RATIONAL AGENTS : PRIORITIZED GOALS, GOAL DYNAMICS, AND AGENT PROGRAMMING LANGUAGES WITH DECLARATIVE GOALS

Shakil M. Khan

A dissertation submitted to the Faculty of Graduate Studies
in partial fulfilment of the requirements
for the degree of

Doctor of Philosophy

Graduate Programme in Department of Electrical Engineering and Computer Science
York University
Toronto, Ontario

June 2018

Abstract

I introduce a specification language for modeling an agent's prioritized goals and their dynamics. I use the situation calculus along with Reiter's solution to the frame problem and predicates for describing agents' knowledge as my base formalism. I further enhance this language by introducing a new sort of *infinite paths*. Within this language, I discuss how to systematically specify prioritized goals and how to precisely describe the effects of actions on these goals. These actions include adoption and dropping of goals and subgoals. In this framework, an agent's intentions are formally specified as the prioritized intersection of her goals. The "prioritized" qualifier above means that the specification must respect the priority ordering of goals when choosing between two incompatible goals. I ensure that the agent's intentions are always consistent with each other and with her knowledge. I investigate two variants with different commitment strategies. Agents specified using the "optimizing" agent framework always try to optimize their intentions, while those specified in the "committed" agent framework will stick to their intentions even if opportunities to commit to higher priority goals arise when these goals are incompatible with their current intentions. For these, I study properties of prioritized goals and goal change. I also give a definition of subgoals, and prove properties about the goal-subgoal relationship.

As an application, I develop a model for a Simple Rational Agent Programming Language (SR-APL) with declarative goals. SR-APL is based on the "committed agent" variant of this rich theory, and combines elements from Belief-Desire-Intention (BDI) APLs and the situation calculus based ConGolog APL. Thus SR-APL supports prioritized goals and is grounded on a formal theory of goal change. It ensures that the agent's declarative goals and adopted plans are consistent with each other and with her knowledge. In doing this, I try to bridge the gap between agent theories and practical agent programming languages by providing a model and specification of an idealized BDI agent whose behavior is closer to what a rational agent does. I show that agents programmed in SR-APL satisfy some key rationality requirements.

Acknowledgements

In the name of The God, most Gracious, most Merciful.

First and foremost, I want to thank my supervisor Yves Lespérance for his constant supply of encouragement throughout the long gestation period of this thesis, for sharing his deep technical insights, for carefully reading numerous drafts, and above all, for infecting me with his passion. Without his efforts, this thesis would probably never have been completed.

It has been an honour to have John-Jules Ch. Meyer as my external examiner, and I greatly thank him for his careful reading of my thesis and valuable feedback. I cannot think of how someone could do a better job of appraising a thesis. I also want to thank the other members of my committee Mikhail E. Soutchanski, Sotirios Liaskos, Zbigniew Stachniak, Johnathan Ostroff, and Franck van Breugel, for their continuing encouragement, stimulating questions, and constructive comments.

I am grateful to all of those with whom I have had the pleasure to work during this and other related projects, in particular, Aijun An, Ahmed Y. Tawfik, and Sotirios Liaskos. Numerous faculty members and fellow graduate students both at York and elsewhere have contributed to this work by their criticisms, discussions, and encouragements. I will mention Steven Shapiro, Sebastian Sardiña, Hojjat Ghaderi, John Thangarajah, Gita R. Sukthankar, M. Birna van Riemsdijk, Parke Godfrey, Hamzeh Roumani, Erick Martínez, Tak-Him Erich Leung, Zahid Abul-Basher, but the list could go on and on. I am grateful to Ulya Yigit and Seela Balkissoon for all of their computer and technical assistance throughout my graduate program. The department's administrative staff have always been cheerfully helpful. Financial support was gratefully received from the Department of Electrical Engineering and Computer Science and the Government of Ontario. Finally, I want to thank my family and friends for their love and support. A very special appreciation goes out to my wife Anika, without whose love, patience, and encouragement, I could not have persevered in this endeavor.

Table of Contents

List of Tables

List of Figures

Index to Symbols and their Definitions

Chapter 1

Introduction

1.1 Introduction

According to cognitive scientists, theory of mind/common sense psychology is a key component of human cognitive capabilities. In at least some cases, it allows us to predict and explain the behavior of others by ascribing high-level mental attitudes such as beliefs, goals, intentions, etc. to them, and assuming that they will behave rationally by performing actions that best satisfy their goals and intentions based on what they believe. Philosopher Daniel Dennett [58] called this adopting the *intentional stance*. This turned out to be one of the most successful abstractions proposed to date in an attempt to unveil the mysteries of human mind: how mind works, and how we are able to predict the behavior of others with surprising accuracy and to adjust our own behavior accordingly. Following on this success, AI researchers have also been asking whether such an abstraction is useful in designing, analyzing, and predicting the behavior of complex artificial systems. Adopting the intentional stance for these systems, often refereed to as *agents*, has at least a couple of advantages. First, it allows the designer of an agent to specify it from a high level. Secondly, and more importantly, it allows an agent to reason about other cooperating or competing agents with little or no knowledge about their internal design, and only in terms of their mental states.

Today the term 'agent' is used in so many different ways and contexts that its use here requires some clarification. I consider an agent to be an artificial, intelligent, adaptive, and autonomous system that can be described using mental states such as beliefs, knowledge, goals, intentions, etc., and whose behavior, at least to some extent, can be predicted in terms of these associated mental states. Agents are typically situated in an environment inhabited by other agents with whom they communicate, cooperate, negotiate, and compete.

1.2 The Context

Research on intelligent agents can be divided into three related areas that focus on integrated treatments of agents: agent theories, architectures, and languages [251]. All three of these areas are concerned with different aspects of the same enterprise, namely, the specification and implementation of intelligent agents.

Agent theories (e.g. [151, 35, 36, 181, 213, 130, 249, 97, 201, 113, 89, 203]) use formal logic to model various mental attitudes of an agent, such as beliefs, desires, goals, intentions, commitments, abilities etc., and the dynamics of these attitudes. They also formalize how these mental attitudes relate to each other and to the agent's behavior, i.e. they specify the inter-attitudinal constraints that are required to manage the rational balance needed among an agent's beliefs, goals, plans, commitments, and intentions. Finally, these theories have also grappled with problems associated with multiple agents, such as the coordination of and communication between a group of agents, and the modeling of joint mental attitudes.

Agent architectures (e.g. [174, 108]) provide mechanisms for managing the mental and physical resources of an agent to meet the demands of complex, dynamic environments.

Agent programming languages (e.g. [205, 172, 140, 51, 13, 47, 101, 186]) attempt to bridge the gap between theory and practice by specifying mechanisms for agents to dynamically select and execute plans to achieve their goals. These languages often have representational primitives corresponding to various mental state components of the agent. An agent performs reasoning on these primitives to decide which plan she should commit to and what action she should execute next. The key issue in agent programming language design concerns the "right way" of programming an agent, i.e.

an autonomous, reactive, proactive, situated, and interacting computing element. In other words, what kind of programming language components should these languages contain so as to allow the programmer to design an agent in the most convenient, most natural, most succinct, most efficient, and most comprehensible way, and how to effectively execute these agent programs.

Recently, there has been a fair amount of work on establishing a link between agent logics and agent programming frameworks by incorporating declarative goals in agent programming [100, 247, 233]. In addition to supporting the definition of procedures which are executed to try to achieve a goal, these programming languages also formalize goals as declarative descriptions of the states of the world which are sought. These declarative goals can be used to decouple plan failure/success from goal failure/success. An agent may be able to successfully execute a plan. However, this does not necessarily mean that the agent was successful in achieving the associated goal, since the environment may interfere with this. Similarly, external interferences may render a user-defined plan impossible to execute; but this does not necessarily mean that the agent will never be able to achieve the associated goal. Thus, these declarative goals play an essential role for monitoring goal achievement and performing recovery when a plan has failed. Since these goals capture the reason for executing plans, it's not hard to see that they are also necessary to model rational deliberation and action, and to model rational response to changes in goals that result from communication, e.g. requests.

1.3 The Problem

There has been much work on modeling agents' mental states, beliefs, goals, and intentions, and how they interact and lead to rational decisions about action. As well, there has been a lot of work on modeling belief change. But the dynamics of motivational attitudes has received much less attention. Most formal models of goals and goal change assume that all goals are equally important and many only deal with achievement goals (an achievement goal is a goal to eventually establish some state property; other types of temporally extended goals include maintenance goals, i.e. goals to maintain some property over some time interval). Moreover, most of these frameworks do not guarantee that an agent's goals will properly evolve when an action/event occurs, e.g. when the agent's beliefs/knowledge changes or a goal is adopted or dropped. Also, they do not model the dependencies between goals and the subgoals and plans adopted to achieve these goals. For instance, subgoals and plans adopted to bring about a goal should be dropped when the parent goal becomes impossible, is achieved, or is dropped. Dealing with these issues is important for developing effective models of rational agency. It is also important for work on belief-desire-intention (BDI) agent programming languages, where handling declarative goals is an active research topic.

In addition, while current agent programming languages with declarative goals have evolved over the past few years, to keep them tractable and practical, they sacrifice some principles of rationality. In particular, while selecting plans to achieve a declarative goal, these agent programming languages ignore other concurrent intentions of the agent. As a consequence, the selected plan may be inconsistent with the agent's other intentions. Thus the execution of such an intended plan can render other contemporary intentions impossible to bring about. Moreover, most of these languages typically rely on syntactic formalizations of declarative goals, subgoals, and their dynamics, whose properties are often not well understood. Again, most assume that all the goals of the agent are equally important. Finally, only a few formalize temporally extended goals and thus most are restricted to achievement goals.

One of the primary reasons for these deficiencies in agent programming languages is the fact that it is quite challenging to formalize an agent programming language that is sufficiently expressive and that can also be implemented relatively efficiently. To cope with the complexity that comes with expressiveness, most existing agent programming languages with declarative goals follow a similar pattern: they start with an agent theory that has very little expressive power, and specify an agent programming language based on this theory. The inherited limited expressiveness of these agent programming languages in turns pushes the agents specified using them further away from rationality and contributes to the aforementioned limitations. The lack of proper formalizations of goals and their dynamics in the agent theory literature also partly contributes to these limitations.

1.4 The Approach

Here, I focus on the specification of prioritized goals and agent programming languages with prioritized declarative goals. First, I develop a logical account of prioritized goals and their dynamics. My model of goals supports the specification of general temporally extended goals, not just achievement goals, and handles goal dynamics and goal-subgoal dependency; a model-theoretic semantics is provided. In my framework, goals are revised appropriately when knowledge changes and when new goals are adopted or existing goals are dropped. I use an action theory based on the situation calculus [148] along with my formalization of paths in the situation calculus as my base formalism for this. I prove that the proposed account has many intuitively desirable properties.

As an application of my theory, I provide a specification for an agent programming language with declarative goals that is based on a version of this rich theory, combining elements from BDI agent programming languages (e.g. [172]) and from the situation calculus based ConGolog agent programming language [51]. While doing this, I address some of the aforementioned problems of current agent programming languages. In particular, my agent programming language supports prioritized goals and is grounded on a formal theory of goal change. I ensure that the agent's declarative goals and her procedural goals (i.e. plans) are consistent with each other and with the agent's knowledge. I also show that agents programmed in my language satisfy some key rationality requirements.

1.5 Contributions

The contributions of this thesis can be divided into the five following parts. First of all, to support modeling temporally extended goals, I introduce a new sort of *infinite paths* in the situation calculus, and propose an axiomatization of infinite paths. The rationale behind introducing infinite paths as a sort is that of expressiveness: it allows first-order quantification over paths and one can also evaluate goals over these infinite paths and handle arbitrary temporally extended goals, e.g. unbounded maintenance goals; such unbounded goals can't be modeled in terms of finite paths. I prove the correctness of my axiomatization and show that infinite paths in the situation calculus are well behaved and indeed correspond to an intuitive notion of paths.

Secondly, I present a formalization of prioritized goals and their dynamics within the agent theory paradigm. An agent specified using this theory always tries to optimize her prioritized goals. My contributions to this end are as follows: I support a rich specification of the goals of an agent. In my agent theory, an agent can have multiple goals/desires at different priority levels, possibly inconsistent with each other. I assume that goals are totally ordered w.r.t. the priority ordering. I define intentions/chosen goals, i.e. the goals that the agent is actively pursuing, as the maximal set of highest priority goals that are consistent with each other and with the agent's knowledge. My model of goals supports the specification of general temporally extended goals, not just achievement goals.

I also specify how these goals evolve when actions/events occur, when the agent's knowledge changes, or when the agent adopts or drops a goal. My formalization of prioritized goal dynamics ensures that the agent always *optimizes her chosen goals*. She will abandon a chosen goal ϕ if an opportunity to commit to a higher priority goal, that is inconsistent with ϕ, arises (cf. Chapter 4 for a concrete example). As such my model displays an idealized form of rationality. In Chapter 4, I discuss how my proposal compares to Bratman's [20] model of practical rationality that takes into consideration the resource-boundedness of real world agents. Note that, being ideally rational, my theory is computationally costly as it requires the agent to constantly deliberate about her intentions, and thus may not be suitable as a foundation for an agent programming language, which are supposed to be more practical, often at the cost of rationality.

I show that my agent theory for “optimizing agents” has some basic desirable properties such as the consistency of chosen goals, that adopting and dropping goals have the expected effects, that agents can introspect their goals, etc. I also prove some properties that specify the conditions under which an agent’s achievement prioritized goals and achievement chosen goals persist. Finally, I show that the framework satisfies some goal change postulates found in the literature [43, 44, 107]. To illustrate the use of the framework, I then specify a personalized travel planning domain in the framework and prove some properties of the specification.

Thirdly, I propose an account of subgoals and their dynamics within my theory of prioritized goals for optimizing agents. More specifically, I give a definition of subgoals and discuss how subgoals change when an agent’s knowledge changes as a result of the execution of an action, or when she adopts a subgoal relative to a parent goal. I show that my formalization of subgoal dynamics ensures that a subgoal is dropped when its parent goal becomes impossible or is dropped, but not necessarily vice versa.

As mentioned above, my formalization of prioritized goals (and subgoals) for optimizing agents presents difficulties for resource-bounded agents. To deal with this, fourthly, I develop a modified version for more committed agents that maintain a commitment to a prioritized goal ϕ as long as ϕ is known to be possible, ϕ is not inconsistent with other (already adopted) higher priority goals, and ϕ is not explicitly dropped. Unlike in the optimizing agent framework, in this committed agent framework the agent's prioritized goals are dropped permanently as soon as they become unrealistic or conflicting with other higher priority goals. So a goal cannot be dropped because a conflicting higher priority goal has become consistent with the agent's higher priority goals (cf. Chapter 6 for a concrete example). As discussed in Chapter 6, this "committed agent" framework is closer to Bratman's original proposal since it accounts for the resource-boundedness of the agent by limiting her deliberation.

I also prove properties about committed agents that are similar to those I proved for optimizing agents, and in light of these properties, I discuss the similarities and the differences between these two frameworks. In particular, I show that prioritized goals/desires are more persistent in the optimizing agent framework than in the committed agent framework, while in the latter chosen goals/intentions are more persistent.

Finally, I contribute to the foundations of BDI agent programming languages/frameworks with declarative goals by developing the specification of a Simple Rational Agent Programming Language, SR-APL, that is based on my theory of prioritized goals and subgoals for committed agents. To this end, I discuss the components of SR-

APL, and give a set of transition rules that specifies the semantics of SR-APL. I focus on developing an expressive agent programming language that allows one to specify agents that are rational without worrying about tractability. I maintain that while efficiency is essential for an agent programming language, one needs to first understand what rationality in a programming context really means. With such an understanding, one can then try to identify restricted versions of the language that are more tractable. This bridges the gap between agent theories and agent programming languages by ensuring that the behavior of any agent specified using the agent programming language is close to what a rational agent does.

Using an example, I discuss how the proposed programming framework compares to existing agent programming languages with declarative goals. In particular, I show that when effects of actions are not reversible or when goals with deadlines are considered, agents specified in other programming languages may behave irrationally in the sense that they can adopt and execute plans that make some of their other goals/plans impossible to achieve/execute. I then show that in the absence of external interferences, an agent specified in SR-APL behaves rationally in that in each state she satisfies some key rationality principles, and the evolution of her agent program is rational. In the future, I would like to investigate restricted versions of SR-APL that are practical, with an understanding of how they compromise rationality.

Parts of these contributions or their preliminary versions have been published earlier. In particular, part of the work on infinite paths appeared in a recent publication in KR 2016 [122]. Some of the work on prioritized goals and subgoals were published in AAMAS 2009 [115], Commonsense 2009 [114], DALT 2009 [117], and AAMAS 2010 [116, 112]. Finally, preliminary versions of the work on simple rational agent programming language appeared in the proceedings of DALT 2010 [118], AAMAS 2011 [120], and ProMAS 2011 [119].

1.6 Organization

The thesis is organized as follows: in the next chapter, I discuss previous work on agent theories and agent programming languages. I focus primarily on existing formalizations of goals and goal change and BDI agent programming languages with declarative goals. In Chapter 3, I present the foundational work that my theories are based on: I discuss a formalization of action in the situation calculus and an account of knowledge and knowledge change in the situation calculus. I then propose my axiomatization of infinite paths in the situation calculus, followed by some properties of paths. I also discuss the situation calculus-based ConGolog agent programming language. In Chapter 4, I develop my account of prioritized goals and their dynamics for optimizing agents. In Chapter 5, I formalize subgoals and their dynamics within the optimizing agent framework. In Chapter 6, I propose a modified version of the framework to model committed agents. Then based on this committed agent framework, I present a model for a rational agent programming language with prioritized declarative goals in Chapter 7. Finally in Chapter 8, I summarize my results and discuss possible future work.

Chapter 2

Literature Review

2.1 Introduction

In this chapter, I review previous work on agent theories, agent architectures, and agent programming languages. In the next section, relevant work on agent theories are considered. In this, my discussion is primarily centered around previous proposals on motivational attitudes such as goals, desires, and intentions, and how they relate to agents' knowledge/beliefs and future actions. In Section 2.3, I review previous work on agent architectures.[1] Then in Section 2.4, I review work on agent programming languages in general, followed by those with declarative goals in Section 2.5.

[1]Some of the material in these sections is adapted from [180].

2.2 Agent Theories

Viewing an entity as an agent involves ascribing high-level cognitive attitudes such as beliefs, goals, desires, and intentions to agents. As mentioned earlier, this is called taking an *intentional stance*. Often, we design agents by representing such attitudes explicitly and implementing reasoning procedures over them. Theories that formalize various aspects of agents are often known as belief-desire-intention (BDI) theories.

Most of the existing BDI theories attempt to deal with one or more of the following questions:

- What are the connections between agents' informational attitudes and their actions, i.e. what are the informational preconditions of actions and what effects do actions have on these attitudes?
- What motivational attitudes (e.g. goal, choice, and intention) are necessary, and how should these attitudes be formalized? How should agents' motivational attitudes change as a result of actions?
- What inter-attitudinal constraints are required to manage the rational balance needed among an agent's beliefs, goals, plans, commitments, and intentions?
- How should one formalize ability, i.e. under what conditions can one expect an agent to succeed in achieving her intentions?

- What does it mean for an agent to behave rationally? How should an agent's future behavior depend on her current mental attitudes?

- How can multiple agents coordinate to achieve a common goal?

In the following, I discuss previous work that attempts to address some of these questions.

2.2.1 Informational Attitudes and Action

One concern of BDI logic theories has been to formalize informational attitudes, such as knowledge and belief, and their relationship to action. This relationship can be broken down into two aspects, namely, the informational preconditions of actions, and the effects of actions on agents' information. In BDI logic theories, the informational attitudes of the agents are almost always modeled using accessibility relations on possible worlds [125, 126, 127]. However, along with the associated computational complexity, possible worlds reasoning suffers from a number of other problems, such as *logical omniscience* (i.e. that of knowing all valid formulae, and that of knowledge being closed under logical consequence). To deal with this, various other approaches has been proposed in the literature [70].

Moore [151, 152] was a pioneer to integrate knowledge and action into a single framework. In his formal theory, he accomplished this by formalizing Hintikka's modal logic of knowledge [102] within McCarthy's first order situation calculus [148]. Variants of this model were later proposed in [134, 188, 50]. Others added informational modalities to other logics of action, such as dynamic logic [232] and CTL* [174, 213].

Moore's main concern was to study the problem of knowledge preconditions for actions, that is, the question of what an agent needs to know in order to be able to perform some action. Moore formalized the ability to perform an action using an agent's knowledge of the referent of the action. In his framework, an agent knows the referent of an action, if it denotes the same action in all of the agent's possible worlds. If the agent knows that the action is also executable, then she is able to perform the action.

Moore, and later others [52, 134, 213, 50], also define what it means for an agent to satisfy the knowledge preconditions of *complex actions*. The underlying concept in these accounts is that to know how to do a parameterized action, the agent must know the "procedure" (i.e. the action function), and know the value of the arguments of the action. For instance, an agent can perform the action of dialing the combination of a safe $Safe_1$ (i.e. $dial(combOf(Safe_1))$) if she knows the procedure the $dial$ action refers to, and knows the value of the term $combOf(Safe_1)$. Also, an agent can perform a complex action if she knows that she can execute a sequence of primitive actions that implements the complex action. However, the agent is not required to know in advance the exact sequence of actions she will execute, since the sequence could depend on information the agent gathers along the way. At all time points the agent must know how to execute the current step of the complex action, know that she will eventually complete the execution of the complex action, and that she will know when the execution is complete. Both Moore's and Singh's theories are limited to determinate complex actions, but Davis, Sardiña et al., and Lespérance et al. handle indeterminate complex actions. Although Moore's framework allows multiple agents, the definition of ability with respect to a complex action involves only action by a single agent. Also, Moore does not address the frame problem for actions, that is, how to specify what remains unchanged after an action is performed. Finally, the framework ignores the formulation of ability to achieve a goal and its relation with being able to perform a complex action. Others, however, do mention how it is a simple matter to add.

A different account was presented in [227], where van der Hoek et al. introduce a primitive capability operator in a propositional modal logic. This operator indicates, for each primitive action and world, whether an agent is capable of performing that action in that world. In this account, the capabilities of complex actions are defined in terms of capabilities of primitive actions. This approach is more flexible than the others, since concepts such as moral capacity can be easily incorporated in this account. However, in this account, one has to specify for each agent, world, and primitive action whether the agent is capable of performing that action in that world. In other words, every instance of a procedure/parameterized action needs to be handled distinctly in this account.

An agent is unable to perform some action when she does not satisfy the informational prerequisites of that action. In that case, it would certainly be useful if the agent had the means to acquire the necessary information. Thus, any agent framework should formalize actions that an agent can use to increase her information. These actions are often known as *informative actions* [152], *test actions* [230], and *knowledge producing actions* [188]. In most agent frameworks, the effect of these actions on agents' knowledge are handled in a similar manner. The result of performing an action that tests the value of a proposition p is that the worlds that disagree with the value of p in the real world are removed from the epistemic accessibility relation. With the situation calculus based formalisms [152, 188, 189], there is an additional requirement that after performing any action a in situation s, the situations that are epistemically related to s are projected/extended by a to get the updated state. In other words, the situations which have not resulted from performing a in an epistemic alternative in s cannot be in the epistemic accessibility relation at $do(a, s)$. So, in this case, even "non-informative actions" affect the knowledge of the agent in the sense that the agent gets to know that it has just performed a. On the other hand, in [230], non-informative actions do not affect the epistemic accessibility relation; also, in their dynamic logic based language, there is no way to say that the agent has just performed action a.

Most of the work already mentioned assumes that new information is consistent with existing knowledge (this is called *belief expansion*). There has also been much work on *belief revision* (and *contraction*) where this may not be the case [86, 231]. Also various researchers have proposed mechanisms for iterated belief change, possibly with noisy sensors [86, 202, 195].

2.2.2 Motivational Attitudes

Along with agents' informational states, a general theory of agency must also take their motivations into account, since agents are expected to act to achieve their goals. To this end, the general trend followed in the literature is to specify a primitive motivational attitude, and then to define compound and more useful motivational attitudes using this. There are two main categories of motivational primitives, namely *goal* [35, 174] (variously known as *choice* [181], *wish* [20], and *preference* [232]), and *intention*. While goals are sometimes allowed to be inconsistent and thus difficult to formulate [20], intentions are mostly considered to be consistent. Another difference between these attitudes lies in the agent's level of commitment towards them. Intention is sometimes primitive [174, 124, 97] and sometimes a defined concept, specified in terms of goals [35, 181, 213]. In his philosophical work [20], Bratman argues that intention is different from goals. He identifies the following important properties of intention:

1. Intentions pose problems for agents; they need to determine a way of bringing about their intentions.

2. Intentions provide a filter for adopting new intentions; intentions that are incompatible with an agent's currently held intentions can not be adopted.

3. Agents will maintain an intention if they attempt to achieve it, the attempt fails, but they still believe the intention is achievable.

Previous work on motivational attitudes mostly concern two aspects of these attitudes, namely, when goals are satisfied, and how long should goals persist. The former differentiates *maintenance goals* and *achievement goals*, while the latter can be used to specify different levels of commitments to a goal. Maintenance goals are propositions that are currently true, and the agent wants it to remain true. Achievement goals, on the other hand, are propositions that are currently false, which the agent would like to be true eventually. Most of the research in the literature focuses on achievement goals.

In the literature, there have been various proposals characterizing different types of persistence of motivational attitudes.

In their linear time temporal model, Cohen and Levesque [35] define a primitive goal modality with its own accessibility relation G similar to the belief modality. Since their model does not have branching futures, it cannot be used to distinguish between some/all branches; rather it can only be used to talk about the actual future. Intuitively, the G-accessible worlds are the ones where all the goals of the agent are satisfied. The goals of the agent are formally defined to be the propositions that are true in all the agent's G-accessible worlds. According to Cohen and Levesque's definition, an agent has a proposition p as an achievement goal if she has the goal that p eventually be true, and believes that it is currently false. Others have adopted a similar primitive motivational operator [174, 181, 232].

Cohen and Levesque point out, following Bratman [20], that since an agent's goals should be compatible with her beliefs, her goal worlds should be constrained to be a subset of the believed worlds. This constraint is known as *realism*. It ensures that the agent does not have an impossible goal. I discuss whether this constraint is actually desirable in the next section.

Cohen and Levesque [35] also investigated the persistence of achievement goals. To ensure that agents do not procrastinate forever, they assume that eventually all achievement goals get dropped. Using this assumption, they then define two types of persistence. According to them, an agent has p as a *persistent goal* if p is one of her achievement goals, and if she does not drop the goal until either she believes the goal has been achieved, or believes that the goal will never be achieved. Cohen and Levesque acknowledge that agents with persistent goals are fanatically committed to their goals. To remedy this, they therefore define the notion of a *relativized persistent goal*, which is a persistent goal with the further condition that the agent may drop the goal if she comes to know that some proposition q has become false. Typically, the goal p is a means to achieving the super-goal q, or q can be some requester agent's intention. The idea behind this is that the agent adopted the goal relative to the condition q being true, so if q becomes false the agent can freely drop the goal. By Cohen and Levesque's definition, it is not known whether an agent has a persistent goal until the agent drops the goal.

Cohen and Levesque [35] define intention as a special kind of persistent goal. In their framework, they distinguish between an intention to perform an action and an intention to achieve a proposition. An agent intends to perform action a, if she has a persistent goal to perform a, immediately after believing she would perform a. Here, the constraint about the agent's belief is required to prevent the agent from intending to accidentally or unknowingly perform the action. An agent intends to achieve proposition p if she has the persistent goal to perform some sequence of actions a, after which p is true. Moreover, immediately before performing a, it is required that the agent believes that she is about to perform some (possibly different) sequence of actions a_0, after which p holds, and that she does not have the goal that it not be the case that p is true immediately after doing a. Note that the believed sequence of actions can be different from the actual sequence. The reason for this is that it allows the agent to intend to bring about a state of affairs without knowing in advance exactly how to do it. It is also required that in at least one of the agent's chosen worlds, the agent performs a, i.e., the sequence of actions that the agent really does, after which p holds. This is meant to rule out cases in which the agent is trying to do, say, a_1 to bring about p, but in the course of doing a_1, she accidentally ends up bringing about p in a completely unforeseen way before completing a_1. Both these definitions of intention can be relativized to a condition q by replacing the persistent goal with a relativized persistent goal. Cohen and Levesque [35] show that their definition of intention satisfies the properties of intention given by Bratman [20].

Rao and Georgeff [174] adopt a primitive intention modality in addition to the goal modality. Their formal language is based on a first-order modal logic that is a variant of the Computation Tree Logic (CTL*) framework [67]. This is a branching-time temporal logic, so it can refer to possible or optional futures (i.e. formulae that hold in at least one branch among the paths emanating from the current situation) and inevitable futures (i.e. formulae that hold in all branches emanating from the current situation), in contrast to Cohen and Levesque's account, where one can only talk about the actual/inevitable future. However, unlike in the latter, there is no way to talk about the actual future, i.e. the path that represents the actual evolution of the world. In their framework, the intention accessible worlds are the worlds that the agent is *committed* to trying to actualize. The intentions of the agent are then defined as the propositions true in all the intention accessible worlds.

Rao and Georgeff [174] studied the persistence of intentions rather than the persistence of goals. Like Cohen and Levesque, they assume that all intentions eventually get dropped. They define three types of commitments, namely, *blind commitment*, *single-minded commitment*, and *open-minded commitment*. An agent is blindly committed to an intention if she maintains her intention to achieve a goal until she believes that the goal holds. An agent is single-mindedly committed to an intention if she maintains her intention to achieve a goal until she believes the goal is true, or until she doesn't believe the goal might eventually be true. Finally, open-minded commitment is the same as single-minded commitment, except that the agent can drop the intention if she drops the goal that the proposition might eventually be true, instead of dropping that belief.

Sadek [181] revises Cohen and Levesque's account to incorporate a branching time temporal logic. He uses a primitive goal modality called *choice*. One problem with Cohen and Levesque's realism constraint is that there is no way to distinguish between the goals due to realism (i.e. a goal that ϕ provided that the agent believes that ϕ is inevitable) and the goals that the agent actually wants. To distinguish between agents' free choices and choices due to the realism constraint, Sadek [181] introduces the concept of *relevant choice*. According to Sadek, an agent i relevantly chooses that ϕ, if she chooses that ϕ, and that if she does not believe that ϕ is not the case, then she chooses that ϕ (i.e. $RC(i, \phi) \doteq C(i, \phi \wedge (\neg B(i, \neg\phi) \supset C(i, \phi)))$). However, this does not fix the "problem". Sadek's definition of relevant choice along with his KD45 logic of choice implies that any chosen proposition is a relevantly chosen one. Thus Sadek's choice modality seems to be similar to Cohen and Levesque's goal modality.

Sadek uses a weaker definition of achievement goals than Cohen and Levesque. In his branching time temporal model, he only requires that in every chosen world, p will eventually be true in *some* possible future. In contrast, Cohen and Levesque requires p to be eventually true in all possible futures (i.e. in their case, the only, inevitable future) of every chosen world. Sadek [181] modified Cohen and Levesque's definition of a persistent goal, by requiring that the agent *choose* not to drop the goal until either the agent believes that the goal has been achieved, or she believes that the goal will never be achieved. Sadek's definition is meant to allow agents to have some awareness that they were adopting a persistent goal. However, a problem with this definition is that it does not guarantee that the persistent goal will actually persist. This is due to the fact that an agent may have a persistence goal at some time-point, but at a later time-point she might change her preferences and thus drop the goal.

In [181], Sadek presents a definition of an intention to achieve a proposition p, where an agent is allowed to include actions by other agents in her plan to achieve p (as long as the initial actions are done by the agent herself). Unfortunately, this definition has some problems. For example, despite claims to the contrary, the given definition allows the agent to intend p when there is no sequence of actions that the agent believes will bring about p.

Another account of both achievement and maintenance goals was presented in [199, 194, 200] where Shapiro et al. define goals using a knowledge accessibility relation K and a primitive "want" accessibility relation W (or H). Intuitively, the W-accessible worlds for an agent are the *happy worlds* where all her goals are satisfied. They then define goal accessible worlds G as the intersection of W and the knowledge accessible relation K, in the sense that a G-accessible world is a W-accessible world that has a K-accessible world in its history. The reason for imposing this constraint is that this assures that agents' goals are realistic, i.e. an agent does not have a goal that she knows is impossible to achieve (this also holds in Cohen and Levesque's framework, as discussed below). Goals are then defined to be all the formulae that are true over the interval $[now, then]$, where now is a K-accessible world, and $then$ is a G-accessible world. Their account is more flexible than others as it handles both types of goals in a uniform way.

One principle that a logic of intention should satisfy is the *side-effect-free* principle, i.e. that an agent should not necessarily intend all the believed "side-effects" of their intentions. For instance, consider the following example (adopted from [35]): suppose that an agent has the intention to go to the dentist to get her teeth fixed, and also believes that getting her teeth fixed will always involve pain. A normal modal logic[2] of intention along with the realism constraint entails that the agent also has the intention to have pain. Thus, even if after the surgery, she finds out that the procedure didn't cause any pain, she will actively pursue her intention to have pain! This causes problems for many of the normal modal logics of intentions seen so far. I discuss this in detail in the next section. To avoid the problem altogether, various researchers have proposed to use non-normal model logics to model motivational attitudes.

Konolige and Pollack's [124] only motivational operator is a primitive intention operator. The main advantage of their account is that it follows directly from their

[2]A normal modal logic is one that satisfies the K axiom, i.e. $G(\phi \supset \psi) \supset (G\phi \supset G\psi)$. Modal operators with a classical possible worlds semantics satisfy this axiom.

non-normal modal semantics that intention is not closed under logical consequence and conjunction. Thus it does not suffer from the side-effect problem, unlike Cohen and Levesque and Sadek's account. In their semantics, intentions are associated with a set of *scenarios* $\mathcal{I}$. A scenario is the set of possible worlds that satisfies some sentence in the non-modal sub-language of their language. An interpretation satisfies $Intend(p)$ if there is an $I \in \mathcal{I}$ that is a scenario for p, i.e. p is true in all the worlds in I and every world that satisfies p is in I. Their non-normal semantics of intention is equivalent to the minimal model semantics of Chellas [31]. Unfortunately, in their framework they have no requirements that intentions persist, and thus their intention modality is closer to what others use as the goal modality.

In an attempt to obtain a minimal logic of intention, Herzig and Longin [97] model intention using a primitive modal operator. Their semantics of intention modality is a non-normal one, and thus in their framework intention is neither closed under logical truth, nor under logical consequence, conjunction, and material implication.

Other researchers have incorporated a procedural motivation component in their framework; sometimes these are used to define the agent's intentions, and sometimes to model the agent's commitment towards actions. In his branching future logic, Singh [213] offers such a model of intention. The underlying temporal logic of his framework is a very expressive one, and one can model true concurrent execution of actions (modeled not just as interleaved actions, but parallel execution of actions), and actions with varying durations. Moreover, unlike Rao and Georgeff's model, Singh's interpretations single out a branch that corresponds to the actual future. Thus, his language can talk about what will really happen. Singh also introduces a procedural motivation component, called a *strategy*. A strategy is an abstract plan or program built-up from constructs such as primitive actions, waiting for conditions, sequences, conditional strategies, and conditional loops, and is viewed as a description of what the agent is currently trying to achieve. In this framework, agents are assigned a strategy in each state. This is modeled using a function C^Y that associates a strategy with an agent at a world and a time. Singh defines intentions in terms of the strategy the agent is following. An agent is said to intend proposition p if it is a necessary consequence of executing her strategy. Singh uses a strong notion of the persistence of intentions, and stipulates that agents do not change their strategies as long as they are able to continue to follow them.

van Linder et al. [232, 149] use a primitive preference modality called *wish*. Wishes are interpreted as a necessity operator over an accessibility relation W. Thus, agents can have contradictory wishes. They offer a strong definition of achievement goals, where the agent is required to know that there is a sequence of primitive actions that she is able perform and whose execution will achieve the goal, i.e. that the goal is *implementable*. In their framework, achievement goals are known to be currently false. Furthermore, they constrain that a goal must be known to be an *explicit preference* of the agent (defined using the *awareness approach* of [69]). To this end, they introduce a special action called $select$. The result of performing $select\ \phi$ marks the wish ϕ as selected. They then define achievement goals as selected wishes that are unfulfilled and implementable. The advantage of this formalization of goals is that it does not suffer from the side-effect problem, and the *transference* problem, i.e. the problem that all logical tautologies are goals of the agent. This is due to the fact that goals are defined not just as wishes, but explicitly chosen wishes/preferences. van Linder et al. require that agents never drop any of their wishes. However, in their framework, goals are dropped as soon as they are fulfilled or become non-implementable, as required by their definition of goals. Due to this, their goals are really intentions.

To model an agent's commitment, van Linder et al. use a semantic primitive called an *agenda*. They define two meta-actions, *commit(a)* and *uncommit(a)*, that update the agenda of the agent so that the agent becomes committed to and uncommitted from the complex action a. An agent can only be committed to a single complex action at any one time. To commit to a complex action, the agent must be able to perform this action and it must achieve one of her goals; she must also have finished executing her previous commitments (i.e. the agenda must be empty). An agenda is a function that maps an agent and a world to a finite sequence of primitive actions and test actions. This action sequence corresponds to one of the terminating executions of the complex action that the agent is committed to achieving. The result of committing to a complex deterministic action (built from primitive actions, tests, sequences, conditional compositions, and iterations) in a given world w is defined in way such that it updates the agent's agenda by adding the appropriate sequence of primitive and test actions to the agenda not only in w, but also in all the knowledge-accessible worlds in w, and in all the worlds that lie along the execution trajectory of the action (i.e. that can be reached from these worlds by performing some prefix of this sequence). This ensures that commitments are known, and that a commitment to a complex action is linked to commitments to its constituents. On the other hand, an agent can uncommit from a complex action if it is no longer useful in achieving any of her goals. Thus the agenda along with the two meta-actions can be used to model an agent's commitment towards actions that achieve one of her goals. While this account links an agent's declarative intentions with her intended actions, it abstracts from how an agent comes to know that a plan is appropriate for a goal. Also, nothing in the framework prevents an agent from performing something that is not in her agenda.

Most of these logical accounts of goals do not deal with goal dynamics properly. In particular, these frameworks do not guarantee that an agent's goals will properly evolve when an action/event occurs, when the agent's beliefs/knowledge changes, or when a goal is adopted or dropped. In their situation calculus based framework discussed above, Shapiro *et al.* [194, 200] incorporate goal expansion and a limited form of goal contraction. They formalize goal dynamics by providing a successor-state axiom [178] for their want or W-accessibility relation. An agent's intentions are expanded when it is requested something by another agent. After the $request(req, agt, \psi)$ action, agt adopts the goal that ψ, unless she has a conflicting goal or is not willing to serve req for ψ. This is modeled by ensuring that (under appropriate conditions) this action causes agt to drop any situations s' in W s.t. ψ does not hold over $[now, s']$ (as discussed above, here now is a knowledge accessible situation in the history of s'). Their framework also handle a limited form of intention contraction. Suppose that the agent req requests agt that ψ and later decides it no longer wants this. The requester req can perform the action $cancelRequest(req, agt, \psi)$, which causes agt to drop the goal that ψ. $cancelRequest$ actions are handled by determining what the W relation would have been if the corresponding $request$ action had never happened. Essentially, this involves restoring the W relation to the way it was before the corresponding $request$ action occurred, considering all the actions that occurred in the history of the current situation starting just after the $request$, and if required removing situations from W to reflect the occurrence of these actions. A $cancelRequest$ action can only be executed if a corresponding $request$ action has occurred in the past. Thus in this framework, an agent remains committed to a requested goal unless the requester cancels this request. Note that, a goal is retained even if the agent learns that it has become impossible, and in this case the agent's goals become inconsistent.

Shapiro and Brewka [196] modify this framework to ensure that goals are dropped when they are believed to be impossible or when they are achieved. In addition, their account assume a priority ordering over the set of requested goals, and in every situation they compute chosen goals by computing a maximal consistent goal set that is also compatible with the agent's beliefs. In their framework, goals are only partially ordered and inconsistencies between goals at the same level (given goals at higher levels and knowledge) can be resolved differently in different models. In fact, the agent's chosen goals in $do(a, s)$ in a model may be quite different from her goals in s, although a did not make any of her goals in s impossible or inconsistent with higher priority goals, simply because the inconsistencies between goals at the same priority level are resolved differently in s and $do(a, s)$. This is rather unintuitive.

There has been a lot of work on belief revision/update, and researchers have proposed postulates to specify the change in an agent's beliefs due to these operators. For instance, the AGM theory [1, 86] can be viewed as the prototypical example of a belief revision specification. However, there has been very little work on goal/intention change postulates. Part of this can be explained by the fact that these postulates are harder to formulate compared to their belief revision counterparts since the agent's future goals depend on both her current beliefs and goals. To the best of my knowledge, there are only two attempts to propose a set of goal change postulates that can be found in the literature. The first set of postulates is proposed by da Costa Pereira et al. in a series of papers on goal revision for rational agents (e.g., see [43, 44, 42]). In their framework, an agent's state S is a triple $\langle \sigma, \gamma, \mathcal{R}_D \rangle$ that consists of a belief-base σ and a desire-base γ (these are presumably achievement goals), each of which is a set of propositional formulae taken from an object language $\mathcal{L}$ containing the standard Boolean connectives, and a desire adoption rule-base $\mathcal{R}_D$. The latter consists of rules,[3] which depending on the agent's current beliefs and desires, allow her to derive new desires, and are meant to serve as a justification for having a desire. Given a state S, a rule whose antecedent is entailed by the agent's current beliefs and desires is called an *active* rule. An agent's desires are updated both as a result of a new/revised belief b and of adoption of a new desire d. When the agent's beliefs are revised/updated, she removes from her desire-base any desire d for which there is no justification in the desire adoption rule-base, i.e. there is no active desire adoption rule in $\mathcal{R}_D$ that can be used to derive d. In addition, she adds the new desires that can be derived from her

[3]These rules are similar to those in PRS [88]; see Section 2.3.3 for details.

active desire adoption rules. Thus γ is closed under the application of rules from $\mathcal{R}_D$. When the agent adopts a new desire d, a new goal update rule with the antecedent that True is added to her rule-base, which in turns makes her add d to her desire-base. The authors then suppose that an intention/goal selection function $\mathcal{I}$ is provided, which given a belief-base and a desire-base, decides which of these desires the agent should actively pursue, i.e. intend.

Their notion of consistency of goals/desires appeals to a specification of consistency of plans for these goals. Consistency of plans is specified in terms of consistency in ordinary propositional logic, as opposed to using a proper formalization for actions and their preconditions and effects in a suitable temporal framework.

To model prioritized desires, they assume a preference relation $\succeq$ over desires in γ that is reflexive and transitive, which they extend to apply to sets of desires.

In the following, I give their postulates which constrain $\mathcal{I}$. Suppose that $\otimes$ is the desire-base γ revision operator, $\oplus$ is the desire adoption rule-base $\mathcal{R}_D$ update operator (that adds an unconditional rule to $\mathcal{R}_D$ when the agent adopts a new desire), and $S_d = \langle \sigma, \gamma \otimes d, \mathcal{R}_D \oplus d \rangle$ is the updated state resulting from the adoption of desire d in S. Then:

I_1 : For all S, $\mathcal{I}(S)$ is a feasible goal set, i.e. a consistent set of goals that are possible.

I_2 : For all S, if $\gamma' \subseteq \gamma$ is a feasible goal set, then $\mathcal{I}(S) \succeq \gamma'$, i.e. a rational agent always selects the most preferable intention set.

I_3 : If d is consistent with $\mathcal{I}(S)$, then $d \in \mathcal{I}(S_d)$.

I_4 : If d is inconsistent with $\mathcal{I}(S)$ and there is an intention i in $\mathcal{I}(S)$ that is conflicting with d and $i \succeq d$, then $\mathcal{I}(S_d) = \mathcal{I}(S)$.

I_5 : If d is inconsistent with $\mathcal{I}(S)$ and for all intentions i in $\mathcal{I}(S)$ that are conflicting with d, we have $d \succeq i$, then $d \in \mathcal{I}(S_d)$ and $i \notin \mathcal{I}(S_d)$.

While these postulates are based on notions of consistency of sets of desires and executability of desires that seems problematic, given an appropriate interpretation, these postulates are in fact sound. I will come back to this issue in Chapter 4.

Another set of goal change postulates is proposed by Icard et al. [107], who studied the problem of joint belief and intention revision as a database management problem [206] in the spirit of the AGM theory [1]. In their logic, they incorporate explicit time indices, (primitive) actions, and pre- and post-conditions of actions. They define a path π as $\mathbb{Z} \rightarrow (P \times Act)$, i.e. a mapping from the set of natural numbers, used to represent time indices t on π, to a set of propositions and "proposition-like formulas", representing what is true at some time index t on π, and what pre- and post-conditions of actions hold at t on π, respectively, and a set of actions, giving the next action a at t

on π. These paths include impossible or "imaginary" ones as there is no requirement that pre-conditions of actions hold; post-conditions are however guaranteed. Given a path π and time index t on π, they have a modal operator for talking about what holds on all paths that share the same past with π up to time t.

A belief set in this framework is defined using a set of paths Π and a total pre-order $\leq$ on this set; in particular it is the minimal set under $\leq$, i.e. $\{\pi \in \Pi : \pi \leq \pi' \text{ for all } \pi' \in \Pi\}$. Note that while paths include imaginary ones, the set Π above is required to be *appropriate* in the sense that if there is a path π in Π where the preconditions of some action a holds at time t, then Π must include at least one path that share the same history as π up to t and where the next action performed is a. Intentions in this framework is simply a finite set of action/time pairs.

A belief-intention-base $\langle B, I\rangle$ consists of a consistent and closed set of formulae B representing the beliefs of the agent, and a finite set of action/time pairs representing the intentions of the agent. Given a set B, $\rho(B)$ is defined to be the set of paths where all formulae in B hold at time 0. The belief base B is said to be *coherent* w.r.t. the intention base I if there is a path in B (presumably, in $\rho(B)$, i.e. in the set of paths induced by B) over which the preconditions of all actions a, s.t. $(a, t) \in I$, is satisfied at the corresponding time t. Since $\rho(B)$, being a belief set, is also appropriate, a path where a happens at t (with the preconditions of a met) must be included in it. Thus, in this framework, belief-intention consistency is defined via the existence of a path over which each of the intended actions a is performed in the appropriate time step t, and a is executable in t. Oddly, there is no requirement that the path is indeed a realistic one; e.g. nothing in their framework seem to constrain that other (non-intended) actions on this path are in fact executable.

Having discussed their framework, let me now list their intention revision postulates. Suppose $\circ$ is the intention revision operator, and $\langle B', I' \rangle$ is the result of revising $\langle B, I \rangle$ with intention (a, t). Then:

$BI_1:$ $\langle B', I' \rangle$ is coherent.

$BI_2:$ If $\langle B, (a, t) \rangle$ is coherent, then $(a, t) \in I'$.

$BI_3:$ If $\langle B, I \cup \{(a, t)\} \rangle$ is coherent, then $I \cup \{(a, t)\} \subseteq I'$.

$BI_4:$ $I' \subseteq I \cup \{(a, t)\}$.

$BI_5:$ $B = B'$.

BI_1 says that intention revision should restore coherence. BI_2 states that the new intention (a, t) takes precedence over existing ones, and thus if coherence can be attained after adding (a, t), the agent must do so, even if this means that she must give up some of the existing intentions. BI_3 and BI_4 together states that if it is possible to maintain coherence by simply adding the new intention, then this is the only change that should be made. BI_4 in addition says that unlike in the case for belief revision, no extraneous intentions are ever added. Finally, BI_5 states that beliefs do not change with intention revision.

Since belief revision should trigger the revision of intentions, for every belief revision operator $*$ they also include its own intention revision operator $\circ^*$. In addition to the AGM belief revision postulates, they propose two more relevant intention revision postulates. Suppose $\langle B, I\rangle * \phi = \langle B', I'\rangle$. Then:

$BI_6:$ $\langle B', I'\rangle = \langle B', I\rangle \circ^* \epsilon$.

$BI_7:$ If $\langle B, I''\rangle * \phi = \langle B'', I'''\rangle$, then $B' = B''$.

BI_6 says that if intention revision is needed to maintain coherence after beliefs are revised, then intentions must be revised (with an empty intention ϵ). BI_7 on the other hand states that the underlying intention set is irrelevant to belief revision.

The authors presented a representation theorem showing that for every belief-intention-base and belief and intention revision function that satisfy the above properties, there is a corresponding model in their theory.

Note that, intentions in this framework are limited to primitive actions. Also, as mentioned above, their notion of consistency via the existence of a path is problematic as the path is not guaranteed to be realistic. Moreover, clearly the second postulate is disputable and is only acceptable if the intention (a, t) is indeed more valuable to the agent than her existing intentions. BI_5 (and BI_7) also seem problematic for a general framework, specially one where agents are considered to be introspective – the agent needs to revise her beliefs about her intentions when the latter change. Finally, their framework inherits the issues with iterated revision from the AGM theory. Otherwise, the rest of the postulates seem sound. I will return to this discussion in Chapter 4.

2.2.3 Interattitudinal Constraints and Properties

In order to capture some of our intuitions about the intentional attitudes and to prove desirable properties about them, the primitive attitudes discussed in previous sections need to be constrained. One constraint introduced by Cohen and Levesque [35] that received some attention is the *realism constraint*. This is a semantic constraint that says that the goal worlds should be a subset of the believed worlds. The reason for imposing this constraint is that it is unintuitive for a rational agent to adopt an achievement goal that she believes will never hold (this property is also known as *intention-belief inconsistency* [20]). This constraint rules out these states of affairs. Since, each of the agent's goals must be satisfied in at least one of the believed worlds, the agent cannot believe the negation of any of its goals. This constraint is also adopted by Sadek [181] and Konolige and Pollack [124], although Konolige and Pollack constrain the intended worlds (in contrast to the goal worlds, as in Cohen and Levesque's framework) to be a subset of the believed worlds. As mentioned earlier, this constraint also follows from Shapiro et al.'s [199] definition of goal.

Rao and Georgeff [173, 174] introduce two variants of the realism constraint, namely, *weak realism* and *strong realism*. Weak realism constrains the intersection of the believed worlds and the chosen/intended worlds to be non-empty, and is thus weaker than realism. Rao and Georgeff [173] also adopt a similar constraint between chosen worlds and intended worlds. The strong realism constraint [174], that can only be defined for worlds that have branching futures, requires that for every believed world, there is a goal world that is a sub-tree of the believed world with the same truth assignments and accessibility relations at all timepoints. They also adopt a similar constraint between goal worlds and intended worlds. This constraint is stronger than realism in the sense that it restricts the agent to choose only goals that it believes it will always be able to achieve, regardless of what happens in the future. Singh [213] imposes a constraint that ensures that if an agent intends a proposition p, then it does not believe that p will never hold in the real future.

As discussed earlier, another property of intentions proposed by Bratman [20] is that agents need not always intend the believed side-effects of their intentions. There are various forms of this "side-effect-free" principle. The weak version states that an agent should be able to consistently intend that p, believe that p always implies q, and not intend that q. Cohen and Levesque [35] and Singh [213] show that their theories are consistent with this. A stronger version of the principle says that an agent should be able to consistently intend that p, and *always* believe that p always implies q, without intending that q. Cohen and Levesque [35], Rao and Georgeff [173, 174] and Sadek [181] show that their theories are consistent with this version of side-effect-free principle. However, for Cohen and Levesque, this holds for the wrong reason, i.e. because the agent may believe that the side-effect already holds, and thus will not have the persistent goal that the side effect hold, and hence the intention to achieve it. Rao and Georgeff's weak and strong realism constraints are meant to ensure that side-effects need not be intended; these constraints do not rule out the existence of an intention/goal-accessible world that does not agree with the belief-accessible worlds on the truth values of a formula, and thus there may be goal/intention-accessible worlds where $p \supset q$ does not hold. However, their theory is still closed under conjunction. Both van Linder et al.[232] and Konolige and Pollack [124] show that their frameworks are consistent with yet another stronger version of the side-effect-free principle, which says that if p logically implies q, and the agent prefers or intends p, then the agent may consistently not prefer q.

Rao and Georgeff [173] argue that although agents' goals should be potentially achievable, no rational agent should be forced to choose all her beliefs; they call this property *belief-goal transference*. They point out that Cohen and Levesque's realism constraint sanctions this property. Since the chosen worlds are a subset of the believed worlds, the agent chooses all of its beliefs. Sadek [181] argues that transference is not problematic, if one can differentiate between choices made due to the realism constraint and choices made freely by the agent. Rao and Georgeff [173], on the other hand, argue that transference is irrational, and show that both strong and weak realism avoid transference. One could claim that the realism constraint causes problems in Cohen and Levesque's framework because their logic deals with the actual future, rather than possible futures. For instance, in Shapiro's branching future framework, the realism constraint seems unproblematic. What the realism constraint sanctions in Shapiro is that you can't have the goal that p if you believe that $\neg p$ is inevitable, and this seems unproblematic.

Finally, another useful property for goals is that of introspection. If an agent has a goal that ϕ, then it is useful to believe/know that she has this as her goal. This is known as positive introspection of goals. On the other hand, if an agent does not have a goal, she should also believe/know this. This is called negative introspection of goals. In their framework, Shapiro *et al.* [194, 200] showed how positive and negative introspection of intentions can be modeled by placing some constraints on K and W which are similar to transitivity and Eucledeanism. They showed that in their framework, if these constraints are prescribed for the initial situations, they continue to hold for all situations since they are preserved by the successor-state axiom for their want accessibility relation W.

2.2.4 Success Theorems and Rational Action

One important aspect of an agent theory is the connection between beliefs, goals, and intentions and the agent's action. Theorems that link these attitudes to an agent's future behavior and the achievement of her goals have been called *Success Theorems* [213], since they characterize conditions under which one can expect an agent to succeed in fulfilling her intentions. Success theorems are important for motivating goal delegation via communication. For instance, to plan a complex action that involves delegating subgoals to other agents, waiting for those subgoals to hold, and then resume working on the goal, it is necessary for the agent to know that the delegated subgoals will eventually become true. Using a success theorem, it is adequate for the agent to know about the other agent's intentions and abilities to infer this.

In [35], Cohen and Levesque present a success theorem which states that if an agent has proposition p as a persistent goal, is always competent with respect to p

(i.e., whenever the agent believes p, p is true), and it is not the case that the agent will believe p will never occur before she drops p, then eventually p will hold. To relate agents' intentions with their actions, Cohen and Levesque assume that all intentions eventually get dropped. This implies that the agents do not procrastinate indefinitely with respect to their intentions (AKA the *no infinite deferral* assumption). According to the definition of persistent goal, it can only be dropped if the agent believes that the goal has been achieved or is impossible to achieve. The latter case is ruled out by the premise of the theorem, therefore the agent eventually believes p. Moreover, since the agent is competent with respect to p, it follows that p eventually holds. Note that, this theorem does not imply that the agent will eventually act, because some external event might achieve p. But Cohen and Levesque claim that if the agent knows that it is the only one that can act to realize p, then under certain (unstated) circumstances, the agent will act. Nevertheless, this account can be criticized since the no infinite deferral assumption should follow from other axioms of the theory, rather than be imposed separately [113]. Rao and Georgeff [174] also offer similar theorems for their three forms of commitments to intentions.

Singh [210] criticizes these success theorems as being too powerful. In particular, he points out that they do not take into account the abilities of agents. He requires that agents always perform actions that they know will ensure the eventual success of their strategies. Using this assumption, he showed that if an agent knows how to follow her strategy, and if her strategy necessarily leads to p (and thus she intends that p), then eventually p will hold. Both Shapiro et al. [198] and Khan and Lespérance [113] also prove success theorems in which the agent is only guaranteed to achieve her intentions if she is *able* to achieve them. The latter consider multiple agents, in the sense that agents are allowed to delegate actions to other agents.

One important concept that has largely been left out of agent theories is that of rational choice of action. If agents are not acting rationally, then they cannot be expected to achieve their intentions even if they have the required commitment. Singh's agents are trying to "achieve strategies". His assumption that agents perform actions that they know will achieve their strategies actually ensures that agents act rationally. In their earlier work, Shapiro et al. [198] formalize strategies as functions from situations to actions (called *Action Selection Function (ASF)*). They use the agent's goals to provide an ordering on ASFs for each situation. For a given situation s, an ASF σ_1 is said to be as good as another ASF σ_2 iff σ_1 achieves all of the agent's goals in all the alternative situations where σ_2 does. They then define an ASF σ to be a rational course of action for a given situation s iff it is maximal in the ordering for s. In [187], Sardiña and Shapiro extended this concept of domination of strategies to deal with *prioritized goals*. Building on Shapiro et al.'s model of goals in [199, 194], Khan and

Lespérance [113] also use an agent's goals to provide an ordering on complex actions (i.e. plans) for each situation. Plans in their framework can include actions by other agents. They then define a plan to be rational for an agent in a situation iff the plan is maximal in that ordering for that agent in that situation, and the plan is *epistemically and intentionally feasible* for the agent. The additional condition on epistemic feasibility ensures that the agent fulfills the knowledge preconditions required to execute the plan. In case the plan includes actions by others, the agent is also required to know that the executing agent fulfills the knowledge preconditions of the actions delegated on her, and that she intends to execute the appropriate actions in the plan when it is her turn to act. In other words, the plan must be both epistemically and intentionally feasible with respect to the agent. Most work on rational action selection has been done in the decision theory setting, where the outcomes of actions are assigned numerical probabilities and utilities [19, 243, 61]. Boutillier [19] provides a framework for analyzing rational action selection in a logical setting using qualitative orderings for preferences and likelihoods.

2.2.5 Multiagent Systems: Collective Mental Attitudes, Communication, and Coordination

So far, I have mainly discussed work on modeling individual agents. The study of collective mental attitudes and collaborative action has also received attention. The motivation for such collaborative action is that it is often the case that although no individual member of a group can bring about some goal, collectively they can achieve it if they coordinate and cooperate with each other. Work in this field is variously known as teamwork, cooperative problem solving, team activity, and cooperative plans.

Researchers in multiagent systems have formalized various collective mental attitudes. Common knowledge can roughly be modeled using infinite nesting of a group knowledge (i.e. “everyone in the group knows that”) modality. It is possible to formulate common knowledge as a fixed point formula [96]. Joint intentions are often formalized as a non-primitive construct built using concepts such as mutual belief and intention [192, 38, 39, 91, 92, 109, 110, 138, 218]. Although joint intentions imply individual commitments for the team members, these collective intentions are usually not reducible to summation of individual intentions. Others formalize group intention from an external perspective by utilizing an explicit social structure for a group [208, 212]. Examples of work on joint ability include [91, 94, 160, 161, 209, 228, 246, 250, 252].

When multiple agents are working together to solve a problem, they need some mechanism for exchanging information, such as beliefs, plans, intentions, synchronization information, and so on. Communication plays an important role in the coordination of multiple agents. Agent communication theories are founded on *Speech act theory* [4, 191], which originated in the philosophy of language. The key idea in speech act theory is that communication acts can be viewed as regular actions. While most actions are performed to change the physical state of the world, communication actions are mainly done to alter the hearer's mental states. Speech acts can be categorized into assertives (e.g. informing), directives (e.g. requesting), commissives (e.g. promising), declaratives, permissives, and prohibitives.

The literature on speech act theory is quite extensive. Cohen and Levesque showed that in a BDI framework, many properties of speech act theory can be derived from an independently motivated theory of rational interaction, which is in turn grounded in the rational theory of action [36, 37]. Following this approach, they proposed formal semantics for speech acts. They model speech acts as attempts to bring about some effects by performing some sequence of events, but with the intent to produce at least some result. For instance, a *request* to achieve ϕ is considered as an attempt by the speaker to have the hearer bring about ϕ, with the intent of at least making the hearer believe that it is mutually believed that the speaker has the goal that the hearer brings about ϕ. Singh [211], on the other hand, argued that the semantics of speech acts corresponds to their satisfaction conditions, and identified these conditions for different types of speech acts. For instance, an assertive is satisfied if its propositional content is true at the time of the utterance. A directive is satisfied if its proposition comes to hold at a later point in the future, and the hearer has the know-how and the intention to achieve it. A commissive is almost like directive except that the role of hearer and speaker is switched, and so forth for permissive, prohibitive, and declarative speech acts. Herzig and Longin [97] proposed some cooperative principles, and showed how these rules can be used to infer the effects of a *yes-no* question and that of a *request* from that of an associated assertive. Their model provides a simpler logical account where only assertives are primitive.

In the ARTIMIS rational agent model [182], Sadek formalizes planning for communicative actions using a backward chaining planning mechanism that utilizes the *feasibility preconditions* (FP) and *perlocutionary effects* (PE) of the speech acts. Here, the FP specify the conditions that have to be satisfied in order to plan for the act, and the PE correspond to the rational effects of the action. For instance, consider the action of i informing j that ϕ is true. The preconditions for this inform action is that i should believe that ϕ holds, and i should not know that j knows that ϕ holds. On the other hand, the rational effects of this action are that j will learn that ϕ holds. Sadek argues that rational effects of a communicative act serve as the reason for planning that act, in the sense that an agent should select an action only if she needs to achieve the rational effect of that action.

One problem with this account is that it fails to specify the conditions under which the rational effects become actual effects; one cannot reason about these conditions. Moreover, the planning mechanism in [182] is incomplete and many rational plans cannot be inferred. Louis [142] extended this framework to incorporate a more general model of planning (state space planning by regression and hierarchical planning) and plan adoption. His framework is more complex, and uses defaults (as does Sadek's). The approach supports multiagent plans and has been implemented. But there is no formalization of epistemically feasible plans, and no success theorem. Also, commitment to a plan is modeled using a special predicate rather than using the intention attitude.

Examples of other interesting agent communication theories include [245, 244, 163, 40, 164, 128]. Using these theories, researchers have developed artificial languages for agent communication (e.g. FIPA-ACL [79], KQML [68]), proposed semantics for agent communication protocols [83, 82, 80, 81], and implemented cooperative spoken dialogue systems (e.g. ARTIMIS [183, 184]).

There have been many proposals for semantics of communication acts based on social commitments [214, 254, 78, 239]. The commitments associated with a conversation would be accessible to an observer and relevant social rules could be enforced. While it is very important to capture and enforce the social aspects of agent communication, i.e. the obligations that go with membership in an agent society, it should be noted that communication cannot be reduced to this public social commitments level [113]. There has also been a suggestion that public social commitment semantics support more efficient reasoning and are more "tractable". However, it has been also pointed out that this is an orthogonal issue [113].

Another way of coordinating agents is to use social laws [9, 207, 155]. These laws are often modeled by incorporating various deontic logic notions [30], such as obligations, prohibitions, and rights into the framework.

2.3 Agent Architectures

The aim of agent architectures is to shift the emphasis from theory to practice. Thus, researchers in this field are concerned with issues surrounding the construction of computer systems that satisfy the properties specified by agent theorists. In classical planning, the agent is given a model of the actions available and their preconditions and effects on the domain states and a goal, and her job is to find a sequence of actions whose execution brings about this goal. Thus classical planning assumes a static environment. However, real environments tend to be dynamic, that is, they often change in unexpected ways at run-time. They may include exogenous actions (i.e. actions by other agents or natural events) and the world may change during planning. The initial state could change before the agent starts executing the plan. The world might not change as a result of plan execution as expected due to the occurrence of exogenous actions. Thus, a classical planning agent will often not do well in such environments. This is one of the areas where work on agent architecture contributes, by taking into account the resource limitations of the agent. Researchers in this area are concerned with designing agents that may have incomplete information about the current state of the world, are not always able to accurately predict the effects of their actions, can deal with external interference, and do not have arbitrary time to deliberate.

One important issue addressed by the agent architecture community is the tradeoff between commitment vs. intention reconsideration, i.e. how strongly should an agent be committed to her intentions in a changing environment. Generally, intentions are considered to be persistent, and are only dropped when they are achieved or they become impossible to achieve, as discussed in Section 2.2. However, prior intended plans may be subject to reconsideration or abandonment when the agent's beliefs change in various ways, for instance, when the agent becomes aware of a more attractive way of achieving her goal. But, according to Bratman [21], "...if an agent constantly reconsiders her plans, they will not limit her deliberation in the way they need to for a resource-bounded agent." Nevertheless, if an opportunity with very high utility arises, the agent should take advantage of this by weighing competing alternatives and reconsidering her current intentions. Thus, there exists a tension between the stability of intended plans that is required for practical reasoning, and the revocability inherent in these plans, as these are often formed based on incomplete information.

Another related issue in agent architecture is the tradeoff between reactivity and deliberation. While agents need some mechanism to support goal-directed reasoning and deliberation, they must also be able to react rapidly to unanticipated changes in the environment. Moreover, since they only have incomplete information about their environment, it is not always possible for them to produce a complete plan for a given goal. Rather, information about how to best achieve a goal can often be acquired after executing some initial part of the plan.

Researchers have proposed various agent architectures that differ depending on how the agent's ability to act is realized. Previous work on agent architecture can be classified roughly into four categories, namely, deliberative architectures, reactive architectures, reactive plan execution architectures, and hybrid architectures. In the following I discuss these categories.

2.3.1 Deliberative Architectures

A deliberative architecture is one that contains an explicitly represented symbolic model of the world (i.e. beliefs, desires, intentions, and actions), and where decisions, such as what actions to perform next, are made via logical reasoning (i.e. planning). Thus deliberative architectures are based on Newell and Simon's [157] *physical-symbol system hypothesis* –they use a physically realizable set of symbols that can be combined to form structures and are capable of running processes that operate on those symbols according to symbolically coded sets of instructions, in order to produce intelligent action. Most innovations in deliberative architecture design have come from the AI planning community. Since deliberative architectures have a planning process as their central component, these architectures can deal with unanticipated goals. However, the disadvantage of a purely deliberative approach lies in the computational complexity of planning: the agent may not be able to find plans in a timely manner. Also, the plans generated by a deliberative architecture often fail in a dynamic environment. Examples of deliberative architectures whose primary component is a planner include the Integrated Planning, Execution, and Monitoring (IPEM) system [2] (based on a sophisticated non-linear planner), and Wood's AUTODRIVE system [248] (a traffic simulation with planning agents).

One particularly interesting class of deliberative architectures is *plan-based* deliberative architecture. The role of commitment to adopted plans or intentions is a critical component in plan-based architectures. Thus, these architectures use adopted plans to limit practical reasoning. The range of reasoning modeled by these frameworks include means-ends analysis (planning), choosing between alternative courses of action (decision analysis), checking consistency of plans and beliefs, and revising beliefs and goals in response to external events.

Building on Bratman's philosophical work in [20], Bratman et al. [21] consider adopted partial plans to structure and focus practical reasoning in their (mostly) deliberative architecture IRMA (Intelligent Resource-bounded Machine Architecture). IRMA has four key data structures: a plan library, and explicit representations of beliefs, desires, and intentions. In this architecture, once an agent adopts a plan, she becomes committed to executing this plan. The agent's commitment to a plan implies that she will not reconsider this adopted plan, unless the environment has changed in a relevant way, and reconsidering this plan will result in a reasonable increase in utility. Also, she will not adopt any further intentions that are inconsistent with achieving her adopted plans.

The adopted plans can be both temporally and structurally partial, meaning that these plans schedule actions for some time period, but not for others, and that these plans specify goals to be achieved leaving open the means to achieve these ends. As discussed above, the motivation for this is that often at plan time, the agent only has partial knowledge about the world, and thus it is not always possible to decide on a complete course of action. These adopted intentions limit the agent's deliberation since they focus means-ends reasoning, and they constrain the number of alternative options for actions that are fed to the decision process.

A partial plan needs to be filled out using means-ends reasoning. Thus, these adopted partial plans focus the means-ends reasoning of the agent. Given a partial plan, the means-ends reasoning process outputs some options for courses of actions that refine this plan. But not all suggested courses of actions will be consistent with the already adopted plans. Thus, before these suggested courses of actions are supplied to the decision-making process, they need to be passed through a 'compatibility filter'. After filtering out the inconsistent plans, the compatible options are then fed to the decision-maker for further deliberation. Bratman et al. remarked that the compatibility filter must be computationally efficient relative to the deliberation process.

In addition to allowing deliberation, IRMA also attempts to provide some form of reactivity unlike most deliberative architectures. IRMA utilizes an 'opportunity analyzer' and 'filter override mechanism' in an attempt to model reactivity. The opportunity analyzer takes the agent's beliefs and goals as input, and watches for opportunities to satisfy the agent's desires when some change in the environment is detected. While doing this, it ignores the agent's adopted plans. When it detects such an opportunity, it suggests a course of action to fulfill the goal to the compatibility filter. While the compatibility filter's job is to detect and eliminate inconsistent plans, the filter override mechanism can be used to allow some of these inconsistent options to be passed to the decision-maker for deliberation. If the filter override mechanism passes such an option to the decision-maker, it must be the case that at least one of the agent's adopted plans are incompatible with this option. So, in that case, the decision-maker needs to decide to either ignore this incompatible option, or to revise the adopted plans that conflict with this option. Note that, although this process is able to handle unanticipated changes in the environment, it is not a completely reactive mechanism, due to the deliberation involved.

2.3.2 Reactive Architectures

Some architectures that have been proposed are completely reactive in nature. In these, all the deliberation is done in advance and compiled into the architecture itself. In some approaches, the designer is responsible for this compilation (e.g. [24]). Another way of doing this is to use an automatic compilation process [179]. Thus these architectures neither contain a symbolic model of the world nor a reasoner for manipulating rules and finding plans (i.e., they are not knowledge-based). Although such architectures are very efficient and can perform simple tasks quite well, a major problem is that they are ineffective in environments that deviate from those expected by the designer. The primary reason for this is that the behavior of the agents in these frameworks is essentially hardwired. Also, it is often hard to design agents with multiple complex goals in these frameworks.

Perhaps the best known reactive agent architecture is Brooks' subsumption architecture [25, 26, 27]. Brooks proposed that intelligent behavior can be generated without explicit representations of symbols and without explicit abstract reasoning, and that intelligence is an emergent property of certain complex systems. He identifies two key properties of intelligence – 'real' intelligence is situated in the world and not disembodied (such as theorem provers or expert systems), and intelligence arises as a result of a system's interaction with its environment.

To demonstrate the validity of his claims, Brooks built a number of robots using his subsumption architecture. A subsumption architecture consists of a hierarchy of task-achieving behaviors/layers. Each layer in this hierarchy is used to implement a certain goal of the robot. These behaviors compete with each-other to exercise control over the robot. Lower layers are used to encode more primitive kinds of behavior, and have precedence over the layers further up in the hierarchy. Each layer's goal subsumes that of the underlying layers. Each of these layers accesses some of the sensor data and generates actions for the actuators. It should be emphasized that the generation of actions in this system is extremely computationally efficient and does not involve any explicit reasoning, or even pattern matching. A layer can inhibit inputs or overrule outputs of the layers below it. This allows the lowest layers to work like fast-adapting mechanisms, while the higher layers control the main direction to be taken in order to achieve the overall goal. Thus this architecture is capable of reacting quickly to changes in the environment.

2.3.3 Reactive Plan Execution Architectures

A reactive plan execution architecture is one that includes a user-defined library of hierarchical plans. Each of these plans consists of a trigger condition (i.e. goal), a precondition (i.e. context), and a body. The trigger condition specifies what the plan is good for, that is, what goal can be achieved using the plan. The context condition describes the conditions under which the plan should be considered for execution. Another component of a reactive plan execution architecture is triggering events. An agent in this kind of architecture responds to events from an event-queue by adopting the appropriate plan, and by eventually executing it. In addition to primitive actions, the body of a plan can contain subgoals (i.e. events), which may in turn trigger the selection and execution of other plans (sub-plans). Thus, in these architectures, the changes in the environment determine which plans should be executed, and how these plans are decomposed.

The most well known version of this architecture is the Procedural Reasoning System (PRS) [88, 87]. A PRS agent consists of a belief-base, a goal-base, a set of plans, and a set of intention structures. Beliefs in PRS are facts about both the external world and the agent's internal state expressed in first-order logic. Goals are represented as temporal formulae, which include formulae for achieving a property, testing for a condition, waiting for a condition to hold, and preserving/maintaining a condition. These goals are meant to be used in the triggering event part of a plan. Like IRMA, PRS also uses plans to structure reasoning. Plans in PRS are complex structures called *Knowledge Areas* (KA). Each of these KAs is a rule, and consists of an invocation condition that specifies when it is applicable, and a body that describes a set of steps to be achieved. The body of a KA can be viewed as a graph with a single start node and possibly multiple end nodes. Arcs in this graph represent subgoals. A successful execution of a KA amounts to achieving each of the subgoals in a path between the start node and one of the end nodes. Intentions of a PRS agent consist of the set of active KA stacks, each of which keeps track of all the subgoals of the original KA.

PRS uses KAs to encode procedural knowledge about the domain. KAs may be activated in a goal-driven fashion, i.e. as a result of acquisition of a new goal, or in a data-driven/reactive fashion, i.e. as a result of some change in the agent's beliefs. These adopted KAs can be used to structure the practical reasoning, since they constitute the entire reasoning process. At each iteration of the PRS interpreter, the set of applicable KAs are determined (by unification) using the agent's beliefs and active goals. Then one or more of these KAs are selected and inserted into the intention structure for execution. Finally, one of the intentions from the (root of the) intention structure is selected, and a step of this intention is executed. This execution can involve an unelaborated subgoal; in that case, this goal is added to the goals of the system. The interpreter then loops to the next iteration, where a new set of applicable KAs are determined based on the perceived changes. Note that, if the selected KA arose as a result of the acquisition of a new goal (called *intrinsic goal*) or a change in belief, then it is inserted into the intention structure as a new intention. On the other hand, if the selected KA was triggered due to the execution of an already existing intention (called *operational goal*), this KA is pushed on top of the KA stack comprising that intention.

PRS uses a special class of KAs (namely, meta-level KAs) to update the beliefs, goals, and intentions of the PRS agent. Meta-level KAs can also be used to control the adoption of lower-level KAs (e.g. in case more than one KA is applicable), create new subgoals, handle failures, reorder the intentions in the intention structure, etc. Thus KAs are very powerful and can be used to capture procedural domain knowledge as well as decision knowledge.

PRS can be used to guarantee some form of reactivity. In fact, it was shown in [88] that there exists a bound on the 'reaction time' of a PRS agent. Note that, in each iteration, the interpreter checks for applicable KAs and places one or more appropriate KAs in the intention structure. This process uses unification and is able to 'react' in a timely manner. Note however that, although a PRS agent is able to promptly recognize changes in the environment and adopt intentions accordingly, she may take arbitrarily long to 'react to the environment' by executing some action. This is due to the fact that there is nothing in the framework that ensures that the process of hierarchical plan decomposition will quickly converge to a sequence of a primitive actions. In other words, the execution of a knowledge area may involve a long and possibly even an infinite chain of subgoaling. Thus the term 'react' above (as used by Georgeff et al. [88]) is used in a weak sense, and should be read as 'recognize'. Nevertheless, it should be noted that PRS assumes that the designer is responsible to ensure that plan decomposition completes in a reasonable time. In practice, these architectures generally respond fairly quickly to changes in the environment.

2.3.4 Hybrid/Cognitive Architectures

There has also been work on architectures that handle the tradeoff between reactivity and deliberation by implementing reactive mechanisms and a deliberation module in two different but interacting layers. Some architectures also include additional layers for plan execution and/or coordination. These architectures are known as layered architectures. A typical layered architecture works as follows: the reactive layer generates potential courses of action in response to time critical events that happen too quickly to be handled by the other layers. It is often implemented using a set of situation-action rules, and thus does not involve complex reasoning. The reactive execution layer (or scheduler) selects precompiled plans to achieve current goals and schedules them for execution. The deliberation layer uses an explicit model of the world and a planner to generate new plans. Finally, the multiagent coordination (A.K.A. the modeling) layer contains models of the cognitive states of other agents in the environment (including human agents). These models are used to manage the dependencies between the activities of the agent and those of other agents (e.g. to identify and resolve goal conflicts). Some of these architectures have control mechanisms to decide which layer controls execution at a given time. Depending on what layers are included in the framework, layered architectures are capable of providing a guaranteed level of responsiveness, performing resource-bounded deliberation to cope with exceptional events, as well as providing the flexibility to adapt ongoing plans as required by changes in the environment. Examples of layered architectures include Touring Machines [71], Inhabited Dynamical Systems [143], and InterRRap [156].

Some proposals were made to capture human-like functionalities and capabilities. Since these architectures model structures for performing a wide variety of cognitive tasks, they are often called cognitive architectures. Examples of cognitive architectures are SOAR [129], Homer [240], and OSCAR [171], to name a few. In addition to ascribing to the agents intentional modalities such as beliefs, goals, and intentions, these architectures often attempt to formalize learning, problem-solving, natural language processing and generation, planning, memory, defeasible reasoning, etc. However, cognitive architectures have not dealt with the main problems faced by researchers in agent architecture, namely, the tradeoff between deliberation and reactivity, and handling resource boundedness.

2.3.5 Relation to Agent Theories

In [175], Rao and Georgeff attempt to relate their agent theory in [174] to a simplified version of the PRS interpreter. This "abstract" interpreter operates on a (logically) closed and consistent set of beliefs, goals, and intentions. Also, the belief-base is closed w.r.t. an agent's plans, i.e. the agent knows all her plans, and all possible de-compositions of her plans are pre-computed. Using these, it generates all the options for action in a single cycle. Then it selects an action for execution, executes this action, and updates the agent's mental states. Rao and Georgeff informally discuss how to constrain various procedures called by this interpreter, and thus implement some of the basic axioms of their theory in this architecture. This includes axioms that relate various mental states (such as belief-goal compatibility), and axioms that model various forms of commitment (e.g. blind commitment). They also have an axiom which is similar to Cohen and Levesque's 'no infinite deferral' assumption (i.e. that all intentions must be eventually dropped). Unfortunately, they do not consider axioms related to agents' abilities required to achieve goals, and axioms that deal with rationality (e.g. that agents should not adopt plans that are very unlikely to achieve a goal). They then present a "practical" interpreter that is similar to the PRS interpreter, in the sense that it operates on a knowledge-base of explicit (and grounded) beliefs and goals that is not closed under logical consequence, and that it computes the decomposition of the adopted plans over an arbitrary number of cycles of the interpreter. At every step of the interpreter, in response to an event, the option generator iterates through the plan library and returns the plans whose invocation condition matches this event and whose context condition follows from the agent's beliefs. The deliberator then utilizes meta-level rules to decide which of these options should be selected. In the next step, one of the intentions is executed. Like in PRS, the execution of an intention may involve triggering of another event, or execution of a primitive action. While Rao and Georgeff acknowledge that this practical interpreter does not obey all the axioms of their agent theory, no suggestions for revising the axioms were given. They did however hint that under certain circumstances, namely when no external events occur during the execution of a goal, the practical interpreter behaves like the abstract one, and satisfies these axioms. However, it is not clear that this is the case. For instance, there is no way of preventing the adoption of a plan that is inconsistent with the agent's adopted intentions, since no lookahead mechanism is incorporated in this practical interpreter. Thus the axiom which states that the agent's intentions should be consistent clearly does not follow from this interpreter.

2.4 Agent Programming Languages

The beginning of the current interest in agent programming languages (APLs, henceforth) might be attributed to Shoham's proposal of Agent-Oriented Programming (AOP) [204, 205], as a 'new programming paradigm based on a societal view of computation', and as a specialization of object-oriented programming. The key idea of this AOP paradigm is to use mentalistic and intentional notions formalized by agent theorists to design and program agents.

Another front that pushed the concept of agent-oriented programming is Rao and Georgeff's PRS architecture. As we have seen, the key concept in the PRS architecture is that of using events for selecting hierarchically decomposed plans, and thus avoiding planning from scratch. In the following, we will see that many APLs in the literature are based on a simplified version of the PRS architecture.

Thus, most of the APLs in the literature can be classified into two classes, namely *deductive reasoning* languages and *reactive plan execution* languages. While the former was derived from various agent theories, logics, and calculi, the roots of the latter can be traced back to reactive plan execution architectures (viz. PRS and dMARS [123]).

Logic based APLs are usually more expressive and strongly grounded into the underlying logic. The latter means that programs written in these can often be verified easily by theorem proving or model checking. However, this expressiveness and ease of verification usually comes at the cost of computational complexity. Most of these languages also suffer from other significant limitations, such as poor scalability and modularity, and no support for physical distribution of the computation, nor for the integration of external packages and languages. Examples of logic-based APLs include AGENT0 [204, 205] (based on modal and deontic logic), Concurrent METATEM [72, 73] (based on modal and temporal logic), the Golog family [140, 51, 55] (based on the situation calculus), FLUX [222] (based on the fluent calculus), and MINERVA [131, 132] (based on a non-monotonic logic).

In contrast, efficiency and modularity are two areas where the reactive plan execution languages shine. In addition, these languages provide means for encoding control knowledge by using user-defined plan/rule libraries. Nevertheless, most of these PRS-based languages often have limited expressiveness. Examples of reactive plan execution languages include AgentSpeak(L) [172], 3APL [99], and CAN [247].

APLs also differ on how they handle several issues. For instance, some of these incorporate BDI concepts such as beliefs, desires, goals, etc (e.g. AgentSpeak(L), 3APL, PLACA, etc.), while others do not (e.g. Golog and Concurrent METATEM). A few of these languages handle incomplete knowledge and sensing actions (e.g. Golog and FLUX). Some languages allow planning with lookahead; others only allow reactive plan selection (from a user-defined plan library) and execution. Examples of APLs that allow offline planning include Golog and CAN-PLAN [185]. Some of these languages allow modeled exogenous actions (e.g. ConGolog), have constructs to support communication (e.g. PLACA, Jason), support multiple agents (e.g. JACK), or provide a programming logic on top of the associated programming language to specify agent properties, such as liveness and safety properties (e.g. GOAL, Dribble).

Besides these two classes of APLs that I have identified, there has also been work on purely behavior-based or reactive languages [26, 27, 3, 145]. However, these are closer to agent architectures than APLs. Also, researchers have developed various agent-oriented software engineering methodologies (e.g. Prometheus [159] and Gaia [253]) and tools. Surveys of these can be found in [12] and [13].

In the following, I review work on APLs in our two classes. Later in Section 2.5, I focus on more recent work on agent programming languages that incorporate declarative goals.

2.4.1 Logic-Based/Deductive Reasoning Languages

AOP, AGENT0, PLACA

Shoham [205] identifies three essential components of an AOP language: a theory for defining the mental state of agents; an interpreted programming language for programming agents, whose semantics must be faithful to the theory; and an ‘agentification’ process, which wraps components of physical systems into agents. He acknowledges that the agentifier is not necessary for systems designed with AOP in mind. However, he envisions applying the AOP framework even to ordinary devices, such as watches and cameras, for which the agentification process is required.

AOP incorporates a quantified multi-modal logic with direct reference to time-points. The theory contains three modalities, namely, belief, commitment (also referred to as choice or decision), and ability. Commitment is a derived operator, and defined in terms of *obligation to oneself*. An example of a formula of the logic is as follows: $CAN_a^7\ open(safe)^9 \supset B_b^7\ CAN_a^7\ open(safe)^9$. This says that if at time 7, agent a can ensure that she is able to open a safe at time 9, then at time 7, agent b believes this. Unfortunately, AOP does not include a formal semantics for this modal logic.

Shoham's first attempt at an AOP language resulted in the AGENT0 programming language [205]. In AGENT0, an agent is specified in terms of a set of initial beliefs and commitments, a set of (fixed) capabilities, and a set of *commitment rules*. The set of commitment rules is a key component in AGENT0, and it determines how the agent acts. A commitment rule associates a message condition and a mental condition with an action. If the message conditions match an incoming message and the mental conditions are true in the agent's current mental state, then the corresponding commitment rule fires, and as a result, the agent becomes committed to the associated action. Commitment to an action in AGENT0 amounts to no more than scheduling an action. The AGENT0 interpreter maintains a database of committed to actions and their scheduled times, and when the appropriate time arrives, the action is executed. Along with *private* actions (i.e. internally executed subroutines), AGENT0 also provides *communicative* actions in the spirit of speech-act theory [4, 191, 40]. The basic loop of the AGENT0 interpreter consists of two steps: in the first step, it reads the current incoming message and updates the mental state (i.e. the agent's beliefs and commitments) by applying *all* applicable commitment rules; in step 2, it executes the commitments for the current time, possibly further updating the mental state.

In her 1993 Doctoral thesis [223], Thomas introduces the PLAnning Communicating Agents (PLACA) language as a more refined implementation of AGENT0. PLACA addresses a severe drawback to AGENT0, namely the inability of agents to plan and to communicate their *declarative* goals (via requests). The overall structure of PLACA is very similar to that of AGENT0. To handle planning, the initial mental state now also contains a *declarative* motivational intention-base, and a *procedural* plan-base. Also, commitment rules are now replaced with *mental-change rules*, each of which associates a set of mental state changes with a set of message conditions/mental conditions/(outgoing) message-list. At every tick of the global clock, these mental change rules are used to update the agent's declarative intentions. Plans on the other hand are fed to the system using an external planner, described as a black box in [224], which is responsible for updating the plan-base at every tick of the clock. For longer deliberations, the planner may request the 'mental-rule checker' module to skip some cycles (while queuing the incoming messages) and allow uninterrupted planning. The architecture also utilizes a separate executor module that is responsible for sending outgoing messages, and for executing the scheduled actions when the time has come.[4] Thus, PLACA separates deliberation about which intentions to adopt from considerations of means of achieving the adopted intentions. While the agent program is used to formalize the former, the latter is modeled using a black-box external planner and not properly fleshed out (i.e. mostly unspecified) by the framework.

In [219], Tan and Weihmayer discuss an AOP-based framework for cooperative problem solving that integrates AGENT0 and the state-space planner PRODIGY [150]. The major difference between PLACA and this framework is that in the latter, planning occurs directly as a result of the firing of commitment rules, and thus is not interruptible. Therefore, planning in this account behaves like a single primitive action. Also, since many rules may fire during a cycle, several planning processes may be triggered, which is computationally demanding and may hamper reactivity.

All of these languages were only intended as prototypes. Thus, various simplifying assumptions were incorporated. For example, AGENT0 lacks a formal semantics. Also, agents can only commit to primitive actions that can be directly executed. AGENT0 and PLACA both assume a global clock. Although PLACA includes declarative goals that trigger planning, it does not formally specify how plan generation and commitment to plans are handled.

[4]Recall that every action is dated, i.e. has a time-stamp associated with it.

Concurrent METATEM and its BDI-Extensions

A problem with AGENT0 and PLACA is that no formal semantics for agent program execution is provided. Also, the execution of these languages cannot be said to truly execute the associated logic. A desirable property of any APL semantics is that it should be strongly coupled to the underlying (BDI) logic. In other words, the program execution semantics should satisfy the underlying logic. This ensures that these two are compatible with each-other; for instance, if the underlying theory sanctions that an agent is (physically and mentally) able to perform some action in some situation, then it is only intuitive that the APL execution semantics agrees with this and that one could derive that there is a legal transition of the agent program with this action in that situation. The Concurrent METATEM language proposed by Fisher and Barringer [74], and its extensions [72, 73] attempt to address these issues.

A Concurrent METATEM system is a collection of concurrently executing objects, whose behavior is specified directly using an executable temporal logic. These objects can communicate via asynchronous broadcast message passing. Each object is specified using an object-interface and a set of executable temporal rules. An object-interface identifies the messages that an object can recognize, together with the messages that it can produce. The temporal rules associated with the objects form the bulk of a Concurrent METATEM program. These have the following general form: 'past and present formula' $\supset$ 'present and future formula', and are assumed to hold at all time-points. In Concurrent METATEM, an object's specification is directly executed to generate its behavior. The execution of an object involves iteratively constructing a model from the corresponding temporal formulae, in the presence of input from the program's environment. Starting from the initial state, at each step the program rules are consulted to check which of the rules have antecedents that are satisfied by the partial model constructed so far. The consequents of all such rules are collected together. These consequents represent constraints on present and future properties of this model, and these along with any outstanding constraints generated in some previous steps are used to construct the current state. Any outstanding constraints are passed to the next step. If at any point, a contradiction is generated, the system may backtrack (i.e. undo some actions) to a previous choice-point and attempt to construct a model for the program in a different way, giving up indicating an execution failure when no choice-points are remaining. Note that, sources of non-determinism (i.e. choice-points) include the execution of a rule whose consequent contains a disjunction or an $\Diamond$-formula, i.e. the execution of eventual satisfaction of some formula.

To summarize, Concurrent METATEM differs from other languages in that it based on a satisfiability point of view. As mentioned, an advantage of Concurrent METATEM is that in this framework, the theory and the programming language are strongly coupled. Also, the semantics of program execution is closely related to the semantics of the associated temporal logic. Thus, verifying agents' properties specified in the language is a viable proposition. In [73], Fisher uses a series of examples to demonstrate that Concurrent METATEM can be used to specify intelligent objects that exhibit an interesting range of behaviors, which include cooperation and competition, negotiation, obeying safety and liveness constraints, and so forth. Also it can be used to specify groups of objects (societies) and hierarchical problem solving objects. Unfortunately, Concurrent METATEM has some limitations. For instance, recall that building a model may require backtracking; it seems that this is not always possible, e.g., in some domains, the effects of actions may not be reversible. Also, like satisfiability, it requires complete knowledge.

Recently, Fisher and Ghidini [75] extended Concurrent METATEM by incorporating the notions of belief, ability, and the "motivational" operator *confidence*. Agents' beliefs are represented using a (KD45) *multi-context logic* [90]. Multi-context logic is a formal framework for modular representation of (nested) beliefs of multiple agents, and is based on the notion of belief contexts. A belief context is a representation of a collection of beliefs that an agent ascribes to herself and to other agents. For example, an agent i may have some beliefs about the world; in addition she may have some beliefs about another agent j, beliefs about j's beliefs about another agent k, etc. In a multi-context logic, each of these sets of beliefs is represented using a distinct formal language, and the interpretation of such a language is local to the belief context it is associated with. A context structure in a multi-context logic contains an infinite tree (where the root of the tree represents the belief context of the agent whose belief is under consideration), and allows one to represent arbitrarily nested beliefs. Although distinct, the contents of different belief contexts can be related. For instance, an obvious relation is the following: if a sentence of the form P is in the belief context for i's beliefs about j (i.e. in the context ij), then a sentence of the form "j believes that P" is in the belief context for i. Another relation says that a sentence of the form P is in ij, only if a sentence of the form "j believes that P" is in the context i. Depending on the relations among different contexts, one can model agents having different reasoning capabilities. As we can see from the above description, the key feature of belief-contexts is modularity. Note that, the extended Concurrent METATEM incorporates the appropriate relations so that agents' beliefs are KD45.

An agent's abilities are constant over time. The semantics of the ability operator is formalized using a function r that associates a belief context to a (fixed) set of formulae. Thus, e.g. if r associates $\{\phi\}$ with the context for agent i, then i has the ability that ϕ; on the other hand if r associates $\{\phi, \psi\}$ with the context for agent k's belief about agent j's belief about agent i, then k believes that j beliefs that i has the ability to achieve ϕ and ψ. Confidence in ϕ, which is a derived attitude, is defined as *believing that ϕ will eventually happen* (i.e. $B_i\Diamond\phi$). As in the original Concurrent METATEM, agents are specified using an agent-interface and a set of temporal rules in this framework. However, rules are now allowed to have modal operators; the rules must be in normal forms that only allow present and future temporal operators.

The key idea in this framework is that the language provides a mechanism for deriving concrete specification of motivations from more abstract ones. Consider the interaction between the following temporal goal formulae (in descending order in terms of abstractness): $B_i\Diamond\phi$, $B_i\Diamond_j\phi$, $\Diamond\phi$, and ϕ, that is, the agent i is confident about ϕ, i is confident about ϕ and believes that j is responsible for bringing about ϕ, ϕ will eventually hold, and ϕ is currently true, respectively. Fisher and Ghidini argue that by providing rules that can be used to derive a more concrete goal formula from one of these abstract goal formulae, we are essentially specifying a *rational* agent. One such rule is as follows:

$$(B_i\Diamond\phi \wedge A_i\phi) \supset \Diamond\phi.$$

This says that, if agent i is confident about ϕ, and is able to achieve ϕ, then ϕ eventually holds. Another example of this is that an agent may move from 'confidence' (i.e. $B_i\Diamond\phi$) to 'confidence in another agent' (i.e. $B_i\Diamond_j\phi$, where $i \neq j$), through deduction or communication. Again, moving from $\Diamond\phi$ to ϕ is essentially a matter of scheduling.

In this framework, various rules can be tailored and various constraints on B_i can be imposed to specify realism, strong-realism, and weak-realism properties (see Section 2). Also, rules can be used to implement sensing actions. Thus, this extension of Concurrent METATEM brings it closer to other BDI languages. Even more recently, Fisher et al. [77, 76] extended Concurrent METATEM to include groups of agents and show how agents can be efficiently organized to collectively solve problems.

One problem with these frameworks is that in these languages, the programmer needs to explicitly specify the behavior of the agents using temporal rules. Thus although verification of agent properties is relatively straightforward, programming even simple agents puts a heavy burden on the programmer. The examples in [73] and [75] show this. Also, while it is possible to write chaining rules (i.e. rules whose consequent fires other rules) in this language, these rules do not exactly correspond to hierarchically decomposed procedures/plans. Finally, this model of agent programming is problematic in the sense that although specifying rational actions (or in this case, rational temporal rules) is left to the programmer, there is nothing to prevent the programmer from writing inconsistent or non-terminating sets of rules.

Golog, ConGolog, and IndiGolog

Another style of agent programming is developed in the logic programming-based Golog family [140, 51, 55]. In Golog, the programmer first declaratively specifies the agent's knowledge of the dynamics of the world (i.e. preconditions and effects of actions), and the initial state of the world in a situation calculus dialect, a first-order language for dynamic domains which incorporates a solution to the frame problem due to Reiter [176, 178]. Then various Golog constructs, such as primitive actions, testing for a condition, sequences, non-deterministic branch, loops, etc. are used to write programs. Given a program and the domain axioms, the interpreter attempts to prove that the program has a terminating execution starting in the initial situation. A sequence of actions for executing the program is uniquely identified by the terminating situation. Once an action sequence is found, the agent executes the program, by executing one action at a time. Thus, Golog redefines the planning problem of 'looking for a legal sequence of actions to achieve some goal' as the problem of 'searching for a legal sequence of actions that amount to a legal execution of the high-level program'. The program can encode search control knowledge.

In contrast to most other logic-based APLs where the agent's state must be explicitly updated by the executing program, Golog and its successors employ an automatic state update mechanism using their background action theory for the domain. Also, unlike other APLs where the designer specifies the agent's behavior using some form of rules, Golog has the programmer specify a high-level non-deterministic program, and the underlying interpreter's task is to search for an execution of this program.

ConGolog extends Golog to include concurrency by providing constructs for non-deterministic iteration, concurrency (with and without priority), and interrupts, which makes it easier to write reactive programs. ConGolog also replaces the *evaluation semantics* of Golog with a *transition semantics*, since a single-step semantics is better suited for concurrency.

The ConGolog interpreter proves that some branch of the non-deterministic program yields a terminating state of the program, and thus resolves the non-determinism in an off-line style using lookahead planning. This offline planning cannot handle dynamically changing worlds too well, especially when sensing and exogenous actions are involved. For instance, consider the following program: $(a|b); sense_q;$ if q then Δ_1 else Δ_2 endIf; ϕ?, which says that the agent should first nondeterministically choose between a and b and execute it, then sense the truth-value of q, and based on this value, should execute either Δ_1 or Δ_2, terminating successfully if the test ϕ? succeeds. Note that, an offline interpreter for this program cannot commit to either a or b in advance, since it does not know which of these will ensure that ϕ, and thus cannot use the sensing action to determine whether q would hold after the action. The only option available to the interpreter is to check if one of the actions a or b will lead to ϕ for *both* values of q. Thus, early occurrences of non-deterministic choices can result in unacceptable delay. The situation gets even worse when loops are involved. To deal with this, the language IndiGolog [55] was proposed. In IndiGolog, programs are executed incrementally to allow for interleaved action, planning, sensing, and exogenous events. Informally, the semantics of incremental execution is as follows: an incremental execution of a program finds a next possible action, executes it in the real world, obtains the sensing results afterwards, and repeats this cycle until the program is finished. Since this makes it possible to quickly execute the actions without much deliberation, this approach is suitable for realistic changing worlds. However, since the program may contain non-deterministic choice points, some lookahead mechanism is required to avoid unsuccessful (dead end) executions. For this reason, a search operator Σ is introduced in IndiGolog. Intuitively, $\Sigma(prog)$ selects from all possible transitions of $prog$ one for which there exists a sequence of further transitions that leads to a terminating configuration. The IndiGolog interpreter automatically monitors the execution of a plan generated by such a search block, and re-plans when the current plan fails or is no longer appropriate due to changes in the environment. IndiGolog also supports a simple form of contingent planning, where the dynamic environment is modeled as a simple deterministic reactive program [135].

Thus, IndiGolog is a powerful language that is able to handle incomplete knowledge, sensing, and exogenous actions, and allows the specification of prompt reactive behavior as well as user-controlled deliberation. Note however that, the standard implementation of IndiGolog makes a *dynamic closed-world assumption*, i.e. it assumes that a program has sufficient knowledge to evaluate a query/test by the time it is evaluated, and if initially the answer to the query is not known, sensing actions will be executed before the query is made. Thus it is assumed that the on-line interpreter has complete knowledge of the relevant fluents by the time the query is evaluated. To avoid this limitation, an extension of the Golog formula evaluator was presented in [238], where the evaluator keeps track of the *possible values* that functional fluents can take in a given history (i.e. a situation along with the sensing results obtained so far). A fluent is said to be known at some history h iff it has only one possible value at h. Note that this only handles limited forms of incomplete knowledge, namely, not knowing the value of a fluent; general disjunctive knowledge is not handled. This work is still at an early stage, and the issue of how to deal with efficient belief update in the presence of incomplete knowledge is still not completely clear.

FLUX

One disadvantage of IndiGolog and some other logic-based non-BDI languages is that the knowledge of the current state is represented indirectly using histories, i.e. via the initial conditions and the actions that the agent has performed so far. A consequence of this is that each time the agent needs to evaluate a condition, she has to consider the entire history of actions and perform regression. Thus these languages do not handle long running agents efficiently. The fluent calculus-based high-level programming language FLUX (FLUent eXecutor) [222] attempts to solve this problem using an explicit state representation and progressing it when an action is performed. FLUX incorporates an implementation of the fluent calculus, a language for reasoning about actions. The fluent calculus provides a basic solution to the frame problem using the concept of state update axioms. It also addresses a variety of other aspects such as, ramifications, qualifications, nondeterministic and concurrent actions, continuous change, and noisy sensors and effectors.

A FLUX agent program consists of three parts, namely a background theory that encodes the agent's internal model of the environment, a kernel that provides the agent with cognitive abilities to reason about her actions and acquired sensor data, and a strategy that specifies the agent's task oriented behavior. The types of incomplete knowledge FLUX can encode are restricted. The underlying inference engine of FLUX is sound but incomplete. However, it can be shown that reasoning in FLUX is linear in the size of the internal state representation. Thus FLUX scales up well to long-term control. FLUX allows the use of full expressive power of logic programming in defining strategies. It also facilitates formal proofs of correctness of strategies.

MINERVA

The MINERVA agent programming language [131, 132] utilizes logic programming and several non-monotonic knowledge representation and reasoning mechanisms to provide a common multiagent framework. A MINERVA agent consists of several specialized concurrently running sub-agents performing various tasks. These agents can read and manipulate a common knowledge base specified in the Multi-dimensional Dynamic Logic Programming language (MDLP). In MINERVA, agents are driven by an observation-deliberation-action cycle. The behavior of these agents is specified in the Knowledge and Behavior Update Language (KABUL).

MDLP provides an extension of Answer Set Programming (ASP). In MDLP, an agent's knowledge is represented using logic programs arranged in an acyclic digraph, in which the vertices are sets of logic programs, and edges represent the relationship between these programs. MDLP benefits from the advantages of ASP, such as default negation, which can be used to deal with incomplete knowledge. Also, it can be used to represent the evolution of knowledge, and preferences.

KABUL is a logic programming language that can be used to specify updates to knowledge bases, and to the KABUL program itself. A KABUL program consists of a set of condition-action rules that encode the agent's behavior. Since actions in KABUL can update both the knowledge base represented in MDLP and the KABUL program itself, it can be used to specify agents that change their behavior over time. Conditions in these rules can refer to external observations, the epistemic state of the agent, as well as to occurrences of exogenous actions. Since no external stimuli are needed to trigger the behavior of the agent, KABUL can be used to specify both reactive and proactive behavior.

Other Logic-Based APLs

There has been work on various other logic-based APLs. Those that gained some popularity include APRIL [146], the deontic logic-based IMPACT [62, 63], the dynamic logic-based Dylog [6], the linear logic-based ε_{hhf} [57], the ambient calculus-based CLAIM [193], the Horn Clauses and Least Herbrand Models based DALI [41], and ReSpecT [158]. Surveys of some of these languages can be found in [144, 12, 13]. In the next section, I discuss some of the reactive plan execution languages.

2.4.2 Reactive Plan Execution Languages

AgentSpeak and its Variants and Implementations (Jason, JACK, Jadex)

As discussed in Section 2, there has been much work on agent theories, and current theories are quite mature and well established. However, there is a large gap between agent theories and BDI APLs.[5] In [172], Rao introduced the AgentSpeak(L) language as an attempt to show a one-to-one correspondence between the model theory, proof theory, and the abstract interpreter of this language. Here, model theory, proof theory, and abstract interpreter refers to the underlying BDI theory, the formal semantics of the programming language (often specified using a transition semantics, as discussed below), and the implemented interpreter for the language, respectively. Rao argues that there is a better chance of unifying theory and practice by taking a simple specification language as the execution model of an agent, and then ascribing mental attitudes to this agent. To this end, he used the Procedural Reasoning System (PRS) and its more recent incarnation the distributed Multi-Agent Reasoning System (dMARS) [123] as a starting point for the AgentSpeak(L) implemented system. To establish the link between agent theories and APLs, it is necessary to have a way of deriving the formal

[5]Here, BDI APLs refers to APLs that incorporate concepts such as beliefs, desires, goals, and intentions. Also, note that the original proposal of Concurrent METATEM, the Golog family, and many other declarative APLs are tightly coupled to the associated logic/theory. However, most of these languages are not *per se* typical BDI-languages.

semantics of program execution from the underlying agent logic. To do this, Rao first defined program states of an APL using agent configurations. An agent configuration consists of a sentential description of an agent's beliefs and her motivational states, derived from associated components of the underlying agent theory. Intentions in a configuration are represented procedurally as in PRS. He then defined program execution or agent behavior as transitions from configuration to configuration. These transitions are guided by a set of transition rules (A.K.A. proof rules), which specify how an agent configuration and its components may change as a result of executing an action, and what actions can be executed. AgentSpeak(L) is based on reactive plan execution architectures, where rather than deliberating on which action to execute next, the agents utilize the changes in the environment to decide which given hierarchical plan should be adopted, and how to decompose and execute this hierarchical plan.

An AgentSpeak(L) agent consists of a belief-base, a set of plans, and a set of intentions. When an agent acquires a new goal, or notices a change in her environment, she may trigger additions or deletions to her goals or beliefs. These events are referred as *triggering events*. Agents in this framework respond to triggering events. Plans in AgentSpeak(L) are rules of the form: $(e : cc \leftarrow p)$. Intuitively, this says that in response to the event e, the agent should adopt the plan-body/intention that p, provided that the context condition cc follows from her belief. The plan-body can be built using sequences of goals and actions. Goals in AgentSpeak(L) are of two types, namely *achievement* goals, and *test* goals. Achievement goals are an abstraction mechanism, and serve the same purpose as procedure calls in imperative programming. In other words, the execution of an achievement goal triggers an event, and as a result, the agent adopts the appropriate plan as her intention, just as the execution of a procedure in imperative programming amounts to the execution of the procedure body. These plans may in turn include achievement goals in them, and in that case when executed, they will trigger the adoption of other plans. Thus achievement goals and plans together provide a mechanism for event-invoked hierarchical decomposition of goals. AgentSpeak(L) also uses these plans to revise agents' beliefs and goals by generating primitive addition/deletion events. Test goals involve testing the belief-base and may be used to compute variable bindings. Intentions in AgentSpeak(L) are stacks of partially instantiated plans. At any time, the agent may have multiple intentions. Initially, each of these intention stacks contain only one element, namely, the plan that was adopted in response to an *external event* (i.e. due to a change in the external environment). The execution of these intentions may involve executing an achievement goal, which triggers the adoption of new intentions. In that case, this new intention is pushed on top of the intention stack that triggered the adoption of this intention.

The overall control flow of the system is determined by the AgentSpeak(L) interpreter, and goes as follows. The interpreter uses a given selection function $S_{\mathcal{E}}$ to determine which pending event to process next. Then it computes the *relevant plans* by checking the plans whose associated event matches (i.e. can be unified) with this event. From these plans, it then computes the set of *applicable plans* by checking whether an instance of the context condition follows from the agent's beliefs. Another selection function $S_{\mathcal{O}}$ is used by the interpreter to choose one of the applicable plans, and this plan is then added to the intention base. The interpreter uses a third selection function $S_{\mathcal{I}}$ to decide which of these intentions should be executed next. As discussed above, these adopted intentions can in turn post so called internal events. Internal events are processed similarly to regular or external events. However, rather than adding the selected applicable plan to the intention base, it is now pushed on top of the intention-stack that posted this event. Only the plans that are on top of an intention stack are considered for execution, and only one of them are executed in each cycle.

For instance, suppose that an agent has the following plans in her plan-base:

$$+!\phi_1 : true \leftarrow !\psi_a; a_1; a_2,$$

$$+!\psi_a : true \leftarrow a_3; a_4.$$

Here $+$ and $!\phi$ refers to an addition event and an achievement goal ϕ, respectively. The first plan says that in response to the event where the agent acquires the goal to achieve ϕ_1, she should adopt the plan to achieve the goal ψ_a first, and then execute the primitive action a_1 followed by a_2. Similarly, the second plan says that in response to the event where the agent acquires the goal to achieve ψ_a, she should adopt the plan to execute the primitive action a_3 followed by a_4. For simplicity, I assume that both of these rules have a $true$ context condition. Now, suppose that the agent acquires the goal to achieve ϕ_1 through some external event (such as, via a $request$ action), and that $S_{\mathcal{E}}$ chooses to process this event next. Since the first rule's trigger condition matches (unifies) with this event, the context condition trivially follows, and there is only one applicable rule (i.e. $S_{\mathcal{O}}$ returns this plan), she will adopt this plan as her intention. Thus, a new intention $[+!\phi_1 : true \leftarrow !\psi_a; a_1; a_2]$ will be added to her intention base (let's call this intention i_1). Similarly, each time she acquires an intention due to an external event, a new intention will be added to the intention-base. Now, suppose that $S_{\mathcal{I}}$ chooses to execute i_1. Since executing the first action of i_1 involves executing the achievement goal ψ_a, it will generate the event that $+!\psi_a$, and this event will be added to the event queue. In the next cycle, suppose that $S_{\mathcal{E}}$ chooses to process the event $+!\psi_a$. In response to this event and after unifying the trigger condition and verifying the context condition, the agent will adopt the intention that $[+!\psi_a : true \leftarrow a_3; a_4]$, since there is only one applicable rule. But, since this event was generated due to the execution of another intention, namely i_1, it will be pushed on top of the stack for

i_1 rather than being added as a new intention. Recall that, when deciding on which intention to execute next, only the plans that are on top of the intention stack are considered. This ensures that the agent will execute $a_3; a_4$ before executing $a_1; a_2$.

An AgentSpeak(L) agent is specified by a tuple $\langle E, B, P, I, A, S_{\mathcal{E}}, S_{\mathcal{O}}, S_{\mathcal{I}} \rangle$, where E is a set of possible events, B is a set of possible base beliefs (defined using a ground set of atoms), P is a set of possible plans, I is a set of possible intentions, A is a set of actions (denoting the possible set of actions that the agent has performed so far), and $S_{\mathcal{E}}, S_{\mathcal{O}}$, and $S_{\mathcal{I}}$ are the three selection functions. An AgentSpeak(L) agent can have a number of executions defined in terms of the configurations reachable from the initial configuration. A BDI configuration is a tuple of $\langle E_i, B_i, I_i, A_i, i \rangle$, where $E_i \subseteq E, B_i \subseteq B, I_i \subseteq I, A_i \subseteq A$, and i is the label of the configuration. Note that, an agent's plans are considered to be static, and thus are not included in a configuration. The semantics of the AgentSpeak(L) programming language is defined using a labeled BDI transition system that specifies how agents can evolve from one configuration to another. A BDI transition system is a pair $\langle \Gamma, \vdash \rangle$, where Γ is a set of BDI configurations, and $\vdash$ is a binary transition relation $\Gamma \times \Gamma$ defined using a set of proof (transition) rules. A BDI derivation/execution is defined to be a (possibly infinite) sequence of BDI configurations $\gamma_0, \gamma_1, \cdots, \gamma_i, \cdots$ such that for all i, $\gamma_i \in \Gamma$, γ_0 is the initial configuration, and for any consecutive pair of configurations (γ_j, γ_{j+1}),

$\gamma_j \vdash \gamma_{j+1}$. For the AgentSpeak(L) programming logic, the notion of ‘refutation’ is defined in terms of an intention: it starts when the agent adopts an intention, and ends when her intention stack becomes empty. Rao argued that this programming logic can be used to formally prove certain properties about an agent’s behavior, such as safety and liveness of the agent system. Also, there is an one-to-one correspondence between the AgentSpeak(L) interpreter and AgentSpeak(L) transition rules.

Since Rao’s original proposal [172], other researchers have proposed various extensions and implementations of the AgentSpeak(L) language. In [60], d’Inverno and Luck use the **Z** specification language to formally specify a complete abstract interpreter for AgentSpeak(L) similar to that given to dMARS [59]. In [153], the operational semantics of AgentSpeak(L) is specified using a more standard Plotkin-style structural approach [168]. The three major implementations of AgentSpeak(L) include JACK [105], Jason [16, 13], and Jadex [22, 170]. In the following, I briefly discuss these, and also point out the extensions provided by these implementations.

The Java Agent Compiler and Kernel (JACK) Intelligent Agents$^{\text{TM}}$ framework [105] is a commercial agent-oriented programming tool developed by Agent-Oriented Software Pty. Ltd. Unlike Jason and Jadex, JACK does not provide a logic-based language to specify agents’ beliefs and intentions; rather, it uses an extension of Java to implement some features of the underlying logic, such as logical variables, events, beliefs, and plans. One of the design goals of JACK was to provide developers with a robust, stable, light-weight product that can be used to develop components of larger environments, such as legacy software systems. To this end, JACK provides three extensions of Java. It extends the Java syntax to include BDI-related keywords, declaration of attributes, and statements. It provides a compiler that compiles these BDI syntactic additions into pure Java classes that can be used by other ordinary Java code. Finally, it incorporates a set of kernel classes that provide the required runtime support to the generated code.

From a functionality point of view, JACK incorporates six components, namely, agent, capability, belief-set, view, event, and plan. The agent construct defines the behavior of an agent by including the capabilities an agent has, the types of events she responds to, and the plans she uses to achieve her goals. The capability concept structures the reasoning capabilities of agents into clusters. Capabilities are built up from events, beliefs, plans, Java code, etc. They simplify agent design by allowing code reuse and encapsulate agent functionality. This allows the agent architect to build up a library of capabilities over time, and create an agent promptly by simply plugging in the required capabilities. JACK uses a generic relational model to represent the agents' beliefs. A belief-set consists of relations of the following form: $relationName(key_1, key_2, \cdots, data_1, data_2, \cdots)$. Each relation can be identified by the relation name and any number of keys. The data fields are used to encode the attributes of a relation. Elements of a belief-set can be retrieved using unification as in Prolog. The view construct is central to JACK's data modeling capability. It is built up from multiple belief-sets or arbitrary Java data structures, and allows general purpose queries to be made about an underlying data model. Events and plans are similar to AgentSpeak(L).

On top of these architecture independent facilities, JACK provides a set of plug-in components that address the requirements for specific agent architectures. Currently it supports two such plug-ins, namely a PRS/dMARS-based BDI interpreter, and a plug-in for building teams of agents, called *SimpleTeam*. The underlying concept behind SimpleTeam is that it allows the programmer to specify a high-level view of the coordinated behavior of a team, and then map this high-level view to the individual activities of the participating agents. JACK also include a graphical agent development environment, a debugger, and an object modeling toolkit to support object transport and interaction with existing applications in Java and C++. JACK has been used to develop commercial applications, such as decision support and defense simulation for analysis applications.

Recently, Bordini and Hübner developed a Java-based open source practical interpreter for AgentSpeak(L) called Jason [16, 13], that incorporates an extension of the AgentSpeak(L) semantics to support speech-act based inter-agent communication [154]. To allow for both closed-world and open-world belief-bases, Jason also allows the use of strong negation in beliefs and plans of agents. Jason also has provision for handling plan failure. This is done by generating a "deletion of goal" event when some action fails (or when some subgoal fails as a result of the absence of an applicable plan for achieving that subgoal), and then handling that goal deletion event by searching for an applicable plan in the rule library, eventually executing one of these plans. However, for this to work properly, the user must define appropriate responses to various plan failures. In case no such plans are defined, the Jason interpreter just drops the intention altogether. In Jason, it is possible to design plan failure handling rules in a way so that plan failure is propagated up the intention stack. To do this, Jason provides some special actions called *internal* actions. Internal actions only change an agent's mental states and have no effect on the world. These actions can be used both in the context and the body of plans. For instance, if the designer under certain conditions wants to propagate a plan failure up the intention stack, he/she can write a plan failure handling rule whose context condition encodes these conditions, and whose body has an internal action that removes the appropriate intentions.

In [11], Bordini et al. aim to provide a more practical programming language by specifying various selection functions of AgentSpeak(L). In particular, they provide specifications of relations between plans and quantitative criteria (such as the quality, duration, and cost of plans, and the deadlines specified for them) for their execution, and then use efficient decision-theoretic task scheduling to automatically guide the choices made by an agent's intention selection function. The design of the Jason APL allows atomic formulae and plans to have *annotations* [241, 154] which can be used by various user-defined selection functions.[6] For instance, annotations within the belief base can be used to register the source of the associated information, and can later be utilized by the (user-defined) belief update function. Annotations in action expressions can be used to implement sophisticated applicable plan selection and efficient decision theoretic intention selection functions. Another interesting feature of Jason is that it can be easily configured to run on various hosts. This is done using an agent communication infrastructure called SACI [106].

Another implemented agent programming framework is the Jadex software framework [22, 170]. Jadex is implemented as an agent reasoning layer that sits on top of the middleware agent infrastructure JADE [10]. The reasoning engine of Jadex is similar to that of AgentSpeak(L). Jadex utilizes both declarative and procedural approaches to define various components of an agent. It uses Java to procedurally define plan bodies (i.e. actions), and the XML language to declaratively define all other men-

[6]However, these sophisticated selection functions are not yet provided with the current distribution of Jason.

tal attitudes. While Jadex provides a semantics for declarative goals (as discussed in the next section), the current implementation does not utilize these goals. The Jadex toolkit comes with a graphical debugger and various other tools to help the application developer. Jadex has been used in various applications such as simulation, scheduling, and mobile computing.

Bordini and Moreira [17] use the transition semantics in [153] to prove various BDI properties of AgentSpeak(L) agents, including the intention-belief inconsistency principle. There has also been some work on automatic verification of AgentSpeak(L)-like programs. In [18, 14, 15], Bordini et al. introduce a toolkit called CASP which can be used to translate a simplified version of AgentSpeak(L) into the input language of existing model checkers for linear temporal logic, such as SPIN [103] and JPF2 [242].

An Abstract Agent Programming Language (3APL)

Another major PRS-based agent programming language that can be found in the literature is An Abstract Agent Programming Language (3APL) [99]. Like AgentSpeak(L), 3APL utilizes a procedural notion of goals/intentions, and specifies a static set of rules, now called *Practical Reasoning* rules or PR-rules, which operate on these goals. However, 3APL differs from AgentSpeak(L) in various ways and I discuss these differences below.

3APL incorporates the notion of *basic actions*, which are used to specify agents' basic capabilities. These actions are viewed as application dependent mental state transformers in that these change agents' beliefs. The specification of belief updates associated with a basic action is formally represented using a partial function $\mathcal{T}$ that returns the result of updating a belief base by performing an action. Note that $\mathcal{T}$ is a partial function, since the action may not be executable in some belief states. In contrast, recall that in AgentSpeak(L) one uses rules to update beliefs by generating a primitive addition/deletion event which triggers some update rules (treated like any other plan).

A 3APL agent consists of a belief-base, a goal-base, and a set of rules. While AgentSpeak(L) agents respond to events, 3APL agents respond to goals in their evolving goal-base. The concept of event is missing from 3APL. 3APL rules are triggered by conditions on the goals and belief-bases, rather than events. If an agent has reacted to some new goal or belief, she should memorize it so that the relevant rule won't fire twice. In [98], it has been shown that 3APL can bi-simulate AgentSpeak(L).[7] In response to a goal in the goal-base, a 3APL agent searches for a rule whose trigger

[7]The underlying idea for this involves the generation of an event-queue from the goal-base, and the creation of an intention-base from the goal-base and rule-base. However, this technique does not cover the deletion of goal events, and the addition and deletion of belief events (i.e. it only handles the addition of goal events).

condition can be unified with this goal, and whose context-condition follows from her beliefs. The agent then replaces the goal in the goal-base with the body of one such rule. Thus, the agent's goal-base in 3APL evolves over time and works like the intention-base in AgentSpeak(L).

3APL provides a richer set of rules than AgentSpeak(L). Also, the plan-body of a rule can now be constructed using various imperative programming constructs, rather than only using sequences as in AgentSpeak(L). PR-rules can be classified into four types, namely, failure rules, reactive rules, plan rules, and optimization rules. The roles of these rules are as suggested by their names. A typical failure or optimization rule is of the form $\pi_h \leftarrow \phi \mid \pi_b$, where π (possibly with subscripts) denotes a plan/program. This says that if π_h is part of the agent's plan, and she believes that ϕ, then this plan should be replaced by π_b. Note that failure rules with empty bodies can be used to drop a goal (plan). Reactive rules are rules with an empty head (i.e. of the form $\leftarrow \phi \mid \pi$), and state that whenever the agent believes that ϕ, it should adopt the plan/goal that π. These rules are used to create new goals. Finally, a plan rule is of the form $G(\overrightarrow{t}) \leftarrow \phi \mid \pi$, and states that when the agent believes that ϕ, one way of achieving the (atomic) goal $G(\overrightarrow{t})$ is π. In addition to facilitating planning for simple achievement goals, these rules in some sense provides a mechanism for revision and monitoring of goals.

Thus, a 3APL agent is a tuple $\langle \Pi, \sigma, \Gamma \rangle$, where Π is a possible goal-base, σ is a possible belief-base, and Γ is a possible PR-base. The operational semantics of 3APL is provided using a transition semantics. 3APL provides two sets of transition rules, one for specifying the execution of individual plans, and another for specifying the execution of an agent. Plan-level execution rules define what it means to execute a single goal, and include rules for executing basic actions, test goals, sequential goals, non-deterministic choice, and application of PR-rules. Agent-level execution rules, which are defined in terms of these plan-level execution rules, specify what it means for an agent to execute multiple goals in parallel. The overall semantics of 3APL is defined in terms of *computations*. A computation is a finite or infinite sequence of mental states such that the first mental state in this sequence is the initial mental state of the agent, and the successive mental states can be derived using the agent-level transitions.

Another novel feature of 3APL is that it separates the semantic specifications for the agent language, and its control structure, by introducing a distinction between an object-level and meta-level semantics. The control structure at the meta-level specifies which goals should be executed and which rules should be applied next. To this end, 3APL introduces a meta-language that includes some meta-actions and a meta-level transition system. To determine which goals (rules) should be executed (applied, respectively) next, the meta-language assumes that there is a fixed user-defined ordering on goals (rules, respectively). The transitions of the meta-actions are derived in terms of the object-level transitions. For instance, if there is a rule in the object-level transition system that says that a goal g is executable, then a meta-level transition rule selects g for execution provided that g is maximal w.r.t. the ordering on the set of all goals. The overall control structure of 3APL is a specialization of the update-act cycle. In the *planning/application phase*, the ordering on rules and goals is used to determine the strongest applicable reactive, plan, or optimization rule, and if there is such a rule, it is then applied to the agent's plan. After this, in the *filtering phase*, the controller uses the ordering on goals to choose a goal, searches for failure rules applicable on that goal, and applies *all* such rules. Finally, in the *execution phase*, it executes a single step of the chosen goal, provided that the first action of the chosen goal is executable. While AgentSpeak(L) has a similar control mechanism provided via the three selection functions $S_{\mathcal{E}}, S_{\mathcal{O}}$, and $S_{\mathcal{I}}$, the major difference is that (the original proposal of) AgentSpeak(L) [172] does not handle plan failure, and thus the filtering phase of 3APL is omitted from AgentSpeak(L).

2.5 Declarative Goals in Agent Programs

As mentioned, an important concept in the context of agent programming is that of declarative goals. I start the discussion on declarative goals by pointing out the differences in expressiveness between declarative and procedural goals, and the advantages of incorporating these goals in APLs.

2.5.1 Advantages of Declarative Goals

Declarative goals in agent programs are necessary for a variety of reasons. The major difference between declarative and procedural goals lies in the way they express an agent's degree of commitment towards a goal. Being committed to a procedural goal amounts to nothing more than being committed to a plan, i.e. to execute a (possibly infinite) sequence of actions that the procedural goal can be decomposed to. On the other hand, being committed to a declarative goal is much more expressive, and it amounts to being committed to one of all possible plans that achieve the goal. Thus, the difference between them can be viewed as commitment towards some means to an end vs. commitment towards an end. For a static environment, where the agent programmer has a complete model of the world and knows about all possible exogenous actions in advance, it may be possible for him/her to specify an extensive set of hierarchical rules that covers all the ways to achieve some goal (like a policy). However, this may require much effort; also this becomes even harder when the agent designer only has partial knowledge about the domain, and cannot predict all possible interruptions (generally a fixed utility function is assumed). Also, from a technical point of view, procedural goals have limited expressiveness. Procedural goals cannot be combined using logical operators. For instance, even if the agent programmer specifies two procedures for achieving two separate goals P and Q, he/she needs to write a third procedure for the conjunctive goal $(P \wedge Q)$.

Motivations for declarative goals in agent programs can be roughly classified into two categories, namely theoretical and practical motivations. From a theoretical point of view, it has been argued that declarative goals are required in order to bridge the gap between agent theories and APLs [100, 247, 233]. The reason for this is that in agent theories, goals are declarative concepts, and thus the incorporation of these goals is viewed as a necessary prerequisite for bridging this gap. From a practical perspective, various advantages of declarative goals have been pointed out in the literature. In the following, I discuss these advantages.

One reason for using declarative goals is to decouple plan execution and goal achievement. A declarative (achievement) goal represents a state that is to be reached. Declarative goals can be used to decide whether a plan was successful in achieving the associated goal or not. The successful execution of a plan does not necessarily indicate the successful achievement of a goal. Also, failure to execute a plan does not mean that the goal can't be achieved, since there might be another way of achieving the goal, one that is not described by the procedural goal. For instance, consider the following example of a hungry cat (from [247]). Initially, the cat knows that some food has been left on the table, and has the goal of reaching the food. If we are using a procedural representation of goals, one way to define this goal is using the cat's plan that $leapOnChair(chair); fromChairJumpToTable(chair, table)$, i.e. it should first jump on a chair that is close to the table, and then jump from the chair to the table, where the food is located. Suppose, with this goal in mind, the cat leaps on to the chair. At this point, a nearby human, realizing the cat's intention, moves the chair further away from the table. Thus, since the second action is no longer executable, the cat's plan has failed, and since we are using a procedural definition of goal, the cat's goal has also failed. Note that, by using only procedural goals, *the reason for performing the plan* is lost. On the other hand, if we were using a declarative definition of goals, we can use $NextTo(Cat, Food)$ as the goal. In that case, even though the cat's plan fails, it can try to plan again using this goal. Consider another example in the same domain: suppose that the cat was successful in jumping on to the table. Note that, this does not necessarily mean that it was successful in reaching food, since somebody might have removed all the food from the table. These examples illustrate that the success and failure of a plan do not tell us much about the success or failure of the associated goal. Once again, this is especially true in dynamic environments where unexpected events may occur, and it is impossible for the designer to predict all possible interruptions caused by exogenous actions.

Declarative goals can also be used to detect fortuitous achievement of goals. For instance, suppose that some food was left on the chair. So after leaping onto the chair, the cat can detect this, and since its goal to reach the food has been achieved, it can drop the plan to jump to the table. Now, consider why an APL that does not incorporate declarative goals, such as 3APL, cannot detect this. Recall that in 3APL, the agent programmer specifies rules that can be triggered due to the presence of procedural goals in the goal-base. Then the procedural goal is decomposed to/replaced with the rule-body, and eventually executed to achieve the goal. Suppose that the programmer only has partial knowledge about the cat's environment, and wrote the following rule that can be used to achieve the cat's goal:

$$goNextToFood(Cat) \leftarrow true \mid leapOnChair(Chair);$$
$$fromChairJumpToTable(Chair, Table).$$

Assume that the programmer does not know that unusual situations such as one where food is left on the chair can occur. In response to the procedural goal $goNextToFood$ (Cat) in the goal-base, the cat will adopt the plan $leapOnChair(Chair); fromChair-$ $JumpToTable(Chair, Table)$. After executing the first action, the cat is on the chair. However, although food is at its current location, the 3APL-cat not knowing that the state that needs to be reached (i.e. $NextTo(Cat, Food)$) has been reached, will not drop the goal of jumping to the table. Thus, if goals are defined procedurally, the cat still has the plan to jump on the table even though its goal has been satisfied. Note that, while a 3APL PR-rule of the form $(\delta \leftarrow \phi \mid nil)$, which says that if the agent believes that ϕ and has the plan that δ, then she should give up this plan, can be used to detect this, it involves the use of a declarative goal ϕ in the context condition. Also, in order to do this, the agent programmer needs to specify an extensive set of such rules. The reason for this is that an exogenous action could occur at any stage during the execution of the plan, and thus the dropping of the current plan due to early achievement of the associated goal should be considered for all possible configurations of the plan. For instance, even for a simple plan $a_1; a_2; a_3$ that includes three primitive actions in sequence, the programmer needs to specify the three following rules: $(a_1; a_2; a_3 \leftarrow \phi \mid nil)$, $(a_2; a_3 \leftarrow \phi \mid nil)$, and $(a_3 \leftarrow \phi \mid nil)$.

Another important motivation for using declarative goals is communication. While an agent can delegate a procedural goal to another agent, she is required to plan for the goal before she can delegate it. Moreover, the requester may not know how to achieve the goal. Thus the use of declarative goals allows distribution of both computation and knowledge, in the sense that the requesting agent need not plan for all of her goals. Declarative goals are also necessary for reasoned responses to communication. For instance, to determine if an agent should adopt a request to achieve a goal, she must know whether this requested goal conflicts with her own goals. Moreover, even if this requested goal conflicts with one of her plans, she (being a very helpful agent) might still decide to adopt it provided that it does not conflict with the associated declarative goal.

Declarative goals are essential for rational behavior. In addition to allowing an agent to plan from scratch when all her plans in the plan library have failed, declarative goals facilitate reasoning about interferences among goals. An agent may have two conflicting goals in the goal base. To decide which goal to achieve, it can do some reasoning with these declarative goals to find out which of these is more important to the agent.

2.5.2 Issues in Agent Programming Languages with Declarative Goals

As mentioned earlier, recently there has been some work on APLs with declarative goals. Before going over these frameworks, I briefly discuss the common issues in APLs with declarative goals.

One issue in APLs with declarative goals concerns the type of goals handled by an APL. Most APLs use achievement goals as the only type of goals. Achievement goals refer to a goal that needs to be achieved once. Some APLs also allow the use of other types of goals, e.g. maintenance goals and perform goals (i.e. a goal to execute some actions). Some also allow the use of inconsistent goal bases (e.g. GOAL [100], Dribble [237]). They assume that two inconsistent achievement goals need to be achieved at different times. Thus, although in these frameworks the goal-base can be inconsistent, the adopted declarative goals (i.e. intentions) must be consistent with each-other. Note that, the need for such inconsistent goal-bases arises from the fact that these APLs take goals to be state formulae, rather than general temporal formulae. For instance, if an agent has two inconsistent achievement goals ϕ and $\neg\phi$, the agent's goal state could be represented by the temporal formula $\Diamond\phi \wedge \Diamond\neg\phi$, i.e. the goal to eventually achieve ϕ and to eventually achieve $\neg\phi$, which is consistent. One exception to this is [187], that defines a goal as a path formula; however it does not deal with the dynamics of declarative goals. Some frameworks model goals with different priorities (e.g. [187]).

Another issue concerns the representation of declarative goals and intentions. In other words, how can one incorporate these goals as a part of the programming language, and what features of these goals are included in the framework. Most APLs keep declarative goals and procedural plans in two separate databases. They then use these goals to trigger some AgentSpeak(L) or 3APL-like rules. Other frameworks treat declarative goals as individual and active components that manage their own state and post events as required (e.g. Jadex [23]). These events are then handled by AgentSpeak(L)-like agents. These frameworks also define the life-cycle of these goals explicitly.

A third issue concerns the consistency of intentions. Two intended goals can be conflicting in various ways, such as:

- A goal can be directly inconsistent with another goal. In that case, all the plans for achieving that goal conflict with all of the plans to achieve the other goal.

- A goal can be inconsistent with some of the plans to achieve another goal, and not others. In that case, any plan to achieve the first goal is in conflict with some of the plans to achieve the second.

- Two goals may be mutually consistent. However, it may be the case that some of the plans to achieve one goal are inconsistent with some of the plans to achieve the other.

Thus, while selecting a plan to achieve a goal, the agent must check that only consistent plans are selected. This ensures that the agent will not commit to and execute a plan that makes one of her goals impossible to achieve. Unfortunately, in most APLs with declarative goals, there is no requirement that a declarative goal be consistent with a procedural goal, i.e. plan. The fact that these APLs maintain intended goals and plans in two separate databases makes it even harder to ensure this consistency. In fact, to the best of our knowledge, no APL in the literature handles these issues.

A fourth issue involves the representation of means-ends relationship between goals and subgoals or plans adopted to achieve these goals, i.e. how these frameworks capture the dependency between subgoals and their parent-goals. Recall that, PRS-based agents adopt plans in response to procedural goals. Similarly, in PRS-based declarative goal oriented APLs, agents adopt plans in response to declarative goals in the goal-base. These plans may in turn involve the achievement of other declarative (sub)goals that may trigger the adoption of other plans. Note that, the only reason the agent intends any of these subgoals and plans is due to her commitment towards the parent goal. In other words, the agent's commitments towards these subgoals can be viewed as conditional intentions, (implicitly) conditioned on intending the super-goal. Thus, if in any situation the agent drops the super-goal, she should also drop all these subgoals. Most APLs do not model this dependency, and thus fail to give up subgoals or plans when the associated goal is dropped. The ones that do, share a similar underlying technique: they introduce some construct in the language that captures the fact that a subgoal was adopted to achieve a goal (and a sub-subgoal was adopted to achieve the subgoal, etc.). Then, when the goal is dropped, they use this information to drop all such subgoals (and the sub-subgoals etc., if any). For instance, GD-3APL [48] attaches a declarative goal with each intended plan, and this information can be used to abandon plans when necessary. However, no mechanism is provided to use this information to effect the appropriate goal contraction. CAN [247] includes a procedural construct that includes both the associated declarative goal and its failure condition with an adopted plan.

Another issue concerns how these declarative goals are used by these frameworks, i.e. which of the aforementioned advantages of declarative goals are realized. As mentioned earlier, the literature identifies three major uses of declarative goals: selecting plans using these goals, decoupling plan success/failure from goal success/failure, and planning for goals on-the-fly.

In the following, I discuss in more detail various declarative goal-oriented APLs found in the literature.

2.5.3 Declarative Goal Oriented Languages

GOAL

The programming language Goal Oriented Agent Language (GOAL) [100] can be identified as one of the first to attempt to incorporate declarative goals in agent programming languages. Like 3APL, GOAL integrates theory and programming in a single framework by providing both an agent programming framework and a programming logic, the latter derived from the operational semantics of the former. Thus statements proven in the logic concern properties of agents specified in the programming language. Unfortunately, GOAL does not take most of the advantages of declarative goals (as discussed in Section 5.1) into account, and only uses declarative goals to select plans.

A GOAL agent keeps a propositional belief-base and a declarative goal-base. To consider the possibility of mutually inconsistent goals (to be achieved at different time steps), the goal-base is allowed to be inconsistent. However, individual goals need to be consistent, and believed propositions cannot be in the goal-base. An agent's goals are then defined to be the formulae that are entailed by an entry in the goal-base, rather than those that are entailed by the entire goal-base. The reason for doing this is that, since the agent can have mutually inconsistent goals, defining goals using the latter can trivialize the logic. Thus agents' goals are modeled using a weak logic that does not include the K axiom, and as a result, goals do not distribute over implication, and two goals cannot be conjoined to form another goal. Note that, this formalization of goals is very "syntactic" and can only handle achievement goals.

A central idea in GOAL is that of *conditional actions*. These actions are used to help agents decide what actions to perform next, and thus can be viewed as very simple action selection rules. Intuitively, a conditional action $\phi \rightarrow do(a)$ states that the agent may consider executing the basic action a if the mental state condition ϕ holds. Basic actions are defined similarly as in 3APL, i.e. using a (unspecified) partial function on the belief-base. Also, two special actions $adopt$ and $drop$ are introduced to respectively adopt a new goal, or drop some old goals. The semantics of $adopt$, $drop$, and conditional actions are specified in terms of the semantics of basic actions. GOAL adopts a simple blind commitment strategy [35]. The authors argue that this is just a default strategy, and that conditional actions (with a $drop$ in the consequence) can be used to override this commitment strategy. However, it is not obvious how any other strategies can be adopted. E.g., say one wants to specify a commitment strategy that enables an agent to give up her goals when they become impossible to achieve. Without a temporal component built into the goal semantics, this is clearly impossible to express.

A GOAL agent is thus defined to be a tuple $\langle \Pi, \Sigma_0, \Gamma_0 \rangle$ consisting of a set of conditional actions Π and an initial mental state $\langle \Sigma_0, \Gamma_0 \rangle$, where Σ_0 is the initial belief-base and Γ_0 is the initial goal-base. While AgentSpeak(L) and 3APL agents search for applicable rules and execute intended actions at every step, a GOAL agent searches for an appropriate conditional action. Since a conditional action associates a basic (primitive) action with a mental condition, the deliberation mechanism of GOAL is indeed a very primitive one. The overall operational semantics of GOAL agents is given using *traces*, which are infinite sequences of consecutive mental states interleaved with scheduled conditional actions, where the first state of each of the traces is the agent's initial mental state.

The specification for basic actions provides the basis for the programming logic of GOAL. Actions are specified using Hoare triples of the form $\{\phi\}\ a\ \{\psi\}$, where ϕ and ψ are mental state formulae. Hoare triples for conditional actions are interpreted relative to the set of traces associated with the GOAL agent, and a time-point in these traces. These user-defined Hoare triples are used to specify preconditions, post-conditions, and frame conditions of actions. On top of the Hoare triples for specifying actions, a temporal logic is defined for specifying properties of GOAL agents. One can then express various *liveness* and *safety* properties of an agent A by considering the temporal formulae that are valid with respect to the set of traces S_A associated with A. It can be shown that such properties are equivalent to a set of Hoare triples. Thus the properties can be proven by showing that the Hoare triples are entailed by the specifications of the actions that appear in the program. Thus it is very straightforward to verify the properties of agents in GOAL.

Note that, a rich notion of action structure is missing in the GOAL programming language. All one has is simple condition-action rules. Moreover, the only deliberation and planning mechanism in GOAL is provided via conditional actions that allow the agents to select primitive actions based on their mental states.

Dribble

Unlike GOAL, Dribble [237] incorporates a procedural motivation component (i.e. plans) in the language. In particular, Dribble takes 3APL's [99] mechanism for *creating and modifying plans* during the execution of the agents, and GOAL's facility for *using declarative goals for selecting actions* into account, and combines these in a single framework. Thus, Dribble uses declarative goals to allow agents to select and modify plans when required. Moreover, as in GOAL, Dribble also includes a dynamic logic on top of its operational semantics to specify and verify properties of agents.

In addition to beliefs and declarative goals, the mental state of a Dribble agent also includes a plan component. Only a single plan can be handled at any one time (no concurrency). The plan of an agent can be changed through application of rules as well as execution of executable actions. Dribble defines two types of rules, namely *Goal rules*, and *Practical Reasoning* (PR) rules. A goal rule g is a pair $\varphi \rightarrow \pi$ such that φ is a propositional formula involving beliefs and goals, and π is a plan. Intuitively a goal rule says that the plan π can be adopted if the mental condition φ holds. Goal rules are used to *select* plans for the first time (i.e. when the plan component of the agent is an empty plan). The condition in φ specifies what the plan π is good for. On the other hand, PR rules are similar to rules in 3APL, and can be used to create plans (possibly from abstract plans), to modify plans, and to model reactive behavior (using rules with empty heads).

Programming a Dribble agent amounts to specifying its initial beliefs and goals and writing sets of goal rules and PR rules. Formally, a Dribble agent is a triple $\langle\langle\sigma_0, \gamma_0, E\rangle, \Gamma, \Delta\rangle$, where $\langle\sigma_0, \gamma_0, E\rangle$ is the initial mental state with initial belief-base σ_0, initial goal-base γ_0, and an empty plan E, and where Γ is a set of goal rules and Δ is a set of PR rules. Note that the initial plan of a Dribble agent is the empty plan E. The reason for this is that a Dribble agent should be able to select a plan (using rules) based on its declarative goal specification, and giving the agent a plan at start up is counter-intuitive in this respect. The operational semantics of the Dribble programming language is specified using a transition system. A computation run $CR(s_0)$ is a finite or infinite sequence $s_0, \cdots, s_n$ or $s_0, \cdots$, where $s_i = \langle\sigma_i, \gamma_i, \pi_i\rangle$ are mental states (where π_i denotes the plan of the agent), and for all i there exists a transition from s_{i-1} to s_i as defined in the transition system for the Dribble agent. The meaning of a Dribble agent $\langle\langle\sigma_0, \gamma_0, E\rangle, \Gamma, \Delta\rangle$ is then defined to be the set of its computation runs $CR(\langle\sigma_0, \gamma_0, E\rangle)$. Thus the first state of the computation runs is the initial mental state of the Dribble agent.

As mentioned, Dribble is an expressive language that improves on GOAL by adding complex plans and rules to manipulate goals. Nevertheless, it has some limitations. Although the authors argue that goal rules and PR rules together can be used as a regression planning mechanism, this is misleading, since no lookahead is incorporated. Also, Dribble does not support exogenous actions; e.g., suppose that the agent has δ as a plan, and some exogenous action happens, which makes the preconditions of δ false forever. While a PR rule with an empty body can be used drop this plan, without a temporal component built into the language, it is impossible to detect that the plan has become forever impossible to execute. Note that, such a mechanism is important if one wants to ensure that the agents do not intend unachievable goals. Moreover, Dribble only allows sequential plan adoption and execution. In other words, agents cannot concurrently commit to two different plans. Finally, one major problem in Dribble is that it uses distinct databases for two types of intentions, i.e. declarative goals and procedural plans, and there is no mechanism for ensuring consistency between these two. Put otherwise, Dribble semantics allows agents with an inconsistent intention-base, e.g. an agent can have a declarative goal ϕ and a plan that makes ϕ unachievable. While other programming languages address some of the problems mentioned earlier, to the best of our knowledge, no BDI agent programming language with both declarative and procedural goals offers a solution to this problem.

Goal Directed 3APL (GD-3APL)

In [48], Dastani et al. present an extension of 3APL [99], called Goal Directed 3APL (GD-3APL), that incorporates both declarative and procedural goals into a single framework. GD-3APL agents are similar to Dribble agents, in that they have beliefs, goals, plans, and rules. The overall semantics is also very similar. The major difference is that GD-3APL uses a more expressive first order language, and that it defines an additional rule type to allow the agents to reason about and modify their declarative goals. GD-3APL provides three types of rules: *Goal Revision* (GR) rules, *Plan Selection* (PS) rules, and *Plan Revision* (PR) rules. GR rules can be used to revise goals, and variants of GR-rules can be used to generate, extend, or drop goals. In some sense, these rules allow agents to reason about their declarative goals. PS rules are like Dribble's Goal rules and PR rules are similar to practical reasoning rules in Dribble and 3APL.

Another advantage of GD-3APL is that an agent's plans/procedural intentions are modeled using a $\langle$plan-body, goal$\rangle$ pair, where the second element is added to record the goal for which the plan was selected. So if the goal gets dropped for some reason or revised by a GR rule before the agent has finished executing this plan, this information is used to also drop the plan. This facilitates decoupling of plan success from goal success. However, exactly how this information can be utilized is left open in the framework. Finally, GD-3APL agents can concurrently handle several goals, by committing to and executing multiple plans, unlike Dribble agents. Note that, GD-3APL does not provide a logic to verify properties of agent programs. Also, an offline lookahead mechanism for planning is still missing.

2APL

Another extension of the 3APL language called 2APL was proposed in [47]. In addition to incorporating first-order features and declarative (achievement) goals, 2APL extends the original proposal of 3APL in many different ways. In particular, 2APL has programming features to support multiple agents, their environments, sensing actions, goal modification actions (for the adoption/dropping of goals), and agent communication (speech acts). Moreover, in addition to the plan selection/generation and plan repair rules, a new type of rules called PC or "Procedure Call" rules is introduced to process internal and external events and received messages. Finally, it extends the plan language of the APL to include a new plan construct to implement atomic (non-interleaving) execution of plans.

A BDI Extension of the Golog-Family

One problem with all of the above frameworks is that although they may support declarative goals, they do not support planning in the sense that there is no lookahead mechanism built into these frameworks. Also, most of these frameworks are not grounded on a formal theory of action, and thus only allow limited reasoning. The agent programming language proposed by Sardiña and Shapiro [187] (let us call it S&S) combines two existing approaches to agent theory (viz. the work in [198]) and to agent programming (namely, IndiGolog) to provide an expressive BDI-agent programming language that supports planning/lookahead. S&S is built on top of a situation calculus-based action theory.

In S&S, an agent program consists of a high-level procedural specification of the agent's behavior (i.e. a single non-deterministic program), a declarative specification of the agent's mental states, and an underlying action theory. The interpreter's job is to search for a *rational* execution of the given program (i.e., one that satisfies the agent's goals, as discussed below). An agent's mental state consists of her beliefs and her goals. S&S incorporates a KD45 model of knowledge and a KD model of goals, both specified in terms of accessibility relations over possible worlds. The model of goals has a temporal component associated with it, and thus it can handle both achievement and maintenance goals. Also, S&S supports prioritized goals through prioritized accessibility relations and all goals are not assumed to be equally important. $(\phi_1 > \phi_2 > \cdots > \phi_n)_s$ is used to represent that the agent in situation s has n prioritized goals, where ϕ_i denotes all the goals of the agent at level i, ϕ_1 being the highest priority goal. Moreover, S&S's language is rich enough to allow queries of whether the agent is able to achieve certain goals in a given situation.

To help the agents decide which plans are preferable, S&S defines an ordering on plans/strategies, which are modeled using action selection functions (as discussed in Section 2.4). It then defines a rational course of action to be a strategy that the agent *knows-how* (i.e. is able) to execute, and she knows is one of the most preferred strategies w.r.t. her prioritized declarative goals.

As discussed above, most reactive plan execution languages that incorporate declarative goals include a pre-compiled plan library. The agents use their declarative goals as triggers to select plans from this library, and hierarchically decompose and execute these plans. Unlike these APLs, S&S uses an IndiGolog-style controller. Recall that, in IndiGolog, the programmer's job is to model the domain using the appropriate axioms, and specify the agent's behavior using a high-level non-deterministic program. Given a starting situation, the IndiGolog interpreter tries to (incrementally) find an execution of this non-deterministic program. S&S uses a similar control strategy. However, the S&S interpreter also needs to take the agent's declarative goals into account.

To this end, the rational-search operator $\Delta_{rat}(\delta : X_G)$ was introduced in S&S. The idea of this construct is that, given a non-deterministic IndiGolog program δ and a set of prioritized goals $X_G = \phi_1 > \phi_2 > \cdots > \phi_n$, the $\Delta_{rat}(\delta : X_G)$ operator will produce a simple and ready to execute terminating deterministic plan (i.e. sequence of actions or conditional plans) δ', whose execution respects both the given program δ, in the sense that δ' is an instantiation of δ, and the set of declarative goals, in that it achieves as many highest priority goals as possible (i.e. it is a rational course of action w.r.t. the given set of declarative goals). S&S assumes that the given program has the highest priority, and thus any declarative goals that conflict with this program will not be achieved. This operator provides a mechanism for combining a procedural specification of behavior and a set of prioritized declarative goals, and is meant to be used by the agent programmer to specify when lookahead is necessary.

To the best of my knowledge, S&S is one of the only two BDI agent programming frameworks[8] that offers deliberation with lookahead, and thus supports planning. Thus, S&S combines a procedural representation of behavior and prioritized declarative goals in an expressive language that is able to model ability and know-how, the temporal aspects of goals and actions, and the relative importance of goals. However, this expressiveness of S&S comes at the cost of complexity – determining whether a plan is rational or not involves searching the plan space defined by the given high-level non-deterministic program, and comparing all the strategies that can be induced by this program. This is clearly problematic in a dynamic environment. In fact it is unknown

[8]Another such language, as discussed below, is CAN-PLAN.

whether there exists a practical procedure to implement this mechanism. Also, it can be argued that agents in S&S are not so goal-directed, since their overall behavior is controlled by the given program.

CAN and CAN-PLAN

As mentioned earlier, another agent programming language that supports lookahead/planning is CAN-PLAN [185], which is an extension of the Conceptual Agent Notation language (CAN) [247]. In contrast to the situation calculus based S&S, CAN-PLAN is a PRS-based language.

The underlying basic infrastructure of CAN is similar to that of AgentSpeak(L). Agents in CAN have a first-order belief-base, a set of plans, and a first-order intention-base, and thus an agent configuration is modeled as a tuple $\langle B, A, \Gamma \rangle$, where B is a possible belief-base, A denotes the actions performed by the agent so far, and Γ represents a possible intention-base. Plan-bodies or programs in CAN can be constructed from primitive actions, operations to add/delete beliefs, tests for conditions, events or achievement goals $!e$, sequences, parallelism, the operator $\llparenthesis \{\cdot : \cdot\} \rrparenthesis$ which is used to represent a set of guarded alternatives as discussed below, the choice operator dual of sequencing $P_1 \rhd P_2$ which executes P_1 and then executes P_2 only if P_1 fails, and the operator $Goal(\phi_s, P, \phi_f)$ (discussed later). CAN uses a set of transition rules to specify the evolution of an agent. Agents in this language respond to events. In response to an event e, a CAN agent uses the plan-library Π with rules of the form $(e' : \phi_i \leftarrow P_i)$ to collect all context condition-plan-body pairs $\phi_i : P_i$ whose event e' can be unified with e, places these pairs inside the special construct $\llparenthesis \{\} \rrparenthesis$ to form the plan $I_1 = \llparenthesis \phi_1 : P_1, \cdots \phi_n : P_n \rrparenthesis$,[9] and inserts I_1 into the intention-base. Another transition rule is then used to (non-deterministically) choose from one of the plan-bodies in I_1 whose context condition holds. Suppose that the context condition ϕ_1 holds, and the agent chooses to try P_1 first. In that case, I_1 will be replaced with the new intention $P_1 \rhd I_2$, where $I_2 = I_1 - \{\phi_1 : P_1\}$, i.e. with the plan that P_1 should be tried first, and if the execution of P_1 fails for some reason, the intention I_2 (which is similar to I_1 but now does not include the pair $\phi_1 : P_1$) will be attempted. Thus, unlike AgentSpeak(L) and similarly to Jason and 3APL, CAN provides a mechanism for handling failure of plans. However, this is very different from 3APL failure rules.

In addition to this basic infrastructure, CAN provides a mechanism for representing both declarative and procedural goals in a uniform manner. For this purpose, it uses the procedural construct $Goal(\psi_s, P, \psi_f)$, which can be read as: the agent should achieve the declarative goal ψ_s using the set of procedures P (which is of the form I_1 discussed above), failing if ψ_f becomes true. CAN provides a set of transition

[9] Note that I simplified the notation a bit by getting rid of the variable bindings.

rules for $Goal(\psi_s, P, \psi_f)$ defined in terms of the above rules. The execution of a $Goal(\psi_s, P, \psi_f)$ construct is specified such that at every step, it only updates the associated P (by executing a step of P), giving up only when the goal ψ_s is achieved, or when it is no longer possible (i.e. when ψ_f holds). Using these rules, CAN guarantees that the execution of $Goal(\psi_s, P, \psi_f)$ will obey some of the properties of declarative goals discussed in Section 5.1. For example, the explicitly specified success condition can be used by the semantic specification to detect early/fortuitous achievement of goals (i.e. achievement of goals before the associated plan has been fully executed) and drop the associated goal and plan. Also, it can be used decouple successful execution of plans from successful achievement of goals. This is done by checking whether the success condition ϕ_s holds after the execution of a plan from P; if the plan has been successfully executed, but the goal has not been achieved yet, the CAN agent will try another plan from P. If the goal remains false after all plans in P have been executed, the CAN agent will retry the plans in P. This mechanism is provided by replacing the intention $Goal(\psi_s, P, \psi_f)$ with $Goal(\psi_s, P \rhd P, \psi_f)$ at the beginning of the execution, and whenever P is not of the form $P_1 \rhd P_2$ (i.e. after it has already tried P but failed to achieve the associated goal). Similarly, the failure condition is specified to decouple plan failure from goal failure, and to remove any committed-to plans when the associated goal has failed (or becomes impossible). This special construct is meant to appear in the plan body part of rules specified by the agent programmer. Thus, this mechanism of CAN can be used for both failure handling and monitoring of declarative goals.

CAN-PLAN [185] extends CAN by including a lookahead mechanism to support offline planning. To perform offline planning, one needs an action theory. To this end, agents in CAN-PLAN are equipped with a simple STRIPS-like action description library Λ that contains rules of the form $a : \psi_a \leftarrow \Phi_a^-; \Phi_a^+$, one for each action a in the domain. Here ψ_a corresponds to the preconditions of a and Φ_a^+ and Φ_a^- denotes the add and delete list of atoms, respectively. CAN-PLAN incorporates the additional $Plan(P)$ operator in the plan language. This operator searches for a complete hierarchical decomposition of P before executing a single step in a similar way to an HTN planner. It is very similar to the IndiGolog Σ search operator, in that $Plan(P)$ can evolve to $Plan(P')$, provided that P can evolve to P' and can reach a final configuration in a finite number of steps. In CAN-PLAN, the agent programmer can mix the $Goal()$ and $Plan()$ operators in various ways to produce different types of failure handling and lookahead. For example, consider the construct $Goal(\phi_s, Plan(Goal(\phi_s, P, \phi_f)), \phi_f)$;

- The external $Goal()$ operator ensures that the agent will use the program $P^* = Plan(Goal(\phi_s, P, \phi_f))$ towards the eventual satisfaction of the goal ϕ_s. The agent is committed to ϕ_s, in that P^* is reinstantiated and retried until ϕ_s holds. Also, it is not necessary to completely execute the plan returned by the planner (i.e. P^*), e.g. if ϕ_s is satisfied before P^* has been fully executed. Finally, the goal ϕ_s is dropped when failure condition ϕ_f becomes true.

- The $Plan()$ operator guarantees that the program P^* has a terminating execution. Note that, an exogenous action might render this program non-executable; however, as mentioned above, in that case the external $Goal()$ operator will call the planner again.

- The internal $Goal()$ operator ensures that the agent will use the program P towards the eventual satisfaction of the goal ϕ_s. Also, at plan-time, P is solved up to the point where the goal is met.

It can be shown that for a restricted class of CAN-PLAN agents,[10] the $Plan()$ operator indeed corresponds to an HTN-planner in the sense that for an agent, there is an execution of $Plan(P)$ in CAN-PLAN, if and only if there is a solution to the corresponding planning problem in a HTN-planner. Also, any execution of $Plan(P)$ corresponds to a HTN-plan solution.

Thus, CAN-PLAN provides a mechanism for on demand lookahead planning to

[10]This constraint restricts the belief-base language of an CAN-PLAN agent to that of an HTN planner.

the agent programmer. While the $Plan()$ operator itself does not consider potential interaction with exogenous actions, in some sense $Goal()$ and $Plan()$ can be mixed to handle external interferences. For instance, $Goal(\phi_s, Plan(P), \phi_f)$ will re-plan if the initial plan obtained fails due to an external interference. Nevertheless, CAN-PLAN's lookahead feature is local in the sense that it does not take into account other concurrent intentions. In other words, the result of planning may include actions that are in conflict with other goals of the agent. Also, while CAN-PLAN uses an action theory for deliberation, it does not utilize this for updating the agent's state, which is a bit inconsistent.

Jadex

Following [45, 229, 136], Braubach et al. [23] propose to treat declarative goals as first class objects in the Jadex framework. Declarative goals in Jadex are individual entities that manage their own state (in contrast to being managed by the agent), and post appropriate events as necessary. Recall that Jadex agents are PRS based agents that have a built-in plan library, and that respond to events. Thus, to handle these declarative goals, all an agent has to do is to listen to and respond to these events posted by the goal objects by adopting, executing, or dropping the appropriate plan. Note that, the current implementation of Jadex [22] does not utilize these declarative goals, and thus it only provides a specification of an extended version of the language.

Braubach et al. studied various goal oriented agent programming languages, architectures, and methodologies, and identified four different types of goals: achievement goals, maintenance goals, perform goals, and query goals. Here, a perform goal specifies some activities to be done, and hence the success of the perform goal depends only on the fact that the activity was performed. They argued that perform goals are different from achievement or maintenance goals, since they do not require any state to be achieved or maintained. Query goals serve the purpose of information acquisition. If there is enough information in the agent's knowledge-base to answer a query goal, it succeeds with that answer; otherwise it becomes the achievement goal of collecting enough information to answer that query.

By analyzing these goals, Braubach et al. identified various states that a goal can be in during its lifecycle. The basic three states of a generic goal are *new, adopted,* and *finished*. An adopted goal can in turn be in various sub-states, such as *option, active,* and *suspended*. Adopting a goal makes it desirable to achieve it and thus it can be seen as an option that the agent can possibly pursue when the actual circumstances allow. To actively pursue a goal, the agent's deliberation mechanism must activate the goal to initiate goal processing. Active goals can be later deactivated by the deliberation mechanism, and saved as an option. For instance, an active goal needs to be deactivated when it conflicts with another higher priority goal. On the other hand, an active or option goal can be suspended when its context becomes invalid. For example, a robot that has a goal of guarding some property during the night can suspend this goal in the day. Unlike options, suspended goals are not fed to the deliberator when it is deciding on what goal to actively pursue next. When the context becomes valid again, the suspended goal is added as an option.

For each of the four goal types, Jadex specifies a refinement of the active state by considering various attributes and the specific life cycle of these goals. For instance, an achievement goal has three sub states of the active state, namely *in process, succeeded*, and *failed*. It also consists of two conditions, a *target condition*, and a *failure condition*. The target condition specifies the world state that the achievement goal wants to bring about, and the failure condition specifies the conditions under which the goal should be dropped and considered to have failed. An active achievement goal will first check the target condition for fulfillment of the goal, and if the condition is met, the goal can be moved to the succeeded state and eventually dropped. If both the target and the failure conditions are false, the active achievement goal can post a goal addition event to the event queue. In response to this event, the agent will eventually adopt an applicable plan, and execute this plan to achieve the goal. At any stage of this execution, if the target or failure conditions are met, the goal will move to the succeeded or failed state, and post a drop event. To handle this event, the agent needs to drop this plan (along with all subgoals adopted for this goal). Braubach et al.'s proposal also specifies how the other three types of goals are processed (see [23] for details).

Thus, like CAN, Jadex allows full decoupling of goals and plans by monitoring declarative goals. In other words, this framework provides a way to decouple both goal-plan failure and success. Moreover, as in CAN, it provides a mechanism for detecting fortuitous or early achievement of goals. Unfortunately, Jadex does not provide a formal semantics for these goals and their dynamics. While it allows the use of incompatible goals, it does not deal with the relationships between these incompatible goals. For instance, it does not require a plan adopted to process a goal to be compatible with another adopted goal.

Other Recent Work

Recently, there has been work on incorporating more expressive types of temporally extended goals in APLwDGs [236, 235, 49]. In their APL-based framework, van Riemsdijk et al. [236] studied various types of goals commonly found in the literature. By investigating the commonalities among these goals, they proposed an abstract unifying framework that handles four types of temporally extended goals, namely achievement goals, query/test goals, perform goals, and limited forms of maintenance goals (e.g. a reactive maintenance goal, i.e. the goal to always make some proposition ϕ eventually true when it becomes false: $\Box(\neg\phi \rightarrow \diamond\phi)$). In contrast to Jadex, they gave a generic definition of goals and a generic transition system that is based on the life-cycle of goals and that includes generic rules specifying the conditions under which such a goal can become active, suspended, or dropped, and under which a plan for a goal can be adopted or dropped. In their framework, they define a generic goal construct as g(C, E, S, π). Here, $S \in \{\text{Active}, \text{Suspended}\}$ is the current state of the goal, and π is the plan of the goal. Also, C is a set of condition-action pairs that specifies when the goal changes state, provided that the agent's plan is non-empty. C is of the form $\langle condition, action\rangle$ with $action \in \{\text{Suspend}, \text{Activate}, \text{Drop}\}$; each of these pairs represents that if the condition follows from the agents beliefs, and the plan π is non-empty, then the action should be applied on the corresponding goal. Similarly, E is a set of condition-action pairs that specifies when the goal changes state, provided that the agent's plan is empty (a distinction between C and E, i.e. conditions that are checked during plan execution and those that are checked when a plan completes or has not been generated yet was made as the authors found it useful to do so). Then for each type of goal, they specify these condition-action pairs for C and E. For instance, for an achievement goal $\Diamond\phi$, the corresponding C is of the form $\{\langle s \vee f, \text{Drop}\rangle\}$, i.e. the goal with its non-empty plan can be dropped if the success condition s of the goal holds, or if its failure condition f holds. On the other hand, E is of the form $\{\langle s \vee f, \text{Drop}\rangle, \langle true, \text{Activate}\rangle\}$, i.e. the goal with its empty plan can be dropped if the success condition s of the goal holds, or if its failure condition f holds. Also, if the plan for the goal is empty, it can be unconditionally activated. The framework thus provides a uniform way of handling multiple types of goals. However, their account do not handle other types of temporally extended goals, e.g. regular maintenance goals (i.e. $\Box\phi$).

Dastani et al. [49] formalized this framework and extended it with various other types of temporally extended goals. They use LTL to specify goals, and showed that many kinds of temporally extended goals can be operationalized via that of achievement and maintenance goals. In particular, they include five additional types of goals that involve maintaining a formula ϕ over an interval: $\Box\phi$ (which maintains ϕ over all states), $\phi\ \mathcal{U}\ \tau$ (where ϕ is maintained until its terminating condition τ becomes true), $\Diamond(\tau \wedge \Box\phi)$ (where ϕ is maintained after its triggering condition τ holds), $\Diamond(\tau \wedge (\phi\ \mathcal{U}\ \tau'))$ (where ϕ is maintained once its triggering condition τ holds and until its terminating condition τ' becomes true), and $\Box(\tau \rightarrow (\phi\ \mathcal{U}\ \tau'))$ (where whenever the triggering condition τ becomes true, ϕ is maintained until its terminating condition τ' holds). They proved that their operational semantics produces correct results by showing that the traces produced by them satisfy the LTL formula. Since this is an abstract framework, the ideas here can be implemented in any APLwDG that provides a proper operational semantics for achievement and maintenance goals. However, being an abstract framework, many of the components are left abstract; for instance, while these frameworks assume the existence of a means-end reasoner that given a new goal g and an existing set of goals and plans, returns a plan that is consistent with the current goals and plans and that can be used to bring about g, no details about the reasoner or how to actually enforce and maintain this consistency are given.

2.6 Conclusion

In this chapter, I reviewed previous research on three related areas that focus on intelligent agents, namely agent theories, agent architectures, and agent programming languages. In these, I focused on logical formulations of motivational attitudes, both from the agent theory and the agent programming language perspectives. In the sequel, I will refer to some of these and compare them with my contributions.

Chapter 3

Foundations

3.1 Introduction

In this chapter, I first discuss previous work that my formalization of prioritized goals and their dynamics is founded on. I start by introducing the situation calculus [148], which is a (mostly) first-order language for representing dynamically changing worlds, and by examining a class of action theories within the situation calculus [178] that can be used to succinctly specify dynamic domains. Then I talk about previous work to formalize knowledge and knowledge change in the situation calculus. Following this, to support modeling temporally extended goals in the situation calculus, in Section 3.5, I introduce a new sort of *infinite paths* in the situation calculus and propose an axiomatization for infinite paths. I also show some properties of the axiomatization. The material in this section is new work. Finally, I review ConGolog [51], a rich programming/process specification language in the situation calculus that the semantics of my proposed agent programming language SR-APL borrows from.

3.2 The Situation Calculus

The formal basis of my agent theory is Reiter's [178] version of the situation calculus [148]. The situation calculus is a sorted, (mostly) first-order language where all changes are the result of named actions. I will use sorts for actions, situations, paths, agents, and domain-specific objects. I use a, s and now, p and $path$, and agt (possibly with decorations) to quantify over actions, situations, paths, and agents, respectively. In addition, I use $\vec{x}$ and $\vec{y}$ to denote sequences of variables, $\forall\vec{x}$ to denote universal quantification over a sequence of variables, and P and Q as predicate variables and F and σ as function variables in second order quantification. I adopt the convention that function symbols begin with a lowercase letter while predicate symbols begin with an uppercase letter. I also assume that unless otherwise noted, all free variables in a formula are implicitly universally quantified in the widest scope. Finally, I use standard notations for logical connectives and quantifiers, and the constants True and False for the true and false propositions.

A situation in the situation calculus represents a possible partial history of the domain. The *initial situations* are situations in which no actions have yet occurred. In general, a situation s describes a possible finite evolution of the domain that results from the occurrence of a certain finite sequence of actions (those in the history of s) starting from the initial situation associated with s.

The set of initial situations correspond to the ways the agent believes the domain might be initially. There can be multiple initial situations to facilitate modeling the fact that the agent may have incomplete knowledge initially. The actual initial state of the domain is represented by the distinguished initial situation constant S_0. The distinguished binary function symbol $do(a, s)$ is used to denote the successor situation of s resulting from action a being performed in situation s. In the situation calculus, actions are denoted by function symbols, possibly with parameters, and situations (world histories) are first order terms. For domains with multiple agents, the actions will include arguments specifying the agents involved in these actions, usually the first few arguments. For example, if $put(agt, x, y)$ stands for an agent agt's action of putting object x on object y, then the situation term $do(put(Agt_1, X_1, Y_1), S_0)$ denotes the situation resulting from Agt_1's putting X_1 on Y_1 when the world is in situation S_0. Also, $do(putDown(Agt_1, X_1), do(walk(Agt_1, P_1), do(pickUp(Agt_1, X_1), S_0)))$ is a situation denoting the world history consisting of the following sequence of actions:

$$[pickUp(Agt_1, X_1), walk(Agt_1, P_1), putDown(Agt_1, X_1)].$$

We want to define situations to be the smallest set that can be obtained by executing a sequence of actions starting from some initial situation. To define the structure of the situations, I adopt a set of *foundational axioms* given by Shapiro [194] that is based on the foundational axioms listed by Lakemeyer and Levesque [130], which in turns extend those given by Reiter [176] by incorporating multiple initial situations to model agents' knowledge and goals. The initial situations are first defined to be those that have no predecessors.

Definition 3.2.1.

$$\text{Init}(s') \overset{\text{def}}{=} \neg\exists a, s.\ s' = do(a, s).$$

Secondly, S_0 is declared to be an initial situation.

Axiom 3.2.2.

$$\text{Init}(S_0).$$

Another axiom is needed to state that performing different actions yields different situations, i.e. that do is injective.

Axiom 3.2.3.

$$\forall a_1, a_2, s_1, s_2.\ do(a_1, s_1) = do(a_2, s_2) \supset (a_1 = a_2 \wedge s_1 = s_2).$$

The next axiom is a second-order induction axiom for situations. It says that if a property P holds for all the initial situations, and if P holds for all successors to a situation s provided that it holds for s, then P holds for all situations.

Axiom 3.2.4.

$$\forall P.\ [(\forall s.\ \text{Init}(s) \supset P(s)) \wedge (\forall a, s.\ P(s) \supset P(do(a, s)))] \supset \forall s.\ P(s)).$$

The next axiom defines precedence for situations: s_1 *strictly precedes* s_2 if s_2 can be obtained by executing a non-empty sequence of actions starting in s_1.

Axiom 3.2.5.

$$\forall s_1, s_2.\ s_1 \prec s_2 \equiv (\exists a, s.\ s_2 = do(a, s) \wedge (s_1 \preceq s)),$$

where $s_1 \preceq s_2$ denotes that s_1 *precedes* s_2 and is defined as follows:

Definition 3.2.6.

$$s_1 \preceq s_2 \overset{\text{def}}{=} s_1 = s_2 \vee s_1 \prec s_2.$$

Shapiro [194] showed that this axiomatization of $\prec$ is equivalent to the one given by Levesque et al. [139].[11]

I also borrow another axiom from Shapiro to recursively define the concept of situations having the same history. Two situations are said to have the *same history* if

[11]Note that [178] uses the notation $\sqsubset$ and $\sqsubseteq$ to denote strict precedence and precedence when situations are not assumed to be executable (see below for a definition of executable situations).

they can be obtained by performing the same sequence of actions, but perhaps starting from different initial situations.

Axiom 3.2.7.

$$\text{SameHist}(s_1, s_2) \equiv (\text{Init}(s_1) \equiv \text{Init}(s_2)) \wedge$$
$$(\neg\text{Init}(s_1) \supset \exists a, s_1', s_2'.\ s_1 = do(a, s_1') \wedge s_2 = do(a, s_2') \wedge \text{SameHist}(s_1', s_2')).$$

Relations and functions whose value may change from situation to situation are called *fluents*, and are denoted by predicate and function symbols taking a situation term as their last argument. These fluents are used to specify the dynamic aspects of the domain. For example, Holding(r, x, s), which is a relational fluent, might mean that a robot r is holding an object x in situation s; position(r, s) is a functional fluent that might denote the position of robot r in situation s. There is also a special predicate Poss(a, s) meaning that the action a is executable (physically possible) in situation s.

3.3 Action Theory

Within this language, action theories can be formulated to describe how the world changes as a result of available actions. I use a theory that includes a set of axioms due to Reiter [178]. In the following, I discuss and provide examples of these axioms. To specify the actions in a domain, one must state the conditions under which it is physically possible to perform these actions. This is done by providing an *action precondition axiom* for each action in the domain. Poss is used to specify these axioms. For each action function f, an axiom of the following form is included in the theory:

$$\text{Poss}(f(\vec{x}), s) \equiv \Phi_f(\vec{x}, s),$$

where Φ_f is a formula whose free variables are among $\vec{x}, s$. For example,

$$\text{Poss}(\mathit{pickup}(agt, x), s) \equiv \forall y\ \neg\text{Holding}(agt, y, s) \wedge \text{NextTo}(agt, x, s) \wedge \neg\text{Heavy}(x)$$

says that the action $\mathit{pickup}(agt, x)$, i.e., an agent agt picking up an object x, is possible in situation s if and only if agt is not already holding something in situation s, she is positioned next to x in s, and x is not heavy.

A situation is called *executable* if every action in its history was executable:

Definition 3.3.1.

$$\text{Executable}(s) \overset{\text{def}}{=} \forall a, s'.\ do(a, s') \preceq s \supset \text{Poss}(a, s').$$

One must also specify how an action affects the state of the world; this can be done by providing *effect axioms*. For relational fluents, these axioms come in two varieties: *positive effect axioms* say what fluents become true when an action is executed under some conditions, while *negative effect axioms* specify what fluents become false when an action is executed under some conditions. For example, the following positive effect axiom says that dropping an object x causes it to become broken provided that x is fragile:

$$\text{Fragile}(x, s) \supset \text{Broken}(x, do(drop(agt, x), s)).$$

For functional fluents, we only need a single type of effect axioms. These specify what functional fluents change their values when an action happens under certain conditions. While these effect axioms provide some causal laws for the domain of application, McCarthy and Hayes [148] showed that they are not sufficient if one wants to reason about change. If the agent wants to form a plan that involves dropping an object, she might need to know that many fluents are not affected by dropping the object. For example, if an agent intends to sort some objects according to their color, she needs to know that the color of an object is not affected by picking it up, walking to another location, or by dropping it. Thus it is usually necessary to add so called *frame axioms* to specify when fluents remain unchanged following an action. Again for relational fluents, frame axioms also come in positive and negative varieties, while for functional fluents, we have a single type of frame axioms. For example, the following frame axiom says that dropping an object does not affect the color of things:

$$\text{color}(y, s) = c \supset \text{color}(y, do(drop(agt, x), s) = c.$$

The *frame problem* [148] arises because the number of these axioms is of the order of the product of the number of fluents and the number of actions [178]. Reiter [176, 178] proposed a solution to the frame problem using what he called *successor-state axioms* (SSA). Building on proposals by Haas [95], Pednault [162], and Schubert [190], he developed a solution to the frame problem for domains that do not involve non-deterministic actions or state constraints. His solution allows the specifier to only state the effect axioms; from these, a single successor-state axiom is obtained for each fluent by performing a syntactic transformation and using a *causal completeness assumption* and a *consistency assumption*. The causal completeness assumption says that before performing the syntactic transformation, effect axioms have been given for *all* possible ways the fluent may change value. The consistency assumption states that the conditions under which a fluent becomes true when an action is executed in some situation and those under which it becomes false are never jointly satisfied. As an alternative to the theory transformation on effect axioms, the axiomatizer can write the successor state axioms directly.

The successor-state axiom for a domain-dependent relational fluent R has the following form:

$$\mathrm{R}(\vec{x}, do(a,s)) \equiv (\gamma_R^+(\vec{x},a,s) \vee (\mathrm{R}(\vec{x},s) \wedge \neg\gamma_R^-(\vec{x},a,s))),$$

where $\gamma_R^+(\vec{x},a,s)$ ($\gamma_R^-(\vec{x},a,s)$, respectively) specifies *all* the conditions under which $\mathrm{R}(\vec{x}, do(a,s))$ becomes true (false, respectively). It is assumed that $\gamma_R^+(\vec{x},a,s)$ and $\gamma_R^-(\vec{x},a,s)$ are never jointly satisfied. Such a successor-state axiom encodes both effect and frame axioms and specifies exactly when the fluent changes.[12] While in general successor-state axioms are more complex than effect axioms, there will be much fewer successor-state axioms (one per fluent) than effect axioms and frame axioms combined. Note that to define regression for situation calculus formulae, Reiter restricts the right-hand side of a successor-state axiom of the above form to be *uniform in* s, which essentially requires s to be the only situation term mentioned in it. But this is too restrictive for my theory, since both the successor-state axioms of my knowledge and goal accessibility relations quantify over situations, making these non-uniform. Therefore, I do not impose this restriction for accessibility fluents. For similar reasons, I also do not require the right-hand side of action precondition axioms, i.e. $\Phi_f(\vec{x}, s)$, to be uniform in s. This does not pose a problem for my theory as I do not address regression here. Note that, Scherl and Levesque [189] have shown how regression can be performed for action theories with knowledge.

Let me give an example of a successor-state axiom for the Broken(x, s) fluent. The actions that affect this fluent are $drop(agt, x)$ and $repair(agt, x)$, which stands for the action of dropping and repairing, respectively, object x by agent agt. The successor-

[12] I do not discuss the ramification problem here; see [141] for a treatment compatible with my theory.

state axiom for Broken is as follows:

$$\text{Broken}(x, do(a, s)) \equiv (a = drop(agt, x) \wedge \text{Fragile}(x, s)) \vee$$
$$(\text{Broken}(x, s) \wedge a \neq repair(agt, x)).$$

It says that an object x is broken in the situation resulting from action a being performed in s, if and only if a is dropping x and x is fragile, or x was already broken in situation s prior to the action and a is not the action of repairing x.

The successor-state axiom for a domain-dependent functional fluent can also be specified in a similar manner. Assume that the only action that affects the color(x, s) fluent is the $paint(agt, x, c)$ action, which stands for the action of painting object x with color c by agent agt. Then the successor-state axiom for the color fluent can be specified as follows:

$$\text{color}(x, do(a, s)) = c \equiv a = paint(agt, x, c) \vee$$
$$(\text{color}(x, s) = c \wedge \neg\exists c'.\ c' \neq c \wedge a = paint(agt, x, c')).$$

It says that an object x has color c in the situation resulting from action a being performed in s, if and only if a refers to the action of painting x with color c, or x was already of color c in situation s prior to the action and a is not the action of painting x with a different color.

To specify successor-state axioms, Reiter relied on the assumption that different action terms represent different actions. For example, the above successor-state axiom for the Broken fluent is only meaningful if for all agt and x, $drop(agt, x)$ and $repair(agt, x)$ represent two different actions. Otherwise, the agent agt might end up with a broken object x after repairing it. This assumption can be formalized using *unique names axioms for actions*. For any pairs of distinct action functions f_1 and f_2, an axiom of the following form is needed:

Axiom 3.3.2.

$$f_1(\vec{x}) \neq f_2(\vec{y}).$$

Also, another axiom of the following form is needed for every action function f:

Axiom 3.3.3.

$$f(\vec{x}) = f(\vec{y}) \supset \vec{x} = \vec{y}.$$

For example, (a) below says that a $drop$ action and a $repair$ action are two different actions, and (b) says that two $drop$ actions are not the same if they do not have the same agent or do not involve dropping the same object:

$a)\ drop(agt, x) \neq repair(agt', x'),$

$b)\ drop(agt, x) = drop(agt', x') \supset (agt = agt' \wedge x = x').$

In general, for n action functions, $O(n^2)$ unique names axioms are required. However, these can be automatically generated from a list of action names and arities.

To define the domain, one also needs to specify what fluents hold initially and the agents' knowledge and goals in the initial situation. This is done using *initial state axioms* that only mention initial situations. For example, consider the following axioms:

$$a)\ \text{Init}(s) \supset \neg\text{Broken}(Obj_1, s),$$

$$b)\ \text{Know}(Agt_1, \neg\text{Broken}(Obj_1, now), S_0).$$

(a) says that the object Obj_1 is intact in all the initial situations, and (b) states that the agent Agt_1 knows in the actual initial situation S_0 that Obj_1 is intact. I will discuss the formalization of knowledge in Section 3.4.

When there are multiple agents in the domain, one also needs to include axioms for identifying the agent of an action, axioms such as:

$$\text{agent}(drop(agt, x)) = agt,$$

which says that the agent of the $drop(agt, x)$ action is agt; this uses a distinguished function 'agent'.

Finally, on some occasions I will need to quantify over formulae, use formulae as arguments of actions, and use predicates that take programs as arguments and these programs may contain formulae in wait/test conditions, etc. Thus I must encode formulae and programs as terms and formalize their relationship to the associated situation calculus formulae. This is tedious but can be done essentially along the lines of [51], where a $Holds$ predicate relating an encoded formula to the corresponding situation calculus formula is axiomatized. Variables in the argument formulae must be represented as terms and substitution must be axiomatized. I assume that we have such an encoding and axiomatization of $Holds$ in the rest of this thesis. For notational simplicity, I will suppress this encoding and use formulae and programs as terms directly. Also, note that if, for a particular domain, the set of formulae that will be used is known in advance, one can write the $Holds$ axioms directly for those formulae only.

A *basic action theory* is a theory that contains the following kinds of axioms:

- Foundational axioms Σ, i.e. Axioms 3.2.2-3.2.7;
- Unique name axioms for actions $\mathcal{D}_{una}$, i.e. Axiom schemata 3.3.2 and 3.3.3;
- Action precondition axioms $\mathcal{D}_{ap}$;
- Successor-state axioms $\mathcal{D}_{ss}$;
- Initial state axioms $\mathcal{D}_{init}$;
- Axioms identifying the agent of each action $\mathcal{D}_{agt}$ (if needed);
- Encoding Axioms $\mathcal{D}_{enc}$ (if needed).

From now on, I will refer to the union of these sets of axioms as $\mathcal{D}_{bat}$:

Definition 3.3.4.

$$\mathcal{D}_{bat} \overset{\text{def}}{=} \Sigma \cup \mathcal{D}_{una} \cup \mathcal{D}_{ap} \cup \mathcal{D}_{ss} \cup \mathcal{D}_{init} \cup \mathcal{D}_{agt} \cup \mathcal{D}_{enc}.$$

Note that here and in the sequel, I just use standard Tarskian semantics of first and second order logic (except for the semantics of SR-APL in Chapter 7). Sorts are handled in the standard way. I define modal operators simply as abbreviations with their semantics in terms of accessibility relations added to the situation calculus language.

I believe that Reiter's relative satisfiability theorem [178], which states that a BAT $\mathcal{D}$ is satisfiable if and only if $\mathcal{D}_{UNA} \cup \mathcal{D}_{S_0}$ is, can be extended to my framework. In Section 3.5, I introduce a new sort of infinite paths in the situation calculus. Note that paths are already there in the standard situation calculus models (satisfying Reiter's foundational axioms), but one needs to use second order quantification to refer to them. I think that the relative satisfiability theorem can be extended to include these paths axioms too. Such infinite paths are used to define the semantics of well known logics such as LTL and CTL*.

A word on axiomatizability/decidability: Reiter has shown that regressable projection queries within the situation calculus can be answered using first-order entailment. Others have proven decidability results for reasoning in interesting fragments of the language; for example, see [220] for an argument-less fluents fragment and [93] for a description logic like 2 variables fragment. Also, recently De Giacomo et al. [53] showed that verification of an expressive class of first-order μ-calculus temporal properties in the situation calculus with bounded action theories (i.e. where in all situations the number of ground fluent atoms that hold is bounded) is decidable.

3.4 Knowledge in the Situation Calculus

I allow the specifier to model the agents in terms of their mental states by including operators to specify agents' information and their motivation. In my framework, I work with knowledge rather than belief. Although much of my formalization should extend to the latter, I leave dealing with belief and belief revision for future work (see [194] for an account in the situation calculus). In the following, I discuss the formal basis of my information operator, i.e. knowledge.

3.4.1 Semantics

The logical foundation of my knowledge operator can be traced back to Hintikka's [102] reinterpretation of Kripke's possible world semantics of necessity [125]. Moore [151] was a pioneer in integrating knowledge and action into a single logical framework. In his formal theory, he accomplished this by formalizing Hintikka's modal logic of knowledge within McCarthy's first order situation calculus [148]. The basic idea is to use situations as possible worlds and to use a relation on situations as the accessibility relation for knowledge.

Scherl and Levesque [188] extended Reiter's successor-state axiom approach to model the effects of actions on agents' knowledge, combining ideas from Reiter and Moore. They use $K(s', s)$ to denote that in situation s, the agent thinks that she could be in situation s'.[13] s' is called a *K-alternative situation* to s. Also, the abbreviation[14] Know(Φ, s) is used to denote that the agent knows that Φ in situation s. The fluents in Φ will usually contain a situation constant or placeholder now that stands for the situation in which Φ must hold, e.g. Broken(obj, now). Φ(s) denotes the formula that results from replacing now with s in Φ. For ease of readability, I will suppress the placeholder where the intended meaning is clear from the context, e.g. Know(Broken$(obj), s)$. Shapiro [194] generalized the Know and K notation to handle multiple agents by adding an agent argument to them. I adopt this convention, but I will suppress the agent argument when dealing with single agent domains.

Scherl and Levesque require that initial situations can only be K-related to other initial situations:

[13]To comply with the situation calculus convention for fluents, which says that the last argument represents the actual situation, Scherl and Levesque reversed the order of the arguments of K.

[14]Note that the knowledge operator Know is an abbreviation or macro and not a predicate in the language, and thus it does not require formulae Φ to be encoded as a first-order term.

Axiom 3.4.1.

$$\text{Init}(s) \wedge K(agt, s', s) \supset \text{Init}(s').$$

As we will see later, the successor-state axiom for K ensures that in all the situations that are K-accessible from $do(a, s)$, a was the last action performed. This implies that all K-related situations share the same history. Since I am dealing with knowledge, I also constrain K to be initially reflexive (which also implies that K is initially serial):

Axiom 3.4.2.

$$\text{Init}(s) \supset K(agt, s, s).$$

Finally, to get positive and negative introspection of knowledge, I require K to be initially transitive and Euclidean:

Axiom 3.4.3.

$$\text{Init}(s) \supset \forall s_1, s_2.\ K(agt, s_1, s) \wedge K(agt, s_2, s_1) \supset K(agt, s_2, s).$$

Axiom 3.4.4.

$$\text{Init}(s) \supset \forall s_1, s_2.\ K(agt, s_1, s) \wedge K(agt, s_2, s) \supset K(agt, s_2, s_1).$$

As shown in Scherl and Levesque [188] and later in Shapiro [194], once these constraints are imposed on the initial situation, they continue to hold after any executable sequence of actions since they are preserved by the successor-state axiom for K.[15]

Using K, the knowledge of an agent, Know(agt, Φ, s), is defined as follows:

Definition 3.4.5.

$$\text{Know}(agt, \Phi, s) \overset{\text{def}}{=} \forall s'.\ K(agt, s', s) \supset \Phi(s').$$

Two useful abbreviations Kwhether and Kref are also defined. Kwhether(agt, Φ, s) means that the agent agt knows whether the formula Φ holds in situation s.

Definition 3.4.6.

$$\text{Kwhether}(agt, \Phi, s) \overset{\text{def}}{=} \text{Know}(agt, \Phi, s) \vee \text{Know}(agt, \neg\Phi, s).$$

Kref(agt, θ) means that the agent agt knows in situation s who/what the term θ refers to.

Definition 3.4.7.

$$\text{Kref}(agt, \theta, s) \overset{\text{def}}{=} \exists t.\ \text{Know}(agt, t = \theta, s).$$

3.4.2 Knowledge Change

Scherl and Levesque [188] showed how to capture the changes in knowledge of agents that result from actions in the successor-state axiom for K. In their framework, an

[15]Scherl and Levesque consider sensing actions as their only type of knowledge-producing actions while Shapiro deals exclusively with some inform communication actions (as discussed below).

agent's knowledge is affected by every action in the sense that she comes to know that the action was performed. It is assumed that agents know the successor-state axioms for actions, so the agents also acquire knowledge about the effects of these actions.[16] In addition, they allow the specifier to include actions that change the agent's knowledge in non-trivial ways. These actions, called *knowledge-producing actions*, come in two varieties: binary sensing actions and non-binary sensing actions. A binary sensing action is a sensing action that senses the truth-value of an associated proposition; e.g., the binary sensing action $sense_{onTable}(agt, b)$ could be performed to sense whether the block b is on the table or not. On the other hand, non-binary sensing actions refer to sensing actions where the agent senses the value of an associated term; e.g., the non-binary sensing action $countBlocksOnTable(agt)$ could be performed to get the number of blocks that are on the table. Following [137], the information provided by a binary sensing action is specified using the predicate SF(a, s), which holds if the action a returns the binary sensing result 1 in situation s. A *guarded sensed fluent axiom* is used to associate an action with the property sensed by this action. For example, one might have a guarded sensed fluent axiom to assert that the action $sense_{onTable}(agt, b)$ tells the agent agt whether the block b is on the table in the situation where it is per-

[16]One consequence of this is that agents are assumed to be aware of all actions that may happen in the environment. This in part allows us to avoid belief revision and its difficulties.

formed, provided that agt, b, and the table are located in the same room:

$$\text{InRoomWithTable}(agt) \wedge \text{Collocated}(agt, b) \supset$$
$$(\text{SF}(sense_{onTable}(agt, b), s) \equiv \text{OnTable}(b, s)).$$

Similarly for non-binary sensing actions, the term $sff(a, s)$ is used to denote the sensing value returned by the action. For example, the following guarded sensed fluent axiom asserts that the action $countBlocksOnTable(agt)$ tells agt the number of blocks that are currently on the table, provided that agt and the table are collocated:

$$\text{InRoomWithTable}(agt) \supset$$
$$(sff(countBlocksOnTable(agt), s) = \text{numOfBlocksOnTable}(s)).$$

When communication between agents is allowed, it is necessary to include additional types of knowledge-producing actions. Shapiro [194] considers the $inform$ communication action as his only type of knowledge-producing action. The $inform(agt_1, agt_2, \Phi)$ action can be used by an agent agt_1 to inform another agent agt_2 that the formula Φ currently holds. agt_1 can inform agt_2 that Φ if she currently knows that Φ:

$$\text{Poss}(inform(agt_1, agt_2, \Phi), s) \equiv \text{Know}(agt_1, \Phi, s).$$

Note that, since in this framework all actions are public (see footnote 16), whenever an $inform$ action occurs, *every* agent learns the informed formula Φ (as they know that since the $inform$ action occurred, its preconditions must have been true, i.e. they know that the informer knows that Φ; this along with the reflexivity of K implies that Φ). This may be problematic in some domains where some sort of privacy is required, e.g., when there are other competing agents in the environment. One solution is to encrypt the content Φ of the $inform$ action [197, 194]. Another solution to this problem was proposed by Lespérance [133]. He further extends the set of knowledge-producing actions in [188] to include two variants of the $inform$ communication action, $informWhether$ and $informRef$. $informWhether(agt_1, agt_2, \Psi)$, which is a primitive action, denotes that the agent agt_1 informs the agent agt_2 about the current truth value of the formula Ψ. agt_1 can inform agt_2 whether Ψ holds if she knows the current truth-value of Ψ:

Axiom 3.4.8.

$$\text{Poss}(informWhether(agt_1, agt_2, \Psi), s) \equiv \text{Kwhether}(agt_1, \Psi, s).$$

On the other hand, the $informRef(agt_1, agt_2, \theta)$ communication action is also a primitive action and it means that the agent agt_1 informs the agent agt_2 of who/what the term θ is. agt_1 can inform agt_2 who/what θ is if she knows the current value of θ:

Axiom 3.4.9.

$$\text{Poss}(informRef(agt_1, agt_2, \theta), s) \equiv \text{Kref}(agt_1, \theta, s).$$

Returning to our discussion on private communication, note that both of these actions hide the actual content of the message. For example, when the $informWhether(agt_1, agt_2, \Phi)$ action happens, every agent learns that the informer agt_1 informed the informee agt_2 whether Φ holds, but does not acquire any knowledge about the actual truth-value of Φ.

The version of the successor-state axiom for K that I adopt specifies how agents' knowledge is updated as a result of sensing actions as well as the $informWhether$ and the $informRef$ communication actions:

Axiom 3.4.10.

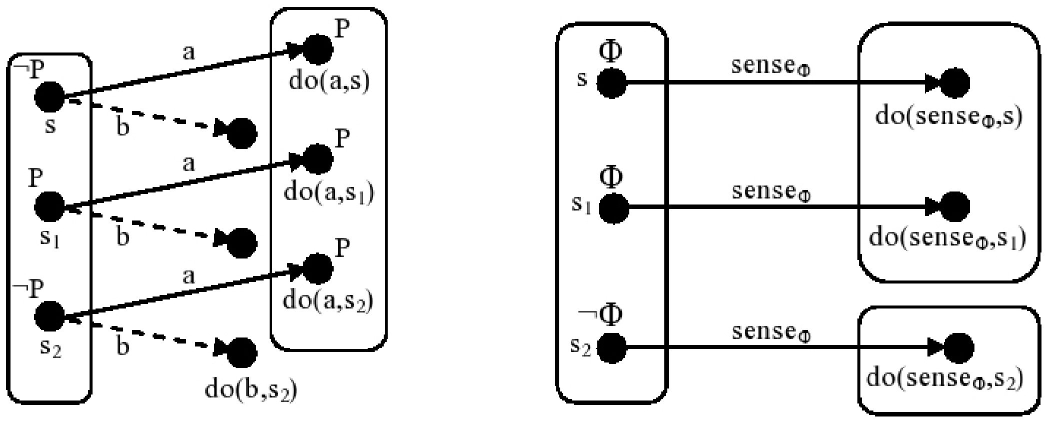
$$\begin{aligned} K(agt, s^*, do(a, s)) \equiv \exists s'. \ [&K(agt, s', s) \wedge s^* = do(a, s') \wedge \text{Poss}(a, s') \\ &\wedge \text{CompatWithKnowledgeAcquired}(agt, a, s', s)], \end{aligned}$$

where

$$\begin{aligned} &\text{CompatWithKnowledgeAcquired}(agt, a, s', s) \stackrel{\text{def}}{=} \\ &\qquad \text{BinarySensingAction}(a) \wedge \text{agent}(a) = agt \supset \text{SF}(a, s') \equiv \text{SF}(a, s) \\ &\quad \wedge \text{NonBinarySensingAction}(a) \wedge \text{agent}(a) = agt \supset sff(a, s') = sff(a, s) \\ &\quad \wedge \forall agt', \Psi. \ a = informWhether(agt', agt, \Psi) \supset \Psi(s') \equiv \Psi(s) \\ &\quad \wedge \forall agt', \theta. \ a = informRef(agt', agt, \theta) \supset \theta(s') = \theta(s). \end{aligned}$$

This says that after an action happens, every agent learns that it has happened. This is formalized by ensuring that any situation that is K-related to $do(a, s)$ must have a as the last action performed. Since action precondition axioms and successor-state axioms are assumed to be common knowledge, agents also learn the preconditions and effects of actions – this is usually called *knowledge update*. Moreover, if the action is a sensing action, the agent performing it acquires knowledge of the associated proposition or term. Furthermore, if the action involves someone informing agt whether Ψ holds, then agt comes to know the truth-value of Ψ afterwards. Finally, if the action involves someone informing agt who/what θ is, then agt knows the value of θ afterwards. Note that this axiom only handles knowledge expansion, not revision. Handling belief revision complicates the framework somewhat, and therefore I focus on knowledge rather than belief. Also, the above axiom only handles completely accurate sensors. After performing a sensing action, the agent learns whether the associated proposition actually holds. See [202, 195] for a treatment of belief revision in the situation calculus and [195] for an account that deals with noisy sensors.

I illustrate the successor-state axiom for K using the scenario in Figure 3.1. In this figure, situations are nodes in the graph, and the edges are labeled by actions. Part of the K-relation is represented by the ovals around the nodes. If a situation s appears in the same box as another situation s', then $K(agt, s', s)$. Finally, in this figure s denotes



(A) Regular Actions

(B) Knowledge Producing Actions

Figure 3.1: An Example of Knowledge Change

the actual situation, i.e. the one representing the true state of the world. First, consider the case for knowledge expansion due to regular actions, as depicted in Figure 3.1(A). Assume that initially s, s_1, and s_2 are accessible from each other. Then after a happens in s, according to the successor-state axiom for K, only $do(a, s), do(a, s_1)$, and $do(a, s_2)$ will be accessible from $do(a, s)$, but not $do(b, s_2)$, etc. Thus, in $do(a, s)$ the agent knows that the action a has just happened and knows that its effects hold. If a makes P become true, then the agent knows that P holds afterwards. Next, consider the case for knowledge expansion as a result of knowledge producing actions, as illustrated in Figure 3.1(B). Assume that initially s, s_1, and s_2 are in the same equivalence class w.r.t. K, and that $\Phi(s), \Phi(s_1)$, and $\neg\Phi(s_2)$. Then after the agent senses the value of Φ in s, according to the successor-state axiom for K, only $do(sense_\Phi, s)$ and $do(sense_\Phi, s_1)$ will be K-accessible from $do(sense_\Phi, s)$, but not $do(sense_\Phi, s_2)$. Since Φ holds in all situations that are K-accessible from $do(sense_\Phi, s)$, the agent will thus know that Φ in $do(sense_\Phi, s)$.

Henceforth, I will refer to the set of axioms for modeling knowledge (i.e. Axioms 3.4.1-3.4.10) as $\mathcal{D}_{know}$. See Chapter 2 for a review of other dynamic epistemic logics/frameworks.

3.5 Infinite Paths in the Situation Calculus

In this section, I set the stage for formalizing agent motivation by introducing a new sort of *infinite paths* in the situation calculus.

Agents' goals are future oriented.[17] For example, an agent might have a goal to eventually achieve some property. Therefore unlike knowledge formulae that take a single situation as argument, goal formulae are evaluated over a (possibly infinite) sequence of consecutive situations, i.e. a path. While some work has been done to capture the notion of paths in the situation calculus, all of these approaches have drawbacks. I discuss some of these here; see the end of this section for a discussion of other

[17] I use the word "goal" here as an umbrella term for any motivational operator, including goals, desires, intentions, etc. See Chapter 2 for the precise distinction between these notions.

related work. While specifying agents' goals and behavior, Shapiro [194] considers only finite paths. He formalized a finite path using a pair of situations representing the beginning state and the ending state of the path. Unfortunately, a temporal framework based on such finite paths has limited expressiveness and can't capture arbitrary temporally extended formulae, e.g. the goal to maintain a property ϕ indefinitely far in the future, $\Box\phi$. Also, quantification over these finite paths requires dealing with a pair of situations explicitly which is somewhat clumsy. Lespérance et al. [134] on the other hand looked at infinite paths. They introduced the notion of *action selection functions* (also called *strategies* in [198]), which are mappings from situations to primitive actions. The idea is that given a situation s, a strategy σ prescribes an action that the agent must perform in s if she were to follow the path induced by this strategy. An infinite path can then be formalized as a tuple (s, σ), where s is the starting situation of the path, and σ is a strategy that defines an infinite sequence of situations by specifying an action for every situation starting from s. Their account however does not have paths as a sort and thus does not allow for first-order quantification over paths.

To support modeling temporally extended goals, I adopt Lespérance et al.'s notion of infinite paths. Following [134], I only consider "realistic" paths; paths involving non-executable actions cannot really occur as they are not realistic. Thus a path in my framework is essentially an infinite sequence of situations, where each situation along the path can be reached by performing some *executable* action in the preceding situation. To allow (first-order) quantification over infinite paths, I in addition introduce a new sort called paths in the language with (possibly sub/super-scripted) variables p ranging over paths. In the next subsection, I give an axiomatization for infinite paths.

Thus my formalization of infinite paths is more general than Shapiro's finite paths. Arbitrary temporally extended formulae such as unbounded maintenance goals can be interpreted using my paths. Moreover, my account is simpler than that of Lespérance et al., and unlike them, I allow quantification over paths, which makes my language easier to use.

Before delving into the technical details, let me point out some notational conventions that I adopt. I will use both state and path formulae. I use uppercase Greek letters and lowercase Greek letters to denote state formulae and path formulae, respectively. A state formula $\Phi(s)$ is a formula that has a free situation variable s in it, whereas a path formula $\phi(p)$ is one that has a free path variable p. State formulae are used in the context of knowledge while path formulae are used in that of goals. As with a state formula that will usually contain a situation constant now, a path formula ϕ will usually contain a path constant/placeholder $path$ that stands for the path over which ϕ must be evaluated. $\phi(p)$ denotes the formula that can be obtained by replacing $path$ with p in ϕ. I often suppress the path variable p in a path formula $\phi(p)$ when the intent is clear from the context.

3.5.1 Axiomatization of Infinite Paths

I now give my axiomatization for infinite paths. I have a predicate OnPath(p, s), meaning that the situation s is on path p. Also, the abbreviation Starts(p, s) means that s is the starting situation of path p. A path p starts with s if and only if s is the earliest situation on p:

Definition 3.5.1.

$$\text{Starts}(p, s) \overset{\text{def}}{=} \text{OnPath}(p, s) \wedge \forall s'.\ \text{OnPath}(p, s') \supset s \preceq s'.$$

As shown in Lespérance et al., one can use action selection functions (ASFs) to model infinite paths. Recall that ASFs or strategies are mappings from situations to primitive actions. The idea is that given a situation s, an ASF F prescribes an action that the agent must perform in s if she were to follow the path induced by this strategy. An infinite path can then be formalized as a tuple (s, F), where s is the starting situation of the path, and F is a strategy that defines an infinite sequence of situations by specifying an action for every situation starting from s. Thus, one way of axiomatizing paths is by making them correspond to such pairs (s, F):

Axiom 3.5.2.

$$(i).\ \forall p.\ (\exists F, s.\ \text{Executable}(F, s) \land \forall s'.\ \text{OnPath}(p, s') \equiv \text{OnPathASF}(F, s, s')),$$

$$(ii).\ \forall F, s.\ \text{Executable}(F, s) \supset$$

$$(\exists p.\ \text{Starts}(p, s) \land \forall s'.\ \text{OnPathASF}(F, s, s') \equiv \text{OnPath}(p, s')).$$

This second-order axiom says that for every path p, there is an action selection function F and a situation s such that F starting in s is executable, and that F produces exactly the same sequence of situations on p starting from situation s. Also, for every executable action selection function F and situation s, there is a path p that starts with s and that corresponds exactly to the sequence of situations produced by F starting from s. Here, OnPathASF(F, s, s') means that the situation sequence defined by (s, F) includes the situation s':

Definition 3.5.3.

$$\text{OnPathASF}(F, s, s') \overset{\text{def}}{=} s \preceq s' \land \forall a, s^*.\ s \prec do(a, s^*) \preceq s' \supset F(s^*) = a.$$

Also, the situation sequence encoded by a strategy F and a starting situation s is executable if and only if s is executable, and for all situations s' on this sequence, the action selected by F in s' is executable in s'.

Definition 3.5.4.

$$\text{Executable}(F, s) \overset{\text{def}}{=} \text{Executable}(s) \land \forall s'.\ \text{OnPathASF}(F, s, s') \supset \text{Poss}(F(s'), s').$$

Another axiom is needed to state that different situation sequences represent different paths.

Axiom 3.5.5.

$$\forall p, p'. \ (\forall s. \ \text{OnPath}(p, s) \equiv \text{OnPath}(p', s)) \equiv p = p'.$$

Note that, for every situation s on a path, there must be an action that is possible in s:

$$\forall p, s. \ \text{OnPath}(p, s) \supset \exists a. \ \text{Poss}(a, s).$$

I consider that situations where no action is possible are "artificial". One can always introduce a dummy action $noOp$ that has the precondition that True, and consequently is always executable. Taking paths to be sequences of executable situations means that there may be infinite sequences of successor situations that are not paths; even if the situations on a prefix of a sequence are executable, the presence of a non-executable situation in the sequence means that it is not a path. One could easily modify the above axiomatization to include paths with non-executable situations, and identify the subset of such paths that are executable.

Also, while I focus on infinite paths, finite (executable) paths can be viewed as prefixes of paths since a finite path can always be extended to an infinite one, e.g. by extending the prefix with an infinite sequence of $noOp$ actions.

I next define some useful path-related constructs that I will need to use. First, I define a set of Linear Temporal Logic (LTL) operators [169, 64].[18] Formulae defined using these operators will be evaluated w.r.t. a path p and a time index/situation s on p. In these, I will use the starting situation s of the path p as the time index, and since s can be obtained from p, I will suppress this index.[19] I say that $\bigcirc\Phi(p)$, i.e. Φ holds *next* over a path p if Φ holds in the successor to the starting situation of p:

Definition 3.5.6.

$$\bigcirc\Phi(p) \overset{\text{def}}{=} \exists s, a.\ \text{Starts}(p, s) \wedge \text{OnPath}(p, do(a, s)) \wedge \Phi(do(a, s)).$$

$\Phi\ \mathcal{U}\ \Psi$ (Φ *until* Ψ) holds over a path p if there is a situation s' on p in which Ψ holds, and Φ continuously holds in every situation from the starting situation of p until s':

Definition 3.5.7.

$$(\Phi\ \mathcal{U}\ \Psi)(p) \overset{\text{def}}{=} \exists s, s'.\ \text{Starts}(p, s) \wedge \text{OnPath}(p, s') \wedge \Psi(s') \wedge \forall s^*.\ s \preceq s^* \prec s' \supset \Phi(s^*).$$

Other LTL operators can be defined as usual, e.g. *eventually* Φ (denoted by $\Diamond\Phi$), *always* Φ (denoted by $\Box\Phi$), Φ *unless* Ψ (denoted by $\Phi\ \mathcal{W}\ \Psi$), Φ *before* Ψ (denoted by $\Phi\ \mathcal{B}\ \Psi$), etc.

[18]In addition, Khan and Lespérance [121] show how arbitrary CTL* [66] formulae can be interpreted over situation calculus with paths defined here.

[19]Note that one can evaluate a temporal formula w.r.t. any time index/situation s along a path p just by evaluating it w.r.t. the suffix p' of p that starts with s (see Definition 3.5.16). Also, the following semantics closely corresponds to the one given by Emerson [64].

Definition 3.5.8.

$$\Diamond\Phi(p) \overset{\text{def}}{=} (\text{True}\, \mathcal{U}\ \Phi)(p).$$

Definition 3.5.9.

$$\Box\Phi(p) \overset{\text{def}}{=} \neg\Diamond\neg\Phi(p).$$

Definition 3.5.10.

$$(\Phi\ \mathcal{W}\ \Psi)(p) \overset{\text{def}}{=} (\Phi\, \mathcal{U}\ \Psi)(p) \vee \Box(\Phi \wedge \neg\Psi)(p).$$

Definition 3.5.11.

$$(\Phi\ \mathcal{B}\ \Psi)(p) \overset{\text{def}}{=} \neg(\neg\Phi\, \mathcal{U}\ \Psi)(p).$$

Let us also introduce a few more definitions. Firstly, I say that ϕ *is weakly inevitable in* s if ϕ holds over all paths that start with s:

Definition 3.5.12.

$$\text{WeaklyInevitable}(\phi, s) \overset{\text{def}}{=} \forall p.\ \text{Starts}(p, s) \supset \phi(p).$$

Secondly, I say that ϕ *is strongly inevitable in* s if ϕ is weakly inevitable in all situations that have the same history as s:

Definition 3.5.13.

$$\text{StronglyInevitable}(\phi, s) \overset{\text{def}}{=} \forall s'.\ \text{SameHist}(s', s) \supset \text{WeaklyInevitable}(\phi, s').$$

Thus, ϕ is strongly inevitable in situation s if ϕ holds over all paths that start with a situation that has the same action history as s.

Thirdly, an agent *knows in s that ϕ is inevitable* if she knows that ϕ is weakly inevitable in s, i.e., ϕ holds over all paths that start with a K-accessible situation in s:

Definition 3.5.14.

$$\text{KInevitable}(\phi, s) \overset{\text{def}}{=} \text{Know}(\text{WeaklyInevitable}(\phi, now), s).$$

Note that $\text{KInevitable}(\phi, s)$ is similar to $\text{StronglyInevitable}(\phi, s)$, except for the fact that in this case ϕ is weakly inevitable only in the situations that are knowledge accessible from s – a subset of the set of situations that share the same history with s.

An agent *knows in s that ϕ is impossible* if she knows that $\neg\phi$ is inevitable in s:

Definition 3.5.15.

$$\text{KImpossible}(\phi, s) \overset{\text{def}}{=} \text{KInevitable}(\neg\phi, s).$$

Finally, I define what it means for a path p' to be a *suffix* of another path p w.r.t. a situation s:

Definition 3.5.16.

$$\begin{aligned}\text{Suffix}(p', p, s) \overset{\text{def}}{=} {} & \text{OnPath}(p, s) \land \text{Starts}(p', s) \\ & \land \forall s'.\ s \preceq s' \supset (\text{OnPath}(p, s') \equiv \text{OnPath}(p', s')).\end{aligned}$$

That is, a path p' is a suffix of another path p w.r.t. a situation s if and only if s is on p, and p' which starts with s, contains exactly the same situations as p starting from s.

3.5.2 Properties

I now show some properties of my axiomatization of paths. All my arguments and claims in the proofs of these will be semantic in nature. I will use the following lemmata in these proofs (here Σ is the set of foundational axioms).

Σ entails that all initial situations are executable:

Lemma 3.5.17.

$$\Sigma \models \forall s.\ \text{Init}(s) \supset \text{Executable}(s).$$

Proof. Follows from Axiom 3.2.5 and Definitions 3.2.1 and 3.3.1. $\square$

Σ entails that doing an action yields a different situation:

Lemma 3.5.18.

$$\Sigma \models \forall a, s.\ s \neq do(a, s).$$

Proof. See Proposition 2.4.1 in [194]. $\square$

Σ entails that a situation s strictly precedes the situation that results from doing an action in s:

Lemma 3.5.19.

$$\Sigma \models \forall a, s.\ s \prec do(a, s).$$

Proof. See Proposition 2.4.2 in [194]. □

Σ entails that $\prec$ is transitive:

Lemma 3.5.20.

$$\Sigma \models \forall s, s_1, s_2.\ s \prec s_1 \wedge s_1 \prec s_2 \supset s \prec s_2.$$

Proof. See Proposition 2.4.6 in [194]. □

Σ entails that $\prec$ is irreflexive:

Lemma 3.5.21.

$$\Sigma \models \forall s.\ s \not\prec s.$$

Proof. See Proposition 2.4.7 in [194]. □

Σ entails that $\prec$ is asymmetric:

Lemma 3.5.22.

$$\Sigma \models \forall s_1, s_2.\ s_1 \prec s_2 \supset \neg(s_2 \prec s_1).$$

Proof. Follows from the transitivity of $\prec$ (i.e. Lemma 3.5.20) and irreflexivity of $\prec$ (i.e. Lemma 3.5.21). □

Σ entails that if a situation strictly precedes another situation, then they are different:

Lemma 3.5.23.

$$\Sigma \models \forall s, s'.\ s \prec s' \supset s \neq s'.$$

Proof. (By contradiction) Fix situations S_1 and S_2 and assume that $S_1 \prec S_2$ and $S_1 = S_2$. If we substitute S_1 for S_2 in the former, we have $S_1 \prec S_1$, but this is contradictory to Lemma 3.5.21. □

Σ entails that the result of doing a in s does not precede s:

Lemma 3.5.24.

$$\Sigma \models \forall a, s.\ do(a, s) \not\prec s.$$

Proof. See Proposition 2.4.9 in [194]. □

Σ entails that a situation s precedes doing an action in s:

Lemma 3.5.25.

$$\Sigma \models \forall a, s.\ s \preceq do(a, s).$$

Proof. See Corollary 2.4.3 in [194]. □

Σ entails that $\preceq$ is reflexive:

Lemma 3.5.26.

$$\Sigma \models \forall s.\ s \preceq s.$$

Proof. Trivial. $\square$

Σ entails that $\preceq$ is antisymmetric:

Lemma 3.5.27.

$$\Sigma \models \forall s, s'.\ s \preceq s' \wedge s' \preceq s \supset s = s'.$$

Proof. See Proposition 2.4.11 in [194]. $\square$

Σ entails that $\preceq$ is transitive:

Lemma 3.5.28.

$$\Sigma \models \forall s, s_1, s_2.\ s \preceq s_1 \wedge s_1 \preceq s_2 \supset s \preceq s_2.$$

Proof. See Proposition 2.4.12 in [194]. $\square$

Σ entails that if $do(a, s)$ is executable, then it is possible to execute a in s, and s is executable:

Lemma 3.5.29.

$$\Sigma \models \forall a, s.\ \text{Executable}(do(a, s)) \supset \text{Poss}(a, s) \wedge \text{Executable}(s).$$

Proof. See Proposition 2.4.16 in [194]. $\square$

Σ entails that if two situations $do(a, s)$ and $do(b, s)$ obtained by performing two actions a and b in the same situation s each precedes a third situation s', then a and b represent the same action.

Lemma 3.5.30.

$$\Sigma \models \forall a, b, s, s'. \ (do(a, s) \preceq s' \land do(b, s) \preceq s') \supset a = b.$$

Proof. (By induction on s') For the base case, fix A_1, B_1, S_1, and S'_1, and assume that:

$$do(A_1, S_1) = S'_1,$$

$$do(B_1, S_1) = S'_1.$$

From this, we have:

$$do(A_1, S_1) = do(B_1, S_1).$$

From this and Axiom 3.2.3, we have $A_1 = B_1$.

For the inductive hypothesis, fix S''_1 and assume that:

$$(do(A_1, S_1) \preceq S''_1 \land do(B_1, S_1) \preceq S''_1) \supset A_1 = B_1. \tag{3.1}$$

Fix action C_1. We have to show that:

$$(do(A_1, S_1) \preceq do(C_1, S''_1) \land do(B_1, S_1) \preceq do(C_1, S''_1)) \supset A_1 = B_1.$$

Assume that $(do(A_1, S_1) \preceq do(C_1, S''_1) \land do(B_1, S_1) \preceq do(C_1, S''_1))$. Then we have 4 cases to consider.

Case 1. $do(A_1, S_1) = do(C_1, S_1'')$ and $do(B_1, S_1) = do(C_1, S_1'')$. In this case, by Axiom 3.2.3, we have $A_1 = C_1$ and $B_1 = C_1$, and thus $A_1 = B_1$.

Case 2. $do(A_1, S_1) = do(C_1, S_1'')$ and $do(B_1, S_1) \prec do(C_1, S_1'')$. From the former and Axiom 3.2.3, we have $S_1 = S_1''$. If we substitute S_1 for S_1'' in the latter, we have $do(B_1, S_1) \prec do(C_1, S_1)$. From this, Axiom 3.2.5, and the fact that do is a function, we have $do(B_1, S_1) \preceq S_1$; but by Definition 3.2.6 and Lemmata 3.5.18 and 3.5.24, this is impossible.

Case 3. $do(A_1, S_1) \prec do(C_1, S_1'')$ and $do(B_1, S_1) = do(C_1, S_1'')$. As in Case 2, this case is also not possible.

Case 4. $do(A_1, S_1) \prec do(C_1, S_1'')$ and $do(B_1, S_1) \prec do(C_1, S_1'')$. From these, Axiom 3.2.5, and the fact that do is a function, we have:

$$do(A_1, S_1) \preceq S_1'' \wedge do(B_1, S_1) \preceq S_1''.$$

Thus we can apply the inductive hypothesis (3.1), which gives us $A_1 = B_1$. □

I say that two situations are co-linear if they are the same or if one of them strictly precedes the other. Assume that s_1 and s_2 are successors of two different situations, obtained by performing two different actions a and b, respectively, in the same situation s. Σ entails that s_1 and s_2 are not co-linear.

Lemma 3.5.31.

$$\Sigma \models \forall s, s_1, s_2, a, b.\ a \neq b \wedge do(a, s) \preceq s_1 \wedge do(b, s) \preceq s_2 \supset$$

$$s_1 \neq s_2 \wedge s_1 \not\prec s_2 \wedge s_2 \not\prec s_1.$$

Proof. (By contradiction) Fix A_1, B_1, S_1, S_1^1, and S_1^2 and assume that:

$$A_1 \neq B_1, \tag{3.2}$$

$$do(A_1, S_1) \preceq S_1^1 \wedge do(B_1, S_1) \preceq S_1^2, \text{ and} \tag{3.3}$$

$$S_1^1 = S_1^2 \vee S_1^1 \prec S_1^2 \vee S_1^2 \prec S_1^1. \tag{3.4}$$

Now, (3.3) above gives us the following four cases:

Case 1. Assume that:

$$S_1^1 = do(A_1, S_1), \text{ and} \tag{3.5}$$

$$S_1^2 = do(B_1, S_1). \tag{3.6}$$

From these, (3.2), and Axiom 3.2.3, we have:

$$S_1^1 \neq S_1^2. \tag{3.7}$$

Thus from (3.4) and (3.7), we have: $S_1^1 \prec S_1^2 \vee S_1^2 \prec S_1^1$. Assume that $S_1^1 \prec S_1^2$. Then from this, (3.5), and (3.6), we have: $do(A_1, S_1) \prec do(B_1, S_1)$. From this, Axiom 3.2.5, and the fact that do is a function, we have: $do(A_1, S_1) \preceq S_1$. From this and Definition

3.2.6, it follows that either $do(A_1, S_1) = S_1$ or $do(A_1, S_1) \prec S_1$. But by Lemma 3.5.18, the former is impossible. By Lemma 3.5.24, the latter is also impossible. It thus follows that:

$$S_1^1 \not\prec S_1^2. \tag{3.8}$$

Similarly, it can be shown that:

$$S_1^2 \not\prec S_1^1. \tag{3.9}$$

But (3.7), (3.8), and (3.9) is contradictory to (3.4).

Case 2a. Assume that:

$$S_1^1 = do(A_1, S_1), \text{ and} \tag{3.10}$$

$$do(B_1, S_1) \prec S_1^2. \tag{3.11}$$

I will show that $\neg(S_1^1 = S_1^2 \vee S_1^1 \prec S_1^2 \vee S_1^2 \prec S_1^1)$ by going over each case, one at a time. First, suppose that $S_1^1 = S_1^2$. From this and (3.10), we have $S_1^2 = do(A_1, S_1)$. From this and (3.11), we have $do(B_1, S_1) \prec do(A_1, S_1)$. From this, Axiom 3.2.5, and the fact that do is a function, we have $do(B_1, S_1) \preceq S_1$. But again, this is impossible by Definition 3.2.6 and Lemmata 3.5.18 and 3.5.24. Thus we have:

$$S_1^1 \neq S_1^2. \tag{3.12}$$

Now suppose that $S_1^1 \prec S_1^2$. Then from this and (3.10), we have:

$$do(A_1, S_1) \prec S_1^2.$$

From this, (3.11), and Definition 3.2.6, we have:

$$do(A_1, S_1) \preceq S_1^2 \wedge do(B_1, S_1) \preceq S_1^2.$$

From this and Lemma 3.5.30, we have $A_1 = B_1$, which is contradictory to (3.2). Thus we have:

$$S_1^1 \not\prec S_1^2. \tag{3.13}$$

Finally, suppose that $S_1^2 \prec S_1^1$. Then from this and (3.10), we have $S_1^2 \prec do(A_1, S_1)$. From (3.11), this, and transitivity of $\prec$ (i.e. Lemma 3.5.20), we have $do(B_1, S_1) \prec do(A_1, S_1)$. But as shown above, this is impossible. Thus:

$$S_1^2 \not\prec S_1^1. \tag{3.14}$$

But (3.12), (3.13), and (3.14) is contradictory to (3.4).

Case 2b. Assume that:

$$do(A_1, S_1) \prec S_1^1, \text{ and} \tag{3.15}$$

$$S_1^2 = do(B_1, S_1). \tag{3.16}$$

The proof for this case is similar to that of Case 2a.

Case 3. Assume that:

$$do(A_1, S_1) \prec S_1^1, \text{ and} \tag{3.17}$$

$$do(B_1, S_1) \prec S_1^2. \tag{3.18}$$

Again, I will show that $\neg(S_1^1 = S_1^2 \vee S_1^1 \prec S_1^2 \vee S_1^2 \prec S_1^1)$ by going over each case separately. First, assume that $S_1^1 = S_1^2$. From this and (3.18), we have $do(B_1, S_1) \prec S_1^1$. From this and (3.17), we have:

$$do(A_1, S_1) \prec S_1^1 \wedge do(B_1, S_1) \prec S_1^1.$$

From this and Definition 3.2.6, we have:

$$do(A_1, S_1) \preceq S_1^1 \wedge do(B_1, S_1) \preceq S_1^1.$$

From this and Lemma 3.5.30, we have $A_1 = B_1$, which is contradictory to (3.2). Thus we have:

$$S_1^2 \neq S_1^1. \tag{3.19}$$

Next, assume that $S_1^1 \prec S_1^2$. Then by this, (3.17), and transitivity of $\prec$ (i.e. Lemma 3.5.20), we have $do(A_1, S_1) \prec S_1^2$. From this, (3.18), and Definition 3.2.6, we have:

$$do(A_1, S_1) \preceq S_1^2 \wedge do(B_1, S_1) \preceq S_1^2.$$

From this and Lemma 3.5.30, we have $A_1 = B_1$, which is contradictory to (3.2). Thus we have:

$$S_1^1 \not\prec S_1^2. \tag{3.20}$$

Finally, assume that $S_1^2 \prec S_1^1$. The proof for this case is similar to the above. Hence we have:

$$S_1^2 \not\prec S_1^1. \tag{3.21}$$

But (3.19), (3.20), and (3.21) is contradictory to (3.4). □

If two situations are both preceded by a third situation and they are not co-linear, then there must be two different situations that precede them, and these situations can be obtained by performing two different actions in the same situation.

Lemma 3.5.32.

$$\Sigma \models \forall s, s_1, s_2.\ s \preceq s_1 \land s \preceq s_2 \land \neg(s_1 = s_2 \lor s_1 \prec s_2 \lor s_2 \prec s_1) \supset$$
$$\exists s', a_1, a_2.\ s \preceq s' \land do(a_1, s') \preceq s_1 \land do(a_2, s') \preceq s_2 \land a_1 \neq a_2.$$

Proof Sketch. Fix S_1, S_1^1, and S_1^2, and assume that:

$$S_1 \preceq S_1^1, \tag{3.22}$$

$$S_1 \preceq S_1^2, \text{ and} \tag{3.23}$$

$$\neg(S_1^1 = S_1^2 \lor S_1^1 \prec S_1^2 \lor S_1^2 \prec S_1^1). \tag{3.24}$$

Now (3.22) and (3.23) above give us 4 cases.

Case 1. Assume that $S_1 = S_1^1$ and $S_1 = S_1^2$. Then we have $S_1^1 = S_1^2$; but then this case is ruled out by (3.24).

Case 2. Assume that $S_1 = S_1^1$ and $S_1 \prec S_1^2$. Then we have $S_1^1 \prec S_1^2$; but then this case too is ruled out by (3.24).

Case 3. Assume that $S_1 \prec S_1^1$ and $S_1 = S_1^2$. Then we have $S_1^2 \prec S_1^1$; again by (3.24), this is also impossible.

Case 4. Assume that:

$$S_1 \prec S_1^1, \text{ and} \tag{3.25}$$

$$S_1 \prec S_1^2. \tag{3.26}$$

Consider the path from S_1 to S_1^1: there must be a situation s' such that $S_1 \prec s' \preceq S_1^1$ and $\neg(s' \preceq S_1^2)$, otherwise S_1^1 and S_1^2 are colinear, contradicting (3.24). Let S' be the unique situation such that:

$$S_1 \prec S' \preceq S_1^1, \tag{3.27}$$

$$\neg(S' \preceq S_1^2), \text{ and} \tag{3.28}$$

$$\forall s^*.\ S_1 \preceq s^* \prec S' \supset s^* \prec S_1^2. \tag{3.29}$$

From (3.27), we have $S_1 \prec S'$. From this and Axiom 3.2.5, it follows that there is an action A_1 and situation S'' such that:

$$S' = do(A_1, S''), \text{ and} \tag{3.30}$$

$$S_1 \preceq S''. \tag{3.31}$$

From (3.30) and (3.27), we have:

$$do(A_1, S'') \preceq S_1^1. \tag{3.32}$$

From (3.31), (3.30), and Axiom 3.2.5, we have: $S_1 \preceq S'' \prec S'$. By this and (3.29), we have: $S'' \prec S_1^2$. From this, (3.30), and (3.28), it follows that there exists an action A_2 such that:

$$A_1 \neq A_2, \text{ and} \tag{3.33}$$

$$do(A_2, S'') \preceq S_1^2. \tag{3.34}$$

The consequent thus follows from (3.31), (3.32), (3.34), and (3.33). $\square$

Σ entails that all non-initial situations are preceded by some initial situation.

Lemma 3.5.33.

$$\Sigma \models \forall s.\ \neg\text{Init}(s) \supset \exists s'.\ \text{Init}(s') \wedge s' \prec s.$$

Proof. By induction on s. $\square$

$\Sigma \cup \mathcal{D}_{know}$ entails that two K-accessible situations share the same action history:

Lemma 3.5.34.

$$\Sigma \cup \mathcal{D}_{know} \models \forall s, s'.\ K(s', s) \supset \text{SameHist}(s', s).$$

Proof. By induction on s. $\square$

$\Sigma \cup \mathcal{D}_{know}$ entails that given a situation s, all K-accessible situations in s are executable:

Lemma 3.5.35.

$$\Sigma \cup \mathcal{D}_{know} \models \forall s, s'.\ K(s', s) \supset \text{Executable}(s').$$

Proof. (By induction on s) In the base case where s is an initial situation, the result follows from Axiom 3.4.1 and Lemma 3.5.17.

For the inductive case, fix S_n and assume that:

$$\forall s'.\ K(s', S_n) \supset \text{Executable}(s').$$

Fix action A_n; we have to show that:

$$\forall s'.\ K(s', do(A_n, S_n)) \supset \text{Executable}(s').$$

Now from Axiom 3.4.10, it follows that:

$$\forall s^*.\ K(s^*, do(A_n, S_n)) \supset \exists s^{**}.\ K(s^{**}, S_n) \wedge s^* = do(A_n, s^{**}) \wedge \text{Poss}(A_n, s^{**}).$$

From this and the inductive hypothesis, it follows that:

$$\forall s^*.\ K(s^*, do(A_n, S_n)) \supset \exists s^{**}.\ s^* = do(A_n, s^{**}) \wedge \text{Executable}(s^{**}) \wedge \text{Poss}(A_n, s^{**}).$$

Finally, from this and Definition 3.3.1, it follows that:

$$\forall s^*.\ K(s^*, do(A_n, S_n)) \supset \exists s^{**}.\ s^* = do(A_n, s^{**}) \wedge \text{Executable}(do(A_n, s^{**})),$$

and thus that $\forall s^*.\ K(s^*, do(A_n, S_n)) \supset \text{Executable}(s^*)$. $\square$

Let $\mathcal{D}_{path}$ consist of the axiomatization for paths, i.e. Axioms 3.5.2 and 3.5.5. Then, my first property captures the conditions under which a situation can be extended to a path: $\Sigma \cup \mathcal{D}_{path}$ entails that for any executable situation, there is a path that starts with that situation, provided that for any situation there exists an executable action.

Proposition 3.5.36.

$$\Sigma \cup \mathcal{D}_{path} \models (\forall s'.\ \exists a.\ \text{Poss}(a, s')) \supset (\forall s.\ \text{Executable}(s) \supset \exists p.\ \text{Starts}(p, s)).$$

Proof. Fix situation S_1 and assume that $\forall s.\ \exists a.\ \text{Poss}(a, s)$ and $\text{Executable}(S_1)$. Construct an action selection function F_1 as follows:

$$F_1(s) = a, \qquad \text{for any situation } s,$$

where a is an arbitrary action that is executable in s, i.e. $\text{Poss}(a, s)$; by the antecedent, such an action is always available. Then by the antecedent, Definitions 3.5.4 and 3.5.3, and by construction of F_1, we have:

$$\text{Executable}(F_1, S_1).$$

The consequent follows from this and Axiom 3.5.2(ii). $\square$

Again, I maintain that situations with no executable actions are "artificial". Henceforth, I will use Proposition 3.5.36 without worrying about the antecedent that there is an executable action for any situation, i.e. that $\forall s'.\ \exists a.\ \text{Poss}(a, s')$; if this assumption does not hold for some theory, we can simply add a dummy action $noOp$ that has a True precondition and that is always executable.

Next, I prove some properties of the starting situation of a path. In particular, I can show that $\Sigma \cup \mathcal{D}_{path}$ entails that (a) any path starts with some situation, (b) the starting situation of any path is unique, and (c) the starting situation of any path is executable.

Proposition 3.5.37.

(a). $\Sigma \cup \mathcal{D}_{path} \models \forall p.\ \exists s.\ \text{Starts}(p, s)$,

(b). $\Sigma \cup \mathcal{D}_{path} \models \forall p, s, s'.\ \text{Starts}(p, s) \land \text{Starts}(p, s') \supset s = s'$,

(c). $\Sigma \cup \mathcal{D}_{path} \models \forall p, s.\ \text{Starts}(p, s) \supset \text{Executable}(s)$.

Proof. (a). Fix path P_1. By Axiom 3.5.2(i), there is a corresponding function F_1 and situation S_1 such that:

$$\forall s.\ \text{OnPath}(P_1, s) \equiv \text{OnPathASF}(F_1, S_1, s). \tag{3.35}$$

From (3.35) and Definition 3.5.3, it follows that:.

$$\forall s.\ \text{OnPath}(P_1, s) \equiv S_1 \preceq s \land \forall a, s^*.\ S_1 \prec do(a, s^*) \preceq s \supset F_1(s^*) = a. \tag{3.36}$$

From this and Definition 3.2.6, we have:

$$\text{OnPath}(P_1, S_1),\ \text{and} \tag{3.37}$$

$$\forall s.\ \text{OnPath}(P_1, s) \supset S_1 \preceq s. \tag{3.38}$$

From 3.37, 3.38, and Definition 3.5.1, it follows that Starts(P_1, S_1). □

(b). Fix path P_1 and starting situations S_1 and S_1'. By the antecedent, we have Starts(P_1, S_1). From this and Definition 3.5.1, we have:

$$\text{OnPath}(P_1, S_1), \tag{3.39}$$

$$\forall s.\ \text{OnPath}(P_1, s) \supset S_1 \preceq s. \tag{3.40}$$

Again, from the antecedent, we have Starts(P_1, S_1'). From this and Definition 3.5.1, we have:

$$\text{OnPath}(P_1, S_1'), \tag{3.41}$$

$$\forall s.\ \text{OnPath}(P_1, s) \supset S_1' \preceq s. \tag{3.42}$$

From 3.40 and 3.41, we have:

$$S_1 \preceq S_1'. \tag{3.43}$$

Moreover, from 3.39 and 3.42, we have:

$$S_1' \preceq S_1. \tag{3.44}$$

The consequent follows from 3.43, 3.44, and Lemma 3.5.27. □

(c). Fix path P_1. By Axiom 3.5.2(i), there is a corresponding function F_1 and situation S_1 such that:

$$\text{Executable}(F_1, S_1), \tag{3.45}$$

$$\forall s.\ \text{OnPath}(P_1, s) \equiv \text{OnPathASF}(F_1, S_1, s). \tag{3.46}$$

As in the proof of Proposition 3.5.37(a), from (3.46) and Definitions 3.5.1 and 3.5.3, it follows that Starts(P_1, S_1). Moreover, by this and Proposition 3.5.37(b), it follows that $\forall s.\ \text{Starts}(P_1, s) \equiv s = S_1$. Finally, from (3.45) and Definition 3.5.4, it follows that Executable(S_1). $\square$

The next two properties deal with the successor situation of a situation on a path that is also on the path. The first states that $\Sigma \cup \mathcal{D}_{path}$ entails that for any situation s on a path p, there is a successor situation $s' = do(a, s)$ on p, and s' can be reached from s by performing an executable action a.

Proposition 3.5.38.

$\Sigma \cup \mathcal{D}_{path} \models \forall p, s.\ \text{OnPath}(p, s) \supset \exists s', a.\ \text{OnPath}(p, s') \wedge s' = do(a, s) \wedge \text{Poss}(a, s)$.

Proof. Fix path P_1. By Axiom 3.5.2(i), there is a corresponding function F_1 and situation S_1 such that:

$$\text{Executable}(F_1, S_1), \tag{3.47}$$

$$\forall s.\ \text{OnPath}(P_1, s) \equiv \text{OnPathASF}(F_1, S_1, s). \tag{3.48}$$

Consider any situation S_n on P_1, i.e., OnPath(P_1, S_n). By (3.48), S_n must also be on the sequence defined by (S_1, F_1):

$$\text{OnPathASF}(F_1, S_1, S_n). \tag{3.49}$$

From (3.49) and Definition 3.5.3, we have:

$$S_1 \preceq S_n, \text{ and} \tag{3.50}$$

$$\forall a, s.\ S_1 \prec do(a, s) \preceq S_n, \supset F_1(s) = a. \tag{3.51}$$

Assume $F_1(S_n) = A_n$. Then from (3.50) and Lemmata 3.5.25 and 3.5.28, we have:

$$S_1 \preceq do(A_n, S_n). \tag{3.52}$$

From (3.52), (3.51), the assumption that $F_1(S_n) = A_n$, and Definition 3.5.3, it follows that the situation $do(A_n, S_n)$ must be on the sequence defined by (S_1, F_1):

$$\text{OnPathASF}(F_1, S_1, do(A_n, S_n)). \tag{3.53}$$

Also, by (3.53) and (3.48), $do(A_n, S_n)$ must be on path P_1:

$$\text{OnPath}(P_1, do(A_n, S_n)). \tag{3.54}$$

Finally, by (3.47), (3.53), and Definition 3.5.4, action A_n must have been executable in S_n:

$$\text{Poss}(A_n, S_n). \tag{3.55}$$

The proposition follows from (3.54) and (3.55). $\square$

Moreover, $\Sigma \cup \mathcal{D}_{path}$ entails that the successor situation of a situation on a path is unique.

Proposition 3.5.39.

$\Sigma \cup \mathcal{D}_{path} \models \forall p, s.\ \text{OnPath}(p, s) \wedge \text{OnPath}(p, do(a, s)) \wedge \text{OnPath}(p, do(b, s)) \supset a = b.$

Proof. (By contradiction) Fix path P_1. By Axiom 3.5.2(i), there is a corresponding function F_1 and situation S_1 such that:

$$\forall s.\ \text{OnPath}(P_1, s) \equiv \text{OnPathASF}(F_1, S_1, s). \tag{3.56}$$

Fix S_1^1, A_1, and A_2 and assume that:

$$\text{OnPath}(P_1, do(A_1, S_1^1)), \tag{3.57}$$

$$\text{OnPath}(P_1, do(A_2, S_1^1)), \text{ and} \tag{3.58}$$

$$A_1 \neq A_2. \tag{3.59}$$

From (3.56), (3.57), and Definition 3.5.3,it follows that:

$$\forall a, s^*.\ S_1 \prec do(a, s^*) \preceq do(A_1, S_1^1) \supset F_1(s^*) = a. \tag{3.60}$$

Similarly, from (3.56), (3.58), and Definition 3.5.3, it follows that:

$$\forall a, s^*.\ S_1 \prec do(a, s^*) \preceq do(A_2, S_1^1) \supset F_1(s^*) = a. \tag{3.61}$$

But from (3.60), (3.61), and (3.59), we have that:

$$F_1(S_1^1) = A_1 \wedge F_1(S_1^1) = A_2 \wedge A_1 \neq A_2,$$

which is contradictory to the fact that F_1 is a function. □

The next property deals with the uniqueness of paths: $\Sigma \cup \mathcal{D}_{path}$ entails that if $p \neq p'$, then there is a situation that is on path p but not on path p'.

Proposition 3.5.40.

$$\Sigma \cup \mathcal{D}_{path} \models \forall p, p'.\ p \neq p' \supset \exists s.\ (\text{OnPath}(p, s) \land \neg\text{OnPath}(p', s)).$$

Proof. Follows from Axiom 3.5.5. $\square$

I can also show that $\Sigma \cup \mathcal{D}_{path}$ entails that all situations on a path are executable.

Corollary 3.5.41.

$$\Sigma \cup \mathcal{D}_{path} \models \forall p, s.\ \text{OnPath}(p, s) \supset \text{Executable}(s).$$

Proof. (By induction on s) Fix path P_1. The base case follows from Propositions 3.5.37(b) and 3.5.37(c). For the inductive hypothesis, fix situation S_1 and assume that:

$$\text{OnPath}(P_1, S_1), \text{ and} \tag{3.62}$$

$$\text{Executable}(S_1). \tag{3.63}$$

From this and Propositions 3.5.38 and 3.5.39, it follows that there is a unique successor situation S_2 and action A_1 such that:

$$\text{OnPath}(P_1, S_2) \land S_2 = do(A_1, S_1), \text{ and} \tag{3.64}$$

$$\text{Poss}(A_1, S_1). \tag{3.65}$$

From (3.63), (3.65), and Definition 3.3.1, it follows that Executable$(do(A_1, S_1))$. $\square$

My next set of properties deal with the structure of situations on paths and shows that paths are essentially linear sequences of situations. First I have $\Sigma \cup \mathcal{D}_{path}$ entails that any pair of situations on the same path are co-linear:

Proposition 3.5.42.

$$\Sigma \cup \mathcal{D}_{path} \models \forall p, s, s'. \text{ OnPath}(p, s) \wedge \text{OnPath}(p, s') \supset s = s' \vee s \prec s' \vee s' \prec s.$$

Proof. (By contradiction) Fix path P_1. By Axiom 3.5.2(i), there is a corresponding function F_1 and situation S_1 such that:

$$\forall s. \text{ OnPath}(P_1, s) \equiv \text{OnPathASF}(F_1, S_1, s). \tag{3.66}$$

Fix S_m and S_n and assume that:

$$\text{OnPath}(P_1, S_m), \tag{3.67}$$

$$\text{OnPath}(P_1, S_n), \tag{3.68}$$

$$\neg(S_m = S_n \vee S_m \prec S_n \vee S_n \prec S_m). \tag{3.69}$$

From (3.66), (3.67), and Definition 3.5.3,it follows that:

$$S_1 \preceq S_m, \text{ and} \tag{3.70}$$

$$\forall a, s^*.\ S_1 \prec do(a, s^*) \preceq S_m \supset F_1(s^*) = a. \tag{3.71}$$

Similarly, from (3.66), (3.68), and Definition 3.5.3, it follows that:

$$S_1 \preceq S_n, \text{ and} \tag{3.72}$$

$$\forall a, s^*.\ S_1 \prec do(a, s^*) \preceq S_n \supset F_1(s^*) = a. \tag{3.73}$$

From (3.69), (3.70), (3.72), and Lemma 3.5.32, it follows that there is a situation S_2 and actions A_1 and A_2 such that:

$$S_1 \preceq S_2 \wedge do(A_1, S_2) \preceq S_m \wedge do(A_2, S_2) \preceq S_n \wedge A_1 \neq A_2. \tag{3.74}$$

But from (3.71), (3.73), and (3.74), we have that:

$$F_1(S_2) = A_1 \wedge F_1(S_2) = A_2 \wedge A_1 \neq A_2,$$

which is contradictory to the fact that F_1 is a function. □

Secondly, I have $\Sigma \cup \mathcal{D}_{path}$ entails that if situations s and s' are on a given path p, then all situations in the interval defined by these two situations are also on p.

Proposition 3.5.43.

$\Sigma \cup \mathcal{D}_{path} \models \forall p, s, s', s^*.\ \text{OnPath}(p, s) \wedge \text{OnPath}(p, s') \wedge s \preceq s^* \preceq s' \supset \text{OnPath}(p, s^*).$

Proof. Fix path P_1. By Axiom 3.5.2(i), there is a corresponding function F_1 and situation S_1 such that:

$$\forall s.\ \text{OnPath}(P_1, s) \equiv \text{OnPathASF}(F_1, S_1, s). \tag{3.75}$$

Fix S_m, S_n, S_p and assume that:

$$\text{OnPath}(P_1, S_m), \tag{3.76}$$

$$\text{OnPath}(P_1, S_p), \tag{3.77}$$

$$S_m \preceq S_n \preceq S_p. \tag{3.78}$$

From (3.75) and (3.76), it follows that:

$$\text{OnPathASF}(F_1, S_1, S_m). \tag{3.79}$$

Similarly, from (3.75) and (3.77), it follows that:

$$\text{OnPathASF}(F_1, S_1, S_p). \tag{3.80}$$

From (3.79) and Definition 3.5.3, it follows that:

$$S_1 \preceq S_m. \tag{3.81}$$

From this, (3.78), and transitivity of $\preceq$ (i.e. Lemma 3.5.28), it follows that:

$$S_1 \preceq S_n. \tag{3.82}$$

From (3.80) and Definition 3.5.3, it follows that:

$$\forall a, s^*.\ S_1 \prec do(a, s^*) \preceq S_p \supset F_1(s^*) = a. \tag{3.83}$$

From this, (3.78), and (3.82), it follows that:

$$\forall a, s^*.\ S_1 \prec do(a, s^*) \preceq S_n \supset F_1(s^*) = a. \tag{3.84}$$

From (3.82), (3.84), and Definition 3.5.3, it follows that OnPathASF(F_1, S_1, S_n). The proposition follows from this and (3.75). □

Finally, I can show that $\Sigma \cup \mathcal{D}_{path}$ entails that two paths can share only one common prefix. Once they branch at some situation, they never merge after that.

Proposition 3.5.44.

$$\begin{aligned}\Sigma \cup \mathcal{D}_{path} \models \forall p_1, p_2, s, a, b, s_1, s_2.\ &\text{OnPath}(p_1, do(a, s)) \wedge \text{OnPath}(p_2, do(b, s))\\ &\wedge a \neq b \wedge s \prec s_1 \wedge s \prec s_2 \wedge \text{OnPath}(p_1, s_1) \wedge \text{OnPath}(p_2, s_2)\\ &\supset s_1 \neq s_2.\end{aligned}$$

Proof. (By contradiction) Fix $P_1, P_2, S_1, A_1, B_1, S_{11}$, and S_{12}, and assume that:

$$\text{OnPath}(P_1, do(A_1, S_1)), \tag{3.85}$$

$$\text{OnPath}(P_2, do(B_1, S_1)), \tag{3.86}$$

$$A_1 \neq B_1, \tag{3.87}$$

$$S_1 \prec S_{11}, \tag{3.88}$$

$$S_1 \prec S_{12}, \tag{3.89}$$

$$\text{OnPath}(P_1, S_{11}), \text{ and} \tag{3.90}$$

$$\text{OnPath}(P_2, S_{12}). \tag{3.91}$$

Also, assume that the consequent is false:

$$S_{11} = S_{12}. \tag{3.92}$$

From (3.85), (3.90), Proposition 3.5.42, and Definition 3.2.6, it follows that:

$$do(A_1, S_1) \preceq S_{11} \vee S_{11} \preceq do(A_1, S_1). \tag{3.93}$$

Now suppose that:

$$S_{11} \prec do(A_1, S_1).$$

Then by Axiom 3.2.5, we have:

$$\exists b, s.\ (do(A_1, S_1) = do(b, s) \wedge S_{11} \preceq s).$$

Then from this and Axiom 3.2.3, it follows that $S_{11} \preceq S_1$. By (3.88) and Lemma 3.5.23, it follows that $S_{11} \neq S_1$. Thus by Definition 3.2.6, we have:

$$S_{11} \prec S_1.$$

Since by Lemma 3.5.22, $\prec$ is asymmetric, it follows from this that $\neg(S_1 \prec S_{11})$. But this contradicts (3.88). Thus it follows that:

$$\neg(S_{11} \prec do(A_1, S_1)). \tag{3.94}$$

From (3.93) and (3.94), it follows that:

$$do(A_1, S_1) \preceq S_{11}. \tag{3.95}$$

Similarly, it can be shown that:

$$do(B_1, S_1) \preceq S_{12}. \tag{3.96}$$

Now from (3.92), we have $S_{11} = S_{12}$. But from (3.95), (3.96), (3.87), and Lemma 3.5.31, this is impossible. $\square$

The next few properties deal with suffixes and prefixes of a given path. The first of these states that $\Sigma \cup \mathcal{D}_{path}$ entails that for any situation s on a path p, there is a suffix of p that starts with s.

Proposition 3.5.45.

$$\Sigma \cup \mathcal{D}_{path} \models \forall p, s.\ \text{OnPath}(p, s) \supset \exists p'.\ \text{Suffix}(p', p, s).$$

Proof. Fix path P_1. By Axiom 3.5.2(i), there is a function F_1 and situation S_1 such that:

$$\text{Executable}(F_1, S_1), \tag{3.97}$$

$$\forall s.\ \text{OnPath}(P_1, s) \equiv \text{OnPathASF}(F_1, S_1, s). \tag{3.98}$$

Fix situation S_n such that:

$$\text{OnPath}(P_1, S_n). \tag{3.99}$$

We will show that there is a path P_n s.t. P_n, that starts with S_n, is a suffix of P_1.

Consider the pair (S_n, F_1). From (3.98) and (3.99), we have that:

$$\text{OnPathASF}(F_1, S_1, S_n). \tag{3.100}$$

From this and Definition 3.5.3, it follows that:

$$S_1 \preceq S_n. \tag{3.101}$$

From (3.100), (3.101), (3.97), and Definitions 3.5.4 and 3.3.1, we have:

$$\text{Executable}(F_1, S_n). \tag{3.102}$$

By (3.102) and Axiom 3.5.2(ii), it follows that there is a path P_n s.t.

$$\text{Starts}(P_n, S_n),\ \text{and} \tag{3.103}$$

$$\forall s.\ \text{OnPathASF}(F_1, S_n, s) \equiv \text{OnPath}(P_n, s). \tag{3.104}$$

Now, we need show that Suffix(P_n, P_1, S_n). From (3.98) and Definition 3.5.3, we have:

$$\forall s.\ \text{OnPath}(P_1, s) \equiv S_1 \preceq s \land \forall a, s^*.\ S_1 \prec do(a, s^*) \preceq s \supset F_1(s^*) = a. \tag{3.105}$$

Similarly, from (3.104) and Definition 3.5.3, we have:

$$\forall s.\ \text{OnPath}(P_n, s) \equiv S_n \preceq s \wedge \forall a, s^*.\ S_n \prec do(a, s^*) \preceq s \supset F_1(s^*) = a. \quad (3.106)$$

From (3.105), (3.106), (3.101), and (3.99), it follows that:

$$\forall s.\ S_n \preceq s \supset \text{OnPath}(P_1, s) \equiv \text{OnPath}(P_n, s). \quad (3.107)$$

Then Suffix(P_n, P_1, S_n) follows from (3.99), (3.103), (3.107), and Definition 3.5.16.

□

Secondly, I can show that given a path p with starting situation $do(a, s)$, $\Sigma \cup \mathcal{D}_{path}$ entails that there is a path p' s.t. p' starts with s, and p is a suffix of p' starting from $do(a, s)$.

Proposition 3.5.46.

$$\Sigma \cup \mathcal{D}_{path} \models \text{Starts}(p, do(a, s)) \supset \exists p'.\ \text{Starts}(p', s) \wedge \text{Suffix}(p, p', do(a, s)).$$

Proof. Fix P_1, A_1, and S_1 and assume that:

$$\text{Starts}(P_1, do(A_1, S_1)). \quad (3.108)$$

By Axiom 3.5.2(i), there is a function F_1 and situation S_2 such that:

$$\text{Executable}(F_1, S_2),\ \text{and} \quad (3.109)$$

$$\forall s.\ \text{OnPath}(P_1, s) \equiv \text{OnPathASF}(F_1, S_2, s). \tag{3.110}$$

From (3.110), Lemma 3.5.26, and Definition 3.5.3, we have:

$$\text{OnPath}(P_1, S_2). \tag{3.111}$$

Again, from (3.110) and Definition 3.5.3, we have:

$$\forall s.\ \text{OnPath}(P_1, s) \supset S_2 \preceq s. \tag{3.112}$$

From (3.111), (3.112), and Definition 3.5.1, we have that:

$$\text{Starts}(P_1, S_2). \tag{3.113}$$

From (3.108), (3.113), and Proposition 3.5.37(b), it follows that:

$$S_2 = do(A_1, S_1).$$

From this and (3.109) and (3.110), it follows that:

$$\text{Executable}(F_1, do(A_1, S_1)), \text{ and} \tag{3.114}$$

$$\forall s.\ \text{OnPath}(P_1, s) \equiv \text{OnPathASF}(F_1, do(A_1, S_1), s). \tag{3.115}$$

Now, consider the pair (S_1, F_1^1), where F_1^1 is defined as follows:

$$\begin{aligned} F_1^1(s) &= A_1, \text{ if } s = S_1 \\ &= F_1(s), \text{ otherwise }. \end{aligned}$$

From (3.114) and Definition 3.5.4, it follows that:

$$\text{Executable}(do(A_1, S_1)).$$

From this and Lemma 3.5.29, it follows that:

$$\text{Poss}(A_1, S_1), \text{ and} \tag{3.116}$$

$$\text{Executable}(S_1). \tag{3.117}$$

From (3.114), (3.116), Definition 3.5.4, and by definition of F_1^1, it follows that:

$$\forall s.\ \text{OnPathASF}(F_1^1, S_1, s) \supset \text{Poss}(F_1^1(s), s). \tag{3.118}$$

From (3.117), (3.118), and Definition 3.5.4, we have that:

$$\text{Executable}(F_1^1, S_1). \tag{3.119}$$

Now, by (3.119) and Axiom 3.5.2(ii), there is a path P_1^1 such that:

$$\text{Starts}(P_1^1, S_1), \text{ and} \tag{3.120}$$

$$\forall s.\ \text{OnPathASF}(F_1^1, S_1, s) \equiv \text{OnPath}(P_1^1, s). \tag{3.121}$$

We need to show that Suffix$(P_1, P_1^1, do(A_1, S_1))$. From Lemma 3.5.25, we have:

$$S_1 \preceq do(A_1, S_1). \tag{3.122}$$

Also, by definition of F_1^1, it follows that:

$$\forall a, s.\ S_1 \prec do(a, s) \preceq do(A_1, S_1) \supset F_1^1(s) = a. \tag{3.123}$$

From (3.122), (3.123), and Definition 3.5.3, it follows that:

$$\text{OnPathASF}(F_1^1, S_1, do(A_1, S_1)).$$

From this and (3.121), it follows that:

$$\text{OnPath}(P_1^1, do(A_1, S_1)). \tag{3.124}$$

From (3.115) and Definition 3.5.3, we have:

$$\begin{aligned}\forall s.\ \text{OnPath}(P_1, s) \equiv\ & do(A_1, S_1) \preceq s \\ & \wedge \forall a, s^*.\ do(A_1, S_1) \prec do(a, s^*) \preceq s \supset F_1(s^*) = a.\end{aligned} \tag{3.125}$$

Similarly, from (3.121) and Definition 3.5.3, we have:

$$\forall s.\ \text{OnPath}(P_1^1, s) \equiv S_1 \preceq s \wedge \forall a, s^*.\ S_1 \prec do(a, s^*) \preceq s \supset F_1^1(s^*) = a. \tag{3.126}$$

Note that, by Lemmata 3.5.19, 3.5.20, and 3.5.23, it follows that:

$$\forall s.\ do(A_1, S_1) \preceq s \supset s \neq S_1.$$

From this, (3.126), and definition of F_1^1, we have:

$$\begin{aligned}&\forall s.\ do(A_1, S_1) \preceq s \supset \\ &\qquad \text{OnPath}(P_1^1, s) \equiv S_1 \preceq s \wedge \forall a, s^*.\ S_1 \prec do(a, s^*) \preceq s \supset F_1(s^*) = a.\end{aligned} \tag{3.127}$$

From (3.125) and (3.127), it follows that:

$$\forall s.\ do(A_1, S_1) \preceq s \supset \text{OnPath}(P_1, s) \equiv \text{OnPath}(P_1^1, s). \tag{3.128}$$

From (3.124), (3.108), (3.128), and Definition 3.5.16, it follows that:

$$\text{Suffix}(P_1, P_1^1, do(A_1, S_1)). \tag{3.129}$$

The consequent follows from (3.120) and (3.129). $\square$

Finally, $\Sigma \cup \mathcal{D}_{path}$ entails that any path that starts with a non-initial situation can be extended in the past; formally, for all situations s_1 and s_2, if s_1 strictly precedes s_2 and there is a path p_2 that starts with s_2, then there must also exist a path p_1 such that p_1 starts with s_1 and p_2 is a suffix of p_1 starting from s_2.

Lemma 3.5.47.

$$\Sigma \cup \mathcal{D}_{path} \models \forall s_1, s_2, p_2.\ s_1 \prec s_2 \wedge \text{Starts}(p_2, s_2) \supset$$
$$\exists p_1.\ \text{Starts}(p_1, s_1) \wedge \text{Suffix}(p_2, p_1, s_2).$$

Proof. (By induction on situation s_2) For the base case, fix S_2^b such that $\text{Init}(S_2^b)$. Then by this, Definition 3.2.1, and Axiom 3.2.5, we have: $\neg\exists s.\ s \prec S_2^b$, and thus the antecedent is false and the thesis follows trivially.

For the inductive hypothesis, fix situation S_2 and assume that:

$$\forall s_1, p_2.\ s_1 \prec S_2 \wedge \text{Starts}(p_2, S_2) \supset \exists p_1.\ \text{Starts}(p_1, s_1) \wedge \text{Suffix}(p_2, p_1, S_2). \tag{3.130}$$

Fix A_2. We have to show that:

$$\forall s_1, p_2.\ s_1 \prec do(A_2, S_2) \wedge \text{Starts}(p_2, do(A_2, S_2)) \supset$$
$$\exists p_1.\ \text{Starts}(p_1, s_1) \wedge \text{Suffix}(p_2, p_1, do(A_2, S_2)).$$

Fix S_1 and P_2 and assume that:

$$S_1 \prec do(A_2, S_2),\ \text{and} \tag{3.131}$$

$$\text{Starts}(P_2, do(A_2, S_2)). \tag{3.132}$$

By (3.132) and Proposition 3.5.46, it follows that there is a path P_3 s.t.:

$$\text{Starts}(P_3, S_2) \wedge \text{Suffix}(P_2, P_3, do(A_2, S_2)). \tag{3.133}$$

Also by (3.131) and Axiom 3.2.5, we have:

$$\exists s, a.\ do(A_2, S_2) = do(a, s) \wedge S_1 \preceq s.$$

By this and Axiom 3.2.3, we have:

$$S_1 \preceq S_2.$$

By this and Definition 3.2.6, it follows that:

$$S_1 = S_2 \vee S_1 \prec S_2.$$

Case 1. Assume that $S_1 = S_2$. Then by (3.132) and Proposition 3.5.46, it follows that:

$$\exists p.\ \text{Starts}(p, S_1) \wedge \text{Suffix}(P_2, p, do(A_2, S_2)),$$

and we are done.

Case 2. Assume that $S_1 \prec S_2$. Then by this, (3.133), and (3.130), it follows that there is a path P_4 s.t.:

$$\text{Starts}(P_4, S_1) \wedge \text{Suffix}(P_3, P_4, S_2). \tag{3.134}$$

We will show that $\text{Suffix}(P_2, P_4, do(A_2, S_2))$. By (3.133) and Definition 3.5.16, we have:

$$\text{OnPath}(P_3, do(A_2, S_2)),\ \text{and} \tag{3.135}$$

$$\forall s'.\ do(A_2, S_2) \preceq s' \supset (\text{OnPath}(P_3, s') \equiv \text{OnPath}(P_2, s')). \tag{3.136}$$

By (3.134) and Definition 3.5.16, we have:

$$\forall s'.\ S_2 \preceq s' \supset (\text{OnPath}(P_3, s') \equiv \text{OnPath}(P_4, s')). \tag{3.137}$$

Since by Lemma 3.5.25, $S_2 \preceq do(A_2, S_2)$, it follows from (3.135) and (3.137) that:

$$\text{OnPath}(P_4, do(A_2, S_2)). \tag{3.138}$$

Also from (3.136) and (3.137), it follows that:

$$\forall s'.\ do(A_2, S_2) \preceq s' \supset (\text{OnPath}(P_2, s') \equiv \text{OnPath}(P_4, s')). \tag{3.139}$$

Finally, from (3.132), (3.138), (3.139), and Definition 3.5.16, we have:

$$\text{Suffix}(P_2, P_4, do(A_2, S_2)).$$

$\square$

3.5.3 Induction Principles

I now prove some second-order induction principles for paths and for situations in a path. First I have $\Sigma \cup \mathcal{D}_{path}$ entails that if some property Q holds for all paths that start with an initial situation, and if whenever Q holds for all paths that start with situation s, then it holds for all paths that start with any successor situation to s, then the property Q holds for all paths.

Theorem 3.5.48 (Induction on Paths)**.**

$$\begin{aligned}\Sigma \cup \mathcal{D}_{path} \models \forall Q.\ [&\{\forall s, p.\ \text{Init}(s) \wedge \text{Starts}(p, s) \supset Q(p)\} \wedge \\ &\{\forall a, s.\ (\forall p.\ \text{Starts}(p, s) \supset Q(p)) \\ &\qquad \supset (\forall p'.\ \text{Starts}(p', do(a, s)) \supset Q(p'))\}] \\ &\supset \forall p.\ Q(p).\end{aligned}$$

Proof. (By contradiction) Fix property Q_1 and assume:

$$\forall s, p.\ \text{Init}(s) \wedge \text{Starts}(p, s) \supset Q_1(p), \tag{3.140}$$

$$\forall a, s.\ (\forall p.\ \text{Starts}(p, s) \supset Q_1(p)) \supset (\forall p'.\ \text{Starts}(p', do(a, s)) \supset Q_1(p')). \tag{3.141}$$

Also assume that there is a path P_1 over which Q_1 is false:

$$\neg Q_1(P_1). \tag{3.142}$$

By Proposition 3.5.37(a,b), P_1 must start with some unique situation, call it S_1:

$$\text{Starts}(P_1, S_1). \tag{3.143}$$

We now prove by induction on s that:

$$\forall s, p.\ \text{Starts}(p, s) \supset Q_1(p).$$

For the base case where s is an initial situation, the thesis follows from (3.140).

For the inductive step, fix S_2 and assume that:

$$\forall p.\ \text{Starts}(p, S_2) \supset Q_1(p). \tag{3.144}$$

Take some arbitrary action A_1. It follows from (3.141) and (3.144) that:

$$\forall p.\ \text{Starts}(p, do(A_1, S_2)) \supset Q_1(p). \tag{3.145}$$

Thus by induction on s, we have:

$$\forall s, p.\ \text{Starts}(p, s) \supset Q_1(p). \tag{3.146}$$

By (3.146) and (3.143), it follows that $Q_1(p)$; but this is contradictory to (3.142). $\square$

Moreover, $\Sigma \cup \mathcal{D}_{path}$ entails that if some property Q holds for the starting situation of a given path p, and if whenever Q holds for a situation s on path p, then it holds for the successor situation to s on p, then the property Q holds for all situations on path p.

Theorem 3.5.49 (Induction on Situations in a Path)**.**

$$\begin{aligned}\Sigma \cup \mathcal{D}_{path} \models \forall p, Q.\ [\{\forall s.\ \text{Starts}(p, s) \supset Q(s)\} \wedge \\ \{\forall a, s.\ (\text{OnPath}(p, s) \wedge Q(s) \wedge \text{OnPath}(p, do(a, s))) \supset Q(do(a, s))\}] \\ \supset \forall s.\ \text{OnPath}(p, s) \supset Q(s).\end{aligned}$$

Proof. (By contradiction) Fix path P_1 and property Q_1 and assume:

$$\forall s.\ \text{Starts}(P_1, s) \supset Q_1(s), \tag{3.147}$$

$$\forall a, s.\ \text{OnPath}(P_1, s) \wedge Q_1(s) \wedge \text{OnPath}(P_1, do(a, s)) \supset Q_1(do(a, s)). \tag{3.148}$$

Also assume that there is a situation S_{P_1} on path P_1 over which Q_1 is false:

$$\text{OnPath}(P_1, S_{P_1}) \wedge \neg Q_1(S_{P_1}). \tag{3.149}$$

By Proposition 3.5.37(a,b), P_1 must start with some unique situation, call it S_1:

$$\text{Starts}(P_1, S_1). \tag{3.150}$$

From this and Definition 3.5.1, we have:

$$\text{OnPath}(P_1, S_1). \tag{3.151}$$

We now prove by induction on s that:

$$\forall s.\ \text{OnPath}(P_1, s) \supset Q_1(s).$$

For the base case where s is the starting situation of P_1, i.e. S_1, the thesis follows from (3.147), (3.150), and (3.151).

For the inductive step, fix S_2 and assume that:

$$\text{OnPath}(P_1, S_2) \wedge Q_1(S_2). \tag{3.152}$$

Take some arbitrary action A_1 such that $\text{OnPath}(P_1, do(A_1, S_2))$. Then by this, (3.152), and (3.148), it follows that:

$$Q_1(do(A_1, S_2)). \tag{3.153}$$

Thus by induction on s, we have:

$$\forall s.\ \text{OnPath}(P_1, s) \supset Q_1(s). \tag{3.154}$$

But this is contradictory to (3.149). □

3.5.4 Correctness of Axiomatization

Next, I prove the correctness of my axiomatization. Note that, a natural way of capturing the notion of infinite path is by specifying it as a mapping from the set of natural numbers to situations on a path. To this end, I use a function σ of the following sort (here $\mathcal{S}$ denotes the set of all situations):

$$\sigma : \mathbb{N} \rightarrow \mathcal{S}.$$

I say that such a function σ models a *path sequence* if σ maps the number 0 to an executable situation (representing the starting situation of the path), and for each number n, there is an action a that is executable in the situation s_n produced by $\sigma(n)$ such that σ maps the immediate successor of n (i.e. $n + 1$) to the situation $do(a, s_n)$.

Definition 3.5.50.

$$\text{PathSeq}(\sigma) \overset{\text{def}}{=} \text{Executable}(\sigma(0)) \wedge \forall n.\ \exists a.\ \text{Poss}(a, \sigma(n)) \wedge \sigma(n + 1) = do(a, \sigma(n)).$$

When I prove results involving path sequences, I will use an axiomatization of the natural numbers, i.e., standard second-order Peano arithmetic, for the natural number sort, denoted by $\Sigma_{\mathbb{N}}$.

I will use the following to prove the completeness theorem. Given Definition 3.5.50, it can be shown that if σ is a path sequence and $i < j$, then the situation given by $\sigma(i)$ precedes the one given by $\sigma(j)$:

Lemma 3.5.51.

$$\Sigma_{\mathbb{N}} \cup \Sigma \cup \mathcal{D}_{path} \models \forall \sigma, i, j.\ \text{PathSeq}(\sigma) \wedge i < j \supset \sigma(i) \prec \sigma(j).$$

Proof. (By induction on n, where $n = j - i$) Fix σ_1 and assume:

$$\text{PathSeq}(\sigma_1). \tag{3.155}$$

For the base case, fix I_1 and J_1 and assume that $J_1 - I_1 = 1$. Then it follows from (3.155) and Definition 3.5.50 that there is an action A_1 s.t.:

$$\sigma_1(J_1) = \sigma_1(I_1 + 1) = do(A_1, \sigma_1(I_1)).$$

From this and Lemma 3.5.19, it follows that $\sigma_1(I_1) \prec do(A_1, \sigma_1(I_1))$, i.e. $\sigma_1(I_1) \prec \sigma_1(I_1 + 1)$, and thus $\sigma_1(I_1) \prec \sigma_1(J_1)$.

For the inductive case, fix I_N, J_N, and N_1 and assume that:

$$J_N - I_N = N_1, \text{ and} \tag{3.156}$$

$$\text{PathSeq}(\sigma_1) \wedge I_N < J_N \supset \sigma_1(I_N) \prec \sigma_1(J_N). \tag{3.157}$$

We have to show that:

$$\text{PathSeq}(\sigma_1) \wedge I_N < J_{N+1} \supset \sigma_1(I_N) \prec \sigma_1(J_{N+1}),$$

where $J_{N+1} - I_N = N_1 + 1$, i.e. by (3.156), $J_{N+1} = J_N + 1$.

Now, from (3.155) and (3.156), it follows that PathSeq$(\sigma_1) \wedge I_N < J_N$. From this and the inductive hypothesis (i.e. (3.157)), we have:

$$\sigma_1(I_N) \prec \sigma_1(J_N). \tag{3.158}$$

Moreover, from (3.155) and Definition 3.5.50 it follows that there is an action A_N s.t.:

$$\sigma_1(J_N + 1) = do(A_N, \sigma_1(J_N)).$$

From this and Lemma 3.5.19, it follows that $\sigma_1(J_N) \prec do(A_N, \sigma_1(J_N))$, i.e. $\sigma_1(J_N) \prec \sigma_1(J_N + 1)$, and thus:

$$\sigma_1(J_N) \prec \sigma_1(J_{N+1}). \tag{3.159}$$

Finally, from (3.158), (3.159), and the transitivity of $\prec$, i.e. Lemma 3.5.20, we have: $\sigma_1(I_N) \prec \sigma_1(J_{N+1})$. □

To show that for every path sequence there is a corresponding path, it is useful to first introduce a corresponding ASF. Given path sequence σ, let F_σ be the ASF defined as follows:

Definition 3.5.52.

$$F_\sigma(s) = a_n, \qquad \text{if } \exists n.\ \sigma(n) = s \wedge \sigma(n+1) = do(a_n, s),$$

$$F_\sigma(s) = b, \qquad \text{otherwise,}$$

where b is some fixed but arbitrary action.

I can show that given a path sequence σ, any situation s that is on the path defined by the corresponding ASF F_σ and the initial situation of the path sequence $\sigma(0)$ is in fact on the path sequence σ at some position n:

Lemma 3.5.53.

$$\Sigma_{\mathbb{N}} \cup \Sigma \cup \mathcal{D}_{path} \models \forall s, \sigma.\ \text{PathSeq}(\sigma) \wedge \sigma(0) \prec s \wedge \text{OnPathASF}(F_\sigma, \sigma(0), s) \supset$$
$$\exists n.\ \sigma(n) = s.$$

Proof. (By induction on s) Fix σ_1. Construct a function from situations to actions F_{σ_1} such that F_{σ_1} is the corresponding ASF to σ_1. Also, assume that:

$$\text{PathSeq}(\sigma_1). \tag{3.160}$$

In the base case where s is an initial situation, $\neg\exists s'.\ s' \prec s$ by Definition 3.2.1 and Axiom 3.2.5, so the antecedent is false and the thesis trivially holds.

For the inductive step, fix S_N and assume that:

$$\sigma_1(0) \prec S_N \wedge \text{OnPathASF}(F_{\sigma_1}, \sigma_1(0), S_N) \supset \exists n.\ \sigma_1(n) = S_N. \tag{3.161}$$

Also, fix action A_N and assume that:

$$\sigma_1(0) \prec do(A_N, S_N), \text{ and} \tag{3.162}$$

$$\text{OnPathASF}(F_{\sigma_1}, \sigma_1(0), do(A_N, S_N)). \tag{3.163}$$

From (3.163) and Definition 3.5.3, it follows that:

$$\forall a, s.\ \sigma_1(0) \prec do(a, s) \preceq do(A_N, S_N) \supset F_{\sigma_1}(s) = a. \tag{3.164}$$

From (3.162) and Axiom 3.2.5, it follows that:

$$\sigma_1(0) \preceq S_N. \tag{3.165}$$

From Lemma 3.5.25, we have $S_N \preceq do(A_N, S_N)$. From this and (3.164), we have:

$$\forall a, s.\ \sigma_1(0) \prec do(a, s) \preceq S_N \supset F_{\sigma_1}(s) = a. \tag{3.166}$$

From (3.165), (3.166), and Definition 3.5.3, we have:

$$\text{OnPathASF}(F_{\sigma_1}, \sigma_1(0), S_N). \tag{3.167}$$

Now, (3.165) and Definition 3.2.6 give us two cases. In the case where $\sigma_1(0) = S_N$, it trivially follows that $\exists n.\ \sigma_1(n) = S_N$. In the case where $\sigma_1(0) \prec S_N$, from this, (3.167), and the induction hypothesis, i.e. (3.161), it follows that $\exists n.\ \sigma_1(n) = S_N$. Thus, in both these cases, there is a N_1 such that:

$$\sigma_1(N_1) = S_N. \tag{3.168}$$

From (3.160), (3.168), and Definition 3.5.50, it follows that there is an action, let us call it A_N^*, s.t.:

$$\sigma_1(N_1 + 1) = do(A_N^*, \sigma_1(N_1)). \tag{3.169}$$

We just need to show that $A_N^* = A_N$. From the definition of F_{σ_1}, it follows that:

$$\forall a.\ \sigma_1(N_1 + 1) = do(a, \sigma_1(N_1)) \supset F_{\sigma_1}(\sigma_1(N_1)) = a. \tag{3.170}$$

From (3.168), (3.169), and (3.170), it follows that:

$$F_{\sigma_1}(S_N) = A_N^*. \tag{3.171}$$

From (3.164), we have $F_{\sigma_1}(S_N) = A_N$. Finally from this and (3.171), we have $A_N = A_N^*$, and thus from this, (3.169), and (3.168), it follows that $\sigma_1(N_1+1) = do(A_N, S_N)$, i.e. $\exists n.\ \sigma_1(n) = do(A_N, S_N)$. □

I say that a path p matches a path sequence σ if σ is indeed a path sequence, $\sigma(0)$ is the starting situation of p, and for all n, s and a, if $\sigma(n)$ is a situation s on path p, then $\sigma(n+1)$ is the successor situation $do(a, s)$ of s on p:

Definition 3.5.54.

$$\begin{aligned}\text{Matches}(p,\sigma) \overset{\text{def}}{=} \ & \text{PathSeq}(\sigma) \wedge (\sigma(0) = s \equiv \text{Starts}(p,s)) \\ & \wedge \forall n, s.\ [\sigma(n) = s \wedge \text{OnPath}(p,s) \supset \\ & \qquad \forall a.\ (\sigma(n+1) = do(a,s) \equiv \text{OnPath}(p, do(a,s)))].\end{aligned}$$

Given this formalization, the task of proving correctness of my axiomatization for infinite paths can be reduced to showing that path sequences are isomorphic to paths defined by $\Sigma \cup \mathcal{D}_{path}$, i.e. that there is an one-to-one mapping between these two. To this end, I first show that for any path p, there is a path sequence σ that matches p.

Theorem 3.5.55 (Soundness)**.**

$$\Sigma_{\mathbb{N}} \cup \Sigma \cup \mathcal{D}_{path} \models \forall p.\ (\exists \sigma.\ \text{PathSeq}(\sigma) \wedge \text{Matches}(p, \sigma)).$$

Proof. Fix path P_1. By Propositions 3.5.37(a), and 3.5.37(c), there is an executable situation S_1 such that P_1 starts with S_1:

$$\text{Starts}(P_1, S_1), \text{ and} \tag{3.172}$$

$$\text{Executable}(S_1). \tag{3.173}$$

By Axiom 3.5.2(i), there is an action selection function F_1 and situation S_2 such that:

$$\text{Executable}(F_1, S_2), \text{ and}$$

$$\forall s'.\ \text{OnPathASF}(F_1, S_2, s') \equiv \text{OnPath}(P_1, s').$$

From this and Definition 3.5.3, it follows that S_2 is the earliest situation of path P_1. Moreover, from Definition 3.5.1 and (3.172), it follows that S_1 is the earliest situation that is also on path P_1. Thus it follows that $S_1 = S_2$ and hence we have:

$$\text{Executable}(F_1, S_1), \text{ and} \tag{3.174}$$

$$\forall s'.\ \text{OnPathASF}(F_1, S_1, s') \equiv \text{OnPath}(P_1, s'). \tag{3.175}$$

Let σ_1 be defined as follows:

$$\sigma_1(0) = S_1, \tag{3.176}$$

$$\sigma_1(n+1) = do(F_1(\sigma_1(n)), \sigma_1(n)), \quad \text{for } n \geq 0. \tag{3.177}$$

We have to prove that PathSeq$(\sigma_1) \wedge$ Matches(P_1, σ_1).

First, let me show that PathSeq(σ_1). By Definition 3.5.50, to show this we have to prove that:

$$\text{(a). Executable}(\sigma_1(0)),\ \text{and}$$

$$\text{(b). } \forall n.\ \exists a.\ \text{Poss}(a, \sigma_1(n)) \wedge \sigma_1(n+1) = do(a, \sigma_1(n)).$$

(a) follows from (3.173) and (3.176). By (3.177), for each n there is indeed an action $a = F_1(\sigma_1(n))$ s.t. $\sigma_1(n+1) = do(a, \sigma_1(n))$. Thus to show (b), we have to prove that $\forall n$. Poss$(F_1(\sigma_1(n)), \sigma_1(n))$. Now, from (3.174) and Definition 3.5.4, it follows that:

$$\forall s'.\ \text{OnPathASF}(F_1, S_1, s') \supset \text{Poss}(F_1(s'), s').$$

Thus, to prove that $\forall n$. Poss$(F_1(\sigma_1(n)), \sigma_1(n))$, we just need to show that:

$$\forall n.\ \text{OnPathASF}(F_1, S_1, \sigma_1(n)).$$

I will show this by induction on n. For the base case, i.e. when $n = 0$, it follows from (3.176) that $\sigma_1(n) = S_1$. Thus we have to show that OnPathASF(F_1, S_1, S_1). This follows trivially from Definitions 3.2.6, 3.5.3, and Lemmata 3.5.20 and 3.5.21 (which imply that there are no situations $do(a, s^*)$ such that $S_1 \prec do(a, s^*) \preceq S_1$). For the inductive case, fix N_1 and assume that:

$$\text{OnPathASF}(F_1, S_1, \sigma_1(N_1)). \tag{3.178}$$

We have to show that OnPathASF$(F_1, S_1, \sigma_1(N_1+1))$. From (3.178) and Definition 3.5.3, we have:

$$S_1 \preceq \sigma_1(N_1), \text{ and} \tag{3.179}$$

$$\forall a, s^*.\ S_1 \prec do(a, s^*) \preceq \sigma_1(N_1) \supset F_1(s^*) = a. \tag{3.180}$$

From (3.180) and Definition 3.5.3, it follows that OnPathASF$(F_1, S_1, \sigma_1(N_1+1))$ holds if the following hold:

(b1). $S_1 \preceq \sigma_1(N_1+1)$, and

(b2). $\forall a, s^*.\ \sigma_1(N_1) \prec do(a, s^*) \preceq \sigma_1(N_1+1) \supset F_1(s^*) = a.$

Now, from (3.177), we have:

$$\sigma_1(N_1+1) = do(F_1(\sigma_1(N_1)), \sigma_1(N_1)). \tag{3.181}$$

From this and Lemma 3.5.25, we have:

$$\sigma_1(N_1) \preceq \sigma_1(N_1+1).$$

(b1) follows from this, (3.179), and the transitivity of $\preceq$ (i.e. Lemma 3.5.28). Moreover, (b2) follows from (3.181) and Lemmata 3.5.20 and 3.5.21 (which imply that there are no situations between $\sigma_1(N_1)$ and $\sigma_1(N_1+1)$). Thus, we have PathSeq(σ_1).

Next, let me show that Matches(P_1, σ_1). We already proved that σ_1 is a path sequence. Thus by Definition 3.5.54, we need to show that:

(c). $\sigma_1(0) = s \equiv \text{Starts}(P_1, s)$ and

(d). $\forall n, s.\ [\sigma_1(n) = s \wedge \text{OnPath}(P_1, s) \supset$

$\qquad \forall a.\ (\sigma_1(n+1) = do(a, s) \equiv \text{OnPath}(P_1, do(a, s)))]$.

(c) follows from (3.172), (3.176) and the uniqueness of starting situations of paths, i.e. Proposition 3.5.37(b). For (d), fix N_1 and $\widehat{S_1}$ and assume that:

$$\sigma_1(N_1) = \widehat{S_1}, \text{ and} \tag{3.182}$$

$$\text{OnPath}(P_1, \widehat{S_1}). \tag{3.183}$$

For the $(\supset)$ direction, fix A_1 and assume that:

$$\sigma_1(N_1 + 1) = do(A_1, \widehat{S_1}).$$

Then by this and (3.177), we have:

$$do(A_1, \widehat{S_1}) = do(F_1(\sigma_1(N_1)), \sigma_1(N_1)).$$

From this and (3.182), we have:

$$do(A_1, \widehat{S_1}) = do(F_1(\widehat{S_1}), \widehat{S_1}). \tag{3.184}$$

From (3.175) and (3.183), we have:

$$\text{OnPathASF}(F_1, S_1, \widehat{S_1}).$$

From this and Definition 3.5.3, it follows that:

$$S_1 \preceq \widehat{S_1}, \text{ and} \tag{3.185}$$

$$\forall a, s^*.\ S_1 \prec do(a, s^*) \preceq \widehat{S_1} \supset F_1(s^*) = a. \tag{3.186}$$

Now, consider the situation $do(F_1(\widehat{S_1}), \widehat{S_1})$. From Lemma 3.5.25, we have:

$$\widehat{S_1} \preceq do(F_1(\widehat{S_1}), \widehat{S_1}). \tag{3.187}$$

From this, (3.185), and the transitivity of $\preceq$ (i.e. Lemma 3.5.28), it follows that:

$$S_1 \preceq do(F_1(\widehat{S_1}), \widehat{S_1}). \tag{3.188}$$

Moreover, from (3.186), (3.187), and Lemmata 3.5.20 and 3.5.21, it follows that:

$$\forall a, s^*.\ S_1 \prec do(a, s^*) \preceq do(F_1(\widehat{S_1}), \widehat{S_1}) \supset F_1(s^*) = a. \tag{3.189}$$

From (3.188), (3.189), and Definition 3.5.3, it follows that:

$$\text{OnPathASF}(F_1, S_1, do(F_1(\widehat{S_1}), \widehat{S_1})).$$

From this and (3.175), it follows that OnPath$(P_1, do(F_1(\widehat{S_1}), \widehat{S_1}))$, i.e. by (3.184) that OnPath$(P_1, do(A_1, \widehat{S_1}))$.

For the $(\subset)$ direction, fix A_1 and assume that:

$$\text{OnPath}(P_1, do(A_1, \widehat{S_1})).$$

Then from this and (3.175), it follows that:

$$\text{OnPathASF}(F_1, S_1, do(A_1, \widehat{S_1})).$$

From this and Definition 3.5.3, it follows that:

$$A_1 = F_1(\widehat{S_1}). \tag{3.190}$$

Now, since by (3.182), $\sigma_1(N_1) = \widehat{S_1}$, it follows by (3.177) that:

$$\sigma_1(N_1 + 1) = do(F_1(\widehat{S_1}), \widehat{S_1}).$$

From this and (3.190), we have:

$$\sigma(N_1 + 1) = do(A_1, \widehat{S_1}).$$

Thus P_1 matches σ_1. $\square$

Conversely, for any path sequence σ, there is a path p that matches σ.

Theorem 3.5.56 (Completeness)**.**

$$\Sigma_{\mathbb{N}} \cup \Sigma \cup \mathcal{D}_{path} \models \forall\sigma.\ \text{PathSeq}(\sigma) \supset \exists p.\ \text{Matches}(p, \sigma).$$

Proof. Fix function σ_1 and assume that:

$$\text{PathSeq}(\sigma_1). \tag{3.191}$$

From this and Definition 3.5.50, it follows that:

$$\text{Executable}(\sigma_1(0)), \text{ and} \tag{3.192}$$

$$\forall n.\ \exists a.\ \text{Poss}(a, \sigma_1(n)) \wedge \sigma_1(n+1) = do(a, \sigma_1(n)). \tag{3.193}$$

Construct a tuple $(\sigma_1(0), F_{\sigma_1})$ such that F_{σ_1}, which is a function from situations to actions, is the corresponding ASF to σ_1. I will now show that Executable$(F_{\sigma_1}, \sigma_1(0))$. Assume otherwise. Then from Definition 3.5.4 and (3.192), it follows that there is a situation S_N such that:

$$\text{OnPathASF}(F_{\sigma_1}, \sigma_1(0), S_N), \text{ and} \tag{3.194}$$

$$\neg\text{Poss}(F_{\sigma_1}(S_N), S_N). \tag{3.195}$$

From (3.194) and Definition 3.5.3, it follows that $\sigma_1(0) \preceq S_N$. This and Definition 3.2.6 give us two cases. In the case where $\sigma_1(0) = S_N$, it trivially follows that $\exists n.\ \sigma_1(n) = S_N$. In the case where $\sigma_1(0) \prec S_N$, from (3.191), the assumption for this case that $\sigma_1(0)$ strictly precedes S_N, (3.194), and Lemma 3.5.53, it follows that $\exists n.\ \sigma_1(n) = S_N$. Thus, for both these cases, we have that there is a n, say N_1, s.t.:

$$\sigma_1(N_1) = S_N. \tag{3.196}$$

Then from this and (3.193), it follows that there is an action A_N s.t.:

$$\sigma_1(N_1+1) = do(A_N, S_N), \text{ and} \tag{3.197}$$

$$\text{Poss}(A_N, S_N). \tag{3.198}$$

From (3.196), (3.197), and the definition of F_{σ_1}, it follows that:

$$F_{\sigma_1}(S_N) = A_N.$$

Finally, from this and (3.198), we have $\text{Poss}(F_{\sigma_1}(S_N), S_N)$; but this is contradictory to (3.195). Thus, we have:

$$\text{Executable}(F_{\sigma_1}, \sigma_1(0)). \tag{3.199}$$

From this and Axiom 3.5.2(ii), it follows that there is a path P_1 such that:

$$\text{Starts}(P_1, \sigma_1(0)), \text{ and} \tag{3.200}$$

$$\forall s.\ \text{OnPathASF}(F_{\sigma_1}, \sigma_1(0), s) \equiv \text{OnPath}(P_1, s). \tag{3.201}$$

Now, we need to show that $\text{Matches}(P_1, \sigma_1)$. By Definition 3.5.54, this amounts to showing that:

(a). $\text{PathSeq}(\sigma_1), \text{ and}$

(b). $\sigma_1(0) = s \equiv \text{Starts}(P_1, s)$ and

(c). $\forall n, s.\ [\sigma_1(n) = s \wedge \text{OnPath}(P_1, s) \supset$

$\forall a.\ (\sigma_1(n+1) = do(a, s) \equiv \text{OnPath}(P_1, do(a, s)))].$

(a) follows from the antecedent, i.e. (3.191). (b) follows from (3.200) and the uniqueness of starting situations of paths, i.e. Proposition 3.5.37(b). For (c), fix $\widehat{N_1}$ and $\widehat{S_1}$ and assume that:

$$\sigma_1(\widehat{N_1}) = \widehat{S_1}, \text{ and} \tag{3.202}$$

$$\text{OnPath}(P_1, \widehat{S_1}). \tag{3.203}$$

For the $(\supset)$ direction, fix $\widehat{A_1}$ and assume that:

$$\sigma_1(\widehat{N_1}+1) = do(\widehat{A_1}, \widehat{S_1}). \tag{3.204}$$

From (3.203) and (3.201), it follows that:

$$\text{OnPathASF}(F_{\sigma_1}, \sigma_1(0), \widehat{S_1}).$$

From this and Definition 3.5.3, we have:

$$\forall a, s.\ \sigma_1(0) \prec do(a,s) \preceq \widehat{S_1} \supset F_{\sigma_1}(s) = a. \tag{3.205}$$

From (3.202), (3.204), and the definition of F_{σ_1}, we have:

$$F_{\sigma_1}(\widehat{S_1}) = \widehat{A_1}. \tag{3.206}$$

Now, suppose $\neg\text{OnPath}(P_1, do(\widehat{A_1}, \widehat{S_1}))$. Then by (3.201), we have:

$$\neg\text{OnPathASF}(F_{\sigma_1}, \sigma_1(0), do(\widehat{A_1}, \widehat{S_1})). \tag{3.207}$$

From Lemma 3.5.51, (3.191), and the fact that $0 < \widehat{N_1} + 1$, we have $\sigma_1(0) \prec \sigma_1(\widehat{N_1} + 1)$. From this and (3.204), we have:

$$\sigma_1(0) \prec do(\widehat{A_1}, \widehat{S_1}).$$

From this, (3.207), and Definition 3.5.3, we have:

$$\exists a, s.\ \sigma_1(0) \prec do(a, s) \preceq do(\widehat{A_1}, \widehat{S_1}) \wedge \neg F_{\sigma_1}(s) = a.$$

From this and (3.205), it follows that $\neg(F_{\sigma_1}(\widehat{S_1}) = \widehat{A_1})$; but this is contradictory to (3.206).

For the $(\subset)$ direction, fix $\widehat{A_2}$ and assume that:

$$\text{OnPath}(P_1, do(\widehat{A_2}, \widehat{S_1})). \qquad (3.208)$$

From (3.191) and Definition 3.5.50, it follows that there is an action, say $\widehat{A_3}$, s.t.:

$$\sigma_1(\widehat{N_1} + 1) = do(\widehat{A_3}, \sigma_1(\widehat{N_1})). \qquad (3.209)$$

I will show that $\widehat{A_2} = \widehat{A_3}$. From (3.209) and (3.202), it follows that:

$$\sigma_1(\widehat{N_1} + 1) = do(\widehat{A_3}, \widehat{S_1}). \qquad (3.210)$$

From (3.208) and (3.201), we have:

$$\text{OnPathASF}(F_{\sigma_1}, \sigma_1(0), do(\widehat{A_2}, \widehat{S_1})).$$

From this and Definition 3.5.3, we have $F_{\sigma_1}(\widehat{S_1}) = \widehat{A_2}$. Finally from this, (3.202), (3.210), and the definition of F_{σ_1}, we have $\widehat{A_2} = \widehat{A_3}$. Thus from this and (3.210), it follows that $\sigma_1(\widehat{N_1}+1) = do(\widehat{A_2}, \widehat{S_1})$. □

Note that my soundness result implies that for any path p, there is a countably infinite number of distinct situations on p: as shown above, p corresponds to a path sequence, and situations along a path sequence are strict successors to each other; this along with Lemmata 3.5.19, 3.5.20, and 3.5.23 imply that these situations are distinct. An alternative way to show that there is a countably infinite number of situations on a path p is to show that (a) there is a situation on p, that (b) for every situation s on p, there is a successor situation $do(a, s)$ that is also on p, and that (c) these situations are distinct. (a) follows from Definition 3.5.1 and Proposition 3.5.37(a), while (b) follows from Proposition 3.5.38. Finally, (c) follows from Lemmata 3.5.19, 3.5.20, and 3.5.23.

3.5.5 Related Work

I have already discussed the work most closely related to my paths at the beginning of this section. Beyond this, there is some work that deals with the temporal aspects of situations, i.e. the starting time of situations and action durations [165, 177], but not temporally extended paths. Another set of approaches introduces some notion of paths while addressing some application of paths and shows how various temporal logic formulae can be interpreted over such paths. Gabaldon [85] was the first to introduce statements of temporal logic (LTL) into the situation calculus. He used these to express search control knowledge for forward-chaining planning. However, he only considers finite paths defined by pairs of situations. Fritz and McIlraith [84] show how an extended version of LTL interpreted over a finite horizon can be compiled into ConGolog [51].

Claßsen and Lakemeyer [32] developed a second-order modal logic inspired by CTL* and dynamic logic to express properties about (possibly) non-terminating ConGolog programs. The authors define infinite "traces" using program configurations. A configuration is a pair (δ, z), where δ is a ConGolog program that remains to be executed and z is a sequence of actions that have been already performed. Given z and world w, they define infinite execution traces of δ as infinite sequences of configurations, s.t. the ending configuration of any finite prefix of the sequence can be reached from the initial configuration (δ, z) and w. Note that, a key difference between this work and my formalization is that while I define paths axiomatically in the situation calculus, they define a modal logic on top of the situation calculus that allows temporal properties over execution of programs to be expressed and the semantics of programs is part of the model theoretic semantics of the logic. For the CTL-like fragment of the language, the authors also propose a verification method based on fixpoint approximation and "characteristic graphs", which can finitely represent a ConGolog program's configuration graph; the method is sound but incomplete.

[56] uses a first-order version of the μ-calculus [65] to specify properties of non-terminating Golog [140] programs. The μ-calculus is a very expressive temporal logic that provides least and greatest fixpoints. Interestingly, the semantics of the μ-calculus operators can be defined without referring explicitly to infinite paths. In the propositional case, it is well known that the μ-calculus subsumes LTL as well as CTL*[111, 5], although translating a CTL* formula often results in a much less readable μ-calculus formula. However, in the first-order case (when quantification over objects across situations is allowed), the μ-calculus *does not subsume* LTL (and thus CTL*), a consequence of results shown in [28]. Thus, my situation calculus with infinite paths, which can be used to define first-order LTL and CTL* over situation calculus theories, can express properties that cannot be expressed in first-order μ-calculus over such theories.

3.6 The ConGolog Agent Programming Language

I now outline the logic programming language ConGolog [51], the concurrent version of Golog [140], which will be used to define the semantics of the Simple Rational Agent Programming Language (SR-APL) in Chapter 7. Within SR-APL, I also specify the agents' plans using the notation of ConGolog.

The ConGolog programming language provides an alternative to AI planning by looking instead at the problem of *high-level program execution*. Instead of looking for a legal sequence of actions to achieve some goal, the ConGolog interpreter searches for a legal sequence of actions that amount to a legal execution of some high-level non-deterministic program, one that specifies the agent's behavior. The more abstract the program is, the more it resembles traditional planning. But a ConGolog program can encode search control knowledge. The formalism differs from other concurrent procedural languages in that the initial state can be incompletely specified. As well, primitive actions can be user defined, to be specific, by axioms in the situation calculus, thus allowing these actions to affect the environment in complex ways. Finally, it also incorporates a rich notion of concurrency, contributing to a level of procedural complexity that hasn't been addressed before (e.g. those arising from the interaction between prioritized concurrency and recursive procedures).

A typical ConGolog program is composed of a sequence of procedure declarations, followed by a complex action. Complex actions can be composed using constructs given in Table 3.1. Here a denotes a situation calculus primitive action, ϕ denotes a situation calculus formula with the situation argument of its fluents suppressed, δ, δ_1, and δ_2 stand for complex actions, $\vec{x}$ is a set of variables, β is a procedure name, and $\vec{p}$ denotes the actual parameters to the procedure. Most of these constructs are self-

a,	primitive action
$\phi?$,	wait for a condition
$(\delta_1; \delta_2)$,	sequence
$(\delta_1 \mid \delta_2)$,	nondeterministic choice between actions
$\pi x.\ \delta$,	nondeterministic choice of arguments
δ^*,	nondeterministic iteration
if ϕ **then** δ_1 **else** δ_2,	conditional
while ϕ **do** δ,	while loop
$(\delta_1 \| \delta_2)$,	concurrency with equal priority
$(\delta_1 \rangle\rangle \delta_2)$,	concurrency with different priorities
$\langle \vec{x} : \phi \rightarrow \delta \rangle$,	interrupt
$\delta^{\|}$,	concurrent iteration
$\beta(\vec{p})$,	procedure call

Table 3.1: Some ConGolog Constructs

explanatory. Intuitively, $\pi x.\delta$ nondeterministically picks a binding for the variable x and performs the program δ for this binding of x. The interrupt construct works as follows: whenever ϕ becomes true for some binding of $\vec{x}$, δ is executed with this binding; after this, the interrupt can fire again. The syntax of procedures is as follows:

proc $\beta(\vec{y})\delta$, where β is the procedure name, $\vec{y}$ denotes the formal parameters to the procedure, and δ is a complex action.

For example, consider the simple program to clear a table in a blocks world:

$$
\begin{aligned}
&\{\textbf{proc}\ removeABlock\\
&\qquad \pi b.\ [\text{OnTable}(b, now)?; pickUp(b); putAway(b)]\\
&\ \textbf{end};\\
&\ removeABlock^{*}; \neg\exists b.\ \text{OnTable}(b, now)?\}
\end{aligned}
$$

Here, first a procedure is defined to remove a block from the table using the non-deterministic choice of argument operator π. The wait action OnTable(b, now)? succeeds only if the chosen argument b is a block that is on the table in the current situation. The main part of the program uses the non-deterministic iteration operator, and says to execute the $removeABlock$ program zero or more times until the table is clear.

The semantics of ConGolog programs are defined using *structural operational semantics* [168], which is based on *transitions*. A transition is a “single step” of computation, i.e. a primitive action. To this end, two special predicates are introduced, *Final* and *Trans*, where Final(δ, s) means that program δ may legally terminate in situation s, and where Trans(δ, s, δ', s') means that program δ in situation s may legally execute one step, ending in situation s' with program δ' remaining. Trans and Final are characterized by giving equivalence axioms for each of the above constructs of ConGolog.[20]

In the following, I give axioms only for the program constructs that I will be using in SR-APL; see [51] for the complete axiomatization of Trans and Final. Also, for test actions $\phi?$, I use the alternate semantics provided in [54] as it simplifies the operational semantics of SR-APL. The axioms Γ_{F} for Final are as follows (here, the construct nil denotes the 'empty' program that terminates immediately):

Axiom 3.6.1.

$$\Gamma_{\mathrm{F}_1}.\ \mathrm{Final}(nil, s) \equiv \mathrm{True},$$

$$\Gamma_{\mathrm{F}_2}.\ \mathrm{Final}(a, s) \equiv \mathrm{False},$$

$$\Gamma_{\mathrm{F}_3}.\ \mathrm{Final}(\phi?, s) \equiv \phi(s),$$

$$\Gamma_{\mathrm{F}_4}.\ \mathrm{Final}([\delta_1; \delta_2], s) \equiv \mathrm{Final}(\delta_1, s) \wedge \mathrm{Final}(\delta_2, s),$$

$$\Gamma_{\mathrm{F}_5}.\ \mathrm{Final}(\pi x.\ \delta, s) \equiv \exists x.\ \mathrm{Final}(\delta, s),$$

$$\Gamma_{\mathrm{F}_6}.\ \mathrm{Final}(\delta^*, s) \equiv \mathrm{True},$$

$$\Gamma_{\mathrm{F}_7}.\ \mathrm{Final}([\delta_1 \parallel \delta_2], s) \equiv \mathrm{Final}(\delta_1, s) \wedge \mathrm{Final}(\delta_2, s).$$

Thus, these axioms define whether the program can be considered to be already in a legally terminated state in the given situation. For example, axiom Γ_{F_4} says that the

[20]However, note that De Giacomo et al. defined interrupts in terms of other constructs; see [51] for details.

program that involves a sequential composition $[\delta_1; \delta_2]$ can be considered completed in situation s if both δ_1 and δ_2 are final/completed in s.

The axioms Γ_{T} characterizing Trans are as follows:

Axiom 3.6.2.

$$\Gamma_{\mathrm{T}_1}.\ \mathrm{Trans}(nil, s, \delta', s') \equiv \mathrm{False},$$

$$\Gamma_{\mathrm{T}_2}.\ \mathrm{Trans}(a, s, \delta', s') \equiv \mathrm{Poss}(a, s) \wedge \delta' = nil \wedge s' = do(a, s),$$

$$\Gamma_{\mathrm{T}_3}.\ \mathrm{Trans}(\phi?, s, \delta', s') \equiv \mathrm{False},$$

$$\begin{aligned}\Gamma_{\mathrm{T}_4}.\ \mathrm{Trans}([\delta_1; \delta_2], s, \delta', s') \equiv \exists \delta_1'.\ (\delta' = [\delta_1'; \delta_2] \wedge \mathrm{Trans}(\delta_1, s, \delta_1', s'))\\ \vee \mathrm{Final}(\delta_1, s) \wedge \mathrm{Trans}(\delta_2, s, \delta', s'),\end{aligned}$$

$$\Gamma_{\mathrm{T}_5}.\ \mathrm{Trans}(\pi x.\delta, s, \delta', s') \equiv \exists x.\ \mathrm{Trans}(\delta, s, \delta', s'),$$

$$\Gamma_{\mathrm{T}_6}.\ \mathrm{Trans}(\delta^*, s, \delta', s') \equiv \exists \delta_1'.\ \delta' = (\delta_1'; \delta^*) \wedge \mathrm{Trans}(\delta, s, \delta_1', s'),$$

$$\begin{aligned}\Gamma_{\mathrm{T}_7}.\ \mathrm{Trans}([\delta_1 \parallel \delta_2], s, \delta', s') \equiv \exists \delta_1'.\ (\delta' = [\delta_1' \parallel \delta_2] \wedge \mathrm{Trans}(\delta_1, s, \delta_1', s')),\\ \vee \exists \delta_2'.\ (\delta' = [\delta_1 \parallel \delta_2'] \wedge \mathrm{Trans}(\delta_2, s, \delta_2', s')).\end{aligned}$$

These axioms thus specify when a configuration (δ, s) with a program δ remaining in situation s can evolve in a single step to a configuration (δ', s'). For example, axiom Γ_{T_2} says that a program involving a primitive action a in a situation s can make a transition to $(nil, do(a, s))$, provided that a is possible in s. After having performed a, nothing remains to be performed. Γ_{T_4} states that a program that is composed of a sequence $[\delta_1; \delta_2]$ in s can evolve to the configuration $([\delta_1'; \delta_2], s')$, provided that the program δ_1 in s can evolve to the program δ_1' in s'. Moreover, this sequential composition can also evolve to the configuration (δ', s'), provided that (δ_1, s) is a final configuration, and the program δ_2 in s can evolve to the program δ' in s'.

The overall semantics of a ConGolog program is specified by the Do predicate, which is defined as follows:

Definition 3.6.3.

$$\text{Do}(\delta, s, s') \overset{\text{def}}{=} \exists \delta'. \, (\text{Trans}^*(\delta, s, \delta', s') \land \text{Final}(\delta', s')),$$

where Trans* is the reflexive transitive closure of the transition relation Trans, which can be defined using the following second-order situation calculus formula:

Definition 3.6.4.

$$\begin{aligned}
&\text{Trans}^*(\delta, s, \delta', s') \overset{\text{def}}{=} \\
&\quad \forall T. \, [\forall \delta_1, s_1. \, T(\delta_1, s_1, \delta_1, s_1) \land \\
&\qquad \forall \delta_1, s_1, \delta_2, s_2, \delta_3, s_3. \, (\text{Trans}(\delta_1, s_1, \delta_2, s_2) \land T(\delta_2, s_2, \delta_3, s_3) \supset T(\delta_1, s_1, \delta_3, s_3))] \\
&\quad \supset T(\delta, s, \delta', s').
\end{aligned}$$

Thus Do(δ, s, s') holds if and only if s' can be reached by performing a sequence of transitions starting with the program δ in s, and the remaining program δ' may legally terminate in s'.

In [51], De Giacomo et al. showed that the axioms for Trans and Final are definitional in the sense that the whole of ConGolog completely characterize Trans and Final for programs without procedures. Thus, these predicates can be eliminated. To give the semantics of ConGolog programs with procedures, De Giacomo et al. relied on second-order definitions of Trans and Final. The reason for this is that since a recursive procedure may do an arbitrary number of procedure calls before it actually performs a primitive action, and since calling procedures does not involve transitions, it is impossible to give first-order equivalence axioms for them. However, they showed that under certain conditions, namely for guarded configurations,[21] the second-order definitions for Trans and Final that is required to handle recursive procedure calls, can be replaced by first-order axioms. See [51] for the details of these.

3.7 Conclusion

In this chapter, I introduced the situation calculus and action theories. I discussed previous work on the formalization of knowledge and its dynamics. I then laid the foundations for the semantics of prioritized goals by introducing infinite paths in the situation calculus, giving an axiomatization of infinite paths, and proving some properties of

[21]A configuration (δ, s) is guarded if and only if for some n, δ makes at most n recursive procedure calls before trying to make an actual program step, i.e. an atomic action. See [51] for a formal definition.

this axiomatization. I will use some of these properties in the proofs of theorems in future chapters. My formalization of infinite paths is relevant for any account of temporal properties or motivational attitudes (which require the former) in the situation calculus. My account of infinite paths is more general than Shapiro's account of finite paths [194]. Moreover, it allows quantification over paths and thus is simpler to use than the one proposed by Lespérance et al. [134]. I also discussed the ConGolog agent programming language. The semantics of my agent programming language SR-APL is based on that of ConGolog.

Chapter 4

A Formalization of Prioritized Goals for Optimizing Agents

4.1 Introduction

Not all of the agent's goals are equally important to her. For example, ensuring that the spacecraft does not explode should in principle be much more important than any other goal that a space agent may have – she must not sacrifice this goal to achieve any number of lower priority goals.[22] Thus, it is useful to support a priority ordering over goals. This information can be used to decide which of the agent's intentions should no longer be actively pursued in case they become mutually inconsistent.

An agent's goals must properly evolve when an action/event occurs, when the

[22]This example is borrowed from [187].

agent's beliefs/knowledge changes, or when a goal is adopted or dropped. Such changes in the agent's goals must be consistent with her knowledge about the world she lives in. For example, the agent should drop an existing goal if she learns that it has been brought about or that it has become impossible to achieve. As discussed in Chapter 2, most work on formalizing goals (e.g., [35, 181, 124, 213, 198, 247, 187, 235]) only deals with static goal semantics and not their dynamics. Those that handle goal dynamics mostly provide a syntactic formalization of goal change, which usually amounts to adding or deleting primitive facts from a goal-base (e.g., [234, 44, 43]). Such formalizations can only capture limited types of temporally extended goals (in particular, most deal with achievement goals exclusively).[23] Moreover, their properties are often not well understood. Finally, these frameworks do not maintain the consistency of intended goals.

In this chapter, I formalize goals with different priorities using a new indexed set of accessibility relations G. I call these goals *prioritized goals* or p-goals. Prioritized goals in my framework are analogous to desires and need not be actively pursued by the agent. As such, they are not required to be consistent with the agent's knowledge or with each-other. In terms of these, I define the consistent set of *chosen goals* or intentions (c-goals, henceforth) that the agent is committed to realize. I then formalize

[23]See Section 4.6 for a discussion on recent work that attempts to handle one or more of these limitations.

p-goal dynamics by giving a successor-state axiom for G that is affected by, among other things, two special actions, $adopt$ and $drop$. A regular (non-adopt/drop) action causes an agent to temporally progress all her goals to reflect the fact that this action has happened. As we will see later, this may render some of her goals impossible to bring about and may make other goals inconsistent with others. On the other hand, an $adopt$ action causes an agent to adopt a new prioritized goal at some specified priority level, while a $drop$ action causes an agent to drop an existing goal from all levels of her goal hierarchy. Since an agent's c-goals are specified in terms of her p-goals, they are automatically updated when her p-goals are revised. I show that agents specified in this framework always try to optimize their chosen goals – they will drop an intended c-goal ϕ whenever an opportunity to commit to a higher priority (but inconsistent with ϕ) goal arise. I then consider some basic properties of my axiomatization: consistency and realism [35]. I also discuss some properties w.r.t. goal change. In particular, I show that adopting a new goal and dropping an existing goal has the desired effects. Also, I identify the restrictions on G that give us positive and negative introspection of goals, and show that if these restrictions are asserted of the initial situations, they persist after any sequence of actions is performed, as they are preserved by the successor-state axiom for G. I then identify the conditions under which an agent's achievement p-goals and achievement c-goals persist. Finally, I discuss an example to illustrate the proposed formalization.

4.2 Prioritized Goals

I specify each p-goal of an agent by its own accessibility relation/fluent G. For a given priority level n, the G relation is specified as a relation on an infinite path p representing a possible evolution of the world and a situation s which stands for the current world. Intuitively, a path p is G-accessible at priority level n in situation s, denoted by $G(p, n, s)$, if the goal of the agent at priority level n in situation s is satisfied over path p and if p starts with a situation that has the same action history as s. The latter requirement ensures that the agent's p-goal accessible paths reflect the actions that have been performed so far.[24] I use a reverse priority ordering on goals – a smaller n represents higher priority, and the highest priority level is 0. Also, I assume that the set of p-goals are totally ordered according to priority. Thus given a priority level, the agent can have only one goal at that level, possibly a complex one, e.g., a conjunctive goal. While some authors contend that this is too strong an assumption for a realistic agent, it could be argued that a strict order is necessary for any resource-bounded agent. I will come back to this issue in Section 4.6.

[24]Since the agent's goals are future oriented, they should be evaluated w.r.t. paths that are consistent with the actions that have been performed in the actual world. The requirement that a G-accessible path starts with a situation that has the same history as the actual current situation enforces this.

I say that an agent has the p-goal that ϕ at level n in situation s if ϕ holds over all paths that are G-accessible at n in s.

Definition 4.2.1.

$$\text{PGoal}(\phi, n, s) \overset{\text{def}}{=} \forall p.\ G(p, n, s) \supset \phi(p).$$

Note that for a given priority level, the PGoal construct can be used to talk about parts (or logical consequences) of the goal at that level, and thus an agent might have many PGoals at some level.

I also define the OPGoal(ϕ, n, s) predicate which holds when ϕ corresponds exactly to the agent's goal at priority level n in situation s, i.e. her *only p-goal* at level n in s.

Definition 4.2.2.

$$\text{OPGoal}(\phi, n, s) \overset{\text{def}}{=} \text{PGoal}(\phi, n, s) \wedge (\forall p.\ \phi(p) \supset G(p, n, s)).$$

An agent has the only p-goal that ϕ at level n in situation s if ϕ is a p-goal at n in s, and any path over which ϕ holds is G-accessible at n in s.

I allow the agent to have infinitely many goals. I expect the modeler to include some specification of what paths are G accessible at the various levels initially. I call these axioms *initial goal axioms*. In many cases, the user will want to specify a finite set of initial p-goals. This can be done by providing a set of axioms as in the example below. But in general, an agent can have a countably infinite set of p-goals, e.g., an agent that has the p-goal at level n to know what the n-th prime number is for all n. The agent's set of p-goals can even be incompletely specified, e.g., the theory might not specify what the p-goals at some level are initially.

The following example illustrates how we can specify the initial p-goals of an agent. We have an agent who initially has the following three p-goals: $\phi_0 = \Box$BeRich, $\phi_1 = \Diamond$GetPhD, and $\phi_2 = \Box$BeHappy at level $0, 1$, and 2, respectively. This domain can be specified using the following two initial goal axioms:

$$\begin{aligned}
&\text{(a) Init}(s) \supset ((G(p, 0, s) \equiv \exists s'.\ \text{Starts}(p, s') \wedge \text{Init}(s') \wedge \phi_0(p))\\
&\qquad\qquad\wedge (G(p, 1, s) \equiv \exists s'.\ \text{Starts}(p, s') \wedge \text{Init}(s') \wedge \phi_1(p))\\
&\qquad\qquad\wedge (G(p, 2, s) \equiv \exists s'.\ \text{Starts}(p, s') \wedge \text{Init}(s') \wedge \phi_2(p))),\\
&\text{(b) Init}(s) \wedge n \geq 3 \supset (G(p, n, s) \equiv \exists s'.\ \text{Starts}(p, s') \wedge \text{Init}(s')).
\end{aligned}$$

(a) specifies the p-goals ϕ_0, ϕ_1, ϕ_2 (from highest to lowest priority) of the agent in the initial situations, and makes $G(p, n, s)$ true for every path p that starts with an initial situation and over which ϕ_n holds, for $n = 0, 1, 2$; each of them defines a set of initial goal paths for a given priority level, and must be consistent. (b) makes $G(p, n, s)$ true for every path p that starts with an initial situation for $n \geq 3$. Thus at these levels, the agent has the trivial p-goal that she be in an initial situation. Given this axiomatization, it can be shown that in my example, I have the following:[25]

Proposition 4.2.3.

(i). For $n = 0, 1, 2,\ \Sigma^{\mathrm{P}} \cup \{\text{(a)}\} \models \text{OPGoal}(\phi_n \wedge \exists s.\ \text{Starts}(s) \wedge \text{Init}(s), n, S_0)$,

(ii). For $n \geq 3,\ \Sigma^{\mathrm{P}} \cup \{\text{(b)}\} \models \text{OPGoal}(\exists s.\ \text{Starts}(s) \wedge \text{Init}(s), n, S_0)$.

Recall that the paths in a G accessibility relation are the ones that the agent wants to actualize independently of what she knows. While p-goals or desires are allowed to be known to be impossible to bring about, an agent's c-goals or intentions must be compatible with what she knows [35]. Not all of the G-accessible paths are realistic in the sense that they start with a K-accessible situation. To filter these out, I define *realistic p-goal accessible paths G_R*:

Definition 4.2.4.

$$G_R(p, n, s) \overset{\text{def}}{=} G(p, n, s) \wedge \exists s'.\ \text{Starts}(p, s') \wedge K(s', s).$$

Thus a path p is G_R-accessible at level n in situation s if it is G-accessible at n in s, and if p starts with a situation that is K-related to s. The G_R relation prunes out the paths from the G relation that are known to be impossible, and since I define c-goals in terms of G_R, this ensures that agents' c-goals are realistic. I say that an agent has

[25]Here, Σ^{P} is an abbreviation for $\Sigma \cup \mathcal{D}_{path}$.

the *realistic p-goal* that ϕ at level n in situation s if ϕ holds over all paths that are G_R-accessible at n in s.

Definition 4.2.5.

$$\text{RPGoal}(\phi, n, s) \overset{\text{def}}{=} \forall p.\ G_R(p, n, s) \supset \phi(p).$$

In our example, assume that initially the agent knows that all of her p-goals are individually possible:

$$\text{(c). } \forall n.\ \exists p.\ G_R(p, n, S_0).$$

Given this, it can be shown that the agent's realistic p-goals in the initial situation S_0 are $\Box$BeRich, $\Diamond$GetPhD, and $\Box$BeHappy in order of priority:[26]

Proposition 4.2.6.

$$\Sigma_{\text{K}}^{\text{P}} \cup \{\text{(a)}\} \models \text{RPGoal}(\Box\text{BeRich}, 0, S_0) \wedge \text{RPGoal}(\Diamond\text{GetPhD}, 1, S_0)$$
$$\wedge\ \text{RPGoal}(\Box\text{BeHappy}, 2, S_0).$$

Using realistic p-goals, I next define c-goals. The idea of how I specify c-goal accessible paths is as follows. The set of G_R-accessibility relations represents a set of prioritized temporal formulae that are candidates for the agent's c-goals. Given G_R,

[26]Here $\Sigma_{\text{K}}^{\text{P}}$ is an abbreviation for $\Sigma \cup \mathcal{D}_{know} \cup \mathcal{D}_{path}$. Also, note that we don't need to use Axiom (c) to prove this proposition – if e.g., $\Box$BeRich is initially known to be impossible, then by Definitions 4.2.4 and 4.2.5, the agent trivially has the realistic p-goal that $\Box$BeRich at level 0, as in that case her set of G_R-accessible paths at level 0 is empty (and similarly for $\Diamond$GetPhD and $\Box$BeHappy).

in each situation I want the agent's c-goals to be the *maximal consistent set* of her higher priority realistic p-goals in that situation. I formalize this iteratively starting with the set of all realistic paths, i.e. paths that start with K-accessible situations. At each iteration i, I take the intersection of this set with the next highest priority set of G_R-accessible paths. If the intersection is not empty, I thus obtain a new chosen set of paths at level i. I call p-goals chosen by this process *active p-goals*. If on the other hand the intersection is empty, then it must be the case that a p-goal at this level is either in conflict with another active higher priority p-goal/a combination of two or more active higher priority p-goals, or is known to be impossible. In that case, all the p-goals at that level are ignored (i.e. marked as *inactive*), and the chosen set of paths at level i is the same as at level $i - 1$. To get the intersection of the first n priority levels, I repeat this until I reach $i = n$. Axiom 4.2.7 specifies this intersection (here **if** ϕ **then** ϕ_a **else** ϕ_b is an abbreviation for $(\phi \supset \phi_a) \wedge (\neg\phi \supset \phi_b)$):

Axiom 4.2.7.

$$
\begin{aligned}
G_\cap(p,n,s) \equiv\ & \textbf{if } (n=0) \textbf{ then} \\
& \qquad \textbf{if } \exists p'.\ G_R(p',n,s) \textbf{ then } G_R(p,n,s) \\
& \qquad \textbf{else } \exists s'.\ \text{Starts}(p,s') \land K(s',s) \\
& \textbf{else} \\
& \qquad \textbf{if } \exists p'.(G_R(p',n,s) \land G_\cap(p',n-1,s)) \\
& \qquad\qquad \textbf{then } (G_R(p,n,s) \land G_\cap(p,n-1,s)) \\
& \qquad \textbf{else } G_\cap(p,n-1,s).
\end{aligned}
$$

In the above axiom, $G_\cap(p,n,s)$ denotes that path p is in the intersection of the set of realistic paths in situation s up to level n. It has two cases, each with two sub-cases. The base iteration, when $n = 0$, specifies $G_\cap$ in terms of the highest priority G_R-accessible paths. If the highest priority goal is realistic, i.e. there is a G_R-accessible path at level 0 in situation s, then the $G_\cap$ relation at level 0 in s consist of the paths that are G_R-accessible at level 0 in s, i.e. $G_\cap(p,0,s) \equiv G_R(p,0,s)$. Otherwise, $G_\cap$ at 0 in s is specified to include all realistic paths in s, i.e. those that start with a K-accessible situation in s, and thus $G_\cap(p,0,s) \equiv \exists s'.\ \text{Starts}(p,s') \land K(s',s)$. For each n s.t. $n > 0$, the $G_\cap$ relation at level n in situation s is specified to be the prioritized intersection of the G_R relation at the first n priority levels (giving priority to the most important goals

– the ones with lower n's – while maintaining consistency). If there is a path that is G_R-accessible at level n in s and that is in the intersection of the first $n-1$ priority levels (i.e. $G_\cap$ at level $n-1$ in s), then this intersection includes all such paths, i.e. $G_\cap(p,n,s) \equiv G_R(p,n,s) \wedge G_\cap(p,n-1,s)$. Otherwise it is the same as that of the intersection of the first $n-1$ priority levels, i.e. $G_\cap(p,n,s) \equiv G_\cap(p,n-1,s)$.

Using this, I define what it means for an agent to have a c-goal at some level n as follows:

Definition 4.2.8.

$$\text{CGoal}(\phi,n,s) \stackrel{\text{def}}{=} \forall p.\ G_\cap(p,n,s) \supset \phi(p).$$

Thus an agent has the c-goal at level n in situation s that ϕ if ϕ holds over all paths that are in the prioritized intersection of the set of G_R-accessible paths up to level n in situation s.

I define c-goal accessible paths in terms of c-goal accessible paths at level n:

Definition 4.2.9.

$$G_\cap(p,s) \stackrel{\text{def}}{=} \forall n.\ G_\cap(p,n,s).$$

Thus a path is c-goal accessible in situation s if for all levels n, it is c-goal accessible at n in s. Using this, I define c-goals as follows:

Definition 4.2.10.

$$\text{CGoal}(\phi, s) \overset{\text{def}}{=} \forall p.\ G_\cap(p, s) \supset \phi(p).$$

That is, the agent has the c-goal that ϕ in situation s if ϕ holds over all of her $G_\cap$-accessible paths in s. Note that, by Axiom 4.2.7 and Definitions 4.2.9 and 4.2.10, a lower priority realistic goal is added as a c-goal only if it is consistent with the higher priority goals that are already chosen as c-goals. In Proposition 4.4.5 below, I formally show that an agent's chosen lower priority goals must be consistent with higher priority ones. Thus, an agent's set of c-goals in some situation is the maximal consistent set of her higher priority realistic p-goals in that situation.

Returning to our example, assume that the agent knows that her p-goal to eventually get a Ph.D. is inconsistent with her highest priority p-goal of always being rich as well as with her p-goal of always being happy, while the latter are consistent with each other:

$$\begin{aligned}&\text{(d). } \neg(\exists p.\ G_R(p, 0, S_0) \wedge G_R(p, 1, S_0)) \wedge \neg(\exists p.\ G_R(p, 1, S_0) \wedge G_R(p, 2, S_0))\\ &\quad \wedge (\exists p.\ G_R(p, 0, S_0) \wedge G_R(p, 2, S_0)).\end{aligned}$$

Then it can be shown that initially our example agent has the c-goals that $\square$BeRich and $\square$BeHappy, but not $\diamond$GetPhD:

Proposition 4.2.11.

$$\Sigma_{\mathrm{K}}^{\mathrm{P}} \cup \{\text{Axiom 4.2.7, (a)–(d)}\} \models \mathrm{CGoal}(\Box\mathrm{BeRich} \wedge \Box\mathrm{BeHappy}, S_0)$$

$$\wedge \neg\mathrm{CGoal}(\Diamond\mathrm{GetPhD}, S_0).$$

Proof Sketch. According to Axiom 4.2.7, the $G_\cap$-accessible paths at level 0 in S_0 are the ones that start with a K-accessible situation in S_0 and where $\Box$BeRich holds, since by Axiom (a) the agent initially has the p-goal that she be in an initial situation and that $\Box$BeRich at level 0, and by Axiom (c) $\Box$BeRich is initially possible, i.e., there is a G_R-accessible path over which $\Box$BeRich holds. Moreover, the $G_\cap$-accessible paths at level 1 in S_0 are the same as at level 0, since by Axiom (d) there is no realistic path over which both $\Diamond$GetPhD and $\Box$BeRich hold. Again, the $G_\cap$-accessible paths at level 2 in S_0 are those that start with a K-accessible situation and over which $\Box$BeRich $\wedge$ $\Box$BeHappy holds, as by Axiom (a) the agent initially has the p-goal that she be in an initial situation and that $\Box$BeHappy at level 2, by Axiom (c) $\Box$BeHappy is initially known to be possible, and by Axiom (d) $\Box$BeRich and $\Box$BeHappy are initially known to be mutually consistent. Finally, the $G_\cap$-accessible paths at any level greater than 2 in S_0 are the same as level 2 since by Axiom (b), any $G_\cap$-accessible path at level 2 is also G_R-accessible at these levels. The proposition thus follows from this and Definitions 4.2.9 and 4.2.10. $\Box$

Note that according to this definition of c-goals, the agent can have a c-goal that ϕ in situation s for various reasons. First of all, ϕ might be known to be inevitable in s, i.e. ϕ might hold over all paths that start with a knowledge-accessible situation (and thus over all G_R-accessible paths) in s. Secondly, ϕ might be an active p-goal at some level n in s. Finally, ϕ might be a consequence of two or more active p-goals at different levels in s.[27] To be able to refer to c-goals for which the agent has a primitive motivation, i.e. c-goals that result from a single only p-goal at some active priority level n, in contrast to those that are known to be inevitable or those that hold as a consequence of two or more active p-goals at different priority levels, I define the notion of *primary c-goals*:

Definition 4.2.12.

$$\text{PrimCGoal}(\phi, s) \overset{\text{def}}{=} \exists n.\ \text{PGoal}(\phi, n, s) \land \exists p.\ G(p, n, s) \land G_\cap(p, n, s).$$

An agent has the primary c-goal that ϕ in situation s, if ϕ is a p-goal at some level n in s, and there is a G-accessible path p at n in s that is also in the prioritized intersection of G_R-accessible paths up to n in s. The conjunct $\exists p.\ G(p, n, s) \land G_\cap(p, n, s)$ is required to ensure that n is indeed an active level, since having a p-goal that ϕ does not necessarily imply that the agent has the c-goal that ϕ. Also, while one might be tempted

[27]By Axiom 4.2.7 and Definitions 4.2.9 and 4.2.10, agents' c-goals are closed under logical consequence.

to define primary c-goals as $\exists n.\ \text{PGoal}(\phi, n, s) \wedge \text{CGoal}(\phi, s)$, this is inadequate since it does not guarantee that the agent choose ϕ due to the presence of her p-goal that ϕ at level n. For example, it might be the case that there is no G_R-accessible path p at n that is in $G_\cap(p, n, s)$ – this might happen if another p-goal at level n becomes impossible or becomes inconsistent with higher priority active goals – and that the agent choose ϕ as a consequence of other chosen G-accessibility levels. As well, defining primary c-goals as $\exists n.\ \text{OPGoal}(\phi, n, s) \wedge \text{CGoal}(\phi, s)$ is also problematic, since it only allows some of the p-goals of the agent, namely the only p-goals, to be her primary c-goals.

Thus if an agent has a primary c-goal that ϕ, then she also has the c-goal that ϕ, but not necessarily vice-versa. It can be shown that initially our example agent has the primary c-goals that $\Box$BeRich and $\Box$BeHappy, but not their conjunction:

Proposition 4.2.13.

$$\Sigma_{\text{K}}^{\text{P}} \cup \{\text{Axiom 4.2.7, (a)–(d)}\} \models \text{PrimCGoal}(\Box\text{BeRich}, S_0)$$
$$\wedge\ \text{PrimCGoal}(\Box\text{BeHappy}, S_0) \wedge \neg\text{PrimCGoal}(\Box\text{BeRich} \wedge \Box\text{BeHappy}, S_0).$$

To some extent, this shows that primary c-goals are not closed under logical consequence. In this sense, my formalization of primary c-goals is related to the non-normal modal formalizations of intentions found in the literature [124], and as such it does not suffer from the "side-effect problem" discussed in Chapter 2. To borrow an example from [35], in this framework an agent can have the primary c-goal to get her teeth fixed and know that this always involves pain, but not have the primary c-goal to have pain.

I also define a useful version of PrimCGoal, PrimCGoal(ϕ, n, s), that makes explicit the level n where ϕ is a primitive chosen goal:

Definition 4.2.14.

$$\text{PrimCGoal}(\phi, n, s) \overset{\text{def}}{=} \text{PGoal}(\phi, n, s) \land \exists p.\ G(p, n, s) \land G_{\cap}(p, n, s).$$

4.3 Goal Dynamics

An agent's goals change when her knowledge changes as a result of the occurrence of an action (including exogenous actions/events), or when she adopts or drops a goal. I formalize this by specifying how an agent's p-goals change. Her c-goals are then obtained using (realistic) p-goals in every new situation as above.

I introduce two actions for adopting and dropping a p-goal, $adopt(\phi, n)$ and $drop(\phi)$. The action precondition axioms for these are as follows. An agent can adopt the p-goal that ϕ at level n in situation s if she does not already have ϕ as her p-goal at some level n' in s:

Axiom 4.3.1.

$$\text{Poss}(adopt(\phi, n), s) \equiv \neg\exists n'.\ \text{PGoal}(\phi, n', s).$$

An agent can drop the p-goal that ϕ in situation s if she has the p-goal that ϕ at some level n in s:

Axiom 4.3.2.

$$\text{Poss}(drop(\phi), s) \equiv \exists n.\ \text{PGoal}(\phi, n, s).$$

I also assume the availability of unique names axioms for $adopt$ and $drop$ as in axiom schemata 3.3.2 and 3.3.3.

In the following, I specify the dynamics of p-goals by giving a successor-state axiom for the G relation, and then discuss each case, one at a time:

Axiom 4.3.3 (SSA for G)**.**

$$\begin{aligned}
&G(p, n, do(a, s)) \equiv \\
&\qquad \forall \phi, m.\ (a \neq adopt(\phi, m) \wedge a \neq drop(\phi) \wedge \text{Progressed}(p, n, a, s)) \vee \\
&\qquad \exists \phi, m.\ (a = adopt(\phi, m) \wedge \text{Adopted}(p, n, m, a, s, \phi)) \vee \\
&\qquad \exists \phi.\ (a = drop(\phi) \wedge \text{Dropped}(p, n, a, s, \phi)).
\end{aligned}$$

The overall idea of the successor-state axiom for G is as follows. First of all, to handle the occurrence of a non-adopt/drop (i.e. a regular) action a, I progress all G-accessible paths at all levels to reflect the fact that this action has just happened; this is done using the Progressed(p, n, a, s) construct, which replaces each G-accessible path p'

with starting situation s' at level n in situation s, by its suffix p provided that it starts with $do(a, s')$:

Definition 4.3.4.

$$\text{Progressed}(p, n, a, s) \overset{\text{def}}{=} \exists p', s'.\ G(p', n, s) \land \text{Starts}(p', s') \land \text{Suffix}(p, p', do(a, s')).$$

Any path over which the next action performed is not a is eliminated from the G-accessibility level being considered.

Secondly, to handle the adoption of a p-goal ϕ at level m, I insert a new temporal goal that ϕ to the agent's goal hierarchy at m by modifying the G-relation in the following manner. The G-accessible paths at all levels above m are progressed as above. The G-accessible paths at level m are the ones that starts with a situation that share the same history with $do(a, s)$ and over which ϕ holds. The G-accessible paths at all levels below m are the ones that can be obtained by progressing the level immediately above it. Thus the agent acquires the p-goal that ϕ at level m, and all the p-goals with priority m or less in s are pushed down one level in the hierarchy.

Definition 4.3.5.

$$\text{Adopted}(p, n, m, a, s, \phi) \overset{\text{def}}{=}$$

$$\textbf{if } (n < m) \textbf{ then } \text{Progressed}(p, n, a, s)$$

$$\textbf{else if } (n = m) \textbf{ then } \exists s'.\ \text{Starts}(p, s') \wedge \text{SameHist}(s', do(a, s)) \wedge \phi(p)$$

$$\textbf{else } \text{Progressed}(p, n - 1, a, s).$$

Finally, to handle the dropping of a p-goal ϕ, I replace the temporal formulae that imply the dropped goal in the agent's goal hierarchy by the trivial proposition that the history of actions in the current situation has occurred. Thus, in addition to progressing all G-accessible paths as above, I add back all paths that share the same history with $do(a, s)$ to the existing G-accessibility levels where the agent has the p-goal that ϕ.

Definition 4.3.6.

$$\text{Dropped}(p, n, a, s, \phi) \overset{\text{def}}{=}$$

$$\textbf{if } \text{PGoal}(\phi, n, s) \textbf{ then } \exists s'.\ \text{Starts}(p, s') \wedge \text{SameHist}(s', do(a, s))$$

$$\textbf{else } \text{Progressed}(p, n, a, s).$$

In our example, recall that our agent has the c-goals/active p-goals in S_0 that $\Box$BeRich and $\Box$BeHappy, but not $\Diamond$GetPhD, since the latter is inconsistent with her higher priority p-goal $\Box$BeRich. Assume that, after the exogenous event/action *goBankrupt* happens in S_0, i.e. in $S_1 = do(goBankrupt, S_0)$, the p-goal $\Box$BeRich becomes impossible while the other p-goals remain possible (assume that a BAT for this domain implying this has been specified):

$$\text{(e). } \neg\exists p.\ G_R(p, 0, S_1) \wedge \exists p'.\ G_R(p', 1, S_1) \wedge \exists p''.\ G_R(p'', 2, S_1).$$

Then in S_1, the agent has the c-goal that $\Diamond$GetPhD, but not $\Box$BeRich nor $\Box$BeHappy:[28]

Proposition 4.3.7.

$$\Sigma_{\mathrm{K}}^{\mathrm{P}} \cup \{\text{Axiom 4.2.7, Axioms 4.3.1–4.3.3, (a)–(e)}\} \models \mathrm{CGoal}(\Diamond\mathrm{GetPhD}, S_1)$$
$$\wedge \neg\mathrm{CGoal}(\Box\mathrm{BeRich}, S_1) \wedge \neg\mathrm{CGoal}(\Box\mathrm{BeHappy}, S_1).$$

$\Box$BeRich is excluded from the set of c-goals since by Axiom (e), it has become impossible to bring about (i.e. unrealistic). Also, since her higher priority active p-goal of eventually getting a Ph.D. is inconsistent with her p-goal of always being happy in S_1,[29] the agent will make $\Box$BeHappy inactive.

Note that, while it might be reasonable to drop a p-goal (e.g., $\Diamond$GetPhD) that is in conflict with another higher priority active p-goal (e.g., $\Box$BeRich in the initial situation), in my framework I keep such p-goals around. The reason for this is that although $\Box$BeRich is initially inconsistent with $\Diamond$GetPhD, the agent might later learn

[28]In the following, I assume that unique names axioms for $goBankrupt$ are given.

[29]Since by the unique name axioms, $goBankrupt$, $adopt$, and $drop$ all refer to different actions, from Axiom 4.3.3, it follows that after the $goBankrupt$ action happens, all G-accessible paths at all priority levels are simply progressed. This and Axiom (d) imply that the p-goals that $\Diamond$GetPhD and $\Box$BeHappy remain inconsistent in S_1.

that $\square$BeRich has become impossible to bring about (e.g., after $goBankrupt$ occurs), and then might want to pursue $\diamond$GetPhD. Thus, it is useful to keep these inactive p-goals since this allows the agent to optimize her chosen goals by taking advantage of such opportunities. As mentioned earlier, c-goals are my analogue to intentions. Recall from Chapter 2 that Bratman's [20] model of intentions limits the agent's practical reasoning – agents do not always optimize their utility and don't always reconsider all available options in order to allocate their reasoning effort wisely. In contrast to this, my c-goals are defined in terms of the p-goals, and at every step, I ensure that the agent optimizes her c-goals so that these are the set of highest priority goals that are consistent given the agent's knowledge. Thus, my notion of c-goal is not as persistent as Bratman's notion of intention. For instance as mentioned above, after the action $goBankrupt$ happens in S_0, the agent will lose the c-goal that $\square$BeHappy, although she did not drop it and it did not become impossible or achieved. In this sense, my model is that of an idealized optimizing agent. There is a tradeoff between optimizing the agent's chosen set of prioritized goals and being committed to chosen goals. In my framework, chosen goals behave like intentions with an automatic filter-override mechanism [20] that forces the agent to drop her chosen goals when opportunities to commit to other higher priority goals arise. In the future, it would be interesting to develop a logical model that captures the pragmatics of intention reconsideration by supporting control over it.

4.4 Properties

I now show that my formalization of prioritized goals for optimizing agents has some desirable properties. Let $\mathcal{D}_{OAgt}$ consist of $\Sigma, \mathcal{D}_{know}, \mathcal{D}_{path},$ and Axioms 4.2.7 and 4.3.1–4.3.3 and the associated definitions as in the two previous sections. Also, given some situation s, let's call a priority level n an *active level* if there is a G-accessible path at level n in s that is also $G_\cap$-accessible up to n in s:

Definition 4.4.1.

$$\text{ActiveLevel}(n, s) \overset{\text{def}}{=} \exists p.\ G(p, n, s) \wedge G_\cap(p, n, s).$$

4.4.1 Basic Properties

An agent's chosen goals are consistent:

Proposition 4.4.2 (Consistency)**.**

$$\mathcal{D}_{OAgt} \models \forall s.\ \neg\text{CGoal}(\text{False}, s).$$

Proof. (By contradiction) Fix situation S_1 and assume that $\text{CGoal}(\text{False}, S_1)$. From this and Definitions 4.2.9 and 4.2.10, we have $\forall p, n.\ G_\cap(p, n, S_1) \supset \text{False}(p)$. Fix such an n, say $n = 0$. Thus we have: $\forall p.\ G_{\cap}(p, 0, S_1) \supset \text{False}(p)$. Since we are dealing with possible worlds, this is only possible when $\neg\exists p.\ G_{\cap}(p, 0, S_1)$. By Axiom 4.2.7, this follows only if there is no K-related situation in S_1, or there is no path p s.t. p starts with a K-related situation in S_1, i.e.:

$$\neg\exists p, s.\ \text{Starts}(p, s) \wedge K(s, S_1). \tag{4.1}$$

But the former is impossible since (as discussed in Chapter 3) by Axiom 3.4.2 and 3.4.10, the K relation is reflexive, and thus $K(S_1, S_1)$. Also, by Lemmata 3.5.35 and 3.5.36, there is indeed a path that starts with S_1. But this is contradictory to (4.1). □

Thus, the agent cannot have both ϕ and $\neg\phi$ as c-goals in a situation s. Even if all of the agent's p-goals become known to be impossible, the set of c-goal accessible paths will be precisely those that start with a K-accessible situation, and thus the agent will only choose the formulae that are known to be inevitable.

I also have the property of realism [35], i.e. if an agent knows that something has become inevitable, then she has this as a c-goal:

Proposition 4.4.3 (Realism)**.**

$$\mathcal{D}_{OAgt} \models \forall s.\ \text{KInevitable}(\phi, s) \supset \text{CGoal}(\phi, s).$$

Proof. Fix S_1 and ϕ_1. It follows from the antecedent (i.e., that KInevitable(ϕ_1, S_1)) and Definitions 3.5.14, 3.5.12, and 3.4.5 that ϕ_1 holds over all paths that starts with a situation that is K-accessible in S_1:

$$\forall p, s.\ \text{Starts}(p, s) \wedge K(s, S_1) \supset \phi_1(p). \tag{4.2}$$

Also, from Axiom 4.2.7 and by induction on n, it follows that:

$$\forall p, n.\ G_\cap(p, n, S_1) \supset \exists s.\ \text{Starts}(p, s) \wedge K(s, S_1). \tag{4.3}$$

The proposition follows from (4.2), (4.3), and Definitions 4.2.9 and 4.2.10. □

Note that this is not necessarily true for p-goals and primary c-goals – an agent may know that something has become inevitable and not have it as her p-goal/primary c-goal, which is intuitive. While the property of realism is often criticized [173, 174], one should view these inevitable goals as something that holds in the worlds that the agent intends to bring about, rather than something that the agent is actively pursuing.

A consequence of Proposition 4.4.2 and 4.4.3 is that an agent does not have a c-goal that is known to be impossible:

Corollary 4.4.4.

$$\mathcal{D}_{OAgt} \models \forall s.\ \text{CGoal}(\phi, s) \supset \neg\text{KImpossible}(\phi, s).$$

Proof. Fix ϕ_1 and S_1. From the antecedent (i.e., that CGoal(ϕ_1, S_1)) and Proposition 4.4.2, we have $\neg$CGoal$(\neg\phi_1, S_1)$. From this and Proposition 4.4.3, we have $\neg$KInevitable$(\neg\phi_1, S_1)$. The consequent follows from this and Definition 3.5.15. □

An agent prefers higher priority goals to lower priority goals, and thus her chosen lower priority goals must be consistent with higher priority ones:

Proposition 4.4.5.

$$\forall n, m, s.\ \text{CGoal}(\phi, m, s) \wedge n > m \supset \text{CGoal}(\phi, n, s).$$

Proof. Fix ϕ_1, N_1, M_1, and S_1. From the antecedent (i.e. that CGoal(ϕ_1, M_1, S_1)) and Definition 4.2.8, we have:

$$\forall p.\ G_\cap(p, M_1, S_1) \supset \phi_1(p). \tag{4.4}$$

From Axiom 4.2.7 and by induction on n, we have: $\forall p, m, n, s.\ n > m \supset G_\cap(p, n, s) \supset G_\cap(p, m, s)$. From this and the antecedent (i.e. that $N_1 > M_1$), we have:

$$\forall p.\ G_\cap(p, N_1, S_1) \supset G_\cap(p, M_1, S_1). \tag{4.5}$$

The consequent follows from (4.4), (4.5), and Definition 4.2.8. □

4.4.2 Dynamic Properties

I next discuss some properties of the framework w.r.t. goal change. First I show that an agent always wants to be in a world that has the same action history as the current situation, provided that initially she wants to be in an initial world.

Proposition 4.4.6 (Correct Action History)**.**

$$\mathcal{D}_{OAgt} \models (\forall p, n, s.\ \text{Init}(s) \land G(p, n, s) \supset \exists s'.\ \text{Starts}(p, s') \land \text{Init}(s'))$$

$$\supset (\forall p, n, s.\ G(p, n, s) \supset \exists s'.\ \text{Starts}(p, s') \land \text{SameHist}(s, s')).$$

Proof. (By induction on s) Follows from Axiom 4.3.3 and Definitions 4.3.4, 4.3.5, 4.3.6, 3.5.16, and 3.2.7. □

Adopting and dropping logically equivalent goals has the same result:

Proposition 4.4.7 (Extensionality w.r.t. Adoption and Drop)**.**

(a). $\mathcal{D}_{OAgt} \models (\forall p.\ \phi_1(p) \equiv \phi_2(p)) \supset$

$$(\forall n, n', s.\ \text{PGoal}(\psi, n', do(adopt(\phi_1, n), s)) \equiv \text{PGoal}(\psi, n', do(adopt(\phi_2, n), s))),$$

(b). $\mathcal{D}_{OAgt} \models (\forall p.\ \phi_1(p) \equiv \phi_2(p)) \supset$

$$(\forall n, s.\ \text{PGoal}(\psi, n, do(drop(\phi_1), s)) \equiv \text{PGoal}(\psi, n, do(drop(\phi_2), s))),$$

(c). $\mathcal{D}_{OAgt} \models (\forall p.\ \phi_1(p) \equiv \phi_2(p)) \supset$

$$(\forall n, s.\ \text{CGoal}(\psi, do(adopt(\phi_1, n), s)) \equiv \text{CGoal}(\psi, do(adopt(\phi_2, n), s))),$$

(d). $\mathcal{D}_{OAgt} \models (\forall p.\ \phi_1(p) \equiv \phi_2(p)) \supset$

$$(\forall s.\ \text{CGoal}(\psi, do(drop(\phi_1), s)) \equiv \text{CGoal}(\psi, do(drop(\phi_2), s))).$$

Proof. (a). Follows from the fact that we use a possible worlds/paths semantics for p-goals.

(b). Similar to that of Proposition 4.4.7(a).

(c). Follows from Definition 4.2.10 and the fact that the $G_\cap$-accessible paths are the same in both situations given the antecedent.

(d). Similar to that of Proposition 4.4.7(c). □

As a consequence, this property also holds for primary c-goals:

Corollary 4.4.8.

(a). $\mathcal{D}_{OAgt} \models (\forall p.\ \phi_1(p) \equiv \phi_2(p)) \supset$

$(\forall n, s.\ \text{PrimCGoal}(\psi, do(adopt(\phi_1, n), s)) \equiv \text{PrimCGoal}(\psi, do(adopt(\phi_2, n), s))),$

(b). $\mathcal{D}_{OAgt} \models (\forall p.\ \phi_1(p) \equiv \phi_2(p)) \supset$

$(\forall s.\ \text{PrimCGoal}(\psi, do(drop(\phi_1), s)) \equiv \text{PrimCGoal}(\psi, do(drop(\phi_2), s))).$

Proof. (a). Follows from Definition 4.2.12 and Proposition 4.4.7(a).

(b). Similar to that of Corollary 4.4.8(a). □

An agent acquires the p-goal that ϕ at level n after she adopts it at n in some situation s:

Proposition 4.4.9 (Adoption-1)**.**

$$\mathcal{D}_{OAgt} \models \text{PGoal}(\phi, n, do(adopt(\phi, n), s)).$$

Proof. Fix ϕ_1, N_1, and S_1. From Axiom 4.3.3 and Definition 4.3.5, we have that the agent's G-accessible paths at N_1 in $do(adopt(\phi_1, N_1), S_1)$ are the ones that start with situations that have the same history as $do(adopt(\phi_1, N_1), S_1)$ and over which ϕ_1 holds:

$$\begin{aligned}&\forall p.\ G(p, N_1, do(adopt(\phi_1, N_1), S_1)) \equiv \\ &\quad \exists s.\ \text{Starts}(p, s) \wedge \text{SameHist}(s, do(adopt(\phi_1, N_1), S_1)) \wedge \phi_1(p).\end{aligned} \tag{4.6}$$

If such a path p exists, then the consequent follows from (4.6) and Definition 4.2.1. Otherwise, the consequent holds trivially from Definition 4.2.1. □

For the next property, I'll need to use the following two lemmata. The first says that if p is in the $G_\cap$-relation up to some level n in situation s, then the starting situation of p must be K-accessible from s:

Lemma 4.4.10.

$$\mathcal{D}_{OAgt} \models \forall p, n, s, s'.\ G_\cap(p, n, s) \wedge \text{Starts}(p, s') \supset K(s', s).$$

Proof. By induction on n and using Axiom 4.2.7 and Definition 4.2.4. □

The second lemma says that an agent's active (and inactive) p-goals below some level n remain active (inactive, resp.) after she adopts a goal ϕ at level n in s, provided that ϕ is consistent with her c-goals up to level $n - 1$:

Lemma 4.4.11.

$$\mathcal{D}_{OAgt} \models$$

$$\neg\text{CGoal}(\neg\exists s', p'.\ \text{Starts}(s') \wedge \text{Suffix}(p', do(adopt(\phi, n), s')) \wedge \phi(p'), n-1, s)$$

$$\supset \forall m.\ m < n \supset (\text{ActiveLevel}(m, s) \equiv \text{ActiveLevel}(m, do(adopt(\phi, n), s)).$$

Proof. By induction on n. □

An agent acquires the primary c-goal (and thus the p-goal and c-goal) that ϕ after she adopts it at some level n in s provided that ϕ is consistent with her c-goals up to level $n-1$; this holds even if she has the inconsistent c-goal at some level that $\neg\phi$ next, provided that she adopts ϕ at a higher priority than all such inconsistent goals:

Proposition 4.4.12 (Adoption-2)**.**

$$\mathcal{D}_{OAgt} \models$$

$$\neg\text{CGoal}(\neg\exists s', p'.\ \text{Starts}(s') \wedge \text{Suffix}(p', do(adopt(\phi, n), s')) \wedge \phi(p'), n-1, s)$$

$$\supset \text{PrimCGoal}(\phi, do(adopt(\phi, n), s)).$$

Proof. Fix ϕ_1, N_1, and S_1. From the antecedent, we have:

$$\begin{aligned}\neg\text{CGoal}(\neg\exists s', p'.\ \text{Starts}(s') \wedge \text{Suffix}(p', do(adopt(\phi_1, N_1), s'))\\ \wedge\ \phi_1(p'), N_1 - 1, S_1).\end{aligned} \tag{4.7}$$

From Proposition 4.4.9, we have:

$$\text{PGoal}(\phi_1, N_1, do(adopt(\phi_1, N_1), S_1)). \tag{4.8}$$

Hence by Definition 4.2.12, to show that PrimCGoal$(\phi_1, do(adopt(\phi_1, N_1), S_1))$, we will need to show that:

$$\exists p.\ G(p, N_1, do(adopt(\phi_1, N_1), S_1)) \wedge G_\cap(p, N_1, do(adopt(\phi_1, N_1), S_1)).$$

Let me first prove that $\exists p.\ G(p, N_1, do(adopt(\phi_1, N_1), S_1))$. From Axiom 4.3.3 and Definition 4.3.5, we have:

$$\begin{aligned}&\forall p.\ G(p, N_1, do(adopt(\phi_1, N_1), S_1)) \equiv\\ &\qquad \exists s.\ \text{Starts}(p, s) \wedge \text{SameHist}(s, do(adopt(\phi_1, N_1), S_1)) \wedge \phi_1(p).\end{aligned} \tag{4.9}$$

From (4.7) and Definition 4.2.8, it follows that there is a path, say P_1, such that P_1 is in the prioritized intersection of G_R-accessible paths up to level $N_1 - 1$ in S_1, that the $adopt(\phi_1, N_1)$ action happens next along P_1, and that ϕ_1 holds afterwards:

$$\begin{aligned}&G_\cap(P_1, N_1 - 1, S_1) \wedge\\ &\exists s', p'.\ \text{Starts}(P_1, s') \wedge \text{Suffix}(p', P_1, do(adopt(\phi_1, N_1), s')) \wedge \phi_1(p').\end{aligned} \tag{4.10}$$

Consider the suffix of P_1 after $adopt(\phi_1, N_1)$ has happened; let's call this path P_2. Note that by (4.10) and Lemma 4.4.10, the starting situation of P_1 is K-accessible in S_1:

$$\text{Starts}(P_1, s) \supset K(s, S_1). \tag{4.11}$$

By this and Lemma 3.5.34, the starting situation of P_1 has the same action history as S_1. Thus it follows that the starting situation of P_2 must have the same history as $do(adopt(\phi_1, N_1), S_1)$:

$$\text{Starts}(P_2, s) \supset \text{SameHist}(s, do(adopt(\phi_1, N_1), S_1)). \tag{4.12}$$

Also, by (4.10), ϕ_1 holds over P_2:

$$\phi_1(P_2). \tag{4.13}$$

From (4.12), (4.13), and (4.9), it follows that P_2 is G-accessible at N_1 after the adopt action has happened in S_1:

$$G(P_2, N_1, do(adopt(\phi_1, N_1), S_1)). \tag{4.14}$$

Next, I will prove that $G_\cap(P_2, N_1, do(adopt(\phi_1, N_1), S_1))$. By Axiom 4.2.7, to prove this, it is sufficient to show that:

$$\begin{aligned}
&\text{If } N_1 = 0: \quad G_R(P_2, N_1, do(adopt(\phi_1, N_1), S_1));\\
&\text{If } N_1 > 0: \quad G_R(P_2, N_1, do(adopt(\phi_1, N_1), S_1))\\
&\qquad\qquad\qquad \wedge G_\cap(P_2, N_1 - 1, do(adopt(\phi_1, N_1), S_1)).
\end{aligned}$$

I will first show that $G_R(P_2, N_1, do(adopt(\phi_1, N_1), S_1))$. Note that by (4.10) and Definition 3.5.16, P_1 is a path and the first action that happens along P_1, i.e. in the starting situation of P_1, is $\mathit{adopt}(\phi_1, N_1)$. From this, Corollary 3.5.41, and Definition 3.3.1 it follows that:

$$\text{Starts}(P_1, s) \supset \text{Poss}(\mathit{adopt}(\phi_1, N_1), s). \tag{4.15}$$

Since $\mathit{adopt}(\phi_1, N_1)$ is not a knowledge-producing action, the K-accessible situations in $\mathit{do}(\mathit{adopt}(\phi_1, N_1), S_1)$ are those that can be obtained by performing this action over some K-accessible situation in S_1, provided that the adopt action is executable in that situation; thus it follows from (4.11), (4.15), and Axiom 3.4.10 that the starting situation of P_2 is K-accessible in $\mathit{do}(\mathit{adopt}(\phi_1, N_1), S_1)$:

$$\text{Starts}(P_2, s) \supset K(s, \mathit{do}(\mathit{adopt}(\phi_1, N_1), S_1)). \tag{4.16}$$

From this, (4.14), and Definition 4.2.4, it follows that:

$$G_R(P_2, N_1, \mathit{do}(\mathit{adopt}(\phi_1, N_1), S_1)). \tag{4.17}$$

Next, I will show that $G_\cap(P_2, N_1 - 1, \mathit{do}(\mathit{adopt}(\phi_1, N_1), S_1))$, where $N_1 > 0$. By (4.7) and Lemma 4.4.11, it follows that:

$$\forall n.\ n < N_1 \supset (\text{ActiveLevel}(n, S_1) \equiv \text{ActiveLevel}(n, \mathit{do}(\mathit{adopt}(\phi_1, n), S_1)).$$

Thus, a priority level that has higher priority than N_1 is active/chosen in $\mathit{do}(\mathit{adopt}(\phi_1, N_1), S_1)$ if and only if it is active/chosen in S_1. Now, consider one such active level M. By Axiom 4.2.7 and (4.10), it follows that $G_R(P_1, M, S_1)$. According to Axiom 4.3.3 and Definitions 4.3.4 and 4.3.5, after the $\mathit{adopt}(\phi_1, N_1)$ action happens in

S_1, the G-accessible paths at all levels that have higher priority than N_1 are simply progressed to reflect that this action has happened. From this, it follows that $G(P_2, M, do(adopt(\phi_1, N_1), S_1))$. Moreover, from this, (4.16), and Definition 4.2.4, we have $G_R(P_2, M, do(adopt(\phi_1, N_1),\ S_1))$. As this holds for all such active levels M where $M < N_1$, it follows from (4.16) and Axiom 4.2.7 that $G_\cap(P_2, N_1 - 1, do(adopt(\phi_1, N_1), S_1))$. Thus by (4.16), (4.17), and Axiom 4.2.7, we have:

$$G_\cap(P_2, N_1, do(adopt(\phi_1, N_1), S_1)). \tag{4.18}$$

The consequent follows from (4.8), (4.14), (4.18), and Definition 4.2.12. □

Recall that the agent's (primary) chosen goals, are like intentions, and as such they act as a filter for adopting newer goals. Proposition 4.4.12 ensures that the agent takes into consideration the priorities of goals when adopting a new goal that is inconsistent with her current chosen goals.

A consequence of this is that an agent acquires the primary c-goal that ϕ after she adopts it at some level n in some situation s, provided that she does not have the c-goal in s that $\neg\phi$ next:

Corollary 4.4.13 (Adoption-3)**.**

$$\mathcal{D}_{OAgt} \models \neg\text{CGoal}(\neg\exists s', p'.\ \text{Starts}(s') \wedge \text{Suffix}(p', do(adopt(\phi, n), s')) \wedge \phi(p'), s)$$
$$\supset \text{PrimCGoal}(\phi, do(adopt(\phi, n), s)).$$

Proof. Note that by Definitions 4.2.8, 4.2.9, and 4.2.10, we have that:

$$\neg\text{CGoal}(\neg\exists s', p'.\ \text{Starts}(s') \wedge \text{Suffix}(p', do(adopt(\phi, n), s')) \wedge \phi(p'), s) \supset$$

$$\neg\text{CGoal}(\neg\exists s', p'.\ \text{Starts}(s') \wedge \text{Suffix}(p', do(adopt(\phi, n), s')) \wedge \phi(p'), n-1, s).$$

Thus the corollary follows from Proposition 4.4.12 as a case of strengthening its antecedent. □

Let ProgOf(ϕ, a) denote the progression of a path formula ϕ after some action a has been performed, which is defined as:

Definition 4.4.14.

$$\text{ProgOf}(\phi, a)(p) \overset{\text{def}}{=} \exists p', s'.\ \text{Starts}(p', s') \wedge \text{Suffix}(p, p', do(a, s')) \wedge \phi(p').$$

Then I can show that after dropping the p-goal that ϕ at n in s, an agent does not have the p-goal (and thus the primary c-goal) that the progression of ϕ at n, i.e. ProgOf$(\phi, drop(\phi))$, provided that ProgOf$(\phi, drop(\phi))$ is not strongly inevitable in $do(drop(\phi), s)$:

Proposition 4.4.15 (Drop)**.**

$$\mathcal{D}_{OAgt} \models \text{PGoal}(\phi, n, s) \wedge \neg\text{StronglyInevitable}(\text{ProgOf}(\phi, drop(\phi)), do(drop(\phi), s))$$
$$\supset \neg\text{PGoal}(\text{ProgOf}(\phi, drop(\phi)), n, do(drop(\phi), s)),$$

Proof. Fix ϕ_1, N_1, and S_1. From the antecedent, we have:

$$\text{PGoal}(\phi_1, N_1, S_1), \tag{4.19}$$

$$\neg\text{StronglyInevitable}(\text{ProgOf}(\phi_1, drop(\phi_1)), do(drop(\phi_1), S_1)). \tag{4.20}$$

We can see from Axiom 4.3.3 and Definition 4.3.6 that after the $drop(\phi_1)$ action has been performed in S_1, each G-accessibility level in S_1 where ϕ_1 is a p-goal is replaced by the set of paths that starts with a situation that has the same history as $do(drop(\phi_1), S_1)$. Thus, by (4.19) and Axiom 4.3.3, we have:

$$G(p, N_1, do(drop(\phi_1), S_1)) \equiv \exists s'.\ \text{Starts}(p, s') \land \text{SameHist}(s', do(drop(\phi_1), S_1)). \tag{4.21}$$

Now, by Definition 3.5.13 and (4.20), there exists a path P_1 that starts with a situation that has the same history as $do(drop(\phi_1), S_1)$ and over which $\neg\text{ProgOf}(\phi_1, drop(\phi_1))$ holds:

$$\exists s'.\ \text{Starts}(P_1, s') \land \text{SameHist}(s', do(drop(\phi_1), S_1)) \land \neg\text{ProgOf}(\phi_1, drop(\phi_1))(P_1). \tag{4.22}$$

The consequent follows from (4.21), (4.22), and Definition 4.2.1. □

Note that, Proposition 4.4.15 does not necessarily hold for CGoal, as ϕ could still be a consequence of the agent's remaining primary c-goals. Also, it does not hold in general for RPGoal, since it might be the case that the progression of ϕ is inevitable over all G_R-accessible paths; however, we could obtain an analogous result by adding the negation of this as a condition.

4.4.3 Goal Introspection

I want my agents to be able to introspect their goals – if an agent has a realistic p-goal that ϕ, she should know that she has this as her goal; moreover if she does not have the realistic p-goal that ϕ, she should know this. In the following, I identify constraints on K and G that yield these properties.[30]

To get positive introspection of realistic p-goals, we need a constraint similar to transitivity, which I call $KGTrans$:

Definition 4.4.16.

$$KGTrans(n, s) \overset{\text{def}}{=} \forall s_1, s_2, p.\ K(s_1, s) \land K(s_2, s_1) \land G(p, n, s_1) \land \text{Starts}(p, s_2) \\ \supset G(p, n, s).$$

If this constraint is satisfied for some priority level n, then the agents will have positive introspection of realistic p-goals at n:[31]

[30]These constraints and associated propositions are closely related to those given by Shapiro [194]; however they are adapted to work for infinite paths.

[31]Note that for this to hold, we also need K to be transitive, but this follows from $\mathcal{D}_{OAgt}$.

Proposition 4.4.17.

$$\mathcal{D}_{OAgt} \models \forall s, n.\ KGTrans(n, s) \supset (\text{RPGoal}(\phi, n, s) \supset \text{Know}(\text{RPGoal}(\phi, n), s)).$$

Proof. (By contradiction) Fix ϕ_1, N_1, and S_1, and assume that:

$$KGTrans(N_1, S_1), \tag{4.23}$$

$$\text{RPGoal}(\phi_1, N_1, S_1). \tag{4.24}$$

Also assume that:

$$\neg\text{Know}(\text{RPGoal}(\phi_1, N_1), S_1). \tag{4.25}$$

From (4.25) and Definitions 4.2.5, 4.2.4, and 3.4.5, it follows that there is a path P_1 and situations S_1^1 and S_1^2 such that:

$$K(S_1^1, S_1) \wedge K(S_1^2, S_1^1) \wedge \text{Starts}(P_1, S_1^2) \wedge G(P_1, N_1, S_1^1) \wedge \neg\phi_1(P_1). \tag{4.26}$$

From this and the transitivity of K (i.e. Axiom 3.4.3), it follows that:

$$K(S_1^2, S_1). \tag{4.27}$$

From this, (4.24), (4.26), and Definitions 4.2.5 and 4.2.4, it follows that:

$$\neg G(P_1, N_1, S_1). \tag{4.28}$$

But it follows from (4.23), (4.26), and Definition 4.4.16 that $G(P_1, N_1, S_1)$, which is contradictory to (4.28). □

To get negative introspection of realistic p-goals, we need a constraint similar to Euclideanism. I call this constraint $KGEuc$:

Definition 4.4.18.

$$KGEuc(n, s) \overset{\text{def}}{=} \forall s_1, s_2, p.\ K(s_1, s) \wedge K(s_2, s) \wedge G(p, n, s) \wedge \text{Starts}(p, s_2)$$

$$\supset G(p, n, s_1).$$

If this constraint is satisfied for some priority level n, then the agents will have negative introspection of realistic p-goals at n:[32]

Proposition 4.4.19.

$$\mathcal{D}_{OAgt} \models \forall s, n.\ KGEuc(n, s) \supset (\neg\text{RPGoal}(\phi, n, s) \supset \text{Know}(\neg\text{RPGoal}(\phi, n), s)).$$

Proof. (By contradiction) Fix ϕ_1, N_1, and S_1, and assume that:

$$KGEuc(N_1, S_1), \tag{4.29}$$

$$\neg\text{RPGoal}(\phi_1, N_1, S_1). \tag{4.30}$$

Also assume that:

$$\neg\text{Know}(\neg\text{RPGoal}(\phi_1, N_1), S_1). \tag{4.31}$$

[32]Again, we need K to be Euclidean, which follows from $\mathcal{D}_{OAgt}$.

From (4.30) and Definitions 4.2.5 and 4.2.4, it follows that there is a path P_1 and situations S_1^1 such that:

$$K(S_1^1, S_1) \wedge \text{Starts}(P_1, S_1^1) \wedge G(P_1, N_1, S_1) \wedge \neg\phi_1(P_1). \tag{4.32}$$

From (4.31) and Definitions 4.2.5, 4.2.4, and 3.4.5, it follows that there is a K-accessible situation in S_1 where the agent indeed has the RPGoal that ϕ_1 at N_1. Pick such a K-accessible situation in S_1; let's call it S_1^2. Thus, we have:

$$K(S_1^2, S_1) \wedge \forall p, s.\ (\text{Starts}(p, s) \wedge K(s, S_1^2) \wedge G(p, N_1, S_1^2) \supset \phi_1(p)). \tag{4.33}$$

From (4.32), (4.33), and the Euclideanism of K (i.e. Axiom 3.4.4), it follows that:

$$K(S_1^1, S_1^2). \tag{4.34}$$

Moreover, from (4.29), (4.32), (4.33), and Definition 4.4.18, it follows that:

$$G(P_1, N_1, S_1^2). \tag{4.35}$$

Finally, from (4.33), (4.32), (4.34), and (4.35), it follows that $\phi_1(P_1)$, which is contradictory to (4.32). $\square$

In the following, I show that for any executable situation, $KGTrans$ and $KGEuc$ persist if they hold in all initial situations as they are preserved by the successor-state axiom for G. First I show the persistence of $KGTrans$:

Theorem 4.4.20.

$$\mathcal{D}_{OAgt} \models (\forall n, s.\ \text{Init}(s) \supset KGTrans(n, s)) \supset$$
$$(\forall n, s.\ \text{Executable}(s) \supset KGTrans(n, s)).$$

Proof. (By induction on s) Assume that:

$$\forall n, s.\ \text{Init}(s) \supset KGTrans(n, s). \tag{4.36}$$

The base case, where s is an initial situation, is trivial. For the inductive case, we fix S_1 and A_1 and assume that:

$$\text{Executable}(do(A_1, S_1)). \tag{4.37}$$

Fix N_1. We need to show that $KGTrans(N_1, do(A_1, S_1))$. From (4.37) and Lemma 3.5.29, we have:

$$\text{Executable}(S_1). \tag{4.38}$$

This, (4.36), and the inductive hypothesis imply:

$$\forall n.\ KGTrans(n, S_1). \tag{4.39}$$

Assume that $S_2 = do(A_1, S_1)$. Let us expand $KGTrans(N_1, S_2)$; fix S_2^1, S_2^2, and P_2, and assume:

$$K(S_2^1, S_2), \tag{4.40}$$

$$K(S_2^2, S_2^1), \tag{4.41}$$

$$G(P_2, N_1, S_2^1), \tag{4.42}$$

$$\text{Starts}(P_2, S_2^2). \tag{4.43}$$

We need to show that $G(P_2, N_1, S_2)$. (4.40), (4.41), and Axiom 3.4.10 imply that there exist S_1^1 and S_1^2 such that:

$$K(S_1^1, S_1) \wedge S_2^1 = do(A_1, S_1^1), \text{ and} \tag{4.44}$$

$$K(S_1^2, S_1^1) \wedge S_2^2 = do(A_1, S_1^2). \tag{4.45}$$

Now, note that by (4.40), (4.41), and the transitivity of K, we have:

$$K(S_2^2, S_2).$$

From this and Lemma 3.5.34, we have:

$$\text{SameHist}(S_2^2, S_2). \tag{4.46}$$

Now, to show that P_2 is G-accessible at level N_1 in situation S_2, we will need to analyze the SSA for G. A close look at it gives us four cases with four different mutually exclusive conditions: Case 1, where one simply progresses the old set of G-accessible paths before the action A_1 has happened to obtain the new set of G-accessible paths after the occurrence of A_1, Case 2, which involves progression with shifting levels, Case 3, which involves processing/filtering the old set of G-accessible paths to handle the adoption of a goal at level N_1, and Case 4 that involves processing them to handle the dropping of a goal at level N_1. Let us discuss each case, one at a time. Thus the successor-state axiom for G (i.e. Axiom 4.3.3) and (4.42) give us four cases:

- **Case 1.** The action A_1 is a regular (non-adopt/drop action), or A_1 does not refer to the adoption of a goal ϕ at level N_1 or at some higher priority level than N_1, or it does not refer to the dropping of a goal ϕ at N_1, i.e.:

  $$\neg(A_1 = \mathit{adopt}(\phi, M) \wedge M \leq N_1) \wedge \neg(A_1 = \mathit{drop}(\phi) \wedge \text{PGoal}(\phi, N_1, S_1)).$$

  By the SSA for G and (4.42), in all these cases P_2 is the simple progression of some path P_1 that was G-accessible at N_1 in S_1^1:

  $$\text{Starts}(P_1, S_1^2) \wedge \text{Suffix}(P_2, P_1, \mathit{do}(A_1, S_1^2)) \wedge G(P_1, N_1, S_1^1). \qquad (4.47)$$

  Definition 4.4.16, (4.39), (4.44), (4.45), and (4.47) imply that:

  $$G(P_1, N_1, S_1). \qquad (4.48)$$

  By Axiom 4.3.3, Definitions 4.3.4, 4.3.5, and 4.3.6, the assumption for this case (i.e. that A_1 is a regular action, or that A_1 refers to the adoption of a goal at a lower priority level than N_1 or to the dropping of a goal at some other level than N_1), (4.47), and (4.48), the progression of P_1 (i.e. P_2) will be retained in the G-relation at N_1 in $S_2 = do(A_1, S_1)$; this is because $\text{Progressed}(P_2, N_1, A_1, S_1)$ holds. Thus we have $G(P_2, N_1, S_2)$ and we are done.

- **Case 2.** A_1 refers to the adoption of a goal ϕ at a higher priority level than N_1, i.e. $A_1 = adopt(\phi, M) \wedge M < N_1$. In this case, P_2 is the progression of some path P_1 that was G-accessible at $N_1 - 1$ in S_1^1 (since adopting the goal ϕ at higher priority than N_1 has pushed all the goals that has priority lower than $M - 1$ down by one level):

  $$\text{Starts}(P_1, S_1^2) \wedge \text{Suffix}(P_2, P_1, do(A_1, S_1^2)) \wedge G(P_1, N_1 - 1, S_1^1).$$

  The rest of the proof for this case is similar to that of Case 1.

- **Case 3.** A_1 refers to the adoption of a goal ϕ at N_1, i.e. $A_1 = adopt(\phi, N_1)$. Then P_2 is included in the G-relation at N_1 in S_2 if it starts with a situation that has the same history as in S_2, and if $\phi(P_2)$ holds. (4.43) and (4.46) imply this former. The latter also holds, otherwise by the SSA for G, P_2 would have not been included in the G-relation at N_1 in S_2^1 (but it is included by (4.42)).

- **Case 4.** A_1 refers to the dropping of a goal ϕ at N_1, i.e. $A_1 = drop(\phi)$, where $\text{PGoal}(\phi, N_1, S_1)$. In this case, P_2 is included in the G-relation at N_1 in S_2 if it starts with a situation that has the same history as in S_2. Again, (4.43) and (4.46) imply this condition.

The theorem thus follows. $\square$

I next show the persistence of $KGEuc$:

Theorem 4.4.21.

$$\mathcal{D}_{OAgt} \models (\forall n, s.\ \text{Init}(s) \supset KGEuc(n, s)) \supset$$
$$(\forall n, s.\ \text{Executable}(s) \supset KGEuc(n, s)).$$

Proof. (By induction on s) Assume that:

$$\forall n, s.\ \text{Init}(s) \supset KGEuc(n, s). \tag{4.49}$$

The base case, where s is initial, is trivial. For the inductive case, we fix S_1 and A_1 and assume that:

$$\text{Executable}(do(A_1, S_1)). \tag{4.50}$$

Fix N_1. We need to show that $KGEuc(N_1, do(A_1, S_1))$. From (4.50) and Lemma 3.5.29, we have:

$$\text{Executable}(S_1). \tag{4.51}$$

This, (4.49), and the inductive hypothesis imply:

$$\forall n.\ KGEuc(n, S_1). \tag{4.52}$$

Assume that $S_2 = do(A_1, S_1)$. Let us expand $KGEuc(N_1, S_2)$; fix S_2^1, S_2^2, and P_2, and assume:

$$K(S_2^1, S_2), \tag{4.53}$$

$$K(S_2^2, S_2), \tag{4.54}$$

$$G(P_2, N_1, S_2), \tag{4.55}$$

$$\text{Starts}(P_2, S_2^2). \tag{4.56}$$

We need to show that $G(P_2, N_1, S_2^1)$. (4.53), (4.54), and Axiom 3.4.10 imply that there exist S_1^1 and S_1^2 such that:

$$K(S_1^1, S_1) \wedge S_2^1 = do(A_1, S_1^1), \text{ and} \tag{4.57}$$

$$K(S_1^2, S_1) \wedge S_2^2 = do(A_1, S_1^2). \tag{4.58}$$

Now, note that by (4.53), (4.54), and the Euclideanism of K, we have:

$$K(S_2^2, S_2^1).$$

By this and Lemma 3.5.34, we have:

$$\text{SameHist}(S_2^2, S_2^1). \tag{4.59}$$

Now, to show that P_2 is G-accessible at level N_1 in situation S_2^1, we will need to analyze the SSA for G. A close look at it gives us four cases with four different mutually exclusive conditions: Case 1, where one simply progresses the old set of G-accessible paths before the action A_1 has happened to obtain the new set of G-accessible paths after the occurrence of A_1, Case 2, which involves progression with shifting levels, Case 3, which involves processing/filtering the old set of G-accessible paths to handle the adoption of a goal at level N_1, and Case 4 that involves processing them to handle the dropping of a goal at level N_1. Let us discuss each case, one at a time. Thus the successor-state axiom for G (i.e. Axiom 4.3.3) and (4.55) give us four cases:

- **Case 1.** The action A_1 is a regular (non-adopt/drop action), or A_1 does not refer to the adoption of a goal ϕ at level N_1 or at some higher priority level than N_1, or it does not refer to the dropping of a goal ϕ at N_1, i.e.:

  $$\neg(A_1 = adopt(\phi, M) \wedge M \leq N_1) \wedge \neg(A_1 = drop(\phi) \wedge \text{PGoal}(\phi, N_1, S_1)).$$

  By the SSA for G and (4.55), in all these cases P_2 is the simple progression of some path P_1 that was G-accessible at N_1 in S_1:

  $$\text{Starts}(P_1, S_1^2) \wedge \text{Suffix}(P_2, P_1, do(A_1, S_1^2)) \wedge G(P_1, N_1, S_1). \tag{4.60}$$

Definition 4.4.18, (4.52), (4.57), (4.58), and (4.60) imply that:

$$G(P_1, N_1, S_1^1). \tag{4.61}$$

By Axiom 4.3.3, Definitions 4.3.4, 4.3.5, and 4.3.6, the assumption for this case (i.e. that A_1 is a regular action, or that A_1 refers to the adoption of a goal at a lower priority level than N_1 or to the dropping of a goal at some other level than N_1), (4.60), and (4.61), the progression of P_1 (i.e. P_2) will be retained in the G-relation at N_1 in $S_2^1 = do(A_1, S_1^1)$, since Progressed(P_2, N_1, A_1, S_1^1) holds. Thus we have $G(P_2, N_1, S_2^1)$.

- **Case 2.** A_1 refers to the adoption of a goal ϕ at a higher priority level than N_1, i.e. $A_1 = \mathit{adopt}(\phi, M) \wedge M < N_1$. In this case, P_2 is the progression of some path P_1 that was G-accessible at $N_1 - 1$ in S_1 (since adopting the goal ϕ at higher priority than N_1 has pushed all the goals that has priority lower than $M - 1$ down by one level):

$$\text{Starts}(P_1, S_1^2) \wedge \text{Suffix}(P_2, P_1, do(A_1, S_1^2)) \wedge G(P_1, N_1 - 1, S_1).$$

  The rest of the proof for this case is similar to that of Case 1.

- **Case 3.** A_1 refers to the adoption of a goal ϕ at N_1, i.e. $A_1 = \mathit{adopt}(\phi, N_1)$. Then P_2 is included in the G-relation at N_1 in S_2^1 if it starts with a situation that has the same history as in S_2^1, and if $\phi(P_2)$ holds. (4.56) and (4.59) imply this former. The latter also holds, otherwise P_2 would not have been included in the G-relation at N_1 in S_2 as in (4.55).

- **Case 4.** A_1 refers to the dropping of a goal ϕ at N_1, i.e. $A_1 = drop(\phi)$, where PGoal(ϕ, N_1, S_1). In this case, P_2 is included in the G-relation at N_1 in S_2^1 if it starts with a situation that has the same history as in S_2^1. Again, (4.56) and (4.59) imply this condition.

The theorem thus follows. □

4.4.4 Persistence Properties

The next two properties concern the persistence of these motivational attitudes. First I have a persistence property for achievement realistic p-goals: if an agent has a realistic p-goal that $\diamond\Phi$ in some situation s, then she will retain this realistic p-goal after some action a has been performed in s, provided that she knows in s that Φ has not yet been achieved, and a is not the action of dropping a p-goal.

Proposition 4.4.22 (Persistence of Achievement Realistic Prioritized Goals)**.**

$$\mathcal{D}_{OAgt} \models \text{RPGoal}(\diamond\Phi, n, s) \land \text{Know}(\neg\Phi, s) \land \forall\psi.\ a \neq drop(\psi)$$
$$\supset \exists n'.\ \text{RPGoal}(\diamond\Phi, n', do(a, s)).$$

Proof. Fix Φ_1, N_1, S_1, and A_1. By the antecedent, we have:

$$\text{RPGoal}(\Diamond\Phi_1, N_1, S_1), \tag{4.62}$$

$$\text{Know}(\neg\Phi_1, S_1), \tag{4.63}$$

$$\forall\psi.\ A_1 \neq drop(\psi). \tag{4.64}$$

Now, by Definitions 4.2.4, 4.2.5, 3.5.8, and 3.5.7, the agent has the realistic p-goal that $\Diamond\Phi_1$ at N_1 in S_1 if the following holds:

$$\forall p.\ G(p, N_1, S_1) \land \exists s.\ \text{Starts}(p, s) \land K(s, S_1) \supset \exists s^*.\ \text{OnPath}(p, s^*) \land \Phi_1(s^*). \tag{4.65}$$

Thus $\Diamond\Phi_1$ will persist after A_1 has been performed in S_1 if there is a level n such that $\Diamond\Phi_1$ holds over all G-accessible paths that start with a K-accessible situation (i.e. all G_R-accessible paths) at n in $do(A_1, S_1)$. After the A_1 action has been performed in S_1, every G relation at any level will be updated in accordance with Axiom 4.3.3. By Axiom 4.3.3, there are three cases to consider: (1) A_1 can be a regular action or the adoption of some goal at a lower priority level than N_1, (2) the adoption of some goal at N_1 or at a higher priority level than N_1, or (3) the dropping of some goal. The last case, i.e. an explicit dropping of a goal is ruled out by (4.64). So let us consider the other two cases, one at a time:

1. $\underline{A_1 \text{ is a regular action or the adoption of some goal at lower priority than } N_1}$:

   First, assume that A_1 is a regular action. Then, by Axiom 4.3.3 and Definition

4.3.4, all G-accessible paths at N_1 will be progressed to reflect the fact that A_1 has just happened. Next, fix Ψ_1 and N_2 and assume that $A_1 = adopt(\Psi_1, N_2)$, where $N_2 > N_1$. Then by Axiom 4.3.3 and Definitions 4.3.5 and 4.3.4, the G-accessible paths at N_1 in S_1 will be progressed to reflect the fact that A_1 has just happened. Thus, for both these cases the G-accessible paths at N_1 in S_1 are simply progressed.

Again, there are two possibilities to consider:

(a) A_1 makes $\Diamond\Phi_1$ impossible to achieve, and thus there are no paths that are G_R-accessible at N_1 in $do(A_1, S_1)$. However this does not cause a problem for persistence of $\Diamond\Phi_1$, since in that case, the agent's set of G_R-accessible paths at N_1 will be empty, and by Definition 4.2.5 the agent trivially has the realistic p-goal that $\Diamond\Phi_1$ at N_1 in $do(A_1, S_1)$.

(b) There is a G_R-accessible path at N_1 in $do(A_1, S_1)$, but $\Diamond\Phi_1$ does not hold over this path. Given the fact that the G-accessible paths at level N_1 in situation $do(A_1, S_1)$ can only be obtained by progressing those at N_1 in S_1, this is only possible if there is a K-accessible situation in S_1, say S_1', where there is a path P_1 that starts with S_1', P_1 is G-accessible at N_1 in S_1, the suffix of P_1 that starts with $do(A_1, S_1')$, let's call it P_2, is G_R-accessible at

N_1 in $do(A_1, S_1)$, and $\Diamond\Phi_1$ does not hold over P_2:

$$K(S'_1, S_1) \wedge \text{Starts}(P_1, S'_1) \wedge G(P_1, N_1, S_1) \wedge \text{Suffix}(P_2, P_1, do(A_1, S'_1))$$
$$\wedge\, G_R(P_2, N_1, do(A_1, S_1)) \wedge \neg\Diamond\Phi_1(P_2). \tag{4.66}$$

By (4.65) and (4.66), it follows that $\exists s.\ \text{OnPath}(P_1, s) \wedge \Phi_1(s)$. By this, (4.66), and Definitions 3.5.8 and 3.5.7, it follows that $\Phi_1(S'_1)$. But by (4.66), (4.63), and Definition 3.4.5, we have $\neg\Phi_1(S'_1)$, a contradiction! Thus it follows that $\Diamond\Phi_1$ holds over all G_R-accessible paths at N_1 in $do(A_1, S_1)$.

2. <u>A_1 is the adoption of some goal at priority greater or equal to N_1:</u>

   Fix Ψ_1 and assume that $A_1 = adopt(\Psi_1, N_2)$, where $N_2 \leq N_1$. Then by Axiom 4.3.3 and Definitions 4.3.5 and 4.3.4, the G-accessible paths at N_1 in S_1 are pushed down one level in the hierarchy, and thus the G-accessible paths at N_1+1 in $do(A_1, S_1)$ are the progressed version of those at level N_1 in S_1, progressed to simply reflect the fact that A_1 has just happened. The rest of the proof is very similar to case (1) with the necessary adjustment to reflect that level N_1 in S_1 indeed refers to level $N_1 + 1$ in $do(A_1, S_1)$. Thus, in this case too, the realistic p-goal $\Diamond\Phi_1$ persists, however at the lower priority level $N_1 + 1$.

The proposition thus follows. □

Note that, we do not need to ensure that $\Diamond\Phi$ is consistent with higher priority active p-goals, since the successor-state axiom for G does not automatically drop such incompatible p-goals from the goal hierarchy. Also, as argued above, the level n where $\Diamond\Phi$ is a realistic p-goal may change, e.g. if the action performed is an $adopt$ action with priority higher than or equal to n. Finally, I believe that the dropping of an unrelated p-goal should not affect persistence, and hence it should be possible to strengthen this proposition. However, I leave this for future work.

The above persistence result can be strengthened to show that a goal $\Diamond\Phi$ persists at its original priority level n (i.e. without possibly shifting its priority level) when additionally adopting goals at priority greater or equal to n is disallowed:

Corollary 4.4.23.

$$\begin{aligned}\mathcal{D}_{OAgt} \models (&\text{RPGoal}(\Diamond\Phi, n, s) \land \text{Know}(\neg\Phi, s)\\ &\land \forall\psi.\ a \neq drop(\psi) \land \forall\psi, m.\ \neg(a = adopt(\psi, m) \land m \leq n))\\ &\supset \text{RPGoal}(\Diamond\Phi, n, do(a, s)).\end{aligned}$$

Proof Sketch. Similar to the proof for Proposition 4.4.22, but with the omission of Case 2, which is ruled out by the antecedent that adoption at level n or at some higher priority level is not allowed, i.e. that $\forall\psi, m.\ \neg(a = adopt(\psi, m) \land m \leq n))$. $\square$

For achievement chosen goals I have the following persistence result: if in some situation s, an agent has the only p-goal at some level n that $\Diamond\Phi$ and that the correct history of actions in s has been performed, and if $\Diamond\Phi$ is also a chosen goal in s (and thus she has the primary c-goal in s that $\Diamond\Phi$), then she will retain the c-goal that $\Diamond\Phi$ at level n after some action a has been performed in s, provided that:

- she knows in s that Φ has not yet been achieved,
- that a is not the action of dropping a p-goal,
- that a is not the action of adopting a p-goal at some higher priority level than n or at n,
- and that at level $n-1$, the agent does not have the c-goal that $\neg\Diamond\Phi$ in $do(a, s)$, i.e. $\Diamond\Phi$ remains consistent with higher priority c-goals after a had been performed in s.

Proposition 4.4.24 (Persistence of Achievement Chosen Goals)**.**

$$\begin{aligned}
\mathcal{D}_{OAgt} \models\ & \text{OPGoal}(\Diamond\Phi \land \exists s'.\ \text{Starts}(s') \land \text{SameHist}(s'), n, s) \land \text{CGoal}(\Diamond\Phi, s) \\
& \land \text{Know}(\neg\Phi, s) \land \forall\psi.\ a \neq \mathit{drop}(\psi) \land \forall\psi, m.\ \neg(a = \mathit{adopt}(\psi, m) \land m \leq n) \\
& \land \neg\text{CGoal}(\neg\Diamond\Phi, n-1, do(a, s)) \\
& \supset \text{CGoal}(\Diamond\Phi, n, do(a, s)).
\end{aligned}$$

Proof. Fix Φ_1, N_1, S_1, and A_1. From the antecedent, we have:

$$\text{OPGoal}(\Diamond\Phi_1 \wedge \exists s'.\ \text{Starts}(s') \wedge \text{SameHist}(s'), N_1, S_1), \tag{4.67}$$

$$\text{CGoal}(\Diamond\Phi_1, S_1), \tag{4.68}$$

$$\text{Know}(\neg\Phi_1, S_1), \tag{4.69}$$

$$\forall\psi.\ A_1 \neq \mathit{drop}(\psi), \tag{4.70}$$

$$\forall\psi, m.\ \neg(A_1 = \mathit{adopt}(\psi, m) \wedge m \leq N_1), \tag{4.71}$$

$$\neg\text{CGoal}(\neg\Diamond\Phi_1, N_1 - 1, \mathit{do}(A_1, S_1)). \tag{4.72}$$

From (4.72) and Definition 4.2.8, there is a path P_1 such that:

$$G_\cap(P_1, N_1 - 1, \mathit{do}(A_1, S_1)) \wedge \Diamond\Phi_1(P_1). \tag{4.73}$$

By (4.73) and Axiom 4.2.7, to prove that $\Diamond\Phi_1$ is a c-goal at N_1 in $\mathit{do}(A_1, S_1)$, it suffices to show that the agent has the realistic p-goal at N_1 in $\mathit{do}(A_1, S_1)$ that $\Diamond\Phi_1$, and that P_1 is indeed G_R-accessible at N_1 in $\mathit{do}(A_1, S_1)$. The latter along with Axiom 4.2.7 ensures that there is at least one path, namely P_1, that is in $G_\cap$ at N_1 in $\mathit{do}(A_1, S_1)$ and over which $\Diamond\Phi_1$ holds, while the former along with Axiom 4.2.7 and the latter stipulates that $\Diamond\Phi_1$ indeed holds over all such paths (i.e. all paths that are in $G_\cap$ at N_1 in $\mathit{do}(A_1, S_1)$).[33]

[33]Note that proving the former condition (i.e. that RPGoal($\Diamond\Phi_1, N_1, \mathit{do}(A_1, S_1)$)) alone is not sufficient since it is possible that N_1 will not be selected as an active level in $\mathit{do}(A_1, S_1)$, e.g. due to the existence of some goal at N_1 in $\mathit{do}(A_1, S_1)$ that is inconsistent with other higher priority goals in $\mathit{do}(A_1, S_1)$.

First I will show that the agent has the realistic p-goal at N_1 in $do(A_1, S_1)$ that $\diamond\Phi_1$. From (4.67) and Definitions 4.2.2 and 4.2.1, we have PGoal$(\diamond\Phi_1, N_1, S_1)$. From this and Definitions 4.2.1, 4.2.4, and 4.2.5, we have:

$$\text{RPGoal}(\diamond\Phi_1, N_1, S_1). \tag{4.74}$$

From (4.74), (4.69), (4.70), (4.71), and Corollary 4.4.23, it follows that $\diamond\Phi_1$ persists at level N_1 after A_1 has been performed in S_1, i.e. RPGoal$(\diamond\Phi_1, N_1, do(A_1, S_1))$.

Next I will show that P_1 is indeed G_R-accessible at N_1 in $do(A_1, S_1)$. Consider an arbitrary path P' such that it starts with a situation S' that is K-accessible from $do(A_1, S_1)$ and such that $\diamond\Phi_1$ holds over P':

$$\diamond\Phi_1(P') \wedge \text{Starts}(P', S') \wedge K(S', do(A_1, S_1)). \tag{4.75}$$

By Axiom 3.2.7, S' must be of the form $do(A_1, S'')$ for some situation S''. Moreover, by Axiom 3.4.10, S'' must have been K-accessible in S_1:

$$K(S'', S_1). \tag{4.76}$$

Let us extend P' in the past to include S'', and let us call the path obtained by doing so, P''. Since P' is a suffix of P'', it follows from (4.75) and Definitions 3.5.8 and 3.5.7 that:

$$\diamond\Phi_1(P''). \tag{4.77}$$

From (4.76) and Lemma 3.5.34, it follows that S'' has the same action history as S_1, i.e. SameHist(S'', S_1). From this and by (4.77), (4.67), and Definition 4.2.2, P'' must be G-accessible at N_1 in S_1, i.e. $G(P'', N_1, S_1)$. From this, (4.76), the fact that P'' starts with S'', and Definition 4.2.4, P'' must be G_R-accessible at N_1 in S_1, i.e. $G_R(P'', N_1, S_1)$. Moreover, by (4.70), (4.71), Axiom 4.3.3, and Definitions 4.3.4 and 4.3.5, since all G_R-accessible paths at N_1 in S_1 are simply progressed when A_1 is performed in S_1, it follows that P' is G_R-accessible at N_1 in $do(A_1, S_1)$. Thus we have:

$$\forall p.\ \Diamond\Phi_1(p) \wedge \exists s.\ \text{Starts}(p, s) \wedge K(s, do(A_1, S_1)) \supset G_R(p, N_1, do(A_1, S_1)). \quad (4.78)$$

By (4.73), Proposition 3.5.37(a), and Lemma 4.4.10, we have:

$$\Diamond\Phi_1(P_1) \wedge \exists s.\ \text{Starts}(P_1, s) \wedge K(s, do(A_1, S_1)).$$

Finally, by this and (4.78), we have $G_R(P_1, N_1, do(A_1, S_1))$. □

Note that, this property also follows if we replace the consequent with CGoal$(\Diamond\Phi, do(a, s))$, or PrimCGoal$(\Diamond\Phi, do(a, s))$, and thus it deals with the persistence of (primary) c-goals. Note however that, it does not hold if we replace the OPGoal in the antecedent with PGoal; the reason for this is that the agent might have a p-goal at level n in s that $\Diamond\Phi$ and the c-goal in s that $\Diamond\Phi$, but not have $\Diamond\Phi$ as a primary c-goal in s, e.g. n might be an inactive level because another p-goal at n has become impossible, and $\Diamond\Phi$ could be a c-goal in s because it is a consequence of two other primary c-goals. Thus even if $\neg\Diamond\phi$ is not a c-goal after a has been performed in s, there is no guarantee that level n will be active in $do(a, s)$ or that all the active p-goals that contributed to $\Diamond\Phi$ in s are still active. As in Proposition 4.4.22, I believe that the dropping of an unrelated p-goal will not affect persistence, and hence it should be possible to strengthen this proposition. Finally, in the future I would like to generalize these two persistence properties to deal with arbitrary temporally extended goals.

4.5 An Example

In this section, I demonstrate the utility of this framework using an application to personalized travel planning over the web. Consider the following scenario: Anika, who lives in Toronto, Canada (YYZ), would like to plan a trip for her holidays. She would like to depart on July 29, returning on the 5th of August. As for the destination, she'd really like to go to Kaafu in the Maldives (MLE), but only if she can redeem her Cool-Air-Miles reward miles for this. Otherwise, she would settle for the Florida Keys (EYW) and save up for next year's vacation. Moreover, if for some reason Florida does not work out, she would like to revisit Varadero, Cuba (VRA) instead. Finally, if she ends up going to the Keys, she would like to visit her best friend who lives there.

The following set of axioms specifies her travel agent's goals initially:[34]

Axiom 4.5.1.

$$
\begin{aligned}
&\text{Init}(s) \supset \\
&\quad ((G(p,0,s) \equiv \exists s'.\ \text{Starts}(p,s') \wedge \text{Init}(s') \wedge \\
&\qquad\qquad \text{During}(\text{At}(\text{Anika}, \text{MLE}), \text{Jul29}, \text{Aug05})(p) \wedge \\
&\qquad\qquad (\text{Redeemed}(\text{Anika}, \text{YYZ}, \text{MLE})\ \ \mathcal{B}\ \ (\text{date} = \text{Jul29}))(p)) \\
&\quad \wedge (G(p,1,s) \equiv \exists s'.\ \text{Starts}(p,s') \wedge \text{Init}(s') \wedge \\
&\qquad\qquad \text{During}(\text{At}(\text{Anika}, \text{EYW}), \text{Jul29}, \text{Aug05})(p)) \\
&\quad \wedge (G(p,2,s) \equiv \exists s'.\ \text{Starts}(p,s') \wedge \text{Init}(s') \wedge \\
&\qquad\qquad \text{During}(\text{At}(\text{Anika}, \text{VRA}), \text{Jul29}, \text{Aug05})(p)) \\
&\quad \wedge (G(p,3,s) \equiv \exists s'.\ \text{Starts}(p,s') \wedge \text{Init}(s') \wedge \\
&\qquad\qquad \text{Between}(\text{AnikaVisitedBFF}, \text{Jul29}, \text{Aug05})(p)).
\end{aligned}
$$

[34]I focus on the goals of the travel planning agent, which are of course derived from those of the agent's client, Anika. I don't model how Anika communicates her preferences to the agent. To simplify, I suppress the agent arguments in the K and the G-relations.

Definition 4.5.2.

$$\text{During}(\Phi, from, to)(p) \overset{\text{def}}{=}$$

$$\exists s_1, s_2.\ \text{OnPath}(p, s_1) \wedge \text{OnPath}(p, s_2) \wedge \text{date}(s_1) = from \wedge \text{date}(s_2) = to$$

$$\wedge\ \forall s.\ \text{OnPath}(p, s) \wedge \text{date}(s) > from \wedge \text{date}(s) < to \supset \Phi(s).$$

Definition 4.5.3.

$$\text{Between}(\Phi, from, to)(p) \overset{\text{def}}{=}$$

$$\exists s.\ \text{OnPath}(p, s) \wedge \text{date}(s) > from \wedge \text{date}(s) < to \wedge \Phi(s).$$

Thus, initially the agent's set of G-accessible paths at the highest priority level comprise those that start with initial situations and over which Anika is located at Kaafu during the week of July 29 to August 5, and she pays for her flight by redeeming her reward miles for this before the departure date. Similarly, initially her G-accessible paths at level 1 (the second highest priority level) are the ones that start with initial situations and over which Anika is located at the Keys during the specified week, and similarly for level 2, but this time at Varadero. Also, initially her G-accessible paths at level 3 are those that start with initial situations and over which Anika visits her best friend between the specified dates.

Finally, for any level that has lower priority than 3, the agent has the trivial goal that she be on a path that starts with an initial situation:

Axiom 4.5.4.

$$\text{Init}(s) \land n \geq 4 \supset (G(p, n, s) \equiv \exists s'.\ \text{Starts}(p, s') \land \text{Init}(s')).$$

Note that, here I model time using a date(s) functional fluent along the lines of Reiter's account [178]. The only action that affects the date(s) fluent is $dateTick$, which increments the date (see Axioms 4.5.6 and 4.5.15 below). For this, I use the date constants Jan01 to Dec31 assuming that the year is 2017 (i.e. not a leap year).

Next, I list the actions that can be performed in this domain, along with their preconditions. An agent agt can fly from location x to location y in some situation s, if x and y refer to two different locations, agt has a ticket from x to y for date d in s, the date of s is indeed d, the flights between these two locations are running in s, and if agt is located at x in s:

Axiom 4.5.5.

$$\begin{aligned} \text{Poss}(fly(agt, x, y), s) \equiv\ & x \neq y \land \exists d.\ \text{HasTicket}(agt, x, y, d, s) \land \text{date}(s) = d \\ & \land \neg\exists d'.\ \text{FlightsCancelled}(x, y, d', s) \land \text{At}(agt, x, s). \end{aligned}$$

There are five exogenous actions in this domain: $dateTick$ increments the calendar date, $addBlacklistLoc(x, d)$ marks location x as blacklisted at least until date d so that the customers of Cool-Air-Miles can no longer redeem their reward miles for an air ticket to x at least until d, $removeBlacklistLoc(x)$ removes x from the blacklisted locations, $startDisruptionBtwn(x, y, d)$ starts some sort of disruption between locations x and y at least until date d, cancelling all flights over this route at least until d, and $endDisruptionBtwn(x, y)$ ends this disruption, restoring the cancelled flights. I assume that $dateTick$ is always possible:

Axiom 4.5.6.

$$\text{Poss}(dateTick, s) \equiv \text{True}.$$

A location x can be added to the blacklist at least until date d in situation s if and only if x is not already in the blacklist until some date d' and if date d is in the future, i.e. the date of s is less than d (note that x is not automatically removed from the blacklist even when the date turns d; rather it must be removed explicitly using a $removeBlacklistLoc$ action):

Axiom 4.5.7.

$$\text{Poss}(addBlacklistLoc(x, d), s) \equiv \neg\exists d'.\ \text{Blacklisted}(x, d', s) \wedge \text{date}(s) < d.$$

A location x can be removed from the blacklist in s if and only if x is currently blacklisted in s and if the expiry date d of the last time x was blacklisted has past, i.e. the date of s is greater than d:

Axiom 4.5.8.

$$\text{Poss}(removeBlacklistLoc(x), s) \equiv \exists d.\ \text{Blacklisted}(x, d, s) \wedge d < \text{date}(s).$$

Similarly, the flights between two locations x and y can be cancelled by causing a disruption between these two locations at least until date d in situation s if and only if the flights between x and y are not already cancelled until some date d' in s and if date d is in the future, i.e. the date of s is less than d (note again that the flights between these two locations are not automatically restored even when the date turns d; rather they must be restored explicitly using an $endDisruptionBtwn$ action):

Axiom 4.5.9.

$$\text{Poss}(startDisruptionBtwn(x, y, d), s) \equiv$$
$$\neg\exists d'.\ \text{FlightsCancelled}(x, y, d', s) \wedge \text{date}(s) < d.$$

Finally, the disruption between two locations x and y can be removed in situation s if and only if the flights between these two locations are currently cancelled in s and if the expiry date d of the disruption between these locations has past, i.e. the date of s is greater than d:

Axiom 4.5.10.

$$\text{Poss}(endDisruptionBtwn(x, y), s) \equiv \exists d.\ \text{FlightsCancelled}(x, y, d, s) \wedge d < \text{date}(s).$$

An agent agt can always purchase a ticket between two locations x and y for any date d (for simplicity, I ignore the monetary aspect, the availability of tickets, etc.):

Axiom 4.5.11.

$$\text{Poss}(purchase(agt, x, y, d), s) \equiv \text{True}.$$

Moreover, an agent can also obtain a ticket by redeeming her Cool-Air-Miles reward miles; in particular, she can redeem her reward miles for an air ticket from location x to location y on date d in situation s if she did not already redeem her Cool-Air-Miles reward miles for this route in s, y is not blacklisted in s for some date d', and if in s she has collected at least as many reward miles as is required to travel from x to y:[35]

Axiom 4.5.12.

$$\text{Poss}(redeemMiles(agt, x, y, d), s) \equiv \neg\text{Redeemed}(agt, x, y, s)$$
$$\wedge \neg\exists d'.\ \text{Blacklisted}(y, d', s) \wedge \text{milesBtwn}(x, y) \leq \text{hasMiles}(agt, s).$$

Finally, Anika can visit her best friend if she is at the keys:

Axiom 4.5.13.

$$\text{Poss}(anikaVisitsBFF, s) \equiv \text{At}(\text{Anika}, \text{EYW}, s).$$

[35]While I could have made this axiom more realistic (e.g. as I did for the preconditions of $removeBlacklistLoc$ and $endDisruptionBtwn$ actions), this does not add much and would have made the axiom/framework unnecessarily complex. For the purpose of this example, such a simplistic model of redemption suffices since, e.g., I assume that initially Anika's travel agent knows that Anika did not redeem her miles for the relevant route/trip, that multiple redemption attempts for the same route will not be considered, etc.

In the following, I specify the fluents in this domain and their successor-state axioms. An agent agt is located at x after a has been performed in situation s if and only if a is the action of agt flying from some location y to x, or if agt was at x in s and a does not refer to the action of agt flying to another location y:

Axiom 4.5.14.

$$\begin{aligned} \text{At}(agt, x, do(a, s)) \equiv{} & \exists y.\ a = fly(agt, y, x) \\ & \vee (\text{At}(agt, x, s) \wedge \neg\exists y.\ a = fly(agt, x, y)). \end{aligned}$$

The current date is x after a has been performed in s if and only if the date was y in s, a is a $dateTick$ action, and x is the successor date to y, or the date was x in s and a does not refer to a $dateTick$ action (I assume that function nextDate(a), which takes a date a and returns the successor date to a, is available):

Axiom 4.5.15.

$$\begin{aligned} \text{date}(do(a, s)) = x \equiv{} & (\exists y.\ \text{date}(s) = y \wedge a = dateTick \wedge \text{nextDate}(y) = x) \\ & \vee (\text{date}(s) = x \wedge \neg a = dateTick). \end{aligned}$$

A location x is blacklisted at least until some date d in $do(a, s)$ if and only if a is the action of adding x to the blacklist until date d or if x was already blacklisted until d in s and a does not refer to the action of removing x from the blacklist:

Axiom 4.5.16.

$$\text{Blacklisted}(x, d, do(a, s)) \equiv a = addBlacklistLoc(x, d)$$
$$\vee\ (\text{Blacklisted}(x, d, s) \wedge \neg a = removeBlacklistLoc(x)).$$

The flights between location x any y are cancelled at least until some date d after a has been performed in s if and only if a is the action of starting a disruption between x and y until date d or if the flights were already cancelled until d in s and a is not the action of ending the disruption between x and y:

Axiom 4.5.17.

$$\text{FlightsCancelled}(x, y, d, do(a, s)) \equiv a = startDisruptionBtwn(x, y, d)$$
$$\vee\ (\text{FlightsCancelled}(x, y, d, s) \wedge \neg a = endDisruptionBtwn(x, y)).$$

An agent agt has redeemed her reward miles for a flight from location x to y after action a has been performed in s if and only if a refers to the action of redeeming them towards a ticket from x to y on some date d or if agt has already redeemed her reward miles for this route in s:

Axiom 4.5.18.

$$\text{Redeemed}(agt, x, y, do(a, s)) \equiv$$
$$\exists d.\ a = redeemMiles(agt, x, y, d) \vee \text{Redeemed}(agt, x, y, s).$$

An agent agt has a ticket for a flight from location x to y for date d after action a has been performed in s if and only if a refers to agt's action of purchasing a ticket from x to y for d in s or to that of redeeming agt's reward miles towards a ticket from x to y for d or if she already had this ticket for this route for d in s:[36]

Axiom 4.5.19.

$$
\begin{aligned}
\text{HasTicket}(agt, x, y, d, do(a, s)) \equiv \\
&(a = purchase(agt, x, y, d) \vee a = redeemMiles(agt, x, y, d)) \\
&\vee \text{HasTicket}(agt, x, y, d, s).
\end{aligned}
$$

An agent agt has n reward miles after a has been performed in s if and only if a is the action of redeeming agt's reward miles for a ticket from location x to y for date d and n is what is left of her reward miles after she redeems the required amount of miles for the ticket, i.e. the difference between the miles she has in s and the distance between x and y, $\text{hasMiles}(agt, s) - \text{milesBtwn}(x, y)$, or if she has n reward miles in s and a is not the action of redeeming them for some ticket:

[36]Again, this is a simplistic model –e.g. here tickets don't get cancelled after they have been used up– but it is adequate for the current example.

Axiom 4.5.20.

$$\text{hasMiles}(agt, do(a, s)) = n \equiv$$

$$(\exists x, y, d.\ a = redeemMiles(agt, x, y, d) \wedge n = \text{hasMiles}(agt, s) - \text{milesBtwn}(x, y))$$

$$\vee\ (\text{hasMiles}(agt, s) = n \wedge \neg\exists x, y, d.\ a = redeemMiles(agt, x, y, d)).$$

Finally, Anika has visited her best friend after an action a has been performed in situation s if a is the action $anikaVisitsBFF$ or if she has already visited her in s:[37]

Axiom 4.5.21.

$$\text{AnikaVisitedBFF}(do(a, s)) \equiv a = anikaVisitsBFF \vee \text{AnikaVisitedBFF}(s).$$

I also need axioms for specifying the nextDate function. For brevity, I just give one example here:

$$\text{nextDate}(\text{Jan01}) = \text{Jan02}.$$

I will call these axioms nextDate axioms.

For these axioms to work, I need to specify unique name axioms for the fly, *date-*$Tick$, *addBlacklistLoc*, *removeBlacklistLoc*, *startDisruptionBtwn*, *endDisruptionBtwn*, $purchase$, $redeemMiles$, and $anikaVisitsBFF$ actions. To this end, I first need an axiom that captures that actions with different action-function names are not the same:

[37]Once again, this simplified model is sufficient for my example.

Axiom 4.5.22. *For all distinct action functions A_1 and A_2 :*

$$\forall \vec{x}, \vec{y}.\ A_1(\vec{x}) \neq A_2(\vec{y}).$$

Thus, for instance, we have that $\forall a, b, c.\ fly(a, b, c) \neq dateTick$, that $\forall a, b, c, d, e.$ $fly(a, b, c) \neq addBlacklistLoc(d, e)$, etc.

Another set of unique names axioms states that two actions with the same name are the same if their arguments are equal:

Axiom 4.5.23. *For all action functions A :*

$$A(x_1, \ldots, x_n) = A(y_1, \ldots, y_n) \supset x_1 = y_1 \wedge \ldots \wedge x_n = y_n.$$

Thus, for example, it follows from this axiom that $fly(a, b, c) = fly(x, y, z) \supset a = x \wedge b = y \wedge c = z$, etc.

Again, I also need unique names axioms for location names:

Axiom 4.5.24.

$$\text{YYZ} \neq \text{MLE} \wedge \text{YYZ} \neq \text{EYW} \wedge \text{YYZ} \neq \text{VRA}$$
$$\wedge\, \text{MLE} \neq \text{EYW} \wedge \text{MLE} \neq \text{VRA} \wedge \text{EYW} \neq \text{VRA}.$$

Furthermore, I need unique names axioms for date constants:

Axiom 4.5.25. *For all distinct date constants d_1 and d_2:*

$$d_1 \neq d_2.$$

Thus, for example, it follows from this axiom that Jul29 $\neq$ Jul30, etc.

Finally, the following initial state axioms specify what the world is like and what the agent knows about the world initially:

Axiom 4.5.26.

(a) $\mathrm{milesBtwn}(\mathrm{YYZ}, \mathrm{MLE}) = 8600$,

(b) $\mathrm{Know}(\mathrm{At}(\mathrm{Anika}, \mathrm{YYZ}), S_0)$,

(c) $\forall loc.\ loc \neq \mathrm{YYZ} \supset \mathrm{Know}(\neg\mathrm{At}(\mathrm{Anika}, loc), S_0)$,

(d) $\mathrm{Know}(\mathrm{date} = \mathrm{Jul26}, S_0)$,

(e) $\forall d.\ \mathrm{Know}(\neg\mathrm{Blacklisted}(\mathrm{MLE}, d), S_0)$,

(f) $\forall loc, d.\ \mathrm{Know}(\neg\mathrm{FlightsCancelled}(\mathrm{YYZ}, loc, d), S_0)$,

(g) $\mathrm{Know}(\neg\mathrm{Redeemed}(\mathrm{Anika}, \mathrm{YYZ}, \mathrm{MLE}), S_0)$,

(h) $\forall x, y, d.\ \mathrm{Know}(\neg\mathrm{HasTicket}(\mathrm{Anika}, x, y, d), S_0)$,

(i) $\mathrm{Know}(\mathrm{hasMiles}(\mathrm{Anika}) = 9000, S_0)$,

(j) $\mathrm{Know}(\neg\mathrm{AnikaVisitedBFF}, S_0)$.

Thus, the distance between Toronto (YYZ) and Kaafu (MLE) is 8600 miles. Moreover, initially the travel agent knows that Anika is only located at Toronto and not elsewhere, that the current date is July 26, that Kaafu is not blacklisted, that all flights from Toronto are operating without any issues, that Anika has not redeemed her reward miles for a ticket from Toronto to Kaafu, that initially Anika does not have any tickets, that initially Anika has 9000 reward miles, and that initially Anika has not visited her best friend.

Henceforth, I use $\mathcal{D}_{TA}$ to denote the set of axioms and definitions required for formalizing our travel agent example, i.e. one that consists of the axioms and the definitions for modeling an optimizing agent $\mathcal{D}_{OAgt}$ and Axioms 4.5.1 – 4.5.26 and the associated definitions. Given this, it can be shown that Anika can only be at one location in any given situation, i.e. that the At relation is functional:

Lemma 4.5.27.

$$\mathcal{D}_{TA} \models \forall loc_1, loc_2, s.\ \text{At}(\text{Anika}, loc_1, s) \wedge \text{At}(\text{Anika}, loc_2, s) \supset loc_1 = loc_2.$$

Proof Sketch. By induction on s using Axioms 4.5.14, 4.5.26 (b) and (c), and the relevant unique names axioms (i.e. 4.5.22, 4.5.23, and 4.5.24). □

Moreover, this result can also be extended for a given date interval:

Lemma 4.5.28.

$$\begin{aligned}\mathcal{D}_{TA} \models \forall loc_1, loc_2, p.\ &\text{During}(\text{At}(\text{Anika}, loc_1), \text{Jul29}, \text{Aug05})(p)\\ &\wedge \text{During}(\text{At}(\text{Anika}, loc_2), \text{Jul29}, \text{Aug05})(p)\\ &\supset loc_1 = loc_2.\end{aligned}$$

Proof. By contradiction. Fix path P_1 and locations L_1 and L_2 and assume:

$$\text{During}(\text{At}(\text{Anika}, L_1), \text{Jul29}, \text{Aug05})(P_1), \tag{4.79}$$

$$\text{During}(\text{At}(\text{Anika}, L_2), \text{Jul29}, \text{Aug05})(P_1), \tag{4.80}$$

$$L_1 \neq L_2. \tag{4.81}$$

By (4.79) and Definition 4.5.2, it follows that there are situations S_1 and S_2 such that:

$$\text{OnPath}(P_1, S_1) \wedge \text{date}(S_1) = \text{Jul29}, \tag{4.82}$$

$$\text{OnPath}(P_1, S_2) \wedge \text{date}(S_2) = \text{Aug05}, \tag{4.83}$$

$$\forall s.\ \text{OnPath}(P_1, s) \wedge \text{date}(s) > \text{Jul29} \wedge \text{date}(s) < \text{Aug05} \supset \text{At}(\text{Anika}, L_1, s). \tag{4.84}$$

Again, by (4.80) and Definition 4.5.2, we have:

$$\forall s.\ \text{OnPath}(P_1, s) \wedge \text{date}(s) > \text{Jul29} \wedge \text{date}(s) < \text{Aug05} \supset \text{At}(\text{Anika}, L_2, s). \tag{4.85}$$

Fix such a situation S^* on P_1 so that the date of S^* is later than Jul29 and earlier than Aug05, say $\text{date}(S^*) = \text{Jul31}$. By (4.82), (4.83), Proposition 3.5.43, Axiom 4.5.15, and nextDate axioms (and the associated unique names axioms), such a situation indeed exists. Thus from this and from (4.84) and (4.85), we have:

$$\text{At}(\text{Anika}, L_1, S^*) \wedge \text{At}(\text{Anika}, L_2, S^*).$$

Then by this and Lemma 4.5.27, we have: $L_1 = L_2$. But this is contradictory to (4.81). □

I can show that initially our agent has the following only p-goals at the various levels:

Proposition 4.5.29.

$$\begin{aligned}\mathcal{D}_{TA} \models\ & \text{OPGoal}(\exists s.\ \text{Starts}(s) \land \text{Init}(s) \land \text{During}(\text{At}(\text{Anika}, \text{MLE}), \text{Jul29}, \text{Aug05}) \\ & \qquad \land (\text{Redeemed}(\text{Anika}, \text{YYZ}, \text{MLE})\ \ \mathcal{B}\ \ (\text{date} = \text{Jul29})), 0, S_0) \\ & \land \text{OPGoal}(\exists s.\ \text{Starts}(s) \land \text{Init}(s) \land \text{During}(\text{At}(\text{Anika}, \text{EYW}), \text{Jul29}, \text{Aug05}), 1, S_0) \\ & \land \text{OPGoal}(\exists s.\ \text{Starts}(s) \land \text{Init}(s) \land \text{During}(\text{At}(\text{Anika}, \text{VRA}), \text{Jul29}, \text{Aug05}), 2, S_0) \\ & \land \text{OPGoal}(\exists s.\ \text{Starts}(s) \land \text{Init}(s) \land \text{Between}(\text{AnikaVisitedBFF}, \text{Jul29}, \text{Aug05}), 3, S_0).\end{aligned}$$

Proof. First, note that from Definitions 4.2.1 and 4.2.2, it follows that to show that OPGoal(ϕ_n, n, S_0) for some formula ϕ_n and some level n, we need to show that $\forall p.\ G(p, n, S_0) \equiv \phi_n(p)$. The proposition follows from this and Axiom 4.5.1, since by Axiom 3.2.2, S_0 is an initial situation, i.e. Init(S_0). $\square$

Also, all of these goals are initially possible:

Proposition 4.5.30.

$$\begin{aligned}\mathcal{D}_{TA} \models\ & \mathrm{RPGoal}(\mathrm{During}(\mathrm{At}(\mathrm{Anika}, \mathrm{MLE}), \mathrm{Jul29}, \mathrm{Aug05}) \\ & \qquad \wedge (\mathrm{Redeemed}(\mathrm{Anika}, \mathrm{YYZ}, \mathrm{MLE})\ \mathcal{B}\ (\mathrm{date} = \mathrm{Jul29})), 0, S_0) \\ & \wedge \mathrm{RPGoal}(\mathrm{During}(\mathrm{At}(\mathrm{Anika}, \mathrm{EYW}), \mathrm{Jul29}, \mathrm{Aug05}), 1, S_0) \\ & \wedge \mathrm{RPGoal}(\mathrm{During}(\mathrm{At}(\mathrm{Anika}, \mathrm{VRA}), \mathrm{Jul29}, \mathrm{Aug05}), 2, S_0) \\ & \wedge \mathrm{RPGoal}(\mathrm{Between}(\mathrm{AnikaVisitedBFF}, \mathrm{Jul29}, \mathrm{Aug05}), 3, S_0).\end{aligned}$$

Proof. I will show this for level 1 only, as the proofs for the other levels are similar. Since by Axiom 3.2.2, S_0 is an initial situation, it follows from Axiom 4.5.1 that:

$$\forall p.\ G(p, 1, S_0) \supset \mathrm{During}(\mathrm{At}(\mathrm{Anika}, \mathrm{EYW}), \mathrm{Jul29}, \mathrm{Aug05})(p). \tag{4.86}$$

We now have the two following cases. First consider the case where there is a path that is G_R-accessible at 1 in S_0. By Definition 4.2.4, we have that all G_R-accessible paths at level 1 in situation S_0 are G-accessible at 1 in S_0, i.e. $\forall p.\ G_R(p, 1, S_0) \supset G(p, 1, S_0)$. From this and (4.86), it follows that:

$$\forall p.\ G_R(p, 1, S_0) \supset \mathrm{During}(\mathrm{At}(\mathrm{Anika}, \mathrm{EYW}), \mathrm{Jul29}, \mathrm{Aug05})(p).$$

For this case, the proposition thus follows from this and Definition 4.2.5.

Next consider the case where there are no G_R-accessible paths at level 1 in S_0.[38] But then the proposition trivially follows from Definition 4.2.5. $\square$

[38]In the proof of Proposition 4.5.31, I show that there is in fact a path in G_R at level 0 in S_0.

However, initially the agent only has the c-goal that Anika be at Kaafu during the week of July 29 to August 5 and that she uses her reward miles before the departure date to buy the ticket for this trip:

Proposition 4.5.31.

$$\begin{aligned}\mathcal{D}_{TA} \models\ & \mathrm{CGoal}(\mathrm{During}(\mathrm{At}(\mathrm{Anika}, \mathrm{MLE}), \mathrm{Jul29}, \mathrm{Aug05}) \\ & \qquad\qquad \wedge (\mathrm{Redeemed}(\mathrm{Anika}, \mathrm{YYZ}, \mathrm{MLE})\ \ \mathcal{B}\ \ (\mathrm{date} = \mathrm{Jul29})), S_0) \\ & \wedge \neg\mathrm{CGoal}(\mathrm{During}(\mathrm{At}(\mathrm{Anika}, \mathrm{EYW}), \mathrm{Jul29}, \mathrm{Aug05}), S_0) \\ & \wedge \neg\mathrm{CGoal}(\mathrm{During}(\mathrm{At}(\mathrm{Anika}, \mathrm{VRA}), \mathrm{Jul29}, \mathrm{Aug05}), S_0) \\ & \wedge \neg\mathrm{CGoal}(\mathrm{Between}(\mathrm{AnikaVisitedBFF}, \mathrm{Jul29}, \mathrm{Aug05}), S_0).\end{aligned}$$

Proof. This can be shown by proving each conjunct, one at a time. Let's start with the first conjunct. In the following, I will use ϕ_0 to denote the c-goal of this conjunct, i.e. $\phi_0 = \mathrm{During}(\mathrm{At}(\mathrm{Anika}, \mathrm{MLE}), \mathrm{Jul29}, \mathrm{Aug05}) \wedge (\mathrm{Redeemed}(\mathrm{Anika}, \mathrm{YYZ}, \mathrm{MLE})$ $\mathcal{B}\ \ (\mathrm{date} = \mathrm{Jul29}))$. I will first show that there is a G_R-accessible path at level 0 in situation S_0 over which ϕ_0 holds. To this end, let me construct such a path P_0 by giving the situations/actions on the path: S_0, $S_1 = do(redeemMiles(\mathrm{Anika}, \mathrm{YYZ}, \mathrm{MLE}, \mathrm{Jul29}),$ $S_0)$, $S_2 = do(dateTick, S_1)$, $S_3 = do(dateTick, S_2)$, $S_4 = do(dateTick, S_3)$, $S_5 =$ $do(fly(\mathrm{Anika}, \mathrm{YYZ}, \mathrm{MLE}), S_4)$, followed by infinitely many $dateTick$ actions. Note that, for this sequence of situations to be a valid path P_0, these situations need to be executable. By Axiom 3.2.2 and Lemma 3.5.17, S_0 is executable; I will show that the rest of the situations are also executable (by showing that these actions are executable in the corresponding situations) later.

Also, note that P_0 starts with S_0, and by Axioms 3.2.2 and 3.4.2, S_0 is an initial situation that is K-accessible from S_0, i.e. $\text{Init}(S_0) \wedge K(S_0, S_0)$. From this, Axiom 4.5.1, and Definition 4.2.4, it follows that P_0 is G_R-accessible at level 0 in S_0 (i.e. $G_R(P_0, 0, S_0)$) if ϕ_0 holds over P_0. I will now argue that this is indeed that case. First note that by Axiom 4.5.12, the $redeemMiles(\text{Anika, YYZ, MLE, Jul29})$ action is possible in S_0 if and only if we have:

$$\neg\text{Redeemed}(\text{Anika}, \text{YYZ}, \text{MLE}, S_0) \wedge$$
$$\neg\exists d.\ \text{Blacklisted}(\text{MLE}, d, S_0) \wedge \text{milesBtwn}(\text{YYZ}, \text{MLE}) \leq \text{hasMiles}(\text{Anika}, S_0).$$

$\neg\text{Redeemed}(\text{Anika}, \text{YYZ}, \text{MLE}, S_0)$ follows from Axiom 4.5.26 (g) and the reflexivity of K. Moreover, $\neg\exists d.\ \text{Blacklisted}(\text{MLE}, d, S_0)$ follows from Axiom 4.5.26 (e) and the reflexivity of K while $\text{milesBtwn}(\text{YYZ}, \text{MLE}) \leq \text{hasMiles}(\text{Anika}, S_0)$ from Axioms 4.5.26 (a), (i), and the reflexivity of K. Thus the $redeemMiles$ action is possible in S_0. Note that after this $redeemMiles$ action has been performed, i.e. in S_1, it follows from Axiom 4.5.19 that Anika has a ticket for a flight from Toronto to the Maldives for July 29, i.e.:

$$\text{HasTicket}(\text{Anika}, \text{YYZ}, \text{MLE}, \text{Jul29}, S_1). \tag{4.87}$$

Also, it follows from Axiom 4.5.18 that she has redeemed her reward miles for this ticket in S_1, i.e.:

$$\text{Redeemed}(\text{Anika}, \text{YYZ}, \text{MLE}, S_1). \tag{4.88}$$

Moreover, it follows from Axiom 4.5.26 (d), the reflexivity of K, and Axiom 4.5.15 (and the unique names for actions axioms[39]) that the current date was not effected by this action:

$$\text{date}(S_1) = \text{Jul26}. \tag{4.89}$$

Furthermore, it follows from Axiom 4.5.26 (b), the reflexivity of K, and Axiom 4.5.14 that Anika's location was not effected by this action:

$$\text{At}(\text{Anika}, \text{YYZ}, S_1). \tag{4.90}$$

Finally, it follows from Axiom 4.5.26 (f), the reflexivity of K, and Axiom 4.5.17 that the flights between Toronto and the Maldives were not effected by this action either:

$$\forall d.\ \neg\text{FlightsCancelled}(\text{YYZ}, \text{MLE}, d, S_1). \tag{4.91}$$

Note that, from (4.88), (4.89), and Definition 3.5.11, it follows that:

$$(\text{Redeemed}(\text{Anika}, \text{YYZ}, \text{MLE})\ \ \mathcal{B}\ \ (\text{date} = \text{Jul29}))(P_0). \tag{4.92}$$

[39]For brevity, I won't mention unique names for actions axioms henceforth.

Secondly, note that by Axiom 4.5.6, the $dateTick$ action is possible in situations S_1, S_2, and S_3. Now, by (4.89) and Axiom 4.5.15, the current date is incremented after each $dateTick$ action happens, i.e. $\text{date}(S_2) = \text{Jul27}$ and:

$$\text{date}(S_3) = \text{Jul28}, \tag{4.93}$$

$$\text{date}(S_4) = \text{Jul29}. \tag{4.94}$$

Also, by (4.87) and Axiom 4.5.19, (4.91) and Axiom 4.5.17, and (4.90) and Axiom 4.5.14, these $dateTick$ actions have no effect on Anika's tickets, flight cancellations, and Anika's location:

$$\text{HasTicket}(\text{Anika}, \text{YYZ}, \text{MLE}, \text{Jul29}, S_4), \tag{4.95}$$

$$\forall d.\ \neg\text{FlightsCancelled}(\text{YYZ}, \text{MLE}, d, S_4), \tag{4.96}$$

$$\text{At}(\text{Anika}, \text{YYZ}, S_4). \tag{4.97}$$

Thirdly, note that by Axiom 4.5.5, (4.95), (4.94), (4.96), and (4.97), the $fly(\text{Anika}, \text{YYZ}, \text{MLE})$ action is possible in S_4. By (4.97) and Axiom 4.5.14, Anika is located at MLE after the fly action happens:

$$\text{At}(\text{Anika}, \text{MLE}, S_5). \tag{4.98}$$

Also, by (4.94) and Axiom 4.5.15, the current date remains the same after the fly action happens:

$$\text{date}(S_5) = \text{Jul29}. \tag{4.99}$$

Fourthly (and finally), by Axiom 4.5.6, all the remaining $dateTick$ actions are possible starting in S_5. As discussed before, these $dateTick$ actions do not affect Anika's location, and thus by (4.98) she remains at MLE in all future situations on path P_0:

$$\forall s.\ \text{OnPath}(P_0, s) \wedge S_5 \preceq s \supset \text{At}(\text{Anika}, \text{MLE}, s). \tag{4.100}$$

By Axiom 4.5.15, each of these $dateTick$ actions flips the date to the next one starting from July 29. In particular, by (4.94) and Axiom 4.5.15, it follows that:

$$\text{date}(S_{12}) = \text{Aug05}, \tag{4.101}$$

where $S_{12} = do(dateTick, do(dateTick, do(dateTick, do(dateTick, do(dateTick,$ $do(dateTick, do(dateTick, S_5)))))))$. Thus, by (4.99), (4.101), the fact that there are no situations between S_5 and $S_6 = do(dateTick, S_5)$, (4.100), and Definition 4.5.2, it follows that:

$$\text{During}(\text{At}(\text{Anika}, \text{MLE}), \text{Jul29}, \text{Aug05})(P_0). \tag{4.102}$$

From (4.102) and (4.92), it follows that ϕ_0 holds over P_0, and thus we have:

$$G_R(P_0, 0, S_0). \tag{4.103}$$

Now, note that by Definitions 4.2.10 and 4.2.9, it follows that $\text{CGoal}(\phi_0, S_0)$ if and only if $\forall p, n.\ G_\cap(p, n, S_0) \supset \phi_0(p)$. I will show this by induction on n using Axiom

4.2.7. The base case follows from (4.103) and Axiom 4.2.7 – since there is a G_R-accessible path at level 0 in situation S_0, namely P_0, the $G_\cap$-accessible paths up to level 0 in S_0 are those that are G_R-accessible at 0 in S_0; by Axiom 4.5.1 and Definition 4.2.4, ϕ_0 holds over all such paths, i.e. $\forall p.\ G_\cap(p, 0, S_0) \supset \phi_0(p)$. For the inductive step, fix level N and assume that:

$$\forall p.\ G_\cap(p, N, S_0) \supset \phi_0(p). \tag{4.104}$$

From Axiom 4.2.7, we can see that the set of $G_\cap$-accessible paths at level $N + 1$ in S_0 is either the same as that of the one at level N in S_0 or a proper subset of it. It thus follows from this and (4.104) that ϕ_0 holds over all G_R-accessible paths at $N+1$ in S_0. Hence, it follows that $\forall p, n.\ G_\cap(p, n, S_0) \supset \phi_0(p)$ and thus we have $\text{CGoal}(\phi_0, S_0)$, i.e. that the first conjunct of the proposition holds.

Next, let us consider the second conjunct; assume that $\phi_1 = \text{During}(\text{At}(\text{Anika}, \text{EY-W}), \text{Jul29}, \text{Aug05})$. By Definitions 4.2.10 and 4.2.9, to show that $\neg\text{CGoal}(\phi_1, S_0)$ we have to prove that $\exists p.\ \forall n.\ G_\cap(p, n, S_0) \wedge \neg\phi_1(p)$. Since I have shown that $\text{CGoal}(\phi_0, S_0)$, by Definitions 4.2.10 and 4.2.9, and Proposition 4.4.2 (which implies that $G_\cap$ cannot be empty), it follows that $\exists p.\ \forall n.\ G_\cap(p, n, S_0) \wedge \phi_0(p)$. Thus from the definition of ϕ_0, we have:

$$\exists p.\ \forall n.\ G_\cap(p, n, S_0) \wedge \text{During}(\text{At}(\text{Anika}, \text{MLE}), \text{Jul29}, \text{Aug05})(p).$$

From this and the fact that At is functional over the date interval (Jul29, Aug05), i.e. Lemma 4.5.28, it follows that:

$$\exists p.\, \forall n.\, G_{\cap}(p, n, S_0) \wedge \neg\text{During}(\text{At}(\text{Anika}, \text{EYW}), \text{Jul29}, \text{Aug05})(p).$$

The conjunct thus follows.

The proof for the third conjunct is similar to that of the second one.

Finally, the proof for the last conjunct is also similar to that of the second one; in this case, we have to use the additional facts that initially the agent knows that Anika has not visited her best friend (i.e. Axiom 4.5.26(j)), that the fluent AnikaVisitedBFF becomes true only if she visits her best friend (i.e. Axiom 4.5.21), and that Anika can only visit her best friend between Jul29 and Aug05 if Anika is located at EYW during this period (i.e. Axiom 4.5.13), which is impossible as the second conjunct holds. □

Now assume that Anika has just learned from a trusted source that the Keys, at this time of the year, is very crowded, teeming with tourists from around the U.S. Since she was looking for a quiet vacation, she gets her agent to change its preferences by making the Keys its least preferred location. In my framework, such a reordering in an agent's preferences can be captured by a sequence of actions that involves dropping one or more p-goals and then re-adopting them at different levels. Recall that dropping an existing goal empties up all the levels where the dropped goal was a p-goal in the sense that the set of paths at each of these levels are replaced with paths that start with current K-accessible situations, essentially replacing the p-goals at these levels with the trivial goal that the correct history of actions in the current situation has occurred. Re-adopting the dropped goal at some new level n on the other hand adds the goal at level n by inserting the set of paths representing the goal at level n and pushing down one level in the goal hierarchy all the other p-goal levels that had priority n or lower before the $adopt$ action occurred.

Continuing with our example, for instance, here this intended reordering can be achieved by first dropping the p-goal that Anika be at the Keys during the specified week and then by re-adopting it at level 3 (or alternatively, by dropping the p-goal that Anika be at Varadero during that period and then re-adopting it at level 1). It can be shown that the agent has the following only p-goals after she drops the p-goal that $\text{During}(\text{At}(\text{Anika}, \text{EYW}), \text{Jul29}, \text{Aug05})$ and adopts it again at level 3 starting in situation S_0, i.e. in S_2, where $S_2 = do(adopt(\text{During}(\text{At}(\text{Anika}, \text{EYW}), \text{Jul29}, \text{Aug05}), 3), S_1)$ and $S_1 = do(drop(\text{During}(\text{At}(\text{Anika}, \text{EYW}), \text{Jul29}, \text{Aug05})), S_0)$:[40]

[40]Note that after the agent has dropped the p-goal at level 1 (i.e. in S_1), her only p-goal at this level becomes just the trivial goal to be on a path that includes the actions done so far, i.e. one that starts with a situation that has the same action history as S_1. I could have defined the optimizing-agent framework to get rid of/compact such “empty” p-goal levels, but this would have complicated the framework further.

Proposition 4.5.32.

$$
\begin{aligned}
\mathcal{D}_{TA} \models\ & \mathrm{OPGoal}(\exists s.\ \mathrm{Starts}(s) \wedge \mathrm{SameHist}(s, S_2) \\
&\qquad\qquad \wedge \mathrm{During}(\mathrm{At}(\mathrm{Anika}, \mathrm{MLE}), \mathrm{Jul29}, \mathrm{Aug05}) \\
&\qquad\qquad \wedge (\mathrm{Redeemed}(\mathrm{Anika}, \mathrm{YYZ}, \mathrm{MLE})\ \ \mathcal{B}\ \ (\mathrm{date} = \mathrm{Jul29})), 0, S_2) \\
&\wedge \mathrm{OPGoal}(\exists s.\ \mathrm{Starts}(s) \wedge \mathrm{SameHist}(s, S_2), 1, S_2) \\
&\wedge \mathrm{OPGoal}(\exists s.\ \mathrm{Starts}(s) \wedge \mathrm{SameHist}(s, S_2) \\
&\qquad\qquad \wedge \mathrm{During}(\mathrm{At}(\mathrm{Anika}, \mathrm{VRA}), \mathrm{Jul29}, \mathrm{Aug05}), 2, S_2) \\
&\wedge \mathrm{OPGoal}(\exists s.\ \mathrm{Starts}(s) \wedge \mathrm{SameHist}(s, S_2) \\
&\qquad\qquad \wedge \mathrm{During}(\mathrm{At}(\mathrm{Anika}, \mathrm{EYW}), \mathrm{Jul29}, \mathrm{Aug05}), 3, S_2) \\
&\wedge \mathrm{OPGoal}(\exists s.\ \mathrm{Starts}(s) \wedge \mathrm{SameHist}(s, S_2) \\
&\qquad\qquad \wedge \mathrm{Between}(\mathrm{AnikaVisitedBFF}, \mathrm{Jul29}, \mathrm{Aug05}), 4, S_2).
\end{aligned}
$$

Proof Sketch. First, note that from Definitions 4.2.1 and 4.2.2, it follows that to show that $\mathrm{OPGoal}(\phi_n, n, s)$ for some formula ϕ_n, some level n, and some situation s, we need to show that $\forall p.\ G(p, n, s) \equiv \phi_n(p)$. Now from the SSA for G, it follows that after the $drop(\mathrm{During}(\mathrm{At}(\mathrm{Anika}, \mathrm{EYW}), \mathrm{Jul29}, \mathrm{Aug05})$ action happens in S_0, i.e. in S_1, the agent's G-accessible paths at all levels are progressed to reflect that this action has just been performed; in addition to this, the SSA for G adds back all paths that share the same history with S_1 to the existing G-accessibility levels where the agent has the p-goal that $\text{During}(\text{At}(\text{Anika}, \text{EYW}), \text{Jul29}, \text{Aug05})$. Thus, from the SSA for G (i.e. Axiom 4.3.3 and Definitions 4.3.4 and 4.3.6) and Axiom 4.5.1, it follows that the agent's G-accessible paths in S_1 at various levels are as follows:

$$\begin{aligned} G(p, 0, S_1) \equiv \exists s.\ & \text{Starts}(p, s) \wedge \text{SameHist}(s, S_1) \wedge \\ & \text{During}(\text{At}(\text{Anika}, \text{MLE}), \text{Jul29}, \text{Aug05})(p) \wedge \\ & (\text{Redeemed}(\text{Anika}, \text{YYZ}, \text{MLE}) \ \mathcal{B} \ (\text{date} = \text{Jul29}))(p), \end{aligned} \tag{4.105}$$

$$G(p, 1, S_1) \equiv \exists s.\ \text{Starts}(p, s) \wedge \text{SameHist}(s, S_1), \tag{4.106}$$

$$\begin{aligned} G(p, 2, S_1) \equiv \exists s.\ & \text{Starts}(p, s) \wedge \text{SameHist}(s, S_1) \wedge \\ & \text{During}(\text{At}(\text{Anika}, \text{VRA}), \text{Jul29}, \text{Aug05})(p), \end{aligned} \tag{4.107}$$

$$\begin{aligned} G(p, 3, S_1) \equiv \exists s.\ & \text{Starts}(p, s) \wedge \text{SameHist}(s, S_1) \wedge \\ & \text{Between}(\text{AnikaVisitedBFF}, \text{Jul29}, \text{Aug05})(p). \end{aligned} \tag{4.108}$$

Moreover, by the SSA for G, it follows that after the $adopt(\text{During}(\text{At}(\text{Anika}, \text{EYW}), \text{Jul29}, \text{Aug05}), 3)$ action happens in S_1, i.e. in S_2, the G-accessible paths at all levels above level 3 are simply progressed to reflect the fact that this action has been performed. Thus from the SSA for G (i.e. Axiom 4.3.3 and Definitions 4.3.4 and 4.3.5) and (4.105), (4.106), and (4.107), it follows that the agent's G-accessible paths at these levels are as follows:

$$\begin{aligned} G(p, 0, S_2) \equiv \exists s.\ &\text{Starts}(p, s) \wedge \text{SameHist}(s, S_2) \wedge \\ &\text{During}(\text{At}(\text{Anika}, \text{MLE}), \text{Jul29}, \text{Aug05})(p) \wedge \\ &(\text{Redeemed}(\text{Anika}, \text{YYZ}, \text{MLE})\ \ \mathcal{B}\ \ (\text{date} = \text{Jul29}))(p), \end{aligned} \tag{4.109}$$

$$G(p, 1, S_2) \equiv \exists s.\ \text{Starts}(p, s) \wedge \text{SameHist}(s, S_2), \tag{4.110}$$

$$\begin{aligned} G(p, 2, S_2) \equiv \exists s.\ &\text{Starts}(p, s) \wedge \text{SameHist}(s, S_2) \wedge \\ &\text{During}(\text{At}(\text{Anika}, \text{VRA}), \text{Jul29}, \text{Aug05})(p). \end{aligned} \tag{4.111}$$

Also, the G-accessible paths at level 3 in situation S_2 are those that start with a situation that has the same action history as S_2 and over which the adopted goal holds, i.e.

$$\begin{aligned} G(p, 3, S_2) \equiv \exists s.\ &\text{Starts}(p, s) \wedge \text{SameHist}(s, S_2) \wedge \\ &\text{During}(\text{At}(\text{Anika}, \text{EYW}), \text{Jul29}, \text{Aug05})(p). \end{aligned} \tag{4.112}$$

Finally, the G-accessible paths at all levels below level 3 in S_2 are the ones that can be obtained by progressing the level immediately above it. Thus from the SSA for G and (4.108), it follows that:

$$\begin{aligned} G(p, 4, S_2) \equiv \exists s.\ &\text{Starts}(p, s) \wedge \text{SameHist}(s, S_2) \wedge \\ &\text{Between}(\text{AnikaVisitedBFF}, \text{Jul29}, \text{Aug05})(p). \end{aligned} \tag{4.113}$$

The proposition then follows from (4.109)–(4.113) and Definitions 4.2.1 and 4.2.2. □

Now suppose that at this point, the agent is informed that Cool-Air-Miles has announced that due to an outbreak of some unknown disease in many Indian ocean islands, they are forced to blacklist the Maldives for a month. Here this can be modeled by having the exogenous action $addBlacklistLoc(\text{MLE}, \text{Aug26})$ occur next. In that case, it can be shown that the agent will choose Varadero over Kaafu in $S_3 = do(addBlacklistLoc(\text{MLE}, \text{Aug26}), S_2)$ since her highest priority p-goal has become impossible to bring about:

Proposition 4.5.33.

$$
\begin{aligned}
\mathcal{D}_{TA} \models\ & \text{KImpossible}(\text{During}(\text{At}(\text{Anika}, \text{MLE}), \text{Jul29}, \text{Aug05}) \\
& \qquad\qquad\qquad \wedge (\text{Redeemed}(\text{Anika}, \text{YYZ}, \text{MLE})\ \ \mathcal{B}\ \ (\text{date} = \text{Jul29})), S_3) \\
& \wedge \neg\text{CGoal}(\text{During}(\text{At}(\text{Anika}, \text{MLE}), \text{Jul29}, \text{Aug05}) \\
& \qquad\qquad\qquad \wedge (\text{Redeemed}(\text{Anika}, \text{YYZ}, \text{MLE})\ \ \mathcal{B}\ \ (\text{date} = \text{Jul29})), S_3) \\
& \wedge \text{CGoal}(\text{During}(\text{At}(\text{Anika}, \text{VRA}), \text{Jul29}, \text{Aug05}), S_3) \\
& \wedge \neg\text{CGoal}(\text{During}(\text{At}(\text{Anika}, \text{EYW}), \text{Jul29}, \text{Aug05}), S_3) \\
& \wedge \neg\text{CGoal}(\text{Between}(\text{AnikaVisitedBFF}, \text{Jul29}, \text{Aug05}), S_3).
\end{aligned}
$$

Proof. I will prove each conjunct, one at a time. Let us consider the first conjunct. I will show this by contradiction. Assume that ϕ_0 stands for $\text{During}(\text{At}(\text{Anika}, \text{MLE}), \text{Jul29}, \text{Aug05}) \wedge (\text{Redeemed}(\text{Anika}, \text{YYZ}, \text{MLE})\ \ \mathcal{B}\ \ (\text{date} = \text{Jul29}))$. By Definitions

3.5.15, 3.5.14, 3.5.12, and 3.4.5, $\neg$KImpossible(ϕ_0, S_3) can be expanded as follows:

$$\begin{aligned}
&\neg\text{KImpossible}(\phi_0, S_3)\\
&\equiv \neg\text{KInevitable}(\neg\phi_0, S_3)\\
&\equiv \neg\text{Know}(\text{WeaklyInevitable}(\neg\phi_0, now), S_3)\\
&\equiv \neg\text{Know}(\forall p.\ \text{Starts}(p, now) \supset \neg\phi_0(p), S_3)\\
&\equiv \neg(\forall s'.\ K(s', S_3) \supset (\forall p.\ \text{Starts}(p, s') \supset \neg\phi_0(p)))\\
&\equiv \exists s'.\ K(s', S_3) \wedge \exists p.\ \text{Starts}(p, s') \wedge \phi_0(p).
\end{aligned}$$

Thus, $\neg$KImpossible(ϕ_0, S_3) holds if there is a path P^* that starts with some situation S^* that is K-accessible from situation S_3 and ϕ_0 holds over P^*:

$$\text{Starts}(P^*, S^*) \wedge K(S^*, S_3) \wedge \phi_0(P^*). \qquad (4.114)$$

From this and Definition 4.5.2, it follows that there is a situation S^{**} such that:

$$\text{OnPath}(P^*, S^{**}) \wedge \text{date}(S^{**}) = \text{Jul}29. \qquad (4.115)$$

Now, from (4.114), Definitions 3.5.11 and 3.5.7, and the definition of ϕ_0, it follows that:

$$\begin{aligned}
&\text{Redeemed}(\text{Anika}, \text{YYZ}, \text{MLE}) \;\; \mathcal{B} \;\; (\text{date} = \text{Jul29})(P^*) \\
&\equiv \neg(\neg\text{Redeemed}(\text{Anika}, \text{YYZ}, \text{MLE}) \;\; \mathcal{U} \;\; (\text{date} = \text{Jul29}))(P^*) \\
&\equiv \neg\exists s, s'.\ \text{Starts}(P^*, s) \wedge \text{OnPath}(P^*, s') \wedge \text{date}(s') = \text{Jul29} \\
&\qquad\qquad \wedge \forall s^*.\ s \leq s^* < s' \supset \neg\text{Redeemed}(\text{Anika}, \text{YYZ}, \text{MLE}, s^*).
\end{aligned}$$

It follows from this, (4.114), and (4.115), that there is a situation S_R that is between the situation interval $[S^*, S^{**})$ and over which Anika has redeemed her Cool Air Miles for a ticket from YYZ to MLE:

$$S^* \preceq S_R \prec S^{**} \wedge \text{Redeemed}(\text{Anika}, \text{YYZ}, \text{MLE}, S_R). \tag{4.116}$$

Now, it follows from Axiom 4.5.26(g) that initially the agent knows that Redeemed is false, i.e. $\text{Know}(\neg\text{Redeemed}(\text{Anika}, \text{YYZ}, \text{MLE}), S_0)$. Fix arbitrary K-accessible situation S_0^k in S_0. Then from Axiom 4.5.26(g) and Definition 3.4.5, we have:

$$\neg\text{Redeemed}(\text{Anika}, \text{YYZ}, \text{MLE}, S_0^k). \tag{4.117}$$

By the SSA for Redeemed, i.e. Axiom 4.5.18, it follows that Redeemed will only become true if the $redeemMiles(\text{Anika}, \text{YYZ}, \text{MLE}, d)$ action happens for some date d. Since by the unique names axioms the $drop$, $adopt$, and $addBlacklistLoc$ actions are not the same as the $redeemMiles$ action, it follows from (4.117) and Axiom 4.5.18

that:

$\neg\text{Redeemed}(\text{Anika}, \text{YYZ}, \text{MLE}, S_3^k)$,

where $S_3^k = do(addBlacklistLoc(\text{MLE}, \text{Aug26}), do(adopt(\phi_2, 3), do(drop(\phi_2), S_0^k)))$ and $\phi_2 = \text{During}(\text{At}(\text{Anika}, \text{EYW}), \text{Jul29}, \text{Aug05})$.

Note that since by Axiom 3.4.10, all situations that are K-accessible in S_3 must be progressed from those that were K-accessible in S_0, the above also holds for any arbitrary K-accessible situation in S_3. From this and (4.114), it thus follows that:

$$\neg\text{Redeemed}(\text{Anika}, \text{YYZ}, \text{MLE}, S^*).$$

From this, (4.116), and Axiom 4.5.18, it follows that there must be a situation s between S^* and S^{**} where an appropriate $redeemMiles$ action occurs:

$$\exists s.\ S^* \prec do(redeemMiles(\text{Anika}, \text{YYZ}, \text{MLE}), s) \prec S^{**}. \qquad (4.118)$$

I will show that this is impossible since the preconditions of $redeemMiles$ will always be false in this interval. Note that by Axiom 4.5.26(e), it follows that the agent knows that initially MLE is not blacklisted for all dates, i.e. $\forall d.\ \text{Know}(\neg\text{Blacklisted}(\text{MLE}, d), S_0)$. Then from Axiom 4.5.26(e) and Definition 3.4.5, for any arbitrary K-accessible situation S_0^k we have:

$$\neg\text{Blacklisted}(\text{MLE}, d, S_0^k). \qquad (4.119)$$

By the SSA for Blacklisted, i.e. Axiom 4.5.16, it follows that MLE will be blacklisted only if the $addBlacklistLoc(\text{MLE}, d)$ action happens for some date d. Since by the unique names axioms the $drop$ and the $adopt$ actions are not the same as the $addBlacklistLoc$ action, it follows from Axiom 4.5.16 that:

$$\forall d.\ \neg\text{Blacklisted}(\text{MLE}, d, S_2^k),$$
$$\text{where } S_2^k = do(adopt(\phi_2, 3), do(drop(\phi_2), S_0^k))).$$

Again, since by Axiom 3.4.10, all situations that are K-accessible in S_2 must be progressed from those that were K-accessible in S_0, the above also holds for any arbitrary K-accessible situation in S_2. From this and Definition 3.4.5, it follows that the agent will also know in S_2 that this is the case:

$$\text{Know}(\forall d.\ \neg\text{Blacklisted}(\text{MLE}, d), S_2). \tag{4.120}$$

But after the $addBlacklistLoc(\text{MLE}, \text{Aug26})$ action happens, by Axiom 4.5.16 it follows that MLE will be blacklisted at least until Aug 26:

$$\text{Blacklisted}(\text{MLE}, \text{Aug26}, S_3^k).$$

Moreover, since by Axiom 3.4.10, all situations that are K-accessible in S_3 must be progressed from those that were K-accessible in S_2, the above also holds for any arbitrary K-accessible situation in S_3. From this and Definition 3.4.5, it follows that the agent will know this is S_3:

$$\text{Know}(\text{Blacklisted}(\text{MLE}, \text{Aug26}), S_3).$$

It follows from this and (4.114) that:

$$\text{Blacklisted}(\text{MLE}, \text{Aug26}, S^*). \tag{4.121}$$

Now by this and the preconditions of $redeemMiles$, i.e. Axiom 4.5.12, for (4.118) to hold, MLE must be removed from the blacklist no later than the date of S^{**}, which by the above definition of S^{**} is Jul 29. But by (4.121) and Axiom 4.5.8, this is impossible as the $removeBlacklistLoc$(MLE) action can only happen in a situation whose date is later than Aug 26. It thus follows from this and the SSA for Blacklisted, i.e. Axiom 4.5.16 that MLE is blacklisted for all situations in the interval $[S^*, S^{**}]$:

$$\forall s.\ S^* \preceq s \preceq S^{**} \supset \text{Blacklisted}(\text{MLE}, \text{Aug26}, s).$$

Thus by this and Axiom 4.5.12 the action $redeemMiles(\text{Anika}, \text{YYZ}, \text{MLE})$ cannot occur between S^* and S^{**}, which is contradictory to (4.118). Hence it follows that:

$$\text{KImpossible}(\phi_0, S_3), \tag{4.122}$$

i.e. the first conjunct of the proposition holds. Note that it also follows that such a path P^* does not exist, i.e.:

$$\neg\exists p, s.\ \text{Starts}(p, s) \land K(s, S_3) \land \phi_0(p). \tag{4.123}$$

I will next show that the second conjunct holds, i.e. $\neg$CGoal(ϕ_0, S_3). This follows from (4.122) and Corollary 4.4.4.

Assume that ϕ_1 stands for During(At(Anika, VRA), Jul29, Aug05). I will next show that the third conjunct holds, i.e. that CGoal(ϕ_1, S_3). In the following I will do this by showing that there is a path that is in the intersection of the G_R-accessible paths (i.e. in $G_\cap$) up to level 1 in S_3 and over which ϕ_1 holds, that this path is also G_R-accessible at level 2 in S_3, and that ϕ_1 holds over all paths that are in G_R-accessible at level 2 in S_3; then I will use these to show that ϕ_1 holds over all paths that are in $G_\cap$ up to any level n. First, let me prove the first one. Let me start by constructing such a path P_1^* by giving the situations/actions on the path; I can then show that $G_\cap(P_1^*, 1, S_3)$. The situations/actions on P_1^* are as follows: S_3, $S_4 = do(purchase(\text{Anika}, \text{YYZ}, \text{VRA}, \text{Jul29}), S_3)$, $S_5 = do(dateTick, S_4)$, $S_6 = do(dateTick, S_5)$, $S_7 = do(dateTick, S_6)$, $S_8 = do(fly(\text{Anika}, \text{YYZ}, \text{VRA}), S_7)$, followed by infinitely many $dateTick$ actions.

Now, note that P_1^* starts with S_3, and by the reflexivity of K (i.e. Axioms 3.4.2 and 3.4.10), we have:

$$\text{Starts}(P_1^*, S_3) \wedge K(S_3, S_3). \tag{4.124}$$

I will now show that ϕ_1 holds over P_1^*. To this end, note that from Definition 4.5.2, it follows that $\phi_1(P_1^*)$ if and only if the following holds:

$$\exists s_1, s_2.\ \text{OnPath}(P_1^*, s_1) \wedge \text{OnPath}(P_1^*, s_2) \wedge \text{date}(s_1) = \text{Jul29} \wedge \text{date}(s_2) = \text{Aug05}$$
$$\wedge\ \forall s.\ \text{OnPath}(P_1^*, s) \wedge \text{date}(s) > \text{Jul29} \wedge \text{date}(s) < \text{Aug05} \supset \text{At}(\text{Anika}, \text{VRA}, s).$$

I will now argue that this is indeed that case. First note that by Axiom 4.5.11, the $purchase(\text{Anika}, \text{YYZ}, \text{VRA}, \text{Jul29})$ action is always possible, and thus it is possible in S_3. Also by Axiom 4.5.19, Anika has the ticket from YYZ to VRA for Jul 29 in S_4:

$$\text{HasTicket}(\text{Anika}, \text{YYZ}, \text{VRA}, \text{Jul29}, S_4). \qquad (4.125)$$

Now by Axiom 4.5.6, the $dateTick$ action is always possible. By Axiom 4.5.26(d) and Definition 3.4.5, it follows that $\text{date}(S_0) = \text{Jul26}$. Moreover, since by the unique names axioms the $drop$, $adopt$, $addBlacklistLoc$, and $purchase$ actions do not refer to the $dateTick$ action, by Axiom 4.5.15 it follows that the date of S_4 is also Jul 26, i.e.: $\text{date}(S_4) = \text{Jul26}$. Moreover, from this and Axiom 4.5.15 it follows that:

$$\text{date}(S_7) = \text{Jul29}. \qquad (4.126)$$

Now, note that by (4.125) and Axiom 4.5.19, it follows that:

$$\text{HasTicket}(\text{Anika}, \text{YYZ}, \text{VRA}, \text{Jul29}, S_7). \qquad (4.127)$$

Moreover, from Axiom 4.5.26(f) and Definition 3.4.5, it follows that:

$$\forall l, d.\ \neg\text{FlightsCancelled}(\text{YYZ}, l, d, S_0).$$

From this, the unique names axioms (which say that none of the actions performed so far is a $startDisruptionBtwn$ action), and Axiom 4.5.17, it follows that:

$$\forall l, d.\ \neg\text{FlightsCancelled}(\text{YYZ}, l, d, S_7). \tag{4.128}$$

Furthermore, from Axiom 4.5.26(b) and Definition 3.4.5, it follows that:

$$\text{At}(\text{Anika}, \text{YYZ}, S_0).$$

From this, the unique names axioms (which say that none of the actions performed so far is a fly action), and Axiom 4.5.14, it follows that:

$$\text{At}(\text{Anika}, \text{YYZ}, S_7). \tag{4.129}$$

Again, from Axiom 4.5.24, it follows that:

$$\text{YYZ} \neq \text{VRA}.$$

From this, (4.127), (4.126), (4.128), (4.129), and Axiom 4.5.5 it follows that the fly(Anika, YYZ, VRA) action is possible in S_7.

Now, by (4.129) and Axiom 4.5.14, Anika will be in VRA after the fly(Anika, YYZ, VRA) action happens in S_7, i.e.:

$$\text{At}(\text{Anika}, \text{VRA}, S_8).$$

Also, since by our construction above none of the rest of the actions on P_1^* is a fly action, Anika will stay in VRA for all situations that follows S_7 on P_1^*:

$$\forall s.\ \text{OnPath}(P_1^*, s) \land S_7 \prec s \supset \text{At}(\text{Anika}, \text{VRA}, s). \tag{4.130}$$

Again, from (4.126), the unique names axioms (which says that the fly action is not a $dateTick$ action), and Axiom 4.5.15, it follows that:

$$\text{date}(S_{14}) = \text{Aug}05, \tag{4.131}$$

where, $S_{14} = do(dateTick, do(dateTick, do(dateTick, do(dateTick, do(dateTick, do(dateTick, S_8))))))$. Recall from above that ϕ_1 holds over P_1^* if there are two situations associated with dates Jul 29 and Aug 05 on P_1^* and Anika is at VRA in all the situations on P_1^* that are within this situation interval (see the unnumbered equation right after (4.124)). It thus follows from (4.126), (4.131), and (4.130) that:

$$\phi_1(P_1^*). \tag{4.132}$$

Next, I will show that $G_\cap(P_1^*, 1, S_3)$. Since by the unique names axioms the $addBlacklistLoc$ action is not an $adopt$ or $drop$ action, the G-accessible paths in S_2 will simply be progressed when it happens. Thus from Proposition 4.5.32, Axiom

4.3.3, and Definitions 4.3.4, 4.2.1 and 4.2.2, it follows that:

$$\forall p.\ G(p,0,S_3) \equiv \exists s.\ \text{Starts}(p,s) \wedge \text{SameHist}(s,S_3) \wedge \phi_0(p), \quad (4.133)$$

$$\forall p.\ G(p,1,S_3) \equiv \exists s.\ \text{Starts}(p,s) \wedge \text{SameHist}(s,S_3), \quad (4.134)$$

$$\forall p.\ G(p,2,S_3) \equiv \exists s.\ \text{Starts}(p,s) \wedge \text{SameHist}(s,S_3) \wedge \phi_1(p). \quad (4.135)$$

From (4.133), it follows that $\forall p.\ G(p,0,S_3) \supset \phi_0(p)$. From this, (4.123), and Definition 4.2.4, it follows that there are no G_R-accessible paths at level 0 in S_3, i.e.:

$$\neg\exists p.\ G_R(p,0,S_3). \quad (4.136)$$

Then by Axiom 4.2.7, it follows that:

$$\forall p.\ G_\cap(p,0,S_3) \equiv \exists s'.\ \text{Starts}(p,s') \wedge K(s',S_3).$$

By (4.124), P_1^* is indeed such a path and thus we have:

$$G_\cap(P_1^*,0,S_3). \quad (4.137)$$

Moreover, from (4.124) and Definition 3.2.7, it follows that:

$$\text{Starts}(P_1^*,S_3) \wedge \text{SameHist}(S_3,S_3). \quad (4.138)$$

Thus by this and (4.134), it follows that $G(P_1^*,1,S_3)$. Moreover, from this, (4.124), and Definition 4.2.4, it follows that $G_R(P_1^*,1,S_3)$. From this, (4.137), and Axiom 4.2.7, it follows that:

$$G_\cap(P_1^*,1,S_3). \quad (4.139)$$

Furthermore, from (4.124), (4.138), (4.132), (4.135), and Definition 4.2.4, it follows that:

$$G_R(P_1^*, 2, S_3), \tag{4.140}$$

$$\forall p.\ G_R(p, 2, S_3) \supset \phi_1(p). \tag{4.141}$$

Now by (4.139), (4.140), (4.141), and Axiom 4.2.7, it follows that ϕ_1 holds over all paths that are in the intersection of the G_R-accessible paths up to level 2 in S_3, i.e.:

$$\forall p.\ G_\cap(p, 2, S_3) \supset \phi_1(p). \tag{4.142}$$

From this and Axiom 4.2.7, it follows that:

$$\forall p.\ (\forall n.\ G_\cap(p, n, S_3)) \supset \phi_1(p).$$

Finally, from this and Definitions 4.2.10 and 4.2.10 and 4.2.9, it follows that the third conjunct holds, i.e. $\text{CGoal}(\phi_1, S_3)$.

Next, I will prove that the fourth conjunct holds. Assume that ϕ_2 stands for $\text{During}(\text{At}(\text{Anika}, \text{EYW}), \text{Jul29}, \text{Aug05})$. Then I need to show that $\neg\text{CGoal}(\phi_2, S_3)$. By Definitions 4.2.10 and 4.2.9, to show that $\neg\text{CGoal}(\phi_2, S_3)$ we have to prove that $\exists p.\ (\forall n.\ G_\cap(p, n, S_3)) \wedge \neg\phi_2(p)$. Since I have shown that $\text{CGoal}(\phi_1, S_3)$, by Definitions 4.2.10 and 4.2.9 it follows that $\exists p.\ (\forall n.\ G_\cap(p, n, S_3)) \wedge \phi_1(p)$. Thus from the definition of ϕ_1, we have:

$$\exists p.\ \forall n.\ G_\cap(p, n, S_0) \wedge \text{During}(\text{At}(\text{Anika}, \text{VRA}), \text{Jul29}, \text{Aug05})(p).$$

From this, the fact that At is functional over the date interval (Jul29, Aug05), i.e. Lemma 4.5.28, and that EYW $\neq$ VRA, i.e. Axiom 4.5.24, it follows that:

$$\exists p.\ \forall n.\ G_{\cap}(p, n, S_0) \wedge \neg\text{During}(\text{At}(\text{Anika}, \text{EYW}), \text{Jul29}, \text{Aug05})(p). \tag{4.143}$$

The conjunct thus follows.

Finally, I will prove that the last conjunct holds; this is related to the proof of the previous conjunct. In this case, note that initially the agent knows that Anika has not visited her best friend (i.e. Axiom 4.5.26(j)), that the fluent AnikaVisitedBFF becomes true only if she visits her best friend (i.e. Axiom 4.5.21), that by the unique names axioms for actions none of the actions performed so far is the $anikaVisitsBFF$ action, and that Anika can only visit her best friend between Jul29 and Aug05 if she is located at EYW during this period (i.e. Axiom 4.5.13). Given that she is not, i.e. (4.143), it follows that:

$$\exists p.\ \forall n.\ G_{\cap}(p, n, S_0) \wedge \neg\text{Between}(\text{AnikaVisitedBFF}, \text{Jul29}, \text{Aug05})(p).$$

From this and Definitions 4.2.10 and 4.2.9 it then follows that the last conjunct holds, i.e.:

$$\neg\text{CGoal}(\text{Between}(\text{AnikaVisitedBFF}, \text{Jul29}, \text{Aug05}), S_3).$$

The proposition thus follows. $\square$

Finally, assume that the agent later learns that as a result of an upcoming hurricane, all flights going from Toronto to Varadero are cancelled for a week. Again, we can model this by having the exogenous action $startDisruptionBtwn(\text{YYZ}, \text{VRA}, \text{Aug02})$ occur next. In that case, it can be shown that the agent will choose the Keys as Anika's destination in $S_4 = do(startDisruptionBtwn(\text{YYZ}, \text{VRA}, \text{Aug02}), S_3)$, since it has become impossible to fly to Varadero before Aug03:

Proposition 4.5.34.

$$
\begin{aligned}
&\mathcal{D}_{TA} \models \\
&\text{Know}(\forall s.\ now \prec s \land \text{date}(s) < \text{Aug03} \supset \neg\text{Poss}(fly(\text{Anika}, \text{YYZ}, \text{VRA}), s), S_4) \\
&\qquad \land \neg\text{CGoal}(\text{During}(\text{At}(\text{Anika}, \text{MLE}), \text{Jul29}, \text{Aug05}) \\
&\qquad\qquad\qquad \land (\text{Redeemed}(\text{Anika}, \text{YYZ}, \text{MLE})\ \mathcal{B}\ (\text{date} = \text{Jul29})), S_4) \\
&\qquad \land \neg\text{CGoal}(\text{During}(\text{At}(\text{Anika}, \text{VRA}), \text{Jul29}, \text{Aug05}), S_4) \\
&\qquad \land \text{CGoal}(\text{During}(\text{At}(\text{Anika}, \text{EYW}), \text{Jul29}, \text{Aug05}), S_4) \\
&\qquad \land \text{CGoal}(\text{Between}(\text{AnikaVisitedBFF}, \text{Jul29}, \text{Aug05}), S_4).
\end{aligned}
$$

Proof Sketch. The proof for this can be approached similarly to that of the previous proposition; in fact, much of this proof sketch is a continuation of the previous one. For the first conjunct, it can be shown that:

- by Axiom 4.5.26(f), the agent initially knows that all flights from YYZ are running;

- by the unique names axioms, none of the actions performed so far is a $startDisr$-$uptionBtwn$ action, and thus by the SSA for FlightsCancelled (Axiom 4.5.17) and the SSA for K (Axiom 3.4.10), the agent still knows in S_3 that all flights from YYZ are operating; thus by Axiom 4.5.9, the agent knows that the $startDis$-$ruptionBtwn(\text{YYZ}, \text{MLE}, \text{Aug02})$ action is possible in S_3; and
- by the SSA for FlightsCancelled (Axiom 4.5.17), the flights from YYZ to VRA are cancelled at least until Aug03 in S_4 and by Axiom 3.4.10, the agent also knows this; moreover, by Axiom 4.5.17 and the preconditions of $endDisruption$-$Btwn$ (Axiom 4.5.10), the agent also knows that this route cannot be restored anytime before Aug03.

Thus by the preconditions of fly, i.e. Axiom 4.5.5, the agent knows that it is not possible to execute the $fly(\text{Anika}, \text{YYZ}, \text{VRA})$ action in a situation that is preceded by a K-accessible situation in S_4 and whose date is earlier than Aug03.

Next consider the last two conjuncts; let's call the content of these c-goals ϕ_2 and ϕ_3, respectively. Note that the last two conjuncts can be proven as before by (i) constructing a path over which ϕ_2 and ϕ_3 hold, (ii) proving that this path is in $G_\cap$ up to level 2 in S_4, (iii) showing that the path is in the G_R relation at level 3 in S_4, (iv)

showing that $\phi_2 \wedge \phi_3$ holds over all paths that are in the G_R relation at level 3 in S_4, and then using these to (v) prove that $\phi_2 \wedge \phi_3$ holds over all paths that are in the $G_\cap$ intersection for all levels n in S_4. Here, I will handle the proof for these two conjuncts simultaneously by constructing a single such path P^* over which ϕ_2 and ϕ_3 hold, proving that $G_\cap(P^*, 2, S_4)$, i.e. step (i) and (ii) above. The proof for (iii), (iv), and (v) are similar to that of the previous proposition, so I will skip them.

I'll start by giving the situations/actions for P^*: $S_4, S_5 = do(purchase(\text{Anika}, \text{YYZ}, \text{EYW}, \text{Jul29}), S_4), S_6 = do(dateTick, S_5), S_7 = do(dateTick, S_6), S_8 = do(dateTick, S_7), S_9 = do(fly(\text{Anika}, \text{YYZ}, \text{EYW}), S_8), S_{10} = do(dateTick, S_9), S_{11} = do(anikaVisitsBFF, S_{10})$, followed by infinitely many $dateTick$ actions. It is straightforward to show that $\phi_2 \wedge \phi_3$ holds over P^*.

Next, I'll show (ii). Note that I have shown earlier that there are no G_R-accessible paths at level 0 in S_3, i.e. $\neg\exists p.\ G_R(p, 0, S_3)$ (see (4.136) in the proof of Proposition 4.5.33). From this and the SSA for G (Axiom 4.3.3), it follows that this also holds for S_4 since by the unique names axioms the $startDisruptionBtwn$ action does not refer to an $adopt$ or a $drop$ action (and thus no paths were added back to the G-relation at level 0 in S_4). Thus by Axiom 4.2.7, it follows that:

$$\forall p.\ G_\cap(p, 0, S_4) \equiv \exists s'.\ \text{Starts}(p, s') \wedge K(s', S_4).$$

Now consider path P^* that starts with S_4; note that it follows from the above and the reflexivity of K that P^* is in $G_\cap$ up to level 0 in S_4 (as it starts with a K-accessible situation in S_4, namely S_4):

$$G_\cap(P^*, 0, S_4). \tag{4.144}$$

Moreover, I have shown earlier (see (4.134) in the proof for Proposition 4.5.33) that:

$$\forall p.\ G(p, 1, S_3) \equiv \exists s.\ \text{Starts}(p, s) \wedge \text{SameHist}(s, S_3).$$

From this, the unique names axioms for actions, and the SSA for G (Axiom 4.3.3), it follows that these G-accessible paths will simply be progressed when the *startDisruptionBtwn* action happens in S_3 since this action is not an *adopt* or a *drop* action, and thus we have:

$$\forall p.\ G(p, 1, S_4) \equiv \exists s.\ \text{Starts}(p, s) \wedge \text{SameHist}(s, S_4).$$

Since by Definition 3.2.7, SameHist(S_4, S_4), it follows that $G(P^*, 1, S_4)$, and from Definition 4.2.4 and the reflexivity of K that $G_R(P^*, 1, S_4)$, and finally from (4.144) and Axiom 4.2.7 that:

$$G_\cap(P^*, 1, S_4). \tag{4.145}$$

Furthermore, previously I have shown that (see (4.137) in the proof of Proposition 4.5.33):

$$\forall p.\ G(p, 2, S_3) \equiv \exists s.\ \text{Starts}(p, s) \wedge \text{SameHist}(s, S_3) \wedge \phi_1(p),$$

where ϕ_1 stands for $\text{During}(\text{At}(\text{Anika}, \text{VRA}), \text{Jul29}, \text{Aug05})$. From this and the SSA for G (Axiom 4.3.3), it follows that these G-accessible paths will simply be progressed when the $startDisruptionBtwn$ action happens in S_3 since the action performed in S_3 is not an $adopt$ or a $drop$ action. Thus from this and the fact that the date in S_4 is the same as in S_3 (by SSA 4.5.15 and the fact that the $startDisruptionBtwn$ action is not a $dateTick$ action, i.e. Axiom 4.5.22), we have:

$$\forall p.\ G(p, 2, S_4) \equiv \exists s.\ \text{Starts}(p, s) \land \text{SameHist}(s, S_4) \land \phi_1(p).$$

But since the agent knows that the $fly(\text{Anika}, \text{YYZ}, \text{EYW})$ action is not executable in any situation before Aug03 (see the third bullet above), it follows from the definition of During (i.e. Definition 4.5.2) that there are no paths that start with S_4 or some situation that has the same action history as S_4 and over which ϕ_1 holds, and thus there are no G_R-accessible paths at level 2 in S_4, i.e. $\neg\exists p.\ G_R(p, 2, S_4)$. Thus from this, (4.144), (4.145), and Axiom 4.2.7, it follows that:

$$G_\cap(P^*, 2, S_4). \tag{4.146}$$

Again, the proof for (iii), (iv), and (v) is similar to that of the previous property. Thus the last two conjuncts hold.

Finally, consider the second and the third conjuncts. Let's call the content of these c-goals ϕ_0 and ϕ_1, respectively. Now by Definitions 4.2.10 and 4.2.9, to show that

$\neg$CGoal(ϕ, S_4) for some ϕ, we have to prove that $\exists p.\ (\forall n.\ G_\cap(p, n, S_4)) \wedge \neg\phi(p)$. This is how both the second and the third conjuncts can be proven. Now, note that in the above, I argued that $\phi_2 \wedge \phi_3$ holds over all paths p that are in the $G_\cap$ intersection for all levels n in S_4 (see (v) above); as argued in the previous proof, neither ϕ_0 nor ϕ_1 can hold over such a path p as by Lemma 4.5.28, At is functional over date intervals. This thus proves that the second and the third conjuncts hold.[41] The proposition thus follows. $\square$

In this section, I used a simple example to illustrate how an agent's prioritized goals can be specified in my framework, how they can be used to derive her chosen goals, and how these chosen goals evolve when the world changes. My formalization of prioritized goals is related to the notion of *preferences* found in the literature [215, 216], and in fact, my framework can be used to specify an agent's preferences. In the above example, I considered the agent's priorities over a couple of preference criteria only, namely the choice of Anika's destination location and the possibility of redeeming Cool Air Miles. In this example, the agent has preferences over mutually exclusive choices. I could have easily extended it to deal with multiple preference criteria over which the agent has independent choices. For instance, assume that Anika

[41]Note that I didn't however prove that $\exists p.\ \forall n.\ G_\cap(p, n, S_4)$. Nevertheless, this follows from Proposition 4.4.2 and Definitions 4.2.10 and 4.2.9.

absolutely requires that she stays in a four-star or better hotel room as she had a very bad experience with lower quality rooms last year. Moreover, she prefers not to spend for a five-star hotel room. Nevertheless, she is willing to upgrade to a five-star room to be in a preferred destination. In this framework, Anika's preferences can be specified using a goal hierarchy whose first few levels are as follows:

$$
\begin{aligned}
&\text{Init}(s) \supset \\
&\quad ((G(p,0,s) \equiv \exists s'.\ \text{Starts}(p,s') \wedge \text{Init}(s') \wedge \\
&\qquad\qquad \text{During}(\text{At}(\text{Anika},\text{MLE}),\text{Jul29},\text{Aug05})(p) \wedge \\
&\qquad\qquad (\text{Redeemed}(\text{Anika},\text{YYZ},\text{MLE})\ \mathcal{B}\ (\text{date}=\text{Jul29}))(p) \wedge \\
&\qquad\qquad \text{During}(\text{BookedRoom}(4*),\text{Jul29},\text{Aug05})(p)) \\
&\quad \wedge (G(p,1,s) \equiv \exists s'.\ \text{Starts}(p,s') \wedge \text{Init}(s') \wedge \\
&\qquad\qquad \text{During}(\text{At}(\text{Anika},\text{MLE}),\text{Jul29},\text{Aug05})(p) \wedge \\
&\qquad\qquad (\text{Redeemed}(\text{Anika},\text{YYZ},\text{MLE})\ \mathcal{B}\ (\text{date}=\text{Jul29}))(p) \wedge \\
&\qquad\qquad \text{During}(\text{BookedRoom}(5*),\text{Jul29},\text{Aug05})(p)) \\
&\quad \wedge (G(p,2,s) \equiv \exists s'.\ \text{Starts}(p,s') \wedge \text{Init}(s') \wedge \\
&\qquad\qquad \text{During}(\text{At}(\text{Anika},\text{EYW}),\text{Jul29},\text{Aug05})(p) \wedge \\
&\qquad\qquad \text{During}(\text{BookedRoom}(4*),\text{Jul29},\text{Aug05})(p)) \\
&\quad \wedge (G(p,3,s) \equiv \exists s'.\ \text{Starts}(p,s') \wedge \text{Init}(s') \wedge \\
&\qquad\qquad \text{During}(\text{At}(\text{Anika},\text{EYW}),\text{Jul29},\text{Aug05})(p) \wedge \\
&\qquad\qquad \text{During}(\text{BookedRoom}(5*),\text{Jul29},\text{Aug05})(p)).
\end{aligned}
$$

Note that, the above specification assumes that the individual utility of the two preference criteria under question is zero – it is unnecessary to book a hotel room if Anika is not going anywhere; also, it does not make sense for her to go somewhere without having a hotel room booked. In this framework, such dependent preferences can be captured by specifying them as conjunctive goals at the same level (as above). I can also simulate cases where preference independence is assumed, as commonly found in the planning with preferences literature [215]. When preferences are independent and additive, rather than modeling them as conjunctions at the same level, we need to specify these preferences at different levels. For example, assume that independently of her travel, Anika prefers to submit a research paper before July 25th rather than not doing so. This can be modeled by adding two prioritized goals to her goal hierarchy; at some higher priority level, she has the p-goal to submit the paper before July 25, while at some lower priority level, she has the p-goal to submit it after July 25. The goal hierarchy looks as follows (here the "$\cdots$" stands for the goals for levels 1 and 2 as in Axiom 4.5.1):

$$\begin{aligned}
&\text{Init}(s) \supset \\
&\quad ((G(p,0,s) \equiv \exists s'.\ \text{Starts}(p,s') \wedge \text{Init}(s') \wedge \\
&\qquad\qquad \text{During}(\text{At}(\text{Anika}, \text{MLE}), \text{Jul}29, \text{Aug}05)(p) \wedge \\
&\qquad\qquad (\text{Redeemed}(\text{Anika}, \text{YYZ}, \text{MLE})\ \ \mathcal{B}\ \ (\text{date} = \text{Jul}29))(p)) \\
&\quad \wedge \cdots \\
&\quad \wedge (G(p,3,s) \equiv \exists s'.\ \text{Starts}(p,s') \wedge \text{Init}(s') \wedge \\
&\qquad\qquad \text{Between}(\text{AnikaVisitedBFF}, \text{Jul}29, \text{Aug}05)(p)) \\
&\quad \wedge (G(p,n,s) \equiv \exists s'.\ \text{Starts}(p,s') \wedge \text{Init}(s') \wedge \\
&\qquad\qquad (\text{PaperSubmitted}(\text{Anika})\ \ \mathcal{B}\ \ (\text{date} = \text{Jul}25))(p)) \\
&\quad \wedge (G(p,n+1,s) \equiv \exists s'.\ \text{Starts}(p,s') \wedge \text{Init}(s') \wedge \\
&\qquad\qquad (\neg\text{PaperSubmitted}(\text{Anika})\ \ \mathcal{B}\ \ (\text{date} = \text{Jul}25))(p)),
\end{aligned}$$

where $n \geq 4$. Note that the actual position of these p-goals relative to other p-goals in the hierarchy does not affect the chosen goals involving travel as long as there is no interaction w.r.t achievability, i.e. as this preference criterion is independent of her other preference attributes (independence of achievability implies independence of preferences).

To summarize, in this section I showed how a realistic travel planning agent can be specified, illustrating the usefulness of my proposed framework. I discussed how the agent's prioritized goals and chosen goals change as a result of actions, including exogenous ones. In addition, I discussed how different types of preferences of the example agent can be modeled within this account.

4.6 Conclusion, Discussion and Future Work

In this chapter, I proposed a possible worlds semantics for specifying the prioritized goals of an agent. I showed how this hierarchy of goals can be used to specify her chosen goals assuming that agents chose as many of their highest priority goals as possible while ensuring their chosen goals are consistent with each other and with the agent's knowledge. Then using a successor-state axiom, I specified how these goals evolve when the world changes. My formalization ensures that an agent's chosen goals are always consistent, that her goals properly evolve as a result of regular actions as well as of adopting and dropping goals, and that agents can introspect their goals. I also showed that in my account, while chosen goals are closed under logical consequence, primary c-goals are not, and thus they are side-effect free. In other words, an agent need not (primarily) intend all known consequences of her (primary) intentions; e.g. she can have the primary c-goal that ϕ and know that ϕ always implies ψ, and still not have the primary c-goal that ψ. In my framework, an agent always tries to optimize her chosen goals, and so agents specified using this framework behave somewhat ideally. Given this simplifying assumption, here I have focused on developing an expressive framework that captures an idealized form of rationality without worrying about tractability. It would be desirable to study restricted fragments of the logic where reasoning is tractable. Also, before defining more limited forms of rationality, one should have a clear specification of what ideal rationality really is so that one understands what compromises are being made.

Note that I could have modeled prioritized goals syntactically by treating the agent's set of p-goals as just an arbitrary set of temporal formulae, and then defining the set of c-goals as a maximal consistent subset of p-goals. However, my possible world semantics has some advantages over this: it clearly defines when goals are consistent with each other and with what is known. One can easily specify how goals change when an action a occurs, e.g. the goal to do a next and then do b becomes the goal to do b next, the goal that $\Diamond\Phi \lor \Diamond\Psi$ becomes the goal that $\Diamond\Psi$ if a makes achieving Φ impossible, etc. Also, it is possible to model introspection of goals with constraints on the accessibility relations G and K.

Unlike Cohen and Levesque, I handle both intended actions and (declarative) goals uniformly (an intended complex action σ can be represented using the goal that $\exists s.$ OnPath$(path, s) \land$Do(σ, now, s), i.e. that the goal accessible paths are such that there is a situation s on each of these paths that can be reached from the starting situation of this path by performing σ; see Chapter 7 for more on this). As in their account, intentions of agents in my framework also persist (e.g. see Proposition 4.4.24). Like their intentions, my chosen goals are also relativized, in particular w.r.t a matching $drop$ action. Put otherwise, under certain assumptions, an agent can drop an intention/primary chosen goal by performing the associated $drop$ action (see Proposition 4.4.15). Like Rao and Georgeff, I also use a branching time temporal logic. Also, my agents are both single-mindedly and open-mindedly committed to their intentions. Finally, unlike both Cohen and Levesque and Rao and Georgeff, I show how agents' intentions can evolve.

As discussed in Chapter 2, recently there have been a few proposals that deal with goal change. Shapiro et al. [200] present a situation calculus based framework where an agent adopts a goal when she is requested to do so by another agent, and remains committed to this goal unless the requester cancels this request; a goal is retained even if the agent learns that it has become impossible, and in this case the agent's goals become inconsistent. Shapiro and Brewka [196] modify this framework to ensure that goals are dropped when they are believed to be impossible or when they are achieved. Although developed independently, these accounts have commonalities to mine in the sense that they also assume a priority ordering over the set of (in their case, requested) goals, and in every situation they compute intentions by computing a maximal consistent goal set that is also compatible with the agent's beliefs. In their framework, goals are only partially ordered and inconsistencies between goals at the same level (given goals at higher levels and knowledge) can be resolved differently in different models. In fact, the agent's intentions in $do(a, s)$ in a model may be quite different from her intentions in s, although a did not make any of her goals in s impossible or inconsistent with higher priority goals, simply because the inconsistencies between goals at the same priority level are resolved differently in s and $do(a, s)$. This is rather unintuitive, in particular in the context of intentions. Recall that one of the defining properties of intentions is that an agent's intentions persist, unless they become impossible to bring about or have already been fulfilled. Note that, while one might argue that a partial order over goals might be more general, allowing this means that additional control information is required to obtain a single goal/intention state after the agent's goals change. In other words, the problem with a partial ordering is that it does not specify what a rational agent should do when two of her goals that have equal priority become inconsistent with each other. My account, on the other hand, imposes a total ordering on goals and thus I was able to ensure that under suitable conditions, an agent's intentions (i.e. chosen goals) persist. Also, I provide a more expressive formalization of prioritized goals: I model goals using infinite paths while they use finite paths, and thus can model many types of goals that Shapiro and Brewka cannot, as discussed in

Chapter 2. Finally, another difference between their framework and mine is that they specify the order on goals syntactically rather than semantically.

My goal dynamics is in line with da Costa Pereira et al.'s goal revision postulates for rational agents [43, 44, 42]. As mentioned in Chapter 2, these postulates are based on notions of consistency of sets of desires and executability of desires that seems problematic. In my framework, I specify executability using a formal action theory (including action precondition axioms), and I interpret consistency among a set of (achievement) goals as the existence of a path starting with the current situation over which all of these goals hold. With this interpretation, I think these postulates are in fact sound. A formal version of their I_1 postulate is shown to hold in my framework by Proposition 4.4.2 and Corollary 4.4.4. Note that I_2 seems problematic unless the ordering $\succeq$ over desires is total, which is the case for my framework. If a partial order is assumed, an agent might have several alternative sets of chosen goals, none of which is better than the others. I formalize and prove I_3 in Corollary 4.4.13. Proposition 4.4.12 shows that I_5 is partially satisfied in my framework (I didn't prove that $i \notin \mathcal{I}(S_d)$). Finally, I believe that I_4 and I_5 are both satisfied in my framework; proving this is left for future work.

While independently motivated, the notion of belief-goal consistency in my framework is similar to the one presented by Icard et al. [107]. Recall from Chapter 2 that intentions in their framework are simply (primitive) action and time index pairs. As mentioned earlier, their account seems problematic since while it ensures that there is a path over which all the intended actions are executed at the appropriate time index and the preconditions of these are satisfied before their execution, there is no guarantee that the path is in fact a realistic one. In other words, this path is allowed to include non-executable actions as long as they are not intended. However as discussed in Chapter 2, given a proper interpretation, many of the postulates they propose seem to be sound. A formal version of their BI_1 postulate is shown to hold in my framework by Proposition 4.4.2. I reject BI_2, as agents are supposed to be committed to their intentions; they may be allowed to give up their existing intentions for a new (and conflicting) one under certain circumstances, but not always. For instance, in my framework an agent can give up a primary c-goal ψ at some level m if she adopts another conflicting goal ϕ *at a higher priority level* n *than* m (Proposition 4.4.12 shows that the new goal ϕ is successfully adopted as long as it is consistent with all higher priority goals than n, and even if it is possibly inconsistent with some goal that has lower priority than n). However, it will not be rational to force the agent to abandon ψ simply because she adopted a new goal, e.g. when she adopts the conflicting goal ϕ at a lower priority level than m. Corollary 4.4.13 shows that BI_3 is partially satisfied in my framework. I believe the persistence property in Proposition 4.4.24 can be generalized to show that BI_3 is indeed satisfied for my framework. Again, I believe that BI_4 also can be shown to hold for primary c-goals (but not for chosen goals, as these are closed under consequence). I leave these for future work. The SSA for G in Axiom 4.3.3 already takes care of postulate BI_6, as an agent's goals here are updated (by progressing the G-accessible paths) even when the action performed is not an $adopt$ or $drop$ action. As we have seen, such updates to G may drop some chosen goals and add some other chosen goals (i.e. deactivate some levels and make others active, as prescribed by Axiom 4.2.7). Finally, as argued before, postulates BI_5 and BI_7 do not hold for my introspective agents since changes in their intentions also change their knowledge about their intentions.

To specify agents' goals, Baral and Zhao introduced their non-monotonic LTL logic called N-LTL [7] and its improved version ER-LTL [8]. Their objective in these papers is to enable the agent specifier to gradually specify the agent's goals in a manner that allows for partial goal retraction/update/revision, strengthening/weakening the conditions (in case of a conditional goal), and even complete change in the goal, i.e. specify goals in an elaboration tolerant manner [147]. Specifying goals in such a way is useful in many time critical domains where complete retraction of a previously specified goal followed by reformulation and re-delegation of an updated version can be costly. They showed how a non-monotonic ER-LTL program can be translated into a monotonic LTL specification. However, the authors do not handle goal dynamics, i.e. discuss how goals evolve when actions/events occur in these frameworks. In my framework, new goals can be adopted and existing goals can be dropped, and p-goals need not be consistent, while the agent constantly re-optimizes her c-goals; more analysis of the elaboration tolerance of my model is left for future work.

In van Benthem et al.'s [225, 226] dynamic epistemic preference logic, agents' preferences are "upgraded" in response to public announcement-like suggestions and commands. In their framework, a suggestion to prefer φ leads to a model change that removes preferences for $\neg\varphi$ over φ. Their notion of preference is quite abstract and covers vast areas including decision theory, optimality theory, game theory, conditional logic, non-monotonic logic, belief revision theory (whose semantics may involve preferences over possible worlds), etc. As argued, my prioritized goals can be used to model preferences. Moreover, I model goals and goal dynamics, i.e. how exactly these goals evolve as a result of regular actions as well as special actions (like $adopt$); they however do not address any of these issues (preferences in their framework can be results of agents' goals, which they don't define).

Arguably, one limitation of my account is that my agents waste resources trying to optimize their c-goals at every step. In Chapter 6, I propose an alternative semantics for specifying the prioritized goals of agents that eliminates the filter override mechanism altogether. That framework can be used to design agents that are very committed to their chosen goals. In the future, I would like to develop a hybrid account where the agent is strongly committed to her chosen goals, and where the filter override mechanism is only triggered under appropriate conditions. Also, it would be interesting to identify a set of postulates for goal change that most people can agree on and examine how they differ from belief change postulates.

Chapter 5

Handling Subgoals

5.1 Introduction

Any proper formalization of goals and their dynamics must also capture the dependencies between goals and subgoals and plans adopted as a means to achieve these goals. In particular, the dynamics of (sub)goals should conform to the following set of intuitive rules:

- The subgoals and plans adopted as means to bring about a goal should be dropped if the parent goal becomes impossible.
- They should also be dropped when the parent goal is brought about fortuitously/in an unexpected way before the subgoals are achieved.
- Again, the abandonment/dropping of a parent goal should remove all of its sub-

goals from the goal hierarchy.

- In contrast to this, the dropping of a subgoal should be independent of that of its parent goal – the agent might realize that one of the subgoals ψ of a goal ϕ has become impossible, and thus might want to drop ψ; however, she may still want to keep ϕ as her goal as there might be other means to fulfill ϕ known to her.

- Finally, these constraints on subgoal dynamics should carry over to subgoals of subgoals of a goal, and thus e.g., if ψ_3 is a subgoal of ψ_2, and ψ_2 is a subgoal of ψ_1, then the dropping of ψ_1 should trigger the dropping of both ψ_2 and ψ_3 from the agent's goal hierarchy.

In this chapter, I introduce a new action for adopting a subgoal with respect to a parent goal. I then extend the successor-state axiom proposed in Chapter 4 to deal with subgoal adoption, and discuss how subgoals change when an agent's knowledge changes as a result of the execution of some (possibly exogenous) action or when she adopts or drops a (sub)goal. I also give a definition of subgoals. After that, I prove some properties; in particular, I show that the proposed formalization of subgoal dynamics ensures that adopting a subgoal with respect to a supergoal has the desired effects, that dropping a supergoal always drops all its subgoals but not necessarily vice versa, and that the subgoal relation is transitive. Finally, returning to our discussion from Chapter 4, I explain why existing belief revision type approaches are not suitable for modeling this relationship between goals and their subgoals.

5.2 Subgoal Dynamics

In this section, I discuss how the framework presented in Chapter 4 can be extended to handle subgoals. To this end, I first introduce an action $adoptRelTo(\psi, \phi)$ for adopting a subgoal ψ relative to a supergoal ϕ. The action precondition axiom for this is as follows:

Axiom 5.2.1.

$$\text{Poss}(adoptRelTo(\psi, \phi), s) \equiv \exists m.\ \text{PGoal}(\phi, m, s) \wedge \neg\exists n.\ \text{PGoal}(\psi, n, s).$$

That is, an agent can adopt a subgoal ψ with respect to a parent goal ϕ in situation s if at some level in s, she has the p-goal that ϕ, and if she does not already have the p-goal that ψ at some level in s. I assume that unique names axioms as in axiom schema 3.3.2 and 3.3.3 are available for $adoptRelTo$.

An agent's subgoals must be dropped when the corresponding parent goal is dropped or becomes impossible. When adopting a subgoal ψ with respect to a supergoal ϕ, in addition to recording the newly adopted goal ψ, we need to model the fact that ψ is a subgoal of ϕ. This information can later be used to drop the subgoal when the parent goal is dropped. One way of modeling this is to ensure that the adoption of a subgoal ψ with respect to a parent goal ϕ adds a new p-goal that contains *both this subgoal and this parent goal, i.e.* $\psi \wedge \phi$. Recall from Chapter 4 that to handle the dropping of a goal ϕ, I require the agent to drop the p-goals at all G-accessibility levels that imply ϕ. Thus, if the agent drops the parent goal ϕ, she will also drop all of its subgoals including ψ, since the G-accessibility levels where the parent goal ϕ holds include the G-accessibility levels where the subgoal ψ holds.

Note that the parent goal ϕ could be a p-goal at multiple levels. I assume that the subgoal ψ is always adopted with respect to the *highest priority supergoal level*, i.e. the highest priority level where ϕ holds. I think that this is reasonable and works in most cases. However, it may be argued that the subgoal ψ should be adopted with respect to all the levels where the supergoal ϕ hold. But this makes subgoal dynamics quite complex to follow. Moreover, if the agent wants to adopt a subgoal ψ with respect to a supergoal ϕ at a different level (say m) than the highest priority level where ϕ holds (say n, where $n < m$), she could do this by adopting ψ with respect to the more specific supergoal (than ϕ) at level m. Nevertheless, when the only p-goal at level n implies the one at level m, we have a problem. But this can also be avoided by using the $adopt$ action instead, and adopting the goal that $\phi \wedge \psi$ at level $m + 1$.[42] Another

[42]This brings up the question of whether one should define $adoptRelTo$ in terms of $adopt$; while I think that this could be done, this would make it harder to determine, given the action history, whether

possible solution is to redefine $adoptRelTo$ to take an argument for the level of the intended supergoal, which I leave for future work. I also assume that the subgoal ψ is always adopted at the level immediately below the supergoal ϕ's level, i.e. at level $n+1$, where n is the highest level where the supergoal ϕ holds. The reason for doing this is that since ψ is a means to the end ϕ, they should have similar priorities. While I think this is reasonable, this could also be generalized if necessary.

To handle subgoals, I add an additional case to the successor-state axiom for G (thus I replace Axiom 4.3.3 in Chapter 4 with the following axiom):

Axiom 5.2.2 (SSA for G)**.**

$$
\begin{aligned}
&G(p,n,do(a,s)) \equiv\\
&\qquad\qquad \forall\phi,\psi.\ (a \neq adopt(\phi) \wedge a \neq adoptRelTo(\psi,\phi) \wedge a \neq drop(\phi) \wedge\\
&\qquad\qquad\qquad\qquad \text{Progressed}(p,n,a,s))\\
&\qquad\qquad \vee\ \exists\phi,m.\ (a = adopt(\phi,m) \wedge \text{Adopted}(p,n,m,a,s,\phi))\\
&\qquad\qquad \vee\ \exists\phi,\psi.\ (a = adoptRelTo(\psi,\phi) \wedge \text{SubGoalAdopted}(p,n,a,s,\psi,\phi))\\
&\qquad\qquad \vee\ \exists\phi.\ (a = drop(\phi) \wedge \text{Dropped}(p,n,a,s,\phi)).
\end{aligned}
$$

The part of the SSA for G that handles subgoal adoption is defined as follows:

a goal is adopted relative to some parent goal or unconditionally adopted. Thus for simplicity, here I introduce $adoptRelTo$ as a primitive action.

Definition 5.2.3.

$$\text{SubGoalAdopted}(p, n, a, s, \psi, \phi) \overset{\text{def}}{=}$$

$$\textbf{if } (\text{AdoptedLevel}(\phi, m, s) \wedge n < m) \textbf{ then } \text{Progressed}(p, n, a, s)$$

$$\textbf{else if } (\text{AdoptedLevel}(\phi, n, s)) \textbf{ then } (\text{Progressed}(p, n-1, a, s) \wedge \psi(p))$$

$$\textbf{else } \text{Progressed}(p, n-1, a, s),$$

where,

Definition 5.2.4.

$$\text{AdoptedLevel}(\phi, n, s) \overset{\text{def}}{=} \text{PGoal}(\phi, n-1, s) \wedge \forall m.\ m < n-1 \supset \neg\text{PGoal}(\phi, m, s).$$

Thus, to handle the adoption of a subgoal ψ with respect to a supergoal ϕ, I add a new p-goal $\phi \wedge \psi$ to the agent's goal hierarchy. My formalization of this uses the abbreviation AdoptedLevel(ϕ, n, s), which says that n is the level where the subgoal should be adopted, that is, the level which is immediately below the highest priority level such that the parent goal ϕ is implied by the only p-goal at this level. The G-accessible paths at all levels above the slot where the subgoal is to be inserted, i.e. m, such that AdoptedLevel(ϕ, m, s) holds, are simply progressed. The G-accessible paths at level m are the ones that can be obtained by progressing the paths from the level immediately above it (i.e. from the highest priority level where ϕ holds) over which ψ holds and eliminating those where ψ does not hold. This guarantees that $\phi \wedge \psi$ holds in this level, and thus as discussed above, the subgoal ψ will be dropped when the supergoal ϕ is dropped. The G-accessible paths at all levels below m are the ones that can be obtained by progressing paths from the level immediately above it. Thus the agent acquires the subgoal that ψ at level m, and all the p-goals with priority m or less in s are pushed down one level in the hierarchy.

Let me give an example of subgoal dynamics; for this I extend the running example of Chapter 4. Assume that we have an agent who initially has the following three p-goals: $\square$BeRich, $\Diamond$GetPhD, and $\square$BeHappy at level 0, 1, and 2, respectively (see second column of Table 5.1). Suppose that the agent knows that one way of always being rich is to always work hard, which in turns can be fulfilled by always being energetic. Assume that with this in mind, our agent adopts the subgoal that $\square$WorkHard with respect to the p-goal that $\square$BeRich, and then adopts the sub-subgoal that $\square$BeEnrg with respect to the subgoal that $\square$WorkHard, starting in S_0. Then the agent's goal hierarchy in $S_1 = do(adoptRelTo(\square\text{WorkHard}, \square\text{BeRich}), S_0)$ should include the p-goal that $\square$WorkHard, and in $S_2 = do(adoptRelTo(\square\text{BeEnrg}, \square\text{WorkHard}), S_1)$ should also include the p-goal that $\square$BeEnrg. According to the successor-state axiom for G, our agent's goal hierarchy in S_1 and in S_2 will be as in Table 5.1.[43] In S_0, the supergoal

[43]For simplicity in Table 5.1, I only show the agent's relevant p-goals rather than her only p-goals (which in addition reflect the actions that have been performed so far).

G-Level	S_0	S_1	S_2	S_3
0	□BeRich	□BeRich	□BeRich	□BeRich
1	◇GetPhD	□BeRich ∧ □WorkHard	□BeRich ∧ □WorkHard	TRUE
2	□BeHappy	◇GetPhD	□BeRich ∧ □WorkHard ∧ □BeEnrg	TRUE
3	TRUE	□BeHappy	◇GetPhD	◇GetPhD
4	TRUE	TRUE	□BeHappy	□BeHappy

Table 5.1: Example of an Agent's Subgoal Dynamics

□BeRich holds at level 0, and thus $\text{AdoptedLevel}(\Box\text{BeRich}, 1, S_0)$ holds, i.e. the agent adopts □WorkHard at priority level 1. Similarly in S_1, the supergoal □WorkHard holds at level 1, and thus $\text{AdoptedLevel}(\Box\text{WorkHard}, 2, S_1)$ holds.

Now, suppose that in S_2, the agent wants to drop the p-goal that □WorkHard. Then in $S_3 = do(drop(\Box\text{WorkHard}), S_2)$, she should no longer have □BeEnrg as a p-goal, but should retain the supergoal that □BeRich. After the agent drops the p-goal that □WorkHard, by the successor-state axiom for G we can see that all the G-accessible levels where □WorkHard holds will be replaced by the trivial only p-goal that the correct history of actions in S_3 has happened (see S_3 in Table 5.1). This shows that dropping □WorkHard results in the dropping of all of its subgoals (in this case □BeEnrg), but that its parent goal □BeRich is retained.

I define the SubGoal relation as follows:

Definition 5.2.5.

$$\text{SubGoal}(\psi, \phi, s) \stackrel{\text{def}}{=} \exists n.\ \text{PGoal}(\phi, n, s) \land \neg\text{PGoal}(\psi, n, s)$$
$$\land \forall m.\ \text{PGoal}(\psi, m, s) \supset \text{PGoal}(\phi, m, s) \land n < m.$$

This says that ψ is a subgoal of ϕ in situation s iff there exists a G-accessibility level n in s such that ϕ is a p-goal at n while ψ is not, and for all G-accessibility levels m in s where ψ is a p-goal, ϕ is also a p-goal and n has higher priority than m, i.e. ψ is not a p-goal at any level higher than n. The intuition behind my notion of subgoals is that it corresponds to the cases where there is some level in the goal hierarchy where the supergoal holds, and the conjunction of the supergoal and the subgoal holds at another lower priority level. Thus the relationship between goals and subgoals is viewed as a special case of prioritized goals. As discussed above, I use the conjunction $\phi \land \psi$ to facilitate the dropping of the subgoal ψ when the supergoal ϕ is dropped. Also, since ψ is a means to the end ϕ, the supergoal ϕ should have higher priority than the subgoal ψ. The above definition captures part of this intuition.

Note that a consequence of this definition is that the same subgoal cannot have two different supergoals (unless one subsumes the other or by transitivity, which I discuss later). This does not pose a problem however, since the preconditions of the $adoptRelTo(\psi, \phi)$ action requires that the agent does not already have the subgoal ψ as a p-goal at some level (and thus, as a subgoal of another parent goal). Thus an agent cannot attempt to adopt a subgoal relative to two different parent goals.

5.3 Properties

I now show that my formalization of subgoals has some desirable properties. Let's define $\mathcal{D}^{SG}_{OAgt}$ as $\mathcal{D}_{OAgt} \backslash \{\text{Axiom 4.3.3}\} \cup \{\text{Axiom 5.2.1, Axiom 5.2.2}\}$ (as well as the associated definitions). First, I can prove that an agent acquires the p-goal that ψ after she adopts it as a subgoal of another goal ϕ in s, provided that she has the p-goal at some level in s that ϕ (which is required for $adoptRelTo(\psi, \phi)$ to be executable in s):

Proposition 5.3.1 (Subgoal Adoption-1)**.**

$$\mathcal{D}^{SG}_{OAgt} \models \exists m.\ \text{PGoal}(\phi, m, s) \supset \exists n.\ \text{PGoal}(\psi, n, do(adoptRelTo(\psi, \phi), s)).$$

Proof. Fix ϕ_1, ψ_1, M_1, and S_1. From the antecedent, we have:

$$\text{PGoal}(\phi_1, M_1, S_1). \tag{5.1}$$

Fix N_1 such that AdoptedLevel(ϕ_1, N_1, S_1) holds. By (5.1) and Definition 5.2.4, such a level N_1 indeed exists. By Axiom 5.2.2 and Definitions 5.2.3, 4.3.4, and 5.2.4, the agent's G-accessible paths in $do(adoptRelTo(\psi_1, \phi_1), S_1)$ at level N_1 are the ones that can be obtained by progressing her G-accessible paths at $N_1 - 1$ in S_1, and over which ψ_1 holds; thus we have:

$$\forall p.\ G(p, N_1, do(adoptRelTo(\psi_1, \phi_1), S_1)) \supset \psi_1(p). \tag{5.2}$$

If such a path exists, then the consequent follows from (5.2) and Definition 4.2.1. Otherwise, the consequent follows trivially by Definition 4.2.1. □

Secondly, I can show that if an agent has the primary c-goal that ϕ at level $n-1$ in situation s, then she acquires the primary c-goal that ψ at level n after she adopts it as a subgoal of ϕ at n in s, provided that she does not have the c-goal at $n-1$ in s that $\neg\psi$ next:

Proposition 5.3.2 (Subgoal Adoption-2)**.**

$$\begin{aligned}\mathcal{D}^{SG}_{OAgt} \models\ & \text{PrimCGoal}(\phi, n-1, s) \wedge \text{AdoptedLevel}(\phi, n, s)\\ & \wedge \neg\text{CGoal}(\neg\exists s', p'.\ \text{Starts}(s') \wedge\\ & \qquad \text{Suffix}(p', do(adoptRelTo(\psi,\phi), s')) \wedge \psi(p'), n-1, s)\\ & \supset \text{PrimCGoal}(\psi, n, do(adoptRelTo(\psi,\phi), s)).\end{aligned}$$

Proof. Fix ϕ_1, ψ_1, N_1, and S_1. From the antecedent, we have:

$$\text{PrimCGoal}(\phi_1, N_1-1, S_1), \tag{5.3}$$

$$\text{AdoptedLevel}(\phi_1, N_1, S_1), \tag{5.4}$$

$$\begin{aligned}&\neg\text{CGoal}(\neg\exists s', p'.\ \text{Starts}(s') \wedge\\ &\qquad \text{Suffix}(p', do(adoptRelTo(\psi_1,\phi_1), s')) \wedge \psi_1(p'), N_1-1, S_1).\end{aligned} \tag{5.5}$$

By (5.4), Axiom 5.2.2, and Definitions 5.2.3, 4.3.4, and 5.2.4, the agent's G-accessible paths in $do(adoptRelTo(\psi_1, \phi_1), S_1)$ at N_1 are the ones that can be obtained by progressing her G-accessible paths at $N_1 - 1$ in S_1, and over which ψ_1 holds:

$$\begin{aligned} &\forall p.\ G(p, N_1, do(adoptRelTo(\psi_1, \phi_1), S_1)) \equiv \\ &\qquad \text{Progressed}(p, N_1 - 1, adoptRelTo(\psi_1, \phi_1), S_1) \wedge \psi_1(p). \end{aligned} \tag{5.6}$$

Regardless of whether such a path exists, it follows from (5.6) and Definition 4.2.1 that:

$$\text{PGoal}(\psi_1, N_1, do(adoptRelTo(\psi_1, \phi_1), S_1)). \tag{5.7}$$

From (5.5) and Definition 4.2.8, it follows that there is a path, say P_1, such that P_1 is in the prioritized intersection of G_R-accessible paths up to level $N_1 - 1$ in S_1, that the $adoptRelTo(\psi_1, \phi_1)$ action happens next along P_1, and that ψ_1 holds over P_1 afterwards:

$$\begin{aligned} &G_\cap(P_1, N_1 - 1, S_1) \wedge \\ &\exists s', p'.\ \text{Starts}(P_1, s') \wedge \text{Suffix}(p', P_1, do(adoptRelTo(\psi_1, \phi_1), s')) \wedge \psi_1(p'). \end{aligned} \tag{5.8}$$

Now, since $G_\cap(P_1, N_1 - 1, S_1)$, by Lemma 4.4.10 P_1 must start with a K-accessible situation in S_1, i.e.:

$$\forall s.\ \text{Starts}(P_1, s) \supset K(s, S_1). \tag{5.9}$$

Also, note that by (5.8), P_1 is a path and the first action that happens along P_1, i.e. in the starting situation of P_1, is $adoptRelTo(\psi_1, \phi_1)$. From this, Corollary 3.5.41, and

Definition 3.3.1 it follows that:

$$\forall s.\ \text{Starts}(P_1, s) \supset \text{Poss}(\mathit{adoptRelTo}(\psi_1, \phi_1), s). \tag{5.10}$$

Consider the suffix of P_1 after the $\mathit{adoptRelTo}(\psi_1, \phi_1)$ action has been performed; let us call this path P_2. Since the $\mathit{adoptRelTo}$ action is not a knowledge-producing action, by (5.9), (5.10), and Axiom 3.4.10 it follows that:

$$\forall s.\ \text{Starts}(P_2, s) \supset K(s, \mathit{do}(\mathit{adoptRelTo}(\psi_1, \phi_1), S_1)). \tag{5.11}$$

By (5.3), Definition 4.2.12, Axiom 4.2.7, and (5.8), it follows that:

$$G_R(P_1, N_1 - 1, S_1). \tag{5.12}$$

By (5.6), (5.12), (5.8), and Definition 4.2.4, it follows that:

$$G(P_2, N_1, \mathit{do}(\mathit{adoptRelTo}(\psi_1, \phi_1), S_1)). \tag{5.13}$$

Again, by (5.13), (5.11), and Definition 4.2.4, it follows that P_2 must be G_R-accessible at N_1 in $\mathit{do}(\mathit{adoptRelTo}(\psi_1, \phi_1), S_1)$:

$$G_R(P_2, N_1, \mathit{do}(\mathit{adoptRelTo}(\psi_1, \phi_1), S_1)). \tag{5.14}$$

Now, I claim that all levels with priority higher than N_1 that are active before the occurrence of the $\mathit{adoptRelTo}(\psi_1, \phi_1)$ action will remain active.[44] To see this,

[44]Recall from Definition 4.4.1 that $\mathit{ActiveLevel}(n, s)$ is defined as $\exists p.\ G(p, n, s) \wedge G_{\cap}(p, n, s)$.

note that the only goal that can become impossible merely by the occurrence of this $adoptRelTo$ action is the goal that states that this action does not happen next, as well as any consequences of this. However, by (5.5) no levels in the goal hierarchy that have such a goal and that have priority higher than N_1 are active in S_1. Put otherwise, by (5.8) and Axiom 4.2.7, there is a path, namely P_1, that is G_R-accessible at all active levels n with priority higher than N_1 in S_1; thus:

$$\forall n.\ n < N_1 \land \text{ActiveLevel}(n, S_1) \supset G_R(P_1, n, S_1). \tag{5.15}$$

Note that, the next action that happens along P_1 is the $adoptRelTo(\psi_1, \phi_1)$ action. Moreover, according to Axiom 5.2.2 and Definitions 5.2.3 and 4.3.4, after the $adoptRel$-$To(\psi_1, \phi_1)$ action happens, the G_R-accessible paths at all levels that have higher priority than N_1 are simply progressed to reflect the fact that this action has happened:

$$\begin{aligned}\forall n.\ n < N_1 \supset (\forall p.\ G(p, n, do(adoptRelTo(\psi_1, \phi_1), S_1)) \equiv \\ \text{Progressed}(p, n, adoptRelTo(\psi_1, \phi_1), S_1)).\end{aligned} \tag{5.16}$$

Then by this, (5.8), and (5.15), all active levels in S_1 that have priority higher than N_1 must have at least one path that can be progressed, namely P_1, and thus any such level that was active in S_1 must also be active in $do(adoptRelTo(\psi_1, \phi_1), S_1)$.

Moreover, I claim that since no active level n with higher priority than N_1 in S_1 became inactive due to the occurrence of $adoptRelTo(\psi_1, \phi_1)$, no inactive level with higher priority than N_1 in S_1 can become active in $do(adoptRelTo(\psi_1, \phi_1), S_1)$. I will prove this by contradiction. Assume that there is a level $M < N_1$ s.t. M is inactive in S_1 and active in $do(adopt(\psi_1, \phi_1), S_1)$. Then, from Definition 4.4.1, it follows that there is a path P_b s.t.:

$$G(P_b, M, do(adoptRelTo(\psi_1, \phi_1), S_1)), \text{ and} \tag{5.17}$$

$$G_\cap(P_b, M, do(adoptRelTo(\psi_1, \phi_1), S_1)). \tag{5.18}$$

From (5.18), Definition 4.4.1, and Axiom 4.2.7, P_b must be G_R-accessible at all levels n in $do(adoptRelTo(\psi_1, \phi_1), S_1)$, where $n < M$ and n is active in S_1 (since, recall that all these levels remain active after the $adoptRelTo(\psi_1, \phi_1)$ action happens). By this, (5.17), Definition 4.2.4, (5.16), and Definition 4.3.4, there is a path P_a that starts with some situation S_{P_a} s.t. P_b is the suffix of P_a starting in $do(adoptRelTo(\psi_1, \phi_1), S_{P_a})$ and P_a is G-accessible at all active levels with higher priority than M in S_1 and at M in S_1. Moreover, by this, (5.18), Lemma 4.4.10 and the SSA for K (i.e. Axiom 3.4.10), it follows that P_a is G_R-accessible at all active levels with higher priority than M in S_1 and at M in S_1. But then, by Axiom 4.2.7 and Definition 4.4.1, M is an active level in S_1, a contradiction. Thus, it follows that:

$$\forall n.\ n < N_1 \supset (\text{ActiveLevel}(n, S_1) \equiv \text{ActiveLevel}(n, do(adoptRelTo(\psi_1, \phi_1), S_1))).$$

Furthermore, by this, (5.8), (5.15), and (5.16), P_2 must be G-accessible from all these active levels in $do(adoptRelTo(\psi_1, \phi_1), S_1)$:

$$\forall n.\ n < N_1 \wedge \text{ActiveLevel}(n, do(adoptRelTo(\psi_1, \phi_1), S_1) \supset$$
$$G(P_2, n, do(adoptRelTo(\psi_1, \phi_1), S_1)).$$

From this, (5.11), and Definition 4.2.4, P_2 must be G_R-accessible from all these active levels in $do(adoptRelTo(\psi_1, \phi_1), S_1)$:

$$\forall n.\ n < N_1 \wedge \text{ActiveLevel}(n, do(adoptRelTo(\psi_1, \phi_1), S_1) \supset$$
$$G_R(P_2, n, do(adoptRelTo(\psi_1, \phi_1), S_1)).$$

Finally, by this, (5.14), Definition 4.4.1, and Axiom 4.2.7, it follows that:

$$G_\cap(P_2, N_1, do(adoptRelTo(\psi_1, \phi_1), S_1)). \tag{5.19}$$

The consequent follows from (5.7), (5.14), (5.19), and Definition 4.2.12. □

I can also show that subgoals are dropped when their parent goal is dropped. More precisely, I show that after dropping the p-goal that ϕ in s, an agent does not have the p-goal (and thus the primary c-goal) that the progression of ψ at level n, provided that ψ is a subgoal of ϕ in s, that ψ is a p-goal at n in s, and that the progression of ψ is not strongly inevitable in $do(drop(\phi), s)$:

Proposition 5.3.3 (Supergoal Drop)**.**

$$\mathcal{D}^{SG}_{OAgt} \models \text{SubGoal}(\psi, \phi, s) \wedge \text{PGoal}(\psi, n, s)$$

$$\wedge \neg\text{StronglyInevitable}(\text{ProgOf}(\psi, drop(\phi)), do(drop(\phi), s))$$

$$\supset \neg\text{PGoal}(\text{ProgOf}(\psi, drop(\phi)), n, do(drop(\phi), s)).$$

Proof. Fix ϕ_1, ψ_1, N_1 and S_1. From the antecedent, we have:

$$\text{SubGoal}(\psi_1, \phi_1, S_1), \tag{5.20}$$

$$\text{PGoal}(\psi_1, N_1, S_1), \tag{5.21}$$

$$\neg\text{StronglyInevitable}(\text{ProgOf}(\psi_1, drop(\phi_1)), do(drop(\phi_1), S_1)). \tag{5.22}$$

From (5.20) and Definition 5.2.5, it follows that for any level n in S_1 where ψ_1 is a p-goal, ϕ_1 is also a p-goal:

$$\forall n.\ \text{PGoal}(\psi_1, n, S_1) \supset \text{PGoal}(\phi_1, n, S_1). \tag{5.23}$$

From (5.22) and Definitions 3.5.13 and 3.5.12, it follows that there is a path P_1 such that:

$$\exists s.\ \text{SameHist}(s, do(drop(\phi_1), S_1)) \wedge \text{Starts}(P_1, s) \wedge \neg\text{ProgOf}(\psi_1, drop(\phi_1))(P_1). \tag{5.24}$$

Now, consider level N_1 in S_1; by (5.21), ψ_1 is a p-goal at N_1 in S_1. By this and (5.23), PGoal(ϕ_1, N_1, S_1), and thus by Axiom 5.2.2, and Definition 4.3.6, we can see that the

G-accessible paths at N_1 in $do(drop(\phi_1), S_1)$ are the ones that start with situations that share the same action history as $do(drop(\phi_1), S_1)$. Since by (5.24), P_1 is such a path, it will be included in the G-relation at N_1 in $do(drop(\phi_1), S_1)$:

$$G(P_1, N_1, do(drop(\phi_1), S_1)). \tag{5.25}$$

The consequent follows from (5.24), (5.25), and Definition 4.2.1. $\square$

Note that, this does not hold if we replace PGoal in the consequent with CGoal since ψ could be a consequence of a combination of other active p-goals, i.e. a non-primary c-goal. Also, this property can be generalized to show that in addition to the above, ψ is indeed not a p-goal at some level n where $\neg$PGoal(ψ, n, s) after the $drop(\phi)$ action happens in s, provided that she don't have the p-goal at n in s that the $drop(\phi)$ action does not happen next or ψ holds after it happens, i.e. that $\neg$PGoal$(\neg\exists s'.\ \text{Do}(drop(\phi), now, s') \vee \psi, n, s)$.

The next property says that dropping a subgoal does not affect the parent goal. In particular, an agent retains the p-goal that the progression of ϕ after she drops a subgoal ψ of some goal ϕ in some situation s:

Proposition 5.3.4 (Subgoal Drop)**.**

$$\begin{aligned}\mathcal{D}^{SG}_{OAgt} \models\ & \text{SubGoal}(\psi, \phi, s) \\ & \supset \exists n.\ \text{PGoal}(\text{ProgOf}(\phi, drop(\psi)), n, do(drop(\psi), s)).\end{aligned}$$

Proof. Fix ϕ_1, ψ_1, and S_1. From the antecedent, we have:

$$\text{SubGoal}(\psi_1, \phi_1, S_1). \tag{5.26}$$

From (5.26) and Definition 5.2.5, it follows that there is a level N_1 in S_1 where ϕ_1 is a p-goal but ψ_1 is not:

$$\text{PGoal}(\phi_1, N_1, S_1) \wedge \neg\text{PGoal}(\psi_1, N_1, S_1). \tag{5.27}$$

Now, by (5.27), Axiom 5.2.2, and Definitions 4.3.6 and 4.3.4, we can see that after the $drop(\psi_1)$ action has been performed in S_1, the agent's G-accessible paths at level N_1 will be the ones that can be obtained by progressing her G-accessible paths at N_1 in S_1; thus from this, (5.27), and Definitions 4.2.1 and 4.4.14, we have:

$$\forall p.\ G(p, N_1, do(drop(\psi_1), S_1)) \supset \text{ProgOf}(\phi_1, drop(\psi_1))(p). \tag{5.28}$$

The consequent follows from (5.28) and Definition 4.2.1. □

I next show that adopting logically equivalent subgoals with respect to logically equivalent supergoals has the same result:

Proposition 5.3.5 (Extensionality w.r.t. Subgoal Adoption)**.**

(a). $\mathcal{D}_{OAgt}^{SG} \models \forall p.(\phi_1(p) \equiv \phi_2(p)) \wedge \forall p.(\psi_1(p) \equiv \psi_2(p)) \supset$

$$\text{PGoal}(\phi^*, do(adoptRelTo(\psi_1, \phi_1), s)) \equiv \text{PGoal}(\phi^*, do(adoptRelTo(\psi_2, \phi_2), s)),$$

(b). $\mathcal{D}_{OAgt}^{SG} \models \forall p.(\phi_1(p) \equiv \phi_2(p)) \wedge \forall p.(\psi_1(p) \equiv \psi_2(p)) \supset$

$$\text{CGoal}(\phi^*, do(adoptRelTo(\psi_1, \phi_1), s)) \equiv \text{CGoal}(\phi^*, do(adoptRelTo(\psi_2, \phi_2), s)).$$

Proof. Similar to that of Proposition 4.4.7. □

As a consequence, this property also holds for primary c-goals:

Corollary 5.3.6.

$$\mathcal{D}_{OAgt}^{SG} \models \forall p.(\phi_1(p) \equiv \phi_2(p)) \wedge \forall p.(\psi_1(p) \equiv \psi_2(p)) \supset$$
$$(\text{PrimCGoal}(\phi^*, do(adoptRelTo(\psi_1, \phi_1), s))$$
$$\equiv \text{PrimCGoal}(\phi^*, do(adoptRelTo(\psi_2, \phi_2), s))).$$

Proof. Follows from Definition 4.2.12, Proposition 5.3.5(a), the SSA for G (i.e. Axiom 5.2.2 and Definitions 5.2.3, 5.2.4, and 4.3.4), and Axiom 4.2.7. □

Finally, I examine the properties of the SubGoal relation. To this end, I first show that the SubGoal relation is irreflexive/strict:

Proposition 5.3.7 (Irreflexivity of Subgoals)**.**

$$\mathcal{D}_{OAgt}^{SG} \models \forall s.\ \neg\text{SubGoal}(\psi, \psi, s).$$

Proof. Trivially follows from Definition 5.2.5. □

Secondly, I show that the SubGoal relation is antisymmetric, i.e. if a goal ψ is a subgoal of another goal ϕ in situation s, then ϕ cannot also be a subgoal of ψ in s:

Proposition 5.3.8 (Antisymmetry of Subgoals)**.**

$$\mathcal{D}_{OAgt}^{SG} \models \forall s.\ \text{SubGoal}(\psi, \phi, s) \supset \neg\text{SubGoal}(\phi, \psi, s).$$

Proof. Trivially follows from Definition 5.2.5. □

Thirdly, it can be shown that the SubGoal relation is transitive, i.e. if ψ_1 is a subgoal of ψ_2 in s, and if ψ_2 is a subgoal of ψ_3 in s, then ψ_1 must also be a subgoal of ψ_3 in s:

Proposition 5.3.9 (Transitivity of Subgoals)**.**

$$\mathcal{D}_{OAgt}^{SG} \models \forall s.\ \text{SubGoal}(\psi_1, \psi_2, s) \wedge \text{SubGoal}(\psi_2, \psi_3, s) \supset \text{SubGoal}(\psi_1, \psi_3, s).$$

Proof. Follows from Definition 5.2.5. □

Finally, as a consequence of Propositions 5.3.9 and 5.3.8, it can be shown that the SubGoal relation is acyclic, since any transitive asymmetric binary relation is acyclic. In other words, it can be shown that for any situation s, if there exists a finite set of distinct formulae $\psi_1, \psi_2, \cdots, \psi_k$ such that $k > 1$ and $\text{SubGoal}(\psi_i, \psi_{i+1}, s)$ for all

$i \in \{1, 2, \cdots, k-1\}$, then it must be the case that $\neg\text{SubGoal}(\psi_k, \psi_1, s)$. Thus the following axiom schema can be shown to be entailed by theory $\mathcal{D}^{SG}_{OAgt}$:

$$(k > 1 \wedge \text{SubGoal}(\psi_1, \psi_2, s) \wedge \cdots \wedge \text{SubGoal}(\psi_{k-1}, \psi_k, s)) \supset \neg\text{SubGoal}(\psi_k, \psi_1, s).$$

While I did not prove persistence properties for supergoals, for achievement supergoals much follows from the ones in the previous chapter (see Section 4.4.4), since achievement supergoals are simply ordinary prioritized achievement goals in the goal hierarchy. Also, persistence of realistic prioritized achievement subgoals seem to hold just as in Proposition 4.4.22. On the other hand, to show persistence of chosen achievement subgoals, we need stronger conditions in the antecedent of Proposition 4.4.24 since subgoals are affected by their supergoals in the sense that levels with subgoals will become inactive if the supergoal becomes impossible (recall from Definition 5.2.5 that the supergoal follows from all levels where the subgoal is a p-goal). Finally, it would be interesting to identify the conditions under which the SubGoal relation persists. I leave these for future work.

5.4 Conclusion

I introduced the $adoptRelTo$ action that can be used to adopt a subgoal relative to a supergoal. I then discussed how we can modify the successor-state axiom for G

in Chapter 4 to handle subgoal adoption. I also proposed a definition of the subgoal relation. Finally, I proved some intuitive properties of subgoal dynamics that are in line with the requirements specified in Section 5.1. The proposed subgoal dynamics ensures that subgoals are automatically dropped when their parent goals become impossible or are dropped, but not vice versa. Again, this notion carries over hierarchies of subgoals in the sense that, for example, a subgoal is also dropped when one of its ancestor goals (w.r.t. the SubGoal relation) becomes impossible or is dropped, but not vice versa.

Note that, while my formalization of subgoal dynamics satisfies most of the requirements discussed in Section 5.1, here I do not handle fortuitous achievement of parent goals. It would be nice to modify the proposed account to handle early achievement of goals by automatically dropping subgoals whose parent goal has been fulfilled. However, this seems quite challenging to do for arbitrary temporally extended goals. The special case of achievement goals should be solvable with some ingenuity. One idea to this end is as follows: instead of adopting the goal that $(\Diamond\Phi \wedge \Diamond\Psi)$ in response to the $adoptRelTo(\Diamond\Psi, \Diamond\Phi)$ action, the agent could adopt the goal that:

$$(\Diamond\Phi \wedge \Diamond\Psi \wedge \neg(\Phi \;\mathcal{B}\; \Psi)),$$

which states that the supergoal Φ must be eventually achieved, and so must be the subgoal Ψ, but the supergoal Φ must not be achieved before the subgoal Ψ.[45] Note that this also allows the case where the supergoal and the subgoal are achieved simultaneously. Thus, if the parent goal Φ is achieved before the subgoal Ψ has been achieved, it will violate the $\neg(\Phi \mathcal{B} \Psi)$ condition and thus the agent will make this goal inactive. As for the general case when arbitrary temporally extended (sub)goals are involved, it seems that one first needs to identify the satisfaction conditions of such goals, and then do a case-by-case analysis and handle each type of goal separately. Making these ideas precise is left for future work.

Note that while one might be tempted to try to employ existing belief revision type approaches to model prioritized goals, subgoals, and their dynamics, this does not work. Modeling prioritized goals using an ordinal conditional function κ (i.e. κ-ranking) and specifying goal dynamics as revising this function appropriately (e.g., as formalized for belief revision originally by Spohn [217] and then by Darwiche and Pearl [46]) seems to work, but only when subgoals are not considered. When one adds the non-trivial relationship between goals and subgoals to the equation, such a formalization produces unintuitive results. For instance, one way of specifying that ψ is a subgoal of ϕ is to ensure that the worlds where $\phi \wedge \psi$ hold are most preferred and thus are assigned a small ordinal (e.g. 0), that the worlds where the supergoal ϕ hold

[45]Let's call this formula *the satisfaction condition of the subgoal $\Diamond\Psi$ relative to the parent goal $\Diamond\Phi$*.

but the subgoal ψ is false (i.e. where $\phi \wedge \neg\psi$ hold) are somewhat less desired and are assigned a higher ranking, and that the rest of the worlds (i.e. where the supergoal ϕ is false) are assigned a still higher ordinal and thus are the least preferred by the agent, i.e.:

$$\kappa(\omega : \phi \wedge \psi) < \kappa(\omega : \phi \wedge \neg\psi) < \kappa(\omega : \neg\phi).$$

Given this κ ranking, contraction by the subgoal ψ (according to the standard approach in [46]) produces the following:

$$\kappa(w : \phi) < \kappa(w : \neg\phi \wedge \neg\psi) < \kappa(w : \neg\phi \wedge \psi).$$

This is a bit strange as all non-ϕ worlds should arguably be equally preferable to the agent after dropping ψ. Moreover, contraction by the supergoal ϕ produces:

$$\kappa(w : \phi \wedge \psi \vee \neg\phi) < \kappa(w : \phi \wedge \neg\psi).$$

Again, intuitively the agent should have no preferences over ψ and $\neg\psi$ worlds. One might wonder why one cannot adjust the standard contraction function to make it work. However, it can be argued that there does not exist an appropriate revision function for κ –modeling the contraction of some goal– that ensures that, e.g., all worlds become equally desirable after the agent drops the supergoal that ϕ (i.e. after contracting κ by ϕ), that also properly handles the contraction by ψ (i.e. the dropping of subgoals) and by ordinary prioritized goals (i.e. the dropping of these goals). That is, while there might be several revision functions for κ, each of which works properly for the dropping of ordinary prioritized goals, subgoals, or supergoals, it is impossible to tailor a single function that works for all of these operations. In other words, since contracting the subgoal ψ should only cause ψ to be dropped while contracting the parent goal ϕ should cause both ϕ and its subgoal ψ (as well as all of its sub-subgoals etc.) to be dropped, we need to have some additional representation to capture this subgoal relation and use this information if we want to handle both types of contractions uniformly. Thus to deal with this, I avoid using such techniques while modeling prioritized goals, subgoals, and their dynamics.

While I made some simplifying assumptions regarding the level of the parent goal relative to which the subgoal is adopted (see Section 5.2), to the best of my knowledge, my account is indeed the first attempt to formalize the dependencies between subgoals and their parent goals using a semantic approach. As discussed in Chapter 2, the existing syntactic approaches to deal with this neither properly handle (sub)goal dynamics nor deal with prioritized goals. While such syntactic accounts of subgoals might enjoy certain advantages, in particular from a computational efficiency standpoint, they suffer from many other serious problems. For instance, consistency between (sub)goals is not automatically maintained. Moreover, in most of these frameworks, subgoals do not properly evolve in response to actions/events, as do ordinary goals. Finally, the many desirable properties of the SubGoal relation discussed in Section 5.3 must also be explicitly enforced, e.g. the acyclicity of the SubGoal relation. My account of subgoals in this chapter avoids these problems.

Chapter 6

A Revised Logical Framework of Prioritized Goals for Committed Agents

6.1 Introduction

In the previous two chapters, I presented a formalization of prioritized goals, subgoals, and their dynamics for agents that *always try to optimize their chosen goals*. In that *"optimizing agent" framework*, an agent's chosen goals are very dynamic. For instance, as mentioned earlier, a currently inactive p-goal ϕ may become active at some later time, for example, if a higher priority active/chosen goal that is currently blocking ϕ — as it is inconsistent with ϕ — becomes impossible. This also means that another currently active/chosen goal ψ may as a result become inactive, not because ψ has become impossible or was dropped, but due to the fact that ψ has lower priority than and is inconsistent with the newly activated goal ϕ. For instance, in my running example in Chapter 4, after the $goBankrupt$ action happens, the agent drops her c-goal that $\Box$BeHappy as the higher priority and conflicting goal that $\Diamond$GetPhD has become active. As discussed in Chapter 4, in that framework chosen goals can be viewed as Bratman et al.'s intentions with an automatic 'opportunity analyzer' and 'filter override mechanism' [21] that forces the optimizing agent to drop her intentions when opportunities to commit to other higher priority goals arise.

Such very dynamic chosen goals/intentions are problematic as a foundation for an agent programming language, as the agents spend a lot of effort in "recomputing" their intentions and plans to achieve them, and their behavior becomes hard to predict for the programmer. To avoid this, here I develop a modified version of the optimizing agent framework that *eliminates agents' (inactive) desires (and as a consequence, the opportunity analyzer and the filter override mechanism) altogether*. Alternatively put, I warrant that in this framework agents' "desires" remain realistic and consistent with their knowledge and intentions.[46] As we will see later, such desires are in fact a subset of the agents' intentions. I achieve this by ensuring that in this framework, *agents' sets of G-accessible paths are the same as that of their G_R-accessible paths*, and that *their p-goals are dropped as soon as they become inactive, i.e. become inconsistent with*

[46]Recall from Chapter 2 that traditionally agents' wishes or desires include goals that are impossible to bring about and that are mutually inconsistent.

their knowledge and/or their intentions. In the resulting *"committed agent" framework*, agents' p-goals are much more dynamic than in the original framework. On the other hand, their c-goals are now much more persistent than before, and are simply the consequential closure of their p-goals, as these must now all be consistent with each other and with the agents' knowledge.

In this chapter, I present the modifications to the optimizing agent framework that are necessary to specify this "committed agent" framework. I then prove some desirable properties about a committed agent's (chosen) goals, goal dynamics, goal introspection, and goal persistence. These show the similarities and differences with the optimizing agent framework. Finally, in light of these properties, I discuss how the committed agent framework compares to the optimizing agent framework, followed by some concluding remarks.

6.2 Specifying Prioritized Goals and Goal Dynamics for Committed Agents

I want to restrict the goals in the agent's goal hierarchy to be realistic and mutually consistent at all times, i.e. consistent with her knowledge and her other goals in all situations. I can obtain a committed agent framework with the following simple changes to my optimizing agent axiomatization: (1) I constrain the agent's initial G-accessible paths at all levels to be realistic, i.e. consistent with her knowledge; (2) I require that initially the agent knows that her p-goals are consistent with each other; (3) I don't allow the agent to adopt a p-goal or subgoal that is inconsistent with her current c-goals, or to drop a p-goal that has become known to be inevitable; and (4) I modify the successor-state axiom for G so that the constraints in (1) and (2) above are preserved. In the following I formalize and discuss each of these changes, one at a time.

First of all, I require that *all initial G-accessible paths at all levels in the agent's goal hierarchy* must be realistic, and thus they *must start with a K-accessible situation*. I specify this by imposing the following constraint on the domain theory:

Assumption 6.2.1 (PGoal Strong Realism)**.**

$$\forall p, n, s.\ \text{Init}(s) \land G(p, n, s) \supset \exists s'.\ \text{Starts}(p, s') \land K(s', s).$$

This, along with Definitions 4.2.1, 4.2.4, and 4.2.5, ensures that given an initial situation s and a priority level n, the agent's set of p-goals at n in s and that of realistic p-goals at n in s are the same, i.e.:

$$\forall s.\ \text{Init}(s) \supset \forall n, \phi.\ \text{PGoal}(\phi, n, s) \equiv \text{RPGoal}(\phi, n, s).$$

Secondly, I require that the agent initially know that all of her p-goals are consistent with each other and with her knowledge. This can be specified by ensuring that *initially* *there exists a realistic path over which all of her p-goals hold.* Thus I require the domain theory to satisfy the following constraint:

Assumption 6.2.2 (PGoal Consistency)**.**

$$\forall s.\ \text{Init}(s) \supset \exists p.\ \forall n.\ G(p, n, s).$$

This thus ensures that the agent's initial (realistic) p-goal hierarchy is consistent. Note that, the additional condition that the starting situation of p be K-accessible in s is unnecessary, since this follows from Assumption 6.2.1.

It is the responsibility of the agent designer to ensure that these two constraints are entailed by the committed agent theory. A pragmatic consequence of these assumptions is that agents' initial p-goals no longer represent true desires, which are usually allowed to be impossible to bring about and/or conflicting with each other, but rather their "goals". In fact, the elimination of desires in this framework considerably simplifies c-goal dynamics and contributes to the persistence of c-goals. I will come back to this issue later. Also, I will show that if these constraints are asserted for the initial situations, they continue to hold for all situations as they are preserved by the (modified) successor-state axiom for G.

Thirdly, recall that in the optimizing agent framework, I allow the agents to adopt desires that are currently inconsistent with their chosen goals. As discussed earlier, while in that framework such adopted desires are initially inactive, they might later become active and trigger the dropping of other (lower priority) chosen goals/intentions. This is consistent with my model of the opportunity analyzer and the filter override mechanism for intentions. Thus in the optimizing agent framework, the adoption of a goal ϕ results in the adoption of the p-goal that ϕ, but not necessarily in the adoption of the c-goal that ϕ. Similarly, the dropping of a goal that ϕ results in the dropping of the p-goal that ϕ, but not necessarily in the dropping of the realistic p-goal or the c-goal that ϕ — ϕ could be inevitable or could be a consequence of other c-goals; moreover, the agent could have ϕ as a realistic p-goal without also having it as a p-goal (recall that when the $drop(\phi)$ action happens, the agent drops ϕ only from the levels where ϕ is a p-goal). Thus the adopt and drop actions in the optimizing agent framework are meant to be viewed as operations over the agents' desires. In contrast, in the committed agent framework, we want our agents to be more committed to their chosen goals; thus they should be *allowed to adopt a new goal only if it is consistent with all their current c-goals*. Similarly, they should be *permitted to drop a goal only if the goal — relative to their sets of primary chosen goals— is indeed dropable*. These changes in the prerequisites for the adopt and drop actions mean that these actions no longer refer to the adoption and dropping of desires of the agent. Rather, they should be viewed as that of (primary) chosen goals or intentions of the agent.

Thus, I update the action precondition axioms for $adopt(\phi, n)$, $adoptRelTo(\psi, \phi)$, and $drop(\phi)$ as follows. An agent can adopt the c-goal that ϕ at level n in situation s if she does not already have ϕ as her p-goal at some level m in s (as before), and she does not intend in s that $\neg\phi$ next:[47]

Axiom 6.2.3.

$$\text{Poss}(adopt(\phi, n), s) \equiv \neg\exists m.\ \text{PGoal}(\phi, m, s) \wedge$$
$$\neg\text{CGoal}(\neg\exists s', p'.\ \text{Starts}(s') \wedge \text{Suffix}(p', do(adopt(\phi, n), s')) \wedge \phi(p'), s).$$

Moreover, an agent can adopt a subgoal ψ w.r.t. the parent goal ϕ if she has the parent goal that ϕ as a p-goal at some level m in s and she does not already have the p-goal that ψ at some level n in s (as before), and she does not intend in s that $\neg\psi$ next:

Axiom 6.2.4.

$$\text{Poss}(adoptRelTo(\psi, \phi), s) \equiv \exists m.\ \text{PGoal}(\phi, m, s) \wedge \neg\exists n.\ \text{PGoal}(\psi, n, s) \wedge$$
$$\neg\text{CGoal}(\neg\exists s', p'.\ \text{Starts}(s') \wedge \text{Suffix}(p', do(adoptRelTo(\psi, \phi), s') \wedge \psi(p'), s).$$

Finally, an agent can *drop* the c-goal that ϕ in situation s if she has ϕ as her p-goal at some level n in s (as before), and if ϕ is not known to be inevitable in s:

[47]This may seem very restrictive as ϕ is required to be consistent with the agent's p-goals even at priority lower than n. But note that if the agent drops the inconsistent lower priority p-goals, she will then be able to adopt ϕ.

Axiom 6.2.5.

$$\text{Poss}(drop(\phi), s) \equiv \exists n.\ \text{PGoal}(\phi, n, s) \wedge \neg\text{KInevitable}(\phi, s).$$

Thus unlike in the optimizing agent framework, in this framework I do not allow an agent to adopt a goal or a subgoal that is inconsistent with her current chosen goals. Also, an agent is not allowed to drop a goal if it is known to be inevitable. Note that, in these axioms, I didn't replace the PGoal operators with CGoals; as I will show later, for any given situation, all p-goal levels are always active, so in this framework having a p-goal that ϕ amounts to having the (primary) c-goal that ϕ.

Finally, I modify the successor-state axiom for G to *preserve Assumptions 6.2.1 and 6.2.2 for all situations and thus eliminate the opportunity analyzer and the filter override mechanism*. I achieve this by replacing the Progressed construct in Axiom 5.2.2 and Definition 4.3.6 with the ProgressedAndFiltered construct, the Progressed construct in Definitions 4.3.5 and 5.2.3 with the Progressed$_{\text{CA}}$ construct, and the SameHist construct in Definitions 4.3.5 and 4.3.6 with the K-relation. In the following, I specify the dynamics of p-goals in the committed agent framework by giving the successor-state axiom for the G relation, and then discuss the changes required for each case, one at a time:

Axiom 6.2.6 (SSA for G)**.**

$$
\begin{aligned}
&G(p, n, do(a, s)) \equiv \\
&\forall \phi, \psi, m.\ (a \neq adopt(\phi, m) \wedge a \neq adoptRelTo(\psi, \phi) \wedge a \neq drop(\phi) \wedge \\
&\qquad\qquad\quad \text{ProgressedAndFiltered}(p, n, a, s)) \\
&\quad \vee \exists \phi, m.\ (a = adopt(\phi, m) \wedge \text{Adopted}_{\text{CA}}(p, n, m, a, s, \phi)) \\
&\quad \vee \exists \phi, \psi.\ (a = adoptRelTo(\psi, \phi) \wedge \text{SubGoalAdopted}_{\text{CA}}(p, n, a, s, \psi, \phi)) \\
&\quad \vee \exists \phi.\ (a = drop(\phi) \wedge \text{Dropped}_{\text{CA}}(p, n, a, s, \phi)).
\end{aligned}
$$

The above axiom and the following definitions are exactly as in the optimizing agent framework, with the exception of the aforementioned changes, so I will only discuss these changes. Again, given some situation, the purpose of these changes is to preserve Assumptions 6.2.1 and 6.2.2 for all possible successor situations (I will show this preservation formally in Section 6.3.2). First, let us consider the case when the action performed is a regular, i.e. non-adopt/drop action; for this, I replaced Progressed(p, n, a, s) with ProgressedAndFiltered(p, n, a, s) (cf. Axiom 6.2.6), which is defined as follows:

Definition 6.2.7.

$$\begin{aligned}
&\text{ProgressedAndFiltered}(p, n, a, s) \overset{\text{def}}{=} \\
&\quad \textbf{if } (n = 0 \land \exists p'.\ \text{Progressed}_{\text{CA}}(p', n, a, s)) \\
&\qquad \textbf{then } \text{Progressed}_{\text{CA}}(p, n, a, s) \\
&\quad \textbf{else if } (n \neq 0 \land \exists p'.\ G_{\cap}(p', n-1, do(a, s)) \land \text{Progressed}(p', n, a, s)) \\
&\qquad \textbf{then } \text{Progressed}_{\text{CA}}(p, n, a, s) \\
&\quad \textbf{else } \exists s'.\ \text{Starts}(p, s') \land K(s', do(a, s)),
\end{aligned}$$

where the $\text{Progressed}_{\text{CA}}$ construct is defined as follows:

Definition 6.2.8.

$$\text{Progressed}_{\text{CA}}(p, n, a, s) \overset{\text{def}}{=} \text{Progressed}(p, n, a, s) \land \exists s'.\ \text{Starts}(p, s') \land K(s', do(a, s)).$$

Here ProgressedAndFiltered plays a similar role to Progressed in the Chapter 4 definition, but also drops goals that have become known to be impossible or inconsistent with higher priority goals. After some action a happens in s, ProgressedAndFiltered(p, n, a, s) replaces a G-accessible path (say with starting situation s^*) at level n in s with its suffix p w.r.t. s' (where $s' = do(a, s^*)$), provided that s' is K-accessible in $do(a, s)$. This is modeled using the $\text{Progressed}_{\text{CA}}$ construct, which only progresses realistic goal paths. In addition, if a makes one or more p-goals at n impossible or inconsistent with higher priority p-goals, then ProgressedAndFiltered(p, n, a, s) adds back to the G-relation at level n any path p that starts with a K-accessible situation in $do(a, s)$. It thus contributes to the maintenance of Assumption 6.2.2 by replacing the only p-goal at all such inactive levels in the agent's goal hierarchy in $do(a, s)$ with the trivial formula that she be in a K-accessible situation in $do(a, s)$, thus effectively dropping these inactive p-goals. Note that any path p that is in the ProgressedAndFiltered(p, n, a, s) relation must start with a situation that is K-accessible in $do(a, s)$. ProgressedAndFiltered thus also ensures that Assumption 6.2.1 is preserved for all levels when the action performed is a regular action.

Next, consider the case where the action performed is an $adopt$ action. Note that, the preconditions for $adopt$ guarantee that for any executable situation, the agent's c-goals are consistent after an $adopt$ action happens; thus if s is executable and a is possible in s, then Assumption 6.2.2 is automatically maintained in $do(a, s)$. To preserve Assumption 6.2.1, in obtaining Definition 6.2.9 from Definition 4.3.5 I replace SameHist with the K-relation for the priority level where the goal is adopted, and Progressed with Progressed$_{\text{CA}}$ for all other levels:

Definition 6.2.9.

$$\text{Adopted}_{\text{CA}}(p, n, m, a, s, \phi) \overset{\text{def}}{=}$$

$$\textbf{if } (n < m) \textbf{ then } \text{Progressed}_{\text{CA}}(p, n, a, s)$$

$$\textbf{else if } (n = m) \textbf{ then } \exists s'.\ \text{Starts}(p, s') \wedge K(s', do(a, s)) \wedge \phi(p)$$

$$\textbf{else } \text{Progressed}_{\text{CA}}(p, n - 1, a, s).$$

The former modification ensures that all the G-accessible paths at the adopted level in $do(a, s)$ are realistic in that each of them starts with a K-accessible situation in $do(a, s)$. The latter warrants that this also holds for all other levels (again, note that $\text{Progressed}_{\text{CA}}(p, n, a, s)$ requires that p starts with a K-accessible situation in $do(a, s)$).

Thirdly, let us consider the case for an $adoptRelTo$ action. As in the previous case, the preconditions for $adoptRelTo$ ensure that for any executable situation, Assumption 6.2.2 is automatically preserved. To maintain Assumption 6.2.1, in obtaining Definition 6.2.10 from Definition 5.2.3 I use $\text{Progressed}_{\text{CA}}$ instead of Progressed for all levels, thus ensuring that the G-accessible paths at all levels in $do(a, s)$ are realistic:

Definition 6.2.10.

$$\text{SubGoalAdopted}_{\text{CA}}(p, n, a, s, \psi, \phi) \overset{\text{def}}{=}$$

$$\textbf{if } (\exists m.\ \text{AdoptedLevel}(\phi, m, s) \wedge n < m) \textbf{ then } \text{Progressed}_{\text{CA}}(p, n, a, s)$$

$$\textbf{else if } (\text{AdoptedLevel}(\phi, n, s)) \textbf{ then } (\text{Progressed}_{\text{CA}}(p, n-1, a, s) \wedge \psi(p))$$

$$\textbf{else } \text{Progressed}_{\text{CA}}(p, n-1, a, s).$$

Finally, to handle a *drop* action, in obtaining Definition 6.2.11 from Definition 4.3.6 I replace the SameHist construct with the K-relation for the levels from where the goal is dropped, and the Progressed construct with ProgressedAndFiltered for all other levels:

Definition 6.2.11.

$$\text{Dropped}_{\text{CA}}(p, n, a, s, \phi) \overset{\text{def}}{=}$$

$$\textbf{if } \text{PGoal}(\phi, n, s) \textbf{ then } \exists s'.\ \text{Starts}(p, s') \wedge K(s', do(a, s))$$

$$\textbf{else } \text{ProgressedAndFiltered}(p, n, a, s).$$

The former preserves Assumption 6.2.1 for the dropped levels, by adding back to these levels only those paths that starts with a K-accessible situation in $do(a, s)$, rather than those that starts with a situation that has the same history as $do(a, s)$ (see Definition 4.3.6). The latter is somewhat different from the above two cases (i.e. that for *adopt* and $adoptRelTo$). This is because the preconditions for $drop$ do not ensure that the agent does not currently have the dropped goal as her c-goal or that she does not intend not to execute this $drop$ action next; such a requirement would be too strong as there is no point in dropping a c-goal unless the agent has it as her c-goal. As discussed above, ProgressedAndFiltered preserves Assumption 6.2.1 for the rest of the levels in the goal hierarchy. Moreover, it also ensures that there is a common G-accessible path in $do(a, s)$ that is accessible from all of these levels, and that starts with a K-accessible situation in $do(a, s)$. This along with the fact that the G relations at the levels from where the goal is dropped include all paths that starts with a K-accessible situation in $do(a, s)$ thus also preserves Assumption 6.2.2. I will prove these results formally in the next section.

These modifications thus ensure that if the occurrence of an action a in some situation s makes a p-goal ϕ at level n impossible or inconsistent with other higher priority p-goals, ϕ (as well as the only p-goal at level n) is dropped from the agent's p-goal hierarchy. Therefore, unlike in the optimizing agent framework, where the agent keeps both active and inactive p-goals in an attempt to keep optimizing her c-goals, in this framework the agent simply drops a p-goal as soon as it becomes inactive.

I can now define the theory $\mathcal{D}_{CAgt}^{SG}$ for modeling committed agents with subgoals as follows:

Definition 6.2.12.

$$\mathcal{D}^{SG}_{CAgt} \overset{\text{def}}{=} \mathcal{D}^{SG}_{OAgt} \backslash \{\text{Axioms 4.3.1, 5.2.1, 4.3.2, 5.2.2}\}$$

$$\cup \{\text{Assumptions 6.2.1 – 6.2.2}\} \cup \{\text{Axioms 6.2.3 – 6.2.6}\}.$$

Note that according to the above successor-state axiom, the agent's G-accessibility relation at some level in $do(a, s)$ depends on some of her G-accessibility relations at other (higher priority) levels in $do(a, s)$. Therefore at a first glance, this axiom may not seem well defined. To show that it is in fact well defined, in the following I prove that this axiom is "Markovian", i.e. that $G(p, n, do(a, s))$ only depends on the K and the G relations in situation s and on the action a. First I show that for any path p, priority level n, and situation s, $G_\cap(p, n, s)$ can be completely specified in terms of a formula that does not mention $G_\cap$:

Proposition 6.2.13. *For all n:*

$$\mathcal{D}^{SG}_{CAgt} \models \forall p, s.\ G_\cap(p, n, s) \equiv \Pi_{G_\cap}(p, n, s),$$

where $\Pi_{G_\cap}(p, n, s)$ is a formula that does not mention $G_\cap$ and whose free variables are among p and s.

Proof. (By induction on n) The base case is trivial, since the right-hand side of Axiom 4.2.7 does not mention $G_\cap$ and only consists of free variables from $\{p, s\}$ when $n = 0$.

For the inductive hypothesis, fix M and assume that the proposition holds for $n = M$, i.e. $\forall p, s.\ G_{\cap}(p, M, s) \equiv \Pi_{G_{\cap}}(p, M, s)$, where $\Pi_{G_{\cap}}(p, M, s)$ is a formula that does not mention $G_{\cap}$ and whose free variables are among p and s. I need to show that the proposition holds for $n = M + 1$. This also trivially follows from Axiom 4.2.7, since the right-hand side of this axiom only mentions occurrences of $G_{\cap}$ at level M with free variables from $\{p, s\}$, and by the inductive hypothesis, all such occurrences can be replaced by $\Pi_{G_{\cap}}(p, M, s)$. $\square$

Next, I show that for any path p, level n, and situation s, $G_{\cap}(p, n, s)$ can be completely specified in terms of a formula that does not mention the G-accessibility relations at levels that have lower priority than n:

Proposition 6.2.14. *For all n:*

$$\mathcal{D}_{CAgt}^{SG} \models \forall p, s.\ G_{\cap}(p, n, s) \equiv \Pi_{G}(p, n, s),$$

where $\Pi_G(p, n, s)$ is a formula that does not mention $G_{\cap}$ and G relations at m, where $m > n$, and whose free variables are among p and s.

Proof Sketch. (By induction on n) Similar to the proof of Proposition 6.2.13. $\square$

Finally, I show that an agent's G-accessible paths in situation $do(a, s)$ can be completely specified in terms of a formula that does not mention her K and G accessibility relations in $do(a, s)$. To be more specific, I show that for any priority level n, we can progressively substitute the right-hand side of the successor-state axiom for G with a formula that talks about the K and the G relations only in situation s:

Proposition 6.2.15. *For all n:*

$$\mathcal{D}^{SG}_{CAgt} \models \forall p, a, s.\; G(p, n, do(a, s)) \equiv \Pi_{KG}(p, n, a, s),$$

where $\Pi_{KG}(p, n, a, s)$ is a formula that mentions the K and the G relations only in situation s, and whose free variables are among $p, a,$ and s.

Proof. (By strong induction on n) For the base case, fix A_1 and S_1 and set $n = 0$. By Axiom 6.2.6 and Definitions 6.2.7, 6.2.9, 6.2.10, 6.2.11, 6.2.8, and 4.3.4, $G(p, 0, do(A_1, S_1))$ can be specified using a formula that involves the G accessibility relation in S_1 and the K accessibility relation in $do(A_1, S_1)$, and whose only free variable is p. In this formula, if we replace the K relation in $do(A_1, S_1)$ by the right-hand side of Axiom 3.4.10, we obtain a formula $\Pi_{KG}(p, 0, A_1, S_1)$ that mentions the K and the G accessibility relations only in situation S_1 and whose only free variable is p. Thus, the G relation at level 0 in $do(A_1, S_1)$ can be expressed with a formula that mentions the K and the G relations only in situation S_1.

For the inductive hypothesis, fix $A_n, S_n,$ and M, and assume that for all n s.t. $0 < n \leq M$, and for all p, $G(p, n, do(A_n, S_n))$ can be completely specified by a formula

$\Pi_{KG}(p, n, A_n, S_n)$ that mentions the K and the G relations in S_n only and whose only free variable is p. I need to show that the proposition holds for $M+1$. Again by Axiom 6.2.6 and Definitions 6.2.7, 6.2.9, 6.2.10, 6.2.11, 6.2.8, and 4.3.4, we can see that G at level $M+1$ in situation $do(A_n, S_n)$, i.e. $G(p, M+1, do(A_n, S_n))$, can be specified using a formula $\Pi'_{KG}(p, M+1, A_n, S_n)$ whose only free variable is p and that involves the G relation in S_n, the K relation in $do(A_n, S_n)$, and the $G_\cap$ relation at level M in $do(A_n, S_n)$; note that the latter appears as $G_\cap(p', M, do(A_n, S_n))$, where p' is bound. By Proposition 6.2.14, we can replace every occurrence of $G_\cap(p', M, do(A_n, S_n))$ in $\Pi'_{KG}(p, M+1, A_n, S_n)$ with a formula $\Pi_G(p', M, do(A_n, S_n))$ that, w.r.t. the G relation, only mentions G at levels m in $do(A_n, S_n)$, where $m \leq M$. By the inductive hypothesis, these G relations in $\Pi_G(p', M, do(A_n, S_n))$ can be expressed using a formula $\Pi''_{KG}(p', M, A_n, S_n)$ that mentions the K and the G relations in S_n only. Moreover from Axiom 4.2.7, we can see that $\Pi_G(p', M, do(A_n, S_n))$ also mentions the K relation in $do(A_n, S_n)$. As in the base case, we can use the right-hand side of Axiom 3.4.10 to replace the K relation in $do(A_n, S_n)$ (both in $\Pi'_{KG}(p, M+1, A_n, S_n)$ and $\Pi_G(p', m', do(A_n, S_n))$) by a formula that mentions the K relation in S_n only. Thus we can obtain a specification of $G(p, M+1, do(A_n, S_n))$ using a formula $\Pi'_{KG}(p, M+1, A_n, S_n)$ that refers to the agent's K and G relations only in situation S_n. $\square$

Note that, the successor-state axiom for G and Axiom 6.2.5 together ensure that for any executable situation s, the agent does not have the realistic p-goal that ϕ after she drops it in s (see Proposition 6.3.28 below). However, this does not necessarily hold for c-goals, as ϕ could be a consequence of the agent's other c-goals. I could have modified the successor-state axiom to identify and drop, e.g. a minimal set of c-goals that contribute to ϕ; but this would have complicated the framework further, e.g. by incorporating techniques used for belief revision to minimize the change in the agent's revised c-goals, etc. I leave this for future work.

6.3 Properties

In this section, I show that my formalization of committed agents has some desirable properties. Many of these or their corresponding versions (in particular those in Sections 6.3.1, 6.3.3, 6.3.4, and 6.3.5) have been already shown to hold in the optimizing agent framework. I also point out the differences between the two frameworks.

6.3.1 Basic Properties

I start with some basic properties. The proofs for these are exactly the same as in the optimizing agent framework since they do not refer to any of the axioms that deal with goal dynamics. First, I can show that an agent's chosen goals are consistent:

Proposition 6.3.1 (Consistency of CGoals)**.**

$$\mathcal{D}^{SG}_{CAgt} \models \forall s.\ \neg \text{CGoal}(\text{False}, s).$$

Proof. Same as that of Proposition 4.4.2. □

Recall that in the optimizing agent framework, an agent is allowed to have a p-goal that is impossible (as her G-accessible paths at some given level can be empty). In contrast, in the committed agent framework the above property can be shown to hold for (realistic) p-goals (and as a consequence, for primary c-goals) for all executable situations. I show this formally in Corollary 6.3.17.

The property of realism also holds for the committed agent framework, and thus all known to be inevitable goals are also chosen/intended:

Proposition 6.3.2 (Realism)**.**

$$\mathcal{D}^{SG}_{CAgt} \models \forall s.\ \text{KInevitable}(\phi, s) \supset \text{CGoal}(\phi, s).$$

Proof. Same as that of Proposition 4.4.3. □

Again, recall that realism does not hold for p-goals/primary c-goals in the optimizing agent framework – an agent may know that something has become inevitable and not have it as her p-goal or primary c-goal. In contrast, this can't be the case in the committed agent framework. In particular, p-goal realism can be shown to hold in this framework (see Corollary 6.3.7). Moreover, primary c-goal realism also holds in the framework for executable situations (by Corollaries 6.3.7 and 6.3.20).

From these two propositions, it follows that no known to be impossible goal is ever chosen/intended:

Corollary 6.3.3.

$$\mathcal{D}^{SG}_{CAgt} \models \forall s.\ \text{CGoal}(\phi, s) \supset \neg\text{KImpossible}(\phi, s).$$

Proof. Same as that of Corollary 4.4.4. $\square$

Once again, unlike in the optimizing agent framework, the above property can also be shown to hold for agents' (realistic) p-goals and primary c-goals for all executable situations (see Corollary 6.3.19). This is because in this framework, all p-goals of the agent are always active, i.e. chosen.

6.3.2 Dynamic Properties I: Preservation of PGoal Strong Realism and Consistency, and its Consequences

I next discuss some properties of goals that are specific to committed agents. In particular, in this section I show that if Assumptions 6.2.1 and 6.2.2 are prescribed to hold for the initial situations, then they will remain true in all situations as they are preserved by the SSA for G. Moreover, I show how a committed agent's prioritized goals and chosen goals are related; to be more specific, I prove that an agent's chosen goals are just the intersection of her p-goals.

First, I show that Assumption 6.2.1 is preserved for all non-initial situations:

Proposition 6.3.4.

$$\mathcal{D}^{SG}_{CAgt} \models \forall p, n, a, s.\ G(p, n, do(a, s)) \supset \exists s'.\text{Starts}(p, s') \wedge K(s', do(a, s)).$$

Proof. (By induction on s) The base case holds by Assumption 6.2.1. The general case trivially follows from Axiom 6.2.6 and Definitions 6.2.7, 6.2.9, 6.2.10, 6.2.11, and 6.2.8, as ProgressedAndFiltered(p, n, a, s), Adopted$_{\text{CA}}(p, n, m, a, s, \phi)$, SubGoal-Adopted$_{\text{CA}}(p, n, a,\ s, \psi, \phi)$, Dropped$_{\text{CA}}(p, n, a, s, \phi)$, and Progressed$_{\text{CA}}(p, n, a, s)$ all by definition ensure that p must start with a K-accessible situation in $do(a, s)$. □

Thus in this framework, an agent's set of G and G_R-accessible paths are equivalent:

Corollary 6.3.5.

$$\mathcal{D}^{SG}_{CAgt} \models \forall p, n, s.\ G(p, n, s) \equiv G_R(p, n, s).$$

Proof. Follows from Assumption 6.2.1, Proposition 6.3.4, and Definition 4.2.4. □

It follows that an agent's set of p-goals at some level n in situation s and that of realistic p-goals are the same:

Corollary 6.3.6.

$$\mathcal{D}^{SG}_{CAgt} \models \forall n, s.\ \text{PGoal}(\phi, n, s) \equiv \text{RPGoal}(\phi, n, s).$$

Proof. Follows from Corollary 6.3.5 and Definitions 4.2.1 and 4.2.5. □

I can also show that the property of realism holds for a committed agent's p-goals:

Corollary 6.3.7.

$$\mathcal{D}^{SG}_{CAgt} \models \forall s.\ \text{KInevitable}(\phi, s) \supset \forall n.\ \text{PGoal}(\phi, n, s).$$

Proof. (By contradiction) Fix ϕ_1 and S_1 and assume that KInevitable(ϕ_1, S_1). From this and Definition 3.5.14, we have that ϕ_1 holds over any path that starts with a situation that is K-accessible in S_1, i.e.:

$$\forall p, s'.\ \text{Starts}(p, s') \wedge K(s', S_1) \supset \phi_1(p). \tag{6.1}$$

Fix N_1 and assume that: $\neg$PGoal(ϕ_1, N_1, S_1). Then by Definition 4.2.1, there exists a G-accessible path P_1 at N_1 in S_1 over which $\neg\phi_1$ holds, i.e. $\neg\phi_1(P_1)$. But by Assumption 6.2.1 and Proposition 6.3.4, P_1 must start with a situation that is K-accessible in S_1, and thus by (6.1), ϕ_1 holds over P_1, i.e. $\phi_1(P_1)$ — a contradiction. □

I next show that Assumption 6.2.2 is preserved by the successor-state axiom for G and thus holds for all situations. In the following, I start by giving some lemmata that

I will need to get this result. The first lemma states that for any situation s and priority level n, any path p that is in the $G_\cap$ relation at n in s must start with a K-accessible situation in s:

Lemma 6.3.8.

$$\mathcal{D}^{SG}_{CAgt} \models \forall p, n, s.\ G_\cap(p, n, s) \supset \exists s'.\ \text{Starts}(p, s') \wedge K(s', s).$$

Proof. (By induction on n). Fix S_1. The base case where $n = 0$ follows trivially from Axiom 4.2.7 and Definition 4.2.4. For the inductive case, fix N_1 and assume that $\forall p.\ G_\cap(p, N_1, S_1) \supset \exists s.\ \text{Starts}(p, s) \wedge K(s, S_1)$. Fix path P_1 s.t. $G_\cap(P_1, N_1 + 1, S_1)$. I need to show that P_1 starts with a K-accessible situation in S_1. Note that by Axiom 4.2.7, path P_1 is in $G_\cap$ at $N_1 + 1$ in S_1 iff:

```
if $\exists p'.\ (G_R(p', N_1 + 1, S_1) \wedge G_\cap(p', N_1, S_1))$
        then $(G_R(P_1, N_1 + 1, S_1) \wedge G_\cap(P_1, N_1, S_1))$
else $G_\cap(P_1, N_1, S_1)$.
```

In both these cases $G_\cap(P_1, N_1, S_1)$ must hold, and thus $\exists s'.\ \text{Starts}(P_1, s') \wedge K(s', S_1)$ follows from the inductive hypothesis. □

The next lemma states that if a path is G_R-accessible for all levels up to m in situation s, then it must be in the $G_\cap$ relation up to m in s:

Lemma 6.3.9.

$$\mathcal{D}_{CAgt}^{SG} \models \forall m, p, s.\ (\forall n.\ n \leq m \supset G_R(p, n, s)) \supset G_\cap(p, m, s).$$

Proof. (By induction on m) Fix situation S_1 and path P_1. The base case, i.e. when $m = 0$, follows trivially from Axiom 4.2.7. For the inductive case, fix M_1 and assume:

$$\forall n.\ n \leq M_1 + 1 \supset G_R(P_1, n, S_1). \tag{6.2}$$

We need to show that $G_\cap(P_1, M_1 + 1, S_1)$. From (6.2) and the inductive hypothesis, we have:

$$G_\cap(P_1, M_1, S_1). \tag{6.3}$$

From Axiom 4.2.7, (6.2), and (6.3), we can see that P_1 is $G_\cap$-accessible at $M_1 + 1$ in S_1 iff both $G_R(P_1, M + 1, S_1)$ and $G_\cap(P_1, M_1, S_1)$ holds. The former follows from (6.2) while the latter from (6.3). □

The next lemma says that if there is a path p that is G_R-accessible for all levels up to level m in situation s, then any path p' that is in the $G_\cap$ relation at some higher or equal priority (w.r.t. m) level n' in s must also be G_R-accessible at n' in s:

Lemma 6.3.10.

$$\mathcal{D}_{CAgt}^{SG} \models \forall m, s.\ (\exists p.\ (\forall n.\ n \leq m \supset G_R(p, n, s))) \supset$$
$$(\forall p', n'.\ n' \leq m \wedge G_\cap(p', n', s) \supset G_R(p', n', s)).$$

Proof. (By contradiction) For the antecedent, fix $M_1, S_1,$ and P_1 and assume:

$$\forall n.\ n \leq M_1 \supset G_R(P_1, n, S_1). \tag{6.4}$$

For the consequent, fix P_1' and N_1' and assume:

$$N_1' \leq M_1 \wedge G_\cap(P_1', N_1', S_1). \tag{6.5}$$

By contradiction, assume:

$$\neg G_R(P_1', N_1', S_1). \tag{6.6}$$

Axiom 4.2.7 gives us two cases. First consider the case when $N_1' = 0$. In that case, from Axiom 4.2.7 we have that a path p is $G_\cap$ accessible at 0 in S_1 iff:

$$\begin{array}{l}\textbf{if } \exists p'.\ G_R(p', 0, S_1) \textbf{ then } G_R(p, 0, S_1)\\ \textbf{else } \exists s'.\ \text{Starts}(p, s') \wedge K(s', S_1).\end{array}$$

Since by (6.5), P_1' is $G_\cap$-accessible at 0 in S_1, and by (6.4), there is a path, namely P_1, that is G_R-accessible at 0 in S_1, P_1' must be G_R-accessible at 0 in S_1 — a contradiction with (6.6).

Now consider the case for $N_1' > 0$. From (6.4) and (6.5), we have:

$$G_R(P_1, N_1', S_1). \tag{6.7}$$

Again, from (6.4), (6.5) and Lemma 6.3.9, it follows that:

$$G_\cap(P_1, N_1' - 1, S_1). \tag{6.8}$$

Since $N'_1 > 0$, from Axiom 4.2.7 we have that a path p is $G_\cap$-accessible at N'_1 in S_1 iff:

if $\exists p'.\ (G_R(p', N'_1, S_1) \wedge G_\cap(p', N'_1 - 1, S_1))$

then $(G_R(p, N'_1, S_1) \wedge G_\cap(p, N'_1 - 1, S_1))$

else $G_\cap(p, N'_1 - 1, S_1)$.

Since by (6.7), there is a path, namely P_1, that is G_R-accessible at N'_1 in S_1 and by (6.8), P_1 is in the $G_\cap$ relation at $N'_1 - 1$ in S_1, the condition in the **if** holds, and thus any path p that is in the $G_\cap$ relation at N'_1 in S_1 must be G_R-accessible at N'_1 in S_1 (i.e. the condition in the **then** must hold for p). Since by (6.5) P'_1 is such a path, it must be G_R-accessible at N'_1 in S_1 — a contradiction with (6.6). $\square$

Finally, the last lemma says that if there is a path p that is in the $G_\cap$ relation at level n in situation s, then it is also in the $G_\cap$ relation at all higher priority (w.r.t. n) levels m in s:

Lemma 6.3.11.

$$\mathcal{D}^{SG}_{CAgt} \models \forall p, n, s.\ G_\cap(p, n, s) \supset (\forall m.\ m < n \supset G_\cap(p, m, s)).$$

Proof. (By induction on n) In the base case, where $n = 0$, the consequent trivially holds. For the inductive case, where $n > 0$, fix N_1, and assume that:

$$\forall p, s.\ G_\cap(p, N_1, s) \supset (\forall m.\ m < N_1 \supset G_\cap(p, m, s)).$$

Also fix P_1 and S_1 and assume that $G_\cap(P_1, N_1 + 1, S_1)$. From this and Axiom 4.2.7, it follows that:

$$G_\cap(P_1, N_1, S_1). \tag{6.9}$$

From this and the inductive hypothesis, it follows that:

$$\forall m.\ m < N_1 \supset G_\cap(P_1, m, S_1). \tag{6.10}$$

The consequent follows from (6.9) and (6.10). $\square$

Using these lemmata, I now prove that Assumption 6.2.2 also hold for any executable situation as it is preserved by the successor-state axiom for G:

Proposition 6.3.12.

$$\mathcal{D}^{SG}_{CAgt} \models \forall s.\ \text{Executable}(s) \supset \exists p.\ \forall n.\ G(p, n, s).$$

Proof. (By induction on s) For the base case, fix situation S_{init} s.t. $\text{Init}(S_{init})$ and assume that $\text{Executable}(S_{init})$. The consequent trivially follows from this and Assumption 6.2.2.

For the inductive case, fix situation S^* and assume that:

$$\exists p.\ \forall n.\ G(p, n, S^*). \tag{6.11}$$

Also, fix action A^* and assume that:

$$\text{Executable}(do(A^*, S^*)). \tag{6.12}$$

I need to show that $\exists p.\ \forall n.\ G(p, n, do(A^*, S^*))$. From (6.12) and Definition 3.3.1, we have:

$$\text{Poss}(A^*, S^*). \tag{6.13}$$

Now, the successor-state axiom for G, i.e. Axiom 6.2.6, gives us four cases. For each of these cases, I will prove that $\exists p.\ \forall n.\ G(p, n, do(A^*, S^*))$. Let us consider them, one at a time.

<u>A^* is a regular action (i.e. not an $adopt$, $adoptRelTo$, or $drop$ action)</u>: In this case, by Axiom 6.2.6, I need to show that there is a path p such that $\forall n.$ ProgressedAndFiltered(p, n, A^*, S^*). I will prove this by strong induction on level n. First consider the base case, i.e. when $n = 0$. From Definitions 6.2.7 and 6.2.8, ProgressedAndFiltered$(p, 0, A^*, S^*)$ holds for some path p at level 0 after A^* has happened in S^* iff:

if $(\exists p', s'.\ \text{Progressed}(p', 0, A^*, S^*) \land \text{Starts}(p', s') \land K(s', do(A^*, S^*)))$

then $\text{Progressed}(p, 0, A^*, S^*) \land \exists s'.\ \text{Starts}(p, s') \land K(s', do(A^*, S^*))$

else $\exists s'.\ \text{Starts}(p, s') \land K(s', do(A^*, S^*))$.

If there exists a G-accessible path P_1 at 0 in S^* whose suffix p' relative to A^* starts with a situation that is K-accessible in $do(A^*, S^*)$, then ProgressedAndFiltered$(P_1, 0, A^*, S^*)$. Otherwise, let P_1 be a path that starts with a situation that is K-accessible in $do(A^*, S^*)$. Since K is reflexive, and $do(A^*, S^*)$ is executable (by (6.12)), by the assumption that there is an executable action in all situations and Proposition 3.5.36, such a path P_1 indeed exists. Clearly, ProgressedAndFiltered$(P_1, 0, A^*, S^*)$ holds.

For the inductive case, fix M and assume that there is a path P_1 s.t. $\forall n.\ n \leq M \supset$ ProgressedAndFiltered(P_1, n, A^*, S^*); I need to show that:

$$\exists p.\ (\forall n.\ n \leq M + 1 \supset \text{ProgressedAndFiltered}(p, n, A^*, S^*)).$$

Note that, by Axiom 6.2.6, the fact that A^* is a regular action, and the inductive hypothesis, it follows that P_1 is G-accessible at all levels up to M in $do(A^*, S^*)$ since ProgressedAndFiltered holds for all these levels. From this and Corollary 6.3.5, we have:

$$\forall n.\ n \leq M \supset G_R(P_1, n, do(A^*, S^*)). \tag{6.14}$$

By (6.14) and Lemma 6.3.10, any path that is in $G_\cap$ at all levels up to and including M in $do(A^*, S^*)$ must also be G_R-accessible at all levels n in $do(A^*, S^*)$ s.t. $n \leq M$:

$$\forall p.\ (\forall n.\ n \leq M\ \supset G_\cap(p, n, do(A^*, S^*))) \supset (\forall n'.\ n' \leq M \supset G_R(p, n', do(A^*, S^*))). \tag{6.15}$$

Now, by Definitions 6.2.7 and 6.2.8, a path p is in ProgressedAndFiltered at level

$M+1$ after A^* has happened in S^* iff:

if $(\exists p'.\ G_\cap(p', M, do(A^*, S^*)) \wedge \text{Progressed}(p', M+1, A^*, S^*)$

$\wedge\ \exists s'.\ \text{Starts}(p', s') \wedge K(s', do(A^*, S^*)))$

then $\text{Progressed}(p, M+1, A^*, S^*) \wedge \exists s'.\ \text{Starts}(p, s') \wedge K(s', do(A^*, S^*))$

else $\exists s.\ \text{Starts}(p, s) \wedge K(s, do(A^*, S^*))$.

Note that, if the condition in the **if** is false, then P_1 is in ProgressedAndFiltered at level $M+1$ after A^* has happened in S^* if it starts with a situation that is K-accessible in $do(A^*, S^*)$. This follows trivially from the inductive hypothesis and Definitions 6.2.7 and 6.2.8, since any path that is in the ProgressedAndFiltered (and the Progressed$_{\text{CA}}$) relation in $do(A^*, S^*)$ must by definition start with a K-accessible situation in $do(A^*, S^*)$.

On the other hand, if there is a path p' s.t. p' is in the $G_\cap$ relation at M in $do(A^*, S^*)$ and p' is also the suffix of a G-accessible path at level $M+1$ in S^* relative to A^*, then set P'_1 to be p'. Thus we have: ProgressedAndFiltered$(P'_1, M+1, A^*, S^*)$. Let me show that P'_1 is also in the ProgressedAndFiltered relation at all higher priority levels than $M+1$. To this end, first note that by the assumption of this subcase that $G_\cap(P'_1, M, do(A^*, S^*))$ and Lemma 6.3.11, it follows that for all n s.t. $0 \leq n \leq M$, $G_\cap(P'_1, n, do(A^*, S^*))$ holds. Moreover, from this, (6.15), and Definition 4.2.4, we have that $\forall n.\ n \leq M \supset G(P_1', n, do(A^*, S^*))$. From this, Axiom 6.2.6, and the fact that A^* is a non-adopt/drop action, we have that $\forall n.\ n \leq M \supset$ ProgressedAndFiltered(P_1', n, A^*, S^*). So there exists a path p s.t. $\forall n.$ ProgressedAndFiltered(p, n, A^*, S^*).

<u>A^* is an *adopt* action</u>: Fix ϕ_1 and N_1, and assume that $A^* = adopt(\phi_1, N_1)$. Now, from (6.13), Axiom 6.2.3, and Definitions 4.2.10 and 4.2.9, we have that there is a path, say P_1, such that P_1 starts with some situation S_1, P_1 is $G_\cap$-accessible at all levels in S^*, and the next action performed over P_1 is $A^* = adopt(\phi_1, N_1)$, after which ϕ_1 holds over the suffix, say P_2, of this path that starts with $do(A^*, S_1)$:

$$\forall n.\ G_\cap(P_1, n, S^*) \wedge \text{Starts}(P_1, S_1) \wedge \text{Suffix}(P_2, P_1, do(A^*, S_1)) \wedge \phi_1(P_2). \quad (6.16)$$

From (6.16) and Lemma 6.3.8, it follows that:

$$\exists s.\ \text{Starts}(P_1, s) \wedge K(s, S^*). \quad (6.17)$$

Again, from (6.16) and Corollary 3.5.41, it follows that:

$$\exists s.\ \text{Starts}(P_1, s) \wedge \text{Poss}(A^*, s). \quad (6.18)$$

Moreover, since A^*, which is an *adopt* action, is not a knowledge-producing action, it follows from (6.17), (6.18), and the SSA for K (i.e. Axiom 3.4.10) that:

$$\exists s.\ \text{Starts}(P_2, s) \wedge K(s, do(A^*, S^*)). \quad (6.19)$$

Note that to prove that $\exists p.\ \forall n.\ G(p, n, do(A^*, S^*))$, by Axiom 6.2.6 and the fact that A^* is an *adopt* action it suffices to show that $\forall n.$ $\text{Adopted}_{\text{CA}}(P_2, n, N_1, A^*, S^*, \phi_1)$. First consider the case where $n = N_1$. By Definition 6.2.9, $\text{Adopted}_{\text{CA}}(P_2, N_1, N_1, A^*,$ $S^*, \phi_1)$ iff P_2 starts with a K-accessible situation in $do(A^*, S^*)$ and $\phi_1(P_2)$. The former follows from (6.19). On the other hand, the latter follows from (6.16).

Next consider the case where $n < N_1$. By Definitions 6.2.9, 6.2.8, and 4.3.4, and (6.19), to show that $\forall n.\ n < N_1 \supset \text{Adopted}_{\text{CA}}(P_2, n, N_1, A^*, S^*, \phi_1)$, it is sufficient to prove that $\forall n.\ n < N_1 \supset G_R(P_1, n, S^*) \wedge \text{Suffix}(P_2, P_1, do(A^*, S^*))$. Now consider the case where $n > N_1$. Again by Definitions 6.2.9, 6.2.8, and 4.3.4, and (6.19), to show that $\forall n.\ n > N_1 \supset \text{Adopted}_{\text{CA}}(P_2, n, N_1, A^*, S^*, \phi_1)$, it is sufficient to prove that $\forall n.\ n > N_1 \supset G_R(P_1, n-1, S^*) \wedge \text{Suffix}(P_2, P_1, do(A^*, S^*))$. Thus, to cover both these cases, I need to show that:

$$\forall n.\ G_R(P_1, n, S^*) \wedge \text{Suffix}(P_2, P_1, do(A^*, S^*)).$$

From this and (6.16), I only need to show that $\forall n.\ G_R(P_1, n, S^*)$. Now, from (6.11) and Corollary 6.3.5, it follows that $\exists p.\ \forall n.\ G_R(p, n, S^*)$. Finally, from this, (6.16), and Lemma 6.3.10, it follows that $\forall n.\ G_R(P_1, n, S^*)$. Thus, there exists a path p, namely P_2, s.t. $\forall n.\ \text{Adopted}_{\text{CA}}(p, n, N_1, A^*, S^*, \phi_1)$.

A^* is an *adoptRelTo* action: The proof for this case is similar to the proof for *adopt* actions.

$\underline{A^* \text{ is an } \mathit{drop} \text{ action}}$: The proof for this case is similar to the one for regular actions. It thus follows that $\exists p.\ \forall n.\ G(p, n, do(A^*, S^*))$. □

As a consequence of Proposition 6.3.12, I can show that if a path p is in the $G_\cap$ relation at some level n for some executable situation s, then p must be G-accessible at all higher priority levels than n in s and at n in s:

Corollary 6.3.13.

$$\mathcal{D}^{SG}_{CAgt} \models \forall p, n, s.\ \text{Executable}(s) \land G_\cap(p, n, s) \supset (\forall m.\ m \leq n \supset G(p, m, s)).$$

Proof. Fix P_1, N_1, and S_1, and assume that:

$$\text{Executable}(S_1), \tag{6.20}$$

$$G_\cap(P_1, N_1, S_1). \tag{6.21}$$

From (6.20) and Proposition 6.3.12, it follows that $\exists p.\ \forall n.\ G(p, n, S_1)$. From this and Corollary 6.3.5, it follows that $\exists p.\ \forall n.\ G_R(p, n, S_1)$. Finally, from this, (6.21), and Lemma 6.3.10, it follows that $\forall n.\ n \leq N_1 \supset G_R(P_1, n, S_1)$. The consequent follows from this and Definition 4.2.4. □

Moreover, I can show that in this framework all of the p-goals of an agent are active and thus her chosen goals are just the intersection of her p-goals. More precisely, for any executable situation s, a committed agent's c-goal- or $G_\cap$-accessible paths in s are exactly those that are G_R-accessible (and G-accessible) at all levels in s, i.e. those that satisfy all her realistic p-goals (and p-goals):

Proposition 6.3.14.

(a). $\mathcal{D}^{SG}_{CAgt} \models \forall s.\ \text{Executable}(s) \supset (\forall p.\ G_\cap(p, s) \equiv \forall n.\ G_R(p, n, s))$,

(b). $\mathcal{D}^{SG}_{CAgt} \models \forall s.\ \text{Executable}(s) \supset (\forall p.\ G_\cap(p, s) \equiv \forall n.\ G(p, n, s))$.

Proof. (a). Fix situation S_1 such that Executable(S_1). By Definition 4.2.9, I need to show that $\forall p.\ (\forall n.\ G_\cap(p, n, S_1)) \equiv (\forall n.\ G_R(p, n, S_1))$. First let us consider the ($\subset$) direction. Fix P_1 and assume:

$$\forall n.\ G_R(P_1, n, S_1). \tag{6.22}$$

I'll prove this by induction on n. For the base case, by (6.22) and Axiom 4.2.7, we have $G_\cap(P_1, 0, S_1)$. For the inductive case, fix M and assume $G_\cap(P_1, M, S_1)$. Then from Axiom 4.2.7, it also follows that $G_\cap(P_1, M+1, S_1)$, as by (6.22) we have $G_R(P_1, M+1, S_1)$ and by the inductive hypothesis we have $G_\cap(P_1, M, S_1)$.

Next, let us consider the ($\supset$) direction. Fix path P_1 and assume:

$$\forall n.\ G_\cap(P_1, n, S_1). \tag{6.23}$$

Since S_1 is executable, by Proposition 6.3.12 and Corollary 6.3.5 it follows that there is a path, say P_2, that is G_R-accessible at all levels in S_1:

$$\forall n.\ G_R(P_2, n, S_1). \tag{6.24}$$

Then it follows from (6.24), (6.23), and Lemma 6.3.10 that $\forall n.\ G_R(P_1, n, S_1)$. □

(b). Follows from Proposition 6.3.14 (a) and Corollary 6.3.5. □

Using this proposition, it can be shown that for any executable situation s, if an agent has a p-goal that ϕ at some level n in s, then she has it as her c-goal in s:

Proposition 6.3.15.

$$\mathcal{D}^{SG}_{CAgt} \models \forall n, s.\ \text{Executable}(s) \land \text{PGoal}(\phi, n, s) \supset \text{CGoal}(\phi, s).$$

Proof. Fix ϕ_1, N_1, and S_1. By Definition 4.2.1 and the antecedent, we have: $\forall p.\ G(p, N_1, S_1) \supset \phi_1(p)$. From this and Definition 4.2.4, we have:

$$\forall p.\ G_R(p, N_1, S_1) \supset \phi_1(p). \tag{6.25}$$

Since by the antecedent S_1 is executable, we can apply Proposition 6.3.14 (a), and get that:

$$\forall p.\ G_\cap(p, S_1) \supset G_R(p, N_1, S_1). \tag{6.26}$$

From (6.25) and (6.26), it follows that: $\forall p.\ G_\cap(p, S_1) \supset \phi_1(p)$. The consequent follows from this and Definition 4.2.10. □

Moreover, for any executable situation s, the consequences of an agent's p-goals in s are also her c-goals in s:

Proposition 6.3.16.

$$\mathcal{D}^{SG}_{CAgt} \models \forall s.\ \text{Executable}(s) \supset$$
$$(\forall p, n_1, n_2.\ \text{PGoal}(\phi_1, n_1, s) \wedge \text{PGoal}(\phi_2, n_2, s) \wedge (\phi_1(p) \wedge \phi_2(p) \supset \psi(p))$$
$$\supset \text{CGoal}(\psi, s)).$$

Proof Sketch. Analogously to the proof of Proposition 6.3.15, it can be shown that both ϕ_1 and ϕ_2 holds over all $G_\cap$-accessible paths in s, and since $\forall p.\ \phi_1(p) \wedge \phi_2(p) \supset \psi(p)$, so does ψ. The consequent follows from this and Definition 4.2.10. $\square$

It is easy to see that this proposition can be generalized for more than two p-goals.

As a consequence of Proposition 6.3.15, I can show that for any executable situation, an agent's p-goals at any level are individually consistent:

Corollary 6.3.17.

$$\mathcal{D}^{SG}_{CAgt} \models \forall s.\ \text{Executable}(s) \supset \forall n.\ \neg\text{PGoal}(\text{False}, n, s).$$

Proof. (By contradiction) Fix S_1 and N_1 and assume: Executable$(S_1) \wedge$ PGoal(False, $N_1, S_1)$. Then by Proposition 6.3.15, we have CGoal(False, S_1). But by Proposition 6.3.1, we have $\neg$CGoal(False, S_1) – a contradiction. $\square$

Moreover for any executable situation, an agent's p-goals at all levels are collectively consistent:

Corollary 6.3.18.

$$\mathcal{D}_{CAgt}^{SG} \models \forall n, s.\ \text{Executable}(s) \land \text{PGoal}(\phi, n, s) \supset \neg\exists n'.\ \text{PGoal}(\neg\phi, n', s).$$

Proof. (By contradiction) Fix S_1, ϕ_1, N_1, and N_2 and assume that:

$$\text{Executable}(S_1), \tag{6.27}$$

$$\text{PGoal}(\phi_1, N_1, S_1), \tag{6.28}$$

$$\text{PGoal}(\neg\phi_1, N_2, S_1). \tag{6.29}$$

By (6.27), (6.28), and Proposition 6.3.15, we have CGoal(ϕ_1, S_1), while from (6.27), (6.29), and Proposition 6.3.15, we have CGoal$(\neg\phi_1, S_1)$. Thus we have CGoal(False, $S_1)$. But by Proposition 6.3.1, we have $\neg$CGoal(False, S_1) – a contradiction. □

Again for any executable situation, an agent's p-goals are not known to be impossible:

Corollary 6.3.19.

$$\mathcal{D}_{CAgt}^{SG} \models \forall n, s.\ \text{Executable}(s) \land \text{PGoal}(\phi, n, s) \supset \neg\text{KImpossible}(\phi, s).$$

Proof. Fix ϕ_1, N_1, and S_1 and assume that Executable$(S_1) \land$PGoal(ϕ_1, N_1, S_1). From this and Proposition 6.3.15, we have CGoal(ϕ_1, S_1). Finally, from this and Corollary 6.3.3, we have $\neg$KImpossible(ϕ_1, S_1). □

Finally for any executable situation s, having the p-goal that ϕ at some level n in s is equivalent to having the primary c-goal that ϕ at n in s:

Corollary 6.3.20.

$$\mathcal{D}_{CAgt}^{SG} \models \forall s.\ \text{Executable}(s) \supset (\forall n.\ \text{PGoal}(\phi, n, s) \equiv \text{PrimCGoal}(\phi, n, s)).$$

Proof Sketch. Fix ϕ_1, N_1, and S_1 and assume that:

$$\text{Executable}(S_1). \tag{6.30}$$

I have to show that $\text{PGoal}(\phi_1, N_1, S_1) \equiv \text{PrimCGoal}(\phi_1, N_1, S_1)$. The ($\subset$) direction trivially follows from the definition of $\text{PrimCGoal}(\phi_1, N_1, S_1)$ (Definition 4.2.14).

For the ($\supset$) direction, I need to show that $\exists p.\ G(p, N_1, S_1) \wedge G_\cap(p, N_1, S_1)$ (by Definition 4.2.14). From (6.30) and Proposition 6.3.12, it follows that there is a path, say P_1, s.t.:

$$\forall n.\ G(P_1, n, S_1), \text{ and thus} \tag{6.31}$$

$$G(P_1, N_1, S_1). \tag{6.32}$$

From (6.31) and Corollary 6.3.5, we have $\forall n.\ G_R(P_1, n, S_1)$, and thus $\forall n.\ n \leq N_1 \supset G_R(P_1, n, S_1)$. Finally, from this and Lemma 6.3.9, it follows that $G_\cap(P_1, N_1, S_1)$. Thus from this and (6.32), it follows that $\exists p.\ G(p, N_1, S_1) \wedge G_\cap(p, N_1, S_1)$. $\square$

Note that none of the main results that I presented in Section 6.3.2 hold for optimizing agents (as expected, Lemmata 6.3.8, 6.3.9, and 6.3.10, which deal solely with properties of Axiom 4.2.7, hold for the optimizing agent framework as well). In particular, recall that an optimizing agent is allowed to have true desires, which can be individually inconsistent, known to be impossible to bring about, or inconsistent with each other and with what she knows. As discussed in Chapter 4, in some of these cases, the agent's sets of G- or G_R- accessible paths at some levels can be empty. Also, some of her lower priority p-goals or desires can be inactive. In contrast, the above results show that in the committed agent framework, all p-goal levels are always considered to be active, and thus can't be empty. The agent's p-goals and realistic p-goals are the same. Finally, her chosen goals or intentions are really the consequential closure of her prioritized goals.

6.3.3 Dynamic Properties II: Extensionality, Adoption, and Drop

I next discuss some properties of a committed agent's goal dynamics. First of all, I can show that a committed agent always wants to be in a situation that has the same action history as the current situation:

Proposition 6.3.21 (Correct Action History)**.**

$$\forall p, n, s.\ G(p, n, s) \supset \exists s'.\ \text{Starts}(p, s') \wedge \text{SameHist}(s, s').$$

Proof. Follows from Assumption 6.2.1, Proposition 6.3.4, and Lemma 3.5.34. $\square$

In contrast, by Proposition 4.4.6 an optimizing agent only wants to be in a world that has the same action history as the current situation, if initially she wants to be in an initial world. Such an additional initial condition is not required in the committed agent framework as Assumption 6.2.1 along with Lemma 3.5.34 already ensure that this is the case.

The next proposition says that adopting and dropping logically equivalent goals has the same result:

Proposition 6.3.22 (Extensionality w.r.t. Adoption and Drop)**.**

(a). $\mathcal{D}_{CAgt}^{SG} \models (\forall p.\ \phi_1(p) \equiv \phi_2(p)) \supset$

$(\forall n, n', s.\ \text{PGoal}(\psi, n', do(adopt(\phi_1, n), s)) \equiv \text{PGoal}(\psi, n', do(adopt(\phi_2, n), s))),$

(b). $\mathcal{D}_{CAgt}^{SG} \models (\forall p.\ \phi_1(p) \equiv \phi_2(p)) \supset$

$(\forall n, s.\ \text{PGoal}(\psi, n, do(drop(\phi_1), s)) \equiv \text{PGoal}(\psi, n, do(drop(\phi_2), s))),$

(c). $\mathcal{D}_{CAgt}^{SG} \models (\forall p.\ \phi_1(p) \equiv \phi_2(p)) \supset$

$(\forall n, s.\ \text{CGoal}(\psi, do(adopt(\phi_1, n), s)) \equiv \text{CGoal}(\psi, do(adopt(\phi_2, n), s))),$

(d). $\mathcal{D}_{CAgt}^{SG} \models (\forall p.\ \phi_1(p) \equiv \phi_2(p)) \supset$

$(\forall s.\ \text{CGoal}(\psi, do(drop(\phi_1), s)) \equiv \text{CGoal}(\psi, do(drop(\phi_2), s))).$

Proof. (a). Follows from the fact that we use a possible worlds/paths semantics for p-goals. □

(b). Similar to that of Proposition 6.3.22(a). □

(c). Follows from Definition 4.2.10 and the fact that the $G_\cap$-accessible paths are the same in both situations given the antecedent. □

(d). Similar to that of Proposition 6.3.22(c). □

Note that although by Proposition 6.3.15, for any executable situation an agent's p-goals are also her c-goals in this framework, Proposition 6.3.22(a) and (b) alone are inadequate for capturing extensionality of chosen goals w.r.t. $adopt$ and $drop$; the reason for this is that the agent might have a c-goal ϕ without necessarily having the p-goal that ϕ, e.g. ϕ can be a consequence of two or more p-goals at different priority levels. Thus (c) and (d) above are also required. Also, as shown in Chapter 4, these results also hold for optimizing agents.

Moreover, adopting logically equivalent subgoals relative to logically equivalent parent goals yields the same goal state:

Proposition 6.3.23 (Extensionality w.r.t. Subgoal Adoption)**.**

(a). $\mathcal{D}^{SG}_{CAgt} \models \forall p.(\phi_1(p) \equiv \phi_2(p)) \wedge \forall p.(\psi_1(p) \equiv \psi_2(p)) \supset$

$$\text{PGoal}(\phi^*, do(adoptRelTo(\psi_1, \phi_1), s)) \equiv \text{PGoal}(\phi^*, do(adoptRelTo(\psi_2, \phi_2), s)),$$

(b). $\mathcal{D}^{SG}_{CAgt} \models \forall p.(\phi_1(p) \equiv \phi_2(p)) \wedge \forall p.(\psi_1(p) \equiv \psi_2(p)) \supset$

$$\text{CGoal}(\phi^*, do(adoptRelTo(\psi_1, \phi_1), s)) \equiv \text{CGoal}(\phi^*, do(adoptRelTo(\psi_2, \phi_2), s)).$$

Proof. (a). Similar to that of Proposition 6.3.22(a). □

(b). Similar to that of Proposition 6.3.22(c). □

Again, as shown in Chapter 5, these results hold for optimizing agents as well.

As a consequence of Proposition 6.3.22(a),(b), and 6.3.23(a), these properties also hold for a committed agent's primary c-goals:

Corollary 6.3.24.

(a). $\mathcal{D}^{SG}_{CAgt} \models \forall p.\ (\phi_1(p) \equiv \phi_2(p)) \supset$

$(\forall n, s.\ \text{PrimCGoal}(\psi, do(adopt(\phi_1, n), s))$

$\equiv \text{PrimCGoal}(\psi, do(adopt(\phi_2, n), s))),$

(b). $\mathcal{D}^{SG}_{CAgt} \models \forall p.\ (\phi_1(p) \equiv \phi_2(p)) \supset$

$\forall s.\ \text{PrimCGoal}(\psi, do(drop(\phi_1), s)) \equiv \text{PrimCGoal}(\psi, do(drop(\phi_2), s)).$

(c). $\mathcal{D}^{SG}_{CAgt} \models \forall p.\ (\phi_1(p) \equiv \phi_2(p)) \wedge \forall p.\ (\psi_1(p) \equiv \psi_2(p)) \supset$

$(\forall s.\ \text{PrimCGoal}(\phi^*, do(adoptRelTo(\psi_1, \phi_1), s))$

$\equiv \text{PrimCGoal}(\phi^*, do(adoptRelTo(\psi_2, \phi_2), s))).$

Proof. (a). Follows from Definition 4.2.12 and Proposition 6.3.22(a). □

(b). Follows from Definition 4.2.12 and Proposition 6.3.22(b). □

(c). Follows from Definition 4.2.12 and Proposition 6.3.23(a). □

Once again, as I showed earlier, these also hold for the optimizing agent framework.

The next few properties deal with (sub)goal adoption and drop and confirm that adopting and dropping (sub)goals has the intended effect. I start by showing that after adopting ϕ at level n in situation s, an agent acquires the p-goal that ϕ at n:

Proposition 6.3.25 (PGoal Adoption)**.**

$$\mathcal{D}^{SG}_{CAgt} \models \forall n, s.\ \text{PGoal}(\phi, n, do(adopt(\phi, n), s)).$$

Proof. Fix ϕ_1, N_1, and S_1. From Axiom 6.2.6 and Definition 6.2.9, we have that the agent's G-accessible paths at N_1 in $do(adopt(\phi_1, N_1), S_1)$ are the ones that start with situations that are K-accessible in $do(adopt(\phi_1, N_1), S_1)$ and over which ϕ_1 holds:

$$\begin{gathered}\forall p.\ G(p, N_1, do(adopt(\phi_1, N_1), S_1)) \equiv \\ \exists s.\ \text{Starts}(p, s) \wedge K(s, do(adopt(\phi_1, N_1), S_1)) \wedge \phi_1(p).\end{gathered} \tag{6.33}$$

If such a path p exists, then the consequent follows from (6.33) and Definition 4.2.1. Otherwise, the consequent holds trivially from Definition 4.2.1. $\square$

Note that the above proof relies on the argument that if there does not exist a path that starts with a situation that is K-accessible in $do(adopt(\phi, n), s)$ and over which ϕ holds, then the agent trivially has the p-goal that ϕ at n after $adopt(\phi, n)$ has been performed in s. However, Axiom 6.2.3 ensures that in this framework for any executable situation s, such a path indeed exists. Note that the property in Proposition 6.3.25 also holds for optimizing agents as shown in Proposition 4.4.9.

Moreover, after adopting a p-goal ϕ at some level n in some situation s, an agent also acquires the c-goal that ϕ provided that s is an executable situation and that the $adopt(\phi, n)$ action is executable in s:

Proposition 6.3.26 (CGoal Adoption)**.**

$$\mathcal{D}^{SG}_{CAgt} \models \forall n, s.\ \text{Executable}(s) \wedge \text{Poss}(adopt(\phi, n), s))$$
$$\supset \text{CGoal}(\phi, do(adopt(\phi, n), s)).$$

Proof. Follows from Definition 3.3.1, Propositions 6.3.25, and 6.3.15. □

Furthermore, the above result also holds for a committed agent's primary c-goals:

Proposition 6.3.27 (Primary CGoal Adoption)**.**

$$\mathcal{D}^{SG}_{CAgt} \models \forall n, s.\ \text{Executable}(s) \wedge \text{Poss}(adopt(\phi, n), s)$$
$$\supset \text{PrimCGoal}(\phi, do(adopt(\phi, n), s)).$$

Proof. Follows from Definition 3.3.1, Proposition 6.3.25, and Corollary 6.3.20. □

In Proposition 4.4.12, a variation of this property is shown to hold for optimizing agents as well. However, in that case, one needs to ensure that the agent does not intend not to execute the $adopt(\phi, n)$ action next, and ϕ is consistent with all her higher priority c-goals in s, i.e. her c-goals up to level $n-1$. Note that in the above proposition, this is guaranteed by the preconditions of the $adopt$ action and the antecedent which ensures that the $adopt(\phi, n)$ action is possible in s.

I can also show that if an agent has a p-goal that ϕ at some level n in some situation s, then she will not have the progression of ϕ as her realistic p-goal after she drops it, provided that it is not the case that the progression of ϕ has become known to be inevitable after the $drop$ action happens in s:

Proposition 6.3.28 (RPGoal Drop)**.**

$$\mathcal{D}^{SG}_{CAgt} \models \forall n, s.\ (\text{PGoal}(\phi, n, s)$$
$$\wedge \neg\text{KInevitable}(\text{ProgOf}(\phi, drop(\phi)), do(drop(\phi), s)))$$
$$\supset \neg\text{RPGoal}(\text{ProgOf}(\phi, drop(\phi)), n, do(drop(\phi), s)).$$

Proof. Fix ϕ_1, N_1, and S_1. From the antecedent, we have:

$$\text{PGoal}(\phi_1, N_1, S_1), \tag{6.34}$$

$$\neg\text{KInevitable}(\text{ProgOf}(\phi_1, drop(\phi_1)), do(drop(\phi_1), S_1)). \tag{6.35}$$

We can see from Axiom 6.2.6 and Definition 6.2.11 that after the $drop(\phi_1)$ action has been performed in S_1, each G-accessibility level in S_1 where ϕ_1 is a p-goal is replaced by the set of paths that start with a situation that is K-accessible in $do(drop(\phi_1), S_1)$. Thus, by (6.34), Axiom 6.2.6, and Definition 6.2.11, we have:

$$G(p, N_1, do(drop(\phi_1), S_1)) \equiv \exists s'.\ \text{Starts}(p, s') \wedge K(s', do(drop(\phi_1), S_1)). \tag{6.36}$$

By (6.35) and Definitions 3.5.14 and 3.5.12, there exists a path P_1 that starts with a situation that is K-accessible from $do(drop(\phi_1), S_1)$, and over which the formula

$\neg$ProgOf$(\phi_1, drop(\phi_1))$ holds:

$$\exists s'.\ \text{Starts}(P_1, s') \wedge K(s', do(drop(\phi_1), S_1)) \wedge \neg\text{ProgOf}(\phi_1, drop(\phi_1))(P_1). \quad (6.37)$$

Thus by (6.36) and (6.37), we have $G(P_1, N_1, do(drop(\phi_1), S_1)))$. The consequent follows from this, (6.37), and Definitions 4.2.4 and 4.2.5. $\square$

Note that, in contrast to the corresponding Proposition 4.4.15 which uses the condition in the antecedent that the progression of ϕ is not strongly inevitable in $do(drop(\phi), s)$, this proposition requires that progression of ϕ is not known to be inevitable in $do(dro-p(\phi), s)$. This is due to the fact that Assumption 6.2.1 and Proposition 6.3.4 ensure that in this framework all G-accessible paths in situation $do(drop(\phi), s)$ always start with a K-accessible situation in $do(drop(\phi), s)$. Thus as long as there is a path that starts with a K-accessible situation in $do(drop(\phi), s)$ and over which the progression of ϕ does not hold, the proposition follows.

Next, I show that similar to an optimizing agent, a committed agent acquires the p-goal that ψ after she adopts it as a subgoal of another goal ϕ in s, provided that she has the p-goal at some level in s that ϕ:

Proposition 6.3.29 (Subgoal Adoption-1)**.**

$$\mathcal{D}^{SG}_{CAgt} \models \exists m.\ \text{PGoal}(\phi, m, s) \supset \exists n.\ \text{PGoal}(\psi, n, do(adoptRelTo(\psi, \phi), s)).$$

Proof. Fix ϕ_1, ψ_1, M_1, and S_1. From the antecedent, we have:

$$\text{PGoal}(\phi_1, M_1, S_1). \tag{6.38}$$

Fix N_1 such that AdoptedLevel(ϕ_1, N_1, S_1) holds. By (6.38) and Definition 5.2.4, such a level N_1 indeed exists. By Axiom 6.2.6 and Definitions 6.2.10, 6.2.8, and 5.2.4, the agent's G-accessible paths in $do(adoptRelTo(\psi_1, \phi_1), S_1)$ at level N_1 are the ones that can be obtained by progressing her G-accessible paths at $N_1 - 1$ in S_1, that start with a K-accessible situation in $do(adoptRelTo(\psi_1, \phi_1), S_1)$, and over which ψ_1 holds; thus we have:

$$\forall p.\ G(p, N_1, do(adoptRelTo(\psi_1, \phi_1), S_1)) \supset \psi_1(p). \tag{6.39}$$

If such a path exists, then the consequent follows from (6.39) and Definition 4.2.1. Otherwise, the consequent follows trivially by Definition 4.2.1. □

Also, I can show that an agent acquires the c-goal that ψ after she adopts it as a subgoal of another goal ϕ in s, provided that s is an executable situation and the $adoptRelTo$ action is executable in s:

Proposition 6.3.30 (Subgoal Adoption-2)**.**

$$\mathcal{D}^{SG}_{CAgt} \models \forall s.\ \text{Executable}(s) \wedge \text{Poss}(adoptRelTo(\psi, \phi), s) \supset$$
$$\text{CGoal}(\psi, do(adoptRelTo(\psi, \phi), s)).$$

Proof. Follows from Definition 3.3.1, Propositions 6.3.29, and 6.3.15. $\square$

Furthermore, the above result also holds for a committed agent's primary c-goals:

Proposition 6.3.31 (Subgoal Adoption-3)**.**

$$\mathcal{D}^{SG}_{CAgt} \models \forall s.\ \text{Executable}(s) \land \text{Poss}(\mathit{adoptRelTo}(\psi, \phi), s) \supset$$
$$\text{PrimCGoal}(\psi, \mathit{do}(\mathit{adoptRelTo}(\psi, \phi), s)).$$

Proof. Follows from Definition 3.3.1, Proposition 6.3.29, and Corollary 6.3.20. $\square$

In contrast, the corresponding proposition for optimizing agents in Chapter 5 (i.e. Proposition 5.3.2) requires that (a) the parent goal ϕ is a primary c-goal at level $n-1$ in s, where n is the adopted level, and that (b) the agent does not intend not to execute the $\mathit{adoptRelTo}(\psi, \phi)$ action next and she does not have the c-goal up to the adopted level that $\neg\psi$ next. Again, note that in the above proposition, we only require the parent goal ϕ to be a p-goal at some level; but this and Corollary 6.3.20 imply that ϕ is a primary c-goal in s. Moreover, (b) also follows from the preconditions of the $\mathit{adoptRelTo}(\psi, \phi)$ action and the antecedent that this action is executable in s.

The last two properties in this section show that dropping subgoals and their supergoals works as intended. First, I prove that subgoals are dropped when their parent goal is dropped. More precisely, I show that after dropping the p-goal that ϕ in s, an agent does not have the p-goal (and thus the primary c-goal) that the progression of ψ

at some level n, provided that ψ is a subgoal of ϕ in s, that ψ is a p-goal at n in s, and that the progression of ψ is not known to be inevitable in $do(drop(\phi), s)$:

Proposition 6.3.32 (Supergoal Drop)**.**

$$\begin{aligned}\mathcal{D}^{SG}_{CAgt} \models\ & \text{SubGoal}(\psi, \phi, s) \wedge \text{PGoal}(\psi, n, s) \\ & \wedge \neg\text{KInevitable}(\text{ProgOf}(\psi, drop(\phi)), do(drop(\phi), s)) \\ & \supset \neg\text{PGoal}(\text{ProgOf}(\psi, drop(\phi)), n, do(drop(\phi), s)).\end{aligned}$$

Proof. Fix ϕ_1, ψ_1, N_1 and S_1. From the antecedent, we have:

$$\text{SubGoal}(\psi_1, \phi_1, S_1), \tag{6.40}$$

$$\text{PGoal}(\psi_1, N_1, S_1), \tag{6.41}$$

$$\neg\text{KInevitable}(\text{ProgOf}(\psi_1, drop(\phi_1)), do(drop(\phi_1), S_1)). \tag{6.42}$$

From (6.40) and Definition 5.2.5, it follows that for any level n in S_1 where ψ_1 is a p-goal, ϕ_1 is also a p-goal:

$$\forall n.\ \text{PGoal}(\psi_1, n, S_1) \supset \text{PGoal}(\phi_1, n, S_1). \tag{6.43}$$

From (6.42) and Definitions 3.5.14, 3.5.12, and 3.4.5, it follows that there is a path P_1 such that:

$$\exists s.\ K(s, do(drop(\phi_1), S_1)) \wedge \text{Starts}(P_1, s) \wedge \neg\text{ProgOf}(\psi_1, drop(\phi_1))(P_1). \tag{6.44}$$

Now, consider level N_1 in S_1; by (6.41), ψ_1 is a p-goal at N_1 in S_1. By this, (6.43), PGoal(ϕ_1, N_1, S_1), and thus by Axiom 6.2.6, and Definition 6.2.11, we can see that the G-accessible paths at N_1 in $do(drop(\phi_1), S_1)$ are the ones that start with situations that are K-accessible in $do(drop(\phi_1), S_1)$. Since by (6.44), P_1 is such a path, it will be included in the G-relation at N_1 in $do(drop(\phi_1), S_1)$:

$$G(P_1, N_1, do(drop(\phi_1), S_1)). \tag{6.45}$$

The consequent thus follows from (6.44), (6.45), and Definition 4.2.1. □

Again, note that the corresponding optimizing agent result (i.e. Proposition 5.3.3) uses StronglyInevitable in the antecedent rather than KInevitable, since unlike in the committed agent framework, G-accessible paths are not required to start with a K-accessible situation in $do(drop(\phi), s)$ in that framework. Also, as with the optimizing agent framework, this proposition does not hold if we replace PGoal in the consequent with CGoal since ψ could be a consequence of a combination of two or more p-goals, i.e. a non-primary c-goal. Finally, as in the optimizing agent framework, this property can be generalized to show that in addition to the above, ψ is indeed not a p-goal at some level n where $\neg$PGoal(ψ, n, s) after the $drop(\phi)$ action happens in s, provided that she don't have the p-goal at n in s that the $drop(\phi)$ action does not happen next or ψ holds after it happens, i.e. that $\neg$PGoal$(\neg\exists s'.$ Do$(drop(\phi), now, s') \vee \psi, n, s)$.

The next property formalizes the conditions under which the dropping of a subgoal does not affect the parent goal. It states that an agent retains the p-goal that the progression of ϕ after she drops a subgoal ψ of some goal ϕ in some executable situation s, provided that the $drop$ action is possible in s, and that she does not have the c-goal in s that the $drop$ action does not happen next:

Proposition 6.3.33 (Subgoal Drop)**.**

$$\begin{aligned}\mathcal{D}^{SG}_{CAgt} \models \forall s.\ (&\text{Executable}(s) \wedge \text{Poss}(drop(\psi), s) \wedge \text{SubGoal}(\psi, \phi, s)\\ &\wedge \neg\text{CGoal}(\neg\exists s'.\ \text{Starts}(s') \wedge \text{OnPath}(do(drop(\psi), s')), s))\\ &\supset \exists n.\ \text{PGoal}(\text{ProgOf}(\phi, drop(\psi)), n, do(drop(\psi), s)).\end{aligned}$$

Proof. Fix ϕ_1, ψ_1, and S_1. From the antecedent, we have:

$$\text{Executable}(S_1) \wedge \text{Poss}(drop(\psi_1), S_1), \tag{6.46}$$

$$\text{SubGoal}(\psi_1, \phi_1, S_1),\ \text{and} \tag{6.47}$$

$$\neg\text{CGoal}(\neg\exists s'.\ \text{Starts}(s') \wedge \text{OnPath}(do(drop(\psi_1), s')), S_1). \tag{6.48}$$

From (6.47) and Definition 5.2.5, it follows that there is a level N_1 in S_1 where ϕ_1 is a p-goal but ψ_1 is not, and for any level n that has priority higher or equal to N_1, ψ_1 is not a p-goal at n in S_1:

$$\text{PGoal}(\phi_1, N_1, S_1), \tag{6.49}$$

$$\forall n.\ n \leq N_1 \supset \neg\text{PGoal}(\psi_1, n, S_1). \tag{6.50}$$

Also, by (6.48) and Definitions 4.2.10 and 4.2.9, there is a path, say P_1, that is in the $G_\cap$ relation at all levels n, and where the next action performed on the path is $drop(\psi_1)$:

$$\forall n.\ G_\cap(P_1, n, S_1), \text{ and} \tag{6.51}$$

$$\exists s'.\ \text{Starts}(P_1, s') \wedge \text{OnPath}(P_1, do(drop(\psi_1), s')). \tag{6.52}$$

By (6.51) and Lemma 6.3.8, it follows that P_1 starts with a K-accessible situation in S_1:

$$\exists s.\ \text{Starts}(P_1, s) \wedge K(s, S_1). \tag{6.53}$$

From (6.46) and Definition 3.3.1, it follows that:

$$\text{Executable}(do(drop(\psi_1), S_1)).$$

From this, (6.51), and Proposition 6.3.14(b), it follows that:

$$\forall n.\ G(P_1, n, S_1). \tag{6.54}$$

Also, from this, it follows that:

$$G(P_1, N_1, S_1). \tag{6.55}$$

Again, from this and (6.49), it follows that:

$$\phi_1(P_1). \tag{6.56}$$

Now, consider the suffix of P_1 that starts with $do(drop(\psi_1), S_{P_1})$, where S_{P_1} is the starting situation of P_1; let us call it P_2. It follows from (6.56) and Definition 4.4.14 that ProgOf$(\phi_1, drop(\psi_1))$ holds over P_2; thus:

$$\text{Suffix}(P_2, P_1, do(drop(\psi_1), S_{P_1})) \land \text{ProgOf}(\phi_1, drop(\psi_1))(P_2). \tag{6.57}$$

Now, note that by (6.50), (6.55), (6.57), Axiom 6.2.6, and Definitions 6.2.11 and 6.2.7, to show the consequent, it is sufficient to prove that:

(a) if $N_1 = 0$ then Progressed$_{\text{CA}}(P_2, N_1, drop(\psi_1), S_1)$, and

(b) if $N_1 \neq 0$ then

$G_\cap(P_2, N_1 - 1, do(drop(\psi_1), S_1)) \land$ Progressed$(P_2, N_1, drop(\psi_1), S_1)$.

If these conditions hold, then this means that the **else** clause in Definition 6.2.7 is never selected and thus no new paths are added to the G relation at N_1 in $do(drop(\psi_1), S_1)$. Thus, in this case, by (6.49), which implies that ϕ_1 holds over all G-accessible paths at N_1 in S_1, and Definition 4.4.14, ProgOf$(\phi_1, drop(\psi_1))$ holds over all G-accessible paths at N_1 in $do(drop(\psi_1), S_1)$, and the consequent follows from this and Definition 4.2.1. I will now prove that (a) and (b) holds.

For (a), assume that $N_1 = 0$. Also, let's call the starting situation of P_2, S_{P_2}, where $S_{P_2} = do(drop(\psi_1), S_{P_1})$. Then by Definitions 6.2.8 and 4.3.4, I need to show that:

$$G(P_1, N_1, S_1) \land \text{Suffix}(P_2, P_1, S_{P_2}) \land K(S_{P_2}, do(drop(\psi_1), S_1)).$$

From (6.55) and (6.57), to prove this I just need to show that $K(S_{P_2}, do(drop(\psi_1), S_1))$. From (6.53), it follows that $K(S_{P_1}, S_1)$. From this, the SSA for K (i.e. Axiom 3.4.10), and the fact that the $drop(\psi_1)$ action is not a knowledge-producing action, it follows that $K(S_{P_2}, do(drop(\psi_1), S_1))$ holds if the $drop(\psi_1)$ action is possible in S_{P_1}. This follows from (6.57), Corollary 3.5.41, and Definition 3.3.1. Thus (6.58) and (a) hold:

$$K(S_{P_2}, do(drop(\psi_1), S_1)). \tag{6.58}$$

For (b), assume that $N_1 \neq 0$. Note that by (6.54), (6.57), and Definition 4.3.4, it follows that Progressed$(P_2, N_1, drop(\psi_1), S_1)$. I thus need to show that $G_{\cap}(P_2, N_1 - 1, do(drop(\psi_1), S_1))$. Note that by Lemma 6.3.9, this holds if $\forall n.\ n < N_1 \supset G_R(P_2, n, do(drop(\psi_1), S_1))$. By Corollary 6.3.5, I simply need to show that:

$$\forall n.\ n < N_1 \supset G(P_2, n, do(drop(\psi_1), S_1)).$$

I will prove this by strong induction on n. First assume that $n = 0$. Then by (6.50), Axiom 6.2.6, and Definitions 6.2.11 and 6.2.7, it follows that P_2 is G-accessible at level 0 in $do(drop(\psi_1), S_1)$ if Progressed$_{\text{CA}}(P_2, 0, drop(\psi_1), S_1)$ holds. This follows from (a) above. For the inductive case, assume that:

$$\forall n.\ n < M_1 \wedge M_1 < N_1 - 1 \supset G(P_2, n, do(drop(\psi_1), S_1)).$$

I need to show that $G(P_2, M_1 + 1, do(drop(\psi_1), S_1))$. Again, by (6.50), Axiom 6.2.6, and Definitions 6.2.11 and 6.2.7, it follows that P_2 is G-accessible at level $M_1 + 1$ in

$do(drop(\psi_1), S_1)$ if:

$$G_{\cap}(P_2, M_1, do(drop(\psi_1), S_1)) \wedge \text{Progressed}_{\text{CA}}(P_2, M_1 + 1, do(drop(\psi_1), S_1)).$$

From the inductive hypothesis, Corollary 6.3.5, and Lemma 6.3.9, it follows that $G_{\cap}(P_2, M_1, do(drop(\psi_1), S_1))$. Finally, from (6.54), (6.57), (6.58), and Definitions 6.2.8 and 4.3.4, it follows that $\text{Progressed}_{\text{CA}}(P_2, M_1{+}1, do(drop(\psi_1), S_1))$. Thus it follows that $\forall n.\ n < N_1 \supset G(P_2, n, do(drop(\psi_1), S_1))$ and hence (b) holds as well. $\square$

I presented a corresponding version of this property for the optimizing agent framework in Proposition 5.3.4. As shown there, unlike in the committed agent framework goals persist under more relaxed conditions after one of their subgoal is dropped. In particular, the situation s is not required to be executable, nor is the $drop(\psi)$ action required to be executable in s. Moreover, the agent is even allowed to have the c-goal that the $drop(\psi)$ action does not happen next. The reason behind these differences is that in the optimizing agent framework, the agent's set of G-accessible paths at some level is allowed to be empty (p-goals are really "desires" there). Also, an agent can have conflicting p-goals and thus having a p-goal that ψ in that framework does not necessarily imply that the agent also has ψ as a c-goal. In contrast, all of the p-goals of a committed agent are also her c-goals, and thus p-goal persistence requires much stronger conditions.

6.3.4 Goal Introspection

Just like an optimizing agent, a committed agent should be able to introspect her goals – if she has a realistic p-goal that ϕ, she should know that she has this as her realistic p-goal; moreover, if she does not have a realistic p-goal that ϕ, she should know this. It is easy to see that in this framework, the constraints on K and G that are required to yield positive and negative introspection are the same as in the optimizing agent framework. Thus, if the $KGTrans$ constraint is satisfied for some priority level n and some situation s, then the agents will have positive introspection of realistic p-goals at n in s:

Proposition 6.3.34.

$$\mathcal{D}^{SG}_{CAgt} \models \forall s, n.\ KGTrans(n, s) \supset (\text{RPGoal}(\phi, n, s) \supset \text{Know}(\text{RPGoal}(\phi, n), s)).$$

Proof. Same as that of Proposition 4.4.17. $\square$

Moreover, if the $KGEuc$ constraint is satisfied for some priority level n in s, then the agents will have negative introspection of realistic p-goals at n in s:

Proposition 6.3.35.

$$\mathcal{D}^{SG}_{CAgt} \models \forall s, n.\ KGEuc(n, s) \supset (\neg\text{RPGoal}(\phi, n, s) \supset \text{Know}(\neg\text{RPGoal}(\phi, n), s)).$$

Proof. Same as that of Proposition 4.4.19. $\square$

I will next show that as in the optimizing agent framework, these constraints also persist for executable situations in this framework if they hold for all initial situations as they are preserved by the (modified) successor-state axiom for G. However, in contrast to Chapter 4, I will prove just one (combined) property. This is necessary to deal with subgoals i.e. the case for $adoptRelTo$ actions.[48] Also, for this I use the following lemma, which says that for any situation s, the level n where a subgoal ψ is adopted w.r.t. a parent goal ϕ in s is unique:

Lemma 6.3.36.

$$\forall n_1, n_2, s.\ (\text{AdoptedLevel}(\phi, n_1, s) \wedge \text{AdoptedLevel}(\phi, n_2, s)) \supset n_1 = n_2.$$

Proof. Follows from Definition 5.2.4. □

I next show the persistence of $KGTrans$ and $KGEuc$:

Theorem 6.3.37.

$$\mathcal{D}^{SG}_{CAgt} \models (\forall n, s.\ \text{Init}(s) \supset KGTrans(n, s) \wedge KGEuc(n, s)) \supset$$
$$(\forall n, s.\ \text{Executable}(s) \supset KGTrans(n, s) \wedge KGEuc(n, s)).$$

[48]Recall that in Chapter 4, I did not prove persistence of $KGTrans$ and $KGEuc$ w.r.t. subgoal adoptions, since subgoals are introduced later in Chapter 5. However, these persistence results can also be shown to hold when subgoal adoptions are allowed by proving a theorem similar to Theorem 6.3.37 below.

Proof. (By induction on s) Assume that:

$$\forall n, s.\ \text{Init}(s) \supset KGTrans(n, s) \wedge KGEuc(n, s). \tag{6.59}$$

The base case, where s is an initial situation, is trivial. For the inductive case, we fix S_1 and A_1 and assume that:

$$\text{Executable}(do(A_1, S_1)). \tag{6.60}$$

Fix N_1. I need to show that $KGTrans(N_1, do(A_1, S_1))$ and $KGEuc(N_1, do(A_1, S_1))$. From (6.60) and Lemma 3.5.29, we have:

$$\text{Executable}(S_1),\ \text{and} \tag{6.61}$$

$$\text{Poss}(A_1, S_1). \tag{6.62}$$

(6.61) and the inductive hypothesis imply:

$$\forall n.\ KGTrans(n, S_1),\ \text{and} \tag{6.63}$$

$$\forall n.\ KGEuc(n, S_1). \tag{6.64}$$

I will next show that $KGTrans(N_1, do(A_1, S_1))$ and $KGEuc(N_1, do(A_1, S_1))$.

<u>Proof of $KGTrans(N_1, do(A_1, S_1))$</u>: Assume that $S_2 = do(A_1, S_1)$. Let us expand

$KGTrans(N_1, S_2)$; fix S_2^1, S_2^2, and P_2, and assume:

$$K(S_2^1, S_2), \tag{6.65}$$

$$K(S_2^2, S_2^1), \tag{6.66}$$

$$G(P_2, N_1, S_2^1), \tag{6.67}$$

$$\text{Starts}(P_2, S_2^2). \tag{6.68}$$

I need to show that $G(P_2, N_1, S_2)$. (6.65), (6.66), and Axiom 3.4.10 imply that there exist S_1^1 and S_1^2 such that:

$$K(S_1^1, S_1) \wedge S_2^1 = do(A_1, S_1^1), \text{ and} \tag{6.69}$$

$$K(S_1^2, S_1^1) \wedge S_2^2 = do(A_1, S_1^2). \tag{6.70}$$

Now, note that by (6.65), (6.66), and the transitivity of K (i.e. Axiom 3.4.3), we have:

$$K(S_2^2, S_2). \tag{6.71}$$

Note that, since K is an equivalence relation (as it is reflexive, i.e. Axiom 3.4.2, and Euclidean, i.e. Axiom 3.4.4), by (6.69), (6.63), and (6.64), it follows that S_1 and S_1^1 have the same set of G-accessible paths at all levels:

$$\forall p, n.\ G(p, n, S_1) \equiv G(p, n, S_1^1). \tag{6.72}$$

Moreover, by the fact that K is an equivalence relation, (6.69), (6.72), Definition 4.2.4, and Axiom 4.2.7, it follows that:

$$\forall p, n.\ G_{\cap}(p, n, S_1) \equiv G_{\cap}(p, n, S_1^1). \tag{6.73}$$

Finally, by (6.69), (6.63), (6.64), Propositions 6.3.34 and 6.3.35, and Definition 5.2.4, it follows that $\forall n.$ AdoptedLevel$(\phi, n, S_1) \supset$ AdoptedLevel(ϕ, n, S_1^1), and from this and Lemma 6.3.36, it follows that:

$$\forall n.\ \text{AdoptedLevel}(\phi, n, S_1) \equiv \text{AdoptedLevel}(\phi, n, S_1^1). \tag{6.74}$$

Now, to show that P_2 is G-accessible at level N_1 in situation S_2, we will need to analyze the SSA for G. A close look at it gives us five cases with five different mutually exclusive conditions that apply in S_1^1: Case 1, where one simply progresses the old set of G-accessible paths before the action A_1 has happened (i.e. in S_1^1) and ensures that these paths start with K-accessible situations in S_2^1 to obtain the new set of G-accessible paths after the occurrence of A_1, Case 2, which involves progression and K-accessibility check with shifting levels, Case 3, which involves processing/filtering the old set of G-accessible paths to handle the adoption of a goal at level N_1, Case 4, which involves processing/filtering the old set of G-accessible paths to handle the adoption of a subgoal w.r.t. a supergoal at level N_1, and Case 5 that involves processing them to handle the dropping of a goal at level N_1 as well as to handle regular actions and drop actions that make p-goals at level N_1 incompatible with other higher priority p-goals and/or with the agent's knowledge (see below for details). Let us discuss each case, one at a time. Thus the successor-state axiom for G (i.e. Axiom 6.2.6) and (6.67) give us five cases:

- **Case 1.** The action A_1 is such that:

  $$\forall p.\ G(p, N_1, do(A_1, S_1^1)) \equiv \text{Progressed}_{\text{CA}}(p, N_1, A_1, S_1^1).$$

  By Axiom 6.2.6 and Definitions 6.2.7, 6.2.9, 6.2.10, and 6.2.11, this is the case if (a1) the action A_1 is a regular (non-adopt/drop action) whose occurrence does not make the p-goals at level N_1 impossible or inconsistent with higher priority goals and with the agent's knowledge (i.e. there is a path P^* starting with situation S_{P^*} that is in $G_\cap$ up to level N_1 in S_1^1 over which A_1 happens next, and $do(A_1, S_{P^*})$ remains K-accessible from $do(A_1, S_1^1)$), or (a2) A_1 is an $adopt$ action, but it does not refer to the adoption of a goal ϕ at level N_1 or at some higher priority level than N_1 (and thus $\neg(A_1 = adopt(\phi, M) \wedge M \leq N_1)$), or (a3) A_1 is an $adoptRelTo$ action, but it does not refer to the adoption of a subgoal ψ w.r.t. a supergoal ϕ at level N_1 or at some higher priority level than N_1 (and thus $\neg(A_1 = adoptRelTo(\psi, \phi) \wedge \exists m.\ \text{AdoptedLevel}(\phi, m, S_1^1) \wedge m \leq N_1)$), or (a4) A_1 is a $drop(\phi)$ action, but it does not refer to the dropping of a goal ϕ at N_1 (i.e. $\neg\text{PGoal}(\phi, N_1, S_1^1)$) and the occurrence of A_1 does not make the p-goals at level

N_1 impossible or inconsistent with higher priority goals and with the agent's knowledge (and thus there is a path P^* starting with situation S_{P^*} that is in $G_\cap$ up to level N_1 in S_1^1 over which $A_1 = drop(\phi)$ happens next, and $do(A_1, S_{P^*})$ remains K-accessible from $do(A_1, S_1^1)$). By the SSA for G (i.e. Axiom 6.2.6 and Definitions 6.2.7, 6.2.9, 6.2.10, and 6.2.11) and (6.67), in all these cases P_2 is the simple progression of some path P_1 that was G-accessible at N_1 in S_1^1, with the additional condition that P_2 must start with a K-accessible situation in S_2^1, i.e. $\text{Progressed}_{\text{CA}}(P_2, N_1, A_1, S_1^1)$. Thus, we have:

$$\text{Starts}(P_1, S_1^2) \wedge \text{Suffix}(P_2, P_1, do(A_1, S_1^2)) \wedge G(P_1, N_1, S_1^1). \tag{6.75}$$

Definition 4.4.16, (6.63), (6.69), (6.70), and (6.75) imply that:

$$G(P_1, N_1, S_1). \tag{6.76}$$

Note that, by (6.72) and the fact that the agent knows what the action A_1 is (i.e. A_1 refers to the same action in all K-accessible situations, let's call this fact (f1)) it can be shown that all the conditions (a1) to (a4) that hold for S_1^1 also hold for S_1. In particular, (a1) follows from the facts that A_1 refers to the same action in S_1 and S_1^1 (i.e. by (f1)), that $S_2 = do(A_1, S_1)$ and $S_2^1 = do(A_1, S_1^1)$ are in the same K equivalence class (by (6.65) and (6.69)), and (6.73). (a2) follows from (f1). (a3) follows from (f1) and (6.74). Finally, (a4) follows from (f1), (6.72), (6.65), (6.69), and (6.73). By this, Axiom 6.2.6, Definitions 6.2.7, 6.2.8, 6.2.9, 6.2.10, and 6.2.11, the assumption regarding A_1 for this case, (6.75), and (6.76), the progression of P_1 (i.e. P_2) will be retained in the G-relation at N_1 in $S_2 = do(A_1, S_1)$; this is because by (6.75), (6.76), and Definition 4.3.4, Progressed(P_2, N_1, A_1, S_1) holds, and by (6.68) and (6.71), P_2 starts with a K-accessible situation in S_2. Thus by these and Definition 6.2.8, Progressed$_{\text{CA}}(P_2, N_1, A_1, S_1)$, and hence we have $G(P_2, N_1, S_2)$.

- **Case 2.** A_1 refers to the adoption of a goal ϕ at a higher priority level than N_1, i.e. $\exists m.\ A_1 = adopt(\phi, m) \wedge m < N_1$ or to the adoption of a subgoal ψ with respect to a parent goal ϕ at a higher priority level than N_1, i.e. $A_1 = adoptRelTo(\psi, \phi) \wedge \exists m.$ AdoptedLevel$(\phi, m, S_1^1) \wedge m < N_1$. In this case, by Definitions 6.2.9 and 6.2.10, P_2 is the progression of some path P_1 that was G-accessible at $N_1 - 1$ in S_1^1, provided that P_2 starts with a K-accessible situation in S_2^1, i.e. Progressed$_{\text{CA}}(P_2, N_1 - 1, A_1, S_1^1)$ (since adopting the (sub)goal at higher priority than N_1 has pushed all the goals that has priority lower than $m - 1$ down by one level):

$$\text{Starts}(P_1, S_1^2) \wedge \text{Suffix}(P_2, P_1, do(A_1, S_1^2)) \wedge G(P_1, N_1 - 1, S_1^1).$$

The rest of the proof for this case is similar to that of Case 1.

- **Case 3.** A_1 refers to the adoption of a goal ϕ at N_1, i.e. $A_1 = \mathit{adopt}(\phi, N_1)$. Since A_1 refers to the same action in S_1 and S_1^1, then by Definition 6.2.9, P_2 is included in the G-relation at N_1 in S_2 if it starts with a situation that is K-accessible in S_2, and if $\phi(P_2)$ holds. (6.68) and (6.71) imply this former. The latter also holds, otherwise by the SSA for G, P_2 would have not been included in the G-relation at N_1 in S_2^1 (but it is included by (6.67)).

- **Case 4.** A_1 refers to the adoption of a subgoal ψ w.r.t. a parent goal ϕ at N_1, i.e. $A_1 = \mathit{adoptRelTo}(\psi, \phi) \wedge \text{AdoptedLevel}(\phi, N_1, S_1^1)$. Note that, as in Case 1, it can be shown that all the assumptions for this case for S_1^1 also hold for S_1; in particular, we have $\text{AdoptedLevel}(\phi, N_1, S_1)$. Then by Definition 6.2.10, P_2 is included in the G-relation at N_1 in S_2 if (a) P_2 is the progression of some path P_1 that was G-accessible at $N_1 - 1$ in S_1, (b) P_2 starts with a K-accessible situation in S_2, and (c) if $\psi(P_2)$ holds. Now, since by (6.67), $G(P_2, N_1, S_2^1)$ holds, from the SSA for G and the assumption for this case (including that for AdoptedLevel) it follows that $G(P_1, N_1 - 1, S_1^1)$. From this, (6.69), (6.70), (6.63), and Definition 4.4.16, it follows that $G(P_1, N_1 - 1, S_1)$. Thus (a) holds. Moreover, (b) follows from (6.68) and (6.71). Finally, (c), i.e. that $\psi(P_2)$, also holds, otherwise by the assumptions for this case regarding A_1 (and AdoptedLevel) and the SSA for G, P_2 would have not been included in the G-relation at N_1 in S_2^1 (but it is included by (6.67)).

- **Case 5.** The action A_1 is such that:

$$\forall p.\ G(p, N_1, do(A_1, S_1^1)) \equiv \exists s'.\ \text{Starts}(p, s') \wedge K(s', do(A_1, S_1^1)).$$

By Axiom 6.2.6 and Definitions 6.2.7 and 6.2.11, this is the case if the action A_1 is a regular action whose occurrence makes the p-goals at level N_1 impossible or inconsistent with higher priority goals and with the agent's knowledge (i.e. there are no paths P^* s.t. P^* starts with situation S_{P^*}, P^* is in $G_\cap$ up to level N_1 in S_1^1, A_1 happens over P^* next, and $do(A_1, S_{P^*})$ remains K-accessible from $do(A_1, S_1^1)$), or A_1 refers to the dropping of a goal ϕ, i.e. $A_1 = drop(\phi)$, where $\neg\text{PGoal}(\phi, N_1, S_1)$, but the occurrence of A_1 makes the p-goals at level N_1 impossible or inconsistent with higher priority goals and with the agent's knowledge (i.e. there are no paths P^* s.t. P^* starts with situation S_{P^*}, P^* is in $G_\cap$ up to level N_1 in S_1^1, $A_1 = drop(\phi)$ happens over P^* next, and $do(A_1, S_{P^*})$ remains K-accessible from $do(A_1, S_1^1)$), or A_1 refers to the dropping of a goal ϕ at N_1, i.e. $A_1 = drop(\phi)$, where $\text{PGoal}(\phi, N_1, S_1^1)$. As in Case 1, it can be shown that by the inductive hypothesis, all these conditions also hold for S_1. Thus, by Definitions 6.2.7 and 6.2.11, P_2 is included in the G-relation at N_1 in S_2 if it starts with a situation that is K-accessible in S_2. Again, (6.68) and (6.71) imply this condition.

It thus follows that $KGTrans(N_1, do(A_1, S_1))$.

Proof of $KGEuc(N_1, do(A_1, S_1))$: Let us expand $KGEuc(N_1, S_2)$; fix S_2^1, S_2^2, and P_2, and assume:

$$K(S_2^1, S_2), \tag{6.77}$$

$$K(S_2^2, S_2), \tag{6.78}$$

$$G(P_2, N_1, S_2), \tag{6.79}$$

$$\text{Starts}(P_2, S_2^2). \tag{6.80}$$

I need to show that $G(P_2, N_1, S_2^1)$. (6.77), (6.78), and Axiom 3.4.10 imply that there exist S_1^1 and S_1^2 such that:

$$K(S_1^1, S_1) \wedge S_2^1 = do(A_1, S_1^1), \text{ and} \tag{6.81}$$

$$K(S_1^2, S_1) \wedge S_2^2 = do(A_1, S_1^2). \tag{6.82}$$

Now, note that by (6.77), (6.78), and the Euclideanism of K (i.e. Axiom 3.4.4), we have:

$$K(S_2^2, S_2^1). \tag{6.83}$$

Note that, since K is an equivalence relation (as it is reflexive, i.e. Axiom 3.4.2, and Euclidean, i.e. Axiom 3.4.4), by (6.81), (6.63), and (6.64), it follows that S_1 and S_1^1

have the same set of G-accessible paths at all levels:

$$\forall p, n.\ G(p, n, S_1) \equiv G(p, n, S_1^1). \tag{6.84}$$

Moreover, by the fact that K is an equivalence relation, (6.81), (6.84), Definition 4.2.4, and Axiom 4.2.7, it follows that:

$$\forall p, n.\ G_\cap(p, n, S_1) \equiv G_\cap(p, n, S_1^1). \tag{6.85}$$

Finally, by (6.81), (6.63), (6.64), Propositions 6.3.34 and 6.3.35, and Definition 5.2.4, it follows that $\forall n.$ AdoptedLevel$(\phi, n, S_1) \supset$ AdoptedLevel(ϕ, n, S_1^1), and from this and Lemma 6.3.36, it follows that:

$$\forall n.\ \text{AdoptedLevel}(\phi, n, S_1) \equiv \text{AdoptedLevel}(\phi, n, S_1^1). \tag{6.86}$$

Now, to show that P_2 is G-accessible at level N_1 in situation S_2^1, we will need to analyze the SSA for G. A close look at it gives us five cases with five different mutually exclusive conditions that apply in S_1: Case 1, where one simply progresses the old set of G-accessible paths before the action A_1 has happened and ensures that these paths start with K-accessible situations in S_2 to obtain the new set of G-accessible paths after the occurrence of A_1, Case 2, which involves progression and K-accessibility check with shifting levels, Case 3, which involves processing/filtering the old set of G-accessible paths to handle the adoption of a goal at level N_1, Case 4, which involves processing/filtering the old set of G-accessible paths to handle the adoption of a subgoal w.r.t. a supergoal at level N_1, and Case 5 that involves processing them to handle the dropping of a goal at level N_1 as well as to handle regular actions and drop actions that make p-goals at level N_1 incompatible with other higher priority p-goals and/or with the agent's knowledge (see below for details). Let us discuss each case, one at a time. Thus the successor-state axiom for G (i.e. Axiom 6.2.6) and (6.79) give us five cases:

- **Case 1.** The action A_1 is such that:

  $$\forall p.\ G(p, N_1, do(A_1, S_1)) \equiv \text{Progressed}_{\text{CA}}(p, N_1, A_1, S_1).$$

  By Axiom 6.2.6 and Definitions 6.2.7, 6.2.9, 6.2.10, and 6.2.11, this is the case if (a) the action A_1 is a regular (non-adopt/drop action) whose occurrence does not make the p-goals at level N_1 impossible or inconsistent with higher priority goals and with the agent's knowledge (i.e. there is a path P^* starting with situation S_{P^*} that is in $G_\cap$ up to level N_1 in S^1 over which A_1 happens next, and $do(A_1, S_{P^*})$ remains K-accessible from $do(A_1, S^1)$), or (b) A_1 is an $adopt$ action, but it does not refer to the adoption of a goal ϕ at level N_1 or at some higher priority level than N_1 (and thus $\neg(A_1 = adopt(\phi, M) \wedge M \leq N_1)$), or (c) A_1 is an $adoptRelTo$ action, but it does not refer to the adoption of a subgoal ψ

w.r.t. a supergoal ϕ at level N_1 or at some higher priority level than N_1 (and thus $\neg(A_1 = adoptRelTo(\psi, \phi) \wedge \exists m.$ AdoptedLevel$(\phi, m, S_1) \wedge m \leq N_1))$, or (d) A_1 is a $drop(\phi)$ action, but it does not refer to the dropping of a goal ϕ at N_1 (i.e. $\neg$PGoal$(\phi, N_1, S_1))$ and the occurrence of A_1 does not make the p-goals at level N_1 impossible or inconsistent with higher priority goals and with the agent's knowledge (i.e. there is a path P^* starting with situation S_{P^*} that is in $G_\cap$ up to level N_1 in S^1 over which $A_1 = drop(\phi)$ happens next, and $do(A_1, S_{P^*})$ remains K-accessible from $do(A_1, S^1))$. By the SSA for G and (6.79), in all these cases P_2 is the simple progression of some path P_1 that was G-accessible at N_1 in S_1, with the additional condition that P_2 must start with a K-accessible situation in S_2, i.e. i.e. Progressed$_{\text{CA}}(P_2, N_1, A_1, S_1)$. Thus, we have:

$$\text{Starts}(P_1, S_1^2) \wedge \text{Suffix}(P_2, P_1, do(A_1, S_1^2)) \wedge G(P_1, N_1, S_1). \qquad (6.87)$$

Definition 4.4.18, (6.64), (6.81), (6.82), and (6.87) imply that:

$$G(P_1, N_1, S_1^1). \qquad (6.88)$$

Note that, by (6.84) and the fact that the agent knows what the action A_1 is (i.e. A_1 refers to the same action in all K-accessible situations, let's call this fact (f1)) it can be shown that all the conditions (a1) to (a4) that hold for S_1^1 also hold for S_1. In particular, (a1) follows from the facts that A_1 refers to the same action in S_1 and S_1^1 (i.e. by (f1)), that $S_2 = do(A_1, S_1)$ and $S_2^1 = do(A_1, S_1^1)$ are in the same K equivalence class (by (6.77) and (6.81)), and (6.85). (a2) follows from (f1). (a3) follows from (f1) and (6.86). Finally, (a4) follows from (f1), (6.84), (6.77), (6.81), and (6.85). By this, Axiom 6.2.6, Definitions 6.2.7, 6.2.8, 6.2.9, 6.2.10, and 6.2.11, the assumption regarding A_1 for this case, (6.87), and (6.88), the progression of P_1 (i.e. P_2) will be retained in the G-relation at N_1 in $S_2^1 = do(A_1, S_1^1)$; this is because by (6.87), (6.88), and Definition 4.3.4, Progressed(P_2, N_1, A_1, S_1^1) holds, and by (6.80) and (6.83), P_2 starts with a K-accessible situation in S_2^1. Thus by these and Definition 6.2.8, Progressed$_{\text{CA}}(P_2, N_1, A_1, S_1^1)$, and hence we have $G(P_2, N_1, S_2^1)$.

- **Case 2.** A_1 refers to the adoption of a goal ϕ at a higher priority level than N_1, i.e. $\exists m.\ A_1 = adopt(\phi, m) \wedge m < N_1$ or to the adoption of a subgoal ψ with respect to a parent goal ϕ at a higher priority level than N_1, i.e. $A_1 = adoptRelTo(\psi, \phi) \wedge \exists m.$ AdoptedLevel$(\phi, m, S_1) \wedge m < N_1$. In this case, by Definitions 6.2.9 and 6.2.10, P_2 is the progression of some path P_1 that was G-accessible at $N_1 - 1$ in S_1, provided that P_2 starts with a K-accessible situation in S_2, i.e. Progressed$_{\text{CA}}(P_2, N_1 - 1, A_1, S_1)$ (since adopting the (sub)goal at higher priority than N_1 has pushed all the goals that has priority lower than $m - 1$ down by one level):

$$\text{Starts}(P_1, S_1^2) \wedge \text{Suffix}(P_2, P_1, do(A_1, S_1^2)) \wedge G(P_1, N_1 - 1, S_1).$$

The rest of the proof for this case is similar to that of Case 1.

- **Case 3.** A_1 refers to the adoption of a goal ϕ at N_1, i.e. $A_1 = \mathit{adopt}(\phi, N_1)$. Since A_1 refers to the same action in S_1 and S_1^1, then by Definition 6.2.9, P_2 is included in the G-relation at N_1 in S_2^1 if it starts with a situation that is K-accessible in S_2^1, and if $\phi(P_2)$ holds. (6.80) and (6.83) imply this former. The latter also holds, otherwise by the SSA for G, P_2 would have not been included in the G-relation at N_1 in S_2 (but it is included by (6.79)).

- **Case 4.** A_1 refers to the adoption of a subgoal ψ w.r.t. a parent goal ϕ at N_1, i.e. $A_1 = \mathit{adoptRelTo}(\psi, \phi) \wedge \text{AdoptedLevel}(\phi, N_1, S_1)$. Note that, as in Case 1, it can be shown that all the assumptions for this case for S_1 also hold for S_1^1; in particular, we have $\text{AdoptedLevel}(\phi, N_1, S_1^1)$. Then by Definition 6.2.10, P_2 is included in the G-relation at N_1 in S_2^1 if (a) P_2 is the progression of some path P_1 that was G-accessible at $N_1 - 1$ in S_1^1, (b) P_2 starts with a K-accessible situation in S_2^1, and (c) if $\psi(P_2)$ holds. Now, since by (6.79), $G(P_2, N_1, S_2)$ holds, from the SSA for G and the assumption for this case (including that for AdoptedLevel) it follows that $G(P_1, N_1 - 1, S_1)$. From this, (6.81), (6.82), (6.64), and Definition

4.4.18, it follows that $G(P_1, N_1 - 1, S_1^1)$. Thus (a) holds. Moreover, (b) follows from (6.80) and (6.83). Finally, (c), i.e. that $\psi(P_2)$, also holds, otherwise by the assumptions for this case regarding A_1 (and AdoptedLevel) and the SSA for G, P_2 would have not been included in the G-relation at N_1 in S_2 (but it is included by (6.79)).

- **Case 5.** A_1 is a regular action whose occurrence makes the p-goals at level N_1 impossible or inconsistent with higher priority goals and with the agent's knowledge (i.e. there are no paths P^* s.t. P^* starts with situation S_{P^*}, P^* is in $G_\cap$ up to level N_1 in S^1, A_1 happens over P^* next, and $do(A_1, S_{P^*})$ remains K-accessible from $do(A_1, S^1)$), or A_1 refers to the dropping of a goal ϕ, i.e. $A_1 = drop(\phi)$, where $\neg$PGoal(ϕ, N_1, S_1), but the occurrence of A_1 makes the p-goals at level N_1 impossible or inconsistent with higher priority goals and with the agent's knowledge (i.e. there are no paths P^* s.t. P^* starts with situation S_{P^*}, P^* is in $G_\cap$ up to level N_1 in S^1, $A_1 = drop(\phi)$ happens over P^* next, and $do(A_1, S_{P^*})$ remains K-accessible from $do(A_1, S^1)$), or A_1 refers to the dropping of a goal ϕ at N_1, i.e. $A_1 = drop(\phi)$, where PGoal(ϕ, N_1, S_1). As in Case 1, it can be shown by the inductive hypothesis that all these conditions also hold for S_1^1. Thus, by Definitions 6.2.7 and 6.2.11, P_2 is included in the G-relation at N_1 in S_2^1 if it starts with a situation that is K-accessible in S_2^1. Again, (6.80) and (6.83) imply this condition.

It thus follows that $KGEuc(N_1, do(A_1, S_1))$. The theorem thus follows. $\square$

6.3.5 Goal Persistence

I next discuss persistence of these motivational attitudes. As in the optimizing agent case, I focus on persistence of achievement goals only. I will need the following definition for this:

Definition 6.3.38.

$$\begin{aligned}\text{KnowProdAct}(agt, a) \doteq\ & \text{BinarySensingAction}(a) \land \text{agent}(a) = agt \\ & \lor \text{NonBinarySensingAction}(a) \land \text{agent}(a) = agt \\ & \lor \exists inf.\ a = informWhether(inf, agt, \Psi) \\ & \lor \exists inf.\ a = informRef(inf, agt, \theta).\end{aligned}$$

That is, an action a is a knowledge-producing action for some agent agt if it is a binary or non-binary sensing action whose agent is agt, or if a involves some informer inf informing agt whether some formula Ψ holds or of the value of some term θ.

Given this, first I can show that if an agent has a (realistic) p-goal that $\Diamond\Phi$ in some executable situation s, then she will retain this p-goal after some action a has been performed in s, provided that:[49]

- she knows in s that Φ has not yet been achieved,
- that a is not the action of dropping a p-goal,
- that a is not a knowledge-producing action for her,[50]
- and that the agent does not have the c-goal not to execute a next in s.

Proposition 6.3.39 (Persistence of PGoals)**.**

$$\begin{aligned}\mathcal{D}^{SG}_{CAgt} \models\ & \text{PGoal}(\Diamond\Phi, n, s) \wedge \text{Executable}(s) \wedge \text{Know}(\neg\Phi, s)\\ & \wedge \neg a = drop(\psi) \wedge \neg\text{KnowProdAct}(a) \wedge \neg\text{CGoal}(\neg\exists s'.\ \text{Do}(a, now, s'), s)\\ & \supset \exists m.\ \text{PGoal}(\Diamond\Phi, m, do(a, s)).\end{aligned}$$

[49]In the following, I use CGoal$(\exists s'.\ \text{Do}(a, now, s'), s)$ as an abbreviation for CGoal(Starts$(now) \wedge \exists s'.\ \text{OnPath}(s') \wedge \text{Do}(a, now, s'), s)$; here now refers to the starting situation of the (suppressed) CGoal-accessible path.

[50]As mentioned in Chapter 4, I suppress the agent argument agt in knowledge and goals. Following this, I also suppress the agent argument in KnowProdAct(a) below.

Proof. Fix Φ_1, N_1, S_1, and A_1. By the antecedent, we have:

$$\text{PGoal}(\Diamond\Phi_1, N_1, S_1), \tag{6.89}$$

$$\text{Executable}(S_1), \tag{6.90}$$

$$\text{Know}(\neg\Phi_1, S_1), \tag{6.91}$$

$$\neg A_1 = drop(\psi), \tag{6.92}$$

$$\neg\text{KnowProdAct}(a), \text{ and} \tag{6.93}$$

$$\neg\text{CGoal}(\neg\exists s'.\ \text{Do}(a, now, s'), S_1). \tag{6.94}$$

Now, by (6.89) and Definitions 4.2.1, 3.5.8, and 3.5.7, since the agent has the p-goal that $\Diamond\Phi_1$ at N_1 in S_1, the following holds:

$$\forall p.\ G(p, N_1, S_1) \supset \exists s^*.\ \text{OnPath}(p, s^*) \wedge \Phi_1(s^*). \tag{6.95}$$

Thus the p-goal that $\Diamond\Phi_1$ will persist after A_1 has been performed in S_1 if there is a level n such that $\Diamond\Phi_1$ holds over all G-accessible paths at n in $do(A_1, S_1)$. After the A_1 action has been performed in S_1, the G relation at every level will be updated in accordance with the SSA for G. By Axiom 6.2.6, there are four cases to consider: (1) A_1 is a regular action, (2) A_1 is the adoption of some goal, (3) A_1 is the adoption of a subgoal w.r.t. some parent goal, or (4) A_1 is the dropping of some goal. The last case, i.e. an explicit dropping of a goal is ruled out by (6.92). So let us consider the other three cases, one at a time:

1. $\underline{A_1 \text{ is a regular action (i.e. neither } adopt\text{, nor } adoptRelTo\text{, nor } drop\text{):}}$

   First, assume that A_1 is a regular action. Then, by Axiom 6.2.6 and Definition 6.2.7, the set of G-accessible paths at N_1 in S_1 will be progressed and filtered to reflect the fact that A_1 has just happened. Note that, from (6.94) and Definition 4.2.10, it follows that there is a path P_1 starting with some situation S_{P_1} that is in the $G_\cap$ relation in S_1 and over which the next action performed is A_1. Let's call the suffix of P_1 starting in $do(A_1, S_{P_1})$, P_2. Thus, we have:

   $$G_\cap(P_1, S_1) \land \text{Starts}(P_1, S_{P_1}) \land \text{Suffix}(P_2, P_1, do(A_1, S_{P_1})). \tag{6.96}$$

   By (6.90), (6.89), Proposition 6.3.15, and Definition 4.2.10, it follows that:

   $$\Diamond\Phi_1(P_1). \tag{6.97}$$

   From (6.96) and Lemma 6.3.8, it follows that:

   $$K(S_{P_1}, S_1). \tag{6.98}$$

   Since P_1 is a path, by (6.96), Definition 3.5.16, Corollary 3.5.41 and Lemma 3.5.29 it follows that:

   $$\text{Poss}(A_1, S_{P_1}). \tag{6.99}$$

   Thus, by (6.98), (6.99), Axiom 3.4.10, and the fact that A_1 is not a knowledge-producing action for the agent (i.e. (6.93)), it follows that:

   $$K(do(A_1, S_{P_1}), do(A_1, S_1)). \tag{6.100}$$

From (6.96), (6.90), and Proposition 6.3.14(b), it follows that:

$$\forall n.\ G(P_1, n, S_1). \tag{6.101}$$

I will now show that the following condition holds (let's call it (a)):

$$\forall p.\ G(p, N_1, do(A_1, S_1)) \equiv \text{Progressed}_{\text{CA}}(p, N_1, A_1, S_1). \tag{a}$$

That is, all G-accessible paths at N_1 in $do(A_1, S_1)$ are suffixes of those that are G-accessible at N_1 in S_1. By the fact that A_1 is a regular action, Axiom 6.2.6, and Definition 6.2.7, to show that (a) holds, I need to show that there exist a path that is the progression by A_1 of a G-accessible path in S_1 that is consistent with higher priority levels than N_1 and knowledge. I will show that P_2 satisfies this condition. Put otherwise, by Axiom 6.2.6 and Definition 6.2.7, I need to show that:

- $\text{Progressed}_{\text{CA}}(P_2, N_1, A_1, S_1)$, if $N_1 = 0$, and
- $G_\cap(P_2, N_1 - 1, do(A_1, S_1)) \land \text{Progressed}(P_2, N_1, A_1, S_1)$, otherwise.

If these hold, then no new paths are added to the set of G-accessible paths at N_1 in $do(A_1, S_1)$, i.e. the **else** clause in Definition 6.2.7 is never selected. First, assume that $N_1 = 0$. Then from (6.101), (6.100), (6.96), and Definitions 6.2.8 and 4.3.4, it follows that $\text{Progressed}_{\text{CA}}(P_2, 0, A_1, S_1)$. Also, from this, Axiom

6.2.6, Definition 6.2.7, and the fact that A_1 is a regular action, we have:

$$G(P_2, 0, do(A_1, S_1)). \qquad (6.102)$$

Next, consider the case where $N_1 \neq 0$. Note that from (6.101), (6.96), and Definition 4.3.4, it follows that:

$$\forall n. \text{ Progressed}(P_2, n, A_1, S_1). \qquad (6.103)$$

Thus I just need to show that $\forall n.\ n \geq 0 \supset G_\cap(P_2, n, do(A_1, S_1))$. I will show this by induction on n. The base case, where $n = 0$, follows from (6.102), Corollary 6.3.5, and Axiom 4.2.7. For the inductive step, fix level K and assume that $G_\cap(P_2, K, do(A_1, S_1))$. From this, (6.103), Axiom 6.2.6, Definition 6.2.7, and the fact that A_1 is a regular action, it follows that $G(P_2, K+1, do(A_1, S_1))$. From this and Corollary 6.3.5, it follows that $G_R(P_2, K+1, do(A_1, S_1))$. Finally, from this, the inductive hypothesis, and Axiom 4.2.7, it follows that $G_\cap(P_2, K+1, do(A_1, S_1))$. Thus condition (a) follows.

Note that despite (a) above, $\Diamond\Phi_1$ can still fail to be a p-goal at N_1 in $do(A_1, S_1)$ if there is a G-accessible path at N_1 in $do(A_1, S_1)$, but $\Diamond\Phi_1$ does not hold over this path. Thus I will next show that this is not the case. Given (a), it follows from Axiom 6.2.6 and Definition 6.2.7 that the G-accessible paths at level N_1 in situation $do(A_1, S_1)$ can only be obtained by progressing those at N_1 in S_1

and checking that these paths start with a K-accessible situation in $do(A_1, S_1)$. Thus $\Diamond\Phi_1$ does not hold over a G-accessible path (and by Corollary 6.3.5, a G_R-accessible path) at N_1 in $do(A_1, S_1)$ if there is a K-accessible situation in S_1, say S_1', where there is a path P_1' that starts with S_1', P_1' is G-accessible at N_1 in S_1, the suffix of P_1' that starts with $do(A_1, S_1')$, let's call it P_2', is G_R-accessible at N_1 in $do(A_1, S_1)$, and $\Diamond\Phi_1$ does not hold over P_2':

$$K(S_1', S_1) \wedge \text{Starts}(P_1', S_1') \wedge G(P_1', N_1, S_1) \wedge \text{Suffix}(P_2', P_1', do(A_1, S_1'))$$
$$\wedge\, G_R(P_2', N_1, do(A_1, S_1)) \wedge \neg\Diamond\Phi_1(P_2'). \tag{6.104}$$

By (6.95) and (6.104), it follows that $\exists s.\ \text{OnPath}(P_1', s) \wedge \Phi_1(s)$. By this, (6.104), and Definitions 3.5.8 and 3.5.7, it follows that $\Phi_1(S_1')$. But by (6.104), (6.91), and Definition 3.4.5, we have $\neg\Phi_1(S_1')$, a contradiction! Thus it follows that $\Diamond\Phi_1$ holds over all G_R-accessible paths, and by Corollary 6.3.5, over all G-accessible paths at N_1 in $do(A_1, S_1)$.

2. <u>A_1 is the adoption of some goal:</u>

   Fix Ψ_1 and N_2 and assume that $A_1 = adopt(\Psi_1, N_2)$. Again, we have two cases, one where the agent adopts the goal Ψ_1 at a lower priority level than N_1, i.e. $N_2 > N_1$, and another where she adopts the goal at level N_1 or at a higher priority level than N_1, i.e. $N_2 \leq N_1$. Let us consider each case, one at a time:

(a) Assume that $N_2 > N_1$. Now, from Axiom 6.2.6 and Definitions 6.2.9, we can see that after the $adopt(\Psi_1, N_2)$ action happens, the G-accessible paths at level N_1 in situation $do(A_1, S_1)$ are those that can be obtained by progressing those at N_1 in S_1 and checking that these paths start with a K-accessible situation in $do(A_1, S_1)$. If there are no such paths, then the agent's G-accessible paths at N_1 in $do(A_1, S_1)$ will be empty, and by Definition 4.2.1, the agent will trivially have the p-goal that $\Diamond\Phi_1$ at N_1 in $do(A_1, S_1)$.

So, assume that there is indeed one such path, but $\Diamond\Phi_1$ does not hold over this path. I will prove by contradiction that this is impossible. Thus $\Diamond\Phi_1$ does not hold over a G-accessible path (and by Corollary 6.3.5, a G_R-accessible path) at N_1 in $do(A_1, S_1)$ if there is a K-accessible situation in S_1, say S_1'', where there is a path P_1'' that starts with S_1'', P_1'' is G-accessible at N_1 in S_1, the suffix of P_1'' that starts with $do(A_1, S_1'')$, let's call it P_2'', is G_R-accessible at N_1 in $do(A_1, S_1)$, and $\Diamond\Phi_1$ does not hold over P_2'':

$$K(S_1'', S_1) \wedge \text{Starts}(P_1'', S_1'') \wedge G(P_1'', N_1, S_1) \wedge \text{Suffix}(P_2'', P_1'', do(A_1, S_1''))$$
$$\wedge\, G_R(P_2'', N_1, do(A_1, S_1)) \wedge \neg\Diamond\Phi_1(P_2''). \tag{6.105}$$

By (6.95) and (6.105), it follows that $\exists s.\ \text{OnPath}(P_1'', s) \wedge \Phi_1(s)$. By this, (6.105), and Definitions 3.5.8 and 3.5.7, it follows that $\Phi_1(S_1'')$. But by (6.105), (6.91), and Definition 3.4.5, we have $\neg\Phi_1(S_1'')$, a contradiction! Thus it follows that $\Diamond\Phi_1$ holds over all G_R-accessible paths, and by Corollary 6.3.5, over all G-accessible paths at N_1 in $do(A_1, S_1)$.

(b) Assume that $N_2 \leq N_1$. Similar to case (a) above, it can be shown that the p-goal that $\Diamond\Phi_1$ persists in this case as well, however at level $N_1 + 1$ as the p-goals at N_1 in S_1 are pushed down one level in the hierarchy after the $adopt$ action happens.

3. $\underline{A_1 \text{ is the adoption of some subgoal w.r.t. some parent goal:}}$ This case is similar to case 2 above.

The proposition thus follows. □

Note that, unlike in the optimizing agent case, we need to ensure that $\Diamond\Phi$ is consistent with higher priority active p-goals after a happens in s, since the successor-state axiom for G in this case automatically drops such incompatible p-goals from the goal hierarchy. In the above proposition, this is guaranteed by checking that the action a is compatible with the agent's current intentions (i.e. that the agent does not have a c-goal not to execute a next). Thus in the absence of such an additional constraint, agents' p-goals may not persist. Moreover, changes in the agent's knowledge via knowledge-producing actions may also force her to drop the p-goal, e.g. if she learns that the goal is indeed impossible to achieve. This shows that agents' p-goals are much more dynamic in the committed agent framework. Also, as in the optimizing agent case, the level n where $\Diamond\Phi$ is a p-goal may change, e.g. if the action performed is an $adopt$ action with priority higher than or equal to n. Again, I believe that the dropping of an unrelated p-goal should not affect persistence, and hence it should be possible to strengthen this proposition. This is left for future work.

I can show that this property also follows if we replace the consequent with CGoal($\Diamond\Phi, do(a, s))$, provided that a is executable in s:

Proposition 6.3.40 (Persistence of CGoals)**.**

$$\begin{aligned}\mathcal{D}^{SG}_{CAgt} \models\ & \text{PGoal}(\Diamond\Phi, n, s) \wedge \text{Executable}(s) \wedge \text{Know}(\neg\Phi, s) \wedge \text{Poss}(a, s) \\ & \wedge \neg a = drop(\psi) \wedge \neg\text{KnowProdAct}(a) \wedge \neg\text{CGoal}(\neg\exists s'.\ \text{Do}(a, now, s'), s) \\ & \supset \text{CGoal}(\Diamond\Phi, do(a, s)).\end{aligned}$$

Proof. From the antecedent and Definition 3.3.1, it follows that Executable$(do(a, s))$. The proposition follows from this and Propositions 6.3.39 and 6.3.15. □

Again, by Corollary 6.3.20, this proposition also follows if the consequent is replaced with a primary c-goal, i.e. PrimCGoal$(\Diamond\Phi, do(a, s))$. Also, this shows that in contrast to the optimizing agent framework, agents' chosen goals are more persistent in the committed agent framework in the sense that when actions are executable, the persistence of p-goals necessarily implies the persistence of chosen goals in this framework.

6.4 Discussion and Conclusion

In this chapter, I presented a variant of my original "optimizing agent" prioritized goals and subgoals framework (proposed in Chapter 4 and 5) that allows one to model more committed agents by using a restricted notion of desires/goals. Here, I required that initially the agent's goals are known to be possible and consistent with each other. I gave a modified successor-state axiom for the goal accessibility relation G and showed that this axiom, along with the modified action precondition axioms for adopt and drop actions, preserve the initially prescribed constraints on the agent's goals for all executable situations.

In this committed agent framework, all of the p-goals of the agent are always realistic and chosen (i.e. p-goals are c-goals too). Put otherwise, an agent's chosen goals in this framework are simply the consequential closure of her prioritized goals. As such, all priority levels are always active. As a consequence, an agent's p-goals are much more dynamic in contrast to the original (optimizing agent) framework – an agent will drop a p-goal if it becomes known to be impossible or inconsistent with other higher priority p-goals and knowledge. On the other hand, an agent's chosen goals in this framework are much more persistent than in the original framework. The agent will not drop a chosen goal ϕ simply because another higher priority chosen goal ϕ' has become impossible, triggering the activation of a third (higher priority than ϕ, lower priority than ϕ', and currently inactive) goal ψ that is inconsistent with ϕ; indeed such a goal ψ would have been dropped by the committed agent earlier, since it is inconsistent with the higher priority goal ϕ'. I proved variants of many of the properties (including introspection and persistence properties) that I showed for optimizing agents and discussed how they differ from those presented earlier in Chapter 4 and 5.

While it can be argued that agents specified in this committed agent framework are somewhat overly committed to their goals in the sense that they will ignore opportunities to bring about more valuable goals in favor of already committed to and conflicting goals, I maintain that this framework is nonetheless quite useful. First, it provides an alternative commitment strategy to that of optimizing agents. Secondly, this simple framework can be useful for dealing with complex domains and illustrating other orthogonal issues therein with more clarity. In fact, in the next chapter, I use this framework as a starting point for defining a rational BDI agent programming language, and discuss some issues with rationality that have not been addressed previously. The optimizing agent framework of Chapter 4 and 5 and the committed agent framework of this chapter sit at opposite end of the continuum. In the former, the agent is continuously reconsidering her commitment to goals and reoptimizing her choice over desires, while in the latter once a goal is no longer chosen/committed to, it is permanently dropped. Essentially, the committed agent thus eliminates “the opportunity analyzer and the filter override mechanism” [21] from the underlying deliberation model. Nevertheless, as mentioned in Chapter 4, it would be certainly interesting to study a hybrid account that achieves some sort of balance between these two different commitment strategies by providing some control over intention reconsideration. I leave this for future work.

Chapter 7

SR-APL : Specifying A Simple Rational Agent Programming Language with Prioritized Goals

7.1 Introduction

This chapter contributes to the foundations of Belief-Desire-Intention Agent Programming Languages/frameworks (BDI APLs), such as PRS [108], AgentSpeak [172], etc. Recall from Chapter 2 that while there has been much recent work on incorporating declarative goals in these APLs [100, 247, 185, 47, 101, 235, 186], to keep them tractable and practical, they sacrifice some principles of rationality. In particular, while selecting plans to achieve a declarative goal, they ignore other concurrent intentions of the agent. As a consequence, *the selected plan may be inconsistent with the agent's other intentions* (I call this the *intention consistency problem*). Thus the execution of such an intended plan can render other contemporary intentions impossible to bring about. Also, these APLs with Declarative Goals (APLwDGs, henceforth) typically rely on syntactic formalizations of declarative goals, subgoals, and their dynamics, whose properties are often not well understood. Often, achievement goals are the only type of temporally extended goals supported in these frameworks (e.g. [100, 47]).

In this chapter, I develop a logical framework for a BDI agent programming language with prioritized declarative goals called Simple Rational APL (SR-APL, henceforth), that addresses most of these deficiencies of previous APLwDGs. SR-APL combines ideas from the situation calculus-based Golog family of APLs (e.g. ConGolog [51]), my expressive semantic formalization of prioritized goals, subgoals, and their dynamics as specified in Chapter 6, and work on BDI APLs. I ensure that an SR-APL agent's chosen declarative goals and adopted plans are consistent with each other and with her knowledge. In doing this, I will address two fundamental questions about rational agency:

1. What does it mean for a BDI agent to be committed to concurrently execute a set of plans next while keeping the option of further commitments to other plans open, in a way that does not allow procrastination?

2. How can one ensure consistency between an agent's adopted declarative goals and adopted plans, given that some of the latter might be abstract, i.e., only

partially instantiated in the sense that they include subgoals for which the agent has not yet adopted a (concrete) plan?

I will show how agents specified in the SR-APL framework satisfy some key rationality requirements. SR-APL is not a practical implemented APL and more work is required to make it practical, perhaps by restricting the proposed representations and reasoning. The framework however tries to bridge the gap between agent theories and practical APLs by providing a model and specification of an idealized BDI agent whose behavior is closer to what a rational agent does. As such, it serves as a basis for understanding how compromises made during the development of a practical APLwDG affect the agent's rationality.

I start the chapter by discussing a motivating example that illustrates the main issues with current APLwDGs. I then describe the components of an SR-APL agent and specify the semantics of SR-APL. Following that, I show that my agents behave in ways that satisfy some key rationality principles. Finally, I summarize my results and discuss possible future work.

7.2 A Motivating Example

Consider a blocks world agent Agt_{BW} with domain $\mathcal{D}_{BW}$, where each block is one of four possible colors: blue, yellow, green, and red, represented by the four predicates

B(b), Y(b), G(b), and R(b), respectively. There is only one action $stack(b, b')$ for stacking block b onto block b'. A block b can be stacked on another block b' in situation s if b and b' represent distinct blocks, both b and b' are *clear* in s, and b is *on the table* in s:

Axiom 7.2.1.

$$\text{Poss}(stack(b, b'), s) \equiv b \neq b' \wedge \text{Clear}(b, s) \wedge \text{Clear}(b', s) \wedge \text{OnTable}(b, s).$$

There are no unstacking actions, so the agent cannot use a block to build two different towers at different times. In the following, I specify the successor-state axioms for the fluents in this domain. First of all, block b is *on the table* after some action a happens in situation s if and only if b is on the table in s and a is not the action of stacking it on top of another block b':

Axiom 7.2.2.

$$\forall b, a, s.\ \text{OnTable}(b, do(a, s)) \equiv \text{OnTable}(b, s) \wedge \neg \exists b'.\ a = stack(b, b').$$

Secondly, block b is *on block* b' after a has happened in s if and only if b is on the table in s and a refers to the action of stacking b on b', or if b is already on b' in s:

Axiom 7.2.3.

$$\forall b, b', a, s.\ \text{On}(b, b', do(a, s)) \equiv (\text{OnTable}(b, s) \wedge a = stack(b, b')) \vee \text{On}(b, b', s).$$

Thirdly, in this domain, the color of the blocks (i.e. *blue, yellow, green,* and *red*) does not change:

Axiom 7.2.4.

$$(i).\ \forall b, a, s.\ \mathrm{B}(b, do(a, s)) \equiv \mathrm{B}(b, s).$$

$$(ii).\ \forall b, a, s.\ \mathrm{Y}(b, do(a, s)) \equiv \mathrm{Y}(b, s).$$

$$(iii).\ \forall b, a, s.\ \mathrm{G}(b, do(a, s)) \equiv \mathrm{G}(b, s).$$

$$(iv).\ \forall b, a, s.\ \mathrm{R}(b, do(a, s)) \equiv \mathrm{R}(b, s).$$

Finally, I say that block b is *clear* in situation s if there are no blocks on b:

Definition 7.2.5.

$$\forall b, s.\ \mathrm{Clear}(b, s) \overset{\text{def}}{=} \neg\exists b'.\ \mathrm{On}(b', b, s).$$

Assume that there are four blocks, B_B, B_Y, B_G, and B_R, one of each color. The agent Agt_{BW} knows the color of these blocks, and knows that initially all the blocks are on the table and are clear:

Axiom 7.2.6.

$$\text{Know}(\forall x.\ \text{OnTable}(x) \equiv x = B_B \vee B_Y \vee B_G \vee B_R, S_0) \wedge$$

$$\text{Know}(\forall x.\ \text{Clear}(x) \equiv x = B_B \vee B_Y \vee B_G \vee B_R, S_0) \wedge$$

$$\text{Know}(\forall x.\ \text{B}(x) \equiv x = B_B, S_0) \wedge \text{Know}(\forall x.\ \text{Y}(x) \equiv x = B_Y, S_0) \wedge$$

$$\text{Know}(\forall x.\ \text{G}(x) \equiv x = B_G, S_0) \wedge \text{Know}(\forall x.\ \text{R}(x) \equiv x = B_R, S_0).$$

Now let us go back to our discussion of BDI agent programming languages. Recall from Chapter 2 that a typical BDI APLwDG uses a user-specified hierarchical plan library Π containing planning rules, a procedural goal-base Γ containing a set of (possibly abstract) plans that the agent is committed to executing, and a declarative goal-base Δ containing the set of declarative goals that the agent is committed to achieving. In response to events in the environment and to goals in Δ, in each cycle the agent interleaves selecting plans from Π to handle such events and goals, adopting them to Γ, and executing actions from plans in Γ. The execution of some of these actions can in turn trigger the adoption of other declarative goals. This process is repeated until all the goals in Δ are successfully achieved and all plans in Γ are successfully executed. In the following, I informally discuss a fundamental problem with such a typical APLwDG, namely the intention consistency problem: when a new plan is adopted, the plan is not required to be consistent with an agent's current intentions, and thus the intended procedural and declarative goals may become inconsistent in these APLwDGs. Later, I will formally prove that an SR-APL agent is free from this problem.

To this end, assume that our agent Agt_{BW} has the following two declarative goals: (1) to eventually have a 2 blocks tower that has a green block on top and a non-yellow block underneath, and (2) to have a 2 blocks tower with a blue block on top and a non-red block underneath; thus $\Delta = \{\Diamond \text{Tower}^{G}_{\bar{Y}}, \Diamond \text{Tower}^{B}_{\bar{R}}\}$, where these goals are defined as follows:

Definition 7.2.7.

$$\text{Tower}^{G}_{\bar{Y}} \overset{\text{def}}{=} \exists b, b'.\ \text{OnTable}(b') \wedge \text{On}(b, b') \wedge \text{G}(b) \wedge \neg\text{Y}(b'),$$

$$\text{Tower}^{B}_{\bar{R}} \overset{\text{def}}{=} \exists b, b'.\ \text{OnTable}(b') \wedge \text{On}(b, b') \wedge \text{B}(b) \wedge \neg\text{R}(b').$$

Assuming that the goal $\Diamond \text{Tower}^{G}_{\bar{Y}}$ has higher priority than $\Diamond \text{Tower}^{B}_{\bar{R}}$, the following initial goal axioms can be used to specify the agent Agt_{BW}'s goals:[51]

Axiom 7.2.8.

$$\text{(a). Init}(s) \supset ((G(p, 0, s) \equiv \exists s'.\ \text{Starts}(p, s') \wedge K(s', s) \wedge \Diamond \text{Tower}^{G}_{\bar{Y}}(p))$$

$$\wedge ((G(p, 1, s) \equiv \exists s'.\ \text{Starts}(p, s') \wedge K(s', s) \wedge \Diamond \text{Tower}^{B}_{\bar{R}}(p))).$$

$$\text{(b). Init}(s) \wedge n \geq 2 \supset (G(p, n, s) \equiv \exists s'.\ \text{Starts}(p, s') \wedge K(s', s)).$$

[51]Note that in these axioms, I require that G-accessible paths start with a K-accessible situation in s (rather than an arbitrary initial situation); this is needed to comply with Assumption 6.2.1.

Suppose that our agent Agt_{BW}'s plan library Π_{BW} has two planning rules:[52]

Definition 7.2.9.

$$\begin{aligned}\Pi_{BW} \overset{\text{def}}{=} \{\ \Diamond\text{Tower}^{G}_{\bar{Y}} : [&\text{OnTable}(b) \wedge \text{OnTable}(b') \wedge b \neq b' \wedge \text{Clear}(b) \\ &\wedge \text{Clear}(b') \wedge \text{G}(b') \wedge \neg\text{Y}(b)] \leftarrow stack(b', b), \\ \Diamond\text{Tower}^{B}_{\bar{R}} : [&\text{OnTable}(b) \wedge \text{OnTable}(b') \wedge b \neq b' \wedge \text{Clear}(b) \\ &\wedge \text{Clear}(b') \wedge \text{B}(b') \wedge \neg\text{R}(b)] \leftarrow stack(b', b)\ \}.\end{aligned}$$

That is, if the agent Agt_{BW} has the goal to have a green and non-yellow tower and knows about a green block b' and a distinct non-yellow block b that are both clear and are on the table, then she should adopt the plan of stacking b' on b, and similarly for the goal of having a blue and non-red tower. Let's define our blocks world domain $\mathcal{D}_{BW}$ as $\mathcal{D} \cup \{$Axiom 7.2.1 – Axiom 7.2.8$\}$.

Now, consider a typical APLwDG, that (without considering the overall consistency of the agent's intentions) simply selects plans whose context condition is satisfied from the rule-base Π for the agent's goals in the declarative goal-base Δ, adds these selected plans to the agent's plan-base Γ, and eventually executes them in an attempt to achieve her goals.[53] I claim that such an APL is not always sound and ra-

[52]Recall that a planning rule of the form $\phi : \Psi \leftarrow \sigma$ means that the agent should consider adopting and executing the plan σ if she has the goal that ϕ and she currently knows/believes that Ψ.

[53]While I will not attempt to do so, it would not be hard to modify the proposed language definition in the next section to specify such an APLwDG.

tional in this example domain. For instance, according to this plan library, one way of building a green non-yellow (and a blue non-red) tower is to construct a green-blue (a blue-green, respectively) tower. While these two plans are individually consistent, they are inconsistent with each other, since the agent Agt_{BW} has only one block of each color. Thus a rational agent should not adopt these two plans. However, the following would be a legal trace for our blocks world domain in such a typical APLwDG (here as usual a configuration $\langle \Gamma, \Delta \rangle$ is a tuple consisting of the set of intended plans Γ and the set of intended declarative goals Δ of the agent; also $\mathcal{C}_1 \Rightarrow \mathcal{C}_2$ denotes a transition from configuration $\mathcal{C}_1$ to configuration $\mathcal{C}_2$ that is allowed by the transition system semantics of the APL):

$$\langle \{\}, \Delta \rangle \Rightarrow \langle \{stack(B_B, B_G)\}, \Delta \rangle \Rightarrow \langle \{stack(B_B, B_G), stack(B_G, B_B)\}, \Delta \rangle \Rightarrow$$
$$\langle \{stack(B_G, B_B)\}, \{\Diamond \text{Tower}_{\bar{Y}}^{G}\} \rangle.$$

The agent Agt_{BW} first moves to configuration $\langle \{stack(B_B, B_G)\}, \Delta \rangle$ by adopting the plan $stack(B_B, B_G)$ in response to $\Diamond \text{Tower}_{\bar{R}}^{B}$, then to $\langle \{stack(B_B, B_G), stack(B_G, B_B)\}, \Delta \rangle$ by adopting $stack(B_G, B_B)$ to handle $\Diamond \text{Tower}_{\bar{Y}}^{G}$, and then to $\langle \{stack(B_G, B_B)\}, \{\Diamond \text{Tower}_{\bar{Y}}^{G}\} \rangle$ by executing the intended action $stack(B_B, B_G)$. At this point, Agt_{BW} is stuck and cannot perform any further transitions and complete successfully. Thus, in such an APL, not only is the agent allowed to adopt two inconsistent plans, but the execution of one of these plans makes other concurrent goals impossible (e.g. the execution of $stack(B_B, B_G)$ makes the higher priority goal $\Diamond\text{Tower}_Y^G$ impossible to achieve).

The problem arises in part because actions are not reversible in this domain; there is no action for moving a block back to the table or for unstacking it. This is common in real world domains, for instance, most tasks with deadlines or that consume resources are such, e.g. doing some errands before noon, a robot delivering mail without running out of battery power, etc. While such irrational behavior could in principle be avoided by using appropriate conditions in the antecedent of the plan-selection rules (e.g. by stating that the agent should only adopt a given plan if she does not have certain other interacting goals), this puts an excessive burden on the agent programmer. Ideally, such reasoning about goals should be delegated to the agent.

7.3 Agent Programming with Prioritized Goals

The proposed framework SR-APL combines elements from BDI agent programming languages such as AgentSpeak [172] and from the ConGolog logic programming language [51], which is defined on top of the situation calculus. In addition, to facilitate monitoring of goal achievement and performing plan failure recovery, I incorporate declarative goals in SR-APL. To specify the operational semantics of plans in SR-APL, I will use (elements of) the semantics of the ConGolog APL (see Chapter 3).

7.3.1 Components of SR-APL

Before going over the operational semantics of SR-APL, I first discuss its various components. First of all, I have a *set of axioms/theory* $\mathcal{D}^{SG}_{CAgt}$ specifying actions that can be done, the initial knowledge and goals of the agent, and their dynamics, as discussed in Chapter 6. Henceforth, I will use $\mathcal{D}$ to refer to this theory. Recall that in the committed agent framework, I allow the agent to have infinitely many p-goals. However, here I assume a finite set of initial p-goals. I also assume that these are all achievement goals $\Diamond\Phi$, where Φ is a situation suppressed formula without temporal operators and without epistemic or goal operators. Since a finite number of achievement p-goals is assumed, I can use the abbreviation NPGoals(n, s), which states that n is the highest priority level starting where (and after which) the agent's goal hierarchy is empty, i.e. where the agent has the trivial p-goal that she is in a K-accessible situation. Thus if NPGoals(n, s) holds, then adopting a p-goal at level n in situation s essentially amounts to adopting the p-goal at a lower priority level than any other existing p-goals in s. I define NPGoals(n, s) as below.

Definition 7.3.1.

$$\text{NPGoals}(n, s) \overset{\text{def}}{=} (\forall m.\ m \geq n \supset \text{NoGoalsAt}(m, s))$$

$$\land\ (\neg\text{NoGoalsAt}(n - 1, s) \lor n = 0),$$

$$\textit{where, } \text{NoGoalsAt}(n, s) \overset{\text{def}}{=} \text{OPGoal}(\exists s'.\ K(s', s) \land \text{Starts}(s'), n, s).$$

That is, NPGoals(n, s) holds in some situation s if n is the highest priority level where the agent's p-goals at level n and all lower priority levels are the trivial p-goal that she be in a K-accessible situation. Recall from the SSA for G (i.e. Axiom 6.2.6) that when a p-goal at some level m is dropped or becomes impossible or inconsistent with other higher priority p-goals and knowledge, the G-accessible paths at m are simply replaced with paths that starts with a current K-accessible situation. In other words, my SSA for G does not compact levels in the sense of removing such an empty level altogether from the goal hierarchy. The above definition takes this into account and thus there may be a level m that has higher priority than n and where the agent has the trivial p-goal that she be in a K-accessible situation, as long as there is at least one "non-empty" level with priority lower than m.

Note that in the situation calculus, both declarative and procedural goals are given declarative semantics. For instance, the Do construct is used to represent an agent's adopted plans: having the goal that $\exists s'.\ \text{Starts}(now) \land \text{OnPath}(s') \land \text{Do}(\sigma, now, s')$

amounts to having the adopted plan that σ. Thus unlike in other APLwDGs, I use the theory $\mathcal{D}$ to uniformly specify *both declarative and procedural goals* of the agent. This considerably simplifies the task of maintaining the consistency between these two types of goals compared to APLs where declarative and procedural goals are stored in two separate goal bases.

Moreover, I also have a *plan library* Π with rules of the form $\phi : \Psi \leftarrow \sigma$, where ϕ must be an achievement goal formula of the form $\Diamond\Phi$, Ψ is an situation suppressed formula expressing a condition on what the agent knows, and σ is a plan; $\phi, \Psi,$ and σ may have free variables in them such that free$(\sigma) \subseteq$ free$(\phi) \cup$ free(Ψ). A rule $\phi : \Psi \leftarrow \sigma$ means that if the agent has some instance of the c-goal that ϕ (i.e. has the ground c-goal that ϕ', where ϕ' is $\phi\theta$ with some substitution θ of the free variables in ϕ) and knows that some ground instance of Ψ, $\Psi'\theta$ holds, then she should consider adopting the plan that $\sigma\theta$ (for the substitution θ of the free variables in ϕ and Ψ). The *plan language* for σ is a simplified version of ConGolog and includes the constructs listed in Table 7.1. I use the ConGolog APL here because it has a situation calculus-based semantics that is well specified and compatible with my agent theory. I could have used any APL with these characteristics. Also, in order to keep the proposed framework in line with current APLwDGs, I only use a subset of the ConGolog constructs. Here I assume that primitive actions a in a plan cannot be exogenous. In addition to these ConGolog con-

nil	The empty program
a	Primitive action
$\phi?$	Waiting for a condition
$\sigma_1; \sigma_2$	Sequence
$adoptRelTo(\Diamond\Phi, \exists s, s'.\ \text{Starts}(s) \wedge \text{OnPath}(s') \wedge \text{DoAL}(\sigma, s, s'))$	Subgoal adoption

Table 7.1: SR-APL Plan Language

structs, the plan language includes the special action for declarative subgoal adoption, $adoptRelTo(\Diamond\Phi, \psi)$ as specified in Chapter 6; here $\Diamond\Phi$ is an achievement declarative subgoal to be adopted and ψ is a path formula relative to which it is adopted. ψ is of the form $\exists s, s'.$ Starts(s) $\wedge$ OnPath(s') $\wedge$ DoAL(σ, s, s'), which roughly says that ψ holds over a path if the plan σ, possibly along with other actions is executed over the path starting from the starting situation of the path (see Section 7.3.2 for the definition of DoAL). Thus $adoptRelTo(\Diamond\Phi, \psi)$ here refers to the adoption of the achievement declarative goal $\Diamond\Phi$ w.r.t. the goal to execute σ, possibly along with other actions. While my account of goal change is expressive enough to handle arbitrary temporally extended goals, here I focus on achievement goals and procedural goals exclusively. Thus I assume that the goal formula ϕ in a rule $\phi : \Psi \leftarrow \sigma$ must be an achievement goal $\Diamond\Phi$. Note that extending my framework to support maintenance goals should be straightforward, since maintenance goals behave like additional constraints on the agent behavior in contrast to achievement goals for which the agent needs to plan for. Nevertheless, I leave this for future work.

I now define an SR-APL agent as a tuple, $\langle \mathcal{D}, \Pi \rangle$. Here, $\mathcal{D}$ is a (committed agent) theory specifying the domain, i.e. the actions and the initial knowledge and goals of the agent, and their dynamics. I assume that $\mathcal{D}$ is complete with respect to the actual initial state, and thus for any situation suppressed formula Φ without K or G, either $\mathcal{D} \models \Phi(S_0)$ or $\mathcal{D} \models \neg\Phi(S_0)$.[54] Also, I constrain the initial goals of the agent in $\mathcal{D}$ to declarative achievement goals only. Finally, Π is a plan library with rules of the form $\phi : \Psi \leftarrow \sigma$ that allows the agent to adopt new plans w.r.t. her declarative achievement goals given her knowledge. ϕ here must be a declarative achievement goal $\Diamond\Phi$, and σ is formed using the constructs in Table 7.1. Thus, for example, our blocks world agent Agt_{BW} in the initial situation can be specified using the program $\langle \mathcal{D}_{BW}, \Pi_{BW} \rangle$.

7.3.2 Semantics of SR-APL

Using these components, I next specify the semantics of SR-APL. I use a transition system [168] for this. As mentioned above, I use my situation calculus domain theory $\mathcal{D}$ to represent both adopted declarative goals and procedural goals/plans and how they

[54]This is required to handle sensing actions and exogenous actions that carry new information properly in the context of my meta-theoretical approach to the semantics of SR-APL below. I will come back to this issue later.

evolve. Initially the goals in $\mathcal{D}$ are restricted to be achievement declarative goals only. As specified by the successor-state axiom for G (i.e. Axiom 6.2.6), the goals in $\mathcal{D}$ are updated by adding plans or other declarative goals to the agent's goal hierarchy when a transition rule makes the agent perform an $adopt$ or $adoptRelTo$ action (note that while the SR-APL plan language in Table 7.1 does not include $adopt$ actions, the transition rules in Table 7.3 utilize both of these actions). I ensure that an agent's declarative goals and adopted plans are consistent with each other and with the agent's knowledge. In my semantics, I specify this by checking that there exists a course of actions that the agent considers possible (i.e. a realistic path), and if she were to follow this path, she would end up realizing all of her declarative goals and executing all of her procedural goals.

Recall from Section 7.1 that specifying such a language raises some fundamental questions about rational agency, for instance: *how to specify a BDI agent's open-ended commitment towards concurrently executing multiple plans while ensuring that the agent does not procrastinate?* An SR-APL agent can work on multiple goals at the same time. Thus at any time, an agent might be committed to several plans that she will be executing in an interleaved fashion. We need a mechanism to model the agent's adopted plans in the goal hierarchy specified by $\mathcal{D}$. One way of specifying an agent's commitment to execute a plan σ next at some level n in $\mathcal{D}$ is to say that she has the p-goal at n that $\exists s, s'$. Starts$(s) \wedge$ OnPath$(s') \wedge$ Do(σ, s, s'), i.e. that each of her G-accessible paths p at n is such that it starts with some situation s, it has the situation s' on it, and s' can be reached from s by executing σ. However, this does not allow for the interleaved execution of several plans, since Do requires that σ be executed before any other actions/plans.

A better alternative is to represent the procedural goal as $\exists s, s'$. Starts$(s) \wedge$OnPath(s') $\wedge$ DoAL(σ, s, s'), which says that the agent has the p-goal at level n to do *at least* the plan σ next, and possibly more. DoAL(σ, s, s') holds if there is an execution of plan σ, possibly interleaved with zero or more actions *by the agent herself*, that starts in situation s and ends in s':[55]

Definition 7.3.2.

$$\text{DoAL}(\sigma, s, s') \overset{\text{def}}{=} \text{Do}(\sigma \| (\pi a.\ \text{Agent}(a) = agt?; a)^*, s, s').$$

In addition to providing a mechanism for combining the already adopted plans, this also allows the agent to be open towards future commitments to other plans. Put otherwise, the DoAL construct in her already committed to plans allows her to accommodate other plans that she might have already or adopt in the future.

However, a new problem with this approach is that it allows the agent to *procras-*

[55]Note that, while my theory supports exogenous actions performed by other agents, I assume that all actions in the plans of agt that specify her behavior must be performed by agt herself.

tinate in the execution of the intended plans in $\mathcal{D}$. For instance, suppose that the agent has the p-goal at priority level n_1 to execute the plan σ_1 and at level n_2 to execute σ_2 next. Then, according to my definition of DoAL, the agent has the intention at level n_1 to execute σ_1 and at level n_2 to execute σ_2, possibly concurrently with other actions next, since I use DoAL to specify those goals. The "other actions" at level n_1 (n_2, respectively) are meant to be actions from the plan σ_2 (σ_1, respectively). However, nothing requires that the additional actions that the agent might execute are indeed from σ_2(σ_1, respectively), and thus this allows her to perform actions that are completely unnecessary as long as they do not make the execution of σ_1 and σ_2 impossible.

To deal with this, I include an additional component, a *procedural intention-base* Γ, in an SR-APL agent configuration. Γ is a finite list of plans that the agent is currently actively pursuing. To avoid procrastination, I will require that any action that the agent actually performs comes from Γ (as specified in the transition rule A_{step} below). In the following, I will use $\Gamma^{\|}$ to denote the concurrent composition of the plans in Γ:[56]

Definition 7.3.3.

$$\Gamma^{\|} \overset{\text{def}}{=} \textbf{if } (\Gamma = [\,]) \textbf{ then } \textit{nil} \textbf{ else } \text{First}(\Gamma)\|(\text{Rest}(\Gamma)^{\|}).$$

[56]I will use various standard list operations, e.g. First (representing the first item of a list), Rest (representing the sublist that contains all but the first item of a list), Cons (for constructing a new list from an item and a list), Member (for checking membership of an item within a list), Remove (for removing all occurrences of a given item from a list), Replace (for replacing all occurrences of a given item by another item in a list), etc.

In SR-APL, a *plan configuration* $\langle \sigma, s \rangle$ is a tuple consisting of an SR-APL plan σ and a ground situation term s. An *agent configuration* on the other hand is a tuple $\langle \Gamma, s \rangle$ that consists of a list of plans Γ and a ground situation term s. The *initial agent configuration* is $\langle [\,], S_0 \rangle$. Note that implicitly an agent configuration also includes the knowledge and the goals of the agent, but that these can be obtained from the (fixed) theory $\mathcal{D}$ and the situation in the configuration.

The semantics of SR-APL are defined by a two-tier transition system. *Plan-level transition rules* specify how a plan written in SR-APL's plan language may evolve. These rules are defined within the language/theory. On top of this, I define *agent-level transition rules* to specify how an SR-APL agent's configuration may evolve. Unlike the plan-level transition rules, these rules are defined meta-theoretically as these involve dealing with the procedural intention-base Γ.

Plan-Level Transition Rules

The plan-level transition rules are simply a subset of the ConGolog transition rules specified in Axioms 3.6.1 and 3.6.2. Since plans are written using SR-APL's plan language in Table 7.1, I only need the subset for the constructs listed in Table 7.1. Here, I will use $\langle \sigma, s \rangle \rightarrow \langle \sigma', s' \rangle$ as an alternative notation for $\mathrm{Trans}(\sigma, s, \sigma', s')$. For the reader's convenience, I list these transition rules again in Table 7.2 (note that the

Γ_{F_1}. $\quad$ $\mathrm{Final}(nil, s) \equiv \mathrm{True},$

Γ_{F_2}. $\quad$ $\mathrm{Final}(a, s) \equiv \mathrm{False},$

Γ_{F_3}. $\quad$ $\mathrm{Final}(\phi?, s) \equiv \phi(s),$

Γ_{F_4}. $\quad$ $\mathrm{Final}([\delta_1; \delta_2], s) \equiv \mathrm{Final}(\delta_1, s) \wedge \mathrm{Final}(\delta_2, s),$

Γ_{T_1}. $\quad$ $\neg\exists \delta', s'.\ \langle nil, s\rangle \rightarrow \langle \delta', s'\rangle,$

Γ_{T_2}. $\quad$ $\langle a, s\rangle \rightarrow \langle \delta', s'\rangle \equiv \mathrm{Poss}(a, s) \wedge \delta' = nil \wedge s' = do(a, s),$

Γ_{T_3}. $\quad$ $\neg\exists \delta', s'.\ \langle \phi?, s\rangle \rightarrow \langle \delta', s'\rangle,$

Γ_{T_4}. $\quad$ $\langle [\delta_1; \delta_2], s\rangle \rightarrow \langle \delta', s'\rangle \equiv \exists \delta_1'.\ (\delta' = [\delta_1'; \delta_2] \wedge \langle \delta_1, s\rangle \rightarrow \langle \delta_1', s'\rangle)$

$\qquad\qquad \vee\ \mathrm{Final}(\delta_1, s) \wedge \langle \delta_2, s\rangle \rightarrow \langle \delta', s'\rangle.$

Table 7.2: Plan-Level Transition Rules

$adoptRelTo$ special action is handled by the same rule as normal actions).

Agent-Level Transition Rules

The agent-level transition rules are given in Table 7.3 and are mostly similar to those of a typical BDI APL.[57] First of all, I have a rule A_{sel} for *selecting and adopting a plan*

[57]In the following, I use $\mathrm{CGoal}(\exists s'.\ \mathrm{DoAL}(\sigma, now, s'), s)$ or simply $\mathrm{CGoal}(\mathrm{DoAL}(\sigma), s)$ as a shorthand for $\mathrm{CGoal}(\exists s'.\ \mathrm{Starts}(path, now) \wedge \mathrm{OnPath}(path, s') \wedge \mathrm{DoAL}(\sigma, now, s'), s)$. Thus, $\mathrm{DoAL}(\sigma)$ or $\exists s'.\ \mathrm{DoAL}(\sigma, now, s')$ here is the path formula $\exists s'.\ \mathrm{Starts}(path, now) \wedge \mathrm{OnPath}(path, s') \wedge \mathrm{DoAL}(\sigma, now, s')$ that has a free path variable/placeholder $path$ that is often suppressed; $path$ will get bound by the context in which the formula $\mathrm{DoAL}(\sigma)$ appears. Similarly, I will use $\mathrm{DoAL}(\sigma)$ as an argument to the $adoptRelTo$ action as in $adoptRelTo(\psi, \mathrm{DoAL}(\sigma))$, as a shorthand for $adoptRelTo(\psi, \exists s'.\ \mathrm{Starts}(now) \wedge \mathrm{OnPath}(s') \wedge \mathrm{DoAL}(\sigma, now, s'))$. Finally, while $\mathrm{DoAL}(\sigma)$ is not technically a program/plan and rather a predicate, here I often abuse the notation and say, e.g. that "$\mathrm{DoAL}(\sigma)$ is executable" rather than that "$\mathrm{DoAL}(\sigma)$ holds", etc.; in these I am actually referring to the

using the plan library Π for some realistic achievement p-goal $\Diamond\Phi$. It states that if:

(a) there is a rule in the plan library Π which says that the agent should adopt an instance of the plan σ if she has an instance of $\Diamond\Phi$ as her p-goal at some level n and knows that some instance of Ψ holds,

(b) $\Diamond\Phi'$ is a (realistic) p-goal with priority n in situation s for which the agent hasn't yet adopted any subgoal,

(c) the agent knows in s that Ψ',

(d) θ_1 unifies Φ and Φ', i.e. mgu$(\Phi, \Phi') = \theta_1$, and θ_2 unifies Ψ and Ψ', i.e. mgu$(\Psi, \Psi') = \theta_2$, and

(e) the adoption of the plan of doing at least $\sigma\theta_1\theta_2$ w.r.t. the p-goal $\Diamond\Phi\theta_1$ is possible in s,

then she can adopt the plan $\sigma\theta_1\theta_2$, adding DoAL$(\sigma\theta_1\theta_2)$ as a subgoal of $\Diamond\Phi\theta_1$ to her goals in the theory $\mathcal{D}$, and adding $\sigma\theta_1\theta_2$ to Γ. Here, I say that a theory $\mathcal{D}$ entails that some goal ϕ has been "handled" in some situation s if and only if there exists a goal ψ such that $\mathcal{D}$ entails that ψ is a subgoal of ϕ in s, i.e.:

program $(\sigma \| (\pi a.\ \text{Agent}(a) = agt?; a)^*)$.

Definition 7.3.4.

$$\text{Handled}(\phi, s, \mathcal{D}) \textit{ iff there exists a } \psi \textit{ such that } \mathcal{D} \models \text{SubGoal}(\psi, \phi, s).$$

Recall that, by Axiom 6.2.4, it follows that if the action of adopting a subgoal ψ relative to a parent goal ϕ is executable in some situation s, then the agent does not have the c-goal that $\neg\psi$ next in s. Assuming that the agent already has the c-goal to execute DoAL($\Gamma^{\|}$) in s,[58] this and condition (e) above imply that the agent does not have the c-goal not to execute $\sigma\theta_1\theta_2$ concurrently with $\Gamma^{\|}$ and possibly other actions next:

$$\text{(I). } \neg\text{CGoal}(\neg\exists s', s''.\ \text{Do}(\mathit{adoptRelTo}(\text{DoAL}(\sigma\theta_1\theta_2), \Diamond\Phi\theta_1), \mathit{now}, s') \\ \wedge \text{DoAL}(\sigma\theta_1\theta_2 \parallel \Gamma^{\|}, s', s''), s).$$

Moreover, from Proposition 6.3.30, it follows that an SR-APL agent acquires the c-goal that ψ after she adopts it as a subgoal of ϕ in s, provided that s is an executable situation and that the $\mathit{adoptRelTo}(\psi, \phi)$ action is executable in s. Thus for any executable situation s, (assuming that the agent already has the c-goal to execute DoAL($\Gamma^{\|}$) in s) we have from this and (e) that:

$$\text{(II). CGoal}(\exists s'.\ \text{DoAL}(\sigma\theta_1\theta_2 \parallel \Gamma^{\|}, \mathit{now}, s'), \\ \mathit{do}(\mathit{adoptRelTo}(\text{DoAL}(\sigma\theta_1\theta_2), \Diamond\Phi\theta_1), s)).$$

[58]See Proposition 7.4.7 below, where I show that under appropriate conditions, this is indeed the case.

$$(\mathrm{A}_{sel})\qquad \frac{\begin{array}{c}\text{Member}(\Diamond\Phi : \Psi \leftarrow \sigma, \Pi),\ \ \mathcal{D} \models \text{PGoal}(\Diamond\Phi', n, s) \text{ for some } n,\\ \neg\text{Handled}(\Diamond\Phi', s, \mathcal{D}),\ \ \mathcal{D} \models \text{Know}(\Psi', s),\ \ \text{mgu}(\Phi, \Phi') = \theta_1,\ \ \text{mgu}(\Psi, \Psi') = \theta_2,\\ \mathcal{D} \models \text{Poss}(\mathit{adoptRelTo}(\text{DoAL}(\sigma\theta_1\theta_2), \Diamond\Phi\theta_1), s)\end{array}}{\langle \Gamma, s\rangle \Rightarrow \langle \text{Cons}(\sigma\theta_1\theta_2, \Gamma), \mathit{do}(\mathit{adoptRelTo}(\text{DoAL}(\sigma\theta_1\theta_2), \Diamond\Phi\theta_1), s)\rangle}$$

$$(\mathrm{A}_{step})\qquad \frac{\begin{array}{c}\text{Member}(\sigma, \Gamma),\ \ \mathcal{D} \models \text{Know}(\langle \sigma, \mathit{now}\rangle \rightarrow \langle \sigma', \mathit{do}(a, \mathit{now})\rangle, s),\\ \mathcal{D} \models \text{PGoal}(\text{DoAL}(\sigma), n, s) \text{ for some } n, \mathcal{D} \models \neg\text{CGoal}(\neg\exists s'.\ \text{Do}(a, \mathit{now}, s'), s)\end{array}}{\langle \Gamma, s\rangle \Rightarrow \langle \text{Replace}(\sigma, \sigma', \Gamma), \mathit{do}(a, s)\rangle}$$

$$(\mathrm{A}_{exo})\qquad \frac{\mathcal{D} \models \text{Exo}(a) \wedge \text{Poss}(a, s)}{\langle \Gamma, s\rangle \Rightarrow \langle \Gamma, \mathit{do}(a, s)\rangle}$$

$$(\mathrm{A}_{clean})\qquad \frac{\text{Member}(\sigma, \Gamma),\ \ \mathcal{D} \models \neg\exists n.\ \text{PGoal}(\text{DoAL}(\sigma), n, s)}{\langle \Gamma, s\rangle \Rightarrow \langle \text{Remove}(\sigma, \Gamma), s\rangle}$$

$$(\mathrm{A}_{rep})\qquad \frac{\begin{array}{c}\mathcal{D} \models \text{Know}(\neg\exists\Gamma', s'.\ \langle \Gamma^{\|}, \mathit{now}\rangle \rightarrow \langle \Gamma', s'\rangle, s)\ \ \mathcal{D} \models \text{Know}(\neg\text{Final}(\Gamma^{\|}, \mathit{now}), s),\\ \text{For all } \sigma \text{ such that Member}(\sigma, \Gamma) \text{ we have:}\\ \mathcal{D} \models \exists n.\ \text{PGoal}(\text{DoAL}(\sigma), n, s),\ \ \text{Handled}(\text{DoAL}(\sigma), s, \mathcal{D}),\\ \mathcal{D} \models \text{Agent}(\vec{a}) = \mathit{agt} \wedge \text{Know}(\exists s'.\ \text{Do}(\vec{a}, \mathit{now}, s') \wedge \exists \Gamma', s''.\ \langle \Gamma^{\|}, s'\rangle \rightarrow \langle \Gamma', s''\rangle, s),\\ \mathcal{D} \models \text{NPGoals}(m, s) \text{ for some } m,\ \ \mathcal{D} \models \text{Poss}(\mathit{adopt}(\text{Do}(\vec{a}), m), s)\end{array}}{\langle \Gamma, s\rangle \Rightarrow \langle \text{Cons}(\vec{a}, \Gamma), \mathit{do}(\mathit{adopt}(\text{Do}(\vec{a}), m), s)\rangle}$$

Table 7.3: Agent-Level Transition Rules

(I) ensures that the adopted subgoal $\sigma\theta_1\theta_2$ is consistent with $\Gamma^{\|}$ (and with all the declarative goals of the agent) in the sense that they can be executed concurrently, possibly along with other actions in s. (II) confirms that $\sigma\theta_1\theta_2$ is indeed intended after the $adoptRelTo$ action has happened. Note that this notion of consistency is a weak one, since it does not guarantee that there is an execution of the program $(\sigma\theta_1\theta_2 \parallel \Gamma^{\|})$ after the $adoptRelTo$ action happens, but rather ensures that DoAL$(\sigma\theta_1\theta_2 \parallel \Gamma^{\|})$ holds. In other words, $\sigma\theta_1\theta_2$ and the plans in Γ *alone* might not be concurrently executable, and additional actions might be required. I will come back to this issue later.

Secondly, I have a transition rule A$_{step}$ for single stepping the agent program by *executing an intended action* from the procedural goal-base Γ. It says that if:

(a) the agent knows that a plan σ in Γ can make a plan-level transition in situation s by performing a primitive action a with plan σ' remaining in $do(a, s)$ afterwards,

(b) DoAL(σ) is a (realistic) p-goal with priority n in s, and

(c) the transition is consistent with the agent's goals in the sense that she does not have the c-goal not to execute a in s,

then the agent can execute a, and Γ and s can be updated accordingly.

Once again I have a weak consistency requirement in condition (c) above. Ideally, I would have added to (c) that the agent can continue from $do(a, s)$ in the sense that she does not have the c-goal not to execute the remaining plan σ' concurrently with the other plans in Γ in $do(a, s)$, i.e. that:

$$\mathcal{D} \models \neg\text{CGoal}(\neg\exists s'.\ \text{Do}(a; \text{Replace}(\sigma, \sigma', \Gamma)^{\|}, now, s'), s).$$

However, this would be too demanding as Γ may be incomplete in the sense that it may include abstract plans that have actions that trigger the adoption of subgoals, for which the execution of $\Gamma^{\|}$ waits; but Γ does not have any adopted plans yet that can achieve these subgoals. Thus $\Gamma^{\|}$ by itself might currently have no complete execution, and will only become completely executable when all such subgoals have been fully expanded.

For example, consider a new agent for our blocks world domain Agt_{BW}^{3T} who has the goal to eventually build a 3 blocks tower, i.e. $\Diamond$3Tower, where 3Tower is defined as follows:

Definition 7.3.5.

$$\text{3Tower} \overset{\text{def}}{=} \exists b, b', b''.\ \text{OnTable}(b) \wedge \text{On}(b', b) \wedge \text{On}(b'', b').$$

The agent Agt_{BW}^{3T}'s initial goals can be defined using the following initial goal axiom:

Axiom 7.3.6.

(a). $\text{Init}(s) \supset G(p, 0, s) \equiv \exists s'.\ \text{Starts}(p, s') \wedge K(s', s) \wedge \Diamond\text{3Tower}(p).$

(b). $\text{Init}(s) \wedge n > 0 \supset G(p, n, s) \equiv \exists s'.\ \text{Starts}(p, s') \wedge K(s', s).$

The agent Agt^{3T}_{BW}'s domain theory $\mathcal{D}^{3T}_{BW}$ is the same as that for $\mathcal{D}_{BW}$, but with the substituted initial goal axioms, i.e. $\mathcal{D}^{3T}_{BW} \overset{\text{def}}{=} \mathcal{D}_{BW} \setminus \{\text{Axiom 7.2.9}\} \cup \{\text{Axiom 7.3.6}\}$. Also, in addition to the rules in Π_{BW}, her plan library Π^{3T}_{BW} as specified below includes a new planning rule that can be used to build a 3 blocks tower.

Definition 7.3.7.

$$
\begin{aligned}
\Pi^{3T}_{BW} \overset{\text{def}}{=} \Pi_{BW} \cup \{\Diamond\text{3Tower} : [&\neg\text{Y}(b) \wedge \text{G}(b') \wedge \text{Y}(b'') \wedge b \neq b' \wedge \text{Clear}(b) \wedge \text{Clear}(b') \\
&\wedge \text{Clear}(b'') \wedge \text{OnTable}(b) \wedge \text{OnTable}(b') \wedge \text{OnTable}(b'')] \\
&\leftarrow \sigma_1\},
\end{aligned}
$$

$\sigma_1 \overset{\text{def}}{=} adoptRelTo(\Diamond\text{Tower}^G_{\bar{Y}}, \text{DoAL}(\sigma_2)); \sigma_2,$

$\sigma_2 \overset{\text{def}}{=} \text{Tower}^G_{\bar{Y}}?; stack(b'', b').$

This new rule says that, if the agent knows about a non-yellow block b, a distinct green block b', and a yellow block b'' that are all clear and on the table, then her goal of building a 3 blocks tower can be fulfilled by adopting the plan that involves adopting the declarative subgoal to eventually build a green non-yellow tower, waiting for the achievement of this subgoal, and then stacking the yellow block on the green block. Now, suppose that in response to her goal $\Diamond$3Tower, the agent Agt^{3T}_{BW} adopted σ_1 as above as a subgoal of this goal using the A_{sel} rule, and thus σ_1 is added to the procedural goal-base Γ. In the next few steps, she will step through the adopted plan

σ_1, executing one action at a time in an attempt to achieve her goal that $\Diamond$3Tower.

Note that in SR-APL, the hierarchical decomposition of a subgoal, e.g. σ_1 above, is a two step process. In the first step, in response to the execution (via A$_{step}$) of the $adoptRelTo(\Diamond\text{Tower}_Y^G, \text{DoAL}(\sigma_2))$ action in her plan σ_1 in Γ, the agent adopts $\Diamond$Tower$_Y^G$ as a subgoal of the remaining plan σ_2, possibly along with other actions, i.e. relative to DoAL(σ_2). Then in the second step, she uses the A$_{sel}$ rule to select and adopt a plan for the subgoal $\Diamond$Tower$_Y^G$. I assume that the subgoal $\Diamond$Tower$_Y^G$ must always be achieved before the supergoal. To do this, I suspend the execution of the supergoal by waiting for the achievement of the subgoal. This can be specified by the programmer by having the supergoal σ_2 start with the wait action Tower$_Y^G$? that waits for the subgoal to be achieved. But this means that σ_2 (and thus σ_1) by itself, i.e. without the DoAL construct, might not have a complete execution as it might get blocked when it reaches Tower$_Y^G$?. Moreover, since σ_2 is a member of Γ, $\Gamma^{||}$ will have a complete execution only when all the subgoals in Γ have been fully expanded. To deal with this in rule A$_{sel}$, I use a weak consistency check that does not verify if $\Gamma^{||}$ is executable, but instead only guarantees that DoAL$(\Gamma^{||})$ is executable. However, my semantics ensures that any action a performed by the agent must not make the concurrent execution of all the adopted plans of the agent possibly with other actions impossible, i.e. it must be consistent with DoAL$(\Gamma^{||})$, since A$_{step}$ requires that doing a must be consistent with all her DoAL procedural goals (and other concurrent declarative goals) in her goal hierarchy, i.e. that $\mathcal{D} \models \neg\text{CGoal}(\neg\exists s'.\ \text{Do}(a, now, s'), s)$. This weak notion of consistency thus addresses another fundamental issue of rational agency, namely, *maintaining consistency between an agent's declarative and procedural goals, while allowing the latter to include abstract plans with unexpanded subgoals*.

In my definition of an SR-APL agent, I require that the theory $\mathcal{D}$ be complete w.r.t. the actual initial state S_0. This is required to handle sensing. Without this assumption, the agent may get stuck due to lack of knowledge about whether some action in the current situation is executable or not, even after performing a corresponding sensing action. For example, consider the plan where the agent senses the value of Φ (which she doesn't know initially) and then perform action a if Φ holds and b if $\neg\Phi$ holds.[59] Suppose also that a is only possible if Φ holds and b is only possible if $\neg\Phi$ holds. If the theory $\mathcal{D}$ did not say what the actual value of Φ is in S_0, then neither a nor b could be performed by the A$_{step}$ rule after the sensing of Φ because it would not be entailed that the agent knows that a is executable after $sense_\Phi$ occurs, and similarly for b. Note that given the above completeness assumption (which for the example says that either $\mathcal{D} \models \Phi(S_0)$ or $\mathcal{D} \models \neg\Phi(S_0)$), if $\mathcal{D}$ entails that Φ holds initially, $\mathcal{D}$ will also entail that

[59]While the SR-APL plan language does not include non-deterministic branch, this can be simulated using the plan $sense_\Phi; adoptRelTo(\Diamond\Psi, \text{True})$ and two planning rules in the rule library Π that condition on the outcome of the sensing.

that the agent knows that a is executable after $sense_\Phi$ occurs, and similarly for the $\neg\Phi$ case.

Note that another way (that does not require this constraint on $\mathcal{D}$) to address this issue with sensing is to update $\mathcal{D}$ after a sensing action happens to add the sensed value observed, i.e. by choosing a sensed fluent value that is consistent with $\mathcal{D}$ and changing $\mathcal{D}$ to $\mathcal{D} \cup \{\text{SF}(a, s)\}$ or $\mathcal{D} \cup \{\neg\text{SF}(a, s)\}$. But this requires storing the updated theory or the sensed values in the agent configuration. The first approach generates the possible executions of the agent for a given environment (the one specified by $\mathcal{D}$). If there is uncertainty about the environment, each model needs to be considered separately. The second approach generates the possible executions for all possible environments (i.e. models of $\mathcal{D}$). For simplicity, here I adopt our approach.

Thirdly, I have a rule A_{exo} for *accommodating exogenous actions*, i.e. actions occurring in the agent's environment that are not under her control. When such an action a occurs in situation s, the agent must update her knowledge and goals by progressing the situation component of her configuration to $do(a, s)$, provided that doing a is possible in s. Note that, the condition in the antecedent of the A_{exo} rule is a strong one since here I require that the theory $\mathcal{D}$ entails that the exogenous action a is possible in situation s, i.e. $\mathcal{D} \models \text{Poss}(a, s)$. This eliminates some possible executions of the agent program that involve exogenous actions. For instance, it may be the case that

$\text{Poss}(a, s)$ holds in some model of $\mathcal{D}$ but does not hold in another. In such a case, the agent should have considered the transition of a in s as a legal one. However, this cannot arise in our case as we assume that $\mathcal{D}$ is complete w.r.t. the actual initial state. I could have dropped this assumption and used the weaker condition that $\mathcal{D} \cup \{\text{Exo}(a)\} \cup \{\text{Poss}(a, s)\}$ is satisfiable in the antecedent of the A_{exo} rule, but again this requires updating the theory with $\text{Know}(\text{Poss}(a, now), s)$ after a happens and storing the theory in the agent configuration. For simplicity, I stick with the condition that $\text{Poss}(a, s)$ be entailed by $\mathcal{D}$ here.

Fourthly, I use the A_{clean} rule for *dropping adopted plans that are no longer intended in the theory $\mathcal{D}$ from the procedural goal-base* Γ. It says that if there is a plan σ in Γ, and executing σ possibly along with other actions is no longer a (realistic) p-goal, then σ should be dropped from Γ. This is required when the occurrence of an exogenous action a forces the agent to drop a plan from her goal hierarchy in $\mathcal{D}$ by making it impossible to execute or rendering it inconsistent with her higher priority (realistic) p-goals. Recall that my theory of prioritized goals for committed agents automatically drops such plans from the agent's goal-hierarchy specified by $\mathcal{D}$ in $do(a, s)$, since by Corollary 6.3.17 and Propositions 6.3.1 and 6.3.16, for any executable situation, an SR-APL agent's p-goals are both individually consistent and collectively consistent with each other and with what she knows.

Finally, I have a rule A_{rep} for *repairing an agent's plans in case she gets stuck*, i.e. when for all plans σ in Γ, the agent has the (realistic) p-goal that $\mathrm{DoAL}(\sigma)$ at some level n in the goal hierarchy specified by theory $\mathcal{D}$, and thus all of these $\mathrm{DoAL}(\sigma)$ goals are still individually executable and collectively consistent, but together they are not concurrently executable *without some non-σ actions* in the sense that $\Gamma^{||}$ has no plan-level transition in s. This could happen as a result of an exogenous action or as a side effect of my weak consistency check in rule A_{sel}, as discussed below. The A_{rep} rule says that if:

(a) the agent knows that $\Gamma^{||}$ does not have a plan-level transition in s (which ensures that A_{step} can't be applied),

(b) she also knows that $\Gamma^{||}$ is not considered to be completed in s,

(c) for each plan σ in Γ:

- the agent currently has the (realistic) p-goal at some level that $\mathrm{DoAL}(\sigma)$ (which ensures that A_{clean} can't be applied), and
- $\mathrm{DoAL}(\sigma)$ has been handled (and thus A_{sel} is not applicable),

(d) there is a sequence of actions $\vec{a}$ such that the agent of these actions is the agent herself, and $\vec{a}$ repairs Γ in the sense that the agent knows that there is a plan-level transition of $\Gamma^{||}$ after $\vec{a}$ has been executed in s, and

(e) it is possible to adopt Do($\vec{a}$) at the lowest priority level in situation s, i.e. at level m, where $NPGoals(m, s)$,

then in an attempt to repair Γ, the agent may adopt $\vec{a}$ at the lowest priority level m.

Why do we need this rule? One reason is because the agent could get stuck due to the occurrence of an exogenous action e, e.g. when e makes the preconditions of a plan σ in Γ false. Note that, DoAL(σ) might still be known to be executable after the occurrence of e, e.g. if there is a known to be executable action r (implicitly represented by the additional actions in the DoAL construct) that can be used to restore the preconditions of σ. In such cases, DoAL(σ) may be retained in the goal hierarchy specified by theory $\mathcal{D}$ after e occurs, and since this does not make the triggering condition of the A$_{clean}$ rule true, so may σ in Γ.

Another reason repair may be needed is that I use a weak consistency check when adopting plans via A$_{sel}$ and executing actions via A$_{step}$. While adopting new plans, the SR-APL semantics does not require the agent to ensure that all possible interleavings of this plan and her already adopted plans be executable. In fact, it does not even require that there be at least one such executable interleaving. Rather, it ensures that there must be at least one possible interleaving of her new and existing plans, *possibly along with additional known to be executable actions* (encoded by the DoAL construct, as depicted in the example below). Similarly, while executing plans from the procedural goal-base Γ, the SR-APL semantics only ensures that the agent does not perform an action that makes one of her goals impossible to bring about. But this action can indeed come from an interleaving that is not further executable without executing additional actions, and thus the agent might get stuck and need to add extra actions using the repair rule to continue from there.

Let me give an example to clarify this. Assume a domain with actions a, b, and r, all of which are initially known to be possible. The execution of b makes the preconditions of a false, while that of r restores them. Our agent has two adopted plans in the goal hierarchy in theory $\mathcal{D}$, DoAL(a) and DoAL(b), and $\Gamma = [a, b]$. Note that an SR-APL agent could end up having such a goal hierarchy (through the A$_{sel}$ rule), since as discussed above, although $b; a$ is not a valid execution of $\Gamma^{\|}$ (since the execution of b breaks the preconditions of a), both $a; b$ and $b; r; a$ are indeed valid executions of DoAL(a) and DoAL(b). Now, since I only do weak consistency checking, my semantics allows the agent to pick a "wrong" choice of plan interleaving, e.g. to perform b as the first action.[60] That is, to execute b using the A$_{step}$ transition rule, we only need to ensure that b has a plan-level transition in s and that this transition is consistent with the agent's goals in $\mathcal{D}$, i.e. with DoAL(a) and DoAL(b), both of which hold. After the execution of b, the agent will get stuck, as there is no action in the remainder of Γ (i.e.

[60]Note that this however does not mean that A$_{step}$ allows the agent to perform an action that makes one of her goals impossible, e.g. to execute b when such a repair action r is not available.

in $[a]$) that she can perform. To deal with this, I include the repair rule that makes the agent plan for and commit to a sequence of actions that can be used to repair Γ, which for my example is r.

Note that, I could have avoided the need for repairing plans in this case by strengthening the conditions of the A$_{step}$ rule to do a strong consistency check by expanding all subgoals in Γ. However, this requires modeling the plan selection/goal decomposition process as part of the consistency check, which I leave for future work. I could have also relied on plan failure recovery techniques [247]. Finally, note that my repair rule does a form of conformant planning; more sophisticated forms of planning such as synthesizing conditional plans that include sensing actions could also be performed.

When the agent has complete information, there must be a repair plan available to the agent (whose actions can be performed by the agent herself) if her goals are consistent. In my framework, since the successor-state axiom for G drops all inconsistent goals/plans, the agent's p-goals are always consistent, and thus if complete information is assumed, it is always possible to repair the remaining plans. Consider another example: assume that our agent has $\mathrm{DoAL}(a)$ and $\mathrm{DoAL}(b)$ as her (realistic) p-goals and $\Gamma = [a, b]$. In addition, assume that she has the c-goal not to execute an action from $\Gamma^{||}$ (i.e. $\mathrm{CGoal}(\neg\exists s', \Gamma'.\ \langle\Gamma^{||}, now\rangle \rightarrow \langle\Gamma', s'\rangle, s)$); then it must be the case that she does not have the c-goal not to execute $\Gamma^{||}$ along with other actions (e.g. some repair action), i.e.:

$$\neg\text{CGoal}(\neg\exists s'.\ \text{DoAL}(a\|b, now, s'), s).$$

Otherwise, one of DoAL(a) or DoAL(b) would have been dropped by the successor-state axiom for G as by Propositions 6.3.1 and 6.3.16, for all executable situations, an agent's p-goals are always consistent with each other. Thus the agent thinks/considers it possible that there exist a plan that can repair Γ. If the agent has complete information, then she must know of such a plan $\vec{a}$. Also, since by Definition 7.3.2, the agent of the "other actions" in the DoAL construct is the agent herself, this means that she is also the agent of $\vec{a}$. If on the other hand the agent has only incomplete information, then a repair plan may need to perform sensing actions and branch on the results. Again, I leave this kind of conditional planning for future work.

Also, note that this rule allows the agent to procrastinate in the sense that in addition to the plan that actually repairs Γ, she is allowed to adopt and execute actions that are unnecessary. This could be avoided by constraining the repair plan $\vec{a}$, e.g. by requiring it to be the shortest or the least costly plan etc. I leave this for future work.

Finally, note that while SR-APL agents rely on a user-specified plan library Π, they can achieve a goal even if such plans are not specified. Indeed the A$_{rep}$ rule can be used as a first principles planner for goals that can be achieved using sequential conformant plans. Thus, given a goal $\Diamond\Phi$, all the programmer needs to do to trigger the planner is to add a plan of the form $(\Diamond\Phi : true \leftarrow \Phi?)$ to the plan library Π. Since the program $\Phi?$ is neither executable nor final, it will eventually trigger the A_{rep} rule, which will make the agent adopt a sequence of actions to achieve Φ.

In my operational semantics, I want to ensure that the procedural goals in Γ are consistent with those in the theory $\mathcal{D}$ before expansion of a subgoal/execution of an action occurs; so I assume that the A_{clean} rule has higher priority than A_{sel} and A_{step}. We can do this by adding appropriate preconditions to the antecedent of the latter, which I leave out for brevity.

To summarize, SR-APL is based on the committed agent theory presented in Chapter 6, and thus it inherits many desirable properties therein. In particular, in SR-APL I formalize both declarative goals and plans uniformly in the same goal hierarchy specified by theory $\mathcal{D}$. I maintain the consistency of adopted declarative and procedural goals by ensuring that there exist a path considered possible by the agent over which all of her adopted declarative goals hold and that encodes the concurrent execution of all of her adopted plans, possibly along with other actions, i.e. $\neg\mathrm{CGoal}(\neg\exists s'.\ \mathrm{DoAL}(\Gamma^{\|}, now, s'), s)$. Whenever the agent's goals/plans become inconsistent due to some external interference, the successor-state axiom for G in $\mathcal{D}$ drops some of the adopted goals/plans, respecting their priority, and consistency of the goal-base is automatically restored. Moreover, in SR-APL I also have a procedural goal-base Γ containing the adopted plans in $\mathcal{D}$, whose sole purpose is to ensure that the agent does not procrastinate w.r.t. her adopted plans. The set of transition rules of SR-APL allows an SR-APL agent to select, adopt, and execute plans from the plan library and thus serves as SR-APL's practical reasoning component. While adopting plans and executing actions, I use a weak consistency check, which avoids checking if the plans in the procedural goal-base Γ are concurrently executable (without additional actions) while ensuring consistency. SR-APL also includes a repair rule that can be used to repair plans if the agent gets stuck (1) due to external interferences, (2) as a result of my weak consistency check, or (3) due to the existence of an adopted declarative goal for which there is no plan specified in the plan library.

7.3.3 Execution Traces

Let me now define some useful notions of program execution in SR-APL.

Definition 7.3.8. *A* labeled execution trace $\mathcal{T}$ relative to a theory $\mathcal{D}$ *is a (possibly infinite) sequence of configurations* $\langle\Gamma_0, s_0\rangle \overset{l_0}{\Rightarrow} \langle\Gamma_1, s_1\rangle \overset{l_1}{\Rightarrow} \langle\Gamma_2, s_2\rangle \overset{l_2}{\Rightarrow} \langle\Gamma_3, s_3\rangle \overset{l_3}{\Rightarrow} \cdots$, *such that* $\langle\Gamma_0, s_0\rangle = \langle[\;], S_0\rangle$ *is the initial configuration, and for all configurations* $\langle\Gamma_i, s_i\rangle$ *in* $\mathcal{T}$*, the agent level transition rule* l_i *can be used to obtain* $\langle\Gamma_{i+1}, s_{i+1}\rangle$.

Here l_i is one of A_{sel}, A_{step}, A_{exo}, A_{clean}, and A_{rep} in general, and in the absence of exogenous actions, l_i can be one of A_{sel}, A_{step}, A_{clean}, and A_{rep}. I will sometime suppress these labels.

Definition 7.3.9. *A* complete trace $\mathcal{T}$ relative to a theory $\mathcal{D}$ *is a finite labeled execution trace relative to* $\mathcal{D}$*,* $\langle \Gamma_0, s_0 \rangle \overset{l_0}{\Rightarrow} \cdots \overset{l_{n-1}}{\Rightarrow} \langle \Gamma_n, s_n \rangle$*, such that* $\langle \Gamma_n, s_n \rangle$ *does not have an agent level transition, i.e.* $\langle \Gamma_n, s_n \rangle \not\Rightarrow$*.*

Returning to our discussion on rationality properties, note that when arbitrary exogenous actions can occur, even the best laid plans can fail. So here I only consider the case where exogenous actions are absent. I model this using the following axiom, which I call NoExo:

Axiom 7.3.10 (NoExo)**.**

$$\forall a. \neg \text{Exo}(a).$$

I could have considered exogenous actions, but in that case I would have to complicate the framework further, e.g. by assuming a fair environment that gives a chance to the agent to perform actions. Moreover, it is not obvious what rational behavior means in such contexts. I leave this for future work.

For our blocks world example, I can show that in the absence of external interferences, my SR-APL agent Agt_{BW} for domain $\mathcal{D}_{BW}$ will not adopt inconsistent plans as seen in Section 7.2 (as rule A$_{sel}$ ensures that all adopted plans can be executed concurrently) and will in fact achieve all her goals, i.e. have a green and non-yellow tower and a blue and non-red tower built. Thus, we have that:

Proposition 7.3.11.

(a). *There exists a complete trace $\mathcal{T}$ relative to $\mathcal{D}_{BW} \cup \{\text{NoExo}\}$ for our agent $\langle \mathcal{D}_{BW} \cup \{\text{NoExo}\}, \Pi_{BW} \rangle$.*

(b). *For all such complete traces $\mathcal{T} = \langle \Gamma_0, s_0 \rangle \Rightarrow \langle \Gamma_1, s_1 \rangle \Rightarrow \cdots \Rightarrow \langle \Gamma_n, s_n \rangle$, I have:* $\mathcal{D}_{BW} \cup \{\text{NoExo}\} \models \text{Final}(\Gamma_n^{||}, s_n) \wedge \text{Tower}_Y^G(s_n) \wedge \text{Tower}_R^B(s_n)$.

(c). *There are no infinite traces relative to $\mathcal{D}_{BW} \cup \{\text{NoExo}\}$.*

Proof Sketch. (a). This can be proven by constructing a trace and proving that it is indeed a valid trace of our blocks world agent $\langle \mathcal{D}_{BW} \cup \{\text{NoExo}\}, \Pi_{BW} \rangle$. The trace $\mathcal{T}$

is as follows:

$$\langle [\ ], S_0 \rangle \overset{A_{sel}}{\Rightarrow}$$

$$\langle [stack(B_G, B_R)], do(A_0, S_0) \rangle \overset{A_{sel}}{\Rightarrow}$$

$$\langle [stack(B_G, B_R), stack(B_B, B_Y)], do(A_1, S_1) \rangle \overset{A_{step}}{\Rightarrow}$$

$$\langle [stack(B_B, B_Y)], do(A_2, S_2) \rangle \overset{A_{step}}{\Rightarrow}$$

$$\langle [\ ], do(A_3, S_3) \rangle,$$

where: $A_0 = adoptRelTo(\text{DoAL}(stack(B_G, B_R)), \Diamond\text{Tower}^G_Y),$

$A_1 = adoptRelTo(\text{DoAL}(stack(B_B, B_Y)), \Diamond\text{Tower}^B_R),$

$A_2 = stack(B_G, B_R),$ and

$A_3 = stack(B_B, B_Y),$

and $S_1 = do(A_0, S_0), S_2 = do(A_1, S_1)$, etc. The rest of the proof concern with showing that $\mathcal{T}$ is indeed a proper trace, i.e. that each of these agent level transitions is possible in the corresponding situation. □

(b). This can be proven by constructing an execution tree for our blocks world agent $\langle \mathcal{D}_{BW} \cup \{\text{NoExo}\}, \Pi_{BW} \rangle$. It can be shown that there are only six execution paths/traces in the tree, all of which involve adopting (via the A$_{sel}$ rule) the two plans/actions $stack(B_G, B_R)$ and $stack(B_B, B_Y)$ in response to the two achievement goals in the goal hierarchy, and executing these plans (via A$_{step}$), since executing these two plans is the only way to achieve Agt_{BW}'s goals. These different traces reflect the order in which each goal is handled and each adopted plan is executed. Any other execution paths are ruled out since the conditions in the antecedent of any other candidate rule is false in the corresponding situation. For example, it can be shown that a trace where the agent tries to execute a plan before adopting it, or one where the agent tries to apply the A$_{exo}$, the A$_{clean}$, or the A$_{rep}$ rule in any situation is not a valid trace since the corresponding transition rules are not applicable given the situation.

Clearly in all these six cases, the agent's goals of building the two towers are achieved, since all of them involve executing the (executable) $stack(B_B, B_Y)$ and the $stack(B_G, B_R)$ actions in an executable situation. □

(c). Follows from the discussion above. □

Thus when exogenous actions cannot occur, any execution of our SR-APL blocks world agent Agt_{BW} terminates and achieves all her goals.

A similar result can be shown for our 3 blocks tower example:

Proposition 7.3.12.

(a). *There exists a complete trace $\mathcal{T}$ relative to $\mathcal{D}^{3T}_{BW} \cup \{\text{NoExo}\}$ for our agent $\langle \mathcal{D}^{3T}_{BW} \cup \{\text{NoExo}\}, \Pi^{3T}_{BW} \rangle$.*

(b). *For all such complete traces $\mathcal{T} = \langle \Gamma_0, s_0 \rangle \Rightarrow \langle \Gamma_1, s_1 \rangle \Rightarrow \cdots \Rightarrow \langle \Gamma_n, s_n \rangle$, I have:*

$$\mathcal{D}^{3T}_{BW} \cup \{\text{NoExo}\} \models \text{Final}(\Gamma^{\|}_n, s_n) \wedge \text{3Tower}(s_n).$$

(c). *There are no infinite traces relative to* $\mathcal{D}^{3T}_{BW} \cup \{\text{NoExo}\}$.

Proof Sketch. (a). This can be proven by constructing a trace and proving that it is indeed a valid trace of our blocks world agent $\langle \mathcal{D}^{3T}_{BW} \cup \{\text{NoExo}\}, \Pi^{3T}_{BW} \rangle$. The trace $\mathcal{T}$ is as follows:

$$\begin{aligned}
&\langle [\ ], S_0 \rangle \overset{A_{sel}}{\Rightarrow} \\
&\langle [\sigma_1], do(A_0, S_0) \rangle \overset{A_{step}}{\Rightarrow} \\
&\langle [\sigma_2], do(A_1, S_1) \rangle \overset{A_{sel}}{\Rightarrow} \\
&\langle [\sigma_2, stack(B_G, B_R)], do(A_2, S_2) \rangle \overset{A_{step}}{\Rightarrow} \\
&\langle [\sigma_2], do(A_3, S_3) \rangle \overset{A_{step}}{\Rightarrow} \\
&\langle [\ ], do(A_4, S_4) \rangle,
\end{aligned}$$

where: $\sigma_1 \doteq adoptRelTo(\Diamond\text{Tower}^{G}_{\bar{Y}}, \text{DoAL}(\sigma_2)); \sigma_2,$

$\sigma_2 \doteq \text{Tower}^{G}_{\bar{Y}}?; stack(B_Y, B_G),$ and

$$A_0 = adoptRelTo(\text{DoAL}(\sigma_1), \Diamond\text{3Tower}),$$

$$A_1 = adoptRelTo(\Diamond\text{Tower}_Y^G, \text{DoAL}(\sigma_2)),$$

$$A_2 = adoptRelTo(\text{DoAL}(stack(B_G, B_R)), \Diamond\text{Tower}_Y^G),$$

$$A_3 = stack(B_G, B_R),$$

$$A_4 = stack(B_Y, B_G).$$

and $S_1 = do(A_0, S_0), S_2 = do(A_1, S_1)$, etc. The rest of the proof concern with showing that $\mathcal{T}$ is indeed a proper trace, i.e. that each of these agent level transitions is possible in the corresponding situation. $\square$

(b). Again, this can be proven by constructing an execution tree for our blocks world agent $\langle \mathcal{D}_{BW}^{3T} \cup \{\text{NoExo}\}, \Pi_{BW}^{3T} \rangle$. It can be shown that there are only two execution paths/traces in the tree; one is given in (a), while the other trace involves fulfilling the subgoal $\Diamond\text{Tower}_Y^G$ using a different substitution for its plan, namely $stack(B_G, B_B)$. Note that although at some point in each trace, the agent will have two plans in the procedural goal-base, the adopted plans must be executed in a particular order in both traces as one of them (i.e. σ_2) cannot be executed before the execution of the other (in this case, either $stack(B_G, B_R)$ or $stack(B_G, B_B)$) since it must wait for the parent goal of the latter to be fulfilled. Any other execution paths are ruled out since the conditions in the antecedent of any other candidate rule is false in the corresponding situation. For example, it can be shown that a trace where the agent tries to execute a plan before adopting it, or one where the agent tries to apply the A$_{exo}$, the A$_{clean}$, or the A$_{rep}$ rule in any situation is not a valid trace since the corresponding transition rules are not applicable given the situation.

Clearly in both cases, the agent's goals of building a 3 tower is achieved, since both involve executing the (executable) $stack(B_G, b)$ (where b is either B_R or B_B) and the $stack(B_Y, B_G)$ actions in an executable situation. □

(c). Follows from the discussion above. □

7.4 Rationality of SR-APL Agents

Let us now return to the general case. I can show that several rationality properties hold for arbitrary SR-APL agents (again I only consider the cases where exogenous actions do not occur). First of all, for all domains $\mathcal{D}$ that are part of an SR-APL agent, in every executable situation the agent's knowledge and c-goals/intentions as specified by $\mathcal{D}$ must be consistent:[61]

Proposition 7.4.1 (Consistency of Knowledge and CGoals)**.**

$$\mathcal{D} \models \forall s.\ \text{Executable}(s) \supset \neg\text{Know}(\text{False}, s) \wedge \neg\text{CGoal}(\text{False}, s).$$

[61]This follows independently from the underlying agent theory $\mathcal{D}^{SG}_{CAgt}$ as discussed/shown in Chapter 3 and 6.

Proof. $\neg$Know(False, s) follows from the facts that I am using a possible worlds semantics for knowledge, that K is initially constrained to be reflexive (and thus serial), and that K continues to be reflexive after any executable sequence of actions since this is preserved by the successor-state axiom for K (as discussed in Chapter 3). The proof for $\neg$CGoal(False, s) on the other hand is as that of Proposition 6.3.1. $\square$

Before proceeding further, let me prove a useful lemma:

Lemma 7.4.2 (Situations on a Trace are Executable).
If $\mathcal{T} = \langle \Gamma_0, s_0 \rangle \overset{l_0}{\Rightarrow} \langle \Gamma_1, s_1 \rangle \overset{l_1}{\Rightarrow} \cdots$ *is a (possibly infinite) trace of an SR-APL agent relative to a theory* $\mathcal{D}$*, then for all* i *such that* $i \geq 0$*, we have that* $\mathcal{D} \models$ Executable(s_i).

Proof. (By induction in s) For the base case, by Axiom 3.2.2 and Lemma 3.5.17, it follows that Executable(S_0). For the inductive step, fix S_i and assume that Executable(S_i). I need to show that the application of any agent-level transition rule that changes the situation to $do(A, S_i)$ for some action A preserves the executability of the updated situation, i.e. that Executable$(do(A, S_i))$. By the inductive hypothesis and Definition 3.3.1, this is the case if Poss(A, S_i) holds. Note that there are four such transition rules (that changes the situation), namely all but rule A$_{clean}$. In all these cases, the antecedent of the rule ensures that Poss(A, S_i) (for A$_{step}$, this follows from the antecedent that $\mathcal{D} \models$ Know$(\langle \sigma, now \rangle \rightarrow \langle \sigma', do(A, now) \rangle, S_i)$, Axiom 3.6.2, the reflexivity of K, i.e.

Axiom 3.4.2, and by induction on the structure of the program σ). It thus follows that Executable$(do(A, S_i))$. $\square$

Secondly, I can show that the procedural goals in Γ and the declarative and procedural goals in the theory $\mathcal{D} \cup \{\text{NoExo}\}$ remain consistent. Let's first define the following notion of consistency:

Definition 7.4.3 (Consistency of goals in Γ and $\mathcal{D}$ in situation s)**.**

The procedural goals in Γ *are consistent with those in the theory* $\mathcal{D}$ *in situation* s *in a configuration* $\langle\Gamma, s\rangle$ *if and only if for all* σ *such that* Member(σ, Γ)*, we have that* $\mathcal{D} \models \text{PrimCGoal}(\text{DoAL}(\sigma), s)$.

Note that the above definition allows additional declarative goals in theory $\mathcal{D}$ in situation s that are not in Γ; however, this is not a problem since it is possible that these goals have not yet been handled in s.

Also, let's define $\mathcal{D}_{\overline{Exo}}$ as follows:

Definition 7.4.4.

$$\mathcal{D}_{\overline{Exo}} \overset{\text{def}}{=} \mathcal{D} \cup \{\text{NoExo}\}.$$

Furthermore, for this I will also need domains that do not involve actions that are knowledge-producing for the agent agt under consideration. I model this using the following axiom which I call NoKnowProdAct:

Axiom 7.4.5 (NoKnowProdAct)**.**

$$\forall a.\ \neg\text{KnowProdAct}(agt, a).$$

Finally, let's define $\mathcal{D}_{\overline{Exo,KPA}}$ as follows:

Definition 7.4.6.

$$\mathcal{D}_{\overline{Exo,KPA}} \overset{\text{def}}{=} \mathcal{D}_{\overline{Exo}} \cup \{\text{NoKnowProdAct}\}.$$

Then we have that:

Proposition 7.4.7 (Consistency of Γ and $\mathcal{D}_{\overline{Exo,KPA}}$)**.**

If $\mathcal{T} = \langle\Gamma_0, s_0\rangle \overset{l_0}{\Rightarrow} \langle\Gamma_1, s_1\rangle \overset{l_1}{\Rightarrow} \cdots$ is a (possibly infinite) trace of an SR-APL agent relative to a theory $\mathcal{D}_{\overline{Exo,KPA}}$ such that for all A_{step} *transitions $\langle\Gamma_{j-1}, s_{j-1}\rangle \overset{A_{step}}{\Rightarrow}$ $\langle\Gamma_j, do(a_{j-1}, s_{j-1})\rangle$ on $\mathcal{T}$ and σ_1, σ_2, if $\sigma_1 \in \Gamma_{j-1}$ and $\sigma_2 \in \Gamma_{j-1}$ and $\sigma_1 \neq \sigma_2$, then $\mathcal{D}_{\overline{Exo,KPA}} \models$ $\text{Know}(\exists\sigma_1'.\ \langle\sigma_1, now\rangle \rightarrow \langle\sigma_1', do(a_{j-1}, now)\rangle, s_{j-1}) \equiv \text{Know}(\neg\exists\sigma_2'.$ $\langle\sigma_2, now\rangle \rightarrow \langle\sigma_2', do(a_{j-1}, now\rangle, s_{j-1})$:*

then for all i such that $i \geq 0$, we have that $s_{i+1} = do(a, s_i)$ for some a, and for all configurations $\langle\Gamma_i, s_i\rangle$, the procedural goals in Γ_i are consistent with those in the theory $\mathcal{D}_{\overline{Exo,KPA}}$ in s_i.

Proof. (By induction on n, where n is the length of a partial trace of $\mathcal{T}$) Fix an arbitrary partial trace of $\mathcal{T}$ of length n. First consider the base case where $n = 1$, i.e. where there is only one configuration in the partial trace. Note that, it follows from Definition 7.3.8 that all traces of length 1 contain only the initial configuration $\langle \Gamma_0, S_0 \rangle$, and Γ_0 is of the form [ ]. Since there are no plans in Γ_0 that agent intend to execute (possibly with other actions) next, and there is just one situation S_0 in the partial trace, the consequent thus trivially follows from Definition 7.4.3.

For the inductive step, assume that the proposition holds for all partial traces of length $n = M$. I need to show that this is the case for all partial traces of length $n = M + 1$, i.e. (given the inductive hypothesis) that for configuration $\langle \Gamma_M, S_M \rangle$ in the partial trace, the procedural goals in Γ_M are consistent with those in the theory $\mathcal{D}_{\overline{Exo,KPA}}$ in S_M and that $S_M = do(A, S_{M-1})$ for some A. Note that by the transition rules in Table 7.3, the only way the situation in the $M + 1^{\text{th}}$ configuration (i.e. S_M) remains unchanged is via the application of the A$_{clean}$ rule. But from this, the antecedent of the A$_{clean}$ rule, and the inductive hypothesis, this is impossible as by the inductive hypothesis there are no plans σ in Γ_{M-1} for which the agent does not have a primary c-goal in S_{M-1} (and by Definition 4.2.12, the p-goal in S_{M-1}) that DoAL(σ). Thus, in the absence of exogenous actions, S_M is of the form $do(A, S_{M-1})$ for some action A; also, only three of the agent-level transition rules can be applied, namely A$_{sel}$, A$_{step}$, and A$_{rep}$. I will now show that the consistency between Γ_M and $\mathcal{D}_{\overline{Exo,KPA}}$ is preserved for the application of all these rules.

The Case for A$_{sel}$: The application of this rule changes the situation to $S_M = do(adoptRelTo(\text{DoAL}(\sigma), \Diamond\Phi), S_{M-1})$ for some σ and Φ, and also adds the plan σ to Γ_M. Since by Lemma 7.4.2, S_M is an executable situation, by Definition 3.3.1, Axiom 6.2.4, and Proposition 6.3.31, the newly adopted goal DoAL(σ) is indeed a primary c-goal at some level in S_M. Thus I just need to show that the goals corresponding to the previously adopted plans in Γ_M are retained in $\mathcal{D}_{\overline{Exo,KPA}}$ in S_M, i.e. those that are above or below the adopted level n, where AdoptedLevel$(\Diamond\Phi, n, S_M)$ (note that I do not need to consider level n itself, since by Axiom 6.2.6 and Definition 6.2.10, any previously adopted plan in Γ_M that is a p-goal at level n in S_M is also a p-goal at level $n-1$ in S_M). To this end, note that since by the antecedent of the A$_{sel}$ rule, the $adoptRelTo(\text{DoAL}(\sigma), \Diamond\Phi)$ action is possible in S_{M-1}, it follows from Axiom 6.2.4 that the agent does not intend in S_{M-1} not to execute DoAL(σ) next. Thus there is a path P_1 starting with some situation S_{P_1} that is in $G_\cap$ in S_{M-1} and over which the next action performed is the $adoptRelTo(\text{DoAL}(\sigma), \Diamond\Phi)$ action, and DoAL(σ) holds over P_2, which is the suffix of P_1 that starts with $S_{P_2} = do(adoptRelTo(\text{DoAL}(\sigma), \Diamond\Phi), S_{P_1})$:

$$\begin{aligned} &G_\cap(P_1, S_{M-1}) \land \text{Starts}(P_1, S_{P_1}) \land S_{P_2} = do(adoptRelTo(\text{DoAL}(\sigma), \Diamond\Phi), S_{P_1}) \\ &\land \text{Suffix}(P_2, P_1, S_{P_2}) \land \exists s.\ \text{OnPath}(P_2, s) \land \text{DoAL}(\sigma, S_{P_2}, s). \end{aligned} \tag{7.1}$$

Now assume that there is a σ^* in Γ_{M-1} and DoAL(σ^*) is a primary c-goal (and thus by definition 4.2.12, a p-goal) at some level H that has higher priority than the adopted level n (i.e. $H < n$) in S_{M-1}, i.e.:

$$\sigma^* \in \Gamma_{M-1} \text{ and } \mathcal{D} \models \text{PrimCGoal}(\text{DoAL}(\sigma^*), H, S_{M-1}) \wedge H < n. \qquad \text{(asm-1)}$$

I will first prove that the progression of $\text{DoAL}(\sigma^*)$ after the $adoptRelTo(\text{DoAL}(\sigma), \Diamond\Phi)$ action happens in S_{M-1}, i.e. $\text{DoAL}(\sigma^*)$ itself, is still a p-goal (and by Lemma 7.4.2 and Corollary 6.3.20, a primary c-goal) at level H after this action occurs. Note that, by (7.1) and Proposition 6.3.14(b), P_1 must also be G-accessible at H in S_{M-1}:

$$G(P_1, H, S_{M-1}). \qquad (7.2)$$

Since P_1 is a path, by Corollary 3.5.41 and Lemma 3.5.29 it follows that the $adoptRel-To(\text{DoAL}(\sigma), \Diamond\Phi)$ action is executable in S_{P_1} :

$$\text{Poss}(adoptRelTo(\text{DoAL}(\sigma), \Diamond\Phi), S_{P_1}). \qquad (7.3)$$

Again, by (7.1) and Lemma 6.3.8, it follows that:

$$K(S_{P_1}, S_{M-1}). \qquad (7.4)$$

Now, from (7.4), (7.3), Axiom 3.4.10, and the fact that the $adoptRelTo(\text{DoAL}(\sigma), \Diamond\Phi)$ action is not a knowledge-producing action, it follows that:

$$K(do(adoptRelTo(\text{DoAL}(\sigma), \Diamond\Phi), S_{P_1}), S_M).$$

Then, after the $adoptRelTo(\text{DoAL}(\sigma), \Diamond\Phi)$ action happens, we can see that by this, (7.1), Axiom 6.2.6, and Definitions 6.2.10, 6.2.8, and 4.3.4, P_2 is retained in the G-relation at H in S_M :

$$G(P_2, H, S_M). \tag{7.5}$$

Also, since by assumption (asm-1) the agent has the p-goal that DoAL(σ^*) at level H in S_{M-1}, there are no paths that are G-accessible at H in S_{M-1} over which DoAL(σ^*) does not hold, and thus by the SSA for G (i.e. Axiom 6.2.6, and Definitions 6.2.10, 6.2.8, and 4.3.4), none in S_M either (over which the progression of DoAL(σ^*) after $adoptRelTo(\text{DoAL}(\sigma), \Diamond\Phi)$ happens in S_{M-1}, i.e. DoAL(σ^*) itself, does not hold). From this, (7.5), and Definition 4.2.1, it follows that the agent has the p-goal (and by Lemma 7.4.2 and Corollary 6.3.20, the primary c-goal) at H in $do(adoptRelTo($ $\text{DoAL}(\sigma), \Diamond\Phi), S_{M-1})$ that $\text{DoAL}(\sigma^*)$. Thus given assumption (asm-1) (i.e. that $H < n$), the agent retains the plan DoAL(σ^*) at H in S_M.

Similarly, it can be shown that if there is a σ' in Γ_{M-1} and DoAL(σ') is a primary c-goal at some level L that has lower priority than the adopted level n (i.e. $L > n$) in S_{M-1}, then the agent will retain this primary c-goal at level $L + 1$ in S_M (this is easy to see by Axiom 6.2.6 and Definition 6.2.10). Thus the consistency between Γ_M and $\mathcal{D}_{\overline{Exo,KPA}}$ is maintained w.r.t. the A$_{sel}$ rule.

The Case for A$_{step}$: From the antecedent of the rule, it follows that $\mathcal{D}_{\overline{Exo,KPA}} \models$

Know($\langle\sigma, now\rangle \rightarrow \langle\sigma', do(A, now)\rangle, S_{M-1}$), for some $\sigma \in \Gamma_{M-1}$, σ', and A. From this, Axiom 3.6.2, the reflexivity of K (i.e. Axiom 3.4.2), and by induction on the structure of the program σ, it can be shown that:

$$\text{Poss}(A, S_{M-1}). \tag{7.6}$$

Moreover, it follows from the antecedent that there is a level N such that:

$$\text{PGoal}(\text{DoAL}(\sigma), N, S_{M-1}), \tag{7.7}$$

$$\neg\text{CGoal}(\neg\exists s'.\ \text{Do}(A, now, s'), S_{M-1}). \tag{7.8}$$

Now, note that since A must come from a plan σ in Γ_{M-1}, it must be of a form allowed in the SR-APL plan language in Table 7.1, and thus must be either an $adoptRelTo$ action or a regular action. If A is an $adoptRelTo$ action, then by this fact and (7.6), the proof is similar to the A$_{sel}$ case (with the exception at level N; since now the $adoptRelTo$ action is coming from the plan σ at level N, the progression of DoAL(σ) after A happens in S_{M-1} in this case is DoAL(σ'); but this is accounted for in the updated Γ_M in the consequent of the A$_{step}$ rule, and thus does not cause a problem). Otherwise, A must be a regular action. Now, from (7.8), it follows that there is a path P_1 starting with some situation S_{P_1} that is in the $G_\cap$ relation in S_{M-1} and over which A happens next. Let's call the suffix of P_1 starting in $do(A, S_{P_1})$, P_2. Thus:

$$\text{Starts}(P_1, S_{P_1}) \wedge G_\cap(P_1, S_{M-1}) \wedge \text{Suffix}(P_2, P_1, do(A, S_{P_1})). \tag{7.9}$$

Since P_1 is a path, by Corollary 3.5.41 and Lemma 3.5.29 it follows that A is executable in S_{P_1} :

$$\text{Poss}(A, S_{P_1}). \tag{7.10}$$

By (7.9) and Lemma 6.3.8, it follows that:

$$K(S_{P_1}, S_{M-1}). \tag{7.11}$$

From (7.11), (7.10), Axiom 3.4.10, and the fact that A is not a knowledge producing action, it follows that:

$$K(do(A, S_{P_1}), S_M). \tag{7.12}$$

Again, by (7.9), Lemma 7.4.2, and Proposition 6.3.14(b), it follows that:

$$\forall n.\ G(P_1, n, S_{M-1}). \tag{7.13}$$

Now, assume that σ^* is a plan in Γ_{M-1}. By the inductive hypothesis, there is a level H such that PrimCGoal(DoAL(σ^*)$, H, S_{M-1})$. I will show that the progression of DoAL(σ^*) after A happens in S_{M-1} is retained as a p-goal (and by Lemma 7.4.2 and Corollary 6.3.20, a primary c-goal) at level H. Note that the progression of DoAL(σ^*) at all levels except at level N is simply DoAL(σ^*) since the action A comes from the plan at level N (note that it is not possible for plans at different priority levels to have the same first action A due to the restriction on $\mathcal{T}$ that for all A$_{step}$ transitions

$\langle \Gamma_{j-1}, s_{j-1} \rangle \overset{A_{step}}{\Rightarrow} \langle \Gamma_j, do(a_{j-1}, s_{j-1}) \rangle$ on $\mathcal{T}$ and σ_1, σ_2, if $\sigma_1 \in \Gamma_{j-1}$ and $\sigma_2 \in \Gamma_{j-1}$ and $\sigma_1 \neq \sigma_2$, then $\mathcal{D}_{\overline{Exo,KPA}} \models$ Know$(\exists \sigma_1'.\ \langle \sigma_1, now \rangle \rightarrow \langle \sigma_1', do(a_{j-1}, now) \rangle, s_{j-1}) \equiv$ Know$(\neg \exists \sigma_2'.\ \langle \sigma_2, now \rangle \rightarrow \langle \sigma_2', do(a_{j-1}, now \rangle, s_{j-1}))$. On the other hand, the progression of DoAL(σ^*) at level N, i.e. that of DoAL(σ), is DoAL(σ') as the action A indeed comes from σ. But this does not pose a problem in the consistency between Γ_M and the p-goal at level N in $do(A, S_{M-1})$ since this is accounted for in Γ_M (as we can see from the consequent of rule A$_{step}$).

Now, by Axiom 6.2.6 and Definitions 6.2.7, 6.2.8, and 4.3.4, to show that the progression of DoAL(σ^*) is still a p-goal (and thus by Lemma 7.4.2 and Corollary 6.3.20, a primary c-goal) at level H, it suffices to show that:

- if $H = 0$ then $\exists p.$ Progressed$_{\text{CA}}(p, H, A, S_{M-1})$, and
- if $H > 0$ then $\exists p.\ G_{\cap}(p, H-1, do(A, S_{M-1})) \wedge$ Progressed(p, H, A, S_{M-1}).

Thus the **else** clause in Definition 6.2.7 is never selected and as such no new paths are added to the updated G relation at level H. Then the paths in G at level H in S_M will be those obtained by progressing the paths in G at H in S_{M-1} after A occurs. Since all paths in G at H in S_M satisfy DoAL(σ^*), the progression of DoAL(σ^*) will be a p-goal at level H in S_M by Definition 4.2.1. I will now show that P_2 is indeed a path that satisfies the above conditions for both $H = 0$ and $H > 0$. First consider the case where $H = 0$; by (7.13), (7.12), (7.9), and Definitions 6.2.8 and 4.3.4, it follows that $\text{Progressed}_{\text{CA}}(P_2, 0, A, S_{M-1})$. Thus by Axiom 6.2.6 and Definition 6.2.7, we also have:

$$G(P_2, 0, do(A, S_{M-1})). \tag{7.14}$$

Next consider the case where $H > 0$. I will prove this by induction on level n. Note that by (7.13), (7.9), and Definition 4.3.4, it follows that:

$$\forall n.\ \text{Progressed}(P_2, n, A, S_{M-1}). \tag{7.15}$$

Now, consider the base case, where $n = 1$; By (7.14), Corollary 6.3.5, and Axiom 4.2.7, it follows that $G_\cap(P_2, 0, do(A, S_{M-1}))$. The base case thus follows from this and (7.15). For the inductive step, fix level $n = L$, where $L \geq 1$, and assume that:

$$G_\cap(P_2, L-1, do(A, S_{M-1})) \land \text{Progressed}(P_2, L, A, S_{M-1}). \tag{7.16}$$

I need to show that:

$$G_\cap(P_2, L, do(A, S_{M-1})) \land \text{Progressed}(P_2, L+1, A, S_{M-1}).$$

From (7.16), Axiom 6.2.6, and Definition 6.2.7, it follows that $G(P_2, L, do(A, S_{M-1}))$. Moreover, from this, (7.16), Corollary 6.3.5, and Axiom 4.2.7, it follows that $G_\cap(P_2, L, do(A, S_{M-1}))$. The inductive case then follows from this and (7.15). It thus follows that the progression of all such DoAL(σ^*) plans are p-goals (and a primary c-goal) at level H in the theory $\mathcal{D}_{\overline{Exo,KPA}}$ in $do(A, S_{M-1})$.

The Case for A_{rep}: This case is similar to the A_{sel} case, since in this case, the action is an *adopt* action that the agent knows is possible in S_{M-1}. $\square$

Thus in the absence of exogenous actions, the A_{clean} transition rule is never used since the agent's plans in Γ and in the theory $\mathcal{D}_{\overline{Exo,KPA}}$ always remain consistent, provided that whenever the A_{step} rule is used, the agent does not choose an action that is a common prefix of two different plans.[62] Note that, if the agent is allowed to choose an action from such a plan, then it is possible that the agent's procedural goal-base Γ and theory $\mathcal{D}_{\overline{Exo,KPA}}$ may become "inconsistent" (i.e. out of sync). Consider the following example. Suppose that $\Gamma = \{[a;b],[a;c]\}$ and $\mathcal{D}_{\overline{Exo,KPA}}$ entails that $\mathrm{PrimCGoal}(\mathrm{DoAL}([a;b]),s)$ and $\mathrm{PrimCGoal}(\mathrm{DoAL}([a;c]),s)$, and thus Γ and $\mathcal{D}_{\overline{Exo,KPA}}$ are consistent in s. Now suppose that the agent performs the action a via the A_{step} rule to get $\Gamma' = \{[b],[a;c]\}$. Note that, the progression by a of $\mathrm{DoAL}([a;b])$ is $\mathrm{DoAL}(b \mid [a;b])$, since the action a could have been produced by the outside action part of DoAL. Similarly, the progression by a of $\mathrm{DoAL}([a;c])$ is $\mathrm{DoAL}(c \mid [a;c])$. Then by Definition 7.4.3, to show that the agent's plans in Γ' and the theory $\mathcal{D}_{\overline{Exo,KPA}}$ are consistent in $do(a,s)$, we need to show that $\mathcal{D}_{\overline{Exo,KPA}}$ entails that:

1. $\mathrm{DoAL}(b \mid [a;b])(p) \supset \mathrm{DoAL}(b)(p)$, and

[62]Recall from Table 7.3 that applications of A_{clean} do not change the situation.

2. $\text{DoAL}(c \mid [a;c])(p) \supset \text{DoAL}([a;c])(p)$.

While the former holds, the latter does not (since a path over which c happens and a never happens is one that satisfies $\text{DoAL}(c \mid [a;c])$ but not $\text{DoAL}([a;c])(p)$). The root cause of this problem is that the A_{step} rule treats the members of the procedural goal-base Γ as concurrently running plans and thus progresses only one plan when an action happens, while in the declarative side the theory $\mathcal{D}_{\overline{Exo,KPA}}$ allows actions to be shared by multiple DoAL goals and thus progresses all of them when the action happens. I could have avoided this issue, e.g. by renaming each instance of an action to include its thread number, etc. I leave this for future work.

Also, note that for this proposition, I assume that the actions involved are not knowledge-producing for the agent. When such knowledge-producing actions are allowed, the agent may learn that one of her plans has become inconsistent with her other (higher priority) plans; recall that in the committed agent framework, such inconsistent plans (i.e. DoAL goals in the theory $\mathcal{D}$) are automatically dropped to maintain consistency. In that case, Γ and $\mathcal{D}_{\overline{Exo,KPA}}$ will become "inconsistent". One could modify the A_{sel} rule to ensure that intended plans are consistent in all K-alternatives. I leave this for future work. Finally, I conjecture that given in addition that the trace $\mathcal{T}$ is complete, when exogenous actions or actions that are knowledge-producing for the agent are allowed, if $s_i = s_{i+1}$, then there exists j such that $0 < i < j < n$ (where n is the length of $\mathcal{T}$) and the goals in Γ_j are consistent with those in the theory $\mathcal{D}$ in s_j. That is, whenever there is some procedural goal in Γ_i that is not a goal w.r.t. the theory $\mathcal{D}$, the A_{clean} rule will remove it from Γ_i, and eventually consistency will be restored. Note that, the potential scenario where the agent must clear/drop an infinite set of goals is ruled out since the trace $\mathcal{T}$ is assumed to be complete, and thus by Definition 7.3.9, $\mathcal{T}$ is required to be a finite sequence whose last configuration does not have any agent level transitions. I leave proving this conjecture for future work.

It follows from Proposition 7.4.7 that in all such configurations $\langle\Gamma, s\rangle$, the agent intends to execute the plans in Γ concurrently starting in s, possibly with other actions:

Corollary 7.4.8 ($\Gamma^{\|}$ is Intended)**.**

If $\mathcal{T} = \langle\Gamma_0, s_0\rangle \overset{l_0}{\Rightarrow} \langle\Gamma_1, s_1\rangle \overset{l_1}{\Rightarrow} \cdots$ *is a (possibly infinite) trace of an SR-APL agent relative to a theory* $\mathcal{D}_{\overline{Exo,KPA}}$ *such that for all* A_{step} *transitions* $\langle\Gamma_{j-1}, s_{j-1}\rangle \overset{A_{step}}{\Rightarrow} \langle\Gamma_j, do(a_{j-1}, s_{j-1})\rangle$ *on* $\mathcal{T}$ *and* σ_1, σ_2*, if* $\sigma_1 \in \Gamma_{j-1}$ *and* $\sigma_2 \in \Gamma_{j-1}$ *and* $\sigma_1 \neq \sigma_2$*, then* $\mathcal{D}_{\overline{Exo,KPA}} \models \mathrm{Know}(\exists\sigma_1'.\ \langle\sigma_1, now\rangle \rightarrow \langle\sigma_1', do(a_{j-1}, now)\rangle, s_{j-1}) \equiv \mathrm{Know}(\neg\exists\sigma_2'.\ \langle\sigma_2, now\rangle \rightarrow \langle\sigma_2', do(a_{j-1}, now\rangle, s_{j-1})$*:*

then for all i *such that* $i \geq 0$*, we have that:*

$$\mathcal{D}_{\overline{Exo,KPA}} \models \mathrm{CGoal}(\exists s'.\ \mathrm{DoAL}(\Gamma_i^{\|}, now, s'), s_i).$$

Proof. The proof is straightforward from Proposition 7.4.7 and Definitions 7.3.3, 7.4.3, 7.3.2, and 4.2.12. □

Finally, my agents evolve in a rational way:

Proposition 7.4.9 (Rationality of Actions in a Trace)**.**

If $\mathcal{T} = \langle \Gamma_0, s_0 \rangle \overset{l_0}{\Rightarrow} \langle \Gamma_1, s_1 \rangle \overset{l_1}{\Rightarrow} \cdots$ *is a (possibly infinite) trace of an SR-APL agent relative to a theory* $\mathcal{D}_{\overline{Exo}}$*, then for all* i *such that* $i > 0$ *and for all* a *such that* $s_i = do(a, s_{i-1})$*, we have:*

(a). $\mathcal{D}_{\overline{Exo}} \models \neg\text{CGoal}(\neg\exists s'.\ \text{Do}(a, now, s'), s_{i-1})$.

(b). *If* $l_{i-1} = A_{step}$ *then: there exists* σ, σ', *such that* $\text{Member}(\sigma, \Gamma_{i-1})$,

$$\mathcal{D}_{\overline{Exo}} \models \text{Know}(\langle \sigma, now \rangle \rightarrow \langle \sigma', do(a, now) \rangle, s_{i-1}),\ \textit{and}$$

$$\mathcal{D}_{\overline{Exo}} \models \text{CGoal}(\exists s'.\ \text{DoAL}(\sigma, now, s'), s_{i-1}).$$

(c). $\mathcal{D}_{\overline{Exo}} \models \forall \phi, \psi, n.\ (a = adoptRelTo(\psi, \phi) \vee a = adopt(\psi, n)) \supset$

$$\neg\text{CGoal}(\neg\exists s', p'.\ \text{Starts}(s') \wedge \text{Suffix}(p', do(a, s')) \wedge \psi(p'), s_{i-1}).$$

Proof. Fix action A and situation S_{i-1}.

(a). First, note that since the transition involves an action, in the absence of exogenous actions, only three of the agent-level transition rules can be applied, namely A_{sel}, A_{step}, and A_{rep}. I will thus prove this by showing that the antecedents of all these rules ensure that the agent does not intend not to execute A next. Let's consider A_{sel} rule first. In this case, $A = adoptRelTo(\psi, \phi)$ for some goals ψ and ϕ. Also, from the antecedent of the rule, it follows that $\text{Poss}(A, S_{i-1})$. The consequent follows from this and Axiom

6.2.4 (and Definitions 3.5.16, 3.6.4, and 3.6.3, and Axioms 3.6.2 and 3.6.1). For the A_{step} rule, the consequent trivially follows from the antecedent of this rule. Finally, the proof for the A_{rep} rule case is similar to that for the A_{sel} rule. □

(b). Since $l_{i-1} = \mathrm{A}_{step}$, if follows from the antecedent of the rule that there is a level n and plans σ, σ', such that:

$$\text{Member}(\sigma, \Gamma_{i-1}), \tag{7.17}$$

$$\mathcal{D}_{\overline{Exo}} \models \text{Know}(\langle \sigma, now \rangle \rightarrow \langle \sigma', do(A, now) \rangle, S_{i-1}), \text{ and} \tag{7.18}$$

$$\mathcal{D}_{\overline{Exo}} \models \text{PGoal}(\text{DoAL}(\sigma), n, S_{i-1}). \tag{7.19}$$

From (7.19), Lemma 7.4.2, and Proposition 6.3.15, it follows that $\text{CGoal}(\text{DoAL}(\sigma),$ $S_{i-1})$. The proposition follows from this, (7.17), and (7.18). □

(c). Since exogenous actions are not possible and a is either an $adoptRelTo$ or an *adopt* action, we only need to consider three agent-level transition rules, namely A_{sel}, A_{rep}, and A_{step} (since SR-APL's plan language includes $adoptRelTo$ actions). For the first two type of rules, it follows from the antecedent of the rule that $\text{Poss}(A, S_{i-1})$ holds. Moreover, for the A_{step} rule, by the antecedent, it follows that there is a σ in Γ_{i-1} such that $\mathcal{D}_{\overline{Exo}} \models \text{Know}(\langle \sigma, now \rangle \rightarrow \langle \sigma', do(A, now) \rangle, S_{i-1})$. From this, Axiom 3.6.2, the reflexivity of K, i.e. Axiom 3.4.2, and by induction on the structure of the program σ, it can be shown that $\text{Poss}(A, S_{i-1})$ holds for this case as well. The proposition then follows from this and Axioms 6.2.4 and 6.2.3. □

This states that SR-APL is sound in the sense that any trace produced by the APL semantics is consistent with the agent's chosen goals. To be precise, (a) if an SR-APL agent performs the action a in situation s_{i-1}, then it must be the case that she does not have the intention not to execute a in s_{i-1}. Moreover, (b) if a is performed via A$_{step}$, then a is indeed intended in s_{i-1} in the sense that she has the intention to execute some plan σ possibly along with some other actions next, and she knows that σ can execute a next. Finally, (c) if a is the action of adopting a subgoal ψ w.r.t. a supergoal ϕ or that of adopting a goal ψ at some level n (performed via A$_{sel}$, A$_{rep}$, or A$_{step}$), then the agent does not have the c-goal in s_{i-1} not to bring about ψ next.

These properties thus show that in the absence of exogenous actions, in all possible executions an SR-APL agent behaves rationally in the sense that her mental states (i.e. her knowledge and adopted declarative and procedural goals) always remain consistent and that any action performed by the agent is consistent with her intentions.

7.5 Conclusion and Future Work

Based on the Committed Agent variant of my rich theory of goals, in this chapter I developed a specification of an rational BDI agent programming framework that handles prioritized goals, provides semantics for goal dynamics and goal-subgoal dependencies, and maintains the consistency of adopted declarative and procedural goals. I also showed that an agent specified in this language satisfies some strong rationality properties. While doing this, I addressed some fundamental questions about rational agency. I model an agent's concurrent commitments by incorporating the DoAL construct in her adopted plans, which allows her to be open-ended towards future commitments to plans, while using a procedural goal-base Γ to prevent procrastination. I formalized a weak notion of consistency between goals and plans that does not require the agent to commit to a means to achieve all adopted goals while checking for consistency.

In addition to the APLs discussed in Chapter 2, there has been work that focuses on maintaining consistency of a set of concurrent intentions. For example, Clement et al. [33, 34] argue that agents should be able to reason about abstract HTN plans and their interactions before they are fully refined. They propose a method for deriving summary information (i.e. external preconditions and effects) of abstract plans and discuss how this information can be used to coordinate the interactions of plans at different levels of abstractions. Thangarajah et al. [221] use such summary information to detect and resolve conflicts between goals at run time. Horty and Pollack [104] propose a decision theoretic approach to compute the utility of adopting new (non-hierarchical) plans, given a set of already adopted plans. While some of these approaches can be integrated in APLs (e.g. [221]), they leave out many aspects of rationality (e.g. they do not say what the agent should do if external interference makes two of her intentions permanently incompatible), and do not deal with declarative goals.

In this chapter, I focused on developing an expressive agent programming framework that yields a rational/robust agent without worrying about tractability. Thus my framework is a specification and model of an ideal APL rather than a practical APL. The idea behind this exercise is to bring current BDI agent programming languages a step closer towards rational agent theories. As discussed, there are a few restrictions to this framework. For instance, recall that to show our otherwise clean consistency result, I had to limit to non-knowledge-producing actions; also I assumed that plans do not share actions. I leave figuring out how to resolve these issues for future work. In the future, I would also like to investigate how one can restrict SR-APL to ensure decidability/tractability.

Chapter 8

Conclusions and Future Work

8.1 Conclusions

In this thesis, I have presented two frameworks for specifying prioritized goals of agents. In developing these frameworks, I have made contributions that can be classified into five categories: setting the stage by formalizing infinite paths within the situation calculus, specifying prioritized goals and goal dynamics for optimizing agents, modeling prioritized goals and goal dynamics for committed agents, capturing the dependencies between goals and their subgoals, and exhibiting the applicability of my approach by developing the BDI agent programming language SR-APL.

Formalizing Paths in the Situation Calculus

I extended Reiter's formalization of the situation calculus [178] by incorporating a new sort of infinite paths in this language. I gave a sound and complete axiomatization of paths and proved many desirable properties about them. The utility of adding paths as a new sort is twofold: it allows for first-order quantifications over paths; moreover I can now have arbitrary temporally extended goals, which can be evaluated over these infinite paths. This sets the stage for my frameworks for prioritized goals and subgoals.

Specifying Optimizing Agents

On top of this extended language, which is further enriched with Scherl and Levesque's Knowledge modality [188], I presented a framework for modeling agent' prioritized goals. My framework supports rich temporally extended desires, which are allowed to be mutually contradictory. It specifies how these desires/goals change when actions/events occur, including goal adoption, goal drop, and exogenous actions. Given an arbitrary situation and the prioritized desires of an agent in that situation, I have a method for deriving the consistent set of chosen goals or intentions of the agent for that situation. This method along with the proposed dynamics of desires ensure that agents specified using this framework always optimize their chosen goals. In particular, my agents behave like Bratman's intentional agents that always trigger the filter override mechanism in an attempt to constantly optimize their chosen goals. Thus agents specified using this optimizing agent framework are very idealized.

Among other intuitively desirable properties, I showed that my agents' chosen goals are consistent, realistic, and possible, that they are aware of their goals, that their (achievement) chosen goals persist, and that their goals properly evolve as a result of various actions (including external events). I also modeled an online travel planning example and proved interesting properties of this domain.

Specifying Committed Agents

Optimizing agents are quite idealized and costly. Real-world agents have limited resources. To deal with this, I proposed another framework where agents are strongly committed to their chosen goals. For this, I revised the optimizing agent framework essentially by forcing the agent to drop desires when they are no longer chosen. An agent specified using this framework drops a desire when the desire becomes impossible, or when it becomes inconsistent with other higher priority goals. Moreover, the agent is not allowed to adopt a desire if it is inconsistent with her current chosen goals. Such a framework, although still does not duly formalize Bratman's original notion of intentions — that agents should be strongly committed to their intentions, but must also be able to revise them under certain (rare) conditions, in particular when doing so increase their utility considerably — is applicable for the real-world. In particular, in contrast to the computationally demanding optimizing agent framework, the committed agent framework is more suitable as a foundation for a BDI agent programming language, albeit at the cost of rationality.

Modeling Goal-Subgoal Dependencies

I extended both these frameworks to include subgoals and their dynamics. In particular, I model subgoal change in a way that properly maintains the dependencies between goals and their parent goals. I showed that when a goal is dropped, all of its subgoals (and theirs, etc.) are also dropped. Moreover, when a chosen goal becomes impossible, all of its subgoals (and theirs, etc.) are also dropped from the agent's set of chosen goals. To the best of my knowledge, this is the first and the only account found in the literature that uses a semantic approach to capture this relationship between goals and subgoals.

Applicability of the Proposed Frameworks

In order to demonstrate the feasibility of using my prioritized goal frameworks to specify multiagent systems, I developed a model for a simple rational agent programming language, SR-APL, that combines the committed agent framework, work on BDI agent programming languages with declarative goals, and the situation calculus based Con-Golog agent programming language [51]. I showed that agents specified using this language behaves rationally, particularly when effects of actions are irreversible or when time sensitive goals are involved. SR-APL thus contributes to bridging the gap between rational agent theories and agent programming languages with declarative goals.

8.2 Future Work

The work presented here can be extended in many ways. In my theory of prioritized goals, I have proposed two frameworks that to some extent lies at the two extremes of the "resource-boundedness vs. tractability" spectrum – the optimizing agent framework formalizes ideally rational agents that always reconsider their intentions; on the other hand, the committed agent framework models over-committed agents that never give up their intentions even if opportunities to commit to higher priority goals arise, and thus effectively minimizing their reasoning costs w.r.t. intention reconsideration. Hence it would be interesting to work on a hybrid account of intention reconsideration where the agent is strongly committed to her chosen goals but where she reconsiders some of her prioritized goals under specific conditions.

I would like to work on a more complete set of AGM-like postulates for prioritized goals and their dynamics. These postulates will serve as a specification for the relationship between an agent's knowledge and her prioritized goals, and for the update/revision and contraction of her prioritized goals.

For both optimizing and committed agent framework, I discussed some properties that show when exactly an agent's chosen achievement goals can be expected to persist. It would be nice to generalize these persistence properties and identify the conditions under which arbitrary temporally extended goals persist.

I have a method for dropping a subgoal when its parent goal is dropped or becomes impossible. However my proposed subgoal dynamics do not give up a subgoal when its parent goal is fulfilled unexpectedly. To this end, I would like to modify my framework to handle early achievement of goals, i.e. automatically drop subgoals whose parent goal have been achieved. Note that this seems quite challenging to do so for arbitrary temporally extended goals, as in that case one first needs to identify the satisfaction conditions of such a goal, and then do a case-by-case analysis and handle each type of goals separately. The special case of achievement goals should be solvable with some effort.

In [166], Pirri and Reiter proved a relative consistency property for basic action theories in the situation calculus that states that such a theory is consistent if and only if the initial state axioms are consistent. While I did not prove such a relative consistency property, I conjecture that it should be possible to extend the results in [166] to show that relative consistency holds for instances of my agent theories.

In the future, I would also like to investigate restricted versions of SR-APL that are practical, with an understanding of how they compromise rationality. I think this can be done. For instance if I assume a finite domain, then reasoning with the underlying theory should be decidable. I could adapt techniques from partial order planning such as summary information/causal links to support consistency maintenance [33, 34, 221]. I could also simply find a global linear plan and cache it, using summary information to revise it when necessary. There are some controller synthesis techniques that can deal with temporally extended goals [167, 29].

Also, it would be desirable to study a version of SR-APL where the agent fully expands an abstract plan and checks its executability before adopting it. Finally, while the underlying agent theory supports arbitrary temporally extended goals, in SR-APL I only consider achievement goals. I would like to relax this in the future.

Bibliography

[1] C. E. Alchourrón, P. Gärdenfors, and D. Makinson. On the Logic of Theory Change: Partial Meet Contraction and Revision Functions. *Journal of Symbolic Logic*, 50(2):510–530, 1985.

[2] J. Ambros-Ingerson and S. Steel. Integrated Planning, Execution and Monitoring. In *Proceedings of the Seventh National Conference on Artificial Intelligence (AAAI-88)*, pages 83–88, St. Paul, MN, USA, 1988.

[3] R. Arkin. *Behavior-Based Robotics*. MIT Press, 1998.

[4] J. L. Austin. *How to Do Things with Words*. Oxford University Press, Oxford, England, 1962.

[5] C. Baier and J.-P. Katoen. *Principles of model checking*. MIT Press, 2008.

[6] M. Baldoni, L. Giordano, A. Martelli, and V. Patil. Modeling Agents in a Logic Action Language. In *Proceeding of the Workshop on Rational Agents (FAPR-00)*, London, UK, 2000.

[7] C. Baral and J. Zhao. Non-monotonic Temporal Logics for Goal Specification. In *Proceedings of the 20th International Joint Conference on Artificial Intelligence (IJCAI-07)*, pages 236–242, 2007.

[8] C. Baral and J. Zhao. Non-monotonic Temporal Logics that Facilitate Elaboration Tolerant Revision of Goals. In *Proceedings of the Twenty-Third AAAI Conference on Artificial Intelligence (AAAI-08)*, pages 406–411, 2008.

[9] M. Barbuceanu, T. Gray, and S. Mankovski. Role of Obligations in Multiagent Coordination. *Applied Artificial Intelligence*, 13(1):11–38, 1999.

[10] F. Bellifemine, F. Bergenti, G. Caire, and A. Poggi. JADE – A Java Agent Development Framework. In R. H. Bordini, M. Dastani, J. Dix, and A. El F. Seghrouchni, editors, *Multi-Agent Programming: Languages, Platforms and Applications*, volume 15 of *Multiagent Systems, Artificial Societies, and Simulated Organizations Series*, pages 125–148. Springer-Verlag, 2005.

[11] R. H. Bordini, A. L. C. Bazzan, R. O. Vicari, and V. R. Lesser. AgentSpeak(XL): Efficient Intention Selection in BDI Agents via Decision-Theoretic

Task Scheduling. In *Proceedings of the First International Joint Conference on Autonomous Agents and Multi-Agent Systems (AAMAS-02)*, pages 1294–1302, 2002.

[12] R. H. Bordini, L. Braubach, M. Dastani, A. El F. Seghrouchni, J. J. Gomez-Sanz, J. Leite, G. O'Hare, A. Pokahr, and A. Ricci. A Survey of Programming Languages and Platforms for Multi-Agent Systems. *Informatica*, 30:33–44, 2006.

[13] R. H. Bordini, M. Dastani, J. Dix, and A. El F. Seghrouchni, editors. *Multi-Agent Programming: Languages, Platforms and Applications*, volume 15 of *Multiagent Systems, Artificial Societies, and Simulated Organizations Series*. Springer-Verlag, 2005.

[14] R. H. Bordini, M. Fisher, C. Pardavila, and M. Wooldridge. Model Checking AgentSpeak. In *Proceedings of the Second International Joint Conference on Autonomous Agents and Multi-Agent Systems (AAMAS-03)*, pages 409–416, 2003.

[15] R. H. Bordini, M. Fisher, W. Visser, and M. Wooldridge. Verifiable Multi-Agent Programs. In *Proceedings of the First International Workshop on Program-*

ming Multi-Agent Systems: Languages, Frameworks, Techniques, and Tools (ProMAS-03), pages 72–89, 2003.

[16] R. H. Bordini and J. F. Hübner. Jason : A Java-based Interpreter for an Extended Version of AgentSpeak. http://jason.sourceforge.net, 2006.

[17] R. H. Bordini and A. F. Moreira. Proving BDI Properties of Agent-Oriented Programming Languages – The Asymmetry Thesis Principles in AgentSpeak(L). *Annals of Mathematics and Artificial Intelligence, Special Issue on Computational Logic in Multi-Agent Systems*, 42(1–3):197–226, 2004.

[18] R. H. Bordini, W. Visser, M. Fisher, C. Pardavila, and M. Wooldridge. Model Checking Multi-Agent Programs with CASP. In W. A. Hunt Jr. and F. Somenzi, editors, *Proceedings of the Fifteenth Conference on Computer-Aided Verification (CAV-03)*, volume 2725 of *LNCS*, pages 110–113. Springer-Verlag, 2003.

[19] C. Boutilier. Toward a Logic for Qualitative Decision Theory. In *Proceedings of the Third International Conference on Principles of Knowledge Representation and Reasoning (KR-92)*, pages 75–86, 1992.

[20] M. E. Bratman. *Intentions, Plans, and Practical Reason.* Harvard University Press, Cambridge, MA, USA, 1987.

[21] M. E. Bratman, D. J. Israel, and M. E. Pollack. Plans and Resource-Bounded Practical Reasoning. *Computational Intelligence*, 4:349–355, 1988.

[22] L. Braubach, A. Pokahr, and W. Lamersdorf. Jadex: A Short Overview. In *Proceedings of the Net. Object Days-04 Conference*, pages 195–207, 2004.

[23] L. Braubach, A. Pokahr, W. Lamersdorf, and D. Moldt. Goal Representation for BDI Agent Systems. In R. H. Bordini, M. Dastani, J. Dix, and A. El F. Seghrouchni, editors, *Proceedings of the Second International Workshop on Programming Multiagent Systems, Languages, and Tools (ProMAS-04)*, volume 3346 of *LNAI*, pages 9–20. Springer-Verlag, 2004.

[24] R. A. Brooks. A Robust Layered Control System for a Mobile Robot. *IEEE Journal of Robotics and Automation*, 2(1):14–23, 1986.

[25] R. A. Brooks. Elephants don't Play Chess. In P. Maes, editor, *Designing Autonomous Agents*, pages 3–15. The MIT Press, 1990.

[26] R. A. Brooks. Intelligence without Reason. In *Proceedings of the Twelfth International Joint Conference on Artificial Intelligence (IJCAI-91)*, pages 569–595, Sydney, Australia, 1990.

[27] R. A. Brooks. Intelligence without Representation. *Artificial Intelligence*, 47:139–159, 1991.

[28] D. Calvanese, G. De Giacomo, M. Montali, and F. Patrizi. On First-Order μ-Calculus over Situation Calculus Action Theories. In *Principles of Knowledge Representation and Reasoning: Proceedings of the Fifteenth International Conference (KR-16)*, pages 411–420, Cape Town, South Africa, 2016.

[29] D. Calvanese, G. De Giacomo, and M. Y. Vardi. Reasoning about Actions and Planning in LTL Action Theories. In *Proceedings of the Eighth International Conference on Principles of Knowledge Representation and Reasoning (KR-02)*, pages 593–602, Toulouse, France, 2002.

[30] J. Carmo and A. J. I. Jones. Deontic Logic and Contrary-to-Duties. In *Handbook of Philosophical Logic*, pages 209–285. Kluwer Academic Publishers, 2000.

[31] B. F. Chellas. *Modal Logic: An Introduction*. Cambridge University Press, 1980.

[32] J. Claßen and G. Lakemeyer. A Logic for Non-Terminating Golog Programs. In *Principles of Knowledge Representation and Reasoning: Proceedings of the Eleventh International Conference (KR-08)*, pages 589–599, Sydney, Australia, 2008.

[33] B. J. Clement and E. H. Durfee. Theory for Coordinating Concurrent Hierarchical Planning Agents Using Summary Information. In *Proceedings of the Sixteenth National Conference on Artificial Intelligence (AAAI-99)*, pages 495–502, Orlando, Florida, 1999.

[34] B. J. Clement, E. H. Durfee, and A. C. Barrett. Abstract Reasoning for Planning and Coordination. *J. of Articial Intelligence Research*, 28:453–515, 2007.

[35] P. R. Cohen and H. J. Levesque. Intention is Choice with Commitment. *Artificial Intelligence*, 42(2–3):213–361, 1990.

[36] P. R. Cohen and H. J. Levesque. Persistence, Intention and Commitment. In P. R. Cohen, J. Morgan, and M. E. Pollack, editors, *Intentions in Communication*, pages 33–69. MIT Press, Cambridge, Mass., USA, 1990.

[37] P. R. Cohen and H. J. Levesque. Rational Interaction as the Basis for Communication. In P. R. Cohen, J. Morgan, and M. E. Pollack, editors, *Intentions in Communication*, pages 221–255. MIT Press, Cambridge, Mass., USA, 1990.

[38] P. R. Cohen and H. J. Levesque. Teamwork. *Nous*, 25(4):487–512, 1991.

[39] P. R. Cohen, H. J. Levesque, and I. Smith. On Team Formation. *Contemporary Action Theory, Synthese*, pages 87–114, 1997.

[40] P. R. Cohen and C. R. Perrault. Elements of a Plan Based Theory of Speech Acts. *Cognitive Science*, 3:177–212, 1979.

[41] S. Costantini and A. Tocchio. A Logic Programming Language for Multi-Agent Systems. In S. Flesca, S. Greco, N. Leone, and G. Ianni, editors, *Proceedings of the 8th European Conference on Logics in Artificial Intelligence (JELIA-02)*, volume 2424 of *LNAI*, pages 1–13. Springer-Verlag, 2004.

[42] P. C. da Costa and A. Tettamanzi. Towards a Framework for Goal Revision. In *Proceedings of the Eighteenth Belgium-Netherlands Conference on Artificial Intelligence (BNAIC)*, pages 99–106, Namur, Belgium, 2006.

[43] P. C. da Costa and A. Tettamanzi. A Belief-Desire Framework for Goal Revision. In *Proceedings of the Eleventh International Conference on Knowledge-Based Intelligent Information and Engineering Systems (KES)*, pages 164–171, 2007.

[44] P. C. da Costa, A. Tettamanzi, and L. Amgoud. Goal Revision for a Rational Agent. In *Proceedings of the Seventeenth European Conference on Artificial Intelligence (ECAI)*, pages 747–178, 2006.

[45] A. Dardenne, A. van Lamsweerde, and S. Fickas. Goal-Directed Requirements Acquisition. *Science of Computer Programming*, 20(1–2):3–50, 1993.

[46] A. Darwiche and J. Pearl. On the Logic of Iterated Belief Revision. *Artificial Intelligence*, 89(1–2):1–29, 1997.

[47] M. Dastani. 2APL: A Practical Agent Programming Language. *Autonomous Agents and Multi-Agent Systems*, 16(3):214–248, 2008.

[48] M. Dastani, M. B. van Riemsdijk, F. Dignum, and J.-J. Ch. Meyer. A Programming Language for Cognitive Agents: Goal Directed 3APL. In *Proceedings of the First International Workshop on Programming Multi-Agent Systems (ProMAS-03)*, volume 3067 of *LNAI*, pages 111–130. Springer-Verlag, 2004.

[49] M. Dastani, M. B. van Riemsdijk, and M. Winikoff. Rich Goal Types in Agent Programming. In *Proceedings of the 10th International Conference on Autonomous Agents and Multiagent Systems (AAMAS-11)*, volume 1-3, pages 405–412, 2011.

[50] E. Davis. Knowledge Preconditions for Plans. *Journal of Logic and Computation*, 4(5):721–766, 1994.

[51] G. De Giacomo, Y. Lespérance, and H. J. Levesque. ConGolog, a Concurrent Programming Language Based on the Situation Calculus. *Artificial Intelligence*, 121(1–2):109–169, 2000.

[52] G. De Giacomo, Y. Lespérance, H. J. Levesque, and S. Sardiña. On the Semantics of Deliberation in IndiGolog – from Theory to Implementation. *Annals of Mathematics and Artificial Intelligence*, 41(2–4):259–299, 2004.

[53] G. De Giacomo, Y. Lespérance, and F. Patrizi. Bounded Situation Calculus Action Theories. *Artificial Intelligence*, 237:172–203, 2016.

[54] G. De Giacomo, Y. Lespérance, and A. R. Pearce. Situation Calculus-based Programs for Representing and Reasoning about Game Structures. In *Proceedings of the Twelfth International Conference on Principles of Knowledge Representation and Reasoning (KR-10)*, Torotno, ON, Canada, May 2010. Morgan Kaufmann Publishing.

[55] G. De Giacomo and H. J. Levesque. An Incremental Interpreter for High-Level Programs with Sensing. pages 86–102, 1999.

[56] G. De Giacomo, E. Ternovskaia, and R. Reiter. Non-Terminating Processes in the Situation Calculus. In *Working Notes of Robots, Softbots, Immobots: Theories of Action, Planning and Control: AAAI-97 Workshop*, 1997.

[57] G. Delzanno and M. Martelli. Proofs as Computations in Linear Logic. *Theoretical Computer Science*, 258(1–2):269–297, 2001.

[58] D. C. Dennett. *The Intentional Stance*. The MIT Press, Cambridge, MA, USA, 1987.

[59] M. d'Inverno, D. Kinny, M. Luck, and M. Wooldridge. A Formal Specification of dMARS. In M. P. Singh, A. S. Rao, and M. Wooldridge, editors, *Intelligent Agents IV–Proceedings of the Fourth International Workshop on Agent Theories, Architectures, and Languages (ATAL-97)*, volume 1365 of *LNAI*, 155–176, 1997. Springer-Verlag.

[60] M. d'Inverno and M. Luck. Engineering AgentSpeak(L): A Formal Computational Model. 8(3):1–27, 1998.

[61] J. Doyle, Y. Shoham, and M. P. Wellman. A Logic for Relative Desire. In Z. W. Ras and M. Zemankova, editors, *Proceedings of the Sixth International Symposium on Methodologies for Intelligent Systems (ISMIS-91)*, 16–31, 1991. Springer-Verlag.

[62] T. Eiter and V. S. Subrahmanian. Heterogeneous Active Agents, II: Algorithms and Complexity. *Artificial Intelligence*, 108(1–2):257–307, 1999.

[63] T. Eiter, V. S. Subrahmanian, and G. Pick. Heterogeneous Active Agents, I: Semantics. *Artificial Intelligence*, 108(1–2):179–255, 1999.

[64] E. A. Emerson. Temporal and Modal Mogic. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science (Vol. B): Formal Models and Semantics*, pages 995–1072. MIT Press, Cambridge, MA, USA, 1991.

[65] E. A. Emerson. Model Checking and the Mu-calculus. In *Descriptive Complexity and Finite Models, Proceedings of a DIMACS Workshop 1996*, pages 185–214, Princeton, New Jersey, USA, 1996.

[66] E. A. Emerson and J. Y. Halpern. "Sometimes" and "Not Never" Revisited: On Branching versus Linear Time Temporal Logic. *Journal of ACM*, 33(1):151–178, 1986.

[67] E. A. Emerson and J. Srinivasan. Branching Time Temporal Logic. In J. W. de Bakker, W.-P. de Roever, and G. Rozenberg, editors, *Linear Time, Branching Time, and Partial Order in Logics and Models for Concurrency*, pages 123–172. Springer-Verlag, Cambridge, MA, USA, 1989.

[68] A.K.S.I External Interfaces Working Group. Specifications of the KQML Agent Communication Language. Working Paper, 1993.

[69] R. Fagin and J. Y. Halpern. Belief, Awareness, and Limited Reasoning. *Artificial Intelligence*, 34:39–76, 1988.

[70] R. Fagin, J. Y. Halpern, Y. Moses, , and M. Y. Vardi. *Reasoning About Knowledge*. The MIT Press, 1995.

[71] I. A. Ferguson. Towards an Architecture for Adaptive, Rational, Mobile Agents. In E. Werner and Y. Demazeau, editors, *Decentralized AI 3–Proceedings of the Third European Workshop on Modeling Autonomous Agents and Multi-Agent Worlds (MAAMAW-91)*, pages 249–262, 1992.

[72] M. Fisher. Concurrent METATEM – A Language for Modeling Reactive Systems. In *Parallel Architectures and Languages, Europe (PARLE)*, volume 694 of *LNCS*, pages 185–196, Munich, Germany, 1993. Springer-Verlag.

[73] M. Fisher. A Survey of Concurrent METATEM – The Language and Its Applications. In D. M. Gabbay and H. J. Ohlbach, editors, *Temporal Logic – Proceedings of the First International Conference*, volume 827 of *LNAI*, pages 480–505. Springer-Verlag, 1994.

[74] M. Fisher and H. Barringer. Concurrent METATEM Processes – A Language for Distributed AI. In *Proceedings of the European Simulation Multiconference (ESM-91)*, Copenhagen, Denmark, 1991.

[75] M. Fisher and C. Ghidini. The ABC of Rational Agent Modeling. In *Proceedings of the First International Joint Conference on Autonomous Agents and Multi-Agent Systems (AAMAS-02)*, pages 849–856, Bologna, Italy, 2002.

[76] M. Fisher, C. Ghidini, and B. Hirsch. Organising Logic-Based Agents. In *Proceedings of Formal Approaches to Agent-Based Systems (FAABS-02)*, volume 2699 of *LNCS*, pages 15–27. Springer-Verlag, 2003.

[77] M. Fisher, C. Ghidini, and B. Hirsch. Programming Groups of Rational Agents. In *Proceedings of the 4th International Workshop on Computational Logic in Multi-Agent Systems (CLIMA-IV)*, pages 16–33, Fort Lauderdale, FL, USA, 2004.

[78] N. Fornara and M. Colombetti. Operational Specification of a Commitment-Based Agent Communication Language. In *Proceedings of the First International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS-02)*, pages 536–542, 2002.

[79] Foundations for Intelligent Physical Agents. FIPA Communicative Act Library Specification, 1997-2002.

[80] Foundations for Intelligent Physical Agents. FIPA Contract Net Interaction Protocol Specification, 1997-2002.

[81] Foundations for Intelligent Physical Agents. FIPA English Auction Interaction Protocol Specification, 1997-2002.

[82] Foundations for Intelligent Physical Agents. FIPA Query Interaction Protocol Specification, 1997-2002.

[83] Foundations for Intelligent Physical Agents. FIPA Request Interaction Protocol Specification, 1997-2002.

[84] C. Fritz and S. A. McIlraith. Decision-Theoretic GOLOG with Qualitative Preferences. In *Proceedings, Tenth International Conference on Principles of Knowledge Representation and Reasoning*, pages 153–163, Lake District of the United Kingdom, 2006.

[85] Alfredo Gabaldon. Precondition Control and the Progression Algorithm. In *Principles of Knowledge Representation and Reasoning: Proceedings of the Ninth International Conference (KR-04)*, pages 634–643, Whistler, Canada, 2004.

[86] P. Gärdenfors. *Knowledge in Flux: Modeling the Dynamics of Epistemic States*. MIT Press, Cambridge, Massachusetts, USA, 1988.

[87] M. P. Georgeff. Situated Reasoning and Rational Behavior. In *Proceedings of the Pacific Rim International Conference on Artificial Intelligence*, 1990.

[88] M. P. Georgeff and F. F. Ingrand. Decision-Making in an Enbedded Reasoning System. In *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence (IJCAI-89)*, pages 972–978, Detroit, MI, 1989.

[89] H. Ghaderi, H. J. Levesque, and Y. Lespérance. A Logical Theory of Coordination and Joint Ability. In *Proceedings of the Twenty-Second Conference on Artificial Intelligence (AAAI-07)*, pages 421–426, 2007.

[90] C. Ghidini and F. Giunchiglia. Local Models Semantics, or Contextual Reasoning = Locality + Compatibility. *Artificial Intelligence*, 127(4):221–259, 2001.

[91] B. J. Grosz and S. Kraus. Collaborative Plans for Complex Group Actions. *Artificial Intelligence*, 86(2):269–357, 1996.

[92] B. J. Grosz and C. L. Sidner. Plans for Discourse. In P. Cohen, J. Morgan, and M. Pollack, editors, *Intentions in Communication*, pages 417–444. The MIT Press, Cambridge, MA, 1990.

[93] Y. Gu and M. Soutchanski. Decidable Reasoning in a Modified Situation Calculus. In *Proceedings of the 20th International Joint Conference on Artificial Intelligence (IJCAI-07)*, pages 1891–1897, 2007.

[94] H. J. Levesque and T. Barrie. Joint Ability. University of Toronto.

[95] A. R. Haas. The Case for Domain-Specific Frame Axioms. In F. M. Brown, editor, *The Frame Problem in Artificial Intelligence: Proceedings of the 1987 Workshop*, pages 343–348. Morgan Kaufmann Publishing, 1987.

[96] J. Y. Halpern and Y. Moses. Knowledge and Common Knowledge in a Distributed Environment. In *Symposium on Principles of Distributed Computing*, pages 50–61, 1984.

[97] A. Herzig and D. Longin. A Logic of Intention with Cooperation Principles and with Assertive Speech Acts as Communication Primitives. In *Proceedings of the First International Joint Conference on Autonomous Agents and Multi-Agent Systems (AAMAS-02)*, Bologna, Italy, 2002.

[98] K. V. Hindriks, F. S. de Boer, W. van der Hoek, and J.-J. Ch. Meyer. A Formal Embedding of AgentSpeak(L) in 3APL. In *Selected Papers from the 11th Australian Joint Conference on Artificial Intelligence on Advanced Topics in*

Artificial Intelligence, volume 1502 of *LNAI*, pages 155–166. Springer-Verlag, 1998.

[99] K. V. Hindriks, F. S. de Boer, W. van der Hoek, and J.-J. Ch. Meyer. Agent Programming in 3APL. *International Journal of Autonomous Agents and Multi-Agent Systems*, 2(4):357–401, 1999.

[100] K. V. Hindriks, F. S. de Boer, W. van der Hoek, and J.-J. Ch. Meyer. Agent Programming with Declarative Goals. In C. Castelfranchi and Y. Lespérance, editors, *Intelligent Agents VII : Proc. of the 7th International Workshop ATAL 2000*, volume 1986 of *LNAI*. Springer, 2000.

[101] K. V. Hindriks, W. van der Hoek, and M. B. van Riemsdijk. Agent Programming with Temporally Extended Goals. In *Proceedings of the Eighth International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS-09)*, pages 137–144. IFAAMAS, 2009.

[102] J. Hintikka. *Knowledge and Belief*. Cornell Uiversity Press, Ithaca, NY, USA, 1962.

[103] G. J. Holzmann. *Design and Validation of Computer Protocols*. Prentice Hall, 1991.

[104] J. F. Horty and M. E. Pollack. Evaluating New Options in the Context of Existing Plans. *Artificial Intelligence*, 127:199–220, 2001.

[105] N. Howden, R. Rönnquist, A. Hodgson, and Andrew Lucas. JACK Intelligent Agents$^{\text{TM}}$– Summary of an Agent Infrastructure. In *Proceedings of the Fifth International Conference on Autonomous Agents*, 2001.

[106] J. F. Hübner. *Um Modelo de Reorganização de Sistemas Multiagentes*. PhD thesis, Universidade de São Paulo, Escola Politécnica, Brazil, 2003.

[107] T. F. Icard III, E. Pacuit, and Y. Shoham. Joint Revision of Beliefs and Intention. In *Principles of Knowledge Representation and Reasoning: Proceedings of the Twelfth International Conference (KR-10)*, Toronto, Ontario, Canada, 2010.

[108] F. F. Ingrand, M. P. Georgeff, and A. S. Rao. An Architecture for Real-Time Reasoning and System Control. *IEEE Expert*, 7(6):34–44, 1992.

[109] N. R. Jennings. Commitments and Conventions: The Foundation of Coordination in Multi-Agent Systems. *The Knowledge Engineering Review*, 8(3):223–250, 1993.

[110] N. R. Jennings. Controlling Cooperative Problem Solving in Industrial Multi-Agent Systems using Joint Intentions. *Artificial Intelligence*, 75(2):195–240, 1995.

[111] E. M. Clarke Jr., O. Grumberg, and D. A. Peled. *Model Checking*. The MIT Press, 1999.

[112] S. M. Khan. Rational Agents: Prioritized Goals, Goal Dynamics, and Agent Programming Languages with Declarative Goals. In *Proceedings of the 9th International Conference on Autonomous Agents and Multiagent Systems (AAMAS-10)*, volume 1–3, pages 1653–1654, Toronto, Ontario, Canada, 2010.

[113] S. M. Khan and Y. Lespérance. ECASL: A Model of Rational Agency for Communicating Agents. In *Proceedings of the Fourth International Joint Conference on Autonomous Agents and Multi Agent Systems (AAMAS-05)*, pages 762–769, Utrecht, The Netherlands, 2005.

[114] S. M. Khan and Y. Lespérance. A Logical Account of Prioritized Goals and their Dynamics. In *Proceedings of the Ninth International Symposium on Logical Formalizations of Commonsense Reasoning (Commonsense-09)*, pages 85–90, Toronto, Ontario, Canada, 2009.

[115] S. M. Khan and Y. Lespérance. Handling Prioritized Goals and Subgoals in a Logical Account of Goal Change. In *Proceedings of the 8th International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS-09)*, volume 2, pages 1155–1156, Budapest, Hungary, 2009.

[116] S. M. Khan and Y. Lespérance. A Logical Framework for Prioritized Goal Change. In *Proceedings of the 9th International Conference on Autonomous Agents and Multiagent Systems (AAMAS-10)*, volume 1–3, pages 283–290, Toronto, Ontario, Canada, 2010.

[117] S. M. Khan and Y. Lespérance. Prioritized Goals and Subgoals in a Logical Account of Goal Change - A Preliminary Report. In M. Baldoni, J. Bentahar, M. B. van Riemsdijk, and J. Lloyd, editors, *Declarative Agent Languages and Technologies VII, 7th International Workshop, DALT 2009, Budapest, Hungary, May 11, 2009. Revised Selected and Invited Papers*, volume 5948 of *LNCS*, pages 119–136. Springer, 2010.

[118] S. M. Khan and Y. Lespérance. Towards a Rational Agent Programming Language with Prioritized Goals. In A. Omicini, S. Sardiña, and W. Vasconcelos, editors, *Working notes of the Eighth International Workshop on Declarative*

Agent Languages and Technologies (DALT-10), pages 18–33, Toronto, Ontario, Canada, 2010.

[119] S. M. Khan and Y. Lespérance. Logical Foundations for a Rational BDI Agent Programming Language (Extended Version). In L. A. Dennis, O. Boissier, and R. H. Bordini, editors, *Programming Multi-Agent Systems - 9th International Workshop, ProMAS 2011, Taipei, Taiwan, May 3, 2011, Revised Selected Papers*, volume 7217 of *LNCS*, pages 3–21. Springer, 2011.

[120] S. M. Khan and Y. Lespérance. SR-APL: A Model for a Programming Language for Rational BDI Agents with Prioritized Goals. In *Proceedings of the 10th International Conference on Autonomous Agents and Multiagent Systems (AAMAS-11)*, volume 1–3, pages 1251–1252, Taipei, Taiwan, 2011.

[121] S. M. Khan and Y. Lespérance. Infinite Paths in the Situation Calculus (Extended Version). Technical Report EECS-2015-05, Electrical Engineering and Computer Science, York University, Toronto, Ontario, Canada, 2015.

[122] S. M. Khan and Y. Lespérance. Infinite Paths in the Situation Calculus: Axiomatization and Properties. In *Principles of Knowledge Representation and Reasoning: Proceedings of the Fifteenth International Conference (KR-16)*, pages 565–568, Cape Town, South Africa, 2016.

[123] D. Kinny. The Distributed Multi-Agent Reasoning System Architecture and Language Specification. Technical report, Australian Artificial Intelligence Institute, Melbourne, Australia, 1993.

[124] K. Konolige and M. E. Pollack. A Representationalist Theory of Intention. In *Thirteenth International Joint Conference on Artificial Intelligence (IJCAI-93)*, pages 390–395, Chambéry, France, 1993.

[125] S. Kripke. Semantical Analysis of Modal Logic I: Normal Propositional Calculi. *Zeitschrift für Mathematische Logik und Grundlagen der Mathematik*, 9:67–96, 1963.

[126] S. Kripke. Semantical Considerations on Modal Logic. *Acta Philosophica Fennica*, 16:83–94, 1963.

[127] S. Kripke. Semantical Analysis of Modal Logic II: Non-Normal Propositional Calculi. In J. W. Addison, L. Henkin, and A. Tarski, editors, *The Theory of Models*, pages 206–220. 1965.

[128] S. Kumar, M. J. Huber, D. McGee, P. R. Cohen, and H. J. Levesque. Semantics of Agent Communication Languages for Group Interaction. In *Proceedings of the Seventeenth National Conference on Artificial Intelligence and Twelfth*

Conference on on Innovative Applications of Artificial Intelligence (AAAI/IAAI-00), pages 42–47, 2000.

[129] J. E. Laird, A. Newell, and P. S. Rosenbloom. SOAR: An Architecture for General Intelligence. *Artificial Intelligence*, 33:1–64, 1987.

[130] G. Lakemeyer and H. J. Levesque. AOL: A Logic of Acting, Sensing, Knowing, and Only-Knowing. In *Principles of Knowledge Representation and Reasoning: Proceedings of the Sixth International Conference (KR-98)*, pages 316–327, 1998.

[131] J. A. Leite. Evolving Knowledge Bases. *Frontiers in Artificial Intelligence and Applications*, 81, 2003.

[132] J. A. Leite, J. J. Alferes, and L. M. Pereira. MINERVA – A Dynamic Logic Programming Agent Architecture. In J.-J. Ch. Meyer and M. Tambe, editors, *Intelligent Agents VIII: Proceedings of the 8th International Workshop on Agent Theories, Architectures, and Languages (ATAL)*, volume 2333 of *LNAI*, pages 141–157. Springer-Verlag, 2002.

[133] Y. Lespérance. On the Epistemic Feasibility of Plans in Multiagent Systems Specifications. In J.-J. C. Meyer and M. Tambe, editors, *Intelligent Agents*

VIII, Agent Theories, Architectures, and Languages, 8th International Workshop (ATAL-2001), Seattle, WA, USA, 2001.

[134] Y. Lespérance, H. J. Levesque, F. Lin, and R. Scherl. Ability and Knowing How in the Situation Calculus. *Studia Logica*, 66(1):165–186, 2000.

[135] Y. Lespérance and H.-K. Ng. Integrating Planning into Reactive High-Level Robot Programs. In *Proceedings of the Second International Cognitive Robotics Workshop*, pages 49–54, Berlin, Germany, 2000.

[136] E. Letier and A. van Lamsweerde. Deriving Operational Software Specifications from System Goals. In *Proceedings of the 10th ACM SIGSOFT Symposium on the Foundations of Software Engineering*, pages 119–128. ACM Press, 2002.

[137] H. J. Levesque. What is Planning in the Presence of Sensing? In *Proceedings of the Thirteenth National Conference on Artificial Intelligence*, pages 1139–1146, Portland, OR, USA, 1996.

[138] H. J. Levesque, P. R. Cohen, and J. T. Nunes. On Acting Together. In *Proceedings of the 8th National Conference on Artificial Intelligence (AAAI-90)*, pages 94–99, San Mateo, California, USA, 1990. Morgan Kaufmann Publishers, Inc.

[139] H. J. Levesque, F. Pirri, and R. Reiter. Foundations for a Calculus of Situations. *Electronic Transactions of AI (ETAI)*, 2(3–4):159–178, 1998.

[140] H. J. Levesque, R. Reiter, Y. Lespérance, F. Lin, and R. B. Scherl. GOLOG: A Logic Programming Language for Dynamic Domains. *Journal of Logic Programming*, 31:59–84, 1997.

[141] F. Lin and R. Reiter. State Constraints Revisited. *Journal of Logic and Computation*, 4(5):655–678, 1994.

[142] V. Louis. *Conception et Mise en Oeuvre de Modèles Formels de Calcul de Plans d'Action Complexes par un Agent Rationnel Dialoguant*. PhD thesis, Université de Caen, Caen, France, 2002.

[143] J. Malec. A Unified Approach to Intelligent Agency. In M. Wooldridge and N. R. Jennings, editors, *Intelligent Agents: Theories, Architectures, and Languages*, volume 890 of *LNAI*, pages 233–244. Springer-Verlag, Germany, 1995.

[144] V. Mascardi, M. Martelli, and L. Sterling. Logic-Based Specification Languages for Intelligent Software Agents. *Theory and Practice of Logic Programming*, 4(4):429–494, 2004.

[145] M. J. Matarić. Behavior-Based Robotics as a Tool for Synthesis of Artificial Behavior and Analysis of Natural Behavior. *Trends in Cognitive Science*, 2(3):82–87, 1998.

[146] F. McCabe and K. Clark. APRIL: Agent Process Interaction Language. In M. Wooldridge and N. R. Jennings, editors, *Intelligent Agents: Theories, Architectures, and Languages*, volume 890 of *LNAI*, pages 324–340. Springer-Verlag, Germany, 1995.

[147] J. McCarthy. Elaboration Tolerance. In *Fourth Symposium on Logical Formalizations of Commonsense Reasoning*, pages 198–216, London, England, 1998.

[148] J. McCarthy and P. J. Hayes. Some Philosophical Problems from the Standpoint of Artificial Intelligence. *Machine Intelligence*, 4:463–502, 1969.

[149] J.-J.Ch. Meyer, W. van der Hoek, and B. van Linder. A Logical Approach to the Dynamics of Commitments. *Artificial Intelligence*, 113(1–2):1–40, 1999.

[150] S. Minton, C. Knoblock, D. Kuokka, Y. Gil, R. Joseph, and J. Carbonell. PRODIGY 2.0: The Manual and Tutorial. Technical Report CMU-CS-89-146, Carnegie Mellon University, USA, 1989.

[151] R. C. Moore. A Formal Theory of Knowledge and Action. In J. R. Hobbs and R. C. Moore, editors, *Formal Theories of the Commonsense World*, pages 319–358. Ablex, 1985.

[152] R. C. Moore. A Formal Theory of Knowledge and Action. In J. F. Allen, J. Hendler, and A. Tate, editors, *Readings in Planning*, pages 480–519. Morgan Kaufmann Publishers, San Mateo, CA, USA, 1990.

[153] À. F. Moreira and R. H. Bordini. An Operational Semantics for a BDI Agent-Oriented Programming Language. In In J.-J. Ch. Meyer and M. J. Wooldridge, editors, *Proceedings of the Workshop on Logics for Agent-Based Systems (LABS-02)*, pages 45–59, 2002.

[154] À. F. Moreira, R. Vieira, and R. H. Bordini. Extending the Operational Semantics of a BDI Agent-Oriented Programming Language for Introducing Speech-Act based Communication. In *Proceedings of the Workshop on Declarative Agent Languages and Technologies (DALT-03)*, volume 2990 of *LNAI*, pages 135–154. Springer-Verlag, 2003.

[155] Y. Moses and M. Tennenholtz. Artificial Social Systems. *Computers and AI*, 14(6):533–562, 1995.

[156] J. P. Müller, M. Pischel, and M. Thiel. Modeling Reactive Behavior in Vertically Layered Agent Architectures. In M. Wooldridge and N. R. Jennings, editors, *Intelligent Agents: Theories, Architectures, and Languages*, volume 890 of *LNAI*, pages 261–276. Springer-Verlag, Germany, 1995.

[157] A. Newell and H. A. Simon. Computer Science as Empirical Enquiry. *Communications of the ACM*, 19:113–126, 1976.

[158] A. Omicini and E. Denti. From Tuple Spaces to Tuple Centres. *Science of Computer Programming*, 41(3):277–294, 2001.

[159] L. Padgham and M. Winikoff. Prometheus: A Methodology for Developing Intelligent Agents. In *Proceedings of the Third International Workshop on Agent-Oriented Software Engineering (AOSE-02)*, pages 174–185, Bologna, Italy, 2002.

[160] M. Pauly. A Logical Framework for Coalitional Effectivity in Dynamic Procedures. *Bulletin of Economic Research*, 53(4):305–324, 2002.

[161] M. Pauly. A Modal Logic for Coalitional Power in Games. *Journal of Logic and Computation*, 12(1):149–166, 2002.

[162] E. P. D. Pednault. ADL: Exploring the Middle Ground between STRIPS and the Situation Calculus. In R. J. Brachman, H. J. Levesque, and R. Reiter, editors, *Proceedings of the First International Conference on Principles of Knowledge Representation and Reasoning (KR-89)*, pages 324–332, Torotno, Ontario, Canada, May 1989. Morgan Kaufmann Publishing.

[163] R. C. Perrault. An Application of Default Logic to Speech Act Theory. In P. R. Cohen, J. Morgan, and M. E. Pollack, editors, *Intentions in Communication*, pages 161–185. MIT Press, Cambridge, Mass., USA, 1990.

[164] R. C. Perrault and J. F. Allen. A Plan-Based Analysis of Indirect Speech Acts. *American Journal of Computational Linguistics*, 6(3–4):167–182, 1980.

[165] J. A. Pinto. *Temporal Reasoning in the Situation Calculus.* PhD thesis, University of Toronto, Toronto, Ontario, Canada, 1994.

[166] F. Pirri and R. Reiter. Some Contributions to the Metatheory of the Situation Calculus. *Journal of the ACM*, 46(3):325–361, 1999.

[167] M. Pistore and P. Traverso. Planning as Model Checking for Extended Goals in Non-Deterministic Domains. In *Proceedings of the Seventeenth International Joint Conference on Artificial Intelligence (IJCAI-01)*, pages 479–484, Seattle, Washington, USA, 2001.

[168] G. Plotkin. A Structural Approach to Operational Semantics. Technical Report DAIMI-FN-19, Computer Science Dept., Aarhus University, Denmark, 1981.

[169] Amir Pnueli. The Temporal Logic of Programs. In *Proceedings of the Eighteenth Annual Symposium on Foundations of Computer Science (FOCS-77)*, pages 46–57, 1977.

[170] A. Pokahr, L. Braubach, and W. Lamersdorf. Jadex: A BDI Reasoning Engine. In R. H. Bordini, M. Dastani, J. Dix, and A. El F. Seghrouchni, editors, *Multi-Agent Programming: Languages, Platforms and Applications*, volume 15 of *Multiagent Systems, Artificial Societies, and Simulated Organizations Series*, pages 149–174. Springer-Verlag, 2005.

[171] J. L. Pollock. *Cognitive Carpentry: A Blueprint for How to Build a Person*. The MIT Press, Cambridge, MA, 1995.

[172] A. S. Rao. AgentSpeak(L): BDI Agents Speak Out in a Logical Computable Language. In W. V. Velde and J. W. Perram, editors, *Agents Breaking Away*, volume 1038 of *LNAI*, pages 42–55. Springer-Verlag, 1996.

[173] A. S. Rao and M. P. Georgeff. Asymmetry Thesis and Side-Effect Problems in Linear Time and Branching Time Intention Logics. In *Proceedings of the*

Twelfth International Joint Conference on Artificial Intelligence (IJCAI-91), pages 498–504, Sydney, Australia, 1991.

[174] A. S. Rao and M. P. Georgeff. Modeling Rational Agents with a BDI-Architecture. In R. Fikes and E. Sandewall, editors, *Proceedings of the 2nd International Conference on Principles of Knowledge Representation and Reasoning (KR-91)*, pages 473–484, San Mateo, CA, USA, 1991. Morgan Kaufmann Publishers.

[175] A. S. Rao and M. P. Georgeff. An Abstract Architecture for Rational Agents. In *Proceedings of the 3rd International Conference on Principles of Knowledge Representation and Reasoning (KR-92)*, pages 439–449, Cambridge, Massachusetts, USA, 1992. Morgan Kaufmann Publishers.

[176] R. Reiter. The Frame Problem in the Situation Calculus: A Simple Solution (sometimes) and a Completeness Result for Goal Regression. In V. Lifschitz, editor, *Artificial Intelligence and Mathematical Theory of Computation: Papers in the Honor of John McCarthy*, pages 359–380. Academic Press, San Diego, CA, USA, 1991.

[177] R. Reiter. Natural Actions, Concurrency and Continuous Time in the Situation Calculus. In *Proceedings of the Fifth International Conference on Principles*

of Knowledge Representation and Reasoning (KR-96), pages 2–13, Cambridge, Massachusetts, USA, 1996.

[178] R. Reiter. *Knowledge in Action. Logical Foundations for Specifying and Implementing Dynamical Systems*. MIT Press, 2001.

[179] S. Rosenschein and L. P. Kaelbling. The Synthesis of Digital Machines with Provable Epistemic Properties. In J. Y. Halpern, editor, *Proceedings of the 1986 Conference on Theoretical Aspects of Reasoning about Knowledge*, pages 83–98, San Mateo, CA, USA, 1986. Morgan Kaufmann Publishers.

[180] S. Shapiro. Integration of Mental Attitudes and Reasoning in Agent Theories, Languages, and Architectures. University of Toronto, 1996.

[181] M. D. Sadek. A Study in the Logic of Intention. In *Principles of Knowledge Representation and Reasoning: Proceedings of the Third International Conference (KR-92)*, pages 462–473, Cambridge, MA, USA, 1992.

[182] M. D. Sadek. Communication Theory = Rational Principles + Communicative Act Models. In *AAAI Workshop on Planning for Interagent Communication*, 1994.

[183] M. D. Sadek, P. Bretier, and E. Panaget. ARTIMIS: Natural Dialogue Meets Rational Agency. In *Proceedings of the Fifteenth International Joint Conference on Artificial Intelligence (IJCAI-97)*, pages 1030–1035, 1997.

[184] M. D. Sadek, A. Ferrieux, A. Cozannet, P. Bretier, F. Panaget, and J. Simonin. Effective Human-Computer Cooperative Spoken Dialogue: the AGS Demonstrator. In *Proceedings of the 4th International Conference on Spoken Language Processing (ICSLP-96)*, Philadelphia, PA, USA, 1996.

[185] S. Sardiña, L. de Silva, and L. Padgham. Hierarchical Planning in BDI Agent Programming Languages: A Formal Approach. In *Proceedings of the Fifth International Joint Conference on Autonomous Agents and Multi-Agent Systems (AAMAS-06)*, pages 1001–1008, Hakodate, Japan, 2003.

[186] S. Sardiña and L. Padgham. A BDI Agent Programming Language with Failure Recovery, Declarative Goals, and Planning. *Autonomous Agents and Multi-Agent Systems*, 23(1):18–70, 2011.

[187] S. Sardiña and S. Shapiro. Rational Action in Agent Programs with Prioritized Goals. In *Proceedings of the Second International Joint Conference on Autonomous Agents and Multi-Agent Systems (AAMAS-03)*, pages 417–424, 2003.

[188] R. B. Scherl and H. J. Levesque. The Frame Problem and Knowledge-Producing Actions. In *Proceedings of the Eleventh National Conference on Artificial Intelligence*, pages 689–695, Washington, DC, USA, 1993. AAAI Press/The MIT Press.

[189] R. B. Scherl and H. J. Levesque. Knowledge, Action, and the Frame Problem. *Artificial Intelligence*, 144:1–39, 2003.

[190] L. K. Schubert. Monotonic Solution of the Frame Problem in the Situation Calculus: An Efficient Method for Worlds with Fully Specified Actions. In H. E. Kyberg, R. P. Loui, and G. N. Carlson, editors, *Knowledge Representation and Defeasible Reasoning*, pages 23–67. Kluwer Academic Press, Boston, MA, USA, 1990.

[191] J. R. Searl. *Speech Acts: An Essay in the Philosophy of Language*. Cambridge University Press, 1969.

[192] J. R. Searl. Collective Intentions and Actions. In P. Cohen, J. Morgan, and M. Pollack, editors, *Intentions in Communication*, pages 401–415. MIT Press, Cambridge, MA, USA, 1990.

[193] A. El F. Seghrouchni and A. Suna. CLAIM: A Computational Language for Autonomous, Intelligent and Mobile Agents. In *Proceedings of the First Inter-*

national Workshop on Programming Multi-Agent Systems: Languages, Frameworks, Techniques, and Tools (ProMAS-03), volume 3067 of *LNCS*, pages 90–110. Springer-Verlag, 2004.

[194] S. Shapiro. *Specifying and Verifying Multiagent Systems Using the Cognitive Agents Specification Language (CASL)*. PhD thesis, University of Toronto, Toronto, Ontario, Canada, 2004.

[195] S. Shapiro. Belief Change with Noisy Sensing and Introspection. In *Proceedings of the Sixth Workshop on Nonmonotonic Reasoning, Action, and Change (NRAC-05)*, Edinburgh, 2005.

[196] S. Shapiro and G. Brewka. Dynamic Interactions between Goals and Beliefs. In *Proceedings of the 20th International Joint Conference on Artificial Intelligence (IJCAI-07)*, pages 2625–2630, Hyderabad, India, 2007.

[197] S. Shapiro and Y. Lespérance. Modeling Multiagent Systems with the Cognitive Agents Specification Language - A Feature Interaction Resolution Application. In C. Castelfranchi and Y. Lespérance, editors, *Intelligent Agents Vol. VII - Proceedings of the Seventh International Workshop on Agent Theories, Architectures, and Languages (ATAL-00)*, volume 1986 of *LNAI*, pages 244–259, Boston, USA, 2001.

[198] S. Shapiro, Y. Lespérance, and H. J. Levesque. Goals and Rational Action in the Situation Calculus - A Preliminary Report. In *Working Notes of the AAAI Fall Symposium on Rational Agency: Concepts, Theories, Models, and Applications*, Cambridge, MA, USA, November 1995.

[199] S. Shapiro, Y. Lespérance, and H. J. Levesque. Specifying Communicative Multi-Agent Systems with ConGolog. In *Working Notes of the AAAI Fall 1997 Symposium on Communicative Action in Humans and Machines*, pages 75–82, Cambridge, MA, USA, 1997. AAAI Press.

[200] S. Shapiro, Y. Lespérance, and H. J. Levesque. Goal Change in the Situation Calculus. *Journal of Logic and Computation*, 17(5):983–1018, 2007.

[201] S. Shapiro, Y. Lespérance, and H.J. Levesque. The Cognitive Agents Specification Language and Verification Environment for Multiagent Systems. In C. Castelfranchi and W. Lewis Johnson, editors, *Proceedings of the 1st Int. Joint Conference on Autonomous Agents and Multiagent Systems*, pages 19–26, Bologna, Italy, July 2002. ACM Press.

[202] S. Shapiro, M. Pagnucco, Y. Lespérance, and H. J. Levesque. Iterated Belief Change in the Situation Calculus. In A. G. Cohn, F. Giunchiglia, and B. Selman, editors, *Principles of Knowledge Representation and Reasoning: Proceedings*

of the Seventh International Conference (KR-00), pages 527–538, San Francisco, CA, USA, 2000. Morgan Kaufmann Publishers.

[203] S. Shapiro, M. Pagnucco, Y. Lespérance, and H. J. Levesque. Iterated Belief Change in the Situation Calculus. *Artificial Intelligence*, 175(1):165–192, 2011.

[204] Y. Shoham. AGENT0: A Simple Agent Language and Its Interpreter. In *Proceedings of the Ninth National Conference on Artificial Intelligence (AAAI-91)*, pages 704–709, Anaheim, California, USA, 1991. AAAI Press.

[205] Y. Shoham. Agent-Oriented Programming. *Artificial Intelligence*, 60(1):51–92, 1993.

[206] Y. Shoham. Logical Theories of Intention and the Database Perspective. *Journal of Philosophical Logic*, 38(6):633–647, 2009.

[207] Y. Shoham and M.Tennenholtz. On Social Laws for Artificial Agent Societies: Off-Line Design. *Artificial Intelligence*, 73(1–2):231–252, 1995.

[208] M. P. Singh. Group Intentions. In *Proceedings of the 10th International Workshop on Distributed Artificial Intelligence (IWDAI-90)*, Texas, USA, 1990.

[209] M. P. Singh. Group Ability and Structure. In *Decentralized AI 2 – Proceedings of the 2nd European Workshop on Modeling Autonomous Agents in a Multi-Agent World (MAAMAW-91)*, pages 127–146, France, 1991.

[210] M. P. Singh. A Critical Examination of the Cohen-Levesque Theory of Intention. In *Proceedings of the Tenth European Conference on Artificial Intelligence (ECAI-92)*, pages 364–368, Vienna, Austria, 1992.

[211] M. P. Singh. A Semantics for Speech Acts. *Annals of Mathematics and Artificial Intelligence*, 8(1–2):47–71, 1993.

[212] M. P. Singh. Intentions for Multiagent Systems. Technical Report KBNL-086-93, Information Systems Division, Microelectronics and Computer Technology Corporation, Austin, TX, USA, 1993.

[213] M. P. Singh. *Multiagent Systems: A Theoretical Framework for Intentions, Know-How, and Communications*. Number 799 in LNAI. Springer-Verlag, Heidelberg, Germany, 1994.

[214] M. P. Singh. A Social Semantics for Agent Communication Languages. In *Issues in Agent Communication: Proceedings of the IJCAI-99 Workshop on Agent Communication Languages*, LNAI, pages 31–45. Springer Verlag, 2000.

[215] S. Sohrabi, J. A. Baier, and S. A. McIlraith. HTN Planning with Preferences. In *Proceedings of the 21st International Joint Conference on Artificial Intelligence (IJCAI-09)*, pages 1790–1797, 2009.

[216] T. C. Son, E. Pontelli, and C. Baral. A Non-monotonic Goal Specification Language for Planning with Preferences. In *Advances in Knowledge Representation, Logic Programming, and Abstract Argumentation - Essays Dedicated to Gerhard Brewka on the Occasion of His 60th Birthday*, volume 9060 of *LNCS*, pages 202–217. Springer, 2015.

[217] W. Spohn. Ordinal Conditional Functions: A Dynamic Theory of Epistemic States. In W. L. Harper and B. Skyrms, editors, *Causation in Decision, Belief Change, and Statistics*, volume 2, pages 105–134. 1987.

[218] M. Tambe. Towards Flexible Teamwork. *Journal of Artificial Intelligence Research*, 7:83–124, 1997.

[219] M. Tan and R. Weihmayer. Integrating Agent-Oriented Programming and Planning for Cooperative Problem Solving. In *Proceedings of the AAAI-92 Workshop on Cooperation among Heterogeneous Intelligent Systems*, 1992.

[220] E. Ternovskaia. Automata Theory for Reasoning About Actions. In *Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence (IJCAI-99)*, pages 153–159, 1999.

[221] J. Thangarajah, L. Padgham, and M. Winikoff. Detecting and Avoiding Interference between Goals in Intelligent Agents. In *Proceedings of the Eighteenth International Joint Conference on Artificial Intelligence (IJCAI-03)*, pages 721–726, Acapulko, Mexico, 2003.

[222] M. Thielscher. FLUX: A Logic Programming Method for Reasoning Agents. *Theory and Practice of Logic Programming*, 5(4–5):533–565, 2005.

[223] S. R. Thomas. PLACA, An Agent-Oriented Programming Language. Technical Report STAN-CS-93-1487, Computer Science Department, Stanford University, Stanford, CA, USA, 1993.

[224] S. R. Thomas. The PLACA Agent Programming Language. In M. Wooldridge and N. R. Jennings, editors, *Intelligent Agents, ECAI-94 Workshop on Agent Theories, Architectures, and Languages, Amsterdam, The Netherlands, August 89, 1994 Proceedings*, volume 890 of *LNAI*, pages 355–370, Germany, 1995. Springer-Verlag.

[225] J. van Benthem and F. Liu. Diversity of Logical Agents in Games. *Philosophie Scientiae*, 8(2):163–178, 2004.

[226] J. van Benthem and F. Liu. Dynamic Logic of Preference Upgrade. *Journal of Applied Non-Classical Logics*, 17(2):157–182, 2007.

[227] W. van der Hoek, B. van Linder, and J.-J. Ch. Meyer. A Logic of Capabilities. Technical Report IR-330, Faculty of Mathematics and Computer Science, Vrije Universiteit Amsterdam, Amsterdam, The Netherlands, 1995.

[228] W. van der Hoek and M. Wooldridge. Cooperation, Knowledge, and Time: Alternating-Time Temporal Epistemic Logic and its Application. *Studia Logica*, 75(1):125–157, 2003.

[229] A. van Lamsweerde. Goal-Oriented Requirements Engineering: A Guided Tour. In *Proceedings the International Joint Conference on Requirements Engineering*, pages 249–263. IEEE, 2001.

[230] B. van Linder, W. van der Hoek, and J.-J. Ch. Meyer. Tests as Epistemic Updates–Pursuit of Knowledge. Technical Report UU-CS-1994-08, Dept. of Computer Science, Utrecht University, Utrecht, The Netherlands, 1994.

[231] B. van Linder, W. van der Hoek, and J.-J. Ch. Meyer. Actions That Make You Change Your Mind. *Knowledge and Belief in Philosophy and AI*, pages 103–146, 1995.

[232] B. van Linder, W. van der Hoek, and J.-J. Ch. Meyer. How to Motivate Your Agents–on Making Promises that You can Keep. In *International Joint Conference on Artificial Intelligence (IJCAI-95) Workshop on Agent Theories, Architectures, and Languages*, Montréal, Canada, 1995.

[233] M. B. van Riemsdijk. *Cognitive Agent Programming : A Semantic Approach*. PhD thesis, Department of Information and Computing Sciences, Universiteit Utrecht, The Netherlands, 2006.

[234] M. B. van Riemsdijk, M. Dastani, F. Dignum, and J.-J. Ch. Meyer. Dynamics of Declarative Goals in Agent Programming. In *Declarative Agent Languages and Technologies II. Second International Workshop*, LNAI, pages 1–18. Springer-Verlag, New York, NY, USA, 2004.

[235] M. B. van Riemsdijk, M. Dastani, and J.-J. Ch. Meyer. Goals in Conflict: Semantic Foundations of Goals in Agent Programming. *Autonomous Agents and Multi-Agent Systems*, 18(3):471–500, 2009.

[236] M. B. van Riemsdijk, M. Dastani, and M. Winikoff. Goals in Agent Systems: A Unifying Framework. In *Proceedings of the 7th International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS-08)*, volume 2, pages 713–720, 2008.

[237] M. B. van Riemsdijk, W. van der Hoek, and J.-J. Ch. Meyer. Agent Programming in Dribble: from Beliefs to Goals using Plans. In *Proceedings of the Second International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS-03)*, pages 393–400. ACM, 2003.

[238] S. Vassos and H. J. Levesque. Progression of Situation Calculus Action Theories with Incomplete Information. In *Proceedings of the 20th International Joint Conference on Artificial Intelligence (IJCAI-07)*, pages 2029–2024, 2007.

[239] M. Verdicchio and M. Colombetti. A Logical Model of Social Commitment for Agent Communication. In *Proceedings of the Second International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS-03)*, pages 528–535, Melbourne, Australia, 2003.

[240] S. Vere and T. Bickmore. A Basic Agent. *Computational Intelligence*, 6:41–60, 1990.

[241] R. Vieira, Á. Moreira, M. Wooldridge, and R. H. Bordini. On the Formal Semantics of Speech-Act Based Communication in an Agent-Oriented Programming Language. *Journal of Artificial Intelligence Research*, 29(221–267), 2007.

[242] W. Visser, K. Havelund, G. Brat, and S. Park. Model Checking Programs. In *Proceedings of the Fifteenth International Conference on Automated Software Engineering (ASE-00)*, pages 3–12. IEEE Computer Society, 2000.

[243] M. P. Wellman and J. Doyle. Preferential Semantics for Goals. In *Proceedings of the Ninth National Conference on Artificial Intelligence (AAAI-91)*, pages 698–703, California, USA, 1991.

[244] E. Werner. Toward a Theory of Communication and Cooperation for Multiagent Planning. In *Proceedings of the Second Conference on Theoretical Aspects of Reasoning about Knowledge (TARK-88)*, pages 129–143. Morgan Kaufman, 1988.

[245] E. Werner. Cooperating Agents : A Unified Theory of Communication and Social Structure. *Distributed Artificial Intelligence*, 2(3–36), 1989.

[246] E. Werner. What Can Agents Do Together? A Semantics for Reasoning about Cooperative Ability. In *Proceedings of the 9th European Conference on Artificial Intelligence (ECAI-90)*, pages 694–701, Stockholm, Sweden, 1990.

[247] M. Winikoff, L. Padgham, J. Harland, and J. Thangarajah. Declarative and Procedural Goals in Intelligent Agent Systems. In *Proceedings of the Eighth International Conference on Principles of Knowledge Representation and Reasoning (KR-02)*, pages 470–481, 2002.

[248] S. Wood. *Planning and Decision Making in Dynamic Domains*. Ellis Horwood Ltd., 1993.

[249] M. Wooldridge. *Reasoning about Rational Agents*. MIT Press, 2000.

[250] M. Wooldridge and N. R. Jennings. Formalizing the Cooperative Problem Solving Process. In *Proceedings of the 13th International Workshop on Distributed Artificial Intelligence (IWDAI-94)*, pages 403–417, WA, USA, 1994.

[251] M. Wooldridge and N. R. Jennings. Agent Theories, Architectures, and Languages: A Survey. In *Proceedings of the workshop on agent theories, architectures, and languages on Intelligent agents*, pages 1–39, New York, NY, USA, 1995. Springer-Verlag.

[252] M. Wooldridge and N. R. Jennings. The Cooperative Problem Solving Process. *Journal of Logic and Computation*, 9(4):563–592, 1999.

[253] M. Wooldridge, N. R. Jennings, and D. Kinny. The Gaia Methodology for Agent-Oriented Analysis and Design. *Journal of Autonomous Agents and Multi-Agent Systems*, 3(3):285–312, 2000.

[254] P. Yolum and M. P. Singh. Commitment Machines. In J.-J. Ch. Meyer and M. Tambe, editors, *Intelligent Agents VIII : 8th International Workshop, ATAL '01*, LNAI 2333, pages 235–247, Seattle, WA, USA, August 2002. Springer-Verlag.