On the Epistemic Feasibility of Plans in Multiagent Systems Specifications

Yves Lespérance

Department of Computer Science, York University
Toronto, ON, Canada M3J 1P3
lesperan@cs.yorku.ca

Abstract. This paper addresses the problem of ensuring that agents' plans are epistemically feasible in multiagent systems specifications. We propose some solutions within the Cognitive Agents Specification Language (CASL). We define a subjective execution construct **Subj** that causes the plan to be executed in terms of the agent's knowledge state, rather than in terms of the world state. The definition assumes that the agent does not do planning or lookahead, and chooses arbitrarily among the actions allowed by the plan. We also define another deliberative execution operator **Delib** for smarter agents that do planning. We show how these notions can be used to express whether a process is epistemically feasible for its agent(s) in several types of situations. More generally, the paper shows how a formalization of epistemic feasibility can be integrated with a transition-system semantics for an agent programming/specification language.

1 Introduction

In the last few years, various frameworks have been proposed to support the formal specification and verification of multiagent systems (MAS) [1, 7, 8, 29]. We have been involved in the development of such a framework, the Cognitive Agents Specification Language (CASL) [25]. CASL combines ideas from agent theory and formal methods, resulting in an expressive specification language that can be used to model and verify complex MAS.

One problem with CASL and some other MAS specification frameworks is that they do not provide good ways of ensuring that agents' plans are *epistemically feasible*, i.e., that the agents have enough knowledge to be able to execute their plans. In a real multiagent setting, each agent's behavior is determined by its own mental attitudes, i.e., its knowledge, goals, etc. At each point in time, agents must select what action to do next based on their plans and the knowledge that they have about the system's state. However in CASL (and some other frameworks), the system's behavior is simply specified as a set of concurrent processes. These processes may refer to agents' mental states — CASL includes operators that model agents' knowledge and goals — but they don't have to. There is no requirement that the modeler specify which agent is executing a given process and that he ensure that the agent has the knowledge required to execute the process.

Consider the following example adapted from Moore [18]. We have an agent, $Robbie$, that wants to open a safe, but does not know the safe's combination. There is also a sec-

ond agent, $Smartie$, that knows the safe's combination. If we take the system's CASL specification to be just the primitive action:

$$dial(Robbie, combination(Safe1), Safe1),$$

that is, $Robbie$ dials the safe's combination and $Smartie$ does nothing, then we have a process that is physically executable and must terminate in a situation where the safe is open.[1] This holds provided that an appropriate specification of the effects and (physical) preconditions of the $dial$ action and of the initial situation has been given. However, this process is not epistemically feasible because the agent does not know the safe's combination.[2] Such a process specification may be adequate if all we want is to identify a set of runs of a system. But it does not capture the internal control of the agents, how their behavior is determined by their own mental state.

The fact that in CASL, system processes are specified from an outside observer's (third-person) point of view can have advantages. In many cases, the internal control programs of the agents are not known. Sometimes, the modeler may only want a very partial model of the system capturing some scenarios of interest. In the case of simple purely reactive agents, the modeler may not want to ascribe mental attitudes to the agents. Also, natural events and processes are best specified objectively. However, this loose coupling between specification and system means that it is easy to write specifications that could not be executed by agents. Often, one would want to ensure that the specifications are epistemically feasible for the agents.

For our example, if we want to ensure that the process is epistemically feasible for the agents, we should use a specification more like the following:

$$\begin{array}{l}(\ \mathbf{KRef}(Robbie, combination(Safe1))?;\\ \quad dial(Robbie, combination(Safe1), Safe1)\ )\\ \|\\ informRef(Smartie, Robbie, combination(Safe1)).\end{array}$$

Here, in the first concurrent process, $Robbie$ waits until it knows what the safe's combination is and then dials it, and in the second process $Smartie$ tells $Robbie$ what the combination is — $\delta_1 \| \delta_2$ represents the concurrent execution of δ_1 and δ_2. We might also want to require that each agent know that the preconditions of its actions are satisfied before it does them, e.g., that $Robbie$ know that it is possible for him to dial a combination on the safe. This sort of requirements has been studied in agent theory under the labels "knowledge prerequisites of action", "knowing how to execute a program", "ability to achieve a goal", "epistemic feasibility", etc. [18, 19, 28, 3, 12, 15, 16]. The

[1] Formally:

$$\begin{array}{l}\exists s\, Do(dial(Robbie, combination(Safe1), Safe1), S_0, s) \wedge\\ \forall s(Do(dial(Robbie, combination(Safe1), Safe1), S_0, s) \supset Open(Safe1, s)).\end{array}$$

The notation is explained in Section 3.

[2] $combination(Safe1)$ is a fluent whose value varies according to the agent's epistemic alternatives; the situation argument can be made explicit by writing $combination(Safe1, now)$; see Section 3.

modeler could explicitly include all these knowledge prerequisites in the process specification. But it would be better if there was a way to simply say that the first process is going to be *subjectively executed* by $Robbie$ and the second process by $Smartie$, and to have all the knowledge prerequisites fall out automatically; something like the following:

$$
\begin{array}{l}
system1 \stackrel{\text{def}}{=} \\
\mathbf{Subj}(Robbie, \mathbf{KRef}(Robbie, combination(Safe1))?; \\
\qquad\qquad dial(Robbie, combination(Safe1), Safe1)) \parallel \\
\mathbf{Subj}(Smartie, informRef(Smartie, Robbie, combination(Safe1))).
\end{array}
$$

In this process specification which we call $system1$, we use a new construct $\mathbf{Subj}(agt, \delta)$, which means that the process specification δ is subjectively executed by agent agt, that is, that δ is executed by agt in terms of his knowledge state. Note that we could have made the example more realistic by having $Robbie$ request $Smartie$ to inform him of the combination and having $Smartie$ respond to such requests, as in the examples of [25]; but here, we prefer to keep the example simple. We return it in Section 3.

In this paper, we explore these issues, and propose an account of subjective plan execution in CASL that ensures that the plan can be executed by the agent based on its knowledge state. Our account of basic subjective execution (**Subj**) assumes that the agent does not do planning or lookahead as it executes its program. We also develop an account of deliberative plan execution (**Delib**) for smarter agents that do planning/lookahead. These notions are defined on top of CASL's transition-system semantics. In fact, one of the paper's contributions is showing how a formalization of epistemic feasibility can be adapted for use with an agent programming/specification language with a transition-system semantics. Note that the paper focuses on developing a reasonable model of agenthood for use in producing better specifications of MAS. This model should later prove useful for obtaining more accurate formal semantics for agent programming languages that interleave sensing and communication with planning and plan execution, e.g., IndiGolog [5].

2 Overview of CASL

The Cognitive Agents Specification Language (CASL) [25] is a formal specification language for multiagent systems. It combines a theory of action [22, 23] and mental states [24] based on the situation calculus [17] with ConGolog [4], a concurrent, nondeterministic programming language that has a formal semantics. The result is a specification language that contains a rich set of operators to facilitate the specification of *complex* multiagent systems. A CASL specification of a system involves two components: a specification of the dynamics of the domain and a specification of the behavior of the agents in the system. Let us describe how these components are modeled.

2.1 Modeling Domain Dynamics

The *domain dynamics* component of a CASL specification states what properties and relations are used to model the state of the system, what actions may be performed by

the agents, what their preconditions and effects are, and what is known about the initial state of the system (the specification may be incomplete). The model can include a specification of the agents' mental states, i.e., what knowledge and goals they have, as well as of the dynamics of these mental states, i.e., how knowledge and goals are affected by communication actions (e.g., inform, request, cancel-request, etc.) and perception actions. This component is specified in a purely declarative way in the situation calculus [17].

Very briefly, the situation calculus is a language of predicate logic for representing dynamically changing worlds. In this language, a possible world history (a sequence of actions) is represented by a first order term called a *situation*. The constant S_0 denotes the initial situation and the term $do(\alpha, s)$ denotes the situation resulting from action α being performed in situation s. Relations (functions) that vary from situation to situation, called *fluents*, are represented by predicate (function) symbols that take a situation term as last argument. The special predicate $Poss(\alpha, s)$ is used to represent the fact that primitive action α is physically possible in situation s.

We use Reiter's solution to the frame problem, where effects axioms are compiled into successor state axioms [22, 23]. Thus, for our example, the domain dynamics specification includes the successor state axiom:

$$Open(x, do(a, s)) \equiv$$
$$\exists agt, c(a = dial(agt, c, x) \wedge c = combination(x, s)) \vee Open(x, s),$$

i.e., the safe x is open in the situation that results from action a being performed in situation s if and only if a is the action of dialing x's correct combination on x or if x was already open in situation s. We also have a successor state axiom for the $combination$ fluent, whose value is not affected by any action:

$$combination(x, do(a, s)) = c \equiv combination(x, s) = c.$$

The specification also includes the action precondition axiom:

$$Poss(dial(agt, c, x), s) \equiv True,$$

i.e., that the $dial$ action is always physically possible. We also specify the agent of the action by:

$$agent(dial(agt, c, x)) = agt.$$

Knowledge is represented by adapting a possible world semantics to the situation calculus [18, 24]. The accessibility relation $K(agt, s', s)$ represents the fact that in situation s, the agent agt thinks that the world could be in situation s'. An agent knows that ϕ if and only if ϕ is true in all his K-accessible situations:

$$\mathbf{Know}(agt, \phi, s) \stackrel{\text{def}}{=} \forall s'(K(agt, s', s) \supset \phi[s']).$$

Here, $\phi[s]$ represents the formula obtained by substituting s for all instances of the special constant now; thus for e.g., $\mathbf{Know}(agt, Open(Safe1, now), s)$ is an abbreviation for $\forall s'(K(agt, s', s) \supset Open(Safe1, s'))$. Often, when no confusion is possible, we suppress now altogether and simply write for e.g., $\mathbf{Know}(agt, Open(Safe1), s)$.

We assume that K is reflexive, transitive, and euclidean, which ensures that what is known is true, and that the agents always know whether they know something (positive and negative introspection). We also use the abbreviations $\mathbf{KWhether}(agt, \phi, s) \stackrel{\text{def}}{=} \mathbf{Know}(agt, \phi, s) \vee \mathbf{Know}(\neg agt, \phi, s)$, i.e., agt knows whether ϕ holds in s and $\mathbf{KRef}(agt, \theta, s) \stackrel{\text{def}}{=} \exists t\, \mathbf{Know}(agt, t = \theta, s)$, i.e., agt knows who/what θ is.

In this paper, we handle the following types knowledge-producing actions: binary sensing actions, e.g., $sense_{Open(Safe1)}(agt)$, where the agent senses the truth-value the associated proposition, non-binary sensing actions, e.g., $read_{combination(Safe1)}(agt)$, where the agent senses the value the associated term, and two generic communication actions, $informWhether(agt_1, agt_2, \phi)$ where agent agt_1 informs agent agt_2 of the truth-value of the proposition ϕ, and $informRef(agt_1, agt_2, \theta)$, where agent agt_1 informs agent agt_2 of the value of the term θ.[3] Following [15], the information provided by a binary sensing action is specified using the predicate $SF(a, s)$, which holds if action a returns the binary sensing result 1 in situation s. For example, we might have an axiom:

$$SF(sense_{Open(Safe1)}(agt), s) \equiv Open(Safe1, s),$$

i.e., the action $sense_{Open(Safe1)}(agt)$ will tell agt whether $Safe1$ is open in the situation where it is performed. Similarly for non-binary sensing actions, we use the term $sff(a, s)$ to denote the sensing value returned by the action; for example, we might have:

$$sff(read_{combination(Safe1)}(agt), s) = combination(Safe1, s),$$

i.e., $read_{combination(Safe1)}(agt)$ tells agt the value of $Safe1$'s combination.

We specify the dynamics of knowledge with the following successor state axiom:

$$\begin{array}{l} K(agt, s^*, do(a, s)) \equiv \\ \exists s'[K(agt, s', s) \wedge s^* = do(a, s') \wedge Poss(a, s') \wedge \\ \quad (BinarySensingAction(a) \wedge agent(a) = agt \supset (SF(a, s') \equiv SF(a, s))) \wedge \\ \quad (NonBinarySensingAction(a) \wedge agent(a) = agt \supset sff(a, s') = sff(a, s)) \wedge \\ \quad \forall informer, \phi(a = informWhether(informer, agt, \phi) \supset \phi[s'] = \phi[s]) \wedge \\ \quad \forall informer, \theta(a = informRef(informer, agt, \theta) \supset \theta[s'] = \theta[s])]. \end{array}$$

This says that that after an action happens, every agent learns that it has happened. 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 of whether ϕ holds, then agt knows this afterwards, and if the action involves someone informing agt of who/what θ is, then agt knows it afterwards ($\theta[s]$ stands for θ with s substituted for now, similarly to $\phi[s]$) . The preconditions of the communication actions are defined by the following axioms:

$$Poss(informWhether(informer, agt, \phi), s) \equiv \mathbf{KWhether}(informer, \phi, s),$$

[3] Since the action $informWhether$ takes a formula as argument, we must encode formulas as terms; see [4] for how this is done. For notational simplicity, we suppress this encoding and use formulas as terms directly.

i.e., $informWhether$ is possible in situation s if and only if $informer$ knows whether ϕ holds, and

$$Poss(informRef(informer, agt, \theta), s) \equiv \mathbf{KRef}(informer, \theta, s),$$

i.e., $informRef$ is possible in situation s if and only if $informer$ knows who/what θ is. There are also axioms stating that the informer is the agent of these actions. Goals and requests are modeled in an analogous way; see [25] for details.

Thus, the dynamics of a domain can be specified in CASL using an action theory that includes the following kinds of axioms:

- initial state axioms, which describe the initial state of the domain and the initial mental states of the agents;
- action precondition axioms, one for each action, which characterize $Poss$;
- successor state axioms, one for each fluent;
- axioms that specify which actions are sensing actions and what fluents they sense, characterizing SF and sff;
- axioms that specify the agent of every action;
- unique names axioms for the actions;
- some domain-independent foundational axioms [10, 23].

2.2 Modeling Agent Behavior

The second component of a CASL model is a specification of the *behavior of the agents* in the domain. Because we are interested in modeling domains involving complex processes, this component is specified procedurally. For this, we use the ConGolog programming/process description language, which provides the following rich set of constructs:

α,	primitive action
$\phi?$,	wait for a condition
$\delta_1; \delta_2$,	sequence
$\delta_1 \mid \delta_2$,	nondeterministic branch
$\pi\, x\, \delta$,	nondeterministic choice of argument
δ^*,	nondeterministic iteration
if ϕ **then** δ_1 **else** δ_2 **endIf**,	conditional
while ϕ **do** δ **endWhile**,	while loop
$\delta_1 \parallel \delta_2$,	concurrency with equal priority
$\delta_1 \rangle\!\rangle \delta_2$,	concurrency with δ_1 at a higher priority
$\delta^{\parallel}$,	concurrent iteration
$\langle\, \boldsymbol{x} : \phi \rightarrow \delta \,\rangle$,	interrupt
$p(\boldsymbol{\theta})$,	procedure call.

The semantics of the ConGolog process description language [4] is defined in terms of *transitions*, in the style of structural operational semantics [21, 9]. A transition is a single step of computation, either a primitive action or testing whether a condition holds in the current situation. Two special predicates are introduced, $Final$ and $Trans$,

where $Final(\delta, s)$ means that program δ may legally terminate in situation s, and where $Trans(\delta, s, \delta', s')$ 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 axioms such as:

$$Trans(\alpha, s, \delta, s') \equiv Poss(\alpha[s], s) \wedge \delta = nil \wedge s' = do(\alpha[s], s),$$
$$Final(\alpha, s) \equiv False,$$

$$Trans([\delta_1; \delta_2], s, \delta, s') \equiv$$
$$\quad Final(\delta_1, s) \wedge Trans(\delta_2, s, \delta, s')$$
$$\quad \vee \exists \delta' (\delta = (\delta'; \delta_2) \wedge Trans(\delta_1, s, \delta', s')),$$
$$Final([\delta_1; \delta_2], s) \equiv Final(\delta_1, s) \wedge Final(\delta_2, s).$$

The first axiom says that a program involving a primitive action α may perform a transition in situation s provided that $\alpha[s]$ is possible in s, with the resulting situation being $do(\alpha[s], s)$ and the remaining program being the empty program nil ($\alpha[s]$ stands for α with s substituted for now, similarly to $\phi[s]$). The second axiom says that a program with a primitive action remaining can never be considered to have terminated. The third axiom says that one can perform a transition for a sequence by performing a transition for the first part, or by performing a transition for the second part provided that the first part has already terminated. The last axiom says that a sequence has terminated when both parts have terminated.[4]

The axioms for the other ConGolog constructs that we will use are as follows:

$$Trans(\phi?, s, \delta, s') \equiv \phi[s] \wedge \delta = nil \wedge s' = s,$$
$$Final(\phi?, s) \equiv False,$$

$$Trans((\delta_1 \| \delta_2), s, \delta, s') \equiv$$
$$\quad \exists \delta_1' (\delta = (\delta_1' \| \delta_2) \wedge Trans(\delta_1, s, \delta_1', s')$$
$$\quad \vee \exists \delta_2' (\delta = (\delta_1 \| \delta_2') \wedge Trans(\delta_2, s, \delta_2', s')),$$
$$Final((\delta_1 \| \delta_2), s) \equiv Final(\delta_1, s) \wedge Final(\delta_2, s),$$

$$Trans((\delta_1 | \delta_2), s, \delta, s') \equiv Trans(\delta_1, s, \delta, s') \vee Trans(\delta_2, s, \delta, s'),$$
$$Final((\delta_1 | \delta_2), s) \equiv Final(\delta_1, s) \vee Final(\delta_2, s),$$

$$Trans(\pi\, v\, \delta, s, \delta', s') \equiv \exists x\, Trans(\delta_x^v, s, \delta', s'),$$
$$Final(\pi\, v\, \delta, s) \equiv \exists x\, Final(\delta_x^v, s),$$

$$Trans(\textbf{if } \phi \textbf{ then } \delta_1 \textbf{ else } \delta_2 \textbf{ endIf}, s, \delta, s') \equiv$$
$$\quad \phi[s] \wedge Trans(\delta_1, s, \delta, s') \vee \neg\phi[s] \wedge Trans(\delta_2, s, \delta, s'),$$
$$Final(\textbf{if } \phi \textbf{ then } \delta_1 \textbf{ else } \delta_2 \textbf{ endIf}, s) \equiv$$
$$\quad \phi[s] \wedge Final(\delta_1, s) \vee \neg\phi[s] \wedge Final(\delta_2, s).$$

Note that to handle recursive procedures, a more complex formalization must be used; see [4] for the details.

[4] Note that we use axioms rather than rules to specify $Trans$ and $Final$ because we want to support reasoning about possible executions of a system given an incomplete specification of the initial situation. Since these predicates take programs (that include test of formulas) as arguments, this requires encoding formulas and programs as terms; see [4] for the details. For notational simplicity, we suppress this encoding and use programs as terms directly.

The overall semantics of a ConGolog program is specified by the Do relation:

$$Do(\delta, s, s') \stackrel{\text{def}}{=} \exists \delta'(Trans^*(\delta, s, \delta', s') \wedge Final(\delta', s')),$$

$$\begin{aligned} Trans^*(\delta, s, \delta', s') \stackrel{\text{def}}{=} \forall T[& \\ & \forall \delta_1, s_1\, T(\delta, s, \delta, s) \wedge \\ & \forall \delta, s(Trans(\delta_1, s_1, \delta_2, s_2) \wedge T(\delta_2, s_2, \delta_3, s_3) \supset T(\delta_1, s_1, \delta_3, s_3)) \\ & \supset T(\delta, s, \delta', s')] \end{aligned}$$

$Do(\delta, s, s')$ holds if and only if s' is a legal terminating situation of process δ started in situation s, i.e., a situation that can be reached by performing a sequence of transitions starting with program δ in situation s and where the program may legally terminate. $Trans^*$ is the reflexive transitive closure of the transition relation $Trans$.[5]

CASL's approach aims for a middle ground between purely intentional (i.e., mental attitudes-based) specifications of agents, which typically allow only very weak predictions about the behavior of agents to be made, and the usual kind of concurrent process specifications, which are too low-level, and don't model mental states at all. Because of its logical foundations, CASL can accommodate incompletely specified models, both in the sense that the initial state of the system is not completely specified, and in the sense that the processes involved are nondeterministic and may evolve in any number of ways.

The latest version of the CASL framework, which supports communication with encrypted speech acts and provides a simplified account of goals is described in [25]. That paper also describes how CASL was used to model a complex multiagent system for feature interaction resolution in telecommunication applications, a system that involves negotiating, autonomous agents with explicit goals. Earlier versions of CASL are described in [26, 13], where the use of the formalism is illustrated with a simple meeting scheduling multiagent system example. A discussion of how the process modeling features of the framework can be used for requirements engineering appears in [11]; this paper also discusses simulation and verification tools that are being developed.

3 Subjective Execution

We define the subjective execution construct $\mathbf{Subj}(agt, \delta)$ introduced in Section 1 as follows:

$$\begin{aligned} & Trans(\mathbf{Subj}(agt, \delta), s, \gamma, s') \equiv \exists \delta'(\gamma = \mathbf{Subj}(agt, \delta') \wedge \\ & \quad [\mathbf{Know}(agt, Trans(\delta, now, \delta', now), s) \wedge s' = s \vee \\ & \quad \exists a(\mathbf{Know}(agt, Trans(\delta, now, \delta', do(a, now)) \wedge agent(a) = agt, s) \\ & \qquad \wedge s' = do(a, s))]), \\ & Final(\mathbf{Subj}(agt, \delta), s) \equiv \mathbf{Know}(agt, Final(\delta, now), s). \end{aligned}$$

This means that when a program is executed subjectively, the system can make a transition only if the agent knows that it can make this transition, and if the transition involves

[5] To define the relation properly, we use second-order logic. For automated reasoning, one could use a first-order version to prove some, but not all, consequences of the theory.

a primitive action, then this action must be performed by the agent himself. A subjective execution may legally terminate only if the agent knows that it may.

Let's go back to the examples of Section 1. Assume that in the initial situation S_0, $Robbie$ does not know what the safe's combination is but $Smartie$ does; formally:

$$\neg\mathbf{KRef}(Robbie, combination(Safe1), S_0) \wedge \\ \mathbf{KRef}(Smartie, combination(Safe1), S_0).$$

Then, we can easily show that $Robbie$ cannot open the by himself, that the program where he just dials the safe's combination is not subjectively executable:

$$\neg\exists s Do(\mathbf{Subj}(Robbie, dial(Robbie, combination(Safe1), Safe1)), S_0, s).$$

To see this, observe that $Robbie$'s not knowing the combination initially amounts to $\neg\exists c \forall s(K(Robbie, s, S_0) \supset combination(Safe1, s) = c)$. Thus, there are K-accessible situations for $Robbie$ in S_0, say s_1 and s_2, where $combination(Safe1, s_1) \neq combination(Safe1, s_2)$. It follows by the unique name axioms for actions that $Robbie$'s dialing the combination is a different action in these situations, i.e., $dial(Robbie, combination(Safe1, s_1), Safe1) \neq dial(Robbie, combination(Safe1, s_2), Safe1)$, and thus that $Robbie$ does not know what that action is in S_0, i.e.,

$$\neg\exists a\, \mathbf{Know}(Robbie, dial(Robbie, combination(Safe1, now), Safe1), S_0).$$

Thus, $Robbie$ does not know what transition to perform in S_0, i.e.,

$$\neg\exists a, \delta\, \mathbf{Know}(Robbie, Trans(DC, now, \delta, do(a, now), S_0),$$

where $DC \stackrel{\text{def}}{=} dial(Robbie, combination(Safe1, now), Safe1)$, and there is no transition where the program is subjectively executed, i.e., $\neg\exists\delta, s\, Trans(\mathbf{Subj}(Robbie, DC), S_0, \delta, s)$. Since the program cannot legally terminate (i.e., is not $Final$ in S_0), the program is not subjectively executable.

If on the other hand $Smartie$ does inform $Robbie$ of what the combination is, as in the $system1$ example, then one can easily show that the processes involved are subjectively executable, i.e., $\exists s Do(system1, S_0, s)$. Note that the test action $\mathbf{KRef}(Robbie, combination(Safe1))?$ in $system1$ is redundant, since as seen earlier, the $dial$ action cannot make a transition unless $Robbie$ knows the combination.

The account also handles cases involving sensing actions. For example, if $Robbie$ first reads the combination of the safe (assume he has it written on a piece of paper), and then dials it, then the resulting process is subjectively executable, i.e.:

$$\exists s Do(\mathbf{Subj}(Robbie, [read_{combination(Safe1)}(Robbie); \\ dial(Robbie, combination(Safe1), Safe1)]), S_0, s).$$

To get a better understanding of this notion, let's look at some of its properties. First, for a primitive action α, an agent can subjectively execute the action if it knows what the action is (including the value of fluent arguments such as $combination(Safe1)$) and knows that it is possible for him to execute it in the current situation:

Proposition 1.

$$Trans(\mathbf{Subj}(agt, \alpha), s, \delta, s') \equiv s' = do(\alpha[s], s) \wedge \delta = \mathbf{Subj}(agt, nil) \wedge \\ \exists a \mathbf{Know}(agt, \alpha = a \wedge Poss(a, now) \wedge agent(a) = agt, s).$$

Secondly, for a test/wait action involving a condition ϕ, an agent can subjectively execute the action only if it knows that ϕ now holds:

Proposition 2.

$$Trans(\mathbf{Subj}(agt, \phi?), s, \delta, s') \equiv s' = s \wedge \delta = \mathbf{Subj}(agt, nil) \wedge \mathbf{Know}(agt, \phi, s).$$

Thirdly, for an if-then-else program where the "then" and "else" branches involve different first transitions, an agent can subjectively execute it if it knows that the condition ϕ is now true and can subjectively execute the "then" branch, or knows that ϕ is now false and can subjectively execute the "else" branch:

Proposition 3.

$$\neg \exists \delta, s' (Trans(\delta_1, s, \delta, s') \wedge Trans(\delta_2, s, \delta, s')) \supset \\ [Trans(\mathbf{Subj}(agt, \mathbf{if}\ \phi\ \mathbf{then}\ \delta_1\ \mathbf{else}\ \delta_2\ \mathbf{endIf}, s, \delta, s') \equiv \\ \mathbf{Know}(agt, \phi, s) \wedge Trans(\mathbf{Subj}(agt, \delta_1), s, \delta, s') \vee \\ \mathbf{Know}(agt, \neg\phi, s) \wedge Trans(\mathbf{Subj}(agt, \delta_2), s, \delta, s')]$$

So we see how with subjective execution, the tests and fluents that appear in the program as well as the preconditions of the actions, are all evaluated against the agent's knowledge state, rather than against the world state.

For deterministic single-agent programs, i.e., programs where $|$ (nondeterministic branch), π (nondeterministic choice of argument), $*$ (nondeterministic iteration), and $\|$ (concurrency) do not occur, we can consider $\exists s' Do(\mathbf{Subj}(agt, \delta), s, s')$ to be an adequate formalization of epistemic feasibility. In this case, it is sufficient that the agent know which transition to perform at each step and know when he can legally terminate. For nondeterministic programs on the other hand, we must consider how the agent chooses which transition to perform among the possibly many that are allowed. **Subj** can be viewed as capturing the behavior of an agent that executes its program in a bold or blind manner. When several transitions are allowed by the program, the agent chooses the next transition arbitrarily; it does not do any lookahead to make a good choice. So it can easily end up in a dead end. For example, consider the program $\mathbf{Subj}(agt, (a; False?)|b)$, in a situation where the agent knows that both actions a and b are possible. Then, the agent might choose to do a transition by performing action a (rather than b), at which point the program $False?$ remains, for which there are no possible transitions and which cannot successfully terminate. If we want to ensure that an agent can subjectively execute a nondeterministic program successfully, we must ensure that every path through the program is subjectively executable and leads to successful termination. Note that this blind execution mode is the default one in the IndiGolog agent programming language [5].

So for nondeterministic programs, the existence of some path through the program (Do) that is subjectively executable is not sufficient to guarantee epistemic feasibility. We must ensure that all paths through the program are subjectively executable. We can

capture this formally by defining a new predicate **AllDo**(δ, s) that holds if all executions of program δ starting in situation s eventually terminate successfully:

$$\begin{array}{l}\textbf{AllDo}(\delta, s) \stackrel{\text{def}}{=} \forall R[\\ \quad \forall \delta_1, s_1(Final(\delta_1, s_1) \supset R(\delta_1, s_1)) \wedge \\ \quad \forall \delta_1, s_1(\exists \delta_2, s_2\, Trans(\delta_1, s_1, \delta_2, s_2) \wedge \\ \qquad\quad \forall \delta_2, s_2(Trans(\delta_1, s_1, \delta_2, s_2) \supset R(\delta_2, s_2)) \\ \qquad\quad \supset R(\delta_1, s_1)) \\ \supset R(\delta, s)].\end{array}$$

AllDo(δ, s) holds if and only if (δ, s) is in the least relation R such that (1) if (δ_1, s_1) can legally terminate, then it is in R, and (2) if some transition can be performed in (δ_1, s_1) and every such transition gets us to a configuration that is in R, then (δ_1, s_1) is also in R.[6] It is easy to see that if all executions of δ in s eventually terminate successfully then some execution eventually terminates successfully, i.e.:

$$\textbf{AllDo}(\delta, s) \supset \exists s' Do(\delta, s, s').$$

So, we formalize *epistemic feasibility* for single-agent programs where the agent executes the program blindly by the predicate **KnowHowSubj**(agt, δ, s), which is defined as follows:

$$\textbf{KnowHowSubj}(agt, \delta, s) \stackrel{\text{def}}{=} \textbf{AllDo}(\textbf{Subj}(agt, \delta), s)$$

i.e., every subjective execution of δ by agt starting in s eventually terminates successfully. For systems involving two agents agt_1 and agt_2 that blindly execute programs δ_1 and δ_2 concurrently, we can define epistemic feasibility as follows:

$$\textbf{KnowHowSubj}(agt_1, \delta_1, agt_2, \delta_2, , s) \stackrel{\text{def}}{=} \textbf{AllDo}(\textbf{Subj}(agt_1, \delta_1) \| \textbf{Subj}(agt_2, \delta_2), s).$$

Since the agents are not doing any lookahead, to guarantee that the process will be executed successfully, we must ensure that no matter how the agents' programs are interleaved and no matter which transitions they choose, the execution will terminate successfully. This can be generalized for processes that involve more than two agents and/or use composition methods other than concurrency. Again, if the agents are all executing their program blindly, then we must require that all executions terminate successfully. So for a multiagent process δ where agents are all blind executors — the agents' programs in δ must all be inside **Subj** operators — we define epistemic feasibility as follows:

$$\textbf{KnowHowSubj}(\delta, s) \stackrel{\text{def}}{=} \textbf{AllDo}(\delta, s).$$

Subj is very similar to the notion called "dumb knowing how" **DKH**(δ, s) formalized in [12] for δs that are Golog programs, i.e., ConGolog programs without concurrency or interrupts. For any deterministic single-agent Golog program δ, we believe that **Subj** and **DKH** are essentially equivalent, in the sense that:

$$\exists s' Do(\textbf{Subj}(agt, \delta), s, s') \equiv \textbf{DKH}(\delta, s).$$

[6] **AllDo** is somewhat similar to the operator $\mathbf{AF}\phi$ in the branching time logic CTL^* [2]. Properties of processes like **AllDo** are often specified in the μ-calculus [20]; see [6] for a discussion in the context of the situation calculus.

We also believe that for nondeterministic single-agent Golog programs δ, we have that:

$$\textbf{AllDo}(\textbf{Subj}(agt, \delta), s) \equiv \textbf{DKH}(\delta, s).$$

(We hope to prove these conjectures in future work.) But note that **Subj** is considerably more general than **DKH**; **Subj** can be used to specify systems involving concurrent processes and multiple agents, as in the $system1$ example.

4 Deliberative Execution

In the previous section, we developed an account of subjective execution that took the agent to be executing the program blindly, without doing any lookahead or deliberation. In this section, we propose another account that captures when a smart agent that does deliberation knows how to execute a program. We use the notation $\textbf{Delib}(agt, \delta)$ for this notion of *deliberative execution*. It is formalized as follows:

$$\begin{array}{l} Trans(\textbf{Delib}(agt, \delta), s, \gamma, s') \equiv \exists \delta'(\gamma = \textbf{Delib}(agt, \delta') \wedge \\ \quad [\textbf{Know}(agt, Trans(\delta, now, \delta', now) \wedge \textbf{KnowHowDelib}(agt, \delta', now), s) \wedge s' = s \vee \\ \quad \exists a(\textbf{Know}(agt, Trans(\delta, now, \delta', do(a, now)) \wedge agent(a) = agt \\ \qquad\qquad \wedge \textbf{KnowHowDelib}(agt, \delta', do(a, now)), s) \wedge s' = do(a, s))]), \end{array}$$

$$\begin{array}{l} \textbf{KnowHowDelib}(agt, \delta, s) \stackrel{\text{def}}{=} \forall R[\\ \quad \forall \delta_1, s_1(\textbf{Know}(agt, Final(\delta_1, now), s_1) \supset R(\delta_1, s_1)) \wedge \\ \quad \forall \delta_1, s_1(\exists \delta_2\, \textbf{Know}(agt, Trans(\delta_1, now, \delta_2, now) \wedge R(\delta_2, now), s) \\ \qquad \supset R(\delta_1, s_1)) \wedge \\ \quad \forall \delta_1, s_1(\exists a, \delta_2\, \textbf{Know}(agt, Trans(\delta_1, now, \delta_2, do(a, now)) \wedge agent(a) = agt \\ \qquad \wedge R(\delta_2, do(a, now)), s_1) \supset R(\delta_1, s_1)) \\ \supset R(\delta, s)], \end{array}$$

$$Final(\textbf{Delib}(agt, \delta), s) \equiv \textbf{Know}(agt, Final(\delta, now), s).$$

This means that the system can make a transition only if the agent knows that it can make this transition and also knows how to deliberatively execute the rest of the program all the way to a final situation. The agent knows how to deliberatively execute δ in s, $\textbf{KnowHowDelib}(agt, \delta, s)$, if and only if (δ, s) is in the least relation R such that (1) if the agent knows that (δ_1, s_1) can legally terminate, then it is in R, and (2) if the agent knows that it can perform a transition in (δ_1, s_1) to get to a configuration that is in R, then (δ_1, s_1) is also in R. The system may legally terminate only if the agent knows that it may. We take $\textbf{KnowHowDelib}(agt, \delta, s)$ to be our formalization of epistemic feasibility for the case of single-agent (deterministic or nondeterministic) programs where the agent does deliberation/lookahead.

Let us look at an example. Consider the following program, which we call $system2$:

$$\begin{array}{l} system2 \stackrel{\text{def}}{=} \\ \quad \textbf{Subj}(Smartie, \pi\, c[dial(Smartie, c, Safe1); Open(Safe1)?]). \end{array}$$

Here, $Smartie$ must nondeterministically choose a combination, dial it, and then verify that the safe is open. An agent that chooses transitions arbitrarily without doing lookahead is likely to choose and dial the wrong combination, since it does not consider the

need to satisfy the test that the safe is open when it chooses its first transition. That is, we can show that:

$$\exists \delta', c(c \neq combination(Safe1, S_0) \wedge \\ Trans(system2, S_0, \delta', do(dial(Smartie, c, Safe1), S_0))).$$

But a deliberative agent that does lookahead would be able to determine that it must dial the right combination. For $system2'$ which is just like $system2$, but with **Subj** replaced by **Delib**, the only possible transition is that where the agent dials the right combination. Thus, we can show that:

$$\exists \delta', s' \, Trans(system2', S_0, \delta', s') \wedge \\ \forall \delta', s' (Trans(system2', S_0, \delta', s') \supset \\ s' = do(dial(Smartie, combination(Safe1), Safe1), S_0)).$$

Let's look at some of the properties of **Delib** and compare it to **Subj**. First, we have that:

Proposition 4. *The properties of propositions 1, 2, and 3 also hold for* **Delib**.

To see where **Delib** and **Subj** differ, consider a program $\alpha; \delta$ involving a sequence that starts with a primitive action. Then, we have that:

Proposition 5.

$$Trans(\mathbf{Subj}(agt, \alpha; \delta), s, \delta', s') \equiv s' = do(\alpha[s], s) \wedge \delta' = \mathbf{Subj}(agt, nil; \delta) \wedge \\ \exists a \, \mathbf{Know}(agt, \alpha = a \wedge Poss(a, now) \wedge agent(a) = agt, s) \quad and$$

$$Trans(\mathbf{Delib}(agt, \alpha; \delta), s, \delta', s') \equiv s' = do(\alpha[s], s) \wedge \delta' = \mathbf{Delib}(agt, nil; \delta) \wedge \\ \exists a \, \mathbf{Know}(agt, \alpha = a \wedge Poss(a, now) \wedge agent(a) = agt \\ \wedge \mathbf{KnowHowDelib}(agt, [nil; \delta], do(a, now)), s).$$

That is, with **Delib**, the agent must not only know what transition/action to perform next, but also know that he will know how to complete the execution of what remains of the program after that transition. Clearly, **Delib** puts much stronger requirements on the agent than **Subj**. The fact that we can count on an agent that deliberates to choose his transitions wisely means that for a nondeterministic program to be epistemically feasible, we no longer need to require that all executions lead to successful termination; it is sufficient that the agent know that no matter how his sensing actions turn out, he will be able to get to some final configuration of the program.

The deliberative execution mode modeled by **Delib** is similar to that used by the IndiGolog agent programming language [5] for programs that are enclosed in a "search block". It is also similar to a notion of ability called $\mathbf{Can}_\perp$ that is formalized in [12]. Essentially, the agent must be able to construct a strategy (a sort of conditional plan) such that if the program is executed according to the strategy, the agent will know what action to perform at every step and be able to complete the program's execution in some bounded number of actions. In [12], we show that $\mathbf{Can}_\perp$ is not as general as one might wish and that there are programs that one would intuitively think a deliberative agent can execute for which $\mathbf{Can}_\perp$ does not hold. These are cases that involve indefinite

iteration and where the agent knows that he will eventually complete the program's execution, but cannot bound the number of actions that have to be performed. [12] proposes an account of ability that also handles these cases. However, the type of deliberation required for this is hard to implement; the **Delib** account corresponds more closely to that implemented in IndiGolog [5].

For the multiagent case, we don't yet have a satisfactory general formalization of epistemic feasibility for agents that deliberate. The problem is that if the processes involve true interactions and the agents' choice of actions must depend on that of other agents, then each agent has to model the other agents' deliberation. Moreover, one must take into account whether the agents are cooperative, competitive, or indifferent. A simple example of this kind of system is one where $Robbie$ wants to open the safe and since it does not the combination is when it starts to deliberate, plans to ask $Smartie$ for it, expecting to get an answer. A mechanism to support a simple version of this kind of deliberation in IndiGolog has been proposed in [14] (where the deliberating agent models other agents as deterministic processes). We would like to extend our formalization to handle this and other cases.

5 Conclusion

In this paper, we have dealt with the problem of ensuring that agents' plans are epistemically feasible in formal specifications of multiagent systems. The problem is tied with that of capturing an adequate notion of agenthood where an agent's choice of actions is determined by its local state. The paper deals with this in the context of the CASL specification language, but the problem arises in other frameworks based on specifying multiagent systems as a set of concurrent processes. We have proposed an account of *subjective plan execution* (**Subj**) that ensures that the plan is executed in terms of what the agent's knows rather than in terms of is true in the world. This account assumes that the agent does not deliberate or do lookahead as it executes its plan. We have also proposed an account of *deliberative plan execution* (**Delib**) for smarter agents that do planning/lookahead. Finally, in terms of these, we have developed two formalizations of *epistemic feasiblilty:* one that captures whether any set of multiagent processes is epistemically feasible when the agents do not deliberate/do lookahead (**KnowHowSubj**), and another that captures whether a process involving a single agent is epistemically feasible where the agent does deliberate/do lookahead (**KnowHowDelib**). The case of multiagent processes where the agents deliberate remains open. Our formalization shows how an account of epistemic feasibility can be integrated with a transition-system semantics for an agent programming/specification language.

Let's examine where other multiagent systems specifications frameworks stand with respect to this problem of ensuring that plans are epistemically feasible. In van Eijk *et al.*'s MAS specification framework [29], which is inspired from standard concurrent systems programming formalisms and adds an account of agents' belief states, programs are executed in a subjective manner. Tests are evaluated in terms of the executing agent's beliefs and primitive actions are treated as belief update operations. However, there is no representation of the agents' external environment and it is not possible to model agents' interactions with their external environment (e.g., to talk about the re-

liability of their sensors or effectors). One can of course represent the environment as an agent, but this is not completely satisfactory; for instance, the world is always complete and never wrong. As well, there is no account of deliberative execution; agents are assumed not to do any lookahead in executing their plans.

The MAS specification framework of Engelfriet *et al.* [7] is a modal logic with temporal and epistemic modalities. It is harder to specify systems with complex behaviors in this type of formalism than in procedural languages like CASL or [29]. If agents are specified according to the proposed methodology, their choice of actions will depend only on their local state. But as in [29], agents don't do any lookahead and there is no account of deliberative execution. Engelfriet *et al.* [7] describe a compositional verification methodology based on their framework.

None of these papers really discuss the conditions under which plans are epistemically feasible. In the case of deterministic single-agent processes, epistemic feasibility reduces to the existence of a successful subjective execution, so both frameworks can be viewed as handling the problem. But there is no treatment for other cases. Neither framework supports the objective execution of processes (the default in CASL).

The work described in this paper is ongoing. In particular, we still need to make a more systematic comparison with earlier work on epistemic feasibility and to develop adequate definitions of epistemic feasibility for the cases of multiagent systems where agents deliberate. We also intend to use our account to produce a more accurate formal semantics for the IndiGolog agent implementation language [5, 14], accounting for its on-line execution mechanism, where sensing, planning/deliberation, and plan execution are interleaved. We would also like examine how our account of subjective/deliberative execution can be adapted to the more general treatment of epistemic attitudes of [27], which allows false beliefs and supports their revision. This would allow us to model cases where the agent executes a process in a way that it thinks is correct, but that may in fact be incorrect. Cases of uninformed or insincere communication could also be handled.

Acknowledgments

I have greatly benefited from discussions on this topic with Hector Levesque, Giuseppe De Giacomo, Steven Shapiro, and David Tremaine. Comments from one of the referees were also very helpful. This research was funded by Communications and Information Technology Ontario and the Natural Science and Engineering Research Council of Canada.

References

1. F. Brazier, B. Dunin-Keplicz, N.R. Jennings, and Jan Treur. Formal specifications of multi-agents systems: A real-world case study. In *Proceedings of the First International Conference on Multi-Agent Systems (ICMAS'95)*, pages 25–32, San Francisco, CA, June 1995. Springer-Verlag.
2. E. Clarke, E.A. Emerson, and A.P. Sistla. Automatic verification of finite-state concurrent systems using temporal-logic specifications. *ACM Trans. Programming, Languages, and Systems*, 8(2):244–263, 1986.

3. Ernest Davis. Knowledge preconditions for plans. *Journal of Logic and Computation*, 4(5):721–766, 1994.
4. Giuseppe De Giacomo, Yves Lespérance, and Hector J. Levesque. ConGolog, a concurrent programming language based on the situation calculus. *Artificial Intelligence*, 121:109–169, 2000.
5. Giuseppe De Giacomo and Hector J. Levesque. An incremental interpreter for high-level programs with sensing. In Hector J. Levesque and Fiora Pirri, editors, *Logical Foundations for Cognitive Agents*, pages 86–102. Springer-Verlag, Berlin, Germany, 1999.
6. Giuseppe De Giacomo, Eugenia Ternovskaia, and Ray Reiter. Non-terminating processes in the situation calculus. In *Working Notes of the AAAI'97 Workshop on Robots, Softbots, Immobots: Theories of Action, Planning and Control*, 1997.
7. Joeri Engelfriet, Catholijn M. Jonker, and Jan Treur. Compositional verification of multi-agent systems in temporal multi-epistemic logic. In J.P. Mueller, M.P. Singh, and A.S. Rao, editors, *Intelligent Agents V: Proceedings of the Fifth International Workshop on Agent Theories, Architectures and languages (ATAL'98)*, volume 1555 of *LNAI*, pages 177–194. Springer-Verlag, 1999.
8. M. Fisher and M. Wooldridge. On the formal specification and verification of multi-agent systems. *International Journal of Cooperative Information Systems*, 6(1):37–65, 1997.
9. M. Hennessy. *The Semantics of Programming Languages*. John Wiley & Sons, 1990.
10. Gerhard Lakemeyer and Hector 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.
11. Yves Lespérance, Todd G. Kelley, John Mylopoulos, and Eric S.K. Yu. Modeling dynamic domains with ConGolog. In *Advanced Information Systems Engineering, 11th International Conference, CAiSE-99, Proceedings*, pages 365–380, Heidelberg, Germany, June 1999. LNCS 1626, Springer-Verlag.
12. Yves Lespérance, Hector J. Levesque, Fangzhen Lin, and Richard B. Scherl. Ability and knowing how in the situation calculus. *Studia Logica*, 66(1):165–186, October 2000.
13. Yves Lespérance, Hector J. Levesque, and Raymond Reiter. A situation calculus approach to modeling and programming agents. In A. Rao and M. Wooldridge, editors, *Foundations of Rational Agency*, pages 275–299. Kluwer, 1999.
14. Yves Lespérance and Ho-Kong Ng. Integrating planning into reactive high-level robot programs. In *Proceedings of the Second International Cognitive Robotics Workshop*, pages 49–54, Berlin, Germany, August 2000.
15. Hector 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, August 1996.
16. Fangzhen Lin and Hector J. Levesque. What robots can do: Robot programs and effective achievability. *Artificial Intelligence*, 101(1–2):201–226, 1998.
17. John McCarthy and Patrick Hayes. Some philosophical problems from the standpoint of artificial intelligence. In B. Meltzer and D. Michie, editors, *Machine Intelligence*, volume 4, pages 463–502. Edinburgh University Press, Edinburgh, UK, 1979.
18. Robert C. Moore. A formal theory of knowledge and action. In J. R. Hobbs and Robert C. Moore, editors, *Formal Theories of the Common Sense World*, pages 319–358. Ablex Publishing, Norwood, NJ, 1985.
19. Leora Morgenstern. Knowledge preconditions for actions and plans. In *Proceedings of the Tenth International Joint Conference on Artificial Intelligence*, pages 867–874, Milan, Italy, August 1987. Morgan Kaufmann Publishing.
20. D. Park. Fixpoint induction and proofs of program properties. In *Machine Intelligence*, volume 5, pages 59–78. Edinburgh University Press, 1970.

21. G. Plotkin. A structural approach to operational semantics. Technical Report DAIMI-FN-19, Computer Science Dept., Aarhus University, Denmark, 1981.
22. Raymond Reiter. The frame problem in the situation calculus: A simple solution (sometimes) and a completeness result for goal regression. In Vladimir Lifschitz, editor, *Artificial Intelligence and Mathematical Theory of Computation: Papers in Honor of John McCarthy*, pages 359–380. Academic Press, San Diego, CA, 1991.
23. Raymond Reiter. *Knowledge in Action: Logical Foundations for Specifying and Implementing Dynamical Systems.* MIT Press, 2001.
24. Richard B. Scherl and Hector J. Levesque. The frame problem and knowledge-producing actions. In *Proceedings of the Eleventh National Conference on Artificial Intelligence*, pages 689–695, Washington, DC, July 1993. AAAI Press/The MIT Press.
25. Steven Shapiro and Yves Lespérance. Modeling multiagent systems with CASL — a feature interaction resolution application. In C. Castelfranchi and Y. Lespérance, editors, *Intelligent Agents VII. Agent Theories, Architectures, and Languages — 7th. International Workshop, ATAL-2000, Boston, MA, USA, July 7–9, 2000, Proceedings*, volume 1986 of *LNAI*, pages 244–259. Springer-Verlag, Berlin, 2001.
26. Steven Shapiro, Yves Lespérance, and Hector 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, November 1997.
27. Steven Shapiro, Maurice Pagnucco, Yves Lespérance, and Hector 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-2000)*, pages 527–538. Morgan Kaufmann, 2000.
28. W. van der Hoek, B. van Linder, and J.-J. Ch. Meyer. A logic of capabilities. In A. Nerode and Yu. V. Matiyasevich, editors, *Proceedings of the Third International Symposium on the Logical Foundations of Computer Science (LFCS'94)*. LNCS Vol. 813, Springer-Verlag, 1994.
29. Rogier M. van Eijk, Frank S. de Boer, Wiebe van der Hoek, and John-Jules Ch. Meyer. Open multi-agent systems: Agent communication and integration. In N.R. Jennings and Y. Lespérance, editors, *Intelligent Agents VI — Proceedings of the Sixth International Workshop on Agent Theories, Architectures, and Languages (ATAL-99)*, Lecture Notes in Artificial Intelligence, pages 218–232. Springer-Verlag, Berlin, 2000.