CHAPITRE PREMIER

# De l'exécutabilité épistémique des plans dans les spécifications de systèmes multi-agents

Yves Lespérance

Department of Computer Science, York University
Toronto, ON, Canada M3J 1P3
lesperan@cs.yorku.ca
http: //www.cs.yorku.ca/∼lesperan

**Résumé**

Cet article concerne le problème de garantir que les plans des agents sont épistémiquement exécutables dans les spécifications de systèmes multi-agents. Nous proposons des solutions dans le cadre du formalisme « Cognitive Agents Specification Language » (CASL). Nous définissions un opérateur d'exécution subjective de plan **Subj** qui fait que le plan est exécuté en fonction des connaissances de l'agent et pas simplement de l'état du monde. La définition suppose que l'agent ne fait pas de planification/anticipation et choisit arbitrairement parmi les actions permises par le plan. Nous définissions aussi un autre opérateur d'exécution délibérative **Delib** pour les agents plus intelligents qui font de la planification et anticipent. Nous montrons comment ces notions permettent d'exprimer le fait qu'un processus est épistémiquement exécutable pour les agents impliqués dans plusieurs types de situations. Plus généralement, l'article montre comment une formalisation de l'exécutabilité épistémique peut être intégrée à une sémantique « système de transitions » pour un langage de programmation/spécification d'agent.

**Mots-clés** : Raisonnement sur la connaissance et l'action, systèmes multi-agents, méthodes formelles

## 1 INTRODUCTION

Ces dernières années, plusieurs cadres formels ont été proposés pour supporter la spécification et la vérification de systèmes multi-agents (SMA)[1, 8, 9,

32]. Nous avons participé au développement d'un tel cadre formel, le « Cognitive Agents Specification Language » (CASL) [27]. CASL s'inspire à la fois des travaux en théorie des agents (logiques croyances-désirs-intentions) et en méthodes formelles de génie logiciel, pour produire un langage de spécification expressif qui peut être utilisé pour modéliser et vérifier des SMAs complexes.

Un problème pour CASL et certains autres formalismes de spécification des SMAs est qu'ils ne fournissent pas de mécanismes pour garantir que les plans des agents sont épistémiquement exécutables, c.-à-d. que les agents ont les connaissances nécessaires pour être capables d'exécuter leurs plans. Dans une véritable situation multi-agent, le comportement de chaque agent est déterminé par ses propres attitudes mentales, ses connaissances, buts, etc. A chaque instant, les agents doivent sélectionner l'action qu'ils exécuteront en fonction de leurs plans et des connaissances qu'ils ont sur l'état du système. Or, il se trouve que dans CASL (et certains autres formalismes), le comportement du système est spécifié simplement comme un ensemble de processus concurrents. Ces processus peuvent faire référence aux états mentaux des agents (CASL fournit des opérateurs qui modélisent leurs connaissances et leurs buts), mais rien de cela n'est requis. L'analyste n'est pas contraint de spécifier quel agent exécute un processus donné et de s'assurer que cet agent a les connaissances requises pour l'exécuter.

Examinons l'exemple suivant, adapté de Moore [20]. Nous avons un agent, $Robie$, qui veut ouvrir un coffre-fort, mais qui ne connaît pas la combinaison de ce coffre. Il y a aussi un second agent, $Futé$, qui sait quelle est la combinaison du coffre. Si nous prenons la spécification en CASL du système comme étant simplement l'action primitive:

$$compose(Robie, combinaison(Coffre1), Coffre1),$$

c.-à-d. que $Robie$ compose la combinaison du coffre et que $Futé$ ne fait rien, alors nous sommes en face d'un processus qui est physiquement exécutable et doit se terminer dans une situation où le coffre est ouvert.[1] Ceci est le cas en supposant qu'une spécification des effets et préconditions (physiques)

[1] Formellement:

$$\exists s\, Do(compose(Robie, combinaison(Coffre1), Coffre1), S_0, s) \wedge \\ \forall s(Do(compose(Robie, combinaison(Coffre1), Coffre1), S_0, s) \supset \\ Ouvert(Coffre1, s)).$$

La notation est expliquée à la section 3.

de l'action $compose$ et de la situation initiale a été donnée (comme nous le faisons à la section 3). Toutefois, ce processus n'est pas épistémiquement exécutable car $Robie$ ne connaît pas la combinaison du coffre.[2] Une telle spécification peut être adéquate si tout ce qu'on veut est d'identifier un ensemble d'exécutions pour le système. Mais ce type de spécification ne capture pas le contrôle interne des agents, comment leur comportement est déterminé par leurs états mentaux.[3]

Si on veut s'assurer que la spécification est épistémiquement exécutable, alors on devrait plutôt donner quelque chose comme suit:

$$
\begin{array}{l}
(\textbf{KRef}(Robie, combinaison(Coffre1))?; \\
compose(Robie, combinaison(Coffre1), Coffre1)) \\
\| \\
informRef(Futé, Robie, combinaison(Coffre1)).
\end{array}
$$

Ici, dans le premier processus concurrent, $Robie$ attends de savoir quelle est la combinaison du coffre, puis la compose, et dans le second processus concurrent, $Futé$ informe $Robie$ de la combinaison. $\delta_1 \| \delta_2$ représente l'exécution concurrente de $\delta_1$ et $\delta_2$. On pourrait aussi vouloir s'assurer que chaque agent *sait* que les préconditions physiques de ses actions sont satisfaites lorsqu'il s'apprête à les accomplir, par exemple, que $Robie$ sait qu'il est physiquement possible pour lui de composer une combinaison sur un coffre-fort. Ce type de préconditions épistémiques a déjà été étudié en théorie des agents sous les termes « knowledge prerequisites of action », « knowing how to execute a program », « ability to achieve a goal » et « epistemic feasibility » [20, 21, 31, 3, 14, 17, 18]. L'analyste pourrait inclure explicitement toutes ces préconditions épistémiques dans la spécification du processus. Mais il serait préférable de pouvoir dire simplement que le premier processus sera ***exécuté subjectivement*** par $Robie$ et le second par $Futé$, et que toutes les

[2] $combinaison(Coffre1)$ est un fluent dont la valeur varie selon les alternatives épistémiques de l'agent; on peut rendre explicite l'argument situation en écrivant $combinaison(Coffre1, now)$; voir la section 3.

[3] Le fait qu'en CASL, les processus du système sont spécifiés du point de vue d'un observateur externe peut avoir des avantages. Dans bien des cas, les programmes de contrôle internes des agents ne sont pas connus. Parfois, l'analyste ne veut qu'un modèle partiel du système capturant certains scénarios d'intérêt. Dans le cas d'agents purement réactifs, l'analyste peut ne pas vouloir attribuer d'attitudes mentales aux agents. Cependant, cette correspondance très libre entre la spécification et le système veut dire qu'il est facile de produire des spécifications qui ne peuvent être exécutées par les agents. Souvent, on veut s'assurer que les spécifications sont épistémiquement exécutables.

préconditions épistémiques en découlent automatiquement; quelque chose comme:

$$
\begin{array}{l}
syst\grave{e}me1 \stackrel{\text{def}}{=} \\
\textbf{Subj}(Robie, \textbf{KRef}(Robie, combinaison(Coffre1))?; \\
\qquad\qquad compose(Robie, combinaison(Coffre1), Coffre1)) \parallel \\
\textbf{Subj}(Futé, informRef(Futé, Robie, combinaison(Coffre1))).
\end{array}
$$

Dans cette spécification que nous appelons $syst\grave{e}me1$, nous utilisons un nouvel opérateur **Subj**$(agt, \delta)$ qui veut dire que le processus $\delta$ est exécuté subjectivement par l'agent $agt$, c.-à-d. que $\delta$ est exécuté par $agt$ en fonction de son état de connaissances. Notons que nous aurions pu rendre l'exemple plus réaliste en spécifiant que $Robie$ fait une requête à $Futé$ pour qu'il l'informe de la combinaison du coffre et que $Futé$ répond à cette requête, comme dans les exemples de [27]; mais ici, nous préférons garder l'exemple simple. Nous y reviendrons à la section 3.

Dans cet article, nous explorerons ces questions, et proposerons un traitement de l'exécution subjective de plans dans le formalisme CASL qui garantit que le plan est exécute par l'agent sur la base de ses états mentaux. Notre traitement de l'exécution subjective (**Subj**) supposera que l'agent ne fait pas de planification et n'essaie pas d'anticiper (« lookahead ») durant l'exécution de son programme. Nous développerons aussi un traitement de l'*exécution délibérative* de plans (**Delib**) applicable à des agent plus intelligents qui font de la planification et anticipent. A partir de ces notions, nous proposerons des formalisations de l'exécutabilité épistémique des plan applicables à plusieurs cas de SMAs. Les notions formalisées sont définies sur la base de la sémantique « système de transitions » de CASL. En fait, une des contributions de l'article est de montrer comment une formalisation de l'exécutabilité épistémique peut être adaptée pour utilisation avec un langage de programmation/spécification d'agent qui a une telle sémantique.

## 2 APERÇU DE CASL

Le « Cognitive Agents Specification Language » (CASL) [27] est un formalisme de spécification pour les systèmes multi-agents (SMA). Il joint une théorie de l'action [24] et des états mentaux [26] basée sur le calcul des situations [19] à ConGolog [4], un langage de programmation concurrente et non-déterministe doté d'une sémantique formelle. Le résultat est un langage

de spécification qui fournit une riche gamme de structures pour faciliter la spécification de *SMAs complexes*. Une spécification d'un système en CASL comprends deux composantes: une spécification de la dynamique du domaine et une spécification du comportement des agents du système. Voyons maintenant comment ces composantes sont modélisées.

## 2.1 Modélisation de la dynamique du domaine

La composante *dynamique du domaine* d'une spécification CASL formalise les propriétés et relations qui sont employés pour modéliser l'état du système, les actions qui peuvent être accomplies par les agents et leurs préconditions et effets, et ce que l'on sait de la situation initiale. Le modèle peut inclure une spécification des états mentaux des agents, c.-à-d. de leurs connaissances et de leurs buts, ainsi que de la dynamique de ces états mentaux, c.-à-d. de comment ils sont affectés par les actions de communication et les actions de perception. Cette composante est spécifiée de façon purement déclarative dans le calcul des situations [19].

Très brièvement, le calcul des situations est un langage de la logique des prédicats qui sert à représenter des mondes dynamiques. Dans ce langage, une histoire possible du monde (une séquence d'actions) est représentée par un terme appartenant à la sorte *situation*. La constante $S_0$ dénote la situation initiale et le terme $do(\alpha, s)$ dénote la situation qui résulte de l'accomplissement de l'action $\alpha$ dans la situation $s$. Les relations (fonctions) qui varient d'une situation à l'autre, appelées *fluents*, sont représentées par des symboles de prédicats (fonctions) qui prennent un terme situation comme dernier argument. Le prédicat spécial $Poss(\alpha, s)$ est utilisé pour dire qu'une action primitive $\alpha$ est physiquement possible dans la situation $s$.

Nous employons la solution de Reiter au problème du cadre (« frame problem »), où les axiomes d'effets sont compilés en axiomes d'état successeur [24, 25]. Ainsi, pour notre exemple, la spécification de la dynamique du domaine inclut l'axiome d'état successeur:

$$\begin{aligned} &Ouvert(x, do(a, s)) \equiv \\ &\quad \exists agt, c(a = compose(agt, c, x) \wedge c = combinaison(x, s)) \\ &\quad \vee Ouvert(x, s), \end{aligned}$$

c.-à-d. que le coffre $x$ est ouvert dans la situation qui résulte de l'accomplissement de l'action $a$ dans la situation $s$ si et seulement si $a$ est l'action de composer la bonne combinaison sur le coffre $x$ ou bien si $x$ était déjà ouvert

dans la situation $s$. Nous avons aussi un axiome d'état successeur pour le fluent $combinaison$ dont la valeur n'est affectée par aucune action:

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

La spécification inclut aussi l'axiome de préconditions:

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

c.-à-d. que l'action $compose$ est toujours physiquement possible. Nous spécifions aussi l'agent de l'action par:

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

La connaissance est représentée en adaptant la sémantique des mondes possibles au calcul des situations [20, 26]. La relation d'accessibilité $K(agt, s', s)$ représente le fait que dans la situation $s$, l'agent $agt$ pense que le monde pourrait être dans la situation $s'$. Un agent sait que $\phi$ dans la situation $s$, **Know**$(agt, \phi, s)$, si et seulement si $\phi$ est vrai dans toutes les situations $s'$ qui lui sont accessibles via $K$ à partir de la situation $s$:

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

Ici, $\phi[s]$ représente la formule obtenue en substituant $s$ pour toutes les instances libres (c.-à-d. en dehors de la portée d'un autre **Know**) de la constante spéciale $now$; ainsi par exemple, **Know**$(agt, Ouvert(Coffre1, now), s)$ est une abréviation de $\forall s'(K(agt, s', s) \supset Ouvert(Coffre1, s'))$. Souvent, lorsque cela ne prête pas à confusion, nous supprimons $now$ et par exemple, écrivons simplement **Know**$(agt, Ouvert(Coffre1), s)$. Nous supposons que la relation $K$ est réflexive, transitive, et euclidienne, ce qui garantit que ce qui est connu est vrai et que les agents savent toujours s'ils savent quelque chose (introspection positive et négative). Nous employons aussi les abréviations suivantes:

$$\textbf{KWhether}(agt, \phi, s) \stackrel{\text{def}}{=} \textbf{Know}(agt, \phi, s) \vee \textbf{Know}(agt, \neg\phi, s),$$

qui signifie qu'$agt$ sait si $\phi$ est vrai dans $s$ et

$$\textbf{KRef}(agt, \theta, s) \stackrel{\text{def}}{=} \exists t\, \textbf{Know}(agt, t = \theta, s),$$

qui veut dire qu'$agt$ sait qui est $\theta$ ou à quoi réfère le terme $\theta$ dans $s$.

Dans cet article, nous traitons les types suivants d'actions productrices de nouvelles connaissances: les actions de perception binaires, ex. $sense_{Ouvert(Coffre1)}(agt)$, où l'agent perçoit la valeur de vérité de la proposition associée, les actions de perception non-binaires, ex. $read_{combinaison(Coffre1)}(agt)$, où l'agent perçoit la valeur du terme associé, et deux actions de communication génériques, $informWhether(agt_1, agt_2, \phi)$ où l'agent $agt_1$ informe l'agent $agt_2$ de la valeur de vérité de la proposition $\phi$, et $informRef(agt_1, agt_2, \theta)$, où l'agent $agt_1$ informe l'agent $agt_2$ de la valeur du terme $\theta$.[4] Suivant l'approche de [17], l'information fournie par une action de perception binaire est spécifiée au moyen du prédicat $SF(a, s)$, qui est vrai si l'action $a$ renvoie l'observation 1 (vrai) dans la situation $s$. Par exemple, on pourrait avoir l'axiome:

$$SF(sense_{Ouvert(Coffre1)}(agt), s) \equiv Ouvert(Coffre1, s),$$

c.-à-d. que l'action $sense_{Ouvert(Coffre1)}(agt)$ dira à $agt$ si $Coffre1$ est ouvert dans la situation où elle est accomplie. De façon analogue, pour les actions de perception non-binaires, nous employons le terme $sff(a, s)$ pour dénoter l'observation renvoyée par l'action; par exemple, on pourrait avoir:

$$sff(read_{combinaison(Coffre1)}(agt), s) = combinaison(Coffre1, s),$$

c.-à-d. que $read_{combinaison(Coffre1)}(agt)$ dit à $agt$ la valeur de la combinaison de $Coffre1$.

Nous spécifions la dynamique des connaissances avec l'axiome d'état successeur suivant:

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

[4] Comme l'action $informWhether$ prend pour argument une formule, il est nécessaire d'encoder les formules en termes; voir [4] pour les détails. Pour simplifier la notation, nous laissons l'encodage implicite et utilisons les formules directement comme des termes.

Ainsi, après qu'une action ait été faite, tous les agents savent que cette action a été faite. De plus, si cette action est une action de perception, alors l'agent qui l'accomplit apprends la valeur de la proposition ou du terme associé. Si par contre l'action est que quelqu'un informe $agt$ de la valeur de vérité de $\phi$, alors $agt$ sait si $\phi$ est vrai dans la situation résultante, et si l'action est que quelqu'un informe $agt$ de la valeur du terme $\theta$, alors $agt$ connaît cette information dans la situation résultante ($\theta[s]$ représente $\theta$ avec $s$ substitué à $now$ comme pour $\phi[s]$). Les préconditions des actions de communications sont définies par les axiomes suivants:

$$Poss(informWhether(informateur, agt, \phi), s) \equiv \\ \textbf{KWhether}(informateur, \phi, s),$$

c.-à-d. que $informWhether$ est possible dans la situation $s$ si et seulement si $informateur$ sait si $\phi$ est vrai, et

$$Poss(informRef(informateur, agt, \theta), s) \equiv \textbf{KRef}(informateur, \theta, s),$$

c.-à-d. que $informRef$ est possible dans la situation $s$ si et seulement si $informateur$ connaît la valeur de $\theta$. Il y a aussi des axiomes qui disent qu'$informateur$ est l'agent de ces actions. Les buts et les requêtes sont aussi modélisés en CASL de façon analogue; voir [27] pour les détails.

Ainsi, la dynamique du domaine est spécifiée en CASL au moyen d'une théorie de l'action contenant des axiomes des types suivants:

- des axiomes d'état initial, qui décrivent l'état initial du domaine et les états mentaux initiaux des agents;
- des axiomes de préconditions d'action, un pour chaque action, qui caractérisent $Poss$;
- des axiomes d'état successeur, un pour chaque fluent;
- des axiomes qui spécifient quelles actions sont des actions de perception et quels fluents elles observent, caractérisant $SF$ et $sff$;
- des axiomes qui spécifient l'agent de chaque action;
- des axiomes de noms uniques (« unique names axioms ») pour les actions;
- certains axiomes fondationnels indépendants du domaine semblables à ceux de [11, 25].

## 2.2 Modélisation du comportement des agents

La seconde composante d'un modèle CASL est une spécification du *comportement des agents* du système. Comme nous voulons pouvoir modéliser des SMAs comportant des processus complexes, cette composante est spécifiée de façon procédurale. Pour cela, nous employons le langage de programmation/spécification de processus ConGolog [4], qui fournit le riche ensemble de structures suivantes:

| | |
|---|---|
| $\alpha$, | action primitive |
| $\phi?$ , | test/attente d'une condition $\phi$ |
| $\delta_1; \delta_2$, | séquence |
| $\delta_1 \mid \delta_2$, | branchement non-déterministe |
| $\pi\, x\, \delta$, | choix d'argument non-déterministe |
| $\delta^*$, | itération non-déterministe |
| **if** $\phi$ **then** $\delta_1$ **else** $\delta_2$ **endIf**, | conditionnelle |
| **while** $\phi$ **do** $\delta$ **endWhile**, | boucle while |
| $\delta_1 \parallel \delta_2$, | exécution concurrente avec priorité égale |
| $\delta_1 \rangle\!\rangle \delta_2$, | exécution concurrente avec $\delta_1$ à priorité plus élevée |
| $\delta^{\parallel}$, | itération concurrente |
| $\langle\, \vec{x} : \phi \rightarrow \delta \,\rangle$, | interruption |
| $p(\vec{\theta})$, | appel de procédure. |

La sémantique du langage ConGolog [4] est définie en termes de *transitions*, dans le style de la sémantique opérationnelle structurale [23, 10]. Une transition est une unité atomique d'exécution de programme, soit l'exécution d'une action primitive, soit l'exécution d'une action de test $\phi?$ dans la situation courante. La sémantique emploie deux prédicats spéciaux, $Final$ et $Trans$. $Final(\delta, s)$ signifie que le programme $\delta$ peut légalement se terminer dans la situation $s$ et $Trans(\delta, s, \delta', s')$ veut dire que le programme $\delta$ dans la situation $s$ peut légalement exécuter une opération atomique, pour arriver dans la situation $s'$ avec le programme $\delta'$ restant à exécuter. $Trans$ et $Final$ sont caractérisés par un ensemble d'axiomes dont les suivants:

$$Trans(\alpha, s, \delta, s') \equiv \quad \text{action primitive}$$
$$Poss(\alpha[s], s) \wedge \delta = nil \wedge s' = do(\alpha[s], s),$$
$$Final(\alpha, s) \equiv False,$$

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

Le premier axiome dit qu'un programme constitué d'une action primitive $\alpha$ peut accomplir une transition dans la situation $s$ à condition que l'action $\alpha$ soit possible dans $s$, que la situation résultante soit $do(\alpha[s], s)$ et que le programme restant soit le programme vide $nil$. Le second axiome dit qu'un programme où il reste une action primitive ne peut jamais être considéré comme terminé. Le troisième axiome dit qu'on peut accomplir une transition pour une séquence en accomplissant une transition pour la première partie ou bien une transition pour la deuxième partie lorsque la première partie a déjà terminé. Le dernier axiome dit qu'une séquence a terminé quand ses deux parties ont terminé.[5]

Les axiomes pour les autres structures ConGolog que nous utiliserons sont les suivants:

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

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

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

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

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

[5] Notez que nous employons des axiomes plutôt que des règles pour spécifier $Trans$ et $Final$ car nous voulons supporter le raisonnement sur les exécutions possibles d'un système même en présence de connaissances incomplètes de son état initial. Notez aussi que puisque ces prédicats prennent des programmes (qui contiennent des tests de formules) comme arguments, il est nécessaire ici aussi d'encoder les formules et programmes en tant que termes; voir [4] pour les détails. Ici, nous laissons l'encodage implicite et utilisons les programmes directement comme des termes.

Notez que pour traiter les procédures récursives, une formalisation plus complexe est nécessaire; voir [4] pour les détails.

La sémantique globale d'un programme ConGolog est spécifiée par la relation $Do$:

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

où $Trans^*$ est la fermeture réflexive transitive de la relation $Trans$. Ainsi, on a $Do(\delta, s, s')$ si et seulement si $s'$ est une situation où le processus peut légalement se terminer après avoir démarré dans la situation $s$, c.-à-d. où $s'$ est une situation qui peut être atteinte en accomplissant une séquence de transitions en partant de la situation $s$ avec le programme $\delta$, et où le programme peut légalement se terminer.[6]

L'approche de CASL vise un compromis entre une spécification purement intentionnelle des agents (c.-à-d. basée sur leurs attitudes mentales, comme dans les logiques croyances-désirs-intentions), qui ne permet pas de faire de prédictions précises sur l'évolution du système, et le type de spécification de systèmes concurrents habituel, qui est de trop bas niveau et ne modélise pas les états mentaux. En raison de ses fondements logiques, CASL peut accommoder les modèles incomplètement spécifiés, soit parce que l'état initial n'est pas complètement spécifié, soit parce que les processus impliqués sont non-déterministes et peuvent évoluer de plusieurs façons. L'approche supporte la vérification, de même que la simulation lorsque la spécification de l'état initial est suffisamment complète.

La version la plus récente de CASL, qui supporte la communication privée avec des actes de langages chiffrés et un traitement plus simple des buts est décrite dans [27]. Cet article montre aussi comment CASL a été employé pour modéliser un système multi-agent relativement complexe qui résout les interactions de services en télécommunications; ce système est constitué d'agents autonomes ayant des buts explicites qui entrent en négociation. Des versions un peu plus anciennes de CASL sont décrites en détail dans [28,

[6] Pour définir la relation $Trans^*$ correctement, nous utilisons la logique du second ordre:

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

Pour le raisonnement automatique on pourrait employer une version du premier ordre pour prouver certaines conséquences (mais pas toutes) de la théorie.

15, 13], où l'utilisation du formalisme est illustrée par un exemple simple de SMA d'organisation de réunions. Une discussion de l'emploi de CASL pour la modélisation des processus et l'analyse de besoins a paru dans [12]; cet article décrit aussi des outils pour la simulation et la vérification qui sont en cours de développement. Dans [29], l'outil pour la vérification est décrit de façon plus détaillée.

## 3 L'EXÉCUTION SUBJECTIVE

Nous définissons l'opérateur d'exécution subjective $\mathbf{Subj}(agt, \delta)$ introduit à la section 1 comme suit:

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

Ceci signifie que lorsqu'un programme est exécuté subjectivement, le système ne peut faire de transition que si l'agent sait qu'il peut faire cette transition et si la transition implique une action primitive, alors cette action est accomplie par l'agent lui-même. Une exécution subjective ne peut légalement se terminer que si l'agent sait qu'elle peut légalement se terminer.

Retournons maintenant aux exemples de la section 1. Supposons que dans la situation initiale $S_0$, $Robie$ ne connaît pas la combinaison du coffre, mais que $Futé$ lui la connaît; formellement:

$$\begin{aligned} &\neg \mathbf{KRef}(Robie, combinaison(Coffre1), S_0) \wedge \\ &\mathbf{KRef}(Futé, combinaison(Coffre1), S_0). \end{aligned}$$

Il est facile de montrer que dans cette situation, $Robie$ ne peut pas ouvrir le coffre par lui-même, et que le programme où il ne fait que composer la combinaison n'est pas subjectivement exécutable:

$$\begin{aligned} \neg \exists s Do(&\mathbf{Subj}(Robie, compose(Robie, combinaison(Coffre1), \\ &Coffre1)), S_0, s). \end{aligned}$$

Pour bien comprendre ce résultat, commençons par observer que le fait que $Robie$ ne connaisse pas la combinaison du coffre initialement veut dire que

l'on a $\neg\exists c \forall s(K(Robie, s, S_0) \supset combinaison(Coffre1, s) = c)$. Ainsi, il y a des situations $K$-accessibles à $Robie$ dans $S_0$, disons $s_1$ et $s_2$, où $combinaison(Coffre1, s_1) \neq combinaison(Coffre1, s_2)$. Il suit par les axiomes de noms uniques pour les actions que la composition de la combinaison du coffre par $Robie$ dénote des actions différentes dans ces deux situations, c.-à-d. que $compose(Robie, combinaison(Coffre1, s_1), Coffre1) \neq compose(Robie, combinaison(Coffre1, s_2), Coffre1)$, et donc que $Robie$ ne sait pas quelle action ce terme dénote dans $S_0$, c.-à-d.

$$\neg\exists a \, \mathbf{Know}(Robie, compose(Robie, combinaison(Coffre1, now), Coffre1), S_0).$$

Donc, $Robie$ ne sait pas quelle transition il peut faire dans $S_0$, c.-à-d.

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

où $CC \stackrel{\text{def}}{=} compose(Robbie, combinaison(Coffre1, now), Coffre1)$, et il n'y a pas de transition où le programme est subjectivement exécuté, c.-à-d. $\neg\exists \delta, s \, Trans(\mathbf{Subj}(Robie, CC), S_0, \delta, s)$. Comme le programme ne peut pas se terminer légalement (c.-à-d. n'est pas $Final$) dans $S_0$, il n'est pas subjectivement exécutable.

Par contre, si $Futé$ informe $Robie$ de la combinaison, comme dans l'exemple $système1$, alors il est facile de voir que les processus impliqués sont subjectivement exécutables, c.-à-d. $\exists s Do(système1, S_0, s)$. Notez que l'on peut éliminer l'action de test $\mathbf{KRef}(Robie, combinaison(Coffre1))$? de $système1$, puisque comme on a vu, la transition pour l'action $compose$ n'est possible que si $Robie$ sait la combinaison.

La formalisation traite aussi correctement les cas d'actions de perception. Par exemple, si $Robie$ commence par lire la combinaison du coffre (supposons qu'elle est écrite sur un morceau de papier) et ensuite la compose, alors le processus obtenu est subjectivement exécutable:

$$\exists s Do(\mathbf{Subj}(Robie, [read_{combinaison(Coffre1)}(Robie); compose(Robie, combinaison(Coffre1), Coffre1)]), S_0, s).$$

Pour mieux comprendre cette notion d'exécution subjective, examinons certaines de ses propriétés. Premièrement, pour une action primitive $\alpha$, un agent ne peut l'exécuter subjectivement que s'il sait quelle est cette action (y compris les valeurs des arguments fluents comme $combinaison(Coffre1)$) et sait que son exécution est physiquement possible dans la situation courante:

**Proposition 1**

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

Deuxièmement, pour une action de test/attente impliquant une condition $\phi$, un agent ne peut l'exécuter subjectivement que s'il sait que $\phi$ est vrai dans la situation courante:

**Proposition 2**

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

Troisièmement, pour un programme « if-then-else » où les branches « then » et « else » requièrent des transitions initiales différentes, un agent peut l'exécuter subjectivement s'il sait que la condition $\phi$ est maintenant vraie et peut exécuter subjectivement la branche « then » ou bien s'il sait que $\phi$ est maintenant fausse et peut exécuter subjectivement la branche « else »:

**Proposition 3**

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

Donc, on voit comment dans l'exécution subjective, les tests et les fluents qui apparaissent dans le programme, de même que les préconditions des actions, sont tous évalués par rapport à l'état de connaissances de l'agent plutôt que par rapport à l'état du monde.

Pour les programmes n'impliquant qu'un seul agent qui sont déterministes, c.-à-d. où les structures $|$ (branchement non-déterministe), $\pi$ (choix d'argument non-déterministe), $*$ (itération non-déterministe) et $||$ (exécution concurrente) n'apparaissent pas, on peut considérer $\exists s' Do(\textbf{Subj}(agt, \delta), s, s')$ comme une formalisation adéquate de l'exécutabilité épistémique. Dans ce cas, il est suffisant que l'agent sache quelle transition accomplir à chaque

étape et sache quand il peut légalement terminer. Par contre pour les programmes non-déterministes, il est nécessaire de considérer comment l'agent choisit la transition qu'il va accomplir parmi toutes celles qui sont possibles. On peut voir **Subj** comme modélisant le comportement d'un agent qui exécute son programme de manière hardie ou aveugle. Lorsque le programme autorise plusieurs transitions différentes, l'agent choisit la prochaine transition de manière arbitraire; il ne tente pas d'explorer où ce choix peut le mener de façon à faire un bon choix. Il peut donc facilement se retrouver dans un cul-de-sac. Par exemple, prenons le programme **Subj**$(agt, (a; False?)|b)$ dans une situation où l'agent sait que les actions primitives $a$ et $b$ sont toutes deux exécutables. Alors l'agent pourrait très bien choisir d'exécuter l'action $a$ (plutôt que $b$), après quoi il lui reste le programme $False?$ à exécuter, lequel n'autorise aucune transition et ne peut légalement se terminer. Si on veut confirmer qu'un tel agent aveugle saura comment exécuter un programme non-déterministe, on doit s'assurer que tout chemin à travers le programme est subjectivement exécutable et mène à une situation finale du programme. Notez que ce mode d'exécution aveugle est celui employé par défaut dans le langage de programmation d'agent IndiGolog [6].

Donc, pour les programmes non-déterministes, l'existence d'un chemin à travers le programme qui est subjectivement exécutable n'est pas suffisante pour garantir l'exécutabilité épistémique. On doit s'assurer que tous les chemins à travers le programme sont subjectivement exécutables. Pour capturer ceci on peut commencer par définir un nouveau prédicat **AllDo**$(\delta, s)$ qui est satisfait si toutes les exécutions du programme $\delta$ à partir de la situation $s$ se terminent avec succès:

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

**AllDo**$(\delta, s)$ est satisfait si et seulement si $(\delta, s)$ est dans la plus petite relation $R$ telle que (1) si $(\delta_1, s_1)$ peut légalement terminer, alors elle est dans $R$ et (2) s'il existe une transition qui peut être faite dans $(\delta_1, s_1)$ et que toutes les transitions à partir de $(\delta_1, s_1)$ mènent à une configuration qui est aussi dans $R$, alors $(\delta_1, s_1)$ est aussi dans $R$.[7] Il est facile d'observer que si toutes les

[7] **AllDo** est similaire à l'opérateur $\mathbf{AF}\phi$ dans la logique temporelle ramifiée $CTL^*$ [2]. Des

exécutions de $\delta$ dans $s$ se terminent avec succès, alors il existe une exécution qui se termine avec succès:

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

Donc, nous formalisons la notion d'*exécutabilité épistémique* pour les programmes n'impliquant qu'un seul agent et où cet agent exécute le programme de manière aveugle par le prédicat **KnowHowSubj**$(agt, \delta, s)$, qui est défini comme suit:

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

c.-à-d. que le programme est épistémiquement exécutable si toute exécution subjective de $\delta$ par $agt$ à partir de $s$ se termine avec succès. Pour les systèmes comprenant deux agents $agt_1$ et $agt_2$ qui exécutent concurremment leurs programmes $\delta_1$ et $\delta_2$ de manière aveugle, on peut définir l'exécutabilité épistémique comme suit:

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

Comme les agents n'anticipent pas, pour garantir que le processus sera exécuté avec succès, il faut s'assurer que peu importe comment les programmes des agents sont entrelacés, et peu importe quelles transitions les agents choisissent, l'exécution se terminera avec succès. On peut généraliser la formalisation pour les processus impliquant plus de deux agents et/ou des méthodes de compositions autres que l'exécution concurrente simple. Encore une fois, si les agents exécutent tous leur programmes de manière aveugle, alors il faut s'assurer que toutes les exécutions se terminent avec succès. Donc pour un processus multi-agent $\delta$ où les agents sont tous des exécutants aveugles (les programmes des agents doivent tous être à l'intérieur d'un opérateur **Subj**), on peut définir l'exécutabilité épistémique comme suit:

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

**Subj** est très semblable à la notion de « dumb knowing how » **DKH**$(\delta, s)$ formalisée dans [14] pour les $\delta$s qui sont des programmes Golog, c.-à-d.

propriétés de processus telles que **AllDo** sont souvent spécifiées dans le $\mu$-calculus [22]; voir [7] pour une discussion dans le contexte du calcul des situations.

des programmes ConGolog sans processus concurrents ni interruptions. Pour tout programme Golog déterministe n'impliquant qu'un seul agent, nous croyons que **KnowHowSubj** et **DKH** sont essentiellement équivalents dans le sens que:

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

Nous croyons aussi que pour tout programme Golog non-déterministe n'impliquant qu'un seul agent $\delta$, nous avons que:

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

(Nous espérons prouver ces conjectures dans des travaux à venir.) Notons cependant que **KnowHowSubj** est considérablement plus général que **DKH**, car il peut être employé pour spécifier des systèmes comprenant des processus concurrents et plusieurs agents, comme dans l'exemple $système1$.

## 4 L'EXÉCUTION DÉLIBÉRATIVE

Dans la section précédente, nous avons développé un traitement de l'exécution subjective qui suppose que l'agent exécute son programme de façon aveugle sans faire de délibération/anticipation. Maintenant, nous allons proposer un autre traitement qui capture les conditions dans lesquelles un agent qui délibère est capable d'exécuter un programme. Nous utiliserons la notation $\textbf{Delib}(agt, \delta)$ pour cette notion d'*exécution délibérative*. Nous la formalisons comme suit:

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

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

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

Ceci signifie que le système ne peut faire de transition que si l'agent sait qu'il peut faire la transition et sait qu'il saura comment compléter l'exécution du programme après avoir fait cette transition. L'agent *sait comment exécuter* $\delta$ dans la situation $s$, **KnowHowDelib**$(agt, \delta, s)$, si et seulement si $(\delta, s)$ est dans la plus petite relation $R$ telle que (1) si l'agent sait que $(\delta_1, s_1)$ peut légalement se terminer, alors $(\delta_1, s_1)$ est dans $R$, et (2) si l'agent sait qu'il peut faire une transition (avec ou sans action) dans $(\delta_1, s_1)$ pour arriver à une configuration qui est dans $R$, alors $(\delta_1, s_1)$ est aussi dans $R$. Le système ne peut légalement se terminer que si l'agent sait qu'il le peut. On peut considérer **KnowHowDelib** comme une formalisation adéquate de la notion d'exécutabilité épistemique pour le cas de systèmes comprenant un seul agent qui délibère.

Examinons un exemple. Considérons le programme suivant, que l'on appellera $système2$:

$$système2 \stackrel{\text{def}}{=} \\ \textbf{Subj}(Futé, \pi\, c[compose(Futé, c, Coffre1); Ouvert(Coffre1)?]).$$

Ici, $Futé$ doit choisir de façon non-déterministe une combinaison (opérateur $\pi$), la composer, et par la suite confirmer que le coffre est ouvert. Un agent qui choisit les transitions qu'il fait de manière arbitraire sans tenter d'anticiper a de grandes chances de choisir une combinaison incorrecte, puisqu'il ne considère pas la nécessité de rendre exécutable l'action de test qui confirme que le coffre est ouvert après avoir fait cette transition. Et effectivement, on peut démontrer que:

$$\exists \delta', c(c \neq combinaison(Coffre1, S_0) \wedge \\ Trans(système2, S_0, \delta', do(compose(Futé, c, Coffre1), S_0))).$$

Par contre, un agent qui délibère et anticipe serait capable de déterminer qu'il *doit* composer la combinaison correcte du coffre. Pour une variante $système2'$, qui est exactement comme $système2$, sauf que **Subj** est remplacé par **Delib**, la seule transition possible est celle où l'agent compose la bonne combinaison. Ainsi, on peut montrer que:

$$\exists \delta', s'(Trans(système2', S_0, \delta', s')) \wedge \\ \forall \delta', s'(Trans(système2', S_0, \delta', s') \supset \\ s' = do(compose(Futé, combinaison(Coffre1), Coffre1), S_0)).$$

Examinons certaines des propriétés de **Delib** et comparons le à **Subj**. Premièrement, on a que:

**Proposition 4**
*Les propriétés des propositions 1, 2 et 3 sont aussi vraies pour* **Delib**.

Pour voir où **Delib** et **Subj** diffèrent, considérons un programme $\alpha; \delta$ composé d'une séquence qui débute par une action primitive. Alors, on a que:

**Proposition 5**

$$
\begin{array}{l}
Trans(\mathbf{Subj}(agt, \alpha; \delta), s, \delta', s') \equiv \\
\quad s' = do(\alpha[s], s) \wedge \delta' = \mathbf{Subj}(agt, nil; \delta) \wedge \\
\quad \exists a\, \mathbf{Know}(agt, \alpha = a \wedge Poss(a, now) \wedge agent(a) = agt, s) \quad \textit{et} \\
Trans(\mathbf{Delib}(agt, \alpha; \delta), s, \delta', s') \equiv \\
\quad s' = do(\alpha[s], s) \wedge \delta' = \mathbf{Delib}(agt, nil; \delta) \wedge \\
\quad \exists a\, \mathbf{Know}(agt, \alpha = a \wedge Poss(a, now) \wedge agent(a) = agt \\
\qquad\qquad \wedge \mathbf{KnowHowDelib}(agt, [nil; \delta], do(a, now)), s).
\end{array}
$$

Ainsi, avec **Delib**, l'agent doit non seulement savoir quelle transition/action faire, mais aussi savoir qu'il saura comment compléter l'exécution de ce qu'il restera du programme après cette transition. **Delib** impose donc des exigences beaucoup plus fortes à l'agent que **Subj**. Le fait que l'on peut faire confiance à un agent qui délibère pour qu'il choisisse ses transitions intelligemment signifie que pour qu'un programme non-déterministe soit épistémiquement exécutable, il n'est plus nécessaire d'exiger que toutes les exécutions se terminent avec succès; il est suffisant que l'agent sache que peu importe le résultat de ses actions de perception, il pourra rejoindre une configuration finale du programme.

Le mode d'exécution délibérative modélisé par **Delib** est similaire à celui utilisé par le langage de programmation d'agent IndiGolog [6] pour les programmes qui sont mis dans un « bloc de recherche ». **Delib** est aussi très similaire à la notion de capacité $\mathbf{Can}_{\perp}$ formalisée dans [14]. Essentiellement, l'agent doit être capable de construire une stratégie (une sorte de plan conditionnel) telle que si le programme est exécuté en accord avec cette stratégie, alors l'agent saura quelle action accomplir à chaque étape et saura comment compléter l'exécution du programme en un nombre borné d'actions. Dans [14], nous montrons que $\mathbf{Can}_{\perp}$ n'est pas aussi général que l'on voudrait et qu'il existe des programmes qu'un agent qui délibère est intuitivement capable d'exécuter et pour lesquels $\mathbf{Can}_{\perp}$ est faux. Il s'agit de programmes itératifs où l'agent sait qu'il complétera éventuellement l'exécution du programme, mais ne peut pas borner le nombre d'actions qu'il devra faire. [14]

propose aussi une formalisation de la capacité plus générale qui traite ces cas correctement. Toutefois, le type de délibération requis pour traiter ces cas correctement est très difficile à implémenter; la notion **Delib** est plus proche du mécanisme implémenté par IndiGolog [6].

Pour le cas multi-agent, nous n'avons pas encore de formalisation générale adéquate de l'exécutabilité épistémique lorsque les agents délibèrent. Le problème est que si le processus comprends de véritables interactions et que le choix d'actions d'un agent dépends de celui des autres agents, alors il faut que les agents modélisent la délibération des autres agents. De plus, il faut tenir compte de la relation entre les agents, s'ils sont coopératifs, compétitifs ou indifférents. Un exemple simple de ce type de cas est un processus où $Robie$ veut ouvrir le coffre et, puisqu'il ne connaît pas la combinaison initialement, planifie de questionner $Futé$ à cet effet, tout en s'attendant à recevoir cette information en réponse. Un mécanisme supportant une version simple de ce type de délibération en IndiGolog a été proposé dans [16] (l'agent qui délibère modélise les autres agents comme des processus déterministes). Nous espérons pouvoir généraliser notre formalisation pour traiter ces cas.

## 5 CONCLUSION

Dans cet article, nous avons traité du problème de garantir que les plans des agents sont épistémiquement exécutables dans les spécifications de systèmes multi-agents. Le problème est lié à celui de formaliser adéquatement la notion d'agent, de façon à ce que les choix d'actions que fait l'agent soient déterminés uniquement par son état local. L'article traite ce problème dans le cadre du langage de spécification CASL, mais le problème est présent peu importe le formalisme de spécification de SMAs employé. Nous avons proposé un traitement de l'*exécution subjective* (**Subj**) qui fait que le plan est exécuté en fonction des connaissances de l'agent plutôt qu'en fonction de l'état du monde. Le traitement suppose que l'agent ne fait pas de planification et choisit arbitrairement parmi les actions permises par le plan. Nous avons aussi proposé un traitement de l'*exécution délibérative* (**Delib**) pour les agents plus intelligents qui font de la planification et anticipent. Finalement, en termes de ces notions, nous avons développé deux formalisations de l'*exécutabilité épistémique:* une première qui modélise l'exécutabilité épistémique pour n'importe quel ensemble de processus multi-agents lorsque les

agents impliqués ne font pas de délibération/anticipation (**KnowHowSubj**), et une autre qui capture l'exécutabilité épistémique pour les processus à un seul agent où celui-ci délibère/anticipe (**KnowHowDelib**). Le cas des processus multi-agents où plusieurs agents délibèrent reste ouvert. Notre formalisation montre aussi comment un traitement de l'exécutabilité épistémique peut être intégré à une sémantique « système de transitions » pour un langage de programmation/spécification d'agent.

Examinons un peu où se situent les autres formalismes de spécification de SMAs par rapport à ce problème. Le formalisme de [32], qui s'inspire des langages employés en sémantique de la programmation concurrente et leur joint un traitement des états mentaux des agents, traite essentiellement l'exécution de programmes de façon subjective. Les tests sont évalués en termes des croyances des agents et les actions primitives sont traitées comme des opérations de mise à jour de leurs croyances. Toutefois, il n'y a pas de représentation du monde extérieur aux agents et il n'est pas possible de modéliser les interactions entre un agent et son environnement extérieur (p. ex., pour parler de la fiabilité de leurs capteurs et effecteurs). Il est possible de représenter l'environnement par un agent, mais ceci n'est pas vraiment naturel. De plus, il n'y a pas de traitement de l'exécution délibérative.

Le formalisme de spécification de SMAs de [8], est une logique avec des modalités temporelles et épistémiques. Une méthode de vérification compositionnelle basée sur le formalisme est aussi proposée. Il est plus difficile de spécifier des systèmes avec des comportements complexes dans une telle logique que dans un formalisme procédural comme CASL ou [32]. Si l'on spécifie les agents selon la méthodologie proposée, l'exécution de leurs actions dépendra uniquement de leur état local. Toutefois, ici non plus, il n'y a pas de mécanisme correspondant à notre exécution délibérative.

Aucun de ces travaux ne discutent les conditions sous lesquelles les plans sont épistémiquement exécutables. Dans le cas des processus déterministes à un seul agent, l'exécutabilité épistémique se réduit à l'existence d'une exécution subjective, et les deux formalismes peuvent donc être vus comme traitant le problème. Mais il n'y a pas de traitement pour les cas multi-agent et agent délibératif. Ni l'un ni l'autre des formalismes ne supporte l'exécution objective de processus (le défaut en CASL).

Les travaux décrits dans cet article se poursuivent. En particulier, il reste à faire une comparaison plus complète avec les travaux antérieurs sur l'exécutabilité épistémique et à produire une formalisation adéquate de cette notion pour le cas des processus où plusieurs agents délibèrent. En nous inspi-

rant des idées proposées ici, nous avons récemment développé dans [5] une meilleure sémantique du mécanisme de délibération présent dans le langage de programmation d'agent IndiGolog. Nous travaillons maintenant à développer une approche à la conception de SMAs qui permettrait de spécifier un système en CASL, de vérifier qu'il satisfait les propriétés désirées (un outil de vérification pour CASL est décrit dans [29]), et par la suite de décomposer le système en agents individuels IndiGolog et obtenir ainsi une spécification exécutable. L'approche garantirait que les propriétés vérifiées sur la spécification CASL resteraient satisfaites par l'ensemble d'agents individuels IndiGolog obtenus. Nous voulons aussi examiner comment nos traitements de l'exécution subjective et délibérative pourraient être adaptés au modèle plus général des connaissances de [30], qui admet les croyances fausses et supporte la révision des croyances. Ceci nous permettrait de modéliser des cas où un agent exécute son programme d'une façon qu'il croit correcte, mais qui est en fait incorrecte. La communication non-sincère ou celle où des informations erronées sont transmises pourrait aussi être traitée.

## Remerciements

J'ai beaucoup bénéficié de discussions sur le sujet de cet article avec Hector Levesque, Giuseppe De Giacomo, Steven Shapiro et David Tremaine. Les commentaires d'un évaluateur anonyme d'ATAL ont aussi été très utiles. Ces recherches ont été soutenues par « Communications and Information Technology Ontario » et le Conseil de recherches en sciences naturelles et en génie du Canada.

## RÉFÉRENCES

[1] F. Brazier, B. Dunin-Keplicz, N.R. Jennings et Jan Treur. Formal specifications of multi-agents systems: A real-world case study. Dans *Proceedings of the First International Conference on Multi-Agent Systems (ICMAS'95)*, pages 25–32, San Francisco, CA, juin 1995. Springer-Verlag.

[2] E. Clarke, E.A. Emerson et 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 et Hector J. Levesque. ConGolog, a concurrent programming language based on the situation calculus. *Artificial Intelligence*, 121: 109–169, 2000.

[5] Giuseppe De Giacomo, Yves Lespérance, Hector J. Levesque et Sebastian Sardiña. On the semantics of deliberation in IndiGolog — from theory to implementation. Dans D. Fensel, F. Giunchiglia, D. McGuiness et M.-A. Williams, editeurs, *Principles of Knowledge Representation and Reasoning: Proceedings of the Eight International Conference (KR-2002)*, pages 603–614, Toulouse, France, avril 2002. Morgan Kaufmann.

[6] Giuseppe De Giacomo et Hector J. Levesque. An incremental interpreter for high-level programs with sensing. Dans Hector J. Levesque and Fiora Pirri, editeurs, *Logical Foundations for Cognitive Agents*, pages 86–102. Springer-Verlag, Berlin, Allemagne, 1999.

[7] Giuseppe De Giacomo, Eugenia Ternovskaia et Ray Reiter. Nonterminating processes in the situation calculus. Dans *Working Notes of the AAAI'97 Workshop on Robots, Softbots, Immobots: Theories of Action, Planning and Control*, 1997.

[8] Joeri Engelfriet, Catholijn M. Jonker et Jan Treur. Compositional verification of multi-agent systems in temporal multi-epistemic logic. Dans J.P. Mueller, M.P. Singh et A.S. Rao, editeurs, *Intelligent Agents V: Proceedings of the Fifth International Workshop on Agent Theories, Architectures and languages (ATAL'98)*, volume 1555 de *LNAI*, pages 177–194. Springer-Verlag, 1999.

[9] M. Fisher et M. Wooldridge. On the formal specification and verification of multi-agent systems. *International Journal of Cooperative Information Systems*, 6(1): 37–65, 1997.

[10] M. Hennessy. *The Semantics of Programming Languages*. John Wiley & Sons, 1990.

[11] Gerhard Lakemeyer et Hector J. Levesque. AOL: A logic of acting, sensing, knowing, and only-knowing. Dans *Principles of Knowledge Representation and Reasoning: Proceedings of the Sixth International Conference (KR-98)*, pages 316–327, 1998.

[12] Yves Lespérance, Todd G. Kelley, John Mylopoulos et Eric S.K. Yu. Modeling dynamic domains with ConGolog. Dans *Advanced Informa-*

*tion Systems Engineering, 11th International Conference, CAiSE-99, Proceedings*, pages 365–380, Heidelberg, Allemagne, juin 1999. LNCS 1626, Springer-Verlag.

[13] Yves Lespérance, Hector J. Levesque, F. Lin, Daniel Marcu, Raymond Reiter et Richard B. Scherl. Fondements d'une approche logique à la programmation d'agents. Dans *Actes des Troisièmes Journées Francophones sur l'Intelligence Artificielle Distribuée et les Systèmes Multi-Agents*, pages 3–14, Chambéry-St. Badolph, France, mars 1995.

[14] Yves Lespérance, Hector J. Levesque, Fangzhen Lin et Richard B. Scherl. Ability and knowing how in the situation calculus. *Studia Logica*, 66(1): 165–186, octobre 2000.

[15] Yves Lespérance, Hector J. Levesque et Raymond Reiter. A situation calculus approach to modeling and programming agents. Dans A. Rao et M. Wooldridge, editeurs, *Foundations of Rational Agency*, pages 275–299. Kluwer, 1999.

[16] Yves Lespérance et Ho-Kong Ng. Integrating planning into reactive high-level robot programs. Dans *Proceedings of the Second International Cognitive Robotics Workshop*, pages 49–54, Berlin, Allemagne, août 2000.

[17] Hector J. Levesque. What is planning in the presence of sensing? Dans *Proceedings of the Thirteenth National Conference on Artificial Intelligence*, pages 1139–1146, Portland, OR, août 1996.

[18] Fangzhen Lin et Hector J. Levesque. What robots can do: Robot programs and effective achievability. *Artificial Intelligence*, 101(1–2): 201–226, 1998.

[19] John McCarthy et Patrick Hayes. Some philosophical problems from the standpoint of artificial intelligence. Dans B. Meltzer and D. Michie, editeurs, *Machine Intelligence*, volume 4, pages 463–502. Edinburgh University Press, Edinburgh, UK, 1979.

[20] Robert C. Moore. A formal theory of knowledge and action. Dans J. R. Hobbs et Robert C. Moore, editeurs, *Formal Theories of the Common Sense World*, pages 319–358. Ablex Publishing, Norwood, NJ, 1985.

[21] Leora Morgenstern. Knowledge preconditions for actions and plans. Dans *Proceedings of the Tenth International Joint Conference on Artificial Intelligence*, pages 867–874, Milan, Italie, août 1987. Morgan Kaufmann Publishing.

[22] D. Park. Fixpoint induction and proofs of program properties. Dans *Machine Intelligence*, volume 5, pages 59–78. Edinburgh University Press, 1970.

[23] G. Plotkin. A structural approach to operational semantics. Technical Report DAIMI-FN-19, Computer Science Dept., Aarhus University, Danemark, 1981.

[24] Raymond Reiter. The frame problem in the situation calculus: A simple solution (sometimes) and a completeness result for goal regression. Dans Vladimir Lifschitz, editeur, *Artificial Intelligence and Mathematical Theory of Computation: Papers in Honor of John McCarthy*, pages 359–380. Academic Press, San Diego, CA, 1991.

[25] Raymond Reiter. *Knowledge in Action: Logical Foundations for Specifying and Implementing Dynamical Systems*. MIT Press, 2001.

[26] Richard B. Scherl et Hector J. Levesque. The frame problem and knowledge-producing actions. Dans *Proceedings of the Eleventh National Conference on Artificial Intelligence*, pages 689–695, Washington, DC, juillet 1993. AAAI Press/The MIT Press.

[27] Steven Shapiro et Yves Lespérance. Modeling multiagent systems with CASL — a feature interaction resolution application. Dans C. Castelfranchi et Y. Lespérance, editeurs, *Intelligent Agents VII. Agent Theories, Architectures, and Languages — 7th. International Workshop, ATAL-2000, Boston, MA, USA, July 7–9, 2000, Proceedings*, volume 1986 de *LNAI*, pages 244–259. Springer-Verlag, Berlin, 2001.

[28] Steven Shapiro, Yves Lespérance et Hector J. Levesque. Specifying communicative multi-agent systems with ConGolog. Dans *Working Notes of the AAAI Fall 1997 Symposium on Communicative Action in Humans and Machines*, pages 75–82, Cambridge, MA, novembre 1997.

[29] Steven Shapiro, Yves Lespérance et Hector J. Levesque. The cognitive agents specification language and verification environment for multiagent systems. Dans C. Castelfranchi et W. Lewis Johnson, editeurs, *Proc. of the 1st Int. Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS-2002)*, pages 19–26, Bologna, Italie, juillet 2002. ACM Press.

[30] Steven Shapiro, Maurice Pagnucco, Yves Lespérance et Hector J. Levesque. Iterated belief change in the situation calculus. Dans A.G. Cohn, F. Giunchiglia et B. Selman, editeurs, *Principles of Knowledge*

*Representation and Reasoning: Proceedings of the Seventh International Conference (KR-2000)*, pages 527–538. Morgan Kaufmann, 2000.

[31] W. van der Hoek, B. van Linder et J.-J. Ch. Meyer. A logic of capabilities. Dans A. Nerode et Yu. V. Matiyasevich, editeurs, *Proceedings of the Third International Symposium on the Logical Foundations of Computer Science (LFCS'94)*. LNCS Vol. 813, Springer-Verlag, 1994.

[32] Rogier M. van Eijk, Frank S. de Boer, Wiebe van der Hoek et John-Jules Ch. Meyer. Open multi-agent systems: Agent communication and integration. Dans N.R. Jennings et Y. Lespérance, editeurs, *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.