De l'exécutabilité épistémique des plans dans les spécifications de systèmes multiagents

Yves Lespérance

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

Résumé Cet article concerne le problème de garantir que les plans des agents sont épistémiquement exécutables dans les spécifications de systèmes multiagents. Nous proposons des solutions dans le cadre du formalisme 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 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 si un plan est épistémiquement exécutable pour un agent dans plusieurs types de situations.

1 Introduction

Durant les dernières années, divers cadres formels ont été proposés pour supporter la spécification et la vérification des systèmes multiagents (SMA)[1, 5, 6, 23]. Nous avons participé au développement d'un tel cadre formel le ≪ Cognitive Agents Specification Language ≫ (CASL) [19]. CASL réunit des idées de la théorie des agents et des méthodes formelles en génie logiciel, pour produire un langage de spécification expressif qui peut être utilisé pour modéliser et vérifier des SMAs complexes.

Un problème avec 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 multiagent, 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 exécuterons en fonction de leurs plans et des connaissances qu'ils ont sur l'état du système. Hors, ils 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 [14]. Nous avons un agent, Robie, qui veut ouvrir un coffre-fort, mais qui ne sait pas la combinaison de ce coffre. Il y

a aussi un second agent, $Futé$, qui sait 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) 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 sait 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:

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

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 δ_1 et δ_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 prérequis a déjà été étudié en théorie des agents sous les

[1] Formellement:

$$\begin{array}{l} \exists s\, Do(compose(Robie, combinaison(Coffre1), Coffre1), S_0, s) \wedge \\ \forall s(Do(compose(Robie, combinaison(Coffre1), Coffre1), S_0, s) \supset \\ \quad Ouvert(Coffre1, s)). \end{array}$$

La notation est expliquée à la section 3.

[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 sénarios 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 pour les agents.

termes $\ll$ knowledge prerequisites of action $\gg$, $\ll$ knowing how to execute a program $\gg$ et $\ll$ ability to achieve a goal $\gg$ [14, 15, 22, 2, 10].

L'analyste pourrait inclure explicitement tous ces prérequis 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 tous les prérequis 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 $\textbf{Subj}(agt, \delta)$ qui veut dire que le processus δ est exécuté subjectivement par l'agent agt et qui garantit que δ est épistémiquement exécutable pour agt. Notons que nous aurions pu rendre l'exemple plus réaliste en ayant $Robie$ faire une requête à $Futé$ de l'informer de la combinaison du coffre et en ayant $Futé$ répondre à cette requête, comme dans les exemples de [19]; mais ici, nous preférons 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 peut être 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. L'article porte principalement sur le développement d'un modèle adéquat de la notion d'agent en vue de l'utiliser pour la production de spécifications de SMAs. Le modèle devrait aussi être utile pour donner une sémantique formelle plus satisfaisante aux langages de programmation et aux architectures d'agent.

2 Aperçu de CASL

Le $\ll$ Cognitive Agents Specification Language $\gg$ (CASL) [19] est un formalisme de spécification pour les systèmes multiagents (SMA). Il joint une théorie de l'action [17] et des états mentaux [18] basée sur le calcul des situations [13] à ConGolog [3], un langage de progrmmation concurrente et nondéterministe 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 CASL comprends deux composantes. La première est une spécification de la *dynamique du domaine*, c.-à-d. des fluents qui sont employés pour modéliser l'état du système, des actions qui peuvent être accomplies par les agents, de leurs préconditions et effets, et de 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 com-

ment ils sont affectés par les actions de communication (e.g. informe, requête, annule-requête, etc.) et les actions de perception. Cette composante est spécifiée de façon purement déclarative dans le calcul des situations. Nous employons la solution de Reiter au problème de frame, où les axiomes d'effets sont compilés en axiomes d'état successeur [17]. Ainsi, pour notre exemple, la spécification de la dynamique du domaine inclut l'axiome d'état successeur:

$$Ouvert(x, do(a, s)) \equiv$$
$$\exists agt, c(a = compose(agt, c, x) \wedge c = combinaison(x, s)) \vee Ouvert(x, s),$$

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 seulemment 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 précondition:

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

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 [14, 18]. 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 ϕ dans la situation s, **Know**(agt, ϕ, s), si et seulement si ϕ 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, $\phi[s]$ represente la formule obtenue en substituant s pour toutes les instances libres (c.-à-d. non-liées par un autre **Know**) de la constante spéciale now; ainsi par exemple, **Know**$(agt, Ouvert(Coffre1, now), s)$ est une abbréviation 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'il savent quelque chose (introspection positive et négative). **KRef**(agt, θ, s) est une abbréviation de $\exists t$ **Know**$(agt, t = \theta, s)$, c.-à-d. qu'agt sait qui est θ (à quoi réfère le terme θ). Ici, nous spécifions la dynamique des connaissances avec l'axiome d'état successeur suivant:

$$K(agt, s^*, do(a, s)) \equiv$$
$$\exists s'[K(agt, s', s) \wedge s^* = do(a, s') \wedge Poss(a, s') \wedge$$
$$\forall informer, \theta(a = informRef(informer, agt, \theta) \supset \theta[s'] = \theta[s])].$$

Ainsi, après qu'une action a été faite, tous les agents savent que cette action a été faite, et si cette action est d'informer agt de qui est θ, agt acquiert cette information dans la situation résultante ($\theta[s]$ représente θ avec s susbtitué à now comme pour $\phi[s]$). Les préconditions de l'action $informRef$ sont définies par l'axiome suivant:

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

c.-à-d. que $informRef$ est possible dans la situation s si et seulement si $informer$ connaît la valeur de θ. Nous spécifions aussi l'agent de l'action par:

$$agent(informRef(informer, agt, \theta)) = informer.$$

Les buts et les requêtes sont modélisés de façon analogue; voir [19] pour les détails. Dans cet article, nous optons pour la simplicité et n'utilisons pas ces notions dans nos exemples.

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/description de processus ConGolog, qui fournit un riche ensemble de structures pour spécifier les processus multiagents: processus concurrents, processus concurrents avec priorités différentes, interruptions, choix nondéterministes, etc.

La sémantique du langage de description de processus ConGolog [3] est définie en termes de *transitions*, dans le style de la sémantique opérationnelle structurale [16, 7]. Une transition est une unité atomique d'exécution de programme, soit l'exécution d'une action primitive, soit l'exécution d'un test d'une condition dans la situation courrante. La sémantique emploie deux prédicats spéciaux, $Final$ et $Trans$. $Final(\delta, s)$ signifie que le programme δ peut légalement se terminer dans la situation s et $Trans(\delta, s, \delta', s')$ veut dire que le programme δ dans la situation s peut légalement exécuter une opération atomique, pour arriver dans la situation s' avec le programme δ' restant à exécuter. $Trans$ et $Final$ sont caractérisés par un ensemble d'axiomes dont les suivants:

$$\begin{aligned}
&Trans(\alpha, s, \delta, s') \equiv && \text{action primitive}\\
&\quad Poss(\alpha[s], s) \wedge \delta = nil \wedge s' = do(\alpha[s], s),\\
&Final(\alpha, s) \equiv False,\\
&Trans([\delta_1; \delta_2], s, \delta, s') \equiv && \text{séquence}\\
&\quad Final(\delta_1, s) \wedge Trans(\delta_2, s, \delta, s')\\
&\quad \vee \exists \delta' (\delta = (\delta'; \delta_2) \wedge Trans(\delta_1, s, \delta', s')),\\
&Final([\delta_1; \delta_2], s) \equiv Final(\delta_1, s) \wedge Final(\delta_2, s).
\end{aligned}$$

Le premier axiome dit qu'un programme constitué d'une action primitive α peut accomplir une transition dans la situation s à condition que l'action α soit possible dans s, que la situation resultante 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.

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 reflexive 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 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 δ, et où le programme peut légalement terminer dans la situation s'.

L'approche de CASL vise un compromis entre une spécification purement intentionnelle (c.-à-d. basée sur les attitudes mentales) des agents, 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 accomoder les modèles incomplètement spécifiés, soit parce que l'état initial n'est pas complètement spécifié, soit parce que les processus impliqués sont nondéterministes et peuvent évoluer de plusieurs façons. L'approche supporte la vérification, de même que la simulation lorsque la description de l'état initial est suffisamment complète.

La dernière version de CASL, qui supporte la communication avec des actes de langages encryptés et un traitement simplifié des buts est décrite dans [19]. L'article montre aussi comment CASL a été employé pour modéliser un système multiagent relativement complexe qui résout les interactions de services en téléphonie; ce système est constitué d'agents autonomes ayant des buts explicites qui entrent en négociation. Des versions antérieures de CASL sont décrites en détail dans [20, 11, 9], où l'utilisation du formalisme est illustrée par un example simple de SMA d'organisation de réunions. Une discussion de l'emploi de CASL pour la modélisarion des processus et l'analyse de besoins a paru dans [8]; cet article décrit aussi des outils de support qui sont en cours de développement.

3 L'exécution subjective

Nous définissons l'opérateur d'exécution subjective $\mathbf{Subj}(agt, \delta)$ introduit dans 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 se terminer légalement que si l'agent sait qu'elle peut terminer légalement.

Avec cette définition, il est facile de démontrer que la spécification de l'exemple $système1$ est exécutable. Supposons que dans la situation initiale S_0, $Robie$ ne connaît pas la combinaison du coffre, mais que $Futé$ lui la connaît; formellement:

$$\neg\exists c\,\mathbf{Know}(Robie, combinaison(Coffre1) = c, S_0) \wedge \\ \exists c\,\mathbf{Know}(Futé, combinaison(Coffre1) = c, S_0)$$

Pour montrer que la spécification de l'exemple $système1$ est exécutable, on prouve que:

$$\exists s Do(système1, S_0, s),$$

Nous omettons les détails de la preuve. Notons que l'on peut éliminer l'action de test $\mathbf{KRef}(Robie, combinaison(Coffre1))?$ de $système1$, puisque la transition pour l'action $compose$ n'est possible que si $Robie$ sait la combinaison; ceci est une conséquence du fait que l'agent doit savoir quelle est l'action qu'il doit accomplir (c.-à-d. $compose(combinaison(Coffre1), Coffre1)$). On peut aussi facilement montrer que si $Futé$ n'informe pas $Robie$ de la combinaison, alors le processus n'est plus executable, c.-à-d.:

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

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

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

Deuxièmement, pour une action de test impliquant une condition ϕ, un agent ne peut l'exécuter subjectivement que s'il sait que ϕ est vrai dans la situation courrante:

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

Donc, on voit comment dans l'exécution subjective, les tests et les fluents qui aparaissent dans le programme, de même que les préconditions des actions, sont tous évalués par rapport à l'état des connaissances de l'agent plutôt que par rapport à l'état du monde. Si on généralisait le modèle pour admettre les croyances fausses et la révision des croyances comme dans [21], ceci nous permettrait de modéliser comment l'agent exécute le programme d'une façon qu'il conçoit comme correcte, mais qui peut en fait être incorrecte.

Une autre propriété importante de **Subj** est la façon dont il traite les programmes nondéterministes. On peut voir **Subj** comme modélisant le comportement d'un agent qui exécute son programme de manière aveugle ou hardie. Lorsque le programme autorise plusieurs transitions différentes, l'agent choisit la prochaine transitions 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; $\delta_1|\delta_2$ représente un choix nondéterministe entre δ_1 et δ_2. Alors l'agent pourrait très bien choisir d'exécuter l'action a, après quoi il lui reste le programme $False?$ à exécuter, ce qui n'est jamais possible. Si on veut confirmer qu'un tel agent aveugle saura comment exécuter un programme nondéterministe, on doit s'assurer que chaque chemin à travers le programme est subjectivement exécutable et mène à une situation finale du programme. Ce mode d'exécution aveugle est celui employé par défault dans le langage de programmation d'agent IndiGolog [4].

Subj est très semblable à la notion de $\ll$ dumb knowing how $\gg$ **DKH**(δ, s) formalisée dans [10] pour les δs qui sont des programmes Golog, c.-à-d. des programmes ConGolog sans processus concurrents ni interruptions. Pour tout programme Golog deterministe n'impliquant qu'un seul agent, nous croyons que **Subj** et **DKH** sont essentiellement équivalents dans le sens que:

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

On peut considérer $\exists s' Do(\textbf{Subj}(agt, \delta), s, s')$ comme une formalisation adéquate de la notion d'exécutabilité épistemique pour le cas de systèmes comprenant un seul agent qui exécute un programme déterministe de manière aveugle.

Pour les programmes Golog nondéterministes n'impliquant qu'un seul agent, nous croyons que l'équivalence est aussi vraie, mais on ne peut l'exprimer avec Do. On doit plutôt dire qu'on a **DKH** si et seulement si tous les chemins à travers le programme sont subjectivement exécutables et mènent à une situation terminale.

Notons que **Subj** est considérablement plus général que **DKH**; **Subj** peut être employé pour spécifier des systèmes comprenant des processus concurrents et plusieurs agents, comme dans l'exemple $système1$. Toutefois, un traitement général de l'exécutabilité épistemique pour ces cas reste à définir.

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. Dans la présente section, nous proposerons un autre traitement qui capture les conditions dans lesquelles un agent qui délibère est capable d'exécuter un programme. Nous utiliserons la notation **Delib**(agt, δ) pour cette notion d'*exécution délibérative*. Nous la formalisons comme suit:

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

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

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

Ceci signifie que le système ne peut faire une transition que si l'agent sait qu'il peut faire cette transition et sait qu'il saura comment compléter l'exécution du programme après avoir fait cette transition. L'agent *sait comment exécuter* δ dans la situation s, **KnowHowDelib**(agt, δ, s), si et seulement si (δ, s) est dans la plus petite relation R telle que (1) si l'agent sait que (δ_1, s_1) peut terminer légalement, alors (δ_1, s_1) est dans R, et (2) si l'agent sait qu'il peut faire une transition dans (δ_1, s_1) pour arriver à une configuration qui est dans R, alors (δ_1, s_1) est aussi dans R. Le système ne peut terminer légalement 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. Supposons que

$$S_1 = do(informRef(Futé, Robie, combinaison(Coffre1)), S_0),$$

c.-à-d. que S_1 est la situation où $Futé$ vient de dire à $Robie$ quelle est la combinaison du coffre. Considérons le programme $système2$ suivant qui sera exécuté dans la situation S_1:

$$\begin{aligned}
&système2 \stackrel{\text{def}}{=} \\
&\qquad \textbf{Subj}(Robie, \pi\, c[compose(Robie, c, Coffre1); Ouvert(Coffre1)?]).
\end{aligned}$$

Ici, $Robie$ doit choisir de façon nondeterministe une combinaison (opérateur π), 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 mauvaise combinaison, 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 demontrer que:

$$\begin{aligned}
&\exists \delta', c(c \neq combinaison(Coffre1, S_1) \wedge \\
&\qquad Trans(système2, S_1, \delta', do(compose(Robie, c, Coffre1), S_1)))
\end{aligned}$$

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:

$$\begin{aligned}
&\exists \delta', s'(Trans(système2', S_1, \delta', s')) \wedge \\
&\forall \delta', s'(Trans(système2', S_1, \delta', s') \supset \\
&\qquad s' = do(compose(Robie, combinaison(Coffre1), Coffre1), S_1))
\end{aligned}$$

Le mode d'exécution délibérative modélisé par **Delib** est similaire à celui utilisé par le langage de programmation d'agent IndiGolog [4] pour les programmes qui sont mis dans un $\ll$ bloc de recherche $\gg$. **Delib** est aussi très similaire à la notion de capacité $\mathbf{Can}_{\perp}$ formalisée dans [10]. 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 [10], nous démontrons 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. [10] propose aussi un autre traitement de la capacité qui capture 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 [4].

Une déficience de la version de **Delib** proposée est qu'elle ne traite pas les programmes où l'agent a besoin de l'intervention d'autres agents durant l'exécution de son programme. Un exemple simple de ceci est un programme similaire à $système2$, mais où $Robie$ ne connaît pas encore la combinaison au moment où il commence à délibérer et où il doit questionner $Futé$ à cet effet, tout en s'attendant à recevoir cette information en réponse. Un mécanisme qui supporte ce type de délibération en IndiGolog est décrit dans [12]. Le problème de développer une formalisation de l'exécutabilité épistemique adéquate en général pour les systèmes à plusieurs agents (cas coopératifs et cas compétitifs) reste ouvert.

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 multiagents. 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 peut importe le formalisme de specification de SMAs employé. Nous avons proposé un traitement de l'exécution subjective de plan (**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. Nous avons aussi montré comment ces notions permettent d'exprimer si un plan est épistémiquement exécutable pour un agent dans plusieurs types de situations.

Examinons un peu où se situent les autres formalismes de spécification de SMAs par rapport à ce problème. Le formalisme de [23], 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 répresentation 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 delibérative.

Le formalisme de [5], est une logique avec des modalités temporelles et épistémiques. Il est plus difficile de spécifier des systèmes avec des comportements complexes dans une telle logique que dans un formalisme procédural comme CASL ou [23]. Si on spécifie les agents selon la méthodologie proposée, l'exécution de leurs actions dependra uniquement de leur état local. Il n'y a pas de mécanisme correspondant à notre exécution délibérative.

Les résultats décrits dans cet article ne sont pas complets et nos travaux dans le domaine se poursuivent. En particulier, il reste à faire une comparaison plus complète avec les travaux antérieurs sur le sujet et à produire une définition adéquate de la notion d'exécutabilité épistémique dans le cas des processus multiagents ou nondéterministes à partir des opérateurs **Subj** et **Delib**. Nous comptons aussi utiliser ce traitement pour produire une meilleure sémantique du langage de programmation d'agent IndiGolog [4, 12] et rendre compte des éléments de ce langage qui supportent l'opération d'un agent qui planifie et agit dans un monde dynamique et incomplètement connu.

Remerciments

Nous avons beaucoup bénéficié de discussions sur le sujet de cet article avec Hector Levesque, Giuseppe De Giacomo, Steven Shapiro et David Tremaine.

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. Ernest Davis. Knowledge preconditions for plans. *Journal of Logic and Computation*, 4(5): 721–766, 1994.
3. 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.
4. Giuseppe De Giacomo et Hector J. Levesque. An incremental interpreter for high-level programs with sensing. Dans Hector J. Levesque et Fiora Pirri, éditeurs, *Logical Foundations for Cognitive Agents*, pages 86–102. Springer-Verlag, Berlin, Allemagne, 1999.
5. 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, éditeurs, *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.

6. 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.
7. M. Hennessy. *The Semantics of Programming Languages*. John Wiley & Sons, 1990.
8. Yves Lespérance, Todd G. Kelley, John Mylopoulos et Eric S.K. Yu. Modeling dynamic domains with ConGolog. Dans *Advanced Information Systems Engineering, 11th International Conference, CAiSE-99, Proceedings*, pages 365–380, Heidelberg, Allemagne, Juin 1999. LNCS 1626, Springer-Verlag.
9. 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.
10. 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.
11. Yves Lespérance, Hector J. Levesque et Raymond Reiter. A situation calculus approach to modeling and programming agents. Dans A. Rao et M. Wooldridge, éditeurs, *Foundations of Rational Agency*, pages 275–299. Kluwer, 1999.
12. 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.
13. John McCarthy et Patrick Hayes. Some philosophical problems from the standpoint of artificial intelligence. Dans B. Meltzer et D. Michie, éditeurs, *Machine Intelligence*, volume 4, pages 463–502. Edinburgh University Press, Edinburgh, UK, 1979.
14. Robert C. Moore. A formal theory of knowledge and action. Dans J. R. Hobbs et Robert C. Moore, éditeurs, *Formal Theories of the Common Sense World*, pages 319–358. Ablex Publishing, Norwood, NJ, 1985.
15. 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.
16. G. Plotkin. A structural approach to operational semantics. Technical Report DAIMI-FN-19, Computer Science Dept., Aarhus University, Denmark, 1981.
17. Raymond Reiter. The frame problem in the situation calculus: A simple solution (sometimes) and a completeness result for goal regression. Dans Vladimir Lifschitz, éditeur, *Artificial Intelligence and Mathematical Theory of Computation: Papers in Honor of John McCarthy*, pages 359–380. Academic Press, San Diego, CA, 1991.
18. 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.
19. Steven Shapiro et Yves Lespérance. Modeling multiagent systems with CASL — a feature interaction resolution application. Dans C. Castelfranchi et Y. Lespérance, éditeurs, *Intelligent Agents VII. Agent Theories, Architectures, and Languages — 7th. International Workshop, ATAL-2000, Boston, MA, USA, July 7–9, 2000, Proceedings*, LNAI. Springer-Verlag, Berlin, 2001. A paraître.
20. Steven Shapiro, Yves Lespérance et Hector J. Levesque. Specifying communicative multiagent 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.
21. 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, éditeurs, *Principles of Knowledge Representation and Reasoning: Proceedings of the Seventh International Conference (KR-2000)*, pages 527–538. Morgan Kaufmann, 2000.

22. W. van der Hoek, B. van Linder et J.-J. Ch. Meyer. A logic of capabilities. Dans A. Nerode et Yu. V. Matiyasevich, éditeurs, *Proceedings of the Third International Symposium on the Logical Foundations of Computer Science (LFCS'94)*. LNCS Vol. 813, Springer-Verlag, 1994.
23. 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, éditeurs, *Intelligent Agents VI — Proceedings of the Sixth International Workshop on Agent Theories, Architectures, and Languages (ATAL-99)*, Lecture Notes in Artificial Intelligence. Springer-Verlag, Berlin, 2000.