IFSIC - Université de Rennes 1
Examen du DEA d'informatique
Module Raisonnement Temporel et Spatial
Énoncé et corrigé de la partie "approches logiques"

Lundi 26 janvier 2004, 14h

2 Approches logiques (45mn)

2.1 Un monde de cinq blocs

On considère un univers de *blocs*, comportant cinq blocs A, B, C, D, E. Ces blocs peuvent être posés l'un sur l'autre, ou directement sur la table, Les contraintes de ce domaine exigent également :

(C1) un seul bloc peut être posé sur un bloc donné,

(C2) un bloc ne peut pas être posé sur plusieurs blocs à la fois, et

(C3) un bloc n'est jamais posé sur lui-même.

Un robot est capable d'effectuer l'une des deux action suivantes :

$PoserSur(x, y)$	poser le bloc x sur le bloc y,
$PoserSurTable(x)$	poser le bloc x sur la table.

On utilise les trois propriétés suivantes pour décrire une situation s de ce monde de blocs :

$Sur(x, y, s)$	le bloc x est posé sur le bloc y,
$SurTable(x, s)$	le bloc x est posé directement sur la table,
$Libre(x, s)$	le bloc x est libre.

On a aussi les contraintes du domaine suivantes :

(C4) un bloc est libre ssi aucun bloc n'est posé sur lui,

(C5) un bloc x est posé directement sur la table ssi, il n'est posé sur aucun bloc.

Les deux actions ont les pré-conditions suivantes :

1. le robot peut poser x sur y ssi les deux blocs x et y sont libres, et
2. il peut poser x sur la table ssi le bloc x est libre.

2.2 Calcul situationnel (de Toronto)

On demande de traduire le 2.1 à l'aide du calcul situationnel "de Toronto" (Reiter) vu en cours.

Question 2.2.1 1. Décrire le vocabulaire exact utilisé (symboles d'objets, d'actions, de fluents, et autres si nécessaire) avec leur sens intuitif (on demande d'utiliser exactement deux symboles d'action et trois de fluents, et d'utiliser pour ces symboles les noms suggérés par l'énoncé).

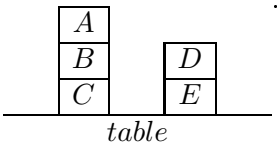
2. Donner les axiomes de pré-conditions d'action pour chacune des deux actions.
3. Donner les formules traduisant les contraintes du domaine.

Question 2.2.2 Donner les formes générales des axiomes d'effets positifs et négatifs pour chacun des trois fluents relationnels. On demande ici de conserver les fluents tels qu'introduits, sans essayer de réduire le nombre de symboles (on laissera les trois symboles de fluents – et un quatrième symbole de prédicat si nécessaire – tels qu'ils s'introduisent naturellement ici).

Question 2.2.3
1. En fait, grâce aux contraintes du domaine, on peut se contenter d'utiliser un seul symbole de fluent relationnel au lieu de trois. On demande ici de réécrire les axiomes d'effets positifs et négatifs de ce seul fluent, en éliminant les deux autres symbole de fluents, ainsi que le quatrième symbole de prédicat s'il a été utilisé.
2. Donner alors l'axiome de succession d'états issu de ces deux axiomes d'effets, en faisant l'hypothèse de fermeture causale. On essaiera de simplifier cet axiome.

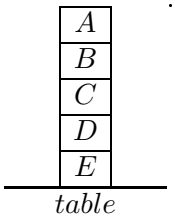
Question 2.2.4 Quelques réflexions sur le sujet tel qu'il est posé.
1. Parmi les formules demandées ci-dessus, lesquelles à votre avis doivent être fournies par l'utilisateur, et lesquelles devraient être calculées automatiquement par le système ?
2. Peut-on vraiment se passer complètement des deux symboles de fluents "inutiles". Sinon, quelle peut être leur utilité ?
3. Tel que l'énoncé est posé, un des deux problèmes principaux de ce type de formalisation (problème de qualification et problème du cadre), est négligé, au sens où les données elles-mêmes le considèrent comme résolu, tandis que le système "résout" l'autre. Quel est celui de ces deux problèmes qui a été négligé ici, et comment cela se voit-il dans les données. Comment le système résout-il l'autre problème ?

Question 2.2.5 On considère la situation initiale suivante : .



Donner les formules particulières décrivant cette situation.

Question 2.2.6 En utilisant les "actions complexes" vues en cours, spécifier un programme (qui pourra alors être facilement écrit en GOLOG, ce qui n'est pas demandé) qui, partant de la situation initiale de la question 2.2.5, permet d'arriver à une situation finale donnée. On prendra le cas de la situation finale suivante, même si le "programme" demandé dépend très peu des situations initiales et finales : .



On ne demande pas de chercher à optimiser, on demande seulement d'utiliser correctement les "actions complexes" vues en cours pour écrire un "programme" de ce genre (en espérant que le système est suffisamment bien conçu pour trouver une solution efficace).

Corrigé

Question 2.2.1 1. Deux symboles d'actions : $PoserSur$ et $PoserSurTable$, symboles de fonction d'arités respectives 2 et 1.

$PoserSur(x, y)$: poser le bloc x sur le bloc y,
$PoserSurTable(x)$: Poser le bloc x sur la table.

Trois symboles de fluents : Sur, $SurTable$ et $Libre$, symboles de prédicat d'arités respectives 3, 2 et 2.

$Sur(x, y, s)$: le bloc x est posé sur le bloc y en situation s
$SurTable(x, s)$: le bloc x est posé directement sur la table en situation s,
$Libre(x, s)$: le bloc x est libre en situation s.

Cinq symboles d'objets A, B, C, D, E, représentant les cinq blocs.

Un prédicat d'arité 2 $Poss$; $Poss(a, s)$: l'action a est possible en situation s.
[x et y variables de type *objet*, s variable de type *situation*, a variable de type *action*.]

2. $Poss(PoserSur(x, y), s) \Leftrightarrow (Libre(x, s) \wedge Libre(y, s) \wedge x \neq y)$.
$Poss(PoserSurTable(x), s) \Leftrightarrow Libre(x, s)$[1].

3. (C1) : $\forall\, x\ y\ z\ s\ \ (Sur(x, y, s) \wedge Sur(z, y, s)) \Rightarrow x = z$;
(C2) : $\forall\, x\ y\ z\ s\ \ (Sur(x, y, s) \wedge Sur(x, z, s)) \Rightarrow y = z$;
(C3) : $\forall\, x\ s\ \ \neg Sur(x, x, s)$[2] ;
(C4) : $\forall\, x\ s\ \ Libre(x, s) \Leftrightarrow (\forall y \neg Sur(y, x, s))$;
(C5) : $\forall\, x\ s\ \ SurTable(x, s) \Leftrightarrow (\forall y \neg Sur(x, y, s))$.
Remarque : La dernière formule entraîne la formule suivante, qu'il est donc inutile de rajouter : $\forall\, x\ s\ \ (SurTable(x, s) \vee (\exists\, y\ \ Sur(x, y, s)))$ (cf deuxième phrase de l'énoncé, en 2.1).

Question 2.2.2 Forme générale de l'axiome d'effets positifs, puis négatifs, pour les trois fluents :

Sur $Sur(x, y, do(a, s)) \Leftarrow [a = PoserSur(x, y) \wedge Poss(a, s)]$.
$\neg Sur(x, y, do(a, s)) \Leftarrow [((\exists z \neq y\ \ a = PoserSur(x, z)) \vee a = PoserSurTable(x)) \wedge Poss(a, s)]$[3].

SurTable $SurTable(x, do(a, s)) \Leftarrow [a = PoserSurTable(x) \wedge Poss(a, s)]$[4].
$\neg SurTable(x, do(a, s)) \Leftarrow [(\exists y\ \ a = PoserSur(x, y)) \wedge Poss(a, s)]$[5].

Libre $Libre(x, do(a, s)) \Leftarrow [Poss(a, s) \wedge \exists yz(Sur(y, x, s) \wedge (a = PoserSur(y, z) \vee a = PoserSurTable(y)))]$[6].
$\neg Libre(x, do(a, s)) \Leftarrow [Poss(a, s) \wedge (\exists\, y\ \ a = PoserSur(y, x))]$[7].

---

[1] On respecte les données qui permettent de poser sur la table un bloc qui s'y trouve déjà. Si on voulait exclure cette possibilité, il suffirait d'ajouter : $\wedge \neg Surtable(x, s)$.

[2] Ici encore, on respecte les données, qui n'exigent pas la contrainte plus forte $\forall\, x\ y\ s\ \ \neg(Sur(x, y, s) \wedge Sur(y, x, s))$. Cette contrainte peut sembler souhaitable. Remarquons toutefois que, *à condition que la situation initiale respecte cette contrainte*, les axiomes de pré-conditions d'actions font que toute situation la respectera. D'autres contraintes souhaitables de ce genre ont été omises dans l'énoncé, afin de ne pas trop le surcharger (pouvez-vous en citer une ?).

[3] On aurait pu ajouter $\wedge Sur(x, y, s)$ mais c'est facultatif, au sens où cela ne modifie pas la formule de succession d'état déduite en question 2.2.3(2), et donc il est préférable de ne pas l'ajouter ici.

[4] Il est préférable de ne pas ajouter $\wedge \neg SurTable(x, s)$ (cf. note 3).

[5] l est préférable de ne pas ajouter $\wedge SurTable(x, s)$ (cf. note 3)

[6] $\exists\, y\ \ Sur(y, x, s)$ équivaut à $\neg Libre(x, s)$ et donc on pourrait croire qu'il est possible de s'en dispenser là encore (cf. note 3), mais *il est indispensable de nommer y* pour écrire la formule complète ici.

[7] Il est préférable de ne pas ajouter $\wedge Libre(x, s)$ (cf. note 3)

Question 2.2.3

1. Grâce aux contraintes (C4) et (C5), traduites respectivement par les deux dernières formules données ci-dessus en question 2.2.1(3), on peut supprimer les fluents $Libre$ et $SurTable$, ne conservant que le fluent Sur.

   Voici comment on peut alors réécrire les formes générales des deux axiomes d'effets du fluent Sur :
   $Sur(x, y, do(a, s)) \Leftarrow [a = PoserSur(x, y) \wedge (Libre(x, s) \wedge Libre(y, s) \wedge x \neq y)]$, soit
   $Sur(x, y, do(a, s)) \Leftarrow [a = PoserSur(x, y) \wedge \forall z \neg Sur(z, x, s) \wedge \forall z \neg Sur(z, y, s) \wedge x \neq y]$.

   $\neg Sur(x, y, do(a, s)) \Leftarrow [(\exists z \neq y \;\; a = PoserSur(x, z) \wedge Poss(a, s)) \vee (a = PoserSurTable(x) \wedge Poss(a, s))]$, soit
   $\neg Sur(x, y, do(a, s)) \Leftarrow [(\exists z \neq y \;\; a = PoserSur(x, z) \wedge Libre(x, s) \wedge Libre(z, s) \wedge x \neq z) \vee (a = PoserSurTable(x) \wedge Libre(x, s))]$, soit
   $\neg Sur(x, y, do(a, s)) \Leftarrow [(\exists z \neq y \;\; a = PoserSur(x, z) \wedge \forall x' \neg Sur(x', x, s) \wedge \forall x' \neg Sur(x', z, s) \wedge x \neq z) \vee (a = PoserSurTable(x) \wedge \forall x' \neg Sur(x', x, s))]$.

2. Voici alors l'axiome de succession d'état du fluent Sur, en faisant l'hypothèse de fermeture causale :
   $Sur(x, y, do(a, s)) \Leftrightarrow \{[a = PoserSur(x, y) \wedge \forall z(\neg Sur(z, x, s) \wedge \neg Sur(z, y, s)) \wedge x \neq y] \vee [Sur(x, y, s) \wedge \neg\exists z \; (a = PoserSur(x, z) \wedge \forall x'(\neg Sur(x', x, s) \wedge \neg Sur(x', z, s) \wedge x \neq z)) \wedge \neg(a = PoserSurTable(x) \wedge \forall x' \neg Sur(x', x, s))]\}$.

Question 2.2.4

1. Les formules des questions 2.2.1 (points (2) et (3)) et du 2.2.2 doivent être fournies par l'utilisateur. Ce sont les formules qui décrivent, assez naturellement, les règles du domaine précis considéré ici.

   Remarque : En réalité d'ailleurs, en ce qui concerne le 2.2.2(2) ce ne sont même pas les formules demandées ici qui doivent être fournies, mais seulement tous les "axiomes d'effets individuels" :
   $Sur(x, y, do(a, s)) \Leftarrow [a = PoserSur(x, y) \wedge Poss(a, s)]$;
   $\neg Sur(x, y, do(a, s)) \Leftarrow (\exists z \neq y \;\; a = PoserSur(x, z) \wedge Poss(a, s))$, et
   $\neg Sur(x, y, do(a, s)) \Leftarrow (a = PoserSurTable(x) \wedge Libre(x, s))$.
   Le système doit savoir combiner les deux dernières formule afin de trouver la forme générale fournie en 2.2.2(2) [cf Prolog par exemple où cela se fait systématiquement et de façon naturelle]. Ici, il n'y a qu'un "axiome d'effet individuel" positif, qui conduit donc directement à la forme générale, et deux "axiomes d'effets individuels" négatifs.

   Les formules du 2.2.3 doivent pouvoir être calculées par le système : en particulier le point (2) transforme les données naturelles en une formule logique plus complexe qui intègre la notion de "persistance" ou "d'inertie" (grâce à la "fermeture causale").

2. Les fluents "inutiles" peuvent servir à décrire une situation particulière, surtout quand on considère un bloc précis : $Libre(x, s)$ ne se réfère explicitement qu'à une seul bloc, celui représenté par la variable d'objet x, tandis que la formule équivalente $\forall y \neg Sur(y, x, s)$ se réfère explicitement à tous les blocs. Il en est de même pour $SurTable(x, s)$. Ces deux "abréviations" ont donc un intérêt pratique certain.

3. Le problème de la qualification des actions est ici considéré comme résolu par les données elles-mêmes : les données se traduisent par des équivalences pour le prédicat $Poss$, ce qui signifie que l'on connaît exactement toutes les conditions qui rendent une action possible.

Par contre, le problème du cadre (ou de l'inertie : seules les choses dont les données mentionnent directement ou non qu'elles peuvent être modifiées sont effectivement modifiables) est traduit par le système. L'utilisateur fournit comme données les axiomes d'effets (positifs et négatifs) de chaque fluent, et le système, en utilisant une formalisation de l'hypothèse de fermeture causale, traduit cela en une formule plus complexe, l'axiome de succession d'état.

Question 2.2.5 Appelons F_0 la formule voulue ici :

$$Sur(x, y, S_0) \Leftrightarrow [(x = A \wedge y = B) \vee (x = B \wedge y = C) \vee (x = D \wedge y = E)].$$

Remarque : Grâce à la formalisation de (C5) fournie en 2.2.1(3), F_0 implique

$$SurTable(x, S_0) \Leftrightarrow (x = C \vee x = E).$$

Question 2.2.6 Soit $F(s)$ la formule décrivant la situation "finale" voulue :

$$Sur(x, y, s) \Leftrightarrow [(x = A \wedge y = B) \vee (x = B \wedge y = C) \vee (x = C \wedge y = D) \vee (x = D \wedge y = E)].$$

Là encore, grâce à la formalisation de (C5) fournie en 2.2.1(3), $F(s)$ implique

$$SurTable(x, s) \Leftrightarrow (x = E).$$

Une solution consiste à ajouter F_0 aux formules données en 2.2.1 et 2.2.2, et à appeler l'action complexe suivante :

$$[(\pi x)\ [\ (\ (\pi y)\ PoserSur(x, y)\ )\ |\ PoserSurTable(x)\ ]\ ]^*\ ;\ F?$$

F désigne ici le "pseudo fluent" constitué de la formule $F(s)$ dans laquelle on a supprimé toute référence à s, comme indiqué dans le cours à propos des "pseudo fluents".

Remarque : Bien sûr, une écriture aussi concise est surtout intéressante si le système sait optimiser la recherche de l'action $[(\pi x)((\pi y)PoserSur(x, y)|PoserSurTable(x))]^*$ afin de satisfaire le test final $(F?)$. Un des objectifs de l'équipe qui a conçu GOLOG est de permettre à un utilisateur de programmer un robot pour effectuer des actions complexes en utilisant naturellement des commandes aussi faciles à exprimer que celle-là.