Intelligence artificielle
ift615
Annexe au Projet 3

1 Introduction

Nous avons vu qu' système expert est composé d'un moteur d'inférence et d'une base de connaissances. La base de connaissance est un ensemble de règles représentant l'expertise proprement dite. Le moteur d'inférence est un programme pour manipuler ces règles, afin de tirer des conclusions, répondre à des questions et faire du diagnostique. Le moteur d'inférence constitue la partie *indépendante des applications* : il peut-être appliqué à des domaines différents en changeant simplement la base des connaissances.

Ce document vous donne quelques suggestions pour passer de A∗ à un moteur d'inférence. Ce ne sont que des suggestions. Il n'y a aucune obligation pour les suivre. Comme vous le verrez ci-après, ces suggestions ne vous donnent pas un moteur d'infrence très efficace. Mais c'est un bon point de départ.

2 De A∗ à un moteur d'inférence

Pour utiliser A^* comme un moteur d'inférence, vous pouvez définir ses trois paramètres (état initial, fonction de *test du but* et fonction *successeur*) en tenant compte des aspects suivants :

1. Les règles de la base de connaissances sont stockées dans une variable globale **rules** et les faits dans une autre variable globale **facts**.

   Vous pouvez voir ces variables juste comme des vecteurs ou des listes. Mieux encore, vous pouvez utiliser des structures plus efficaces. Une règle devrait contenir au moins trois éléments : la précondition, la postcondition et un commentaire (par exemple, comme commentaire,

vous pouvez mettre juste des numéros de règles; ces commentaires serviront pour afficher un diagnostique compréhensible pour les usagers). La précondition est un vecteur ou une liste de faits, alors que la postcondition c'est juste un fait. Un fait, à son tour, c'est juste un littéral. Enfin, un littéral est un prédicat ou la négation d'un prédicat.

2. Un état de l'espace de recherche est une pile de faits (c-à-d., pile des sous-buts), représentant la *mémoire de travail* du système expert.

3. L'état initial contient le fait exprimant le problème à diagnostiquer.

4. La fonction de *test du but* prend un état comme argument et retourne *vrai* si l'état est vide.

5. La fonction *successeur* prend un état s comme argument et retourne un vecteur ou une liste d'éléments, telle que chaque élément est à son tour un vecteur ou une liste contenant, dans l'ordre, un état successeur s' et un unificateur (c-à-d., substitution) représentant le nom que A^* va donner à la transition entre s et s'.

   Donc, étant donné un état s comme argument et les variables globales **facts** et **rules**, la fonction *successeur* est définie comme suit :

   (a) Soit g le but en tête de la pile de l'état s.

   (b) Pour chaque fait unifiable avec g, on obtient un successeur correspondant s' comme suit :

      i. Supprimez g de s pour obtenir un état s'';
      ii. trouvez l'unificateur le plus général (upg) du g avec le fait;
      iii. enfin, appliquez l'upg à chaque fait dans s'' pour obtenir s'.

   (c) Pour chaque règle dont la postcondition est unifiable avec g, on obtient un état successeur s' et l'unificateur correspondant comme suit :

      i. supprimez g de s;
      ii. trouvez l'unificateur le plus général (upg) de g avec la postcondition de la règle;
      iii. appliquez l'upg à chaque fait dans s pour obtenir un état s'';
      iv. appliquez l'upg à chaque précondition de la règle pour obtenir une liste P de préconditions;

v. renommez les variables dans P sauf celles qui viennet d'etre modifiées par l'upg;[1]

vi. enfin, ajouter les faits dans P en tête de s'' pour obtenir s'.

(d) Le résultat retourné par la fonction *successeur* est la liste de paires etat/nom-de-transition obtenues en associant les s' calculés aux deux étapes précédentes avec des noms de transitions qui sont, pour chaque état, une paire composée de l'unificateur correspondant et de la règle ou du fait correspondant.

(e) Si cette liste est vide, c-à-d., si g ne peut être unifié avec aucune règle ou aucun fait, alors la fonction *successeur* doit afficher un message demandant à l'usager de préciser si "oui" ou "non" g est vrai et pour quelle instantiation. Si la réponse est "oui", on procède comme si g (ou l'instantiation de g) se trouvait dans **facts**. Sinon, on retourne une liste vide.

Par exemple, supposons qu'on fait le diagnostic d'un véhicule et qu'il n'y a pas de règle ni de fait unifiable avec "allumes(phares)". Dans ce cas, la fonction *successeur* doit poser la question à l'usager, par exemple en affichant le message

allumes(phares) ?

Si l'usager répond "oui", alors la fonction *successeur* retourne une liste contenant un seul successeur s' obtenu en supprimant le sous-but de s. Le fait devrait être ajouté à **facts**. Par contre, si l'usager répond "non", la fonction *successeur* retourne une liste vide.

L'affichage des questions peut se faire de manière plus expressive. Par exemple, au lieu d'afficher

allumes(phares) ?

la fonction *successeur* devrait afficher

Est-ce que les phares sont allumés ?

[1] Pour renommer les variables, il vous faudra une fonction *rename-variables* qui prend comme argument une expression (*exp*) et, optionnelement, une liste de variables (*except*); elle retourne une nouvelle expression equivalente à celle donnee, sauf que toutes les variables ne se trouvant pas dans *except* sont renommées. Dans votre cas, *except* contiendra les variables $?x$ telles qu'on a une paire ($?z$ $?x$) dans l'upg pour un $?z$ quelconque. Un exemplaire de la fonction (en Scheme) est disponible sur le site Web (section Projet 3) à titre indicatif.

Les commentaires associés aux règles peuvent intervenir pour afficher des messages plus expressifs.

3 Exemple : diagnostique d'un véhicule

Dans l'exemple suivant, je ferai abstraction des commentaires associés aux règles dans la base de connaissance et aux transitions dans A^*.

3.1 Base de Connaissances

1. IF atteint_essence(moteur) AND not(demarre(moteur))
   THEN probleme_est(bougies)
2. IF demarre(moteur) AND not(allumes(phares))
   THEN probleme_est(batterie)
3. IF not(demarre(moteur)) AND allumes(phares)
   THEN probleme_est(demarreur)
4. IF il_y_a_essence(reservoir_essence) AND il_y_a_essence(carburateur)
   THEN atteint_essence(moteur)

3.2 Comment ça fonctionne ?

Supposons que vous avez codé ces règles dans la variable **rules** et que vous avez mis les faits que vous connaissez dans la variable **facts**. Pour démarrer le diagnostique d'un problème avec un véhicule, vous appelez simplement A^*, avec comme arguments l'état initial ((*probleme_est* ?x)), la fonction de *test du but* définie dans l'étape 4 de la section 3.1 et la fonction *successeur* définie dans l'étape 5 de la section 3.1.

Initialement la variable *open* de A^* contient uniquement l'état correspondant au problème initial :

$$open = (\ (probleme\_est\ ?x)\ )$$

Ensuite, pour expanser l'état initial, A^* unifie le fait en tête avec les postconditions des règles. Dans notre cas, (*probleme_est* ?x) est unifiable avec la postcondition des règles 1, 2 et 3. Ces unifications donnent la liste des successeurs

$$\begin{array}{l}(\ ((atteint\_essence\ moteur)\ (not\ (demarre\ moteur)))\\ \ ((demarre\ moteur)\ (not\ (allumes\ phares)))\\ \ ((not\ (demarre\ moteur))\ (allumes\ phares))\ ).\end{array}$$

En fait, chaque successeur devrait être associé avec l'unificateur correspondnat, c-à-d., respectivement ((?*x bougies*)), ((?*x batterie*)) et ((?*x demarreur*)). Mais, pour plus de clarité, je vais en faire abstraction dans ce scénario.

Puisque l'état expansé est enlevé de *open* et que ses successeurs sont ajoutés, après l'expansion de l'état initial *open* devient :

(((*atteint_essence moteur*) (*not* (*demarre moteur*)))
((*demarre moteur*) (*not* (*allumes phares*)))
((*not* (*demarre moteur*)) (*allumes phares*))).

L'itération suivante de A^* choisit l'état en tête de *open* , c-à-d.,

((*atteint_essence moteur*) (*not* (*demarre moteur*))).

Comme le fait en tête de cet état, (*atteint_essence moteur*), peut être unifié seulement avec la règle 4 (avec l'unificateur vide), on obtient un seul successeur en enlevant le fait en tête de l'état et en ajoutant ensuite les préconditions de la règle 4 :

((*il_y_a_essence reservoir_essence*) (*il_y_a_essence carburateur*) (*not* (*demarre moteur*))).

Puisque l'état expansé est enlevé de *open* et que les successeurs obtenus sont ajoutés en tête, maintenant *open* vaut :

(((*il_y_a_essence reservoir_essence*) (*il_y_a_essence carburateur*) (*not* (*demarre moteur*)))
((*demarre moteur*) (*not* (*allumes phares*)))
((*not* (*demarre moteur*)) (*allumes phares*))).

L'itération suivante de A^* choisit l'état en tête de *open*, c-à-d.,

(*il_y_a_essence reservoir_essence*) (*il_y_a_essence carburateur*) (*not* (*demarre moteur*))).

Puisque aucune règle n'est unifiable avec le premier fait, c-à-d.,

(*il_y_a_essence reservoir_essence*),

la fonction *successeur* doit poser la question :

(il_y_a_essence reservoir_essence) ?

Supposons qu'on réponde "oui". Dans ce cas, en enlevant le premier sous-but de l'état courant[2], on obtient une liste de successeurs avec un seul état

(((*il_y_a_essence carburateur*) (*not* (*demarre moteur*)))).

[2]La réponse "oui" signifie que, dorénavant, le sous-but fait partie de la base des faits **facts*

Donc *open* devient :

(((*il_y_a_essence carburateur*) (*not* (*demarre moteur*)))
((*demarre moteur*) (*not* (*allumes phares*)))
((*not* (*demarre moteur*)) (*allumes phares*))).

L'itération suivante de A^* conduira à la question (*il_y_a_essence carburateur*). Supposons une réponse positive.

Ainsi, à l'itération suivante l'état en tête de *open* sera ((*not* (*demarre moteur*))). Cela conduit éventuellement à la question

(not (demarre moteur)) ?

Supposons qu'on réponde "non". Dans ce cas, on n'a pas de successeur, donc *open* devient :

(((*demarre moteur*) (*not* (*allumes phares*)))
((*not* (*demarre moteur*)) (*allumes phares*))).

L'itération suivante de A^* choisit encore l'état en tête de *open* . Ceci conduit à la question

(demarre moteur) ?

Supposons encore une réponse positive.[3] En conséquence, *open* devient :

(((*not* (*allumes phares*)))
((*not* (*demarre moteur*)) (*allumes phares*))).

Finalement la prochaine itération conduit à la question :

(not (allumes phares)) ?

En supposant une réponse positive, *open* devient :

(()
((*not* (*demarre moteur*)) (*allumes phares*))).

Puisque on vient d'atteindre un état vide, A^* s'arrête. Ainsi le système expert peut répondre

[3]En fait, si l'usager reste cohérent, c'est la seule réponse possible vu que le système a déjà posé la question (*not*(*demarre moteur*)) auparavant et que la réponse "non" a été donnée. Dans un cas pareil, puisque les réponses sont stockées dans **facts**, une implémentation plus judicieuse de la fonction *successeur* devrait permettre de conclure qu'en fait (*not* (*demarre moteur*)) est vrai sans devoir poser la question à l'usager.

(probleme_est batterie)

On peut en plus donner une explication du diagnostique, puisque A^* garde une trace du chemin qui mène de l'état initial à l'état final. Comme on le sait, dans A^* cette trace est gardée automatiquement par le mécanisme des pointeurs des états vers leurs parents optimaux.[4]

À vous de faire preuve d'imagination pour programmer une fonction *successeur* ayant autant d'options intéressantes que possibles. J'ai mentionné certaines options en classe. Par exemple, vous pourriez afficher des message plus parlant plutôt que simplement d'afficher des faits du genre

(not (allumes phares)) ?

Vous pourriez aussi faire en sorte que l'usager puisse questionner le système expert sur le fondement de certaines questions que ce dernier pose aussi à l'usager. Par exemple, quand le système expert pose la question

(not (allumes phares)) ?

l'usager pourrait répondre "pourquoi ?" En se basant sur la règle, ceci provoquerait une réponse de la part du système expert du genre "puisque le moteur ne démarre pas, si les phares ne sont pas allumées, je pourrai conclure que le problème vient de la batterie".

Puisqu'on ne vise pas à trouver un chemin optimal, au lieu de A^* il est souvent plus efficace de faire une recherche en profondeur, avec une pile à la place de *open*, mais sans *close*. Vous pouvez obtenir une telle implémentation facilement en s'inspirant de ce qui précéde.

[4]En fait, on pourrait garder une trace autrement en associant à chaque état la règle appliquée et l'instatiation des variables.