Subsomptions en Prolog

Introduction

Le but de ce TME est de programmer en Prolog un algorithme permettant d'inférer des subsomptions pour la logique de description $\mathcal{FL}^-$. On procédera de façon progressive en ajoutant des règles pour gérer des cas de plus en plus complexes. On s'assurera toujours de faire des inférences correctes : si on trouve que C subsume D selon nos règles, on aura la garantie que C subsume bien D. L'inverse ne sera pas forcément vrai, le programme, construit progressivement, pouvant être incomplet.

Représentation

Exercice 1 Représentation préfixe en *Prolog*

Implanter les connaissances de l'exercice de TD sur les animaux (TD 4 Exercice 4) en *Prolog* dans une représentation préfixe. On traduit les différents opérateurs de concepts et éléments des T-Box et A-Box de la façon suivante :

	$\mathcal{FL}^-$	*Prolog*
Intersection	$C \sqcap D$	`and(C, D)`
Quantification existentielle	$\exists R$	`some(R)`
Quantification universelle	$\forall R.C$	`all(R,C)`
Subsomption	${}^{T}\boxed{C \sqsubseteq D}$	`subs(C,D)`
Equivalence	${}^{T}\boxed{C \equiv D}$	`equiv(C,D)`
Instance de concept	${}^{A}\boxed{I : C}$	`inst(IND,C)`
Instance de rôle	${}^{A}\boxed{\langle I, J\rangle : R}$	`instR(I,R,J)`

Subsomption structurelle pour $\mathcal{FL}^-$

Le but de cette section est d'écrire un ensemble de règles permettant de répondre à des questions du type "est-ce qu'on peut prouver que C est subsumé par D ?" (soit "est-ce que $C \sqsubseteq_s D$?")

Exercice 2 Concepts atomiques

On suppose dans un premier temps que la T-Box ne contient que des expressions du type $A \sqsubseteq B$ où A et B sont des concepts atomiques et que C et D sont aussi atomiques. Pour vérifier si $C \sqsubseteq_s D$ dans ce contexte, on propose les règles suivantes :

```
subsS1(C,C).
subsS1(C,D):-subs(C,D),C\==D.
subsS1(C,D):-subs(C,E),subsS1(E,D).
```

1. Que traduisent ces règles ?
2. Tester ces règles sur les requêtes $canari \sqsubseteq_s animal$, $chat \sqsubseteq_s etreVivant$.

3*. Tester la requête $chien \sqsubseteq_s souris$. Quel problème pose cette requête ?

Pour corriger ce problème, on introduit un troisième argument contenant la liste des subsomptions déjà faites dans une branche. On définit `subsS(C,D)`, qui doit être vérifié quand $C \sqsubseteq_s D$ avec le prédicat auxiliaire `subsS(C,D,L)` où L contient la liste des concepts utilisés dans la preuve de la subsomption. Pour éviter des preuves infinies, on s'interdira toujours de réutiliser un concept déjà présent dans L.

On réécrit donc les règles sur `subsS1(C,D)` pour définir `subsS(C,D,L)`, en rajoutant avant tout appel récursif $E \sqsubseteq_s D$ la condition que E ne soit pas dans L et en ajoutant E à L dans l'appel récursif :

```
subsS(C,D) :- subsS(C,D,[C]).
subsS(C,C,_).
subsS(C,D,_):-subs(C,D),C\==D.
subsS(C,D,L):-subs(C,E),not(member(E,L),subsS(E,D,[E|L]),E\==D.
```

4. Tester ces nouvelles règles avec *canari* $\sqsubseteq_s$ *animal*, *chat* $\sqsubseteq_s$ *etreVivant*, *chien* $\sqsubseteq_s$ *canide*, et *chien* $\sqsubseteq_s$ *chien*. Toutes ces requêtes doivent réussir.
5. Tester les requêtes *souris* $\sqsubseteq_s \exists$*mange* et *lion* $\sqsubseteq_s \forall$*mange.animal*. Pourquoi ces requêtes réussissent-elle bien que $\exists$*mange* ne soit pas un concept atomique ?
6. Tester les requêtes *chat* $\sqsubseteq_s$ *humain* et *chien* $\sqsubseteq_s$ *souris* qui doivent échouer.
7*. Que devraient renvoyer les requêtes *chat* $\sqsubseteq_s X$ et $X \sqsubseteq_s mammifere$? Vérifier que c'est bien le cas (il n'est pas nécessaire d'éliminer les réponses doubles ou le false final).

On suppose maintenant que la T-Box peut aussi contenir des expressions du type $A \equiv B$.

8. Ecrire des règles permettant de dériver $A \sqsubseteq B$ et $B \sqsubseteq A$ à partir de $A \equiv B$.
9. Tester avec la requête *lion* $\sqsubseteq_s \forall$*mange.animal*.
10*. A-t-on plus intérêt à dériver ${}^{T}\boxed{A \sqsubseteq B}$ ou $A \sqsubseteq_s B$? pourquoi ? cela, cumulé aux règles précédentes, suffit-il à répondre à toute requête atomique ?

Exercice 3 Gestion de l'intersection

On s'intéresse maintenant aux requêtes contenant des intersections de concepts. On étend donc la définition du prédicat `subsS` de la façon suivante.

```
subsS(C,and(D1,D2),L):-D1\=D2,subsS(C,D1,L),subsS(C,D2,L).
subsS(C,D,L):-subs(and(D1,D2),D),E=and(D1,D2),not(member(E,L)), subsS(C,E,[E|L]),E\==C.
subsS(and(C,C),D,L):-nonvar(C),subsS(C,D,[C|L]).
subsS(and(C1,C2),D,L):-C1\=C2,subsS(C1,D,[C1|L]).
subsS(and(C1,C2),D,L):-C1\=C2,subsS(C2,D,[C2|L]).
subsS(and(C1,C2),D,L):-subs(C1,E1),E=and(E1,C2),not(member(E,L)),subsS(E,D,[E|L]),E\==D.
subsS(and(C1,C2),D,L):-Cinv=and(C2,C1),not(member(Cinv,L)),subsS(Cinv,D,[Cinv|L]).
```

1. Tester cet ensemble de règles avec les requêtes *chihuahua* $\sqsubseteq_s mammifere \sqcap \exists aMaitre$, *chien* $\sqcap$ $\exists aMaitre \sqsubseteq_s$ *pet* et *chihuahua* $\sqsubseteq_s$ *pet* $\sqcap$ *chien*.
2*. Pour chacune des règles, indiquer la situation traitée par la règle et donner un exemple de requête qui ne marcherait pas sans la règle.
3. Ces règles suffisent-elle pour gérer toute requête ne contenant que des concepts atomiques ou avec intersection vis-à-vis d'une T-Box ne contenant que des subsomptions entre concepts atomiques ou avec intersection ?

Exercice 4 Gestion des rôles

On s'intéresse maintenant aux requêtes contenant des rôles qualifiés ($\forall R.C$).

1. Ecrire une règle permettant de répondre à une requête du type $\forall R.C \sqsubseteq_s \forall R.D$, où C et D sont des concepts à partir de C et D.
2. Tester les requêtes *lion* $\sqsubseteq_s \forall$*mange.etreVivant* et $\forall$*mange.canari* $\sqsubseteq_s$ *carnivoreExc*.

3. Tester les requêtes $carnivoreExc \sqcap herbivoreExc \sqsubseteq_s \forall mange.nothing$ et $(carnivoreExc \sqcap herbivoreExc) \sqcap animal \sqsubseteq_s nothing$. Si besoin, rajouter des règles pour qu'elles réussissent. Qu'en est-il de la requête $(carnivoreExc \sqcap animal) \sqcap herbivoreExc \sqsubseteq_s nothing$? pourquoi ? (on n'exige pas de modifier le programme pour que cette dernière requête réussisse).

4*. Ecrire une règle permettant de répondre à une requête du type $\forall R.I \sqsubseteq_s \forall R.C$ à partir de l'instance I et du concept C en faisant appel aux assertions de la A-Box.

5. Est-il nécessaire d'écrire des règles similaires pour les concepts de la forme $\exists R$?

6. Que devraient renvoyer les requêtes $lion \sqsubseteq_s X$ et $X \sqsubseteq_s predateur$? Voir si c'est bien le cas avec vos règles (il n'est pas nécessaire d'éliminer les réponses doubles ou le false final).

Exercice 5 Complétude

Un ensemble de règles ainsi produit est complet pour un langage donné s'il peut prouver toute subsomption $C \sqsubseteq D$ correcte à partir du moment où tous les termes de la T-Box et de la requête appartiennent au langage donné.

L'ensemble de règle que vous avez écrit est-il complet pour $\mathcal{FL}^-$?

Pour aller plus loin : Preuve d'instantiations

Cette partie n'est pas obligatoire.

Exercice 6 Gestion des instances

On veut pouvoir répondre à des requête sur des instances, pour demander si un individu I est une instance d'un concept plus général C. On note une telle requête de preuve d'instantiation $I, C :_s$.

1. Ecrire une ou plusieurs règles pour pouvoir répondre à une requête $I :_s C$ où C est un concept sans rôles. Il est possible de faire appel aux preuves de subsomptions précedemment définies.
2. Tester avec les requêtes $felix :_s mammifere$ et $princesse :_s pet$.
3. Ecrire une ou plusieurs règles pour pouvoir répondre à une requête $I :_s \exists R$ et tester les requêtes $felix :_s \exists mange$, $princess :_s \exists aMaitre$, et $felix :_s \exists aMaitre$.
4. On veut maintenant pouvoir vérifier des instances de concepts du type $\forall R.C$.
   (a) Définir un prédicat `contreExAll(I,R,C)` qui est vérifié quand il existe I_2 telle que ${}^{A}\boxed{\langle I, I_2\rangle : R}$ et I_2 n'est pas une instance de C.
   (b) Ecrire une règle qui vérifie $I :_s \forall R.C$ en utilisant `contreExAll(I,R,C)`.
   (c) Tester les requêtes $felix :_s \forall mange.animal$, $titi :_s \forall mange.personne$, qui doivent toutes deux réussir, et $felix :_s mammifere$, qui doit échouer.
5. Faut il modifier les règles de 4.4 pour utiliser le test d'instance structurelle plutôt que juste la A-Box ? Pourquoi ?
6. Que donnent les requêtes $felix :_s pet$ et $felix :_s carnivoreExc$? Que faudrait-il faire pour qu'elle réussissent ?