

Subsumptions en Prolog

Introduction

Le but de ce TME est de programmer en Prolog un algorithme permettant d'inférer des subsumptions pour la logique de description \mathcal{FL}^- . On procède de façon progressive en ajoutant des règles pour gérer des cas de plus en plus complexes. On s'assure toujours de faire des inférences correctes : si on trouve que C subsume D selon nos règles, on a la garantie que C subsume bien D . L'inverse n'est 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 2) 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}^-	<i>Prolog</i>
Intersection	$C \sqcap D$	<code>and(C, D)</code>
Quantification existentielle	$\exists R$	<code>some(R)</code>
Quantification universelle	$\forall R.C$	<code>all(R, C)</code>
Subsumption	$\begin{matrix} T \\ C \sqsubseteq D \end{matrix}$	<code>subs(C, D)</code>
Equivalence	$\begin{matrix} T \\ C \equiv D \end{matrix}$	<code>equiv(C, D)</code>
Instance de concept	$\begin{matrix} A \\ I : C \end{matrix}$	<code>inst(IND, C)</code>
Instance de rôle	$\begin{matrix} A \\ \langle I, J \rangle : R \end{matrix}$	<code>instR(I, R, J)</code>

Note : pour les chiens et canides, on demande de traduire séparément les deux phrases *alors que les chiens sont des canidés* et *Les seuls canidés considérés sont les chiens* par deux subsumptions (sans traduire cela en une équivalence).

Subsumption 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 $\text{canari} \sqsubseteq_s \text{animal}$, $\text{chat} \sqsubseteq_s \text{etreVivant}$.
- 3*. Tester la requête $\text{chien} \sqsubseteq_s \text{souris}$. 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 $\text{subsS}(C,D)$, qui doit être vérifié quand $C \sqsubseteq_s D$ avec le prédicat auxiliaire $\text{subsS}(C,D,L)$ où L contient la liste des concepts utilisés dans la preuve de la subsomption. Pour éviter des preuves infinies, on s'interdit toujours de réutiliser un concept déjà présent dans L .

On réécrit donc les règles sur $\text{subsS1}(C,D)$ pour définir $\text{subsS}(C,D,L)$, 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 $\text{canari} \sqsubseteq_s \text{animal}$, $\text{chat} \sqsubseteq_s \text{etreVivant}$, $\text{chien} \sqsubseteq_s \text{canide}$, et $\text{chien} \sqsubseteq_s \text{chien}$. Toutes ces requêtes doivent réussir.
5. Tester la requête $\text{souris} \sqsubseteq_s \exists \text{mange}$. Pourquoi cette requête réussit-elle bien que $\exists \text{mange}$ ne soit pas un concept atomique ?
6. Tester les requêtes $\text{chat} \sqsubseteq_s \text{humain}$ et $\text{chien} \sqsubseteq_s \text{souris}$ qui doivent échouer.
- 7*. Que devraient renvoyer les requêtes $\text{chat} \sqsubseteq_s X$ et $X \sqsubseteq_s \text{mammifere}$? 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 avant et après ajout des règles avec la requête $\text{lion} \sqsubseteq_s \forall \text{mange.animal}$. Note : il est ici supposé que vous ayez bien traduit les définitions des carnivores et herbivores exclusifs des `equiv` et non par deux `subs`.
- 10*. A-t-on plus intérêt à dériver $\boxed{A \sqsubseteq B}$ 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. Il est donc important de travailler dans le même fichier et de garder le même nom de prédicat (`subsS`).

```

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 $\text{chihuahua} \sqsubseteq_s \text{mammifere} \sqcap \exists a \text{Maitre}$, $\text{chien} \sqcap \exists a \text{Maitre} \sqsubseteq_s \text{pet}$ et $\text{chihuahua} \sqsubseteq_s \text{pet} \sqcap \text{chien}$.
- 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 . Il s'agit là encore d'étendre la définition de `subsS` et non de créer un nouveau prédicat.
2. Tester les requêtes $\text{lion} \sqsubseteq_s \forall \text{mange.etreVivant}$ et $\forall \text{mange.canari} \sqsubseteq_s \text{carnivoreExc}$.

3. Tester les requêtes $carnivoreExc \sqcap herbivoreExc \sqsubseteq_s \forall \text{mange.nothing}$ et $(carnivoreExc \sqcap herbivoreExc) \sqcap animal \sqsubseteq_s \text{nothing}$. Si besoin, rajouter des règles pour qu'elles réussissent. Qu'en est-il de la requête $(carnivoreExc \sqcap animal) \sqcap herbivoreExc \sqsubseteq_s \text{nothing}$? 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 subsumption $C \sqsubseteq D$ correcte à partir du moment où tous les termes de la T-Box et de la requête appartiennent au langage donné.

L'ensemble de règles écrites 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 requêtes 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'instanciation $I, C :_s$.

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 subsumptions précédemment définies.
2. Tester avec les requêtes $felix :_s \text{mammifere}$ et $princesse :_s \text{pet}$.
3. Ecrire une ou plusieurs règles pour pouvoir répondre à une requête $I :_s \exists R$ et tester les requêtes $felix :_s \exists \text{mange}$, $princesse :_s \exists \text{aMaitre}$, et $felix :_s \exists \text{aMaitre}$.
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 tel que $\boxed{A \langle I, I_2 \rangle : R}$ 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 $felix :_s \forall \text{mange.animal}$, $titi :_s \forall \text{mange.personne}$, qui doivent toutes deux réussir, et $felix :_s \text{mammifere}$, 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 $felix :_s \text{pet}$ et $felix :_s \text{carnivoreExc}$? Que faudrait-il faire pour qu'elles réussissent?