SORBONNE UNIVERSITÉ

Projet LRC

Logique et représentation des connaissances

Subsomptions en Prolog

Antoine Théologien & Gabriel Zamy

Décembre 2024

Introduction

Ce rapport est consacré au projet de LRC portant sur les subsombtions en prolog. Nous y apporterons les réponses aux différents exercices, principalement aux questions mises en commentaires.

Exercice 1

subs(chat, felin)

- Traduction : Les chats sont des félins
- FL :

$$\text{chat} \sqsubseteq \text{felin}$$

subs(chihuahua, and(chien, pet))

- Traduction : Un chihuahua est à la fois un chien et un animal de compagnie.
- FL :

$$\text{chihuahua} \sqsubseteq (\text{chien} \sqcap \text{pet})$$

subs(and(animal, some(aMaitre)), pet)

- Traduction : Un animal qui a un maˆıtre est un animal de compagnie.
- FL :

$$(\text{animal} \sqcap \exists \text{aMaitre}) \sqsubseteq \text{pet}$$

subs(some(aMaitre), all(aMaitre, personne))

- Traduction : Toute entité qui a un maˆıtre ne peut avoir qu'un (ou plusieurs) maˆıtre(s) humain(s)
- FL :

$$\exists \text{aMaitre} \sqsubseteq \forall \text{aMaitre.personne}$$

subs(and(all(mange, nothing), some(mange)), nothing)

- Traduction : On ne peut pas à la fois ne rien manger (ne manger que des choses qui n'existent pas) et manger quelque chose
- FL :

$$(\forall \text{mange.nothing} \sqcap \exists \text{mange}) \sqsubseteq \text{nothing}$$

equiv(carnivoreExc, all(mange, animal))

- Traduction : Un carnivore exclusif est défini comme une entité qui mange uniquement des animaux
- FL :

$$\text{carnivoreExc} \equiv \forall \text{mange.animal}$$

Exercice 2

1.

subsS1(C, C).
Traduction : Tout concept C est subsumé par lui-même ($C \sqsubseteq_s C$).

subsS1(C, D) :- subs(C, D), C \== D.
Traduction : Si $C \sqsubseteq D$ (c'est-à-dire que subs(C, D) est explicitement dans la TBox) et que $C \neq D$, alors $C \sqsubseteq_s D$.

subsS1(C, D) :- subs(C, E), subsS1(E, D).
Traduction : Si $C \sqsubseteq E$ et $E \sqsubseteq_s D$, alors $C \sqsubseteq_s D$. (Cette règle exploite la transitivité.)

Pour les requêtes $canari \sqsubseteq_s animal$ et $chat \sqsubseteq_s etreVivant$, on obtient true, true, et enfin false qui correspond à la fin du parcours de l'arbre.

2.

En utilisant le mode *trace*, on remarque que la requête $chien \sqsubseteq_s souris$ semble tomber dans une boucle infinie, qui pourrait être causée par le fait que l'on ne vérifie pas si une subsomption à déjà été testée ou non.

3.

On teste donc les requêtes suivantes :

- Pour $chat \sqsubseteq_s etreVivant$, on attend le résultat $true$, on obtient bien $true$.
- Pour $chien \sqsubseteq_s canide$, on attend le résultat $true$, on obtient bien $true$.
- Pour $chien \sqsubseteq_s chien$, on attend le résultat $true$, on obtient bien $true$.
- Pour $chien \sqsubseteq_s souris$, on attend donc $false$, que l'on obtient bien en exécutant la requête. En observant la trace, on remarque qu'à chaque exploration, le concept depuis lequel l'arbre est exploré est bien ajouté dans la liste des concepts visités. Cela permet d'éviter les boucles infinies que nous avions précédemment. Comme aucune règle ne permet de vérifier la requête, on obtient bien le résultat $false$.

4.

En testant la requête $souris \sqsubseteq_s \exists mange$, on obtient bien $true$. Cela fonctionne car la règle $subsS1(C, D) : -subs(C, D), C \equiv D.$ n'oblige pas D a être un concept atomique. Ainsi, comme on a la règle : $subs(animal, some(mange))$, et qu'une souris est bien un animal, on obtient $true$.

5.

- Pour $chat \sqsubseteq_s X$, on devrait obtenir :
  - $X = chat$;
  - $X = felin$;
  - $X = mammifer$;
  - $X = animal$;
  - $X = etreVivant$;
  - $X = some(mange)$;
  - $false$;

En testant, on obtient bien ce résultat.

- Pour $X \sqsubseteq_s mammifere$, on devrait obtenir :
  - $X = felin$;
  - $X = canide$;
  - $X = souris$;
  - $X = chat$;
  - $X = chien$;
  - $X = lion$;
  - $false$;

  En testant, on obtient bien ce résultat.

6.

On travaille donc avec la requête suivante : $lion \sqsubseteq_s \forall mange.animal$. Avant l'ajout, on obtient $false$.
Après l'ajout des règles dans le fichier prolog, on obtient donc $true$, puisque les subsomptions sont maintenant définies grâce aux équivalences.

Exercice 3

1.

En testant, les trois requêtes retournent toutes $true$.

2.

subsS(C, and(D1, D2), L) :- D1 $\bar{\text{D}}$2, subsS(C, D1, L), subsS(C, D2, L).

- Utilité : Cette règle permet de vérifier si un concept C est un sous-concept d'une conjonction de concepts D1 et D2.
- Echec sans la règle : subsS(chien,and(mammifere,animal))).

subsS(C, D, L) :- subs(and(D1, D2), D), E = and(D1, D2), not(member(E, L)), subsS(C, E, [E—L]), E = C.

- Utilité : Vérifie si un concept C est un sous-concept de D, lui-même défini comme une conjonction de concepts.
- Echec sans la règle : subsS(chien,pet)). Dans le cas où la base contient une règle comme subs(and(animal, some(aMaitre)), pet), Prolog ne pourrait pas inférer que chien est un sous-concept de pet via cette conjonction sans cette règle.

subsS(and(C, C), D, L) :- nonvar(C), subsS(C, D, [C—L]).

- Utilité : Gère le cas où l'on a une conjonction du même concept, C et C par exemple.
- Echec sans la règle : subsS(and(chien, chien), mammifere)).

subsS(and(C1, C2), D, L) :- C1 $\bar{\text{C}}$2, subsS(C1, D, [C1—L]). et subsS(and(C1, C2), D, L) :- C1 $\bar{\text{C}}$2, subsS(C2, D, [C2—L]).

- Utilité : Vérifient qu'une conjonction est un sous-concept d'un concept D, en examinant séparemment le concept de gauche et de droite.
- Echec sans la règle : subsS(and(chien,some(aMaitre),mammifere)).

subsS(and(C1, C2), D, L) :- subs(C1, E1), E = and(E1, C2), not(member(E, L)), subsS(E, D, [E—L]), E = D.

- Utilité : On gère ici le cas où une composante C1 peut être remplacées par un sous-concept E1, puis on teste là conjonction alors formée E1 et C2 pour la subsomption.
- Echec sans la règle : subsS(and(chien,some(aMaitre)),pet). Si la base contient subs(and(animal, some(aMaitre)), pet) et subs(felin, animal), Prolog ne pourrait pas conclure que and(felin, some(aMaitre)) est un sous-concept de pet sans cette règle, car elle ne traiterait pas la substitution de felin par animal dans la conjonction.

subsS(and(C1, C2), D, L) :- Cinv = and(C2, C1), not(member(Cinv, L)), subsS(Cinv, D, [Cinv—L]).

- Utilité : Permet de gérer l'ordre des concepts dans la conjonction.
- Echec sans la règle : subsS(and(some(aMaitre), chien), pet). Si la base contient subs(and(chien, some(aMaitre)), pet), Prolog ne pourrait pas conclure que and(some(aMaitre), chien) est un sous-concept de pet sans cette règle, car il ne considérerait pas l'inversion de l'ordre des concepts dans la conjonction.