



**SORBONNE
UNIVERSITÉ**

Projet LRC

Logique et représentation des connaissances

Subsumptions en Prolog

Antoine Théologien & Gabriel Zamy

Décembre 2024

Introduction

Ce rapport est consacré au projet de LRC portant sur les subsomptions 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 maître 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 maître ne peut avoir qu'un (ou plusieurs) maître(s) humain(s)
- FL :

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

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 :

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

equiv(carnivoreExc, all(mange, animal))

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

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

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 $\text{subs}(C, D)$ 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 $\text{canari} \sqsubseteq_s \text{animal}$ et $\text{chat} \sqsubseteq_s \text{etreVivant}$, 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 $\text{chien} \sqsubseteq_s \text{souris}$ semble tomber dans une boucle infinie, qui pourrait être causée par le fait que l'on ne vérifie pas si une subsumption a déjà été testée ou non.

3.

On teste donc les requêtes suivantes :

- Pour $\text{chat} \sqsubseteq_s \text{etreVivant}$, on attend le résultat *true*, on obtient bien *true*.
- Pour $\text{chien} \sqsubseteq_s \text{canide}$, on attend le résultat *true*, on obtient bien *true*.
- Pour $\text{chien} \sqsubseteq_s \text{chien}$, on attend le résultat *true*, on obtient bien *true*.
- Pour $\text{chien} \sqsubseteq_s \text{souris}$, 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 avons précédemment. Comme aucune règle ne permet de vérifier la requête, on obtient bien le résultat *false*.

4.

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

5.

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

En testant, on obtient bien ce résultat.

- Pour $X \sqsubseteq_s \text{mammifere}$, on devrait obtenir :

- $X = \text{felin}$;
- $X = \text{canide}$;
- $X = \text{souris}$;
- $X = \text{chat}$;
- $X = \text{chien}$;
- $X = \text{lion}$;
- false ;

En testant, on obtient bien ce résultat.

6.

On travaille donc avec la requête suivante : $\text{lion} \sqsubseteq_s \forall \text{mange}.\text{animal}$. 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 $\bar{\sqsubseteq}$ D2, 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 \equiv 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 $\bar{\sqsubseteq}$ C2, subsS(C1, D, [C1—L]). et subsS(and(C1, C2), D, L) :- C1 $\bar{\sqsubseteq}$ C2, subsS(C2, D, [C2—L]).

- Utilité : Vérifient qu'une conjonction est un sous-concept d'un concept D, en examinant séparément 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 remplacée par un sous-concept E1, puis on teste la 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.

Exercice 4

1.

On écrit alors la règle suivante : **subsS**(all(R, C), all(R, D), L) :- **subsS**(C, D, L).

2.

On teste alors les règles suivantes :

- – Requête : **subsS**(lion, all(mange, etreVivant)).
– Résultat : true
- – Requête : **subsS**(all(mange,canari),carnivoreExc).
- Résultat : true

3.

On teste alors les règles suivantes :

- – Requête : **subsS**(and(carnivoreExc, herbivoreExc), all(mange, nothing)).
– Résultat : true
- – Requête : ?- **subsS**(and(and(carnivoreExc, herbivoreExc), animal),nothing).
– Résultat : true

On a alors ajouté les règles suivantes pour gérer les incohérences des intersections :

- **subsS**(and(carnivoreExc, herbivoreExc), all(mange, nothing), L) :- not(member(nothing, L)).
- **subsS**(and(carnivoreExc, herbivoreExc), nothing, L) :- not(member(nothing, L)).
- **subsS**(and(and(carnivoreExc, herbivoreExc), animal), nothing, L) :- not(member(nothing, L)).
- **subsS**(and(all(R, nothing), some(R)), nothing, L) :- not(member(nothing, L)).

On teste enfin la dernière requête : **subsS**(and(and(carnivoreExc, animal), herbivoreExc), nothing), qui comme prévu échoue. Celle-ci échoue car le programme ne traite pas explicitement l'incohérence entre carnivoreExc et herbivoreExc lorsqu'ils sont combinés avec un autre concept, ici animal. De plus, carnivoreExc et herbivoreExc sont incohérents en eux-mêmes, mais cette incohérence n'est pas propagée par défaut.

4.

Non, il n'est pas nécessaire d'écrire des règles similaires pour les concepts de la forme \exists , puisque ces cas sont déjà gérés par les règles pour les subsumptions.

5.

- Pour $\text{lion} \sqsubseteq_s X$:
 - Requête : $\text{subsS}(\text{lion}, X)$.
 - Résultat attendu : toutes les subsumptions de lion, c'est à dire : félin, carnivoreExc, mammifere, animal, etreVivant, predateur, some(mange), et tous les all(mange,Y).
 - Résultat obtenu : Le résultat attendu : félin, carnivoreExc, mammifere, animal, etreVivant, predateur, some(mange), et les all(mange,Y).
- Pour $X \sqsubseteq_s \text{predateur}$
 - Requête : $\text{subsS}(X, \text{predateur})$.
 - Résultat attendu : Tous les concepts qui subsumptionnent predateur, donc carnivoreExc, et par conséquent les concepts qui en héritent.
 - Résultat obtenu : Le résultat attendu : carnivoreExc, lion..

Exercice 5

Pour que l'ensemble soit complet, il faut donc que l'on puisse prouver toute subsumption correcte à partir du moment où tous les termes sont associés dans la TBOX. Or, dans l'exercice 4, nous avons une requête qui avec nos informations incluses dans la TBOX devrait fonctionner, mais qui retourne tout de même false, puisque nous n'avons pas géré les cas d'incohérence entre carnivoreExc et HerbivoreExc et d'autres concepts. Par conséquent, nous ne pouvons pas affirmer que cet ensemble est complet pour FL-. Nous avons cependant toutes les autres règles qui nous permettent de gérer la sémantique de cette logique, notamment, pour les conjonctions et les opérateurs de type $\forall R$.