Logique et Représentation des Connaissances

Master DAC, Sorbonne Université, 2022-2023

Ben Kabongo (21116436)

Rapport

Ecriture en Prolog d'un démonstrateur basé sur l'algorithme des tableaux pour la logique de description ALC

Dans le cadre de l'unité d'enseignement **Logique et Représentation des Connaissances**, nous avons codé, en utilisant le langage de programmation **Prolog**, un démonstrateur de propositions basé sur l'algorithme des tableaux, en logique de description ALC.

Dans ce rapport, nous allons présenter les différents prédicats que nous avons implémentés dans notre démonstrateur de propositions, lister et expliquer les paramètres, ainsi que leurs utilités respectives.

Partie 1 : **Etape préliminaire de vérification et de mise en forme de la Tbox et de la Abox**

1. allConcepts(L) : **L** contient la liste des concepts atomiques et des concepts complexes définis dans la TBox.
2. allInstances(L) : **L** contient la liste des instances définies dans la ABox.
3. allRules(L) : **L** contient la liste des rôles définis dans la TBox.
4. isAtomicConcept(X), isComplexConcept(X), isConcept(X) : vérifient respectivement si **X** est un concept atomique, complexe et atomique ou complexe de la TBox.
5. isIstance(I), isRule(R) : vérifient respectivement si **I** est une instance de la ABox et si **R** un rôle de la TBox.

6. **concept(X)** : vérifie si X est une définition de concept sémantiquement et syntaxiquement correct, d'après la grammaire donnée par ALC et les conventions données dans l'énoncé.

   Le prédicat est vrai si X respecte les patrons suivants : *C, not(C), and(C1, C2), or(C1, C2), some(R, C), all(R, C), (I, C)* et *(I1, I2, R)*, où C, C1, C2 sont des concepts et R des rôles de la TBox (sinon échec), I, I1, I2, des instances de la ABox (sinon échec).

   La définition récursive permet de vérifier que si par exemple *concept(C1)* et *concept(C2)* sont vrais alors *concept(and(not(C1), or(C2, C1)))* est aussi une définition correcte de concept syntaxiquement et sémantiquement.

7. isIn(C1, C2) : vérifie si le concept **C1** apparaît dans la définition du concept **C2.** Le prédicat commence par tester l'égalité des deux concepts. Il est vrai s'ils sont égaux. Sinon, si C2 est un concept complexe, le prédicat considérera ensuite son écriture équivalente définie dans la TBox et comparera C1 à tous les concepts qui interviennent dans la définition de C2. Les récursions sont récursives. On s'arrête quand on a rencontré C1 ou si plus aucune substitution n'est possible (tous les concepts sont des concepts atomiques).

   Un concept est autoréférent s'il apparaît dans sa propre définition. Le prédicat *isIn* nous permet ensuite de facilement définir le prédicat *autoref*.

8. **autoref(X)** : vérifie si **X** est un concept autoréférent. Par définition, les concepts atomiques ne peuvent être autoréférents. Le prédicat vérifie donc s'il existe une définition du concept X dans la TBox et appelle ensuite le prédicat *isIn* pour déterminer si X apparaît dans sa propre définition donnée dans la TBox.

9. atomize(C, NC) : ce prédicat donne dans **NC** une écriture équivalente du concept **C** qui ne contient que des concepts atomiques et qui est sous forme normale négative.

10. **traitement_Tbox(TBox)** : ce prédicat donne la TBox sous forme de liste. Pour chaque définition *equiv(C1, C2)* donnée dans TBox, le prédicat renvoie *(C1, NC2)* où *NC2* est une réécriture sous forme normale négative de *C2* avec des concepts atomiques.

    Le prédicat utilise à cet effet les prédicats *atomize* pour la réécriture des concepts et le prédicat *element_TBox(L, TBox).* Ce dernier reçoit la liste des concepts complexes de la TBox et constitue la liste **TBox** comme expliqué ci-dessus.

11. traitement_i_Abox(Abi) : ce prédicat donne la liste des assertions des concepts de la ABox. Pour chaque *inst(I, C)* donnée dans la ABox, le prédicat renvoie *(I, NC) où NC* est une réécriture sous forme normale négative de C avec des concepts atomiques.

Le prédicat utilise à cet effet les prédicats *atomize* pour la réécriture des concepts et le prédicat *element_i_ABox(L, Abi)*. Ce dernier reçoit la liste de couple instance et concept et constitue la liste **Abi** comme expliqué ci-dessus.

12. traitement_r_Abox(Abr) : ce prédicat donne la liste des assertions des rôles de la ABox. Pour chaque *instR(I1, I2, R)* donnée dans la ABox, le prédicat renvoie *(I1, I2, R)* et constitue ainsi la liste **Abr.**

13. **traitement_Abox(Abi, Abr)** : ce prédicat utilise les prédicats *traitement_i_Abox* et *traitement_r_Abox* pour constituer respectivement la liste des assertions de concepts et la liste des assertions de rôles. **Abi** est la liste des assertions des concepts et **Abr** est la liste des assertions de rôles.

14. **premiere_etape(TBox, Abi, Abr)** : ce prédicat utilise les prédicats *traitement_Tbox* et *traitement_Abox* pour constituer respectivement les listes **TBox** pour la TBox, **Abi** pour les assertions de concepts de la ABox et **Abr** pour les assertions de rôles de la ABox.

Partie 2 : **Saisie de la proposition à démontrer**

15. saisie_identificateur(I) : demande à l'utilisateur de saisir un identificateur **I.**

16. saisie_concept(C) : demande à l'utilisateur de saisir un nom de concept **C** et recommence tant que le concept saisi n'est pas syntaxiquement et sémantiquement correct.

17. **aquisition_proposition_type1(Abi, Abe, TBox)** : l'utilisateur a choisi de saisir une proposition de type *I:C*. Le programme lui demande donc de saisir un identificateur puis un concept correct.

    Abi est la liste des assertions des concepts de la ABox, **TBox** est la liste des concepts de la TBox, **Abe** est la liste étendue des assertions des concepts.

    Après une saisie correcte d'une proposition *(I, C),* l'assertion de concept *(I, NNC)* est concatené avec **Abi** pour constituer la **Abe.** *NNC* est la négation de *C* réécrite sous forme normale négative avec des concepts atomiques de la TBox.

18. **aquisition_proposition_type2(Abi, Abe, TBox)** : l'utilisateur a choisi de saisir une proposition de type *C1* ⊓ *C2*. Le programme lui demande donc de saisir un premier concept correct *C1* puis un second *C2*.

    Abi est la liste des assertions des concepts de la ABox, **TBox** est la liste des concepts de la TBox, **Abe** est la liste étendue des assertions des concepts.

Après une saisie correcte des concepts *C1* et *C2* l'assertion de concept *(I, NNC)* est concatené avec **Abi** pour constituer la **Abe.** *I* est un nom d'instance généré automatiquement et *NNC* est la négation de *C1* ⊓ *C2* réécrite sous forme normale négative avec des concepts atomiques de la TBox.

19. **deuxieme_etape(TBox, Abi, Abr)** : ce prédicat utilise les prédicats définis ci-haut pour acquérir la proposition de l'utilisateur. La troisième étape consiste à utiliser la méthode des tableaux pour déterminer si l'assertion est vraie ou pas en fonction de la TBox et de la ABox.

Partie 3 : **Démonstration de la proposition**

20. **tri_Abox(Abi, Lie, Lpt, Li, Lu, Ls)** : ce prédicat effectue un tri des assertions de concepts de la ABox, représentée ici par la liste **Abi**.
    A la suite de ce tri, les listes suivantes contiennent chacune une forme donnée d'assertions de concepts :
    - **Lie :** some(R, C)
    - **Lpt :** all(R, C)
    - **Li :** and(C1, C2)
    - **Lu** : or(C1, C2)
    - **Ls** : not(C) ou C : avec *C* un concept atomique.

21. **evolue((I, C), Lie, Lpt, Li, Lu, Ls, Lie1, Lpt1, Li1, Lu1, Ls1)** : ce prédicat ajoute l'assertion de concept *(I, C)* dans la liste adéquate en fonction de sa forme. Les listes avec les suffixes 1 sont les listes de mises à jour.

22. write_bracket_concept(C), write_concept(C) : ces prédicats affichent un concept et gère le parenthésage lorsqu'il s'agit des concepts complexes.

    Par exemple, si en paramètre le prédicat reçoit *and(some(aEcrit, livre), or(personne, sculpture)*, le prédicat *write_concept* produira l'affichage formaté suivant : *(∃ aEcrit . livre) ⊓ (personne ⊔ sculpture)*.

23. write_concept_assert((I, C)), write_concept_assert_list(L) : ces prédicats affichent respectivement une assertion de concepts de la ABox et une liste d'assertions de concepts.

    Par exemple, si en paramètre le prédicat reçoit *(michelAnge, and(some(aEcrit, livre), or(personne, sculpture))*, le prédicat *write_concept_assert* produira l'affichage formaté suivant : *michelAnge : (∃ aEcrit . livre) ⊓ (personne ⊔ sculpture)*.

Le prédicat *write_concept_assert_list* effectue le même affichage pour une liste. Il est par la suite utilisé pour l'affichage de tous les formats de listes d'assertions de concepts de la ABox.

24. write_rule_assert((I1, I2, R)), write_rule_assert_list(L) : ces prédicats affichent respectivement une assertion de rôles de la ABox et une liste d'assertions de rôles.

    Par exemple, si en paramètre le prédicat reçoit *(michelAnge, david, aCree)*, le prédicat *write_rule_assert* produira l'affichage formaté suivant : *< michelAnge, david > : aCree.*

    Le prédicat *write_concept_assert_list* effectue le même affichage pour une liste.

25. **affiche_evolution_Abox(Lie1, Lpt1, Li1, Lu1, Ls1, Abr1, Lie2, Lpt2, Li2, Lu2, Ls2, Abr2) :** ce prédicat affiche l'évolution d'un état de la Abox étendue vers un état suivant, les 6 premiers paramètres décrivant l'état de départ, les 6 suivants l'état d'arrivée. Les listes **Lie*, Lpt*, Li*, Lu*** et **Ls*** respectent le format expliqué ci-haut, et les listes **Abr*** représentent les assertions des rôles de la ABox.

    Les différents affichages des listes utilisent par conséquent les prédicats *write_concept_assert_list* et *write_rule_assert_list* respectivement pour les listes d'assertions de concepts et les listes d'assertions de rôles.

26. test_clash(Ls) : ce prédicat vérifie s'il y a un clash (c'est-à-dire une assertion et son contraire) dans la ABox. On se limite ici à ne regarder que les assertions de concepts atomiques, la liste **Ls** donc, partant du principe que l'application des règles de résolution (par la méthode des tableaux) produit au final que des assertions de concepts atomiques.

Nous nous sommes fortement inspirés de l'arbre de résolution donné dans l'énoncé pour écrire les prédicats qui suivent. Les prédicats de résolution et d'application des règles dépendent fortement les uns des autres.

A chaque changement dans la ABox, chaque prédicat fait appel au prédicat *evolue* pour modifier en conséquence les listes de la ABox et au prédicat *affichage_evolution_Abox* afin d'afficher les différentes modifications et donc la trace de résolution.

27. **resolution(Lie, Lpt, Li, Lu, Ls, Abr)** : ce prédicat lance la résolution d'un noeud développé par l'arbre de résolution. **Il commence par tester s'il y'a clash** en faisant un appel au prédicat *test_clash*. S'il n'y a pas de clash, il fait appel au prédicat *complete_some*.

28. **complete_some(Lie, Lpt, Li, Lu, Ls, Abr)** : ce prédicat tente d'appliquer la règle *il existe*.

Si la liste **Lie** est vide, la règle ne peut être appliquée ; le prédicat délègue donc la résolution au prédicat *transformation_and*.

Si la liste n'est pas vide, on applique la règle et on fait à nouveau appel au prédicat *resolution* avec la ABox modifiée.

29. **transformation_and(Lie, Lpt, Li, Lu, Ls, Abr)** : ce prédicat tente d'appliquer la règle *et*.

    Si la liste **Li** est vide, la règle ne peut être appliquée ; le prédicat délègue donc la résolution au prédicat *deduction_all*.

    Si la liste n'est pas vide, on applique la règle et on fait à nouveau appel au prédicat *resolution* avec la ABox modifiée.

30. **deduction_all(Lie, Lpt, Li, Lu, Ls, Abr)** : ce prédicat tente d'appliquer la règle *pour tout*.

    Si la liste **Lpt** est vide, la règle ne peut être appliquée ; le prédicat délègue donc la résolution au prédicat *transformation_or*.

    Sinon si la liste n'est pas vide, c'est-à-dire qu'il y a une tête de liste de la forme : *(A,all(R,C).*

    On vérifie si dans la liste des assertions des rôles **Abr** il existe une assertion de la forme *(A, B, R)*, condition sine qua none à l'application de la règle *pour tout*. Tant qu'il existe une telle assertion, on la supprime de la ABox et on applique la règle. On rappelle le prédicat *deduction_all* avec la nouvelle liste **Abr** et la même liste **Lpt** pour appliquer la règle de l'assertion *(A,all(R,C)* une fois pour toute.

    S'il n'existe pas ou plus d'assertion de rôle de la forme *(A, B, R)* dans **Abr,** la règle ne peut plus être appliquée, on fait donc à nouveau appel au prédicat *resolution* avec la ABox modifiée.

31. **transformation_or(Lie, Lpt, Li, Lu, Ls, Abr)** : ce prédicat tente d'appliquer la règle *ou*.

    Si la liste **Lu** est vide, la règle ne peut être appliquée ; on indique que plus aucune règle ne peut être appliquée.

    Si la liste n'est pas vide, on applique la règle. Cette règle produit deux noeuds de résolutions. Au sein du prédicat, on construit les listes respectives des deux nouvelles ABox et on fait appel au prédicat *resolution* pour les deux noeuds.

32. **troisieme_etape(Abe, Abr)** : ce prédicat tri la liste des assertions de concepts étendues **Abe** en faisait appel au prédicat *tri_Abox* et lance la résolution en faisant appel au prédicat *resolution.*

33. **programme** : ce prédicat lance le programme principal, effectue les traitements adéquats sur la TBox et la ABox, invite ensuite l'utilisateur à rentrer la proposition à démontrer et lance la démonstration.