Master d'Ingéniérie Informatique de Paris 7
M1: Prolog et programmation par contraintes

Partiel du 19 novembre 2004 - Durée: 2 heures

Documents autorisés; le barème est donné à titre indicatif.

Exercice 1 (**2 points**)

Quels sont les résultats pour les buts suivants?

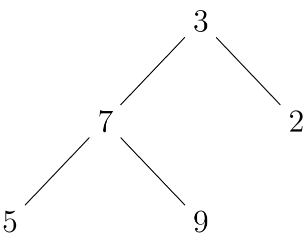
1. `f(g(X,a),Y)=f(Y,g(b,Z)).`
2. `X==3.`
3. `not(X==3).`
4. `X is 3.`
5. `not(X is 3).`
6. `[X,a|R]=[3,Y,1].`
7. `!,false.`
8. `!;false.`
9. `assert(p(a)),assert(p(b)),p(X).`
10. `assert(p(a)),assert(p(b)),setof(X,p(X),R).`

Exercice 2 (**3 points**)

Pour chaque prédicat de la liste suivante:
- décrire la relation qu'il définit (c.à.d. décrire "ce que le prédicat calcule").
-Indiquer les éventuels paramètres d'entrée.

1. ```
   a(0,0):-!.
   a(X,Y):- V is X-1, a(V,Z), Y is Z + X.
   ```
2. ```
   b(X,0,1):-!.
   b(X,Y,R):- Z is Y-1, b(X,Z,T), R is T*X.
   ```
3. ```
   c(0,L,L).
   c(X,[Z|L],T):- Y is X-1, c(Y,L,T).
   ```
4. ```
   d([],[]).
   d([X|L],R):- d(L,T), R=[X|[X|T]].
   ```
5. ```
   e(X,Y,[X,Y|Z]).
   e(X,Y,[Z|T]):-e(X,Y,T).
   ```
   ```

Exercice 3 (**5 points**) On considère la représentation en Prolog des arbres binaires étiquetés sur les entiers exemplifiée par:



est représenté par `arbre(3,arbre(7,5,9),2)`.

1. Ecrire un prédicat `est_arbre/1` tel que le but `est_arbre(terme)` réussit si et seulement si `terme` est la représentation d'un arbre binaires étiqueté sur les entiers.

2. Ecrire un prédicat `somme(Arbre,Somme)`, pour calculer la somme des étiquettes d'un arbre.

   Un *parcours* est une liste de `gauche` et `droite`. Un parcours est *valide* pour un arbre `t` si, en suivant le parcours depuis la racine de `t`, en allant vers le fils gauche ou droit comme indiqué par le parcours, on aboutit à une feuille (exemple: les parcours valides pour l'arbre ci-dessus sont `[gauche,gauche]`, `[gauche,droite]` et `[droite]`).

3. Ecrire un predicat `valide(Parcours,Arbre)` tel que le but `valide(p,a)` réussit si et seulement si `p` est valide dans `a`.

4. Ecrire un predicat `prolonge(Parcours,Arbre,Prolongement)` pour vérifier si un parcours donné peut être étendu en parcours valide. Utilisation typique:

```
  ?- prolonge([gauche],arbre(3,arbre(7,5,9),2),L).
L = [gauche] ? ;
L = [droite] ? ;
no
```

Exercice 4 (**5 points**)

On considère les faits:

```
q(2). q(4).
r(3). r(9).
s(1). s(2).
```

et la règle:

```
p(X):-q(Y),r(Z),s(T), X is Y*Z+T.
```

Nous allons étudier des variantes de cette règle, dans lesquelles des coupures seront intercalées dans le corps de la règle.

Un exemple de variante "à une coupure":

```
p(X):-q(Y),r(Z),!,s(T), X is Y*Z+T.
```

Un exemple de variante "à deux coupures":

```
p(X):-q(Y),!,r(Z),s(T), X is Y*Z+T,!.
```

1. Il existe cinq variantes “à une coupure” de cette règle. Pour chacune d’entre elles, donner les résultats du but `p(X)` pour le programme composé par l’ensemble de faits ci-dessus et la variante en question.

2. Pour un de ces cinq programmes, au choix, dessiner l’arbre de dérivation de `p(X)`.

   On dit qu’une variante `R` *est simulée par* une variante `R’` si les résultats du but `P(X)` pour `R` et `R’` sont les mêmes.

3. Montrer (à l’aide d’un argument général, ne pas énumérer tous les cas!) que toute variante à deux coupures est simulée par une variante à une coupure.

Exercice 5 (**2.5 points**)
On considère les faits:

```
q(a). q(b). q(c).
r(b). r(c). r(d).
s(c). s(d). s(e).
```

et les règles:

a) `p(X):-q(X),r(X),s(X).`

b) `p(X):-q(X),r(X),not(s(X)).`

c) `p(X):-q(X),not(r(X)),s(X).`

d) `p(X):-not(q(X)),r(X),s(X).`

e) `p(X):-q(X),not(r(X)),not(s(X)).`

Donner les résultats du but `p(X)` pour chacune de ces règles (prise séparemment).

Exercice 6 (**2.5 points**)
On considère les programmes:

P1:	**P2:**
`p(a).`	`p(X):-p(a).`
`p(X):-p(a).`	`p(a).`

1. Quels sont respectivement l’univers de Herbrand U_1 et le plus petit modèle de Herbrand H_1 de **P1**? Et ceux de **P2**? Soit C_1 l’ensemble des conséquences logiques de **P1** vérifiées dans H_1.

2. Soit B_1 (respectivement B_2) l’ensemble de buts clos qui reussissent pour **P1** (respectivement **P2**). Exhiber un élément de B_1 qui n’est pas dans C_1 et un éléments de C_1 qui n’est pas dans B_2.

Exercice 7 (**facultatif**) On considère le programme suivant, qui vérifie si un mot, sous la forme d’une liste de lettres, appartient au langage $\{a^n b^n \mid n \geq 1\}$:

```
p(X):-q(q0,X,[]).
q(q1,[],[]).
q(q0,[a|X],S):-q(q0,X,[a|S]).
q(q0,[b|X],[a|S]):-q(q1,X,S).
q(q1,[b|X],[a|S]):-q(q1,X,S).
```

1. tracer la rêquete:

   `p([a,a,b,b]).`

2. Montrer que tout automate à pile (déterministe ou non) peut être simulé par un programme prolog (géneraliser la méthode utilisée dans le programme ci-dessus).