

Ce sujet comporte 3 pages.

Tous les documents sont autorisés.

Le barème est indicatif.

Toutes les réponses doivent être justifiées (de façon concise, mais précise).

Partie I - Arbres de recherche (6 pts)

Soit Prog le programme PROLOG ci-dessous :

$p(1, Y) :- r(Y) .$	$q(X) :- c(Z, Z), !, d(X, Z) .$
$p(X, Y) :- q(X), Y \text{ is } X+3 .$	$q(5) .$
$p(3, 2) :- a(T), b(T) .$	
$c([], [5,3]) .$	$d(E, [E]) .$
$c([1], [X]) .$	$d(12, [X L]) .$
$c([4,2], [4,2]) .$	
$r(91) .$	$a(72) .$
$r(62) .$	$b(4) .$
	$b(23) .$

Question 1 (4 pts)

Construire l'arbre de recherche du but $p(X, Y)$.

Quels sont les témoins de la preuve de $\text{Prog} \vdash \exists X \exists Y . p(X, Y)$?

Soit toto le prédicat suivant :

$\text{toto}([T|Q], [Q|T]) .$

Question 2 (2 pts)

- Que fait le prédicat toto ? Faites part de vos réflexions sur ce prédicat.

- Que produit $\text{toto}([1,2,3,4], R)$? Indiquer le détail des unifications effectuées.

Partie II - Logique (4 pts)

Question 3 (3 pts)

Construire une preuve en calcul des séquents pour chacun des énoncés suivants :

$\neg(A \wedge B) \Rightarrow \neg A \vee \neg B$ et $A \Rightarrow (B \Rightarrow A)$

Question 4 (1 pt)

Montrer que l'énoncé suivant n'a pas de preuve en calcul des séquents.

$(A \vee B) \Rightarrow A$

Partie III - Programmation logique (10 pts)

Voici la spécification partielle d'un générateur de code sous forme d'une grammaire à attributs (extraite du cours de compilation).

Spécification :

```
B → E1 rop E2
    B.place := nouvelle_variable()
    B.code :=      E1.code @@ E2.code
                @@ {rop.vallex, B.place, E1.place, E2.place}
| not B1
    B.place := nouvelle_variable()
    B.code := B1.code @@ {not, B.place, B1.place, _}
| B1 bop B2
    B.place := nouvelle_variable()
    B.code :=      B1.code @@ B2.code
                @@ {bop.vallex, B.place, B1.place, B2.place}
| id
    B.place := id.place
    B.code := code_vide
| bool
    B.place := nouvelle_variable()
    B.code := {const si bool.vallex == vrai alors 1 sinon 0 fsi, B.place, _, _}
```

Rappels :

- Les instructions sont représentées par un quadruplet {codop, destination, source1, source2}.
- Le signe @@ représente la concaténation de séquences d'instructions.
- Les non-terminaux B et E engendrent respectivement les expressions booléennes et les expressions arithmétiques. On distingue leurs occurrences par des indices, mais B, B₁ et B₂ sont bien le même non-terminal.
- Les non-terminaux rop, bop, id et bool engendrent respectivement les opérateurs relationnels, les opérateurs binaires, les identificateurs de variables booléennes, et les constantes booléennes.
- Les attributs code, place, vallex permettent la production du code : code contient le code produit, place l'identificateur de la variable qui contient le résultat de l'évaluation de ce code, et vallex la chaîne de caractères reconnue par un non-terminal.
- not est un terminal.
- La fonction nouvelle_variable() produit un nouvel identificateur de variable à chaque appel.

Exemple : le code produit par l'analyse de l'expression $a > 3$ ou $b == a+5$ pourrait être le suivant :

```
{>, v1, a, 3}
{+, v2, a, 5}
{==, v3, b, v2}
{ou, v4, v1, v3}
```

qui laisse le résultat du calcul dans la place v4.

On se propose de programmer en PROLOG ce générateur de code. On ne s'intéresse qu'à la programmation du non-terminal B, car on suppose que les autres non-terminaux sont déjà programmés.

La fonction nouvelle_variable() sera codée en PROLOG par le prédicat nouvelle_variable/3 suivant :

```
% nouvelle_variable(+CompteurIn, -Id, -CompteurOut) : Id est l'identificateur de la variable de numéro CompteurIn ; le prochain numéro sera CompteurOut.  
nouvelle_variable(CompteurIn, var(CompteurIn), CompteurOut) :-  
    CompteurOut is CompteurIn+1.
```

Question 5 (1 pt)

Imaginer un codage en PROLOG de la représentation des instructions.

Question 6 (2 pts)

On veut programmer ce générateur de code en DCG. La grammaire DCG analysera une liste d'unités lexicales représentant une expression booléenne à analyser selon la spécification proposée, et elle produira, par calcul d'attributs, le code associé.

Donner le principe de codage de la grammaire DCG : choix des structures de données, arguments d'entrée et de sortie des non-terminaux DCG permettant la circulation des attributs.

Question 7 (5 pts)

Donner le codage en DCG de la spécification.

Question 8 (2 pts)

Identifier des difficultés (inefficacité, ...) que pourrait connaître votre grammaire DCG à l'exécution, et donner des pistes pour les résoudre.