Département Informatique

Rajouter sur les pages à remplir un cadre pour les différents aspects de la notation + une feuille explicative du barème correspondant (pas forcément à distribuer aux étudiants d'aillers), cf corrigé!!!

Examen de Compilation - 4ème Année

Lundi 30 janvier 2006 - 13h30-16h30

Durée de l'épreuve : 3h.
Responsable : M. Ducassé (82 52)
Aucun document autorisé sauf une page A4 manuscrite (recto-verso).

L'énoncé comporte 20 pages, dont les pages 13 à 20 sont **à détacher, remplir et joindre à la copie.** Des définitions, possiblement utiles à cet examen, se trouvent en annexe, pages 10 à 12. Le barème et les temps sont donnés à titre indicatif.

1 Questions de cours (*1/4 heure, 2 points*)

Question 1.1 *Quelles sont les étapes de compilation indépendantes de la machine cible ? Pour chaque étape, donner son nom, ce qu'elle prend en entrée et ce qu'elle rend en sortie. Il n'est pas demandé de définition, ni d'explication.*

Question 1.2 *Pour chacune des paires d'expressions Micro ML de type suivantes, dire si les deux expressions s'unifient. Si oui donner la substitution résultat, si non dire à quel moment l'algorithme d'unification va générer un échec.*

1. *fonc(T1, tuple(bool, liste(T1))) et fonc(ent, tuple(T2, T2))*
2. *fonc(T1, tuple(bool, liste(T1))) et fonc(ent, tuple(T2, T3))*

Problème

Le problème est divisé en deux parties : la section 2 traite de questions d'analyse lexicale et d'analyse syntaxique, la section 3 traite de questions d'analyse sémantique. Ces parties sont indépendantes. Notez que la partie sur l'analyse syntaxique est beaucoup plus longue que celle sur l'analyse

sémantique. Il est recommandé de **lire toutes les questions** *du problème avant de commencer à y répondre. En particulier, les figures donnent beaucoup d'informations. Il est conseillé de ne comprendre de ces figures que ce qui est* **nécessaire pour répondre aux questions.** *Les informations dans les notes de bas de page (*footnotes*) ne sont pas nécessaires à cet examen. Elles sont essentiellement destinées à ceux qui voudraient réfléchir au problème a posteriori.*

Le problème aborde l'analyse de programmes Prolog. L'énoncé de cet examen utilise une version simplifiée de la syntaxe d'ECLiPSe Prolog[1]. La figure 5 page 6 donne un exemple de programme spécifié en Prolog suivant cette syntaxe simplifiée. *Ne pas perdre de temps à comprendre ce programme Prolog. C'est sans importance pour la suite.*

2 Analyses lexicale et syntaxique (*1 heure 3/4, 11,5 points*)

Le but de cette première partie est de comprendre quelques spécificités de la syntaxe d'ECLiPSe Prolog en s'appuyant sur les connaissances acquises en cours.

Question 2.1 Le manuel ECLiPSe définit des classes de caractères (cf figure 1) et des groupes de caractères (cf figure 2). *Quelle est la propriété vue en cours qui ne serait pas vérifiée si les groupes de caractères étaient traités comme des classes de caractères ? Justifier brièvement.*

Question 2.2 *Le traitement des groupes de caractères n'a pas été vu en cours. Dire en une phrase comment vous pensez qu'il faut les traiter.*

Question 2.3 La figure 3 spécifie une partie des unités lexicales et lexèmes valides. *Quel est le formalisme utilisé pour cette spécification ? Cela vous surprend-il ? Justifier cette dernière réponse.*

Question 2.4 `EOCL` *signifie "End Of CLause". Paraphraser sa définition.*

[1]ECLiPSe User Manual, Release 5.8, Joachim Schimpf et al., http ://eclipse.crosscoreop.com/eclipse/

upper_case	UC	all upper case letters
underline	UL	_
lower_case	LC	all lower case letters
digit	N	digits
blank_space	BS	space, tab and nonprintable ASCII characters
end_of_line	NL	line feed
atom_quote	AQ	’
string_quote	SQ	"
solo	SL	! ; ([{)] } , |
line_comment	CM	%
escape	ES	\
first_comment	CM1	/
second_comment	CM2	*
symbol	SY	# + - . : < = > ? @ ^ ‘ ~ $ &

FIG. 1 – Classes de caractères d’ECLiPSe Prolog

alphanumerical	ALP	UC UL LC N
any character	ANY	
non escape	NES	ANY except ES
sign	SGN	+ -

FIG. 2 – Groupes de caractères d’ECLiPSe Prolog

```
EOCL = . (BS | NL | <end of file>)  | <end of file>

ATOM = (LC ALP*)
        | (SY | CM1 | CM2 | ES)+
        | (AQ (NES | ES ANY+)* AQ)
        | SL
        | []
        | {}
        | |

INT  = [SGN] N+

STRING = SQ (NES | ES ANY+ | SQ BS* SQ)* SQ

VAR = (UC | UL) ALP*
```

FIG. 3 – Spécification partielle des unités lexicales et lexèmes d'ECLiPSe Prolog

Question 2.5 *En utilisant les figures 1 et 3 justifier pourquoi cette définition n'est pas aussi simple que ce à quoi on aurait peut-être pu s'attendre. En particulier, dire quelles seraient les conséquences sur la partie analyse du compilateur si la restriction précédente n'existait pas.*

Une syntaxe simplifiée d'ECLiPSe Prolog est spécifiée par la grammaire non-contextuelle G_1 définie à la figure 4. La figure 5 donne un exemple de programme spécifié en Prolog suivant cette syntaxe simplifiée. *On répète de ne pas perdre de temps à comprendre le programme Prolog. C'est sans importance pour la suite.*

Question 2.6 *Donner les ensembles V_N et V_T de la grammaire G_1.*

On se propose d'essayer de faire une analyse SLR de la grammaire G_1.

Attention, les questions 2.7 à 2.10 doivent impérativement être rédigées directement sur les pages 13 et 15 de cet énoncé (que vous rendrez avec votre copie). Les questions 2.11 à 2.15 sont à rédiger sur la copie.

1 $prog \longrightarrow clS$
2 $clS \longrightarrow clause\ EOCL$
3 $clS \longrightarrow clause\ EOCL\ clS$
4 $clause \longrightarrow head$
5 $clause \longrightarrow head\ :\text{-}\ goals$
6 $clause \longrightarrow :\text{-}\ goals$
7 $head \longrightarrow term\_g$
8 $goals \longrightarrow term\_g$
9 $goals \longrightarrow goals\ ,\ goals$
10 $term\_g \longrightarrow ATOM$
11 $term\_g \longrightarrow ATOM\ (\ termlist\ )$
/* Note : no space before (*/
12 $termlist \longrightarrow term$
13 $termlist \longrightarrow term\ ,\ termlist$
14 $list \longrightarrow [\ listexpr\ ]$
15 $list \longrightarrow .(term,\ term)$
16 $listexpr \longrightarrow term$
17 $listexpr \longrightarrow term \mid term$
18 $listexpr \longrightarrow term\ ,\ listexpr$
19 $term \longrightarrow term\_g$
20 $term \longrightarrow VAR$
21 $term \longrightarrow list$
22 $term \longrightarrow STRING$
23 $term \longrightarrow INT$

FIG. 4 – G_1 : Grammaire simplifiée d'ECLiPSe Prolog

```
explore(Goal) :-
        db(fact, Goal).

explore(Goal) :-
        db(Rule, then(if(Cond), Goal)),
        explore(Cond).

explore(and(Goal1, Goal2)) :-
        explore(Goal1),
        explore(Goal2).

db(fact, eats(sheba, meat)).

%c_rule concludes about carnivores
db(c_rule, then(if(and(isa(A, mammal), eats(A, meat))),
                isa(A, carnivore))).
```

FIG. 5 – Un extrait de système expert qui classifie les animaux, programmé en Prolog

Question 2.7 *Calculer les ensembles d'items SLR demandés page 13 pour la grammaire G_1.*

Question 2.8 *Donner les ensembles "suivant" pour les non-terminaux de la grammaire, comme spécifié page 13. On ne demande pas de faire un calcul formel.*

Question 2.9 *Remplir les parties des tables d'analyse SLR, comme demandé page 15, avec ce qui peut être construit à l'aide des ensembles précédemment calculés. On indiquera toutes les possibilités données par le calcul d'items dans les cases correspondantes.*

Question 2.10 *Essayer de résoudre les conflits détectés. Il est interdit de changer la grammaire. On peut toutefois s'appuyer sur des propriétés bien connues, ou sur des propriétés sans conséquence pour le programmeur. Dans ce cas, il faudra bien expliciter ces propriétés. Pour chacun des conflits, bien dire si on peut ou non le résoudre et si oui comment. Justifiez vos réponses.*

Question 2.11 *Dans le cours il a été dit qu'en analyse LR, il ne pouvait jamais y avoir de conflit décaler/décaler. Illustrer ce propos en utilisant l'ensemble d'items I_1 précédemment calculé.*

Question 2.12 *Pensez-vous que la construction des tables permette d'expliquer la note attachée à la règle 11 ? Justifier brièvement.*

11 term_g ⟶ ATOM (termlist)
/ Note : no space before (*/*

En Prolog, les programmeurs peuvent déclarer comme opérateur n'importe quel atome. Ces opérateurs peuvent être préfixes, infixes ou postfixes, une priorité doit aussi leur être attachée. Les opérateurs définis par les programmeurs ont de gros avantages pour les utilisateurs. Par exemple, dans le programme de la figure 5, en définissant `if` comme un opérateur préfixe, `then`, `and`, `eats`, et `isa` comme des opérateurs infixes, la règle suivante

```
db(c_rule, then(if(and(isa(A, mammal), eats(A, meat))),
               isa(A, carnivore))).
```

peut devenir ce qui suit, qui est beaucoup plus lisible[2].

[2]Le parenthésage dépend des priorités relatives des opérateurs, non spécifiées ici.

```
db(c_rule, if   (A isa mammal) and  (A eats meat)
           then A isa carnivore).
```

Question 2.13 *Peut-on maintenant expliquer la note attachée à la règle 11 ? Justifier.*

Question 2.14 *Si on autorise les opérateurs définis par les programmeurs, que peut-on dire du vocabulaire terminal de l'analyse syntaxique ? Qu'en déduit-on pour la construction des tables LR ?*

Question 2.15 *Sur un "blog", trouvé par hasard, on peut lire en substance :* "J'ai essayé pendant plusieurs semaines de programmer un analyseur syntaxique de Prolog avec des outils courants LR et j'ai abandonné à cause des opérateurs". *Quel commentaire cela vous inspire-t-il ?*

3 Analyse sémantique (*1 heure, 6,5 points*)

Dans la phase d'analyse sémantique, on souhaite générer pour tout le programme la liste des prédicats définis avec leur "arité". L'arité d'un prédicat est le nombre de ses arguments. Une prédicat sans argument est d'arité 0. Par exemple, pour le programme de la figure 5 la liste retournée sera `[explore/1, db/2]`.

Question 3.1 *Donner une partie de l'arbre syntaxique concret de l'analyse selon G_1 du programme de la figure 5 page 6. Cette partie doit être suffisante pour tester les calculs d'attributs demandés dans la suite.*

On n'hésitera pas à utiliser un intercalaire seulement à cette fin. Il est conseillé de construire cet arbre au fur et à mesure des besoins. Pour les parties non détaillées utiliser "...". On pourra annoter cet arbre si nécessaire lors de la résolution des questions suivantes, pourvu qu'il reste lisible.

Les questions 3.2 et 3.4 doivent être rédigées directement sur les pages 16 à 20.

Pour la construction de grammaires à attributs, on introduira les attributs explicitement, en indiquant leur objet, leur type et s'ils sont synthétisés ou hérités. On définira avec soin les structures de données utilisées. Les fonctions sémantiques devront être définies avec précision. Il n'est **pas** demandé de code Yacc, il s'agit de donner une spécification comme fait en cours et TD. **Les variables globales sont interdites.**

Question 3.2 Dans un premier temps on se propose de collecter tous les prédicats apparaissant dans les têtes de clauses sans se soucier des doublons. Par exemple, pour le programme de la figure 5 la liste retournée sera

`[explore/1, explore/1, explore/1, db/2, db/2]`.

Construire une grammaire attribuée GA_1, attachée à la grammaire non contextuelle G_1, qui permet de collecter cette liste.

Question 3.3 *Si on souhaitait collecter dans une autre liste les prédicats utilisés*[3]*, quelles règles devraient être modifiées ? Donner une esquisse des modifications. Il n'est pas demandé de spécifier formellement ce nouveau calcul.*

[3]par exemple pour détecter ceux qui ne sont pas définis

En ECLiPSe Prolog les clauses qui définissent un même prédicat doivent être contiguës afin de permettre la compilation incrémentale[4].

On se propose de tirer partie de cette particularité afin d'éviter d'ajouter des doublons dans la liste des prédicats. Il s'agit de n'insérer dans la liste que les nouveaux prédicats.

Question 3.4 *Sur les mêmes pages que pour GA_1 mais avec une* **couleur différente**, *construire une grammaire attribuée GA_2 qui étend et modifie la grammaire GA_1 afin de ne pas insérer de doublons.*

Rayer délicatement avec la deuxième couleur les règles sémantiques de GA_1 qui doivent être modifiées pour GA_2. Les règles rayées doivent, bien entendu, rester lisibles.

Question 3.5 *Comparer ce dernier calcul à une solution qui consisterait à enlever a posteriori les doublons de la liste résultat de GA_1.*

Fin des questions

4 Annexes

4.1 Construction des tables SLR

4.1.1 L'opération Fermeture

Si I est un ensemble d'items pour une grammaire $G = (V_T, V_N, S, P)$, $Fermeture(I)$ est l'ensemble d'items construit à partir de I par les deux règles :

1. Placer chaque item de I dans $Fermeture(I)$.

2. Si $A \longrightarrow \alpha \bullet B\beta \in Fermeture(I)$,
   si $B \longrightarrow \gamma \in P$
   alors
   ajouter $B \longrightarrow \bullet\, \gamma$ à $Fermeture(I)$.

[4]Les prédicats sont compilés indépendemment les uns des autres, dès que toutes les clauses définissant un même prédicat ont été collectées.

4.1.2 L'opération Transition

Si I est un ensemble d'items et si X est un symbole de la grammaire ($X \in (V_T \cup V_N)$)
alors $Transition(I, X) = Fermeture(J)$
où $J = \{A \longrightarrow \alpha X \bullet \beta \quad tq \quad A \longrightarrow \alpha \bullet X\beta \in I\}$

4.1.3 Construction de la collection canonique d'ensembles d'items LR(0)

C est un ensemble d'ensembles d'items
$C = \{Fermeture(\{S' \longrightarrow \bullet S\})\}$
répéter
pour chaque ensemble d'items I de C et pour chaque symbole de la grammaire $X \in (V_T \cup V_N)$
tels que $Transition(I, X) \neq \emptyset$
et non encore dans C

ajouter $Transition(I, X)$ **à** C

jusqu'à ce qu'aucun nouvel ensemble d'items ne puisse plus être ajouté à C

4.1.4 Construction des tables d'analyse SLR

1. Construire $C = \{I_0, ..., I_n\}$ la collection canonique des ensembles d'items LR(0) pour G'. Les états i correspondent aux I_i.
   $I_0 = Fermeture(\{S' \longrightarrow \bullet S\})$.

2. Table "Action". Pour tous les états i :
   - Pour $[A \longrightarrow \alpha \bullet a\beta] \in I_i$ tq $Transition(I_i, a) = I_j$ et $a \in V_T$
     alors Action[i, a] = "décaler j".

   - Pour $[A \longrightarrow \alpha \bullet] \in I_i$ tq $A \neq S'$ et pour tous les $a \in Suivant(A)$
     alors
     Action[i, a] = "réduire par $A \longrightarrow \alpha$"

- Si $[S' \longrightarrow S \bullet\ ] \in I_i$
  alors Action[i, \$] = "accepter".

- Toutes les autres entrées sont positionnées à "erreur".

- **Si les règles précédentes engendrent des conflits, la grammaire n'est pas SLR**. L'algorithme échoue et ne produit pas d'analyseur.

3. Table "Successeur" :
   Pour $A \in V_N$ tq $Transition(I_i, A) = I_j$
   alors successeur(i, A) = j.

Nom : .....

Construction de tables SLR *(Il y a 4 questions dans cette partie)*

Question 2.7 Calculer les ensembles d'items canoniques SLR suivants pour G_1

Fermeture($\{\text{prog} \longrightarrow \bullet \text{ clS}\}$) $= I_0$

$Transition(I_0, clause) = I_1$

$Transition(I_0, head) = I_2$

$Transition(I_0, term\_g) = I_3$

$Transition(I_0, ATOM) = I_4$

$Transition(I_2, \text{“ :-”}) = I_5$

$Transition(I_5, goals) = I_6$

$Transition(I_6, \text{“,”}) = I_7$

$Transition(I_7, goals) = I_8$

$Transition(I_8, \text{“,”}) =$

Question 2.8 Donner les ensembles "suivant" pour les non-terminaux *clause*, *head*, *term_g* et *goals* de G_1.

Nom : .....

Question 2.9 Tables SLR partielles.
En utilisant les résultats précédents, remplir ce que vous pouvez des 9 lignes spécifiées des tables SLR. **Ne faire figurer que les terminaux et non-terminaux intervenant dans ces lignes.** Ne pas calculer d'ensembles d'items supplémentaires. Indiquer toutes les possibilités données par le calcul d'items dans les cases correspondantes.

0														
1														
2														
3														
4														
5														
6														
7														
8														

Question 2.10 Pour chaque conflit détecté, dire si on peut ou pas le régler. Justifiez les réponses. On pourra, le cas échéant, utiliser le dos de cette page.

Nom : .....

Questions 3.2 et3.4. Construction des grammaires attribuées GA_1 et GA_2. Utilisez une **couleur différente** pour chaque grammaire à attributs. Les règles 19 à 23 ne sont pas nécessaires ici, elles sont donc omises.

Description des attributs

Fonctions sémantiques

Nom : .....

1 prog $\longrightarrow$ clS

2 clS $\longrightarrow$ clause EOCL

3 clS $\longrightarrow$ clause EOCL clS

4 clause $\longrightarrow$ head

5 clause $\longrightarrow$ head :- goals

6 clause $\longrightarrow$:- goals

7 head $\longrightarrow$ term_g

8 goals $\longrightarrow$ term_g

9 goals $\longrightarrow$ goals , goals

Nom : .....

10 term_g $\longrightarrow$ ATOM

11 term_g $\longrightarrow$ ATOM (termlist)

12 termlist $\longrightarrow$ term

13 termlist $\longrightarrow$ term , termlist

14 list $\longrightarrow$ [listexpr]

15 list $\longrightarrow$.(term, term)

16 listexpr $\longrightarrow$ term

17 listexpr $\longrightarrow$ term | term

18 listexpr $\longrightarrow$ term , listexpr