sciences-centreimpression

De: no-reply@www.usherbrooke.ca
Envoyé: 30 juillet 2015 14:22
À: sciences-centreimpression
Objet: COMMANDE EXAMENS IFT313 5 août
Pièces jointes: IFT313_Final_E2015.pdf

TYPE-EXAMEN	FINAL
SIGLE-COURS	IFT313
TITRE-COURS	Introduction aux langages formels
PROFESSEUR	Richard St-Denis
DATE-HEURE	Mercredi 5 août à 9 h
AUTORISE-PAR	Richard St-Denis et André Mayers
NOMBRE-PAGES	12
NOMBRE-COPIE-PROF	25
IMPRESSION-QUESTIONNAIRE	Recto-verso plié broché en cahier
NOMBRE-FEUILLES-BLANCHES	
NOMBRE-PAPIER-GRAPHIQUE	
NOMBRE-CAHIERS	
CONSENTEMENT-AGES	1
REMARQUES	Impression en cahier de cet examen et Impression couleur s.v.p. (demandes du Pr St-Denis). Richard.St-Denis@usherbrooke.ca
E-MAIL	
FIRST-NAME	
LAST-NAME	
NICK-NAME	
SPAMSHIELD	true

UNIVERSITÉ DE SHERBROOKE

DÉPARTEMENT D'INFORMATIQUE

IFT 313

Introduction aux langages formels

EXAMEN FINAL[1]

Le mercredi 5 août 2015 de 9 h 00 à 12 h 00 Professeur : Richard St-Denis

- Toute documentation est permise.
- **Tout appareil électronique, incluant le téléphone cellulaire et l'ordinateur, est interdit.**
- Ne dégrafez pas ce questionnaire.
- Répondez dans les espaces prévus à cet effet.
- **Personne ne peut quitter la salle d'examen avant 9 h 15.**
- **Personne ne peut quitter son siège entre 11 h 50 et 12 h 00.**
- La correction est, entre autres, basée sur le fait que chacune de vos réponses soit :
  - claire, c'est-à-dire lisible et compréhensible pour le correcteur ;
  - précise, c'est-à-dire exacte ou sans erreur ;
  - complète, c'est-à-dire que toutes les étapes de résolution du problème sont présentes ;
  - concise, c'est-à-dire que la méthode de résolution est la plus courte possible.

Nom : _____ Prénom : _____

Signature : _____ Matricule : _____

Question	Barème
1	/8
2	/8
3	/8
4	/8
5	/8
total :	/40

1. Ce questionnaire comporte douze (12) pages.

Vous pouvez utiliser des abréviations dans vos réponses afin d'éviter d'écrire des noms trop longs, mais à la condition qu'elles soient indicatives. Par exemple, vous pouvez remplacer *instance-variable-identifier* par *IVI* et *expression* par *E*. Évitez les doublons.

1. (8 points) Voici une partie de la définition simplifiée d'un identificateur (*identifier* en anglais) pour le langage de programmation *Ruby* présentée sous la forme de règles de production d'une grammaire hors contexte écrites dans la notation non étendue. Les symboles terminaux sont en caractères gras.

   [1] *identifier* → *class-variable-identifier*

   [2] *identifier* → *instance-variable-identifier*

   [3] *class-variable-identifier* → **@ @ char** *identifier-characters*

   [4] *instance-variable-identifier* → **@ char** *identifier-characters*

   [5] *identifier-characters* → *identifier-characters* **char**

   [6] *identifier-characters* → λ

   a) (2 points) Donnez toutes les raisons qui expliquent pourquoi cette grammaire n'est pas LL(1).

   b) (4 points) Calculez les ensembles d'items LR(0) pour cette grammaire.

c) (2 points) Est-ce que cette grammaire est LR(0) ? Justifiez votre réponse.

2. (8 points) Une grammaire linéaire à gauche est une grammaire hors contexte dont les règles de production sont de la forme suivante :

$$A \to \lambda \quad A \to a \quad A \to Ba.$$

a) (2 points) Est-ce qu'une grammaire linéaire à gauche peut être ambiguë ? Si oui, donnez un exemple d'une telle grammaire. Autrement, expliquez pourquoi ?

b) (6 points) Supposons une grammaire linéaire à gauche non ambiguë. Est-ce qu'une telle grammaire est toujours LR(1) et même LR(0) ?

Pour répondre à cette question prenez, par exemple, la grammaire $G = (\{A, B\}, \{a, b, c\}, P, A)$, où P : $A \to Ba$ $\quad B \to Bb$

$A \to a$ $\quad B \to c$

$A \to \lambda$

ou tout autre grammaire linéaire à gauche et calculez les items LR(1). (4 points)

Puis, à partir des conclusions déduites des ensembles d'items précédents, tentez de formuler des arguments qui permettent de répondre positivement ou négativement aux deux questions suivantes.

Est-ce qu'une grammaire linéaire à gauche non ambiguë est LR(1) ? (1 point)

Est-ce qu'une grammaire linéaire à gauche non ambiguë est LR(0) ? (1 point)

3. (8 points) Répondez aux questions suivantes à partir des deux versions de la grammaire du langage DFL qui se trouvent sur des feuilles séparées.

a) (3 points) À partir de la **version 1** de la grammaire du langage DFL, calculez les *lookaheads* de longueur 1 des règles de production suivantes :

[1] *expression* → *primitive_expression rest* _____

[2] *expression* → *letblock_expression rest* _____

[3] *expression* → *conditional_expression rest* _____

[4] *expression* → *for_expression rest* _____

[5] *expression* → *forall_expression rest* _____

[6] *expression* → *application_expression rest* _____

[7] *rest* → λ _____

[8] *rest* → , *expression rest* _____

Placez vos réponses à la droite des règles de production dans les espaces prévus à cet effet et indiquez le détail des calculs les plus importants dans les lignes suivantes.

b) (1 point) Pourquoi cette partie de la grammaire n'est pas LL(1) ?

c) (3 points) À partir de la **version 2** de la grammaire du langage DFL, calculez les *lookaheads* de longueur 1 des règles de production suivantes :

[1] *expression* → *primitive_expression rest* _____

[2] *expression* → *letblock_expression rest* _____

[3] *expression* → *conditional_expression rest* _____

[4] *expression* → *for_expression rest* _____

[5] *expression* → *forall_expression rest* _____

[6] *rest* → λ _____

[7] *rest* → , *expression* _____

Placez vos réponses à la droite des règles de production dans les espaces prévus à cet effet et indiquez le détail des calculs les plus importants dans les lignes suivantes.

d) (1 point) Est-ce que cette partie de la grammaire est LL(1) ?

4. (8 points) Répondez aux deux question suivantes.

   a) (4 points) À partir de la **version 2** de la grammaire DFL et de la méthode de la descente récursive prédictive pour une grammaire LL(1), écrivez la procédure *ReconnaîtreExpression* à partir de la règle de production suivante (écrite dans la notation de l'outil *ell*). Il est strictement interdit d'optimiser le code.

```
expression : ( primitive_expression | letblock_expression | conditional_expression |
               for_expression | forall_expression ) [ ',' expression ] .
```

b) (4 points) À partir de la **version 2** de la grammaire DFL, donnez une partie du contenu de la table d'un analyseur syntaxique descendant à l'aide d'un automate à pile pour une grammaire LL(1) après avoir traduit les deux règles suivantes (écrites dans la notation de l'outil *ell*) dans la notation non étendue. En plus des variables `letblock_expression` et `for_expression`, vous devez considérer toutes les nouvelles variables qui apparaissent dans les règles de production dans la notation non étendue.

```
letblock_expression : 'let' inputlist ':=' restricted_expression
                      ( ';' inputlist ':=' restricted_expression )* 'in' expression 'end' .
for_expression : 'for' inputlist ':=' expression 'do' 'if' primitive_expression 'then'
                 ( iter_expression 'else' expression | expression 'else' iter_expression ) 'end' .
```

Règles de production dans la notation non étendue :

Partie de la table :

5. (8 points) Voici à nouveau une partie de la **version 2** de la grammaire du langage DFL, mais modifiée pour des fins de simplication.

[1]	*program*	→ **NAME** *expression* **end**
[2]	*expression*	→ *primitive_expression rest*
[3]	*expression*	→ *letblock_expression rest*
[4]	*expression*	→ *conditional_expression rest*
[5]	*primitive_expression*	→ **const**
[6]	*letblock_expression*	→ **let**
[7]	*conditional_expression*	→ **if**
[8]	*rest*	→ λ
[9]	*rest*	→ , *expression*

Voici les items LR(1) pour cette partie de la grammaire. Les abréviations suivantes ont été utilisées pour des fins de concision :

— P pour program ;

— E pour *expression* ;

— PE pour *primitive_expression* ;

— LE pour *letblock_expression* ;

— CE pour *conditional_expression* ;

— R pour *rest*.

$I_0 = (\{[S \rightarrow \bullet P, \{\#\}]\})$
$S \rightarrow \bullet P \ \{\#\}$
$P \rightarrow \bullet$**NAME** E **end** $\{\#\}$

$I_1 = goto(I_0, P)$
$S \rightarrow P \bullet \ \{\#\}$

$I_2 = goto(I_0,$ **NAME**$)$
$P \rightarrow$ **NAME** $\bullet E$ **end** $\{\#\}$
$E \rightarrow \bullet PE \ R \ \{\text{end}\}$
$E \rightarrow \bullet LE \ R \ \{\text{end}\}$
$E \rightarrow \bullet CE \ R \ \{\text{end}\}$
$PE \rightarrow \bullet$**const** $\{\text{end}/,\}$
$LE \rightarrow \bullet$**let** $\{\text{end}/,\}$
$CE \rightarrow \bullet$**if** $\{\text{end}/,\}$

$I_3 = goto(I_2, E)$
$P \rightarrow$ **NAME** $E \bullet$ **end** $\{\#\}$

$I_4 = goto(I_2, PE)$

$E \rightarrow PE \bullet R$ {end}

$R \rightarrow \bullet$ {end}

$R \rightarrow \bullet, E$ {end}

$I_5 = goto(I_2, LE)$

$E \rightarrow LE \bullet R$ {end}

$R \rightarrow \bullet$ {end}

$R \rightarrow \bullet, E$ {end}

$I_6 = goto(I_2, CE)$

$E \rightarrow CE \bullet R$ {end}

$R \rightarrow \bullet$ {end}

$R \rightarrow \bullet, E$ {end}

$I_7 = goto(I_2, \textbf{const})$

$PE \rightarrow \textbf{const} \bullet$ {end/,}

$I_8 = goto(I_2, \textbf{let})$

$LE \rightarrow \textbf{let} \bullet$ {end/,}

$I_9 = goto(I_2, \textbf{if})$

$CE \rightarrow \textbf{if} \bullet$ {end/,}

$I_{10} = goto(I_3, \textbf{end})$

$P \rightarrow \textbf{NAME}\ E\ \textbf{end} \bullet$ {#}

$I_{11} = goto(I_4, R)$

$E \rightarrow PE\ R \bullet$ {end}

$I_{12} = goto(I_4, ,)$

$R \rightarrow , \bullet E$ {end}

$E \rightarrow \bullet PE\ R$ {end}

$E \rightarrow \bullet LE\ R$ {end}

$E \rightarrow \bullet CE\ R$ {end}

$PE \rightarrow \bullet \textbf{const}$ {end/,}

$LE \rightarrow \bullet \textbf{let}$ {end/,}

$CE \rightarrow \bullet \textbf{if}$ {end/,}

$I_{13} = goto(I_5, R)$

$E \rightarrow LE\ R \bullet$ {end}

$goto(I_5, ,) = I_{12}$

$I_{14} = goto(I_6, R)$

$E \rightarrow CE\ R \bullet$ {end}

$goto(I_6, ,) = I_{12}$

$I_{15} = goto(I_{12}, E)$

$R \rightarrow , E \bullet$ {end}

$goto(I_{12}, PE) = I_4$

$goto(I_{12}, CE) = I_6$

$goto(I_{12}, \textbf{let}) = I_8$

$goto(I_{12}, LE) = I_5$

$goto(I_{12}, \textbf{const}) = I_7$

$goto(I_{12}, \textbf{if}) = I_9$

a) (1 point) Est-que cette partie de la grammaire est LR(1) ? Justifiez votre réponse.

b) (1 point) Est-que cette partie de la grammaire est LALR(1) ? Justifiez votre réponse.

c) (1 point) Est-que cette partie de la grammaire est SLR(1) ? Justifiez votre réponse.

d) (1 point) Est-que cette partie de la grammaire est LR(0) ? Justifiez votre réponse.

e) (4 points) Construisez la table *action* et la table *goto* pour un analyseur syntaxique ascendant LALR(1) pour cette partie de la grammaire. Donnez le contenu des tables pour les dix premiers états.

FIN DE L'EXAMEN

sciences-centreimpression

De: no-reply@www.usherbrooke.ca
Envoyé: 30 juillet 2015 14:24
À: sciences-centreimpression
Objet: COMMANDE EXAMENS IFT313 (Annexes) 5 août
Pièces jointes: lIFT313_Annexe_Final_E2015.pdf

TYPE-EXAMEN	FINAL
SIGLE-COURS	IFT313
TITRE-COURS	Introduction aux langages formels
PROFESSEUR	Richard St-Denis
DATE-HEURE	Mercredi 5 août à 9 h
AUTORISE-PAR	Richard St-Denis et André Mayers
NOMBRE-PAGES	4
NOMBRE-COPIE-PROF	25
IMPRESSION-QUESTIONNAIRE	Recto-verso broché
NOMBRE-FEUILLES-BLANCHES	
NOMBRE-PAPIER-GRAPHIQUE	
NOMBRE-CAHIERS	
CONSENTEMENT-AGES	1
REMARQUES	Annexe à imprimer recto-verso. Richard.St-Denis@usherbrooke.ca
E-MAIL	
FIRST-NAME	
LAST-NAME	
NICK-NAME	
SPAMSHIELD	true

Version 1 de la grammaire du langage DFL

```
program : 'program' '(' 'input' inputlist ')' 'yield' outputlist ';'
          (proceduredef)* expression 'end' '.' .

proceduredef : 'define' NAME ':='
               'procedure' '(' 'input' inputlist ')' 'yield' outputlist ';'
               ( proceduredef )* expression 'end' ';' .

inputlist : typedeclaration ( ',' typedeclaration )* .

typedeclaration : NAME ( ',' NAME )* ':' type .

outputlist : type ( ',' type )* .

namelist : NAME ( ',' NAME )* .

expression : ( primitive_expression | letblock_expression |
               conditional_expression | for_expression |
               forall_expression | application_expression )
               ( ',' expression )* .

letblock_expression : 'let' inputlist ':=' restricted_expression
                      ( ';' inputlist ':=' restricted_expression )*
                      'in' expression 'end' .

restricted_expression : ( primitive_expression | application_expression)
                        ( ',' (primitive_expression | application_expression ) )* .

conditional_expression : 'if' primitive_expression 'then'
                         expression 'else' expression 'end' .

for_expression : 'for' inputlist ':=' expression 'do'
                 'if' primitive_expression 'then'
                 ( iter_expression 'else' expression |
                   expression 'else' iter_expression ) 'end' .

iter_expression : 'iter' namelist ':=' restricted_expression
                  ( ';' namelist ':=' restricted_expression )*'end' .

forall_expression : 'forall' NAME 'in' '[' range ']' NAME ':='
                    expression 'construct' NAME 'end' .

application_expression : NAME '(' expression ')' .
```

```
primitive_expression : simple_expression relop simple_expression
                     | simple_expression .

simple_expression : term
                  | sign term
                  | term addop simple_expression .

term : factor
     | factor mulop term .

factor : NAME
       | const
       | '(' primitive_expression ')'
       | NAME '[' expression ']'
       | application_expression
       | 'not' factor .

sign : '-' .

addop : '+' | '-' | 'or' .

mulop : '*' | '/' | 'and' .

relop: '<' | '>' | '=' | '<>' | '<=' | '>=' .

type : standard_type
     | 'array' '[' range ( ',' range )* ']' 'of' standard_type .

range : const '..' const .

standard_type : 'integer' | 'real' | 'boolean' | 'char' .

const : INTEGER | REAL .
```

Version 2 de la grammaire du langage DFL

```
program : 'program' '(' 'input' inputlist ')' 'yield' outputlist ';'
          ( proceduredef )* expression 'end' '.' .

proceduredef : 'define' NAME ':='
               'procedure' '(' 'input' inputlist ')' 'yield' outputlist ';'
               ( proceduredef )* expression 'end' ';' .

inputlist : typedeclaration || ',' .

typedeclaration : ( NAME || ',' ) ':' type .

outputlist : type || ',' .

namelist : NAME || ',' .

expression : ( primitive_expression | letblock_expression |
               conditional_expression | for_expression |
               forall_expression ) [ ',' expression ] .

letblock_expression : 'let' inputlist ':=' restricted_expression
                      ( ';' inputlist ':=' restricted_expression )*
                      'in' expression 'end' .

restricted_expression : primitive_expression || ',' .

conditional_expression : 'if' primitive_expression 'then'
                         expression 'else' expression 'end' .

for_expression : 'for' inputlist ':=' expression 'do'
                 'if' primitive_expression 'then'
                 ( iter_expression 'else' expression |
                   expression 'else' iter_expression ) 'end' .

iter_expression : 'iter' ( namelist ':=' restricted_expression || ';' ) 'end' .

forall_expression : 'forall' NAME 'in' '[' range ']' NAME ':='
                    expression 'construct' NAME 'end' .

primitive_expression : simple_expression rest_primitive .

rest_primitive : ( '<' | '>' | '=' | '<>' | '<=' | '>=' ) simple_expression
               | .
```

```
simple_expression : term rest_simple
                  | '-' term .

rest_simple : ( '+' | '-' | 'or' ) simple_expression
            | .

term : factor rest_term .

rest_term : ( '*' | '/' | 'and' ) term
            | .

factor : NAME rest_factor
       | const
       | '(' primitive_expression ')'
       | 'not' factor .

rest_factor : '[' expression ']'
            | '(' expression ')'
            | .

type : standard_type
     | 'array' '[' ( range || ',' ) ']' 'of' standard_type  .

range : const '..' const .

standard_type : 'integer' | 'real' | 'boolean' | 'char'  .

const : INTEGER | REAL .
```