ANNEE UNIVERSITAIRE 2010/2011
1ERE SESSION DE PRINTEMPS

Parcours : CSB6
Code UE : INF354
Epreuve : Analyse Syntaxique
Date : 6 Mai 2011
Heure : 11H
Durée : 1H30
Documents : autorisés
Epreuve de : Mme Frédérique Carrère

Exercice 1

Le programme make sert à appeler des commandes générant des fichiers. Il détermine automatiquement quels fichiers nécessitent d'être regénérés, et exécute les commandes permettant cette génération. Ce programme utilise un fichier Makefile pour déterminer quels fichiers (cibles) doivent être mis à jour, de quels autres fichiers (prérequis) ils dépendent et si nécessaire, quelle commande appeler. Ce fichier Makefile contient une série de règles de la forme

```
cible1 cible2... : prérequis1 prérequis2...
            commande1
            commande2 ...
```

Le but de cet exercice est d'analyser les règles d'un fichier Makefile. Par exemple le fichier suivant :

```
CC=gcc
CFLAGS=-W -Wall -ansi -pedantic
EXEC=hello

all: $(EXEC)

hello: hello.o main.o
        $(CC) -o $@ $^

main.o: hello.h

%.o: %.c
        $(CC) -o $@ -c $< $(CFLAGS)

.PHONY: clean

clean:
        rm -f ${objects}\
              $(EXEC)\
              *~
```

Les tabulations et les retours à la ligne ont une valeur de séparateurs dans certains cas et sont de simples blancs dans d'autres. L'analyseur lexical devra donc les traiter différemment selon les cas.
Un mot est une suite non vide de caractères qui ne contient pas

- des blancs (espace, tabulation ou retour à la ligne)
- des caractères réservés (les deux points ou le point-virgule qui servent de séparateurs)

Un mot ne peut contenir une contre-oblique \ que si elle est immédiatement suivie d'un caractère différent du retour à la ligne. Les retours à la lignes précédés d'une contre-oblique \ seront interprétés comme de simples espaces et ignorés.
Les commandes sont séparées,

- soit par un retour à la ligne suivi d'une tabulation,
- soit par un point-virgule.

Les règles sont séparées par au moins un retour à la ligne.
Les token transmis à l'analyseur syntaxique seront MOT, SR (un séparateur de règles), SC (un séparateur de commandes).

Ecrivez un programme `lex` qui renvoient les bonnes valeurs de lexèmes (MOT, SR, SC et les caractères ' :' et '=') rencontrés. Dans le cas où le lexème est un mot, mettez la chaîne reconnue dans une variable *mot* de type $char*$.

Exercice 2

La grammaire suivante permet l'analyse syntaxique des lignes de commandes d'un fichier Makefile.

```
%token MOT SR SC
%%
Lignes  : Lignes Regle SR
          |
;

Regle     : Mots ':' Mots Commandes
          | Mots ':' Commandes
;

Mots    : Mots MOT
        | MOT
;

Commandes : Commandes SC Mots
          |
;
%%
```

1. A l'aide du fichier `y.output` donné en annexe, indiquez tous les états présentant un conflit pour l'algorithme LR(0).
2. Calculez les ensembles `Suivant` pour les non-terminaux `Regle` et `Commandes` de la grammaire.

3. Expliquez en détail comment l'algorithme SLR(1) permet de déterminer les actions à effectuer dans l'état 7.
4. En reprenant le Makefile donné en exemple dans l'exercice 1, indiquez l'état atteint après la lecture de " `main.o : hello.h` ". Détaillez l'évolution de la pile et les actions effectuées jusqu'à la lecture de "`%.o`".

Exercice 3

On souhaite analyser des expressions booléennes contenant également des affections de variables. Par exemple l'expression $3 < x = y = 2 * 4 < z = 8 < 10$; dont la valeur booléenne est *true*. Pour cela on utilise la grammaire suivante écrite dans le format reconnu par `yacc` :

```
%%
%token  ENTIER IDENT

%left '<' '>'
%left '+' '*'

%%
calcul  :          /* vide */
        |       calcul expr_bool ';'
;
expr_bool :  expr_bool '<' expr_bool
          |  expr_bool '>' expr_bool
          |  expr
;

expr : IDENT '=' expr
     | expr_num
;

expr_num : expr_num '+' expr_num
         | expr_num '*' expr_num
         | ENTIER
        ;
%%
```

1. On veut associer à chaque symbole de la grammaire deux attributs : un booléen et un entier (le type booléen est *bool* défini dans `stdbool.h`, une variable booléenne vaut *true* ou $false$).
   Indiquez les lignes à rajouter dans le fichier yacc pour pouvoir traiter ce type d'attributs.
2. Donnez les règles de lex permettant de reconnaître les entiers et d'initialiser leurs attributs. Le booléen vaudra *true* pour tous les entiers.
3. Dessinez l'arbre syntaxique de l'expression $3 < x = y = 2 * 4 < z = 8 < 10$; en respectant les priorités indiquées dans le fichier `yacc`.
4. Donnez des règles de calcul d'attributs qui permettent de calculer l'attribut booléen de toutes les expressions reconnues par la grammaire.

Annexe :

```
   0  $accept : Lignes $end
   1  Lignes : Lignes Regle SR
   2         |
   3  Regle : Mots ':' Mots Commandes
   4        | Mots ':' Commandes
   5  Mots : Mots MOT
   6       | MOT
   7  Commandes : Commandes SC Mots
   8            |

state 0
$accept : . Lignes $end  (0)
Lignes : .  (2)

.  reduce 2

Lignes  goto 1

state 1
$accept : Lignes . $end  (0)
Lignes : Lignes . Regle SR  (1)

$end  accept
MOT  shift 2
.  error

Regle  goto 3
Mots  goto 4

state 2
Mots : MOT .  (6)

.  reduce 6

state 3
Lignes : Lignes Regle . SR  (1)

SR  shift 5
.  error

state 4
Regle : Mots . ':' Mots Commandes  (3)
Regle : Mots . ':' Commandes  (4)
Mots : Mots . MOT  (5)
```

```
MOT  shift 6
':'  shift 7
.  error

state 5
Lignes : Lignes Regle SR .  (1)

.  reduce 1

state 6
Mots : Mots MOT .  (5)

.  reduce 5

state 7
Regle : Mots ':' . Mots Commandes  (3)
Regle : Mots ':' . Commandes  (4)
Commandes : .  (8)

MOT  shift 2
SR  reduce 8
SC  reduce 8

Mots  goto 8
Commandes  goto 9

state 8
Regle : Mots ':' Mots . Commandes  (3)
Mots : Mots . MOT  (5)
Commandes : .  (8)

MOT  shift 6
SR  reduce 8
SC  reduce 8

Commandes  goto 10

state 9
Regle : Mots ':' Commandes .  (4)
Commandes : Commandes . SC Mots  (7)

SC  shift 11
SR  reduce 4

state 10
Regle : Mots ':' Mots Commandes .  (3)
Commandes : Commandes . SC Mots  (7)
```

```
SC  shift 11
SR  reduce 3

state 11
Commandes : Commandes SC . Mots  (7)

MOT  shift 2
.  error

Mots  goto 12

state 12
Mots : Mots . MOT  (5)
Commandes : Commandes SC Mots .  (7)

MOT  shift 6
SR  reduce 7
SC  reduce 7

6 terminals, 5 nonterminals
9 grammar rules, 13 states
```