Introduction

Soit les règles de grammaire de déclaration de fonction à la C suivantes :

```
decl→type ID '(' lparam ')' ';'
lparam → param  rparam | ε
rparam → ',' param  rparam | ε
param → type ID
type→ INT | FLOAT
```

L'axiome est `decl` .

Le vocabulaire non terminal est : V_n={decl, lparam, rparam, param, type}.

Le vocabulaire terminal est : $\Sigma = \{',', ID, '(', INT, FLOAT, ')', ';'\}$ où `ID` représente un identificateur composé d'une lettre suivie de chiffres ou de lettres; où `INT` représente le mot-clé int , et `FLOAT` le mot-clé float.

Les espaces sont filtrés lors de l'analyse lexicale.

Analyse Descendante

Théorie

1. Donnez la liste des premiers et suivants pour chaque symbole non terminal
2. Calculez la table d'analyse descendante de l'automate à pile et représentez-la en abbrégeant les symboles non terminaux : decl (`de`), lparam (`lp`) rparam (`rp`), param (`p`), type (`ty`)
3. Effectuez la reconnaissance du mot `int f(float f,int x );` en représentant sur chaque ligne : la pile, le suffixe du mot commençant par le jeton courant, la règle de grammaire activée.
4. Dessinez l'arbre de dérivation correspondant à la reconnaissance précédente
5. La grammaire G est-elle LL(1) ? Justifiez votre réponse.

Pratique

On souhaite calculer le nombre de paramètres dans une déclaration.

1. Comment définir en bison et en flex, l'unique type C de l'attribut sémantique associé à chaque symbole ?
2. Ecrire un source flex réalisant l'analyse lexicale.
3. Ecrire un analyseur descendant récursif programmation réalisant cet interprèteur en utilisant la fonction `yylex()` définie précédemment à l'aide du source flex. L'interpréteur doit boucler tant que la ligne saisie n'est pas "q" (pour quitter).

Analyse Ascendante

Théorie

1. Dressez la collection canonique SLR de cette grammaire en remplaçant.
2. Dressez la table d'analyse ascendante de cette grammaire.
3. Combien de conflits possède cette table ? Pour chacun, énumérez le type de conflit, l'état et le jeton. Comment Bison résoud-il les conflits par défaut ?
4. Cette grammaire est-elle analysable en analyse descendante ? Pourquoi ?
5. Cette grammaire est-elle ambigüe ? Donnez un argument sans tenter la preuve.

Pratique

On souhaite écrire, à l'aide de bison et flex, un interpréteur qui calcule et affiche le nombre de paramètres d'une expression.

1. Ecrire un analyseur ascendant en bison, sans conflit, réalisant l'interprèteur sans changer la grammaire.
2. A l'aide de l'automate à pile produit par bison, analysez le mot `int f(float f,int x );` en décrivant les états successifs de
   la pile ainsi que les règles utilisées par bison (4 colonnes : pile, flot d'entrée, action, valeur sémantique calculée lors des
   réductions).
3. Dessiner l'arbre de dérivation associé à cette reconnaissance.