TP 3: Analyse syntaxique par descente récursive

J. Brunel, M. Couzinier, H. Cassé, M. Strecker

Maîtrise IUP ISI

1 Descente récursive

On s'intéresse à analyser un langage avec des expressions et une commande conditionnelle, donné par la grammaire suivante:

S → "print" NUM
| "if" E "then" S "else" S
E → NUM "=" NUM

Les terminaux NUM sont des nombres vus dans les TP précédents.

On suppose qu'on a la définition de terminaux suivante (voir fichier `recdesc.h`):

```
#define TOKTYPE enum token
TOKTYPE {IF, THEN, ELSE, BEG, END, PRINT, SEMI, NUM, EQ};
```

Pour l'instant, on ne s'occupe que de l'analyse syntaxique et on suppose que l'analyse lexicale a stocké les terminaux dans un tableau `tstr`. La position actuelle dans `tstr` est donnée par `tstr_ptr` et dans `tok`, on emmagasine le token actuel :

```
TOKTYPE tstr [] = {NUM, EQ, NUM};
int tstr_ptr;
TOKTYPE tok;
```

La définition de `tstr` en dessus correspond au résultat de l'analyse lexicale d'une expression comme "1 = 2". Quelles seront les listes de terminaux pour les phrases suivantes?

- `if 1 = 2 then print 3 else print 4`
- `if 1 = 2 then print else print 4`
- `if 1 = 2 then if 2 = 2 then print 3 else print 5 else print 4`

Créez un fichier `recdesc_tab.c` qui contient les `#include` nécessaires et les définitions précédentes. Puis, programmez:

- une fonction `start` qui met `tstr_ptr` à 0 et lit le premier token

- une fonction `advance` qui passe au terminal suivant et incrémente `tstr_ptr` de 1
- une fonction `error` qui imprime un message d'erreur.

`advance` et `error` sont utilisées dans `eat` qui compare le terminal attendu, `t`, avec le terminal actuel :

```
void eat(TOKTYPE t) {
  if (tok == t) advance(); else error();
}
```

Il est maintenant facile de coder la grammaire. Par exemple, pour la règle E, on obtiendra :

```
void E (void) {
  printf("Recognizing rule E\n");
  eat(NUM); eat(EQ); eat(NUM);
}
```

Codez les règles pour le nonterminal S et appelez S dans la fonction `main`:

```
int main (void) {
  start();
  S();
  return 0;
}
```

Testez votre analyseur syntaxique avec les listes de terminaux données plus haut.

2 Intégrer l'analyseur lexical Lex

Nous allons maintenant remplacer la lecture d'un tableau de terminaux par des appels à Lex.

Lex lit un fichier pointé par `yyin` et fournit des terminaux par des appels consécutifs à `yylex`. Donc, rajoutez une ligne

```
extern FILE *yyin;
```

Et adaptez les fonctions `start` et `advance` par des appels de la forme

```
tok = yylex();
```

Mis à part ces deux fonctions, l'analyse syntaxique est indépendante de la représentation des terminaux. Pour finir, il suffit de créer un fichier Lex `recdesc.l` (dont un squelette se trouve sur la page Web) et d'en rajouter les productions pour les terminaux de notre langage.

Après compilation et édition des liens, vous pouvez tester les exemples précédents, et l'expression `if 1 = 2 then print x else print y`. Adaptez le fichier Lex si vous n'obtenez pas le résultat attendu.

3 Actions sémantiques

D'habitude, on ne se limite pas à constater qu'un texte est syntaxiquement bien formé - on veut le manipuler. Donc il faut ajouter des actions sémantiques à chaque production de la grammaire.

Puisque notre langage ne contient que des constantes, on peut évaluer les commandes et ainsi déterminer quel nombre serait calculé. Par exemple, l'évaluation de `if 1 = 2 then print 3 else print 4` imprime le nombre 4.

Modifiez l'implantation de façon à ce que

- la fonction `E` retourne 0 si la comparaison évalue à faux et 1 sinon.
- la fonction `S` retourne la valeur qui serait calculée.

Bien sûr, il faut changer aussi les fonctions `start`, `advance` et `eat` pour prendre en compte la valeur du terminal NUM que l'on peut récupérer dans la variable `yytext`.

4 Un langage plus complexe ...

Quels problèmes se posent si on veut enchaîner les commandes (S), séparées par des point-virgules ? Transformez la grammaire de façon à permettre une implantation directe selon le schéma utilisé plus haut. La grammaire est-elle ambiguë ? Si oui, pour quelles raisons ? Faites des choix raisonnables et introduisez, si nécessaire, des mots-clé qui éliminent les ambiguïtés.

Testez sur les exemples :

- `if 1 = 2 then print 3; print 5 else print 4`
- `if 1 = 2 then print 3 else print 4; print 7`