

Projet de Compilation IUP 1

Université Paris 7 — 2001-2002

1 Objet du projet. Modalités de réalisation

Le projet est à réaliser en C, en utilisant `lex` et `yacc`. La date de remise du source (par mail) et du rapport, est fixée au mercredi 15 mai 2002. Chaque membre d'un groupe doit réaliser une partie non triviale du travail demandé qu'il devra exposer lors de la soutenance. Par ailleurs, chaque membre d'un groupe doit connaître les options générales choisies par le groupe ainsi que les difficultés rencontrées. Le projet sera présenté dans un rapport synthétique illustré par des jeux d'exemples, exposera les problèmes rencontrés et les choix effectués pour les résoudre. Le programme devra enfin pouvoir tourner sur plusieurs machines de l'UFR.

2 Sujet 1 : un petit shell (2 personnes)

Écrire un *shell* permettant d'interpréter des commandes et de lancer des processus avec synchronisation éventuelle. Il sera compatible `sh`, et fournira au moins les fonctionnalités suivantes :

- exécution de commandes simples, soit internes, soit externes, avec arguments éventuels ;
- les commandes internes suivantes devront être supportées : `source`, `alias`, `unalias`, `cd`, `exec`, `exit`, `export`, `pwd`, `test` et `umask` ;
- construction de lignes de commandes utilisant :
 - pipelines (`|`),
 - listes de pipelines construites avec le caractère de séquence (`;`),
 - listes de pipelines construites avec le caractère de concurrence (`&`),
 - listes de pipelines construites avec les opérateurs d'exécution conditionnelle (`&&` et `||`),
 - les parenthèses permettant d'exécuter une liste de commande dans un sous-shell ;
- définition et utilisation de paramètres, avec mécanisme d'expansion de paramètres ;
- redirections (`<`, `>`, `>>`) ;
- les mécanismes habituels de *quoting*, utilisant les caractères `\`, `"` et `'` ;
- structures de contrôle
 - `if...then...[elif ... then ...] else...fi`
 - `while ... ; do ... done`
 - `for nom [in ... ;] do ... ; done`

Il n'est pas demandé de supporter le *job-control*.

3 Sujet 2 : générateur d'analyseur syntaxique LR (3 personnes)

On demande la réalisation d'un générateur d'analyseur syntaxique basé sur les techniques LR. Le générateur lit une spécification de grammaire dans le fichier qui lui est donné en argument et génère un analyseur SLR(1) pour cette grammaire. La spécification de la forme du fichier décrivant la grammaire est la même que pour `yacc`. Les trois sections devront être présentes. Dans la première partie, on permettra en particulier le bloc littéral que le générateur se contentera de copier dans le code de l'analyseur. Les directives `%union`, `%token`, `%type`, `%right`, `%left` et `%nonassoc` seront aussi traitées.

La forme de chaque règle de grammaire devra aussi être du même type que celle employée par `yacc` :

```
non-terminal:  membre droit de la règle 1 { action 1 }
```

```
| membre droit de la règle 2 { action 2 }  
...  
;
```

chaque règle étant alors considérée comme distincte. Un membre droit peut par ailleurs être vide.

Le générateur d'analyseur syntaxique doit fournir :

- La liste des ensembles **Premier(X)** et **Suivant(X)** pour chaque non-terminal X.
- l'automate LR(0), consultable sous forme texte.
- La table d'analyse SLR(1), consultable sous forme texte.
- Le code d'un analyseur syntaxique prenant en entrée une chaîne de terminaux, construisant l'arbre de dérivation associé et indiquant les éventuelles erreurs.

Les *tokens* pourront être donnés par l'utilisateur sous forme littérale. Si la grammaire n'est pas SLR(1), l'analyseur doit l'indiquer. Pour lever les conflits, on choisira la même stratégie que celle utilisée par `yacc`. On introduira donc, en plus de la directive `%token`, les directives `%right`, `%left` et `%nonassoc`.

Les *tokens* pourront porter un attribut. Comme dans `yacc`, on pourra indiquer au niveau de la spécification de la grammaire des calculs d'attributs à effectuer. On se restreindra à des attributs synthétisés pour lesquels un seul parcours en profondeur de l'arbre de dérivation suffit. Dans une action, les attributs seront notés `$$`, `$1`, `$2`,... avec la même signification qu'en `yacc`. Le type des attributs sera de même donné dans une directive `%union`. Chaque action sera du type `$$=f($i,$j,...)`. Enfin, l'« initialisation » des attributs au niveau des feuilles de l'arbre se fera via la variable `yyval`.