Département d'Informatique
MASTER IFA 1ère année
Vendredi 9 janvier 2009

Contrôle terminal de Compilation

Durée : 3h. Documents autorisés : les polycopiés distribués en cours. Veiller à la présentation.

I Analyse syntaxique

Dans la grammaire suivante, les non-terminaux L et R représentent respectivement une l-valeur (une adresse mémoire) et une r-valeur (une valeur qui peut être stockée en mémoire), et le terminal * représente l'opérateur "contenu de" :

$$\begin{array}{lcl} S & \longrightarrow & L = R \mid R \\ L & \longrightarrow & * R \mid \textbf{id} \\ R & \longrightarrow & L \end{array}$$

1. Construire la table d'analyse LR(1).
2. Construire la table d'analyse LaLR(1).
3. La grammaire est-elle LR(1), LaLR(1) ? Justifier votre réponse.

II Traduction dirigée par la syntaxe

1 Affectations simultanées

Certains langages comme MAPLE permettent les affectations simultanées de plusieurs variables distinctes. Exemples :

- `(x,y) := (1,2)` attribue la valeur `1` à la variable `x`, et la valeur `2` à la variable `y`.
- après exécution de la séquence `(x,y) := (2,3); (x,y) := (x+y,x*y)`, la variable `x` a la valeur `5` et la variable `y` a la valeur `6`.
- `(x,y) := (y,x)` échange les valeurs de `x` et `y`.

1. Donner une grammaire pour les affectations simultanées de plusieurs variables. Le non-terminal E désignera une expression de la valeur d'une variable (ex. `x+y` ou `x-y`) et le terminal id, un identificateur. Il est inutile de donner les règles associées aux expressions. (La grammaire autorisera un nombre de variables (à gauche du `:=`) différent du nombre d'expressions (à droite du `:=`).)
2. Donner un schéma de traduction en analyse ascendante permettant de générer des quadruplets. On utilisera des temporaires pour mémoriser la valeur des variables. Par exemple :

   `(x,y) := (y,x)` se traduit par :
   ```
   T1 := y
   T2 := x
   x := T1
   y := T2
   ```

   où `T1` et `T2` sont des variables temporaires générées par le compilateur. (On fera abstraction de la partie analyse sémantique qui vérifie que les types sont cohérents, que toutes les variables (à gauche du `:=`) sont distinctes et que le nombre de ces variables est égal au nombre d'expressions (à droite du `:=`).) Expliquer quels sont les attributs et fonctions utilisés. (On utilisera notamment les attributs `E.ptr`, `id.ptr` et la fonction `newtemp()` vus en cours.)

2 Commandes gardées de Djikstra

On souhaite compiler un langage incluant une structure de contrôle inspirée des *commandes gardées* de Djikstra. La forme générale de cette structure est :

`(` B1 `->` S1 `[]` B2 `->` S2 `[]` ... `[]` Bn `->` Sn `)*`

où les Bi (i=1..n) sont des expressions booléennes et les Si sont des instructions (affectations ou structure de contrôle). Sa signification est :

```
do
   if B1 then S1
         else if B2 then S2
                    else  ... if Bn then Sn
while (B1 ∨ B2 ∨ ...∨ Bn)
```

L'exécution termine lorsque tous les booléens sont à faux ($\vee$ désigne le "ou" logique). La grammaire du langage est :

S	$\longrightarrow$	`(` M C `)*` | `begin` L `end` | A
L	$\longrightarrow$	L `;` M S | S
C	$\longrightarrow$	B `->` M S | C `[]` B `->` M S
M	$\longrightarrow$	ϵ

où le non-terminal B désigne une expression booléenne, le non-terminal A, une affectation et le non-terminal M, un marqueur ajouté à la grammaire dans le but de réaliser la traduction.

Associer une action à chacune de ces règles de grammaires de manière à définir un schéma de traduction en analyse ascendante qui génére des quadruplets. On suppose que les expressions booléennes se traduisent par une position de quadruplet comme il a été vu en cours. Expliquer quels sont les attributs et fonctions utilisés. (On utilisera notamment les attributs `B.true`, `B.false`, `S.next`, `M.quad`, la fonction `complete` et la variable `nextquad` vus en cours.)

III Optimisations

On considère le code intermédiaire issu de la traduction d'affectations simultanées (cf. II 1.) et on applique les optimisations suivantes : propagation de copie et élimination de code mort. Donner le code intermédiaire et le code optimisé correspondant aux programmes :

1. `(x,y) := (y,x)`
2. `(x,y,z) := (z,x,y)`
3. `(x,y,z) := (y,z,x)`