Examen de compilation

Christian Rinderknecht

Lundi 2 février 2004

Durée : deux heures. Documents et calculatrices interdits.

1 Syntaxe

1. Soient les non-terminaux `Expression`, `Term`, `Sign`, `Factor` et `Expressions`. Soit `Expressions` le symbole de départ. Soient les terminaux `number` et `ident`. Soit `epsilon` le mot vide (ε). Soient les productions

```
Expressions ::= Expression
              | Expression "," Expressions
Expression  ::= Term
              | Expression "+" Term
Term        ::= Sign Factor
              | Term "*" Sign Factor
Sign        ::= epsilon
              | "-"
Factor      ::= ident
              | number
              | "(" ")"
              | "(" Expression ")"
              | Factor "(" ")"
              | Factor "(" Expressions ")"
```

   Questions :

   (a) Factorisez la grammaire à gauche et supprimez les récursions à gauche.

   (b) Pour chaque non-terminal de cette nouvelle grammaire, calculez ses ensembles de premiers lexèmes ($\mathcal{P}$) et de lexèmes suivants ($\mathcal{S}$).

   (c) En déduire quels sont les non-terminaux qui peuvent dériver ε.

   (d) Construisez la table d'analyse descendante. La grammaire est-elle LL(1) ?

   (e) Donnez une grammaire équivalente en EBNF (question indépendante).

2. Trouvez une grammaire EBNF non récursive qui spécifie la syntaxe des nombres à virgule flottante dont la spécification est la suivante :
   — Un nombre à virgule flottante est constitué de trois parties : un signe, une mantisse et un exposant. Elle sont toujours placées dans cet ordre. Le signe et l'exposant sont optionnels.
   — Un signe est formé d'un seul caractère `+` ou `-`.
   — La mantisse est une suite non vide de chiffres contenant un caractère `.` (point décimal) qui est placé au début, au sein ou à la fin de la suite de chiffres.
   — L'exposant est constitué de trois parties : un caractère `e`, un signe et une suite non vide de chiffres. Ces trois parties sont toujours placées dans cet ordre. Le signe est optionnel.

   Voici quelques exemples : `-3.14159  1.4e10  127.0e-2  -0.1e-1  .0`

2 Sémantique

Soit le langage fonctionnel Micro-ML dont la syntaxe concrète est spécifiée par la grammaire en BNF

```
Expression ::= integer
             | "(" Expression ")"
             | ident
             | "let" ident "=" Expression "in" Expression
             | "fun" ident "->" Expression
             | Expression Expression
```

La syntaxe abstraite associée est

type expr = Int **of** *int*
| Var **of** *string*
| Let **of** *string* * *expr* * *expr*
| Fun **of** *string* * *expr*
| App **of** *expr* * *expr*

La sémantique opérationnelle $\rho \vdash e \twoheadrightarrow v$ est la plus petite relation définie par les règles d'inférence

$$\mathsf{Const}\ n \twoheadrightarrow \dot{n} \quad \textsc{const} \qquad \frac{x \in \mathrm{dom}(\rho)}{\rho \vdash \mathsf{Var}\ x \twoheadrightarrow \rho(x)}\ \textsc{var}$$

$$\frac{\rho \vdash e_1 \twoheadrightarrow v_1 \qquad \rho \oplus x \mapsto v_1 \vdash e_2 \twoheadrightarrow v_2}{\rho \vdash \mathsf{Let}\,(x, e_1, e_2) \twoheadrightarrow v_2}\ \textsc{let}$$

$$\rho \vdash \mathsf{Fun}\,(x, e) \twoheadrightarrow \mathsf{Clos}\,(x, e, \rho) \quad \textsc{abs}$$

$$\frac{\rho \vdash e_1 \twoheadrightarrow \mathsf{Clos}\,(x_0, e_0, \rho_0) \qquad \rho \vdash e_2 \twoheadrightarrow v_2 \qquad \rho_0 \oplus x_0 \mapsto v_2 \vdash e_0 \twoheadrightarrow v_0}{\rho \vdash \mathsf{App}\,(e_1, e_2) \twoheadrightarrow v_0}\ \textsc{app}$$

où le type des valeurs est :

type *value* = Int **of** *int* | Clos **of** *string* * *expr* * (*string* → *value*);;

et l'environnement ρ est une fonction partielle des variables vers les valeurs.

1. Montrez que la grammaire algébrique est ambiguë.
2. Prouvez que le programme 1 2 ne peut être évalué. Quelle différence voyez-vous entre cet exemple et (**fun** f → f f) (**fun** f → f f), donné dans le cour ?
3. Donnez la définition standard de la fonction $\mathcal{L}$ qui donne les variables libres d'une expression de Micro-ML.
4. Vérifiez que les constructions **let** x = e_1 **in** e_2 et (**fun** x → e_2) e_1 sont équivalentes du point de vue de l'évaluation — c'est-à-dire que l'une produit une valeur v si et seulement si l'autre produit également v.