RICM1 - 2005/06
Langage et Programmation 2

TP2

Rappel

Les sources du TP sont à rendre *impérativement* le lundi qui précède le TP suivant. Il s'agit donc pour ce TP, organisé sur deux séances, du lundi 20 mars.

Objectifs du TP

L'objectif final est de parvenir à un programme qui permet d'évaluer des expressions. L'écriture d'un tel programme se fait en quatre étapes :

1. Définition du type de la structure de données représentant les expressions
2. Réalisation d'un analyseur lexical
3. Réalisation d'un analyseur syntaxique
4. Écriture de la fonction `evaluation` proprement dite

On ne traitera que des expressions bien formées. On procédera incrémentalement, chaque phase élargissant l'ensemble des expressions que l'on souhaite traiter.
– expressions additives
– expressions arithmétiques
– expressions booléennes
– let-expressions

Dans un premier temps on considèrera donc les expressions additives puis arithmétiques. Si, par exemple, on donne la chaîne de caractères "$4 - 2 * 3$", l'évaluation devra répondre 6. On supposera qu'une expression est composée au moins d'un entier.

1 Expressions additives

L'analyseur lexical

Le travail de l'analyseur est de prendre une chaîne de caractère et de la transformer en flot (*stream*) de lexèmes (*tokens*) pour qu'elle puisse être traitée par l'analyseur syntaxique. Pour réaliser l'analyseur lexical, vous aurez besoin de :
– Une fonction intermédiaire qui transforme la chaîne de caractères en flot de caractères.
– Le schéma de Horner.
– La définition d'un type *token* qui définit chaque symbole des expressions arithmétiques (`type token = TPlus | ...`).

L'expression "$4-2+3$" sera alors tranformée en un flot, en l'occurence `[< TEnt 4 ; TMoins ; TEnt 2 ; TPlus ; TEnt 3 >]`

Le type des expressions

Comme on l'a vu en TD, la structure de représentation des expressions arithmétiques la plus appropriée est la structure d'arbre. L'expression "$4-2+3$" sera alors représentée par l'arbre `Plus(Moins(Int 4, Int 2),Int 3)`.

Le type d'une expression est :

```
type expr =
  | Int of int
  | Plus of expr * expr
  | Moins of expr * expr
```

L'analyseur syntaxique

L'analyseur syntaxique doit transformer le flot de tokens en un arbre de type `expr`. Par exemple, le flot `[< TEnt 4 ; TMoins ; TEnt 2 ; TPlus ; TEnt 3 >]` doit être transformé en l'arbre `Plus(Moins(Int 4, Int 2),Int 3)`.

Grammaire Pour pouvoir contruire l'arbre, vous aurez besoin de la grammaire des expressions additives.

$$\begin{array}{lcl} E & \longrightarrow & T\ A \\ A & \longrightarrow & '+'\ T\ A \ \mid\ '-'\ T\ A \ \mid\ \epsilon \\ T & \longrightarrow & \textit{entier} \ \mid\ '('\ E\ ')' \end{array}$$

Signification des non terminaux : E expression, T terme, A suite d'additions.

L'évaluation

La fonction d'évaluation correspond à un parcours de l'arbre :

```
let rec evaluation e =
  match e with
    | Int n -> n
    | Plus (e1,e2) -> (evaluation e1) + (evaluation e2)
    | Moins (e1,e2) -> (evaluation e1) - (evaluation e2)
```

Discussion

Réalisez un autre analyseur lexical à partir de la grammaire suivante. Observez la manière différente de construire le résultat. Expliquez pourquoi cet analyseur boucle.

$$
\begin{array}{lcl}
E & \longrightarrow & E\,'+'\,T \mid E\,'-'\,T \mid T \\
T & \longrightarrow & \mathit{entier} \mid '('\,E\,')'
\end{array}
$$

2 Expressions arithmétiques

Note : Ne vous sentez pas obligé d'attendre la seconde séance pour démarrer cette phase !

L'analyseur lexical

Introduire des *lexèmes* pour les opérateurs de multiplication, division et pour les parenthèses.

Le type des expressions

L'expression "$(4-2)*3$" sera alors représentée par l'arbre `Mult(Moins(Int 4, Int 2),Int 3)`.

```
type expr =
  | Int of int
  | Plus of expr * expr
  | Moins of expr * expr
  | Mult of expr * expr
  | Div of expr * expr
```

L'analyseur syntaxique

L'analyseur syntaxique doit transformer le flot de tokens en un arbre de type *expr*. Par exemple, le flot `[< TParou ; TEnt 4 ; TMoins ; TEnt 2 ; TParferm ; TMult ; TEnt 3 >]` doit être transformé en l'arbre `Mult(Moins(Int 4, Int 2),Int 3)`.
Il faudra donc améliorer la grammaire précédente (en veillant à ce qu'elle reste récursive à droite) pour qu'elle prenne en compte la multiplication et la division.

L'évaluation

La fonction d'évaluation correspond toujours à un parcours de l'arbre.

3 Expressions booléennes

Reproduire la démarche ci-dessus pour des expressions purement booléeenes (sur les deux constantes notées `true` et `false`).

Il faut donc compléter l'analyseur lexical en introduisant un constructeur pour les identificateurs (suites de lettres, éventuellement suites de lettres et de chiffres commençant par une lettre).

Les deux seuls identificateurs réellement utilisés à ce stade sont donc `true` et `false`, mais d'autres apparaîtront par la suite.

4 Expressions arithmétiques et booléennes

Introduire une expression booléenne pour le test d'égalité entre deux expressions arithmétiques et ajouter une construction `si ...alors ...sinon ...` dans les expressions arithmétiques.

Grammaire Pour pouvoir contruire l'arbre, vous aurez besoin de la grammaire suivante :

$$\begin{array}{rcl}
E & \longrightarrow & 'si'\ E\ 'alors'\ E\ 'sinon'\ E \mid CNJ\ D \\
D & \longrightarrow & '\vee'\ CNJ\ D \mid \epsilon \\
CNJ & \longrightarrow & L\ C \\
C & \longrightarrow & '\wedge'\ L\ C \mid \epsilon \\
L & \longrightarrow & '\neg'\ L \mid EC\ CMP \\
CMP & \longrightarrow & '='\ EC \mid \epsilon \\
EC & \longrightarrow & T\ A \\
A & \longrightarrow & '+'\ T\ A \mid '-'\ T\ A \mid \epsilon \\
T & \longrightarrow & F\ SM \\
M & \longrightarrow & '\times'\ F\ M \mid '/'\ F\ M \mid \epsilon \\
F & \longrightarrow & entier \mid '('\ E\ ')' \mid 'vrai' \mid 'faux'
\end{array}$$

Signification des non terminaux : E expression, T terme, A suite d'additions, F facteur, M suite de multiplications, CNJ conjonction, CMP comparaison, L littéral, C suite de conjonctions, D suite de disjonctions, EC expressions comparables.

5 Let-expressions (*bonus*)

Introduire une construction `let ...in ...`. Il faut prévoir une notion d'identificateur dans les lexèmes et une notion de variable dans les expressions.

Les *productions* à ajouter à la grammaire sont :

$$\begin{array}{rcl}
E & \longrightarrow & 'soit'\ ident\ '='\ E\ 'dans'\ E \\
F & \longrightarrow & ident
\end{array}$$

Notez que pour l'évaluation, il vous faudra introduire une notion d'environnement.