Analyse syntaxique : le noyau ML

Julien FORGET

4 mai 2009

L'objectif de ces TPs est de construire le *front-end* complet d'un noyau fonctionnel de type ML que nous appellerons MINIML. Le travail se décompose en deux phases : analyse syntaxique puis typage. Nous n'étudierons pas la partie *back-end*, à savoir la génération de code à partir du résultat du front-end.

Le travail effectué au cours des séances de TPs sera relevé et noté (à la fin du cycle d'enseignement), vous prendrez donc soin de bien présenter et de bien commenter votre code.

Préambule : la compilation séparée

Le code que nous allons développer sera séparé en plusieurs fichiers, ce qui nécessitera d'utiliser la compilation séparée. En OCAML la compilation séparée est très liée au système de *modules*. Sans entrer dans les détails :

- A chaque fichier $fic.ml$ correspond un module Fic (attention à la majuscule au début).
- On accèdera dans un fichier $fic1.ml$ aux déclarations réalisées dans un fichier extérieur $fic2.ml$ en rajoutant la ligne `open Fic1` au début du fichier $fic2.ml$.

Attention, ceci est une vision très schématique pour simplifier ce TP. Ce n'est pas la seule manière de faire de la compilation séparée et l'utilisation des modules ne se limite absolument pas à la compilation séparée.

1 Présentation du langage

Le langage que nous allons étudier, est une version minimaliste de noyau fonctionnel de la famille ML (dont font partie SML, Haskell et OCaml).

Une **expression** exp pourra être un des éléments suivants :

- *Variable* : `x:int`, `y:bool`, ...
- *Constante entière* : 1, 2, 3, ...
- *Constante booléenne* : true, false.
- *Opérateur prédéfini* : +, -, *, /, =.
- *Branchement conditionnel* : **`if`** `exp` **`then`** `exp1` **`else`** `exp2`
- *Définition de fonction* : **`fun`** `x:typ -> exp`. On accepte un seul paramètre à la fois et il doit être typé (`bool` ou `int`).

– *Application de fonction* : `exp1 exp2`.

Exemple d'expression valide : `(fun x:int -> x+2) 4`.

Un **programme** est une liste d'expressions se terminant chacune par " ; ;".

2 Analyse syntaxique

1. Arbre de Syntaxe abstraite.
   (a) Dans un fichier `ast.ml`, définissez le type `expr` correspondant aux expressions à l'aide d'un type union. On utilisera un cas différent pour chaque opérateur prédéfini.
   (b) A l'aide du type `expr`, définissez l'expression `fun x -> x+2` et évaluez là dans l'interpréteur OCAML.
   (c) Définissez le type d'un programme `prog`.
2. Analyse lexicale.
   (a) Dans un fichier `parser.mly`, déclarez les lexèmes de MINIML : entier, identifiant et les différents mots et symboles réservés du langage. Ajoutez aussi un lexème `EOF` pour gérer la fin de fichier.
   (b) Dans un fichier `lexer.mll`, définissez l'analyseur lexical reconnaissant ces lexèmes. On ajoutera deux règles particulières :
   – `[' ' '\n'] {token lexbuf}` : permet de sauter les blancs et les retours à la ligne.
   – `eof {EOF}` : détecte la fin de fichier.
3. Analyse grammaticale
   (a) Commencez par compléter les déclarations du parser en ajoutant les règles d'associativité et de priorité des différents lexèmes.
   (b) Définissez la règle de grammaire `exp` permettant de reconnaître une expression.
   (c) Définissez la règle de grammaire `prog` permettant de reconnaître un programme. Attention, un programme doit se terminer par `EOF`.
   (d) Complétez la partie déclarations en ajoutant la déclaration du symbole de départ et de son type.
4. Dans un fichier `main.ml`, définissez la fonction principale permettant d'effectuer l'analyse syntaxique d'un fichier source MINIML.
   (a) Pour simplifier, le nom du fichier à analyser sera spécifié dans le programme principal à l'aide d'une variable `source`.
   (b) La fonction principale se contente de faire appel au parser et au lexer appliqués sur ce fichier source (voir cours).
   (c) Compilez à l'aide du `Makefile` fourni pour ce TP :
      i. Créez un fichier `.depend` (`touch .depend`).

ii. Calculez les dépendances de données liant les différents fichiers (`make depend`).

iii. Lancez la compilation (`make`).

(d) Vérifiez sur quelques exemples que votre programme ne lève pas d'erreur à l'exécution.