Projet Programmation 2 — 1ère partie

Analyseur pour un mini Java

A rendre au plus tard le 22 avril 2005 à 19h.

Ce document et les différents fichiers nécessaires pour la réalisation du projet sont disponibles à l'adresse `http://www.liafa.jussieu.fr/~sighirea/cours/prog2`.
Consultez également la liste de diffusion du cours sur `http://list.info.licence.univ-paris7.fr/`.

Table des matières

1 But du projet et fichiers fournis

La première partie du projet consiste à réaliser un analyseur lexical et syntaxique pour un sous-ensemble du langage Java, qu'on appellera mini-Java et qui est décrit dans ce document. La deuxième partie du projet ajoutera à cet analyseur un vérificateur de types.
Pour cette première partie, plusieurs fichiers utiles sont fournis :

- `javasyntax.ml` contient la syntaxe abstraite du langage, donnée sous la forme de types OCAML.
- `analyseur.ml` définit plusieurs fonctions d'analyse syntaxique spécialisées pour lire un programme ou des expressions mini-Java à partir d'une chaîne ou à partir d'un fichier. On pourra y ajouter d'autres fonctions si c'est utile.
- `TESTS/*.java` contienent quelques exemples de programmes.
- `A_LIRE_1` indique le travail à effectuer pour la première partie (fichiers à écrire) et comment compiler puis utiliser ces fichiers.

Dans le cadre de la première partie du projet, vous devez écrire deux fichiers :
- `javalexer.mll` doit contenir une description d'analyseur lexical pour le langage.
- `javaparser.mly` doit contenir une description d'analyseur syntaxique pour le langage.

Les fonctions du fichier `analyseur.ml` font appel à ces deux analyseurs.

La syntaxe concrète acceptée par ces deux analyseurs est décrite informellement dans la section **??**. Avant d'écrire les analyseurs, il faudra écrire une grammaire BNF qui formalise la syntaxe concrète du langage. Ceci vous aidera à écrire ensuite les définitions de lexèmes en OCAMLLEX et la grammaire OCAMLYACC. **L'écriture de la grammaire BNF fait partie du projet**.

2 Syntaxe abstraite de mini-Java

Un programme mini-Java correspond au type `javaprog`.

```
type  'a option = Nopt | Some of 'a;;
type   status = InstStatus | ClassStatus;;

type javaprog = JavaProg of javaclassdecl list
```

```
and javaclassdecl = {classname: string;
                     classfather: string option;
                     fielddecls: javafielddecl list;
                     methdecls: javamethdecl list}

and javamethdecl = {methname: string;
                    methrestype: javatype;
                    methparams: (string * javatype) list;
                    methstatus: status;
                    methbody: javastat}

and javafielddecl = {fieldname: string;
                     fieldtype: javatype;
                     fieldstatus: status;
                     fieldinit: javaexp option}

and javavardecl = {varname: string;
                   vartype: javatype;
                   varinit: javaexp option}

and javatype = JavaVoidType | JavaBoolType | JavaIntType
             | JavaIdType of javaident
             | JavaNullType

and javaident = string

and javaexp = JavaConstExp of javaconst
            | JavaThisExp
            | JavaVarExp of javaident
            | JavaNewClassExp of javaident
            | JavaMethCall of javaexp * string * (javaexp list)
            | JavaCall of string *  (javaexp list)
            | JavaFieldExp of javaexp * string
            | JavaUnopExp of javaunop * javaexp
            | JavaBinopExp of javabinop * javaexp * javaexp
            | JavaAssign of javaexp * javaexp

and javaconst = JavaInt of int | JavaTrue | JavaFalse | JavaNull

and javaunop = JavaUMOINS | JavaNOT

and javabinop = JavaAND | JavaOR | JavaPLUS | JavaMOINS | JavaMULT | JavaDIV
              | JavaEGAL | JavaNE | JavaLT | JavaGT | JavaLE | JavaGE

and javastat = JavaBlock of (javavardecl list) * (javastat list)
             | JavaComput of javaexp
             | JavaIf of javaexp * javastat * javastat
             | JavaFor of javaexp * javaexp * javaexp * javastat
             | JavaReturn of javaexp option
             | JavaNop
```

;;

3 Syntaxe concrète de mini-Java

Les mots-clé du langage seront écrits en caractères accentués.

3.1 Programme

Un programme mini-Java est une liste de dćlarations de classes. Par exemple :

```
// fichier TESTS/test.java
class Point {
  // declaration champs et methodes
}
class Pixel extends Point {
  // declaration champs et methodes
}
```

3.2 Déclaration de classe

Une déclaration de classe est introduite par le mot-clé **class** et spécifie le nom d'une classe mère (optionnel, introduit par le mot clé **extends**) suivi de la liste des champs et des méthodes de cette classe entre des accolades ('{', '}'). Il est important de noter qu'une classe mini-Java ne contient pas de déclarations explicites de constructeurs.
La déclaration d'une classe peut être suivie d'un caractère ';'.

3.3 Déclaration de champ

Un champ est déclaré en spécifiant son statut, sont type et son nom. Toute déclaration doit être finie par un caractère ';'.
L'exemple suivant déclare `x` et `y` comme deux champs entiers ayant le statut de champs d'instance, et un champ entier `nbPoints` ayant le statut champ de classe (mot-clé **static**).

```
int x;
int y;
static int nbPoints;
```

3.4 Déclaration de méthode

Une déclaration de méthode spécifie dans l'ordre : le statut de la méthode (de classe, **static**, ou par défaut d'instance), le type de retour de la méthode, le nom de la méthode, sa liste de paramètres formels entre paranthèses ('(' et ')'), et son corps qui est un bloc d'instructions. Les paramètres formels sont séparés par une virgule. Par exemple :

```
int gety() {
  return y;
}
static Point createPoint(int x0, int y0)
{
  Point p = new Point();
  p.x=x0;
  p.y=y0;
```

```
    return p;
  }
```

3.5 Types

Un type mini-Java peut être un type entier (**int**), un type booléen (**bool**), un type vide (**void**), ou un type référence vers un objet d'une classe.
Par exemple, le champ `x` de la classe `Point` est de type entier ; la méthode `createPoint` ci-dessus renvoie un résultat de type référence vers un objet de la classe `Point`.
En mini-Java, on n'a pas de types tableau.

3.6 Expressions

Les expressions de Java considérées en mini-Java sont :
- les nombres entiers, les constantes booléennes (**true** ou **false**) ou **null** ;
- les variables et la variable spéciale **this** ;
- la création d'un objet, par exemple **`newPoint()`** ;
- l'appel d'une méthode de la classe courante, par exemple `getx()` ;
- l'appel d'une méthode sur un objet, par exemple `p.getx()` ;
- l'accès à un champ d'un objet, par exemple `p.x` ;
- l'appel d'un opérateur arithmétique ou logique parmi `||`, &&, `==, !=, !, <, <=, >, >=, +, -, *, /`.
- l'affectation d'une variable ou d'un champ.

3.7 Instructions

Les instructions de Java considérées en mini-Java sont :
- le bloc d'instruction qui doit contenir un liste de déclarations de variable suivie d'une liste d'instructions ; les déclarations de variables sont toujours en début de bloc ;
- une expression suivie du caractère ';' ;
- l'instruction conditionnelle **if** ;
- la boucle **for** sans déclaration de variables dans l'expression d'initialisation ;
- le retour d'une méthode avec ou sans envoi de résultat, **return** (terminée par un ';') ;
- l'instruction vide ';'.

4 Travail à réaliser pour la première partie

1. Programmer en OCAML un analyseur lexical et syntaxique pour le langage mini-Java. L'analyseur lexical sera décrit dans le fichier `javalexer.mll` et l'analyseur syntaxique dans le fichier `javaparser.mly`.
2. Ajouter aux exemples donnés, d'autres exemples permettant de tester le bon fonctionnement de l'analyseur.
3. Dans un document annexe `doc.tex` ou `doc.txt`, rédiger la première partie de votre rapport de projet. Dans une première partie, cette documentation doit contenir la grammaire BNF que vous avez utilisée pour votre analyseur. Dans une deuxième partie, vous devez commenter chacun des exemples testés et le comportement de votre analyseur sur ces exemples.

5 Remise du projet

La première partie du projet sera envoyé aux enseignants avant sa soutenance en utilisant le programme `/ens/prieur/Licence/Prog2/remise` au plus tard le 22 avril à 19h. Un fichier d'explication `remise.txt` se trouve dans le même répertoire.

Ce qui doit être envoyé est un fichier d'archive Unix `<login>.tar` contenant (au minimum) :
- Le fichier `javalexer.mll`.
- Le fichier `javaparser.mly`.
- Le fichier `javasyntax.ml` fourni.
- Le fichier `analyseur.ml` fourni, eventuellement modifié.
- Le fichier `Makefile` qui permet de compiler le projet.
- Le répertoire `TESTS` contenant vos tests.
- La documentation dans le fichier `doc.tex` ou `doc.txt`.

Pour crér ce fichier, dans le répertoire où se trouve votre projet, taper :

```
tar cvf <login>.tar Makefile javalexer.mll javaparser.mly javasyntax.ml analyseur.ml
TESTS doc.tex(txt).
```