UPMC/master/info/APS-4I503

Janvier 2015

Syntaxe

Analyse des Programmes et Sémantiques

Étude d'un *langage de programmation*.

Ce ne sera pas apprendre à programmer avec ce langage.
Ce sera savoir parler de *tous les programmes* que l'on peut écrire avec ce langage.

Que peut-on dire de *tous les programmes* ?

Analyse des programmes

Le programme comme *objet* d'étude

Quantitaif : nombre de lignes de code, nombre de classes définies, performences temps/espace
Qualitatif : *typage*, robustesse, complétude fonctionnelle

Analyse du *code source*

Nota : tout n'est pas atteignable de cette manière

Code source

SYNTAXE

Définir la syntaxe = définir la *grammaire* du langage

Définition inductive

Une (méta)syntaxe pour les grammaires : BNF

Analyse code source = Analyse *dirigée par la syntaxe*

Notre langage

BOPL : un noyau d'un *langage à objets*

Rèf : Palsberg, J. et Schwartzbach, M. I., Object-Oriented Type Systems. Wiley, 1994.

On y ajoutera des *extensions*

Basic **O**bject **P**rogramming **L**anguage

Un programme BOPL

```
program

  class Point is
  vars
     Int x, y ;
  methods
    Point add(Point p)
    let
      Point ret ;
    in
    begin
      ret := new Point ;
      ret.x := (self.x) + (p.x) ;
      ret.y := (self.y) + (p.y) ;
      return ret
    end
  end
```

```
let
  Point p1, p2, p3 ;
in

begin
  p1 := new Point ;
  p1.x := 1 ;
  p1.y := 2 ;
  p2 := new Point ;
  p2.x := 5 ;
  p2.y := 10 ;
  p3 := p1.add(p2) ;
  writeln(p3)
end
```

Structure d'un programme BOPL

- En tête – *mot clef* `program` :
  - Déclaration(s) de classe(s) – mots clef `class` et `is` ;
    - Variables d'instances – `vars`
    - Méthode(s) – `methods`
      - Signature
      - Variables locales
      - Corps de la méthode
  - Déclarations de variables (locales) – mots clef `let` et `in` ;
  - Corps du programme (*main*) – mots clef `begin` et `end`.

Hiérarchie – *indentation* – *arborescence*

Grammaire 1

Les programmes BOPL

Prog	: :=	`program` Classes Locals InstrList
Classes	: :=	ε | Class Classes
Locals	: :=	ε | `let` VarDecs `in`
InstrList	: :=	`begin` InstrSeq `end`

Typographie et éléments de grammaires

- *symboles non terminaux* (Petites Capitales)
- *symboles terminaux* (`Machine à écrire` ou sans sérif)
- *élément vide* : ε
- *(méta)symbole réservés* (: := et |)
- séparateurs (esp. tab. et crlf)

Nota : définition *récursive* de Classes

Grammaire 2

Définition de classe

$$
\begin{array}{lll}
\textsc{Class} & ::= & \texttt{class}\ \text{id}\ \textsc{Extends}\ \texttt{is} \\
 & & \textsc{VarDecList}\ \textsc{MethodList} \\
 & & \\
\textsc{Extends} & ::= & \varepsilon \mid \texttt{extends}\ \textsc{ClassType} \\
\textsc{VarDecList} & ::= & \varepsilon \mid \texttt{vars}\ \textsc{VarDecs} \\
\textsc{MethodList} & ::= & \varepsilon \mid \texttt{methods}\ \textsc{Methods}
\end{array}
$$

Avec id : ensemble des *identificateurs* (symboles *terminaux*)

Grammaire 3

Types/classes et variables

ClassType	: :=	`Obj` | `Void` | `Int` | `Bool` | id
VarDecs	: :=	VarDec | VarDecs VarDec
VarDec	: :=	ClassType Ids `;`
Ids	: :=	Ids `,` id

- 4 types/classes prédéfinies (mots réservés)
- définitions récursives des suites (non vides) VarDecs et Ids
- symboles séparateurs réserves (`;` et `,`)

Grammaire 4

Déclarations de méthodes

Methods	: :=	Method | Methods Method
Method	: :=	ClassType id (ArgDecList) Locals InstrList
ArgDecList	: :=	ε | ArgDecs
ArgDecs	: :=	ArgDec | ArgDecs ; ArgDec
ArgDec	: :=	ClassType id
InstrSeq	: :=	Instr | InstrSeq ; Instr

Nota : 3 clauses pour la liste des paramètres (possiblement vide et séparateur infixe)

Grammaire 5

Instructions

```
Instr  ::=  Exp.id ( ArgList )
         |  id := Exp
         |  Exp.id := Exp
         |  return Exp
         |  writeln Exp
         |  if Exp then InstrList else InstrList
         |  while Exp do InstrList
```

Grammaire 6

Expressions

Exp	: :=	Exp + Term | Exp - Term
	|	Exp or Term
	|	Exp.id | Exp.id (ArgList)
	|	super.id (ArgList)
	|	Exp instanceof ClassType
Term	: :=	Term * Fact | Term / Fact
	|	Term and Fact | Fact
Fact	: :=	Fact = Basic | Fact < Basic | Basic
Basic	: :=	not Exp | num | id | true | false | nil
	|	new ClassType
	|	(Exp)
ArgList	: :=	ε | Args
Args	: :=	Exp | Args , Exp

Priorité des opérateurs : Term, Fact, Basic