`sem`

Application des techniques de sémantique dénotationnelle à l'implantation de l'évaluateur d'un langage fonctionnel et impératif (ouf)

Pascal Manoury

Février 2007

Brève description du «langage source»

Le «langage source» est le noyau d'un langage de programmation incluant des traits fonctionnels et impératifs.

Un programme est constitué d'une suite de déclarations suivi d'une suite d'instructions.

Une déclaration introduit soit un symbole de constante, un symbole de variable, un symbole de fonction ou un symbole de procédure. Les fonctions et procédures récursives doient être explicitement déclarées comme telles.

Les instructions sont l'affectation, l'alternative, les boucles, les blocs (avec déclarations locales), une instruction d'échappement et un mécanisme d'exception. L'alternative unaire (sans `else`) est admise. Il existe cinq formes de boucles (voir grammaire ci-dessous).

Les expressions du langage suivent la syntaxe LISP. Les symboles fonctionnels prédéfinis sont tous en MAJUSCULE.

Le langage connaît les types des booléens, des entiers, des chaînes de caractères. Il autorise la construction de structures de listes à la manière de LISP (listes fonctionnelles). Le langage n'autorise pas la définition de types par l'utilisateur.

Le langage n'est pas typé.

Lexique

Symboles et mots réservés

```
, ; = ( ) [ ] { } := ..
Cst Var Fun FunRec Proc ProcRec
If Else Loop While Until For In
Break Try With Do Raise
Print Println
```

Autres unités lexicales

Alphabet : caractères affichables

Ensemble de caractères :

- `alpha` : caractères alphabétiques (majuscules et minuscules)
- `num` : caractères numériques décimaux

– `alphanum` : union des ci-dessus

Unités lexicales

– identificateurs : `alpha (alphanum|'?')*`
– entiers : `'-'? num+`
– chaînes de caractères : toute suite de caractères de l'alphabet incluse entre guillemets (") ; un guillemet à l'intérieur d'une chaîne doit être précédé du caractère d'échappement \ ; le caractère (d'échappement) \ à l'intérieur d'une chaîne de caractères doit être précédé de lui même.

Valeurs et fonctions prédéfinies

```
IF NIL CONS CAR CDR TRUE FALSE
PAIR? NIL?
AND OR NOT EQ? LE? LT? GE? GT?
ADD MUL SUB DIV
```

Grammaire

Les non terminaux sont notés en Petites Majuscules. les terminaux sont notés en caractères `typographiques`, les ensembles de terminaux sont `id` pour les identificateurs, `num` pour les constantes entières et `str` pour les chaînes de caractères.

```
Prog  ::= Decs Stats

Decs  ::=
        | Dec Decs

Dec   ::= Cst id = Exp ;
        | Var id ;
        | Fun ( id Ids ) = Exp ;
        | FunRec ( id Ids ) = Exp ;
        | Proc id ( Prms ) = Stat
        | ProcRec id ( Prms ) = Stat

Stat  ::= id ( Args ) ;
        | id := Exp ;
        | { Decs Stats } ;
        | If Exp Stat
        | If Exp Stat Else Stat

        | Loop Stat
        | Loop While Exp Stat
        | Loop Until Exp Stat
        | Loop For ( id In Rng ) Stat

        | Break ;

        | Try Stat With Catches
        | Raise id ;
```

```
Catches  ::=  id Do Stat
           |  id Do Stat Else Catches

Ids      ::=  id Ids

Prms     ::=  id
           |  id , Prms

Stats    ::=
           |  Stat Stats

Exp      ::=  num
           |  str
           |  id
           |  ( IF Exp Exp Exp )
           |  ( id Exps )

Exps     ::=  Exp
           |  Exp Exps

Rng      ::=  Exp
           |  [ Exp .. Exp ]

Args     ::=  Exp
           |  Exp , Exps
```

Remarque

Le traitement du mécanisme d'exception n'est pas entièrement achevé. Les exceptions ne sont pas utilisables dans les expressions. De plus, l'environnement des continuations (les *catches*) associées à un *try* ne passe pas aux procédures. L'exemple ci-dessous échoue :

```
eleph@oxygene:~/LMD/CL/Prog/src$ cat ex16.src
Proc ex(x) = {
  If (EQ? x 0)
    Raise EXCEPTION;
  Else
    Println("Normal");
};

Println("Finit mal:");
Try ex(0);
With EXCEPTION Do Println("Exception");
eleph@oxygene:~/LMD/CL/Prog/src$ ./sem ex16.src
Finit mal:
Fatal error: exception Failure("eval fatal: <: "EGET: NOT FOUND"")
```