Petitのインタプリタの実装

氏名：大久保雄飛
学生番号：2014SE081
所要時間：約2日

1 Petit

Petit[1]は、コンピュータサイエンス入門(論理とプログラム意味論)において、プログラミング言語の意味論の具体例として挙げられているプログラミング言語で、構文規則が少なく、非常に簡素なプログラミング言語である。

1.1 構文領域と構文規則

構文領域とは記号の列からなる集合である。Petitの構文領域は以下の三つからなる。

ξ : **Var**　プログラム内の変数記号全体
ϵ : **Exp**　式の全体
τ : **Pro**　プログラムの全体

ここで、ξ ϵ τ はその構文領域上のメタ変数である。構文領域は、その構文に対して構文規則を与えることで、具体的に定義できる。Petitの構文規則はBNFで以下のように与えられる。

(i)　ξ ::= A | B | ... | Z
(ii)　ϵ ::= 0 | ξ | suc ϵ
(iii)　τ ::= ξ:=ϵ | τ;τ | for ϵ times do τ end

1.2 意味領域と意味関数

与えられた構文領域に対して、適切な意味を与えるために、数学的な領域と対応があると、プログラムの具体的な構造や仕組みを考えず、抽象的に扱えるので都合が良い。意味領域とは、それを目的とした数学的領域のことである。そのために、Petitでは、以下の三つの意味領域を考える。

ν : **N**　0以上の自然数の全体
σ : **S = (Var → N)**　コンピュータの状態全体
θ : **C = (S → S)**　コンピュータの状態の変換全体

ここで、ν σ θ はその意味領域上のメタ変数である。次に、構文領域から意味領域への意味関数を定義する。

$\mathcal{E}$: **Exp → (S → N)**
$\mathcal{C}$: **Pro → C**

1.2.1 意味関数 $\mathcal{E}$ の定義

$$\begin{aligned}\mathcal{E}[\![l]\!](\sigma) &= 0\\ \mathcal{E}[\![\xi]\!](\sigma) &= \sigma(\xi)\\ \mathcal{E}[\![\int\sqcap\rfloor\epsilon]\!](\sigma) &= \mathcal{E}[\![\epsilon]\!](\sigma)+\infty\end{aligned}$$

1.2.2 意味関数 $\mathcal{C}$ の定義

$$\begin{aligned}\mathcal{C}[\![\xi := \epsilon]\!](\sigma) &= update(\xi, \mathcal{E}[\![\epsilon]\!](\sigma))(\sigma)\\ \mathcal{C}[\![\tau_l;\tau_\infty]\!](\sigma) &= \mathcal{C}[\![\tau_\infty]\!](\mathcal{C}[\![\tau_l]\!](\sigma))\\ \mathcal{C}[\![\{\wr\nabla\epsilon\sqcup\rangle\Updownarrow\rceil\int\lceil\wr\tau\rceil\backslash\lceil]\!](\sigma) &= iterate(\mathcal{C}[\![\tau]\!], \mathcal{E}[\![\epsilon]\!](\sigma))(\sigma)\end{aligned}$$

ここで、関数 $update$ と $iterate$ を次のように定義する

$$update : \mathbf{Var}\times\mathbf{N}\to\mathbf{C}$$
$$update(\xi,\nu)(\sigma) = \sigma\{\xi:\nu\}$$

ただし、

$$\sigma\{\xi:\nu\}(\eta) = \begin{cases}\nu & if\ \eta=\xi\\ \sigma(\eta) & if\ \eta\neq\xi\end{cases}$$

$$iterate : \mathbf{C}\times\mathbf{N}\to\mathbf{C}$$
$$iterate(\theta,\nu) = \underbrace{\theta\circ\theta\circ ...\circ\theta}_{\nu回}$$

ここで、$\circ$ は関数の合成を表す。

2 実装

本章では Petit の実装について説明する。Petit のプログラムを評価するためには、前提として、字句解析器と構文解析器が必要であったので、与えられた LBNF から両解析器を生成する BNFC[2] を用いた。LBNF は Labelled BNF の略称で、BNF の拡張であり、BNF で定義される各構文規則に任意のラベルを与え、各構文規則をラベルのみで識別可能にする。LBNF を用いて Petit の構文規則を書き、以下のようになった。

```
Ass . Stm ::= Ident ":=" Exp ;
Con . Stm ::= Stm ";" Stm ;
For . Stm ::= "for" Exp "times" "do" Stm "end" ;

Zer . Exp ::= "0" ;
Var . Exp ::= Ident ;
Suc . Exp ::= "suc" Exp ;
```

ここで、ASS Con For などがラベルである。ただし、例外として、**Var** は Capital Letter に限定されておらず、BNFC に定義済みの修飾子 **Ident** を用いた。Petit の実装にはプログラミング言語 Ocaml を利用するので、BNFC を用いて、Ocaml 用の両解析器と、Petit の構文を Ocaml 上で抽象的に扱うための型を生成した。以下はそのデータ構造の定義である。

```
type ident = Ident of string
and stm =
    Ass of ident * exp
  | Con of stm * stm
  | For of exp * stm

and exp =
    Zer
  | Var of ident
  | Suc of exp
```

構文規則 (i) と Ident が対応し、構文規則 (ii) と対応するのが Zer Var Suc であり、構文規則 (iii) と対応するのが Ass Con For である。また、コンピュータの状態全体 **S** と対応するデータ型として、以下のデータ型を定義した。

```
type store = (ident * int) list
```

意味関数 $\mathcal{E}$ と $\mathcal{C}$ に対応する Ocaml の関数として、それぞれ evalExp と evalStm を定義した。また、補助関数 *update* と *iterate* に対応する Ocaml の関数として、それぞれ update と iterate を定義した。 以下はそれら四つの関数定義である。

```
let rec evalExp (s : store) : exp -> int =
  function
    | Zer  -> 0
    | Var id -> assoc id s
    | Suc e -> (evalExp s e) + 1


let rec evalStm (s : store) : stm -> store =
  function
    | Ass (id, e) -> update (id, evalExp s e) s
    | Con (st1, st2) -> let s1 = evalStm s st1 in
                        evalStm s1 st2
    | For (e, st) -> let n = evalExp s e in
                     iterate s st n

and iterate (s : store) (st : stm) : int -> store =
  function
    | 0 -> s
    | n -> iterate (evalStm s st) st (n-1)

let rec update (x, vx) : store -> store =
  function
    | [] -> [(x, vx)]
    | (y, vy) :: ys -> if x = y
                       then (y, vx) :: ys
                       else (y, vy) :: update (x, vx) ys
```

3 Petit プログラムの評価

前章で定義した Ocaml 関数と、BNFC で生成された字句解析器と構文解析を用いて、Petit のプログラムを評価するインタプリタを作成した。以下のプログラムは変数 a1 と a2 にそれぞれ 6 と 2 を代入し、a1+a2 を計算し、計算結果を変数 r に代入するプログラムである。

```
a1 := suc suc suc suc suc suc 0;
a2 := suc suc 0;

r := a1;
for a2 times do
  r := suc r
end
```

作成したインタプリタによって、上記のプログラムを評価した結果を以下に示す。

```
[Result]
a1 : 6, a2 : 2, r : 8,
```

参考文献

[1] https://www.iwanami.co.jp/.BOOKS/00/9/0061900.html
コンピュータサイエンス入門 (論理とプログラム意味論) P7-14

[2] https://bnfc.readthedocs.io/en/latest/
BNF Converter