プログラミング言語 Petit のインタプリタの実装

氏名：南山太郎
学生番号：20XXSE001

1 Petit の定義

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，例として挙げられているプログラミング言語である．Petit は，構文規則が少なく，意味関数の定義が簡潔である．

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

$\xi : \mathbf{Var}$　プログラム内の変数記号全体
$\epsilon : \mathbf{Exp}$　式の全体
$\tau : \mathbf{Pro}$　プログラムの全体

の 3 つからなる．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

$$\xi ::= A \mid B \mid \cdots \mid Z \tag{1}$$
$$\epsilon ::= 0 \mid \xi \mid suc\ \epsilon \tag{2}$$
$$\tau ::= \xi := \epsilon \mid \tau;\ \tau \mid \mathsf{for}\ \epsilon\ \mathsf{times\ do}\ \tau\ \mathsf{end} \tag{3}$$

と与えられる．

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．Petit では，3 つの意味領域

$\nu : \mathbf{N}$	0 以上の自然数の全体
$\sigma : \mathbf{S} = (\mathbf{Var} \to \mathbf{N})$	コンピュータの状態全体
$\theta : \mathbf{C} = (\mathbf{S} \to \mathbf{S})$	コンピュータの状態の変換全体

を考える．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

$$\mathcal{E} : \mathbf{Exp} \to (\mathbf{S} \to \mathbf{N})$$
$$\mathcal{C} : \mathbf{Pro} \to \mathbf{C}$$

を定義する．

1.2.1 意味関数 $\mathcal{E}$ の定義

式の評価は以下のように定義される：

$$\mathcal{E}[\![0]\!](\sigma) = 0$$
$$\mathcal{E}[\![\xi]\!](\sigma) = \sigma(\xi)$$
$$\mathcal{E}[\![suc\ \epsilon]\!](\sigma) = \mathcal{E}[\![\epsilon]\!](\sigma) + 1$$

1.2.2 意味関数 $\mathcal{C}$ の定義

文の実行は以下のように定義される：

$$\mathcal{C}[\![\xi := \epsilon]\!](\sigma) = update(\xi, \mathcal{E}[\![\epsilon]\!](\sigma))(\sigma)$$
$$\mathcal{C}[\![\tau_0;\ \tau_1]\!](\sigma) = \mathcal{C}[\![\tau_1]\!](\mathcal{C}[\![\tau_0]\!](\sigma))$$
$$\mathcal{C}[\![\mathsf{for}\ \epsilon\ \mathsf{times\ do}\ \tau\ \mathsf{end}]\!](\sigma) = iterate(\mathcal{C}[\![\tau]\!], \mathcal{E}[\![\epsilon]\!](\sigma))(\sigma)$$

ここで，補助関数 $update$ は

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

と定義される．ただし，

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

である．補助関数 $iterate$ は

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

と定義される．ここで，$\circ$ は関数の合成を表す．

```
Asn. 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 ;
```

図 1: Petit の構文規則（ファイル Petit.cf）

```
(* OCaml module generated by the BNF converter *)



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

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

図 2: Petit の構文領域（ファイル AbsPetit.ml）

2 Petit の実装

本章では Petit の実装について説明する．Petit のプログラムを実行するためには，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 を用いた Petit の構文規則は図 1 の通りである．ここで，`ASS`, `Con`, `For` などがラベルである．ただし，例外として，`Var` は大文字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．生成された型を含むファイルの中身を図 2 に示す．

構文規則 1 と `Ident` が対応し，構文規則 2 と対応す

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

図 3: Petit の意味領域

```
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
```

図 4: Petit の意味関数

るのが `Zer`, `Var`, `Suc` であり，構文規則 3 と対応するのが `Asn`, `Con`, `For` である．また，コンピュータの状態全体 **S** と対応する意味領域の型 `store` を図 3 のように定義した．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．これら 4 つの意味関数を図 4 に示す．

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

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

図 5: Petit のプログラム例

3　Petit プログラムの実行

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器を用いて，Petit のプログラムを実行するインタプリタを作成した．図 5 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 と a2 の和を計算し，計算結果を変数 r に保存するプログラムである．

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] 田辺　誠，中島玲二，長谷川真人：コンピュータサイエンス入門: 論理とプログラム意味論，岩波書店 (1999).

[2] The BNF Converter (online), available from 〈https://bnfc.digitalgrammars.com/〉 (accessed 2020-03-23).