パラメータ付き０Ｌシステムプログラミング言語の仕様と実装*

西田 泰伸†
富山県立大学工学部、 939-0398 富山県小杉町黒河 5180

1 はじめに

L システムは A. Lindenmayer が提唱して以来、形式言語理論に加え、自然や人工物の形の記述モデルとして多数の研究者を引きつけてきた [2]。その中でもパラメータ付き 0L システムは簡単な規則ですぐれた記述能力を持つため、さまざまな「形」のシュミレータに応用されている。[4, 1, 6, 3]

パラメータ付き 0L システムの規則は次の形式をしている。

$$\langle \text{頭部} \rangle : \langle \text{条件部} \rangle \rightarrow \langle \text{置換部} \rangle$$

ここで、〈頭部〉は 0 以上の仮パラメータの付いた記号、〈条件部〉は仮パラメータと定数からなる式、〈置換部〉は

$$\langle \text{記号}(\text{式}_1, \ldots, \text{式}_k) \rangle$$

の一般形をした仮パラメータ付き記号の有限の長さの並びである。「式」は普通の算術式とし、仮パラメータや定数は実数値をとる。実数値のパラメータ（実パラメータ）を持つ記号をモジュールという。特定のモジュールの列を初期列として与える。現在のモジュール列のそれぞれのモジュールについて、

1. 頭部の記号とパラメータの数が等しい
2. 仮パラメータに対応するモジュールの実パラメータを代入したとき、条件部が 0 以外の値を持つ

となる規則を選び、そのモジュールを選ばれた規則の（式の仮パラメータに対応する実パラメータを代入して計算した）置換部で置き換える。条件を満たす規則がないときは以前のモジュールは変化しない。このようにして、パラメータ付き 0L システムは初期列から次々とモジュール列を導出する。

この規則をシュミレータによって模倣するのではなく、そのままプログラミング言語として実行可能なコードとみなすアイデアが提案されている [5]。本研究ではこの発想を具体化した言語仕様（パラメータ 0L システムプログラミング言語、PLL）を作り、C 言語へのトランスレータを作ることにより実装した。ここでは、言語仕様と実装の概略について報告する。

*A Specification and an Implementation of a Parameterized 0L System Programming Language

†Taishin Y. Nishida, Toyama Prefectural University Kosugi-machi, Toyama 939–0398, Email: nishida@pu-toyama.ac.jp

2 パラメータ付き 0L システムプログラミング言語 (PLL) の仕様

文献 [5] では、パラメータ付き 0L システムプログラミング言語 (PLL) を C 言語を拡張する方式で提案している。ここでもその考え方を踏襲し、C 言語に PLL 特有の文を追加する。つぎに、追加する主な PLL 特有の文の構文を拡張 BNF 記法（*- 閉包も使う）で記述する。

1. ⟨RULE⟩ ::= PLMODULE ⟨symbol⟩ (double ⟨token⟩ [, double ⟨token⟩]*) ⟨RULE BODY⟩
2. ⟨RULE BODY⟩ ::= { [C_statement | ⟨PRODUCE⟩]*}
3. ⟨PRODUCE⟩ ::= produce ⟨SUCCESSOR⟩ ;
4. ⟨SUCCESSOR⟩ ::= ⟨ATOM⟩ | ⟨ATOM⟩, ⟨SUCCESSOR⟩
5. ⟨ATOM⟩ ::= ⟨symbol⟩[(C_expression [, C_expression]*)]
6. ⟨SETAXIOM⟩ ::= setaxiom ⟨SUCCESSOR⟩ ;
7. ⟨APPLY PRODUCTION⟩ ::= apply_production ;

ここで、⟨symbol⟩ と ⟨token⟩ は通常の C の識別子である。C_statement と C_expression はそれぞれ C で許される文と式である。この拡張 BNF の 1 から 5 は頭部が同じ記号を持つ規則をまとめて記述する。条件部は C の条件文、`if— else` と `switch` で表現され、置換部は produce 文で表現される。setaxiom と apply_production はそれぞれ初期列の設定と導出を行う。

3 グラフィック表現

前節の仕様はモジュール列の導出を定める。ある用途にはこの仕様で十分である。例えば、次のプログラムは

```
PLMODULE Fib(double n1, double n){
  produce Fib(n, n1 + n) ;
}
void main(void){
  setaxiom Fib(1.0, 1.0), Fib(1.0, 1.5) ;
  for(;;){
```

```
    apply_production ;
  }
}
```

ふたつのフィボナッチ数列 (1, 1, 2, 3, 5, 8, ...) と (1, 1.5, 2.5, 4, 6.5, 10.5, ...) を同時に計算し続ける。

しかしながら、パラメータ付き 0L システムは主に形の記述モデルとして用いられるため、モジュール列を図形として解釈する必要がある。われわれは文献 [4] のタートルグラフィック解釈を採用する。そこでは記号 |, $, [,], {, }, ., ~, %, !, +, −, ^, &, \, /, ', f, F, G はタートル命令となり[1]、PLL の規則では書き換えられない。PLL で書き換えられず（C で識別子として許されない記号を使う）ため、FINAL_PLMODULE 宣言を導入した。次のプログラムはタートルグラフィック解釈により、図 1 で示される樹木状の形を導出する。

```
FINAL_PLMODULE +
FINAL_PLMODULE −
FINAL_PLMODULE [
FINAL_PLMODULE ]
FINAL_PLMODULE F

PLMODULE make_branch(double x, double y){
  if(x >= 0.1){
    produce F(x),
      [, +(10.0), make_branch(x * 0.9, y * 0.707), ],
      [, −(25.0), make_branch(x * 0.7, y * 0.707), ];
  }else{
    produce F(x);
  }
}
void main(void){
  int i;
  setaxiom make_branch(1.0,1.0);
  for(i=0; i<10; i++){
    apply_production;
  }
}
```

4 実装

PLL の実装は PLL 特有の文を C の関数群に変換するトランスレータと、導出・図形表示などを行うライブラリからなる。トランスレータによる変換の例を二つ示す。

setaxiom make_branch(1.0, 1.0); ⇒ pll_initialize1();
append(2, &make_branch, Hash(make_branch), 1.0, 1.0);
pll_initialize2();

apply_production; ⇒ apply_production();

[1] 例えば F(x) は長さ x の線を描く、+(θ) はタートルの向きを右に θ （度）回す、[と] は枝の始まりと終わりを示すなど

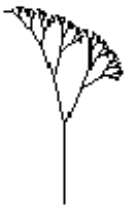

Figure 1: PLL によって導出される樹木状の形

ここで、pll_initialize1() と pll_initialize2() はモジュール列を記憶する領域の確保やグラフィックの初期化をする関数、apply_production() は一回の導出をする関数である。append は PLL の規則に従って次のモジュールを作り出す関数であり、プログラマが定義した（トランスレータにより変換されている）規則へのポインタ &make_branch と、記号名のハッシュ値 Hash(make_branch) を、実パラメータと共に引数としてとる。これらの関数と、グラフィック表示用の関数がライブラリで実装されている。

PLL はコンパイラであるから従来の L システムシュミレータより高速であり、L システムを使ったモデル化に有用であると考えられる。この実装はフリーソフトウェアとして、ソースプログラムを公開する予定である。

References

[1] J. Hanan (1997), Virtual plants — integrating architectural and physiological models, *Environmental Modelling & Software,* 12:35–42.

[2] 西田 泰伸 (2001), L システム，歴史と展望, 信学論 D-I, **J84-D-I**, pp. 18–30.

[3] Y. Parish and P. Müller (2001), Procedural modeling of cities, ACM SIGGRAPH 2001, Los Angeles CA USA 12–17 Aug., pp. 301–308.

[4] P. Prusinkiewicz and A. Lindenmayer (1990), *The Algorithmic Beauty of Plants,* Springer-Verlag, Berlin.

[5] P. Prusinkiewicz and J. Hanan (1992), L-systems: from formalism to programming languages, in: G. Rozenberg and A. Salomaa (eds), *Lindenmayer Systems,* Springer-Verlag, Berlin, pp. 193–211.

[6] P. Prusinkiewicz (2000), Simulation modeling of plants and plant ecosystems, *CACM* 43:85–93.