ショートノート

関数型プログラムによる属性評価を行う教育用属性文法インタプリタ†

桃内佳雄†† 岡田哲彦††* 宮本衛市††

プログラミング手法と言語処理系について，直接的かつ多面的に学習することのできる教育用システムとして，属性文法インタプリタを構成した．本システムを用いて，たとえば，次のような項目についての基本的な学習を行うことができる．（1）属性文法による非手続き的プログラミング，（2）LISPKIT-LISP-B による関数型プログラミング，（3）コンパイラなどの言語処理系の基本的な構造．

1. まえがき

プログラミング手法および言語処理系の諸側面の基本を学習することのできるコンパクトなシステムの構築は，計算機科学を学ぶ学生の教育にとって有用である．われわれは，複数のプログラミング手法と言語処理系の基本について，直接的かつ多面的に学習することのできる教育用システムとして，属性文法インタプリタを構成した．属性文法は，プログラム言語の形式的定義，プログラミング方法論，コンパイラ生成系，および非手続き型プログラミングのための計算モデルなどのために広く用いられている[1)~4)]．この属性文法を，教育用というもう一つの視点から見直してみると，「そのさまざまな可能性のゆえに，プログラミング手法や言語処理系の教育用システムを構築するための柔軟な枠組を与えている」という属性文法のもつ可能性を見いだすことができる．本システムはそのような可能性を実現しようとする一つの試みである．

2. システムの設計方針とその実現方法

設計方針とその実現方法について以下に述べる．

Ⅰ．複数のプログラミング手法を学習できること．

属性の評価規則を Henderson の LISPKIT-LISP[5)] に基づく言語 (LIPKIT-LISP-B) による関数型プログラムとして記述する．これにより，属性文法による非手続き的プログラミング，および LISPKIT-LISP-B による関数型プログラミングの基本について直接的に学習することができる．また，システム自体は手続き的プログラム言語 PASCAL により記述した．

Ⅱ．言語処理系の形式的な定義および基本的な構造について学習できること．

これは，属性文法のもつ，コンパイラなどの言語処理系を形式的に記述する能力における可能性に依存している．実際にわれわれは，Wirth による PL/0 コンパイラ[6)] を含め，種々の言語処理系を本システムでの属性文法による非手続き的プログラムにより記述し，それを本システム（属性文法インタプリタ）に入力することによって，PL/0 コンパイラなどの言語処理系の構成を行っている．

Ⅲ．システムが使いやすいものであること．

属性文法は，基本的な教育用ということで，記述能力は多少落ちても処理のしやすさということを考えて，L-属性文法に制限し，その記述方法も，3章でその詳細を述べるが，簡潔で理解しやすい方法を考案した．さらに，システムの処理過程の追跡のために，3種類の情報出力機能をもたせた．それらは，入力された属性文法の文法グラフの出力，入力文の構文解析過程のトレース，および構文解析・属性評価過程におけるスタック状態の出力である．3章で構文解析過程のトレースの例を示す．

Ⅳ．システム自体の理解からもプログラミング手法や言語処理系について学習できること．

プログラミングの学習にとって，プログラムを読むことも一つの有効な方法である．本システムは，PASCAL で約1,400行のプログラムとして簡潔に構造化されて作成されている．また，LISPKIT-LISP-B による評価規則のプログラムは，システムに内蔵の LISP コンパイラにより，同じく内蔵の仮想機械

† An Attribute Grammar Interpreter for Programming Education by YOSHIO MOMOUCHI, TETSUHIKO OKADA and EIICHI MIYAMOTO (Division of Information Engineering Graduate School of Engineering, Hokkaido University).

†† 北海道大学大学院工学研究科情報工学専攻
* 現在 (株)日立製作所中央研究所

SECD マシンのコードに翻訳され，SECD マシン上で実行される．LISP コンパイラと SECD マシンは文献5) に基づいており，文献5) を参考書としてそれらについても学習することができる．

以上の記述中で触れたことのほかに，本システムは，①属性文法に基づく処理は構文解析と並行して属性評価を行う下向き構文解析法による，②構文解析は後戻り処理を行い，左再帰性の簡単な処理方略を含む，③属性文法の入力時と入力文の構文解析時に簡単なエラー回復処理を行う，などの特徴をもつ．

3. システムの概要

3.1 システムの構成

インタプリタは**図1**に示すように，大別して五つのモジュールから構成される．文法入力部は入力される属性文法をその内部表現である文法グラフに変換する．LISP コンパイラは LISPKIT-LISP-B により記述されている属性文法の評価規則をインタプリタに内蔵の仮想機械 SECD マシンのコードに翻訳する．字句解析部は入力文を字句の列として認識し，リスト構造に変換する．構文解析・属性評価部は文法入力部が作成した文法グラフに従って，字句解析された入力文の構文解析と SECD マシンによる属性評価を行い，その結果を出力する．

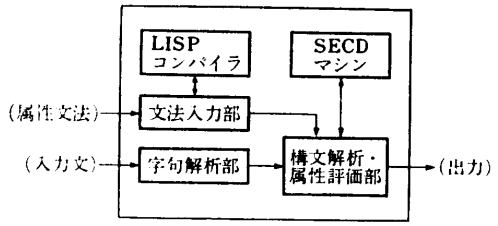

図 1 システムの構成
Fig. 1 System organization.

```
ATTRIBUTES[
<START>          CODE
<EXP>            CODE
<EXP_REC>        CODE
<TERM>           CODE
<TERM_REC>       CODE
<FACTOR>         CODE    ]

(SET (NULL   .(LAMBDA(X)(EQ X NIL)))
     (REV    .(LAMBDA(X Y)(IF (NULL X) Y
                (REV (CDR X) (CONS (CAR X) Y)) )))
     (APP    .(LAMBDA(X Y)(REV (REV X NIL) Y)))
     (APP_3 .(LAMBDA(X Y Z)(APP X (APP Y Z))))      )

<START>    ::= <EXP>
   [ 0 (AS CODE_0 CODE_1) ]
<EXP>      ::= <TERM> <EXP_REC>
   [ 0 (AS CODE_0 (APP CODE_1 CODE_2)) ]
<EXP_REC>  ::= '+' <TERM> <EXP_REC>
   [ 0 (AS CODE_0 (APP_3 CODE_1 '(ADD) CODE_2)) ]
             ! '-' <TERM> <EXP_REC>
   [ 0 (AS CODE_0 (APP_3 CODE_1 '(SUB) CODE_2)) ]
             !
   [ 0 (AS CODE_0 NIL) ]
<TERM>     ::= <FACTOR> <TERM_REC>
   [ 0 (AS CODE_0 (APP CODE_1 CODE_2)) ]
<TERM_REC> ::= '*' <FACTOR> <TERM_REC>
   [ 0 (AS CODE_0 (APP_3 CODE_1 '(MUL) CODE_2)) ]
             ! '/' <FACTOR> <TERM_REC>
   [ 0 (AS CODE_0 (APP_3 CODE_1 '(DIV) CODE_2)) ]
             !
   [ 0 (AS CODE_0 NIL ]
<FACTOR>   ::= '(' <EXP> ')'
   [ 0 (AS CODE_0 CODE_1) ]
             ! <NAME>
   [ 0 (AS CODE_0 (CONS NAME_1 NIL)) ]
             ! <NUMBER>
   [ 0 (AS CODE_0 (CONS VALUE_1 NIL)) ]
#
```

図 2 属性文法の例
Fig. 2 An example of attribute grammar.

インタプリタの実行においては，まず属性文法を入力して処理系を構成した後に，その処理系によって処理されるべき入力文を入力する．

3.2 属性文法

属性文法は，属性宣言部と関数定義部と文法規則部とで構成する．**図2**に算術式を逆ポーランド記法に変換する処理を記述する属性文法の例を示す．

属性宣言部では，ATTRIBUTES に続けて“[”と“]”のなかに文法規則で用いる非終端記号とその属性を宣言する．特別な非終端記号として，開始記号〈START〉と文字列を表す〈NAME〉と自然数を表す〈NUMBER〉がある．〈NAME〉と〈NUMBER〉は，それぞれ文字列を値とする NAME と自然数を値とする VALUE という属性とともに暗黙のうちに宣言されているものとして処理される．関数定義部は，属性宣言部の後に位置し，“(SET”と“)”で括られたなかに属性の評価規則のなかで用いる関数をラムダ記法を用いて定義する．定義せずに用いることのできる組込み関数として，CAR, CDR, CONS, ATOM, EQ, IF, AS(SIGN) など，19 個を用意している．文法規則は，書換え規則とそれに付随する評価規則の組を単位として構成する．書換え規則は BNF 記法によって，評価規則は LISPKIT-LISP-B によって記述する．たとえば，図2の中の次の文法規則を例としてその記述方法を説明する．

〈TERM〉::=〈FACTOR〉〈TERM-REC〉
[0 (AS CODE-0 (APP CODE-1 CODE-2))]

第1行目が書換え規則で，第2行目が評価規則である．書換え規則の非終端記号(“〈”と“〉”で括られている）には左から，0, 1, 2, …と番号が仮定され，また，評価規則の評価位置として，::=の直後から各記号の間に，1, 2, … と番号が仮定され，最後尾の位置を0とする．上例では次のようになる．

(非終端記号番号：0　1　2)
〈TERM〉::=〈FACTOR〉〈TERM-REC〉
(評価位置番号：　1　2　0)

評価規則はこの番号を用いて記述する．上例では，評価位置 0 で，右辺の〈FACTOR〉の属性 CODE

```
*** START LPAG ***
DO YOU WANT SYNTAX GRAPH ? (Y OR N)
?N
DO YOU WANT TO TRACE PARSING ? (Y OR N)
?Y
DO YOU WANT TO TRACE STACK ? (Y OR N)
?N
------  INPUT DATA  ------
       X*2+Y/3 QUIT
------  START PARSING  ------
  <EXP>
    <TERM>
      <FACTOR>
        <NAME>
          !-------------------------  X
      <TERM_REC>
        !-------------------------  *
        <FACTOR>
          <NAME>
          *** FAILURE IN <NAME>
          <NUMBER>
            !-------------------------  2
        <TERM_REC>
    <EXP_REC>
      !-------------------------  +
      <TERM>
        <FACTOR>
          <NAME>
            !-------------------------  Y
        <TERM_REC>
          !-------------------------  /
          <FACTOR>
            <NAME>
            *** FAILURE IN <NAME>
            <NUMBER>
              !-------------------------  3
          <TERM_REC>
      <EXP_REC>
<<< SUCCEEDED IN PARSING >>>
( ( X 2 MUL Y 3 DIV ADD ) )
*** END LPAG ***
END
```

図 3 構文解析過程のトレースを含む実行例
Fig. 3 A sample run with the trace of parsing.

〈CODE-1〉と〈TERM-REC〉の属性 CODE (CODE-2) とを逆付加し (APP), 左辺の〈TERM〉の属性 CODE (CODE-0) に代入する (AS) ことを表している. 各非終端記号の属性を CODE-1 の "-1" のように非終端記号番号を付加することにより区別する. また, 評価規則はL-属性文法における制限に従って書かれなければならない.

3.3 実 行 例

図2の属性文法を入力して処理系を構成した後に, 入力文 "X*2+Y/3" を入力し, その逆ポーランド記法を出力している, 構文解析過程のトレースを含む実行例を図3に示す. "***FAILURE…" は後戻り処理を示す.

4. む す び

本システムは, 簡単な言語処理プログラムの模型実験のためにも用いることができ, PL/0 コンパイラの構成とその改良を通して, その有効性を確かめている[7].

参 考 文 献

1) Watt, D. A.: An Extended Attribute Grammar for PASCAL, *SIGPLAN Notices*, Vol. 14, No. 2, pp. 60-74 (1979).
2) Noonan, R. E.: Structured Programming and Formal Specification, *IEEE Trans. Softw. Eng.*, Vol. SE-1, No. 4, pp. 421-425 (1975).
3) 佐々政孝: コンパイラ生成系, 情報処理, Vol. 23, No. 9, pp. 802-817 (1982).
4) 片山卓也: 属性文法計算モデル, 情報処理, Vol. 24, No. 2, pp. 147-155 (1983).
5) Henderson, P.: *Functional Programming—Application and Implementation*, Prentice Hall, Englewood Cliffs (1980).
6) Wirth, N.(片山卓也訳): アルゴリズム+データ構造=プログラム, 日本コンピュータ協会, 東京 (1979).
7) 岡田哲彦: 属性文法による言語処理のためのソフトウェアツール (LPAG) の開発, 北大大学院工学研究科情報工学専攻修士論文 (1983).

(昭和58年9月8日受付)
(昭和58年12月13日採録)