意味情報を利用したLL(1)構文解析の一手法*

4Q−1

金谷英信　中川裕之　中田育男
筑波大学

1 はじめに

文脈自由文法に意味規則を付加した属性文法は、コンパイラの仕様記述に向いており、コンパイラ生成系への応用が広く研究されてきた。属性文法を用いたコンパイラの記述では、その記述量などが問題となったが、属性文法の生成規則に正規右辺文法を用いる正規右辺属性文法が提案され、記述法の改善が試みられている。

正規右辺属性文法では、正規表現を使用することにより、構文規則の記述がコンパクトになり、記述の読解性が良い。また文法がELL(1)文法である場合、効率の良い再帰降下型の構文解析器をつくることができる[1]。

しかし、LL(1)文法は決定性文法であり、左から右に下降型解析しながらただ1つの先読み記号で正しい選択肢を選べなくてはならない。一般のプログラミング言語の多くはこの条件に抵触する部分があるので、再帰降下型の構文解析をするためには、構文規則の書換えを必要とし、読解性を非常に悪くする。

この問題の多くは、意味情報を用いて解析を行なえば構文解析の書換えをせずに解決できる。正規右辺文法で属性文法をわかりやすい表現にする為に、文法記号（非終端記号、または終端記号）に附属して属性情報や意味規則を記述することを我々は考えている[4]。さらに、その記述を構文解析にも利用することを考えた。

そこで、我々は正規右辺属性文法に意味情報を反映する属性を付加することを提案し、その属性を用いて構文解析する（attribute-directed-parsing）構文解析器を生成する生成系の試作を行なっている。意味規則はLL構文解析と親和性の高いL属性文法を基本とし、右から左へ属性値が決定するようなL属性でないものも扱える1パス型属性文法[4]用いている。

本論文では、属性条件指示の記述方法と、その文法記述をもとにattribute-directed-parsingを行なう構文解析器の生成、解析処理方法について述べる。

2 構文規則への属性条件の付加

ステートメントが手続き呼びだしか、代入文かの文法記述を行なう場合、直観的には次のような記述を行なう。

statement ::= assign | call
assign ::= ident ':=' expression
call ::= ident proc-params

これは、非終端記号assign、callの開始記号集合がいずれも{ident}でありLL(1)でない。このような文法は、一般に次のように左括りだしの技法を用いて書換えを行なう[2]。

statement ::= ident(rest-assign | rest-call)
rest-assign ::= ':=' expression
rest-call ::= proc-params

このような書き換えは、文法設計者の意図する文法構造を崩すものであり、また、読解性が極端に悪くなる。この記述をrule-splittingによるattribute-directed-parsing[3]ができるように記述すると文法の書換えが不要になる。

*A method of ELL(1) parsing with semantic analysis effects
Hidenobu KANAYA, Hiroyuki NAKAGAWA, Ikuo NAKATA
(Univ. of Tsukuba)

statement → assign **where** is-var(inh(statement),syn*(assign))
statement → call **where** is-proc(inh(statement),syn*(call))
.....

しかし、このような記述はどこでattribute-directed-parsingによる分岐が必要なのかを考慮しなくてはならない。できればLL文法を意識せずに、設計者が考えた通りにその文法を宣言的に記述できる方がよい。本生成系では、構文規則とは別に分岐条件を記述するのではなく、文法規則に属性条件を付加する。文法記述中で、構文解析時に解析情報として用いる属性と属性値を下線部のように、終端記号のあとに<,>でくくって次のように記述する。

・終端記号 < 属性名 == 属性値 >

この表現を用いると次のようになる。

［文法R1］
statement ::= assign | call
assign ::= var-ident ':=' expression
call ::= call-ident proc-params
var-ident ::= ident< ↑*mode* == *var* >
call-ident ::= ident< ↑*mode* == *proc* >

3 本システムにおける評価方法

［文法R1］では、構文要素としては同じ識別子identが現れても< ↑*mode* == *var* >といった属性値を解析情報として用いながら構文解析をLL(1)のごとくに行なうことができる。属性値を解析情報として用いて構文解析する処理は、大まかに次のようになる。

- 解析情報に用いる属性の属性値を入手する
- 入手した属性値をもとに正しい分岐を行なう

3.1 完全宣言型言語の構文解析

構文解析時に解析に用いる属性値が必ず決定しているような完全宣言型の言語では、上記のような場合に問題なく属性値を用いて構文解析を行なえる。上記のような［文法R1］をもとに構文解析器を生成するには、解析情報として用いる属性の属性値を入手する手続きを意味処理として自動的に追加しなくてはならない。構文解析で用いる情報は記号表に格納されている場合がほとんどなので、本生成系で生成される構文解析器は属性値を記号表から入手するようにしている。

本生成系では基本的な意味関数を用意しており、それを用いて記号表などの記述を宣言的に行なえる[4]。本生成系で行なう属性値を用いた構文解析では、このようにして記述された記号表に格納される情報を取り出して用いるものとする。

前述の［文法R1］を例に考える。ここで非終端記号statementの生成規則の分岐は属性値をもとに行なわれる。非終端記号statementのdirector集合は次のようになる。

・［assign］ FIRST{ ident< ↑*mode* == *var* >}
・［call］ FIRST{ ident< ↑*mode* == *proc* >}

この情報をもとに非終端記号statementの生成規則から次のような構文解析器の一部が生成される。

```
tatement(){
 if (token == ident){
   att_val = get_mode("mode",token_stirings);
           /* 現在の先読みしている記号列をキーとして
              属性 mode の属性値を記号表から検索 */
   if (att_val==var){ assign(); }
   else if (att_val==proc){ call(); }
 } else { error(); }
}
```

3.2 完全宣言型でない言語の構文解析

言語によっては、関数の定義や宣言よりも前にその関数の呼びだし文を書くことを許すような、完全宣言型でない言語もある。その場合には前節で述べたような attribute-directed-parsing はできない。例えば、次のようなプログラムを考える。

```
main(){ .....
sub(); — (a)
.....}
sub(){ ......} — (b)
```

(a) の手続き呼びだしのステートメントを構文解析するときには、識別子 sub の属性値 *mode* はまだ決まっていない。手続き sub として定義され、識別子 sub の属性 *mode* の属性値が決定し、記号表に登録されるのは (b) まで解析が進まなくてはならない。

3.2.1 完全宣言型でない言語の場合の記述法

次のように、完全宣言型でない言語の場合にも対応する表現を導入する。

・終端記号 < 属性名 = 属性値 >

この表現は、その属性値が決まっていない場合もありうることを意味する。構文解析の際には、属性値が記号表に登録されている場合は属性値を使って構文解析を行ない、属性値が記号表に登録されていない場合は、この終端記号に続く終端記号（または非終端記号）の first 集合を見て構文解析を続行する。これを使って、例のようなプログラムを受理できるように［文法 R1］を修正すると次のようになる。

［文法 R2］

statement ::= assign | call

assign ::= var-ident ':=' expression

call ::= call-ident proc-params

var-ident ::= ident< ↑*mode* == *var* >

call-ident ::= ident< ↑*mode* = *proc* >

3.2.2 完全宣言型でない言語の場合の評価

構文解析の際には、次のように処理が進む。

- 属性値が決定しているときは、その値を用いて構文解析を行なう。
- 属性値が決定していないときは、1 トークン先読みし構文解析する。

属性値が決定しない場合の構文解析では 1 トークン先で予測解析を行なうので、その為の director 集合が必要になる。この集合を second 集合と呼ぶこととし、director 集合の収集の際に同時に集める。［文法記述 R2］での非終端記号 statement の director 集合は次のようになる。

・［ assign ］ FIRST{ ident< ↑*mode* == *var* >}

・［ call ］ FIRST{ ident< ↑*mode* = *proc* >}
SECOND { FIRST{proc-params} }

属性値が決定していない場合に記号表へ属性値の検索を行なうと、属性値として undef が返るので、「< 属性名 == 属性値 >」のような記述ではエラーとし、「< 属性名 = 属性値 >」のような記述では second 集合を用いて構文解析を行なえばよい。また、second 集合を用いて予測的構文解析を行なったときは、記述してある属性名、属性値、トークンの内容を記号表に格納しておけば、あとで属性値が決定したとき（例の (b) を構文解析したときなど）にチェックすることができる。本生成系では、記号表へのメンバー登録の際にこのように格納された情報と登録する内容との整合性を自動的にチェックし、second 集合を用いて予測的構文解析を行なったときの意味と定義部（または宣言部）での意味が合致しているかを調べている。これによって、予測的な構文解析を行なった際にも、1 パスでエラーチェックが可能となる。

［文法 R2］をもとに構文解析器の一部を生成すると次のようになる。

```
statement(){
  if (token == ident){
    att_val = get_mode("mode",token_strings);
            /* 現在の先読みしている記号列をキーとして
               属性 mode の属性値を記号表から検索 */
    if(att_val==var){ assign(); }
    else if(att_val==proc){ call(); }
    else if(att_val==undef){
     get_token();
     if(token==Second(call)){
         push_back_token();
         set_after_check(token_strings,"mode","proc");
                /* 記号表に記号列、属性名、属性値
                   をセット。後のチェックに利用する。 */
         call(); }
    } else {error(); }}
    else { error(); }
}
```

4 まとめ

正規右辺属性文法の中に attribute directed parsing が宣言的に記述できる表現を導入し、左括りだしによる構文規則の書き換えの大部分が不要となった。また、その表現を使った文法記述をもとに生成される構文解析器の処理方法を述べた。

意味処理の結果を反映する属性を正規右辺文法の中に簡潔な表現で埋め込み、その属性に条件付けをすることで、解りやすいコンパイラ記述とすることができた。また、その記述をもとに効率の良い 1 パスコンパイラを生成するコンパイラ生成系を設計した。属性値指示を文法記述の中に宣言的に記述することで文法設計者の意図する文法構造を崩すことがなくなり、生成される構文解析器は指示された属性値を用いて LL 構文解析が可能となった。完全宣言型でない言語を記述する場合にも同じような宣言的記述をもとに、属性値が決定していれば attribute-directed-parsing を行ない、属性値が決まっていなければトークン先読みを行なって予測的構文解析を行なう構文解析器を生成することができた。構文解析器が予測的構文解析を行なった場合にトークン内容と記述された属性名、属性値を実際の属性値決定の際の整合性チェックに用いることで、1 パスの枠組のなかで完全宣言型でない言語の識別子の宣言と使用に関する意味的エラーを自動的に発見することができた。

今後は各種の手続き型言語の記述、処理系の生成を行ない、生成系の有効性、正当性の評価を行なっていきたい。

参考文献

[1] 丁亜希, 中田育男: 正規右辺文法の再帰降下パーサの効率のよい生成法, 情報処理学会論文誌 Vol.30,No.2,pp.197-203,1989.

[2] E.F.Elsworth, M.A.B.Parkes:Automated Compiler Construction based on Top-down Syntax Analysis and Attribute Evaluation,SIGPLAN NOTICES Vol.25,No.8,pp.37-42,1990.

[3] Watt.D.A:Rule Splitting and Attribute Directed Parsing,LNSC Vol.94,pp.363-392.1980.

[4] 中川裕之, 金谷英信, 中田育男: 拡張 1 パス型属性文法の提案, 本大会発表,1992.