拡張1パス型属性文法の提案*

4Q－2

中川裕之　金谷英信　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与した属性文法はコンパイラの仕様記述に向いており、また、比較的効率の良い属性評価器が得られることよりコンパイラ生成系への応用が広く研究されてきた。属性評価器の自動生成にあたっては効率良く、また、効率の良い属性評価器を生成することが問題とされてきたが、これはKnuthが提案した属性文法の部分クラスを定義することで解決が試みられ、その結果、実用的な属性評価器の自動生成系が得られるようになった。その一方で、属性文法の記述量が問題となり、記述量を少なくするために記述法を改善する試みがなされた。その1つとして、属性文法の生成規則に正規右辺文法を用い、更にその正規表現に対応した形で属性評価規則にも正規表現を許した正規右辺属性文法が提案された[2][3][1]。しかし、その問題点は右辺に不定個数現れる得る構文要素に対応する意味規則を独立に書くことが難しいという点である。

我々は正規右辺文法の中に簡潔な表現の意味規則を埋め込み、分かり易い記述で効率の良い1パスコンパイラを生成するコンパイラ生成系を考案した。構文規則は構文解析器との対応が容易なLL(1)文法を、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、これを拡張した。L属性文法は左から右へと属性評価を行なえる1パス可能な属性文法でその記述は比較的分かり易い。しかし、意味規則を素直に表現しようとすると属性値が右から左に決まる形になることが多い。それをL属性文法の範囲で記述するためには構文規則の書換えなどが必要になる。そう言った場合でも、もとの右から左に属性値が決まる表現のままで1パスで評価をする方法を考案した。

本論文では、本システムで用いる属性評価規則の記述方法と1パスで評価する方法について述べる。以下、2節で本システムで用いる意味評価規則の記述方法について述べ、3節で簡単な生成規則例を用いてL属性でないものを1パスで評価する方法について4節では従来の1パス型属性文法ではうまく表現できない場合の拡張した属性評価の方法について述べる。

2　本システムにおける属性評価規則の表記法とその意味

2.1　属性の記述方法

属性は矢印付きの添字で表し、例えばident(↑name)という形をとった場合、上向きの矢印は、nameがidentの合成属性であることを示す。非終端記号では上述の合成属性の他に継承属性も扱うことができ、例えば、condition(↓truep, ↓falsep)の↓truep,↓falsepの下矢印は、継承属性であることを示す。

2.2　意味操作関数の記述方法

属性文法における意味規則は宣言的なものであるが記号表への登録や目的コードの生成は翻訳の最中に行なわれる操作と考える方が素直である。また、ある構文要素に関する意味規則はその要素に隣接して記述した方が分かり易い。特に正規右辺文法ではプログラム上でその要素に対応するものの出現個数が不定であるので分かり易くなる。

終端記号及び非終端記号の記述では属性名についている↑,↓は合成属性、継承属性を表すが、意味操作関数のパラメータでは出力パラメータ、入力パラメータを表す。例えばキーワードdclで始まり型と1つ以上の変数が続く変数宣言を記述する次の文法を考える。

s::= 'dcl' type idlist ';' .

idlist::={ ident ","} .

identは識別子を表す終端記号であり、{ident","}はidentを,で区切って並べたものである。ここで記号表登録を行なう意味操作は、各変数について、その名前と型を一緒に登録するように記述したい。そのため下記の生成規則に示すようにidentの直後に各変数の名前nameと変数の型typを記号表に登録する関数§TABLE(↓name, ↓typ)を書いてある。

s::='dcl' type(↑typ) idlist(↓typ) ':'.

idlist(↓typ)::={ident(↑name)§TABLE(↓name,↓typ)","}.

2.3　本システムで用いる基本的意味関数

コンパイラの意味処理で基本的なものは記号表の操作とコード生成である。本システムではmemberとappendという2つの基本的な意味関数を用意し、この関数を用いて記述された意味については、以下に述べる処理を自動的に扱えるようにした。記号表の操作についてはmember関数を用いて、あるキーをもつエントリが記号表にあるかどうかの検索を、append関数は記号表にエントリの登録を行なうのに用いる。コード生成についてはappend関数を用いてコード格納領域に順次コードの格納を行なう。

3　拡張した1パス型属性文法の評価方法

上述の記述方法を用いて、本システムで扱う拡張1パス型属性文法とその評価方法について述べる。

3.1　従来のL属性文法における問題点

L属性文法は、左から右へと1パスで属性評価が可能な属性文法で、その記述は比較的分かり易い。しかし、一般には右の方で決まる属性の値を左の方で使う場合も結構ある。このような場合、L属性の範囲では記述できないから無理にL属性文法で表現するためには構文規則を変更しなくてはならず、その結果、記述が複雑となってしまう。例えば、pascalでは、変数の型宣言は、以下のように記述する。

```
var v1,v2,v3,v4:INTEGER ;
```

この例では各変数名の宣言を行ない、最後にこれらの変数の型が示されている。この場合、意味を自然に記述すれば各変数についてその名前と型を記号表登録を行なうように記述することになるが、不定個の変数名の後に型が現れるのでその属性値は右から左に決まってしまう。そして、そのような記述から1パスで構文解析と一緒に属性評価を行なえるようにする1つの方法として、不定個の変数をリストにつなぎ、型が現れた時点でそのリストの各要素に型をつけて登録するのが考えられる[1]。これは、左から右へと言う意味解析に無理に記述をあわせる方法であるため、余計な記述や操作となってしまう。

*A practical one-pass attribute evaluation

Hiroyuki NAKAGAWA,Hidenobu KANAYA,Ikuo NAKATA
(Univ. of Tsukuba)

3.2 本システムにおける属性評価規則の拡張

上述のように意味解析に合わせて生成規則を書き換えるのではなく、もともと考えた意味のまま生成規則を記述できた方が良い。しかし、それは従来のL属性文法に対する1パス型評価器では属性評価ができない。そこで従来の手書きのコンパイラで使用しているバックパッチの手法を属性評価に導入した拡張1パス型属性文法を提案する。この属性文法を用いた上述の変数宣言の記述はバックパッチによる属性評価を許したことで下記のような記述になり、右から左への属性の受渡しで属性評価が行なえる。

variable::='var' idlist(↓typ)':'type(↑typ).
idlist(↓typ)::={ident(↑name)§TABLE(↓name,↓typ)","}.

§TABLE 実行時に渡されるパラメータの内、*↓typ* は決定していない。しかし、*↓typ* は、*type* を構文解析した時点で決まるものであるから、その時点でバックパッチすればよい。この評価方法の導入により構文規則が簡潔になり容易に意味規則との対応がつく。

4 評価時における属性評価方法の拡張

前節では生成規則の記述の中で、属性値が右の方で決まるため左の方で使う時にはその属性値が決まっていない場合の属性評価方法について述べた。従来の1パス型属性文法にはL属性、LR属性などがあるが、属性文法の記述を見ただけではL属性などではないことが分からない場合の表現ができないという問題がある。本システムではこういった場合でも用意した基本的な意味関数で表現された範囲であれば、これを自動的に扱えるようにした。本節では簡単な例を挙げて生成規則と意味関数の記述方法、そして、その評価方法について述べる。

4.1 従来の1パス型属性文法での評価方法

従来、L属性文法で表現する時、goto文などではラベルLを生成し"jump L"命令を生成、後でアセンブラのパスでLの番地を決めている。この場合、呼ばれる度に違うラベルを生成するという特別な関数が必要になり、かつ、ラベル解決のパスが必要になる。それを避けるため、以下のように記述はできるだけ素直に書くようにする。

生成規則 R1

```
goto::='goto' ident(↑name)§TABLEGET(↓name,↑address)
            §INSTRUCTION(JMP,↓address) .
```

生成規則 R2

```
label ::= ident(↑name) ':' §CODE(↑address)
            §TABLE(↓name,↓address).
```

§TABLEGET はラベル名称をキーに記号表を検索しその番地を得る意味関数であり、§INSTRUCTION は中間コードの作成を、§CODE は現在の目的コード格納番地を得るという意味関数である。

上述のような記述では属性評価時に属性値が決まっていないということがおきる。つまり、コンパイラは goto L という文に出会った時、ラベルLが既に定義されたものであれば、記号表より、その番地を得て goto のための命令を生成すればよいが、飛び越しが前向きの場合、その goto 文でラベルLが初めて現れた場合もある。その場合には、飛び越し番地を指定しないままの goto 命令を生成しなくてはならなくなる。本システムでは基本的な意味関数で用いる入出力パラメータにバックパッチの手法を導入することで、この問題の解決を図った。

4.2 意味関数における属性評価方法

4.2.1 出力パラメータをもつ意味関数の記述と評価方法

生成規則 R1 の意味関数§TABLEGET(↓name,↑address) はラベル名称である入力パラメータの↓name をキーに記号表の検索を行ない、その番地を出力パラメータ↑address に返すもので、その記述は以下のように行なう。

§TABLEGET(↓label,↑address):- member([label,↑address],symbol).

symbol はコンパイラ設計者によってその構造が定義された記号表の名称、[label,↑address] は記号表のエントリ、基本関数で用いられている↑address は変数を意味する。ここで、基本関数 member を実行することで、出力パラメータ↑ address の結果は以下の2種類が考えられ、これに対応できる評価器の生成を行なうようにする。

- エントリが記号表になく出力パラメータ ↑address の値が未定
- エントリが記号表にあり出力パラメータ ↑address の値が決定
- エントリが記号表にあるが出力パラメータ ↑address の値が未定

4.2.2 入力パラメータをもつ意味関数の記述と評価方法

生成規則 R2 の意味関数§TABLE(↓label,↓address) はラベル名称をキーに記号表の検索を行ない二重登録でないことを確認後、記号表にラベルの名称と番地の登録を行なうものであり、生成規則 R1 の §INSTRUCTION(JMP,↓ address) は、中間コード格納領域に goto 命令を格納するもので、それらの記述を下記に示す。

```
§TABLE(↓label,↓address):- not(member([label,address],symbol)),
            append( symbol,[label,address]).
§INSTRUCTION(↓opcode,↓operand ):-
            append(program,[opcode,operand]).
```

ここで入力パラメータ↓address の状態によって入力パラメータ↓ address の評価は以下の2種類の処理を考慮した評価器となる。

- 入力パラメータ ↓operand の値が決まっていないため、[opcode,operan をバックパッチリストにつなぐ
- 入力パラメータ ↓address が決定した値で [label,address] のエントリを記号表に書き込む、そしてバックパッチリストにつながれている要素（例えば [opcode,opernd]）がある場合は入力パラメータ ↓address の値でバックパッチ処理を行なう

5 まとめ

正規右辺文法の中に簡潔な表現の意味規則を埋め込み分かり易い記述で効率の良い1パスコンパイラを生成する生成系の属性評価規則の記述方法と拡張した1パス型属性文法の評価方法について述べた。生成規則の記述の中で右で決まった属性値を左で用いる場合にもそのまま書けるようにしたことで読解性を向上させ、その属性評価にバックパッチの手法を導入することで1パスで評価できるようにした。今後は、各種の手続き型言語の生成を行なって評価方法の正当性を確認して行きたい。

参考文献

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

[2] Jullig,R,K. and DeRemer,F.:Regular Right-Part Attribute Grammars,Proceedings of the ACM SIGPLAN'84 Symposium on Compiler Construction,SIGPLAN Notices, Vol.19,No.6(1984),pp.171-178

[3] 丁亜希, 渡辺美樹, 中田育男, 佐々政孝:
正規右辺属性文法と1パス再帰降下属性評価器の生成,
情報処理学会論文誌 vol.30,No.2(1989),pp.204-212.