3H－7

属性文法に基づいたインクリメンタルなPascal-Sコンパイラ
The Incremental Pascal-S Compiler based on Attribute Grammar

亀山 裕亮*　中井 央*　中田 育男*　山下 義行*

1　はじめに

あるソースプログラムに対して属性評価が行なわれる際に、ソースプログラムの変更に伴なって、その変更箇所から影響を受ける範囲のみをコンパイルすることをインクリメンタルなコンパイルという

また構文解析と同時に解析木を作らずに、属性評価を行なう属性文法のクラスにLR属性文法[3]がある。LR属性文法によるコンパイラの記述から作成されるコンパイラは、1パス型のコンパイラであり実行効率のよいものである。

本論文では1パス型属性文法であるLR属性文法に基づいた、インクリメンタルなコンパイラにおいて、コード生成を含めた時に起こる問題点をあげ、その解決策について述べる。

2　インクリメンタルな属性評価

この章ではLR属性文法に基づいたインクリメンタルな解析法について述べる。

ある入力記号列 $w = x_0y_1x_1$ 及びその一部を変更した $w' = x_0z_1x_1$ があり(x,y,zはトークン列)、この入力記号列 w,w' に対して構文解析を行なった時の解析木が、それぞれ図1のように t_1,t_2 となるとする。ここで、y_1 を含む部分木を t'_1、z_1 を含む部分木を t'_2 だとすると図1中で影をつけた2つの部分が等しくなるものを見つければ、t'_1 を t'_2 で起き変えることで、それ以降の t_2 の解析を省略することができる。このための条件を構文照合条件という。構文照合条件に関する詳細は[2]を参照されたい。

次にLR属性文法に基づいたインクリメンタルな属性評価について述べる。

いま図1において、t'_2 への還元が起こったとする。もし構文照合条件が成り立つとすると、この時、t'_2 の合成属性の値さえ t'_1 の合成属性の値と同じであれば、古い t'_1 以下の部分木を新しいもので置き換えることで、それ以降の属性評価を省略することができる。

これはLR属性文法はL属性であるので、左から右への属性依存しかなく、また、t'_2 をルートとする部分木から外側へ属性を渡す唯一の方法は t'_2 の合成属性を通じてのみであるため、t'_2 の合成属性が以前の評価時の値と同じであれば、以降の属性評価の結果も同じになることによる。

*筑波大学 University of Tsukuba

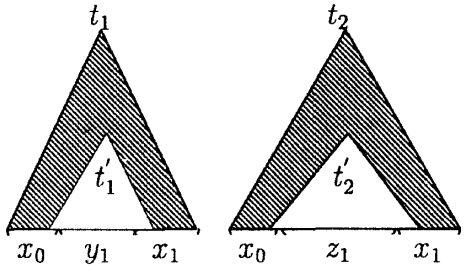

図 1: 入力列がそれぞれ $x_0y_1x_1, x_0z_0x_1$ の場合の解析木。

以上のことから、構文照合条件を属性の評価も含めたものに拡張した属性照合条件を定義する。

属性照合条件

構文照合条件が成り立ったときの、解析木 t'_1 と t'_2 の合成属性が同じである。 □

この属性照合条件が成り立ったところで解析を終了することができる。

3　コード属性と属性照合条件

前章で述べたようにインクリメンタルなコンパイラでは、属性照合条件を調べるために、前の解析で作られた合成属性と再解析で新しく作られた合成属性とを比較しなくてはいけない。ところがこの時にオブジェクトコードのようなものにおいては問題が生じることがある。

次のような例を用いてこのことを考えてみる。

変更箇所を含む生成規則 (X は非終端記号である)

$X_0 \rightarrow X_1 \cdots X_n$

において、それぞれの非終端記号Xはコードのため合成属性を持っていて X_0 に還元される時には、それぞれの非終端記号の持つ合成属性をリストで連結していくとする。インクリメンタルな再解析で X_0 に還元された時にコード属性の属性照合条件を調べるのであるが、再解析によって変更箇所のコード属性は新たに作り直されてしまうので、前の解析で作られた属性とは別のものとなってしまう。そのため最初の解析と再解析とで作られるコード属性が一致することはなく、照合条件は成り立たない。

前章で述べたインクリメンタルな属性評価器にこ

のようなコード属性を含めた場合、変更箇所の前まではコードを再利用しながら解析を進めていき、変更箇所の再解析により作られるコード属性に対しては、先ほど述べたように属性照合条件がなりたたないため、入力の最後まで解析を行ないコードを繋いでいくことになる。この時、変更箇所より後ろの入力を解析する際、部分木が前回と同じになっていることがわかれば、その部分木の解析を省略することができる [2]。

しかし、部分木の省略ができたとしても、この方法では変更箇所の解析が終った後も入力の最後まで解析を続けてしまうため、再解析時間がソースプログラム中での変更箇所の位置に影響されるという問題点がある。特にソースの前半に変更箇所があると、再解析の範囲が大きくなるので効率が悪い。そこで、この問題を解決するためにコード属性の属性照合条件について考えてみることにする。

まずコードに関する属性の性質について考えてみると、合成属性について次の後置的合成属性というものがある [5]。

定義 1 (後置的合成属性) 各生成規則に対して、合成属性 a の意味規則が次の条件を満たしているとき、その属性 a を後置的合成属性という。ただし、$\omega_i(0 \leq i \leq n)$ は終端記号、$X_j(0 \leq j \leq n)$ は非終端記号、S は開始記号、$\|$ は属性を連結する演算子である。また、終端記号は属性を持たないと仮定する。

$$X_0 \rightarrow \omega_0 X_1 \omega_1 \cdots X_n \omega_n, (X_0 \neq S)\ ^{1} \quad \cdots (1)$$
$$X_0.a := X_1.a \parallel \cdots \parallel X_n.a \parallel tail. \qquad \square$$

ある合成属性が後置的であれば、その属性はソースプログラムに出現する順番に繋がれていき途中で入れかわることがない。もしコード属性がこの性質を満たしているならば同様に、ソースプログラムに出現する順番に繋がれていき途中で入れかわることがない。

この時 (1) のような生成規則について考えてみる。入力の始めから X_0 の前の部分に対応するコードを α、X_0 の後ろから入力の最後までに対応するコードを γ とすると、最初の解析により $(\alpha\ X_1.a \cdots\ X_n.a\ \gamma)$ というコードが作られる (ここで本論文ではコードというものはリスト構造により実現する)。

次に再解析が行なわれ、変更箇所を含む $X_i(1 \leq i \leq n)$ への還元が起った時、もし変更箇所より後ろの $X_{i+1}, \cdots, X_n$ に関して上で述べた部分木の再利用が可能であるならば、X_0 への還元が起った後で $(\alpha\ X_1.a \cdots X_i \cdots X_n.a)$ というコードが作られている。ここで、$X_n.a$ は最初の解析の時に作られている属性であるので、$X_n.a$ には最初の解析の時に残りの入力に対するコードである γ が連結されている。そのため、再解析の時 $X_n.a$ が再利用されると $X_n.a$ には γ のコードすなわち X_0 以降のコードが連結されているため、$(\alpha\ X_1.a \cdots X_i \cdots X_n.a\ \gamma)$ という全体のコードができていることになる。

以上のことを考慮してコード属性に関する属性の照合条件は次のように定めることができる。

コード属性の属性照合条件

(1) の生成規則において、X_0 への還元が起ったときにコードの属性 $X_n.a$ が再利用された属性である。 $\square$

コード属性の属性照合条件が成り立ったところでコンパイルを終了することができる。

4 実験結果とまとめ

本論文では、目的コードとして Pascal-P4 コード [4] を出力するインクリメンタルなコンパイラを作成するために、1 パス型属性評価器生成系 Rie [1] により作成されたインクリメンタルな Pascal-S の属性評価器にコード属性を付加した。

実験として 1000 行ほどの Pascal-S のプログラムを用意し、ソースプログラムの前半部分と後半部分に "writeln;" の 1 文を挿入した場合の、最初の解析時間と再解析時間を調べてみた。ここで、方法 1 とはコード属性に対しては属性照合条件を適用しない方法、方法 2 が方法 1 の改良であるコード属性の属性照合条件を適用する方法である。実験は SGI の Indy(OS:IRIX5.2, CPU: MIPS R4600 100MHz) 上で pixie を用いて行なった。pixie は実行にかかったマシンインストラクションとサイクル数を報告してくれるコマンドである。実験結果を図 2に示す。

変更箇所	ソースの前半		ソースの後半	
	最初の解析	再解析	最初の解析	再解析
方法 1	30,905,724	311,821	30,906,791	99,863
方法 2	30,905,724	26,167	30,906,791	27,298

図 2: 実験結果 (単位:machine cycle)

実験結果より、どちらの方法でも最初の解析に比べて、再解析の時間が 1/100 ~ 1/1000 で済んでいることがわかった。次に方法 1 と方法 2 の比較をしてみると、方法 1 では前半に変更箇所があった場合は後半に変更箇所があった場合と比べて 3 倍程度多くの時間がかかってしまっている。また方法 1 は方法 2 よりも前半に変更箇所があった場合では 10 倍、後半に変更箇所があった場合でも 3 倍の解析時間がかかっている。この実験より、方法 2 では変更箇所の位置によらず、再解析時間はほぼ一定であることが解った。

参考文献

[1] 石塚浩志, 佐々政孝: 属性文法によるコンパイラ生成系 (1986).

[2] 中井央: コンパイラにおけるインクリメンタルな解析法, 博士論文, 筑波大学大学院工学研究科 (1997).

[3] Sassa, M., Ishizuka, H. and Nakata, I.: A Contribution to LR-attributed Grammars, *J. Inf. Process.*, Vol. 8, No. 3, pp. 196-206 (1985).

[4] Steven Pemberton, M. D.: *Pascal Implementation - The P4 Compiler*, Ellis Horwood Limited (1982). (邦訳) 竹市正人・木村友則: 「Pascal の言語処理系」, 近代科学社 (1984).

[5] 渡辺喜道, 徳田雄洋: 属性値予測による 1 パス属性文法の評価法, 情報処理学会論文誌, Vol. 37, No. 3, pp. 384-392 (1996).

[1] [5] ではここに tail という生成規則に依存した定数があるが本論文では省略した。