ショートノート

構造エディタにおける下降型パーサのための構文木の圧縮化技法[†]

佐藤　豊[††]　板野肯三[†††]

構造エディタでは，プログラムの内部表現として保持する構文木のために，大量の記憶を消費する．この問題を解決するために，構造エディタ向きの下降型パーサにおいて，構文木の表現を簡略化するための手法を考案した．この方法では，下降時に明らかに必要なノードだけを作り，上昇時に必要になったときにノードを生成して挿入する．これによって，構文木のノード数が大幅に減少するだけでなく，パーサの解析速度も向上し，プログラムを編集した後のインクリメンタルな再構文解析にも自然に適応する．

1. まえがき

構造エディタでは，プログラムの構文や意味の情報を編集に利用するために，プログラム全体の属性付き構文木を保持しなければならない．そのための記憶量は一般にかなり大きく，もとのソースプログラムの数十倍にもなるので[1),5)]，構文木の表現を簡略化することが望まれる．その解決法としては，プログラムの構造の表現として，導出の過程を完全に表現した解析木(parse tree)ではなく，構文の骨組みだけを表す抽象構文木(abstract syntax tree)を用いる方法が知られている[2)~4)]．しかし，構造エディタの操作性を改善するために，プログラムを文字列として柔軟に編集することを許すと，もとの構文木の構造が変更されることがある．このとき，変更の影響が及ぶ部分だけの構文をインクリメンタルに再解析するためには，過去の構文解析の過程を回復する必要があるが，抽象構文木から解析木を再現するのは非効率である．

そこで，導出過程を保存した解析木を用いながら，構文的に冗長なノードを省略することにより，解析木を圧縮することを考える．使用するパーサは，再帰下降型のインクリメンタル・パーサ[6)]とする．下降型パーサは基本的に，下降する時にのみ無条件にノードを生成する．これに対して，下降時に必要なノードだけを作って下降し，上昇時に必要になったノードを生成して挿入する方法を考案した．この方法は，パーサの高速化にも役立ち，プログラムが変更された場合の再解析にも自然に適応する．また，その実現には，基本となるパーサに簡単な機構を付加するだけでよい．本論文では，この方法で省略が可能となるノードの種類について述べ，パーサの実現方法を簡単に説明する．

2. 解析木中の冗長なノード

プログラムの属性付き解析木のなかで，空でない子ノードをただ一つだけ持つノードは，多くの場合，構文を表す上で冗長なノードである．このような冗長なノードは，属性や意味規則を持たないならば，実際に不要なノードである．したがって，これらを除去することによって，必要な情報を失うことなく，解析木の表現を簡略化することができる．以下では，簡単のために，構文上の冗長性だけに注目する．各種の表記法は文献4)に従う*．非終端記号 X からの導出として，

$$X \overset{*}{\Rightarrow} W\xi \Rightarrow \gamma\xi \overset{*}{\Rightarrow} Y\alpha$$
$$X \overset{*}{\Rightarrow} W\xi \Rightarrow \rho\xi \overset{*}{\Rightarrow} Z\alpha$$

の二つの最左導出があるとき，$\gamma \neq \rho$ は FIRST($Y\alpha$) ∩ FIRST($Z\alpha$)$=\phi$** を意味しているということが常に真である場合を仮定する（これは，文法が LL(1) であるなら，常に正しい）．このとき，導出過程 $X \overset{*}{\Rightarrow} Y\alpha$ は，X と Y の組から一意に決めることができ，$\alpha \overset{*}{\Rightarrow} \varepsilon$ となる場合，この導出の過程は省略可能である．したがって，この導出過程で生成されるノードは冗長である．

冗長なノードをなくすには，構文解析を終えた後に

---

† A Method of Simplifying Parse Tree for the Top-down Parser of a Structure Editor by YUTAKA SATO (Doctoral Program in Engineering, University of Tsukuba) and KOZO ITANO (Institute of Information Sciences and Electronics, University of Tsukuba).

†† 筑波大学工学研究科
††† 筑波大学電子・情報工学系

* 小文字のギリシャ文字で長さ0以上の記号列を，大文字のローマ字で非終端記号を表す．また，⇒ は直接導出可能，$\overset{*}{\Rightarrow}$ は生成規則を0回以上適用して導出可能であることを表す．

** FIRST(η) は，記号列 η から導出できる終端記号列の先頭に来ることのできる終端記号の集合．

削除するのが簡単であるが，この場合，プログラムが変更された後に再解析が正しく行えることが保証されず，一度作ったノードを削除するコストも余分である．冗長なノードをはじめから作らないためには，

$$X \overset{+}{\Rightarrow} Y\alpha \Rightarrow \beta\alpha$$

のような導出があるとき，X から β を直接導出できるよう，新たに

$$X \rightarrow \beta$$

なる生成規則を文法に追加すればよい．しかし，こうすると，文法が LL(1) でなくなって LL(1) パーサでは解析できず，再帰下降パーサにおいても，バックトラックが増加してしまう．これを避けるには，X から $\beta\alpha$ が導出される過程で最初に使用される生成規則を，

$$X \rightarrow \beta\alpha$$

$$X \rightarrow \delta \qquad (\mathrm{FIRST}(\beta) \cap \mathrm{FIRST}(\delta) = \phi)$$

のように分解すればよい．しかし，この方法をとると一般に，δ を表現するための構文の規模は，かなり大きくなる．また，構文が複雑に変形されてしまって，意味を対応させにくくなる．

3. 本方式の概要

以下に述べる方法は，上述のような問題を生じることなしに，冗長なノードをはじめから作らない方法であり，もとの文法を変形せずに，パーサの性能を損なうことなく，簡単な機構の追加だけで実現できる．

（1） 基 本 方 針

前章に述べた省略可能な導出過程 $X \overset{+}{\Rightarrow} Y\alpha$ に対しては，Y から X へ逆にたどることができる．そこで，この過程を省略して導出を進め，この部分は後で必要になったときに挿入することを考える．これは，パーサの機構からみると，冗長になる可能性のあるノードを下降時には作らず，上昇時に必要な場合にのみ作成して挿入することになる．冗長なノードを生成する可能性のある生成規則は，次のような形をもつ．

$$A \rightarrow [\alpha]\ B\ [\beta]$$

下降時にこの生成規則が選択された場合，入力記号が FIRST (α) に含まれないなら，この生成規則を省略してBの解析に移る．上昇時には，この生成規則が省略されていることを知っている．もし，FIRST (β) に次の入力記号を含むなら，この生成規則を挿入して，β の解析に移る．このように，ノードを必要に応じて挿入する方法は，プログラムをエディタで編集した後に，インクリメンタルに再解析を行うときに起こる，新たなノードの挿入に自然に適応する．

（2） 実現上の制約

実際のパーサの性能や実現を考えて，省略が可能な生成規則は，下降時に省略可能か否かの判定と，上昇時に挿入するか否かの判定が，現在の非終端記号と入力記号から決定できるものに制限する．上述の形で表される生成規則のすべてに対して，$\alpha \overset{*}{\Rightarrow} \varepsilon$ かつ $\beta \overset{*}{\Rightarrow} \varepsilon$ であるあらゆる場合を実際に省略可能とすると，解析の下降時に $\alpha \overset{*}{\Rightarrow} \varepsilon$ であるか否かを判定する必要があり，これは性能的に不利であり，パーサの実現も複雑になる．そこで，

$$A \rightarrow B\ [\beta]$$

となるような生成規則だけを考える．さらに，この形式の生成規則が文法中に唯一である場合に限り，その生成規則を省略可能とする．これは，もし

$$A \rightarrow B\ [\beta]$$

$$C \rightarrow B\ [\gamma]$$

のような形で，Bが二つ以上の生成規則の右辺に表れていると，Bの解析が成功した後の上昇時に，どの生成規則を挿入するかを，実際の親ノードの生成規則に依存して判断する必要があるからである．また，たとえ $\mathrm{FIRST}(\beta) \cap \mathrm{FIRST}(\gamma) = \phi$ であっても，プログラムが構文エラーを含む場合には，誤った挿入が行われてしまう可能性がある．

4. パーサの構成例

基本とするパーサは，表駆動型の下降型パーサである．パーサに付加するアルゴリズムは基本的に3章(1)で述べたものであるが，高速化のためにこのアルゴリズムをパース制御表で置き換える．パーサ自体に加える変更は，上昇時の簡単な判定操作だけである．説明のために，**図1**に示すような言語Cの文法の一部を例として用いる．

（1） パース制御表の構成

以下の説明では，文法中の非終端記号の参照経路をパスと呼び，生成規則を省略して下降することをバイパスと呼ぶことにする．下降時に使用する制御表は，下降型のパーサで一般に使われる，非終端記号と入力記号の組から，遷移先の生成規則の番号を求めるものである．ただし，省略が可能な生成規則に対するエントリには，中間の生成規則をバイパスした最終的な遷移先の生成規則の番号を格納する．図1の文法に対する下降時の制御表は，**表1**(1)のようになる．

上昇時に使用する制御表は，解析を終えた非終端記

```
#1 : S    →  E ;
#2 : E    →  AE  [, E]
#3 : AE   →  CE  [AO AE]
#4 : CE   →  BE  [? BE : CE]
#5 : BE   →  UE  [BO BE]
#6 : UE   →  PE  [UO]
#7 : PE   →  ID  [PO]
#8 : PE   →  CO
#9 : ID   →  identifier
#10: CO   →  constant
     AO   →  = | += | -= | …
     BO   →  + | - | * | / | …
     UO   →  ++ | --
     PO   →  ( E ) [PO] | [ E ] [PO] |
              …
```

S:Statement E:Expression
AE:Assignment Expression
CE:Conditional Expression
BE:Binary Expression UE:Unary Expression
PE:Primary Expression
AO:Assignment Operator BO:Binary Operator
UO:Unary Operator PO:Primary Operator

図 1 Cの文法の一部
Fig. 1 A part of the grammar of language C.

号と，次の入力記号の組から，挿入すべきノードの生成規則を求めるものである．親の生成規則が省略可能でない非終端記号と，挿入すべきノードがない組み合わせに対しては，この表のエントリは空とする．省略可能なパスに現れる非終端記号に対しては，そのパスを逆にたどって，省略可能な生成規則 #N: X→Y[α] (#N は生成規則の識別番号)に対する FIRST(α) の和集合を作成し，この集合の各要素に，挿入すべきノードの生成規則番号 #N を対応させる．図1の文法に対する上昇時の制御表は，表1(2)のようになる．

(2) パーサのアルゴリズム

下降時には，上述の下降時の制御表を用いることによって，省略可能な生成規則が自動的にバイパスされる．上昇時には，まず，現在のノードと親ノードの間にバイパスされた生成規則があるか否かを調べる．これは，親ノードの仮定する子ノードの非終端記号と，実際の子ノードの非終端記号が一致するか否かで判定する．もし省略が行われていたなら，上昇時の制御表をひいて，挿入すべき生成規則があるなら，対応するノードを生成して挿入する．図1の文法に基づく，**図 2**(1)の例の完全な解析木は，図2(2)のように表されるが，本方式では，**図 3** のような過程で簡略な解析木を生成する．

表 1 パース制御表
Table 1 Parsing table.

(1) 下降時の制御表

非終端記号	入力記号		
	identifier	constant	…
S	#9	#10	
E	#9	#10	
AE	#9	#10	
CE	#9	#10	
BE	#9	#10	
UE	#9	#10	
PE	#9	#10	

(2) 上昇時の制御表

非終端記号	入力記号						
	,	=	?	<	++	(	…
S							
E	#2						
AE	#2	#3					
CE	#2	#3	#4				
BE	#2	#3	#4	#5			
UE	#2	#3	#4	#5			
PE	#2	#3	#4	#5	#6		
ID	#2	#3	#4	#5	#6	#7	
CO	#2	#3	#4	#5	#6		

(1) プログラム例

x = f ( ) ;

(2) 完全な解析木

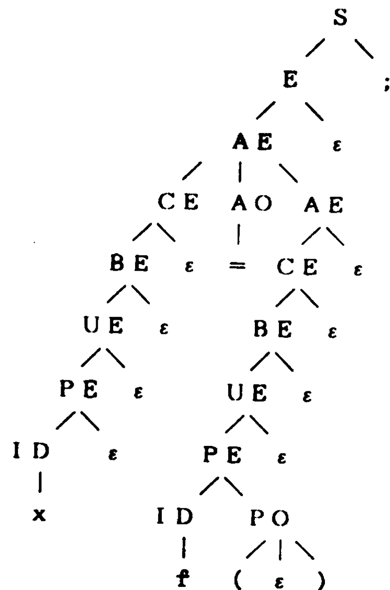

図 2 完全な解析木の例
Fig. 2 An example of a complete parse-tree.

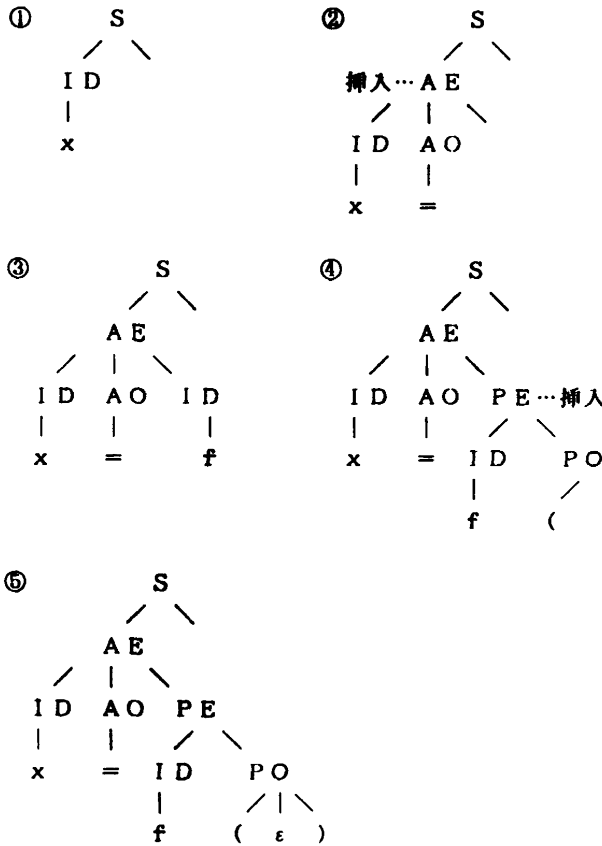

図 3　簡略化された解析木の生成過程
Fig. 3 The generating process of a simplified parse-tree.

5. むすび

上述の機構は，実際に言語Cの下降型インクリメンタル・パーサ[6]に組み込まれた．省略可能な生成規則は，特に式に関する構文に多いので，割合として式を多く含むプログラムほど記憶の節約効果は大きい．UNIX 上の5千行ほどのプログラムをこのパーサで解析したところ，解析木のノード数は 50% 以下に削減されていた．また，無駄なノードの生成を行わないため，パーサの速度を 30% 以上高速化できた．解析木のノード全体の中で，上昇時に挿入されたノードは約 13% であった．一方，解析木の冗長性を減らすと，意味解析や実行の高速化にも効果がある．実際に，この解析木を直接実行するインタプリタ[5]の速度は，10 % 程度高速化した．

本論文で述べた方式は，上昇型の解析における還元と同様な操作を含んでいるが，現在の非終端記号と一つの先読み記号だけで，動作が決定できるという性質は保っている．このために省略の適用範囲を制限したが，実際にはもっと広い範囲を省略することができる．例えば，ある非終端記号に至る省略可能なパスが複数存在する場合には，上昇時のパース制御表に親ノードの生成規則に関するもう一つの次元を加えることで，対応することができる．

参考文献

1) Schwartz, M. D. et al.: Incremental Compilation in Magpie, Proceedings of the ACM SIGPLAN '84 Symposium on Compiler Construction, *SIGPLAN Notices*, Vol. 19, No. 6, pp. 122-131 (1984).
2) Reps, T. and Teitelbaum, T.: The Synthesizer Generator, ACM SIGSOFT/SIGPLAN Software Engineering Symposium on Practical Software Development Environment, *SIGPLAN Notices*, Vol. 19, No. 5, pp. 42-48 (1984).
3) Donzeau-Gouge, V. et al.: Document Structure and Modularity in Mentor, *SIGPLAN Notices*, Vol. 19, No. 5, pp. 141-148 (1984).
4) Aho, A. V. et al.: *Compilers—Principles, Techniques, and Tools*, Addison-Wesley, Reading, Mass. (1986).
5) 佐藤，板野：構造エディタとインタプリタの統括的記述とその生成系，コンピュータソフトウェア（掲載予定）.
6) 佐藤，板野：構造エディタのためのインクリメンタル LL パーサの構成法，テクニカルノート HLLA-163，筑波大学電子・情報工学系 (1986).

（昭和 61 年 10 月 15 日受付）
（昭和 61 年 12 月 10 日採録）

佐藤　豊（正会員）
昭和 35 年生．昭和 57 年筑波大学第三学群情報学類卒業．昭和 59 年同大学院修士課程理工学研究科修了．現在同大学院博士課程工学研究科に在学中．プログラミング・システムのユーザ・インタフェースおよび構成法の研究に従事．ソフトウェア科学会会員．

板野　肯三（正会員）
昭和 23 年生．昭和 46 年東京大学理学部物理学科卒業．昭和 48 年同大学大学院修士課程修了．昭和 51 年同博士課程単位取得後退学．理学博士．筑波大学計算機センタ準研究員，同大学電子・情報工学系助手，講師を経て，現在，同助教授．コンピュータアーキテクチャ，オペレーティングシステム，プログラミング言語に興味を持つ．ソフトウェア科学会，IEEE，ACM 各会員．