ショートノート

コード生成向きパタン照合における簡易な競合解消方法†

渡邊 坦†† 久島 伊知郎††

中間語をテンプレートと構文解析手法でパタン照合し，オブジェクトコードに変換する方法は，コード生成の有力な一方法である．中間語の構文規則は曖昧性が高いので，その場合，パタン照合の競合解消が重要課題となる．本論文では，還元条件を各生成規則に対して指定できるボトムアップ型構文解析系において，部分パタンを還元してコードに変換する際，そのコードに固有の標識を付け，その部分パタンを先頭とするより長いパタンが検出された時，その標識まで遡ってコードを生成し直す方法を示す．これはある種のシフト／還元競合を誤りの恐れなく解消する簡易な方法であり，効率の良いオブジェクトの生成に使える．

1. はじめに

コンパイラにおけるコード生成の有力な方法として，中間語を構文解析してパタン照合する Graham-Glanville 法等がある[1),2)]．このパタン照合では，構文の合致するパタンが複数あれば，どれを選ぶかを属性情報で決めるなどの方法で競合解消が行われる．

構文解析をパーサジェネレータで生成したボトムアップな解析系で行う場合，構文に曖昧性があれば，シフト/還元競合ではシフト優先，還元/還元競合では演算の優先順位に従うことで一部対処できる[3)]．還元条件や相続属性指定により，属性に対する条件を細かく指定する方法もある[4)]．文脈依存な条件の扱い方として，条件節をマーカ非終端記号に変換して構文に組み込み，条件節の評価時に先読み記号を修正する方法[5)]や，条件を満たすマーカのみが構文則に適合するとみなす方法[6)]などがある．

しかし，これらはいずれも，構文上の分岐点かそれ以前に条件節を評価して，どの生成規則を選択するかを決めるものであり，構文上の分岐点を越えたあとでしか評価できない条件判定のある場合には適用できない．このような条件判定は，ソースプログラム言語ではあまり必要とされないが，最適化変換後の中間語からコードを生成する時にはしばしば必要になる．本報告では，これに対処する簡易なコード生成方法を示す．

† A Simple Technique of Conflict Resolution for Pattern Matching in Code Generation by TAN WATANABE (Systems Development Laboratory, Hitachi, Ltd.) and ICHIRO KYUSHIMA (3rd Department, Systems Development Laboratory, Hitachi, Ltd.).

†† (株)日立製作所システム開発研究所

2. 競合解消における問題点

コード生成では，中間語のデータ構造だけに注目したパタンでは不十分で，アドレスモードやオペランドの型，レジスタ番号などの属性を考慮しなければならない．すべての属性を構文要素として扱うと，パタンの数が膨大になるので，属性の組み合わせによる場合分けは，競合解消の条件節で指定することが多い．

例として，オペレータとアドレスモードの各ペアを一つの構文要素として扱い，レジスタ番号や，変数名，定数などをその属性として表す場合をあげる．これでは，$A:=B+C$; は，A，B，Cがメモリにあれば

LOADMEM ADDMEM STOREMEM

という中間語パタンで表現し，AとBがレジスタでCが定数であれば，

LOADREG ADDCON STOREREG

と表現するものとする．

二つのパタンPとQにおいて，QはPの後ろに構文要素の列を付加して長くした形をしているとき，Qのパタンに合う入力があると，一般には，それをPで部分的に還元するより，Qで還元した方がより適切な変換ができるとみなされる．これは，シフト/還元の競合はシフト優先とすることで選択できる．構文が合致した時に属性も調べる還元条件を加えると，さらに細かく場合分けできる．しかし，還元条件は構文の照合を終えてから判定するので，還元条件に合わないと判明した時点では，構文の短い他のパタンで変換することはもはやできない．ごく簡単な例で説明すると，

StatementSeg→AssignSeg|…|DefaultPattern

で始まる生成規則群において，

DefaultPattern→

```
LOADREG    {オペランド2レジスタをオペランド1レジスタに転送;}
| ADDCON    {オペランド1レジスタに定数を加算;}
| STOREREG  {オペランド1レジスタをオペランド2レジスタに転送;}
```

のように，各々の要素パタンに対応するコード生成規則（{ } 内は還元時のアクション指定）と，これらが同じレジスタをオペランドとして

LOADREG　ADDCON　STOREREG

という列を作っている場合には

```
AssignSeg→LOADREG  ADDCON
              STOREREG
    {レジスタインクリメント命令への変換指定;}
```

というコード生成規則を与えるとすると，レジスタが三つの要素パタンを通じて同じか否かは最後まで見ないと分からないので，LOADREG や ADDCON を通り過ぎた後でしか，前者を選ぶか後者を選ぶかを判定できない．レジスタの不一致が判明した時は，前者の生成規則との構文上の分岐点は通り過ぎているので，その LOADREG や ADDCON に対するコード生成規則は適用できない．マーカ非終端記号を挿入して条件判定する場合も，そのマーカまでの照合が終わった状態で判定するので，条件が合わないとき，その時点以前に構文的に異なってしまったパタンは採用できない．

効率の良いオブジェクトを生成するには，条件付きの長いパタンに適合したらそれによる変換をし，そうでなければ（短い）一般規則に従って変換することが必要なので，このような問題はしばしば発生する．

3. コード生成やり直しによる競合解消

ボトムアップな構文解析で，同じ非終端記号に還元されるパタン P と Q において，Q が P の後ろに構文要素列を付加した形の場合，長いパタン Q の還元条件が満たされなかったら短いパタン P に対するコードを生成することは，コード生成のやり直しを許せば簡単に実現できる．すなわち，P を認識した時点でひとまず P に対するコードを生成しておき，さらに進んで Q の構文を認識したとき，Q の還元条件が満たされていれば，P によるコードを捨てて Q で還元したコードを生成し，Q の還元条件が満たされていなければ，P によるコードをそのまま残せばよい．

いったん生成したコードをどこから生成し直すかを決めるため，P のように，他のパタンの先頭にくる可能性のある部分パタンに対しては，それによって生成したコードの先頭に或る標識を記入し，P を先頭の部分パタンとする Q のようなパタンで還元する時に，その標識の所まで戻る．具体的には次のようにする．

（1） 生成したコードは，ラベル定義点など，ある区切りがくるまでメモリに保持する．

（2） 他のパタンの先頭に来る可能性のある部分パタン P に対しては，それを右辺とする生成規則 $X \rightarrow P$ を導入し，後戻りの可能性のある文脈では，$X \rightarrow P$ の還元時に生成するコードの先頭に，生成コード一連番号等の一意な標識を付け，それを非終端記号 X の一つの属性とする．

（3） パタン Q の先頭にパタン P が含まれるとき，すなわち，生成規則 $Y \rightarrow Q$ の Q が $Q = PQ'$ と表現できるとき，これを $Y \rightarrow XQ'$ と表現し，$Y \rightarrow XQ'$ で還元するときのコード生成用アクション部では，$X \rightarrow P$ で付与した標識まで後戻りしてから，XQ' に対するコードを生成する．$X \rightarrow P$ で付ける標識を $X.CodeId と表し，コード生成位置を $X.CodeId の標識まで後退させるとともにその後退の成否を示す関数を PopCode($X.CodeId) として，$Y \rightarrow XQ'$ による還元のアクション部を

```
if PopCode($X.CodeId) then
  XQ' に対するコードの生成;
```

と表す．PopCode($X.CodeId) は，$X$ に対して生成したコードがメモリ外に掃き出されていたり，X が後戻り用標識を生成しない還元条件のもとに還元されていたりして，求める標識を検出できない時に false となる．

[例]

```
StatementSeg→AssignSeg|…|DefaultPattern
AssignSeg→LoadReg AddCon StoreReg
  #reducecondition (
    ($LoadReg.Op1=$AddCon.Op1) and
    ($AddCon.Op1=$StoreReg.Op1) and
    ($LoadReg.Op2=$StoreReg.Op2) and
    (not $LoadReg.Reuse))
  {if PopCode($LoadReg.CodeId) then
      レジスタインクリメント命令の生成;}
…
DefaultPattern→
  LoadReg|AddCon|StoreReg|…
LoadReg→LOADREG
```

```
    {$LoadReg. CodeId:=ObjectCodeId 等の属性
     設定;
     LOADREG の Op 2 を Op 1 に移す命令列の
     生成;}
  AddCon→ADDCON
    {$AddCon. CodeId:=ObjectCodeId 等の属性
     設定;
     ADDCON の Op 2 を Op 1 に加える命令列
     の生成;}
  ...
```

この例では，全部大文字の構文要素 LOADREG 等は，2章の場合と同じく終端記号としての中間語要素を表し，頭文字など一部が大文字で他は小文字の構文要素は，非終端記号を表す．$ は構文要素の属性を引用する時の印であり，$LoadReg.Op1 は LoadReg の第1オペランドを表す属性を示し，$LoadReg.CodeId は LOADREG を LoadReg へ還元したとき生成したコードの標識を示すというように使う．$LoadReg.Reuse は，LoadReg のオペランド1のレジスタが後続の命令列の中で再利用されていれば true，いなければ false を表す．このコード生成規則により，DefaultPattern の変換を選ぶと

```
Trans Rt, Ra; Add Rt, #1; Trans Ra, Rt;
```

というコードに変換されるものも，属性条件が合えば

```
Add Ra, #1;
```

と変換され，効率良いオブジェクトが得られる．

競合解消条件を上記のように記述することは容易であり，より良いパタンに遭遇したときにどこまで戻って生成し直すかについても，標識がつけてあるので，誤る可能性が少なくなる．したがって，競合関係について思い煩うことなく，対象とする機種の特性を利用できるパタンを多数あげ，その変換形式を指示することにより，効率の良いオブジェクトを生成できる．

4. おわりに

生成規則ごとに還元条件を指定できるボトムアップ構文解析法によるコード生成方法に対し，簡単で誤りの恐れのないコード選択方法を示した．この方法は，部分的やり直しをするので速度低下が心配されるが，一つの適用例では，795 個の命令生成においてやり直しで廃棄したコードは 9.3% であり，実用上支障ない範囲と思われる．ただし，これはパタンを充実させるにつれて増えるであろう．他のパタンの先頭に来る可能性のあるパタンは，全体のパタン 300 個中の 11% であった．属性も構文要素として扱ってパタンを構文だけで表現するならば，やり直しせずにすますことも可能であろうが，その場合は，構文表が巨大になるばかりでなく，処理する構文要素数が多くなるので，やり直しのないほうが速いとは即断できない．しかし，速度については今後評価が必要である．

謝辞 貴重な助言をいただいた査読者，ならびに，本研究の機会を与えていただいた元日立製作所小田原工場早川ソフト技術センタ長と同社システム開発研究所堂免所長，有益な討論をいただいた同研究所神野主任研究員と小泉研究員に感謝いたします．

参考文献

1) Glanville, R. S. and Graham, S. L.: A New Method for Compiler Code Generation, *5th ACM Symposium on Principles of Programming Languages*, pp. 231-240 (1978).
2) Balachandran, Dhamdhere, D. M. and Biswas, S.: Efficient Retargetable Code Generation Using Bottom-up Tree Pattern Matching, *Comput. Lang.*, Vol. 15, No. 3, pp. 127-140 (1990).
3) Aho, A. V., Sethi, R. and Ullman, J. D.: *Compilers—Principles, Techniques, and Tools*, Addison-Wesley Publishing Co., Reading, Massachusetts (1986).
4) 久島，神野，橋本：パーサジェネレータ PGEN の開発，第 40 回情報処理学会全国大会論文集，pp. 670-671 (Mar. 1990).
5) Ganapathi, M.: Semantic Predicates in Parser Generators, *Comput. Lang.*, Vol. 14, No. 1, pp. 25-33 (1989).
6) McKenzie, B. J.: LR Parsing of CFGs with Restrictions, *Softw. Pract. Exper.*, Vol. 20, No. 8, pp. 823-832 (1990).

(平成3年6月21日受付)
(平成4年3月12日採録)

渡邊 坦（正会員）

1962年京都大学理学部数学科卒業．同年日本アイ・ビー・エム(株)入社．1967年(株)日立製作所に入社後，同社中央研究所，システム開発研究所にて，設計計算システムやプログラム開発用ツールの研究から，計算機言語とコンパイラの研究へと進んできた．1974年情報処理学会論文賞受賞．工学博士．ACM, IEEE Computer Society, 日本ソフトウェア科学会各会員．

久島伊知郎（正会員）

1986年東京大学理学部情報科学科卒業．同年(株)日立製作所に入社．以来，同社システム開発研究所にて，プログラミング言語，コンパイラの研究開発に従事．