循環属性文法に基づく生成系 Jun について

3J－6

佐々木 晃　徳田 雄洋　脇田 建　佐々 政孝
東京工業大学 情報理工学研究科　Email:sasaki@is.titech.ac.jp

1 はじめに

属性文法は、その強力な記述力と定式化の簡潔さから、古くからその理論に関する研究や、ソフトウェアへの応用に関する研究が盛んに行われてきた。その応用面では特に、様々なコンパイラ生成系の記述文法として属性文法が多く用いられており、著者らも属性文法を用いたコンパイラの自動生成の研究を行っている。本稿であげる Jun は、最適化器、コード生成器、インタプリタ・デバッガといったバックエンドをすべて属性文法によって記述できるコンパイラ生成系である。これまでに、最適化器までを自動生成できるコンパイラ生成系、特に属性文法によって記述できるものは少ない。

Jun は 1989 年に初版が開発されたが、システムが生成する評価器を大幅に改良するため、著者が改めて実装し直し、現在公開に向けて準備中である。本稿ではこの Jun についての概説を行う。

2 属性文法によるバックエンドの記述

属性文法によるコンパイラ記述の長所は、記述法が簡潔、宣言的という点にあるが、特にバックエンドの自動生成に関して他の方法と比べて勝っている点として

(1) コード生成器における式の最適コード生成では、従来の手続き的な方式に比べ宣言的に記述できる。これは、中間木のパターンマッチングでは記述できない。

(2) これまで、手続き的な手法で作られることの多かった、最適化器も宣言的かつ自然に記述できる。

などが挙げられる。ところが、この際問題となるのは(2)で、属性文法の古い定式化では属性の依存関係に循環がある場合、属性値は定まらないとされてきたが、最適化器を属性文法を用いて自然に記述しようとすると属性依存関係にサイクルが出来てしまう。

この解決法として、Farrow はある条件下では循環があっても属性値を求めることが可能である**有限帰納属性文法** [1] を提唱した。Jun では、この有限帰納属性文法を採用し、循環を許す強力な記述力を実現した。

3 生成系 Jun

生成系 Jun は、Common Lisp（単に Lisp ともいう）上で開発され、Jun が生成する評価器も Lisp 上で動作する。我々の研究室では、Jun と Rie という二つの生成系を組み合わせて、コンパイラの開発を行っている(図1)。Rie は Jun と同じく属性文法による記述から、構文解析、意味解析を同時に1パスで行う効率のよい評価器を自動生成するコンパイラ生成系であり、フロントエンドの自動生成に用いている。Jun が生成する評価器の入力は、フロントエンドによって生成された中間木で、実際はS式で表される抽象構文木である。各バックエンド（すなわち、Jun の生成する属性評価器）は、同じ中間木を共通に用いるようになっている。

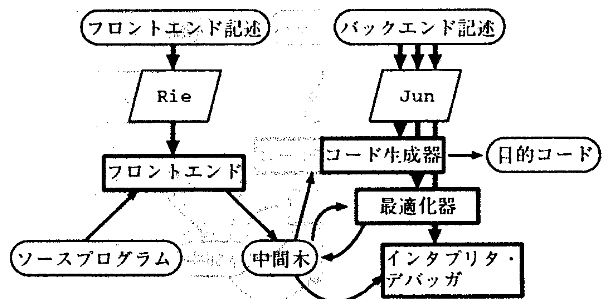

図1 属性文法によるコンパイラ自動生成

4 有限帰納属性文法に基づく評価器

次ページの図4は、データフロー解析の生存変数を求める属性文法記述である。拡張依存グラフ（図2）から分かるように、この記述には生成規則(*)で依存関係にサイクルが含まれているが、有限帰納属性文法に従っているので、Jun が生成する評価器では属性の計算を行うことができる。

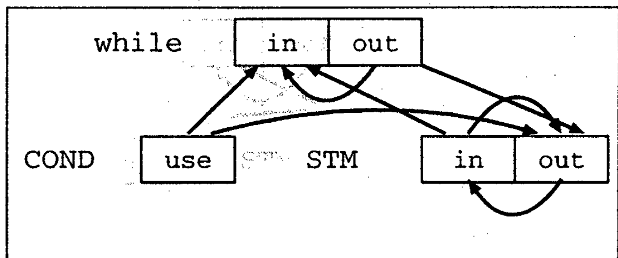

図2 拡張依存グラフ

この例では、`STM.out` と `STM.in` の間にサイクルが出来るが、{`STM.out`, `STM.in`} のようにサイクルに含まれる属性を一つの集合にまとめたものを CDC（circular dependency class）と呼ぶ。また、サイクルに含まれない属性はそれ自身のみで一つの CDC である。このとき、各 CDC をノードとすれば、これらのノード間の依存関係にはサイクルは含まれない（図3）。すなわち、サ

A summary of Jun : a compiler generator based on a circular attribute grammar
A. Sasaki, T. Tokuda, K. Wakita and M. Sassa
Tokyo Institute of Technology

イクルに含まれる属性値が定まれば、木全体の属性値を決めることができる。

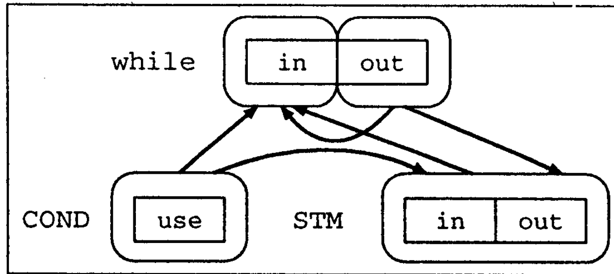

図 3 CDC

さて、属性の依存関係にサイクルが存在しても属性値を定めることができる条件は、

1. サイクルに含まれる属性の値域が完全半順序集合であること、かつ
2. それらを定める関数が単調で収束すること

であり、これが有限帰納属性文法であるための必要十分条件である。この条件のもとでは、初期値をボトム（またはトップ）として、属性規則に従って評価を繰り返すことにより最小（最大）不動点を求めることができるので、これらをサイクル内の属性の属性値と定めればよい。

5 例

図4に、Jun の入力となる記述(これを Jun 記述と呼ぶ)の例を示す。これは、データフロー解析における生存変数を求める Jun 記述である。まず、中間木の構造を表すためのノードとクラスを宣言する(図4(1))。ノードは評価器の入力となる中間木に現れる節であり、クラスはノードの集合である。中間木のあるノードから子ノードを見た場合に、似た役割をする子ノードを集めてクラスとすることにより、複数のノードのパターンに対する意味規則をひとまとめにして記述できる。

属性はクラス単位ごとに宣言する(図4(2))。**synt**、**inh** はそれぞれ合成、継承属性を表す。修飾子 **export** は、属性値を次のフェーズのために残す際に用いる。

木の構造の定義及び属性評価規則の定義(図4(3))は、次の形で行う。

木の構造 { 木に対応する意味規則 }

ただし、木の構造は

クラス名またはノード名 => クラス名$_1$,,, クラス名$_i$

のように記述する。また、サイクルに含まれる属性はユーザが明記する(図4(#))。

このような記述から生成された評価器は、属性評価規則にしたがって中間木上の属性を計算し、バックエンドとして動作する。

```
%class
  STM ::= stm_s | assign | if | while
          proc_call| null_stm;        ...(1)
...
%attribute
  C_BLOCK, C_PROCDECL, C_PROC_ID, STM =>
      out : inh,
      in : synt export;               ...(2)
  C_CONSTDEF, C_VARDECL,
  EXP, COND, BOP, UOP, C_NUMBER =>
      use : synt;
...
%semantics
  stm_s => STM, STM;                  ...(3)
    { STM[2].out = stm_s.out;
      STM[1].out = STM[2].in;
      stm_s.in = STM[1].in }
  null_stm;
    { null_stm.in = null_stm.out }
  assign => C_ID, EXP;
    { assign.in
      = (union EXP.use
          (set-diff assign.out C_ID.use)) }
  if => COND, STM;
    { STM.out = if.out ;
      if.in = (union COND.use STM.in) }
  while => COND, STM;                 ...(*)
    { %circle STM.out, STM.in ;       ...(#)
      STM.out =
        (union
          (union COND.use while.out)
          STM.in);
      while.in =
        (union
          (union COND.use while.out)
          STM.in) }
...
```

図 4 Jun 記述の例: 生存変数の解析

6 おわりに

著者らはこれまでに、Pascal の小さなサブセットである PL/0 言語の最適化器の一部、68000用 および sparc 用コード生成器、インタプリタ・デバッガといったバックエンドを、Jun を用いて開発した。なお、PL/0 でまだ実験されていない、ポインタ、配列、型に関する最適化等を属性文法によってうまく扱えるかどうかを調べるために、現在C言語のサブセットの開発をすすめている。

また、属性文法デバッガ [2] の実装を行い、属性文法によるプログラミング環境の向上にもつとめている。

謝辞 本研究の一部は、文部省科学研究費の補助を受けた。

参考文献

[1] R. Farrow. Automatic generation of fixed-point-finding evaluator for circular, but well-defined, attribute grammars. *SIGPLAN '86 Symp. on Compiler Construction*, pp. 85-98, ACM, 1986.

[2] 大久保琢也, 佐々木晃, 脇田建, 佐々政孝. 属性文法に対するデバッガの開発. 日本ソフトウェア科学会第11回大会論文集, pp. 345-348, 1994.