6F−4

ソフトウェア自動合成シェルSOFTEX/S (4)
−項書換え規則の互換性検証システム−

三木 雅人　佐藤 明良　山之内 徹　渡辺 正信
NEC C&Cシステム研究所

1 はじめに

ソフトウェア自動合成シェルSOFTEX/S[1]の変換規則の互換性検証システムを作成した。このシステムは、SOFTEX/Sによって生成された専用ジェネレータの出力プログラムが構文的に正しいことを保証するために、専用ジェネレータ生成のための変換規則作成者の強力な支援環境となる。

2 項書換え規則の互換性

多ソート項書換え規則の互換性とは、多ソート項が書換え規則によって書換えられた結果が多ソート項になることである。この互換性を満たすための必要十分条件は既に知られている[2]。また、文脈自由文法から多ソート項集合への同型写像が存在することもわかっており、文脈自由文法で書かれたプログラムを多ソート項集合にマッピングして、この多ソート項の項書換え規則の互換性を検証することで、元のプログラムが変換の前後で構文的に等価なことを検証することが可能となる。SOFTEX/Sでは、C++を完全に包含した仕様記述言語DSL/C++[1]のプログラムからC++のプログラムへの変換を行なっており、本システムによって、DSL/C++の枠を出ずに変換でき、変換されたプログラムの構文的正当性を保証することが可能となる。従って、変換規則作成者は、作成した変換規則を網羅テストを通さずに検証でき、専用ジェネレータ作成の手間の削減が図れる。

今回のシステムで扱う多ソート項集合は、関数の定義域には複数種類のソートの出現が可能だが(定義域多相)、値域には単一のソートしか出現しない(値域単相)。一般に多ソート代数では、定義域多相、値域多相の方が記述できる範囲が広くなり、その場合の互換性検証についても研究が行なわれている[2]。しかし今回は、値域多相では計算処理時間の増大が予想されること、また文脈自由言語と同型な多ソート項が値域単相であり、値域多相の要求がそれほどないことから、値域単相を採用した。

3 多ソート項書換え規則の互換性必要十分条件

定義域多相、値域単相の場合の多ソート書換え規則が互換性を満たすための必要十分条件について述べる。多ソート書換え規則の定義は文献[2]に述べられている。

定義 1 [互換性]　多ソート書換え規則 $l \rightarrow r$ が互換(compatible)であるとは、$\forall t \in T(\Sigma, V);\ t \Rightarrow_{l \rightarrow r} t'$ ならば $t' \in T(\Sigma, V)$ である時、その時に限る。ここで、tは多ソート項、Σ は多ソート指標、Vは変数集合とする。

定義 2 [ソートのロケーション] 多ソート指標を $\Sigma =< S, F >$ とするとき、ソート記号集合 S からソート付関数記号集合族 $F(S \times ... \times S \times \square \times S \times ... \times S)$ への写像 $Locations$ を以下のように定義する:

$$Locations(s) \stackrel{\mathbf{def}}{=} \{f(s_1, ..., s_{i-1}, \square, s_{i+1}, ..., s_n, s_R) \mid s, s_1, ..., s_n, s_R \in S;\ f_{s_1 ... s_{i-1} s_i s_{i+1} ... s_n s_R} \in F;\ s = s_i\}$$

ここで、□はソート記号sが現れている関数記号中の定義域の場所の置き換えである。

定義 3 [互換ソート順序 $\succeq$] 多ソート指標を $\Sigma =< S, F >$ とする時、ソート記号集合 S 上の疑順序 $\succeq$(互換ソート順序)を以下のように定義する:

$\forall s_1, s_2 \in S$ に対して $Locations(s_1) \subseteq Locations(s_2)$ の時、その時に限り

$$s_1 \succeq s_2$$

定理 1 [多ソート書換え規則互換性の必要十分条件] 多ソート指標を $\Sigma =< S, F >$ とし、ソート記号集合 S 上の順序を $\succeq$ とする時、多ソート書換え規則 $l \rightarrow r$ は

$$Sort(l) \succeq Sort(r)$$

の時その時に限り互換である。ただし、$Sort(t)$ は項 t の値域ソート記号を表す。

4 互換性検証システム

上述した互換性必要十分条件を用いて、変換規則の互換性検証システムを作成した。システム構成図を図1に示す。

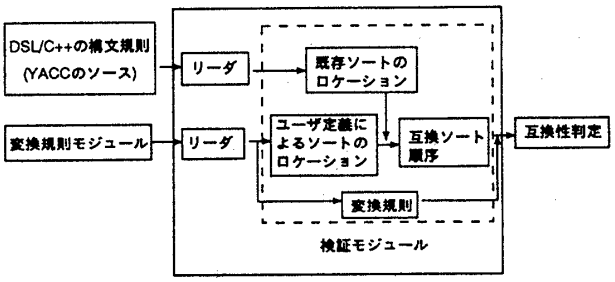

図 1:

システムは、二種類のファイルを読み込むためのリーダと検証モジュールから構成されている。また、検証はモジュール単位で行なわれる。互換性検証に必要なソートの文脈情報は、SOFTEX/Sの仕様記述言語DSL/C++の構文規則と、ユーザ定義の変換規則内の関数定義部から得られる。システムのユーザは、検証したい変換規則モジュールを入力すればよく、システムはこの中の関数定義部からソートのロケーションを生成し、これと、DSL/C++の構文規則から既に生成されている既存ソートのロケーションとをリンクしてか

Software Synthesis Shell SOFTEX/S (4) −Compatibility Checker for Term Rewriting Rules−
Masato MIKI, Akiyoshi SATO, Toru YAMANOUCHI, Masanobu WATANABE
NEC Corporation

ら、互換ソート順序を決定する。互換性検証は、モジュール内の変換規則を1つずつ互換ソート順序に適合するかを調べることで行なわれる。

5 検証例

本システムを用いた、変換規則の検証例を次に示す。変換規則の記述言語は、DSL/C++であり、vardef 'sort VAR; は, ソート sort の変数の宣言、termdef 'sort('sorts) term; は引数のソートがsortsで、termで構成される項のソートがsortである関数の宣言、relation name eql(l,r) は、変換規則名がrelation、左辺項がl、右辺項がrである変換規則である。[. list .] は、リスト list を表す。

```
vardef 'expression X,X1,X2;
vardef 'statement Y,Y1,Y2,L;
termdef 'statement('expression,'statement) P;
termdef 'selection_statement('statement) cond;
termdef 'selection_statement('expression,'statement)
        cond2;
termdef 'selection_statement('expression,
        'statement,'statement) cond3;
relation r1 eql (cond([. p(X,Y) .]),cond2(X,Y))
relation r2 eql (cond([. p(X,Y) : L .]),
        cond3(X,Y,cond(L)))
relation r3 eql (cond2(X,Y),
        'selectoin_statement[if(X) Y])
relation r4 eql (cond3(X,Y1,Y2),
                'selection_statement[if(X) Y1 else Y2])
relation r5 eql ('selection_statement[if(0) Y],
                'compound_statement[{}])
```

この変換規則集合は、cond文(条件とアクションを引数とする項pのリスト)をcのif文に変換するものである。文献[3]のC++の構文規則の中から、上記の変換規則内のソートが生成規則の左辺に出現するものを選択すると、

```
initializer : '=' expression
primary_expression : '(' expression ')'
                (expression に関してはこの他多数あり)
statement_list : statement
statement_list : statement_list  statement
                (statement に関してはこの他多数あり)
statement : selection_statement
statement : compound_statement
fct_body : compound_statement
```

定義2に従いながら、各ソートのロケーションを作成すると、

```
Locations(expression)={initializer('=',□),
      primary_expression('(',□,')'),...}
Locations(statement)={statement_list(□),
      statement_list(statemen_list,□),...}
Locations(selection_statement)={statement(□)}
Locations(compound_statement)={statement(□),
      fct-body(□)}
```

この中で、ロケーションの間に包含関係が存在するのは、

```
Locations(selection_statement)⊆
            Locations(compound_statement)
```

であり、定義3から次の互換ソート順序が決定される。

```
selection_statement≥compound_statement
```

各変換規則は、左辺項のソートはselection_statementであり、右辺項のソートはselection_statementかcompound_statementであるので、定理1により、互換性を満たすことが検証される。しかし、ルールr5を次のように変更すると、左辺項のソートと右辺項のソートが互換ソート順序を満たさないために互換性に反する。

```
relation r5 eql ('compound_statement[{if(X) Y}],
                'selectoin_statement[if(X) Y])
```

実際このルールは、例えば

```
main() {if(X) Y} ⇒ main() if(X) Y
```

という構文的に正しくない変換を行なう。

6 評価

作成したシステムを評価するために、SOFTEX/Sを用いて作成した2つの専用ジェネレータ、SOFTEX/SEM[1]、SOFTEX/SDL[1]について、変換規則モジュールの互換性を検証した。結果は以下の通りである。

	全ルール数	互換性のないルール数
SOFTEX/SEM	51	3
SOFTEX/SDL	380	7

互換性を満たさないルールは、例えば、

左辺項のソート		右辺項のソート
declaration	⟶	identifier
statement_list	⟶	identifier
statement	⟶	statement_list

のような構文的に誤った変換を行なっており、このようなルールが検出できたことから、互換性検証システムの有用性が示された。

7 今後の課題

今回作成したシステムでは、ユーザが関数を定義する時に、その関数の定義域、値域のソートを完全に記述しなければ、検証が行なえないという問題点がある。実際に、関数によっては、定義域、値域のソートが定義時に記述されなくても、他の関数定義や変換規則ルールから見当がつく場合も多く、これらを全て記述するのは、ユーザにとってかなりの負担である。これを、解決するために、記述されない部分のソートを記述されたソートの情報から矛盾なく推論する方式を現在検討している。

また、今回採用した定義域多相、値域単相の記述能力の限界を調べ、必要ならば値域多相に対応するように拡張する予定である。

8 まとめ

SOFTEX/Sの変換規則の互換性検証システムを作成し、このシステムがユーザの専用ジェネレータ作成を支援することについて報告した。今後は、様々な専用ジェネレータ作成時に、本システムを利用することで、システムの有効性を評価するとともに上記の機能拡張を行なっていく予定である。

参考文献

[1] Yamanouchi, T., Sato, A., Tomobe, M., Takeuchi, H., Takamura, J. and Watanabe, M.: Software Synthesis Shell SOFTEX/S, *Proc. of the 7th Knowledge-Based Software Engineering Conference*, 1992.

[2] 佐藤明良, 山之内徹 , 渡辺正信 : 多ソート書換え規則の互換性, 日本ソフトウェア科学会 第9回大会全国大会論文集 1992.

[3] Ellis,M. and Stroustrup , B. : The Annotated C++ Reference Manual , *Addison-Wesley* 1990