3U－10　宣言型計算モデルに基づく仕様記述からの手続き型プログラムの生成*

安斉克浩　赤間清　宮本衛市†
北海道大学工学部‡

1　はじめに

手続き型言語によりプログラムを組む際の負担を軽減するため、Prologのプログラムから C言語のプログラムを自動生成する [1] システム等が研究されているが、一般の Prologは節やボディアトムに順序を含むなど、純粋な宣言型言語ではない。

これに対し、本研究では純粋な宣言型言語を仕様として用い、仕様から手続き型プログラムをプログラム変換によって生成する研究を行なった。この変換には計算の正当性や計算の制御の表現力などの優れた特徴を持つ、宣言型計算モデルに基づくルールを基礎としたプログラム変換という手法を用いている。

この研究の特徴としては仕様の汎用性と、正当性の保証の可能性が挙げられるが、これらはそれぞれ、純粋な宣言型言語を用いること、ルールを基礎としたプログラム変換という手法によって得られる。

2　仕様の記述

仕様は、述語の定義部と述語の使用限定部からなる。述語の定義部は有限個の確定節の集合であり、純粋な論理プログラムである。この定義部では述語の宣言的定義のみを与えており、それゆえ述語の特定の使用法には依存しないという特徴を持つ。

述語の使用限定部は、述語の使用法を指定するもので、実行時の入出力状態（引数の「型」と「返り値」）を宣言する。型には LIST 型、NUMBER 型、ALL 型などがあり、返り値は計算結果の返る引数を定める（ans の引数で示す）。

仕様記述の例として、任意の数に対し、その数を2倍して5を足した数を得る‘m2a5‘という述語の仕様を示す。定義部は純粋な論理プログラムで定義でき、使用限定部も容易に記述できる (図1)。

述語の定義部：

```
(m2a5 *x *y) ← (*tmp = (x *x 2)),
               (*y   = (+ *tmp 5)).
```

述語の使用限定部：

```
(ans *z) ← (m2a5 <NUMBER> *z:<ALL>).
```

図 1: 仕様 (m2a5)

3　手順の導出

仕様のうち、定義部は純粋な宣言型プログラムであり、節、ボディアトムに順序性はない。この仕様から手続き型プログラムを生成するには実行の際の動作手順を導出する必要がある。ここでは先程の m2a5 という述語を例に手順を導出するが、その前に m2a5 の2つのボディアトムを変換し得る変換ルールの一部を記述しておく（図2）。

さて、使用限定部より第1引数が入力、第2引数が出力であるので、m2a5 が実行される時点で定義部の引数のうち*x は具体化されるが*y は変数のままである。ここで、m2a5 の2つのボディアトムは変換ルールより、共に2つの変数のうち1つが具体化されなければ実行できないため、はじめに *tmp = .. がルール1により変換され、次いで *b = .. がルール3により変換される。

ここで、使用限定部の記述を図3のようにすると、上と同様に、はじめに *b = .. がルール4によって、次いで *tmp = .. がルール2によって変換されることになり、手順は先程とは逆になり、述語の意味も'5を引いて2で割る'に変化することになる。

4　プログラム生成の概略

仕様から手続き型プログラムへの変換は5行程に分けて行なわれるが、ここではそれらを簡単に説明

*Generation of procedual programs from specification based on declarative computation model
†Katsuhiro Anzai, Kiyoshi Akama, Eiichi Miyamoto
‡Hokkaido Univ.

```
(Rule 1 (Head (*c = (x *a *b)))
        (Cond (numberp *a)
              (numberp *b))
        (Exec (mul *a *b *c))
        (Body (empty)))

(Rule 2 (Head (*c = (x *a *b)))
        (Cond (numberp *c)
              (numberp *b))
        (Exec (div *c *b *a))
        (Body (empty)))

(Rule 3 (Head (*c = (+ *a *b)))
        (Cond (numberp *a)
              (numberp *b))
        (Exec (add *a *b *c))
        (Body (empty)))

(Rule 4 (Head (*c = (+ *a *b)))
        (Cond (numberp *c)
              (numberp *b))
        (Exec (sub *c *b *a))
        (Body (empty)))
```

図 2: 変換ルール

```
(ans *z) ← (m2a5 *z:<ALL> <NUMBER>).
```

図 3: 使用限定部 (m2a5)

する。

フェーズ1　ここでは述語の使用限定部に書かれた引数情報と述語の定義部とを組み合わせることにより変換ルール群を生成する。変換ルール群とは、引数間の背反性等を考慮して、述語の実行内容を引数によって場合分けしたプログラム変換ルールの集合である。

フェーズ2　実際にプログラム変換を実行するフェーズである。システムがはじめから持っている変換ルールと、フェーズ1で生成された変換ルール群とを用いて、述語の使用限定部に対しプログラム変換を繰り返し行なう。この際、入力引数の違いにより、動作がどのように変化するかを解析し、状態遷移図（図4）[1]を得る。

[1] 2つのリストをつなぐ append 述語の一般的な定義では第1引数の値により状態が変化する。値が cons の場合には再帰によって元の状態に戻り (S0 → S2)、nil の場合には終了する (S0 → S1)。

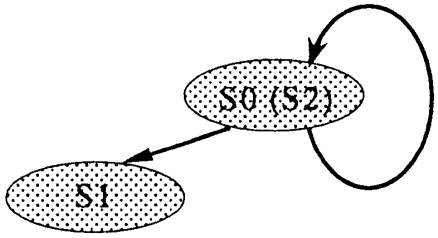

図 4: 状態遷移図 (append)

フェーズ3　ここでは変換の過程で得られた状態遷移図から、実際の実行の手順となる TREE を生成する。これは状態遷移図を元に、状態から状態への移行は条件分岐として処理し、各状態において実行すべき実行列を手順通りに並べたもので簡単な手続き型プログラムとなっている。

フェーズ4　ここでは手続きレベルでの最適化を行なう。具体的には再帰からループへの変換、レジスタアロケーションなどが挙げられる。

フェーズ5　ここでは得られた TREE を目的の言語仕様に合わせて1対1に変換を行ない、必要なデータ構造を実現する。この結果、目的とする手続き型プログラムが出来上がる。

5　おわりに

本研究では、汎用性の高い、手順を表していない確定節の集合で記述された仕様からルールを基礎としたプログラム変換という手法を用いて、厳密に正当性を保証した形で手続き的手順を導出し、手続き型プログラムを生成する手法を示した。

今後の課題としては、多種の仕様を網羅できるシステムの構築、及び、プログラム変換ルールのさらなる洗練と、またそれによるシステムの改良などが挙げられる。

参考文献

[1] K. Katamine On the Translation of Prolog into C, Proc of LPC'93, pp.31-38(1993)

[2] K. Akama: Unfolding for Generalized Logic Programs, *Preprints Work. Gr. for Artifficial Intelligence, IPSJ* 83-3-AI, pp.43-52(1992)