2J−7　変換履歴の再利用による高レベルのプログラム変換

小堀賢司，西谷泰昭 (群馬大学)

1. はじめに

プログラム変換は効率の悪いプログラムを効率の良いプログラムに変換する。しかし、従来の部分計算法やunfold/fold変換のようなプログラム変換法は、例えば、二分探索のような技法を直接利用したプログラムを生成することが難しい。このため、このような高レベルの知識はルールとしてシステムに保持されなければならない。本稿では、まず高階関数で簡潔に記述したプログラムを人間の高レベルの知識を用いるために対話的に変換する。そして、その変換履歴を一般化し他の類似の問題に適用することを考える。

2. プログラムの記述

高階関数は（1）表現力が豊かなので簡潔に記述でき（2）仕様としてそのまま使うことのできる形式的な定義を書き下せるという利点がある [1][3]。そこで高階関数でプログラムを記述する。しかし、高階関数で定義したプログラムは効率面では満足できないので、これを変換によって効率良くする。ここでは、高階関数として fold , filter , map を用いる [3]。例えば、fold の定義は以下のようになる。

```
(fold op init l)
      =
(if (null l) nil
    (op (car l)(fold op init (cdr l))))
```

例えばソートの定義は高階関数を使うと

```
(sort l) = (fold insert nil l)
```

と定義できる。

3. 変換履歴と知識の表現

効率良く問題を特くプログラムは入力データの性質やプログラム実行時の実行環境をうまく利用している。変換システムは、効率の良いプログラムに変換するために、プログラミングに関する種々の知識を扱わなければならない。このような知識を分類すると

1. プログラム言語の性質
2. 基本データに関する性質
3. 問題固有の性質

に分けられる。3はさらにソートされた列に関する性質のように、他の問題に適用できる一般的なものと、真に固有のものに分けられる。この区別は明らかでないが、人間はそれを見分けて一般的な部分を他の問題に利用している。3の知識を利用するためには問題固有の性質に対応できる変換ルールを持ち、さらに強力な証明系をそなえたシステムが必要である。しかし、これは以下のことで補うことができると考える。

1. 証明系の部分はユーザがシステムと対話する。
2. 効率の悪いプログラムから良いプログラムへの変換が1つのルールによって行なわれると考えれば、「ルール = 変換履歴」が成り立つ。すなわち、変換履歴がプログラミングに関する知識を表している。よって対話的な変換により、問題固有であるがある程度一般的な性質にも対応できるさまざまなルールを作ることができる。

4. 変換履歴の再利用

4.1. 仕様と変換履歴の類似

プログラム1 ≅ プログラム2であるとする（図1）。プログラム1と2の構造は、それぞれの仕様を変換履歴によって形式的に変換した結果である。よって、変換履歴を逆にたどれば仕様1と仕様2が類似していて、変換履歴1と変換履歴2も類似していると考えることができる。

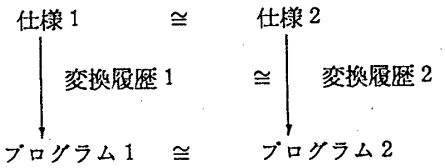

図1 仕様と変換履歴の類似

従って、変換履歴1を一般化し仕様2に適用することによりプログラム2が生成できる [2]（図2）。一般化の方法については4.2.で述べる。

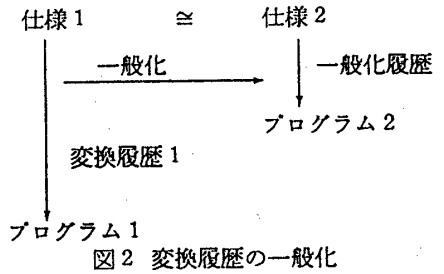

図2 変換履歴の一般化

4.2. 変換履歴の一般化

一般化された変換履歴は、図2の変換履歴1を部分的に縮約あるいは抽象化したもので、履歴のスキーマとなる。変換履歴の一般化は、以下の2つのプロセスで行なわれる。

1. 仕様と変換履歴の抽象化

   仕様のどの関数や引数を抽象化するかを決定する。これは対話的に行なわれる。次に変換履歴の関数や

Reuse of program transformation histories
Kenji Kobori , Yasuaki Nishitani
Gunma University

引数を仕様で抽象化したもので置き換える。抽象化したものに依存したルールの変換により新しい式が出現しているならば抽象化する。大文字はパラメータ、小文字はインスタンスを表す。

例 仕様の抽象化

```
  (exist x l) = (filter λ i.(= x i) l)
=>
  (F x L)     = (filter λ i.(P x i) L)
```

2. 履歴の縮約

これは変換履歴をルール化するプロセスである。一般的な部分と特殊な部分で処理が異なる。特殊な部分とは、変換履歴の中で抽象化した部分を用いて新しい関数が出現する変換である。一般的な部分は特殊的でない部分をいう。

(a) 一般的な部分

一般的な部分の変換の始まりから終りに変換するようなルールにする。

例

```
(F x L)
  ↓ 定義で展開
(filter λ j.(P x j) L)
  ↓ Lの分解
(filter λ j.(P x j) (append L1 L2))
   where L = (append L1 L2)
  ↓ filterのルールの適用
(append (filter λ j.(P x j) L1)
        (filter λ j.(P x j) L2))
```

縮約 =>

```
(F x L)
  ↓
(append (filter λ j.(P x j) L1)
        (filter λ j.(P x j) L2))
   where L = (append L1 L2)
```

(b) 特殊な部分

新しい抽象化部分が出現する変換とそれを使った変換は残し、それ以外は削除する。

例

```
(not (P x (car L)))
  |ルールの適用
  ↓ (P a b) => (and (P1 a b)(P2 a b))
(not (and (P1 x (car L))
          (P2 x (car L))))
  ↓ ;; P1 と P2 が新しい出現
(or (not (P1 x (car L)))
    (not (P2 x (car L))))
  ↓
(if (not (P1 x (car L))) t
    (not (P2 x (car L))))
```

短縮 =>

```
if
  ∃ P1,P2( (P x (car L))  =
  (and (P1 x (car L))(P2 x (car L))))
then
  (not (P x (car L)))
    ↓
  (if (not (P1 x (car L))) t
      (not (P2 x (car L))))
```

4.3. 一般化履歴の適用

一般化された仕様と適用される仕様が類似していれば、一般化履歴を類似した仕様に適用できる。一般化履歴の適用は以下のように行なう。

1. 仕様のマッチングをとる。
2. 適用される仕様が、一般化された仕様の性質を満たしているか調べる。
3. 関数の引数がマッチするか調べる。マッチしない場合、一般化された仕様とマッチするように補助関数を導入する。
4. 一般化履歴を適用する。特殊部分は、適用可能ならば適用する。

5. 再利用の結果

5.1 再利用の例

リスト l の中にある要素 x が存在するとき l に含まれるすべての x を返し、存在しないとき nil を返す関数 (exist x l) を以下のように定義する。

```
(exist x l) = (filter λ i.(= x i) l)
```

l がソートされたものを常にとるならば、この関数を二分探索法を用いて定義できる。対話的に変換を行ない二分探索法によるプログラムを得、その変換履歴を一般化する。

次に、平方根を求める関数に一般化した変換履歴を適用する。定義を以下に示す。

```
(sqrt n)
   =
(filter λ i.(and (<= (* i i) n)
                 (< n (* (1+ i)(1+ i))))
         (mk-int 1 n))
```

適用の結果、二分探索法を用いたプログラムを得た。

5.2. 応用例

```
(sort (append '(4 6 2 8 5) l))
          l は数字の要素からなるリスト
```

という式を普通に部分計算すると

```
(insert 4 (insert 6 ( ... (insert 5 l)))))
```

となり効率化できない。しかし、(sort l) をマージソートに変換する変換履歴を一般化したものをルールとして利用することにより

```
(sort (append '(2 4 5 6 8) l))
```

と変換できる。

6. まとめ

本研究では、変換履歴を一般化することにより高レベルの知識を用いたプログラム変換が可能であることを提案した。これを類似した仕様に適用し、問題固有の性質を用いた変換を行なった。また、一般化した変換履歴をルールとして部分計算に利用することにより効率的なプログラムが生成できた。

参考文献

[1] Simon Thompson: "Executable Specification and Program Transformation", FIFTH International workshop on software specification and design 5.1989.
[2] Allen Goklberg: "Reusing Software Developments", Proceeding of the Forth ACM SIGSOFT Symposium on Software Development Environment 1990
[3] R. バード,P. ワドラー "Functional Programming", 近代科学社, 1991