完備化手続きを用いたプログラム合成の研究

吉政　洋美

北陸先端科学技術大学院大学 情報科学研究科

1999 年 2 月 15 日

キーワード： 切り払い手法, 融合規則, 項書き換え系, Knuth-Bendix 完備化手続き.

P.Wadler によって提案された切り払い手法(1990) は、関数型プログラミング言語において、畳み込み-合成規則（The foldr/build rule）を用いることによって不必要であると思われるデータ構造を削除する手法である。関数型言語のプログラム変換法として融合変換を用いた切り払い手法が有効であることが知られている。F.Bellegarde(1995) では、項書換え系で切り払い手法のための融合規則 (Fusion rule) が表現できることを示している。ここでの融合ルールは、定義された書き換え規則の右辺の中での融合項 (Fusion term) が構成子に基づいた系 R の実行によって見つけられることにより、構成子に基づいた系 R_h が統合されるというものである。これは、ある書き換え規則を関数型プログラミング言語において定義し、それに対して合成規則を付け加えることによって、融合規則 $s \rightarrow h(x_1, x_2, \cdots, x_n)$ が生成されるシステムである。

F.Bellegarde の切り払い手法によるプログラム自動生成は、完備化手続きのアルゴリズムより導出される。新しい記号 h を用いて、$h(x_1, x_2, ..., x_n)$ を左辺におき、合成させたい融合項 t に対して、等式 $E : h(x_1, ..., x_n) = t$ を作る。そして、プログラム自動生成によって、E の等式を書き換え規則 $R : t \rightarrow h(x_1, ..., x_n)$ を生成させる手続きをとる。本研究は、Knuth-Bendix 完備化手続きを用いて等式の左辺と右辺が入れ替わった形の書き換え規則と合成された書き換え規則の集合が得られるような E の等式と順序付けについての例を収集した。合成された書き換え規則が得られるような成功例には、融合規則 (Fusion rule) が適用される。項書換え系に対する融合変換が完備化手続きに基づいて実現できる

ことを示している。プログラム合成手順を示すと、以下の図のようになる。

R : { 項書換え規則 }

E : $h(x) = t$

⇓ 順序付け

融合ルール R_h : $t \to h(x)$

$R \cup R_h$

⇓ 完備化

$$\begin{cases} h([]) \to t_1 \\ h(x :: xs) \to t_2 \end{cases}$$

Knuth-Bendix 完備化手続きのシステムは、関数型プログラミング言語 Standard ML of New Jersey (1993) を用いて作成し、実験は、主に Dell 社のノート型パソコン Inspiron 3000 で行なった。Knuth-Bendix 完備化手続きへの拡張によって F.Bellegarde による融合規則にある制限が取り外せることが本研究でわかった。例えば、F.Bellegarde によるプログラム合成では、融合規則 R_h の融合項 t が線形でなければならないという制限が課せられるが、実験により、$x + x \to d(x)$ での $x + x$ のような非線形のものでもうまく合成されるということがわかった。Knuth-Bendix 完備化手続きによる切り払い手法を用いたプログラム合成は、主に 長さを表す関数の集合 R_{length} 、和を表す関数の集合 R_+ 、積を表す関数の集合 $R_\times$ のような再帰的プログラムの集合に対して、あらゆる関数を新しい記号 (Fresh symbol) を用いることによって、等式 E を作り、合成させた形のものを Knuth-Bendix 完備化手続きにより、出力させて実験を行なった。

これは、関数記号を辞書式経路順序付けにより、完備化させるものである。この実験を行なった際、順序付けの設定によって、完備化せずに停止しないものや完備化する前に失敗してしまうケースが多く観察される。そこで、どうすれば完備化を成功させるかその法則性を発見するために 46 通りの入力データに対して、順序付けを変えることによって、731 通りのプログラム合成を Knuth-Bendix 完備化手続きのシステムを用いて実験を行なった。そこで、121 通りの入力データに対して完備化手続きの出力が成功し、それぞれの出力データの中に生成規則が得られた。その生成規則などを本論文において、報告する。

プログラム合成の実験の結果、順序付けに着目して考察してみると、新しく導入する関数記号 (Fresh symbol) は、一番大きく持ってくると成功しやすいことがわかる。そして、完備化が成功する例を見てみると、順序付けによって、完備化するまでのステップ数が異なることもある。同じプログラム合成であっても、順序付けが異なると完備化される途中で失敗して停止したり、発散して停止しなくなったりすることが多い。このようにして、順序付けがプログラム合成の完備化に対して、大きく左右していることがよくわかる。プログラム合成を成功させるためには、順序付けは、注意深く設定する必要がある。

プログラム合成の一例を示す。等式 $E : h(x, y, z) = length((x@y) :: z)$ を完備化により、$R_h : length((x@y) :: z) \to h(x, y, z)$ を生成させて、$R_+ \cup R_{length} \cup R_@ \cup R_h$ のプログラム合成を試みる。すると、$h > length > @ > + > [] > :: > 0 > s$ の順序付けでのみ 10 ステップで完備化ができる。そして、次のような生成規則が出力される。

- $h(x :: y, z, w) \to length((x :: (y@z)) + w)$
- $h(x, y, z) \to length(s(x@y) + z)$
- $h(x, y, 0) \to length(x@y)$

このような実験の結果を、本論文で 27 種類の成功例を紹介する。そして、成功例ばかりではなく、失敗例についても触れる。