知能と複雑系 117－2
(1999. 7. 22)

等価変換ルールの探索に基づくプログラム合成

小池 英勝† 赤間 清†† 宮本 衛市†

† 北海道大学大学院 工学研究科 システム情報工学専攻
札幌市北区北 13 条西 8 丁目. Tel. 011-706-6814

†† 北海道大学 情報メディア教育研究総合センター
札幌市北区北 11 条西 5 丁目. Tel. 011-706-6814

{koike,akama,miya}@complex.eng.hokudai.ac.jp

等価変換による問題解決では，等価変換ルールの集合で問題解決の手続きを記述する．等価変換ルールを用いると，正当で効率的な問題解決が可能である．論理プログラミングでは，関係を正しく記述しても解を得られない場合がある (reverse の例題など)．そのような問題に対してもこの方法では正しく解を与えることができる．これまでに，確定節の集合から多数の正当な等価変換ルールの集合を生成するための基礎理論が提案されている．本論文では，等価変換ルールの集合から探索によって，問題を効率的に解決するための等価変換ルールを発見する方法を提案する．この方法は，正当で効率的なプログラムを自動生成するための基礎を与える．

ルール　プログラム合成　等価変換　自動生成

Program Synthesis Based on Searching for Equivalent Transformation Rules

Hidekatsu Koike† Kiyoshi Akama†† Eiichi Miyamoto†

† Division of System and Information Engineering, Hokkaido University
North 13 West 8 Kita-ku Sapporo. Tel. 011-706-6814

†† Center of Information and Multimedia Studies, Hokkaido University
North 11 West 5 Kita-ku Sapporo. Tel. 011-706-6814

{koike,akama,miya}@complex.eng.hokudai.ac.jp

In problem solving based on equivalent transformation(ET), a procedure is represented by a set of ET rules, which enables us to sove problems correctly and efficiently. There exist some problems that can be solved correctly by the ET paradigm, but cannot be solved by natural logic programs, though they define problems correctly. A theoretical foundation for generating a large class of correct ET rules has been proposed. In this paper we develop a method of searching the class of rules for efficient ET rules. This method lays a foundation of synthesizing correct and efficient programs.

rule, program synthesis, equivalent transformation, automatic generation

1 はじめに

人にとってわかり易い記述から，効率的なプログラムを自動的に合成できれば，ソフトウェア構築における人の負担を大きく減らすことができる．

本研究では，等価変換による問題解決 [4] を基礎として，プログラム合成システムを構築する．このシステムは，問題を宣言的に記述した確定節の集合を入力として，そこからプログラムを自動合成する．等価変換による問題解決では，等価変換ルールの集合がプログラムである．等価変換ルールは，問題記述の宣言的意味を保存したまま変換するルールである．等価変換の枠組みを用いる利点を以下に示す．

- 高いアルゴリズム記述能力がある
- 個々の等価変換ルールを独立したプログラムの部品と捉えることができる

効率的なプログラムを合成するには，それを記述するものに高いアルゴリズム記述能力が必要である．この能力が不充分であれば，合成可能なプログラムに始めから大きな制限が付いてしまう．

プログラム合成を行うためには，プログラムが小さな部品に分解でき，その部品の正当性は他の影響を受けないことが重要である．プログラムの部品を 1 つずつ作って全体を構築していくことが出来れば，複雑なプログラム合成の問題を単純で小さな問題に分解できる．また，部品の追加による全体の正当性への影響が無ければ，副作用を考慮すること無しに次々に追加していくことによって全体を構築できる．プログラムを等価変換の集合と捉えることでこれらのことが実現できる [6].

本枠組ではルールの追加が自由に行えるので，比較的容易に自動生成可能なものはシステムに任せ，少数の自動生成が困難なルールを人が追加することができる．つまり，システムと人の分業を容易に行うことのできる構造を持っている．したがって，この枠組は，実用的なシステムを他の枠組よりも容易に構築できると考えられる．

ルール生成は，メタ問題記述とメタルールを用いて行う．このルール生成法では得られる等価変換ルールが多数存在するので，その中から効率の良いルールを探索する．

本プログラム合成システムは，問題を正しく解くプログラムの合成だけではなく，合成されるプログラムの効率化という側面も持っている．本プログラム合成の枠組では，この 2 つの側面を継ぎ目無く捉えることが出来る．

以降の節で，探索に基づくプログラムの合成法を，実際に構築したシステムと共に説明する．

2 プログラム合成システムの概要

はじめに，システムがどのようにプログラムを合成するかについて，概要と基本要素について述べる．

2.1 入力と出力

本システムは，基本的に問題記述を入力とし，等価変換ルールの集合を出力する．問題記述は，関係を表す確定節の集合と質問を表す節からなる．問題記述の具体例を以下に示す．

問題記述($p1$)

確定節集合

$reverse([X|Y], Z)$
$\quad \leftarrow reverse(Y, R), append(R, [X], Z).$

$reverse([\,], [\,]) \leftarrow .$

$append([\,], X, X) \leftarrow .$

$append([A|X], Y, [A|Z]) \leftarrow append(X, Y, Z).$

質問

$ans(X) \leftarrow reverse(X, [1, 2]).$

$reverse$ は第 1 引数のリストと第 2 引数のリストの要素の順番が逆になる関係を表す．$append$ は第 1 引数と第 2 引数を結合したものが第 3 引数であるという関係を表す．質問の節は，リスト $[1, 2]$ を逆にしたものは何かという質問を表し，答えが変数 X に入る．

この問題記述を与えると，プログラム合成システムは以下のような等価変換ルールの集合を生成する．

$(r1)$ $reverse(\&X,\ [\&A|\&Y])$
$\quad \rightarrow \{\&X = [\#B|\#W]\},$
$\quad\quad reverse(\#W, \#Z),$
$\quad\quad append(\#Z, [\&B], [\&A|\&Y]).$

$(r2)$ $reverse(\&X,\ [\,]) \rightarrow \{\&X = [\,]\}.$

$(r3)$ $append(\&A, [\&B], [\&C, \&D|\&E])$
$\quad \rightarrow \{\&A = [\&C|\#F]\},$
$\quad\quad append(\#F, [\&B], [\&D|\&E]).$

$(r4)$ $append(\&A, [\&B|\&C], [\&D])$
$\quad \rightarrow \{\&A = [\,], \&C = [\,], \&B = \&D\}.$

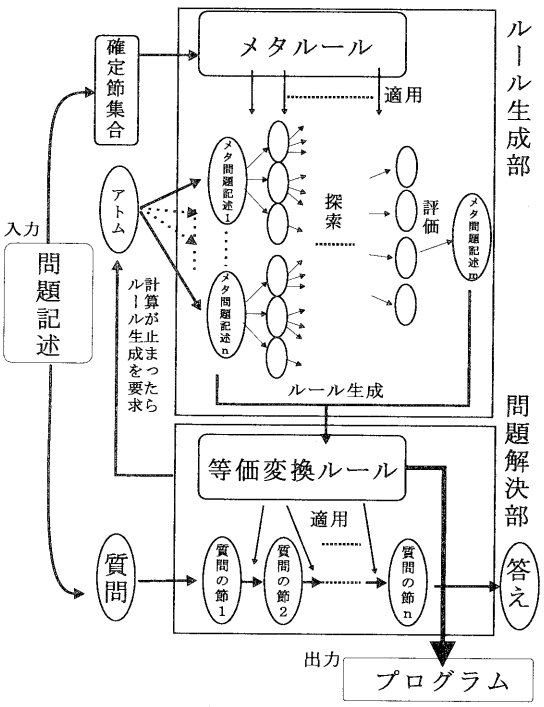
このルールは、入力として与えられた質問とそれに類似した質問に解を与えることができる．また，単に問題を解けるだけではなく効率的に問題が解けるルールが出力されている．システムは，問題を表す関係と具体的な質問を入力とすることによって，その質問を解くために特化した効率的なルールを生成する．更に，生成されたルールは与えられた問題だけではなく類似した多数の問題を解くことが出来る．

2.2 論理プログラミングの問題点

問題記述 ($p1$) の確定節集合は，論理プログラムでもあり正しく関係が記述されている．実際論理の枠組で，質問 $reverse([1,2],X)$ に対して正しく解を求めることが出来る．しかし，質問 $reverse(X,[1,2])$ を与えても無限ループになり解を得ることは出来ない．質問の意味としては同じにもかかわらず，この質問を解くためには確定節を書き換えなければならない．本システムでは，確定節集合はそのままで質問を変えても，そこから生成した等価変換ルールで問題解決を行うので，正しく解を求めることが出来る．

2.3 プログラム合成の要素

システムの構成を $Fig.1$ に図示する．



$Fig.1$　システムの構成

本プログラム合成法は，大きく分けて問題解決とルール生成の 2 つから成る．以下に，それぞれの構成要素を列挙する．

問題解決

- **問題記述** 関係を宣言的に記述した確定節の集合と，質問を表す節の集合から成る．
- **等価変換ルール** 本システムの出力で，等価変換ルールの集合がプログラムとなる．
- **変換** 等価変換ルールで問題記述を変換することにより，質問に対する答えを計算する．

ルール生成

- **メタ問題記述** 問題記述を抽象的に表現したもの．
- **メタルール** メタ問題記述を等価変換するためのルール．
- **探索** 生成可能なルールの集合から効率的なルールを探索する．
- **ルールの評価** 探索で個々のルールを評価する．

2.4 等価変換ルール生成法の概要

システムは問題記述が与えられると，始めにメタルールを作る．次に問題解決部は質問の節を等価変換ルールで変換して，質問の解を求めるための計算を行う．計算が終了していないのに必要な等価変換ルールが無い場合，ルール生成部で新しい等価変換ルールを生成する．システムは，問題解決部の質問の節からアトムを選択する．そのアトムからメタ問題記述を作り，それをメタルールで変換していく．そしてメタ問題記述の変換列から等価変換ルールを作る．一般に生成可能なルールは複数あるので，その中から効率的なルールを探索する．等価変換ルールが生成されると，問題解決部に処理が戻り計算が継続される．この一連の流れを計算が終了するまで繰り返す．問題解決部の計算が終了したということは，必要な等価変換ルールは全て生成できたということなのでプログラム合成は終了する．

3 問題解決と等価変換ルール

本節では，等価変換ルールとそれを用いた問題解決について述べる．

3.1 等価変換ルール

2 つの集合 R_P，R_R が与えられているとする．R_P の中の要素は宣言的記述のアトムにマッチするメタアトムとして用いられる．R_R 中の要素は実行メタアトムとして用いられる．R_R と R_P の具体例は以下のようなものである．

$R_P = \{ans, primes, filter, initial, append, \ldots\}$
$R_R = \{=, >, \leq, divisible, \ldots\}$

メタアトムとは，変数に & 変数と # 変数が現れるアトムである．また，このような変数が現れる項をメタ項と呼ぶ．メタアトムの変数に通常の変数や項を代入することで，通常のアトムが得られる．先頭に&のついた変数を&変数と呼び，任意の項を代入できる．先頭に#がついた変数を#変数と呼び通常の変数を代入できる．これらの変数の詳細は [3] にある．等価変換ルールには通常の変数は表れない．

R_P と R_R 上の等価変換ルールとは以下の形のものである．

$\langle rulename \rangle$:
$H, \{C_S\} \rightarrow \{E_{S_1}\}, B_{S_1};$
$\rightarrow \{E_{S_2}\}, B_{S_2};$
$\cdots$
$\rightarrow \{E_{S_n}\}, B_{S_n}.$
ただし $(n \geq 1)$

ここで，$\langle rulename \rangle$ はルール名である．H は，R_P 中のメタアトム，C_s と E_{S_i} は，空列又は R_R 中のメタアトムの列，B_{S_i} は空列又は R_P 中のメタアトムの列である．H をヘッド，C_S を適用条件，E_{S_i} を実行部，E_{S_i} と B_{S_i} の組をボディと呼ぶ．ルール名，適用条件，実行部は省略可能である．

等価変換ルールは適用の際，代入 θ によってメタアトムが通常のアトムに具体化される．θ は，& 変数への項の代入と，# 変数への通常の変数の代入からなる．ボディにアトム b を持った確定節 C を考える．等価変換ルールが C のアトム b に適用可能であるとは，ヘッド H が代入 θ によってアトム b にマッチ $(H\theta = b)$ して，$C_S\theta$ が真のときで，かつ，そのときに限る．

アトム b にルールが適用された節 C は，n 個以下の節に変換される．$E_{S_i}\theta$ の実行が成功した節だけが残り，$E_{S_i}\theta$ の実行で発生した代入 σ を用いて，節 $C\sigma$ の $b\sigma$ は $B_{S_i}\theta\sigma$ で置き換えられる．

等価変換ルールの具体例は，以下のようなものである．

$(r1)$ $reverse(\&X, [\&A|\&Y])$
$\rightarrow \{\&X = [\#B|\#W]\},$
$reverse(\#W, \#Z),$
$append(\#Z, [\#B], [\&A|\&Y]).$

3.2 問題解決

等価変換パラダイムでは，等価変換ルールで問題記述の質問を表す節を変換していくことで解を求める．

たとえば，2.1 節の問題記述の質問の節は，同節の等価変換ルール $(r1) \sim (r4)$ を用いて以下のように変換される．

1) $ans(X) \leftarrow reverse(X, [1,2]).$
2) $ans([B1|W1]) \leftarrow reverse(W1, Z1),$
$append(Z1, [B1], [1,2]).$
$(r1)$ より
3) $ans([B1|W1]) \leftarrow reverse(W1, [1|F1]),$
$append(F1, [B1], [2]).$
$(r3)$ より
4) $ans([2|W1]) \leftarrow reverse(W1, [1]).$
$(r4)$ より
5) $ans([2, B2|W2]) \leftarrow reverse(W2, Z2),$
$append(Z2, [B2], [1]).$
$(r2)$ より
6) $ans([2, 1|W2]) \leftarrow reverse(W2, []).$
$(r4)$ より
7) $ans([2, 1]) \leftarrow .$
$(r2)$ より

ここでは，1) から 7) の 7 ステップの変換で解が得らている．変換の過程について説明する．等価変換ルール $r1$ によって 1) から 2) に変換される．$r3$ によって 2) から 3) へ変換される．$r4$ によって 3) から 4) へ変換される．以下同様に 5) から 7) まで変換される．7) の節で解が $[2,1]$ であることを表している．

4 ルール生成の基礎

本節では，ルール生成の基本要素と，ルール生成法の概要について述べる．

4.1 メタ問題記述

メタ問題記述とは，問題記述を抽象的に表したものである．1 つのメタ問題記述は多数の問題記述を代表する．メタ問題記述はメタ節の集合である．メタ節は，ダミーヘッドとメタアトムからなる．ダミーヘッドは，任意のヘッドを表す．メタ節，メタアトムはそれぞれ，節，アトムを抽象的に表す．メタ節の具体例は以下のようなものである．

(mc) $h \leftarrow reverse(\&A, [1,2]).$

このメタ節は，第 1 引数が任意の項で，第 2 引数が $[1,2]$ であるような $reverse$ アトムがボディに現れる全ての節を代表している．ここで，ダミーヘッド h は任意のヘッドを表す．上のメタ節 (mc) だけからなるメタ問題記述は，(mc) が表す節を 1 つでも持つ全ての問題記述を表す．このメタ問題記述は，2.1 節の問題記述 $(p1)$ を含む多数の問題記述を表す．メタ問題記述はメタルールに

よって等価変換される．メタ問題記述を他のメタ問題記述へ等価変換することは、そのメタ問題記述が表す多数の問題記述の等価変換を一度に行うことに等しい．

4.2 メタルール

メタルールは，以下のような形をしている．

$\langle rulename \rangle$:
$H_1, \ldots, H_m, \{C_S\} \rightarrow \{E_{S_1}\}, B_{S_1};$
$\rightarrow \{E_{S_2}\}, B_{S_2};$
$\ldots$
$\rightarrow \{E_{S_n}\}, B_{S_n}.$
ただし $(m \geq 1, n \geq 1)$

ルール名，適用条件，実行部は省略可能である．等価変換ルールと似た形をしているが，変換対象がメタ問題記述であること，現われる変数が $*$ 変数と % 変数であること，ヘッドに複数のアトムを許すことが異なる．

$*$ 変数とは先頭に $*$ のついた変数で，任意のメタ項が代入できる．% 変数とは，先頭に % のついた変数で#変数が代入できる．

メタルールは適用の際，代入 θ_{meta} によって $*$ 変数と % 変数を持つアトムが，メタアトムに具体化される．θ_{meta} は，$*$ 変数へのメタ項の代入と，% 変数への # 変数の代入からなる．メタルールがメタ節 C に適用可能であるとは，C がボディにメタアトム $b_1, \ldots, b_m$ を含み，全てのヘッド H_h が代入 θ_{meta} によって異なるアトム b_i にそれぞれマッチ $(H_h\theta_{meta} = b_i)$ して，$C_S\theta_{meta}$ が真のときで，かつ，そのときに限る．節のボディの書き換えは等価変換ルールと同様に行われる．

メタルールは，確定節から平坦化と呼ばれる機械的な書き換えを用いて得られるルールと，あらかじめシステムに用意しておくルールに分類できる．

あらかじめ用意しておくルールには，等式制約を処理するものや公式を表すものなどがある．メタルールの具体例としては以下のようなものがある．

$equal([*A|*X],[*B|*Y]),\{*A == *B\}$
$\rightarrow equal(*X,*Y).$

$\{*A == *B\}$ はルール適用条件で，$*A$ と $*B$ が等しいとき真になる．

4.3 問題記述からのメタルールの作成

メタルールは，与えられた問題記述から機械的な書き換えで作ることが出来る．具体例で説明する．2.1 節の問題記述 $(p1)$ が与えられたとする．この節の，確定節集合を平坦化する．平坦化とは以下のような節の書き換えの操作である．

- 節のヘッドの引数を全て，名前の異なる変数にする．
- ヘッドに現れた引数の形を *equal* 述語を使って表す．

$(p1)$ の確定節集合は以下のように書き換えられる．

$(c1)$ $reverse(X,Z) \leftarrow equal(X,[Xs|Y]),$
$reverse(Y,R), append(R,[Xs],Z).$
$(c2)$ $reverse(X,Z) \leftarrow equal(X,[]), equal(Z,[]).$
$(c3)$ $append(X,Y,Z) \leftarrow equal(X,[]), equal(Y,Z).$
$(c4)$ $append(X,Y,Z) \leftarrow equal(X,[A|Xs]),$
$equal(Z,[A|Xs]), append(Xs,Y,Zs).$

そして，この確定節からメタルールを作る．始めに，それぞれの確定節のヘッドに注目して同じものに分ける．ヘッドが等しいとは，ヘッドアトムの名前が一致して更に引数の個数が一致しているときである．変数名が違う場合は，それぞれ対応する変数名が一致するように付け直す．上の確定節集合から得られるメタルールを以下に示す．

$(m1)$ $reverse(*X,*Z)$
$\rightarrow equal(*X,[\%Xs|\%Y]),$
$reverse(\%Y,\%R),$
$append(\%R,[\%Xs],*Z);$
$\rightarrow equal(*X,[]), equal(*Z,[]).$
$(m2)$ $append(*X,*Y,*Z)$
$\rightarrow equal(*X,[]), equal(*Y,*Z);$
$\rightarrow equal(*X,[\%A|\%Xs]),$
$equal(*Z,[\%A|\%Zs]),$
$append(\%Xs,*Y,\%Zs).$

$m1$ は，$c1$ と $c2$ から作られる．$m2$ は，$c3$，$c4$ から作られる．このように同じヘッドを持つ節が複数個存在する場合は，それらの節から複数のボディを持つルールを 1 つ作る．

4.4 メタ問題記述の変換

具体例で説明する．次のような 1 つのメタ節からなるメタ問題記述が与えられたとする．

$h \leftarrow reverse(\&A,[]).$

そして，メタルールで以下のようにに変換していく．

1) $h \leftarrow reverse(\&A,[]).$
2) $h \leftarrow equal(\&A,[\#X|\#X1]),$
$reverse(\#X1,\#R), append(\#R,[\#X],[]).$
$h \leftarrow equal(\&A,[]), equal([],[]).$

$(m1)$ より

3) $h \leftarrow equal(\&A, [\#X|\#X1]),$
$reverse(\#X1, \#R),$
$equal(\#R, []), equal([\#X], []).$

$h \leftarrow equal(\&A, [\#X|\#X1]),$
$reverse(\#X1, \#R),$
$equal(\#R, [\#A, \#X2]), equal([], [\#A, \#Zs]),$
$append(\#X2, [\#X], \#Z).$

$h \leftarrow equal(\&A, []), equal([], []).$

$(m2)$ より

4) $h \leftarrow equal(\&A, []).$

(等式制約に関するメタルール) より

3) から，4) への変換には等式制約に関するメタルールが適用される．このメタルールはあらかじめシステムに用意しておく．

ここで示した変換の列は，多数の中の一例に過ぎない．例えば，2) には，再び $(m1)$ を適用することが可能である．このようにメタ節 (メタ問題記述) の変換は一般に非決定的である．

4.5 等価変換ルールの生成

4.4 節の 1) から 4) への変換の列はどの段階も等価変換により得られたものである．よって，1) から 4) への直接の書き換えも等価変換であるので，1) と 4) から新しい等価変換ルールを作ることが出来る．このようにして 2.1 節の $r2$ が得られる．

5 問題解決部

本節では，実際に具体的な問題を計算しながら，どのようなルールを生成するかを決定する問題解決部について説明する．

5.1 変換の制御

問題解決部は，与えられた問題記述を等価変換ルールを用いてどのように変換するかを制御する．変換の制御は，

- 変換対象の節の選択
- 節の対象アトムの選択
- 適用ルールの決定

から成る．これらの項目は全て非決定性を含むため，3 つが全て決まって，はじめて制御が一意に定まる．

変換対象選択の条件としては，

(1) 以前に選択していないものを選ぶ．

(2) 変換されてから時間のたったものを選ぶ．

この条件は，節，アトムの選択どちらにも適用され，必ず全ての節の全てのアトムが変換されることを保証する．このことによってある特定のアトムが展開されつづけることによる無限ループを防いでいる．変換によって新しく生成されたアトムは，一度選択されていると考える．

5.2 ルール生成の要求

問題解決部は，適用可能な等価変換ルールが無くなった時点で，計算途中の質問の節からボディアトムを選びそれをルール生成部に渡す．アトムを選択することで，そのアトムを持つ節に適用可能なルールの生成を要求する．アトムの選択肢は，通常複数ある．そのために，選択は以下のような条件で行う．

- リストを引数に取るボディアトムを選択する．
- 現れる変数の数が少ないアトムを選択する．

6 ルール生成部

本節では，システムのルール生成部が行う等価変換ルールの生成について述べる．

6.1 メタ問題記述の作成

ルール生成部は，ルール生成の始めに問題解決部から与えられたアトムから，メタ問題記述を作る．

メタ問題記述は以下の手順で作る．

(1) 与えられたアトムを表す全てのメタアトムを列挙する．

(2) 列挙されたメタアトム 1 つずつからメタ節を作る．

ここで作られるメタ問題記述は，1 つのメタ節からなる．つまり作られたメタ節と同じ数のメタ問題記述が作られる．もしアトム $reverse(X, [1, 2])$ が与えられたとすると，

$h \leftarrow reverse(\&X, [1, 2]).$
$h \leftarrow reverse(\&X, [\&Y, 2]).$
$h \leftarrow reverse(\&X, [1, \&Y]).$
$h \leftarrow reverse(\&X, [\&Y, \&Z]).$
$h \leftarrow reverse(\&X, [\&Y|\&Z]).$
$h \leftarrow reverse(\&X, \&Y).$

の6つのメタ節が作られる．下の節ほど一般的な形をしている．

どのメタ節からも，正しい等価変換ルールが得られるが，与えたメタ節のボディが得られるルールのヘッドになるので，メタアトムの引数に変数のみが現れるものからは，一般的的だが非効率なルールが生成されることがある．また，あまりに特殊過ぎると殆ど適用されないルールが生成されることがある．よって，理論的には全てのメタ節でルールを生成して，それらを比較して最も効率的なルールを選択する方法をとる．

しかし，このように複数のメタ節からルールを作って比較する方法は処理に時間がかかる．本システムの実際の実装では，ダウンマッチングと呼ばれる方法を導入している．この方法は1回のルール生成で全てのルールを生成して比較したのと等しい効果を得られる．ダウンマッチングについての詳細は別の論文で議論する．

6.2 メタ問題記述の変換の制御

メタ問題記述の変換は，プログラム合成の核をなす部分である．ルール生成部は多数の可能なメタ問題記述の変換を試みる．具体的には，メタ問題記述に適用可能なメタルールが同時に複数存在する場合は，それぞれの適用を全て試すことになる．しかし，実際のシステムで全てを試すことは計算量の増大で不可能な場合がある．よって，変換回数の上限の設定と探索木の枝刈りによって有限時間でルール生成が終了するようにする．

6.3 探索木の枝刈り

メタ問題記述の変換列が生成される過程には非決定性を含み，全ての変換過程のパスの探索を行うと計算爆発を起こしてしまう場合がある．よって，探索木の枝を適切に刈ることが必要になる．しかし，闇雲に枝を刈ると効率的な等価変換ルール生成の機会を逃してしまう．そのため，なるべく効率化に関係の無い探索の枝を刈るようにしたい．本システムでの枝刈りの基準を以下に示す．

(1) 変換中に許すアトムの数の最大値
(2) 互いに逆変換にあるルールの適用
(3) 探索中に現れた条件
(4) 重複する探索木の削除

(1) は，メタ問題記述が持つメタアトムの総数が，ある定めた数より多くなった場合，そのサーチパスを破棄することを意味する．(2) は，互いに逆変換の関係にあるメタルールの連続した適用を禁止する．(3) は，ある変換で発生した条件を後の変換で使うことで探索の分岐を減らす．たとえば，ある変数が奇数と偶数の可能性があり，探索のパスがそれぞれの場合に分岐したとする．このとき，分岐以降は変数にそれぞれ奇数，偶数であるという条件を付けることが出来る．よって，その分岐以降は，その変数に関する分岐を避けることが出来る．(4) は，多数の重複する探索のパスを1つにまとめることで，余分な探索を減らす．

6.4 ルールの評価

多数のルールの中から効率的なルールを選択するには，ルールの効率を判断するための基準が必要である．

以下の基準で評価する．

1. 分岐 (ボディ) の数
2. 自己再帰性
3. 展開されるアトムの数
4. マッチするパターン (ヘッドの形) の一般的性
5. 無限ループ可能性

これらの基準を評価関数 F_e で定式化する．

$$F_e(x) = \{W_{br} \times Br(x) + W_{si} \times Si(x) + W_p \times P(x)\} \times Loop(x) + user(x)$$

x には評価するルールが1つ入る．ここで，$Br(x)$，$Si(x)$，$P(x)$，$Loop(x)$ はそれぞれ分岐数，自己再帰を考慮したアトムの数え上げ，パターンの評価，ループの有無を整数の返り値とする関数である．また，W_x はそれぞれの関数の重みである．

関数 $user(x)$ で，ユーザーが，特定のアトムを含まない等価変換ルールを候補のルール中で高い評価になるように指定できる．指定には，アトムの名前とそれを含んだときにペナルティーとなる点数の2項組みを指定する．

7 等価変換ルール生成の具体例

本システムは問題記述 ($p1$) を入力として与えると，等価変換ルール ($r1$) ～ ($r4$) を出力する．また，このルールの集合は正しく解を与える．

ここで，問題記述 ($p1$) の確定節集合は同じで，質問の節を

$$ans(X) \leftarrow reverse([1,2], X).$$

に変えた問題記述 ($p2$) を考える．この質問の節は $reverse$ の引数である X とリストの位置が $p1$ と逆になっている．等価変換ルール ($r1$) ～ ($r4$) は，いずれもヘッドにマッチしないのでこの節に適用できない．問題記述 ($p2$) を

システムに与えると次のような等価変換ルールの集合を出力する．

$(r5)$ $reverse([\&A|\&X], \&Y) \rightarrow$
$reverse(\&X, \#W)$,
$append(\#W, [\&A], \&Y)$.
$(r6)$ $reverse([], \&X) \rightarrow \{\&X = []\}$.
$(r7)$ $append([], \&Y, \&Z)$
$\rightarrow \{\&Y = \&Z\}$.
$(r8)$ $append([\&A], \&Y, \&Z)$
$\rightarrow \{\&Z = [\&A|\&Y]\}$.

このルールは問題記述 $(p2)$ の質問に正しく解を与える．

更に，確定節集合は同じで質問の節が

$ans(X) \leftarrow reverse([1,2,3], X)$.

である問題記述 $(p3)$ を考える．この問題記述をシステムに与えると，等価変換ルール $(r5) \sim (r8)$ に加えて，

$(r9)$ $append([\&A|\&X], \&Y, \&Z)$
$\rightarrow \{\&Z = [\&A|\#W]\}$,
$append(\&X, \&Y, \#W)$.

を生成する．

システムから出力されたルール $(r1) \sim (r9)$ は，同じ確定節集合から生成されているので混合して用いても正当性は保証される．よってこれらのルールを混合することで与えた質問

$ans(X) \leftarrow reverse(X, [1,2])$.
$ans(X) \leftarrow reverse([1,2], X)$.
$ans(X) \leftarrow reverse([1,2,3], X)$.

の全てに解を与えることができる．更に，このルールの集合は，類似の質問

$ans(X) \leftarrow reverse([1], X)$.
$ans(X) \leftarrow reverse(X, [1,2,3])$.
$ans(X) \leftarrow reverse([1,2,X], [3,Y,1])$.

などにも解を与えることができる．

本システムは，与えられた具体的な問題を解くために必要なルールを生成する．このことによってその問題に特化した効率的なルールを生成できる．等価変換ルール $(r1) \sim (r4)$ は，$reverse$ の第 1 引数が変数である場合に特化している．$append$ に関するルール $(r3)$，$(r4)$ も $(r1)$ の適用によって生成される $append$ を変換することに特化している．$(r5) \sim (r8)$ は，$reverse$ の第 2 引数が変数の場合でかつ第 1 引数のリストの要素が 2 個以下の場合に特化している．$(r5) \sim (r8)$ に $(r9)$ を追加することによって，任意の個数のリストに対応できる．また，このとき $(r9)$ が適用可能な節の集合は，$(r8)$ が適用可能な節の集合を全て含むので，$(r8)$ を取り除くことが出来る．

8 終わりに

本研究では，効率的なプログラムの合成のために等価変換ルールを探索する方法を提案した．また，問題記述から効率的なプログラムを合成するシステムを構築した．本論文で示したシステムは，現在様々な例題を与え実行し，そこから新たな問題を発見してその問題に対応できるように改善している段階にある．今後は，問題記述に一階述語論理を導入し，更に適用可能な問題のクラスを拡大する予定である．

参考文献

[1] 淵　一博, 古川　康一, 溝口　文雄：プログラム変換, 共立出版 (1987)

[2] 小池英勝, 赤間清, 宮本衛市：仕様からの等価変換ルールの生成法, 電子情報通信学会技術研究報告 KBSE98-8, pp.33-40 (1998)

[3] 小池英勝, 赤間清, 宮本衛市：等価変換ルールの生成方法の理論的基礎, 情報処理学会研究報告 98-ICS-144, pp.13-18 (1998)

[4] 赤間清, 繁田良則, 宮本衛市：論理プログラムの等価変換による問題解決の枠組, 人工知能学会誌, Vol.12, No.2, pp.90-99 (1997).

[5] 赤間清，岡田浩一，宮本衛市：宣言的プログラムの推論規則, 人工知能学会誌，Vol.12, No.4, pp.114-121 (1997)

[6] 畑山満美子, 赤間清，宮本衛市：等価変換ルールの追加による知識処理システムの改善, 人工知能学会誌 Vol.12，No.6，pp.861-869 (1997)

[7] Pettorossi,A. and Proietti,M.: Transformation of Logic Programs:Foundations and Techniques, *The Journal of Logic Programming*, Vol.19/20,pp.261-320(1994).