数理解析研究所講究録
1125巻 2000年 150-161

効率的な等価変換ルールの探索に基づくプログラム合成

Program Synthesis Based on Searching for Efficient Equivalent Transformation Rules

小池 英勝[†] Hidekatsu KOIKE
赤間 清[††] Kiyoshi AKAMA
宮本 衛市[†] Eiichi MIYAMOTO

†北海道大学大学院 システム情報工学専攻
Division of System and Information Engineering, Hokkaido University
††北海道大学 情報メディア教育研究総合センター
Center of Information and Multimedia Studies, Hokkaido University

概要

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（プログラム）を記述する．等価変換ルールを用いると，正当で効率的な手続の記述が可能である．論理プログラミングでは，節の集合(論理プログラム)で宣言的に関係を正しく記述しても，実際には解を得られない場合が多い．本論文が提案する方法では，宣言的に記述された問題に対して，高速に正しく解を与える手続を自動合成できる．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 [3]．本論文では，この理論を用いて，問題を解決するための等価変換ルールの候補を多数生成し，探索によって効率的な手続きを自動合成する方法を提案する．

1 はじめに

人にとってわかり易い問題記述から，その問題を解くための効率的なプログラムを自動的に合成することができれば，問題解決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の枠組 [5] を基礎として，プログラム合成を行う．この合成法は，問題を宣言的に記述した確定節の集合を入力として，そこから等価変換ルールの集合を出力する．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 [5] を保存したまま変換するルールである．

正当な等価変換ルールを多数生成する方法 [2] とその理論的基礎 [3] が既に提案されている．多数の正当な等価変換ルールの集合から効率的なものだけを選びプログラムを構成すれば正当で効率的なプログラムを合成できる．よって等価変換の枠組みで行うプログラム合成は，多数の正当な等価変換ルールの生成と，そのルールの集合から効率的なルールを探索することが中心になる．

本プログラム合成は，問題を正しく解くプログラムの合成と，合成されるプログラムの効率化を同時に行う．現在このプログラム合成を自動的に行うシステムが試作されている [4]．本論文では等価変換ルールの生成と探索を中心に，プログラム自動合成システムの実現について議論する．

2 本プログラム合成の特徴

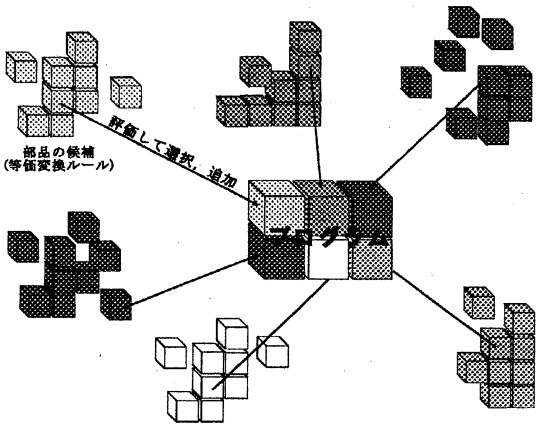

Fig.1 プログラム合成の概念

2.1 プログラムの部品としての等価変換ルール

与えられた仕様を満たすプログラムは一般に多数存在する．プログラム自動合成を行う際，その中から効率的なものを出力したい．しかし，プログラムを１つずつ合成して評価するのでは計算量が大きくなりすぎる．そこで，１つのプログラムを合成する問題を，そのプログラムを構成する多数の部品を生成する問題に置き換える．そして多数のプログラムを評価する代わりに，部品の候補を１つずつ評価選択して全体を作る (*Fig*.1)．このことによって複雑なプログラム合成の問題は，比較的小規模で簡単な問題に分解されたことになる．

上記のことを行うにはプログラムの部品に以下のような性質が必要である．

- 個々の部品が正当ならば，プログラム全体としても正当になる
- 個々の部品が効率的ならば，プログラム全体としても効率的になる

等価変換ルールは，追加による全体への正当性の影響がない．しかも個々のルールが効率的であれば全体としても効率的になる [7]．したがって上の性質を満たしている．このことから，本研究では「プログラムの部品＝等価変換ルール」と捉えることにより，プログラム合成を行う．

効率の良い等価変換ルールとは，簡単にいえば，適用したときに節を増加させないルールである．これらのルールはボディを高々１つだけ持つ．プログラムを構成する等価変換ルールが全て高々１つのボディを持つということが，そのプログラムが効率的であるかどうかの大まかな目安になる．等価変換ルールについては 4.2節で述べる．

等価変換ルールの生成は，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要求を満たす正当な等価変換ルールの候補を多数生成できる．よって，多数のプログラムの部品の候補を生成できる．メタ問題記述とメタルールは 5節で，等価変換ルールの評価に関しては 7.4節で述べる．

2.2 特徴

本プログラム合成は以下のような特徴を持つ．

(1) プログラムの部品を順次生成して追加することによってプログラム合成を行う
(2) 高いアルゴリズム記述能力を持つプログラムのクラスを採用している
(3) 多数の候補の中からプログラムを探索する

(1) について説明する．上で述べたように，プログラムが小さな部品に分解でき，その正当性や効率が他の部品から完全に独立していれば，部品を１つずつ作り追加していくことによってプログラム全体を構築することができる．等価変換ルールは，追加による正当性への副作用が無い．しかも個々のルールが効率的であれば全体としても効率的になる．したがって，等価変換ルールでプログラムを構成することによって (1) を実現することができる．

(2) について説明する．効率的なプログラムを低コストで合成するには，まずプログラムを表現する手段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．本プログラム合成法ではプログラムを等価変換ルールで記述する．等価変換ルールで記述されたプログラムは他の枠組みと比較して十分に正当かつ高速に問題を解けることが示されている [6] [7] [9]．

3.1節で述べる問題記述 ($p1$) の確定節集合は Prolog のプログラムと見ることができる．しかし，質問 $reverse(X,[1,2])$ を与えると，多くの Prolog の処理系は無限ループに陥り解を得ることは出来ない．これは，確定節のアルゴリズム表現力に限界があるためである．このような問題に対しても，柔軟に計算を行い解を得ることのできるプログラムを等価変換ルールで自然に記述することができる．そして，本プログラム合成法ではそのプログラムを自動

合成できる．

(3) について説明する．一般に与えられた仕様を満たすプログラムは多数存在する．例えば画像を処理するプログラムを考えたとき，専用のプロセッサを搭載したコンピュータとそうでないコンピュータでは，最適なプログラムは異なる．また，メモリを十分に搭載したコンピュータとそうではないコンピュータでも，同様に最適なプログラムは異なる．

このように，プログラムの評価は実行環境やユーザの要求に大きく左右される．したがって，プログラム合成とは，単に与えられた仕様を満たした1つのプログラムを合成するのではなく，多数のプログラムの候補の中から，与えられた評価関数に基づき効率的なプログラムを探索することと捉えるのが良い．本プログラム合成法は、等価変換ルールの探索によってこのことを実現する．

本プログラム合成では評価関数が外側からコントロールできるので，実行環境の変化に対するプログラムの評価の変化にも柔軟に対応することができる．

3 プログラム合成の概要

本節では，プログラム合成法の構成要素と，手順について大まかに説明する．

3.1 入力と出力

本プログラム合成では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．

関係を表す確定節集合

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

質問

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

問題記述 $(p1)$

$reverse$ 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,「リストの順序を逆にして $[1,2]$ が得られたときの，もとのリストは何か.」を表している．等価変換ルールによる問題記述の変換は，質問の節に対して行われる．関係を表す節が別の節に変換されることは無い．

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．等価変換ルールのシンタックスの説明は 4.2節にある．

$(r1)$ $reverse(\&X, [\&A|\&Y])$
$\quad \rightarrow \{\&X = [\#B|\#W]\},$
$\quad\quad reverse(\#W, \#Z),$
$\quad\quad append(\#Z, [\#B], [\&A|\&Y]).$
$(r2)$ $reverse(\&X, []) \rightarrow \{\&X = []\}.$
$(r3)$ $append(\&A, [\&B], [\&C, \&D|\&E])$
$\quad \rightarrow \{\&A = [\&C|\#F]\},$
$\quad\quad append(\#F, [\&B], [\&D|\&E]).$
$(r4)$ $append(\&A, [\&B|\&C], [\&D])$
$\quad \rightarrow \{\&A = [], \&C = [], \&B = \&D\}.$

これらのルールは、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 (4.3節，8節)．

3.2 プログラム合成システムの要素

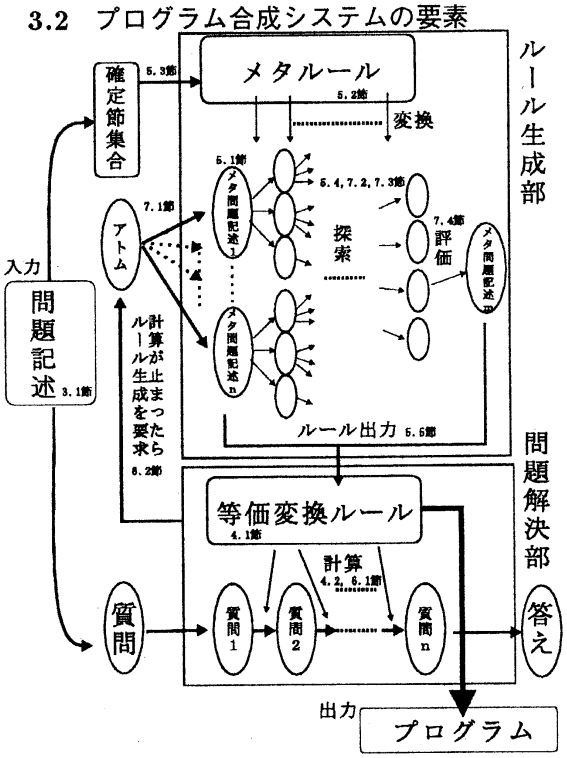

Fig.2 システムの構成

プログラム合成の大まかな構成を示した概念図が *Fig.2* である．図に書かれた節番号はその部分を説

明する節を示す．また，プログラム合成の大まかな手順の流れ図が *Fig.*3 である．

本プログラム合成システムは，大きく分けて問題解決とルール生成の 2 つのサブシステムから成る．以下に，システムの入出力とサブシステムの構成要素を列挙する．後ろについている番号はその要素を説明している節を表す．

システムの入出力

- **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．
- **プログラム** 本システムの出力で，等価変換ルールの集合．

問題解決

- **計算** (4.3，6.1節) 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求める．
- **ルール生成要求** (6.2節) 必要な等価変換ルールをアトムを使って要求する．

ルール生成

- **メタ問題記述** (5.1節) 多数の問題記述を 1 つの表現で表したもの．
- **メタルール** (5.2節) メタ問題記述を等価変換するためのルール．
- **メタ計算** (5.4節) メタ問題記述をメタルールで変換する．
- **探索** (7.2，7.3節) 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．
- **ルールの評価基準** (7.4節) 探索で個々のルールを評価するときに使う．

3.3 プログラム合成法の手順

プログラム合成の手順を示した *Fig.*3 を簡単に説明する．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適用可能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．ルール生成の際システムは，始めに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 (メタ計算) 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．この方法では一般に生成可能な等価変換ルールは複数あるので，その中から効率的なルールを選択する．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．

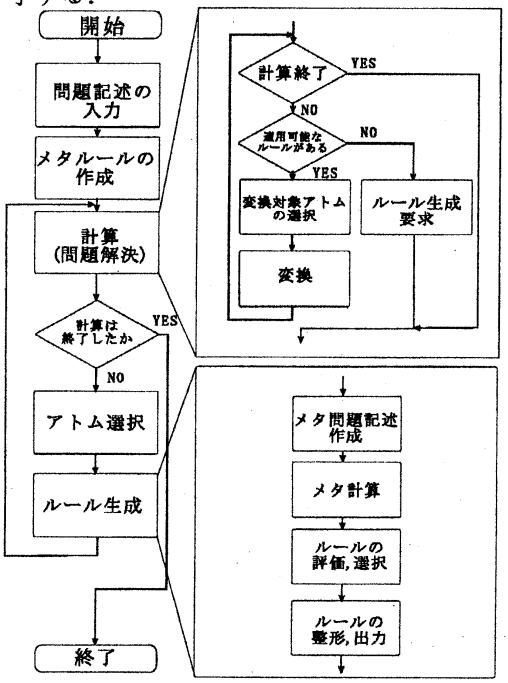

*Fig.*3 プログラム合成の手順

4 等価変換ルールと問題解決

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．

4.1 メタアトム

等価変換ルールを表現するために，& 変数，# 変数，メタ項，メタアトムを導入する．& 変数とは，先頭に&の付いた変数で，任意の項を代入できる．# 変数とは，先頭に#が付いた変数で，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．メタ項とは，通常の変数の替わりに & 変数や # 変数が現れる項である．メタアトムとは，通常の項の替わりにメタ項を持つアトムである．

メタアトムの変数に通常の変数や項を代入することによって，通常のアトムが得られる．

† 実際に変換されるのは質問の節だけである．

4.2 等価変換ルール

2つの互いに疎な述語記号の集合 R_P, R_R が与えられているとする．R_R 中の述語記号は等価変換ルールの条件部や実行部のメタアトムを作るために用いられる．R_P 中の述語記号は条件部と実行部と以外に現れるメタアトムを作るために用いられる．R_R と R_P の具体例は以下のようなものである．

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

R_P で作られたメタアトムは，ルールの適用の際，ある節のボディアトムを書き換えるアトムとして使われる．R_R で作られたメタアトムは，代入，等式，大小関係の比較，文字の表示などルール適用の際にプリミティブな動作を行うアトムとして使われる．

R_P と R_R 上の等価変換ルールとは以下の様な形をしている．

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

ここで，$\langle rulename \rangle$ はルール名でルールの優先度を設定するために使われる‡．H は，R_P で作られたメタアトム，C_s と E_{S_i} は，R_R で作られたメタアトムの列 (空列を含む)，B_{S_i} は R_P で作られたメタアトムの列 (空列を含む) である．H をヘッド，C_S を条件部，E_{S_i} を実行部，E_{S_i} と B_{S_i} の組を i 番目のボディと呼ぶ．ルール名，条件部，実行部は省略可能である．

等価変換ルールが持つ全てのメタアトムは，ルール適用の際，代入 θ によって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．# 変数へ代入する通常の変数の名前は，変換対象の節に現れないものを任意に選ぶ．また，異なる # 変数には異なる名前の変数を代入する．

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．

節 C のボディアトム b にルールが適用されると，始めにルールのボディの数 (n 個) の C のコピー $C_i (i = 1, \ldots, n)$ が作られる．そして，$E_{S_i}\theta$ 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_i を用いて，節 $C_i\theta\sigma_i$ のアトム $b\theta\sigma_i$ が $B_{S_i}\theta\sigma_i$ で置き換えられる．適用したルールの，実行部が成功したボディが l 個であった場合，節 C は，l 個の新しい節に変換される．

4.3 問題解決 (計算)

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によって解を求める．

たとえば，3.1節の問題記述の質問の節は，3.1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) $ans(X) \leftarrow \mathbf{reverse(X, [1,2])}.$
2) $ans([B1|W1]) \leftarrow reverse(W1, Z1),$
$\mathbf{append(Z1, [B1], [1,2])}.$
$(r1)$ より
3) $ans([B1|W1]) \leftarrow reverse(W1, [1|F1]),$
$\mathbf{append(F1, [B1], [2])}.$
$(r3)$ より
4) $ans([2|W1]) \leftarrow \mathbf{reverse(W1, [1])}.$
$(r4)$ より
5) $ans([2, B2|W2]) \leftarrow reverse(W2, Z2),$
$\mathbf{append(Z2, [B2], [1])}.$
$(r1)$ より
6) $ans([2,1|W2]) \leftarrow \mathbf{reverse(W2, [])}.$
$(r4)$ 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．太字で強調されたアトムがルール適用によって書き換えられている．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．

‡ 本論文では，ルール名を使う例は現れない．
§ ユニフィケーションとは異なり，変換対象を変更していないことに注意されたい．
¶ 実行部が省略されているボディは成功したと考える．

5 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．

5.1 メタ問題記述

メタ問題記述とは，問題記述の集合を表したものである．1つのメタ問題記述は，そのメタ問題記述が表す共通の特徴を持った全て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，任意の節のヘッド（アトム）を表す．メタアトムは4.1節で述べた．メタ問題記述中のメタ節は，共通の性質を持った節の集合を表す．

メタ節の具体例は以下のようなものである．

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

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，3.1節の問題記述$(p1)$を含む多数の問題記述を表す．メタ問題記述は次の節で説明するメタルールによって等価変換される．

5.2 メタルール

メタルールは，以下の様な形をしている．

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

4.2節で述べた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．等価変換ルールと同様に、ルール名，条件部，実行部は省略可能である．

$*$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%変数を持つ全てのアトムが，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$H_1,\ldots,H_m$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，問題記述の関係を表す節集合から自動的に得られるルールと，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．

自動的に得られるメタルールの具体例は5.3節にある．あらかじめ用意しておくメタルールの例として，等式制約を処理するメタルールを以下に示す．

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

$\{*A == *B\}$は条件部で，$*A$と$*B$が等しいとき真になる．

5.3 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．3.1節の問題記述$(p1)$が与えられたとする．始めに，与えられた問題記述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．

ただし$equal$は引数同士が等しいことを表す述語とする．

$(p1)$の確定節集合は平坦化によって以下の様に書き換えられる．

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

そして，この確定節からメタルールを作る．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．等しいヘッドを持つ節から，その節の数だけボディを持つ1つのルールを作る．例として，上の確定節集合から得られるメタルールは以下の様になる．

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

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

$m1$は，$c1$と$c2$から作られる．$m2$は，$c3$と$c4$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．

変数は，ヘッドに現れるものは$*$変数に，それ以外のものは$\%$変数にする．これらの理論的な説明は[3]にある．

5.4 メタ問題記述の変換(メタ計算)

具体例で説明する．ここで示す例は，3.1節の等価変換ルール$(r2)$の生成の例である．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．

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

与えられたメタ問題記述を，メタルール$m1$，$m2$と等式制約に関するメタルールで以下のように変換していく．

1) $h \leftarrow$ **reverse**$(\&\mathbf{A}, [\,])$.

2) $h \leftarrow equal(\&A, [\#X|\#X1]),$
$reverse(\#X1, \#R),$
append$(\#\mathbf{R}, [\#\mathbf{X}], [\,])$.
$h \leftarrow equal(\&A, [\,]), equal([\,], [\,]).$
$(m1)$より

3) $h \leftarrow equal(\&A, [\#X|\#X1]),$
$reverse(\#X1, \#R),$
$equal(\#R, [\,]),$
equal$([\#\mathbf{X}], [\,])$.
$h \leftarrow equal(\&A, [\#X|\#X1]),$
$reverse(\#X1, \#R),$
$equal(\#R, [\#A, \#X2]),$
equal$([\,], [\#\mathbf{A}, \#\mathbf{Zs}]),$
$append(\#X2, [\#X], \#Z).$
$h \leftarrow equal(\&A, [\,]),$ **equal**$([\,], [\,]).$
$(m2)$より

4) $h \leftarrow equal(\&A, [\,]).$
(等式制約に関するメタルール)より

太字で強調されたメタアトムがメタルールによって書き換えられている．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．3)の1番目と2番目のメタ節の$equal([\#X], [\,])$，$equal([\,], [\#A, \#Zs])$は，関係が成立することは無いので，等式制約に関するメタルールによって消去される．

ここで示した変換の列1)～4)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ので，多数の変換列が得られる．

5.5 等価変換ルールの生成

5.4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$(r2)$が得られる．5.4節で述べたように変換列は多数生成されるので，多数の等価変換ルールが生成可能である．この様にして得ることができる等価変換ルールひとつひとつがプログラムの部品の候補である．

6 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．

6.1 変換の制御

問題解決部は，等価変換ルールで問題記述を変換

して計算を行う．変換の制御は，

- 節 (変換対象節) の選択
- 変換対象節のボディのアトム (変換対象アトム) の選択
- 適用ルールの決定

から成る．これら 3 つの項目にはいずれにも非決定性を含む場合があるので以下の様なヒューリスティックを用いる．

変換対象選択の条件としては，

(1) 以前に選択していない節，アトムを選ぶ．
(2) 変換されてから時間のたった節，アトムを選ぶ．

この条件は，節，アトムの選択どちらにも用いられ，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続けることによる無限ループを防いでいる．実験ではこの条件の採用で，選択順序を固定した場合に起こる無限ループを避けることができた．逆に状況が悪化したケースは今のところ無い．

ルールは，記述された順序で調べ最初に適用可能と判明したものを選択している．

6.2 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によって，そのアトム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，選択は以下の様なヒューリスティックで行う．

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

これらの条件を設けた目的は，アトムの引数がなるべく単純な変数になっていない物を選ぶためである．7節で述べるメタ問題記述の作成方法は，引数が変数のアトムより，具体化されたアトムから多くの種類のメタ問題記述を生成できる．しかし，この条件によって常に最適な選択が起こるとは保証されていない．

7 ルール生成部

本節では，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．

7.1 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を用いて，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評価の高い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，共通する計算を同時に行う方法で計算量を大幅に減らすことに成功している．このテクニックについては別の論文で議論する．

7.2 メタ問題記述の変換 (メタ計算) 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．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．変換回数の上限は今のところ人が設定している．上限が小さすぎることによって起こる不都合は，最適なルールの候補を生成する前に終了してしまうことである．このような場合でも，実際に計算は正しく行われる場合が多い．正しく行われない場合とは無限ループに陥る場合である．本プログラム合成方法は等価変換ルールのみを出力するので，誤った答えを出して終了することはない．

7.3 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．この値をあまり小さくしすぎると畳み込みによる最適化が行えるパスを見逃してしまう可能性がある．この最適な値は，期待したルールが得られる最小値となるが，その値より大きくても計算量が実用的な範囲にさえなればよいので，簡単なヒューリスティックで与えてよい場合が多い．(2) は無駄な変換を防ぐ為に，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によって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つにまとめることによって，余分な探索を減らす．

7.4 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) の一般性
(5) 無限ループ可能性
(6) ユーザーの指定したアトムの有無

これらの基準を評価関数 F_e で定式化する．

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

関数 $F_e(x)$ はルールの評価を整数の点数としてあらわす．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，特定のアトムを含む，又は，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の点数の2項組みを指定する．

基準 (1) ～ (6) を簡単に説明する．(1)，(3) は，適用後の計算量を減らすという観点から数は少ないほど良い．(2) は，畳み込みにより計算量が減った結果が再帰構造として現れる可能性があるので再帰したほうが良い．(4) は，高い評価のルールはなるべく適用可能性を高くするという考えから，一般的な方が良い．(5) の無限ループチェックは，1つのルールを見て発見できるものだけである．最も簡単なものは，ヘッドとボディが全く同じものである．(6) は環境や要求によって変わる評価を指定されたあるアトムの出現の有無で表現するためのものである．

等価変換ルールの評価には上にあげた原則は存在するが評価関数の表現の仕方は，一意に決まらない．実際のシステムでは評価関数の中の重みは，複数の性質の異なる例題を実行して共通に使える値を定めている．しかし，この原則に従って評価を行え

ば重みに多少の幅があっても最適なルールが得られる場合が多い．

8 等価変換ルールの生成例と効率

8.1 自動生成されたルールの追加

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 $(r1) \sim (r4)$ を出力する．このルールの集合は正しく $(p1)$ の解を与える．

ここで，問題記述 $(p1)$ の質問の節を

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

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 $[1,2]$ の位置が $p1$ と逆になっている．等価変換ルール $(r1) \sim (r4)$ 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によって与えた質問

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

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 $(r1)$ の適用によって質問の節に現れる $append$ アトム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，$(r8)$ を取り除くことも出来る．

8.2 本プログラム合成法で可能な効率化

本プログラム合成法で可能な効率化は

1. unfold/fold 変換による効率化
2. 新述語の定義による効率化
3. 再帰呼び出しの除去

などがある．

(1) の効率化が現れる例題では，問題記述 $(p1) \sim (p3)$ や，素数を生成する問題などがある．素数を生成する問題 [4] も問題記述 $(p1)$ と同様に論理の枠組では無限ループになり解を得ることができない．

(2) は，論理や関数プログラミングのプログラム変換の例題でよく知られている効率化であり [1]，同等の効率化を本枠組みで行うことができる [2]．新述語の定義は，生成した等価変換ルールのボディの B_{S_i} に現れる2つ以上のアトムの組み合わせをボディとする新しい節を問題記述に追加する．新述語の定義の仕方は多数存在するので原理的には全ての定義を試して最も評価の高いものを選択する．実際のシステムでは，自動的に新述語の定義を行えるが，この詳細の説明は本論文では割愛する．

(3) の例題としては，階乗の計算や，リストの長さを求める計算，本論文の問題記述 $(p2)$ などがある．これらの例題で再帰除去を実際のシステムで行

うことに成功している．

他には，余りを求める計算 mod と指数計算 exp が組み合わされた計算の例がある．問題記述では，mod と exp を組み合わせて計算が定義されているが，計算中の桁あふれを抑えるために評価関数に exp があるルールは低い評価を与えるように設定する．すると exp の全く現れない計算を行うルールの集合を出力した [4]．

以下に，新述語の定義による再帰呼び出し除去の例を示す．問題記述として $(p2)$ を与える．システムは，はじめに $(r5)$ を生成するが，このルールのボディから新しい定義

$$auto(X,Y,Z) \leftarrow reverse(X,S), append(S,Y,Z).$$

を問題記述に追加する．そして，ルール生成をやり直す．

最終的に以下のような等価変換ルールの集合を生成する．

$(r10)$ $reverse([\&A|\&X],\&Y) \rightarrow auto(\&X,[\&A],\&Y).$

$(r11)$ $auto([\&A|\&X],\&Y,\&Z) \rightarrow auto(\&X,[\&A|\&Y],\&Z).$

$(r12)$ $auto([],\&Y,\&Z) \rightarrow \{\&Y = \&Z\}.$

このルールの集合は，$(r5) \sim (r9)$ と比較してスタッキングの時間，記憶領域が改善されている．

8.3 合成されるプログラムの性能の下限

問題記述 $(p1) \sim (p3)$ について，システムが探索を行わずに生成する等価変換ルールは以下のものである．

$(u1)$ $reverse(\&X,\&Z) \rightarrow \{\&X = [\#Xs|\#Y])\}$, $reverse(\#Y,\#R)$, $append(\#R,[\#Xs],\&Z)$; $\rightarrow \{\&X = [],\&Z = []\}$.

$(u2)$ $append(\&X,\&Y,\&Z) \rightarrow \{\&X = [],\&Y = \&Z\}$; $\rightarrow \{\&X = [\#A|\#Xs]$, $\&Z = [\#A|\#Zs]\}$, $append(\#Xs,\&Y,\#Zs)$.

これらのルールは，5.3節のメタルール $(m1)$, $(m2)$ の $*$ 変数を & 変数に，% 変数を # 変数にそれぞれ書き換え，$equal$ を実行部に移したものになっている．そして，これらのルールは引数が & 変数だけのアトムのみが現れるメタ問題記述 $h \rightarrow reverse(\&X,\&Y)$. や $h \rightarrow appned(\&X,\&Y,\&Z)$. を作り 1 回だけ変換して得られるものに等しい．これらのルールは，Prolog のレゾリューションに相当する unfold 変換である．unfold 変換は等価変換である [8]．このことからシステムは少なくとも Prolog が行う計算に相当する等価変換ルールを出力することができる．

9 比較

本論文のプログラム合成を論理プログラムのプログラム合成，プログラム変換などと比較する．ここでは，2節で挙げた本プログラム合成法の特徴 (1) ~ (3) に関して比較する．

特徴 (1) は，プログラム合成をルール生成という小さな問題の集合に分解しているということである．これには，出力するプログラムの構成要素 (等価変換ルール) の相互の独立性が必要である．しかし，論理の枠組ではプログラムの構成要素 (確定節など) の独立性が不十分なので，本合成法のような小問題への分解は行えない．よって，論理プログラムのプログラム合成やプログラム変換はより複雑で難しい問題にならざるを得ない．

特徴 (2) は，出力されたプログラムが高いアルゴリズム記述能力を持つということである．論理プログラミングでは $reverse$ の例で示したように，単に問題を確定節で宣言的に記述しただけでは，解を得られない例が多数存在する．論理プログラミングの解法は等価変換の立場から見ると，ヘッドの引数が変数のみで，条件部の無い unfold 変換ルールのみを使って計算していることに相当する．これは，ルールの発火条件が緩いので，計算手続きが非決定的になり易く，よって制御が難しくなる．このために計算の実行は非効率的になったり，無限ループに陥る場合がある．等価変換ルールは，4節で示したように，ヘッドと条件部で適用条件を詳細に記述できるので，計算を柔軟に制御できる．

特徴 (3) は，本プログラム合成法が，仕様を満たす多数のプログラムを生成可能であり，その中から評価の高いものを選択できることを意味している．論理の枠組でのプログラム合成やプログラム変換の研究は，与えられた仕様やプログラムから 1 つのプ

ログラムだけを生成するものが多く，探索でより良いプログラムを探す有効な方法は知られていない．

以上により本プログラム合成法の特徴 (1) ～ (3) は，本論文の新規性と考えられる．

10 おわりに

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] 小池英勝, 赤間清, 宮本衛市：等価変換ルールの探索に基づくプログラム合成, 電子情報通信学会技術研究報告 SS99-20, pp.41-48 (1999)

[5] 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).

[6] 吹田慶子, 赤間清, 宮本衛市：領域知識と状況を利用した未知単語の理解, 電子情報通信学会技術研究報告 SS97-52, pp17-24 (1998)

[7] 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)

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

[9] 北野智丈, 赤間清, 宮本衛市：参照制約に基づく否定文と疑問文の意味理解, 電子情報通信学会技術研究報告 KBSE98-14,pp33-40(1998).