知能と複雑系 117－1
(1999. 7. 22)

メタ記述の変換によるアルゴリズム生成の基礎

秋元 亨介　赤間 清　宮本 衛市

北海道大学工学部情報工学科

札幌市北区北 13 条西 8 丁目. Tel. 011-706-6814

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

等価変換パラダイムでは、宣言的記述をルールを用いて等価変換することにより、問題の解決を行う。本論文では、仕様からアルゴリズムを生成する問題を等価変換パラダイムを用いて研究する。問題の仕様を問合せ集合 Q と宣言的記述 P のペア、アルゴリズムを等価変換ルール集合 R と見なし、(Q,P) から R を生成する方法を提案する。この方法では、Q から得たメタ節を P から導かれたメタルールで等価変換することによって、ルールを生成する。" 新しい節の定義"、" メタルールの追加"、" 評価に基づいたルールの選択"、" アトムパターンの生成" などのステップからなるルール生成のアルゴリズムを提案し、それを所属問題に適用する。

宣言的記述　等価変換　メタ節　メタルール　ルール生成

A Foundation for Algorithm Generation by Transforming Meta-descriptions

Kyousuke Akimoto　Kiyoshi Akama　Eiichi Miyamoto

Department of Information Engineering, Faculty of Engineering, Hokkaido University

North 13 West 8 Kita-ku Sapporo. Tel. 011-706-6814

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

In the equivalent transfomation (ET) paradigm, problems are solved by ET of declarative descriptions by using rules. In this paper, we investigate how to generate an algorithm from a specification based on the ET paradigm. We regard a specification as a pair of a quiry set Q and a declarative description P, an algorithm as a set R of ET rules, respectively, and propose a method of generating R from (Q, P). In this method we generate new ET rules by equivalently transfomating meta-clauses that are constructed from $Q$by using meta-rules that are constructed from P. We also propose an algorithm for rule generation (consisting of such steps as " definiting new clauses," "adding meta rules," "selecting rules based on evaluation mapping," and "generating atom patterns"), and apply it to membership problems.

declarative description　equivalent transfomation　meta-clause　meta-rule　rule generation

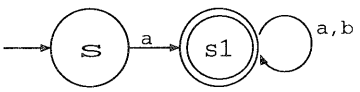
1 はじめに

「問題の仕様からその問題を解くアルゴリズムを作る」ということは重要なことである。本論文では、仕様からアルゴリズムを生成する問題を等価変換パラダイムを用いて研究する。等価変換パラダイムでは、問題を理論的に「問合せ集合と宣言的記述のペアから等価変換ルールの集合を作る」という形で捉え、解決する。その上で、「Σ上の言語 L を認識する手続きを作る」という問題を取り上げる。以下では、具体的な例を用いてこれを示すとともに、メタ記述の変換によってルールを生成する方法も示す。

2 問題設定

2.1 例題の設定

「Σをアルファベット $\{a,b\}$ とする。このとき、
言語 L_0は a で始まるΣ上の文字列集合であると
する。この言語 L_0を認識する手続き R_0を作れ。」
という例題を設定する。この問題を直観的に解くと、手続き R_0は以下のようなオートマトンで表される。



この問題は、問題の仕様 S から手続き R を得る問題 (以下では手続き生成問題と呼ぶ) である。

2.2 等価変換パラダイムによるアプローチ

本論文では、等価変換パラダイムを採用し、この手続き生成問題を解決する。等価変換パラダイムにおける計算とは、「宣言的記述を 意味を保存して ルールによって 変換する」ものである。本論文では、宣言的記述は確定節の集合である。

本論文では、等価変換パラダイムに基づいて、以下のように手続き生成問題を捉える。

1. 仕様 $S=(Q,P)$　　Q:問合せの集合
　　P:宣言的記述 (確定節の集合)
2. 手続き R =等価変換ルールの集合

つまり、「問題の仕様から手続きを作る」という手続き生成問題を、「問合せ集合と宣言的記述のペアから等価変換ルールの集合を作る」という形で捉える。

2.3 例題の定式化

前節の枠組で、2.1 節の例題を捉えると、仕様 S_0は、問合せ Q_0と宣言的記述 P_0を用いて、以下のように定式化することができる。

$S_0=(Q_0,P_0)$
$Q_0=\{s(X)|X\in\Sigma^*\}$
$P_0=\{s(X)\leftarrow X=aY\}$

ただし、X、Y はΣ上の文字列を表す変数である。

手続き R_0は、等価変換ルールの集合として以下のように与えられる。

rule1 : $s(\varepsilon)\rightarrow\langle false\rangle$.
rule2 : $s(a\&W)\rightarrow s1(\&W)$.
rule3 : $s(b\&W)\rightarrow\langle false\rangle$.
rule4 : $s1(\varepsilon)\rightarrow\langle ture\rangle$.
rule5 : $s1(a\&W)\rightarrow s1(\&W)$.
rule6 : $s1(b\&W)\rightarrow s1(\&W)$.

2.4 等価変換ルールによる言語 L_0の認識

R_0は確かに 求めるべき手続きになっていることを示す。すなわち、R_0を用いると、Σ上の任意の文字列αが与えられたとき、αが L_0に属すか否かを判定する認識問題を解決する手続きになっていることを示す。「文字列 αが 言語 L_0に属する」という条件は、

$yes\in M(P')$ [1]

で表される。したがって、R_0のルールを次のように使って等価変換を行うことによって判定することができる。始めに、次の P'を作る。

$P'=P\cup\{yes\leftarrow s(\alpha)\}$

P'を R_0で等価変換して、もし、

$P''=P\cup\{yes\leftarrow\}$

に変換できれば答えは yes であり、

$P''=P$

に変換できれば答えは no である。例えば、文字列 aab の場合には、

$yes\leftarrow s(aab)$. by rule2
$yes\leftarrow s1(ab)$. by rule5
$yes\leftarrow s1(b)$. by rule6
$yes\leftarrow s1(\varepsilon)$. by rule4
$yes\leftarrow$.

のように yes 節は単位節に変換され、yes という答えが得られる。

ここで、P_1から P_2に問題を等価変換するとは、$M(P_1)=M(P_2)$ を満たす条件のもとで P_1を変形し P_2を得ることである。よって、もし各ステップが等価変換であることが保証されれば、この解答の正当性も保証される。

2.5 ルール集合とオートマトンの対応

2.3 節で示した等価変換ルール集合は、2.1 節のオートマトンと同一視できることを示す。ルール R_0の中で、例えば、

[1] $M(P)$ は、宣言的記述 P の意味を表す。

rule2: $s(a\&W) \rightarrow s1(\&W)$.

のルールは、状態 s と状態 $s1$ がラベル a の弧でつながれていることに対応させる。また、問合せが $s(\alpha)$ で与えられた場合は、s が初期状態であることに対応させ、

rule4: $s1(\varepsilon) \rightarrow \langle ture \rangle$.

の形のルール ($body$ にアトムが無いルール) は、$s1$ が最終状態であることに対応させる。

以上のように捉えると、2.3節で示した等価変換ルール集合は、2.1節のオートマトンとちょうど対応する。また、この意味で、等価変換に基づく 手続き生成問題は、例題1のような問題を形式的に捉える方法の1つと見なすことができる。

2.6 問題仕様

問題仕様 S として、宣言的記述 P が、「引数がただ一つである述語 s とΣ上の文字列を表す変数 X を用いて、$s(X)$ をヘッドとし、任意のアトムでボディが構成される確定節」で表され、問合せが、「すべての文字列αに対して、$s(\alpha)$ を集めた集合」で表されるものを採用し、そこから問題を解く効率的な等価変換ルールを生成する。

より具体的には、対象とする問題の仕様 S が、問合せ Q と宣言的記述 P を用いて、

$S = (Q, P)$

$Q_0 = \{s(X) | X \in \Sigma^*\}$

$P = \{s(X) \leftarrow atom_0, \cdots.\}$

のように与えられたとき、 $P' = P \cup \{yes \leftarrow s(X)\}$ の yes 節を等価変換するためのルール集合を生成する。

3 ルールの生成方法

2章の等価変換ルールを生成するために、以下のような方法を提案する。

3.1 メタ表現の導入

まず、ルールを生成するために必要な表現方法として、メタ節とメタルールを導入する。

3.1.1 メタ節

メタ節は、&変数、#変数、定数を引数に持つ0個以上のメタアトムからなるボディと、引数を持たない任意の述語 (ここでは、$head$) からなるヘッドで作られる次のような節である。

$head \leftarrow atom_0, atom_1, \cdots.$

例えば、$head \leftarrow s(a\&X)$. はメタ節である。

3.1.2 メタルール

メタルールは、メタ節を等価変換するためのルールである。これは、節集合 (定義節など) から作ることができる。しかし、通常の節を変換する等価変換ルールよりも、メタルールは一段上のレベルの変換ルールである。そこで、変数は、*変数と%変数を用いる。

例えば、$s(*X) \rightarrow *X = a\%Y$. はメタルールである。

3.1.3 メタ表現に用いられる変数

メタ節の中で用いられる変数 (&変数と#変数) について説明する。&変数とは、$\&R$ のように&のついている変数であり、「任意の項を代入できる」変数である。#変数とは、$\#R1$ のように#のついている変数であり、「変換前の節に出現しない任意の新変数を代入できる」変数である。等価変換ルールに用いられる変数は、この&変数と#変数である。

また、メタルールの中で用いられる変数 (*変数と%変数) について説明する。*変数とは、$*R$ のように*のついている変数であり、「任意の項を代入できる」変数である。具体的には、定数および変数 (&変数と#変数) を代入できる。%変数とは、$\%R1$ のように%のついている変数であり、「変換前の節に出現しない任意の#変数を代入できる」変数である。メタルールに用いられる変数は、この*変数と%変数である。

3.2 対象となる問題を記述

ここからが、ルール生成の過程である。まず、ルールを生成する基となる、問題仕様を宣言的に記述する。本論文で扱う例題では、新たに定義した節 C_0と、定義節 D とあわせて記述する。

3.3 メタルールの準備

メタ節を変換するメタルールをあらかじめ用意しておく。これらのメタルール群を MR_0とする。ここで、MR_0は、

1. 定義節 D から得られるメタルール
2. 新たに定義した節 C_0から得られるメタルール

である。これらを用いて、メタルール MR_0を生成することを

$MR_0 = genMR(\{D \cup \{C_0\}\})$

と表す。

メタルールを得る一つの方法として、以下の手順があげられる。

1. 確定節の平坦化を行う。この時、節のヘッドの形を揃える。

2. 平坦化した節のヘッドに現われる変数をすべて*変数にする。
3. それ以外の変数を%変数にする。
4. $Head$ を一つにまとめ、矢印の向きを変更する。

例えば、

$$\begin{aligned} accepts(R,X) \leftarrow\ & equal(X,[]),\\ & accepts\_empty(R).\\ accepts(R,X) \leftarrow\ & equal(X,[S|X1]),\\ & next(R,S,R1),\\ & accepts(R1,X1). \end{aligned}$$

のように平坦化された節から、

$$\begin{aligned} accepts(*R,*X) \rightarrow\ & equal(*X,[]),\\ & accepts\_empty(*R).\\ \rightarrow\ & equal(*X,[\%S|\%X1]),\\ & next(*R,\%S,\%R1),\\ & accepts(\%R1,\%X1). \end{aligned}$$

のようなメタルールを得ることができる。

3.4 ルール生成ルーチン

本節では、ルール生成を表す gen を定義する。
ルールを生成するには、

1. 対象となるアトムのパターン p
2. ルール生成の計算を行うメタルールの集合 MR
3. ルールを選別する評価基準に基づく制御 $ctrl$

が必要である。制御 $ctrl$ は、「どのアトムにどのメタルールを適用するか」を表したものである。一般には、同じパターンで、同じメタルールの集合を用いても、適用順序を変更すれば、生成されるルールも異なる。

ルールは、メタ節をメタルールで変換して、変換が終了した段階で生成される。例えば、アトムパターン p を変換し、アトムパターン p' を得たとき、

$head \leftarrow p.$
$\cdots$(メタルールの適用)
$head \leftarrow p'.$

という変換から、
$r = p \rightarrow p'.$
というルール r が生成される。これを
$p' = gen(p, MR, ctrl)$
$r = (p \rightarrow p')$
と表す。また、ルール生成が正常に行われなかったとき(無限ループに陥るなどして停止しない場合など)は、$null$ を返すものとする。

3.5 評価基準の設定

本節では、評価関数 $eval$ を定義する。

前章のようにアトムパターン、メタルール、制御によってルールが一つ生成される。しかし、そのルールが有効なルールどうかは保証できない。そこで、生成されたルールに対して、実際に出力するルール群に加えるかどうかを決めるために、評価基準を設定する。

本論文では、評価基準として以下の形のものをあげる。この形のどれかであれば、それをルール群に加える。

- $atom \rightarrow \langle ture \rangle$.
- $atom \rightarrow \langle false \rangle$.
- $atom \rightarrow atom'$.

生成したルール r が、この基準を満たすとき、
$eval(r) = T$
とし、それ以外のときは、
$eval(r) = F$
とする。

3.6 メタルール生成ルーチン

本節では、新しく節の定義を表す $define$ と、メタルール生成を表す $genMR$ を定義する。

生成されたルールが評価基準を満たせば、そのルールは新たに実際に出力するルール群に加えられる。一方、評価基準を満たさなかった場合は、変換が終了した段階でのメタ節から、新たに確定節 C_{new} を作る。その確定節からメタルール mr を作り、メタルール群に追加することにより、それ以降のルール生成ルーチンで用いることができる。

メタアトム p' から新たに確定節 C_{new} を作ることを
$C_{new} = define(p')$
と表し、確定節からメタルール mr を作ることを
$mr = genMR(\{C_{new}\})$
と表す。

3.7 アトムパターンの生成

本節では、新しいアトムパターンを生成することを表す $genAtom$ と、アトムパターンの特殊化を表す $specialize$ を定義する。

$genAtom$ は、新しく節を定義するために必要なアトムパターン Pat_{new} を生成することであり、
$Pat_{new} = genAtom$
と表す。また、$specialize$ は、引数となるアトムパターンの集合の各要素を、対象とするデータドメインを用いて、よりパターンを特殊化する操作である。例えば、$s(\&X)$ というアトムは、

$\{s(\varepsilon), s(a\&X), s(b\&X)\} = specialize(s(\&X))$
というアトムパターンに特殊化される。

3.8 ルール生成システム

ルール生成は、次のような手続きになっている。ここで、生成されるルールの集合を GR、メタルールの集合を MR、アトムパターンの集合を Pat とする。

ルール生成アルゴリズムを疑似的な言語で記述する。

```
GR := ∅,MR := MR0,Pat := Pat0;
MR0 := genMR({D ∪ {C0}});
Pat0 := specialize(s(&X));
while(Pat ≠ ∅){
   Pat' := Pat;
   for p in Pat'{
      p' := gen(p, MR, ctrl);
      r := (p → p');
      if(r ≠ null){
         if(eval(r) = T){
               GR := GR ∪ {r};
               Pat := Pat − {p};
         }else{
          Patnew := genAtom;
          Pat := Pat ∪ {Patnew};
          Cnew := define(p');
          MR := MR ∪ {genMR({Cnew})};
          p'' := gen(p', MR, ctrl);
          r := (p' → p'');
          GR := GR ∪ {r};
          Pat := Pat − {p};
          }
      }
   }
   Pat := specialize(Pat);
}
return GR;
```

4 正規表現によって受理される言語

本章では、前章で提示したアルゴリズムを他の例にも適用するために、正規表現によって受理される言語について説明する。

4.1 正規表現

まず、Σをアルファベットとし[2]、Σ上の正規表現 R を集合 Reg として次のとおりに定義する。

[2]本論文では、簡単のために、Σの元 a, b についてのみ扱う。

$$Reg ::= void|a|b|(Reg; Reg)|(Reg + Reg)|Reg^*$$

ただし、$void$ は空集合を表す正規表現である。また、正規言語 L, L_1, L_2 を表す正規表現をそれぞれ r, r_1, r_2 で表すとき、$(r_1; r_2)$ は L_1L_2 を表し、$(r_1 + r_2)$ は $L_1 \cup L_2$ を表し、(r^*) は $L^*(= \bigcup_{i=0}^{\infty} L^i, L^0 = \{\varepsilon\}, L^i = LL^{i-1}(i \geq 1))$ を表す。ここで、εは空文字列を表す。このとき、正規表現 $void^*$は、集合 $\{\varepsilon\}$ を表している。また、簡単のために、単項演算子'$*$'は最も優先順位が高く、次に';'、最後に'+'という規則に従うものとし、更に、例えば $(R1; R2; R3)$ は $(R1; (R2; R3))$ を表すものとする。

4.2 正規表現によって受理される文字列

前節で定義した正規表現 R に受理される文字列について考える。Σ上の文字列 X は、a、b で構成されるリストまたは、空リストであるとする。文字列 X が正規表現 R に受理されるとは、正規表現 R によって受理される言語に文字列 X が属しているということである。

4.3 正規表現によって受理される文字列の判定方法

4.1 節、4.2 節に基づき、任意に与えた文字列 X が正規表現 R に受理されるかどうかを判定するために $accepts(R, X)$ の形で得られるような定義節を以下に示す。

```
accepts(R, []) ← accepts_empty(R).
accepts(R, [S|X]) ← next(R, S, R1),
                    accepts(R1, X).
accepts_empty((R1; R2)) ← accepts_empty(R1),
                          accepts_empty(R2).
accepts_empty((R1 + R2)) ← accepts_empty(R1).
accepts_empty((R1 + R2)) ← accepts_empty(R2).
accepts_empty(R*) ←.
next(R, S, R1) ← transf(R, TR),
                 deriv(TR, S, L),
                 make_regexpr(L, R1).
transf(void, void) ←.
transf(S, S) ← symbol(S).
transf((void; R), void) ←.
transf((S; R), (S; R)) ← symbol(S).
transf(((R1; R2); R3), R4) ←
      transf((R1; R2; R3), R4).
transf(((R1 + R2); R3), R4) ←
      transf(((R1; R3) + (R2; R3)), R4).
transf((R1*; R2), R3) ←
```

$$transf((R2 + (R1; (R1^*; R2))), R3).$$
$$transf((R1 + R2), (R3 + R4)) \leftarrow transf(R1, R3),$$
$$transf(R2, R4).$$
$$transf(R1^*, (void^* + R2)) \leftarrow transf((R1; R1^*), R2).$$
$$deriv(void, S, []) \leftarrow.$$
$$deriv(void^*, S, []) \leftarrow.$$
$$deriv(S, S, [void^*]) \leftarrow symbol(S).$$
$$deriv(S1, S2, []) \leftarrow$$
$$symbol(S1), symbol(S2), S1 \neq S2.$$
$$deriv((S; R), S, [R]) \leftarrow symbol(S).$$
$$deriv((S1; R), S2, []) \leftarrow$$
$$symbol(S1), symbol(S2), S1 \neq S2.$$
$$deriv((R1 + R2), S, L) \leftarrow deriv(R1, S, L1),$$
$$deriv(R2, S, L2),$$
$$append(L1, L2, L).$$
$$make\_regexpr([R], R) \leftarrow.$$
$$make\_regexpr([R1, R2|Rs], (R1 + T)) \leftarrow$$
$$make\_regexpr([R2|Rs], T).$$
$$symbol(a) \leftarrow.$$
$$symbol(b) \leftarrow.$$

4.4 述語の解説

述語 $accepts(R, X)$ は、文字列 X が正規表現 R に受理されるかどうかを計算する。これは、空列を受理する場合と、有限個の文字列を受理する場合、それぞれの計算を行う。

述語 $accepts\_empty(R)$ は、正規表現 R が空文字列つまり、εを受理するかどうかを計算する。

述語 $next(R, S, R1)$ は、S がΣ上の文字、X がΣ上の文字列であり、Z を受理する正規表現 $R1$ が存在し、文字列 $X = S; Z$ が R に受理される場合に適用され、結果的には $R1$ を計算することになる。

ここで、$R1$ は、$transf(R, TR)$、$deriv(TR, S, L)$ および、$make\_regexpr(L, R1)$ によって計算される。

述語 $transf(R, TR)$ は、正規表現 R を等価な正規表現 TR に変換する。この変換は、正規表現の代数則に従って行われ、TR は集合 $Treg$ として次のとおりに定義される。

$$TReg ::=$$
$$void|void^*|a|b|(a; Reg)|(b; Reg)|(TReg + TReg)$$

この定義から分かるように、変換された正規表現 TR $(E_1 + \cdots + E_n (n \geq 1))$ の $E_1, \cdots, E_n (n \geq 1)$ それぞれの先頭の部分は単純な正規表現 $(void, void^*, a, b)$ で表されている。また、述語 $transf(R, TR)$ によって変換された正規表現は、必ず述語 $deriv(TR, S, L)$ が適用できる形になっている。

述語 $deriv(TR, S, L)$ は、変形された正規表現 TR によって文字列 X の先頭の文字 S が受理されるかどうかを判定する。判定後、S が受理される場合は正規表現の残りの部分がリストの要素として、受理されない場合は空リストとして L に代入される。正規表現 TR は $E_1 + \cdots + E_n (n \geq 1)$ の形で与えられているので、L は $(E_1, \cdots, E_n)$ のようなリストになる。

述語 $make\_regexpr(L, R1)$ は、リスト L から正規表現 $R1$ を計算して導く。複数の正規表現 $E_1, \cdots, E_{n-1}, E_n$ の要素で構成されるリストから計算された正規表現 $R1$ は、$(E_1 + (\cdots + (E_{n-1} + E_n) \cdots))$ の形で表される。

5 特定の正規表現によって受理される文字列の判定

ある特定の正規表現に注目した場合、前章で示した定義節に対して3章のアルゴリズムを適用すると、任意の文字列すべてについて正規表現によって受理されるかどうかを yes、no で判定することができる。以下では、その例を示す。

正規表現 $R = (a + b; a^*; b)^*; b$ が与えられた場合について、以下のような節を定義する。

$$s(X) \leftarrow accepts((a + b; a^*; b)^*; b, X).$$

これと 4.3 節の定義節をを宣言的記述で表し、3章で示したアルゴリズムを適用する。その結果、Σ 上の文字列 X が Σ 上の正規表現 R に受理されるどうかを判定するための、以下のような等価変換ルールが生成される。

$$s([]) \rightarrow \langle false \rangle.$$
$$s([a|\&X]) \rightarrow s(\&X).$$
$$s([b|\&X]) \rightarrow s1(\&X).$$
$$s1([]) \rightarrow \langle true \rangle.$$
$$s1([a|\&X]) \rightarrow s2(\&X).$$
$$s1([b|\&X]) \rightarrow s(\&X).$$
$$s2([]) \rightarrow \langle false \rangle.$$
$$s2([a|\&X]) \rightarrow s2(\&X).$$
$$s2([b|\&X]) \rightarrow s(\&X).$$

ここで、$h([x|\&X]) \rightarrow k(\&X).$ の変換では、ラベル x の弧で状態 h と状態 k が結びつけられていて、s が初期状態であり、$sh([]) \rightarrow.$ が状態 sh が最終状態であることを表していると考えると、これは正規表現 $R = (a + b; a^*; b)^*; b$ で記述された言語を受理する決定性有限オートマトンを表していると言える。また、このルールを用いることによって、文字列 X が正規表現 $(a+b; a^*; b)^*; b$ によって受理されるかどうかの判定をより効率的に行うことができる。

6 ルールの生成例

本章では、5.1節で示したルールの生成について例を示す。

6.1 ルール生成例 1

まず、以下のような節を定義する。

$$s(X) \leftarrow accepts((a+b;a^*;b)^*;b,X).$$

この節と4.3節で示した定義節から、メタルールを作る。以下に、用いるメタルールの一部を示す。

$$\begin{aligned}
&accepts(*R,*X) \rightarrow equal(*X,[]),\\
&\qquad\qquad\qquad accepts_empty(*R).\\
&\qquad\qquad\rightarrow equal(*X,[\%S|\%X1]),\\
&\qquad\qquad\quad next(*R,\%S,\%R1),\\
&\qquad\qquad\quad accepts(\%R1,\%X1).\\
&s(*X) \rightarrow accepts((a+b;a^*;b)^*;b,*X).\\
&accepts((a+b;a^*;b)^*;b,*X) \rightarrow s(*X).
\end{aligned}$$

6.2 ルール生成例 2

6.2.1 文字列 []

メタ節

$head \leftarrow s([]).$ を変換していくと、

$$\begin{aligned}
&head \leftarrow accepts((a+b;a^*;b)^*;b,[]).\\
&head \leftarrow accepts_empty((a+b;a^*;b)^*;b).\\
&\qquad\cdots\cdots
\end{aligned}$$

となり、節が消滅する。これにより、

$s([]) \rightarrow \langle false \rangle.$

というルールが生成される。

6.2.2 文字列 $[a|\&X]$

メタ節

$head \leftarrow s([a|\&X]).$ を変換していくと、

$$\begin{aligned}
&head \leftarrow accepts((a+b;a^*;b)^*;b,[a|\&X]).\\
&head \leftarrow next((a+b;a^*;b)^*;b,a,\#R1),\\
&\qquad\qquad accepts(\#R1,\&X).\\
&\qquad\cdots\cdots\\
&head \leftarrow accepts((a+b;a^*;b)^*;b,\&X).\\
&head \leftarrow s(\&X).
\end{aligned}$$

となる。これにより、

$s([a|\&X]) \rightarrow s(\&X).$

というルールが生成される。

6.2.3 文字列 $[b|\&X]$

メタ節

$head \leftarrow s([b|\&X]).$ を変換していくと、

$$\begin{aligned}
&head \leftarrow accepts((a+b;a^*;b)^*;b,[b|\&X]).\\
&head \leftarrow next((a+b;a^*;b)^*;b,b,\#R1),\\
&\qquad\qquad accepts(\#R1,\&X).\\
&\qquad\cdots\cdots\\
&head \leftarrow accepts(void^*+a^*;b;(a+b;a^*;b)^*;b,\&X).
\end{aligned}$$

となる。

6.2.4 節の定義

ここで、アトム $s1(\&X)$ を生成し、以下のような節を定義する。

$$\begin{aligned}
&s1(X) \leftarrow\\
&\qquad accepts(void^*+a^*;b;(a+b;a^*;b)^*;b,X).
\end{aligned}$$

この節から、以下のような新たなメタルールを作る。

$$\begin{aligned}
&s1(*X)\\
&\qquad\rightarrow accepts(void^*+a^*;b;(a+b;a^*;b)^*;b,*X).\\
&accepts(void^*+a^*;b;(a+b;a^*;b)^*;b,*X)\\
&\qquad\rightarrow s1(*X).
\end{aligned}$$

これをメタルール群に加え、変換を続ける。ここでは、

$$\begin{aligned}
&head \leftarrow accepts(void^*+a^*;b;(a+b;a^*;b)^*;b,\&X).\\
&head \leftarrow s1(\&X).
\end{aligned}$$

と変換が進み、

$s([b|\&X]) \rightarrow s1(\&X).$

というルールが生成される。

6.3 これ以降のルール生成過程

これ以降のルール生成の過程を以下に簡単に示す。

1. メタ節 $head \leftarrow s1([])$ の変換
2. ルール $s1([]) \rightarrow \langle ture \rangle$ の生成
3. メタ節 $head \leftarrow s1([a|\&X])$ の変換
4. アトム $s2(\&X)$ の生成
5. 確定節を定義し、メタルールを追加
6. ルール $s1([a|\&X]) \rightarrow s2(\&X)$ の生成
7. メタ節 $head \leftarrow s1([b|\&X])$ の変換
8. ルール $s1([b|\&X]) \rightarrow s(\&X)$ の生成
9. メタ節 $head \leftarrow s2([])$ の変換
10. ルール $s2([]) \rightarrow \langle false \rangle$ の生成

11. メタ節 $head \leftarrow s2([a|\&X])$ の変換
12. ルール $s2([a|\&X]) \rightarrow s2(\&X)$ の生成
13. メタ節 $head \leftarrow s2([b|\&X])$ の変換
14. ルール $s2([b|\&X]) \rightarrow s(\&X)$ の生成

この段階で、すべてのルールが評価基準を満たす形で得られ、ルール生成が完了する。

7 比較

4.3節で示した定義節から、直接導かれる等価変換ルールを用いることによって、受理を判定することは可能である。例えば、

$R = (a + b; a^*; b)^*; b$
$X = [a, b, b, a, b]$

を与えた場合には、以下のように変換が行われ、

$$
\begin{aligned}
yes \leftarrow\ & accepts((a + b; a^*; b)^*; b, [a, b, b, a, b]) \\
yes \leftarrow\ & next((a + b; a^*; b)^*; b, a, R1), \\
& accepts(R1, [b, b, a, b]). \\
yes \leftarrow\ & transf((a + b; a^*; b)^*; b, TR), \\
& deriv(TR, a, L), \\
& make\_regexpr(L, R1), \\
& accepts(R1, [b, b, a, b]). \\
& \cdots\cdots \\
yes \leftarrow\ & next(void^* + a^*; b; (a + b; a^*; b)^*; b, b, R3), \\
& accepts(R3, [a, b]). \\
& \cdots\cdots \\
yes \leftarrow .&
\end{aligned}
$$

yes 節は単位節に変換される。したがって、これは yes となる。

ここで、用いた等価変換ルールの一部を以下に示す。

$$
\begin{aligned}
accepts(\&R, \&X) \rightarrow\ & equal(\&X, []), \\
& accepts\_empty(\&R). \\
\rightarrow\ & equal(\&X, [\#S|\#X1]), \\
& next(\&R, \#S, \#R1), \\
& accepts(\#R1, \#X1).
\end{aligned}
$$

この等価変換ルールは、メタルールを作る場合と同様の手順で作られるが、変数は、*変数の代わりにに&変数を、%変数の代わりに#変数を用いる。

上のルールと生成した5.1節のルール、それぞれで等価変換を行った場合を比較すると、

上のルール	905 ステップ 200msec
生成したルール	6 ステップ 1msec 以下

で計算することができる。これを比較すると、計算コストということに関して、生成したルールの方が効率的だと言える。

8 おわりに

本論文では、アルゴリズムを生成するということを純粋な確定節レベルの仕様から等価変換ルールの集合を生成すると捉え、その方法を示すアルゴリズムを提案した。これは、" 新しい節の定義"、" メタルールの追加"、" 評価に基づいたルールの選択"、" アトムパターンの生成" のステップがあることが特徴である。

今後は、計算機システム上で実行するための実装を試みるとともに、このアルゴリズムを様々な例題に対し適用する。それによって、アルゴリズムを拡張する必要性が出てくる可能性がある。特に、" 新しい節の定義" ということに関して、様々なパターンを試し、拡張していくことが必要である。

参考文献

[1] 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)

[2] 赤間清, 清水伴訓, 宮本衛市:宣言的プログラムの等価変換による問題解決, 人工知能学会誌, Vol.13, No.6, pp.944-952 (1998)

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

[4] 小池英勝, 赤間清, 宮本衛市:等価変換ルール生成の基礎理論, 情報処理学会研究報告, 98-ICS-144,pp.13-18 (1998)

[5] 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)

[6] Sato T. and Tamaki H.: *Transformational Logic Program Synthesis*, Proc. of The International Conference on Fifth Generation Computer Systems 1984, pp.195-201 (1984)

[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).