パターンに基づくプログラム変換システム

Program Transformation System based on Template

千葉勇輝　青戸等人　外山芳人
Yuki CHIBA　Takahito AOTO　Yoshihito TOYAMA

東北大学 電気通信研究所
RIEC, Tohoku University
{chiba,aoto,toyama}@nue.riec.tohoku.ac.jp

パターンに基づくプログラム変換は Huet と Lang (1978) により提案されたプログラム効率化の手法である．千葉ら (2005) は，項書き換えを計算モデルとして用いたパターンに基づくプログラム変換の枠組みを提案した．この枠組みでは，項書き換えにおける定理自動証明の手法を応用することで，発展的変換パターンに基づくプログラム変換の正当性を自動的に検証することが出来る．本論文では，この枠組みを実現したシステム RAPT について報告する．RAPT は，与えられた発展的変換パターンに基づいて与えられた項書き換えシステムを変換するとともに，その変換の正当性を自動的に検証する．

1　はじめに

パターンに基づくプログラム変換は Huet と Lang により提案されたプログラム効率化の手法である [5]．彼らは，パターン照合にラムダ計算の第 2 階照合アルゴリズムを利用し，パターンに基づくプログラム変換を実現した．以後，より柔軟かつ効率的なプログラム変換を実現するため，様々な照合アルゴリズムの研究が行われている [2, 3, 9]．一方，Huet と Lang は表示的意味論に基づく変換の正当性の検証法も与えているが，以後，変換の正当性についてはあまり研究が行われていない．

著者らは文献 [1] において，項書き換えを計算モデルに用いたパターンに基づくプログラム変換の枠組みを提案した．この枠組みでは，プログラムは項書き換えシステム，変換パターンはパターン変数を含む項書き換えシステムにより記述される．そして，項パターン照合アルゴリズムにより，パターンに基づくプログラム変換を実現している．

また，この枠組みでは，項書き換えにおける定理自動証明の手法を応用し，発展的変換パターンに基づくプログラム変換の正当性を自動的に検証することが出来る．検証に成功するとき，入出力プログラムの等価性が操作的意味論に基づき保証される．変換の正当性を保証するためのプログラムの帰納的性質も帰納的定理証明法を用いて自動証明される．プログラムの帰納的性質の検証は，Huet と Lang の検証法ではユーザー自身の保証に委ねられていることに注意しておく．

本論文では，この枠組みを実現したプログラム変換システム RAPT (Rewriting-based Automated Program Transformation system) について報告する．RAPT は，与えられた発展的変換パターンに基づいて与えられた項書き換えシステムを変換するとともに，その変換の正当性を自動的に検証する．

2　パターンに基づくプログラム変換

以下に，リストの総和を再帰的に求める関数 sum を定義する項書き換えシステム $\mathcal{R}$ を示す．ここで，$0, \mathsf{s}(0), \mathsf{s}(\mathsf{s}(0)), \ldots$ は自然数 $0, 1, 2, \ldots$ を表す．

$$\mathcal{R} \left\{ \begin{array}{lcl} \mathsf{sum}([\,]) & \to & 0 \\ \mathsf{sum}(x : y) & \to & +(x, \mathsf{sum}(y)) \\ +(0, x) & \to & x \\ +(\mathsf{s}(x), y) & \to & \mathsf{s}(+(x, y)) \end{array} \right.$$

変換パターンはパターン変数を用いて抽象化した項書き換えシステム (項書き換えパターン) によって構成される．以下の変換パターン $\mathcal{P} \Rightarrow \mathcal{P}'$ を用いて項書き換えシステム $\mathcal{R}$ を変換することを考える．

$$\mathcal{P} \left\{ \begin{array}{lcl} \mathsf{f}(\mathsf{a}) & \to & \mathsf{b} \\ \mathsf{f}(\mathsf{c}(u, v)) & \to & \mathsf{g}(u, \mathsf{f}(v)) \\ \mathsf{g}(\mathsf{b}, u) & \to & u \\ \mathsf{g}(\mathsf{d}(u, v), w) & \to & \mathsf{d}(u, \mathsf{g}(v, w)) \end{array} \right.$$

$$\mathcal{P}'\ \left\{\begin{array}{lcl} \mathsf{f}(u) & \to & \mathsf{f}_1(u,\mathsf{b}) \\ \mathsf{f}_1(\mathsf{a},u) & \to & u \\ \mathsf{f}_1(\mathsf{c}(u,v),w) & \to & \mathsf{f}_1(v,\mathsf{g}(w,u)) \\ \mathsf{g}(\mathsf{b},u) & \to & u \\ \mathsf{g}(\mathsf{d}(u,v),w) & \to & \mathsf{d}(u,\mathsf{g}(v,w)) \end{array}\right.$$

項書き換えシステム $\mathcal{R}$ を変換パターン $\mathcal{P} \Rightarrow \mathcal{P}'$ を用いて変換するには，項書き換えパターン $\mathcal{P}$ と項書き換えシステム $\mathcal{R}$ とのパターン照合問題を解く必要がある．文献 [1] において著者らは，パターン照合問題の解を項準同型写像によって表わし，項パターン照合問題を解くアルゴリズム **Match** を提案した．アルゴリズム **Match** を用いると，項書き換えパターン $\mathcal{P}$ と項書き換えシステム $\mathcal{R}$ のパターン照合問題の解として以下の項準同型写像 φ が得られる．

$$\varphi = \left\{\begin{array}{ll} \mathsf{f} \mapsto \mathsf{sum}(\Box_1), & \mathsf{g} \mapsto +(\Box_1,\Box_2), \\ \mathsf{a} \mapsto [\,], & \mathsf{b} \mapsto \mathsf{0}, \\ \mathsf{c} \mapsto \Box_1{:}\Box_2, & \mathsf{d} \mapsto \mathsf{s}(\Box_2) \end{array}\right\}$$

項準同型写像 φ を項書き換えパターン $\mathcal{P}'$ に適用することで以下の項書き換えシステム $\mathcal{R}'$ が得られる．

$$\mathcal{R}'\ \left\{\begin{array}{lcl} \mathsf{sum}(x) & \to & \mathsf{sum1}(x,\mathsf{0}) \\ \mathsf{sum1}([\,],x) & \to & x \\ \mathsf{sum1}(x{:}y,z) & \to & \mathsf{sum1}(y,+(z,x)) \\ +(\mathsf{0},x) & \to & x \\ +(\mathsf{s}(x),y) & \to & \mathsf{s}(+(x,y)) \end{array}\right.$$

この $\mathcal{R}'$ が変換によって得られる項書き換えシステムとなる．ここで，パターン変数 f_1 は $\mathcal{P}$ に出現しないためパターン照合の解 φ の定義域にも出現しないが，可読性のため，ここでは $\mathcal{R}'$ 中の f_1 を $\mathsf{sum1}$ とおいている．

入力項書き換えシステム $\mathcal{R}$ と出力項書き換えシステム $\mathcal{R}'$ を比較すると，$\mathcal{R}$ は再帰的に関数 sum を定義しているのに対し，$\mathcal{R}'$ では反復的な定義となっている．再帰的なプログラムより反復的なプログラムの方が効率的であることはよく知られている．

3 プログラム変換の正当性

プログラム変換の正当性を保証するためには，プログラムの帰納的な性質が必要になる場合がある．例えば，前節における項書き換えシステム $\mathcal{R}$ から $\mathcal{R}'$ へのプログラム変換において，変換の正当性を保証するためには関数 $+$ の結合律と $+(\mathsf{0},n) = +(n,\mathsf{0})$ という 2 つの帰納的性質を必要とする．そこで，変換の正当性を保証する際必要となる帰納的性質を抽象化し，変換パターンに組み込む．

定義 1 (変換パターン) パターン変数を含む項を項パターンという．項パターン上の項書き換えシステムを項書き換えパターン，項パターンによる等式の有限集合を仮定とよぶ．変換パターンとは，2 つの項書き換えパターン $\mathcal{P},\mathcal{P}'$ と仮定 $\mathcal{H}$ の組 $\langle \mathcal{P},\mathcal{P}',\mathcal{H}\rangle$ のことである．

変換パターンに基づくプログラム変換は，形式的には以下のように定義される．

定義 2 (パターンに基づくプログラム変換) $\mathscr{G}$ を関数記号の集合とする．以下の 3 つの条件を満たす CS 準同型写像 φ が存在するとき，項書き換えシステム $\mathcal{R}$ が変換パターン $\langle \mathcal{P},\mathcal{P}',\mathcal{H}\rangle$ によって項書き換えシステム $\mathcal{R}'$ に $\mathscr{G}$ のもとで変換されるという．

1. $\mathcal{R} = \varphi(\mathcal{P})$,
2. $\mathcal{R}' = \varphi(\mathcal{P}')$,
3. $\varphi(\mathcal{H})$ が $\mathscr{G}$ のもとで $\mathcal{R}$ の帰納的定理となる．

定義 2 における条件 3 は変換パターンの仮定部分を照合解により具体化したものが，入力プログラムの帰納的性質となっていることを表わしている．また，項準同型写像が CS 準同型写像であることは，以下の定理 1 が成立するために必要となる．関数記号の集合 $\mathscr{G}$ がパラメータとなっているのは，正当性の議論において必要なためである．

変換の正当性を定式化するためには，項書き換えシステムの等価性を定義する必要がある．以下では，関数記号の集合を $\mathscr{F}$，構成子記号の集合を $\mathscr{F}_{\mathrm{c}}$ と記す．変数の出現しない項を基底項とよび，関数記号の集合 $\mathscr{G} \subseteq \mathscr{F}$ 上の基底項の集合を $\mathrm{T}(\mathscr{G})$ と記す．また，項書き換えシステム $\mathcal{R}$ による簡約関係を $\to_{\mathcal{R}}$ で表わし，その反射推移閉包を $\overset{*}{\to}_{\mathcal{R}}$ と記す．

定義 3 (項書き換えシステムの等価性) $\mathscr{G}$ を $\mathscr{F}_{\mathrm{c}} \subseteq \mathscr{G} \subseteq \mathscr{F}$ となる関数記号の集合とし，$\mathcal{R},\mathcal{R}'$ を項書き換えシステムとする．任意の基底項 $s \in \mathrm{T}(\mathscr{G})$ と基底構成子項 $t \in \mathrm{T}(\mathscr{F}_{\mathrm{c}})$ に対して，$s \overset{*}{\to}_{\mathcal{R}} t$ の時かつその時に限り $s \overset{*}{\to}_{\mathcal{R}'} t$ となるとき，$\mathcal{R}$ と $\mathcal{R}'$ は $\mathscr{G}$ のもとで等価であるという．

定義 3 による項書き換えシステムの等価性は，考慮する関数記号の集合を制限していることに注意する．プログラム変換において，一方のプログラムには出現

```
FUNCTIONS
 sum:  List -> Nat;
 cons:  Nat * List -> List;
 nil:  List;
 +:  Nat * Nat -> Nat;
 s:  Nat -> Nat;
 0:  Nat

RULES
 sum(nil()) -> 0();
 sum(cons(x,ys)) -> +(x,sum(ys));
 +(0(), x) -> x;
 +(s(x),y) -> s(+(x,y))
```

```
INPUT
 ?f(?a()) -> ?b();
 ?f(?c(u,v)) -> ?g(u,?f(v));
 ?g(?b(),u) -> u;
 ?g(?d(u,v),w) -> ?d(u,?g(v,w))

OUTPUT
 ?f(u) -> ?f1(u,?b());
 ?f1(?a(),u) -> u;
 ?f1(?c(u,v),w) -> ?f1(v,?g(w,u));
 ?g(?b(),u) -> u;
 ?g(?d(u,v),w) -> ?d(u,?g(v,w))

HYPOTHESIS
 ?g(?b(),u) = ?g(u,?b());
 ?g(?g(u,v),w) = ?g(u,?g(v,w))
```

図 1: RAPT に対する入力例

するが，もう一方には出現しない関数記号が存在することが多い．例えば，前節の項書き換えシステム $\mathcal{R}$, $\mathcal{R}'$ において，sum1 は $\mathcal{R}'$ には出現するが $\mathcal{R}$ には出現しない．そのため，例えば $\mathsf{sum1}([1,2,3],0) \xrightarrow{*}_{\mathcal{R}'} 6$ だが，$\mathsf{sum1}([1,2,3],0) \xrightarrow{*}_{\mathcal{R}} 6$ とはならない．このため，$\mathsf{sum1} \notin \mathscr{G}$ なる関数記号の集合 $\mathscr{G}$ のもとで等価性を考える必要がある．

文献 [1] において，著者らは，発展的変換パターンの概念を導入し，同じ変換パターンを用いたプログラム変換の正当性検証における共通部分をパターンの段階に抽象化した．発展的変換パターンによる変換の正当性は以下のように検証される．

定理 1 (Theorem 2 of [1]) $\mathscr{G}, \mathscr{G}'$ を $\mathscr{F}_{\mathrm{c}} \subseteq \mathscr{G}, \mathscr{G}' \subseteq \mathscr{F}$ となる関数記号の集合とする．また，$\mathscr{G}$ 上の項書き換えシステム $\mathcal{R}$ が，発展的変換パターン $\langle \mathcal{P}, \mathcal{P}', \mathcal{H} \rangle$ によって $\mathscr{G}'$ 上の項書き換えシステム $\mathcal{R}'$ へ $\mathscr{G}$ のもとで変換されるとする．このとき，以下の 3 つの条件が成立するならば，$\mathcal{R}$ と $\mathcal{R}'$ は $\mathscr{G} \cap \mathscr{G}'$ のもとで等価である．

1. $\mathcal{R}$ は左線形の構成子システム，
2. $\mathcal{R}$ は $\mathscr{G}$ のもとで十分完全かつ合流性を持つ，
3. $\mathcal{R}'$ は $\mathscr{G}'$ のもとで十分完全である．

定理 1 における入出力項書き換えシステムに必要とされる条件は一般のプログラムを考慮すると自然な条件である．また，発展的変換パターンはいくつかの推論規則を用いて構成する [1]．

4 プログラム変換システム RAPT

RAPT は多ソート項書き換えシステムをプログラム変換の対象とする．文献 [1] の理論的枠組みが多ソート項書き換えシステムに対してもそのまま適用できることは容易に確認できる．多ソート項書き換えシステムを用いる理由は，定理 1 に基づき変換の正当性を保証するためには，入出力項書き換えシステムの十分完全性を検証する必要があるが，通常のプログラムは，多ソート項書き換えシステムでモデル化しないと十分完全にならないためである．

RAPT は与えられた多ソート項書き換えシステムを，与えられた変換パターンに基づいて変換し，得られた多ソート項書き換えシステムと入力された多ソート項書き換えシステムの等価性の検証に成功した場合，それを出力する．ただし，等価性の保証には入力される変換パターンが発展的である必要がある．RAPT に対する入力例を図 1 に示す．FUNCTIONS および RULES セクションは入力多ソート項書き換えシステムを，INPUT, OUTPUT, HYPOTHESIS セクションは変換パターンを与えている．

RAPT は 6 つの部分から構成されている．RAPT の全体構成を図 2 に示す．実線の矢印はデータの流れを表わし，点線の矢印は各部分で得られた情報がどこで用いられるかを示している．

1. 入力検証部
   入力項書き換えシステムが左線形構成子システ

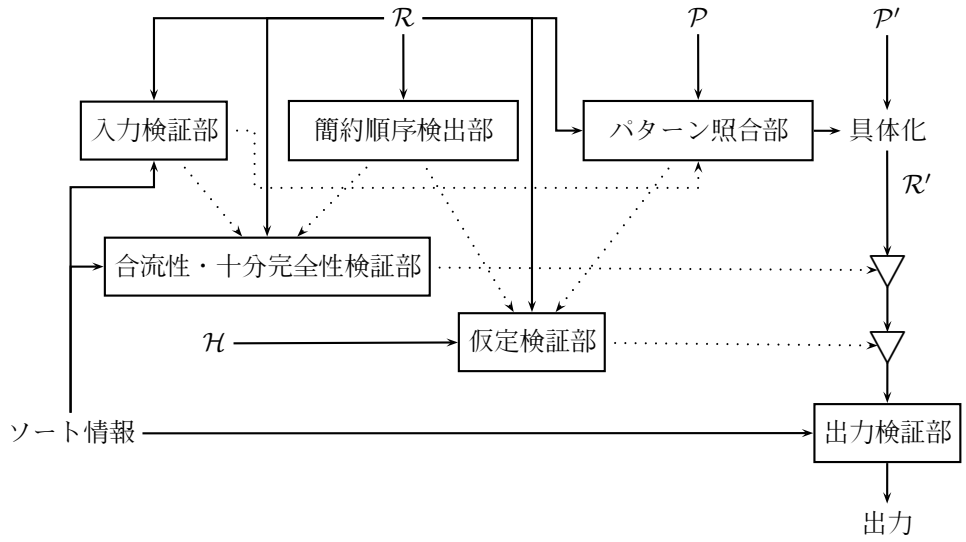

図 2: RAPT の全体構成

ムであるか判定し，型チェックを行う．

2. 簡約順序検出部
   辞書式経路順序に基づいて入力項書き換えシステムの停止性証明を行う．証明に成功した場合，関数記号の優位順序を検出する．優位順序の検出には，文献 [4] の制約解消アルゴリズムを利用している．

3. 合流性・十分完全性検証部
   入力項書き換えシステムの合流性および十分完全性の自動証明を行う．入力項書き換えシステムの停止性検証はすでに済んでいるため，合流性の判定には危険対の合流性を判定すればよい．同様に，十分完全性についても擬簡約性を判定すればよい．擬簡約性の判定には補代入アルゴリズム [6, 8] を用いた．

4. パターン照合部
   照合アルゴリズム **Match** を用いて項書き換えパターンから項書き換えシステムへの CS 準同型写像を見つける．**Match** によるパターン照合は，項書き換えパターンと項書き換えシステムの照合問題を書き換え規則の対応関係に応じた項パターンと項の項パターン照合問題に帰着する．RAPT では，主関数から副関数へと先に得られた照合解を利用しながら照合問題を解いている．

5. 仮定検証部
   パターン照合部で得られた CS 準同型写像を用いて仮定を具体化し等式を得る．その等式が入力項書き換えシステムの帰納的定理であるかを，書き換え帰納法 [7] を用いて自動証明する．書き換え帰納法には，入力プログラムの停止性を保証する簡約順序が必要だが，これは簡約順序検出部で得られた関数記号の優位順序から得る．

6. 出力検証部
   得られた出力項書き換えシステムに対して，停止性，左線形性，型整合性，十分完全性を判定する．

以下に図 1 を入力として与えたときの RAPT の出力を示す．

```
[ sum(u) -> f1(u, 0()),
f1(nil(), u) -> u,
f1(cons(u, v), w) -> f1(v, +(w, u)),
+(0(), u) -> u,
+(s(v), w) -> s(+(v, w)) ]
```

5 結論

本研究ではパターンに基づくプログラム自動変換システム RAPT をその理論的枠組みと合わせて紹介した．RAPT は多ソート項書き換えシステムを対象としたプログラム変換システムで，従来ほとんど取り扱われていない正当性の検証を，定理自動証明の手法を応用することで自動的に行う．RAPT の実装には Standard ML of New Jersey (`http://www.smlnj.org/`) を用いた．ソースコードは 5,000 行程度である．

パターンに基づくプログラム変換の実装としては MAG が知られている [3]．MAG と RAPT の大きな

違いはプログラムが持つ帰納的性質の検証へのアプローチである．MAG ではユーザーに委ねられているのに対し，RAPT では定理自動証明の手法を利用することで自動的に検証を行う．定理自動証明の技術をプログラム変換に応用したシステムはほとんど知られておらず，RAPT はプログラム変換技術と定理自動証明技術の結合へ向けた最初のステップと考えられる．

謝辞

有益な示唆を頂いた菊池健太郎氏に感謝します．本研究の一部は，科学研究費 14580357, 17700002, 16016202 の補助を受けて行われた．

参考文献

[1] Y. Chiba, T. Aoto, and Y. Toyama. Program transformation by templates based on term rewriting. In *Proceedings of the 7th ACM-SIGPLAN International Conference on Principles and Practice of Declarative Programming (PPDP 2005)*, pages 59–69. ACM Press, 2005.

[2] R. Curien, Z. Qian, and H. Shi. Efficient second-order matching. In *Proceedings of the 7th International Conference on Rewriting Techniques and Applications*, volume 1103 of *LNCS*, pages 317–331. Springer-Verlag, 1996.

[3] O. de Moor and G. Sittampalam. Higher-order matching for program transformation. *Theoretical Computer Science*, 269:135–162, 2001.

[4] N. Hirokawa and A. Middeldorp. Tsukuba termination tool. In *Proceedings of the 14th International Conference on Rewriting Techniques and Applications*, volume 2706 of *LNCS*, pages 311–320. Springer-Verlag, 2003.

[5] G. Huet and B. Lang. Proving and applying program transformations expressed with second order patterns. *Acta Informatica*, 11:31–55, 1978.

[6] A. Lazrek, P. Lescanne, and J. J. Thiel. Tools for proving inductive equalities, relative completeness, and ω-completeness. *Information and Computation*, 84:47–70, 1990.

[7] U. S. Reddy. Term rewriting induction. In *Proceedings of the 10th International Conference on Automated Deduction*, volume 449 of *LNAI*, pages 162–177, 1990.

[8] J. J. Thiel. Stop loosing sleep over incomplete data type specifications. In *Proceedings of the 11th Annual ACM Symposium on Principles of Programming Languages*, pages 76–82, 1984.

[9] T. Yokoyama, Z. Hu, and M. Takeichi. Deterministic second-order patterns. *Information Processing Letters*, 89(6):309–314, 2004.