社団法人　電子情報通信学会
THE INSTITUTE OF ELECTRONICS,
INFORMATION AND COMMUNICATION ENGINEERS

信学技報
TECHNICAL REPORT OF IEICE.

# 融合変換を模倣するプログラム生成変換の戦略

長島　正憲†　酒井　正彦†　西田　直樹†　坂部　俊樹†　草刈圭一朗†

† 名古屋大学大学院情報科学研究科　〒 464–8603 名古屋市千種区不老町
E-mail: † nagashima@sakabe.i.is.nagoya-u.ac.jp, {sakai,nishida,sakabe,kusakari}@is.nagoya-u.ac.jp

**あらまし**　等号付き一階述語論理に基づくプログラム生成変換 $\mathcal{GS}$ [1], [2] は，一般の等式論理では扱うことのできない限量子や論理和を含む仕様を取り扱うことができる．$\mathcal{GS}$ は，実行不可能な仕様から実行可能なプログラムを生成できる場合があり，プログラムの自動生成に応用が期待できる．しかし，どのようなプログラムや仕様に対してプログラム生成が成功するかは全く不明であった．本稿では，プログラム融合変換として知られる Deforestation [3] を $\mathcal{GS}$ が模倣できることを示す．

**キーワード**　プログラム融合変換，プログラム生成，限量子付き等式仕様，項書換え系，被覆集合

# Simulating Fusion Transformation by Program-Generation Transformation

Masanori NAGASHIMA†, Masahiko SAKAI†, Naoki NISHIDA†,

Toshiki SAKABE†, and Keiichirou KUSAKARI†

† Graduate School of Information Science, Nagoya University
Furo-cho, Chikusa-ku, Nagoya, 464–8603 Japan
E-mail: † nagashima@sakabe.i.is.nagoya-u.ac.jp, {sakai,nishida,sakabe,kusakari}@is.nagoya-u.ac.jp

**Abstract**　Program-Generation Transformation $\mathcal{GS}$ [1], [2] based on first-order logic with equality can handle specifications with quantifiers and logical ORs in them. There are some cases in which $\mathcal{GS}$ can transform unexecutable specifications into executable programs, so $\mathcal{GS}$ is expected to be applied to automatic program generation. However, we have not known conditions for specifications when the program generation succeeds. In this report, we show that $\mathcal{GS}$ can simulate Deforestation [3] known as a fusion transformation.

**Key words**　fusion transformation, program generation, quantified equational specification, TRS, covering set

## 1. はじめに

我々が提案したプログラム生成変換 $\mathcal{GS}$ は，そのままでは実行不可能な未知プログラムの仕様を論理式で表し，実行可能な既知プログラムを表す項書換え系の下でその論理式を変換していくことで，未知プログラムの実行可能な定義を導出することを目的としたシステムである．例えば，自然数を 1/2 倍する既知プログラム $\mathsf{h}$ を表す TRS は

$$\mathcal{R}_1 = \left\{ \begin{array}{c} \mathsf{h}(\mathsf{0}) \to \mathsf{0}, \\ \mathsf{h}(\mathsf{s}(\mathsf{0})) \to \mathsf{0}, \\ \mathsf{h}(\mathsf{s}(\mathsf{s}(x))) \to \mathsf{s}(\mathsf{h}(x)) \end{array} \right\}$$

と表現できる．このとき，自然数を 2 倍する未知プログラム $\mathsf{d}$ の仕様となる論理式集合は

$$\mathcal{E}_1 = \left\{ \begin{array}{c} \forall x.\, \mathsf{d}(\mathsf{h}(x)) \approx x, \\ \forall x.\, \mathsf{h}(\mathsf{d}(x)) \approx x \end{array} \right\}$$

と書ける．$\mathcal{E}_1$ は $\mathsf{d}$ の定義としては十分であるが，プログラムとして実行可能ではない．従来の等式論理の枠組みでは $\mathcal{E}_1$ から実行可能な $\mathsf{d}$ の定義を得ることはできなかった．しかし論理的な意味に着目すると，仕様 $\forall x.\, \mathsf{h}(\mathsf{d}(x)) \approx x$ を満足するには $\exists$ を利用して展開した別の仕様 $\forall x.\, \exists y.\, (\mathsf{d}(x) \approx y \wedge \mathsf{h}(y) \approx x)$ を満足すればよいことが分かる．さらに $\mathsf{d}$ をプログラムとみなすならば，限量子の解釈としてデータのみを考慮すれば十分であるので，$\exists y.$ を $\vee$ で場合わけした仕様

$$\forall x. \left( \begin{array}{ll} & \mathsf{d}(x) \approx \mathsf{0} \wedge \mathsf{h}(\mathsf{0}) \approx x \\ \vee & \mathsf{d}(x) \approx \mathsf{s}(\mathsf{0}) \wedge \mathsf{h}(\mathsf{s}(\mathsf{0})) \approx x \\ \vee & \exists z. \left( \mathsf{d}(x) \approx \mathsf{s}(\mathsf{s}(z)) \wedge \mathsf{h}(\mathsf{s}(\mathsf{s}(z))) \approx x \right) \end{array} \right)$$

を満たすことで元の仕様を満たしていることがいえる．このよ

うに等式論理の枠組みにとどまらず，限量子や論理和を扱うことのできる一階述語論理の枠組みで変換を考えることで，従来行われなかったプログラム生成が可能になる．$\mathcal{GS}$ はプログラム生成のために本質であると思われる変換を一般化し変換規則として与えたものである．これらの変換規則は，変換結果が仕様を満足することを保証しているので，$\mathcal{GS}$ はプログラム生成システムとして健全である．

しかし，逆に $\mathcal{GS}$ にどのような既知プログラムや仕様を与えた場合にプログラム生成が成功するか（プログラム生成システムとしての完全性が成り立つ十分条件）は全く知られていなかった．$\mathcal{GS}$ をプログラムの自動的な生成に応用するためには，変換の適用範囲に関する議論が不可欠である．そこで本稿では，関数型プログラムにおける合成関数の効率化手法である Deforestation を取り上げ，その変換を $\mathcal{GS}$ が模倣できることを示すことで，Deforestation が対象とするプログラムのクラスにおいて，$\mathcal{GS}$ が完全であることを明らかにする．

## 2. 準　　備

本稿は，一階述語論理と項書換え系の一般的な記法や概念に従う [4], [5], [6].

**論理記号**として $\neg, \wedge, \vee, \forall, \exists$ の 5 個を用いる．$\mathcal{X}$ を**変数**の（可算無限）集合，$\mathcal{F}$ を**関数記号**の集合，$\mathcal{P}$ を**述語記号**の集合とする．各関数記号および述語記号には，**アリティ**と呼ばれる自然数が対応づけられている．アリティが $n$ である関数記号 $f \in \mathcal{F}$ は，$n$ 引数の関数記号であるという．特に，0 引数の関数記号を**定数記号**という．同様に，アリティが $n$ である述語記号 $P \in \mathcal{P}$ は，$n$ 引数の述語記号であるという．特に，0 引数の述語記号を**命題記号**という．

関数記号の集合 $\mathcal{F}$ と変数の集合 $\mathcal{X}$ 上の**項**は，次のように再帰的に定義される．

（1） 変数 $x \in \mathcal{X}$ は項である．

（2） 項 $t_1, \ldots, t_n$ と $n$ 引数の関数記号 $f \in \mathcal{F}$ に対して，$f(t_1, \ldots, t_n)$ は項である．

ただし，項 $f()$ は $f$ と略記する．$\mathcal{F}, \mathcal{X}$ 上のすべての項の集合を $\mathcal{T}(\mathcal{F}, \mathcal{X})$ で表す．変数を含まない項を**基底項**と呼び，その集合 $\mathcal{T}(\mathcal{F}, \emptyset)$ は $\mathcal{T}(\mathcal{F})$ と略記する．項の並び $t_1, \ldots, t_n$ は $\overrightarrow{t}$ のように表現することがある．項 $t$ に現れる変数の集合を $\mathcal{V}ar(t)$ で表す．どの変数も項中に高々1 回しか現れないとき，その項は線形であるという．項 $s$ と $t$ が同一であるときは $s \equiv t$ と記述する．

関数記号の集合 $\mathcal{F}$，述語記号の集合 $\mathcal{P}$，および変数の集合 $\mathcal{X}$ 上の**論理式**は，次のように再帰的に定義される．

（1） 項 $t_1, \ldots, t_n \in \mathcal{T}(\mathcal{F}, \mathcal{X})$ と $n$ 引数の述語記号 $P \in \mathcal{P}$ に対して，**原始論理式** $P(t_1, \ldots, t_n)$ は論理式である．

（2） 論理式 $U, V$ と変数 $x \in \mathcal{X}$ に対して，$(\neg U)$, $(U \wedge V)$, $(U \vee V)$, $(\forall x.U)$, $(\exists x.U)$ はすべて論理式である．

ただし，論理式 $P()$ は $P$ と略記する．また，論理記号の優先順位を $\forall, \exists > \neg > \wedge > \vee$ と定め，括弧を省略する場合がある．$\mathcal{F}, \mathcal{P}, \mathcal{X}$ 上のすべての論理式の集合を**一階言語**といい，$\mathcal{L}(\mathcal{F}, \mathcal{P}, \mathcal{X})$ で表す．$\xi x.$ は $\forall x.$ もしくは $\exists x.$ のいずれかであることを示すために用いる．$\forall x_1. \cdots \forall x_n.U$, $\exists x_1. \cdots \exists x_n.U$, $\xi_1 x_1. \cdots \xi_n x_n.U$ はそれぞれ $\overrightarrow{\forall x}.U$, $\overrightarrow{\exists x}.U$, $\overrightarrow{\xi x}.U$ のように表現することがある．論理式 $U$ に現れる変数の集合を $\mathcal{V}ar(U)$ で表す．論理式 $U, V$ が同一であるときは $U \equiv V$ と記述する．なお，限量子 $\forall, \exists$ により束縛される変数の名前替えによって同一になる論理式はすべて同一であるとみなす．また，$\wedge, \vee$ に関しては交換律と結合律が成り立っているものとする．複数の $\forall$ の並び，あるいは複数の $\exists$ の並びは，変数の束縛の順序を入れ替えても同一であるとする．

述語記号 $\mathcal{P}$ の集合に**等号**と呼ばれる特別な 2 引数述語記号を含む一階言語を**等号付き一階言語**といい，$\mathcal{L}^{\approx}(\mathcal{F}, \mathcal{P}, \mathcal{X})$ のように明示する．また，原始論理式 $\approx (s, t)$ は**等式**といい，$s \approx t$ と略記する．等号の交換律は成り立っているものとする．すなわち，等式 $s \approx t$ と $t \approx s$ は同一である．

変数から項への写像 $\sigma : \mathcal{X} \to \mathcal{T}(\mathcal{F}, \mathcal{X})$ は，$x \neq \sigma(x)$ となる $x$ が有限個であるとき**代入**であるという．代入 $\sigma$ は $x \neq \sigma(x)$ となる $x \mapsto \sigma(x)$ を並べて，$\{x_1 \mapsto \sigma(x_1), \ldots, x_n \mapsto \sigma(x_n)\}$ のように書く．代入 $\sigma$ は項から項への写像に自然に拡張される．すなわち，$\sigma(t)$ は項 $t$ 中の各変数 $x$ を項 $\sigma(x)$ に置き換えて得られる項である．特に，$\sigma(t) \in \mathcal{T}(\mathcal{F})$ となるとき $\sigma$ を**基底代入**という．代入 $\sigma$ は，さらに論理式から論理式への写像に自然に拡張される．すなわち，$\sigma(U)$ は論理式 $U$ 中の各項 $t$ を項 $\sigma(t)$ に置き換えて得られる項である．ただし，$U$ 中の限量子で束縛される変数は $\sigma$ の定義域や値域に含まれない変数になるよう名前替えされているものとする．なお，$\sigma(t), \sigma(U)$ は，それぞれ $t\sigma, U\sigma$ と略記する．

特別な定数記号 $\Box$ をちょうど 1 個含む項 $C[\,] \in \mathcal{T}(\mathcal{F} \cup \{\Box\}, \mathcal{X})$ を**項文脈**と呼ぶ．項文脈 $C[\,]$ の $\Box$ を項 $t$ に置き換えて得られる項を $C[t]$ と表す．同様に，特別な命題記号 $\diamond$ をちょうど 1 個含む論理式 $\Gamma\langle\,\rangle \in \mathcal{L}(\mathcal{F}, \mathcal{P} \cup \{\diamond\}, \mathcal{X})$ を**論理式文脈**と呼ぶ．論理式文脈 $\Gamma\langle\,\rangle$ の $\diamond$ を論理式 $U$ に置き換えて得られる論理式を $\Gamma\langle U\rangle$ と表す．

項 $l, r \in \mathcal{T}(\mathcal{F}, \mathcal{X})$ からなる等式 $l \approx r$ は $l \notin \mathcal{X}$ かつ $\mathcal{V}ar(l) \supseteq \mathcal{V}ar(r)$ であるとき書換え規則といい，$l \to r$ で表す．書換え規則の有限集合を項書換え系 (TRS) という．$\mathcal{R}$ を TRS, $C[\,]$ を項文脈，$\sigma$ を代入，$l \to r \in \mathcal{R}$ とするとき，$s \equiv C[l\sigma]$ は $t \equiv C[r\sigma]$ に書換えられるといい，$s \to_{\mathcal{R}} t$ と書く．$=_{\mathcal{R}}$ は $\to_{\mathcal{R}}$ の反射・推移・対称閉包を表す．

## 3. プログラム生成変換 $\mathcal{GS}$

本節では，プログラム生成変換 $\mathcal{GS}$ [1], [2] を簡単に説明する．$\mathcal{GS}$ は，既知プログラムに相当する TRS と，その下で未知プログラムの仕様を表現した論理式集合から，未知プログラムの実行可能な定義を得るためのシステムである．

プログラム定義や仕様を TRS や論理式とみなすならば，変数割り当てと限量子の解釈には基底項を考えれば十分である．$\mathcal{GS}$ には限量子の解釈を基底項に限定する変換規則が含まれるが，それらの規則は以下で定義される被覆集合 [7] の概念を用いている．

［定義 3.1］（被覆集合, 強被覆集合） $S$ を項の有限集合，$\mathcal{R}$ を

項書換え系とする．任意の基底項 $t \in \mathcal{T}(\mathcal{F})$ に対して，ある項 $s \in S$ とある代入 $\sigma$ が存在して $t =_{\mathcal{R}} s\sigma$ を満たすとき，$S$ を $\mathcal{R}$-**被覆集合**と呼ぶ．特に，各 $t$ に対してそのような $s$ がただ 1 通りに決まるとき，$S$ を $\mathcal{R}$-**強被覆集合**と呼ぶ．すべての $\mathcal{R}$-被覆集合の集合，$\mathcal{R}$-強被覆集合の集合をそれぞれ $\mathcal{CS}(\mathcal{R}), \mathcal{SCS}(\mathcal{R})$ で表す． □

次に，TRS による項上の書換え関係を拡張して，論理式集合 $\mathcal{E}$ 上の書換え関係 $\rightarrow_{\mathcal{E}}$ が次のように定義される．

［定義 3.2］ $U, V$ を論理式とする．ある論理式文脈 $\Gamma\langle\ \rangle$，項文脈 $C[\ ]$，代入 $\sigma$，および，**書換え規則**と呼ばれる閉論理式 $\forall\overrightarrow{x}.\exists\overrightarrow{y}.(l \approx r \wedge W) \in \mathcal{E}$ が存在して

$$U \equiv \Gamma\langle C[l\sigma] \approx t\rangle,\ V \equiv \Gamma\langle \exists\overrightarrow{y}.(C[r\sigma] \approx t \wedge l\sigma \approx r\sigma \wedge W\sigma)\rangle$$

であるとき，$U$ は $V$ に**書き換えられる**といい，$U \rightarrow_{\mathcal{E}} V$ と表す．ただし，書換え規則は束縛変数の名前替えによって，変数条件

$$\mathcal{V}ar(l) = \{\overrightarrow{x}\},\ \{\overrightarrow{y}\} \cap \mathcal{V}ar(\Gamma\langle C[\ ] \approx t\rangle) = \emptyset$$

を満たしているものとする．$=_{\mathcal{E}}$ は $\rightarrow_{\mathcal{E}}$ の反射・推移・対称閉包を表す． □

プログラム生成変換 $\mathcal{GS}$ は次のように定義される．

［定義 3.3］（プログラム生成変換 $\mathcal{GS}$） プログラム生成変換 $\mathcal{GS}$ は，論理式集合と TRS の対に適用される，以下の 6 つの変換規則で構成される．ただし，各変換規則において論理式文脈 $\Gamma\langle\ \rangle$ は単調 [2] である．

（1） **Decomposition**

$$\frac{\mathcal{E} \cup \left\{\Gamma\langle C[t] \approx s\rangle\right\};\ \mathcal{R}}{\mathcal{E} \cup \left\{\Gamma\langle \exists x.(t \approx x \wedge C[x] \approx s)\rangle\right\};\ \mathcal{R}}\ \textit{if}\ \ x \notin \mathcal{V}ar(\Gamma\langle C[t] \approx s\rangle)$$

（2） **Composition**

$$\frac{\mathcal{E} \cup \left\{\Gamma\langle \exists x.(x \approx t \wedge U)\rangle\right\};\ \mathcal{R}}{\mathcal{E} \cup \left\{\Gamma\langle U\sigma\rangle\right\};\ \mathcal{R}}\ \textit{if}\ \ x \notin \mathcal{V}ar(t),\ \sigma = \{x \mapsto t\}$$

（3） $\forall$-**Expansion**

$$\frac{\mathcal{E} \cup \left\{\Gamma\langle \forall x.U\rangle\right\};\ \mathcal{R}}{\mathcal{E} \cup \left\{\Gamma\langle \bigwedge_i \forall\overrightarrow{y_i}.U\sigma_i\rangle\right\};\ \mathcal{R}}\ \textit{if}\ \ \bigcup_i\{t_i\} \in \mathcal{CS}(\mathcal{R}),\ \{\overrightarrow{y_i}\} = \mathcal{V}ar(t_i),\ \sigma_i = \{x \mapsto t_i\}$$

（4） $\exists$-**Expansion**

$$\frac{\mathcal{E} \cup \left\{\Gamma\langle \exists x.U\rangle\right\};\ \mathcal{R}}{\mathcal{E} \cup \left\{\Gamma\langle \bigvee_i \exists\overrightarrow{y_i}.U\sigma_i\rangle\right\};\ \mathcal{R}}\ \textit{if}\ \ \bigcup_i\{t_i\} \in \mathcal{CS}(\mathcal{R}),\ \{\overrightarrow{y_i}\} = \mathcal{V}ar(t_i),\ \sigma_i = \{x \mapsto t_i\}$$

（5） **Deduction**

$$\frac{\mathcal{E} \cup \left\{U\right\};\ \mathcal{R}}{\mathcal{E} \cup \left\{V\right\};\ \mathcal{R}}\ \textit{if}\ \ U \rightarrow_{\mathcal{E} \cup \mathcal{E}_{\mathcal{R}}} V$$

（6） **Var-Elimination**

$$\frac{\mathcal{E} \cup \left\{\Gamma\langle \forall x.(\bigvee_i \exists\overrightarrow{y_i}.(x \approx t_i \wedge U_i))\rangle\right\};\ \mathcal{R}}{\mathcal{E} \cup \left\{\Gamma\langle \bigwedge_i \forall\overrightarrow{y_i}.U_i\sigma_i\rangle\right\};\ \mathcal{R}}\ \textit{if}\ \ \bigcup_i\{t_i\} \in \mathcal{SCS}(\mathcal{R}),\ \{\overrightarrow{y_i}\} = \mathcal{V}ar(t_i),\ \sigma_i = \{x \mapsto t_i\}$$

論理式集合と TRS の対 $\mathcal{E};\mathcal{R}$ が $\mathcal{GS}$ の変換規則を一回適用して $\mathcal{E}';\mathcal{R}$ になったとき $\mathcal{E};\mathcal{R} \Longrightarrow_{\mathcal{GS}} \mathcal{E}';\mathcal{R}$ と書く．$\Longrightarrow_{\mathcal{GS}}^{\mathbf{Dec}}$, $\Longrightarrow_{\mathcal{GS}}^{\mathbf{Com}}$, $\Longrightarrow_{\mathcal{GS}}^{\forall\textbf{-Exp}}$, $\Longrightarrow_{\mathcal{GS}}^{\exists\textbf{-Exp}}$, $\Longrightarrow_{\mathcal{GS}}^{\mathbf{Ded}}$, $\Longrightarrow_{\mathcal{GS}}^{\mathbf{V\text{-}Eli}}$ は，それぞれ変換規則 **Decomposition**, **Composition**, $\forall$-**Expansion**, $\exists$-**Expansion**, **Deduction**, **Var-Elimination** による 1 ステップの変換を表す． □

変換 $\mathcal{E};\mathcal{R} \Longrightarrow_{\mathcal{GS}}^{*} \mathcal{E}';\mathcal{R}$ によって得られる論理式集合 $\mathcal{E}'$ は TRS $\mathcal{R}$ の下で変換前の論理式集合 $\mathcal{E}$（仕様）を満足する [1], [2].

［例 3.4］ 自然数を 1/2 倍するプログラム $\mathsf{h}$ を定義する TRS $\mathcal{R}_1$ と，自然数を 2 倍するプログラム $\mathsf{d}$ の仕様となる論理式集合 $\mathcal{E}_1$ が以下で与えられている．

$$\mathcal{R}_1 = \left\{\begin{array}{c}\mathsf{h}(\mathsf{0}) \rightarrow \mathsf{0},\\ \mathsf{h}(\mathsf{s}(\mathsf{0})) \rightarrow \mathsf{0},\\ \mathsf{h}(\mathsf{s}(\mathsf{s}(x))) \rightarrow \mathsf{s}(\mathsf{h}(x))\end{array}\right\}$$

$$\mathcal{E}_1 = \left\{\begin{array}{c}\forall x.\mathsf{d}(\mathsf{h}(x)) \approx x,\\ \forall x.\mathsf{h}(\mathsf{d}(x)) \approx x\end{array}\right\}$$

$\mathcal{E}_1;\mathcal{R}_1$ を $\mathcal{GS}$ によって変換する [2].

$$\mathcal{E}_1;\mathcal{R}_1 = \left\{\begin{array}{c}\forall x.\mathsf{d}(\mathsf{h}(x)) \approx x,\\ \forall x.\mathsf{h}(\mathsf{d}(x)) \approx x\end{array}\right\};\mathcal{R}_1$$

$$\Longrightarrow_{\mathcal{GS}}^{*} \left\{\begin{array}{c}\forall x.\exists y.\left(\mathsf{h}(x) \approx y \wedge \mathsf{d}(y) \approx x\right),\\ \mathsf{d}(\mathsf{0}) \approx \mathsf{0} \vee \mathsf{d}(\mathsf{0}) \approx \mathsf{s}(\mathsf{0}),\\ \forall u.\mathsf{d}(\mathsf{s}(u)) \approx \mathsf{s}(\mathsf{s}(\mathsf{d}(u))),\\ \forall u.\mathsf{h}(\mathsf{d}(u)) \approx u\end{array}\right\};\mathcal{R}_1$$

変換によって得られた論理式集合は $\mathsf{d}$ を実行可能な定義

$$\mathcal{R}_2 = \left\{\begin{array}{c}\mathsf{d}(\mathsf{0}) \rightarrow \mathsf{0},\\ \mathsf{d}(\mathsf{0}) \rightarrow \mathsf{s}(\mathsf{0}),\\ \mathsf{d}(\mathsf{s}(x)) \rightarrow \mathsf{s}(\mathsf{s}(\mathsf{d}(x)))\end{array}\right\}$$

を含んでいる． □

定義 3.3 の変換規則では論理式文脈 $\Gamma\langle\rangle$ の単調性 [2] を要求しているが，一般の論理式文脈に関する単調性判定はそれほど簡単ではない．しかし，本稿の目的である Deforestation の模倣に際しては，$\wedge$ と $\forall$ と $\approx$ のみを含む論理式文脈しか扱わない．よって，以降の議論においては単調性は常に成り立つと仮定してよい．

## 4. $\mathcal{GS}$ による Deforestation の模倣

関数記号の集合 $\mathcal{F}$ は被定義記号の集合 $\mathcal{F}_D$ と構成子記号の集合 $\mathcal{F}_C$ に分割されているとする．すなわち $\mathcal{F} = \mathcal{F}_D \uplus \mathcal{F}_C$ である．

次のように定義される線形項 $tt$ を treeless 項 [8] という．

$$tt ::= x \quad (x \in \mathcal{X})$$

$$
\begin{array}{lll}
(1)\ \mathcal{D}[\![\mathsf{app}(\mathsf{app}(xs,ys),zs)]\!] = & \text{let} & \\
 & \quad \mathsf{app}'([\,],ys,zs) & \approx \ \mathcal{D}[\![\mathsf{app}(\mathsf{app}([\,],ys),zs)]\!] \\
 & \quad \mathsf{app}'(x::xs,ys,zs) & \approx \ \mathcal{D}[\![\mathsf{app}(\mathsf{app}(x::xs,ys),zs)]\!] \\
 & \text{in} & \\
 & \quad \mathsf{app}'(xs,ys,zs) & \\
 & & \\
(2)\ \mathcal{D}[\![\mathsf{app}(\mathsf{app}([\,],ys),zs)]\!] & = \ \mathcal{D}[\![\mathsf{app}(ys,zs)]\!] & \\
 & = \ \mathsf{app}(ys,zs) & \\
 & & \\
(3)\ \mathcal{D}[\![\mathsf{app}(\mathsf{app}(x::xs,ys),zs)]\!] & = \ \mathcal{D}[\![\mathsf{app}(x::\mathsf{app}(xs,ys),zs)]\!] & \\
 & = \ \mathcal{D}[\![x::\mathsf{app}(\mathsf{app}(xs,ys),zs)]\!] & \\
 & = \ \mathcal{D}[\![x]\!]::\mathcal{D}[\![\mathsf{app}(\mathsf{app}(xs,ys),zs)]\!] & \\
 & = \ x::\mathsf{app}'(xs,ys,zs) &
\end{array}
$$

図 1　Deforestation $\mathcal{D}[\![\mathsf{app}(\mathsf{app}(xs,ys),zs)]\!]$

$$
\quad | \quad c(tt_1,\ldots,tt_n) \quad (c\in F_C,\ tt_1,\ldots,tt_n : treeless)
$$
$$
\quad | \quad h(x_1,\ldots,x_n) \quad (h\in\mathcal{F}_D,\ x_1,\ldots,x_n\in\mathcal{X})
$$

また，次のように定義される線形項 $p$ をパターン項という．

$$
p ::= x \quad (x\in\mathcal{X})
$$
$$
\quad | \quad c(x_1,\ldots,x_n) \quad (c\in\mathcal{F}_C,\ x_1,\ldots,x_n\in\mathcal{X})
$$

Deforestation で取り扱うプログラム [8] は，次のように定義される関数である．

［定義 4.1］（treeless 関数定義）　各被定義記号について，以下のどちらかの形式で与えられる書換え規則の集合 (TRS) を treeless 関数定義という．

（1）　$f$ 形式

$$
f(x_1,\ldots,x_n)\to tf
$$

（2）　$g$ 形式

$$
\begin{array}{l}
g(p_1,x_1,\ldots,x_n)\ \to\ tg_1 \\
\qquad\qquad\vdots \\
g(p_m,x_1,\ldots,x_n)\ \to\ tg_m
\end{array}
$$

ここで，$tf, tg_1,\ldots,tg_m$ は treeless 項，$p_1\equiv c_1(y_1,\ldots,y_{n_1})$, $\cdots$, $p_m\equiv c_m(y_1,\ldots,y_{n_m})$ はパターン項である．また，書換え規則の左辺は線形である．　□

［例 4.2］　2 つのリストを連結するプログラム app は以下の treeless 関数定義 $\mathcal{R}_3$ で表現できる．

$$
\mathcal{R}_3=\left\{\begin{array}{c}\mathsf{app}([\,],ys)\to ys,\\ \mathsf{app}(x::xs,ys)\to x::\mathsf{app}(xs,ys)\end{array}\right\}
$$

□

Deforestation の定義に先立ち，次のような特別な項文脈を定義しておく．

$$
\begin{array}{lll}
\cdots\Box\cdots & ::= & \Box \\
 & | & g(\cdots\Box\cdots,x_1,\ldots,x_n)
\end{array}
$$

以上の準備の下で，Deforestation は次のように定義される．

なお，以降では Deforestation の入力項となる合成関数は第 2 引数以降が全て変数であると仮定する．Deforestation が合成の対象とする $g$ 形式関数は第 1 引数においてのみパターンマッチングを許しているので，このこと自体は妥当な制約である．

［定義 4.3］（Deforestation）　文献 [8] 形式の定義を用いる．

$$
\begin{array}{l}
\mathcal{D}[\![x]\!]=x \\
\mathcal{D}[\![c(t_1,...,t_n)]\!]=c(\mathcal{D}[\![t_1]\!],...,\mathcal{D}[\![t_n]\!]) \\
\mathcal{D}[\![\cdots f(t_1,...,t_n)\cdots]\!]\quad (f: f\text{ 形式の関数定義}) \\
\ =\ \mathcal{D}[\![\cdots tf\{x_1\mapsto t_1,...,x_n\mapsto t_n\}\cdots]\!] \\
\mathcal{D}[\![\cdots g(c_i(s_1,...,s_{n_i}),t_1,...,t_n)\cdots]\!]\quad (g: g\text{ 形式の関数定義}) \\
\ =\ \mathcal{D}[\![\cdots tg_i\{y_1\mapsto s_1,...,y_{n_i}\mapsto s_{n_i},x_1\mapsto t_1,...,x_n\mapsto t_n\}\cdots]\!] \\
\mathcal{D}[\![\cdots g(x_0,x_1,...,x_n)\cdots]\!]\quad (g: g\text{ 形式の関数定義}) \\
\ =\quad \text{let} \\
\qquad\qquad g'(p_1,x_1',...,x_l') \ \to\ \mathcal{D}[\![\cdots g(p_1,x_1,...,x_n)\cdots]\!] \\
\qquad\qquad\qquad\vdots \\
\qquad\qquad g'(p_m,x_1',...,x_l') \ \to\ \mathcal{D}[\![\cdots g(p_m,x_1,...,x_n)\cdots]\!] \\
\qquad \text{in} \\
\qquad\qquad g'(x_0,x_1',...,x_l') \\
\qquad \{x_1',...,x_l'\}=\mathcal{V}ar(\cdots g(x_0,x_1,...,x_n)\cdots)\setminus\{x_0\}
\end{array}
$$

□

［例 4.4］　treeless 関数定義として例 4.2 の $\mathcal{R}_3$ を考える．3 つのリストを連結するプログラムは app を用いると，$\mathsf{app}(\mathsf{app}(xs,ys),zs)$ と書ける．この合成関数は Deforestation によって図 1 のように変換される．この変換結果を TRS とみなすと次のようになる．

$$
\mathcal{R}_4=\left\{\begin{array}{c}\mathsf{app}'([\,],ys,zs)\to\mathsf{app}(ys,zs),\\ \mathsf{app}'(x::xs,ys,zs)\to x::\mathsf{app}'(xs,ys,zs)\end{array}\right\}
$$

□

$\mathcal{GS}$ の変換においては，Deforestation のように新しい関数記号を導入することはない．しかし，Deforestation によって導入される関数は $\mathcal{GS}$ にとっては，生成すべきプログラムの仕様とみなすのが自然である．$\mathcal{D}[\![t]\!]$ の模倣に際しては，以下で定義

$$\mathcal{S}_{\mathsf{app}(\mathsf{app}(xs,ys),zs)};\mathcal{R}_3 = \left\{\ \forall xs.\forall ys.\forall zs.\, h^{\mathsf{app},\mathsf{app}}(xs,ys,zs) \approx \mathsf{app}(\mathsf{app}(xs,ys),zs)\ \right\};\mathcal{R}_3$$

（以下の 1 ステップの変換が図 1 の (1) $\mathcal{D}[\![\mathsf{app}(\mathsf{app}(xs,ys),zs)]\!]$ を模倣する.）

$$\Longrightarrow_{\mathcal{GS}}^{\forall\textbf{-Exp}} \left\{ \begin{array}{rc} & \forall ys.\forall zs.\, h^{\mathsf{app},\mathsf{app}}([\,],ys,zs) \approx \mathsf{app}(\mathsf{app}([\,],ys),zs) \\ \wedge & \forall x.\forall xs.\forall ys.\forall zs.\, h^{\mathsf{app},\mathsf{app}}(x::xs,ys,zs) \approx \mathsf{app}(\mathsf{app}(x::xs,ys),zs) \end{array} \right\};\mathcal{R}_3$$

（以下の 1 ステップの変換が図 1 の (2) $\mathcal{D}[\![\mathsf{app}(\mathsf{app}([\,],ys),zs)]\!]$ を模倣する.）

$$\Longrightarrow_{\mathcal{GS}}^{\textbf{Ded}} \left\{ \begin{array}{rc} & \forall ys.\forall zs.\, h^{\mathsf{app},\mathsf{app}}([\,],ys,zs) \approx \mathsf{app}(ys,zs) \\ \wedge & \forall x.\forall xs.\forall ys.\forall zs.\, h^{\mathsf{app},\mathsf{app}}(x::xs,ys,zs) \approx \mathsf{app}(\mathsf{app}(x::xs,ys),zs) \end{array} \right\};\mathcal{R}_3$$

（以下の 3 ステップの変換が図 1 の (3) $\mathcal{D}[\![\mathsf{app}(\mathsf{app}(x::xs,ys),zs)]\!]$ を模倣する.）

$$\Longrightarrow_{\mathcal{GS}}^{\textbf{Ded}} \left\{ \begin{array}{rc} & \forall ys.\forall zs.\, h^{\mathsf{app},\mathsf{app}}([\,],ys,zs) \approx \mathsf{app}(ys,zs) \\ \wedge & \forall x.\forall xs.\forall ys.\forall zs.\, h^{\mathsf{app},\mathsf{app}}(x::xs,ys,zs) \approx \mathsf{app}(x::\mathsf{app}(xs,ys),zs) \end{array} \right\};\mathcal{R}_3$$

$$\Longrightarrow_{\mathcal{GS}}^{\textbf{Ded}} \left\{ \begin{array}{rc} & \forall ys.\forall zs.\, h^{\mathsf{app},\mathsf{app}}([\,],ys,zs) \approx \mathsf{app}(ys,zs) \\ \wedge & \forall x.\forall xs.\forall ys.\forall zs.\, h^{\mathsf{app},\mathsf{app}}(x::xs,ys,zs) \approx x::\mathsf{app}(\mathsf{app}(xs,ys),zs) \end{array} \right\};\mathcal{R}_3$$

$$\Longrightarrow_{\mathcal{GS}}^{\textbf{Ded}} \left\{ \begin{array}{rc} & \forall ys.\forall zs.\, h^{\mathsf{app},\mathsf{app}}([\,],ys,zs) \approx \mathsf{app}(ys,zs) \\ \wedge & \forall x.\forall xs.\forall ys.\forall zs.\, h^{\mathsf{app},\mathsf{app}}(x::xs,ys,zs) \approx x::h^{\mathsf{app},\mathsf{app}}(xs,ys,zs) \end{array} \right\};\mathcal{R}_3$$

図 2　Deforestation $\mathcal{D}[\![\mathsf{app}(\mathsf{app}(xs,ys),zs)]\!]$ を模倣する $\mathcal{GS}$ 変換系列

される論理式集合 $\mathcal{S}_t$ を $\mathcal{GS}$ に与える仕様とする.

［定義 4.5］　$t$ を項とする．$t$ 中に存在する $g$ 形式関数のネスト数の最大値を $N$ とする．このとき，論理式集合 $\mathcal{S}_t$ を以下のように定める.

$$\mathcal{S}_t = \{\overrightarrow{\forall x}.\, h^{g_1,\cdots,g_k}(\overrightarrow{x}) \approx th^{g_1,\cdots,g_k} \mid 2 \leqq k \leqq N\}$$

ここで，新しい関数 $h^{g_1\cdots g_k}$ は $g$ 形式関数 $g_1,\cdots,g_k$ に対して，次のように再帰的に定義される.

（1）　$h^{g_1}(x_0,\overrightarrow{x_1}) \to g_1(x_0,\overrightarrow{x_1})$

（2）　$h^{g_1,\cdots,g_{k-1}}(x_0,\overrightarrow{x_{k-1}},\cdots,\overrightarrow{x_1}) \to th$ のとき,

$$h^{g_1,\cdots,g_{k-1}g_k}(x_0,\overrightarrow{x_k},\overrightarrow{x_{k-1}},\cdots,\overrightarrow{x_1}) \to th^{g_1,\cdots,g_{k-1}}\{x_0 \mapsto g_k(x_0,\overrightarrow{x_k})\}$$

□

［例 4.6］　$\mathcal{D}[\![\mathsf{app}(\mathsf{app}(xs,ys),zs)]\!]$ を模倣する場合，$\mathcal{GS}$ に与えるべき仕様は,

$$\mathcal{S}_{\mathsf{app}(\mathsf{app}(xs,ys),zs)} = \left\{\ \forall xs.\forall ys.\forall zs.\, h^{\mathsf{app},\mathsf{app}}(xs,ys,zs) \approx \mathsf{app}(\mathsf{app}(xs,ys),zs)\ \right\}$$

である.　□

$\mathcal{GS}$ によって Deforestation を模倣できることは，Deforestation の 1 回分の再帰呼出しに対して，常に対応する $\mathcal{GS}$ 変換が存在すること（補題 4.7）を保証することで示される.

［補題 4.7］

（1）　$\mathcal{D}[\![t]\!] = t'$ かつ $t'$ が let を含まないとする．このとき，あらゆる項 $s$，論理式 $W$ に対して，$\{\overrightarrow{\forall x}.\, s \approx t \wedge W\};\mathcal{R} \Longrightarrow_{\mathcal{GS}}^{\textbf{Ded}^*} \{\overrightarrow{\forall x}.\, s \approx t' \wedge W\};\mathcal{R}$ である．ここで，$\{\overrightarrow{x}\} = \mathcal{V}ar(s \approx t)$ である.

（2）　$\mathcal{D}[\![t]\!] =$ let

$$\begin{array}{rcl} g'(p_1,x'_1,...,x'_l) & \to & \mathcal{D}[\![u_1]\!] \\ & \vdots & \\ g'(p_m,x'_1,...,x'_l) & \to & \mathcal{D}[\![u_m]\!] \end{array}$$

in

$$g'(x_0,x'_1,...,x'_l)$$

であるとする．このとき,

$$\mathcal{S}_t;\mathcal{R} \Longrightarrow_{\mathcal{GS}}^{\forall\textbf{-Exp}} \mathcal{S}_t \cup \{\wedge_i\, \overrightarrow{\forall z_i}.\forall x'_1.\,...\forall x'_l.\, g'(p_i,x'_1,...,x'_l) \approx u_i\};\mathcal{R}$$

を満たす変換が存在する．ここで，$\{\overrightarrow{z_i}\} = \mathcal{V}ar(p_i)$ である.

［証明 ］

（1）　$\mathcal{D}[\![\cdot]\!]$ の再帰呼出しの回数 $M$ に関する帰納法による.

($M=1$ の場合)　$t \equiv x$ または $t \equiv c$ のいずれかの場合に相当する．どちらの場合も $t' \equiv t$ であるから，明らかに成り立つ.

($M>1$ の場合)　$s$ を任意の項，$W$ を任意の論理式とする.

$t \equiv c(t_1,...,t_n)$ であるとき，$t' = \mathcal{D}[\![t]\!] = c(\mathcal{D}[\![t_1]\!],...,\mathcal{D}[\![t_n]\!])$. $\mathcal{D}[\![t_1]\!] = t'_1,\ldots,\mathcal{D}[\![t_n]\!] = t'_n$ とすると，任意の $i$ に対して帰納法の仮定から $\{\overrightarrow{\forall x_i}.\, s \approx t_i \wedge W\} \Longrightarrow_{\mathcal{GS}}^{\textbf{Ded}^*} \{\overrightarrow{\forall x_i}.\, s \approx t'_i \wedge W\}$ である．ここで $\{\overrightarrow{x_i}\} = \mathcal{V}ar(s \approx t_i)$．また $\{\overrightarrow{x}\} = \{\overrightarrow{x_1}\} \cup \cdots \cup \{\overrightarrow{x_n}\}$ であるので $\{\overrightarrow{\forall x}.\, s \approx t \wedge W\} = \{\overrightarrow{\forall x}.\, s \approx c(t_1,t_2,...,t_n) \wedge W\} \Longrightarrow_{\mathcal{GS}}^{\textbf{Ded}^*} \{\overrightarrow{\forall x}.\, s \approx c(t'_1,t_2,...,t_n) \wedge W\} \Longrightarrow_{\mathcal{GS}}^{\textbf{Ded}^*} \{\overrightarrow{\forall x}.\, s \approx c(t'_1,t'_2,...,t_n) \wedge W\} \Longrightarrow_{\mathcal{GS}}^{\textbf{Ded}^*} \cdots \Longrightarrow_{\mathcal{GS}}^{\textbf{Ded}^*} \{\overrightarrow{\forall x}.\, s \approx c(t'_1,t'_2,...,t'_n) \wedge W\} = \{\overrightarrow{\forall x}.\, s \approx t' \wedge W\}$ である.

$t \equiv \cdots g(c_i(s_1,...,s_{n_i}),t_1,...,t_n)\cdots$ であるとき，$t' = \mathcal{D}[\![t'']\!]$ かつ $t'' \equiv \cdots tg_i\{y_1 \mapsto s_1,...,y_{n_i} \mapsto s_{n_i},x_1 \mapsto t_1,...,x_n \mapsto t_n\}\cdots$ である．$g(c_i(y_1,...,y_{n_i}),x_1,...,x_n) \to tg_i \in \mathcal{R}$ であるから $\{\overrightarrow{\forall x}.\, s \approx t \wedge W\} \Longrightarrow_{\mathcal{GS}}^{\textbf{Ded}} \{\overrightarrow{\forall x}.\, s \approx t'' \wedge W\}$．また，帰納法の仮定より $\{\overrightarrow{\forall x}.\, s \approx t'' \wedge W\} \Longrightarrow_{\mathcal{GS}}^{\textbf{Ded}^*} \{\overrightarrow{\forall x}.\, s \approx t' \wedge W\}$ したがって $\{\overrightarrow{\forall x}.\, s \approx t \wedge W\} \Longrightarrow_{\mathcal{GS}}^{\textbf{Ded}^*} \{\overrightarrow{\forall x}.\, s \approx t' \wedge W\}$.

$t \equiv \cdots f(t_1,...,t_n)\cdots$ であるときも同様.

（2）　$t \equiv \cdots g(x_0,x_1,...,x_n)\cdots$ の形をしているはずである．また，各 $i$ に対して $u_i \equiv \cdots g(p_i,x_1,...,x_n)\cdots$ である．このとき $\mathcal{S}_t$ のつくりかたから，$\forall x_0.\forall x'_1.\,...\forall x'_l.\, g'(x_0,x'_1,...,x'_l) \approx \cdots g(x_0,x_1,...,x_n)\cdots \in \mathcal{S}_t$ である．さらに $\{p_1,\ldots,p_m\} \in \mathcal{CS}(\mathcal{R})$ なので,

$\mathcal{S}_t;\mathcal{R} \Longrightarrow_{\mathcal{GS}}^{\forall\textbf{-Exp}} \mathcal{S}_t \cup$
$\{\wedge_i\, \overrightarrow{\forall z_i}.\forall x'_1.\,...\forall x'_l.\, g'(p_i,x'_1,...,x'_l) \approx \cdots g(p_i,x_1,...,x_n)\cdots\};\mathcal{R}$
$= \mathcal{S}_t \cup \{\wedge_i\, \overrightarrow{\forall z_i}.\forall x'_1.\,...\forall x'_l.\, g'(p_i,x'_1,...,x'_l) \approx u_i\};\mathcal{R}$

□

補題 4.7 からプログラム生成変換 $\mathcal{GS}$ は Deforestation を模倣できるといえる．

［定理 4.8］ $\mathcal{GS}$ は Deforestation を模倣する． □

最後に，Deforestation による変換を実際に $\mathcal{GS}$ で模倣する一例を挙げる．

［例 4.9］ 図 1 の変換は，例 4.6 の仕様 $\mathcal{S}_{\mathsf{app}(\mathsf{app}(xs,ys),zs)}$ からはじまる $\mathcal{GS}$ の変換系列によって模倣できる（図 2）． □

## 5. ま と め

本稿では，我々が提案したプログラム生成変換 $\mathcal{GS}$ が Deforestation を模倣できることを示した．このことにより，少なくとも Deforestation が対象とするプログラムと（関数合成を表現する）仕様に対しては，$\mathcal{GS}$ によるプログラム生成が必ず成功することが明らかになった．

プログラム融合変換を全自動で行うためには，補題 4.7 で示した模倣手法をアルゴリズムとしてより明示的に示す必要がある．

Deforestation が対象とするプログラムと仕様だけではなく，プログラム生成システムとしての $\mathcal{GS}$ の完全性を保証する他の条件を発見することは，今後の課題としてより興味深い．

**謝辞**

本研究は一部，文部科学省科学研究費補助金（萌芽研究 16650005），ならびに，文部科学省 21 世紀 COE プログラム「社会情報基盤のための音声・映像の知的統合」の助成を受けている．

## 文 献

[1] 長島, 酒井, 坂部, 草刈：“限量子付き等式理論の変換に基づく仕様からのプログラム生成”, コンピュータソフトウェア, **21**, 4, pp. 49–54 (2004).

[2] 長島正憲：“限量子付き等式仕様の変換に基づくプログラム生成に関する研究”, 修士学位論文, 名古屋大学 (2004).

[3] P. L. Wadler: “Deforestation: Transforming programs to eliminate trees”, Theoretical Computer Science, **73**, 2, pp. 231–248 (1990).

[4] F. Baader and T. Nipkow: “Term Rewriting and All That”, Cambridge University Press (1998).

[5] M. Ben-Ari: “Mathematical Logic for Computer Science”, 2nd ed., Springer-Verlag, London (2001).

[6] E. Mendelson: “Introduction to Mathematical Logic”, 4th ed., Chapman & Hall, London (1997).

[7] D. A. Plaisted: “Semantic confluence tests and completion methods”, Information and Control, **65**, 2/3, pp. 182–215 (1985).

[8] A. B. Ferguson and P. L. Wadler: “When will deforestation stop?”, Proceedings of 1988 Glasgow Workshop on Functional Programming, Rothesay, Isle of Bute, pp. 39–56 (1988).