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.

# $\mathcal{G}ene\mathcal{S}ys$ によるプログラム生成例と Introduction 規則の追加

近藤　悟†　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: †skondo@sakabe.i.is.nagoya-u.ac.jp, ††{sakai,nishida,sakabe,kusakari}@is.nagoya-u.ac.jp

**あらまし**　プログラム生成系 $\mathcal{G}ene\mathcal{S}ys$ は一階述語論理で記述された仕様からの実行可能なプログラム生成を目的とした手法である．本論文では，よく知られたいくつかのプログラム合成例に $\mathcal{G}ene\mathcal{S}ys$ が適用できることを示す．また，$\mathcal{G}ene\mathcal{S}ys$ では実行可能なプログラムを生成できない仕様の一例を挙げ，等式を追加する新たな変換規則 **Introduction** によりこの仕様からのプログラム生成が可能になることを示す．
**キーワード**　プログラム生成，項書換え系，等式仕様

# Example programs generated by $\mathcal{G}ene\mathcal{S}ys$ and proposal of Introduction rule

Satoru KONDO†, 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: †skondo@sakabe.i.is.nagoya-u.ac.jp, ††{sakai,nishida,sakabe,kusakari}@is.nagoya-u.ac.jp

**Abstract**　Program Generation System $\mathcal{G}ene\mathcal{S}ys$ is a method for generating executable programs from specifications described in first-order logic. In this paper, we show some examples known as successful results obtained by program composition are applicable to GeneSys. We also propose a new conversion rule called Introduction, which enables $\mathcal{G}ene\mathcal{S}ys$ to generate programs from a specification that the original system fails.
**Key words**　program generation, term rewriting system, equational specification

## 1. はじめに

作成したいプログラムの満たすべき性質を記述した仕様からプログラムを作成する作業は自動化が困難な知的作業である．一方で，プログラムを対象とする自動変換はいくつか存在している．複数の関数の組み合わせからなるプログラムを単一の関数に融合することにより，合成関数を計算する効率的なプログラムを生成する融合変換や，計算を効率化する一定のパターンにプログラムを埋め込むことで効率の良いプログラムを得る変換手法 [3] などがある．しかし，これらの変換方法はプログラムとして実行できない仕様を対象とすることができない．仕様からプログラムへの変換を機械的に実行する手法としてプログラム生成系 $\mathcal{G}ene\mathcal{S}ys$ [1] が提案されている．$\mathcal{G}ene\mathcal{S}ys$ は実行可能な既知プログラムを参照しながら論理式で表された仕様を変換する推論規則で構成される．これまでに，プログラム融合変換の一手法である Deforestation [4] を $\mathcal{G}ene\mathcal{S}ys$ の自動実行で模倣する戦略が提案されている [2]．また，プログラムとして実行不可能な仕様からのプログラム生成例として，自然数を 1/2 倍するプログラム H と自然数を 2 倍する関数 D の仕様を表わす論理式集合から関数 D を計算するプログラムを生成できることが示されている．しかし，Deforestation 以外の入力クラスに対する $\mathcal{G}ene\mathcal{S}ys$ の能力についてはよく分かっていない．

本論文では，まず，プログラム効率化としてよく知られているプログラム合成例に $\mathcal{G}ene\mathcal{S}ys$ が適用できることを示す．次に，新しい関数記号を含む論理式を追加する変換規則 **Introduction** を提案する．この規則により，論理式の変換過程で仕様または既知プログラムに定義されていない新しい関数記号を取り扱うことができる．未定義の関数記号が必要になるために従来の $\mathcal{G}ene\mathcal{S}ys$ ではプログラムを生成することができない仕様の一例を挙げ，変換規則 **Introduction** によりこの仕様からのプログラム生成が可能になることを示す．最後に，研究を効率的に進めるために開発した，ユーザとの対話を通じて $\mathcal{G}ene\mathcal{S}ys$ に

よる論理式変換を支援する実験的ツールを紹介する．

## 2. 準　　備

本節では，一階述語論理と項書換え系に関する記法や概念を定義する [5], [6], [7].

**論理記号**は $\neg, \wedge, \vee, \Rightarrow, \forall, \exists$ の 6 個である．**変数**の（可算無限）集合を $\mathcal{X}$，**関数記号**の集合を $\mathcal{F}$，**述語記号**の集合を $\mathcal{P}$ 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．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，$\neg U$, $U \wedge V$, $U \vee V$, $U \Rightarrow V$, $\forall x.U$, $\exists x.U$ はすべて論理式である．

$\xi x.$ 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，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．

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．特に，$\sigma(x) \in \mathcal{T}(\mathcal{F})$ となるとき $\sigma$ を**基底代入**という．ただし，限量子で束縛される変数は名前替えによって $\sigma$ の定義域や値域に含まれない変数になっているものとする．

特別な定数記号 $\Box$ をちょうど 1 個含む項 $C[\ ]$ 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 $\Gamma\langle\ \rangle$ を**論理式文脈**と呼び，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．

論理式 $U, V$ が全ての解釈において等しい真理値を持つとき，$U$ と $V$ は**意味論的同値**であるといい，$U \cong V$ と記述する．論理式集合 $\mathcal{E}$ の全ての論理式を真にするあらゆる解釈が $U$ も真にするとき，$U$ は $\mathcal{E}$ の**意味論的帰結**であるといい，$\mathcal{E} \models U$ と書く．

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. プログラム生成系 $\mathcal{G}ene\mathcal{S}ys$

本節では，プログラム生成系 $\mathcal{G}ene\mathcal{S}ys$ [1] の概要を説明する．

与えられた論理式集合 $\mathcal{E}$ に対して，論理式集合上の二項関係 $\to_{\mathcal{E}}$ を次のように定義する．

［定義 3.1］ $U, V$ 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 $\overrightarrow{\forall x}.\overrightarrow{\xi y}.(l \approx r \wedge W) \in \mathcal{E}$ が存在して

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

であるとき，$U$ は $V$ に**書き換えられる**といい，$U \to_{\mathcal{E}} V$ 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

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

を満たしているものとする． □

プログラム生成系 $\mathcal{G}ene\mathcal{S}ys$ の枠組みにおいては，論理式の変換をプログラム生成とみなす．また，プログラムは入力としてデータ (基底項) を受け取る．この観点からすると，$\mathcal{G}ene\mathcal{S}ys$ の変換における変数割り当てと限量子の解釈には基底項のみを考慮すれば十分である．$\mathcal{G}ene\mathcal{S}ys$ 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 [8] の概念を用いている．

［定義 3.2］（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．すべての基底項 $t \in \mathcal{T}(\mathcal{F})$ 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． □

プログラム生成系 $\mathcal{G}ene\mathcal{S}ys$ は次のように定義される．

［定義 3.3］ (プログラム生成系 $\mathcal{G}ene\mathcal{S}ys$) プログラム生成系 $\mathcal{G}ene\mathcal{S}ys$ は論理式集合と TRS で構成される対に適用される，以下の 6 つの変換規則で構成される．ただし，各変換規則は論理式文脈 $\Gamma\langle\ \rangle$ が単調 [1] である場合にのみ適用できるものとする．

（1） **Decomposition**

$$\frac{\mathcal{E} \cup \{\Gamma\langle C[t] \approx s\rangle\} \ ; \ \mathcal{R}}{\mathcal{E} \cup \{\Gamma\langle \exists x.\, (t \approx x \wedge C[x] \approx s)\rangle\} \ ; \ \mathcal{R}}$$

*if* $x \notin \mathcal{V}ar(\Gamma\langle C[t] \approx s\rangle)$

（2） **Composition**

$$\frac{\mathcal{E} \cup \{\Gamma\langle \exists x.\, (x \approx t \wedge U)\rangle\} \ ; \ \mathcal{R}}{\mathcal{E} \cup \{\Gamma\langle U\sigma\rangle\} \ ; \ \mathcal{R}}$$

*if* $x \notin \mathcal{V}ar(t),\ \sigma = \{x \mapsto t\}$

（3） $\forall$-**Expansion**

$$\frac{\mathcal{E} \cup \{\Gamma\langle \forall x.\, U\rangle\} \ ; \ \mathcal{R}}{\mathcal{E} \cup \{\Gamma\langle \bigwedge_i \overrightarrow{\forall y_i}.\, U\sigma_i\rangle\} \ ; \ \mathcal{R}}$$

*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**

$$\frac{\mathcal{E} \cup \{\Gamma\langle \exists x.\, U\rangle\} \ ; \ \mathcal{R}}{\mathcal{E} \cup \{\Gamma\langle \bigvee_i \overrightarrow{\exists y_i}.\, U\sigma_i\rangle\} \ ; \ \mathcal{R}}$$

*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**

$$\frac{\mathcal{E} \cup \{U\} \ ; \ \mathcal{R}}{\mathcal{E} \cup \{V\} \ ; \ \mathcal{R}}$$

*if* $U \rightarrow_{\mathcal{E} \cup \mathcal{E}_{\mathcal{R}}} V$

（6） **Var-Elimination**

$$\frac{\mathcal{E} \cup \{\Gamma\langle \forall x.\, (\bigvee_i \overrightarrow{\exists y_i}.\, (x \approx t_i \wedge U_i))\rangle\} \ ; \ \mathcal{R}}{\mathcal{E} \cup \{\Gamma\langle \bigwedge_i \overrightarrow{\forall y_i}.\, U_i\sigma_i\rangle\} \ ; \ \mathcal{R}}$$

*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 $\mathcal{G}ene\mathcal{S}ys$ の変換規則を一回適用して $\mathcal{E}';\mathcal{R}$ になったとき $\mathcal{E};\mathcal{R} \Rightarrow_{\mathcal{GS}} \mathcal{E}';\mathcal{R}$ と書く．$\Rightarrow_{\mathcal{GS}}^{\mathbf{Dec}}$, $\Rightarrow_{\mathcal{GS}}^{\mathbf{Com}}$, $\Rightarrow_{\mathcal{GS}}^{\forall\text{-}\mathbf{Exp}}$, $\Rightarrow_{\mathcal{GS}}^{\exists\text{-}\mathbf{Exp}}$, $\Rightarrow_{\mathcal{GS}}^{\mathbf{Ded}}$, $\Rightarrow_{\mathcal{GS}}^{\mathbf{V\text{-}Eli}}$ は，それぞれ変換規則 **Decomposition**, **Composition**, $\forall$-**Expansion**, $\exists$-**Expansion**, **Deduction**, **Var-Elimination** による変換を表す． □

$\mathcal{G}ene\mathcal{S}ys$ はプログラム生成系として健全である．すなわち，変換 $\mathcal{E};\mathcal{R} \Rightarrow_{\mathcal{GS}}^{*} \mathcal{E}';\mathcal{R}$ 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）を満足することが証明されている [1].

定義 3.3 の変換規則では論理式文脈 $\Gamma\langle\ \rangle$ の単調性を要求しているが，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，否定記号を含まない論理式文脈は常に単調性を持つことが証明できる [1]．本論文で取り扱う論理式は論理記号 $\neg$, $\Rightarrow$ を扱わない．このため，以降の議論においては論理式文脈の単調性は常に成り立つと仮定してよい．

## 4. プログラム生成例

本節では，$\mathcal{G}ene\mathcal{S}ys$ により仕様からプログラムを生成できるいくつかの新しい変換例を示す．以下で示すこれらの変換例は，手作業によるプログラム合成でよく用いられる例題である．

［例 4.1］ 2つのリストを連結する関数 App と，App を用いたリストを反転させる関数 Rev を定義する TRS $\mathcal{R}_1$ と，Rev より効率の良いリスト反転関数 FRev 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．

$$\mathcal{R}_1 = \left\{ \begin{array}{c} \mathsf{App}([\,], ys) \rightarrow ys, \\ \mathsf{App}(x :: xs, ys) \rightarrow x :: \mathsf{App}(xs, ys), \\ \mathsf{App}(\mathsf{App}(xs, ys), zs) \rightarrow \mathsf{App}(xs, \mathsf{App}(ys, zs)), \\ \mathsf{Rev}([\,]) \rightarrow [\,], \\ \mathsf{Rev}(x :: xs) \rightarrow \mathsf{App}(\mathsf{Rev}(xs), x :: [\,]) \end{array} \right\}$$

$$\mathcal{E}_1 = \left\{ \ \forall xs.\, \forall ys.\, \mathsf{FRev}(xs, ys) \approx \mathsf{App}(\mathsf{Rev}(ys), xs) \ \right\}$$

$\mathcal{E}_1;\mathcal{R}_1$ を $\mathcal{G}ene\mathcal{S}ys$ によって変換する (図 1)．ただし，各変換ステップに割り当てられた番号は何回目の変換かを表わしている ($\overset{\mathbf{i}}{\Rightarrow}_{\mathcal{GS}}$ は $i$ 回目の変換)．この番号は後で変換を参照するときに利用する．

変換によって得られた論理式集合から FRev を実行可能な TRS $\mathcal{R}_2$ が得られる．

$$\mathcal{R}_2 = \left\{ \begin{array}{c} \mathsf{FRev}(xs, [\,]) \rightarrow xs, \\ \mathsf{FRev}(xs, y :: ys) \rightarrow \mathsf{FRev}(y :: xs, ys) \end{array} \right\}$$

□

［例 4.2］ Fibonacci 数を計算する Fib，自然数の加算を求める関数 Add，値の組 Pair から値を取り出す関数 Fir，Sec を定義する TRS $\mathcal{R}_3$ と，より効率良く Fibonacci 数を求める関数 FFib の仕様となる論理式集合 $\mathcal{E}_3$ が以下で与えられている．

$$\mathcal{R}_3 = \left\{ \begin{array}{c} \mathsf{Fib}(0) \rightarrow 0, \\ \mathsf{Fib}(\mathsf{s}(0)) \rightarrow \mathsf{s}(0), \\ \mathsf{Fib}(\mathsf{s}(\mathsf{s}(x))) \rightarrow \mathsf{Add}(\mathsf{Fib}(x), \mathsf{Fib}(\mathsf{s}(x))), \\ \mathsf{Add}(0, y) \rightarrow y \\ \mathsf{Add}(\mathsf{s}(x), y) \rightarrow \mathsf{s}(\mathsf{Add}(x, y)), \\ \mathsf{Fir}(\mathsf{Pair}(x, y)) \rightarrow x, \\ \mathsf{Sec}(\mathsf{Pair}(x, y)) \rightarrow y \end{array} \right\}$$

$$\mathcal{E}_3 = \left\{ \begin{array}{c} \forall x.\, \mathsf{Fir}(\mathsf{FFib}(x)) \approx \mathsf{Fib}(x), \\ \forall x.\, \mathsf{Sec}(\mathsf{FFib}(x)) \approx \mathsf{Fib}(S(x)) \end{array} \right\}$$

$\mathcal{E}_3;\mathcal{R}_3$ を $\mathcal{G}ene\mathcal{S}ys$ を用いて変換し，FFib を計算するプログラムを生成する．(図 2). □

TRS の規則の左辺の先頭に表れない関数記号を構成子記号と呼ぶ．等式の両辺が先頭に共通の構成子記号を持つとき，その構成子記号の引数に表れる項同士についても等式関係が成立する．これは，根の位置に構成子記号を持つ項は先頭で書き換えられないことから容易に証明できる．図 2 の 6 ステップ目の変換では，この性質により構成子記号 Pair を消去している．この例のように，論理式の等価変換を行った上で変換規則を適用すると変換を進めるために有効な場合がある．変換ステップ $\cong$ は論理式の等価変換規則による変換を表わす．

$$\mathcal{E}_1;\mathcal{R}_1 = \left\{\ \forall xs.\,\forall ys.\,\mathsf{FRev}(xs,ys) \approx \mathsf{App}(\mathsf{Rev}(ys),xs)\ \right\};\mathcal{R}_1$$

$$\overset{\mathbf{1}}{\Rightarrow}_{\mathcal{GS}}^{\forall\text{-}\mathbf{Exp}} \left\{ \begin{array}{c} \forall xs.\,\forall ys.\,\mathsf{FRev}(xs,ys) \approx \mathsf{App}(\mathsf{Rev}(ys),xs), \\ \left( \begin{array}{c} \forall xs.\,\mathsf{FRev}(xs,[\,]) \approx \mathsf{App}(\mathsf{Rev}([\,]),xs) \\ \wedge\quad \forall xs.\,\forall y.\,\forall ys.\,\mathsf{FRev}(xs,y::ys) \approx \mathsf{App}(\mathsf{Rev}(y::ys),xs) \end{array} \right) \end{array} \right\};\mathcal{R}_1$$

$$\overset{\mathbf{2}}{\Rightarrow}_{\mathcal{GS}}^{\mathbf{Ded}} \left\{ \begin{array}{c} \forall xs.\,\forall ys.\,\mathsf{FRev}(xs,ys) \approx \mathsf{App}(\mathsf{Rev}(ys),xs), \\ \forall xs.\,\mathsf{FRev}(xs,[\,]) \approx xs, \\ \forall xs.\,\forall y.\,\forall ys.\,\mathsf{FRev}(xs,y::ys) \approx \mathsf{App}(\mathsf{Rev}(ys),y::xs) \end{array} \right\};\mathcal{R}_1$$

$$\overset{\mathbf{3}}{\Rightarrow}_{\mathcal{GS}}^{\mathbf{Ded}} \left\{ \begin{array}{c} \forall xs.\,\forall ys.\,\mathsf{FRev}(xs,ys) \approx \mathsf{App}(\mathsf{Rev}(ys),xs), \\ \forall xs.\,\mathsf{FRev}(xs,[\,]) \approx xs, \\ \left( \begin{array}{c} \forall xs.\,\forall y.\,\forall ys.\,\mathsf{FRev}(xs,y::ys) \approx \mathsf{FRev}(y::xs,ys) \\ \wedge\quad \forall xs.\,\forall y.\,\forall ys.\,\mathsf{FRev}(y::xs,ys) \approx \mathsf{App}(\mathsf{Rev}(ys),y::xs) \end{array} \right) \end{array} \right\};\mathcal{R}_1$$

図 1　FRev のプログラム生成変換

$$\mathcal{E}_3;\mathcal{R}_3 = \left\{ \begin{array}{c} \forall x.\,\mathsf{Fir}(\mathsf{FFib}(x)) \approx \mathsf{Fib}(x), \\ \forall x.\,\mathsf{Sec}(\mathsf{FFib}(x)) \approx \mathsf{Fib}(S(x)) \end{array} \right\};\mathcal{R}_3 \quad \overset{\mathbf{1}}{\Rightarrow}_{\mathcal{GS}}^{\mathbf{Dec}} \left\{ \begin{array}{c} \forall x.\,\exists y.\,\mathsf{FFib}(x) \approx y \wedge \mathsf{Fir}(y) \approx \mathsf{Fib}(x), \\ \forall x.\,\exists y.\,\mathsf{FFib}(x) \approx y \wedge \mathsf{Sec}(y) \approx \mathsf{Fib}(\mathsf{s}(x)) \end{array} \right\};\mathcal{R}_3$$

$$\overset{\mathbf{2}}{\Rightarrow}_{\mathcal{GS}}^{\exists\text{-}\mathbf{Exp}} \left\{ \begin{array}{c} \forall x.\,\exists y.\,\exists z.\,\mathsf{FFib}(x) \approx \mathsf{Pair}(y,z) \wedge \mathsf{Fir}(\mathsf{Pair}(y,z)) \approx \mathsf{Fib}(x), \\ \forall x.\,\exists y.\,\exists z.\,\mathsf{FFib}(x) \approx \mathsf{Pair}(y,z) \wedge \mathsf{Sec}(\mathsf{Pair}(y,z)) \approx \mathsf{Fib}(\mathsf{s}(x)) \end{array} \right\};\mathcal{R}_3$$

$$\overset{\mathbf{3}}{\Rightarrow}_{\mathcal{GS}}^{\mathbf{Ded}} \left\{ \begin{array}{c} \forall x.\,\exists y.\,\exists z.\,\mathsf{FFib}(x) \approx \mathsf{Pair}(y,z) \wedge y \approx \mathsf{Fib}(x), \\ \forall x.\,\exists y.\,\exists z.\,\mathsf{FFib}(x) \approx \mathsf{Pair}(y,z) \wedge z \approx \mathsf{Fib}(\mathsf{s}(x)) \end{array} \right\};\mathcal{R}_3 \overset{\mathbf{4}}{\Rightarrow}_{\mathcal{GS}}^{\mathbf{Com}} \left\{ \begin{array}{c} \forall x.\,\exists z.\,\mathsf{FFib}(x) \approx \mathsf{Pair}(\mathsf{Fib}(x),z), \\ \forall x.\,\exists y.\,\mathsf{FFib}(x) \approx \mathsf{Pair}(y,\mathsf{Fib}(\mathsf{s}(x))) \end{array} \right\};\mathcal{R}_3$$

$$\overset{\mathbf{5}}{\Rightarrow}_{\mathcal{GS}}^{\mathbf{Ded}} \left\{ \begin{array}{c} \forall x.\,\exists z.\,\exists u. \\ \left( \begin{array}{c} \mathsf{Pair}(u,\mathsf{Fib}(\mathsf{s}(x))) \approx \mathsf{Pair}(\mathsf{Fib}(x),z) \\ \wedge\quad \mathsf{FFib}(x) \approx \mathsf{Pair}(u,\mathsf{Fib}(\mathsf{s}(x))) \end{array} \right), \\ \forall x.\,\exists y.\,\mathsf{FFib}(x) \approx \mathsf{Pair}(y,\mathsf{Fib}(\mathsf{s}(x))) \end{array} \right\};\mathcal{R}_3 \overset{\mathbf{6}}{\cong} \left\{ \begin{array}{c} \forall x.\,\exists z.\,\exists u. \\ \left( \begin{array}{c} u \approx \mathsf{Fib}(x) \\ \wedge\quad \mathsf{Fib}(\mathsf{s}(x))) \approx z \\ \wedge\quad \mathsf{FFib}(x) \approx \mathsf{Pair}(u,\mathsf{Fib}(\mathsf{s}(x))) \end{array} \right), \\ \forall x.\,\exists y.\,\mathsf{FFib}(x) \approx \mathsf{Pair}(y,\mathsf{Fib}(\mathsf{s}(x))) \end{array} \right\};\mathcal{R}_3$$

$$\overset{\mathbf{7}}{\Rightarrow}_{\mathcal{GS}}^{\mathbf{Com}} \left\{ \begin{array}{c} \forall x.\,\mathsf{FFib}(x) \approx \mathsf{Pair}(\mathsf{Fib}(x),\mathsf{Fib}(\mathsf{s}(x))), \\ \forall x.\,\exists y.\,\mathsf{FFib}(x) \approx \mathsf{Pair}(y,\mathsf{Fib}(\mathsf{s}(x))) \end{array} \right\};\mathcal{R}_3 \overset{\mathbf{8}}{\Rightarrow}_{\mathcal{GS}}^{\forall\text{-}\mathbf{Exp}} \left\{ \begin{array}{c} \left( \begin{array}{c} \mathsf{FFib}(0) \approx \mathsf{Pair}(0,\mathsf{s}(0)) \\ \wedge\quad \forall x.\,\mathsf{FFib}(\mathsf{s}(x)) \approx \mathsf{Pair}(\mathsf{Fib}(\mathsf{s}(x)),\mathsf{Fib}(\mathsf{s}(\mathsf{s}(x)))) \end{array} \right), \\ \forall x.\,\exists y.\,\mathsf{FFib}(x) \approx \mathsf{Pair}(y,\mathsf{Fib}(\mathsf{s}(x))) \end{array} \right\};\mathcal{R}_3$$

$$\overset{\mathbf{9}}{\cong} \left\{ \begin{array}{c} \mathsf{FFib}(0) \approx \mathsf{Pair}(0,\mathsf{s}(0)), \\ \forall x.\,\mathsf{FFib}(\mathsf{s}(x)) \approx \mathsf{Pair}(\mathsf{Fib}(\mathsf{s}(x)),\mathsf{Add}(\mathsf{Fib}(x),\mathsf{Fib}(\mathsf{s}(x)))), \\ \forall x.\,\exists y.\,\mathsf{FFib}(x) \approx \mathsf{Pair}(y,\mathsf{Fib}(\mathsf{s}(x))) \end{array} \right\};\mathcal{R}_3$$

$$\overset{\mathbf{10}}{\Rightarrow}_{\mathcal{GS}}^{\mathbf{Ded}} \left\{ \begin{array}{c} \forall x.\,\exists z.\,\mathsf{FFib}(x) \approx \mathsf{Pair}(z,\mathsf{Fib}(\mathsf{s}(x))), \\ \mathsf{FFib}(0) \approx \mathsf{Pair}(0,\mathsf{s}(0)), \\ \forall x. \left( \begin{array}{c} \left( \begin{array}{c} \mathsf{FFib}(\mathsf{s}(x)) \\ \approx\quad \mathsf{Pair}(\mathsf{Sec}(\mathsf{FFib}(x)),\mathsf{Add}(\mathsf{Fir}(\mathsf{FFib}(x)),\mathsf{Sec}(\mathsf{FFib}(x)))) \end{array} \right) \\ \wedge\quad \mathsf{Fir}(\mathsf{FFib}(x)) \approx \mathsf{Fib}(x) \\ \wedge\quad \mathsf{Sec}(\mathsf{FFib}(x)) \approx \mathsf{Fib}(\mathsf{s}(x)) \end{array} \right), \\ \forall x.\,\exists y.\,\mathsf{FFib}(x) \approx \mathsf{Pair}(y,\mathsf{Fib}(\mathsf{s}(x))) \end{array} \right\};\mathcal{R}_3$$

図 2　FFib のプログラム生成変換

## 5. 変換規則 Introduction の導入

合成関数の逆関数プログラムを生成する場合に，変換を進める過程で新しい関数記号が必要になることがある．

関数 $F, G$ およびそれらの合成関数 $I\ (x) \approx F(G(x))$ が既知プログラムとして定義され，$I$ の逆関数を表わす性質 $I^{-1}$ の仕様 $I^{-1}(I\ (x)) \approx x$ が与えられたとする．このとき，$I$ の逆関数である $I^{-1}(x)$ は $F$ および $G$ の逆関数 $F^{-1}$, $G^{-1}$ を生成することにより，$I^{-1}(x) \approx G^{-1}(F^{-1}(x))$ と表現することができる．しかしながら，$\mathcal{G}ene\mathcal{S}ys$ の 6 つの変換規則では新しい関数記号を作り出すことができない．この問題は関数記号 $F^{-1}$, $G^{-1}$ の仕様をそれぞれ与えることで，$F^{-1}$, $G^{-1}$ のプログラム生成を目指す部分問題に帰着できる．この要求に対応できるように，新しい関数記号を含む論理式を加える変換規則 **Introduction** を定義する．

［定義 5.1］　変換規則 **Introduction**

$$\frac{\mathcal{E}\ ;\ \mathcal{R}}{\mathcal{E} \cup \{P\}\ ;\ \mathcal{R}} \qquad \textit{if}\quad P \text{ は新しい関数記号を含む論理式.}$$

$\Rightarrow_{\mathcal{GS}}^{\mathbf{Int}}$ は変換規則 **Introduction** による変換を表す．　□

**Introduction** を $\mathcal{G}ene\mathcal{S}ys$ の変換規則として利用するためには **Introduction** が健全であること，すなわち，**Introduction** による変換で得られた論理式が変換前の論理式を満たしている必要がある．

等号付き一階述語論理の $\models$ に基づく意味論で考えれば，変換によって得られた論理式集合 $\mathcal{E}'$ が元の論理式集合 $\mathcal{E}$ を満たすためには，$\mathcal{E}' \models \mathcal{E}$ が成り立てばよい．

［定理 5.2］（**Introduction** の健全性） $\mathcal{E};\mathcal{R} \Rightarrow_{\mathcal{GS}}^{\mathbf{Int}} \mathcal{E}';\mathcal{R}$ ならば $\mathcal{E}' \cup \mathcal{E}_{\mathcal{R}} \models \mathcal{E} \cup \mathcal{E}_{\mathcal{R}}$．

［証明 ］ $\mathcal{E} \subset \mathcal{E}'$ であることから明らかである． □

**Introduction** の健全性により，**Introduction** をプログラム生成系 $\mathcal{G}ene\mathcal{S}ys$ の変換規則として利用することができる．

［例 5.3］ 自然数を 2 倍する関数 $\mathsf{D}$，2 のべき乗を計算する $\mathsf{Exp2}$ を定義する TRS $\mathcal{R}_4$ と，2 を底とする対数の整数部分を計算する $\mathsf{Log2}$ の仕様となる論理式集合 $\mathcal{E}_4$ が以下で与えられている．

$$\mathcal{R}_4 = \left\{ \begin{array}{c} \mathsf{D}(0) \to 0, \\ \mathsf{D}(\mathsf{s}(x)) \to \mathsf{s}(\mathsf{s}(\mathsf{D}(x))), \\ \mathsf{Exp2}(0) \to \mathsf{s}(0), \\ \mathsf{Exp2}(\mathsf{s}(x)) \to \mathsf{D}(\mathsf{Exp2}(x)) \end{array} \right\}$$

$$\mathcal{E}_4 = \left\{ \ \forall x.\, \mathsf{Log2}(\mathsf{Exp2}(x)) \approx x \ \right\}$$

$\mathcal{E}_4;\mathcal{R}_4$ を $\mathcal{G}ene\mathcal{S}ys$ によって変換する (図 3)．ただし，関数 $\mathsf{D}^{-1}$ は変換規則 **Introduction** により追加した関数記号であり，$\mathsf{D}$ の逆関数となっている．

変換によって得られた論理式集合から $\mathsf{Log2}$ を定義する TRS $\mathcal{R}_5$ が得られる．

$$\mathcal{R}_5 = \left\{ \begin{array}{c} \mathsf{Log2}(\mathsf{s}(0)) \to 0 \\ \mathsf{Log2}(y) \to \mathsf{s}(\mathsf{Log2}(\mathsf{D}^{-1}(y))) \end{array} \right\}$$

**Introduction** は論理式を追加する自由度の高い変換規則である．このため，冗長であったり矛盾する論理式であっても追加することが可能になり，必要な論理式を手作業で記述しなければならない．機械的に適用できる規則として実装するためには，追加することで変換が進む可能性のある論理式を特定する戦略が必要になる．実装の一案として，例 5.3 の場合のような逆関数の仕様を追加する機能に限定すれば，機械的に追加する論理式を有限個列挙することができる．

## 6. $\mathcal{G}ene\mathcal{S}ys$ ツールの概要

$\mathcal{G}ene\mathcal{S}ys$ による変換は多くの枝分かれが発生し，手作業で変換の全容を把握することは難しい．$\mathcal{G}ene\mathcal{S}ys$ による論理式変換を補助するため，機械的に実行できる工程を自動化するツールを実装した．ツールは，論理式 $\mathcal{E}$ (仕様)， TRS $\mathcal{R}$ (既知プログラム) を入力として受け取り，ユーザの選択した変換規則に対して，それを入力の論理式に適用して生成できる論理式の一覧を提示する．そして，ユーザは提示された候補一覧から生成したい論理式を選択する．この対話を繰り返すことで変換を進める．また， TRS による **Deduction** が可能な場合は規則の選択に先立って自動的に実行する．そのため，本ツールにおける変換規則 **Deduction** は論理式による書換え (定義 3.1) のみを指す．ツール動作の流れを次に示す．

$\mathcal{G}ene\mathcal{S}ys$ **ツールの処理手順**

（1） $\mathcal{R}$ による **Deduction** を可能な限り適用する．

（2） ユーザは論理式に対して適用したい変換規則を選択．**Introduction** を選択した場合は追加したい論理式を続けて入力する．

（3） 論理式と手順 (2) で選択された規則に対するすべての変換候補の一覧をユーザに提示．

（4） ユーザは提示された変換候補の一覧から生成したい論理式を選択．

（5） ツール終了を選択した場合は，手順 (4) で選択した論理式までの変換過程を出力して終了．

それ以外は選択した論理式を新たな入力として手順 (1) に戻り，変換を繰り返す．

手順 (2) では変換規則の選択前に，簡単なコマンド入力により一回の規則適用で可能な変換を閲覧することができる．リストから生成したい論理式を選択する仕組みにより，最も煩雑な作業である論理式の手入力を極力避けることができる．また，手順 (2) の変換規則は定義 3.3 の 6 つの変換規則，**Introduction** 規則のほかに，いくつかの論理式の等価変換も実装されている．

## 7. おわりに

本論文では，いくつかのプログラム合成例に $\mathcal{G}ene\mathcal{S}ys$ が適用可能であることを示した．また，新たな関数の仕様を導入する変換規則 **Introduction** をプログラム生成系 $\mathcal{G}ene\mathcal{S}ys$ に追加する提案をした．変換規則 **Introduction** による変換を必要とするプログラム生成例を挙げ，**Introduction** が $\mathcal{G}ene\mathcal{S}ys$ のプログラム生成に有効であることを明らかにした．

例 4.1 では，App の結合則をあらかじめ入力しておく必要性が，変換過程で判明した．しかしながらこの規則は App の帰納的性質として導けるものであり，仕様に含まれない方が自然である．今後の課題として，変換を進めるために有効な性質を特定することができるか検討する必要がある．さらに，それらの性質の正しさを書換え帰納法 [9] などを用いて証明する機能の追加も有用であると考えている．

また，本論文で紹介したツールは，$\mathcal{G}ene\mathcal{S}ys$ による変換を効率的に行うために有効である．しかし，ツールによる変換は現在のところ一方通行のため指定のステップに戻って変換をやり直すという処理ができない．作業のより一層の効率化のためにツールの機能拡充が必要になる．一方，$\mathcal{G}ene\mathcal{S}ys$ による変換には，変換対象の論理式が複雑になるにつれて変換候補の数が爆発的に増加する問題がある．本当に必要な変換候補が沈み込まないように，変換に有効でない候補を除去する戦略が求められる．ツールには変換候補の枝刈りを行う戦略を十分ではないが一部組み込んだ．しかし，論理式を変換する戦略の大部分はユーザの判断に委ねられており，システムの自動化に向けて有効な戦略の発見が必要である．

さらにいくつか今後の課題を挙げる．まず，手作業による変換では論理式の等価変換は自然に行われるが，ツール上で等価変換をするためには規則として組み込む必要がある．しかし，

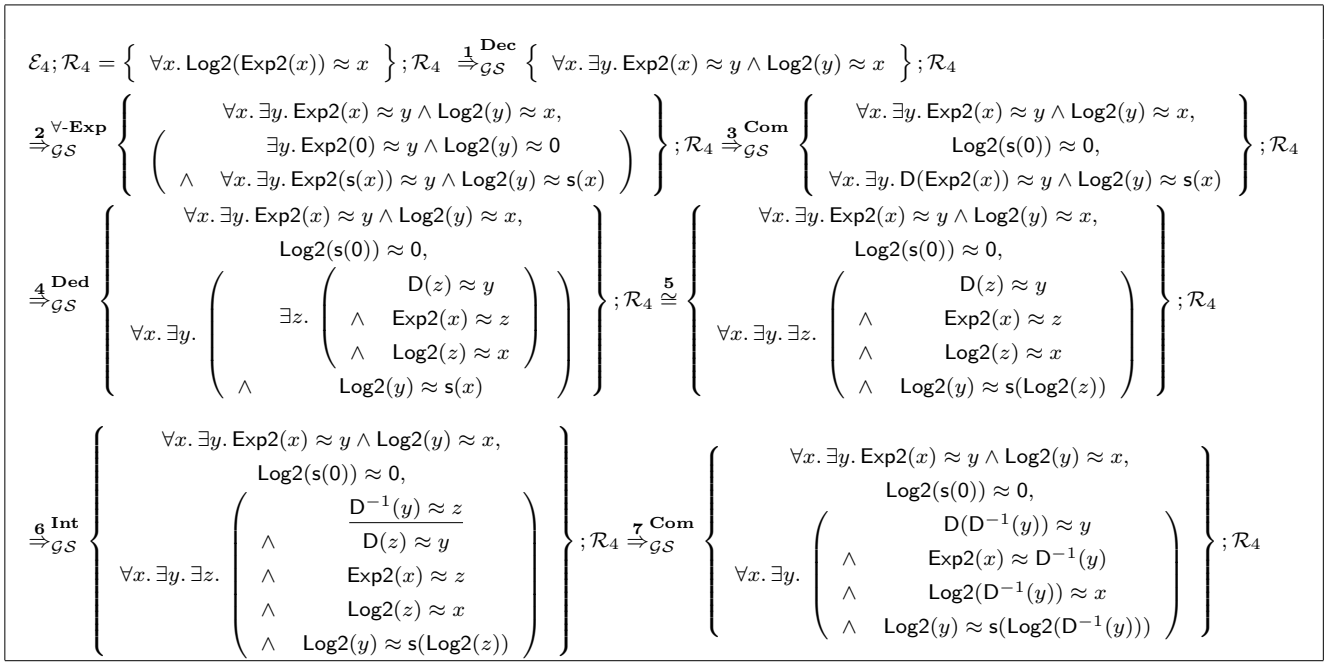


図 3　関数 Log2 のプログラム生成変換

有効な等価変換をすべて規則として追加することは変換の自動化を困難にする上，無駄な変換候補の増大によりツールの使い勝手を損なう．このため，等価変換をツール上で実現する方針が必要になる．次に，$\mathcal{G}ene\mathcal{S}ys$ がプログラム生成系として完全となるような仕様の条件を見つけ，$\mathcal{G}ene\mathcal{S}ys$ のプログラム生成能力を明らかにすることが挙げられる．最後に，目的のプログラムが生成できたことを検証する仕組みを定義できれば，$\mathcal{G}ene\mathcal{S}ys$ による変換の自動化も期待できる．

**謝辞**　本研究は一部，科研費#18500011，#16650005，#17700009 ならびに名古屋大学 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) の補助を受けている．

## 文　献

[1] 長島正憲, 酒井正彦, 坂部俊樹, 草刈圭一朗. 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, Vol. 21, No. 4, pp. 49–54, 2004.

[2] 長島正憲, 酒井正彦, 西田直樹, 坂部俊樹, 草刈圭一朗. 融合変換を模倣するプログラム生成変換の戦略. 電子情報通信学会, Vol. 104, No. 466, pp. 43–48, 2004.

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

[4] P. L. Wadler. Deforestation: Transforming programs to eliminate trees. *Theoretical Computer Science*, Vol. 73, No. 2, pp. 231–248, 1990.

[5] F. Baader and T. Nipkow. *Term Rewriting and All That*. Cambridge University Press, 1998.

[6] M. Ben-Ari. *Mathematical Logic for Computer Science*. Springer-Verlag, second edition, 2001.

[7] E. Mendelson. *Introduction to Mathematical Logic*. Chapman & Hall, fourth edition, 1997.

[8] D. A. Plaisted. Semantic confluence tests and completion methods. *Information and Control*, Vol. 65, No. 2/3, pp. 182–215, 1985.

[9] U. S. Reddy. Term rewriting induction. In *Proceedings of the 10th international conference on Automated deduction*, LNAI 449, pp. 162–177, 1990.