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

信学技報
IEICE Technical Report
SS2007-45(2007-12)

プログラム生成系 $\mathcal{G}ene\mathcal{S}ys$ における等式仕様への否定の導入

近藤　悟† 酒井 正彦†† 坂部 俊樹†† 草刈 圭一朗†† 西田 直樹††

† 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,sakabe,kusakari,nishida}@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，$\mathcal{G}ene\mathcal{S}ys$ で否定記号を含む論理式を扱うために既存の変換規則を拡張し，否定記号を用いた仕様からのプログラム生成の例を示す．また，否定記号に関連する変換規則などを新たに追加し，これによりプログラムの生成が可能になる例を示す．
キーワード　プログラム生成，項書換え系，等式仕様，否定記号

Extending program-generation system $\mathcal{G}ene\mathcal{S}ys$ for allowing negation in equational specifications

Satoru KONDO†, Masahiko SAKAI††, Toshiki SAKABE††,
Keiichirou KUSAKARI††, and Naoki NISHIDA††

† 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,sakabe,kusakari,nishida}@is.nagoya-u.ac.jp

Abstract　Program-Generation System $\mathcal{G}ene\mathcal{S}ys$ has been proposed, which generated executable programs from first order equational specifications without negation. In this paper, we extend conversion rules of $\mathcal{G}ene\mathcal{S}ys$ in order to handle formulas with negation, and show a program-generation example from a specification with negation. We also propose new conversion rules related to negation and give a example that indicates benefits of the new rules.
Key words　program generation, term rewriting system, equational specification, negation

1. はじめに

プログラムを対象とするいくつかの自動変換が提案されている．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 [4] などがある．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ず，一般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と考えられている．

仕様からプログラムへの変換手法としては，これまでにプログラム生成系 $\mathcal{G}ene\mathcal{S}ys$ [1], [3] が提案されている．$\mathcal{G}ene\mathcal{S}ys$ は項書換え系で与えられた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変換規則で構成され，健全性，すなわち，変換によりプログラムが得られればその正しさが保証されるという特長を持つ．その他にも，プログラム融合変換の一手法である Deforestation [5] 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].

本論文では，まず，$\mathcal{G}ene\mathcal{S}ys$ が扱える論理記号に否定記号を追加し，否定記号を導入した後の $\mathcal{G}ene\mathcal{S}ys$ もまた変換系としての健全性を保持できるように変換規則を修正する．また，これによりプログラムの生成に成功する例を示す．

否定記号の導入により仕様の表現力が高まった一方で，必要な情報が揃っているにも関わらず従来の変換規則では変換を進められないことも多い．これを改善するため２つの変換規則を追加する．１つ目の変換規則は，論理式中の等式を経験として利用し，類似の等式に含まれる変数の値を推論する変換規則 **Imply-Deduction**$_+$ である．２つ目は，条件付き項書換え系で表された既知プログラムに対して，条件の成立を仮定して等式による推論をおこなう変換規則 **R-Deduction**$_+$ である．また，これらの２つの変換規則の活用により可能になるプログラ

ム生成例を示す.

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 [6], [7], [8].

$\mathcal{S}$ を型の集合とする．変数 の（可算無限）集合を $\mathcal{X}$，関数記号の集合を $\mathcal{F}$ とする．各変数や関数記号には型が対応付けられており，次の型関数 τ によって表す．変数 $x \in \mathcal{X}$ が型 S を持つとき $\tau(x) = S$，関数記号 $f \in \mathcal{F}$ が引数の型 $S_1 \cdots S_n \in \mathcal{S}^*$，返値の型 S を持つとき $\tau(f) = (S_1 \cdots S_n, S)$ である．関数記号の集合 $\mathcal{F}$ と変数の集合 $\mathcal{X}$ 上の型付き項は，次のように再帰的に定義される．

（1） 変数 $x \in \mathcal{X}$ は型 $\tau(x)$ の項である．

（2） 型 $S_1, \ldots, S_n$ の項 $t_1, \ldots, t_n$ と $\tau(f) = (S_1 \cdots S_n, S)$ の関数記号 $f \in \mathcal{F}$ に対して，$f(t_1, \ldots, t_n)$ は型 S の項である．ただし，項 $f()$ は f と略記する．変数を含まない項を基底項と呼び，その集合 $\mathcal{T}(\mathcal{F}, \emptyset)$ を $\mathcal{T}(\mathcal{F})$ と略記する．項の並び $t_1, \ldots, t_n$ は $\vec{t}$ 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．

論理記号は $\neg, \wedge, \vee, \Rightarrow, \forall, \exists$ の 6 個である．述語記号は 2 引数の $\approx$ のみである．関数記号の集合 $\mathcal{F}$，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） 同一の型を持つ項 s, t について等式 $s \approx t$ 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，限量子 $\forall x.$ 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．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，複数の同じ限量子の並びは，変数の束縛の順序を無視する．本論文では，等式 $s \approx t$ と $t \approx s$ は同一の等式とみなす．

変数にそれと同一の型の項を対応付ける写像 $\sigma : \mathcal{X} \to \mathcal{T}(\mathcal{F}, \mathcal{X})$ 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 σ を基底代入という．

特別な定数記号 $\square$ をちょうど 1 個含む項 C を項文脈と呼ぶ．項文脈 C の $\square$ 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 Γ を論理式文脈と呼び，論理式文脈 Γ 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．また，限量子の解釈を基底項に限定した意味論的帰結を $\models_g$ と表す [1]．命題：$\mathcal{E} \models_g V$ ならば $\mathcal{E} \models_g U$ が成り立つようなすべての論理式集合 $\mathcal{E}$ と論理式 U, V に対して，命題：$\mathcal{E} \models_g \Gamma\langle V \rangle$ ならば $\mathcal{E} \models_g \Gamma\langle U \rangle$ も成り立つとき，Γ は単調であるという．逆に $\mathcal{E} \models_g \Gamma\langle U \rangle$ ならば $\mathcal{E} \models_g \Gamma\langle V \rangle$ が成り立つとき，Γ は逆単調であるという．

条件付き書換え規則は次の条件を満たす 3 項組 $(l, r, Cond)$ であり，$l \to r \Leftarrow Cond$:

- $l, r \in \mathcal{T}(\mathcal{F}, \mathcal{X})$ かつ $l \notin \mathcal{X}$
- $Cond$ は条件部と呼ばれ，$true$ または $Cond_1 \wedge \cdots \wedge Cond_n$ $(n > 0)$ の形式をしている．

ここで $Cond_i$ は項 s_i, t_i からなる等式 $s_i \approx t_i$ である．条件部が true である規則 $l \to r \Leftarrow true$ を単に書換え規則と呼び，$l \to r$ と省略する．書換え規則の有限集合を項書換え系（TRS），条件付き書換え規則の有限集合を条件付き項書換え系（CTRS）という．書換え規則 $l \to r \Leftarrow Cond$ において規則の条件部もしくは右辺のみに現れる変数を余剰変数と呼ぶ．条件付き項書換え系 $\mathcal{R}$ に対して，論理式集合 $\mathcal{E}_\mathcal{R}$ を次のように定める．

$$\mathcal{E}_\mathcal{R} = \{\forall \vec{x}.\exists \vec{y}.Cond \Rightarrow l \approx r \mid l \to r \Leftarrow Cond \in \mathcal{R},$$
$$\{\vec{x}\} = \mathcal{V}ar(l),\ \{\vec{y}\} \text{ は余剰変数の集合 } \}$$

プログラム生成系 $\mathcal{G}eneSys$ [1], [3] は既知プログラムとして与えられる項書換え系を参照しながら仕様を表現した論理式を変換する変換規則で構成される．例えば，既知プログラムとしてある関数 F の動作が項書換え系 $\mathcal{R}$ で定義されているとする．このとき，逆関数 G の仕様 $\mathcal{E}$ は，$\mathcal{E} = \{\forall x.\mathrm{F}(\mathrm{G}(x)) = x\}$ と与えることができる．$\mathcal{G}eneSys$ による変換で G の動作を定義する論理式 $\mathcal{E}'$ が得られたとすると，$\mathcal{E}'$ と $\mathcal{E}_\mathcal{R}$ のもとで $\mathcal{E}$ が成立する（$\mathcal{E}' \cup \mathcal{E}_\mathcal{R} \models_g \mathcal{E}$）．この性質を $\mathcal{G}eneSys$ の健全性という．この性質から，変換により実行可能な形式の論理式が得られれば，それは仕様を満たすプログラムになっていることが保証される．

3. 否定を導入したプログラム生成系 $\mathcal{G}eneSys$

否定の導入に先立ち，多様な仕様を扱うために有用な型を導入する．これは，変数の場合分けに相当する変換規則 $\forall$-**Expansion** において意図しない場合分けを行わないようにするためである．以下では，$\mathcal{G}eneSys$ は型付きに自然な拡張がなされているものとして議論する．

関数の仕様を表す論理式に否定記号を利用することで，より自然な仕様を定義できることがある．例えば自然数の減算 Sub の仕様を，自然数の加算 Add を用いて $\{\forall x.\forall y.\mathrm{Add}(\mathrm{Sub}(x, y), y) \approx x\}$ と与える．しかしながら，実際にはこの仕様は $x \geq y$ であるときのみ成り立つものであるため，意図しない動作をする関数が生成される可能性がある．そこで，論理記号 $\Rightarrow$ と自然数上の順序を定義する関数 Ge を用いて $\{\forall x.\forall y.\mathrm{Ge}(x, y) \approx \mathrm{True} \Rightarrow \mathrm{Add}(\mathrm{Sub}(x, y), y) \approx x\}$ のように成立する条件を記述することにより正しく仕様を書くことが可能になる．しかしながら，これまでの $\mathcal{G}eneSys$ では否定記号を用いることができなかったためこのような論理式を扱うことができなかった．そこで，プログラム生成系 $\mathcal{G}eneSys$ で否定記号を含む論理式を扱えるように変換規則の定義を拡張する．

$\mathcal{G}eneSys$ で否定記号を扱うことを許していなかった理由は以下のとおりである．論理式 $\Gamma\langle U_1 \rangle$ から変換規則の適用によ

り $\Gamma\langle U_2\rangle$ が得られるとすると，*GeneSys* では健全性を保つために，Γ は単調な論理式文脈に制限し，かつ $U_2 \cup \mathcal{E}_\mathcal{R} \models_g U_1$ が成立するように変換規則を設計していたためである．否定記号がなければどの論理式文脈も単調であるため，確かに $\Gamma\langle U_2\rangle_+ \cup \mathcal{E}_\mathcal{R} \models_g \Gamma\langle U_1\rangle_+$ が成立し，変換の健全性を保つことができる．一方，Γ が逆単調な文脈 Γ_- である場合には逆向きの関係 $U_1 \cup \mathcal{E}_\mathcal{R} \models_g U_2$ が成立するように変換規則を設計すれば健全性は保たれる．そこで，否定記号を含む逆単調な文脈を扱えるように各々の変換規則の見直しを行った．以下では，Γ_+ は単調な論理式文脈を，Γ_- は逆単調な論理式文脈を表すことにする．

まず，全称限量子で束縛された変数の場合分けを行う変換規則 ∀-**Expansion** を見直し，単調，逆単調それぞれの文脈に適用できる ∀-**Expansion**$_+$，∀-**Expansion**$_-$ を定義する．ただし，名前に $+$，$-$ が付いた変換規則はそれぞれ単調，逆単調の文脈にのみ適用できることを表す．各変換規則は *if* 節の条件を満たすときに上の論理式から下への変換を行うことができる．

- ∀-**Expansion**$_+$

$$\frac{\mathcal{E} \cup \{\Gamma\langle \forall x_{:A}.\, U\rangle_+\} \;;\; \mathcal{R}}{\mathcal{E} \cup \{\Gamma\langle \bigwedge_i \forall \overrightarrow{y_{i:B_i}}.\, U\sigma_i\rangle_+\} \;;\; \mathcal{R}}$$

if $\quad \{t_1, \ldots, t_n\} \in \mathcal{CS}(A),\ \sigma_i = \{x \mapsto t_i\},\ \{\overrightarrow{y_i}\} = \mathcal{V}ar(t_i)$

- ∀-**Expansion**$_-$

$$\frac{\mathcal{E} \cup \{\Gamma\langle \forall x_{:A}.\, U\rangle_-\} \;;\; \mathcal{R}}{\mathcal{E} \cup \{\Gamma\langle \bigwedge_i \forall \overrightarrow{y_{i:B_i}}.\, U\sigma_i\rangle_-\} \;;\; \mathcal{R}}$$

if $\quad \sigma_i = \{x \mapsto t_i\},\ \{\overrightarrow{y_i}\} = \mathcal{V}ar(t_i),\ \tau(t_i) = A$

GeneSys では場合分けが網羅すべき集合を定義するために被覆集合 [9] の概念を利用している．条件付き項書換え系 $\mathcal{R}$ として，型 A のすべての基底項 s に対して，ある項 $t \in K$ とある代入 σ が存在して $s = t\sigma$ を満たすとき，項の集合 K を型 A の被覆集合であるといい，型 A のすべての被覆集合を $\mathcal{CS}(A)$ で表す．型 A を持つ変数 x を $t_1, \ldots, t_n$ の n 個の場合に分けるとき，単調な論理式文脈で場合分けるために必要な条件は項集合 $\{t_1, \ldots, t_n\}$ が変数 x の型 A の被覆集合になっていることである．また，逆単調な文脈の場合には項 t_i は型 A のいかなる項であっても健全性は崩れないが，項集合 $\{t_1, \ldots, t_n\}$ が型 A の定義する集合そのものと一致するとき任意の文脈に適用できるため，等価な変換規則 ∀-**Expansion** として利用できる．

∀-**Expansion** 以外の変換規則を図 1 に示す．∀ の場合と同様に，∃ で束縛された変数の場合分けを行う変換規則 ∃-**Expansion** も逆単調な文脈に適用できる規則として定義する．一方，ほかの 4 つの変換規則は等価な変換をおこなう規則であり，文脈の単調性に関係なく適用できる．**E-Deduction** は論理式中の等式を規則として推論する変換規則であり，**R-Deduction** は既知プログラムによる推論を定義している．また，合成関数を分解する **Decomposition**，関数を合成する **Composition** も同様に等価変換規則になっている．ただし，これまで項書換え系で表現していた既知プログラムについて，条件付き書換え系 (CTRS) で表現する拡張をおこなった．これに対応するため，**R-Deduction** は条件部 *Cond* が成立するときのみ適用できるように変更した．

- ∃-**Expansion**$_+$

$$\frac{\mathcal{E} \cup \{\Gamma\langle \exists x_{:A}.\, U\rangle_+\} \;;\; \mathcal{R}}{\mathcal{E} \cup \{\Gamma\langle \bigvee_i \exists \overrightarrow{y_{i:B_i}}.\, U\sigma_i\rangle_+\} \;;\; \mathcal{R}}$$

if $\quad \sigma_i = \{x \mapsto t_i\},\ \{\overrightarrow{y_i}\} = \mathcal{V}ar(t_i),\ \tau(t_i) = A$

- ∃-**Expansion**$_-$

$$\frac{\mathcal{E} \cup \{\Gamma\langle \exists x_{:A}.\, U\rangle_-\} \;;\; \mathcal{R}}{\mathcal{E} \cup \{\Gamma\langle \bigvee_i \exists \overrightarrow{y_{i:B_i}}.\, U\sigma_i\rangle_-\} \;;\; \mathcal{R}}$$

if $\quad \{t_1, \ldots, t_n\} \in \mathcal{CS}(A),\ \sigma_i = \{x \mapsto t_i\},\ \{\overrightarrow{y_i}\} = \mathcal{V}ar(t_i)$

- **E-Deduction**

$$\frac{\mathcal{E} \cup \{\Gamma\langle C[l\sigma] \approx t\rangle\} \;;\; \mathcal{R}}{\mathcal{E} \cup \{\Gamma\langle \xi_{\overrightarrow{y_{:\tau(y)}}}.\, C[r\sigma] \approx t \wedge l\sigma \approx r\sigma \wedge W\sigma\rangle\} \;;\; \mathcal{R}}$$

if $\quad \forall \overrightarrow{x_{:\tau(x)}}.\, \exists \overrightarrow{y_{:\tau(y)}}.\, l \approx r \wedge W \in \mathcal{E},\ \mathcal{V}ar(l) = \{\overrightarrow{x}\},\ \{\overrightarrow{y}\} \bigcap \mathcal{V}ar(\Gamma\langle C \approx t\rangle) = \phi$

- **Decomposition**

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

if $\quad x \notin \mathcal{V}ar(\Gamma\langle C[t] \approx s\rangle),\ \tau(t) = A$

- **R-Deduction**

$$\frac{\mathcal{E} \cup \{\Gamma\langle C[l\sigma] \approx t \wedge (Cond)\sigma\rangle\} \;;\; \mathcal{R}}{\mathcal{E} \cup \{\Gamma\langle C[r\sigma] \approx t \wedge (Cond)\sigma\rangle\} \;;\; \mathcal{R}} \quad \textit{if} \quad l \to r \Leftarrow Cond \in \mathcal{R}$$

- **Composition**

$$\frac{\mathcal{E} \cup \{\Gamma\langle \exists x_{:A}.\, (x \approx t \wedge U)\rangle\} \;;\; \mathcal{R}}{\mathcal{E} \cup \{\Gamma\langle U\sigma\rangle\} \;;\; \mathcal{R}} \quad \textit{if} \quad \begin{array}{l} x \notin \mathcal{V}ar(t), \\ \sigma = \{x \mapsto t\} \end{array}$$

図 1　*GeneSys* の変換規則

否定記号を考慮した *GeneSys* の拡張により，次の例のような否定を含む論理式仕様からのプログラム生成が可能になる．

[例 3.1]　自然数 Nat の加算を計算する関数 Add (Nat∗Nat ↦ Nat) および自然数上の順序を判定する Ge (Nat∗Nat ↦ Bool) を定義する CTRS $\mathcal{R}_1$ と，自然数の減算を計算する関数 Sub (Nat ∗ Nat ↦ Nat) 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．

$$\mathcal{R}_1 = \left\{ \begin{array}{c} \mathrm{Add}(x, 0) \to x \\ \mathrm{Add}(x, \mathrm{S}(y)) \to \mathrm{S}(\mathrm{Add}(x, y)) \\ \mathrm{Ge}(0, \mathrm{S}(y)) \to \mathrm{False} \\ \mathrm{Ge}(x, 0) \to \mathrm{True} \\ \mathrm{Ge}(\mathrm{S}(x), \mathrm{S}(y)) \to \mathrm{Ge}(x, y) \end{array} \right\}$$

$$\mathcal{E}_1 = \left\{ \begin{array}{c} \forall x_{:\mathrm{Nat}}.\, \forall y_{:\mathrm{Nat}}.\, \mathrm{Sub}(\mathrm{Add}(x, y), y) \approx x, \\ \forall x_{:\mathrm{Nat}}.\, \forall y_{:\mathrm{Nat}}.\, \mathrm{Ge}(x, y) \approx \mathrm{True} \\ \Rightarrow \mathrm{Add}(\mathrm{Sub}(x, y), y) \approx x \end{array} \right\}$$

型 Nat および Bool は以下のように定義する．

$$\mathcal{S}_{\mathcal{R}_1} = \left\{ \; \mathrm{Nat} := 0 \mid \mathrm{S}(\mathrm{Nat}),\ \mathrm{Bool} := \mathrm{True} \mid \mathrm{False} \; \right\}$$

$\mathcal{E}_1;\mathcal{R}_1$ を *GeneSys* によって変換する (図 2)．$\Rightarrow_{\mathcal{GS}}^{Rule}$ は *GeneSys* の変換規則 *Rule* による変換を表す．また，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す．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 ($\overset{i}{\Rightarrow}_{\mathcal{GS}}$ は i 回目の変換)．変換によって得られた論理

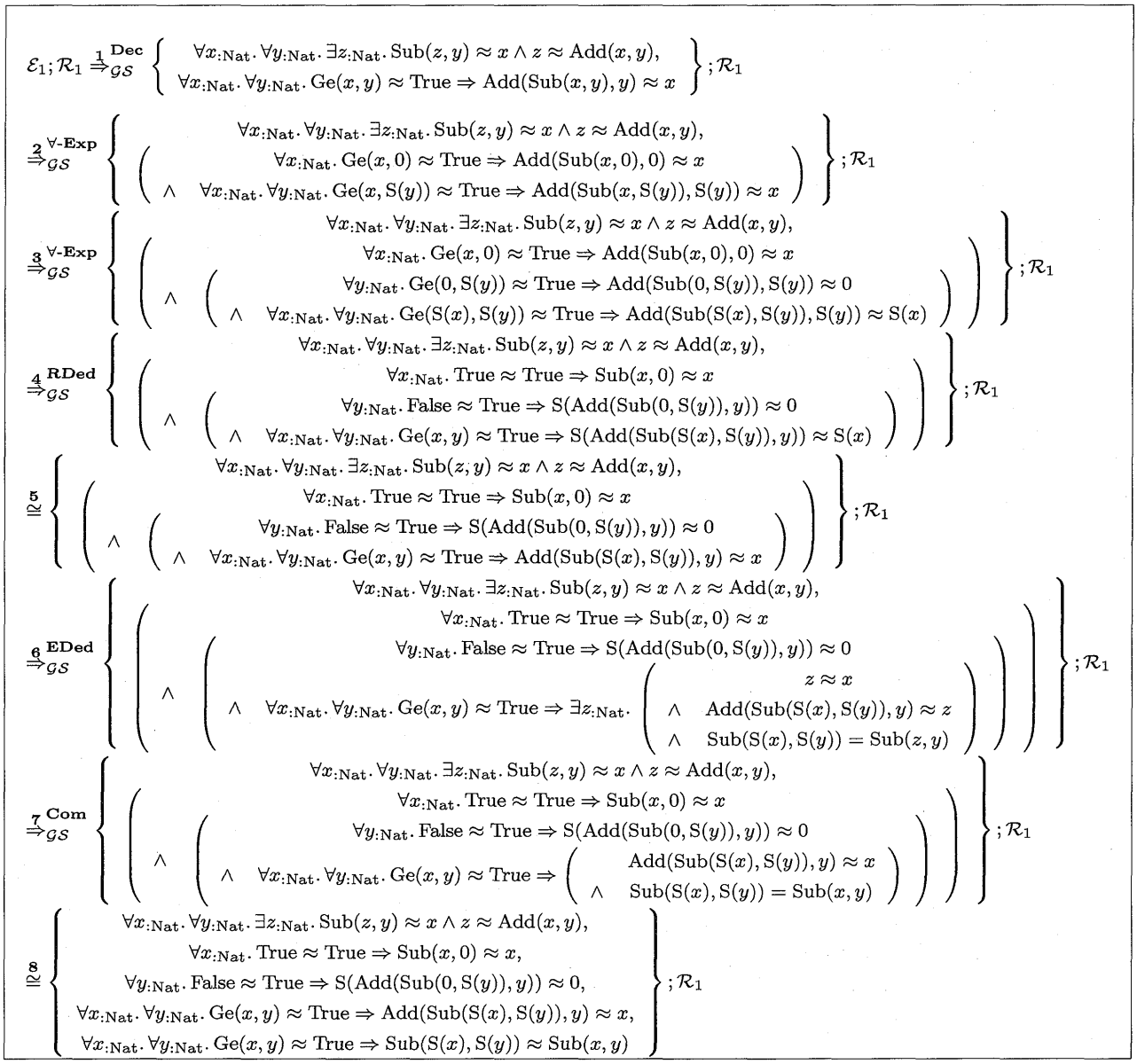

図 2　Sub のプログラム生成変換

式集合から Sub を定義する CTRS $\mathcal{R}_2$ が取り出せる.

$$\mathcal{R}_2 = \left\{ \begin{array}{r} \mathrm{Sub}(x,0) \to x \\ \mathrm{Sub}(\mathrm{S}(x),\mathrm{S}(y)) \to \mathrm{Sub}(x,y) \\ \Leftarrow \mathrm{Ge}(x,y)\approx \mathrm{True} \end{array} \right\}$$

□

論理式 $\mathcal{E}$, $\mathcal{E}'$ について $\mathcal{E} \models_g \mathcal{E}'$ が成立するとき, $\mathcal{E}'$ を $\mathcal{E}$ の帰納的定理という. $\mathcal{E} \models_g \mathcal{E}'$ が成り立つとき, $\mathcal{E} \models_g \mathcal{E} \cup \mathcal{E}'$ が成立するため, $\mathcal{E} \cup \mathcal{E}'$ から 帰納的定理 $\mathcal{E}'$ を除く変換は健全性を持つことが分かる. したがって, 例 3.1 の変換で得られた論理式集合から帰納的定理を除去し, プログラム $\mathcal{R}_2$ を取り出せることは容易に確かめることができる. 変換規則の健全性より, 図 2 の変換で得られた論理式集合は仕様を満たしている. これは, 仕様として与えた論理式集合 $\mathcal{E}_1$ が, プログラムとして抽出できた $\mathcal{R}_2$ と既知プログラム $\mathcal{R}_1$ のもとで帰納的定理になっていることからも確認できる.

4. 新しい変換規則の導入とプログラム生成例

本節では従来の *GeneSys* では取り出せなかった仕様の持っている情報を自然に推論できる 2 つの変換規則を新たに定義する. また, それらの変換規則を利用したプログラム生成の一例を挙げ, 変換規則の有用性を示す.

4.1 論理式集合の情報を引き出す変換規則

論理式集合を用いた変換には等価変換規則である **E-Deduction** が定義されている. 論理式に基づく推論により有用な変換を行う規則 **Imply-Deduction**$_+$ を次に定義する.

- **Imply-Deduction**$_+$

$$\frac{\mathcal{E} \cup \left\{ \Gamma\langle \forall \overrightarrow{x_{:\tau(x)}}.\exists z_{:\tau(z)}.C[z]\approx s \rangle_+ \right\};\ \mathcal{R}}{\mathcal{E} \cup \left\{ \Gamma\langle \forall \overrightarrow{x_{:\tau(x)}}.\exists z_{:\tau(z)}.C[z]\approx s \wedge z\approx t \rangle_+ \right\};\ \mathcal{R}}$$

$$\textit{if} \quad \forall \overrightarrow{x_{:\tau(x)}}.C[t]\approx s \in \mathcal{E},\ z \notin \mathcal{V}ar(s),\ \mathcal{V}ar(t) \subseteqq \{\overrightarrow{x}\}$$

論理式 $\exists z.C[z]\approx s$ が成立するためには, 変数 z にある値 t' を代入した等式 $C[t']\approx s$ が成り立てばよい. 既に得られてい

る論理式集合に等式 $C[t] \approx s$ が含まれているならば，少なくとも $z \approx t$ であるときに等式が成立することが分かっている．このとき，変数 z が値 t を持つという仮定は自然なものである．

4.2 書換え規則の条件部を利用した変換規則

例 3.1 は論理式仕様に否定記号を含む変換を示している．一方で，入力として与えられた書換え規則に否定記号が含まれる(条件部を持つ) 場合には従来の変換規則のみでは不十分なことが分かってきた．

条件付き書換え規則 $l \to r \Leftarrow Cond \in \mathcal{R}$ について，書換えが行える条件 $Cond$ が成立しているか分からない場合でも書換えを進めたいことがある．このため，条件 $Cond$ の成立を仮定して書換えを進める次の変換規則 **R-Deduction$_+$** を定義する．

- **R-Deduction$_+$**

$$\frac{\mathcal{E} \cup \{\Gamma\langle C[l\sigma] \approx t\rangle_+\} \ ; \ \mathcal{R}}{\mathcal{E} \cup \{\Gamma\langle \exists \overrightarrow{y:B}.\, C[r\sigma] \approx t \wedge (Cond)\sigma\rangle_+\} \ ; \ \mathcal{R}}$$

$$if \quad l \to r \Leftarrow Cond \in \mathcal{R},\ \{\overrightarrow{y}\} \text{ は余剰変数の集合}$$

条件 $Cond$ が成立する場合には変換を進めるために有効だが，成立を仮定すると矛盾を生じる場合には論理式は False になりプログラムの生成に失敗してしまう．この規則の適用の際には，論理式に対して条件が成立するかどうかの見極めが必要になる．

否定の導入により再定義した変換規則に，本節で定義した2つの変換規則を加えた $GeneSys$ が健全性を持つ，すなわち，生成できたプログラムが仕様を満たすことを示す必要がある．$GeneSys$ の健全性は，限量子の解釈を基底項のみに限定した $\models_g$ に基づいて定義される．

[定理 4.1]（$GeneSys$ の健全性） $\mathcal{E};\mathcal{R} \ \Rightarrow_{GS} \ \mathcal{E}';\mathcal{R}$ ならば $\mathcal{E}' \cup \mathcal{E}_{\mathcal{R}} \models_g \mathcal{E} \cup \mathcal{E}_{\mathcal{R}}$

[証明] **Imply-Deduction$_+$** および **R-Deduction$_+$** 以外の変換規則により変換された場合については [1] で示されているのと同様に証明できる．**Imply-Deduction$_+$** による変換の場合は $\mathcal{E} \subset \mathcal{E}'$ より明らか．また，**R-Deduction$_+$** は $(Cond)\sigma$ が成り立っていることから $l\sigma \approx r\sigma$ であり，$C[l\sigma] \approx t$ が成立．論理式文脈が単調であることからこの規則による変換は健全性を持つ． □

変換規則 **Imply-Deduction$_+$** および **R-Deduction$_+$** の追加により，条件部を持つ書換え規則を入力とする次のようなプログラム生成が可能になる．

[例 4.2] 第1引数と第2引数のリストの連結が第3引数と一致するか判定する関数 Appcheck (NatL $*$ NatL $*$ NatL $\mapsto$ Bool) を定義する CTRS $\mathcal{R}_3$ と，2つのリストを連結する関数 App (NatL $*$ NatL $\mapsto$ NatL) 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．

$$\mathcal{R}_3 = \left\{ \begin{array}{c} \mathrm{Appcheck}(\mathrm{Nil}, ys, zs) \to \mathrm{True} \ \Leftarrow \ ys \approx zs \\ \mathrm{Appcheck}(x :: xs, ys, z :: zs) \to \mathrm{Appcheck}(xs, ys, zs) \\ \qquad\qquad \Leftarrow \ x \approx z \\ \mathrm{Appcheck}(x :: xs, ys, \mathrm{Nil}) \to \mathrm{False} \\ \mathrm{Appcheck}(xs, y :: ys, \mathrm{Nil}) \to \mathrm{False} \end{array} \right\}$$

$$\mathcal{E}_3 = \left\{ \begin{array}{l} \forall xs_{:\mathrm{NatL}}.\, \forall ys_{:\mathrm{NatL}}. \\ \quad \mathrm{Appcheck}(xs, ys, \mathrm{App}(xs, ys)) \approx \mathrm{True} \end{array} \right\}$$

型 Nat，NatL および Bool は以下のように定義する．

$$\mathcal{S}_{\mathcal{R}_3} = \mathcal{S}_{\mathcal{R}_1} \bigcup \left\{ \ \mathrm{NatL} := \mathrm{Nil} \mid \mathrm{Nat} :: \mathrm{NatL}, \ \right\}$$

$\mathcal{E}_3;\mathcal{R}_3$ を $GeneSys$ によって変換する (図 3)．$\Rightarrow_{GS}^{\mathbf{RDed}_+}$，$\Rightarrow_{GS}^{\mathbf{Imp\text{-}Ded}_+}$ はそれぞれ変換規則 **R-Deduction$_+$**，**Imply-Deduction$_+$** による変換を表す．変換によって得られた論理式集合から App を定義する CTRS $\mathcal{R}_4$ が得られる．

$$\mathcal{R}_4 = \left\{ \begin{array}{c} \mathrm{App}(\mathrm{Nil}, ys) \to ys \\ \mathrm{App}(x :: xs, ys) \to x :: \mathrm{App}(xs, ys) \end{array} \right\}$$

□

7ステップ目における **Imply-Deduction$_+$** による変換では，等式 $\mathrm{Appcheck}(xs, ys, zs) \approx \mathrm{True}$ から変数 zs の値が $\mathrm{App}(xs, ys)$ であるという推論をしている．App は仕様 $\mathcal{E}_3$ を満たすように定義されるはずであり，仕様を満たすように変数の値を推論することは自然である．実際に，プログラムとして抽出できた $\mathcal{R}_4$ と既知プログラム $\mathcal{R}_3$ のもとで仕様として与えた論理式集合 $\mathcal{E}_3$ が帰納的定理になっていることから $\mathcal{R}_3$ は仕様を満たすプログラムになっている．

5. おわりに

本論文では，$GeneSys$ に型を導入することで複数の異なるデータ構造を同時に取り扱えようになった．また，否定を含む仕様および既知プログラムを入力として扱えるように $GeneSys$ の変換規則を拡張した．より直観に近い自然な仕様を与えることが可能になり，これまで正しい仕様を与えられなかったために変換に失敗していた減算のプログラム生成が成功する例を示した．4節では新しい2つの変換規則を追加し，これらの変換規則の活用により条件部付き書換え規則を入力とする問題にも対応できる例を示した．追加した2つの変換規則はともに健全性を持つが，等価な変換規則ではない．このため，恒偽となる不正な論理式を導き出す適用により変換が失敗する可能性を持っている．

2つのリストを受け取り，1つ目のリストを整列させたリストと2つ目のリストが一致するか判定する関数を既知プログラムとして，挿入ソートプログラムを生成する例題に取り組んでいる．論理式と書換え規則がともに否定を含むため，適用できる変換規則が制限され変換が進まない状況が発生しており，このプログラム生成が今後の課題となっている．さらに課題として，全探索による自動的な変換をおこなうツールの実装が挙げられる．このためには，プログラムが生成できたことを検証する仕組みが必要である．この際，どのような論理式をプログラムと判断するかという定義が重要になる．また，探索の範囲が広すぎる場合には効率的な変換の枝刈りも併せて必要になる．

謝辞 本研究は一部，科研費#18500011，#16650005，#17700009 の補助を受けている．

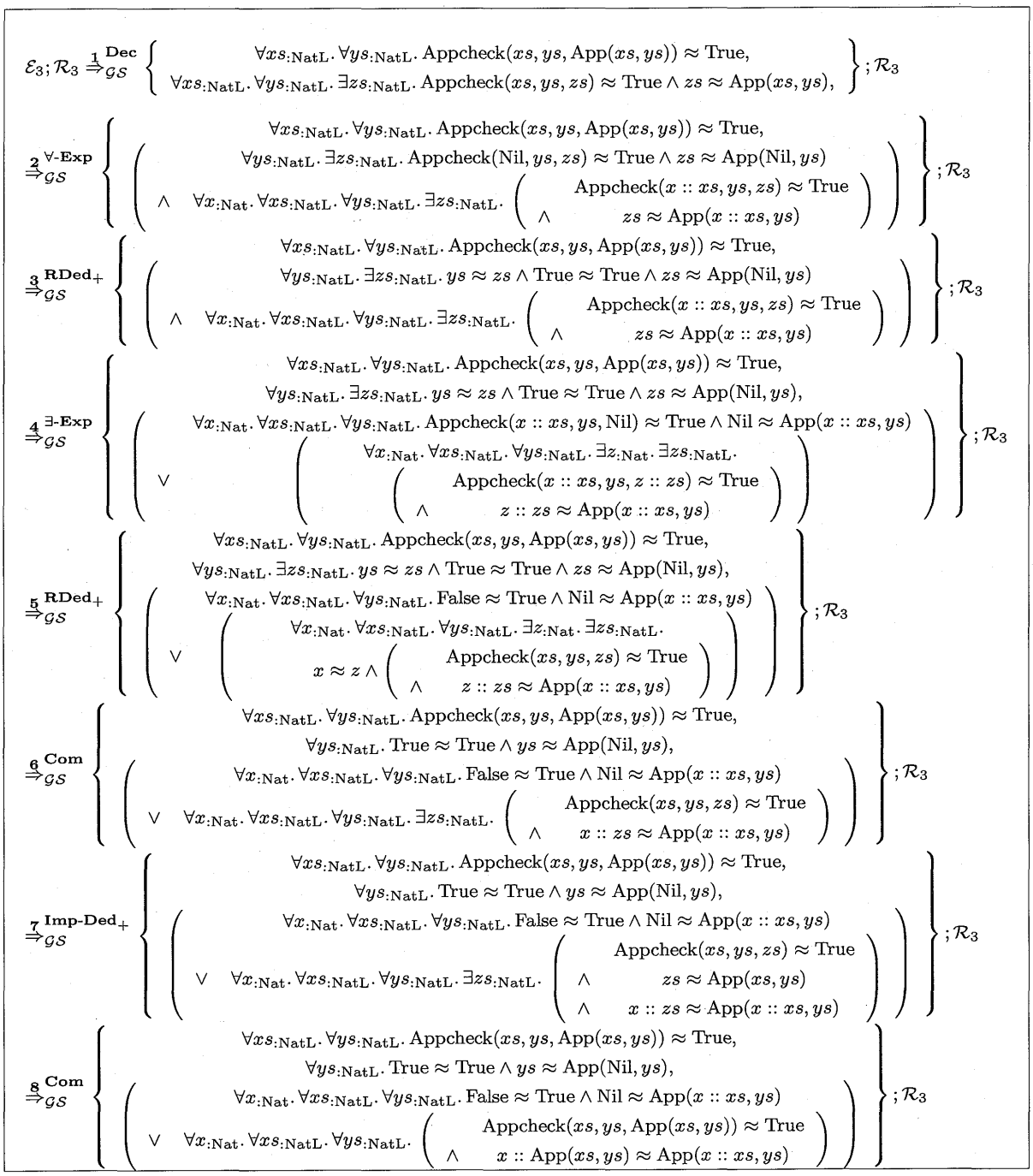

図 3　App のプログラム生成変換

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. 電子情報通信学会技術報告 (SS2004-33), Vol. 104, No. 466, pp. 43–48, 2004.

[3] 近藤悟, 酒井正彦, 西田直樹, 坂部俊樹, 草刈圭一朗. *GeneSys* によるプログラム生成例と Introduction 規則の追加. 電子情報通信学会技術報告 (SS2006-46), Vol. 106, No. 324, pp. 37–42 2006.

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

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

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

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

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

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