可逆プログラミング言語
構造化可逆言語 (SRL) から非構造化可逆言語 (RL) への
翻訳器の試作

2017SE050 宮本 昌武
2017SE051 水野 幹大
2017SE058 野端 祐人
指導教員：横山哲郎

1　はじめに

従来のプログラミング言語は、計算は不可逆的に実行される。可逆とは、計算過程において、常に直前と直後の状態が一意に定まり、情報が消失しないことをいう。すなわち、不可逆的に実行されるということは、計算は前方には決定的だが、後方には決定的ではない。逆に、そのプログラムの実行過程が必ず可逆になるような言語設計がなされているプログラミング言語を可逆プログラミング言語という。

本研究では、可逆プログラミング言語である構造化可逆言語 (SRL) から非構造化可逆言語 (RL) の翻訳器を、純粋な関数型プログラミング言語である Haskell を用いて実装する。手順として、SRL と RL の構文を BNF で書く 、すなわち、 字句解析器・構文解析器・プリティプリンタをプログラムする。次に、双方の表示的意味論、操作的意味論や、翻訳規則をプログラムし、翻訳器を実装する。更に、SRL/RL の評価規則、逆変換規則をプログラムし、インタープリタや逆変換規則を実装する。最後に、[1][2] で行われた SRL/RL の拡張を行う。

2　関連研究

2.1　可逆フローチャート

可逆フローチャートは、従来のフローチャートが、幅広く使われている従来のプログラミング言語のモデルとして使用されているように、可逆プログラミング言語のモデルとして使用されている。従来のフローチャートは、制御フローや、命令型のプログラムの構造を表している。但し、常に標準計算の基本理論を可逆言語に直接引き継げるとは限らない。更に、可逆フローチャートは、従来のフローチャートと表面的に似ているにもかかわらず、重要な違いが存在する。それは、どのステップの計算過程でも、情報が失われることがなく、全てのステップが、順方向か逆方向に決定的になることである。これにより、可逆フローチャートの、計算能力や、可逆プログラミングの特性、理論的で、実用的な従来のフローチャートの形式との違いを正確に捉えることができる。可逆フローチャートは、幅広い目的で使用される。コンパイラによって生成されたマシンコードの低レベルな側面と、高いレベルの汎用的なブロック構造化言語や、反復や条件付き文に対応する。

2.1.1　SRL

可逆フローチャートの有用性を示すため、高水準言語である構造化可逆言語 (SRL) という具体的な可逆プログラミング言語が存在する。この言語は、横山教授らによって作成された可逆プログラミング言語である。[3]

構文規則については、図 1、図 3、図 4 に示す。SRL は、プログラムが再帰的に繰り返されている構文形式のブロックで構成されている。ブロックは、ステップ演算、ブロックのシーケンス、if 文から成る条件付き文、ループの 4 つの構文がある。ステップ操作に関して、push x1 x2 は、x1 の値を x2 の先頭に移動し、x1 をゼロクリアする。pop x1 x2 は,x2 の先頭を、ゼロクリアする必要がある最初の引数 x1 に移動する。skip については、何もしない。x ^= 0 の省略形とみなせる。

2.1.2　RL

SRL と同じく、可逆フローチャートの有用性を示すため、低水準言語である非構造化可逆言語 (RL) という具体的な可逆プログラミング言語が存在する。この言語も、横山教授らによって作成された可逆プログラミング言語である。[3]

構文規則については、図 2、図 3、図 4 に示す。RL のブロックはラベル l、アサーション k から来るステップ演算、ジャンプの要素から構成されている。アサーションからの取得は無条件がある。つまり、ブロック l から行われる。非構造化プログラムには、 1 つの entry と 1 つの exit が含まれている。

SRL,RL の翻訳規則を図 5、図 6 に示す。このように、相互に翻訳規則が存在し、翻訳可能であるが、SRL から RL へ翻訳する機械は、現状存在していない。

2.1.3 構文

SRL,RL の 2 つの可逆フローチャート言語の構文を以下図 1、2、3、4 に示す。

$$
\begin{array}{lcl}
p & ::= & b \\
b & ::= & a \\
 & | & b \quad b \\
 & | & \text{if } e \text{ then } b \text{ else } b \text{ fi } e \\
 & | & \text{from } e \text{ do } b \text{ loop } b \text{ until } e
\end{array}
$$

図 1 SRL のブロックの構文 ([3] より)

$$
\begin{array}{lcl}
q & ::= & d^{+} \\
d & ::= & l \ : \ k \ a^{*} \ j \\
k & ::= & \text{from } l \\
 & | & \text{fi } e \text{ from } l \text{ else } l \\
 & | & \text{entry} \\
j & ::= & \text{goto } l \\
 & | & \text{if } e \text{ goto } l \text{ else } l \\
 & | & \text{exit}
\end{array}
$$

図 2 RL の構文 ([3] より)

$$
\begin{array}{rcl}
a & ::= & x \oplus = \ e \ \mid \ x[e] \oplus = \ e \ \mid \ \text{push } x \ x \ \mid \ \text{pop } x \ x \ \mid \ \text{skip} \\
e & ::= & c \ \mid \ x \ \mid \ x[e] \ \mid \ e \otimes e \ \mid \ \textbf{top} \ x \ \mid \ \textbf{empty} \ x \\
c & ::= & 0 \ \mid \ 1 \ \mid \ \cdot \cdot \cdot \ \mid \ 4294967295 \\
\otimes & ::= & \oplus \ \mid \ * \ \mid \ / \ \mid \ = \ \mid \ < \ \mid \ > \ \mid \ <= \ \mid \ > \ \mid \ != \\
\oplus & ::= & + \ \mid \ - \ \mid \ \hat{} \
\end{array}
$$

図 3 可逆ステップ演算と式の構文 ([3] より)

$$
\begin{array}{rl}
\text{SRL}: & p \in \text{SRL} \quad b \in \text{Blk} \\
\text{RL}: & q \in \text{RL} \quad d \in \text{RLblk} \quad j \in \text{Jump} \quad k \in \text{From} \quad l \in \text{Label} \\
\text{SRL, RL}: & a \in \text{Step} \quad e \in \text{Exp} \quad c \in \text{Const} \quad x \in \text{Var} \quad \oplus, \otimes \in \text{Op}
\end{array}
$$

図 4 SRL と RL の構文ドメイン ([3] より)

2.1.4 SRL,RL の翻訳規則

SRL から RL への翻訳規則を図 5 に、RL から SRL への翻訳規則を図 6 に示す。

$\tau_{SRL}[[b]]$ =
l_0 : entry
goto l_1
$\tau[[b]]$ (l_0 , l_1 , l_2 , l_3)
l_3 : from l_2
exit
where l_0 , l_1 , l_2 , l_3 are fresh

$\tau[[b_1\ b_2]]$ (l_0 , l_1 , l_4 , l_5) =
$\tau[[b_1]]$ (l_0 , l_1 , l_2 , l_3)
$\tau[[b_2]]$ (l_2 , l_3 , l_4 , l_5)
where l_2 , l_3 are fresh

$\tau[[a]]$ (l_0 , l_1 , l_2 , l_3) =
l_1 : from l_0
a
goto l_2
l_2 : from l_1
goto l_3

$\tau[[\text{if } e_1 \text{ then } b_1 \text{ else } b_2 \text{ fi } e_2\ ]]$ (l_0 , l_1 , l_6 , l_7) =
l_1 : from l_0
if e_1 goto l_2 else l_4
$\tau[[b_1]]$ (l_1 , l_2 , l_3 , l_6)
$\tau[[b_2]]$ (l_1 , l_4 , l_5 , l_6)
l_6 : fi e_2 from l_3 else l_5
goto l_7
where l_2 , l_3 , l_4 , l_5 are fresh

$\tau[[\text{from } e_1 \text{ do } b_1 \text{ loop } b_2 \text{ until } e_2\ ]]$ (l_0 , l_1 , l_6 , l_7) =
l_1 : fi e_1 from l_0 else l_6
goto l_2
$\tau[[b_1]]$ (l_1 , l_2 , l_3 , l_4)
$\tau[[b_2]]$ (l_4 , l_5 , l_6 , l_1)
l_4 : from l_3
if e_2 goto l_7 else l_5
where l_2 , l_3 , l_5 , l_6 are fresh

図 5 SRL から RL への翻訳規則 ([3] より)

$$
\begin{aligned}
\tau_{RL}[[q]] \;=\;& \text{if } \wedge_{i=0}^{n} \wedge_{j=1}^{n+1} \wedge_{k=0}^{2} \neg x_{i,j}^{k} \text{ fi true} \\
& x_{0.1}^{0} \;\hat{}= \text{ true} \\
& \text{from } \mathrm{x}_{0.1}^{0} \text{ do} \\
& \quad \tau_{blks}[[q]] \\
& \text{until } \mathrm{x}_{n,n+1}^{0} \\
& x_{n,n+1}^{0} \;\hat{}= \text{ true} \\
& \text{if } \wedge_{i=1}^{n} \wedge_{j=0}^{n+1} \wedge_{k=0}^{2} \neg x_{i,j}^{k} \text{ fi true}
\end{aligned}
$$

$$
\begin{aligned}
\tau_{blks}[[d_1\ d_2 \cdot\cdot\cdot d_n]] &= \tau_{blks}[[d_1]](\tau_{blks}[[d_2]](\cdot\ \cdot\ (\tau_{blks}[[d_n]](E_{\text{false}}))\ \cdot\ \cdot)) \\
\tau_{blk}[[l_i\ :\ k\ a^*\ j]](F) &= \tau_{flow}[[k]](i\ , \tau_{step}[[a^*]](i\ , \tau_{flow}[[j]](i\ , \mathrm{F})))
\end{aligned}
$$

$$
\tau_{flow}[[\text{fi } e \text{ from } l_j \text{ else } l_k]](i\ , F) = \text{if } x_{j,i}^{0} \ \vee\ x_{k,i}^{0} \text{ then if } x_{j,i}^{0} \text{ then } P_{j,i} \text{ else } P_{k,i} \text{ fi } e \text{ else } F \text{ fi } x_{i,i}^{1}
$$

$$
\tau_{flow}[[\text{if } e \text{ goto } l_j \text{ else } l_k]](i\ , F) = \text{if } x_{i,i}^{2} \text{ then if } e \text{ then } R_{i,j} \text{ else } R_{i,k} \text{ fi } x_{i,j}^{0} \text{ else } F \text{ fi } x_{i,j}^{0} \ \vee\ x_{i,k}^{0}
$$

$$
\begin{aligned}
\tau_{flow}[[\text{from } l_j]](i\ , F) &= \text{if } x_{j,i}^{0} \text{ then } P_{j,i} \text{ else } F \text{ fi } x_{i,i}^{1} \\
\tau_{flow}[[\text{goto } l_j]](i\ , F) &= \text{if } x_{i,i}^{2} \text{ then } R_{i,j} \text{ else } F \text{ fi } x_{i,j}^{0}
\end{aligned}
$$

$$
\begin{aligned}
\tau_{flow}[[\text{entry}]](i\ , F) &= \text{if } x_{j,i}^{0} \text{ then } P_{j,i} \text{ else } F \text{ fi } x_{i,i}^{1} \\
\tau_{flow}[[\text{exit}]](n\ , F) &= \text{if } x_{n,n}^{2} \text{ then } R_{n,n+1} \text{ else } F \text{ fi } x_{n,n+1}^{0}
\end{aligned}
$$

$$
\tau_{steps}[[a^*]](i\ , F) = \text{if } x_{i,i}^{1} \text{ then } a^*\ ;\ P_{i,i} \text{ else } F \text{ fi } x_{i,i}^{2}
$$

図 6 RL から SRL への翻訳規則

([3] より)

図 5 について、SRL のブロック b が 2 つで構成されている場合、2 つの新しいラベルを用いて、個別に変換される。ステップ演算である a は、l_0 から受け取るブロック l_1 と、制御を l_3 に渡すブロック l_2 に変換される。条件文と、ループ文には、互いに類似しているが、b_1 と b_2 によるブロック間の制御フローつなぎが異なる。2 つのブロックは、内部の制御フローをつなぐ 4 つの新しいラベルを使用して、τ への再帰的な呼び出しによって個別に変換される。

また、本研究では、SRL から RL への翻訳器を実装することを目的とするため、図 6 については利用しない。

2.1.5 SRL/RL の評価規則

2.1.6 SRL/RL の逆変換規則

2.1.7 SRL/RL の拡張

2.2 可逆プログラミング言語

可逆プログラミング言語には、SRL、RL の他に、高水準の命令型可逆言語である Janus[4], R[], SyReC[], 低水準の命令型可逆言語である Bob[], PISA[] などが存在する。

2.2.1 Janus

Janus とは初の可逆構造化プログラミング言語であると思われる。シンプルでかつ強力であり、可逆言語を設計するうえで便利なモデルとなる。ステートメントは可逆制御フロー演算子（条件分岐、ループ）、スタック操作（push,pop）、ローカル変数ブロック、プロシージャ呼び出し、スキップまたはステートメントシーケンスから構成されている。

2.2.2 R

2.2.3 SyReC

2.2.4 Bob

2.2.5 PISA

3 目的

SRL と RL の 字句解析器・構文解析器・プリティプリンタを書き、表示的意味論/操作的意味論も書く。更に、SRL から RL への翻訳規則を書き、翻訳器を Haskell で実装する。また、SRL/RL の評価規則や逆変換規則を書き、インタープリタや逆変換規則も実装する。

4 実装

4.1 SRL から RL への翻訳機の実装

4.2 インタープリタの実装

4.3 逆変換規則の実装

4.4 SRL/RL の拡張部分の実装

5 おわりに

参考文献

[1] Moriyama, K.: Theoretical properties of reversible flowchart programming languages, Master's thesis, University of Copenhagen (2009).

[2] Moriyama, K.: An Introduction to Reversible Programming Using Simple Reversible Flowchart Languages, Master's thesis, University of Copenhagen (2009).

[3] Tetsuo Yokoyama, Holger Bock Axelen, R.G.: Fundamentals of reversible flowchart languages, *Theoretical Computer Science*, Vol.611, pp.87–115 (2016).

[4] Tetsuo Yokoyama, Holger Bock Axelen, R.G.: Principles of a Reversible Programming Language, *Proc.Computing frontiers*, pp.43–54 (2008).