構造化可逆言語から非構造化可逆言語へのトランスレータの試作

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の計算は非可逆的に実行される．計算が可逆であるとは，その過程において，直前と直後の状態がたかだか一意に定まり，情報が消失しないことをいう．すなわち，非可逆的に計算が実行されるということは，その計算の前方，または後方に決定的ではないことをいう．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 [1]．そのため，非可逆な計算において情報が消失した場合，その情報分の熱エネルギーが周囲に放出される．しかし，可逆計算の場合，ランダウアーの原理に基づくエネルギー消費がなく，すなわち情報消失による発熱もない．それだけでなく可逆プログラミング言語はコンピュータサイエンスの様々な分野で需要がある．例えば，量子コンピューティング，双方向モデル変換，平均時間計算量といったプログラムプロパティの静的分析，プログラム逆変換といった複雑なプログラミング変換などの研究において可逆プログラミング言語が用いられている．

プログラミング言語は，高水準言語と低水準言語の大きく 2 つに分類することができ，可逆プログラミング言語においても同じことが言える．また，高水準言語から低水準言語へのトランスレートをする時，トランスレータの性質に正確性と効率性の 2 点が挙げられる [2]．正確性とは，翻訳が意味論を維持する性質であり，効率性は翻訳が計算複雑性を維持する性質である．ソースプログラムとターゲットプログラムは，リソースの使用量に関して同じ漸近的な複雑さを持っていることが望ましい．正確性と効率性は実用的な（非可逆）トランスレータが備えておくべき性質であり，可逆プログラミング言語が対象であるトランスレータにおいても備えておくべき性質である．

本研究の目的は大きく分けて 3 つある．第 1 の目的は SRL から RL への意味論と内包的クリーン性を維持する正確で効率的なトランスレータの実装を行うことである．埋込みや Bennett の一般解法 [3, 4] を用いてトランスレートした場合に得られる可逆プログラムは，内包的にクリーンではなく，実行ステップ数に比例するメモリを追加で使用してしまう．我々は翻訳結果のプログラムが必ず内包的なクリーン性をもつような入力プログラムの走査が 1 回のみですむようなトランスレータを実現する．可逆コンパイラ特有の効率化やそれを非可逆コンパイラでも用いられる効率化と組み合わせた効率化のアイデアを試すために，こうした単純な可逆プログラミング言語間の実際に動作するトランスレータの実装が有効に活用されることが期待される．

第 2 の目的は量子プログラミング言語や双方向変換のための言語など可逆性を有する新しいプログラミング言語を設計するためのガイドラインを示すことである．SRL および RL の構文および意味論の拡張を行うことで，単純な言語である SRL および RL の構造化および非構造化可逆プログラミング言語に関する理論やツールを活かすことが期待できる．

第 3 の目的は拡張された SRL から拡張された RL へのトランスレータへの拡張，ならびに SRL および RL のインタープリタの拡張が容易であることを示すことである．実装するトランスレータとインタープリタは，構造化可逆プログラミング言語および非構造化可逆プログラミング言語の両方を対象言語としている．我々はこれらの拡張を実際に行って，これらの拡張が容易になるような実装上の工夫を示す．

2　関連研究

2.1　可逆フローチャート

従来のフローチャートが，幅広く使われているプログラミング言語のモデルであるように，可逆フローチャートは，可逆プログラミング言語のモデルとして意図されている．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が，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．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，順方向および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．

図 1 で，可逆フロチャートの制御フローを示す．上 3 つが順方向の制御フローで，下 3 つがインバートされた制御フローである．B，B はブロックで，e は式である．インバートすると矢印やブロックがインバートされるだけでなく，テストとアサーションが入れ替わるようになっている．

2.2　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[1] という具体的な可逆プログラミング言語が存在する．

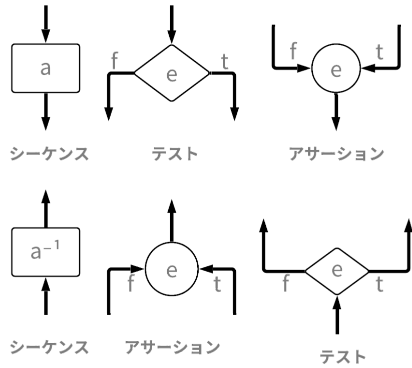


図 1　可逆制御フロー

```
p  ::=  b                                                    (SR プログラム)
b  ::=  a                                                    (ステップ操作)
     |  b  b                                                 (シーケンス)
     |  if e then b else b fi e                              (条件)
     |  from e do b loop b until ea                          (ループ)
     |  rif e b rfi e                                        (逆実行（拡張）)
a  ::=  x ⊕= e  |  x[e] ⊕= e  |  push x x  |  pop x x  |  skip
                                                             (ステップ操作)
     |  x ⇔ x  |  x[e] ⇔ x  |  x ⇔ x[e]  |  x[e] ⇔ x[e]
                                                             (ステップ操作（拡張）)
e  ::=  c  |  x  |  x[e]  |  e ⊗ e  |  top x  |  empty x     (式)
c  ::=  0  |  1  |  ···  |  4294967295                       (0 以上の整数定数)
⊗  ::=  ⊕  |  *  |  /  |  =  |  <  |  >  |  <=  |  >  |  !=  (演算子)
⊕  ::=  +  |  -  |  ^                                        (演算子)
```

図 2　SRL の構文 ([1]，[5] より)

一部の可逆プログラミング言語ではプログラム内で実行方向を変えることができ，それにより順方向計算と逆方向計算の両方のコードを共有することができる．しかし，SRL は実行方向を変える構文をもっていない．森山は SRL に Conditional reverse といった実行方向を変えられる構文を追加した [5]．

図 2 に SRL の構文規則, 図 3 に SRL の構文領域を示す．SRL プログラムはブロックであり，再帰的に構成されている．ブロックは，ステップ演算，シーケンス，if 文から成る条件，ループの 4 つからなるが，森山によって rif 文が追加された．更に，左辺と右辺を入れ替える 4 つのステップ操作も追加された．

```
SRL :  p ∈ SRL      b ∈ Blk
       a ∈ Step     e ∈ Exp     c ∈ Const     x ∈ Var     ⊕, ⊗ ∈ Op
```

図 3　SRL の構文領域 ([1] より)

図 4 は，SRL ブロックをそれぞれ可逆フローチャート図で表したものであり，全てのブロック全体をブロック b' とし，ステップ演算については，ステップ演算 a を実行する．シーケンスについては，ブロック $b1$, $b2$ の順に実行する．条件については，式 $e1$ が真ならば，$b2$ を，$e1$ が偽な

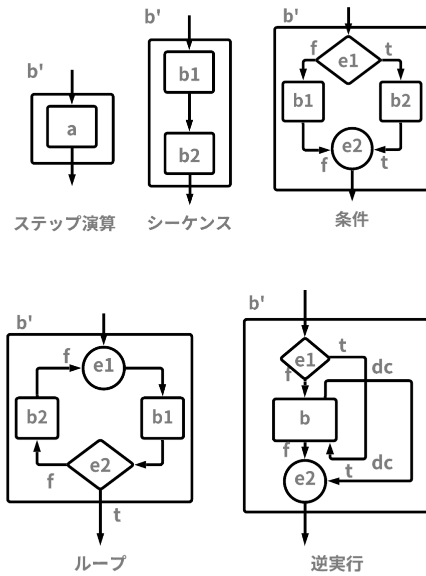


図 4　SRL ブロック

らば，$b1$ を実行する．式 $e2$ については，$b2$ を実行した後ならば真，$b1$ を実行した後ならば偽でなければならない．ループについて，式 $e1$ から，$b1$ を実行し，$e2$ が真ならば，実行を終了し，偽ならば $b2$ を実行する．この操作を $e2$ が真になるまで繰り返す．逆実行については，dc で順実行しているなら逆実行，逆実行しているなら順実行へと方向を変換する．

2.3　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[1] という具体的な可逆プログラミング言語が存在する．

更に，SRL と同様に，森山は RL に Reverse jump といった実行方向を変えられる構文を追加した [5]．

図 5 に RL の構文規則, 図 6 に SRL の構文領域を示す．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，RL プログラムには，1 つの entry と 1 つの exit が含まれている．さらに，森山によってアサーション k，ジャンプ j に，それぞれ rfrom , rgoto 文が追加された．

3　設計・実装

3.1　構文木，字句解析器・構文解析器・プリティプリンタ

本研究では，BNFC を用いて構文木，字句解析器・構文解析器・プリティプリンタを作成した．LBNF を記述する際，SRL と RL の 2 つのファイルに分割して行った．これは，SRL，RL において共通の構文である式は両方のファイルに記述しなければならないという手間があるが，プログラムの見やすさを考慮したものである．

q ::= d^{+} (RL プログラム)
d ::= l : k a^{*} j (RL ブロック)
k ::= from 文 l | fi e from l else l | entry (アサーション)
| rfrom l (rfrom 文（拡張）)
j ::= goto l | if e goto l else l | exit (ジャンプ)
| rgoto l (rgoto 文（拡張）)
a ::= x $\oplus$= e | x[e] $\oplus$= e | push x x | pop x x | skip (ステップ操作)
| $x \Leftrightarrow x$ | x[e] $\Leftrightarrow x$ | $x \Leftrightarrow x$[e] | x[e] $\Leftrightarrow x$[e] (ステップ操作（拡張）)
e ::= c | x | x[e] | $e \otimes e$ | top x | empty x (式)
c ::= 0 | 1 | $\cdots$ | 4294967295 (0 以上の整数定数)
$\otimes$ ::= $\oplus$ | * | / | = | < | > | <= | > | != (演算子)
$\oplus$ ::= + | - | ^ (演算子)

図 5 RL の構文 ([1], [5] より)

RL : q ∈ RL d ∈ RLblk j ∈ Jump k ∈ From l ∈ Label
a ∈ Step e ∈ Exp c ∈ Const x ∈ Var $\oplus, \otimes$ ∈ Op

図 6 RL の構文領域 ([1] より)

$\mathcal{T}_{\mathrm{SRL}}[\![b]\!]$ =
l_0 : entry
goto l_1
$\mathcal{T}[\![b]\!]$ (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

$\mathcal{T}[\![b_1\ b_2]\!]$ (l_0, l_1, l_4, l_5) =
$\mathcal{T}[\![b_1]\!]$ (l_0, l_1, l_2, l_3)
$\mathcal{T}[\![b_2]\!]$ (l_2, l_3, l_4, l_5)
where l_2 , l_3 are fresh

$\mathcal{T}[\![a]\!]$ (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

$\mathcal{T}[\![$if e_1 then b_1 else b_2 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
$\mathcal{T}[\![b_1]\!]$ (l_1, l_2, l_3, l_6)
$\mathcal{T}[\![b_2]\!]$ (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

$\mathcal{T}[\![$from e_1 do b_1 loop b_2 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
$\mathcal{T}[\![b_1]\!]$ (l_1, l_2, l_3, l_4)
$\mathcal{T}[\![b_2]\!]$ (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 7 SRL から RL への翻訳規則 ([1] より)

3.2 トランスレータ

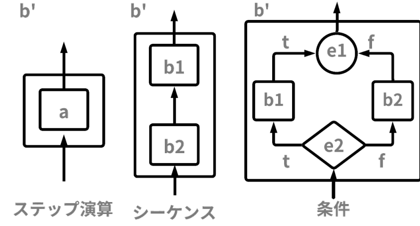
トランスレータは，まず初めに Ident，Exp，Step の型変換を行っている．次に SRL のブロックを受け取った文によって適切な RL のブロックにトランスレートする．最後に entry 文，body，exit 文を連結させて SRL から RL へのトランスレートが完了する．SRL のブロック b が 2 つで構成されている場合，2 つの新しいラベルを用いて，個別にトランスレートされる．ステップ演算である a は，l_0 から受け取るブロック l_1 と，制御を l_3 に渡すブロック l_2 にトランスレートされる．条件文と，ループ文には，互いに類似しているが，b_1 と b_2 に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，$\mathcal{T}$ への再帰的な呼び出しによって個別にトランスレートされる

また，拡張部分の `rif` 文について，SRL から RL への翻訳規則が [5]，[6] に定義されていないため，今回新たに拡張した．図 9（左）について，l_4, l_5 が fresh される．l_2 において，`rgoto` によって実行方向が逆になり，l_3 へと移動し，$\mathcal{T}[\![b]\!](l_2, l_3, l_4, l_5)$ を実行する．l_5 において，`rfrom` によって実行方向が逆になり，l_6 へと移動する．図 9(右）について，l_2, l_3, l_4, l_5 が fresh される．l_1 において，e1 が真ならば l_2，すなわち $\mathcal{T}[\![b]\!](l_1, l_2, l_3, l_6)$ へ，偽ならば l_4，すなわち $\mathcal{T}[\![b]\!](l_1, l_4, l_5, l_6)$ へ移動する．l_6 において，e1 が真ならば，e2 も真であり，e1 が偽ならば，e2 も偽である．

$\mathcal{T}_{bwd}[\![b]\!]$ (l_1, l_2, l_3, l_6) =
l_2 : from l_1 rgoto l_3
$\mathcal{T}[\![b]\!]$ (l_2, l_3, l_4, l_5)
l_5 : rfrom l_4 goto l_6
where l_4, l_5 fresh

$\mathcal{T}[\![$rif e_1 b 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
$\mathcal{T}_{bwd}[\![b]\!]$ (l_1, l_2, l_3, l_6)
$\mathcal{T}[\![b]\!]$ (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 fresh

図 8 SRL から RL への翻訳規則 (拡張部分)



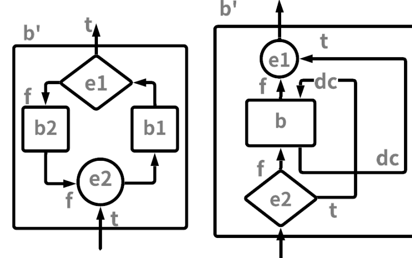


図 9 インバータ

3.3 インバータ

構文解析された SRL の「＋と－」，「`pop` と `push`」をそれぞれインバートすることで SRL インバータは作成される．図 9 は，SRL Inverter のブロックをそれぞれ可逆フローチャート図で表したものであり，それぞれ全体をブロック b とする. 次に RL の場合「`goto` と `from`」,「`exit` と `entry`」,「`rfrom` と `rgoto`」それぞれインバートすることで RL インバータは作成される．実装においては，SRL Inverter をトランスレータすることで実装した.

3.4 インタープリタ

インタープリタは式の評価，ステップ演算の評価，SRL の操作的意味論，RL の操作的意味論を定義する．式の評価では主にストアの値を出力，四則演算，関係演算子を定義する．ステップ演算の評価では主にストアの値やストアの配列に対する四則演算，`skip`，`pop`，`push` の定義をする．SRL の操作的意味論では主に `if` 文，`loop` 文を定義する．RL の操作的意味論では主に `jump` 文，`from` 文を定義する．更に，森山の論文 [5] に示されている意味論の一部，RIFTRUE,BLOCK1,BLOCK2 のを改良して実装した．その意味論を図 10 に示す．

$$\frac{\sigma \vdash_{expr} e_1 \Rightarrow v_1 \quad v_1 \neq 0 \qquad \sigma \vdash_{block} \mathcal{T}_{SRL}[\![b]\!] \Rightarrow \sigma' \qquad \sigma' \vdash_{expr} e_2 \not\Rightarrow v_2 \quad v_2 \neq 0}{\sigma \vdash_{block} \texttt{rif}\ e_1\ b\ \texttt{rfi}\ e_2 \Rightarrow \sigma'}\ \text{RIFTRUE}$$

$$\frac{\sigma \vdash_{from} k \Rightarrow (l',\ \overline{x}) \qquad \sigma \vdash_{assigns} al \Rightarrow \sigma \qquad \sigma \vdash_{jump} j \Rightarrow (l_0,\ \overline{y})}{\sigma \vdash_{block} (l_0,\ l:\ k\ al\ j,\ (x,b)) \Rightarrow (\sigma',\ l',\ (b,y))}\ \text{BLOCK2}$$

$$\frac{\sigma \vdash_{from} k \Rightarrow (l_0,\ x) \qquad \sigma \vdash_{assigns} al \Rightarrow \sigma' \qquad \sigma' \vdash_{jump} j \Rightarrow (l',\ y)}{\sigma \vdash_{block} (l_0,\ l:\ k\ al\ j,\ (x,f)) \Rightarrow (\sigma',\ l',\ (f,y))}\ \text{BLOCK1}$$

図 10 改良した SRL,RL の操作的意味論（拡張部分）([5] より)

$$\begin{array}{rcll}
\sigma & \in \texttt{Store} & = & \texttt{Lvalue} \rightharpoonup \texttt{Value} \\
u & \in \texttt{Lvalue} & = & \texttt{Var} \cup \{x[v] \mid x \in \texttt{Var},\ v \in \texttt{Value}\} \\
v & \in \texttt{Value} & = & \mathbb{Z}_{32} \cup \texttt{Svalue} \\
s & \in \texttt{Svalue} & = & \{\texttt{nil}\} \cup \{v :: s \mid v \in \mathbb{Z}_{32},\ s \in \texttt{Svalue}\} \\
l & \in \texttt{Lavel'} & = & \texttt{Label} \cup \{\texttt{entry},\ \texttt{exit}\} \\
x,y & \in \texttt{Direction} & = & \{\texttt{forward},\ \texttt{backward}\}
\end{array}$$

図 11 意味論の値 ([1] より)

図 10 の改良部分について説明する．σ から σ' への変換については，構文領域 x のプログラム b の実行よって行われ，$\sigma \vdash_x b \Rightarrow \sigma'$ と表される．RIFTRUE は，SRL 文であり，rif 文の true 分岐で，真ならばブロックを逆方向実行する．偽ならば，([5] より) に示されている式 RIFFALSE にしたがって，ブロックを順方向に実行する．$\mathcal{T}_{SRL}[\![b]\!]$ は SRL 言語である b をインバートすることを表す．BLOCK1,2 については，RL 文であり，あるブロックから別のブロックへの遷移を表す．x, y は前方，または後方を示す変数で，$\overline{x}$, $\overline{y}$ は x, y とはそれぞれ逆方向を示す．f, b はそれぞれ前方（forward），後方（backword）を示す．(x,f) とは，方向が x から f に変換されることを示す．(f,y), (x,b), (b,y) についても同様である．

図 12 は，RIFTRUE を可逆フローチャート図で表したものと，BLOCK の Jump の仕組みを図で表したものである．RIFTRUE について，e1 が真の時，b をインバートする．e1 が偽の時はインバートせず b の評価をする．normal jump については，別のラベルへ向かう際に方向転換をせず，reverse jump については，別のラベルへ向かう際に方向転換を行う．

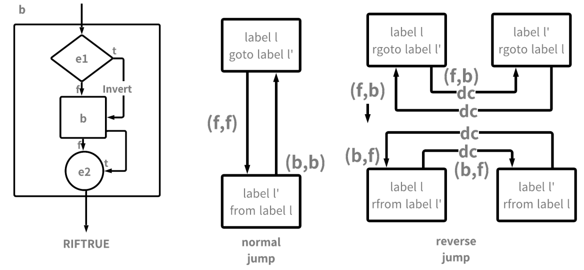


図 12 改良した SRL,RL の操作的意味論（拡張部分）

4 おわりに

本研究では，SRL から RL へのトランスレータを実装した．さらに両言語のインタープリタを実装し，トランスレータおよびインタープリタの機能拡張を行った．外延的のみならず内包的にクリーンなトランスレータを実装することができ，それにより 1 回の走査で SRL プログラムを RL プログラムにトランスレートすることが可能となった．機能拡張を容易にするために翻訳規則の作成や，インバータを使用して実装などの実装上の工夫を示すこともできた．

参考文献

[1] Yokoyama, T., Axelen, H.B. and Glück, R.: Fundamentals of reversible flowchart languages, *Theoretical Computer Science*, Vol.611, pp.87–115 (2016).

[2] Axelsen, H.B.: Clean Translation of an Imperative Reversible Programming Language, *Proc. Compiler Construction* (Knoop, J., Ed.), Lecture Notes in Computer Science, Vol.6601, pp.144–163 (2011).

[3] Landauer, R.: Irreversibility and Heat Generation in the Computing Process, *IBM Journal of Research and Development*, Vol.5, No.3, pp.183–191 (1961).

[4] Bennett, C.H.: Logical Reversibility of Computation, *IBM Journal of Research and Development*, Vol.17, No.6, pp.525–532 (1973).

[5] Moriyama, K.: Theoretical properties of reversible flowchart programming languages (2009). Term report.

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