可逆言語の拡張とその可逆性及び正当性の証明

M2021SE008 水野 幹大
指導教員：横山哲郎

1 はじめに

可逆計算では，どの状態でも直前と直後の状態がたかだか一意に定まり，情報が消失せず，したがって情報損失による熱が発生しない．ランダウアーの原理によれば，情報が消失されるとき，熱が発生し，エネルギーを必要とする [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 [2]．

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という．可逆言語で書かれたプログラムは全て可逆性が保証される [3]．多くの可逆言語において可逆プログラムは意味の逆を解釈できる．一方，現在主流の電子計算機では通常は非可逆な計算がされており，プログラムの逆解釈はできない．ひとつの理由は，ビットはゼロクリアされると情報損失し，それ以前の値を知ることができないからである [4]．

可逆言語は計算機科学の様々な分野で需要がある．例えば，デバッグ [5]，プログラム逆変換といった複雑なプログラミング変換，及び従来の古典的なコンピュータに比べて現在の課題を高速で解決する可能性をもつ量子計算などで用いられている．

可逆流れ図言語は，単純でありながらプログラムの構造や制御フローを自然に記述できるプログラミング言語理論の基礎を支える言語のひとつである．可逆流れ図言語としてゴミ出力なしで可逆的な解釈ができるクリーン性をもつ高水準言語 SRL 及び低水準言語 RL が知られており，これらの言語を用いて可逆プログラミング言語理論の新しい知見が得られてきた。しかし，これらの言語は記述力が弱く，プログラムの再利用が容易ではない．また，非可逆言語で広く使われている言語機構の自然な導入の可否，構造化演算子における逆意味の適切な扱い方，翻訳や解釈の最適化手法に未知な部分がある．これらを整備していくために厳密に形式化された意味論を備えた言語の拡張が望まれる．

文献 [6] において逆解釈を選択できる構造化演算子及び実行方向を変更するジャンプが SRL と RL にそれぞれ導入された．これらの拡張により可逆流れ図言語の記述力が増し，モジュール性が向上した．しかし，操作的意味論や拡張 SRL から拡張 RL への翻訳規則を定めることが不十分であり，かつ可逆言語が持つべき性質である可逆性並びに逆変換及び翻訳の正当性は未証明である．

本研究では，第一に，SRL と RL について，可逆プログラミング言語がもつべき性質である可逆性並びに逆変換及び翻訳の正当性を示す．第二に，拡張 SRL と拡張 RL の意味規則や翻訳規則を完全に整備し，拡張 SRL と拡張 RL が可逆性をもち，それらの逆変換器と翻訳機が正当性をもつことを示す．

2 関連研究

効率的な翻訳系や解釈系の構築には厳密に形式化された意味論が必要である．可逆言語の基礎的な理論整備の一環として，可逆言語 Janus の操作的意味論と表示的意味論の等価性 [7] 及び操作的意味論を用いた可逆化解釈系の形式化 [8] が示されている．さらに，性質を形式的に記述して定理証明支援系での証明がなされることがある [7]．可逆言語の分析や最適化のための基礎を得るため，正当性が保たれることは厳密に示されてはいないものの，Janus から可逆な静的単一代入（RSSA）形式への可逆的翻訳系が Kutrib らによって初めて実現された [9]．可逆言語に新たな言語要素が追加された場合にも，その形式化をして基本的な性質が保たれることを厳密に示すことは重要である．

3 準備

3.1 逆解釈と逆変換

解釈，逆解釈，変換，逆変換の関係を図 1 に示す．あるプログラム p と値 x から解釈によって値 y が得られるとき，そのプログラム p と値 y から値 x を得ることを p の逆解釈とよぶ．また，あるプログラム p から変換器によって p' が生成され，そのプログラムと値 x から値 y が得られるとき，値 y から値 x が得られるプログラム p^{-1} をプログラム p から生成することを逆変換とよぶ．

3.2 SRL と RL

SRL では，構造化されたシーケンス，条件，ループのみの制御の流れを用いる．SRL の構文規則を図 2 に示す．SRL プログラムは再帰的に構文が定められている SRL ブロック b からなり，SRL ブロック b はステップ操作，シーケンス，条件，ループからなる．

ステップ操作 a は，状態を更新する可逆的な代入，スタック操作などからなる．式 e は，0 以上の整数定数，変

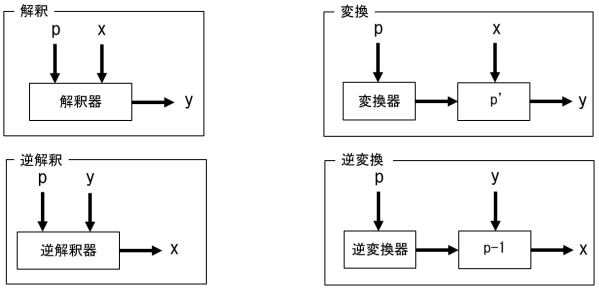

図 1 解釈，逆解釈，変換，逆変換の関係

$$
\begin{array}{lll}
p & ::= & b \quad \text{(SRL プログラム)} \\
b & ::= & a \quad \text{(ステップ操作)} \\
 & | & b\ b \quad \text{(シーケンス)} \\
 & | & \texttt{if}\ e\ \texttt{then}\ b\ \texttt{else}\ b\ \texttt{fi}\ e \quad \text{(条件)} \\
 & | & \texttt{from}\ e\ \texttt{do}\ b\ \texttt{loop}\ b\ \texttt{until}\ e \quad \text{(ループ)}
\end{array}
$$

図 2　SRL の構文 [3]

$$
\begin{array}{lll}
q & ::= & d^+ \quad \text{(RL プログラム)} \\
d & ::= & l\ :\ k\ a^*\ j \quad \text{(RL ブロック)} \\
k & ::= & \texttt{from}\ l \mid \texttt{fi}\ e\ \texttt{from}\ l\ \texttt{else}\ l \mid \texttt{entry} \quad \text{(アサーション)} \\
j & ::= & \texttt{goto}\ l \mid \texttt{if}\ e\ \texttt{goto}\ l\ \texttt{else}\ l \mid \texttt{exit} \quad \text{(ジャンプ)}
\end{array}
$$

図 3　RL の構文 [3]

数，添字式付き配列変数，二項演算式などからなる．

RL では任意の RL ブロックから任意の RL ブロックへ制御は流れることができる．RL の構文規則を図 3 に示す．RL プログラム q は，1 個以上の RL ブロック d の並びからなる．RL ブロック d は，一意のラベル l，アサーション k，0 個以上のステップ操作 a^* の並び，ジャンプ j からなる．アサーション k は，l からジャンプしてきたことを示す `from` l，式 e が真ならば，l_1 から，偽ならば l_2 からジャンプしてきたことを示す `fi` e `from` l_1 `else` l_2，プログラムの入口である `entry` からなる．ジャンプ j は，どこのラベルにジャンプするかを示す `goto` l，式 e が真ならば，l_1 に，偽ならば l_2 にジャンプすることを示す `if` e `goto` l_1 `else` l_2，プログラムの出口である `exit` からなる．

ステップ操作 a と式 e は SRL と共通である．

SRL と RL の意味論は文献 [3] に定められたものを使う．

我々は，可逆言語が持つべき性質である，可逆性，逆変換及び翻訳の正当性を SRL と RL がもつことを確認した．この結果は，5 章で得られた結果に含まれるので本稿では詳細を省略する．

3.3　拡張 SRL と拡張 RL

文献 [6] では，SRL ブロック b に構造化された `rif` 文が，RL のアサーション k に `rfrom` 命令，ジャンプ j に `rgoto` 命令が拡張された:

$$
\begin{array}{lll}
b & ::= & \cdots \mid \texttt{rif}\ e\ b\ \texttt{rfi}\ e \\
k & ::= & \cdots \mid \texttt{rfrom}\ l \\
j & ::= & \cdots \mid \texttt{rgoto}\ l
\end{array}
$$

この拡張により，ユーザにとって陽に逆意味論へのアクセスが可能になり，順方向と逆方向の意味をもつコード片によるモジュラリティ・可読性の向上とそうしたモジュールの正当性の確認が容易になることが確認された．例えば，こうした非伝統的コード共有による Bennett 法によって可逆 Turing 機械の記述がより簡潔になった [6].

拡張 SRL の文 `rif` e_1 `then` b `rfi` e_2 における制御フローを図 4 に示す．式 e_1 が真ならば t のラベルの付いた（赤の）矢印に，偽ならば f のラベルの付いた（青の）矢印に沿って制御が流れる．更に，式 e_1 が真のときは式 e_2 も必ず真，式 e_1 が偽のときは式 e_2 も必ず偽でなければならない．ここで，SRL ブロック b に関して，式 e_1 が真の場合は b の意味の逆解釈を，偽の場合は解釈をそれぞれ行う．

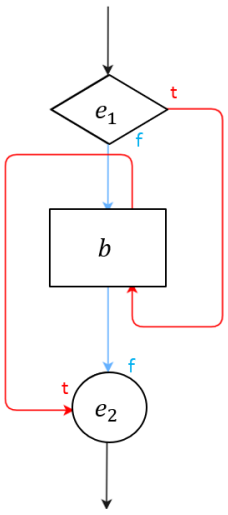

図 4　拡張 SRL の条件の制御フロー

拡張 RL のジャンプにおける，全 8 通りの制御フローを図 5 に示す．実行方向を実行中に変更しない場合が，順方向からの実行，逆方向からの実行，また，それらの逆実行を含めて 4 通りある．また，実行方向を実行中に変更する場合には，`rfrom` 命令，`rgoto` 命令にそれぞれ順方向から逆方向へ，逆方向から順方向へ，また，それらの逆方向の実行の 4 通りがある．文献 [6] では拡張 RL の意味規則の整備や拡張 SRL と拡張 RL の性質の証明が不十分であった．

4　拡張 SRL と拡張 RL の意味論と翻訳規則

本章では我々の整備した RL の意味論や翻訳規則を示す．

意味論の値　実行方向 dir が順方向と逆方向であることをそれぞれ表す `fwd` と `bwd` を意味論の値に追加した．

拡張 RL の操作的意味論　`from` 命令 k, `jump` 命令 j を状態 σ において実行し，それぞれ得られたラベルを l，実行方向を dir とする判断を次のように定める:

$$
\begin{array}{l}
\sigma \vdash_{from}\ k \Rightarrow (l, dir) \\
\sigma \vdash_{jump}\ j \Rightarrow (l, dir)
\end{array}
$$

状態が σ のとき，実行方向が dir で，l_1 から l_2 へ制御

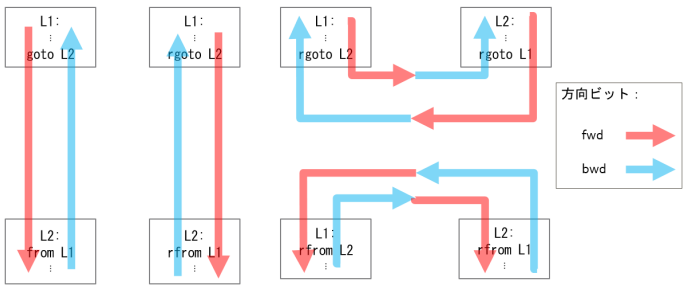

図 5　拡張 RL のジャンプの制御フロー（[6] の図を改変）

$$\frac{\Gamma_q(l_2) = k\ al\ j \quad \sigma \vdash_{steps} al \Rightarrow \sigma' \quad \sigma \vdash_{from} k \Rightarrow (l_1, \texttt{fwd}) \quad \sigma' \vdash_{jump} j \Rightarrow (l_3, \texttt{fwd})}{(\sigma, (l_1, l_2), \texttt{fwd}) \Rightarrow_q (\sigma', (l_2, l_3), \texttt{fwd})}\ \text{FF1}$$

$$\frac{\Gamma_q(l_2) = k\ al\ j \quad \sigma \vdash_{steps} al \Rightarrow \sigma' \quad \sigma \vdash_{from} k \Rightarrow (l_1, \texttt{fwd}) \quad \sigma' \vdash_{jump} j \Rightarrow (l_3, \texttt{fwd})}{(\sigma', (l_3, l_2), \texttt{bwd}) \Rightarrow_q (\sigma, (l_2, l_1), \texttt{bwd})}\ \text{FF2}$$

$$\frac{\Gamma_q(l_2) = k\ al\ j \quad \sigma \vdash_{steps} al \Rightarrow \sigma' \quad \sigma \vdash_{from} k \Rightarrow (l_1, \texttt{fwd}) \quad \sigma' \vdash_{jump} j \Rightarrow (l_3, \texttt{bwd})}{(\sigma, (l_1, l_2), \texttt{fwd}) \Rightarrow_q (\sigma', (l_2, l_3), \texttt{bwd})}\ \text{FB1}$$

$$\frac{\Gamma_q(l_2) = k\ al\ j \quad \sigma \vdash_{steps} al \Rightarrow \sigma' \quad \sigma \vdash_{from} k \Rightarrow (l_1, \texttt{fwd}) \quad \sigma' \vdash_{jump} j \Rightarrow (l_3, \texttt{bwd})}{(\sigma', (l_3, l_2), \texttt{bwd}) \Rightarrow_q (\sigma, (l_2, l_1), \texttt{fwd})}\ \text{FB2}$$

$$\frac{\Gamma_q(l_2) = k\ al\ j \quad \sigma' \vdash_{steps} al \Rightarrow \sigma \quad \sigma' \vdash_{from} k \Rightarrow (l_1, \texttt{bwd}) \quad \sigma \vdash_{jump} j \Rightarrow (l_3, \texttt{fwd})}{(\sigma, (l_1, l_2), \texttt{bwd}) \Rightarrow_q (\sigma', (l_2, l_3), \texttt{fwd})}\ \text{BF1}$$

$$\frac{\Gamma_q(l_2) = k\ al\ j \quad \sigma' \vdash_{steps} al \Rightarrow \sigma \quad \sigma' \vdash_{from} k \Rightarrow (l_1, \texttt{bwd}) \quad \sigma \vdash_{jump} j \Rightarrow (l_3, \texttt{fwd})}{(\sigma', (l_3, l_2), \texttt{fwd}) \Rightarrow_q (\sigma, (l_2, l_1), \texttt{bwd})}\ \text{BF2}$$

$$\frac{\Gamma_q(l_2) = k\ al\ j \quad \sigma' \vdash_{steps} al \Rightarrow \sigma \quad \sigma' \vdash_{from} k \Rightarrow (l_1, \texttt{bwd}) \quad \sigma \vdash_{jump} j \Rightarrow (l_3, \texttt{bwd})}{(\sigma, (l_1, l_2), \texttt{bwd}) \Rightarrow_q (\sigma', (l_2, l_3), \texttt{bwd})}\ \text{BB1}$$

$$\frac{\Gamma_q(l_2) = k\ al\ j \quad \sigma' \vdash_{steps} al \Rightarrow \sigma \quad \sigma' \vdash_{from} k \Rightarrow (l_1, \texttt{bwd}) \quad \sigma \vdash_{jump} j \Rightarrow (l_3, \texttt{bwd})}{(\sigma', (l_3, l_2), \texttt{fwd}) \Rightarrow_q (\sigma, (l_2, l_1), \texttt{fwd})}\ \text{BB2}$$

図 6 拡張 RL のジャンプに関する計算状況の更新

が流れる計算状況を $(\sigma, (l_1, l_2), dir)$ と表し，その更新を次のように表す:

$$(\sigma, (l_1, l_2), dir) \Rightarrow_q (\sigma', (l_2, l_3), dir')$$

文献 [3] の計算状況の更新において，更新の前後に同一の dir を追加することで，ジャンプに関する計算状況の更新を除いた全ての意味規則が得られる．図 6 にジャンプに関する計算状況の更新についての全ての意味規則を示す．各意味規則は図 5 のそれぞれの制御フローを表す．これらの意味規則からなる意味論を拡張 RL の意味論とする．

拡張 SRL から拡張 RL への翻訳　SRL から RL への翻訳器 [3] に拡張を行った．図 7 に拡張部分を示す．この翻訳規則の結果は，l_2, l_3, l_4, l_5 を異なるラベルとして，l_1 は，l_0 から遷移され，式 e_1 が真ならば l_2，偽ならば l_4 へ遷移することを表す．l_6 は式 e_2 が真ならば l_3，偽ならば l_5 から遷移され，l_7 へ遷移することを表す．SRL ブロック b の逆変換 $\mathcal{I}_{blk}[\![b]\!]$ を用いて，b の意味の逆を実現した．

$$\begin{aligned}
&\mathcal{T}[\![\texttt{rif}\ e_1\ b\ \texttt{fi}\ e_2]\!]\ (l_0, l_1, l_6, l_7) = \\
&\quad l_1\ : \texttt{from}\ l_0 \\
&\qquad\quad \texttt{if}\ e_1\ \texttt{goto}\ l_2\ \texttt{else}\ l_4 \\
&\quad \mathcal{T}[\![\mathcal{I}_{blk}[\![b]\!]]\!]\ (l_1, l_2, l_3, l_6) \\
&\quad \mathcal{T}[\![b]\!]\ (l_1, l_4, l_5, l_6) \\
&\quad l_6\ : \texttt{fi}\ e_2\ \texttt{from}\ l_3\ \texttt{else}\ l_5 \\
&\qquad\quad \texttt{goto}\ l_7 \\
&\quad \texttt{where}\ l_2,\ l_3,\ l_4,\ l_5\ \texttt{fresh}
\end{aligned}$$

図 7 拡張 SRL から拡張 RL への翻訳規則の拡張部分

5 拡張言語の可逆性と逆変換・翻訳の正当性

可逆言語が持つべき性質である，可逆性，逆変換及び翻訳の正当性を拡張 SRL と拡張 RL がもつことを示す．以降では，状態 σ を状態 σ' に更新する x に関する構文要素 t についての判断を $\sigma \vdash_x t \Rightarrow \sigma'$ で表す．

可逆性　拡張 SRL，拡張 RL 共に含まれるステップ操作 a，SRL ブロック b，SRL プログラム p，RL ブロック d，RL プログラム q について，可逆性（前方決定性と後方決定性）をもつことを示した．また，式 e については前方決定性をもつことのみ証明を行った．基本的には前方決定性と後方決定性を分けて示した．

補題 1 [ステップ操作の可逆性]

$$\sigma \vdash_{step} a \Rightarrow \sigma' \wedge \sigma \vdash_{step} a \Rightarrow \sigma'' \Longrightarrow \sigma' = \sigma''$$
$$\sigma' \vdash_{step} a \Rightarrow \sigma \wedge \sigma'' \vdash_{step} a \Rightarrow \sigma \Longrightarrow \sigma' = \sigma''$$

補題 2 [拡張 SRL プログラムの実行の可逆性]

$$\sigma \vdash_{blk} b \Rightarrow \sigma' \wedge \sigma \vdash_{blk} b \Rightarrow \sigma'' \Longrightarrow \sigma' = \sigma''$$
$$\sigma' \vdash_{blk} b \Rightarrow \sigma \wedge \sigma'' \vdash_{blk} b \Rightarrow \sigma \Longrightarrow \sigma' = \sigma''$$

補題 3 [拡張 RL プログラムの実行の可逆性]

$$\sigma \vdash_{RL} q \Rightarrow \sigma' \wedge \sigma \vdash_{RL} q \Rightarrow \sigma'' \Longrightarrow \sigma' = \sigma''$$
$$\sigma' \vdash_{RL} q \Rightarrow \sigma \wedge \sigma'' \vdash_{RL} q \Rightarrow \sigma \Longrightarrow \sigma' = \sigma''$$

ステップ操作 a については，導出木

$$\frac{\begin{array}{c}\vdots\\ \sigma' \vdash_{step} \texttt{push}\ x_1\ x_2 \Rightarrow \sigma\end{array}}{\sigma \vdash_{step} \texttt{pop}\ x_1\ x_2 \Rightarrow \sigma'}$$

の根で σ から σ' が得られる判断をもつのに対して，根の前提では σ' から σ が得られる判断をもつ．したがって，前方決定性の証明に後方決定性が必要である．同様に，後方決定性の証明に前方決定性が必要である．我々は前方決定性と後方決定性を同時に示すようにして両者を帰納法の仮定に用いた．

SRL ブロック b についても，導出木

$$\frac{\cdots\ \begin{array}{c}\vdots\\ \sigma' \vdash_{blk} b \Rightarrow \sigma\end{array}\ \cdots}{\sigma \vdash_{blk} \texttt{rif}\ e_1\ b\ \texttt{rfi}\ e_2 \Rightarrow \sigma'}$$

に同様の構造があるので，前方決定性と後方決定性を同時に示した．

逆変換の正当性　ステップ操作 a，SRL ブロック b，SRL プログラム p，RL ブロック d，RL プログラム q について，逆変換の正当性をもつことを示した．

補題 4 [ステップ操作の逆変換の実行の正当性]

$$\sigma \vdash_{step} a \Rightarrow \sigma' \Longleftrightarrow \sigma' \vdash_{step} \mathcal{I}_{step}[\![a]\!] \Rightarrow \sigma$$

補題 5 [拡張 SRL プログラムの逆変換の実行の正当性]

$$\sigma \vdash_{blk} b \Rightarrow \sigma' \Longleftrightarrow \sigma' \vdash_{blk} \mathcal{I}_{blk}[\![b]\!] \Rightarrow \sigma$$

補題 6 [拡張 RL プログラムの逆変換の実行の正当性]

$$\sigma \vdash_{rlblk} d \Rightarrow \sigma' \Longrightarrow \sigma' \vdash_{rlblk} \mathcal{I}_{rlblk}[\![d]\!] \Rightarrow \sigma$$
$$\sigma \vdash_{RL} q \Rightarrow \sigma' \Longrightarrow \sigma' \vdash_{RL} \mathcal{I}_{RL}[\![q]\!] \Rightarrow \sigma$$

ここで，$\mathcal{I}_x$ は構文 x についての逆変換器である [3]．

ステップ操作 a については構造帰納法を用いた．SRL ブロック b については導出木の高さに関する帰納法を用いた．これは導出木

$$\dfrac{\cdots \quad \dfrac{\vdots}{\sigma'' \vdash_{loop} (e_1, b_1, b_2, e_2) \Rightarrow \sigma'''}}{\sigma \vdash_{loop} (e_1, b_1, b_2, e_2) \Rightarrow \sigma'''}$$

の根とその前提に同一の構造 (e_1, b_1, b_2, e_2) が現れており合成的ではないので構造帰納法では直接的に証明できなかったからである．SRL プログラム p については SRL ブロックが逆変換の正当性をもつことを用いた．RL ブロック d については自明であり，RL プログラム q については RL ブロック d の逆変換の正当性を用いた．

翻訳の正当性　SRL ブロック b から RL ブロック d 及び SRL プログラム p から RL プログラム q への翻訳の正当性をもつことを証明した．

SRL ブロック b から RL ブロック d への翻訳については，SRL ブロックの逆変換の正当性を示した場合と同様に合成的ではない部分があったので，導出木の高さに関する帰納法を用いて示した．

補題 7 [$\mathcal{T}_{SRL}$ の正当性]

$$\sigma \vdash_{SRL} p \Rightarrow \sigma' \implies \sigma \vdash_{RL} \mathcal{T}_{SRL}[\![p]\!] \Rightarrow \sigma'$$

6　おわりに

本研究では，SRL と RL について，可逆言語がもつべき性質である可逆性や，逆変換及び翻訳の正当性を保持することを示した．また，証明法として，構造帰納法や導出木の高さに関する帰納法といった，SRL，RL に新たに拡張した際，今まで証明された性質が損なわれないことを証明する方法を精査した．基本的には構造帰納法を用いて証明を行ったが，操作的意味論で前提と結論に同様の式が存在し，構造帰納法を用いて直接的に証明することができないものに対して導出木の高さに関する帰納法を用いたり，特に可逆性について，前方決定性と後方決定性を別々に構造帰納法を用いて直接的に証明できないものに対して，前方決定性と後方決定性を同時に仮定したりして，直接的に証明できる工夫を施した．

次に，拡張 SRL と拡張 RL についての操作的意味論や翻訳規則の整備を行った．また，拡張 SRL と拡張 RL やそれらの逆変換や翻訳についても拡張前の SRL と RL のときと同様の工夫によって可逆性や正当性を持つことを示した．これらは，逆呼び出し文をもつ Janus や可逆ジャンプ命令をもつ PISA の意味論の整備においても活用されることが期待される．

今後の課題は定理証明支援系を用いての証明である．

参考文献

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

[2] 森田憲一：可逆計算，近代科学社 (2012).

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

[4] Krakovsky, M.: Taking the Heat, *Commun. ACM*, Vol.64, No.6, p.18–20 (2021).

[5] Ikeda, T. and Yuen, S.: A Reversible Debugger for Imperative Parallel Programs with Contracts, *Reversible Computation. Proceedings*, LNCS, Vol.13354, Springer, pp.204–212 (2022).

[6] Moriyama, K.: An Introduction to Reversible Programming Using Simple Reversible Flowchart Languages (2009).

[7] Paolini, L., Piccolo, M. and Roversi, L.: A Certified Study of a Reversible Programming Language, *21st International Conference on Types for Proofs and Programs (TYPES 2015)* (Uustalu, T., Ed.), Leibniz International Proceedings in Informatics (LIPIcs), Vol.69, Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, Germany, pp.7:1–7:21 (2018).

[8] Fernández, M. and Mackie, I.: A reversible operational semantics for imperative programming languages, *Formal Methods and Software Engineering. Proceedings* (Lin, S., Hou, Z. and Mahony, B.P., Eds.), LNCS, Vol.12531, Springer, pp.91–106 (2020).

[9] Kutrib, M., Meyer, U., Deworetzki, N. and Schuster, M.: Compiling Janus to RSSA, *Reversible Computation. Proceedings* (Yamashita, S. and Yokoyama, T., Eds.), LNCS, Vol.12805, Springer, pp.64–78 (2021).