2020/07/17 18:07:48

比較結果

古いファイル :

200717youshi.pdf

13 ページ (290 KB)

VS

新しいファイル :

200619youshi.pdf

7 ページ (220 KB)

変更の合計

210

テキストのみの比較

内容

72 件の置換

78 件の挿入

60 件の削除

スタイルと注釈

0 件のスタイル

0 個の注釈

最初の変更に移動 (1 ページ)

可逆プログラミング言語 ROOPL のインタープリタの実装

2017SE004 青柳 裕樹　2017SE057 新美 伊織　2017SE086 竹市 翔哉
指導教員：横山 哲郎

1　はじめに

可逆計算は反転可能な 2 方向性計算モデルである．つまり，計算過程においてどの状態からもその直前と直後の状態が一意に決まる．計算は一方向に進み，その中で，前の状態が一意に決まり，その状態へ到達できるように情報損失を避けながら計算が進む．したがって，可逆計算は，計算過程において完璧に情報を保つことができるため，熱放散の問題の有効な解決策として，考えられている．

これまで，マイクロプロセッサの使用電力の最小化に対して多くの取り組みがされてきたが，現代の計算モデルでは理論上の限界が存在する．ランダウアーの原理では，次のように主張されている．ビットの消去や 2 つの計算経路の結合など論理的に非可逆な操作には，情報処理機器やその環境において，必ずエントロピーの増加が伴う [1]．つまり，システム上での情報の消去は常にエネルギー消費の増加につながるということである．また，n ビットの情報を消去するために必要なエネルギーの量は，$n \cdot K_B \cdot T \cdot \log 2$ である．ここで，T は回路の温度，K_B はボルツマン定数である．これはランダウアの限界として知られており，情報の消去に関わる計算のエネルギー消費量はこれが限界値とされている．しかし，可逆計算においては，情報の損失がないため，ランダウアの限界の対象とならない．つまり，可逆な計算のみを行うならば，エネルギーの消費量を小さくするのに限界はないということである．

現代の可逆化へのアプローチは，非可逆プログラムの可逆化のアイデアに基づいている．非可逆プログラムを可逆にするには，プログラム実行中のすべての中間の値のトレースを保存しておけばよい．このテクニックはランダウアの埋め込みとして知られている [1]．このテクニックでは，どのような非可逆プログラムにも適用でき，完璧な情報の保存を実現するとされている．しかし，このテクニックには，プログラムの実行時間に比例したスペースが必要であり，実用的ではないと言える．そこで，このような埋め込みをする必要がなく，完全に可逆性を実現する，可逆プログラミング言語がある．可逆プログラミング言語は個々に可逆な実行ステップからできている．それらをある計算状態から次の計算状態への関数とみなしたとき，それらはすべて単射であり，前方決定性と後方決定性が保証されている．非可逆のプログラミング言語では，多対一の関係になりえるため，その逆関数は一対多となるため，後方決定性が保証されない．したがって，前の状態を一意に決定することができない．可逆言語において，すべての可逆プログラムは一つの逆プログラムを持つため，それぞれの実行ステップは可逆で，それは単純な再帰降下で実現できる．また，プログラムのサイズの観点からもプログラムを逆変換することに追加のコストがかからない．

2　関連研究

2.1　可逆プログラミング言語

可逆プログラミング言語とは, プログラムの実行過程が必ず可逆になるように言語が設計されているプログラミング言語である.

可逆プログラミング言語の種類としては，可逆手続き型プログラミング言語の Janus[2]，可逆命令型プログラミング言語の R[3]，可逆関数型プログラミング言語の RFUN[4]，可逆アセンブリ言語の PISA[3], 可逆オブジェクト指向言語の Joule^R[5] や ROOPL[6] などがある.

Janus は, 局所的に可逆なプログラムの文と逆意味論への直接アクセスをもつ可逆手続き型言語であり, 整数, 固定サイズの整数配列及び可変サイズの整数スタックの 3 つのデータ型をもつ. プログラムは main 関数とそれに続くいくつかの関数から成る. プログラムの開始点は main 関数であり,main 関数内で宣言された静的変数の値がプログラムの出力となる. 構文規則や意味規則で制限を与えることにより可逆性を実現している. それにより, 可逆プログラムで可能な逆実行をするためのメカニズムを備えている.

R は, 命令形の可逆プログラミング言語であり, コンパイラ型言語である.R と R から PISA へのコンパイラは,PISA プログラムの作成を容易にするために開発された.R の構文は,LISP と C を混合したものであり, ネストされた文で表される. プログラムは main ルーチンと sub ルーチンから成り,main ルーチンがプログラムの開始点となる.

RFUN は, 関数型言語 FUN に構文を追加し, 可逆性を持たせた可逆関数型プログラミング言語である. プログラムは, いくつかの関数定義から成り, 各関数は 1 つの引数をもつ.

PISA は, 可逆コンピュータアーキテクチャ Pendulum のための可逆アセンブリ言語である.

Morgensen らによる研究では，形式化されていなかったオブジェクト指向型の可逆プログラミング言語 ROOPL[7] が示されている．

【要チェック：Theseus[8]．James と Sabry の言語 [9]．並行計算の言語 [10]．ハードウェア記述言語（HDL）SyReC[11]，可逆オブジェクト指向言語 JOOL[12]/Joole^R[5]，サーベイ論文の可逆プログラミング言語の部分も参照 [13]】

2.2 可逆フローチャート

フローチャートとは，プロセスの各ステップを基本となる処理を長方形，条件分岐する処理は菱形など，特別な処理を意味する記号を組み合わせて表現し，流れをそれらの図形の間の矢印で表すことで，アルゴリズムやプロセスを表現する図である．フローチャートの書き方は，JIS（日本産業規格（旧名：日本工業規格））で規格化されている．現在では, プログラムの構造や，プロセスを表すことにも使われている．

一方で, 可逆フローチャートは，可逆プログラミング言語の構造やプロセスを表すのに適したフローチャートである．通常のフローチャートと異なり，演算が可逆なものだけに制限され，また，フローの結合点では，必ず可逆化のための条件式が必要となる．可逆フローチャートは通常のフローチャートに可逆化を適応することにより，非可逆フローチャートもシミュレートすることができる．

2.2.1 SRL, RL

SRL と RL は，可逆フローチャートを表す可逆プログラミング言語である [14]．SRL は if 文や繰り返し文などからなる高水準の構造化プログラミング言語である．RL はジャンプ命令や goto 文などのブロックのかたまりからなる低水準の非構造化プログラミング言語である．また，これらは相互に変換可能である．

2.3 可逆オブジェクト指向プログラミング言語 ROOPL

可逆オブジェクト指向プログラミング言語 ROOPL は，オブジェクト指向とユーザ定義型がサポートされている可逆プログラミング言語である [6]．ROOPL は静的に型付けされ，継承，カプセル化，部分型多相もサポートされている．

2.4 可逆オブジェクト指向プログラミング言語 ROOPL++

可逆オブジェクト指向プログラミング言語 ROOPL++ は,ROOPL を拡張した言語である.[15]ROOPL では, オブジェクトが construct/destruct ブロック内でしか存在できなく, 使いにくいものになっていた. さらに, 配列型の不足により, 言語の表現力にも問題があった. このような欠点を解決するため,ROOPL++ が設計された.ROOPL++ は, 新しいオブジェクト定義文の追加により, オブジェクトが construct/destruct ブロック外でも存在でき, 配列型も追加されているため, 複雑な可逆プログラムの作成が可能である.

2.4.1 構文

ROOPL++ の構文と構文領域を図 1 に示す.ROOPL++ のプログラムは, フィールドとメソッドを持つ 1 つ以上のクラス定義から成る. プログラムは引数のない main メソッドから実行され,main メソッドが含まれているクラスのフィールドの値がプログラムの実行結果となる.

2.4.2 意味領域と意味関数

ROOPL++ の意味を表示的意味論で記述する．意味領域を図 2 に, 意味関数を図 3 示す.

Locs は環境やストアで用いられるロケーションを表し, その値は非負整数である.*Envs* は環境を表し, 識別子からロケーションへの部分関数である.*ClassEnvs* はクラス環境を表し, クラス名からフィールドとメソッドの組への部分関数である.*Stores* はストアを表し, ロケーションから値への部分関数である. また, 環境やストアは演算子 $[\cdot \mapsto \cdot]$ で更新される. 演算子 $[\cdot \mapsto \cdot]$ の定義を以下に示す.

$$\gamma[x' \mapsto l](x) = \begin{cases} l & \text{if } x' = x \\ \gamma(x) & \text{if } x' \neq x \end{cases}$$

$$\mu[l' \mapsto v](l) = \begin{cases} v & \text{if } l' = l \\ \mu(l) & \text{if } l' \neq l \end{cases}$$

環境の省略記法 $[x_1 \mapsto l_1, \ldots, x_n \mapsto l_n]$ は, 識別子 x_1 から x_n をロケーション l_1 から l_n にそれぞれ対応付けた環境に更新することを表す. ストアの省略記法 $[l_1 \mapsto v_1, \ldots, l_n \mapsto v_n]$ も同様にロケーション l_1 から l_n を値 v_1 から v_n にそれぞれ対応付けたストアに更新することを表す. *Objects* はオブジェクトを表し, そのオブジェクトのクラス ID とそのオブジェクトのフィールドから値への部分関数の組である.*Values* は値を表し, それは, 整数, オブジェクト, ロケーション, ロケーションのベクトルのいずれかである.

意味関数 $\mathcal{E}$ は式を受け取り値を返す関数,$\mathcal{S}$ は文, 環境, クラス環境, ストアを受け取り文の実行により更新されたストアを返す関数,$\mathcal{P}$ はプログラムを受け取り環境とストアの組を返す関数である.

2.4.3 意味関数 $\mathcal{E}$ の定義

意味関数 $\mathcal{E}$ の各式に対する定義を図 4 に示す. 整数 n の場合, 値としての整数 $\overline{n}$ が返される. 変数 x の場合, まず環境 γ から x に対応するロケーションを求める. そしてストア μ からそのロケーションに対応する値が返される.2 つの式の演算の場合, まず e_1, e_2 を評価し, その 2 つの値を関数 $[\![\otimes]\!]$ で評価した値を返す. ここで, 関数 $[\![\otimes]\!]$ の定義を図 5 に示す. 配列の場合, まず式 e が評価され, インデックス v を求める. そして, 環境 γ とストア μ から識別子 x に対応する, ロケーションが格納されている配列 l' を求め, その配列からインデックスに対応するロケーションを求める. 最後に, ストア μ からそのロケーションに対応する値が返される.

2.4.4 意味関数 $\mathcal{S}$ の定義

意味関数 $\mathcal{S}$ の各文に対する定義を図 6,7 に示す.skip 文では, ストア μ に変化が起きずそのままのストアが返され

$$
\begin{array}{rcll}
prog & ::= & cl^{+} & \text{(program)} \\
cl & ::= & \textbf{class}\ c\ (\textbf{inherits}\ c)^{?}\ (t\ x)^{*}\ m^{+} & \text{(class definition)} \\
d & ::= & c \mid c\texttt{[}e\texttt{]} \mid \textbf{int}\texttt{[}e\texttt{]} & \text{(class and arrays)} \\
t & ::= & \textbf{int} \mid c \mid \textbf{int}\texttt{[]} \mid c\texttt{[]} & \text{(data type)} \\
y & ::= & x \mid x\texttt{[}e\texttt{]} & \text{(variable identifiers)} \\
m & ::= & \textbf{method}\ q(t\ x,\ \ldots,\ t\ x)\ s & \text{(method)} \\
s & ::= & y\ \odot\texttt{=}\ e \mid y\ \texttt{<=>}\ y & \text{(assignment)} \\
 & \mid & \textbf{if}\ e\ \textbf{then}\ s\ \textbf{else}\ s\ \textbf{fi}\ e & \text{(conditional)} \\
 & \mid & \textbf{from}\ e\ \textbf{do}\ s\ \textbf{loop}\ s\ \textbf{until}\ e & \text{(loop)} \\
 & \mid & \textbf{construct}\ c\ x\quad s\quad \textbf{destruct}\ x & \text{(object block)} \\
 & \mid & \textbf{local}\ t\ x\ \texttt{=}\ e\quad s\quad \textbf{delocal}\ t\ x\ \texttt{=}\ e & \text{(local variable block)} \\
 & \mid & \textbf{new}\ d\ y \mid \textbf{delete}\ d\ y & \text{(object con- and destruction)} \\
 & \mid & \textbf{copy}\ d\ y\ y \mid \textbf{uncopy}\ d\ y\ y & \text{(reference con- and destruction)} \\
 & \mid & \textbf{call}\ q(x,\ \ldots,\ x) \mid \textbf{uncall}\ q(x,\ \ldots,\ x) & \text{(local method invocation)} \\
 & \mid & \textbf{call}\ x\texttt{::}q(x,\ \ldots,\ x) \mid \textbf{uncall}\ x\texttt{::}q(x,\ \ldots,\ x) & \text{(method invocation)} \\
 & \mid & \textbf{skip} \mid s\ s & \text{(statement sequence)} \\
e & ::= & \overline{n} \mid x \mid x\texttt{[}e\texttt{]} \mid \texttt{nil} \mid e\ \otimes\ e & \text{(expression)} \\
\odot & ::= & \texttt{+} \mid \texttt{-} \mid \texttt{\^{}} & \text{(operator)} \\
\otimes & ::= & \odot \mid \texttt{*} \mid \texttt{/} \mid \texttt{\%} \mid \texttt{\&} \mid \texttt{|} \mid \texttt{\&\&} \mid \texttt{||} \mid \texttt{<} \mid \texttt{>} \mid \texttt{=} \mid \texttt{!=} \mid \texttt{<=} \mid \texttt{>=} & \text{(operator)}
\end{array}
$$

$prog \in Programs$ $s \in$ Statements $n \in$ Constants
$cl \in$ Classes $e \in$ Expressions $x \in$ VarIDs
$t \in$ Types $\odot \in$ ModOps $q \in$ MethodIDs
$m \in$ Methods $\otimes \in$ Operators $c \in$ ClassIDs

図 1 ROOPL++ の構文と構文領域 ([15] より転載)

$$
\begin{array}{rl}
l : Locs & = \mathbb{N}_0 \\
\gamma : Envs & = (VarIDs \to Locs) \\
\mu : Stores & = (Locs \to Values) \\
\Gamma : ClassEnvs & = (ClassIDs \to (fs \times ms)\ ) \\
fs : Fields & = \{t_1\ f_1\ \cdots\ t_n\ f_n\} \\
ms : Methods & = \{\textbf{method}\ q_1\ (\cdots)\ s_1\ \cdots\ \textbf{method}\ q_n\ (\cdots)\ s_n\} \\
\mu : Stores & = (Locs \to Values) \\
Objects & = \{\ \langle c_f,\ \gamma_f \rangle \mid c_f \in ClassIDs\ \wedge\ \gamma_f \in Envs\ \} \\
v : Values & = \mathbb{Z} \cup\ Objects \cup\ Locs \cup\ [Locs]
\end{array}
$$

図 2 意味領域 ([15] から一部改変)

る. 代入文では, 識別子 x の値と式 e の値を演算子 $\odot$ で評価し, 値 (v) を得る. その後, 識別子 x を v に対応付けたストアが返される.loop 文では,e_1 の値が真 (0 でない) 場合, 文 s_1 が実行され, その後ストア $\mu'(s_1$ 実行後のストア) のもとで意味関数 $\mathcal{L}$ が実行される.e_1 が偽の場合は $_2$ も偽で

$$
\begin{array}{l}
\mathcal{E} : Expressions \to (Envs \times Stores) \to Values \\
\mathcal{S} : Statement \to (Envs \times ClassEnvs) \to Stores \to Stores \\
\mathcal{P} : Programs \to (Envs \times Stores)
\end{array}
$$

図 3 意味関数 ([15] を参考に作成)

なければならない. この条件式 e_2 は文を逆にする際に必要となり, これにより可逆化を実現することができる. また,loop 文の意味論に現れる意味関数 $\mathcal{L}$ は $\mathcal{S}$ と同様に環境, ストア, マップを受け取り, 更新されたストアを返す関数であり,loop 文の意味を簡潔に記述するために定義されている.$\mathcal{L}$ は,e_2 の値が偽 (0) の場合,s_2,s_1 の順に文が実行されその結果であるストアのもとで再び $\mathcal{L}$ が実行される.e_2 の値が真である場合は, ループが終了しそのままのストアが返される. if 文では,e_1 の値が真である場合, 文 s_1 が実行されその結果のストアが返される.(e_2 の値も真とならなければならない.)e_1 の値が偽である場合, 文 s_2 が実行されその結果のストアが返される.(e_2 の値も偽とならなければな

$$
\begin{aligned}
\mathcal{E}[\![n]\!](\gamma,\mu) &= \overline{n} \\
\mathcal{E}[\![x]\!](\gamma,\mu) &= \mu(\gamma(x)) \\
\mathcal{E}[\![nil]\!](\gamma,\mu) &= 0 \\
\mathcal{E}[\![e_1 \otimes e_2]\!](\gamma,\mu) &= [\![\otimes]\!](\mathcal{E}[\![e_1]\!](\gamma,\mu), \mathcal{E}[\![e_2]\!](\gamma,\mu)) \\
\mathcal{E}[\![x\,[e]\,]\!](\gamma,\mu) &= \mu(l'[v]) \\
&\text{where } v = \mathcal{E}[\![e]\!](\gamma,\mu),\ l' = \mu(\gamma(x))
\end{aligned}
$$

図 4　意味関数 $\mathcal{E}$ の定義 ([15] から一部改変)

らない.) ローカルでのメソッド呼び出し文 call は, 現在のクラスに定義されているメソッドを呼び出すための文である. ここではまず, 識別子 this に対応する現在のクラスのオブジェクト (クラス名 c とクラスフィールド γ' の組) を取り出す. 次に, クラス環境から現在のクラスのフィールドとメソッドを取り出し, 呼び出されているメソッド q を探し出す. そして, メソッド q の仮引数に実引数のロケーションを対応付けたものを γ' に追加し, 拡張する (γ''). この操作は実引数を参照渡ししていることを表す. 最後に, 環境 γ'' とクラス環境 Γ のもとでメソッドの本体を実行し, その結果のストアを返す. ローカルでのメソッド逆呼び出し文 uncall は, メソッドを逆実行し, その結果のストアを返す文である.

2.4.5　意味関数 $\mathcal{P}$ の定義

意味関数 $\mathcal{P}$ の定義を図 8 に示す. ここではまず関数 gen により全てのクラスからマップが生成される.

2.5　可逆プログラミング言語の処理系

プログラミング言語の処理系とは, プログラミング言語で書かれたプログラムの実行や変換をするためのプログラムである. プログラムを逐次解析しながら実行するプログラムをインタプリタ, プログラムを機械語やその他のプログラミング言語に変換するプログラムをコンパイラまたはトランスレータという. 現在, 可逆プログラミング言語の処理系として, インタープリタは,Janus インタプリタ,PISA インタプリタの PHPISA や Pendulum VM,R-WHILE インタプリタなどが実装されている. コンパイラは,ROOPL を低水準言語の PISA に変換する ROOPLtoPISA,R 言語を Pendulum アセンブリ言語に変換する R コンパイラなどがある. 【以下も要チェック】

- ROOPL to PISA コペンハーゲン大 3 年生
- Pendulum VM (PISA インタープリタ) コペンハーゲン大 3 年生
- PISA オンラインインタプリタ PHPISA フロリダ大学学部生
- pendvm　　MIT の学生
- phpisa　　MIT の学生

$$
\begin{aligned}
[\![+]\!](v_1,\ v_2) &= v_1 + v_2 \\
[\![\%]\!](v_1,\ v_2) &= v_1\ mod\ v_2 \\
[\![-]\!](v_1,\ v_2) &= v_1 - v_2 \\
[\![\&]\!](v_1,\ v_2) &= v_1\ and\ v_2 \\
[\![*]\!](v_1,\ v_2) &= v_1 \times v_2 \\
[\![|]\!](v_1,\ v_2) &= v_1\ or\ v_2 \\
[\![/]\!](v_1,\ v_2) &= \frac{v_1}{v_2} \\
[\![\hat{}]\!](v_1,\ v_2) &= v_1\ xor\ v_2 \\
[\![\&\&]\!](v_1,\ v_2) &= \begin{cases} 0 & \text{if } v_1 = 0 \vee v_2 = 0 \\ 1 & \text{otherwise} \end{cases} \\
[\![<=]\!](v_1,\ v_2) &= \begin{cases} 1 & \text{if } v_1 \leq v_2 \\ 0 & \text{otherwise} \end{cases} \\
[\![||]\!](v_1,\ v_2) &= \begin{cases} 0 & \text{if } v_1 = v_2 = 0 \\ 1 & \text{otherwise} \end{cases} \\
[\![>=]\!](v_1,\ v_2) &= \begin{cases} 1 & \text{if } v_1 \geq v_2 \\ 0 & \text{otherwise} \end{cases} \\
[\![<]\!](v_1,\ v_2) &= \begin{cases} 1 & \text{if } v_1 < v_2 \\ 0 & \text{otherwise} \end{cases} \\
[\![=]\!](v_1,\ v_2) &= \begin{cases} 1 & \text{if } v_1 = v_2 \\ 0 & \text{otherwise} \end{cases} \\
[\![>]\!](v_1,\ v_2) &= \begin{cases} 1 & \text{if } v_1 > v_2 \\ 0 & \text{otherwise} \end{cases} \\
[\![!=]\!](v_1,\ v_2) &= \begin{cases} 1 & \text{if } v_1 \neq v_2 \\ 0 & \text{otherwise} \end{cases}
\end{aligned}
$$

図 5　関数 $[\![\otimes]\!]$ の定義 ([15] より転載)

- R コンパイラ　　MIT の学生
- R-WHILE　　コペンハーゲン大の学生ら
- Janus ブレーメン大学の研究者やコペンハーゲン大の学生
- Syrec ブレーメン大学の研究者
- Bob コペンハーゲン大
- 量子プログラミング

3　目的

可逆な計算機の実現には, 可逆プログラミング言語は重要であり, また, 可逆言語のインタープリタは, 可逆アルゴリズムを開発する際に助けとなる. 我々の知る限りでは, 現段階では,ROOPL++ や JOULE といった可逆オブジェクト指向プログラミング言語は開発されているが, インタープリタは実装されていない. したがって, 可逆オブジェクト指向プログラミング言語を試すことができない[要出典]

$$\mathcal{S}[\![\textbf{skip}]\!]_{\gamma,\Gamma}(\mu) = \mu$$

$$\mathcal{S}[\![x \odot= e]\!]_{\gamma,\Gamma}(\mu) = \mu[\gamma(x) \mapsto v]$$
$$\text{where } v = [\![\odot]\!](\ \mu(\gamma(x)), \mathcal{E}[\![e]\!](\gamma,\mu)\ )$$

$$\mathcal{S}[\![x_1 \texttt{<=>} x_2]\!]_{\gamma,\Gamma}(\mu) = \mu[\gamma(x_1) \mapsto v_2,\ \gamma(x_2) \mapsto v_1]$$
$$\text{where } v_1 = \mu(\gamma(x_1)),\ v_2 = \mu(\gamma(x_2))$$

$$\mathcal{S}[\![\textbf{from } e_1 \textbf{ do } s_1 \textbf{ loop } s_2 \textbf{ until } e_2]\!]_{\gamma,\Gamma}(\mu)$$
$$= \mathcal{L}[\![(e_1, s_1, s_2, e_2)]\!]_{\gamma,\Gamma}(\mu') \ \text{ if } \mathcal{E}[\![e_1]\!](\gamma,\mu) \neq 0$$
$$\text{where } \mu' = \mathcal{S}[\![s_1]\!]_{\gamma,\Gamma}(\mu)$$

$$\mathcal{L}[\![(e_2, s_1, s_2, e_2)]\!]_{\gamma,\Gamma}(\mu)$$
$$= \begin{cases} \mu & \text{if } \mathcal{E}[\![e_2]\!](\gamma,\mu) \neq 0 \\ \mathcal{L}[\![(e_1, s_1, s_2, s_2)]\!]_{\gamma,\Gamma}(\mu'') & \text{if } \mathcal{E}[\![e_2]\!](\gamma,\mu) = 0 \wedge \mathcal{E}[\![e_1]\!](\gamma,\mu') = 0 \\ \quad \text{where } \mu'' = \mathcal{S}[\![s_1]\!]_{\gamma,\Gamma}(\mu'),\ \mu' = \mathcal{S}[\![s_2]\!]_{\gamma,\Gamma}(\mu) \end{cases}$$

$$\mathcal{S}[\![\textbf{if } e_1 \textbf{ then } s_1 \textbf{ else } s_2 \textbf{ fi } e_2]\!]_{\gamma,\Gamma}(\mu)$$
$$= \begin{cases} \mu' & \text{if } (\mathcal{E}[\![e_1]\!](\gamma,\mu) \neq 0) \ \wedge \ (\mathcal{E}[\![e_2]\!](\gamma,\mu') \neq 0) \\ \quad \text{where } \mu' = \mathcal{S}[\![s_1]\!]_{\gamma,\Gamma}(\mu) & \\ \mu' & \text{if } (\mathcal{E}[\![e_1]\!](\gamma,\mu) = 0) \ \wedge \ (\mathcal{E}[\![e_2]\!](\gamma,\mu') = 0) \\ \quad \text{where } \mu' = \mathcal{S}[\![s_2]\!]_{\gamma,\Gamma}(\mu) & \end{cases}$$

$$\mathcal{S}[\![\textbf{call } q(y_1, \ldots, y_n)]\!]_{\gamma,\Gamma}(\mu) = \mathcal{S}[\![s]\!]_{\gamma'',\Gamma}(\mu)$$
$$\text{where}$$
$$\mu(\gamma(this)) = \langle c,\ \gamma' \rangle \quad \Gamma(c) = (\ fields,\ methods\ )$$
$$(\ \textbf{method } q(t_1\ z_1, \ldots, t_n\ z_n)\ s\ ) \ \in \ methods$$
$$\gamma'' = \gamma'[\ this \mapsto \gamma(this),\ z_1 \mapsto \gamma(y_1), \ldots,\ z_k \mapsto \gamma(y_k)\ ]$$

$$\mathcal{S}[\![\textbf{uncall } q(y_1, \ldots, y_n)]\!]_{\gamma,\Gamma}(\mu) = \mu'$$
$$\text{where } \mathcal{S}[\![\textbf{call } q(y_1, \ldots, y_n)]\!]_{\gamma,\Gamma}(\mu') = \mu$$

$$\mathcal{S}[\![\textbf{call } x_0 :: q(y_1, \ldots, y_n)]\!]_{\gamma,\Gamma}(\mu) = \mathcal{S}[\![s]\!]_{\gamma'',\Gamma}(\mu)$$
$$where$$
$$l = \gamma(x_0) \quad l' = \mu(l) \quad \mu(l') = \langle c, \gamma' \rangle \quad \Gamma(c) = (\ fields,\ methods\ )$$
$$(\ \textbf{method } q(t_1\ z_1, \ldots, t_n\ z_n)\ s\ ) \ \in \ methods$$
$$\gamma'' = \gamma'[\ this \mapsto \gamma(x_0),\ z_1 \mapsto \gamma(y_1), \ldots,\ z_n \mapsto \gamma(y_k)\ ]$$

$$\mathcal{S}[\![\textbf{call } x[e] :: q(y_1, \ldots, y_n)]\!]_{\gamma,\Gamma}(\mu) = \mathcal{S}[\![s]\!]_{\gamma'',\Gamma}(\mu)$$
$$where$$
$$v = \mathcal{E}[\![e]\!](\gamma,\mu) \quad l = \mu(\gamma(x))[v] \quad l' = \mu(l)$$
$$\mu(l') = \langle c,\ \gamma' \rangle,\ \Gamma(c) = (\ fields,\ methods\ )$$
$$(\ \textbf{method } q(t_1\ z_1, \ldots, t_n\ z_n)\ s\ ) \ \in \ methods$$
$$\gamma'' = \gamma'[\ this \mapsto \gamma(x_0),\ z_1 \mapsto \gamma(y_1), \ldots,\ z_n \mapsto \gamma(y_k)\ ]$$

図 6　意味関数 $\mathcal{S}$ の定義 1([15] から一部改変)

$\mathcal{S}[\![\textbf{uncall } y :: q(y_1, \ldots, y_n)]\!]_{\gamma,\Gamma}(\mu) = \mu'$
where $\mathcal{S}[\![\textbf{call } y :: q(y_1, \ldots, y_n)]\!]_{\gamma,\Gamma}(\mu') = \mu$

$\mathcal{S}[\![\textbf{construct } c\ x \quad s \quad \textbf{destruct } x]\!]_{\gamma,\Gamma}(\mu) = \mu''[l'_1 \mapsto 0, \ldots, l'_n \mapsto 0]$
where
$\Gamma(c) = \Big(\overbrace{\{\langle t_1, f_1\rangle, \ldots, \langle t_n, f_n\rangle\}}^{fields}, methods \Big) \quad \gamma' = [f_1 \mapsto l'_1, \ldots, f_n \mapsto l'_n]$
$\{ l, l'_1, \ldots, l'_n \} \cap \mathrm{dom}(\mu) = \emptyset \quad \mu' = \mu\,[l'_1 \mapsto 0, \ldots, l'_n \mapsto 0, l \mapsto l', l' \mapsto \langle c, \gamma'\rangle]$
$\mu'' = \mathcal{S}[\![s]\!]_{\gamma[x\mapsto r]}(\mu') \quad \mu''(l'_1) = 0, \ldots, \mu''(l'_n) = 0$

$\mathcal{S}[\![x\texttt{[}e_1\texttt{]} \odot= e_2]\!]_{\gamma,\Gamma}(\mu) = \mu[l' \mapsto w']$
where $v_1 = \mathcal{E}[\![e_1]\!](\gamma,\mu),\ v_2 = \mathcal{E}[\![e_2]\!](\gamma,\mu),\ l' = \mu(\gamma(x))\texttt{[}v_1\texttt{]},\ w = \mu(l'),\ w' = [\![\odot]\!](w, v_2)$

$\mathcal{S}[\![\textbf{new } c\ x]\!]_{\gamma,\Gamma}(\mu) = \mu\Big[l \mapsto l_0,\ l_0 \mapsto \langle c, \gamma'\rangle,\ l_1 \mapsto 0, \ldots, l_m \mapsto 0\Big]$
where
$\Gamma(c) = \Big(\overbrace{\{\langle t_1, f_1\rangle, \ldots, \langle t_m, f_m\rangle\}}^{fields}, methods \Big) \quad \gamma' = [f_1 \mapsto l_0, \ldots, f_m \mapsto l_0]$
$\gamma(x) = l \quad \{ l_0, \ldots, l_m \} \cap \mathrm{dom}(\mu) = \emptyset$

$\mathcal{S}[\![\textbf{new } c\ x\texttt{[}e\texttt{]}]\!]_{\gamma,\Gamma}(\mu) = \mu\Big[l \mapsto l_0,\ l_0 \mapsto \langle c, \gamma'\rangle,\ l_1 \mapsto 0, \ldots, l_m \mapsto 0\Big]$
where
$\Gamma(c) = \Big(\overbrace{\{\langle t_1, f_1\rangle, \ldots, \langle t_m, f_m\rangle\}}^{fields}, methods \Big) \quad \gamma' = [f_1 \mapsto l_0, \ldots, f_m \mapsto l_0]$
$v = \mathcal{E}[\![e]\!](\gamma,\mu) \quad l = \mu(\gamma(x))[v] \quad \{l_0, \ldots, l_m\} \cap \mathrm{dom}(\mu) = \emptyset$

$\mathcal{S}[\![\textbf{delete } c\ y]\!]_{\gamma,\Gamma}(\mu) = \mu'$
where $\mathcal{S}[\![\textbf{new } c\ y]\!]_{\gamma,\Gamma}(\mu') = \mu$

$\mathcal{S}[\![\textbf{new } a\texttt{[}e\texttt{]}\ x]\!]_{\gamma,\Gamma}(\mu) = \mu\big[l \mapsto \{l'_1, \ldots, l'_n\},\ l'_1 \mapsto 0, \ldots, l'_n \mapsto 0\big]$
where $n = \mathcal{E}[\![e]\!](\gamma,\mu),\ l = \gamma(x),\ \{ l'_1, \ldots, l'_n \} \cap \mathrm{dom}(\mu) = \emptyset$

$\mathcal{S}[\![\textbf{delete } a\texttt{[}e\texttt{]}\ x]\!]_{\gamma,\Gamma}(\mu) = \mu'$
where $\mathcal{S}[\![\textbf{new } a\texttt{[}e\texttt{]}\ x]\!]_{\gamma,\Gamma}(\mu') = \mu$

$\mathcal{S}[\![\textbf{copy } c\ x\ x']\!]_{\gamma,\Gamma}(\mu) = \mu[l' \mapsto v]$
where $\gamma(x) = l,\ \gamma(x') = l',\ \mu(l) = v$

$\mathcal{S}[\![\textbf{uncopy } c\ x\ x']\!]_{\gamma,\Gamma}(\mu) = \mu'$
where $\mathcal{S}[\![\textbf{copy } c\ x\ x']\!]_{\gamma,\Gamma}(\mu') = \mu$

$\mathcal{S}[\![\textbf{local } c\ x = e_1 \quad s \quad \textbf{delocal } x = e_2]\!]_{\gamma,\Gamma}(\mu) = \mu'$
where
$v_1 = \mathcal{E}[\![e_1]\!](\gamma,\mu),\ v_2 = \mathcal{E}[\![e_2]\!](\gamma,\mu),\ r \notin \mathrm{dom}(\mu)$
$\mu' = \mathcal{S}[\![s]\!]_{\gamma[x\mapsto r],\Gamma}(\mu[r \mapsto v_1]),\ \mu'' = \mu'[r \mapsto v_2],\ \mu''(x) = v_2$

図 7　意味関数 $\mathcal{S}$ の定義 2([15] から一部改変)

$$\mathcal{P}[\![cl_1 \ldots cl_n]\!] = (\gamma, \mu')$$

where

$$\Gamma = \mathrm{gen}(cl_1,\ \ldots,\ cl_n) \qquad \Gamma(c) = \Big(\overbrace{\{\langle t_1,\ f_1\rangle,\ \ldots,\ \langle t_i,\ f_i\rangle\}}^{fields},\ methods\Big)$$
$$\Big(\ \mathbf{method\ main()}\ s\ \Big) \in methods \qquad \gamma = [f_1 \mapsto 1,\ \ldots,\ f_i \mapsto i,\ this \mapsto i+1]$$
$$\mu = [1 \mapsto 0,\ \ldots,\ i \mapsto 0,\ i+1 \mapsto i+2,\ i+2 \mapsto \langle c,\ \gamma\rangle] \qquad \mu' = \mathcal{S}[\![s]\!]_{\gamma,\Gamma}(\mu)$$

図 8　意味関数 $\mathcal{P}$ の定義([15] から一部改変)

が現状である. そのため, 本研究では ROOPL++ のインタープリタを実装し, そして, 誰でも試せるようにオンラインインタープリタを実装することを目的とする.

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] Lutz, C.: Janus: a time-reversible language (1986). *Letter to R. Landauer.*

[3] Frank, M.P.: Reversibility for Efficient Computing, PhD Thesis (1999).

[4] Yokoyama, T., Axelsen, H.B. and Glück, R.: Towards a Reversible Functional Language, Vol.7165, Springer-Verlag, pp.14–29 (2012).

[5] Schultz, U.P.: Reversible Object-Oriented Programming with Region-Based Memory Management, *Reversible Computation* (Kari, J. and Ulidowski, I., Eds.), Cham, Springer-Verlag, pp.322–328 (2018).

[6] Haulund, T.: Design and Implementation of a Reversible Object-Oriented Programming Language, Master's thesis, University of Copenhagen (2017).

[7] Haulund, T., Mogensen, T.Æ. and Glück, R.: Implementing Reversible Object-Oriented Language Features on Reversible Machines, *Reversible Computation* (Phillips, I. and Rahaman, H., Eds.), Cham, Springer-Verlag, pp.66–73 (2017).

[8] James, R.P. and Sabry, A.: Theseus: A High Level Language for Reversible Computing (Work-in-progress report), *Conference on Reversible Computation* (2014).

[9] James, R.P. and Sabry, A.: Isomorphic Interpreters from Logically Reversible Abstract Machines, Vol.7581, Springer-Verlag, pp.57–71 (2013).

[10] Ulidowski, I., Phillips, I. and Yuen, S.: Reversing Event Structures, *New Generation Computing*, Vol.36, No.3, pp.281–306 (2018).

[11] Wille, R., Offermann, S. and Drechsler, R.: SyReC: A programming language for synthesis of reversible circuits, pp.184–189 (2010).

[12] Schultz, U.P. and Axelsen, H.B.: Elements of a Reversible Object-Oriented Language, *Reversible Computation* (Devitt, S. and Lanese, I., Eds.), Cham, Springer International Publishing, pp.153–159 (2016).

[13] Mezzina, C.A., Schlatte, R., Glück, R., Haulund, T., Hoey, J., Holm Cservenka, M., Lanese, I., Mogensen, T.Æ., Siljak, H., Schultz, U.P. and Ulidowski, I.: *Software and Reversible Systems: A Survey of Recent Activities*, pp.41–59, Springer International Publishing (2020).

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

[15] Cservenka, M.H.: Design and Implementation of Dynamic Memory Management in a Reversible Object-Oriented Programming Language, Master's thesis, University of Copenhagen (2018).