Reversible Computation and Reversible Programming Languages

Tetsuo Yokoyama

Graduate School of Information Science
Nagoya University
Example: Fibonacci-Pairs

```
procedure fib
    if n=0 then x1 += 1
    test
    x2 += 1
    else n -= 1
    call fib
    x1 += x2
    x1 <=> x2
fi
```

Assertion: x1 = x2

Global store (three integer variables, initially zero)
Forward & Backward Computation

procedure main_fwd
    n += 4
    call fib

procedure main_bwd
    x1 += 5
    x2 += 8
    uncall fib

[ReillyFederighi65, LutzDerby82]
Janus: a Reversible Language

- To our knowledge, the first reversible structured language
  - Suggested for a class at Caltech [Lutz and Dervy 1982]
- Imperative language
- Global store, no local store
- Scalar and array types, integer values
- Structured control operators (IF, LOOP)
- Simple procedures (correspond to loops)
  - No return value, side effects on global store
Syntax of Janus

\[
p ::= vdec^\ast \ (\text{procedure } id \ s)^+ \\
vdec ::= x \mid x[c] \\
s ::= x \oplus= e \mid x[e] \ominus= e \mid \text{if } e \text{ then } s \text{ else } s \text{ fi } e \mid \text{from } e \text{ do } s \text{ loop } s \text{ until } e \mid \text{call id } \mid \text{uncall id } \mid \text{skip } \mid s \ s \\
e ::= c \mid x \mid x[e] \mid \sim e \mid e \oslash e \\
c ::= 0 \mid 1 \mid \cdots \mid 4294967295 \\
\oplus ::= + \mid - \mid \sim \\
\oslash ::= \oplus \mid * \mid / \mid \% \mid \slash \mid \& \mid | \mid \&\& \mid | | \\
< \mid > \mid = \mid != \mid <= \mid >=
\]

Assignment operations
Reversible Conditional
Reversible Loop
Procedure call/uncall
32-bit integers
Control Flow Operators

Remark: Circles are assertions

if $e_1$ then $s_1$ else $s_2$ fi $e_2$

from $e_1$ do $s_1$ loop $s_2$ until $e_2$

Remark: Circles are assertions
Local Inversion of CFOs

**Conditional (IF)**

\[
\text{if } e_1 \text{ then } s_1 \text{ else } s_2 \text{ fi } e_2
\]

**Loop (DO-UNTIL-LOOP)**

\[
\text{from } e_1 \text{ do } s_1 \text{ loop } s_2 \text{ until } e_2
\]

**Local inversion**

\[
\text{if } e_2 \text{ then } s_1^{-1} \text{ else } s_2^{-1} \text{ fi } e_1
\]

\[
\text{from } e_2 \text{ do } s_1^{-1} \text{ loop } s_2^{-1} \text{ until } e_1
\]

⇒ Conditional and Loop are reversible.
Skip and Sequence

\[
\frac{\sigma \vdash_{stmt} s_1 \Rightarrow \sigma'}{\sigma \vdash_{stmt} s_2 \Rightarrow \sigma''} \quad \frac{\sigma' \vdash_{stmt} s_2 \Rightarrow \sigma''}{\text{Seq}} 
\]

\[
\frac{\sigma \vdash_{stmt} \text{skip} \Rightarrow \sigma}{\text{Skip}}
\]

\[
\text{Local inversion}
\]

\[
\Rightarrow \text{Skip and sequence are reversible.}
\]
Procedure Call / Uncall

\[
\begin{align*}
\sigma |\!\!\!\vdash_{stmt} & \Gamma(id) \Rightarrow \sigma' \\
\sigma |\!\!\!\vdash_{stmt} & \text{call } id \Rightarrow \sigma' \\
\sigma' |\!\!\!\vdash_{stmt} & \Gamma(id) \Rightarrow \sigma \\
\sigma |\!\!\!\vdash_{stmt} & \text{uncall } id \Rightarrow \sigma'
\end{align*}
\]

Call

Uncall

\[
\Gamma \in \text{Idens[Janus]} \vdash \text{Stmts[Janus]}
\]

\[
\begin{align*}
\text{call id} & \leftrightarrow \text{uncall id} \\
\text{uncall id} & \leftrightarrow \text{call id}
\end{align*}
\]

Local inversion

⇒ Procedure call / uncall is reversible.
Abbreviation: \( x \oplus= e \iff x := x \oplus e \)

If variable \( x \) must not occur in expression \( e \), this is again an example of reversible update.

\( \Rightarrow \) C-like Assignments are reversible.
Evaluation of Expressions

Judgment: \( \sigma \vdash_{\text{expr}} e \Rightarrow v \)

\( \sigma \vdash_{\text{expr}} c \Rightarrow [c] \) \hspace{1cm} \text{Con}

\( \sigma \vdash_{\text{expr}} x \Rightarrow \sigma(x) \) \hspace{1cm} \text{Var}

\( \sigma \vdash_{\text{expr}} e_1 \Rightarrow v_1 \) \hspace{0.5cm} \( \sigma \vdash_{\text{expr}} e_2 \Rightarrow v_2 \) \hspace{0.5cm} \( [[\odot]](v_1, v_2) = v \) \hspace{1cm} \text{BinOp}

\( \odot \in \{+, -, \sim, \ldots\} \)

Store \( \sigma : \text{Var} \Rightarrow \text{Val} \)

\( \Rightarrow \) Evaluation of expressions is \text{fwd deterministic.} \nBut it is not \text{backward deterministic.} \)
Non-injective Binary Operators

• Some of the binary operators (others are similar)

\[ [+](v_1, v_2) = (v_1 + v_2) \mod 2^{32} \]

\[ [\neq](v_1, v_2) = \begin{cases} 0 & \text{if } v_1 \neq v_2 \\ 1 & \text{if } v_1 = v_2 \end{cases} \]

• No binary operator is injective.
• There does not exist a unique inverse operation.

**Question:** Why does this *not* harm the reversibility of statements?
Answer: Reversible Update

Evaluation of RHS (Irreversible)

\[ \sigma \vdash_{expr} e \Rightarrow v \]

\[ \sigma \uplus \{ x \mapsto v_1 \} \vdash_{stmt} x \odot = e \Rightarrow \sigma \uplus \{ x \mapsto v_2 \} \]

\[ v_2 = \llbracket \odot \rrbracket (v_1, v) \]

- \( \sigma \vdash_{expr} e \Rightarrow v \) is fwd deterministic.
- Variable \( x \) must not occur in expression \( e \).
- Function \( \lambda v'. \llbracket \odot \rrbracket (v', v) \) is injective for any \( v \) when \( \odot \) is +, -, or \( \sim \).
- \( \Rightarrow \) It has an inverse function.

\( \Rightarrow \) C-like Assignments are reversible.
Theorem: Janus Statements are Reversible

\[ \forall s \in \text{Stmts}[\text{Janus}], \exists s' \in \text{Stmts}[\text{Janus}], \forall \sigma, \sigma' \in \text{Stores}[\text{Janus}]. \]

\[ \sigma \vdash_{stmt} s \Rightarrow \sigma' \iff \sigma' \vdash_{stmt} s' \Rightarrow \sigma \]

Remarks:

- Evaluation of expressions is not reversible. But this does not harm this reversibility.
- Referential transparency: \( s = s' \Rightarrow s_1 s s_2 = s_1 s' s_2 \)
- We cannot write irreversible programs in Janus.
Criteria of Computational Strength

R-Turing completeness

A reversible language is called r-Turing complete if it can simulate reversible Turing machines (RTM), cleanly.

RTM in Janus:
Less than 40 lines

```
procedure inst
    if q=q1[pc] then
        if s=s1[pc] then // Symbol rule:
            q += q2[pc]-q1[pc] // set q to q2[pc]
            s += s2[pc]-s1[pc] // set s to s2[pc]
        else // Shift rule:
            if s1[pc]=SLASH then
                q += q2[pc]-q1[pc] // set q to q2[pc]
            if s2[pc]=RIGHT then
                call pushtape(s,left) // push s on left
                uncall pushtape(s,right) // pop right to s
            else
                if s2[pc]=LEFT then
                    call pushtape(s,right) // push s on right
                    uncall pushtape(s,left) // pop left to s

procedure pushtape
    fi s2[pc]=LEFT
    fi s2[pc]=RIGHT
    fi s1[pc]=SLASH
    fi s=q2[pc]
fi empty(stk)
```
• Assignment:
  – Zero-cleared copying, Zero-clearing by a constant

\[
\begin{align*}
\{ x := 0, y := v \} \\
x ^\dagger &= y \\
\{ x := v, y := v \}
\end{align*}
\]

\[
\begin{align*}
\{ x := v, y := v \} \\
x ^\dagger &= y \\
\{ x := 0, y := v \}
\end{align*}
\]

• Garbage manipulation:
  – Temporary stack

```
procedure alloc_tmp
    tmp_sp += 1
    tmp <=> tmp_stack[tmp_sp]
```

\[
\begin{align*}
\{ x := v, \ldots \} \\
x <=> \text{tmp} \\
call alloc\_tmp \\
\{ x := 0, \ldots \}
\end{align*}
\]
• New modularity:
  – Code sharing by call and uncall

• Call-uncall (Garbage collection)
  – Local Bennett’s method [Bennett 1973]:

```plaintext
procedure main_fwd
    n += 4
    call fib

procedure main_bwd
    x1 += 5
    x2 += 8
    uncall fib

procedure fib

same procedure!
(code sharing)

call f
// copy the result of f
uncall f
```
Two Approaches to Inversion of Program

- Inverse Interpretation:

  proc \rightarrow \text{out}  
  \text{UNCALL proc} \rightarrow \text{in}

- Program Inversion:

  proc \rightarrow \text{out}  
  \text{Inverter} \rightarrow \text{CALL proc}^{-1} \rightarrow \text{in}

In Janus, any statements have its inverse.
procedure rfft(int re, int im, int N)
local int k=0, int j=0, int m=1
from m=1 // Stage of butterflies
do from k=0
do from j=0
  do re[k+j+m] += lift1(k,j,m,im) // Butterfly:
    // 1. lifting
    im[k+j+m] += lift2(k,j,m,re)
    re[k+j+m] += lift1(k,j,m,im)
    call flipswap(k,j,m,re,im)
    +=(re[k+j],re[k+j+m])
    +=(im[k+j],im[k+j+m])
  j += 1
until j=m
j ^= m
k += m*2
until k=N
k ^= N
call double(m)
until m=N
delocal int k=0, int j=0, int m=N
Computational Kernel

**Ordinary butterfly**

Add and subtract real and imaginary parts of a and b.

**Reversible butterfly**

Reversible updates!

Lifting

[OraintaraChenNguyen02]
Concluding Remarks

• As any computation model does, reversible computation model itself is theoretically of interest.

• **Formalized** reversible language Janus.
  – Janus: the first reversible language suggested for a class at Caltech [Lutz 1986].

• **Proved** that Janus is reversible.

• Explored the connection between program inversion and reversible computing.

• Demonstrated the practical and nontrivial reversible programs
  – fast Fourier transform

• **Shown** the computational strength of the language by implementing a **reversible Turing machine interpreter**.
Related Work: History of (Clean) Reversible High-level Programming Languages

- **Janus** [Lutz and Derby 1982]
  - The *first* reversible language. Imperative.

- **psiLisp** [Baker 1992]
  - The reversible Lisp-like functional language w/destructive updates.

- **R** [Frank 1997]
  - R compiler generates PISA code, which runs on the reversible processor Pendulum [Vieri 1999].

- **Inv** [Mu, Hu, and Takeichi 2004]
  - An injective *functional* language.

- **Gries’ invertible language** [Gries 1981]
  - Locally invertible CFOs
References


