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
        x2 += 1
    else
        n -= 1
        call fib
    fi

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

procedure fib

procedure forward

procedure backward

same procedure!
(code sharing)

[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

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

Remark: Circles are assertions

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

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

Remark: Circles are assertions
Local Inversion of CFOs

Local inversion

Conditional (IF)

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

Loop (DO-UNTIL-LOOP)

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

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

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

\[\Rightarrow \text{ Conditional and Loop are reversible.}\]
Skip and Sequence

\[
\sigma \vdash_{stmt} s_1 \Rightarrow \sigma' \quad \sigma' \vdash_{stmt} s_2 \Rightarrow \sigma'' \quad \text{Seq} \quad \sigma \vdash_{stmt} \text{skip} \Rightarrow \sigma \\
\Rightarrow \text{Skip and sequence are reversible.}
\]
Procedure Call / Uncall

\[
\frac{\sigma \vdash_{stmt} \Gamma(id) \Rightarrow \sigma'}{\sigma \vdash_{stmt} \text{call } id \Rightarrow \sigma'} \quad \text{Call}
\]

\[
\frac{\sigma' \vdash_{stmt} \Gamma(id) \Rightarrow \sigma}{\sigma \vdash_{stmt} \text{uncall } id \Rightarrow \sigma'} \quad \text{Uncall}
\]

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

⇒ Procedure call / uncall is reversible.
C-like Assignments

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

\[
\begin{align*}
\text{Judgment:} \quad \sigma \vdash_{\text{expr}} e \Rightarrow v \\
\text{Store} \quad \text{Exp} \quad \text{Val}
\end{align*}
\]

\[
\begin{align*}
\sigma \vdash_{\text{expr}} c \Rightarrow \llbracket c \rrbracket \quad \text{Con} \\
\sigma \vdash_{\text{expr}} x \Rightarrow \sigma(x) \quad \text{Var}
\end{align*}
\]

\[
\begin{align*}
\sigma \vdash_{\text{expr}} e_1 \Rightarrow v_1 \\
\sigma \vdash_{\text{expr}} e_2 \Rightarrow v_2 \\
\llbracket \odot \rrbracket(v_1, v_2) = v \\
\sigma \vdash_{\text{expr}} e_1 \odot e_2 \Rightarrow v \\
\odot \in \{+, -, \sim, \ldots\}
\end{align*}
\]

\[
\text{BinOp}
\]

\[
\begin{align*}
\text{Store } \sigma : \text{Var} \Rightarrow \text{Val}
\end{align*}
\]

⇒ Evaluation of expressions is \textit{fwd} deterministic. But it is not \textit{backward} deterministic.
Non-injective Binary Operators

• Some of the binary operators (others are similar)

\[
\llbracket + \rrbracket(v_1, v_2) = (v_1 + v_2) \mod 2^{32} \\
\llbracket - \rrbracket(v_1, v_2) = \begin{cases} 
0 & \text{if } v_1 \neq v_2 \\
1 & \text{if } v_1 = v_2
\end{cases}
\]

• 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

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

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

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

\( \Rightarrow \) It has an inverse function.

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

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

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

```plaintext
procedure main
    ... RTM, tape and constants decl. and init.
    from q=QS
    do call inst(q,left,s,right,q1,s1,s2,q2
        pc += 1
        if pc=PC_MAX then
            pc ^= PC_MAX
        fi pc=0
    until q=QF

procedure pushtape
    if empty(stk) && (s=BLANK) then
        s ^= BLANK // zero-clear s
    else
        push(s,stk)
    fi empty(stk)

procedure inst
    if q=q1[pc] then
        if s=s1[pc] then
            q += q2[pc]-q1[pc] // set q to q2[pc]
            s += s2[pc]-s1[pc] // set s to s2[pc]
        else
            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
                else
                    uncall pushtape(s,right) // pop right to s
                fi
            else
                if s2[pc]=LEFT then
                    call pushtape(s,right) // push s on right
                    uncall pushtape(s,left) // pop left to s
                fi
            fi
        fi
    fi
fi
```
• Assignment:
  – Zero-cleared copying, Zero-clearing by a constant

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

• Garbage manipulation:
  – Temporary stack

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

```
x <=> tmp
x <=> tmp
{ x := 0, ... } 
```

• New modularity:
  – Code sharing by call and uncall

procedure main_fwd
  n += 4
  call fib

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

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

    call f
    // copy the result of f
    uncall f

same
procedure!
(code sharing)
Two Approaches to Inversion of Program

- Inverse Interpretation:

- Program Inversion:

In Janus, any statements have its inverse.
Reversible Integer FFT (radix-2) [CF08]

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 \( j=0 \)
      \( \text{do} \)
      \( \text{re}[k+j+m] += \text{lift1}(k,j,m,im) \)
      \( \text{im}[k+j+m] += \text{lift2}(k,j,m,re) \)
      \( \text{re}[k+j+m] += \text{lift1}(k,j,m,im) \)
      \( \text{call \ flipped}(k,j,m,\text{re},im) \)
      \( +=(\text{re}[k+j],\text{re}[k+j+m]) \)
      \( +=(\text{im}[k+j],\text{im}[k+j+m]) \)
    \( j += 1 \)
  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

// Butterfly:
// 1. lifting

// 2. addsub

Computational Kernel
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

• Yokoyama T., Glück R., A reversible programming language and its invertible self-interpreter. In: Partial Evaluation and Program Manipulation. Proceedings. 144-153, ACM Press 2007.

• Axelsen H.B., Glück R., Yokoyama T., Reversible machine code and its abstract processor architecture. In: Diekert V., et al. (eds.), Computer Science - Theory and Applications. Lecture Notes in Computer Science, Vol. 4649, 56-69, Springer-Verlag 2007.

• Yokoyama T., Axelsen H.B., Glück R., Principles of a reversible programming language. In: Conference on Computing Frontiers. Proceedings. 43-54, ACM Press 2008.

• Yokoyama T., Axelsen H.B., Glück R., Reversible Flowchart Languages and the Structured Reversible Program Theorem. In: International Colloquium on Automata, Languages and Programming. Lecture Notes in Computer Science, Vol. 5126, pp. 258-270, Springer-Verlag, 2008.