Invited Tutorial at RC 2009

Reversible Computation and Reversible Programming Languages

Tetsuo Yokoyama

Graduate School of Information Science
Nagoya University

Example: Fibonacci-Pairs

```
n x1 x2
```

global store (three integer variables, initially zero)

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

reversible procedure

test (n=0)

swap values of x1, x2 (x1 <=> x2)

assertion (x1=x2)

Forward & Backward Computation

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

procedure forward

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

procedure backward

```
procedure fib
```

same procedure!
(code sharing)

[ReillyFederighi65,LutzDerby82]

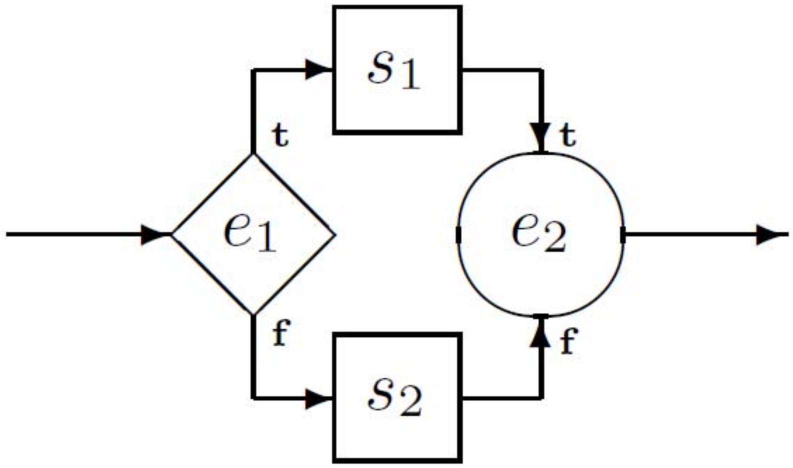
Janus: a Reversible Language

- 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

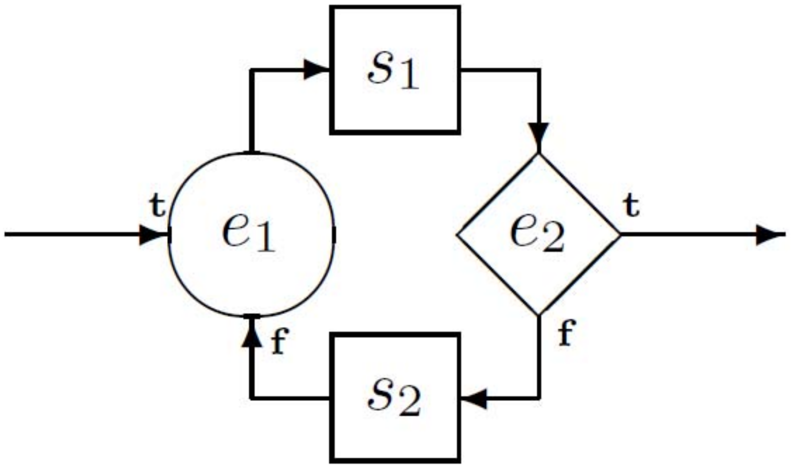
Syntax of Janus

$$
\begin{array}{rcll}
p & ::= & vdec^{*}\ (\texttt{procedure}\ id\ s)^{+} & \\
vdec & ::= & x \mid x\texttt{[}c\texttt{]} & \\
s & ::= & x\ \oplus\texttt{=}\ e \mid x\texttt{[}e\texttt{]}\ \oplus\texttt{=}\ e \mid & \text{Assignment operations} \\
 & & \texttt{if}\ e\ \texttt{then}\ s\ \texttt{else}\ s\ \texttt{fi}\ e \mid & \text{Reversible Conditional} \\
 & & \texttt{from}\ e\ \texttt{do}\ s\ \texttt{loop}\ s\ \texttt{until}\ e \mid & \text{Reversible Loop} \\
 & & \texttt{call}\ id \mid \texttt{uncall}\ id \mid \texttt{skip} \mid s\ s & \text{Procedure call/uncall} \\
e & ::= & c \mid x \mid x\texttt{[}e\texttt{]} \mid \texttt{\~{}}e \mid e \odot e & \\
c & ::= & \texttt{0} \mid \texttt{1} \mid \cdots \mid \texttt{4294967295} & \text{32-bit integers} \\
\oplus & ::= & \texttt{+} \mid \texttt{-} \mid \texttt{\^{}} & \\
\odot & ::= & \oplus \mid \texttt{*} \mid \texttt{/} \mid \texttt{\%} \mid \texttt{*/} \mid \texttt{\&} \mid \texttt{|} \mid \texttt{\&\&} \mid \texttt{||} \mid & \\
 & & \texttt{<} \mid \texttt{>} \mid \texttt{=} \mid \texttt{!=} \mid \texttt{<=} \mid \texttt{>=} &
\end{array}
$$

Control Flow Operators



(a) Conditional



(b) Loop

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**)

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

⇕

`if` e_2 `then` s_1^{-1} `else` s_2^{-1} `fi` e_1

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

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

⇕

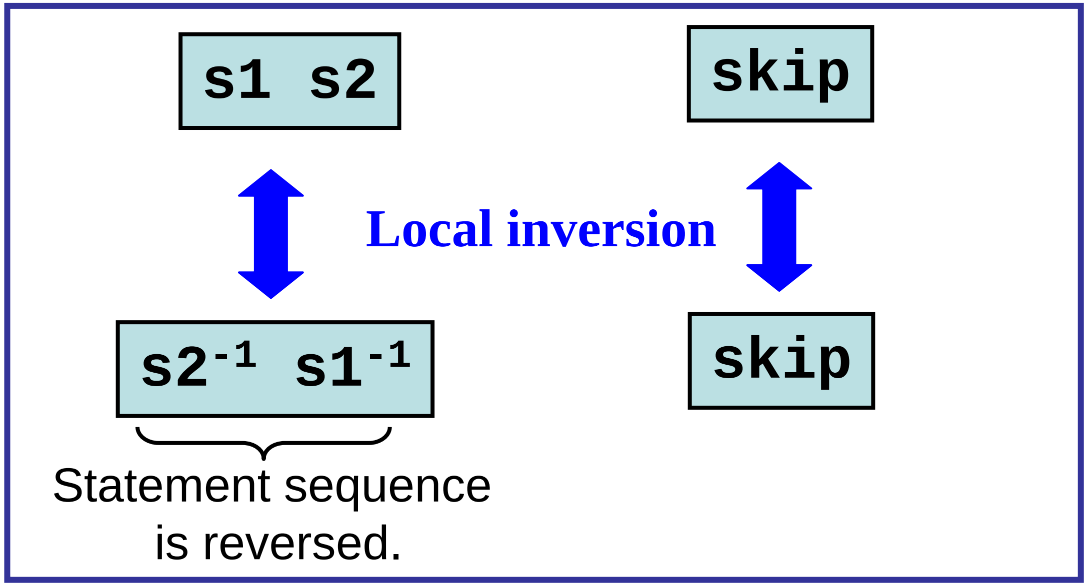
`from` e_2 `do` s_1^{-1} `loop` s_2^{-1} `until` e_1

Local inversion

⇒ Conditional and Loop are reversible.

Skip and Sequence

$$\frac{\sigma \vdash_{stmt} s_1 \Rightarrow \sigma' \quad \sigma' \vdash_{stmt} s_2 \Rightarrow \sigma''}{\sigma \vdash_{stmt} s_1\ s_2 \Rightarrow \sigma''}\ \mathsf{Seq} \qquad \frac{}{\sigma \vdash_{stmt} \texttt{skip} \Rightarrow \sigma}\ \mathsf{Skip}$$



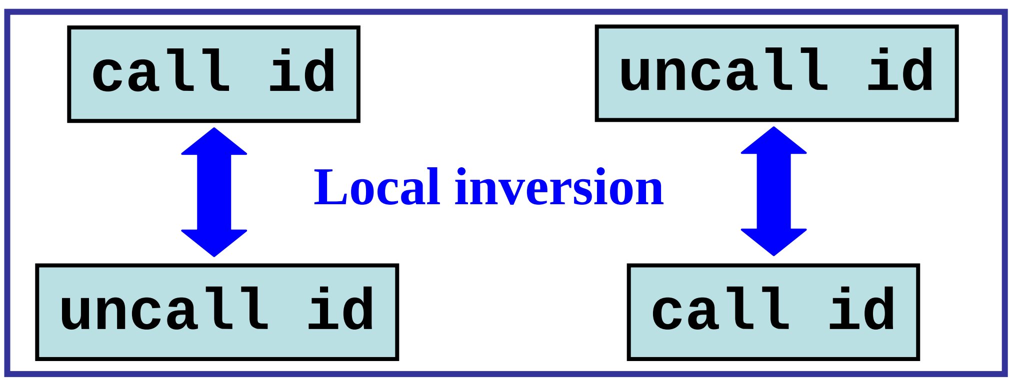
$\Rightarrow$ Skip and sequence are reversible.

Procedure Call / Uncall

$$\frac{\boxed{\sigma} \vdash_{stmt} \Gamma(id) \Rightarrow \boxed{\sigma'}}{\sigma \vdash_{stmt} \texttt{call}\ id \Rightarrow \sigma'} \quad \text{Call}$$

$$\frac{\boxed{\sigma'} \vdash_{stmt} \Gamma(id) \Rightarrow \boxed{\sigma}}{\sigma \vdash_{stmt} \texttt{uncall}\ id \Rightarrow \sigma'} \quad \text{Uncall}$$

$$\Gamma \in \text{Idens[Janus]} \rightharpoonup \text{Stmts[Janus]}$$



$\Rightarrow$ Procedure call / uncall is reversible.

C-like Assignments

x += e	**x -= e**	**x ^= e**
⇕	⇕	⇕ **Local inversion**
x -= e	**x += e**	**x ^= e**

Abbreviation: **x** ⊕**= e** ⇔ **x := x** ⊕ **e**
If variable **x** must not occur in expression **e**,
this is again an example of reversible update.

⇒ C-like Assignments are reversible.

Evaluation of Expressions

Judgment: $\sigma \vdash_{expr} e \Rightarrow v$

Store (σ), Exp (e), Val (v)

$$\frac{}{\sigma \vdash_{expr} c \Rightarrow [\![c]\!]}\ \textsf{Con} \qquad \frac{}{\sigma \vdash_{expr} x \Rightarrow \sigma(x)}\ \textsf{Var}$$

$$\frac{\sigma \vdash_{expr} e_1 \Rightarrow v_1 \quad \sigma \vdash_{expr} e_2 \Rightarrow v_2 \quad [\![\odot]\!](v_1, v_2) = v}{\sigma \vdash_{expr} e_1 \odot e_2 \Rightarrow v}\ \textsf{BinOp}$$

$$\odot \in \{+, -, \hat{}, \ldots\}$$

$$\textsf{Store}\ \sigma : \textsf{Var} \Rightarrow \textsf{Val}$$

$\Rightarrow$ Evaluation of expressions is fwd deterministic.
But it is not backward deterministic.

Non-injective Binary Operators

- Some of the binary operators (others are similar)

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

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

- 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)

Reversible Update

$$\frac{\sigma \vdash_{expr} e \Rightarrow v \qquad v_2 = [\![\oplus]\!](v_1, v)}{\sigma \uplus \{x \mapsto v_1\} \vdash_{stmt} x \oplus\!= e \Rightarrow \sigma \uplus \{x \mapsto v_2\}} \text{ AssVar}$$

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

$\Rightarrow$ It has an inverse function.

$\Rightarrow$ C-like Assignments are reversible.

Theorem: Janus Statements are Reversible

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

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 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                      // Symbol rule:
     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                // Shift rule:
       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
         fi s2[pc]=LEFT
       fi s2[pc]=RIGHT
     fi s1[pc]=SLASH
   fi s=s2[pc]
 fi q=q2[pc]
```

- Assignment:
  - Zero-cleared copying, Zero-clearing by a constant

```
{ x := 0, y := v }
x ^= y
{ x := v, y := v }
```

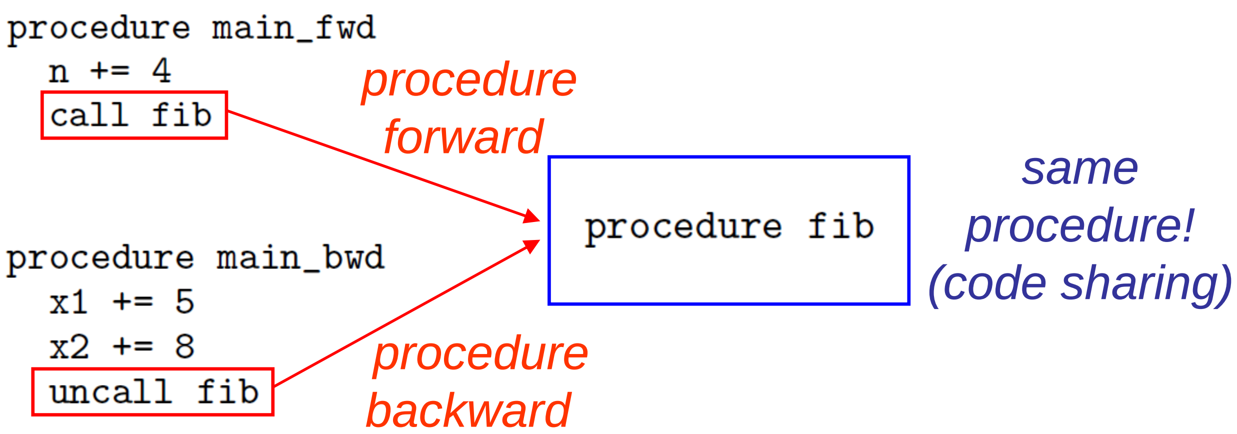
```
{ x := v, y := v }
x ^= y
{ x := 0, y := v }
```

- Garbage manipulation:
  - Temporary stack

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

```
{ x := v, … }
x <=> tmp
call alloc_tmp
{ x := 0, … }
```

- New modularity:
  - Code sharing by call and uncall

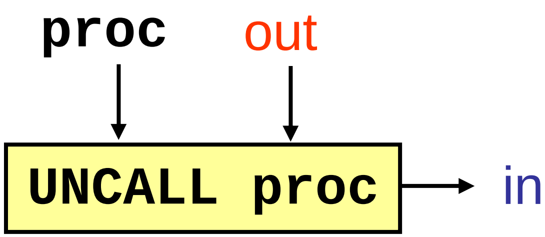


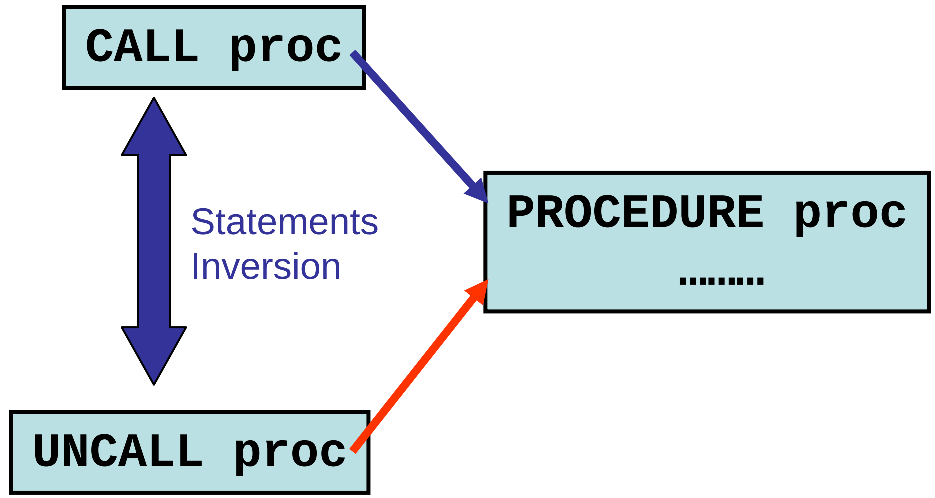
- Call-uncall (Garbage collection)
  - Local Bennett's method [Bennett 1973]:

```
call f
// copy the result of f
uncall f
```

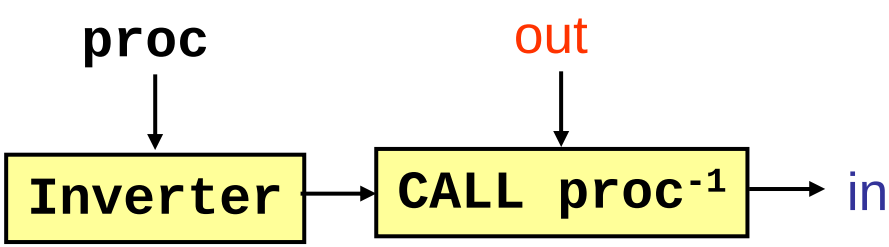
Two Approaches to Inversion of Program

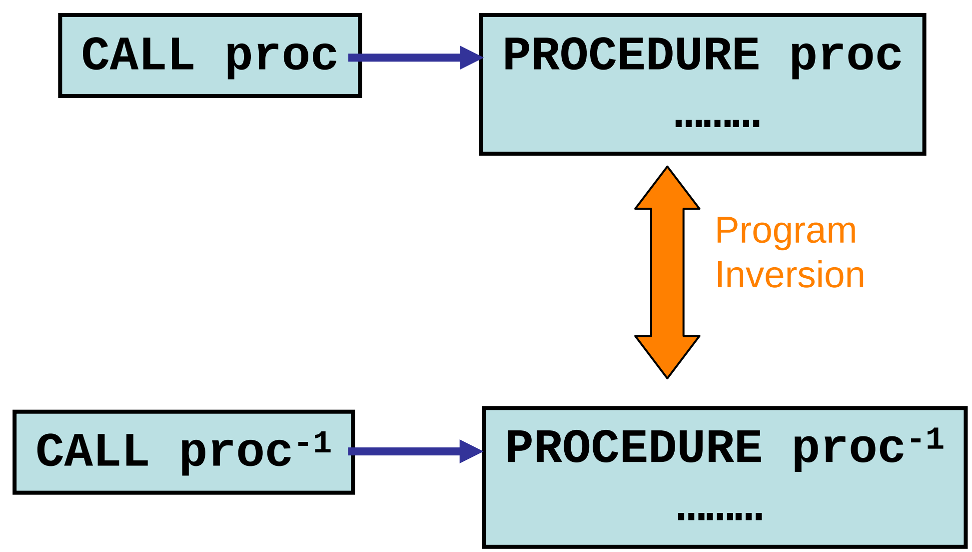
- Inverse Interpretation:





- Program Inversion:





In Janus, any statements have its inverse.

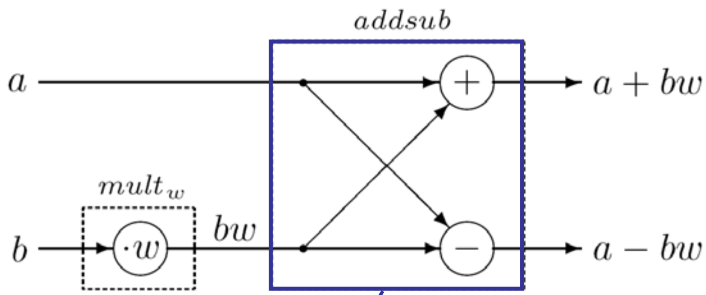
Reversible Integer FFT (radix-2) [CF08]

```
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                                   // Butterfly:
          do re[k+j+m] += lift1(k,j,m,im)            // 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])                  // 2. addsub
             +-=(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

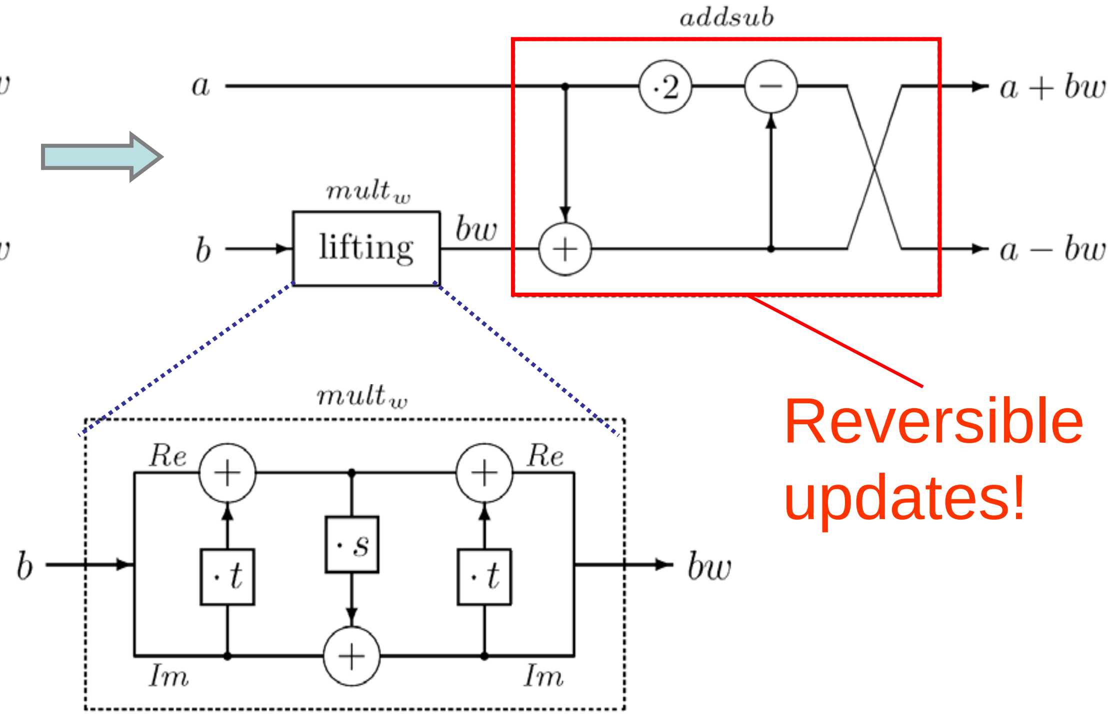
Computational Kernel

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.