Concepts of programming languages

Janus

Joris ten Tusscher, Joris Burgers, Ivo Gabe de Wolff, Cas van der Rest, Orestis Melkonian

Janus

A reversible programming language

Not turing complete!

Reversibility

Every statement can be reverted. No history is stored.

```
x += y * 3
```

Reversibility

Every statement can be reverted. No history is stored.

```
x += y * 3
```

```
x -= y * 3
```

Injective functions

Reversible languages can only compute injective functions.

$$\forall x, y : f(x) = f(y) \implies x = y \tag{1}$$

Every output has only a single input.

Injective functions

Reversible languages can only compute injective functions.

$$\forall x, y : f(x) = f(y) \implies x = y \tag{1}$$

Every output has only a single input.

$$h(x) = (x, g(x)) \tag{2}$$

Turing completeness

Turing machines can compute non-injective functions.

Reversible languages are not turing complete.

Reversible Turing complete.

Turing machines

Infinite tape of memory

Finite set of states

Transition function

- Current state
- Current symbol on tape
- Write symbol
- Move tape pointer
- Next state

Turing machines

Forward deterministic: given any state and tape, there is at most one transition *from* that state.

Backward deterministic: given any state and tape, there is at most one transition *to* that state.

P is the class of forward deterministic turing machines, NP of non-deterministic turing machines.

Reversible Turing complete: a language that can simulate forward and backward deterministic turing machines.

What do reversible languages compute

Given a forward deterministic turing machine that computes $f(x)$,

There exists a reversible turing machine that computes $x \to (x, f(x))$.

More memory.

Example

`fib`: calculates (n+1)-th and (n+2)-th Fibonacci number.

```
procedure fib
    if n = 0 then
        x1 += 1    ; -- 1st Fib nr is 1.
        x2 += 1    ; -- 2nd Fib nr is 1.
    else
        n -= 1
        call fib
        x1 += x2
        x1 <=> x2
    fi x1 = x2
```

Example

`fib`: calculates (n+1)-th and (n+2)-th Fibonacci number.

```
procedure fib
    if n = 0 then
        x1 += 1     ; -- 1st Fib nr is 1.
        x2 += 1     ; -- 2nd Fib nr is 1.
    else
        n -= 1
        call fib
        x1 += x2
        x1 <=> x2
    fi x1 = x2      ; -- Why do we need this?
```

Example

`fib`: calculates (n+1)-th and (n+2)-th Fibonacci number.

```
procedure fib
    if n = 0 then
        x1 += 1     ; -- 1st Fib nr is 1.
        x2 += 1     ; -- 2nd Fib nr is 1.
    else
        n -= 1
        call fib
        x1 += x2
        x1 <=> x2
    fi x1 = x2      ; -- Why do we need this?
```

- Q: How do we calculate the inverse?

Example

`fib`: calculates (n+1)-th and (n+2)-th Fibonacci number.

```
procedure fib
    if n = 0 then
        x1 += 1    ; -- 1st Fib nr is 1.
        x2 += 1    ; -- 2nd Fib nr is 1.
    else
        n -= 1
        call fib
        x1 += x2
        x1 <=> x2
    fi x1 = x2     ; -- Used for inverting the if-statement.
```

Q: How do we calculate the inverse?

$$\mathcal{I}[\![\texttt{if } e_1 \texttt{ then } s_1 \texttt{ else } s_2 \texttt{ fi } e_2]\!] \quad = \texttt{if } e_2 \texttt{ then } \mathcal{I}[\![s_1]\!] \texttt{ else } \mathcal{I}[\![s_2]\!] \texttt{ fi } e_1$$

Example

`fib`: calculates (n+1)-th and (n+2)-th Fibonacci number.

```
procedure fibInverse
    if x1 = x2 then
        x2 -= 1          ; -- 2nd Fib nr is 1.
        x1 -= 1          ; -- 1st Fib nr is 1.
    else
        x1 <=> x2
        x1 -= x2
        call fibInverse
        n += 1
    fi n = 0
```

Example

fib: calculates (n+1)-th and (n+2)-th Fibonacci number.

- Q: What does the inverse of fib do?

```
procedure fibInverse
    if x1 = x2 then
        x2 -= 1          ; -- 2nd Fib nr is 1.
        x1 -= 1          ; -- 1st Fib nr is 1.
    else
        x1 <=> x2
        x1 -= x2
        call fibInverse
        n += 1
    fi n = 0
```