

Strict Sequential Constructiveness

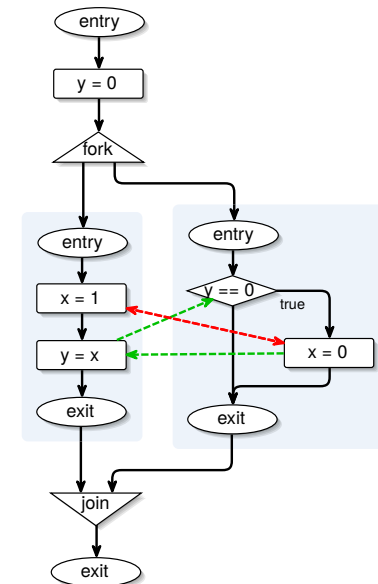
Alexander Schulz-Rosengarten
Reinhard von Hanxleden, Michael Mandler

Motivation: Program P10

```
1 module P10
2   int x, y;
3   {
4     y = 0;           //S1
5     fork
6       x = 1;        //S2
7       y = x         //S3
8     par
9       if y == 0 then //S4
10        x = 0        //S5
11      end
12    join
13  }
```

SC-admissible Schedule

S1 — S2 — S3 — S4

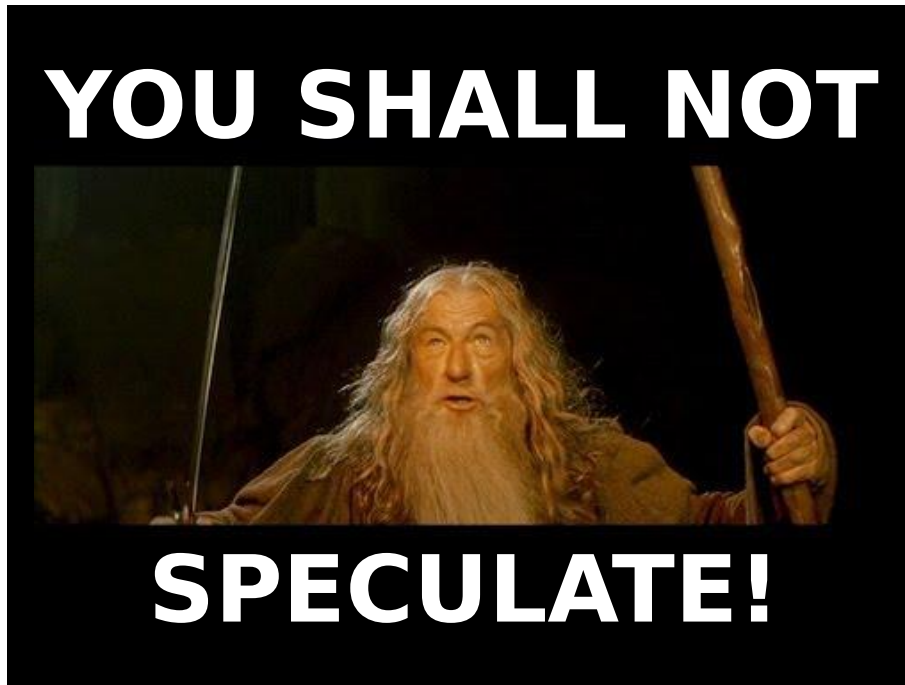


Problem

P10

- is reactive (\exists SC-admissible Run)
- is determinate (\forall SC-admissible Runs : same determinate macro responses)
- is Sequentially Constructive
- but is executed in a speculative manner

Why restricting sequential constructiveness?



Problem

P10:

- is reactive (\exists SC-admissible Run)
- is determinate (\forall SC-admissible Runs : same determinate macro responses)
- is Sequentially Constructive
- but is executed in a speculative manner

Problem

The SC MoC allows speculation

\Rightarrow SC programs may form non-constructive (delay sensitive) circuits

Restricting Sequential Constructiveness

Strict Sequential Constructiveness
is
Sequential Constructiveness without speculation

How can we eliminate speculation?

Restricting Sequential Constructiveness

Idea

Ground SC in constructiveness in the spirit of Esterel

Constructive Esterel:

- has no speculation
- always transforms into delay-insensitive (constructive) circuits

but

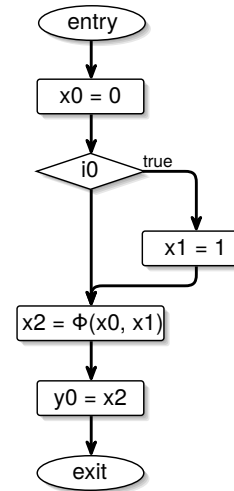
- requires globally consistent signal states
- has no shared variables (write & read)

Concept



- 1 Transformation into SSA form
 - ▶ sequential variable behavior
 - ▶ iur protocol
- 2 Translation into Esterel
 - ▶ signal encoding
 - ▶ SSA functions encoding
- 3 Esterel constructiveness check

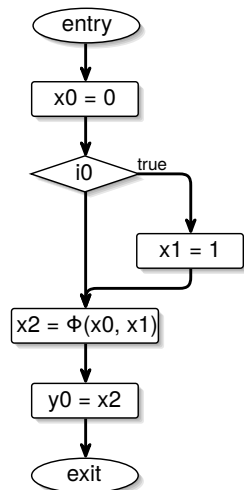
Static Single Assignment Form



- Each variable is assigned only once (statically)
- Only one reaching definition for each read (def-use-chains)
- Minimal placement of ϕ -nodes using a dominator analysis
- Intermediate representation based on a CFG

What about SCGs with concurrency?

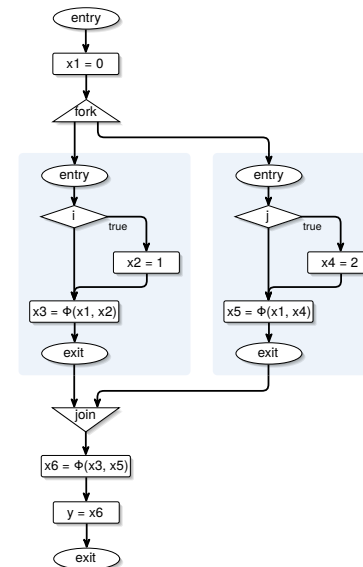
Static Single Assignment Form



Procedure

- 1 Split up variables into versions
- 2 Introduce ϕ -functions to merge variable versions

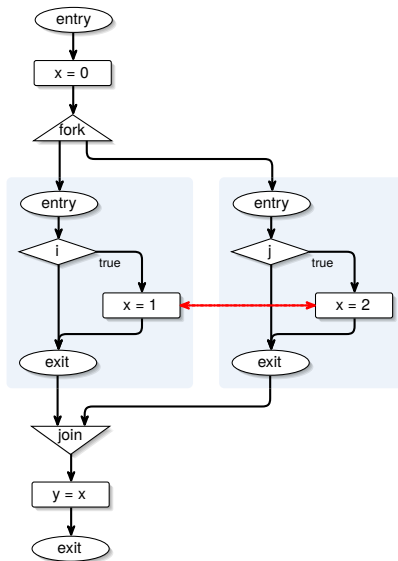
Static Single Assignment Form with Concurrency



Problem

ϕ -functions cannot handle concurrency

Static Single Assignment Form with Concurrency



i	j	y
false	false	0
false	true	2
true	false	1
true	true	reject

Static Single Assignment Form for SC Programs

SC-specific merge functions:

- Sequential override
- Encode concurrency
- Detect confluent writes
- Reject conflicting writes

Variable Representation: $\langle x^p, x \rangle$

Inspired by valued signals.

x^p : Presence signal

x : Actual variable value

```

1 || conc( $\langle x_i^p, x_i \rangle, \langle x_j^p, x_j \rangle$ ) :=
2 ||   present  $x_i^p$  then
3 ||     present  $x_j^p$  then
4 ||       if  $x_i == x_j$  then
5 ||         return  $\langle x_i^p, x_i \rangle$ 
6 ||       else
7 ||         reject
8 ||     else
9 ||       return  $\langle x_i^p, x_i \rangle$ 
10 ||  else
11 ||    present  $x_j^p$  then
12 ||      return  $\langle x_j^p, x_j \rangle$ 
13 ||  else
14 ||    return  $\langle absent, nil \rangle$ 

```

Static Single Assignment Form for SC Programs

SC-specific merge functions:

- Sequential override
- Encode concurrency
- Detect confluent writes
- Reject conflicting writes

Variable Representation: $\langle x^p, x \rangle$

Inspired by valued signals.

x^p : Presence signal

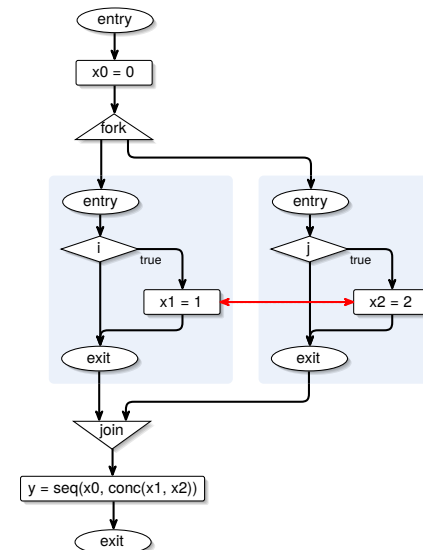
x : Actual variable value

```

1 || seq( $\langle x_i^p, x_i \rangle, \langle x_j^p, x_j \rangle$ ) :=
2 ||   present  $x_j^p$  then
3 ||     return  $\langle x_j^p, x_j \rangle$ 
4 ||   else
5 ||     present  $x_i^p$  then
6 ||       return  $\langle x_i^p, x_i \rangle$ 
7 ||     else
8 ||       return  $\langle absent, nil \rangle$ 

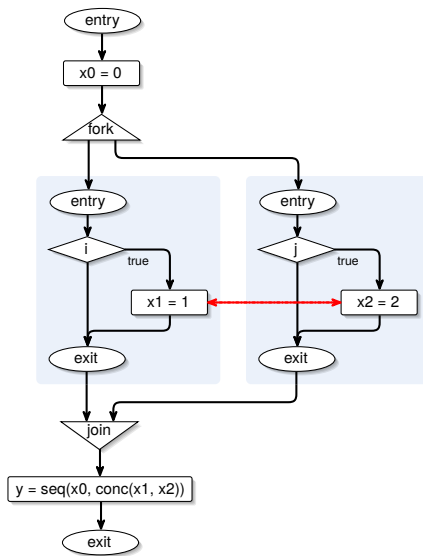
```

Static Single Assignment Form for SCGs



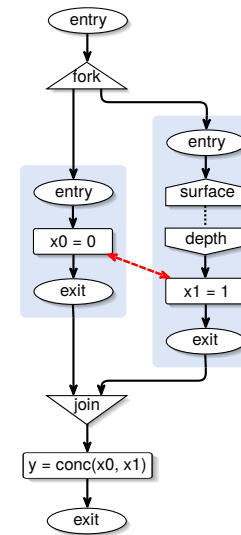
$seq(x_0, conc(x_1, x_2))$

Static Single Assignment Form for SCGs



i	j	y
false	false	0
false	true	2
true	false	1
true	true	reject

SSA Form: Delays



```

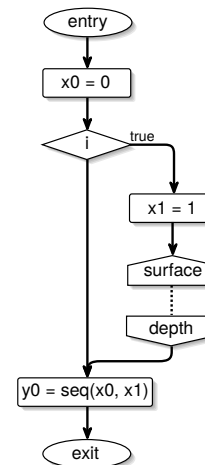
1  conc( $\langle x_i^p, x_i \rangle, \langle x_j^p, x_j \rangle$ ) :=
2  present  $x_i^p$  then
3    present  $x_j^p$  then
4      if  $x_i == x_j$  then
5        return  $\langle x_i^p, x_i \rangle$ 
6      else
7        reject
8    else
9      return  $\langle x_i^p, x_i \rangle$ 
10 else
11   present  $x_j^p$  then
12     return  $\langle x_j^p, x_j \rangle$ 
13   else
14     return  $\langle absent, nil \rangle$ 

```

SSA Form: Further Aspects

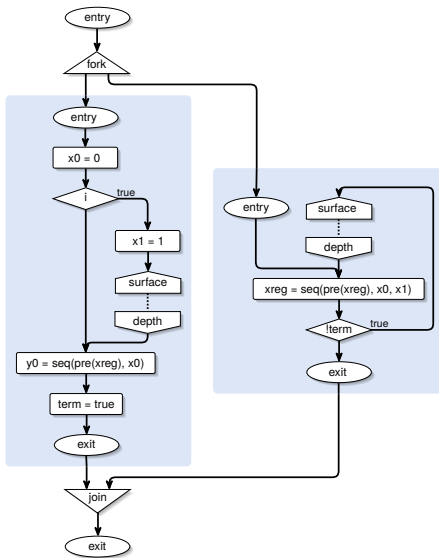
- Delays
 - Merge functions use a variable with signals and implicit reset
- Loops
 - Merge expressions require explicit sequential ordering
- Updates
 - iur protocol ordering
 - Confluent by definition
- Interface
 - SSA renaming

SSA Form: Delays



- Presence signals are reset to absent
- Runtime concurrent conflicts can be detected
- Merge function cannot resolve value without write in the same tick

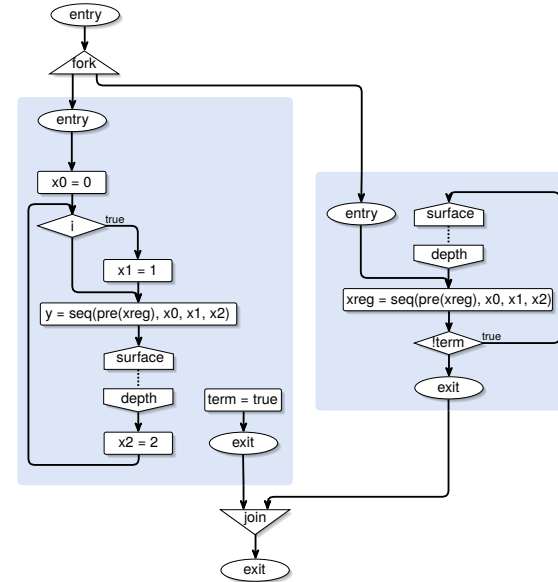
SSA Form: Delays Solved



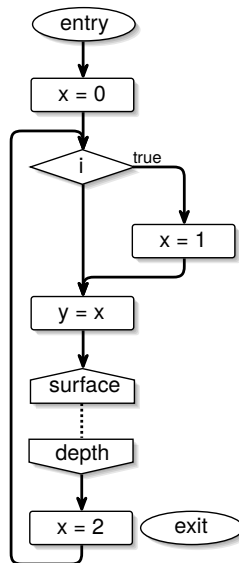
Solution:

- Resolve and save variable values in each tick
- Store values in register variables
- Use *pre* to consider values of the previous tick in merge expressions
- Reduce merge expression based on tick borders

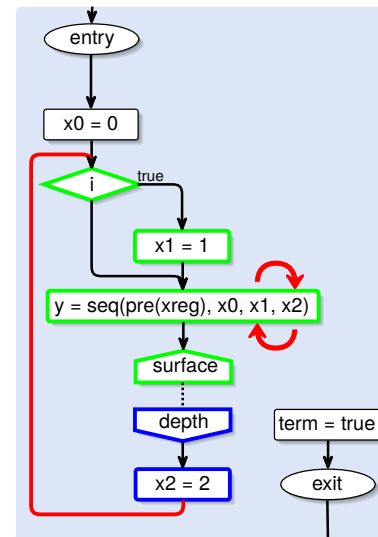
SSA Form: Loops



SSA Form: Loops



SSA Form: Loop Handling

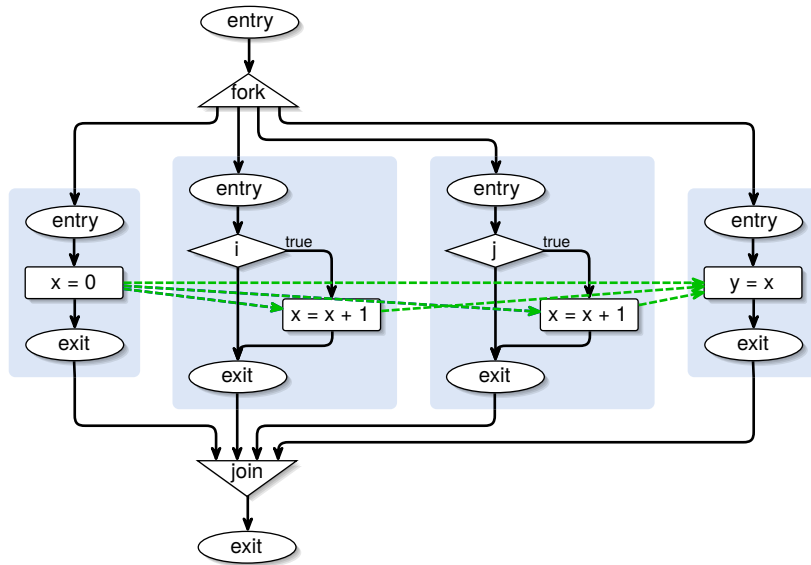


- Merge expressions require static ordering
- Wrong ordering due to simple structure analysis

Solution:

- Surface-Depth analysis
- Requires a pause that is always executed
- Switch order of writes in the surface with depth

SSA Form: Updates



SSA Form: Update Handling

$x = x + 1 \rightarrow x_{up} = 1$
 \downarrow
 $\text{combine}(+, x_{init}, x_{up})$

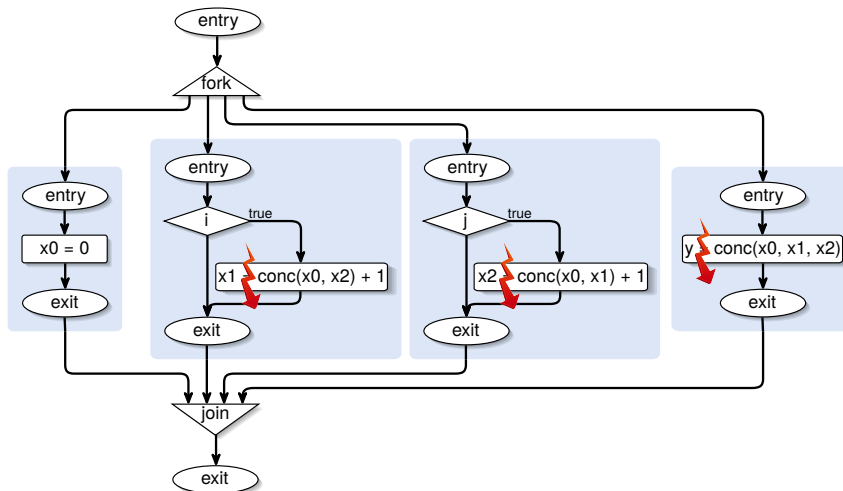
```

1 | combine( $f, \langle x^p, x \rangle, \langle x_{up}^p, x_{up} \rangle$ ) :=
2 |   present  $x^p$  then
3 |     present  $x_{up}^p$  then
4 |       return  $\langle x^p, f(x, x_{up}) \rangle$ 
5 |     else
6 |       return  $\langle x^p, x \rangle$ 
7 |   else
8 |     present  $x_{up}^p$  then
9 |       reject
10 |    else
11 |      return  $\langle absent, nil \rangle$ 

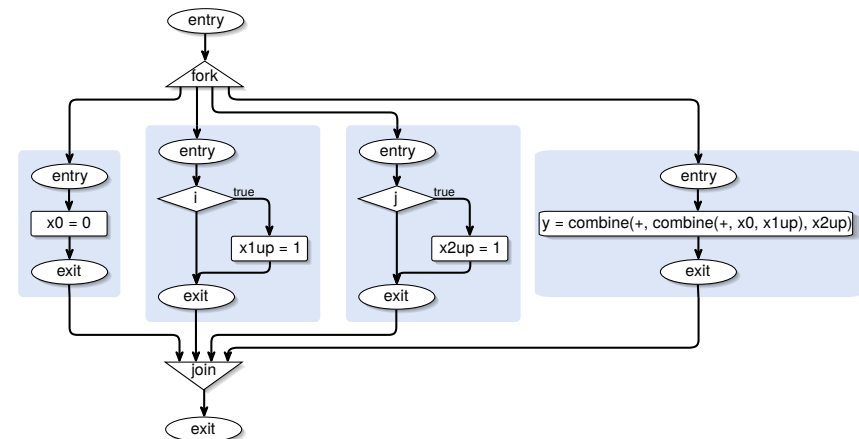
```

- Special seq function for updates
- Requires partial static schedule to generate merge expressions

SSA Form: Updates



SSA Form: Updates Solved



SSA Form: Interface

```

1 module IO
2 input int I;
3 output int O;
4 {
5   if I < 0 then
6     I = 0
7   end;
8   O = I;
9   pause;
10  O = O * I
11 }

```

- SSA renaming should not violate the original interface
- Inputs must be read from the environment in each tick
- Inputs can be locally overridden
- Outputs must be conveyed to the environment in each tick



SSA Form: Interface Solved

```

1 module IO-SSA
2 input int I;
3 int IO;
4 output int O;
5 int O0, O1, Oreg;
6 bool term = false;
7 {
8   fork
9     if I < 0 then
10      IO = 0
11    end;
12    O0 = seq(I, IO)
13    pause;
14    O1 = pre(Oreg) * I;
15    term = true
16  par
17    PauseLoop:
18      Oreg = seq(pre(Oreg), O0, O1);
19      O = Oreg;
20    if !term then
21      pause;
22      goto PauseLoop
23    end
24  join
25 }

```

Translation into Esterel

Translation of

- program structure
- variables

Esterel data-types:

- Variables
- Valued signals
- Pure signals

Presence Encoding

$x_i^p \setminus x_i$	present	absent
present	true	false
absent	undef	undef

Dual-Rail Encoding

$x_i \setminus \text{not_}x_i$	present	absent
present	illegal	true
absent	false	undef

Dual-Rail Encoding

```

xi = true      →      emit xi

xi = false    →      emit not_xi

xi = e        →
    present errorExpr(e) then
        emit error
    else
        [
            present trueExpr(e) then
                emit xi
            end
            ||
            present falseExpr(e) then
                emit not_xi
            end
        ]
    end

```

Dual-Rail Encoding

```

xi:
  trueExpr: xi
  falseExpr: not_xi

conc(ei, ej):
  errorExpr: (trueExpr(ei) ∧ falseExpr(ej)) ∨
              (falseExpr(ei) ∧ trueExpr(ej))
  trueExpr: trueExpr(ei) ∨ trueExpr(ej)
  falseExpr: falseExpr(ei) ∨ falseExpr(ej)

seq(ei, ej):
  trueExpr: trueExpr(ej) ∨ (¬falseExpr(ej) ∧ trueExpr(ei))
  falseExpr: falseExpr(ej) ∨ (¬trueExpr(ej) ∧ falseExpr(ei))

```

Dual-Rail Encoding

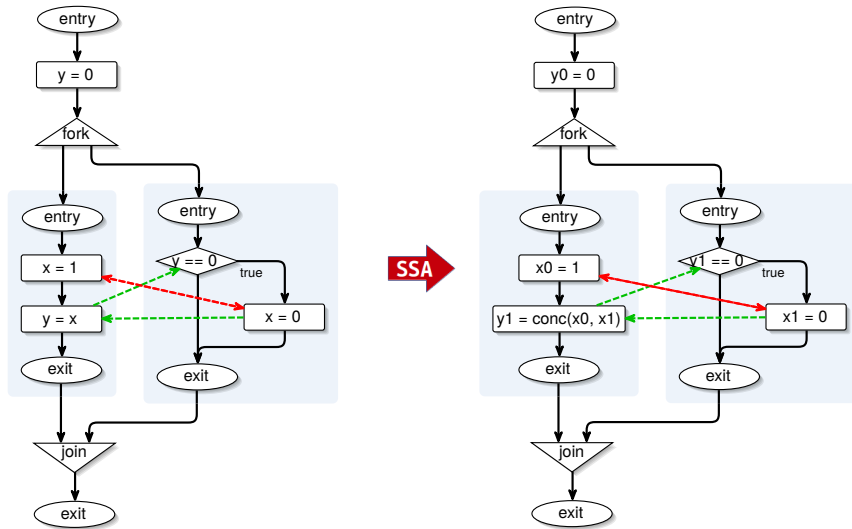
```

if (e) then
  //then-block
else
  //else-block
end
→
    present errorExpr(e) then
        emit error
    else
        [
            present trueExpr(e) then
                % then-block
            end
            ||
            present falseExpr(e) then
                % else-block
            end
        ]
    end

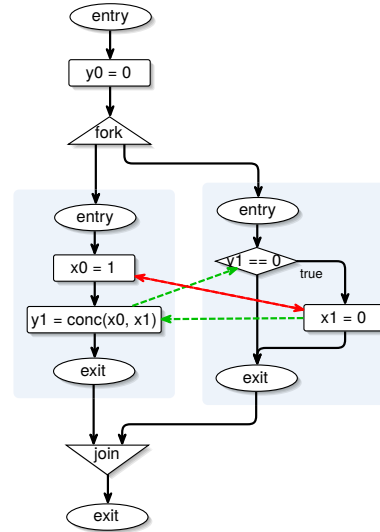
```



Back to P10

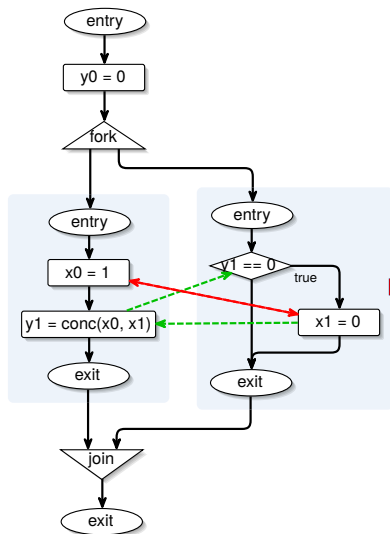


Back to P10



Not constructive in Esterel
 ⇒ not Strict SC

Back to P10

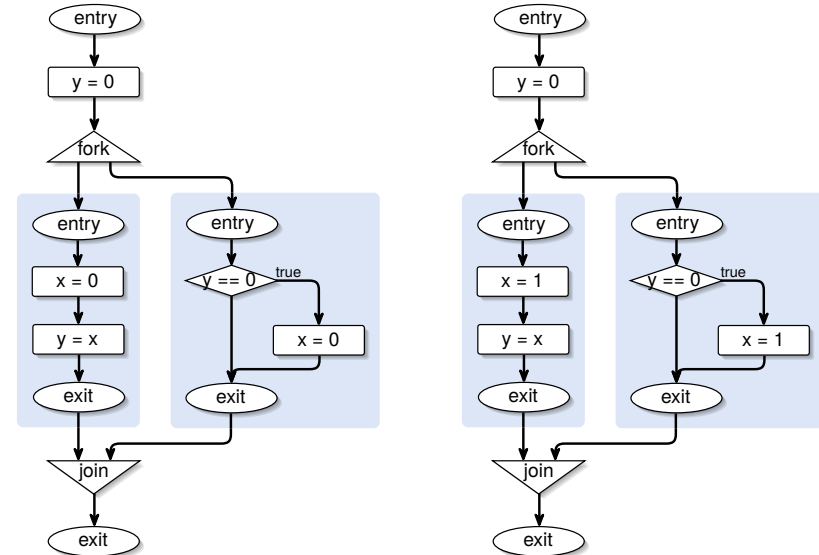


Esterel

```

1 module P10:
2   signal x0, not_x0, x1, not_x1 in
3   signal y0, not_y0, y1, not_y1 in
4   signal error in
5   [
6     emit not_y0;
7     [
8       emit x0;
9       present (x0 and not_x1) or
10        (not_x0 and x1) then
11         emit error
12       else
13         [
14           present x0 or x1 then
15             emit y1
16           end
17         ||
18         present not_x0 or not_x1 then
19           emit not_y1
20         end
21       end
22     ||
23     present not_y1 then
24       emit not_x1
25     end
26   ]
27 ||
28   signal err in
29   present err then
30     present err else emit err end
31   end
32   end signal
    
```

Confluent P10 Variants



Confluent P10 Variants

```
1 | [
2 |   emit not_y0;
3 |   [
4 |     emit not_x0;
5 |     present (x0 and not_x1) or (
6 |       not_x0 and x1) then
7 |       emit error
8 |     else
9 |       [
10 |        present x0 or x1 then
11 |          emit y1
12 |        end
13 |        ||
14 |        present not_x0 or not_x1 then
15 |          emit not_y1
16 |        end
17 |      ]
18 |    ||
19 |    present not_y1 then
20 |      emit not_x1
21 |    end
22 |  ]
23 | ||
24 | signal err in
25 | present error then
26 |   present err else emit err end
27 | end
28 | end signal
29 | ]
```

Strict Sequential Constructiveness

40 / 42

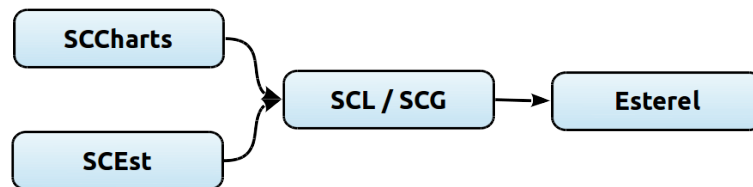
Future Work

- Optimized translation for SCEst
- Code optimization based on SSA
- Loop unrolling for (bounded) instantaneous loops
- Dynamic scheduling of updates
- Reduction merge expression insertion

Strict Sequential Constructiveness

42 / 42

New Compile Chain



Strict Sequential Constructiveness

41 / 42