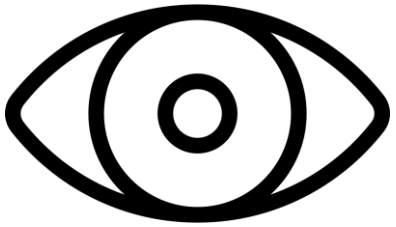
From Lustre to Graphical Models and SCCharts

Lena Grimm

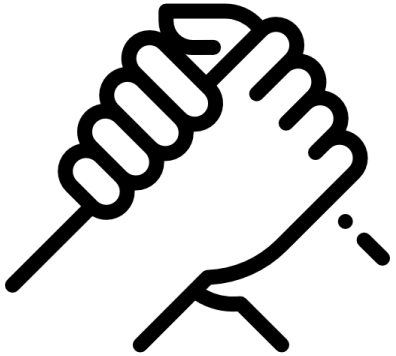
Kiel University

Motivation



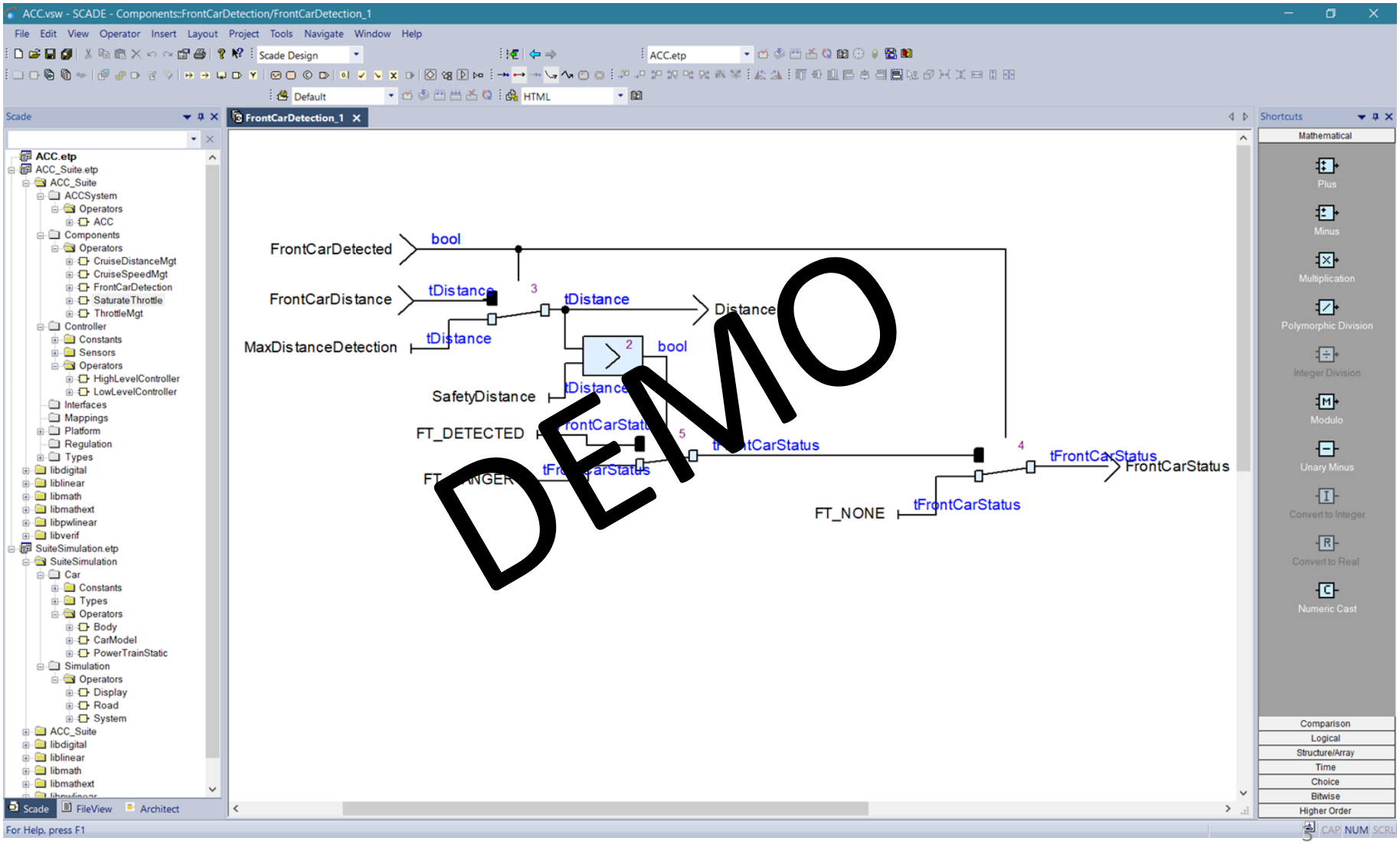
Graphical Model

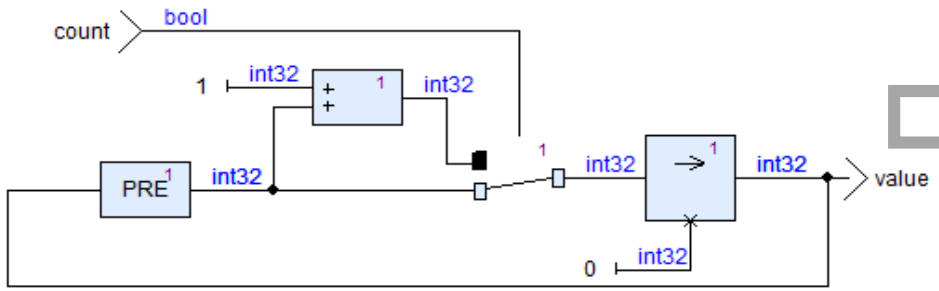
Validation with SCCharts



Sequential Constructiveness

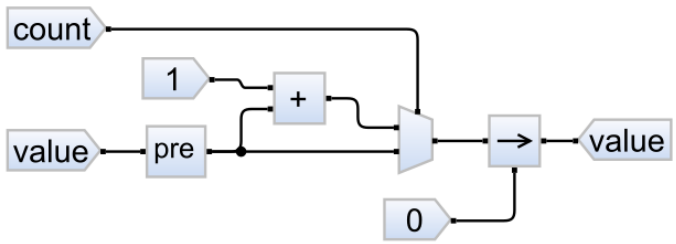
SCADE


ACC.vsw - SCADE - Components::FrontCarDetection/FrontCarDetection_1
File Edit View Operator Insert Layout Project Tools Navigate Window Help
Scade Design
ACC.etp
Default
HTML
Scade
FrontCarDetection_1
Shortcuts
Mathematical
Plus
Minus
Multiplication
Polymorphic Division
Integer Division
Modulo
Unary Minus
Convert to Integer
Convert to Real
Numeric Cast
Comparison
Logical
Structure/Array
Time
Choice
Bitwise
Higher Order
ACC.etp
ACC_Suite.etp
ACC_Suite
ACCSystem
Operators
ACC
Components
Operators
CruiseDistanceMgt
CruiseSpeedMgt
FrontCarDetection
SaturateThrottle
ThrottleMgt
Controller
Constants
Sensors
Operators
HighLevelController
LowLevelController
Interfaces
Mappings
Platform
Regulation
Types
libdigital
liblinear
libmath
libmathext
libpwlinear
libverif
SuiteSimulation.etp
SuiteSimulation
Car
Constants
Types
Operators
Body
CarModel
PowerTrainStatic
Simulation
Operators
Display
Road
System
ACC_Suite
libdigital
liblinear
libmath
libmathext
Scade
FileView
Architect
FrontCarDetected
bool
FrontCarDistance
tDistance
3
tDistance
Distance
MaxDistanceDetection
tDistance
2
bool
SafetyDistance
tDistance
FT_DETECTED
5
4
tFrontCarStatus
FrontCarStatus
FT_NONE
tFrontCarStatus
DEMO
For Help, press F1
CAP NUM SCRL

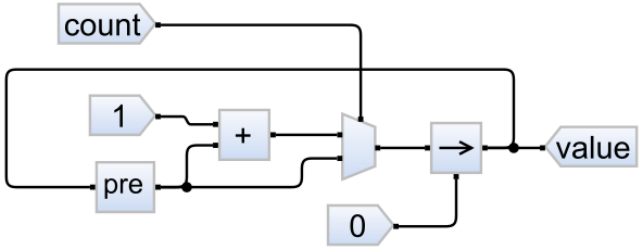



```
L1 = count;
value = L7;
L2 = 1;
L3 = pre L7;
L4 = L2 + L3;
L5 = if L1 then (L4) else (L3);
L6 = 0;
L7 = (L6) -> (L5);
```

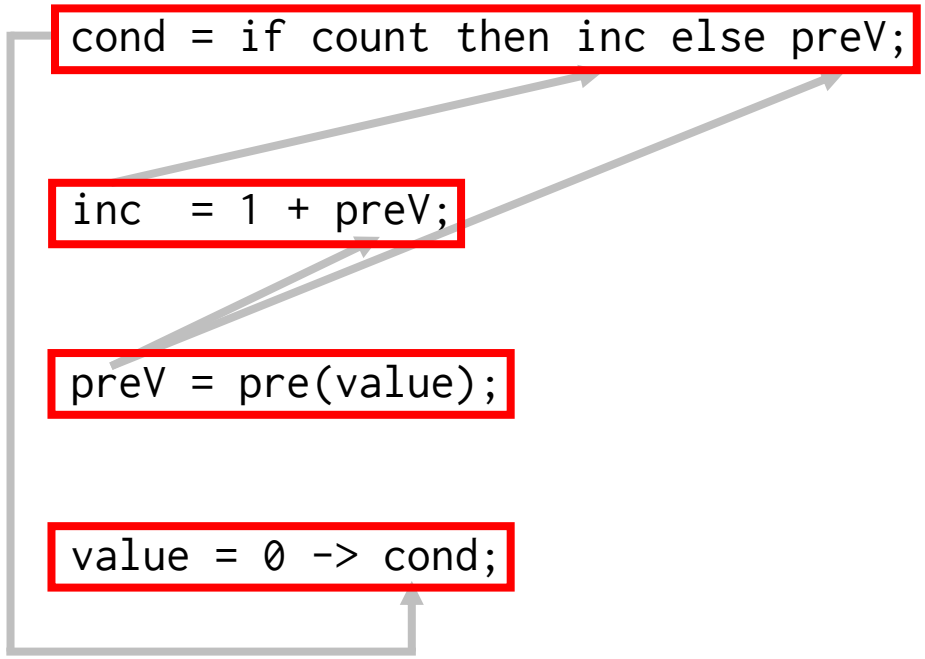
```
node Counter (count:bool)
  returns (value:int)
var L1:bool;
L2, L3, L4, L5, L6, L7:int;
let
  L1 = count;
  value = L7;
  L2 = 1;
  L3 = pre L7;
  L4 = L2 + L3;
  L5 = if L1 then (L4) else (L3);
  L6 = 0;
  L7 = (L6) -> (L5);
tel
```

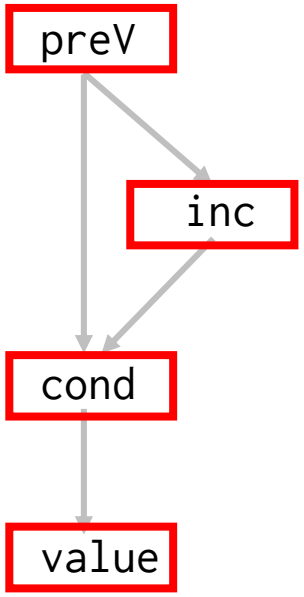


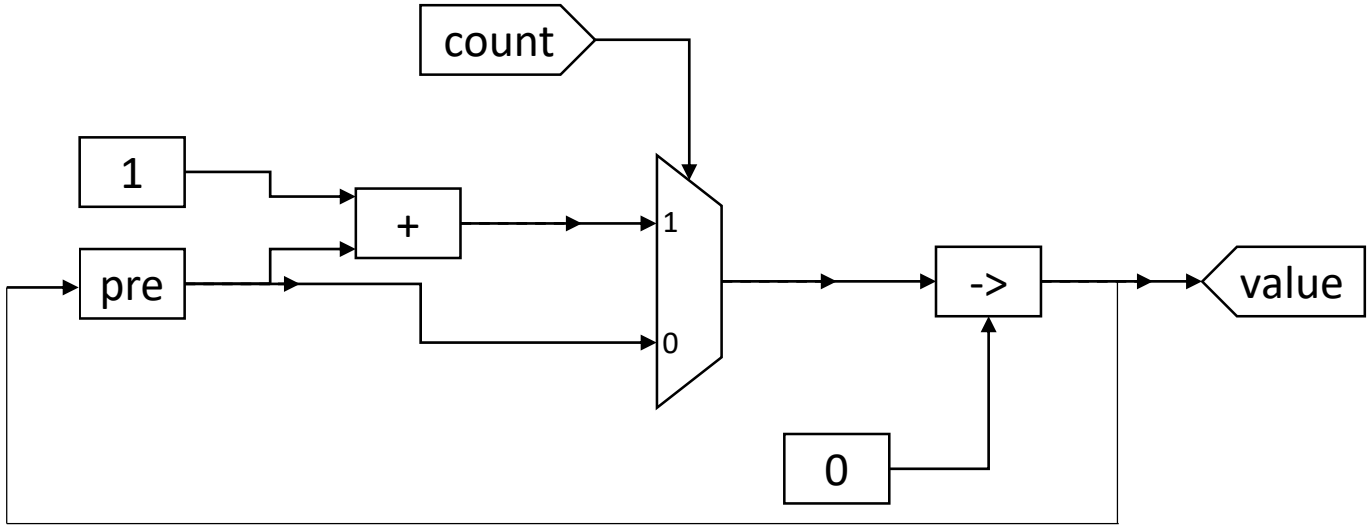
GENERATED



Input: count:bool **Output: value:int**



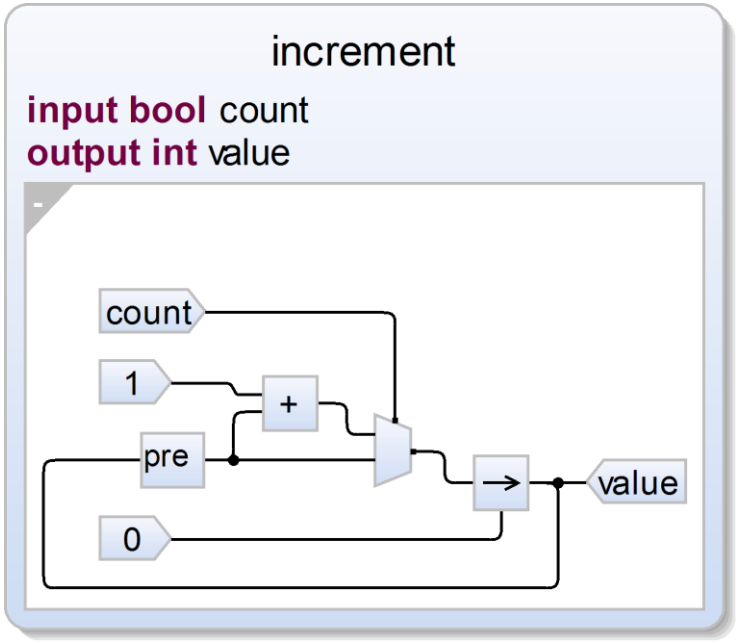




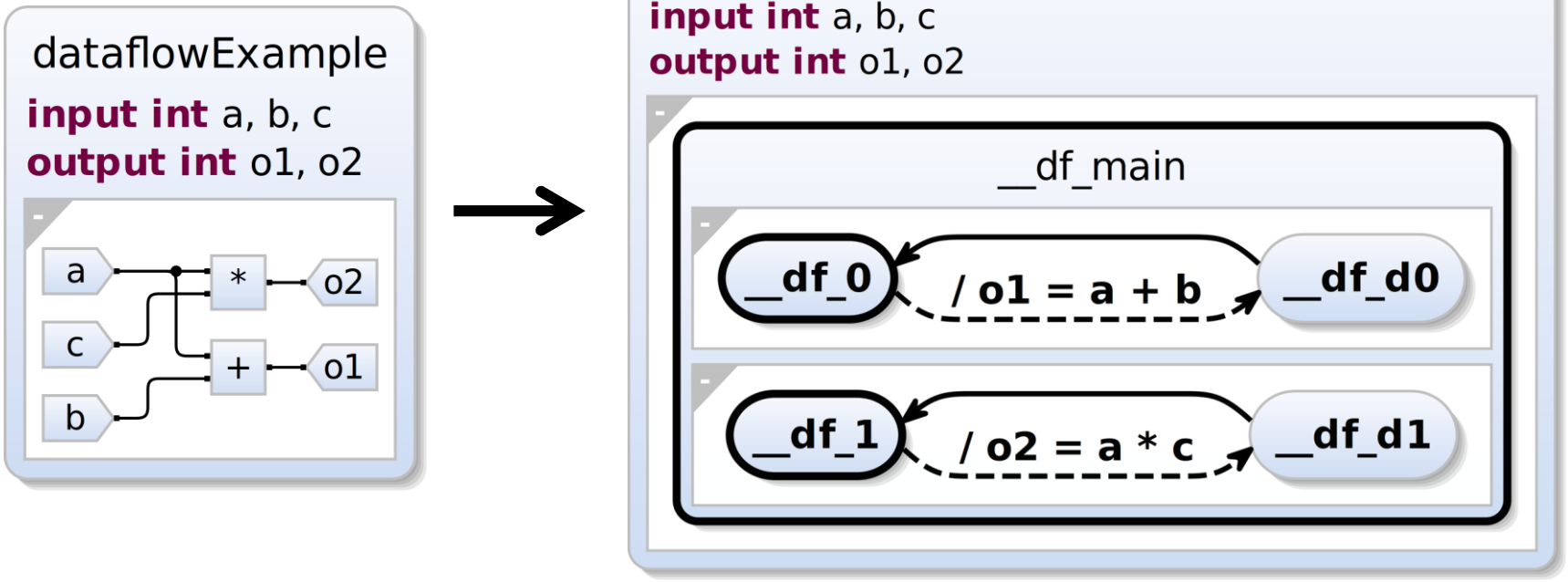
KIELER Demo

```
node increment (count:bool) returns (value:int)
var cond:int;
    inc:int;
    preV:int;

let
  cond = if count then inc else preV;
  inc  = 1 + preV;
  preV = pre(value);
  value = 0 -> cond;
tel
```

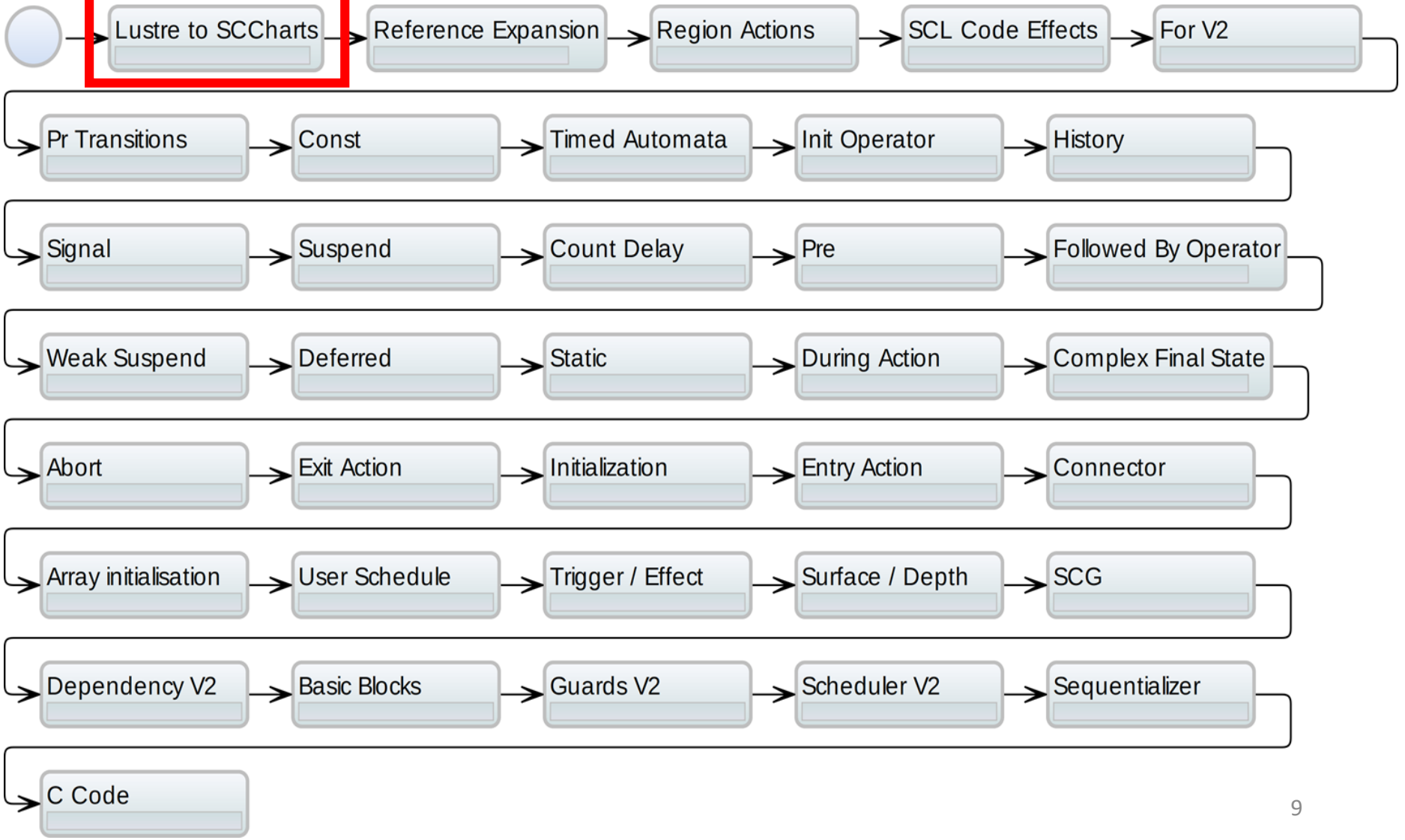
```
scchart increment {
  input bool count
  output int value

  dataflow {
    int cond, inc, preV

    cond = count ? inc : preV
    inc = 1 + preV
    preV = pre(value)
    value = 0 -> cond
  }
}
```



SCCharts Dataflow Semantics



KIELER Compilation Chain



Lustre Clock Calculus

Lustre

x	1	2	3	4	5	6	7	8	9
clk	true	false	true	false	false	true	false	true	true
x when clk	1		3			6		8	9

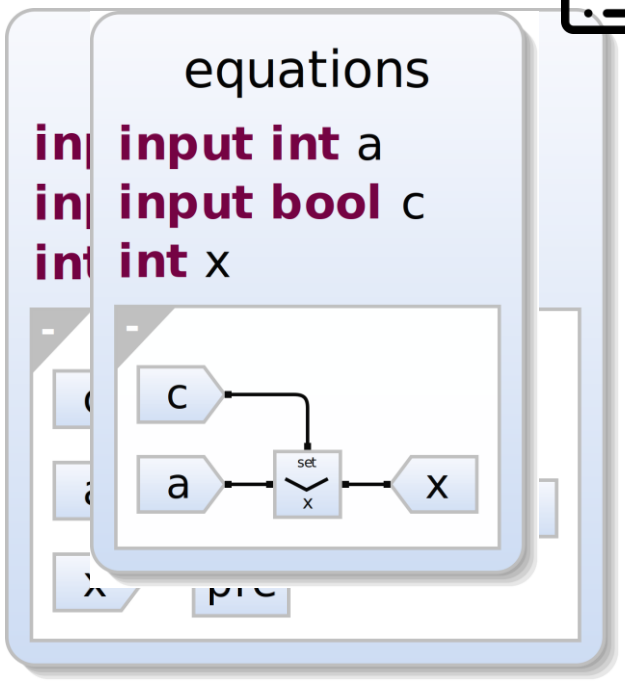
when, current?

SCCharts

x	1	2	3	4	5	6	7	8	9
clk	true	false	true	false	false	true	false	true	true
clk? x	1	1	3	3	3	6	6	8	9

When Operator

```
node equations(a:int; c:bool)
  returns ();

var x:int when c;
let
  x = a when c;
tel.
```




```
scchart equations {
  input int a
  input bool c
  int x
  dataflow {
    x = c ? a : pre(x)
  }
}
```

Lustre When Operator

clk	true	false	true	false	true	true	false	false	true
x	true	false	false	true	true	false	false	false	true
y	true	false	false	true	false	false	true	true	false

xClk = x when clk	true		false		true	false			true
y when xClk	true				false				false

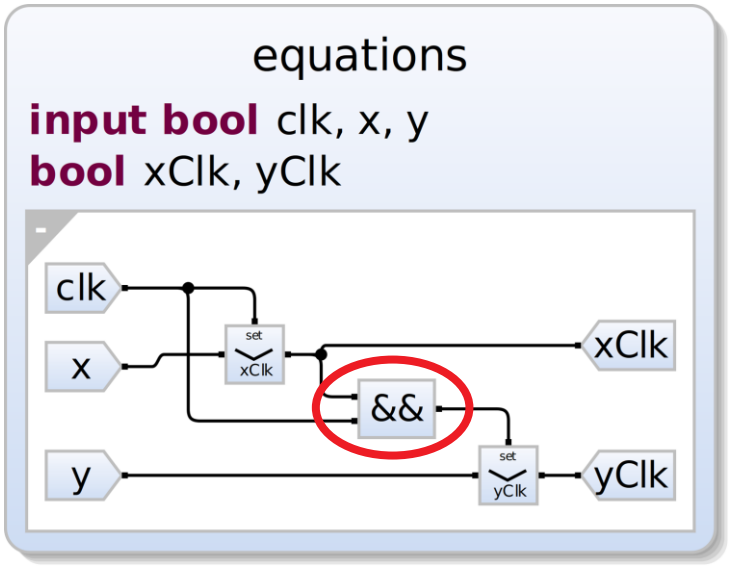
When Operator with Variables I

clk	true	false	true	false	true	true	false	false	true
x	true	false	false	true	true	false	false	false	true
y	true	false	false	true	false	false	true	true	false

xClk = clk? x	true	true	false	false	true	false	false	false	true
xClk? y	true	**false**	false	false	false	false	false	false	false

Hierarchical When

```
node equations(clk,x,y:bool)
  returns ();

var xClk:bool when clk;
    yClk:bool when xClk;
let
  xClk = x when clk;
  yClk = y when xClk;
tel.
```




```
scchart equations {
  input bool clk, x, y
  bool xClk, yClk

  dataflow {
    xClk = clk ? x
    yClk = (xClk && clk) ? y
  }
}
```

When Operator with Variables II

clk	true	false	true	false	true	true	false	false	true
x	true	false	false	true	true	false	false	false	true
y	true	false	false	true	false	false	true	true	false

xClk = clk? x	true	true	false	false	true	false	false	false	true
(clk&&xClk)? y	true	true	false	false	false	false	false	false	false

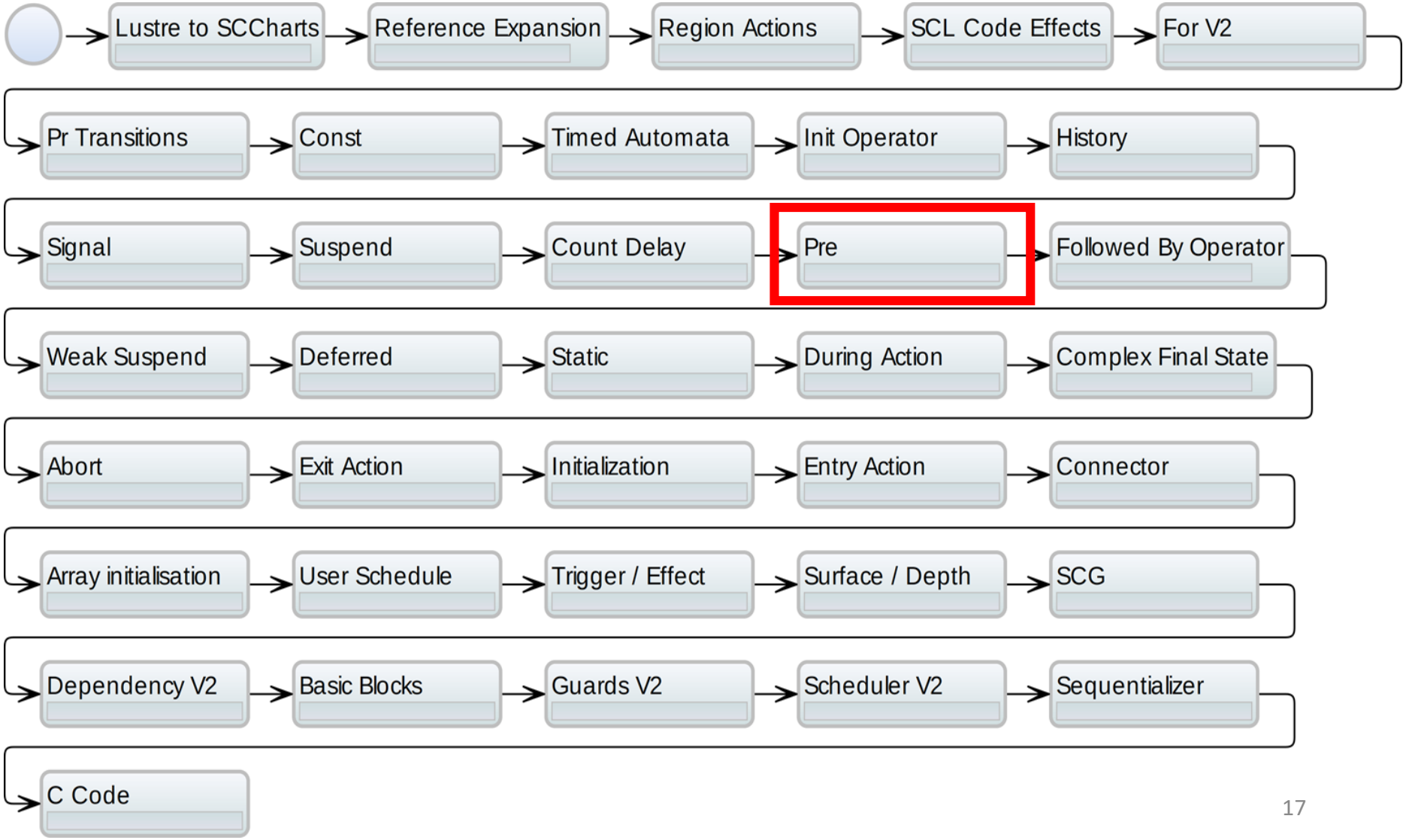
Current?

clk	true	false	true	false	true	true	false	false	true
x	true	false	false	true	true	false	false	false	true
y	true	false	false	true	false	false	true	true	false

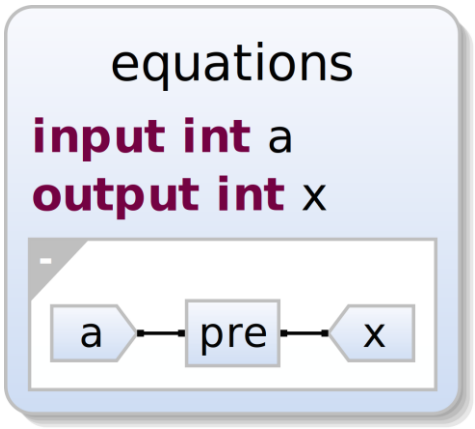
xClk = clk? x	true	true	false	false	true	false	false	false	true
(clk&&xClk)? y	true	true	false	false	false	false	false	false	false

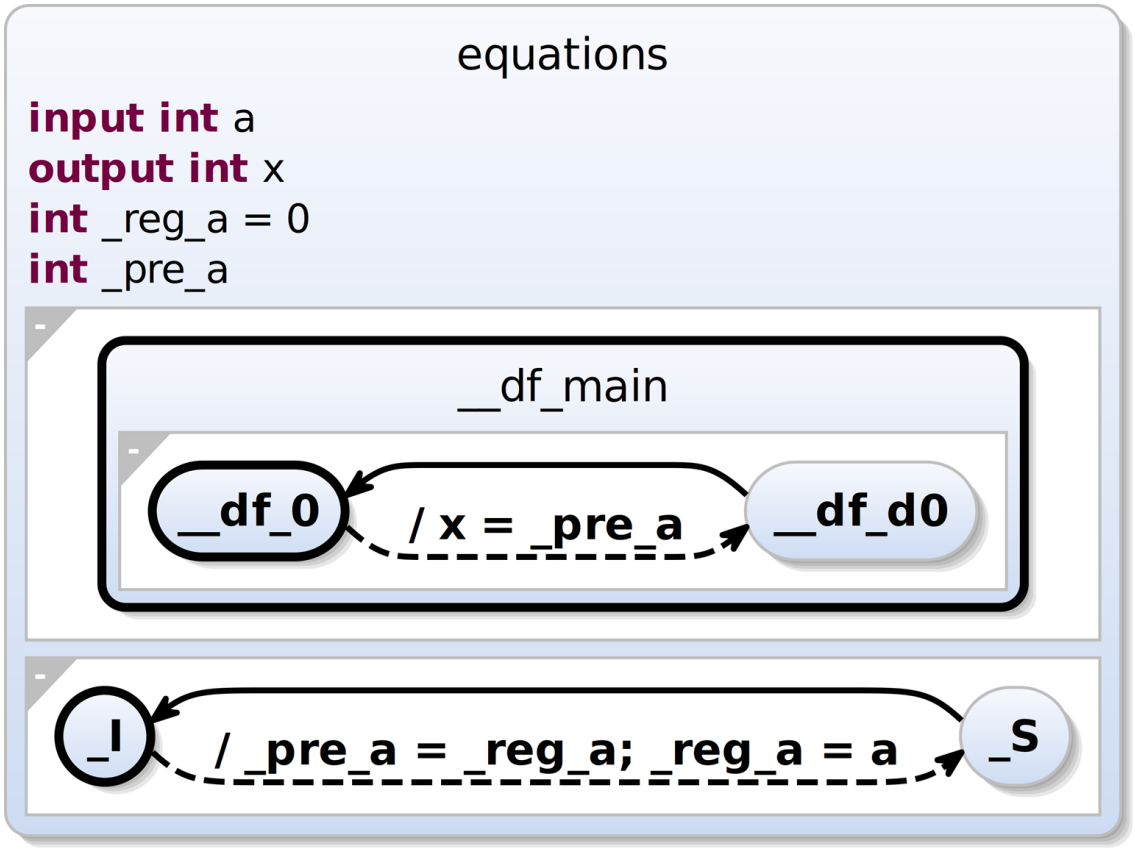
Always implicit *current* through variables

KIELER Compilation Chain



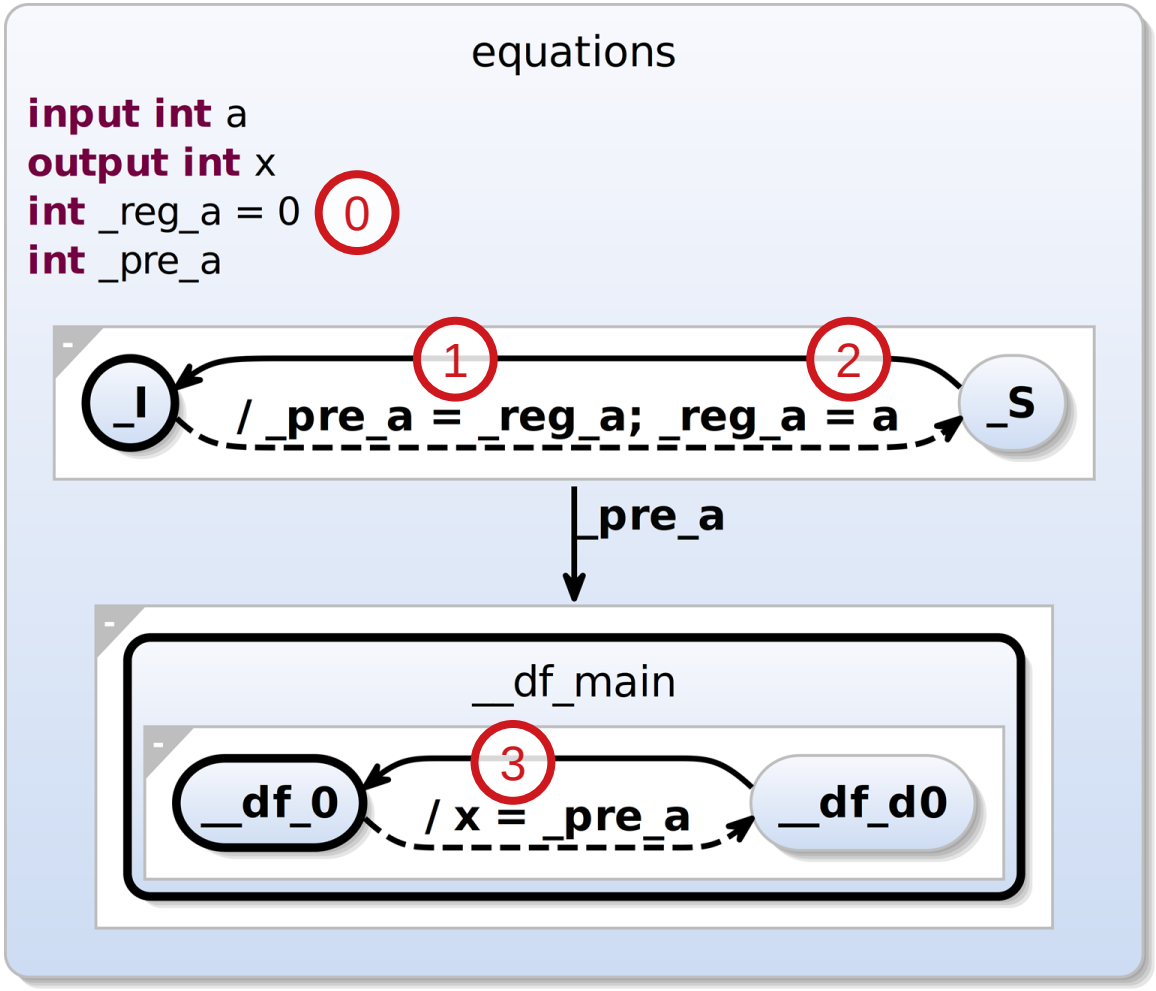
Pre Operator in SCCharts






```
scchart equations {
     input int a
     output int x

     dataflow {
          x = pre(a)
     }
}
```

Pre Operator in SCCharts Induced Dataflow View



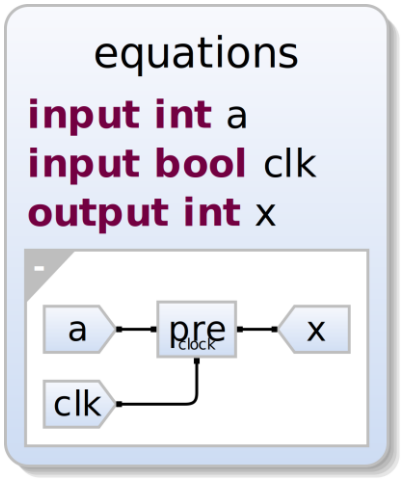
Lustre Pre Operator and Clocks

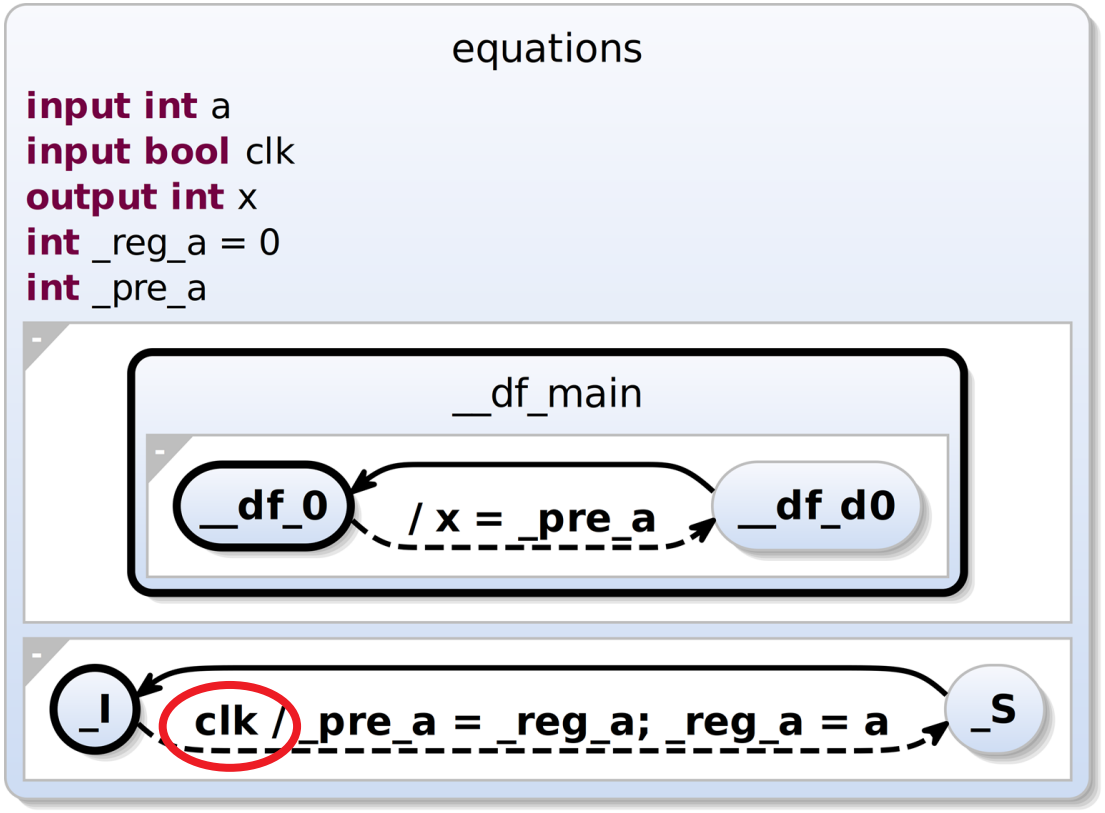
clk	true	false	false	true	true	false	true	false	true
x	1	2	3	4	5	6	7	8	9
xClk = x when clk	1			4	5		7		9
pxClk = pre(xClk)	nil			1	4		5		7
pre(pxClk)	nil			nil	1		4		5

Pre Operator and Clocks with Variables

clk	true	false	false	true	true	false	true	false	true
x	1	2	3	4	5	6	7	8	9
xClk = clk? x	1	1	1	4	5	5	7	7	9
pxClk = pre(xClk)	nil	1	1	1	4	5	5	7	7
pre(pxClk)	nil	nil	1	1 **nil**	1	4	5 **4**	5	7 **5**

Clocked Pre Operation in SCCharts





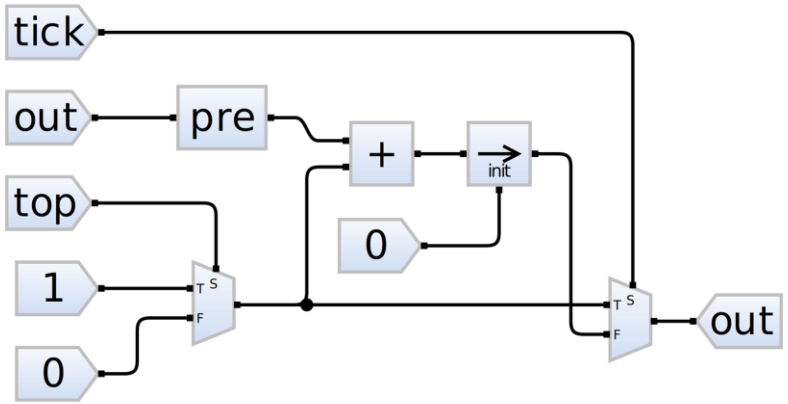

```
scchart equations {
     input int a
     input bool clk
     output int x

     dataflow {
          x = pre(a, clk)
     }
}
```

Clocked Pre Operator with Variables

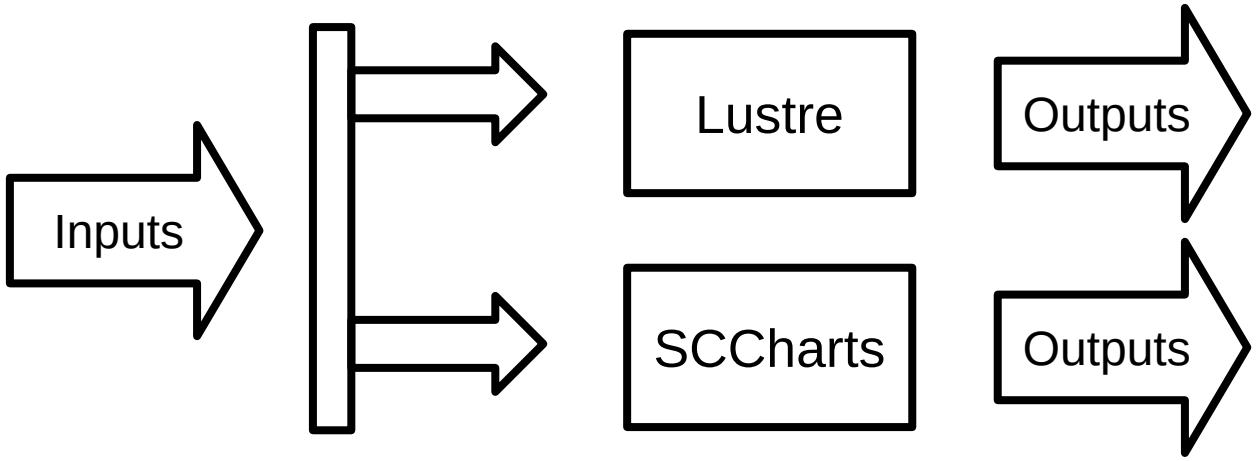
clk	true	false	false	true	true	false	true	false	true
x	1	2	3	4	5	6	7	8	9
xClk = clk? x	1	1	1	4	5	5	7	7	9
pxClk = pre(xClk)	nil	1	1	1	4	5	5	7	7
pre(pxClk)	nil	nil	1	nil	1	4	5	5	7

Model Recovery

```
node counting(tick:bool;top:bool)
    returns (out:int);
var v:int;
let
  v = if top then 1 else 0;
  out = if tick
        then v
        else (0 -> pre out + v);
tel.
```



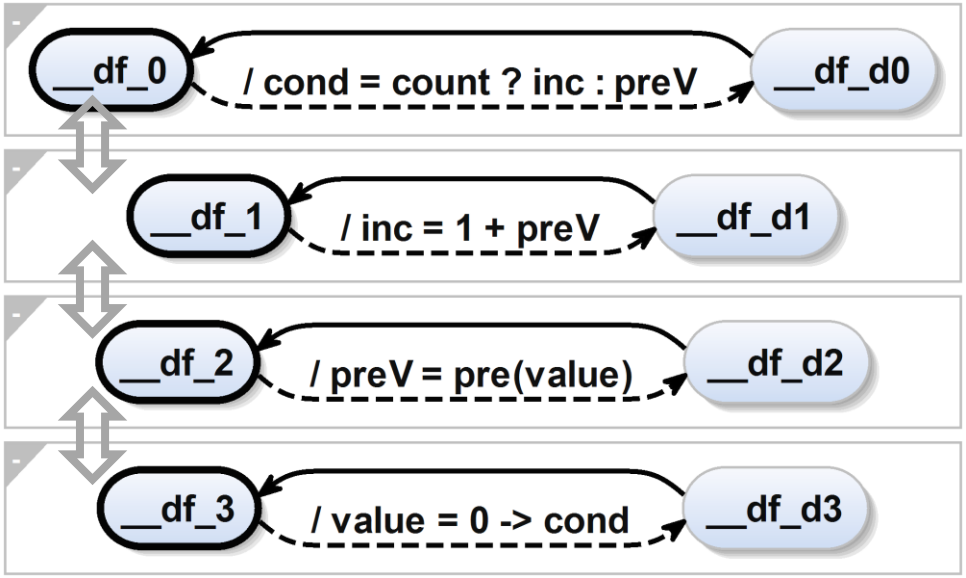
Behavior Preservation



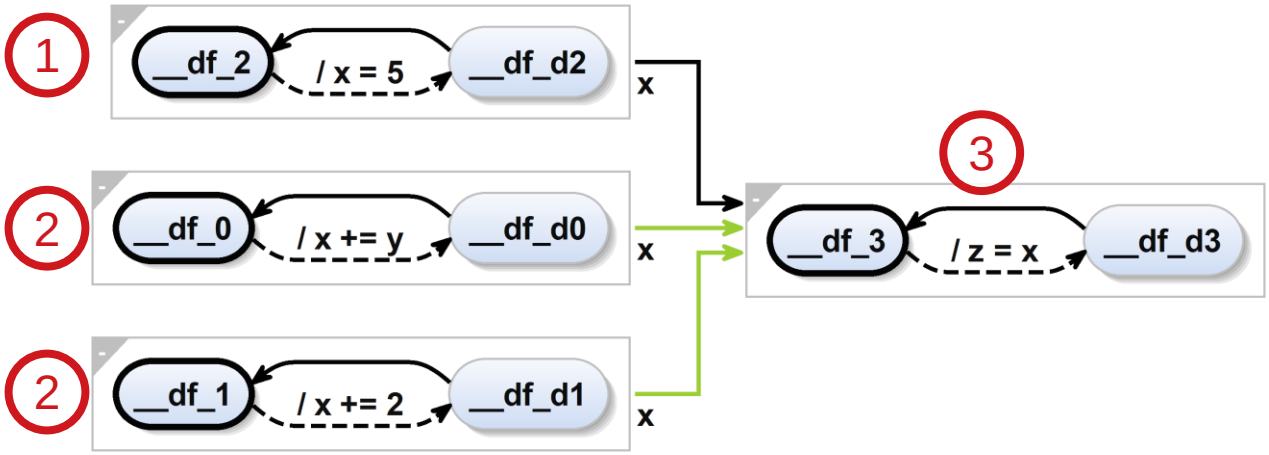
Sequential Constructiveness Concurrency

```
node increment (count:bool)
       returns (value:int)
var cond:int;
    inc:int;
    preV:int;

let
  cond = if count then inc else preV;
  inc  = 1 + preV;
  preV = pre(value);
  value = 0 -> cond;
tel
```

Initialize-Update-Read



```
…
   x += y;
   x += 2;
   x =  5;
   z =  x;
…
```



Conditioned Updates:

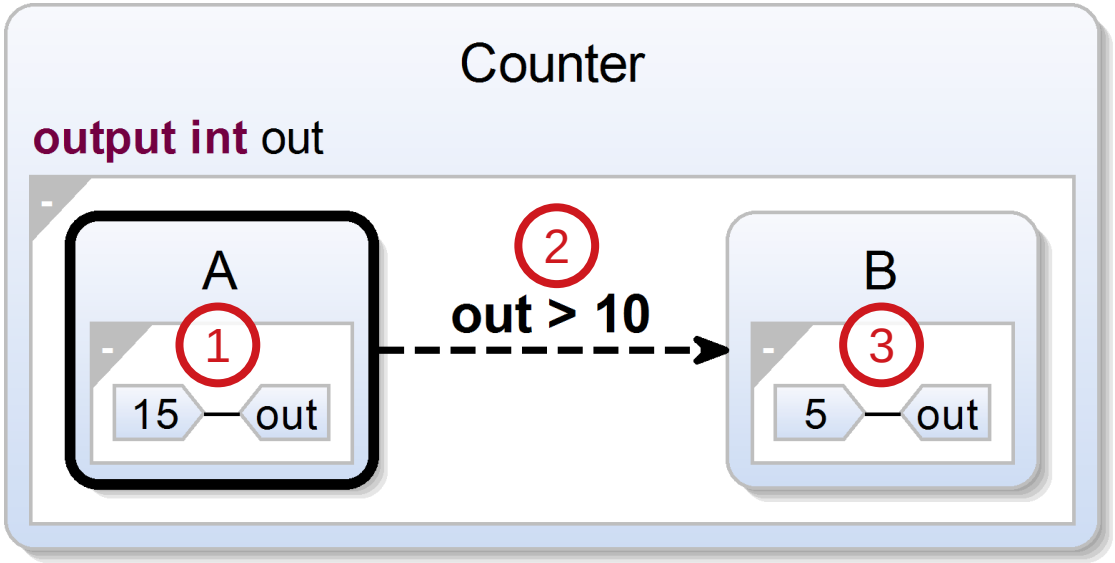
```
…
   x =  5;
   x += b? y;
   x += c? 3;
…
```

Sequential Constructiveness Sequentiality

```
node simpleInc ()
       returns (value:int)
let
  value = 0 -> value + 1;
tel
```

No Register Variable needed

Sequential Constructiveness
Sequentiality with Automata



> Execute behavior of two states within one tick

Thank you!