
LÜTJENBURG

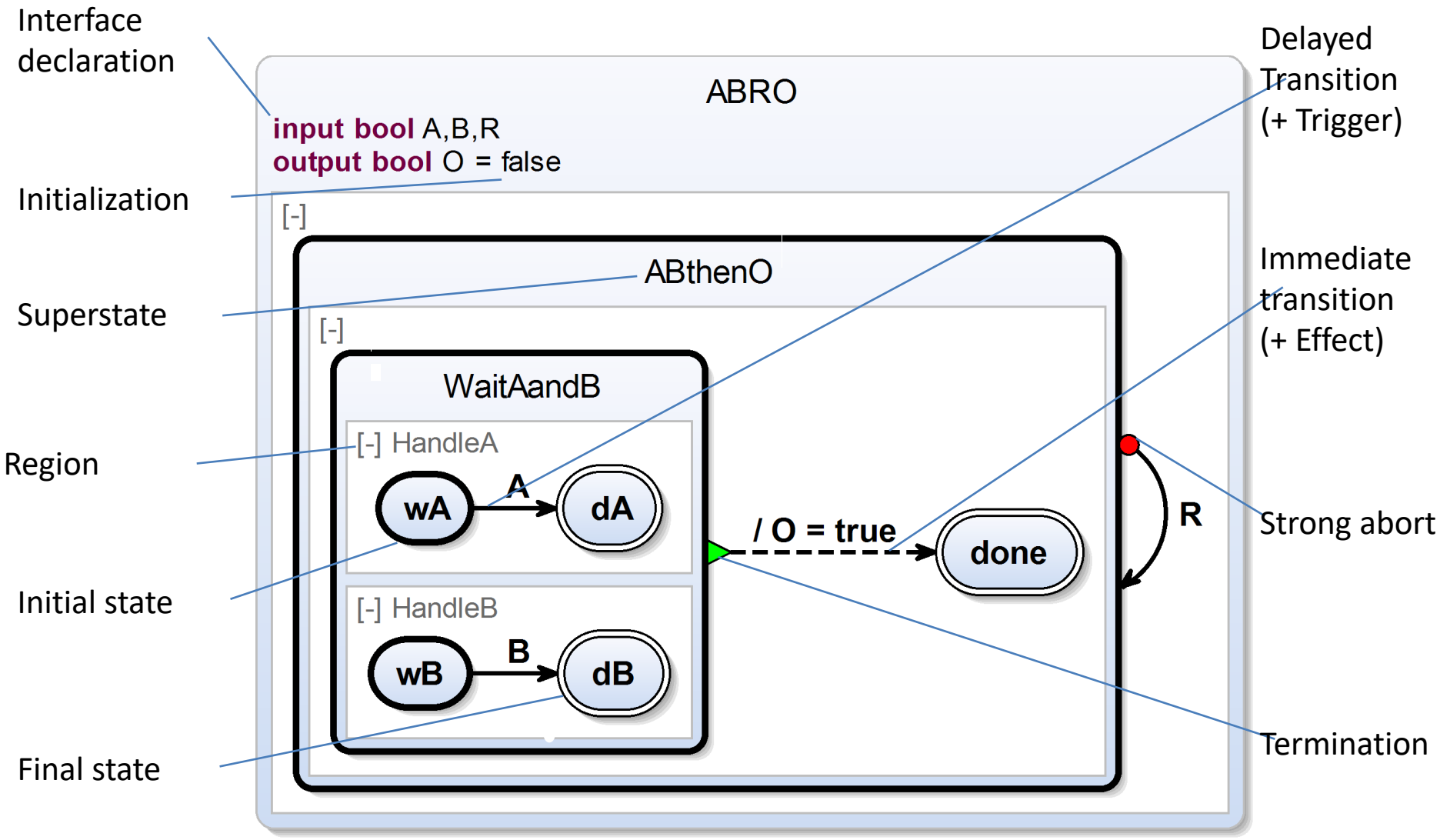
SCCharts Twelve Years Later
A Reflection on Sequential Constructiveness and Text-First Modeling

Reinhard von Hanxleden

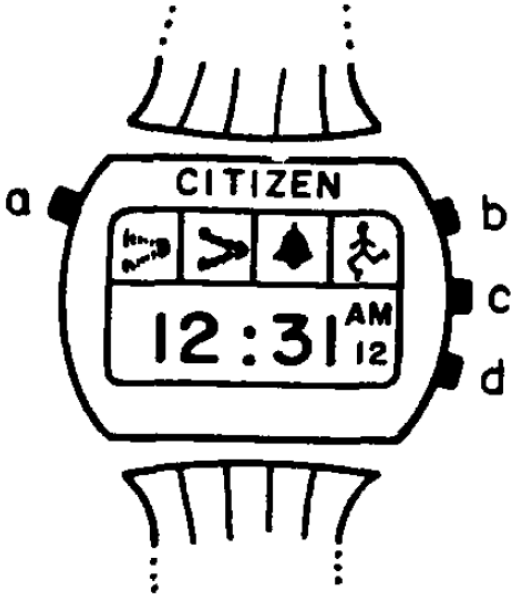
rvh@informatik.uni-kiel.de

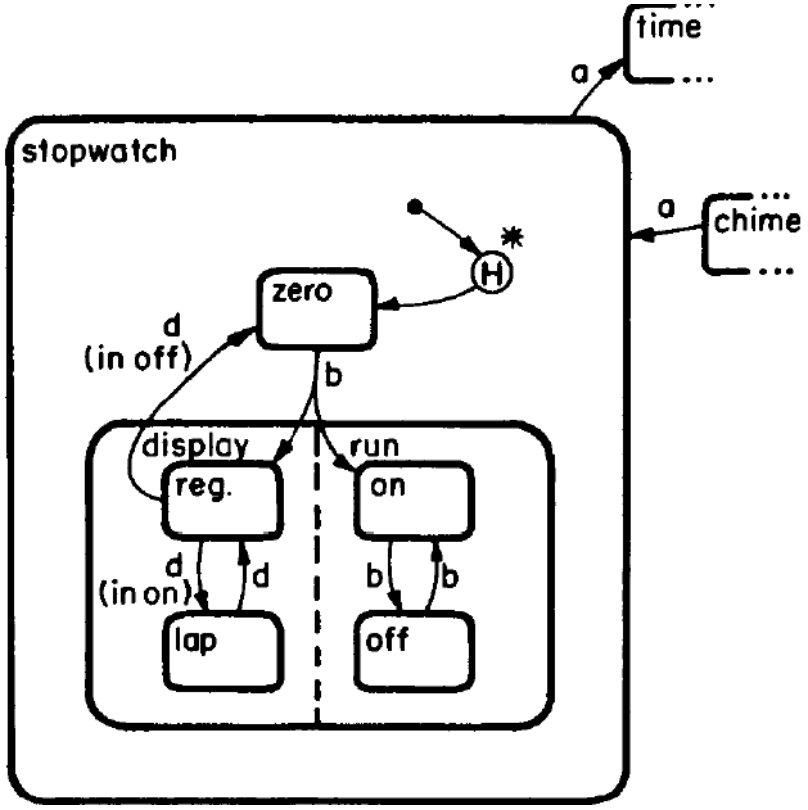
Presentation at SYNCHRON, Nov. 2025, Aussois, France

Based on keynote at LCTES 2025, June 17, 2025


Interface declaration
Initialization
Superstate
Region
Initial state
Final state
Delayed Transition (+ Trigger)
Immediate transition (+ Effect)
Strong abort
Termination
ABRO
input bool A,B,R
output bool O = false
[-]
ABthenO
[-]
WaitAandB
[-] HandleA
wA
A
dA
[-] HandleB
wB
B
dB
/ O = true
done
R

1980s: Statecharts

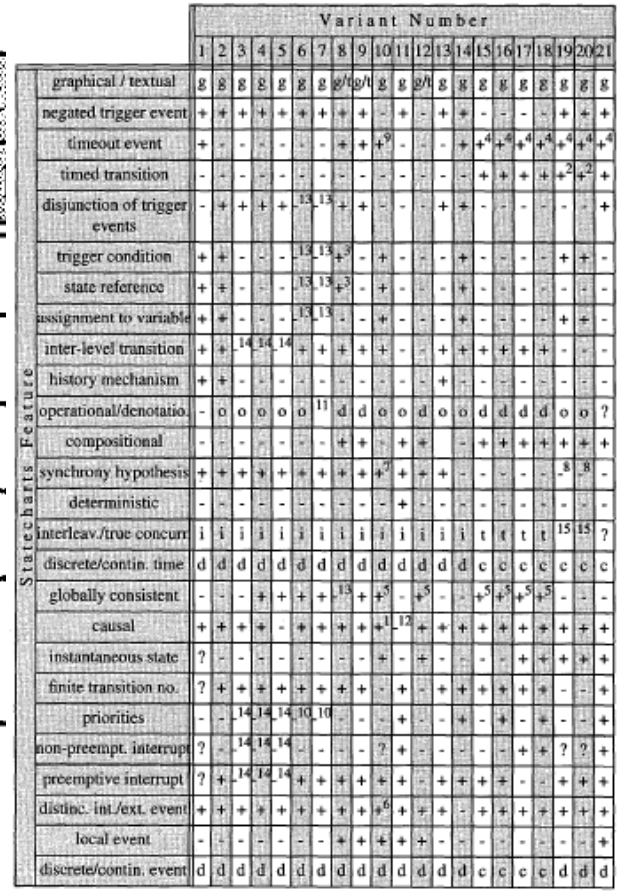




Harel
Statecharts: A Visual Formalism for Complex Systems
Science of Computer Programming, 1987

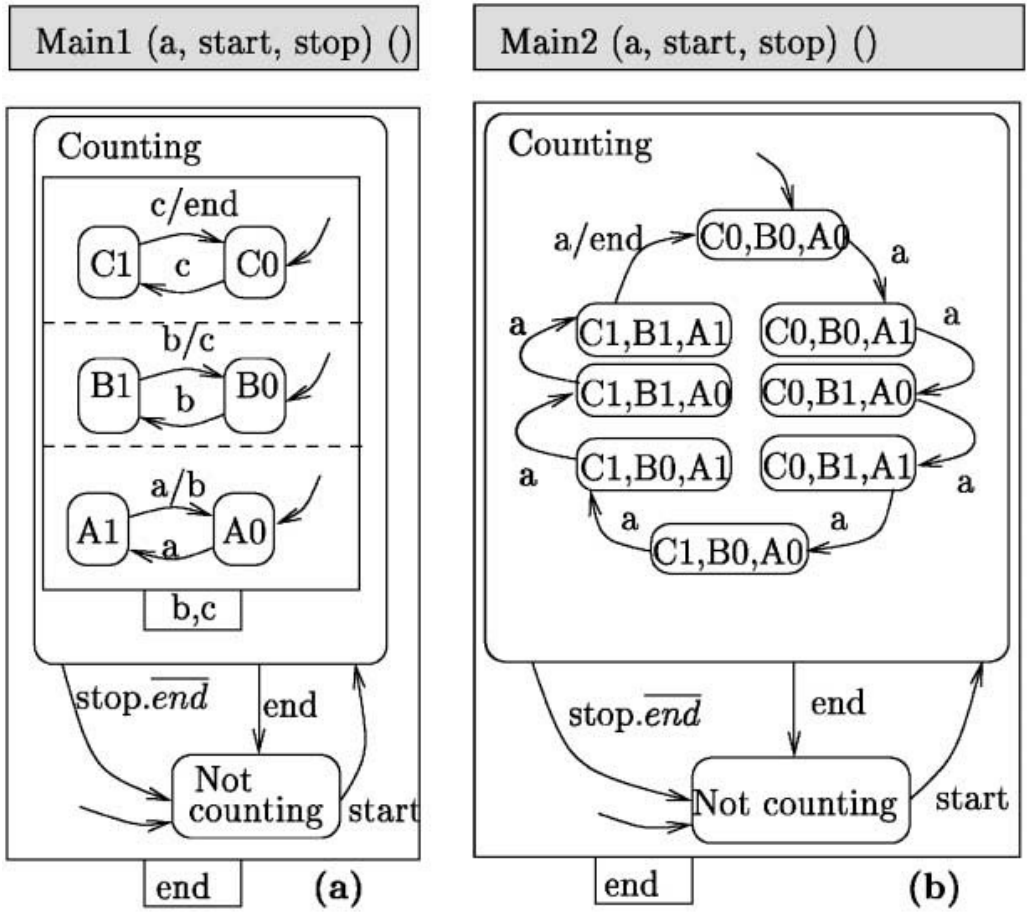
1990s: Many Statecharts

	Variant									
	1	2	3	4	5	6	7	8	9	10
graphical / textual	g	g	g	g	g	g	g	g/t	g/t	g
negated trigger event	+	+	+	+	+	+	+	+	+	-
timeout event	+	-	-	-	-	-	-	+	+	+[9]
timed transition	-	-	-	-	-	-	-	-	-	-
disjunction of trigger events	-	+	+	+	+	-[13]	-[13]	+	+	-



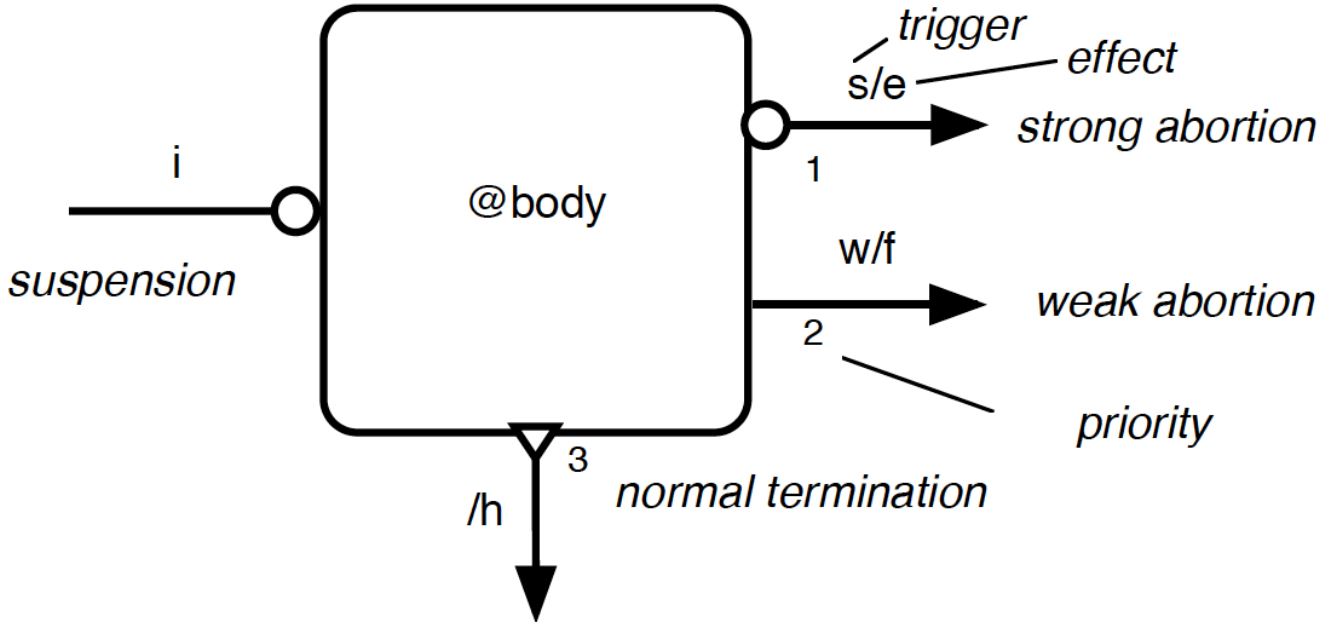
Michael von der Beeck
A Comparison of Statecharts Variants
Formal Techniques in Real-Time and Fault-Tolerant Systems, LNCS 863, 1994

1991: Argos



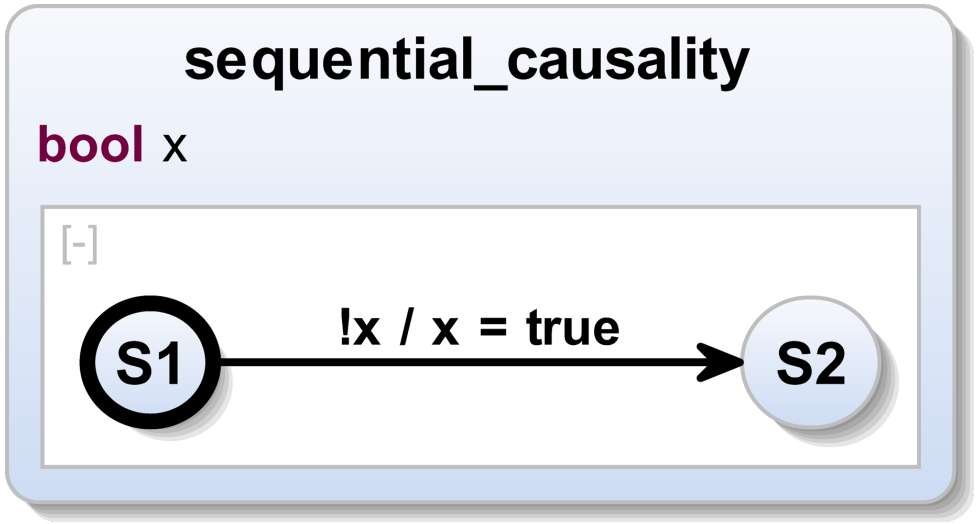
Florence Maraninchi
The Argos language: Graphical Representation of Automata and Description of Reactive Systems
IEEE Workshop on Visual Languages, Kobe, Japan, 1991

1995: SyncCharts, a.k.a. Safe State Machines



Charles André
SyncCharts: A Visual Representation of Reactive Behaviors
Research Report 95-52, I3S, Sophia Antipolis, 1995

Limitations of Strict Synchrony




```
if (!x) {
  ...
   x = true;
}
```

Not allowed in SyncCharts, Esterel, etc.!

SCCharts – Motivation

Preserve nice properties of synchronous programming

- Determinacy, sound semantic basis
- Static causality (i.e., determinacy) checking, no run-time surprises
- Efficient synthesis

Reduce the pain

- Make it easy to adopt for mainstream programmer
- Reject only models where determinacy is compromised
- Approach: harness scheduling information from sequential/imperative constructs

von Hanxleden, Duderstadt, Motika, Smyth, Mendler, Aguado, Stephen, O'Brien
SCCharts: sequentially constructive statecharts for safety-critical applications – HW/SW-synthesis for a conservative extension of synchronous statecharts
PLDI '14

Sequential Constructiveness

Idea: Sequential control flow overrides „write before read“

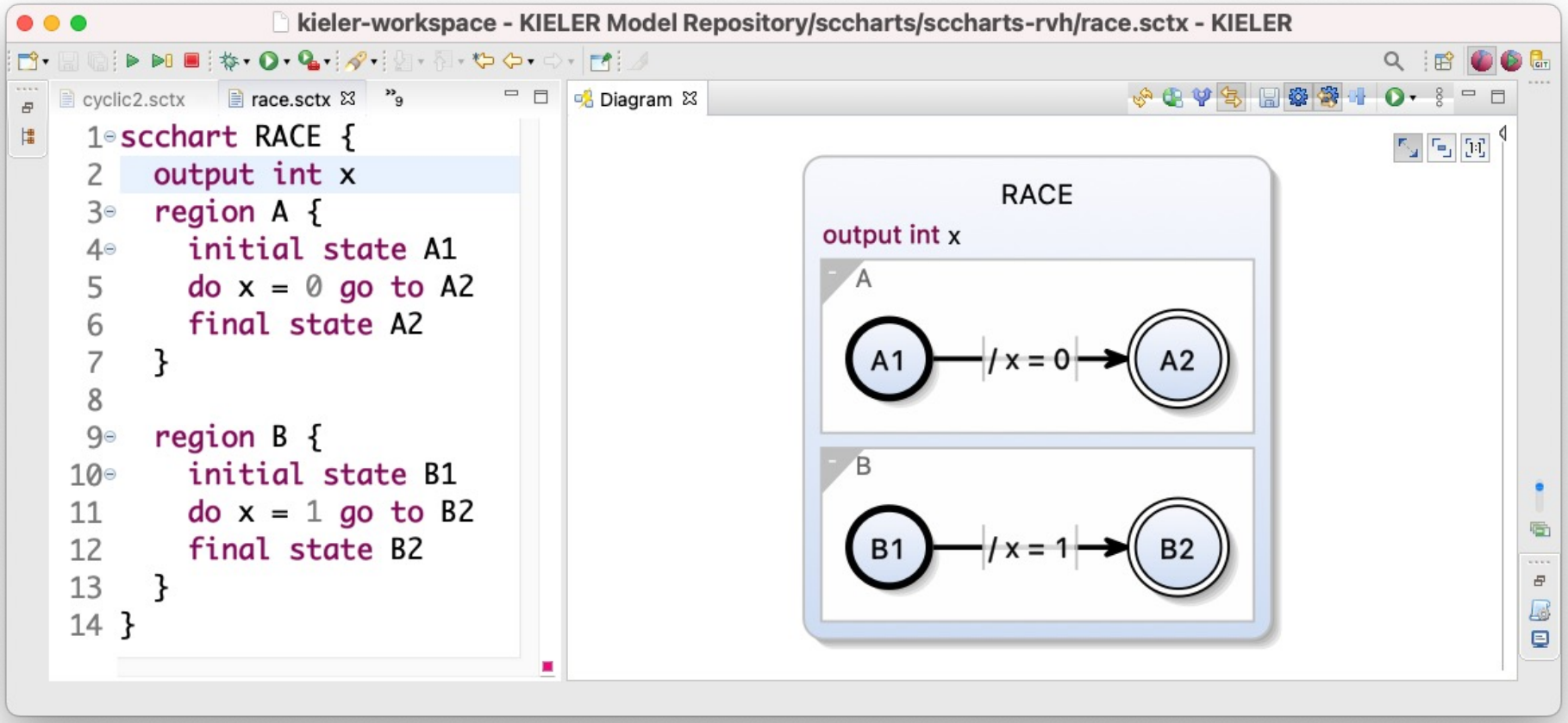
Writes visible **only** to reads that are

1. sequential successors or
2. concurrent

Reinhard von Hanxleden, Michael Mendler, et al.
Sequentially Constructive Concurrency—A Conservative Extension of the Synchronous Model of Computation.
ACM TECS ’14

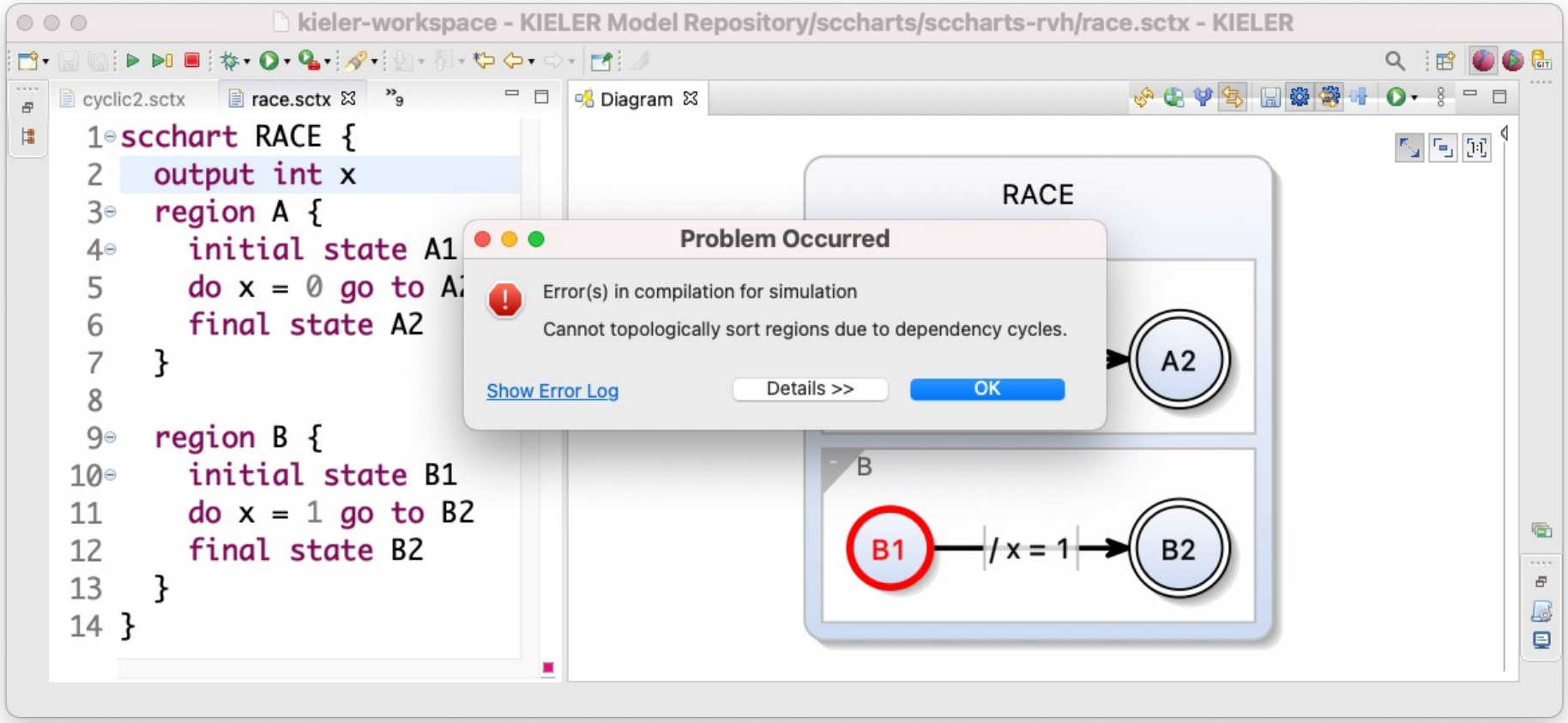
Must Still Reject Some Models …

Concurrent accesses may lead to causality cycles



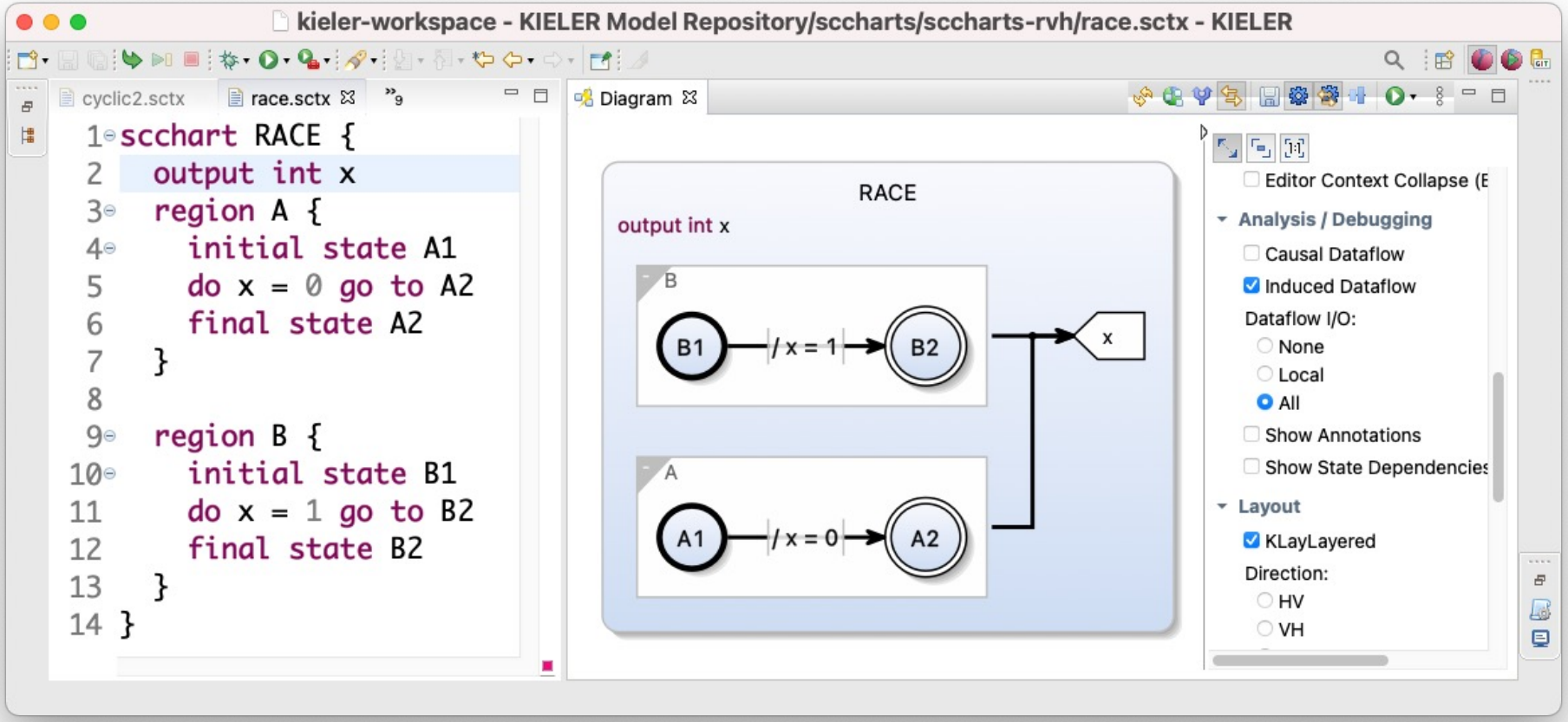
Must Still Reject Some Models …

Concurrent accesses may lead to causality cycles

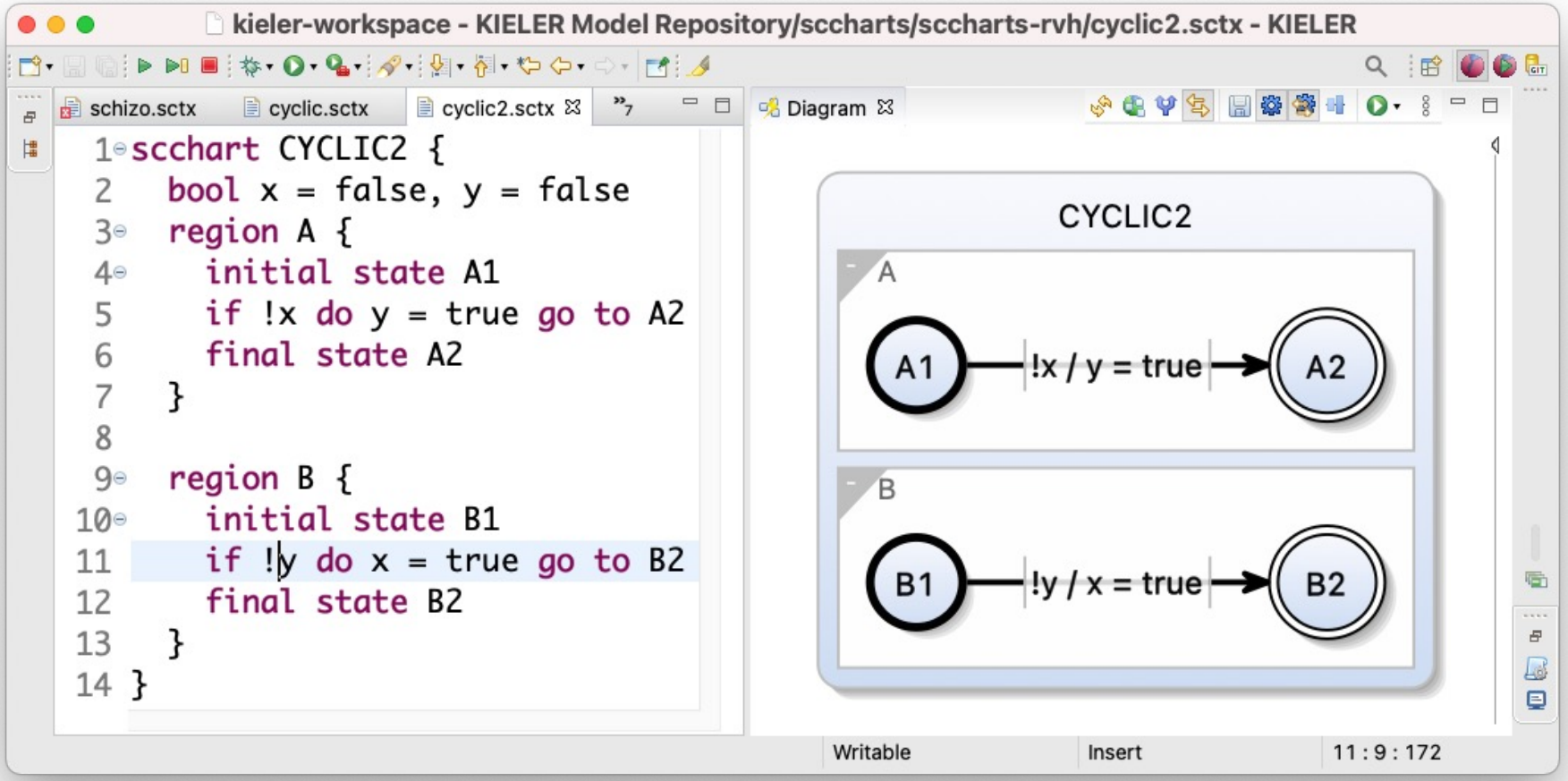


Must Still Reject Some Models …

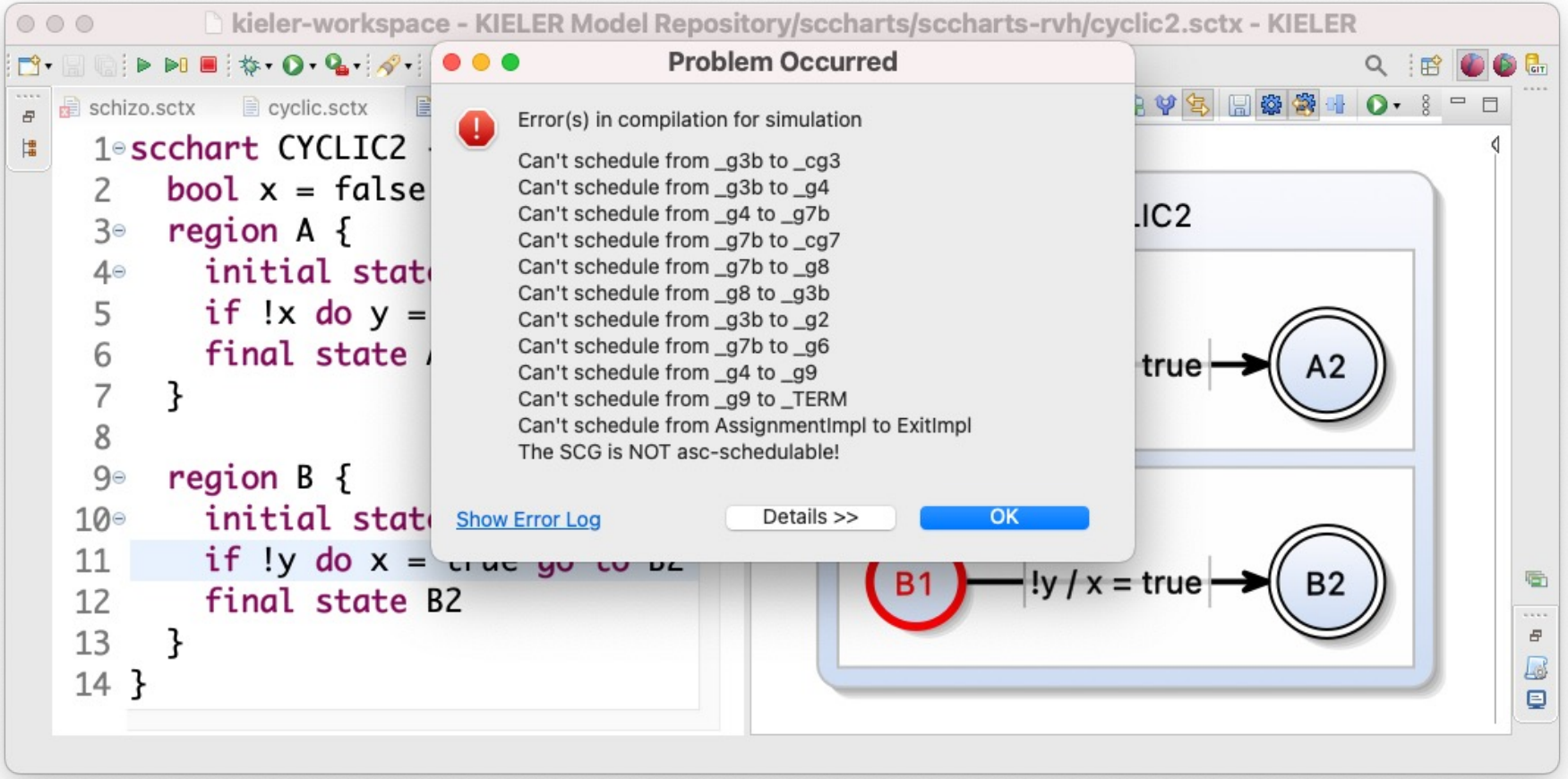
Concurrent accesses may lead to causality cycles



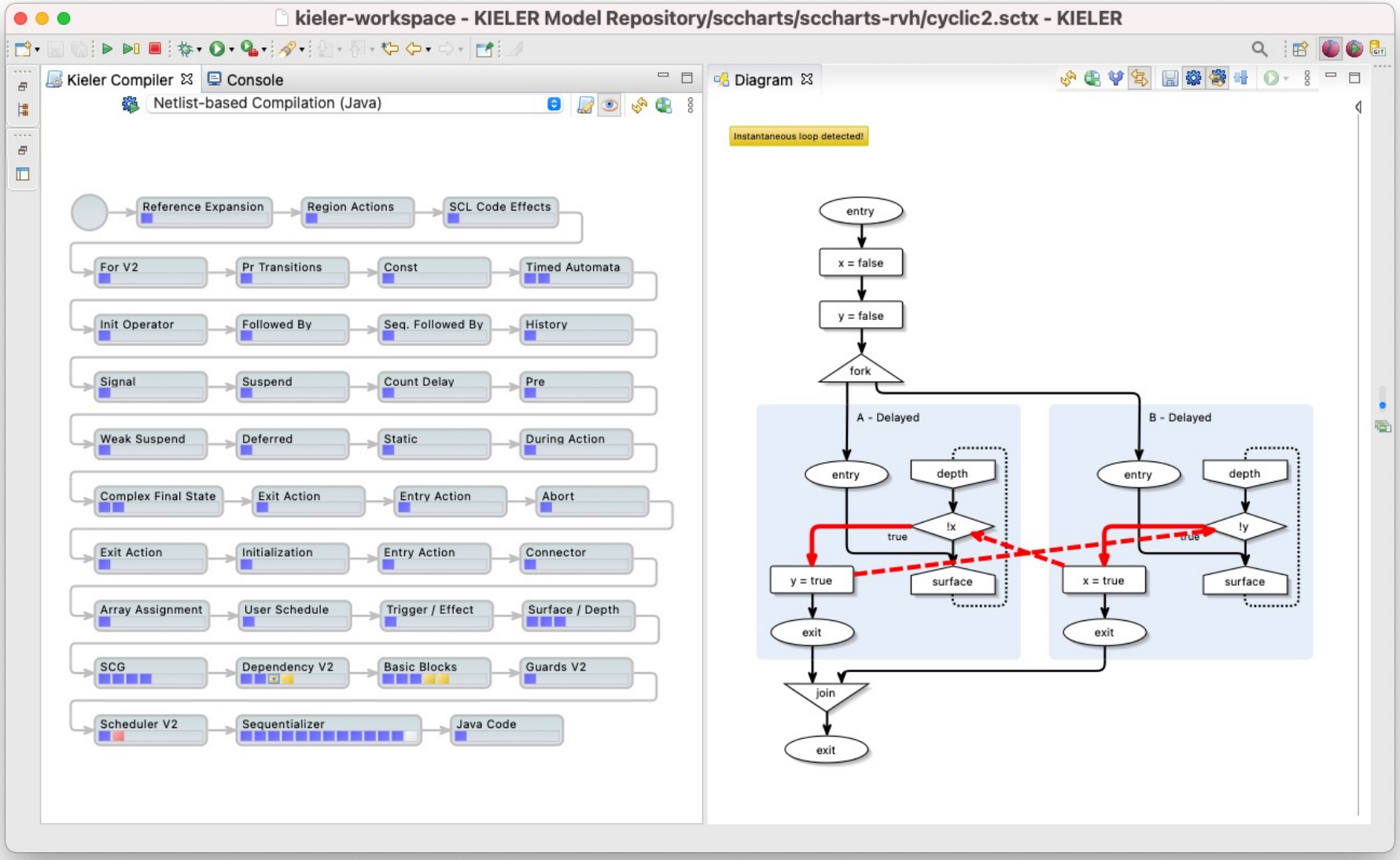
Another Example



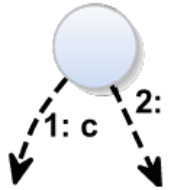
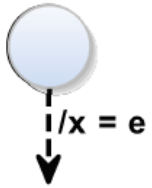
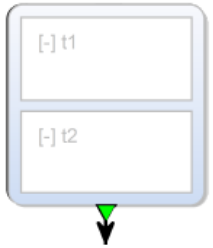
Another Example

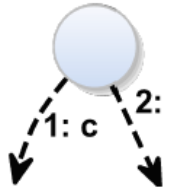
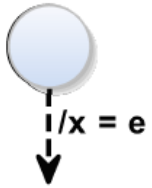
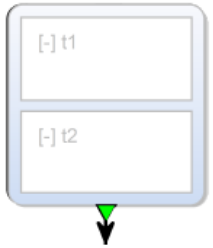


Another Example

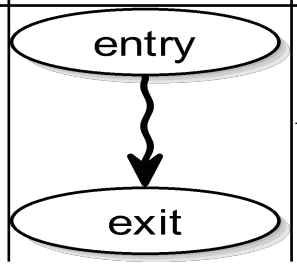
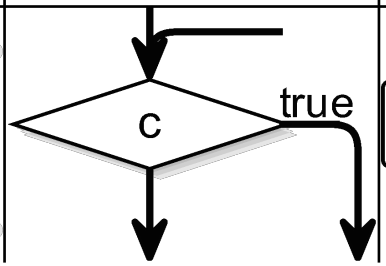
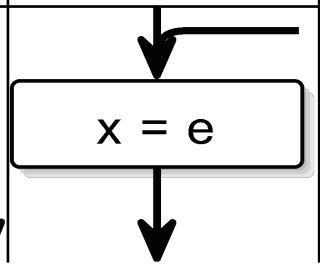
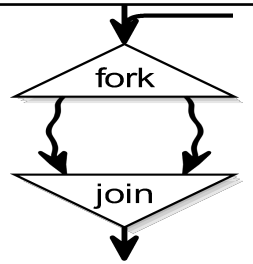
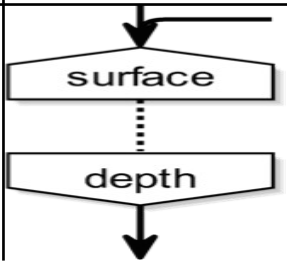


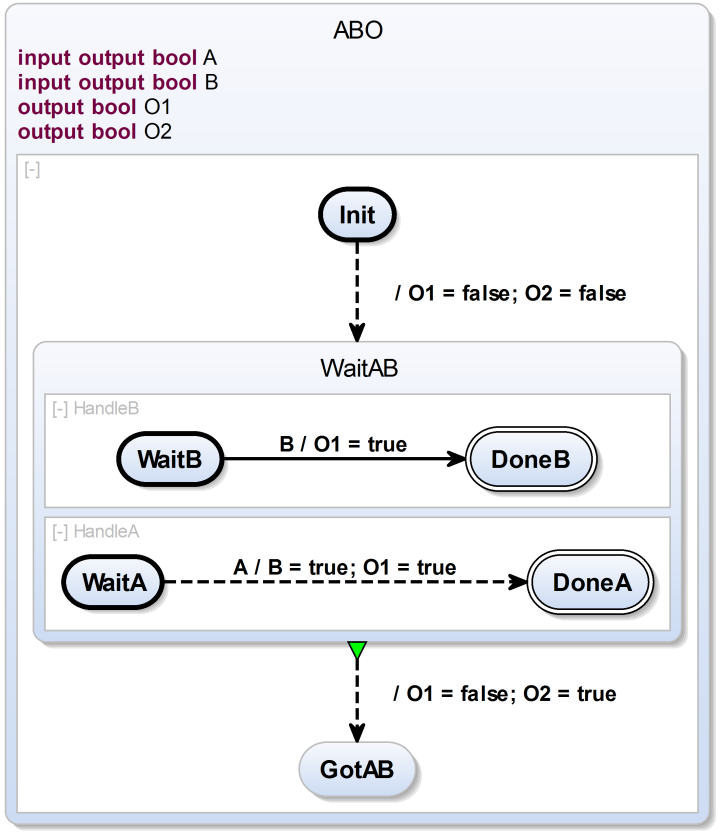
SCChart Building Blocks

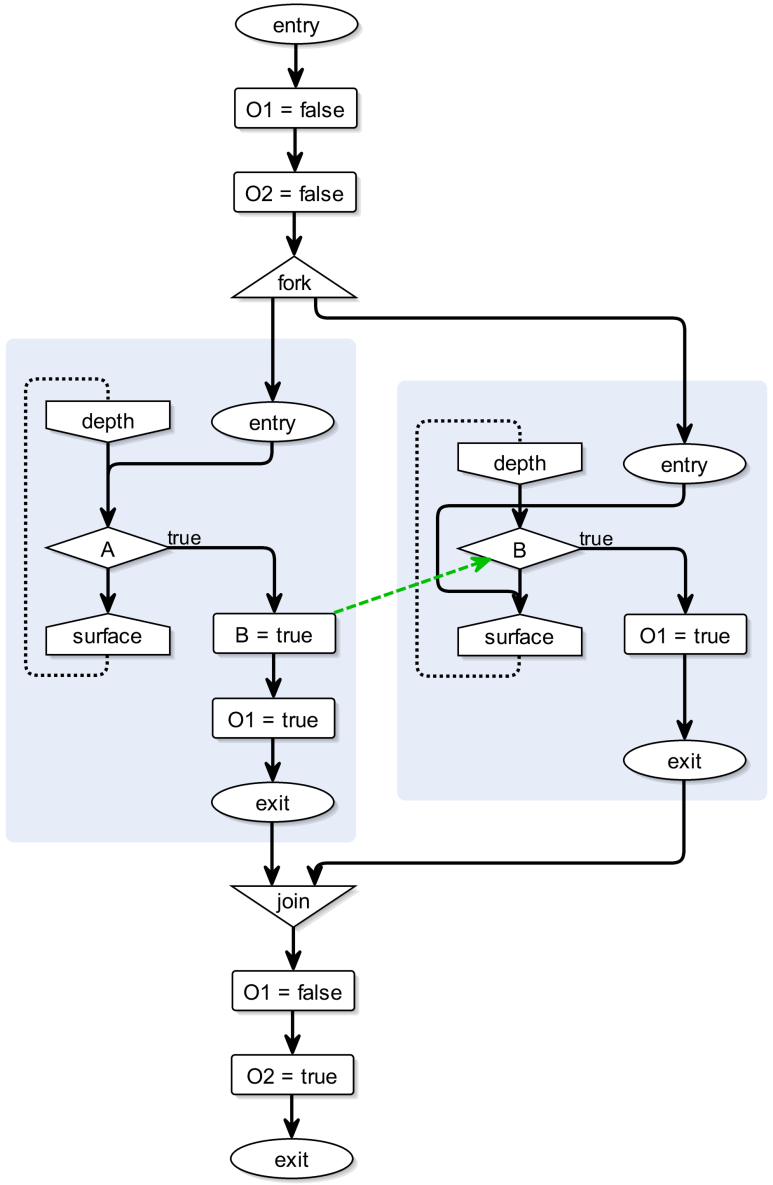
	Region	Trigger	Effect	Superstate	State
Normalized Core SCCharts					

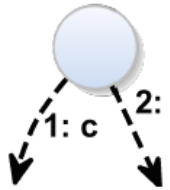
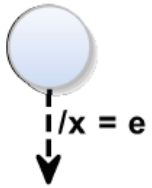
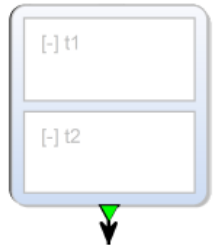
	Region	Trigger	Effect	Superstate	State
Normalized Core SCCharts					

M2M Mappings

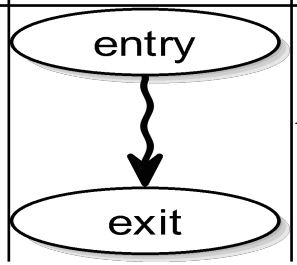
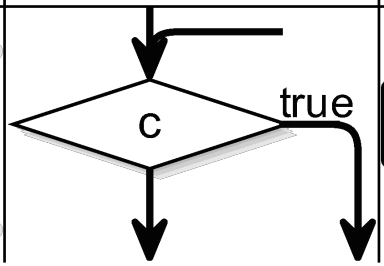
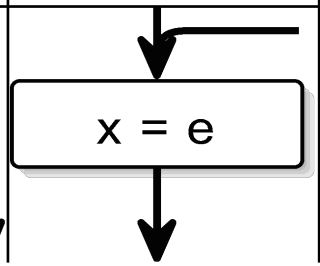
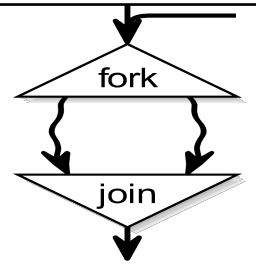
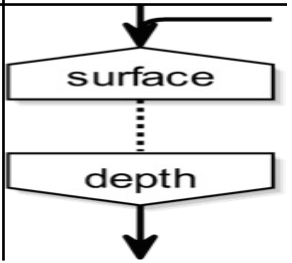
	Thread	Conditional	Assignment	Concurrency	Delay
SCL	*t*	if (*c*) s_1 else s_2	*x* = *e*	fork t_1 par t_2 join	pause
SCG					


ABO
input output bool A
input output bool B
output bool O1
output bool O2
[-]
Init
/ O1 = false; O2 = false
WaitAB
[-] HandleB
WaitB
B / O1 = true
DoneB
[-] HandleA
WaitA
A / B = true; O1 = true
DoneA
/ O1 = false; O2 = true
GotAB

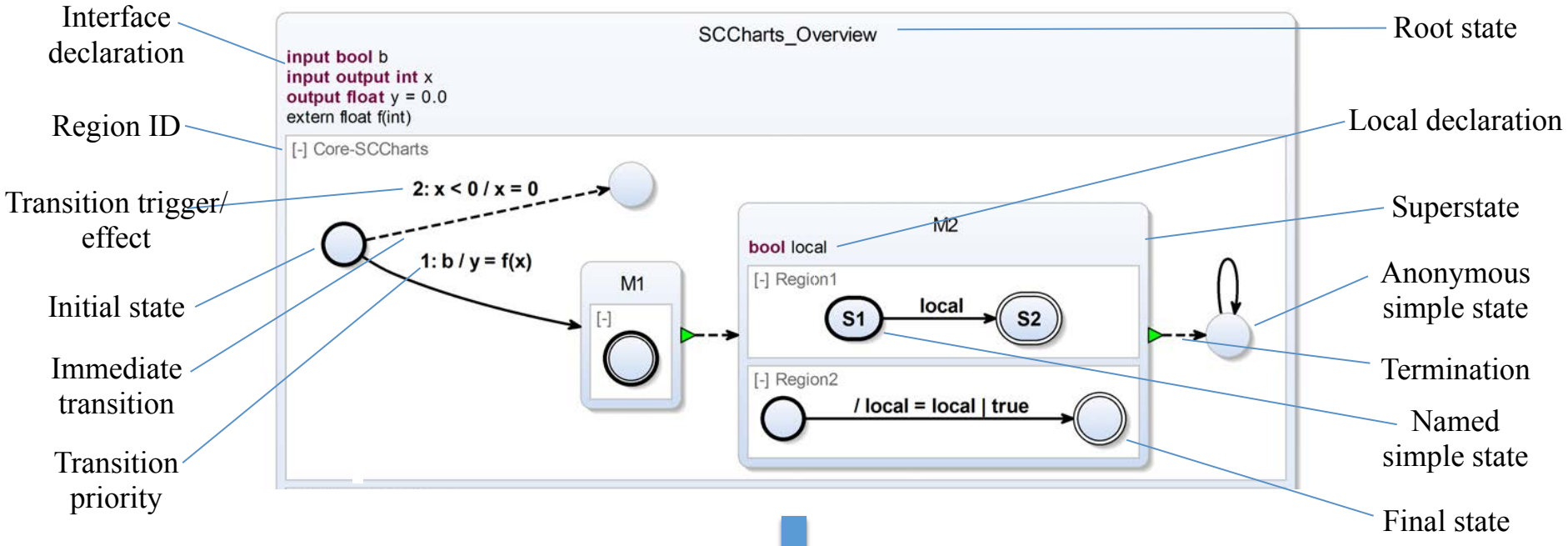

entry
O1 = false
O2 = false
fork
depth
entry
A
true
surface
B = true
O1 = true
exit
depth
entry
B
true
surface
O1 = true
exit
join
O1 = false
O2 = true
exit

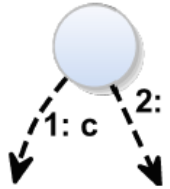
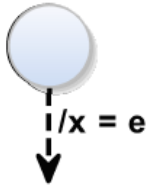
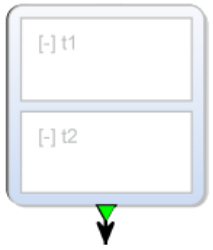
	Region	Trigger	Effect	Superstate	State
Normalized Core SCCharts					

M2M Mappings

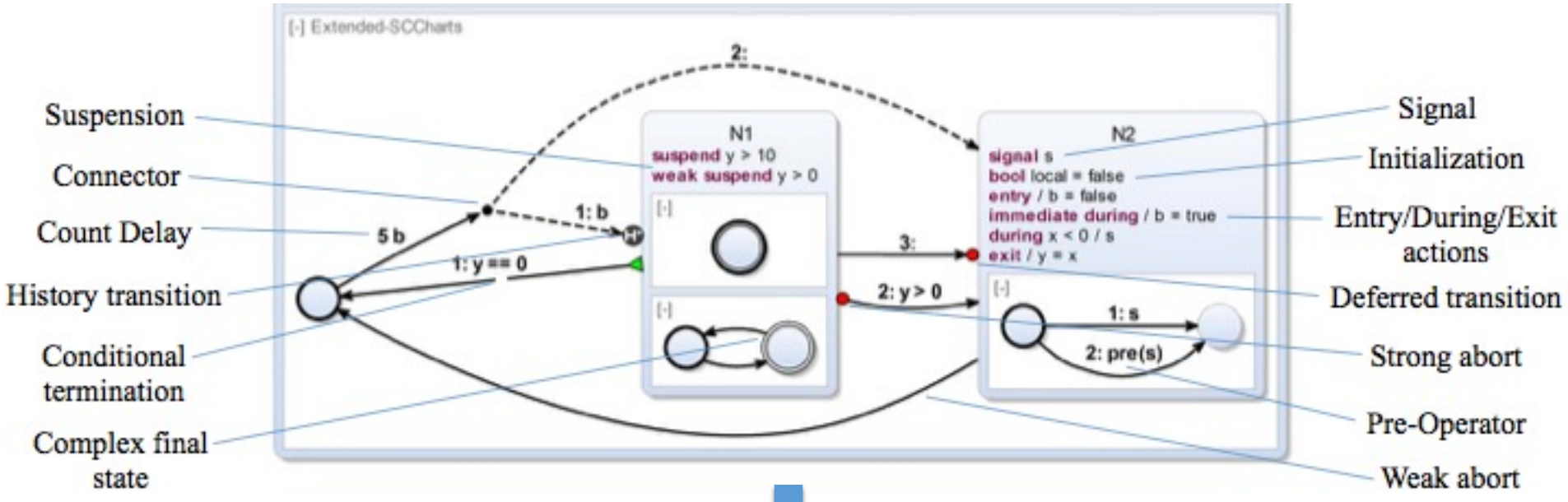
	Thread	Conditional	Assignment	Concurrency	Delay
SCL	*t*	if (*c*) s_1 else s_2	*x* = *e*	fork t_1 par t_2 join	pause
SCG					

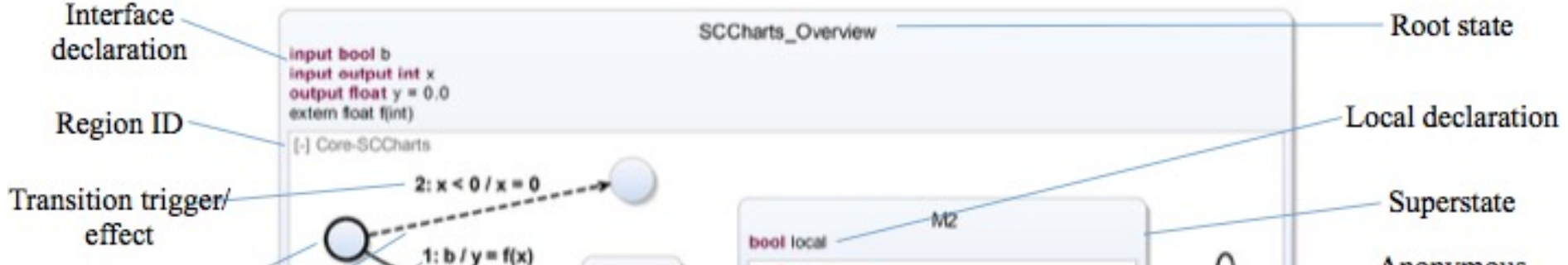
Some Syntactic Sugar: Core SCCharts



	Region	Trigger	Effect	Superstate	State
Normalized Core SCCharts					

More Syntactic Sugar: Extended SCCharts

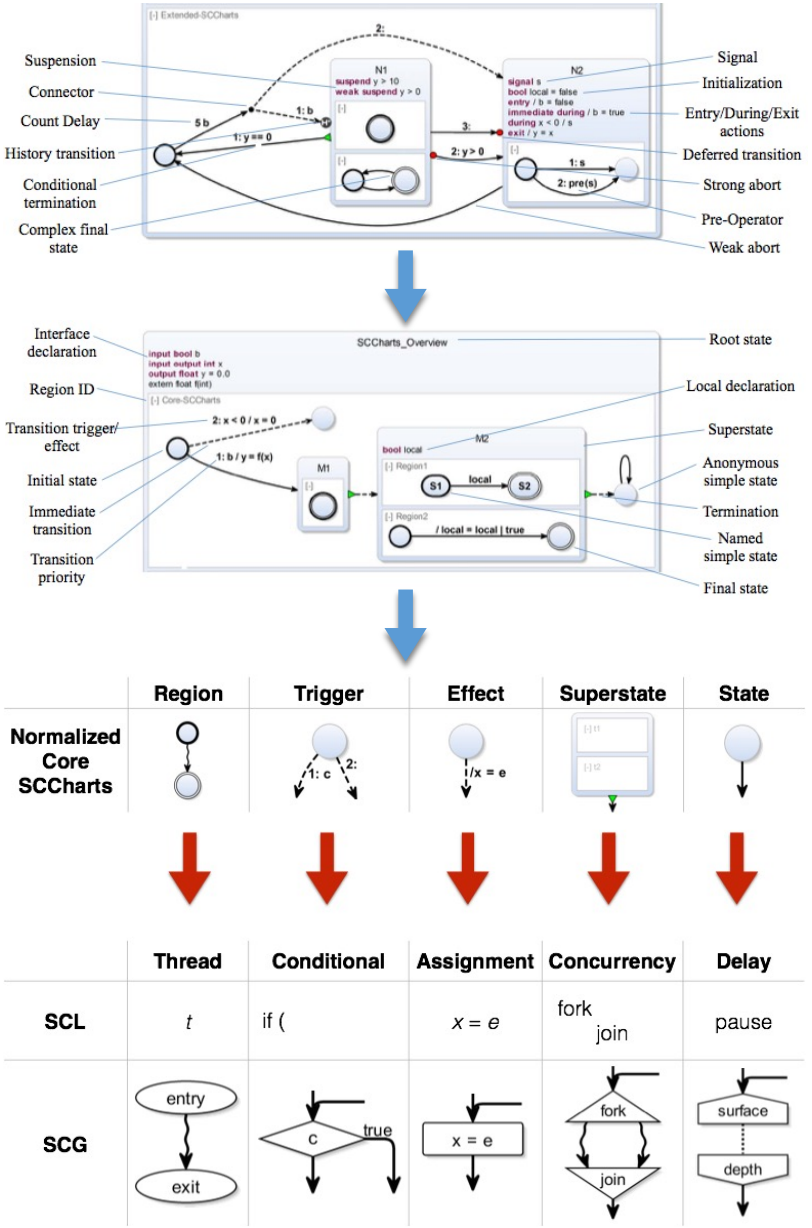




Taking Stock

SCCharts defined by M2M Transformations

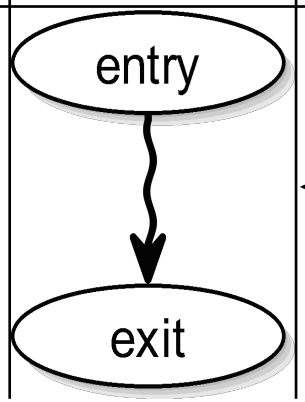
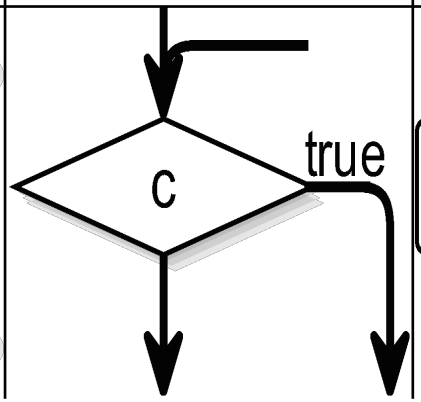
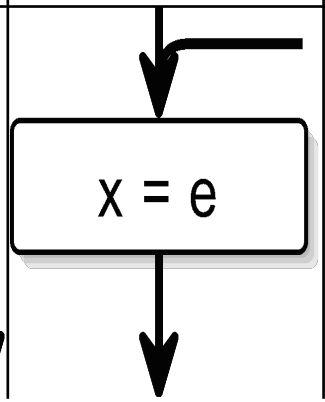
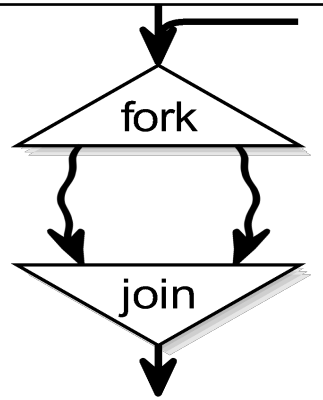
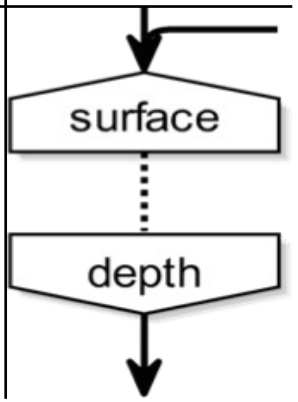
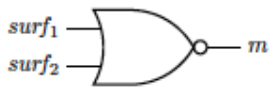
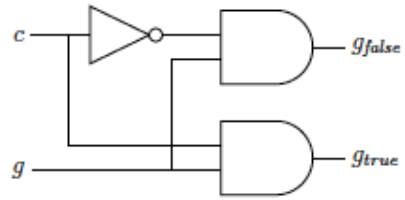
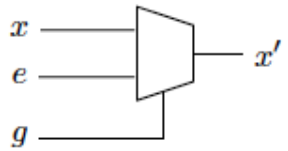
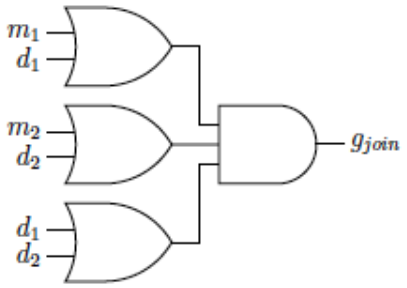
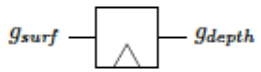
- Extended SCCharts
- Core SCCharts
- Normalized Core SCCharts
- SCL/SCG



Downstream Compilation

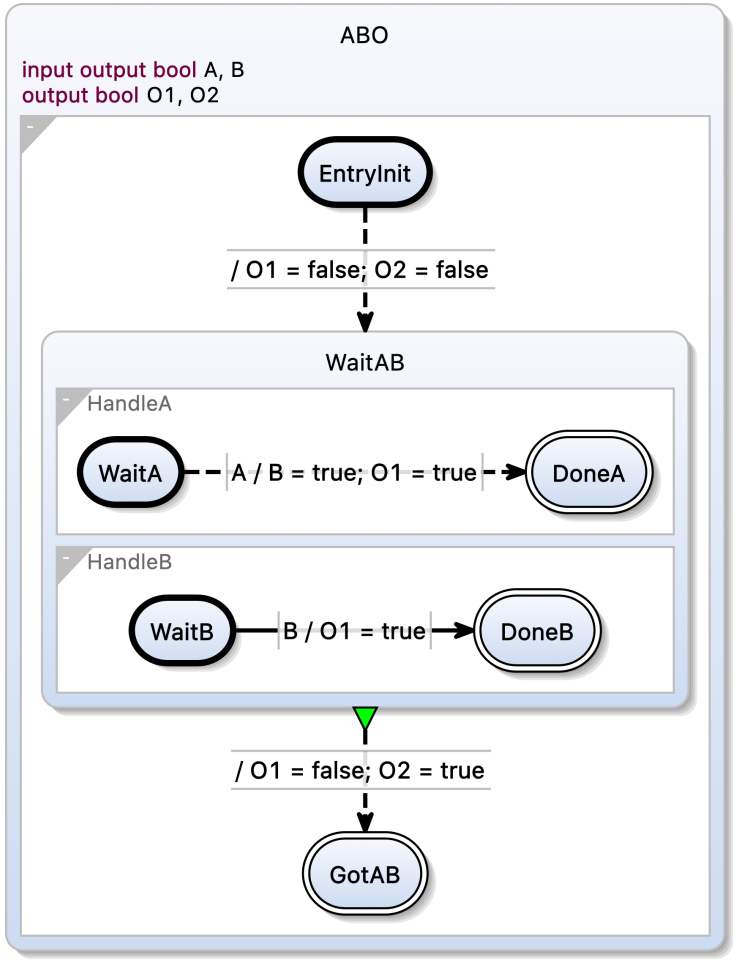
	Dataflow Approach	**Priority Approach**
Accepts instantaneous loops	–	+
Can synthesize hardware	+	–
Can synthesize software	+	+
Size scales well (linear in size of SCChart)	+	+
Speed scales well (execute only active parts)	–	+
Instruction-cache friendly (good locality)	+	–
Pipeline friendly (little/no branching)	+	–
WCRT predictable (simple control flow)	+	+/–
Low execution time jitter (simple/fixed flow)	+	–
Variable number (guard variables)	–	+

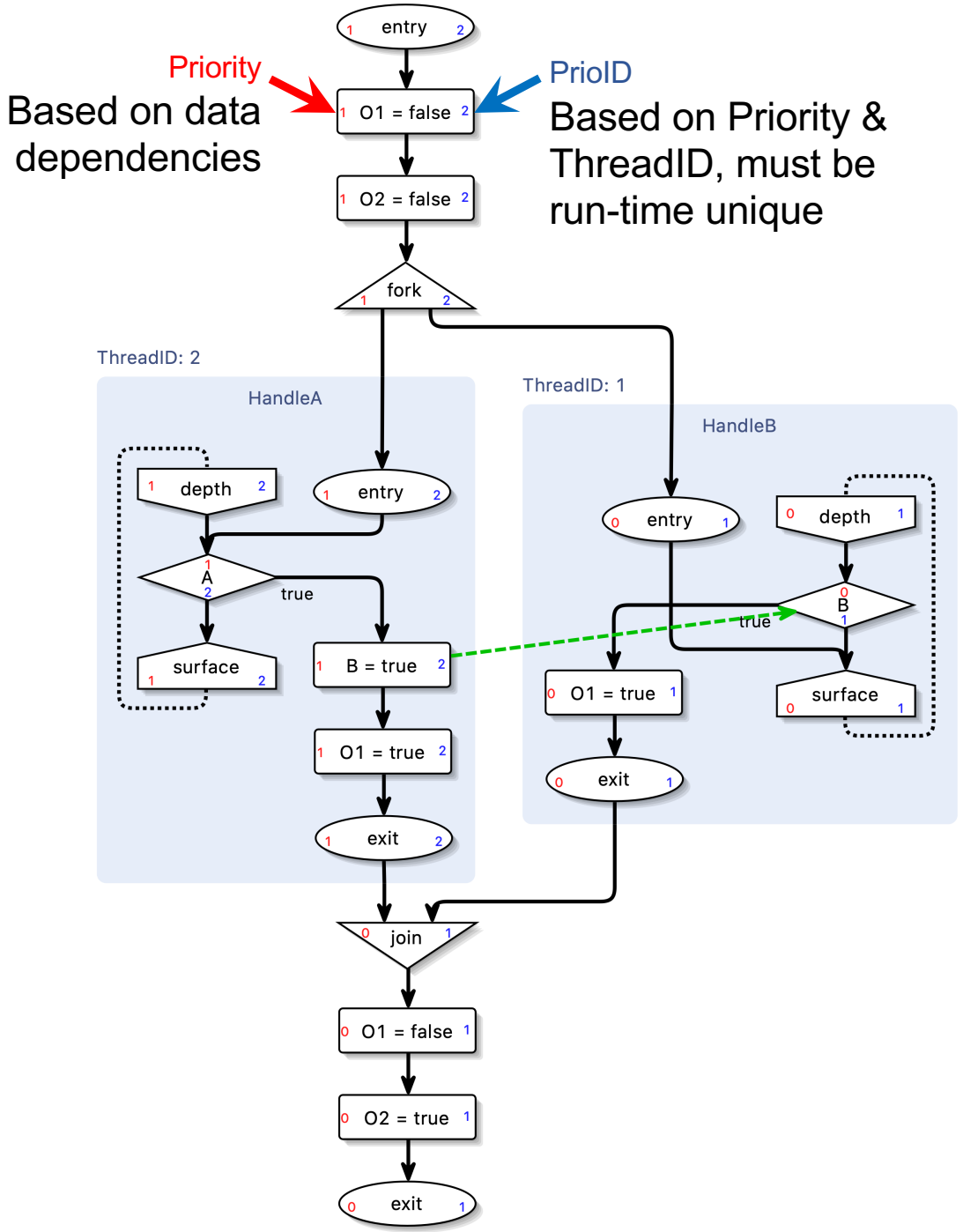
von Hanxleden, Duderstadt, Motika, Smyth, Mendler, Aguado, Stephen, O'Brien
SCCharts: sequentially constructive statecharts for safety-critical applications – HW/SW-synthesis for a conservative extension of synchronous statecharts
PLDI '14

	Thread	Conditional	Assignment	Concurrency	Delay
SCL	*t*	if (*c*) *s1* else *s2*	*x* = *e*	fork *t1* par *t2* join	pause
SCG					
Data-Flow Code	$d = g_{exit}$ $m = \neg \bigvee_{surf \in t} g_{surf}$	$g = \bigvee g_{in}$ $g_{true} = g \wedge c$ $g_{false} = g \wedge \neg c$	$g = \bigvee g_{in}$ $x' = g\ ?\ e : x$	$g_{join} = (d_1 \vee m_1) \wedge (d_2 \vee m_2) \wedge (d_1 \vee d_2)$	$g_{surf} = \bigvee g_{in}$ $g_{depth} =$ pre (g_{surf})
Circuits					

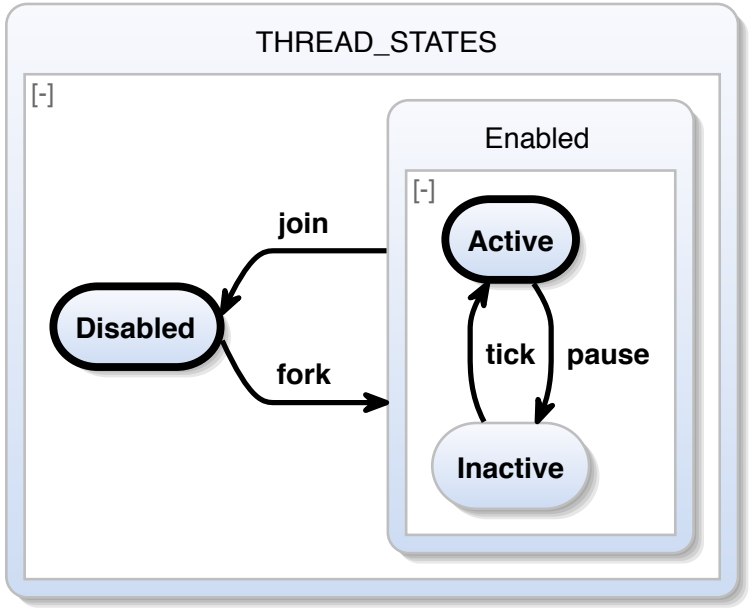
Priority-Based Compilation

- More software-like
- Don't emulate control flow with guards/basic blocks, but with program counters/threads
- Priority-based thread dispatching
- SCL$_P$: SCL + PrioIDs
- In C: implemented as macros, using computed gotos
- In Java: no macros, no gotos
  - emulate gotos with while + break

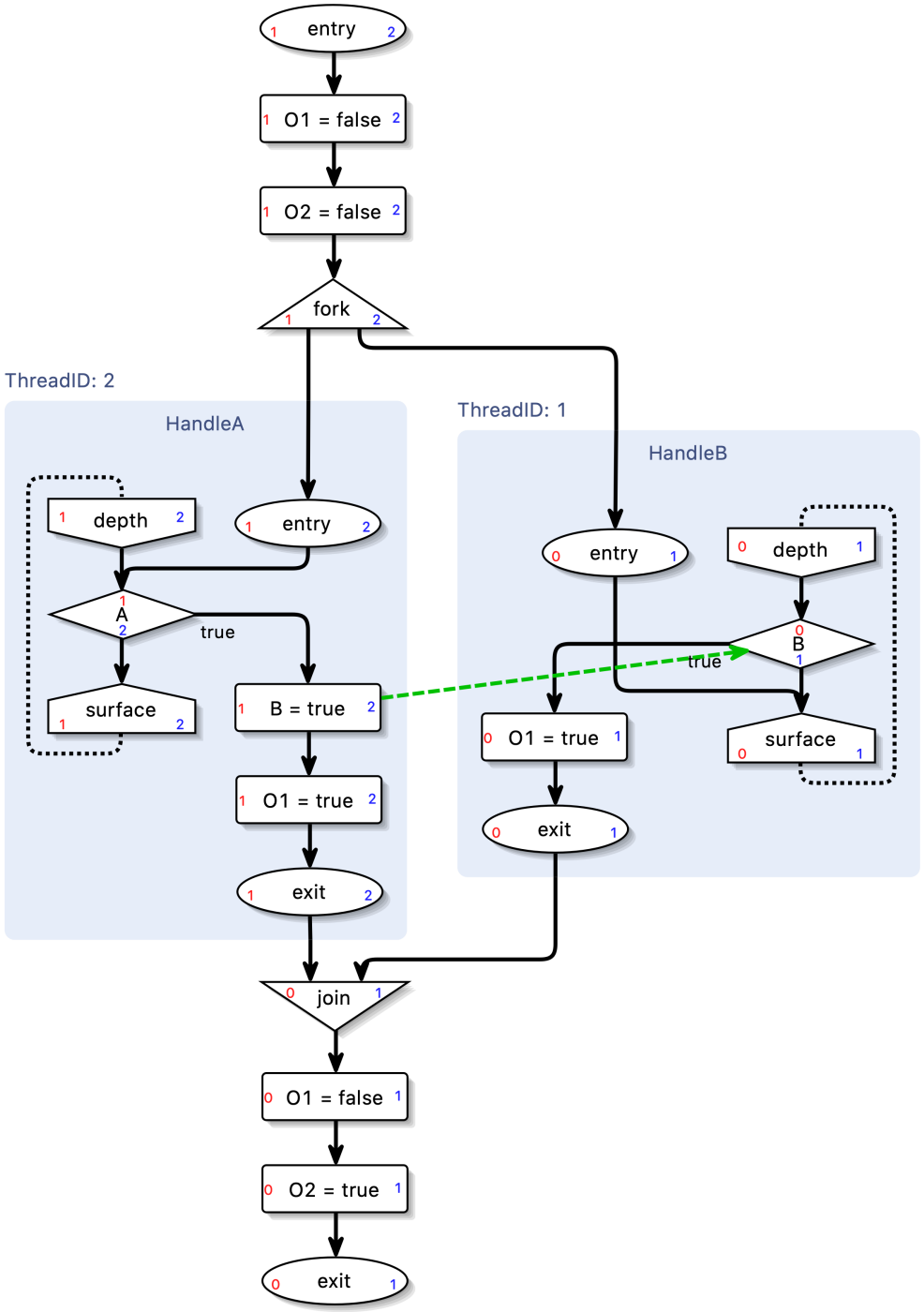

ABO
input output bool A, B
output bool O1, O2
EntryInit
/ O1 = false; O2 = false
WaitAB
HandleA
WaitA
A / B = true; O1 = true
DoneA
HandleB
WaitB
B / O1 = true
DoneB
/ O1 = false; O2 = true
GotAB


Priority
Based on data dependencies
PrioID
Based on Priority & ThreadID, must be run-time unique
entry
O1 = false
O2 = false
fork
ThreadID: 2
HandleA
depth
entry
A
true
surface
B = true
O1 = true
exit
ThreadID: 1
HandleB
entry
depth
B
true
O1 = true
surface
exit
join
O1 = false
O2 = true
exit

```
1 // Declare Boolean type
2 typedef int bool;
3 #define false 0
4 #define true 1
5
6 // Generate "_L<line-number>" label
7 #define _concat_helper(a, b) a ## b
8 #define _concat(a, b) _concat_helper(a, b)
9 #define _label_    _concat(_L, __LINE__)
10
11 // Enable/disable threads with prioID p
12 #define _u2b(u)      (1 << u)
13 #define _enable(p) _enabled |= _u2b(p); _active |= _u2b(p)
14 #define _isEnabled(p) ((_enabled & _u2b(p)) != 0)
15 #define _disable(p) _enabled &= ~_u2b(p)
```



```
17 // Set current thread continuation
18 #define _setPC(p, label) _pc[p] = &&label
19
20 // Pause, resume at <label> or at pause
21 #define _pause(label) _setPC(_cid, label); goto _L_PAUSE
22 #define pause      _pause(_label_); _label_:
23
24 // Fork/join sibling thread with prioID p
25 #define fork1(label, p) _setPC(p, label); _enable(p);
26 #define join1(p)   _label_: if (_isEnabled(p)) { _pause(_label_); }
27
28 // Terminate thread at "par"
29 #define par        goto _L_TERM;
30
31 // Context switch (change prioID)
32 #define _prio(p)  _deactivate(_cid); _disable(_cid); _cid = p; \
33  _enable(_cid); _setPC(_cid, _label_); goto _L_DISPATCH; _label_:
```

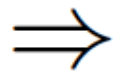


SCL$_P$:

```
85 int tick()
86 {
87   tickstart(2);
88   O1 = false;
89   O2 = false;
90
91   fork1(HandleB,
         1) {
92     HandleA:
93     if (!A) {
94       pause;
95       goto HandleA
           ;
96     }
97     B = true;
98     O1 = true;
99
100  } par {
102    HandleB:
103    pause;
104    if (!B) {
105      goto HandleB
           ;
106    }
107    O1 = true;
108  } join1(2);
109
110  O1 = false;
111  O2 = true;
112  tickreturn;
113 }
```

ABO SCL$_P$ I

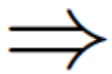
```
85 int tick()
86 {
87   tickstart(2);
88   O1 = false;
89   O2 = false;
90
91   fork1(HandleB,
         1) {
92     HandleA:
93     if (!A) {
94       pause;
95       goto HandleA
             ;
96     }
97     B = true;
98     O1 = true;
99
100  } par {
```



```
85 int tick()
86 {
87   if (_notInitial) { _active = _enabled;
          goto _L_DISPATCH; } else { _pc[0]
          = &&_L_TICKEND; _enabled = (1 <<
         0); _active = _enabled; _cid = 2;
         ; _enabled |= (1 << _cid); _active
          |= (1 << _cid); _notInitial = 1;
         } ;
88   O1 = 0;
89   O2 = 0;
90
91   _pc[1] = &&HandleB; _enabled |= (1 <<
         1); _active |= (1 << 1); {
92     HandleA:
93     if (!A) {
94       _pc[_cid] = &&_L94; goto _L_PAUSE;
             _L94:;
95       goto HandleA;
96     }
97     B = 1;
98     O1 = 1;
99
100  } goto _L_TERM; {
```

ABO SCL$_P$ II

```
102    HandleB:
103    pause;
104    if (!B) {
105      goto HandleB
             ;
106    }
107    O1 = true;
108  } join1(2);
109
110  O1 = false;
111  O2 = true;
112  tickreturn;
113 }
```



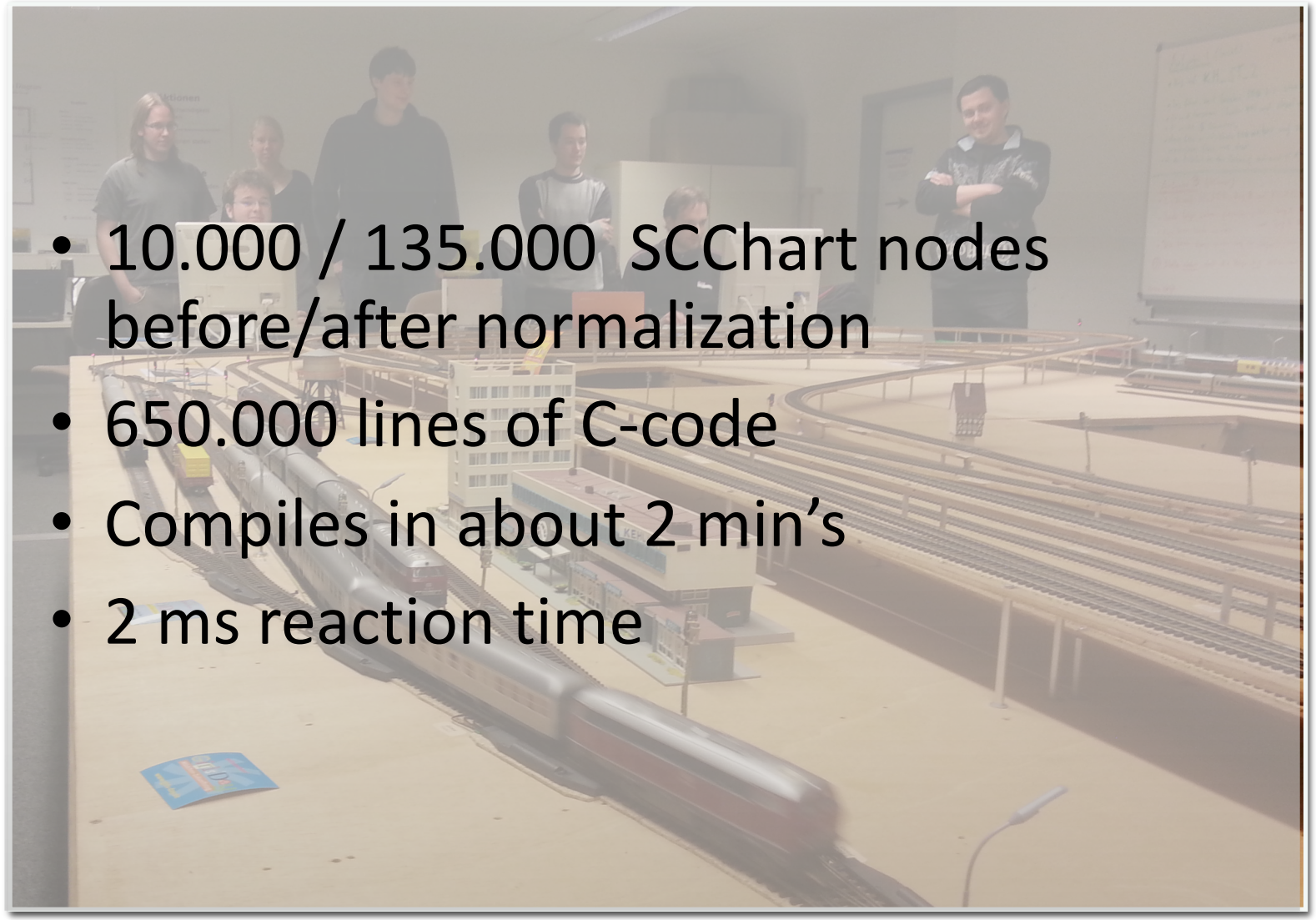
```
102    HandleB:
103    _pc[_cid] = &&_L103; goto _L_PAUSE;
           _L103:;
104    if (!B) {
105      goto HandleB;
106    }
107    O1 = 1;
108  } _L108: if (((_enabled & (1 << 2)) !=
           0)) { _pc[_cid] = &&_L108; goto
         _L_PAUSE; };
109
110  O1 = 0;
111  O2 = 1;
112  goto _L_TERM; _L_TICKEND: return (
         _enabled != (1 << 0)); _L_TERM:
         _enabled &= ~(1 << _cid); _L_PAUSE
         : _active &= ~(1 << _cid);
         _L_DISPATCH: __asm volatile("bsrl
         %1,%0\n" : "=r" (_cid) : "r" (
         _active) ); goto *_pc[_cid];
113 }
```

Take-Home Message

- Can do wonderful things with C preprocessor!
- Can do wonderful things with computed gotos!
- Can do wonderful things with embedded assembler!

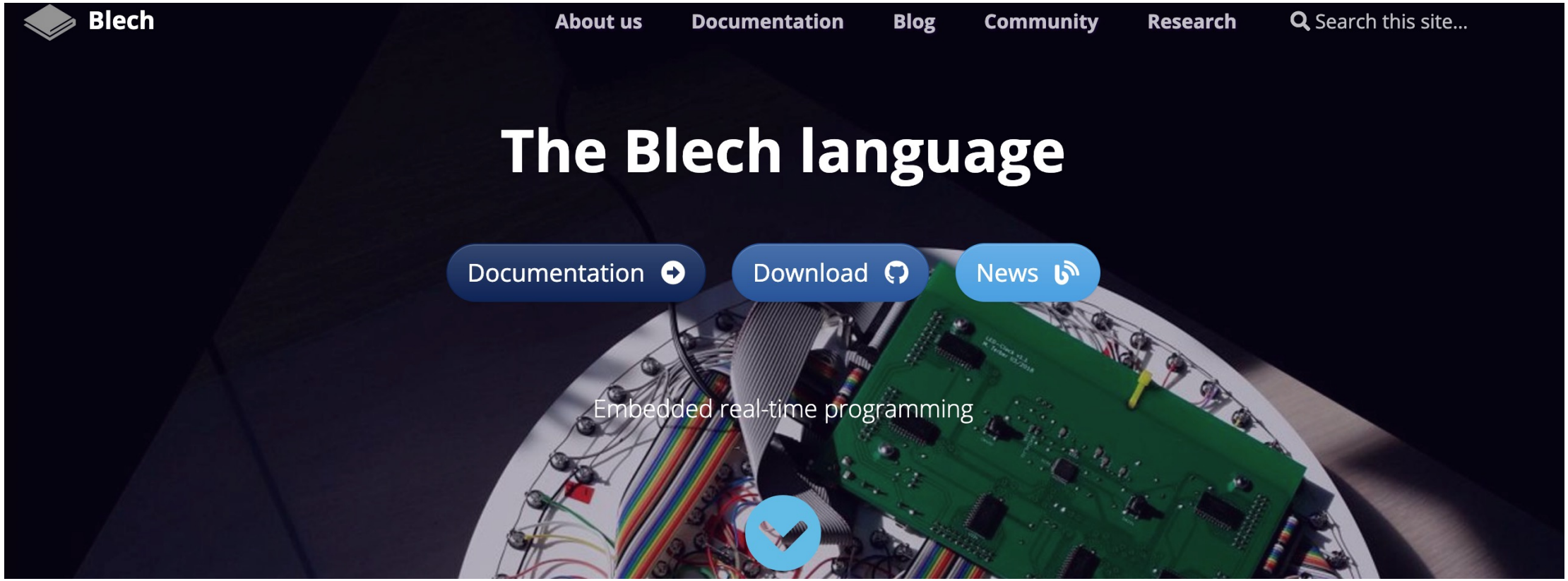
SCCharts – Classroom-Tested

SCCharts – Classroom-Tested



- 10.000 / 135.000 SCChart nodes before/after normalization
- 650.000 lines of C-code
- Compiles in about 2 min’s
- 2 ms reaction time

Another Sequentially-Constructive Language: Blech



www.blech-lang.org

Lucas, Schulz-Rosengarten, von Hanxleden, Gretz, Grosch
Extracting Mode Diagrams from Blech Code
FDL 2021

Take-Home Message

- Small number of core constructs sufficient for reactive control flow!
- On top of that, can build powerful constructs as syntactic sugar

Advantage:

- Can keep core semantics clean and simple
- Can easily adapt to different syntheses (hw + sw)

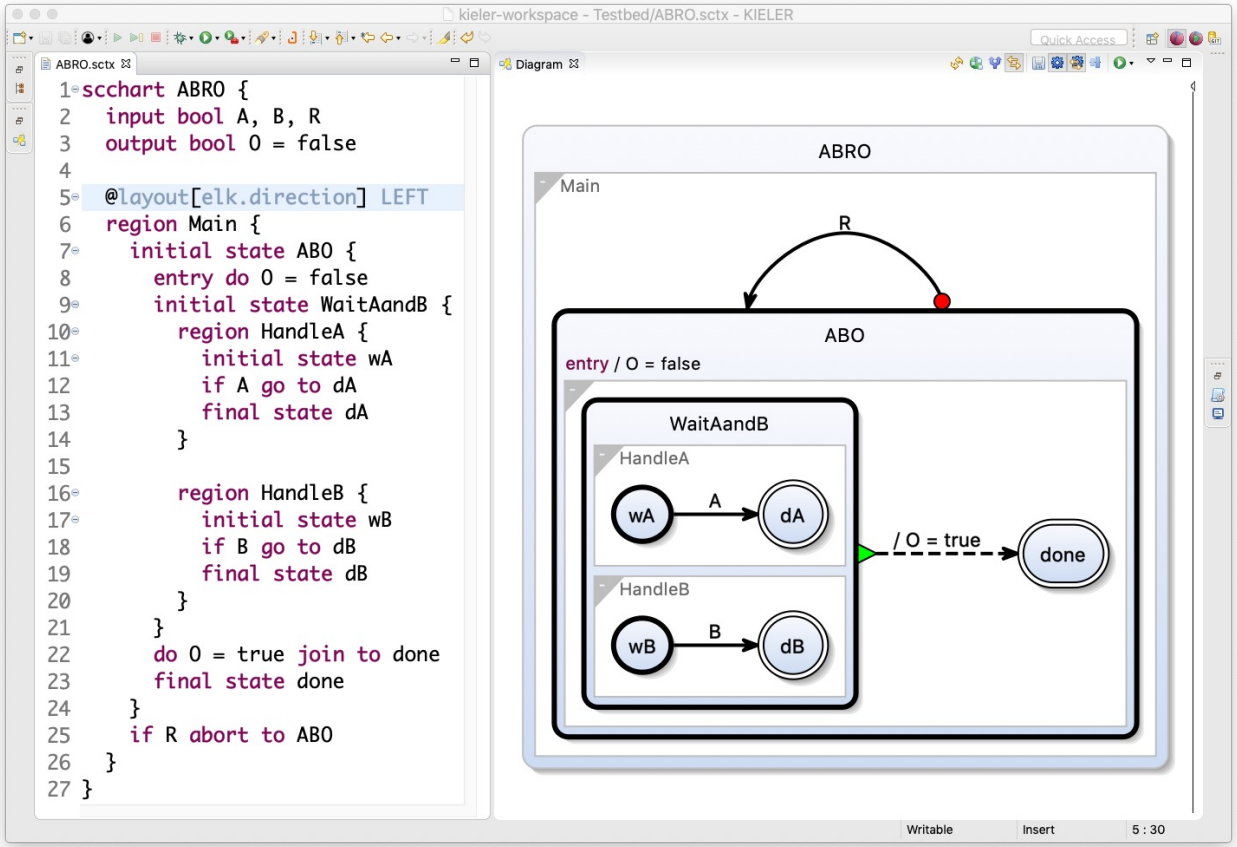
BUT:

- Resulting code difficult to map back to original model!
- This motivated “Interpreter-Approach”
- … which also scales better, as it facilitates module-reuse

Smyth, Motika, von Hanxleden
Synthesizing Manually Verifiable Code for Statecharts
Reactive and Event-based Languages & Systems (REBLS '18)

Text-First Modeling in SCCharts

- SCCharts have textual and graphical syntax
- In KIELER tool, modeler writes textually, tool automatically synthesizes graphical views
- Uses auto-layout from Eclipse Layout Kernel (ELK)



Pragmatics-Aware Modeling

Free user of tedious mechanical work, such as . . .

- manual placing of graphical objects
- manual navigation in complex models

Focus on pragmatics:

- New interaction methodologies
- New analysis methodologies
- New ways to synthesize models

Our experimental platforms:

KIELER

The Key to Efficient Modeling

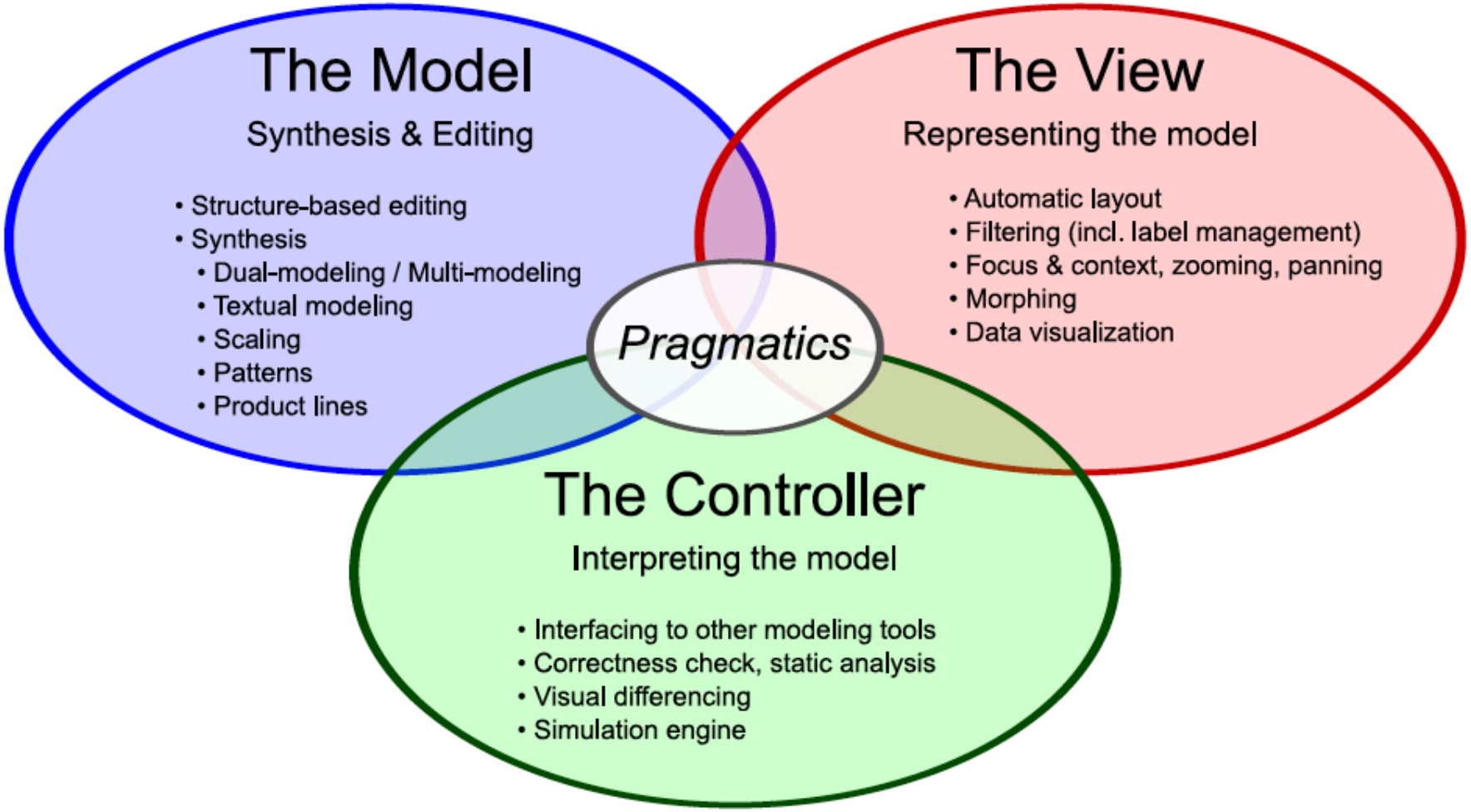
Eclipse Layout Kernel

Key to Pragmatics: The MVC Paradigm

- A **model** represents knowledge.
  A model could be a single object (rather uninteresting),
  or it could be some structure of objects.
- A **view** is a (visual) representation of its model.
  It would ordinarily highlight certain attributes of the model and suppress others.
  It is thus acting as a presentation filter.
- A **controller** is the link between a user and the system.
  It provides the user with input by arranging for relevant views to present themselves in appropriate places on the screen.

Trygve Reenskaug
Models - Views - Controllers
Xerox PARC technical note, 1979

Key to Pragmatics: The MVC Paradigm

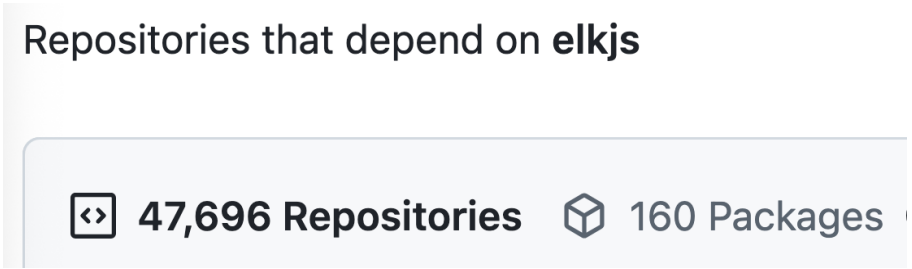


Fuhrmann, von Hanxleden
On the Pragmatics of Model-Based Design
15th Monterey Workshop 2008, LNCS 6028 (2010)

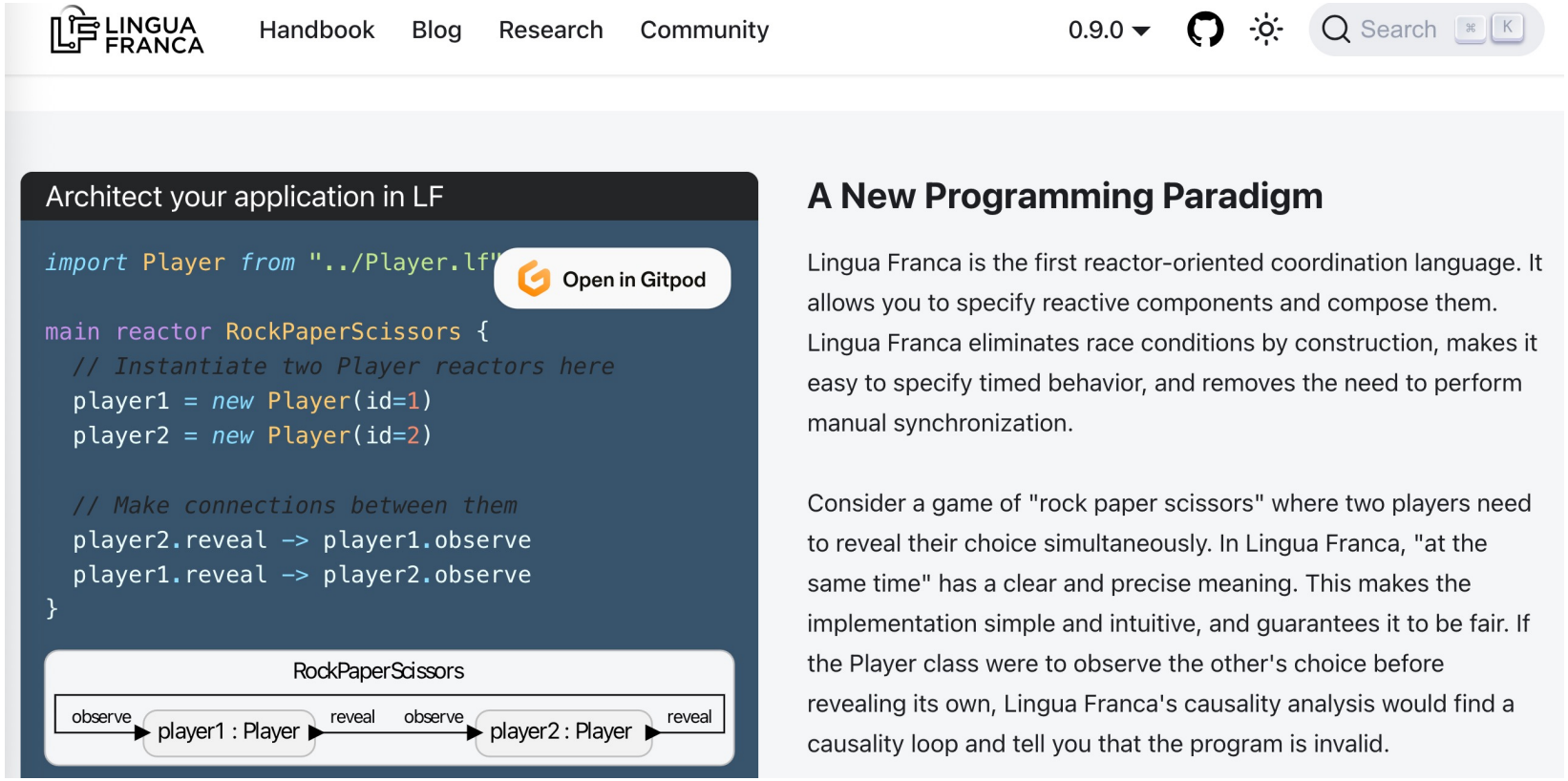
Pragmatics is Catching On …

*In our experience over many years my colleagues and I concluded that **textual modeling** is the only practical way, but that a **graphical view** of the models is a must-have as well. Your technology closes exactly that gap.*

Dr. Andreas Seibel, BSH Hausgeräte GmbH
E-Mail from Oct. 6, 2017



Another Text-First Language: Lingua Franca



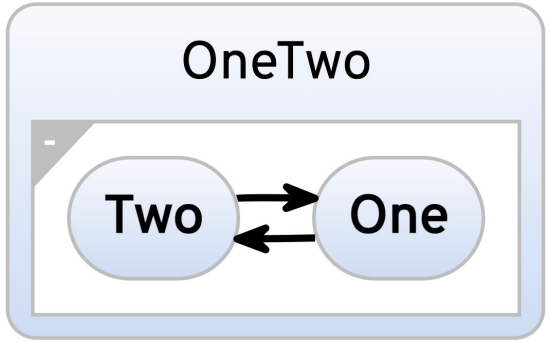
www.lf-lang.org

von Hanxleden, Lee, et al.
Pragmatics Twelve Years Later: A Report on Lingua Franca
ISoLA 2022

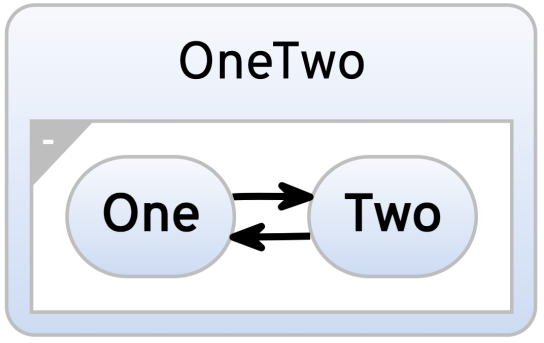
Outlook: Model Order

```
scchart OneTwo {
  state One
  go to Two

  state Two
  go to One
}
```



View



Alternative View

- Both options equally “good” from perspective of automatic layout!
- The problem goes back to the heart of graph drawing
  - A graph is a pair (*V*, *E*), where *V* is a **set** of vertices, *E* is a **set** of edges
- Approach: replace “set” (unordered!) by “list” (ordered!)
- Derive model order from textual input

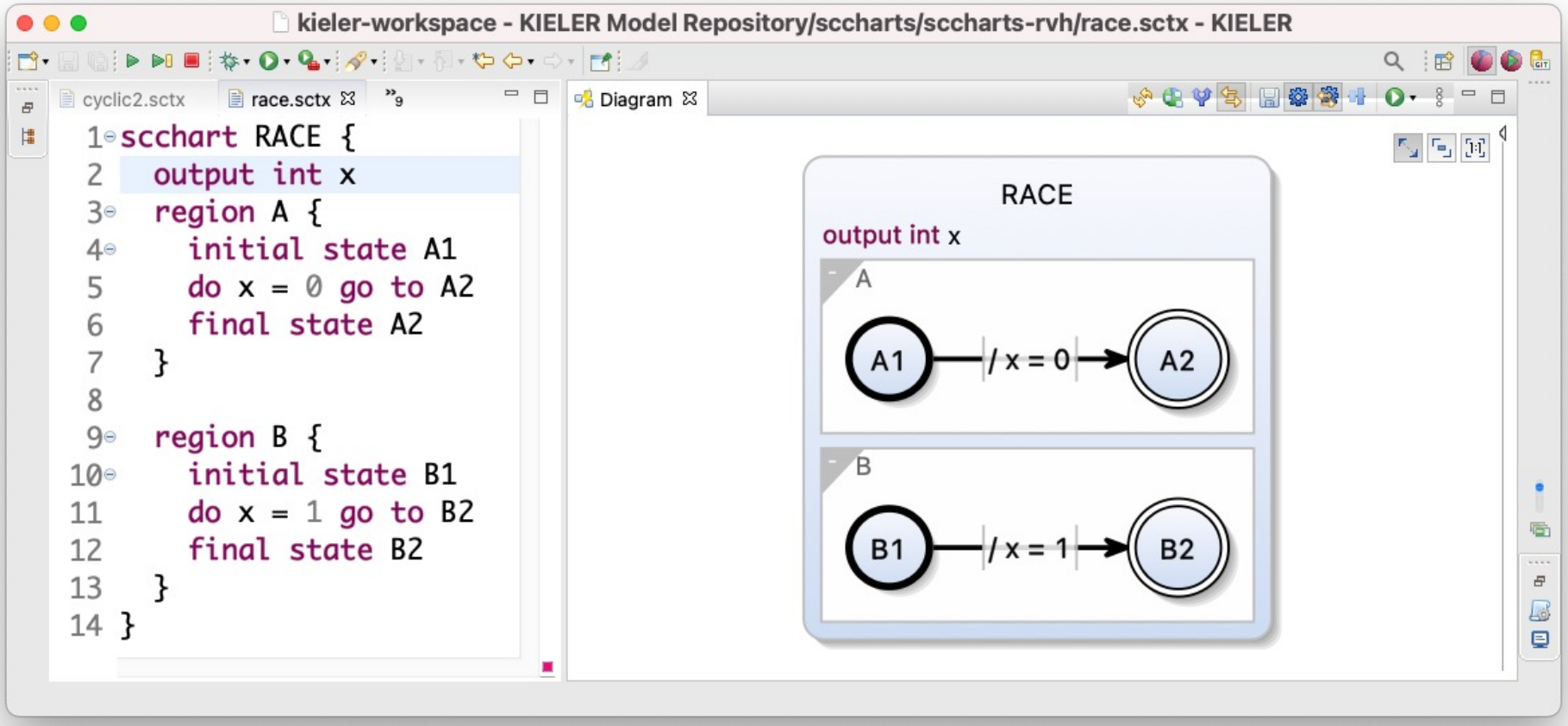
Domrös, Riepe, von Hanxleden
Model Order in Sugiyama Layouts
IVAPP 2023

Take-Home Message

- Automatic layout is **practical**
- Users love …
  - … to **not** spend precious life time on manual layout
  - … to have **control** over how a diagram looks
- Users do not love …
  - … surprising or unstable layouts
  - … having to learn layout options or annotations, it should “just work”
- Should revise definition of **graph**!

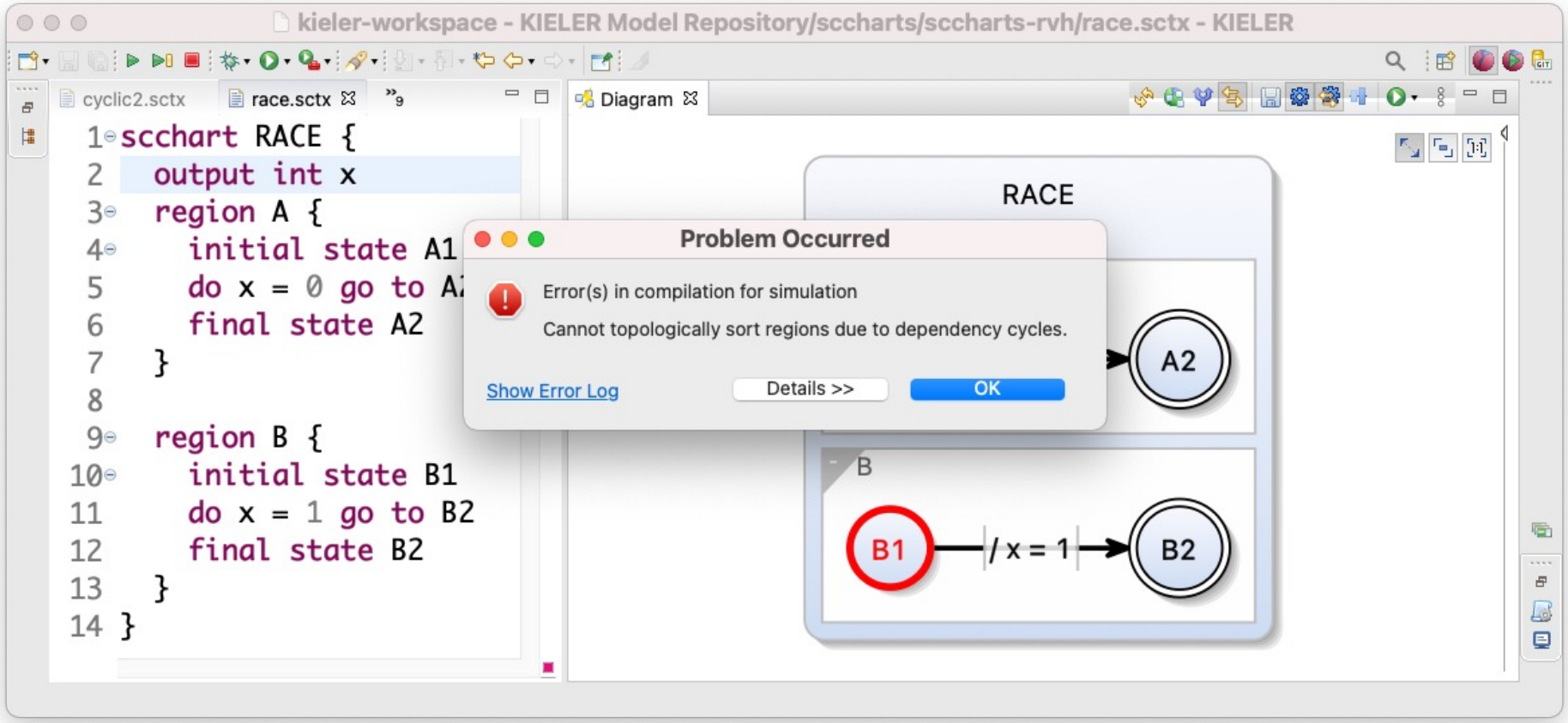
Outlook: Concurrent Sequential Constructiveness

Recall: Concurrent accesses may lead to causality cycles



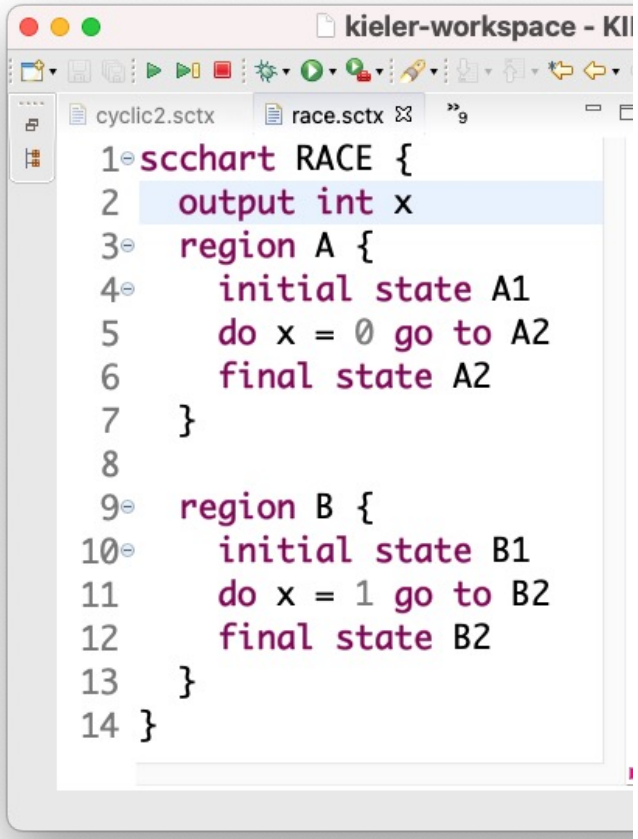
Outlook: Concurrent Sequential Constructiveness

Recall: Concurrent accesses may lead to causality cycles



Outlook: Concurrent Sequential Constructiveness

Recall: Concurrent accesses may lead to causality cycles



- But why not let the order (of regions) prescribe the schedule!
- I.e., within a tick, first schedule region A, then region B
- See e.g. some Statechart dialects, or PRET-C

Andalam, Roop, Girault
Deterministic, predictable and light-weight multithreading using PRET-C
DATE 2010

Outlook: Concurrent Sequential Constructiveness

- Still deterministic
- Still under programmer control

Advantage:

- No more (?) nasty causality issues
- Simpler semantics

The price to pay:

- No back-and-forth scheduling within tick
- But is that *really* a problem? Watch this space …

Bonus: Concurrent Sequential Constructiveness for Esterel

	Esterel	SCEst
`O = 1 || O = 2`	Rejected	~~Rejected~~
`present Done else … emit Done end`	Rejected	Accepted
`emit O(1); emit O(?O + 1)`	Rejected	Accepted
`emit O(1); pause; emit O(pre(?O)+1)`	Accepted	Accepted

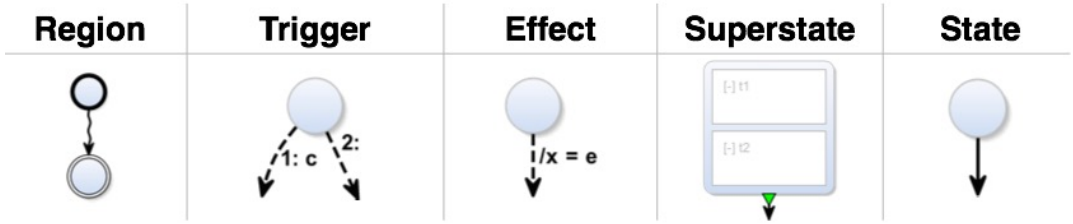
Accepted!

Smyth, Motika, Rathlev, von Hanxleden, Mendler
SCEst: Sequentially Constructive Esterel
ACM TECS 2018 (MEMOCODE 2015)

Wrap-Up

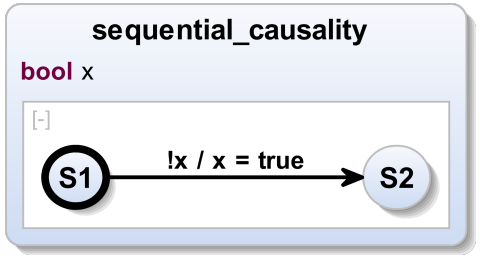
Language

- 5 core constructs
- Smörgåsboard of extensions



Model of Computation

- Relaxed synchrony
- Still determininistic



Text-First Modeling

- KIELER + ELK provide infrastructure
- Model order valuable

KIELER
The Key to Efficient Modeling

Still plenty of things to do: Variants on SC MoC, optimize code generation, pragmatics improvements …

SCCharts – Conclusion

- Sequential constructiveness
  - … is natural for programmers and proven in practice
  - … so far, tricky to formalize precisely
  - … should take even more advantage of **textual order**
  - **Easy control** of scheduling is key!
- Text-first modeling
  - … is natural for programmers and proven in practice
  - … harnesses power of automatic layout
  - … should take even more advantage of **textual order**
  - **Easy control** of layout is key!

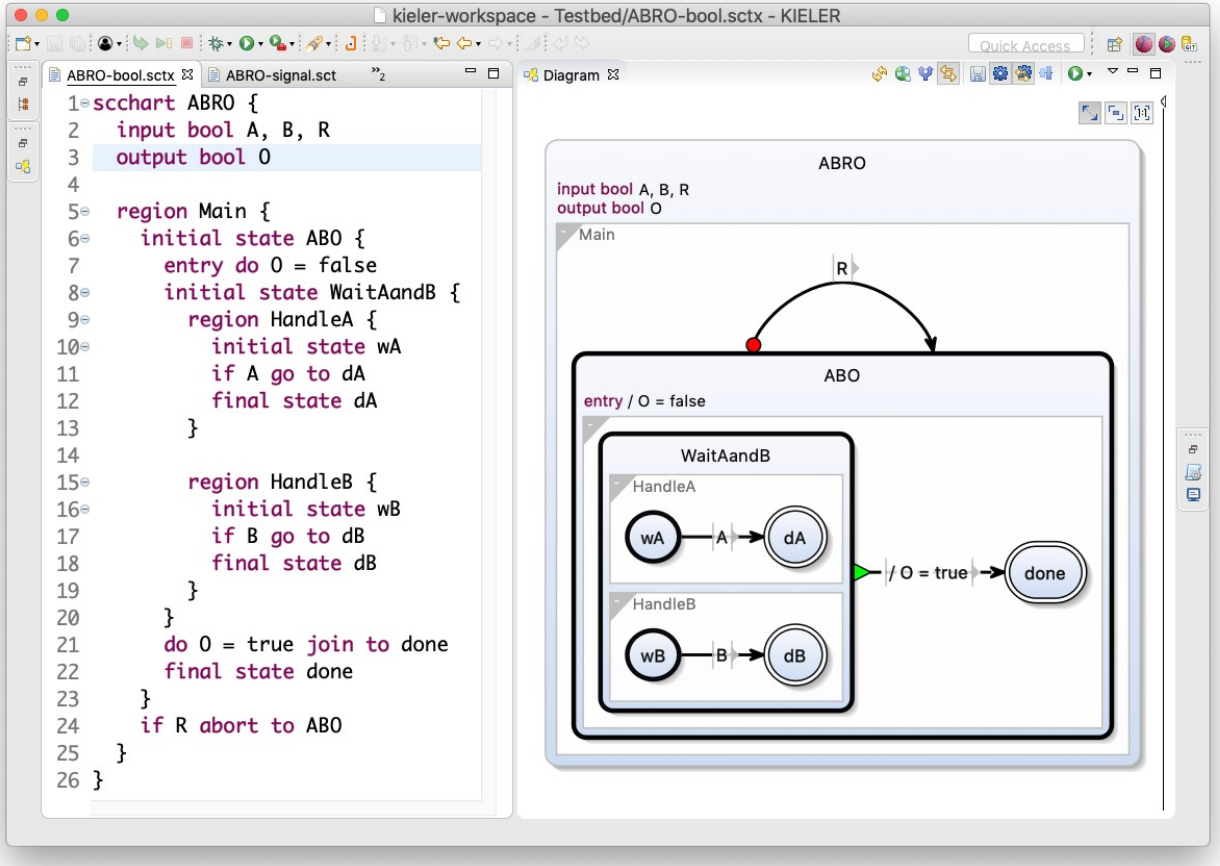
Thank you!

EXTRAS

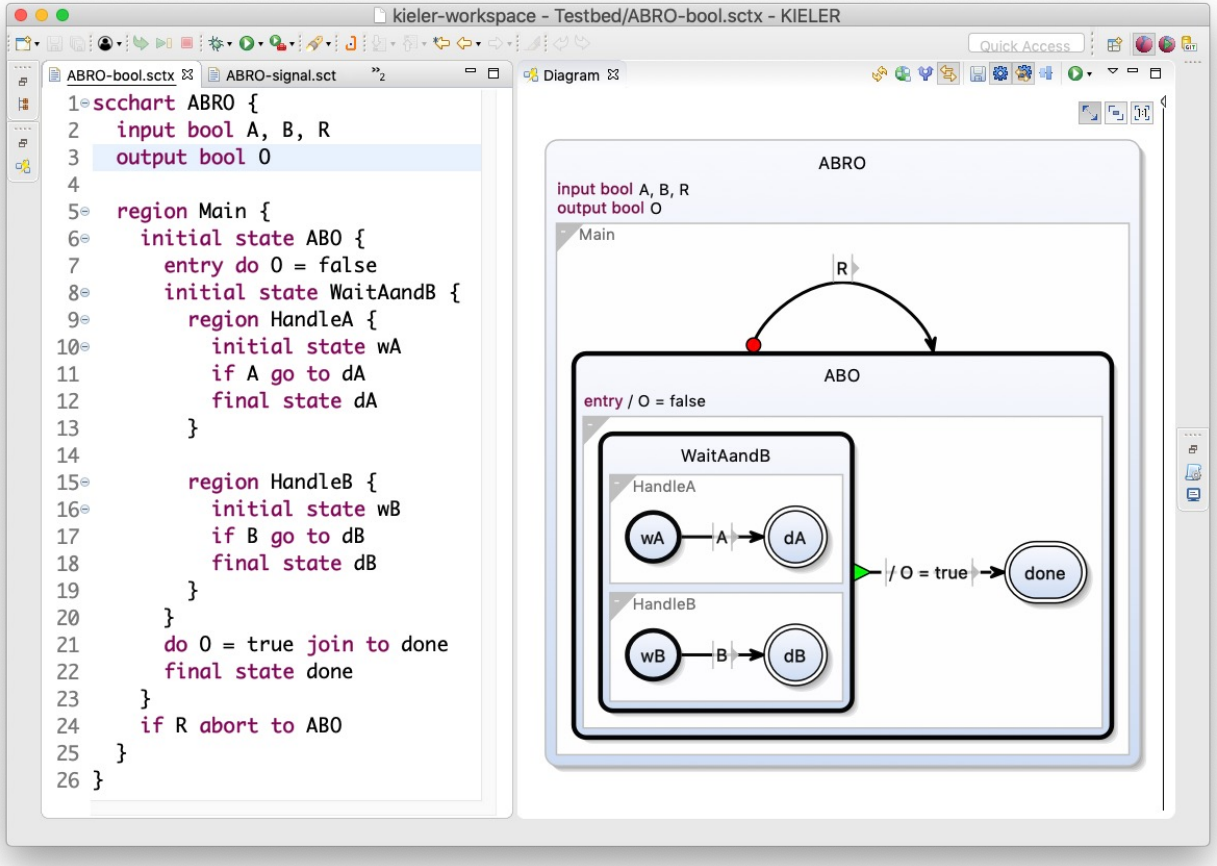
Booleans vs. Signals

`bool`:

- true or false
- Persistent across ticks



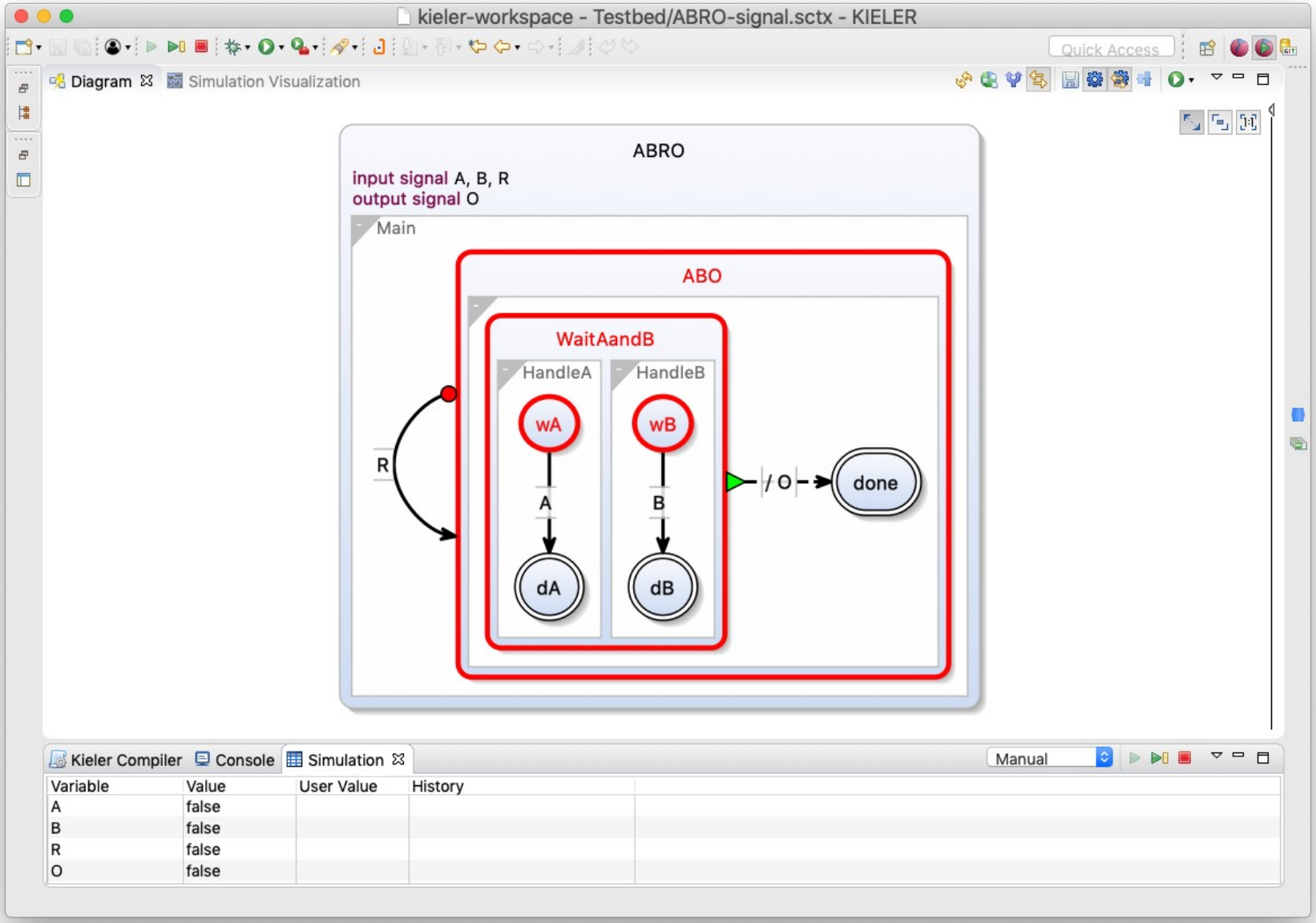
Booleans vs. Signals



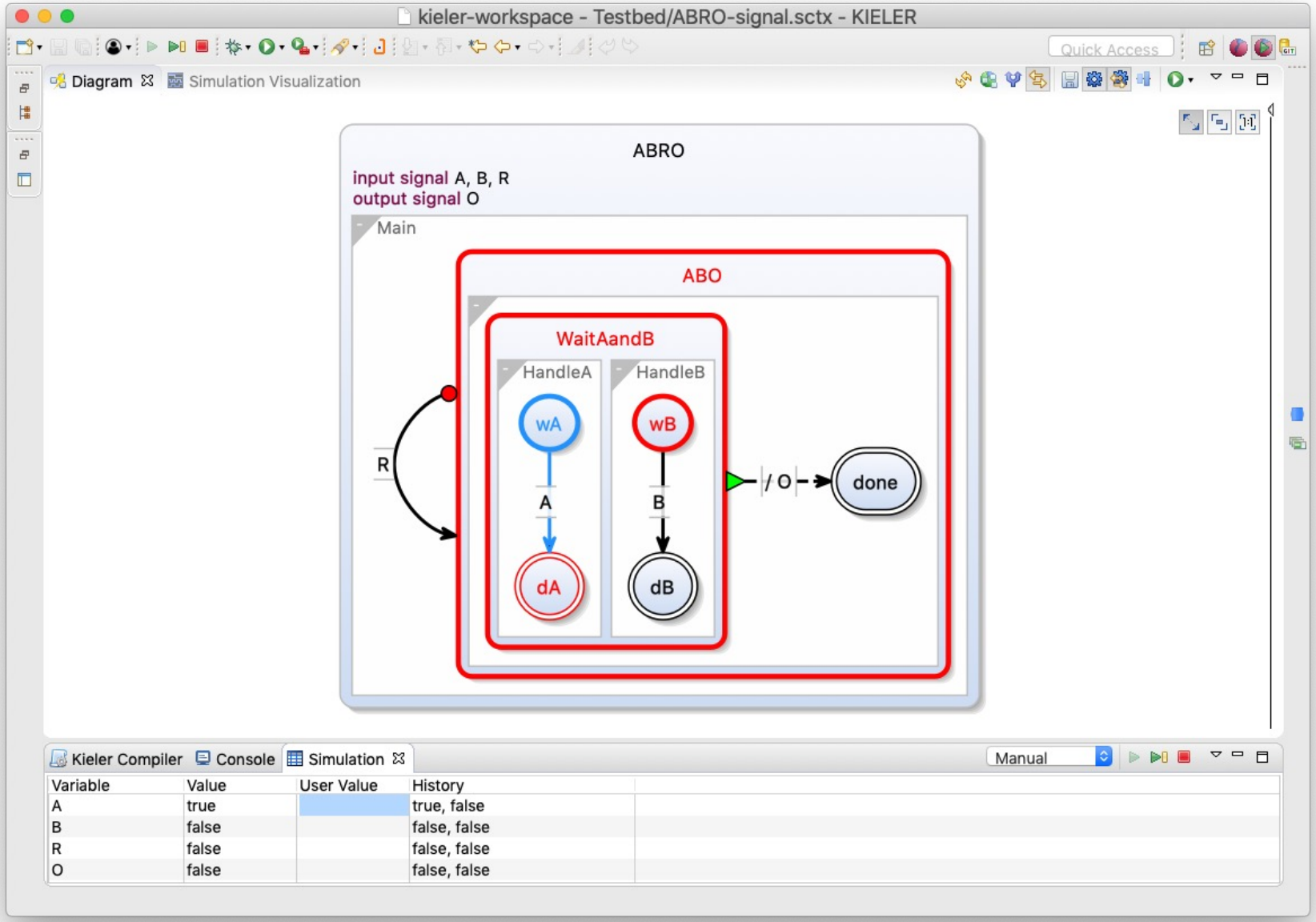
`signal`:

- true (“present”)
  or false (“absent”)
- Re-initialized to absent
  (unless input signal) at each tick
- Conceptually, correspond to *events*
- … and beyond these *pure signals*, there are also *valued signals*,
  which carry a – persistent – value of some type (including bool) …

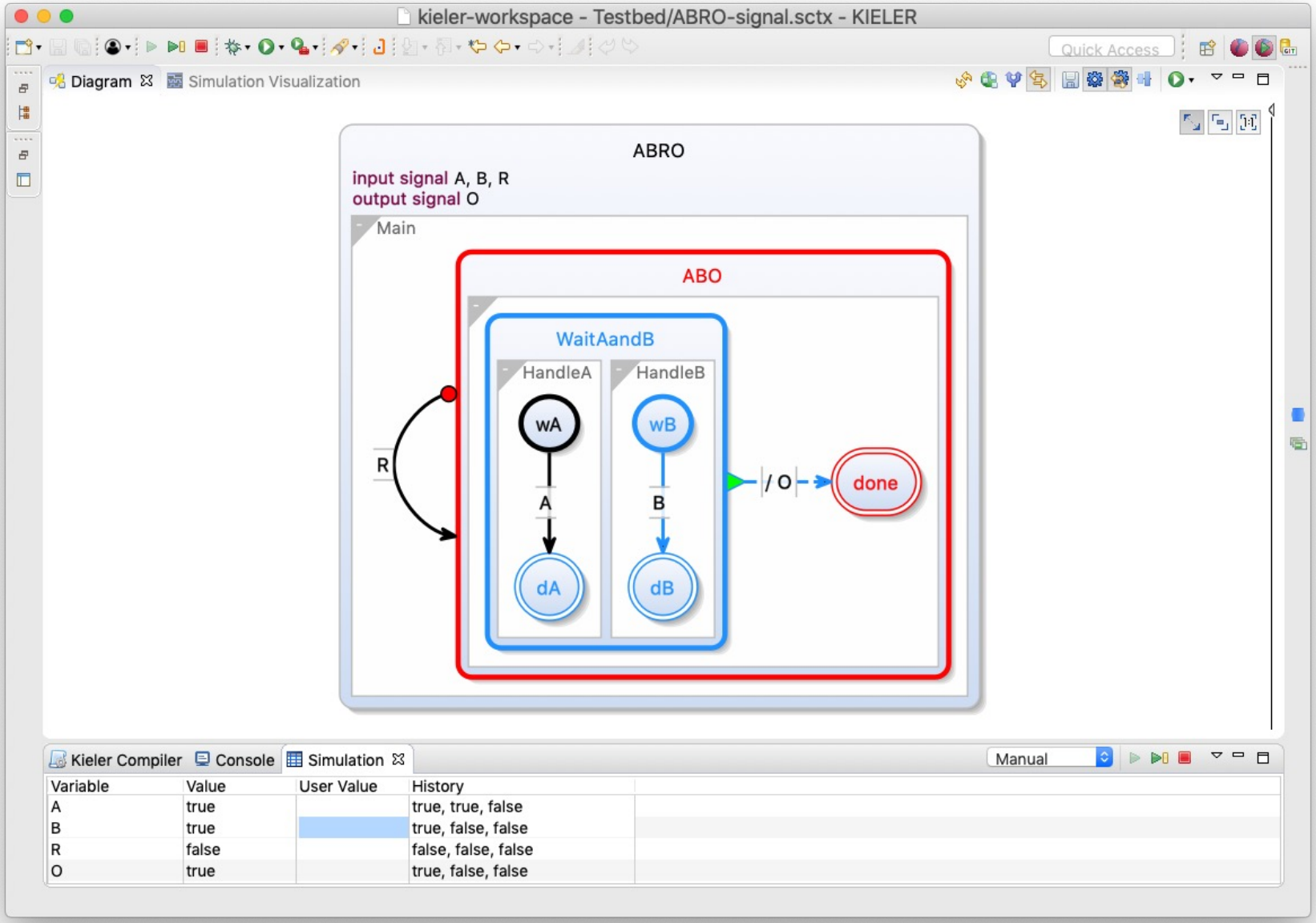
Simulation


kieler-workspace - Testbed/ABRO-signal.sctx - KIELER
Quick Access
Diagram
Simulation Visualization
ABRO
input signal A, B, R
output signal O
Main
ABO
WaitAandB
HandleA
HandleB
wA
wB
R
A
B
/ O
done
dA
dB
Kieler Compiler
Console
Simulation
Manual
Variable
Value
User Value
History
A
false
B
false
R
false
O
false

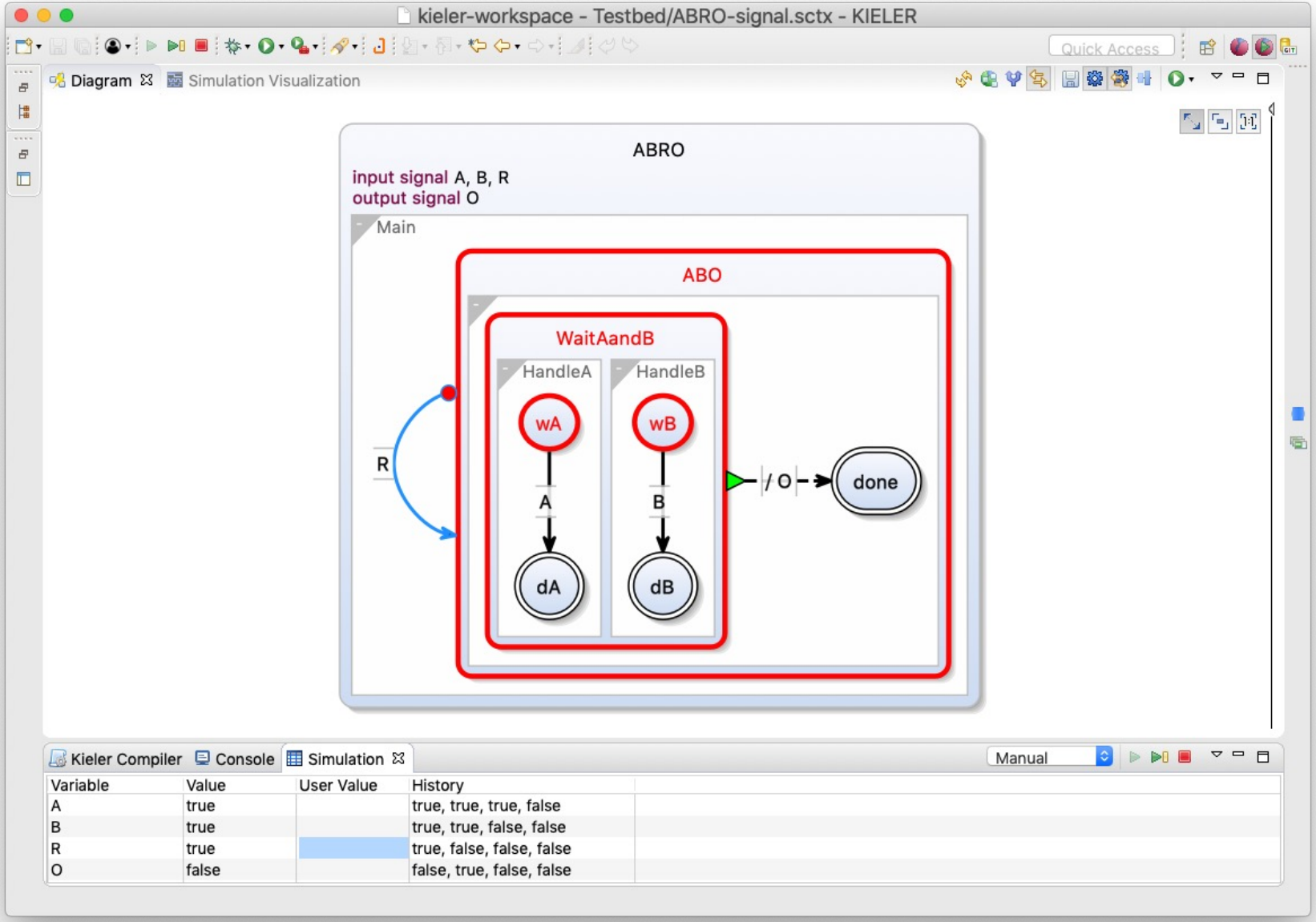
Simulation


kieler-workspace - Testbed/ABRO-signal.sctx - KIELER
Quick Access
Diagram
Simulation Visualization
ABRO
input signal A, B, R
output signal O
Main
ABO
WaitAandB
HandleA
HandleB
wA
wB
R
A
B
/ O
done
dA
dB
Kieler Compiler
Console
Simulation
Manual
Variable
Value
User Value
History
A
true
true, false
B
false
false, false
R
false
false, false
O
false
false, false

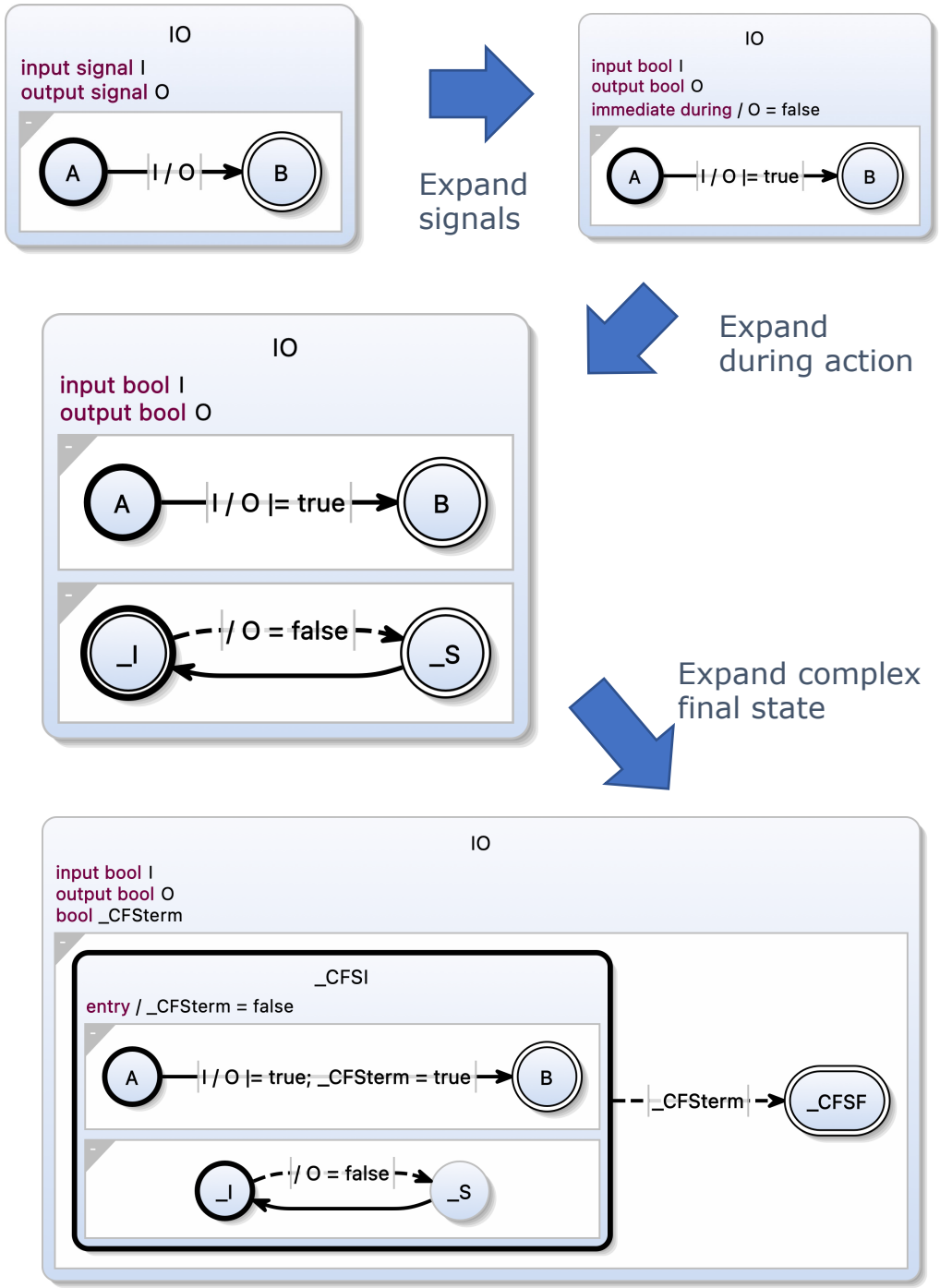
Simulation

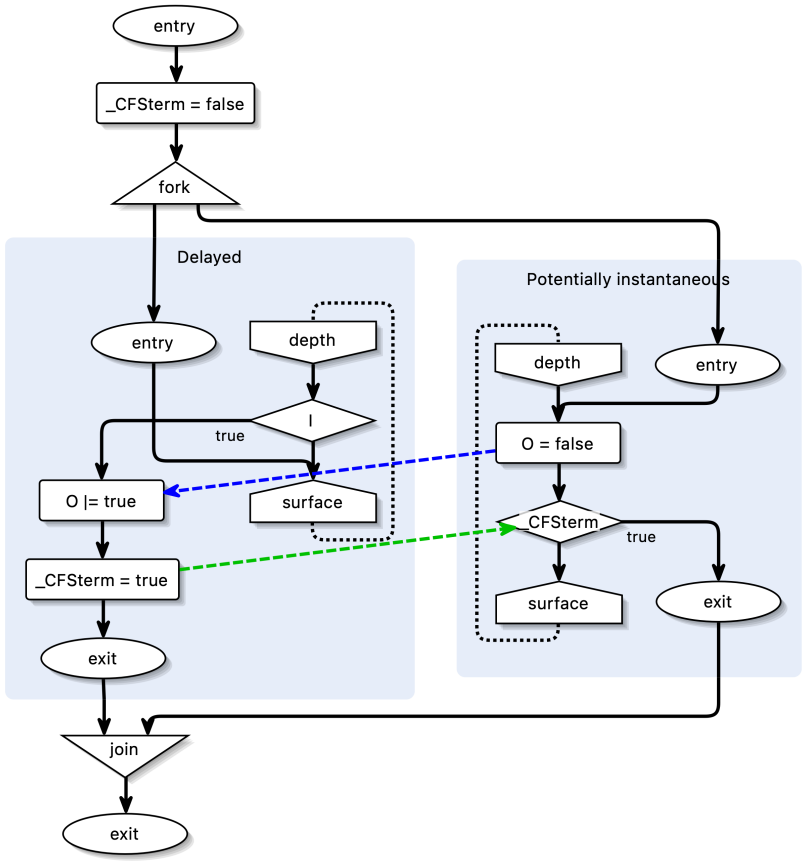

kieler-workspace - Testbed/ABRO-signal.sctx - KIELER
Quick Access
Diagram
Simulation Visualization
ABRO
input signal A, B, R
output signal O
Main
ABO
WaitAandB
HandleA
HandleB
wA
wB
R
A
B
/ O
done
dA
dB
Kieler Compiler
Console
Simulation
Manual
Variable
Value
User Value
History
A
true
true, true, false
B
true
true, false, false
R
false
false, false, false
O
true
true, false, false

Simulation


kieler-workspace - Testbed/ABRO-signal.sctx - KIELER
Quick Access
Diagram
Simulation Visualization
ABRO
input signal A, B, R
output signal O
Main
ABO
WaitAandB
HandleA
HandleB
wA
wB
R
A
B
/ O
done
dA
dB
Kieler Compiler
Console
Simulation
Manual
Variable | Value | User Value | History
A | true | | true, true, true, false
B | true | | true, true, false, false
R | true | | true, false, false, false
O | false | | false, true, false, false

Inits/Updates – Enable Signals!

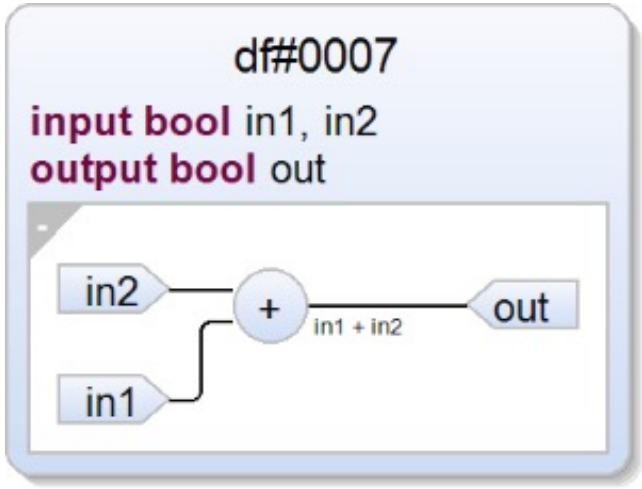




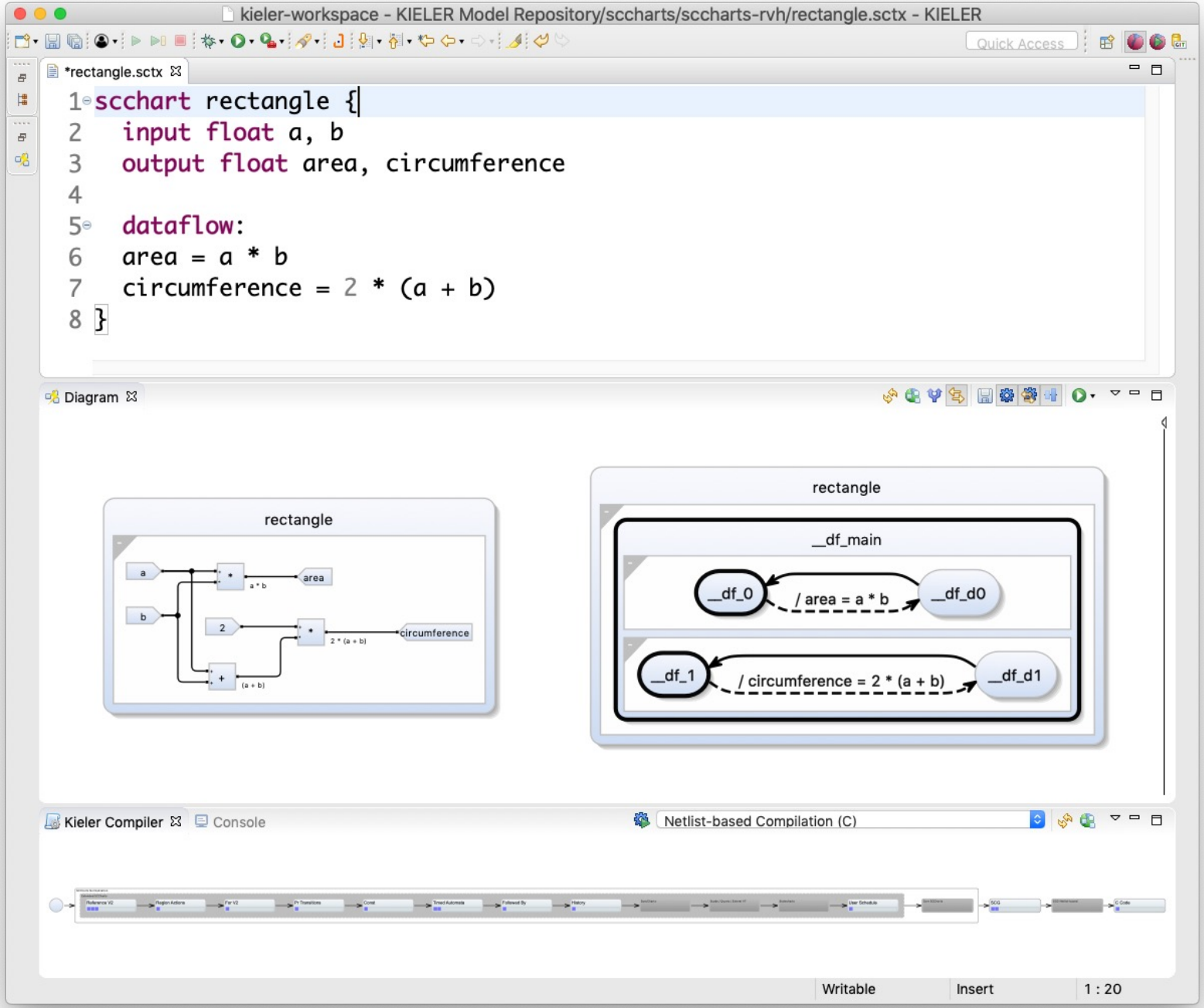
Dataflow SCCharts

```
scchart add {
  input bool in1, in2
  output bool out

  dataflow:
  out = in1 + in2
}
```



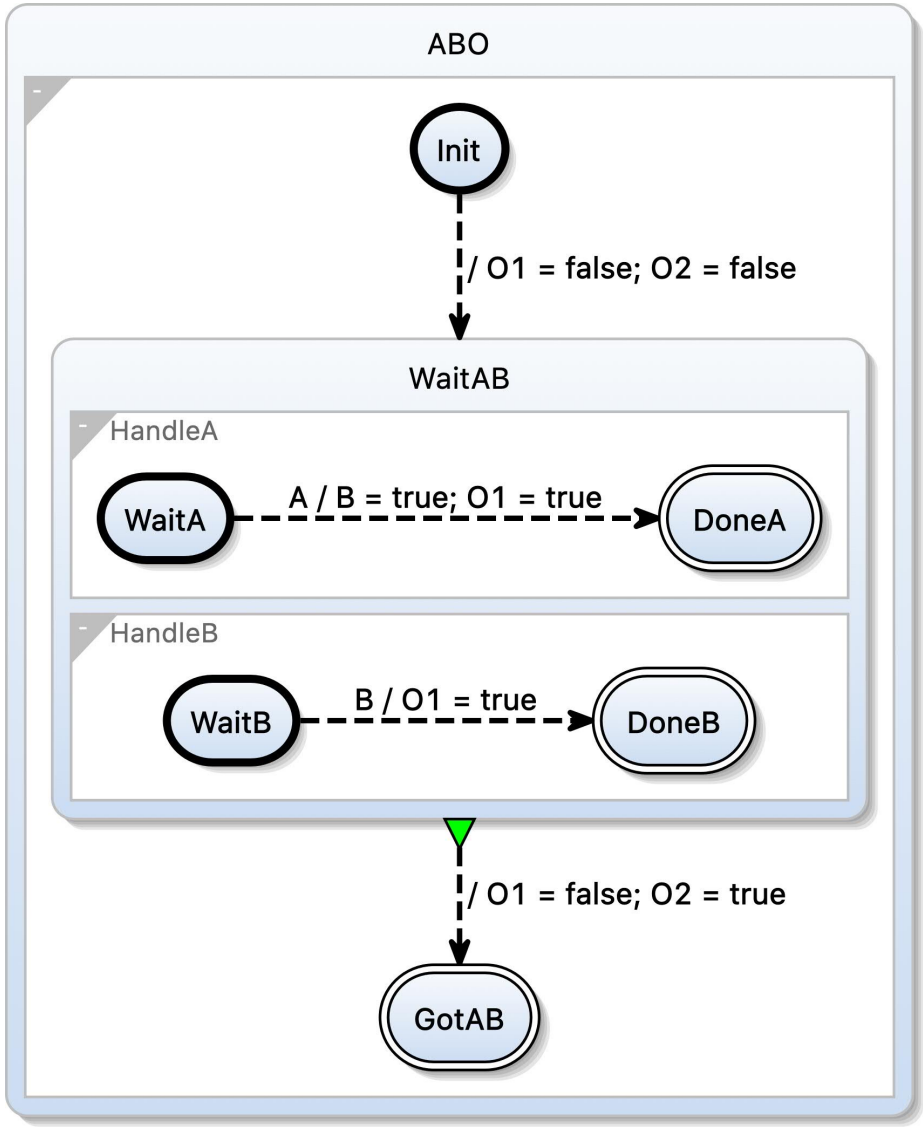
Semantically, dataflow equations correspond to concurrent assignments,
as in immediate during actions,
following iur-scheduling


kieler-workspace - KIELER Model Repository/sccharts/sccharts-rvh/rectangle.sctx - KIELER
Quick Access
*rectangle.sctx
1 scchart rectangle {
2 input float a, b
3 output float area, circumference
4
5 dataflow:
6 area = a * b
7 circumference = 2 * (a + b)
8 }
Diagram
rectangle
a
b
2
area
circumference
a * b
2 * (a + b)
(a + b)
rectangle
__df_main
__df_0
/ area = a * b
__df_d0
__df_1
/ circumference = 2 * (a + b)
__df_d1
Kieler Compiler
Console
Netlist-based Compilation (C)
Writable
Insert
1 : 20

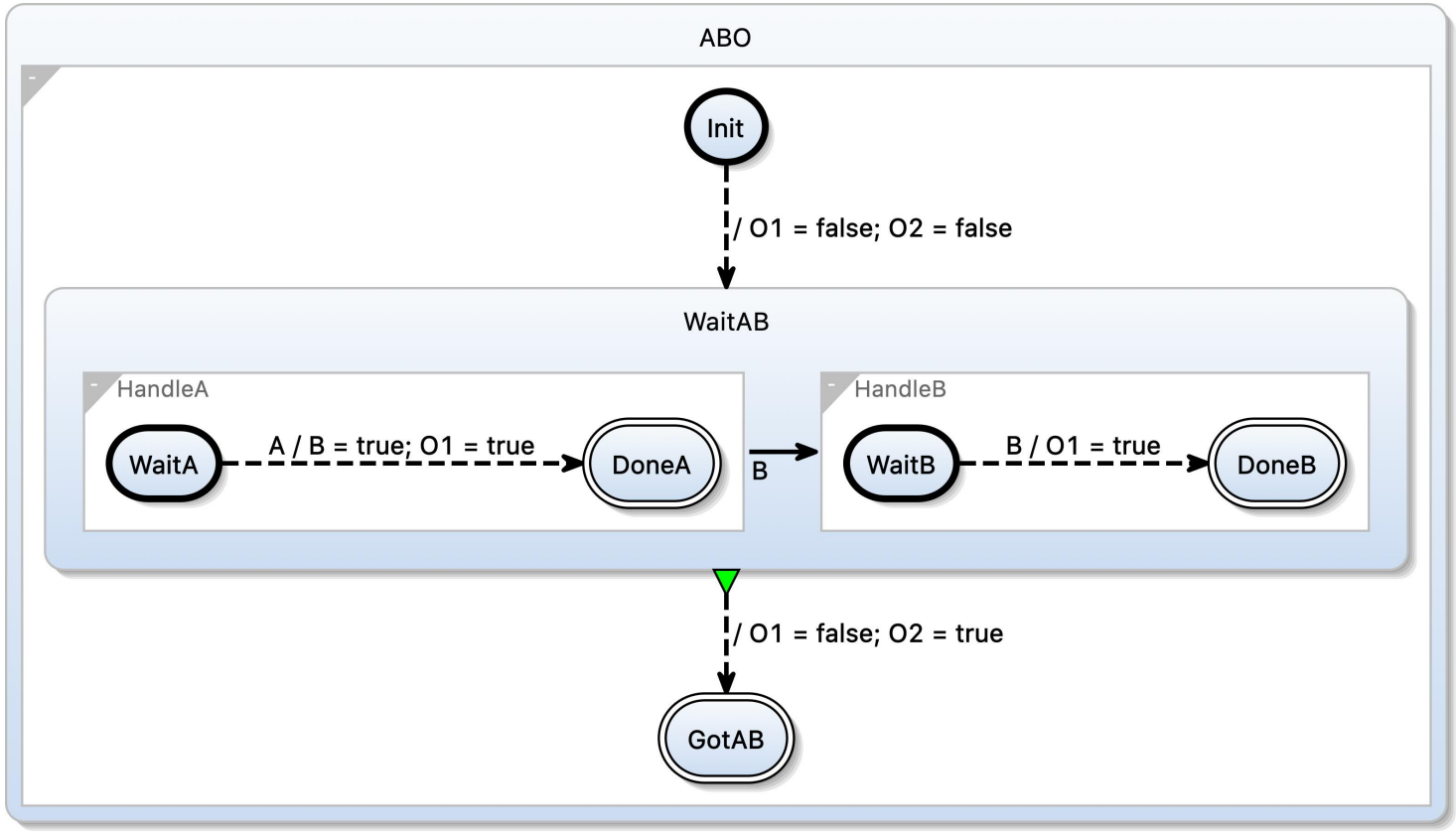
More Language Features

- The semantic kernel of SCCharts has been stable since beginning
- Beyond kernel, have host language interaction, for regions, reference charts/inheritance, dataflow, ...
- Like other languages (e.g., Java), the feature set of SCCharts keeps evolving – also based on your input!
- When using SCCharts, should consult current documentation and experiment with different features to see what might suit you best

Induced Dataflow

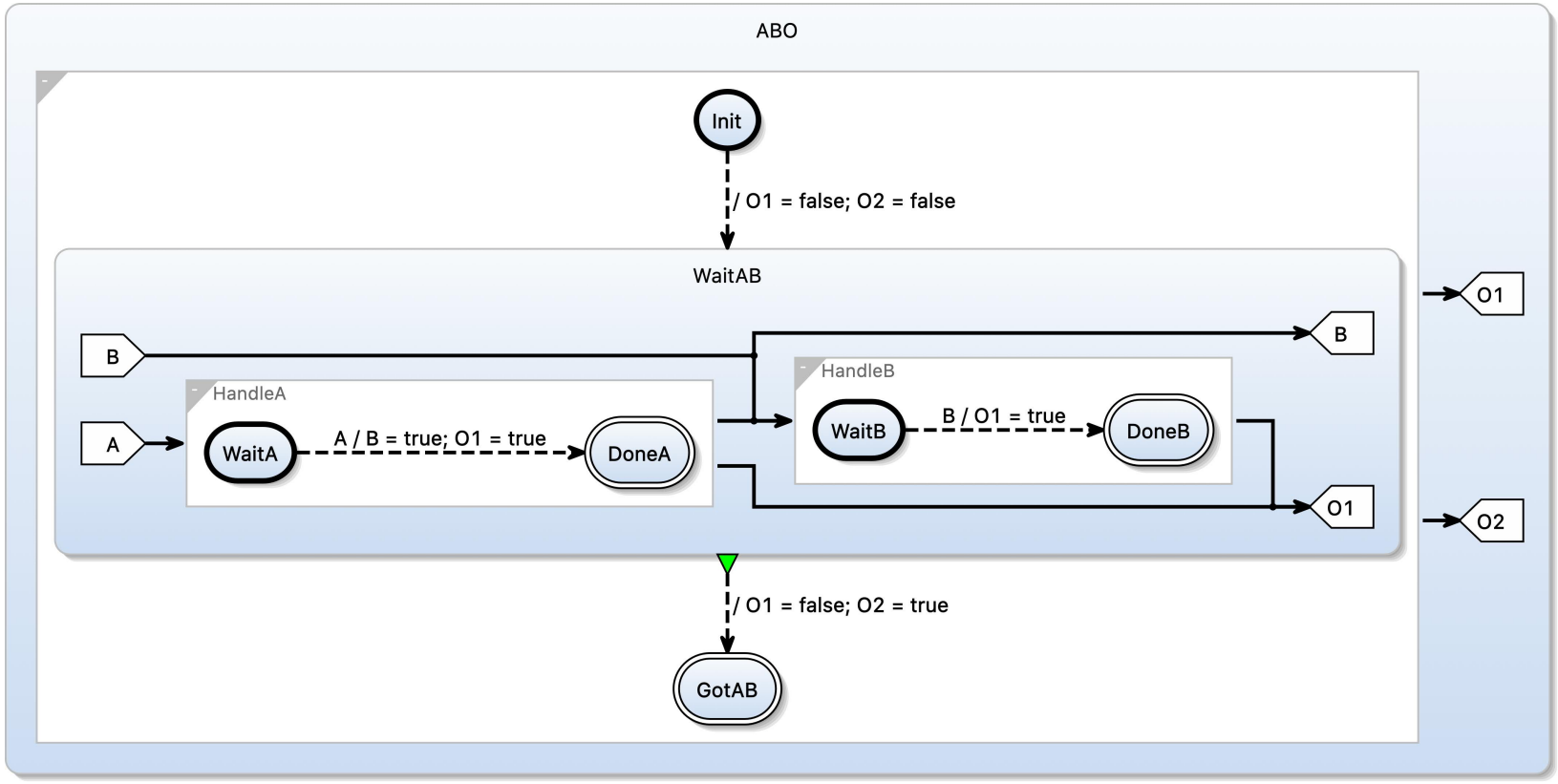


Induced Dataflow



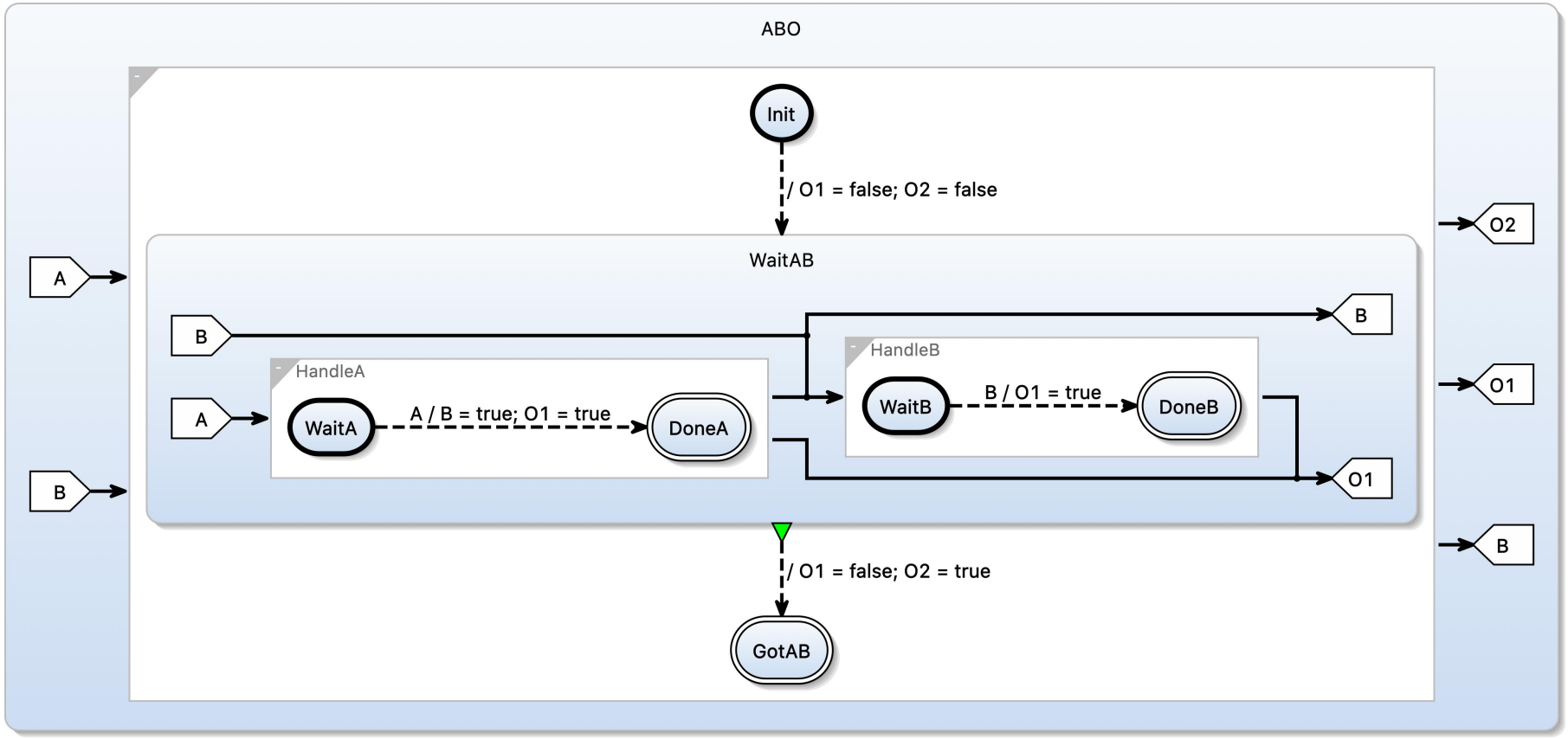
Wechselberg, Schulz-Rosengarten, Smyth, von Hanxleden
Augmenting State Models with Data Flow
Principles of Modeling: Essays Dedicated to Edward A. Lee on the Occasion of His 60th Birthday, Springer, 2018

Induced Dataflow



Wechselberg, Schulz-Rosengarten, Smyth, von Hanxleden
Augmenting State Models with Data Flow
Principles of Modeling: Essays Dedicated to Edward A. Lee on the Occasion of His 60th Birthday, Springer, 2018

Induced Dataflow



Wechselberg, Schulz-Rosengarten, Smyth, von Hanxleden
Augmenting State Models with Data Flow
Principles of Modeling: Essays Dedicated to Edward A. Lee on the Occasion of His 60th Birthday, Springer, 2018

Customized Skins

```
import "DF-0007"
#skinpath "skin"

scchart df#1000 {
  input int I, I2
  output int O
  ref df#0007 A, A2

  dataflow:
  A = {true, false}
  A2 = {true, A.out}
  O = A2.out
}
```