

SCCharts: Sequentially Constructive Statecharts for Safety-Critical Applications

Reinhard von Hanxleden¹, Björn Duderstadt¹, Christian Motika¹,
Steven Smyth¹, Michael Mendler², Joaquin Aguado²,
Stephen Mercer³, and Owen O'Brien³

¹Christian-Albrechts-Universität zu Kiel

²Bamberg University

³National Instruments

June 9–11, 2014, Edinburgh, UK

A Scheduling Problem: Parallelization



von Hanxleden, Kennedy

Relaxing SIMD Control Flow Constraints Using Loop
Transformations

PLDI'92



von Hanxleden, Kennedy

Give-N-Take — A Balanced Code Placement
Framework

PLDI'94

A Scheduling Problem: Parallelization



[von Hanxleden, Kennedy](#)

Relaxing SIMD Control Flow Constraints Using Loop
Transformations
PLDI'92



[von Hanxleden, Kennedy](#)

Give-N-Take — A Balanced Code Placement
Framework
PLDI'94



wikipedia

Ken Kennedy
(1945 – 2007)

A Scheduling Problem: Parallelization



von Hanxleden, Kennedy

Relaxing SIMD Control Flow Constraints Using Loop
Transformations

PLDI'92



von Hanxleden, Kennedy

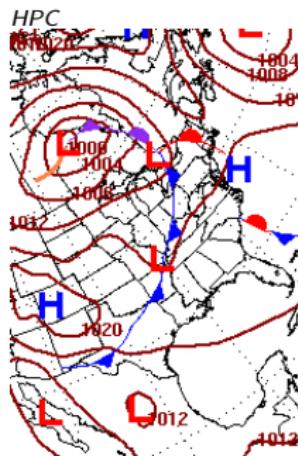
Give-N-Take — A Balanced Code Placement
Framework

PLDI'94



wikipedia

Ken Kennedy
(1945 – 2007)



A Scheduling Problem: Parallelization



von Hanxleden, Kennedy

Relaxing SIMD Control Flow Constraints Using Loop
Transformations

PLDI'92



von Hanxleden, Kennedy

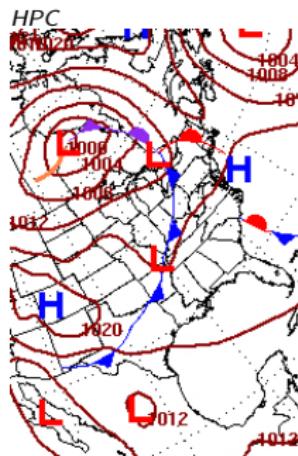
Give-N-Take — A Balanced Code Placement
Framework

PLDI'94



wikipedia

Ken Kennedy
(1945 – 2007)



Starting point: Sequential program
Semantics: Given by “;”

A Scheduling Problem: Parallelization



von Hanxleden, Kennedy

Relaxing SIMD Control Flow Constraints Using Loop
Transformations

PLDI'92



von Hanxleden, Kennedy

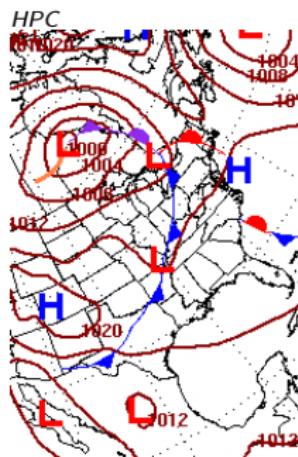
Give-N-Take — A Balanced Code Placement
Framework

PLDI'94



wikipedia

Ken Kennedy
(1945 – 2007)



Starting point: Sequential program

Semantics: Given by ";"

Compiler: Eliminates (some) ;'s

Result: Concurrent program

A Scheduling Problem: Parallelization



von Hanxleden, Kennedy

Relaxing SIMD Control Flow Constraints Using Loop
Transformations

PLDI'92



von Hanxleden, Kennedy

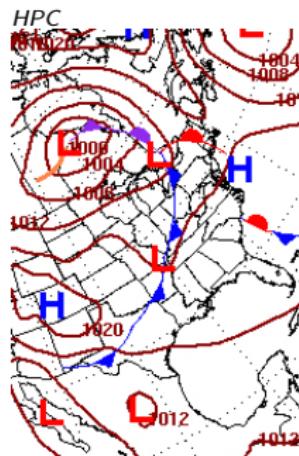
Give-N-Take — A Balanced Code Placement
Framework

PLDI'94



wikipedia

Ken Kennedy
(1945 – 2007)



Starting point: Sequential program

Semantics: Given by “;”

Compiler: Eliminates (some) ;’s

Result: Concurrent program

How much speedup do we get?

Another Scheduling Problem: Statecharts



Harel

Statecharts: A Visual
Formalism for Complex
Systems

Science of Computer
Programming '87

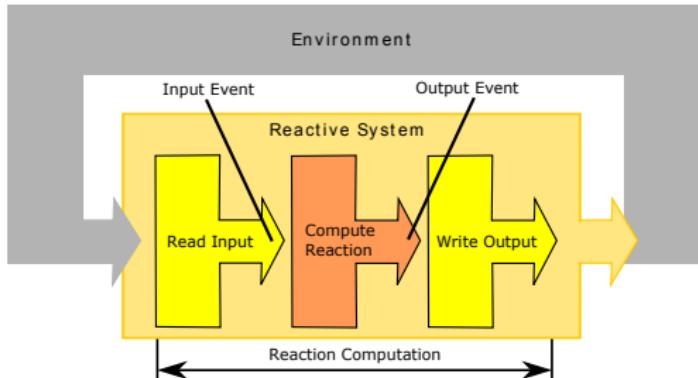
Another Scheduling Problem: Statecharts



Harel

Statecharts: A Visual
Formalism for Complex
Systems

Science of Computer
Programming '87



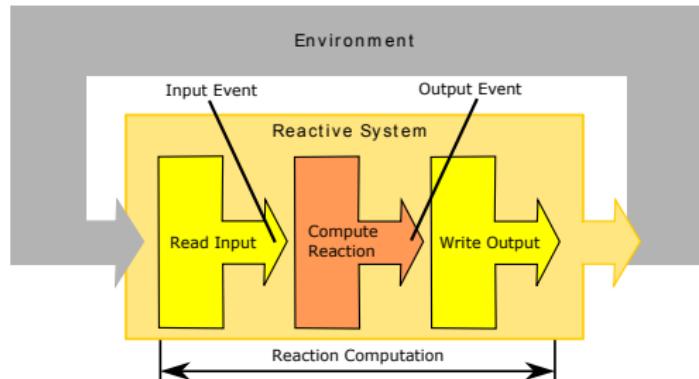
Another Scheduling Problem: Statecharts



Harel

Statecharts: A Visual
Formalism for Complex
Systems

Science of Computer
Programming '87



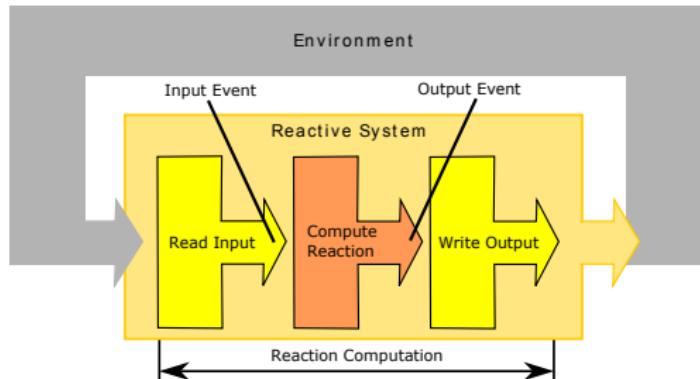
Another Scheduling Problem: Statecharts



Harel

Statecharts: A Visual
Formalism for Complex
Systems

Science of Computer
Programming '87



Starting point: Concurrent Statechart
Semantics: Synchronous (?)

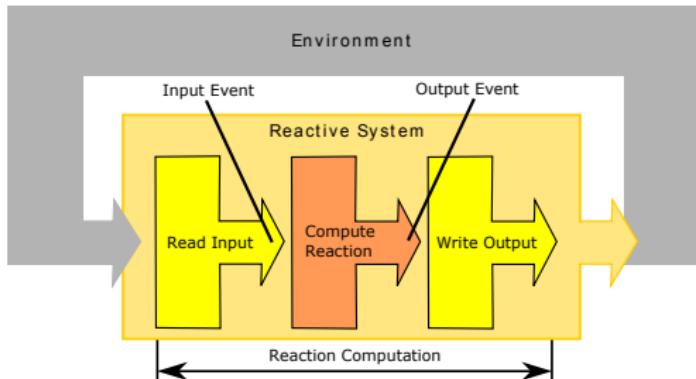
Another Scheduling Problem: Statecharts



Harel

Statecharts: A Visual
Formalism for Complex
Systems

Science of Computer
Programming '87



Starting point: Concurrent Statechart

Semantics: Synchronous (?)

Compiler: Inserts ;'s

Result: Sequential program

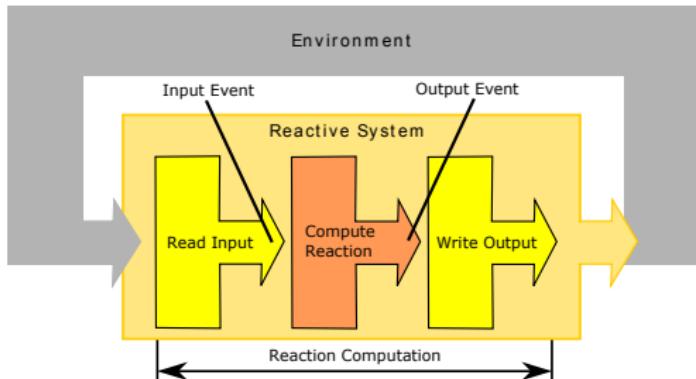
Another Scheduling Problem: Statecharts



Harel

Statecharts: A Visual
Formalism for Complex
Systems

Science of Computer
Programming '87



Starting point: Concurrent Statechart

Semantics: Synchronous (?)

Compiler: Inserts ;'s

Result: Sequential program

How do we get this deterministic?

Add Synchronous Languages



Berry, Gonthier

The Esterel Synchronous
Programming Language:
Design, Semantics,
Implementation
Science of Computer
Programming '92



André

Computing SyncCharts
Reactions
ENTCS '94

Add Synchronous Languages



Berry, Gonthier

The Esterel Synchronous
Programming Language:
Design, Semantics,
Implementation
Science of Computer
Programming '92

Starting point: Concurrent
Statechart

Semantics: **Synchronous (!),
deterministic**



André

Computing SyncCharts
Reactions
ENTCS '94

Add Synchronous Languages



Berry, Gonthier

The Esterel Synchronous
Programming Language:
Design, Semantics,
Implementation
Science of Computer
Programming '92



André

Computing SyncCharts
Reactions
ENTCS '94

Starting point: Concurrent
Statechart

Semantics: **Synchronous (!),
deterministic**

Compiler: **Rejects some ;'s**

Result: Sequential program

Add Synchronous Languages



Berry, Gonthier

The Esterel Synchronous
Programming Language:
Design, Semantics,
Implementation
Science of Computer
Programming '92



André

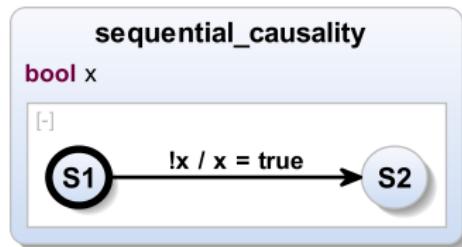
Computing SyncCharts
Reactions
ENTCS '94

Starting point: Concurrent
Statechart

Semantics: **Synchronous (!),
deterministic**

Compiler: **Rejects some ;'s**

Result: Sequential program



Add Synchronous Languages



Berry, Gonthier

The Esterel Synchronous
Programming Language:
Design, Semantics,
Implementation
Science of Computer
Programming '92



André

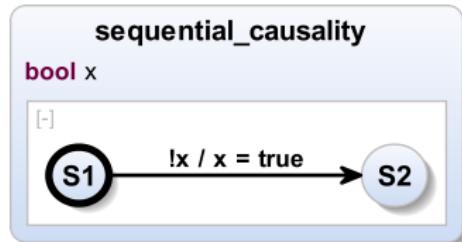
Computing SyncCharts
Reactions
ENTCS '94

Starting point: Concurrent
Statechart

Semantics: **Synchronous (!),
deterministic**

Compiler: **Rejects some ;'s**

Result: Sequential program



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

Add Synchronous Languages



Berry, Gonthier

The Esterel Synchronous Programming Language:
Design, Semantics,
Implementation
Science of Computer Programming '92



André

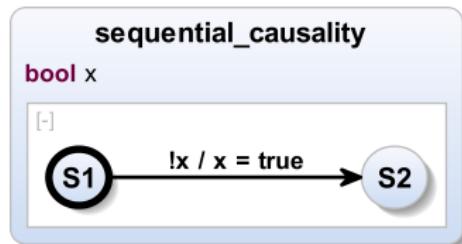
Computing SyncCharts
Reactions
ENTCS '94

Starting point: Concurrent Statechart

Semantics: **Synchronous (!), deterministic**

Compiler: **Rejects some ;'s**

Result: Sequential program



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

This is deterministic — but gets rejected under strict synchrony!

Add Sequential Constructiveness



von Hanxleden, Mendler, et al.

Sequentially Constructive Concurrency—A Conservative Extension of the Synchronous Model of Computation

ACM TECS '14

SCL

SCG

Add Sequential Constructiveness



von Hanxleden, Mendler, et al.

Sequentially Constructive Concurrency—A Conservative Extension of the Synchronous Model of Computation

ACM TECS '14

	Thread	Concurrency
SCL	t	fork t_1 par t_2 join
SCG	 entry exit	 fork join

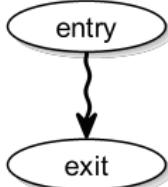
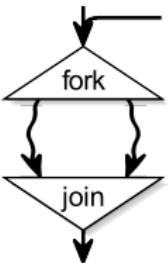
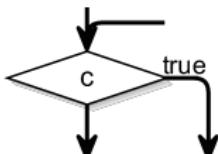
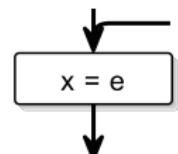
Add Sequential Constructiveness



von Hanxleden, Mendler, et al.

Sequentially Constructive Concurrency—A Conservative Extension of the Synchronous Model of Computation

ACM TECS '14

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

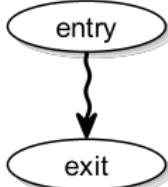
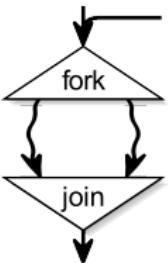
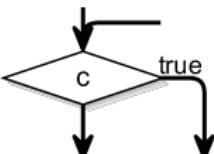
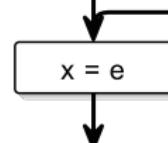
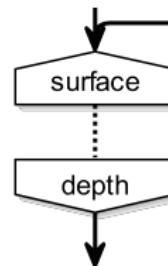
Add Sequential Constructiveness



von Hanxleden, Mendler, et al.

Sequentially Constructive Concurrency—A Conservative Extension of the Synchronous Model of Computation

ACM TECS '14

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

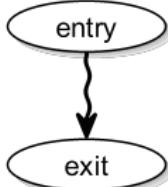
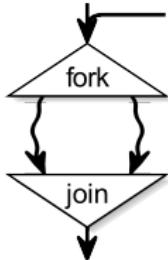
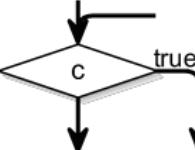
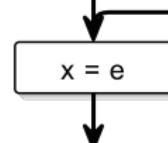
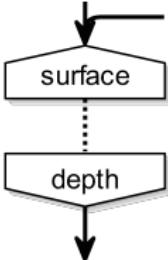
Add Sequential Constructiveness



von Hanxleden, Mendler, et al.

Sequentially Constructive Concurrency—A Conservative Extension of the Synchronous Model of Computation

ACM TECS '14

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

- ▶ Schedule *sequential* statements according to ;
- ▶ Schedule *concurrent* statements as

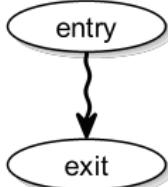
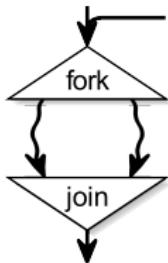
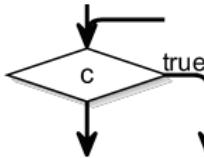
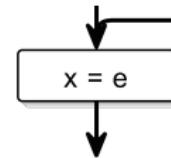
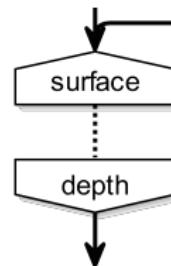
Add Sequential Constructiveness



von Hanxleden, Mendler, et al.

Sequentially Constructive Concurrency—A Conservative Extension of the Synchronous Model of Computation

ACM TECS '14

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

- ▶ Schedule *sequential* statements according to ;
- ▶ Schedule *concurrent* statements as
 1. Initialize (" $x = 0$ ")

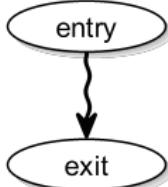
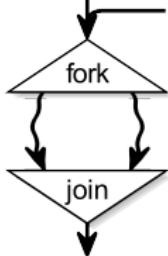
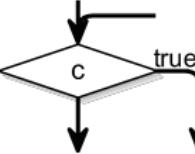
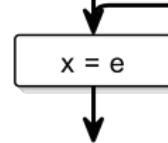
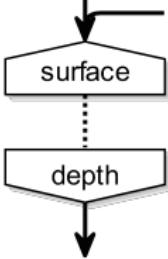
Add Sequential Constructiveness



von Hanxleden, Mendler, et al.

Sequentially Constructive Concurrency—A Conservative Extension of the Synchronous Model of Computation

ACM TECS '14

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

- ▶ Schedule *sequential* statements according to ;
- ▶ Schedule *concurrent* statements as
 1. Initialize (" $x = 0$ ")
 2. Update (" $x++$ ")

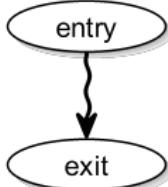
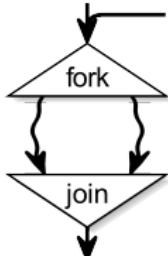
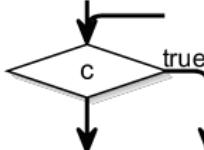
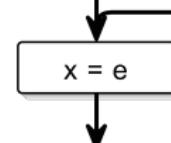
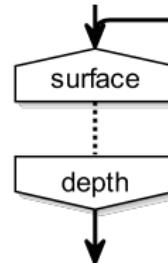
Add Sequential Constructiveness



von Hanxleden, Mendler, et al.

Sequentially Constructive Concurrency—A Conservative Extension of the Synchronous Model of Computation

ACM TECS '14

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

- ▶ Schedule *sequential* statements according to ;
- ▶ Schedule *concurrent* statements as
 1. Initialize (" $x = 0$ ")
 2. Update (" $x++$ ")
 3. Read (" $y = x$ ")

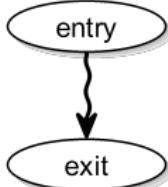
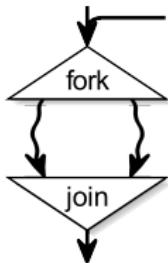
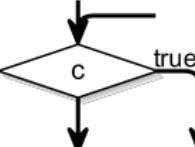
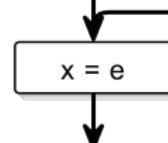
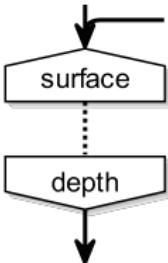
Add Sequential Constructiveness



von Hanxleden, Mendler, et al.

Sequentially Constructive Concurrency—A Conservative Extension of the Synchronous Model of Computation

ACM TECS '14

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

- ▶ Schedule *sequential* statements according to ;
- ▶ Schedule *concurrent* statements as
 1. Initialize (" $x = 0$ ")
 2. Update (" $x++$ ")
 3. Read (" $y = x$ ")
- ▶ Only reject programs that have *concurrent* causality problems

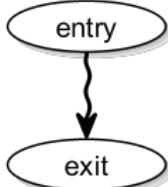
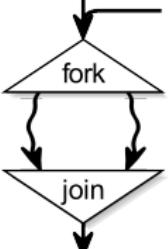
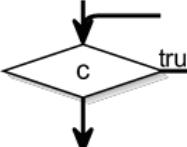
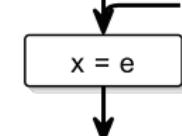
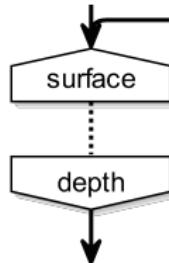
Add Sequential Constructiveness



von Hanxleden, Mendler, et al.

Sequentially Constructive Concurrency—A Conservative Extension of the Synchronous Model of Computation

ACM TECS '14

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

- ▶ Schedule *sequential* statements according to ;
- ▶ Schedule *concurrent* statements as
 1. Initialize (" $x = 0$ ")
 2. Update (" $x++$ ")
 3. Read (" $y = x$ ")
- ▶ Only reject programs that have *concurrent* causality problems, i.e., multiple concurrent initializations

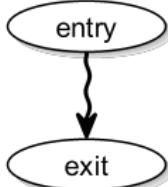
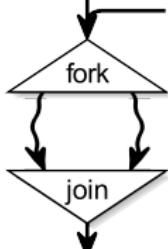
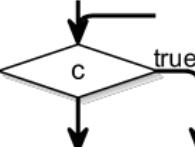
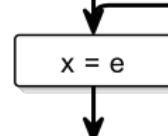
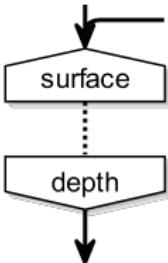
Add Sequential Constructiveness



von Hanxleden, Mendler, et al.

Sequentially Constructive Concurrency—A Conservative Extension of the Synchronous Model of Computation

ACM TECS '14

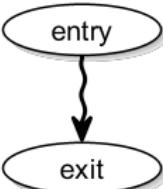
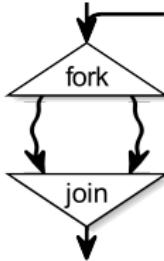
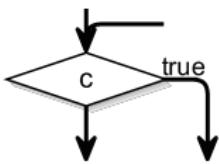
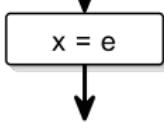
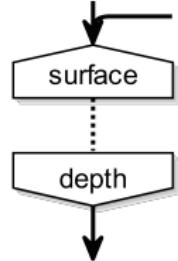
	Thread	Concurrency	Conditional	Assignment	Delay
SCL	t	fork t_1 par t_2 join	if (c) s_1 else s_2	$x = e$	pause
SCG					

- ▶ Schedule *sequential* statements according to ;
- ▶ Schedule *concurrent* statements as
 1. Initialize (" $x = 0$ ")
 2. Update (" $x++$ ")
 3. Read (" $y = x$ ")
- ▶ Only reject programs that have *concurrent* causality problems, i.e., multiple concurrent initializations

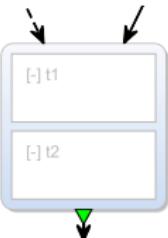
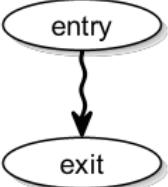
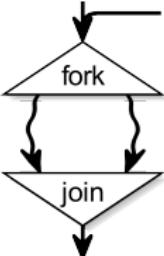
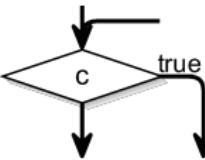
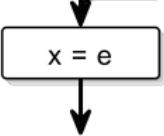
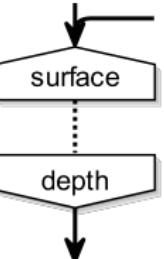
How does this fit to Statecharts? How do we compile this?
⇒ This work

SCG/SCL + Statechart Syntax \implies Normalized SCCharts

SCCharts

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

SCG/SCL + Statechart Syntax \implies Normalized SCCharts

	Region	Superstate			
SCCharts					
SCL	Thread	Concurrency	Conditional	Assignment	Delay
SCG					

SCG/SCL + Statechart Syntax \implies Normalized SCCharts

	Region	Superstate	Trigger	Effect	
SCCharts					
SCL	Thread	Concurrency	Conditional	Assignment	Delay
SCG					

SCG/SCL + Statechart Syntax \implies Normalized SCCharts

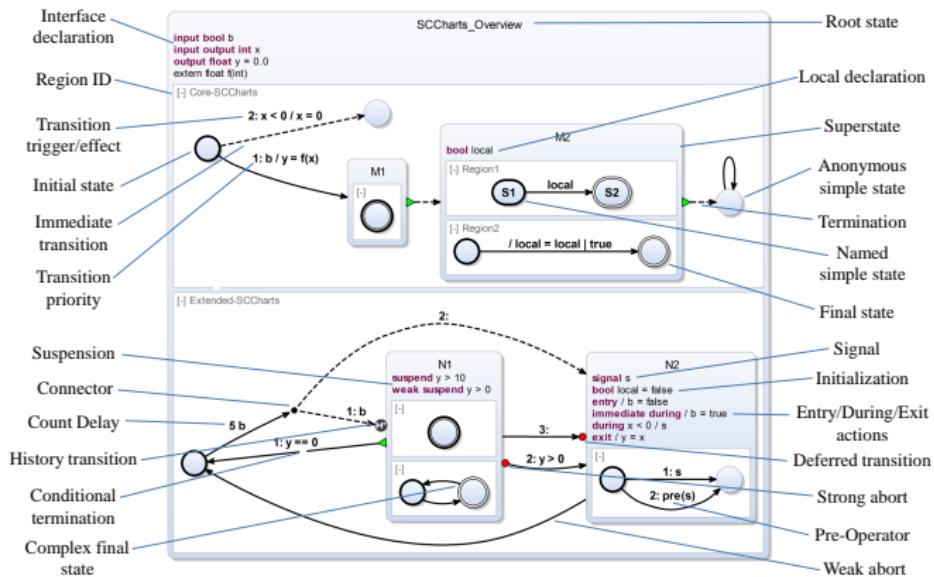
	Region	Superstate	Trigger	Effect	State
SCCharts					
SCL	Thread	Concurrency	Conditional	Assignment	Delay
SCG					

SCG/SCL + Statechart Syntax \implies Normalized SCCharts

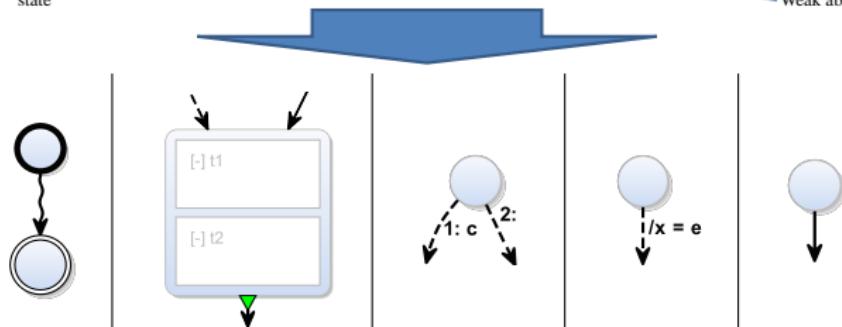
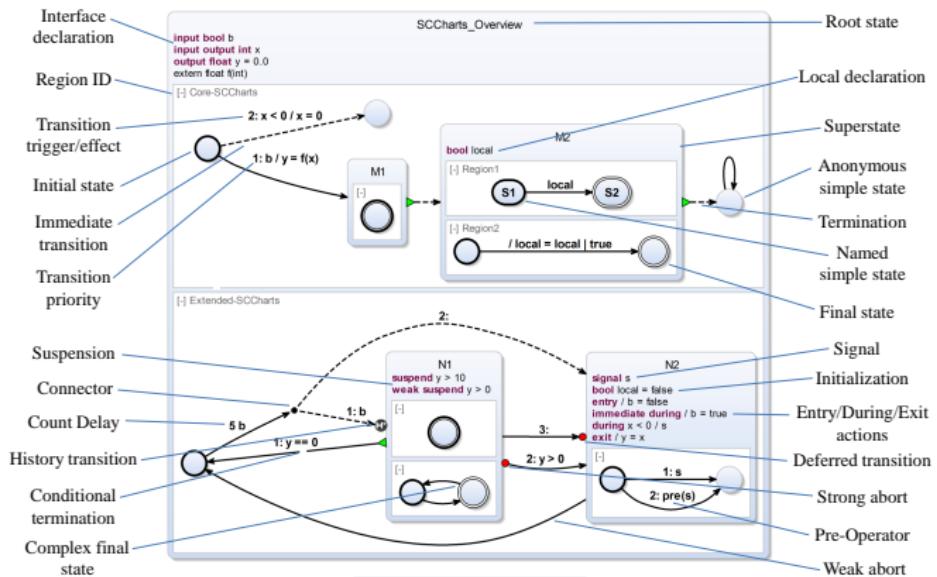
	Region	Superstate	Trigger	Effect	State
SCCharts					
SCL	t	fork t_1 par t_2 join	if (c) s_1 else s_2	$x = e$	pause
SCG					

Now add some syntactic sugar . . .

SCCharts Overview



SCCharts Overview



Example: ABRO

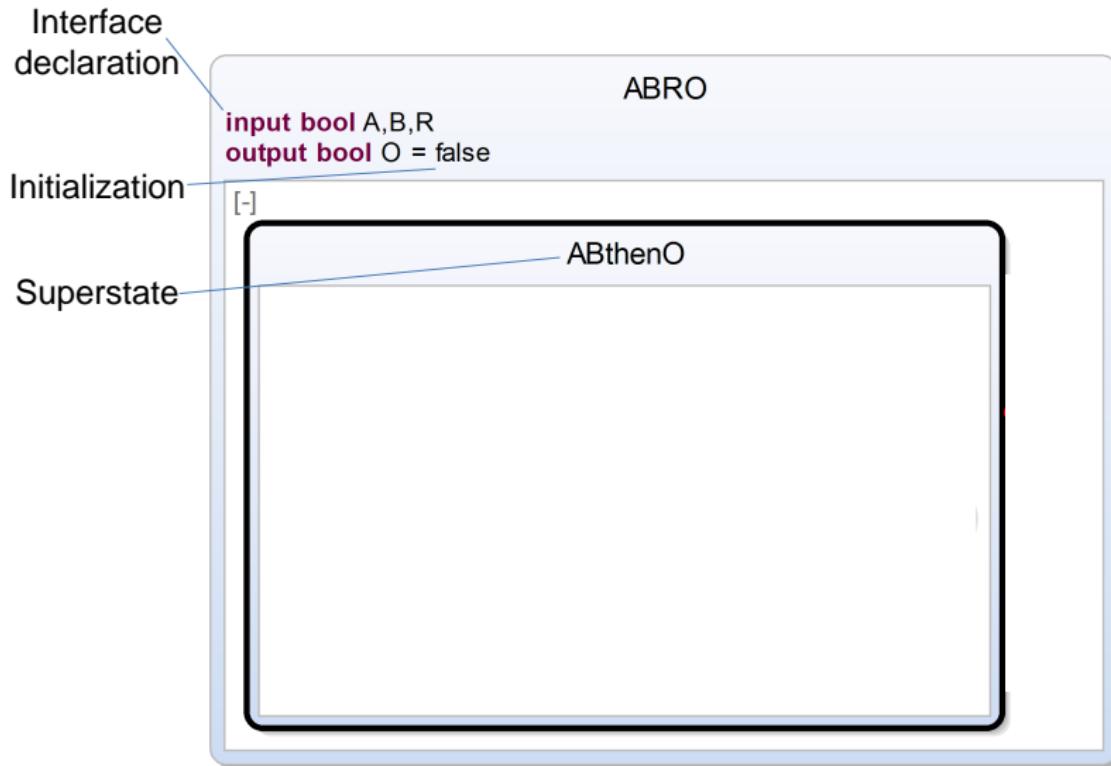
Interface
declaration

Initialization

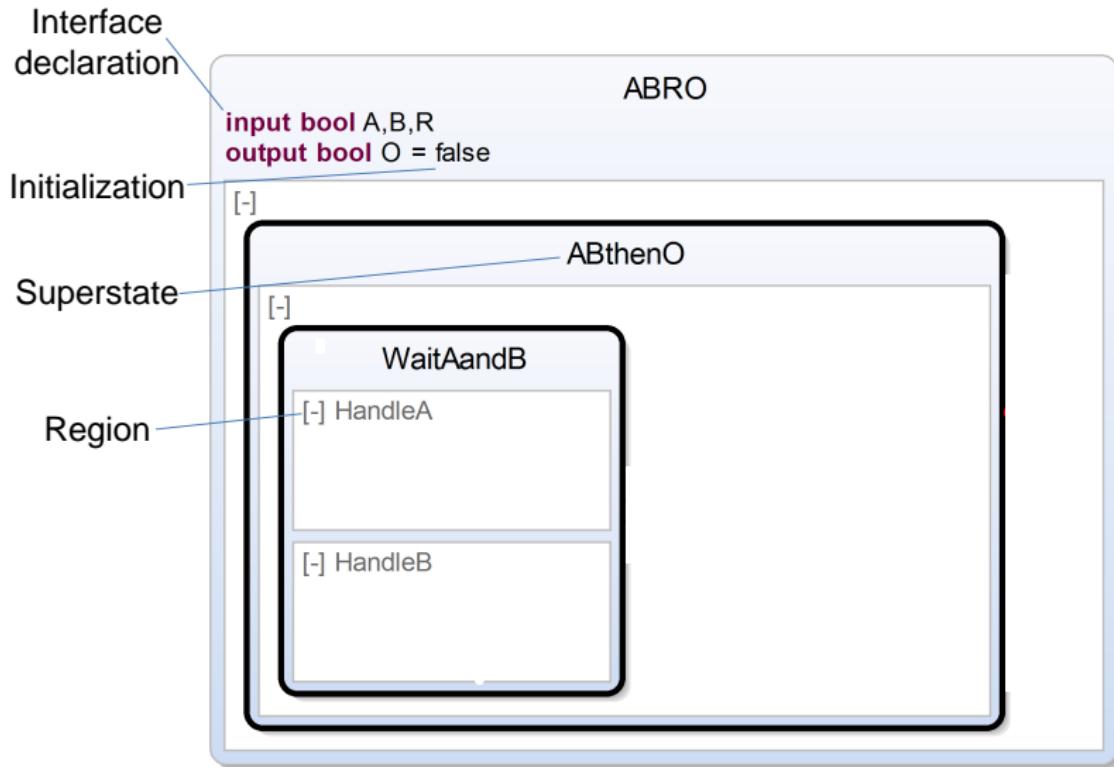
ABRO

input bool A,B,R
output bool O = false

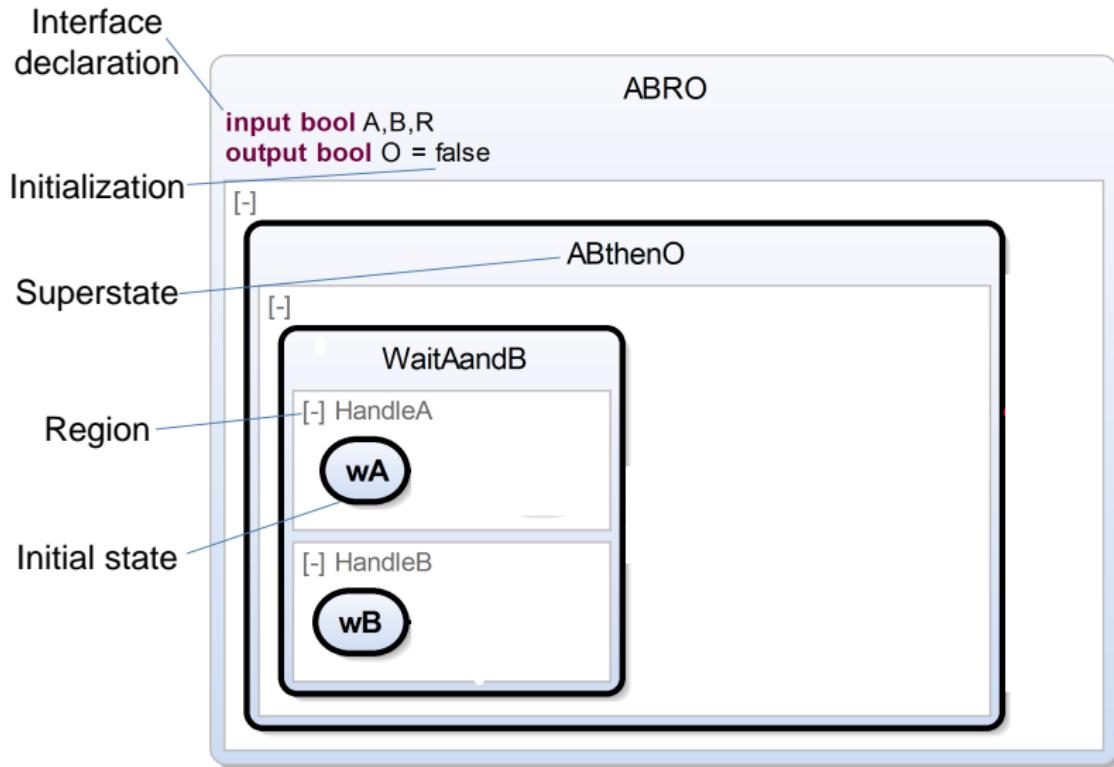
Example: ABRO



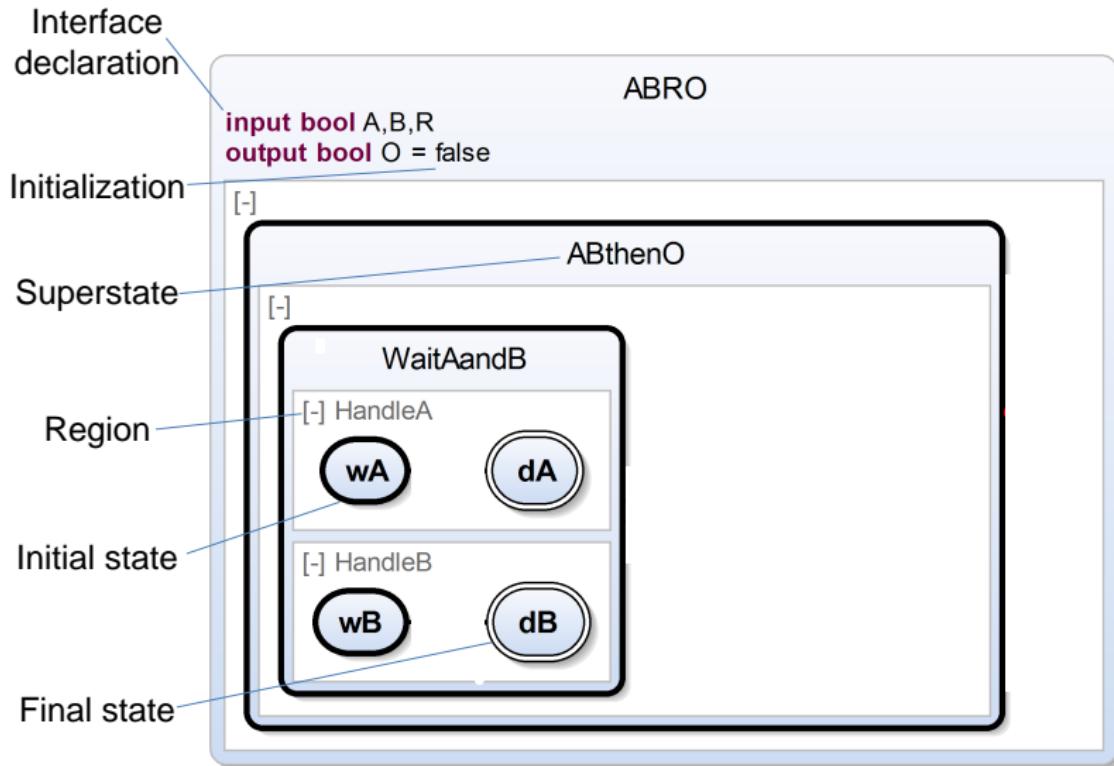
Example: ABRO



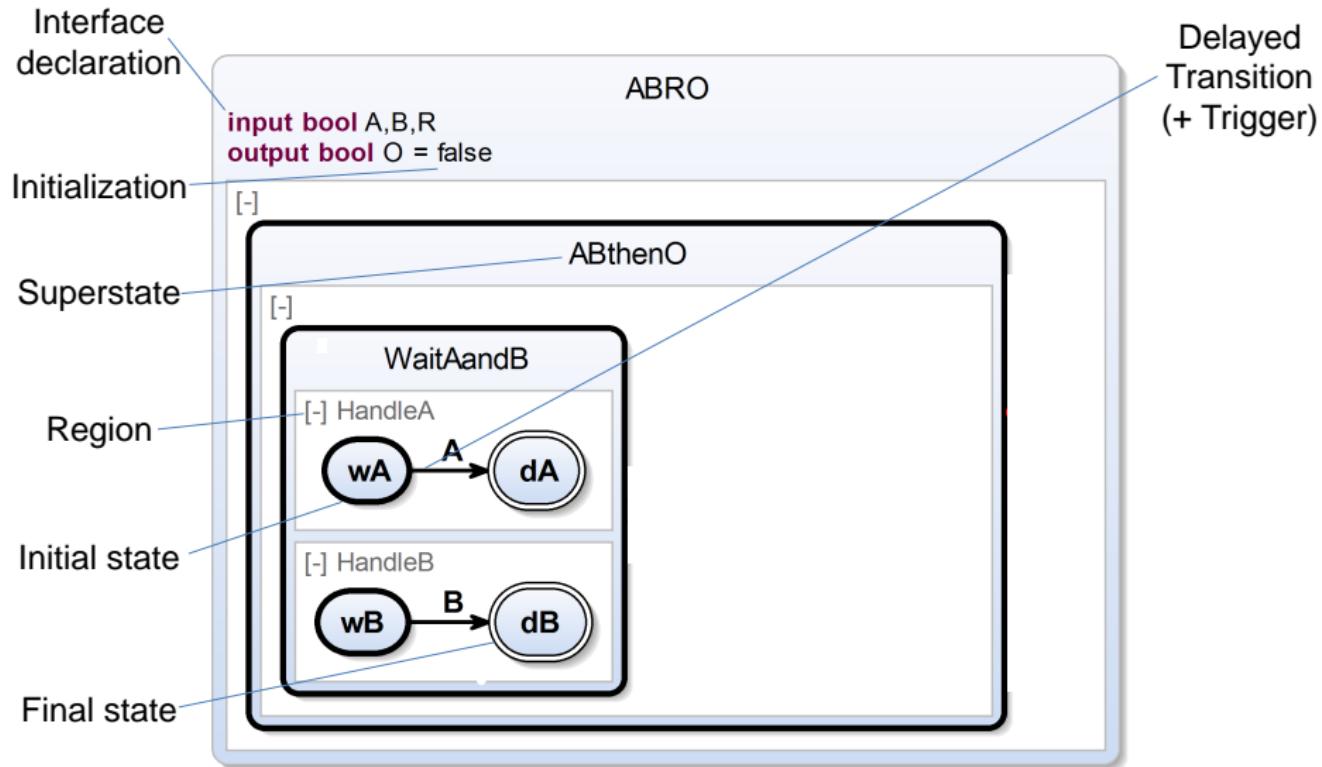
Example: ABRO



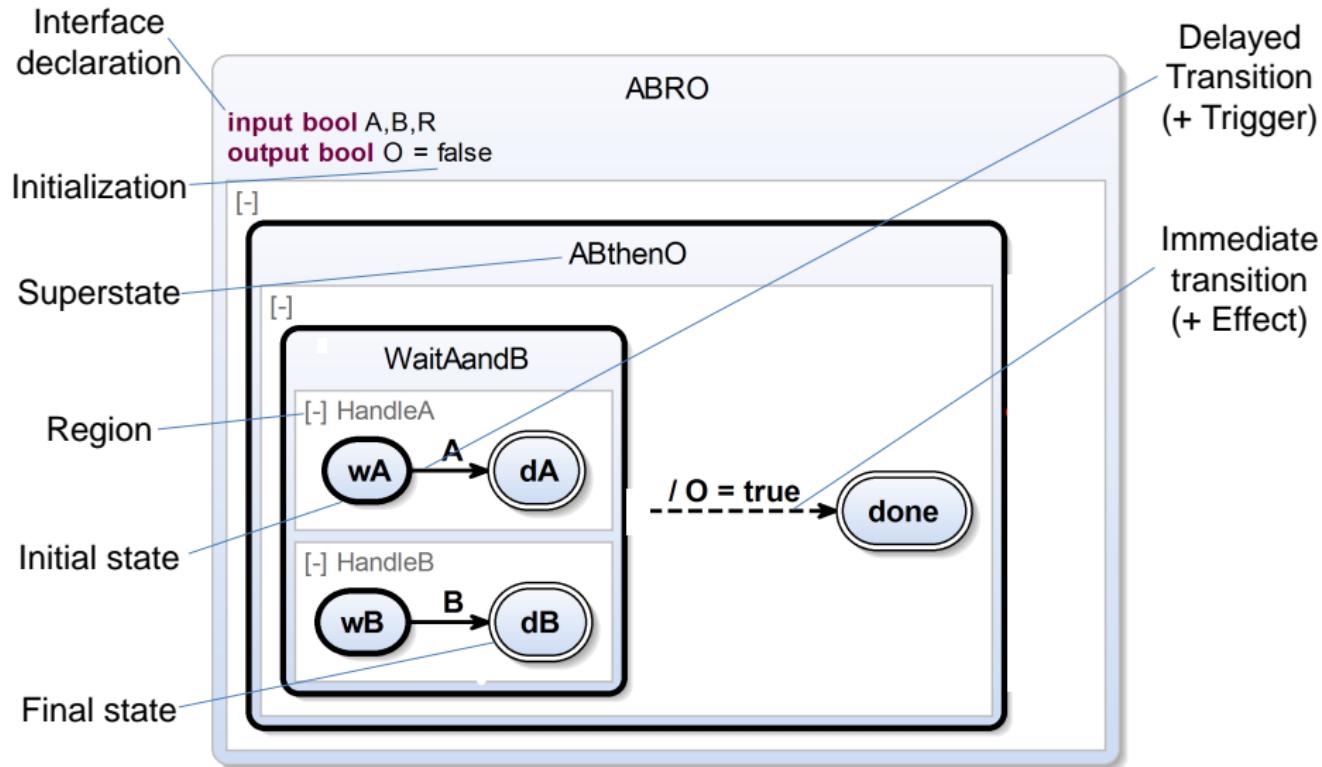
Example: ABRO



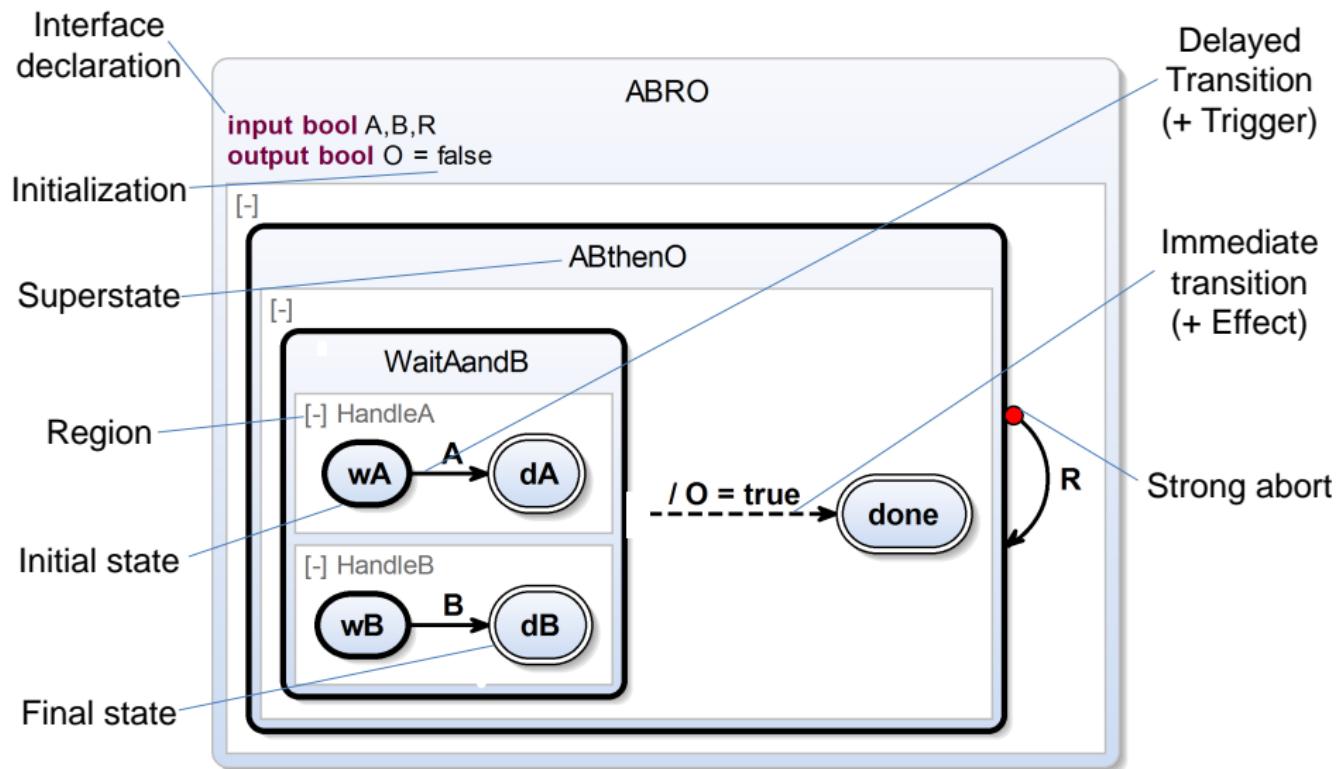
Example: ABRO



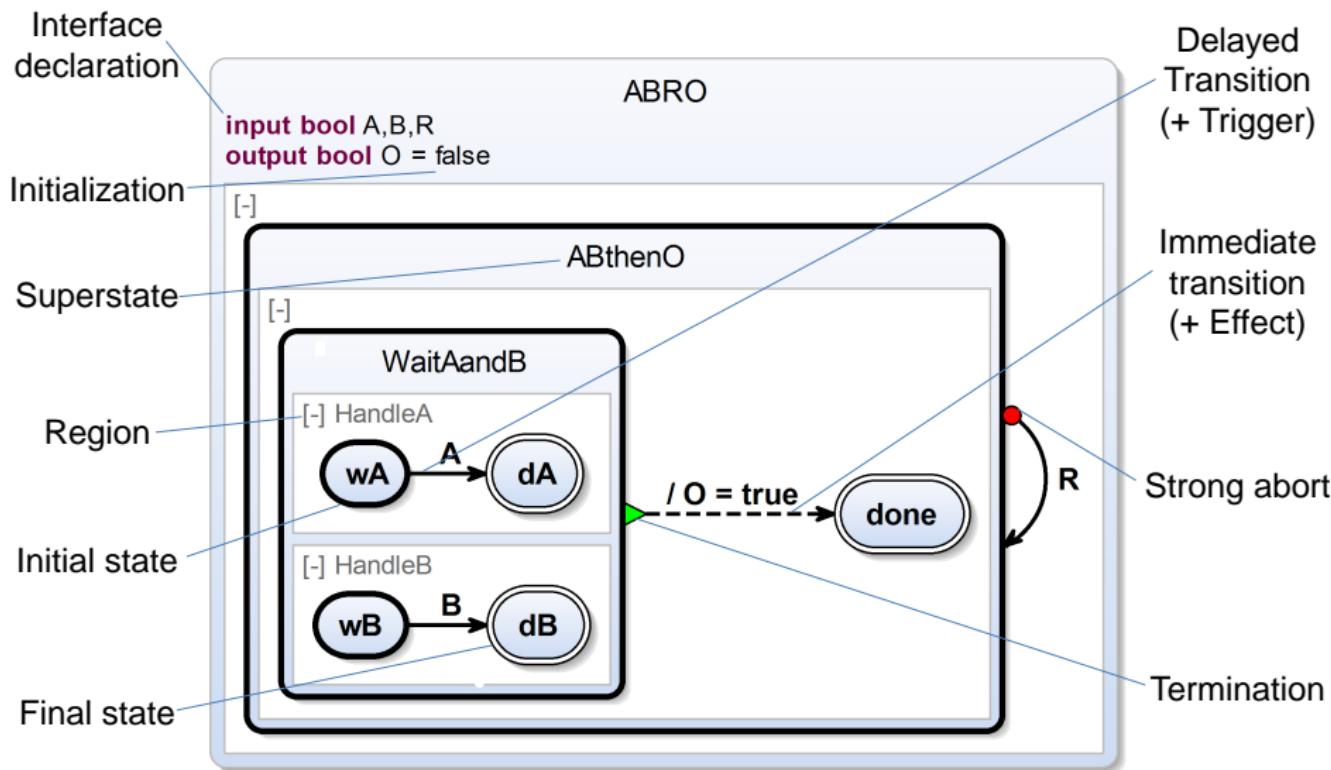
Example: ABRO



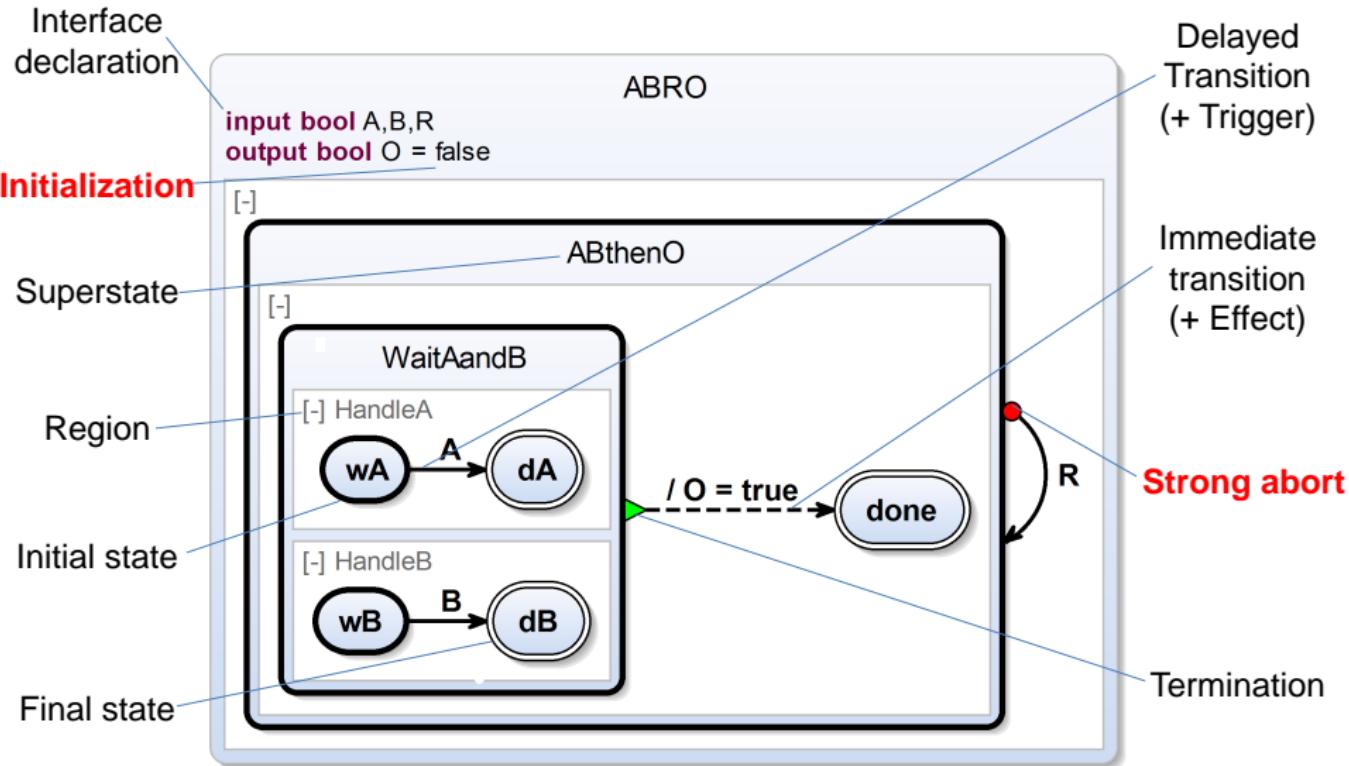
Example: ABRO



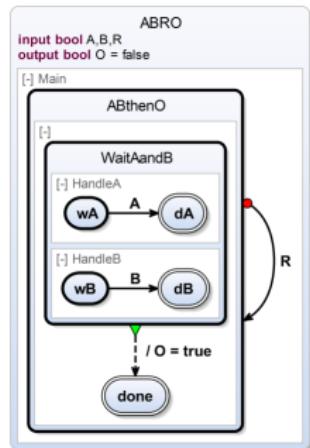
Example: ABRO



Example: ABRO—With Extended SCChart Features

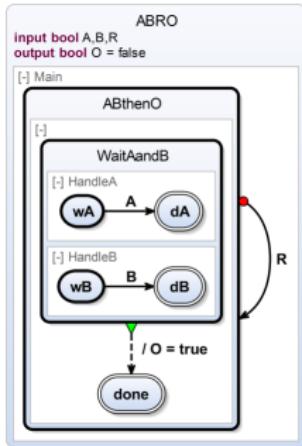


What do we want to do?



Model

What do we want to do?



CODE

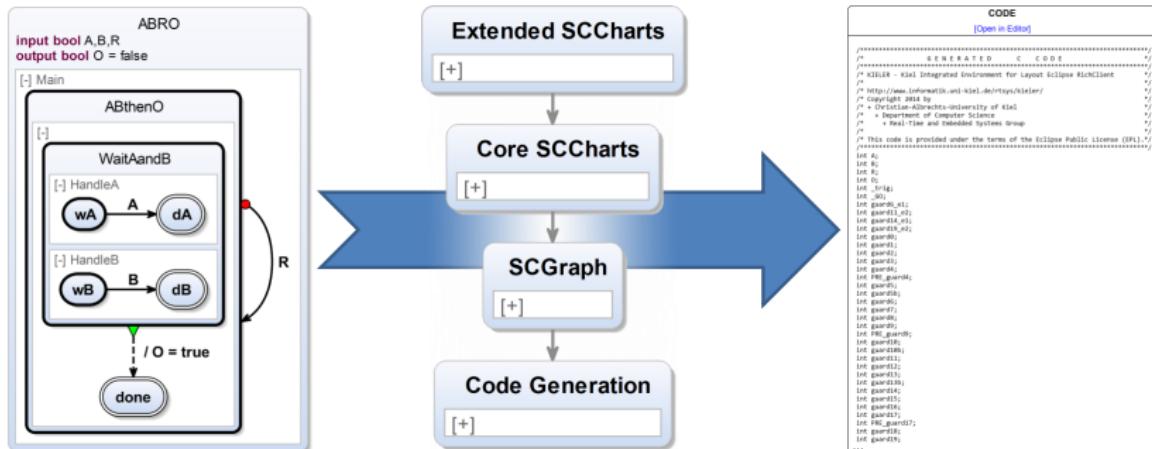
(Open in Editor)

```
***** GENERATED C CODE ****
/*
 * KIELER - kiel Integrated Environment for Layout Eclipse #KIELClient
 */
/* Copyright 2014 by
 * Institute for Microeconomics, University of Kiel
 * Department of Computer Science
 * + Real-Time and Embedded Systems Group
 */
/* This code is provided under the terms of the Eclipse Public License (EPL). */
***** GENERATED C CODE ****
int A;
int B;
int R;
int O;
int O1;
int O2;
int O3;
int O4;
int O5;
int O6;
int O7;
int O8;
int O9;
int O10;
int O11;
int O12;
int O13;
int O14;
int O15;
int O16;
int O17;
int O18;
int O19;
int O20;
int O21;
int O22;
int O23;
int O24;
int O25;
int O26;
int O27;
int O28;
int O29;
int O30;
int O31;
int O32;
int O33;
int O34;
int O35;
int O36;
int O37;
int O38;
int O39;
int O40;
int O41;
int O42;
int O43;
int O44;
int O45;
int O46;
int O47;
int O48;
int O49;
int O50;
int O51;
int O52;
int O53;
int O54;
int O55;
int O56;
int O57;
int O58;
int O59;
int O60;
int O61;
int O62;
int O63;
int O64;
int O65;
int O66;
int O67;
int O68;
int O69;
int O70;
int O71;
int O72;
int O73;
int O74;
int O75;
int O76;
int O77;
int O78;
int O79;
int O80;
int O81;
int O82;
int O83;
int O84;
int O85;
int O86;
int O87;
int O88;
int O89;
int O90;
int O91;
int O92;
int O93;
int O94;
int O95;
int O96;
int O97;
int O98;
int O99;
int O100;
int O101;
int O102;
int O103;
int O104;
int O105;
int O106;
int O107;
int O108;
int O109;
int O110;
int O111;
int O112;
int O113;
int O114;
int O115;
int O116;
int O117;
int O118;
int O119;
int O120;
int O121;
int O122;
int O123;
int O124;
int O125;
int O126;
int O127;
int O128;
int O129;
int O130;
int O131;
int O132;
int O133;
int O134;
int O135;
int O136;
int O137;
int O138;
int O139;
int O140;
int O141;
int O142;
int O143;
int O144;
int O145;
int O146;
int O147;
int O148;
int O149;
int O150;
int O151;
int O152;
int O153;
int O154;
int O155;
int O156;
int O157;
int O158;
int O159;
int O160;
int O161;
int O162;
int O163;
int O164;
int O165;
int O166;
int O167;
int O168;
int O169;
int O170;
int O171;
int O172;
int O173;
int O174;
int O175;
int O176;
int O177;
int O178;
int O179;
int O180;
int O181;
int O182;
int O183;
int O184;
int O185;
int O186;
int O187;
int O188;
int O189;
int O190;
int O191;
int O192;
int O193;
int O194;
int O195;
int O196;
int O197;
int O198;
int O199;
int O200;
int O201;
int O202;
int O203;
int O204;
int O205;
int O206;
int O207;
int O208;
int O209;
int O210;
int O211;
int O212;
int O213;
int O214;
int O215;
int O216;
int O217;
int O218;
int O219;
int O220;
int O221;
int O222;
int O223;
int O224;
int O225;
int O226;
int O227;
int O228;
int O229;
int O230;
int O231;
int O232;
int O233;
int O234;
int O235;
int O236;
int O237;
int O238;
int O239;
int O240;
int O241;
int O242;
int O243;
int O244;
int O245;
int O246;
int O247;
int O248;
int O249;
int O250;
int O251;
int O252;
int O253;
int O254;
int O255;
int O256;
int O257;
int O258;
int O259;
int O260;
int O261;
int O262;
int O263;
int O264;
int O265;
int O266;
int O267;
int O268;
int O269;
int O270;
int O271;
int O272;
int O273;
int O274;
int O275;
int O276;
int O277;
int O278;
int O279;
int O280;
int O281;
int O282;
int O283;
int O284;
int O285;
int O286;
int O287;
int O288;
int O289;
int O290;
int O291;
int O292;
int O293;
int O294;
int O295;
int O296;
int O297;
int O298;
int O299;
int O300;
int O310;
int O320;
int O330;
int O340;
int O350;
int O360;
int O370;
int O380;
int O390;
int O400;
int O410;
int O420;
int O430;
int O440;
int O450;
int O460;
int O470;
int O480;
int O490;
int O500;
int O510;
int O520;
int O530;
int O540;
int O550;
int O560;
int O570;
int O580;
int O590;
int O600;
int O610;
int O620;
int O630;
int O640;
int O650;
int O660;
int O670;
int O680;
int O690;
int O700;
int O710;
int O720;
int O730;
int O740;
int O750;
int O760;
int O770;
int O780;
int O790;
int O800;
int O810;
int O820;
int O830;
int O840;
int O850;
int O860;
int O870;
int O880;
int O890;
int O900;
int O910;
int O920;
int O930;
int O940;
int O950;
int O960;
int O970;
int O980;
int O990;
int O1000;
int O1010;
int O1020;
int O1030;
int O1040;
int O1050;
int O1060;
int O1070;
int O1080;
int O1090;
int O1100;
int O1110;
int O1120;
int O1130;
int O1140;
int O1150;
int O1160;
int O1170;
int O1180;
int O1190;
int O1200;
int O1210;
int O1220;
int O1230;
int O1240;
int O1250;
int O1260;
int O1270;
int O1280;
int O1290;
int O1300;
int O1310;
int O1320;
int O1330;
int O1340;
int O1350;
int O1360;
int O1370;
int O1380;
int O1390;
int O1400;
int O1410;
int O1420;
int O1430;
int O1440;
int O1450;
int O1460;
int O1470;
int O1480;
int O1490;
int O1500;
int O1510;
int O1520;
int O1530;
int O1540;
int O1550;
int O1560;
int O1570;
int O1580;
int O1590;
int O1600;
int O1610;
int O1620;
int O1630;
int O1640;
int O1650;
int O1660;
int O1670;
int O1680;
int O1690;
int O1700;
int O1710;
int O1720;
int O1730;
int O1740;
int O1750;
int O1760;
int O1770;
int O1780;
int O1790;
int O1800;
int O1810;
int O1820;
int O1830;
int O1840;
int O1850;
int O1860;
int O1870;
int O1880;
int O1890;
int O1900;
int O1910;
int O1920;
int O1930;
int O1940;
int O1950;
int O1960;
int O1970;
int O1980;
int O1990;
int O2000;
int O2010;
int O2020;
int O2030;
int O2040;
int O2050;
int O2060;
int O2070;
int O2080;
int O2090;
int O2100;
int O2110;
int O2120;
int O2130;
int O2140;
int O2150;
int O2160;
int O2170;
int O2180;
int O2190;
int O2200;
int O2210;
int O2220;
int O2230;
int O2240;
int O2250;
int O2260;
int O2270;
int O2280;
int O2290;
int O2300;
int O2310;
int O2320;
int O2330;
int O2340;
int O2350;
int O2360;
int O2370;
int O2380;
int O2390;
int O2400;
int O2410;
int O2420;
int O2430;
int O2440;
int O2450;
int O2460;
int O2470;
int O2480;
int O2490;
int O2500;
int O2510;
int O2520;
int O2530;
int O2540;
int O2550;
int O2560;
int O2570;
int O2580;
int O2590;
int O2600;
int O2610;
int O2620;
int O2630;
int O2640;
int O2650;
int O2660;
int O2670;
int O2680;
int O2690;
int O2700;
int O2710;
int O2720;
int O2730;
int O2740;
int O2750;
int O2760;
int O2770;
int O2780;
int O2790;
int O2800;
int O2810;
int O2820;
int O2830;
int O2840;
int O2850;
int O2860;
int O2870;
int O2880;
int O2890;
int O2900;
int O2910;
int O2920;
int O2930;
int O2940;
int O2950;
int O2960;
int O2970;
int O2980;
int O2990;
int O3000;
```

Model

Code

What do we want to do?

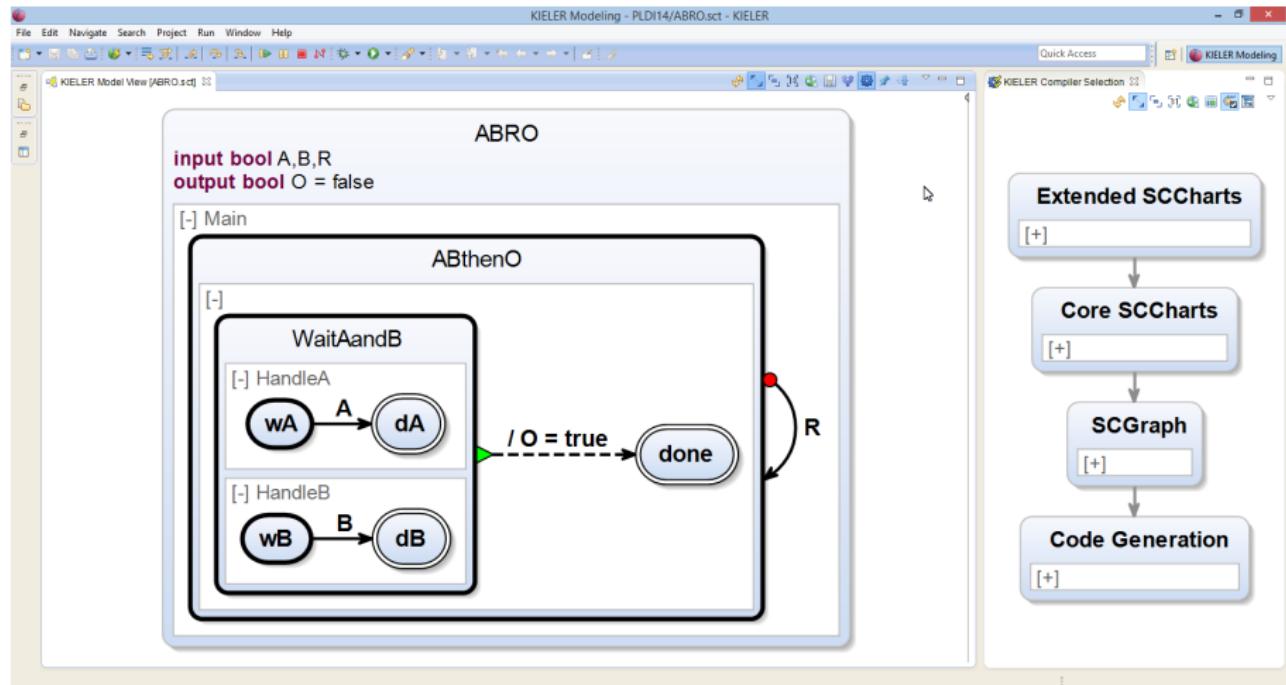


Model

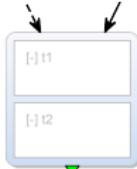
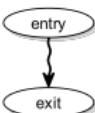
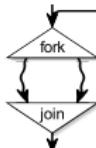
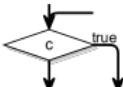
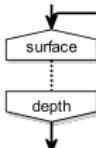
Transformation
Stages

Code

SCCharts Compiler in KIELER Eclipse Richt Client



One Approach to Code Generation: Data-Flow

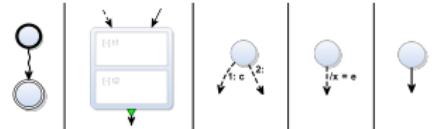
	Region (Thread)	Superstate (Concurrency)	Trigger (Conditional)	Effect (Assignment)	State (Delay)
SCCharts					
SCG					
SCL	t	$\text{fork } t_1 \text{ par } t_2 \text{ join}$	$\text{if } (c) s_1 \text{ else } s_2$	$x = e$	pause

One Approach to Code Generation: Data-Flow

	Region (Thread)	Superstate (Concurrency)	Trigger (Conditional)	Effect (Assignment)	State (Delay)
SCCharts					
SCG					
SCL	t	$\text{fork } t_1 \text{ par } t_2 \text{ join}$	$\text{if } (c) s_1 \text{ else } s_2$	$x = e$	pause
Data- Flow Code	$d = g_{exit}$ $m = \neg \bigvee_{surf \in t} g_{surf}$	$g_{join} = (d_1 \vee m_1) \wedge (d_2 \vee m_2) \wedge (d_1 \vee d_2)$	$g = \bigvee g_{in}$ $g_{true} = g \wedge c$ $g_{false} = g \wedge \neg c$	$g = \bigvee g_{in}$ $x' = g ? e : x$	$g_{depth} = \text{pre}(g_{surf})$
Circuits					

Summary

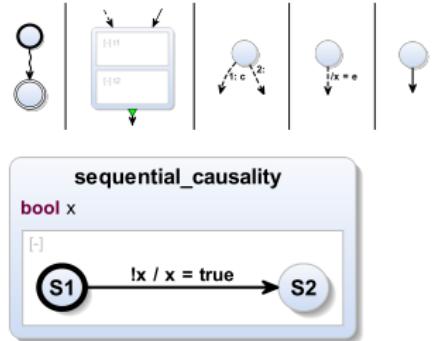
Starting point: Concurrent Statechart
5 Core Constructs +
Smörgåsbord of Extensions



Summary

Starting point: Concurrent Statechart
5 Core Constructs +
Smörgåsbord of Extensions

Semantics: Deterministic
Sequentially constructive
; not source of errors,
instead resolves errors

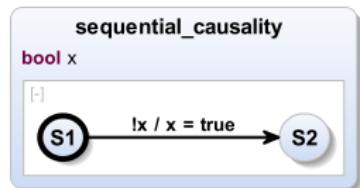
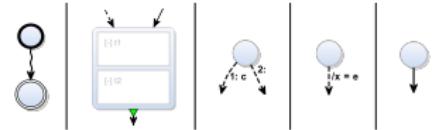


Summary

Starting point: Concurrent Statechart
5 Core Constructs +
Smörgåsbord of Extensions

Semantics: Deterministic
Sequentially constructive
; not source of errors,
instead resolves errors

Compiler: Incremental transformations
No conceptual breaks
Currently stress-tested
in class room



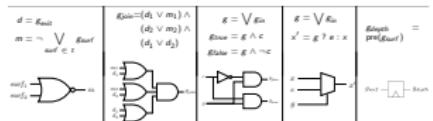
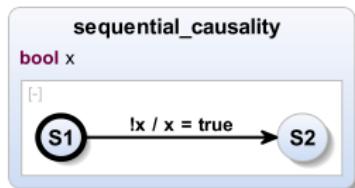
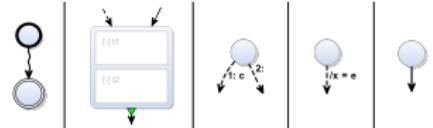
Summary

Starting point: Concurrent Statechart
5 Core Constructs +
Smörgåsbord of Extensions

Semantics: Deterministic
Sequentially constructive
; not source of errors,
instead resolves errors

Compiler: Incremental transformations
No conceptual breaks
Currently stress-tested
in class room

Result: SW or HW
Sequential or parallel



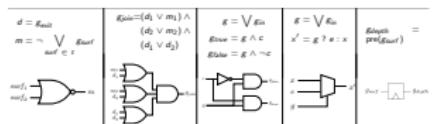
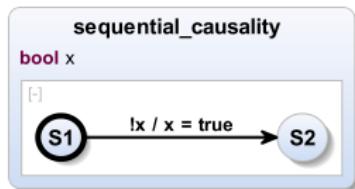
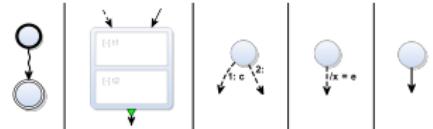
Summary

Starting point: Concurrent Statechart
5 Core Constructs +
Smörgåsbord of Extensions

Semantics: Deterministic
Sequentially constructive
; not source of errors,
instead resolves errors

Compiler: Incremental transformations
No conceptual breaks
Currently stress-tested
in class room

Result: SW or HW
Sequential or parallel



What compiler/optimization machinery can be put to work here?

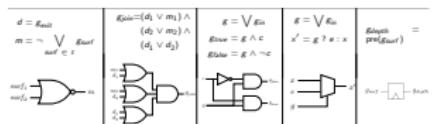
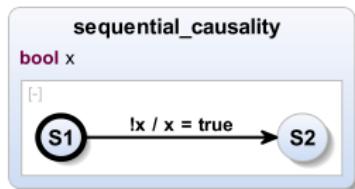
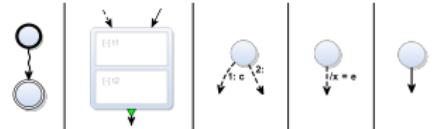
Summary

Starting point: Concurrent Statechart
5 Core Constructs +
Smörgåsbord of Extensions

Semantics: Deterministic
Sequentially constructive
; not source of errors,
instead resolves errors

Compiler: Incremental transformations
No conceptual breaks
Currently stress-tested
in class room

Result: SW or HW
Sequential or parallel



What compiler/optimization machinery can be put to work here?
Thanks!

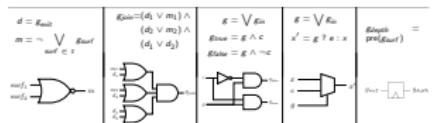
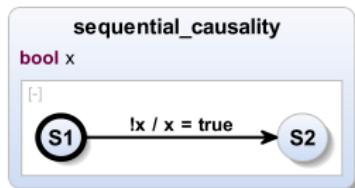
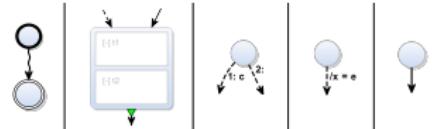
Summary

Starting point: Concurrent Statechart
5 Core Constructs +
Smörgåsbord of Extensions

Semantics: Deterministic
Sequentially constructive
; not source of errors,
instead resolves errors

Compiler: Incremental transformations
No conceptual breaks
Currently stress-tested
in class room

Result: SW or HW
Sequential or parallel

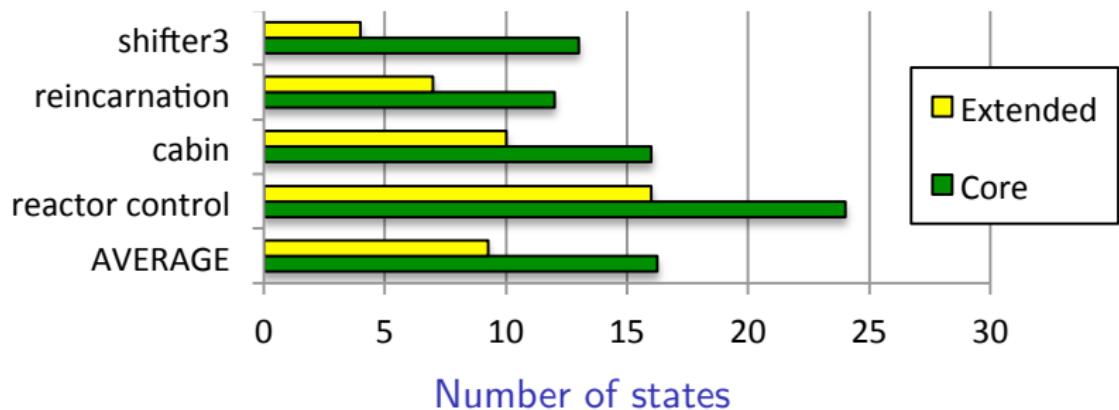


What compiler/optimization machinery can be put to work here?

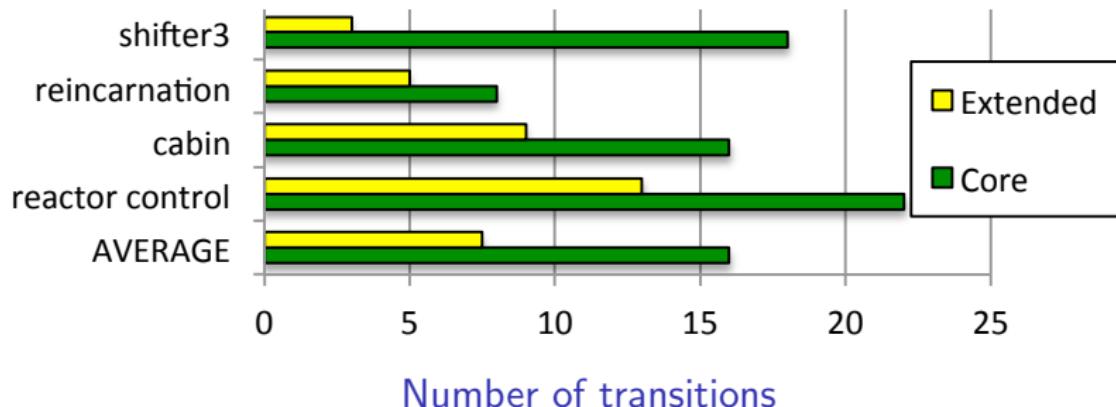
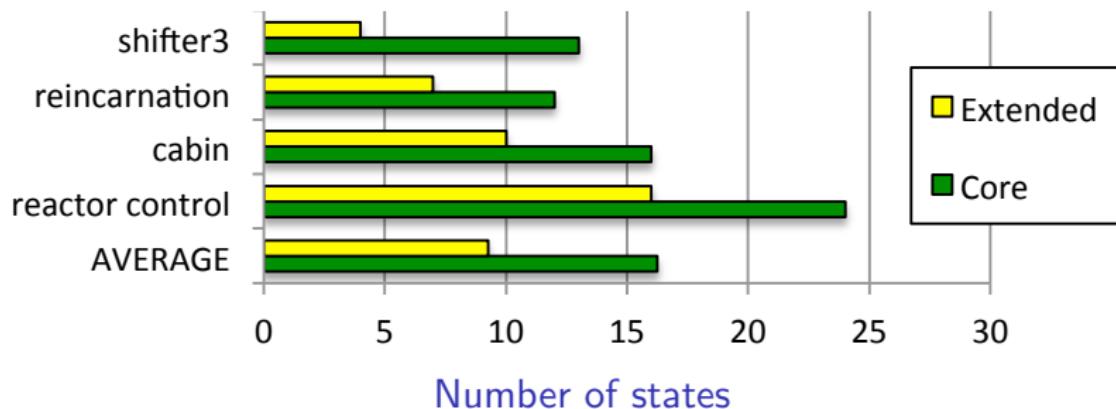
Thanks!

Questions or comments?

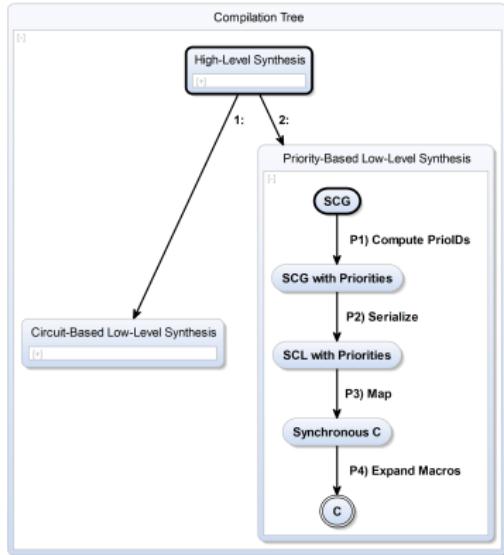
Effects of Expanding Expansions



Effects of Expanding Expansions

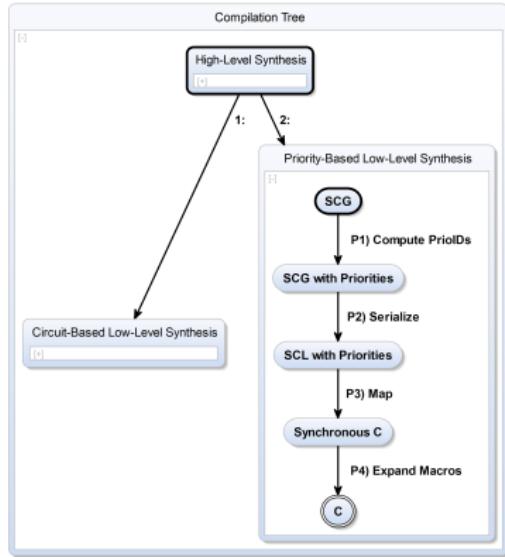


Low-Level Synthesis II: The Priority Approach



- ▶ More software-like
- ▶ Don't emulate control flow with guards/basic blocks, but with program counters/threads
- ▶ Priority-based thread dispatching
- ▶ SCL_P : $SCL + \text{PrioIDs}$
- ▶ Implemented as C macros

Low-Level Synthesis II: The Priority Approach

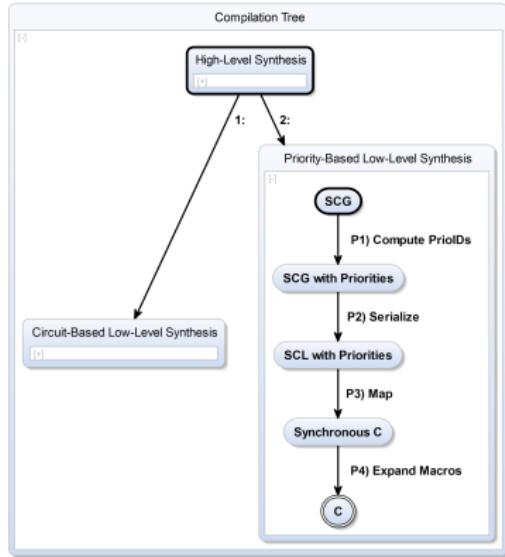


- ▶ More software-like
- ▶ Don't emulate control flow with guards/basic blocks, but with program counters/threads
- ▶ Priority-based thread dispatching
- ▶ SCL_P: SCL + PrioIDs
- ▶ Implemented as C macros

Differences to Synchronous C [von Hanxleden '09]

- ▶ No preemption ⇒ don't need to keep track of thread hierarchies
- ▶ Fewer, more light-weight operators
- ▶ RISC instead of CISC

Low-Level Synthesis II: The Priority Approach



- ▶ 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
- ▶ Implemented as C macros

Differences to Synchronous C [von Hanxleden '09]

- ▶ No preemption ⇒ don't need to keep track of thread hierarchies
- ▶ Fewer, more light-weight operators
- ▶ RISC instead of CISC
- ▶ More human-friendly syntax

SCL_P Macros I

```
1 // Boolean type
2 typedef int bool;
3 #define false 0
4 #define true 1
5
6 // Enable/disable threads with prioID p
7 #define _u2b(u)          (1 << u)
8 #define _enable(p)       _enabled |= _u2b(p); \
9                          active |= _u2b(p)
10 #define _isEnabled(p)   (( _enabled & _u2b(p)) != 0)
11 #define _disable(p)     _enabled &= ~_u2b(p)
12
13 // Set current thread continuation
14 #define _setPC(p, label) _pc[p] = &&label
```

SCL_P Macros II

```
17 #define _pause(label)      _setPC(_cid, label); \
18                           goto _L_PAUSE
19
20 // Pause, resume at pause
21 #define _concat_helper(a, b) a ## b
22 #define _concat(a, b)        _concat_helper(a, b)
23 #define _label_              _concat(_L, __LINE__)
24 #define pause               _pause(_label_); _label_ :
25
26 // Fork/join sibling thread with prioID p
27 #define fork1(label, p)     _setPC(p, label); _enable(p);
28 #define join1(p)           _label_ : if (_isEnabled(p)) \
29                           { _pause(_label_); }
30
31 // Terminate thread at "par"
32 #define par                goto _L_TERM;
```

ABO SCL_P I

```
85 int tick()
86 {
87     tickstart(2);
88     O1 = false;
89     O2 = false;
90
91     fork1(HandleB, 1)
92         {
93             HandleA:
94             if (!A) {
95                 pause;
96                 goto HandleA;
97                 ;
98             }
99             B = true;
100            O1 = true;
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); {
92         HandleA:
93         if (!A) {
94             _pc[_cid] = &&_L94; goto _L_PAUSE;
95             _L94:;
96             goto HandleA;
97         }
98         B = 1;
99         O1 = 1;
100     } goto _L_TERM; {
```

ABO SCL_P II

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



```
102     HandleB:  
103         _pc[_cid] = &&_L103; goto _L_PAUSE;  
104         _L103;;  
105         if (!B) {  
106             goto HandleB;  
107         }  
108         O1 = 1;  
109         } _L108: if (((enabled & (1 << 2)) != 0)) {  
110             _pc[_cid] = &&_L108; goto _L_PAUSE;  
111         };  
112         O1 = 0;  
113         O2 = 1;  
114         goto _L_TERM; _L_TICKEND: return (  
115             enabled != (1 << 0)); _L_TERM:  
116             enabled &= ~(1 << _cid); _L_PAUSE:  
117             active &= ~(1 << _cid); _L_DISPATCH:  
118             __asm volatile("bsrl %1,%0\n" : "=r" (  
119                 _cid) : "r" (active)); goto *_pc[_cid];  
120 }
```

Comparison of Low-Level Synthesis Approaches

Circuit	Priority
---------	----------

Comparison of Low-Level Synthesis Approaches

	Circuit	Priority
Accepts instantaneous loops	-	+
Can synthesize hardware	+	-
Can synthesize software	+	+

Comparison of Low-Level Synthesis Approaches

	Circuit	Priority
Accepts instantaneous loops	-	+
Can synthesize hardware	+	-
Can synthesize software	+	+
Size scales well (linear in size of SCChart)	+	+

Comparison of Low-Level Synthesis Approaches

	Circuit	Priority
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)	+	-