SCCharts: Sequentially Constructive Statecharts for Safety-Critical Applications

Reinhard von Hanxleden[1], Björn Duderstadt[1], Christian Motika[1], **Steven Smyth**[1], Michael Mendler[2], Joaquin Aguado[2], Stephen Mercer[3], and Owen O'Brien[3]

[1]Christian-Albrechts-Universität zu Kiel
[2]Bamberg University
[3]National Instruments

June 9–11, 2014, Edingburgh, 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)

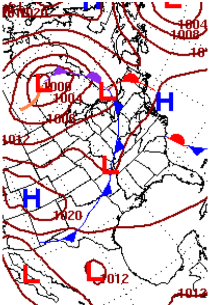
A Scheduling Problem: Parallelization

- 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)

HPC


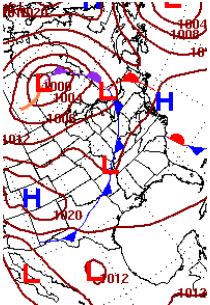
A Scheduling Problem: Parallelization

- 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)

HPC


Starting point: Sequential program

Semantics: Given by ";"

A Scheduling Problem: Parallelization

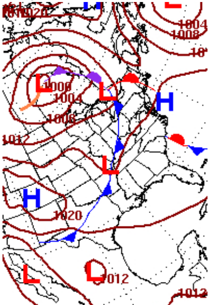
von Hanxleden, Kennedy
Relaxing SIMD Control Flow Constraints Using Loop Transformations
PLDI'92

von Hanxleden, Kennedy
Give-N-Take — A Balanced Code Placement Framework
PLDI'94

wikipedia

Ken Kennedy
(1945 – 2007)

HPC



Starting point:	Sequential program
Semantics:	Given by ";"
Compiler:	Eliminates (some) ;'s
Result:	Concurrent program

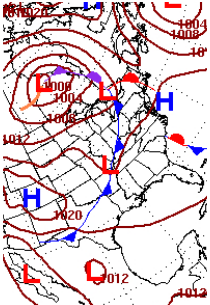
A Scheduling Problem: Parallelization

- 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)

HPC



Starting point:	Sequential program
Semantics:	Given by ";"
Compiler:	Eliminates (some) ;'s
Result:	Concurrent program

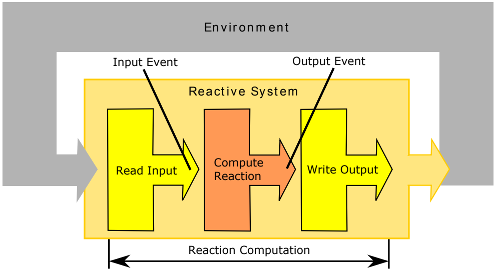
How much speedup do we get?

Another Scheduling Problem: Statecharts

Harel
Statecharts: A Visual Formalism for Complex Systems
Science of Computer Programming '87

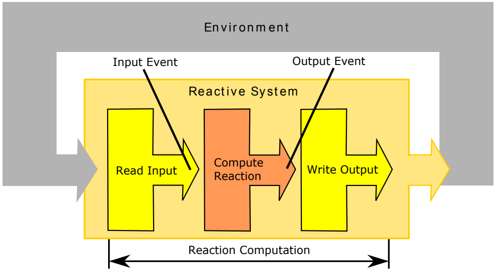
Another Scheduling Problem: Statecharts

Harel
Statecharts: A Visual Formalism for Complex Systems
Science of Computer Programming '87



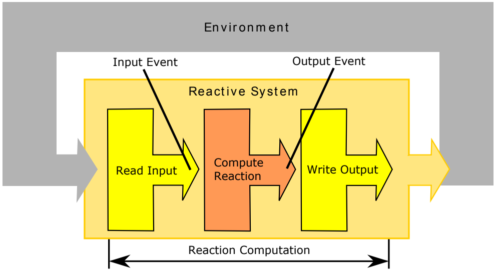
Another Scheduling Problem: Statecharts

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

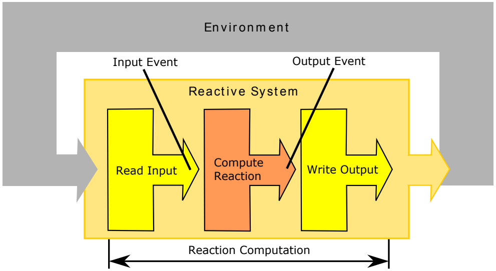


Starting point: Concurrent Statechart

Semantics: Synchronous (?)

Another Scheduling Problem: Statecharts

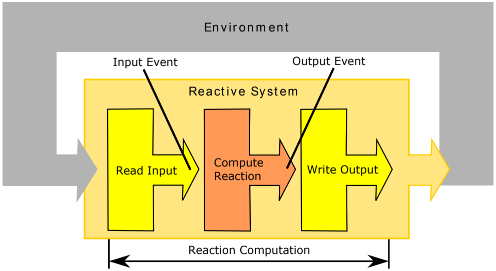
Harel
Statecharts: A Visual Formalism for Complex Systems
Science of Computer Programming '87



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

André
Computing SyncCharts Reactions
ENTCS '94

Starting point:	Concurrent Statechart
Semantics:	Synchronous (!), 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

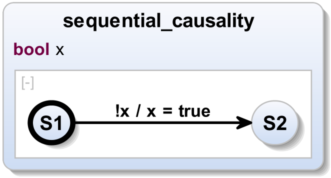
Starting point:	Concurrent Statechart
Semantics:	Synchronous (!), deterministic
Compiler:	Rejects some ;'s
Result:	Sequential program

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

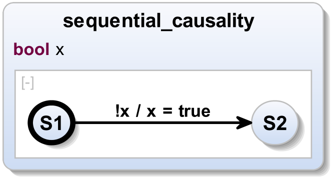


Add Synchronous Languages

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



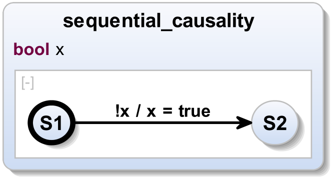
```
if (!x) {
   ...
   x = true;
}
```

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

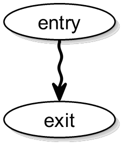
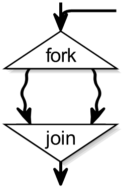
von Hanxleden, Mendler, et al.
Sequentially Constructive Concurrency—A Conservative Extension of the Synchronous Model of Computation
ACM TECS '14

SCL

SCG

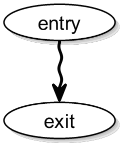
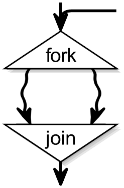
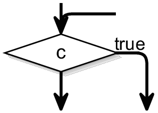
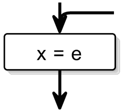
Add Sequential Constructiveness

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			

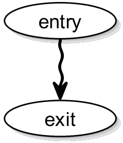
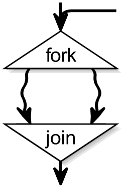
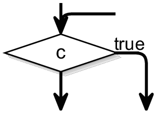
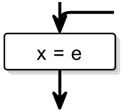
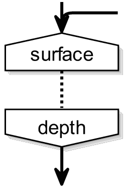
Add Sequential Constructiveness

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				

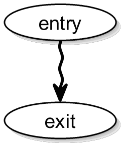
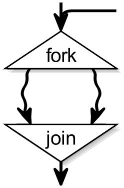
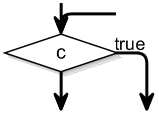
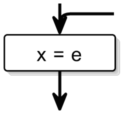
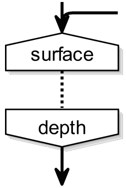
Add Sequential Constructiveness

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

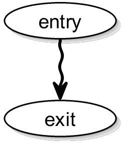
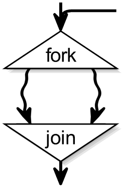
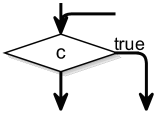
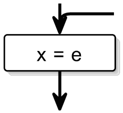
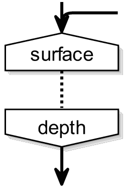
von Hanxleden, Mendler, et al.
Sequentially Constructive Concurrency—A Conservative Extension of the Synchronous Model of Computation
ACM TECS '14

	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

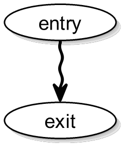
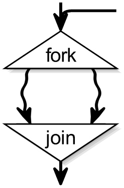
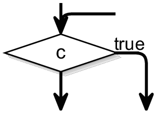
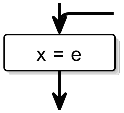
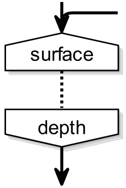
von Hanxleden, Mendler, et al.
Sequentially Constructive Concurrency—A Conservative Extension of the Synchronous Model of Computation
ACM TECS '14

	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

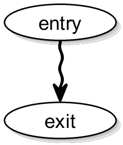
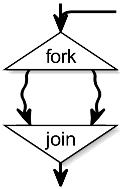
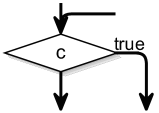
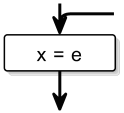
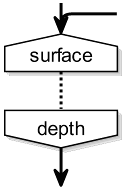
von Hanxleden, Mendler, et al.
Sequentially Constructive Concurrency—A Conservative Extension of the Synchronous Model of Computation
ACM TECS '14

	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

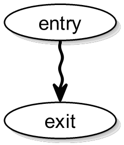
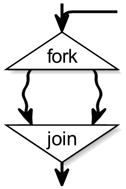
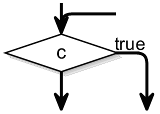
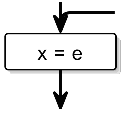
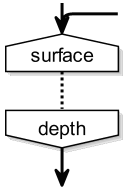
von Hanxleden, Mendler, et al.
Sequentially Constructive Concurrency—A Conservative Extension of the Synchronous Model of Computation
ACM TECS '14

	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

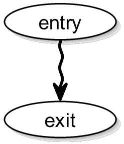
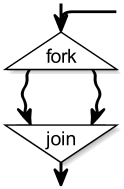
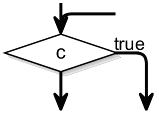
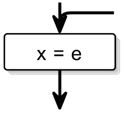
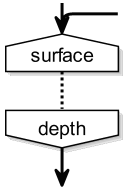
von Hanxleden, Mendler, et al.
Sequentially Constructive Concurrency—A Conservative Extension of the Synchronous Model of Computation
ACM TECS '14

	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

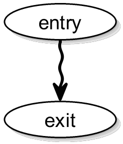
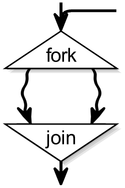
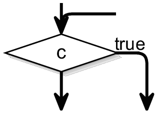
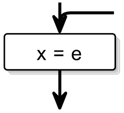
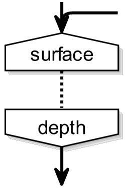
von Hanxleden, Mendler, et al.
Sequentially Constructive Concurrency—A Conservative Extension of the Synchronous Model of Computation
ACM TECS '14

	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

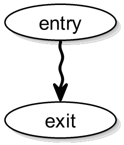
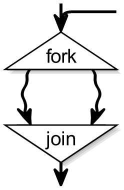
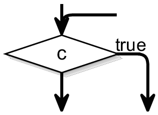
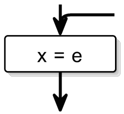
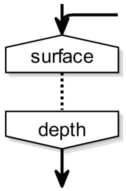
von Hanxleden, Mendler, et al.
Sequentially Constructive Concurrency—A Conservative Extension of the Synchronous Model of Computation
ACM TECS '14

	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					

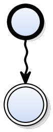
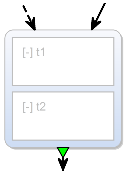
- Schedule *sequential* statements according to ;
- Schedule *concurrent* statements as
  1. Initialize ("x = 0") 2. Update ("x++") 3. Read ("y = x")
- Only reject programs that have *concurrent* causality problems, i. e., multiple concurrent initializations

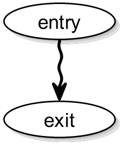
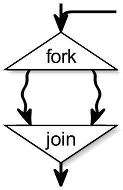
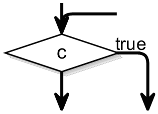
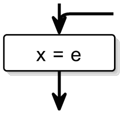
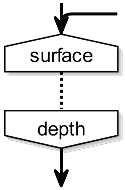
How does this fit to Statecharts? How do we compile this?
$\Longrightarrow$ This work

SCG/SCL + Statechart Syntax $\Longrightarrow$ Normalized SCCharts

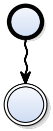
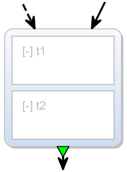
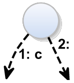
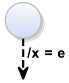
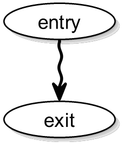
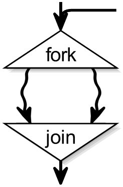
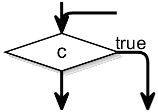
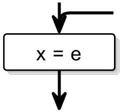
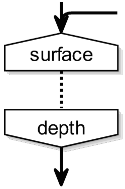
SCCharts					
	Thread	Concurrency	Conditional	Assignment	Delay
SCL	t	fork t_1 par t_2 join	if (c) s_1 else s_2	$x = e$	pause
SCG					

SCG/SCL + Statechart Syntax $\Longrightarrow$ Normalized SCCharts

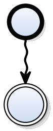
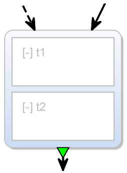
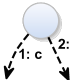
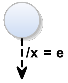
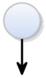
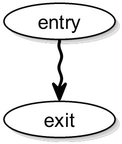
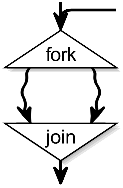
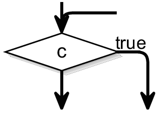
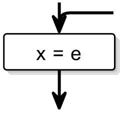
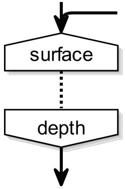
	Region	Superstate
SCCharts		

	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					

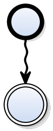
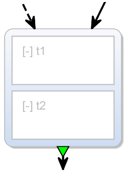
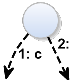
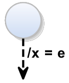
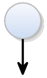
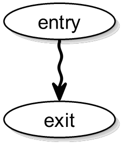
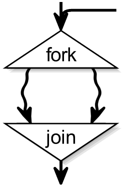
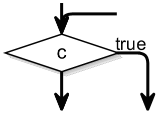
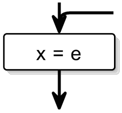
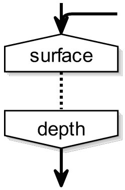
SCG/SCL + Statechart Syntax $\Longrightarrow$ Normalized SCCharts

	Region	Superstate	Trigger	Effect	
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 $\Longrightarrow$ Normalized SCCharts

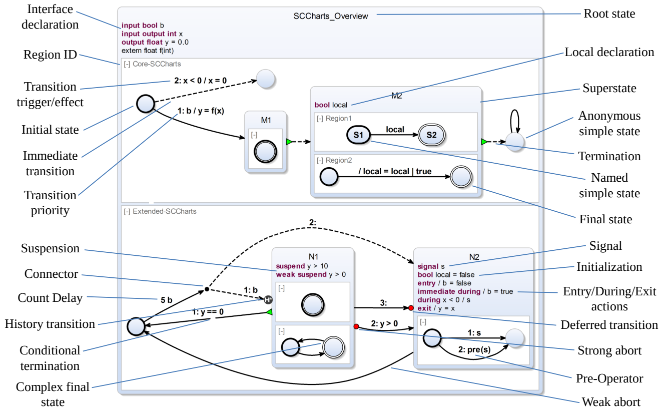
	Region	Superstate	Trigger	Effect	State
SCCharts					
	Thread	Concurrency	Conditional	Assignment	Delay
SCL	t	fork t_1 par t_2 join	if (c) s_1 else s_2	$x = e$	pause
SCG					

SCG/SCL + Statechart Syntax $\Longrightarrow$ Normalized SCCharts

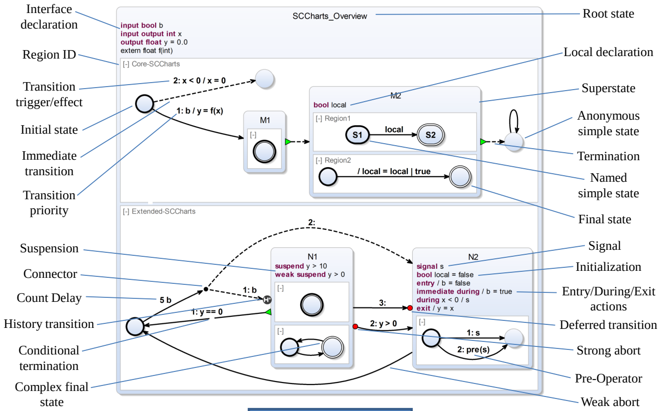
	Region	Superstate	Trigger	Effect	State
SCCharts					
	Thread	**Concurrency**	**Conditional**	**Assignment**	**Delay**
SCL	t	fork t_1 par t_2 join	if (c) s_1 else s_2	$x = e$	pause
SCG					

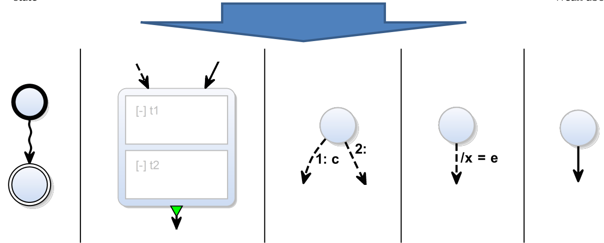
Now add some syntactic sugar . . .

SCCharts Overview

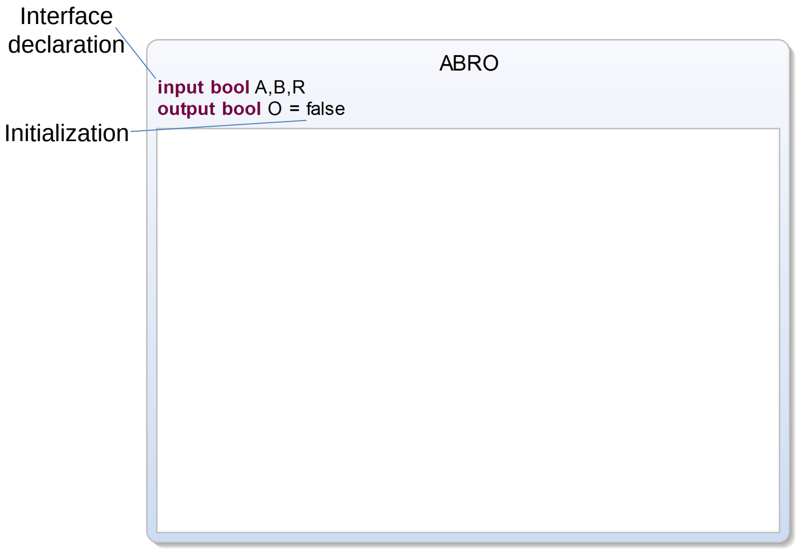


SCCharts Overview



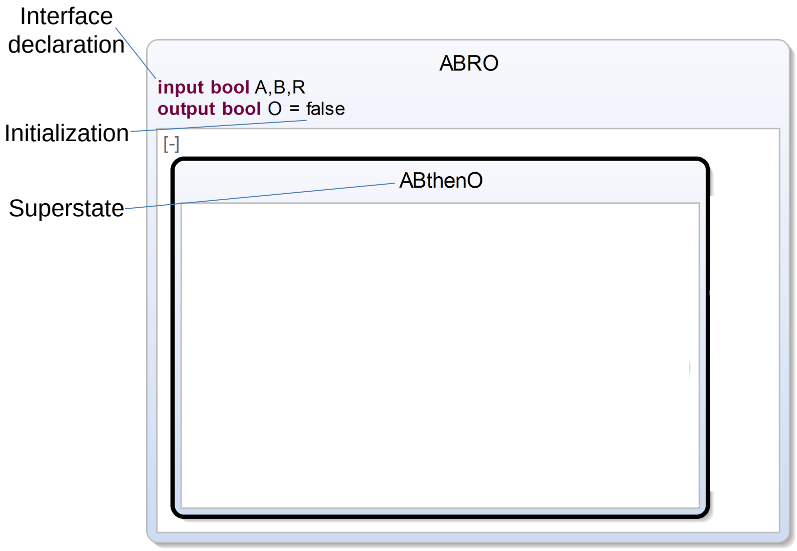


Example: ABRO



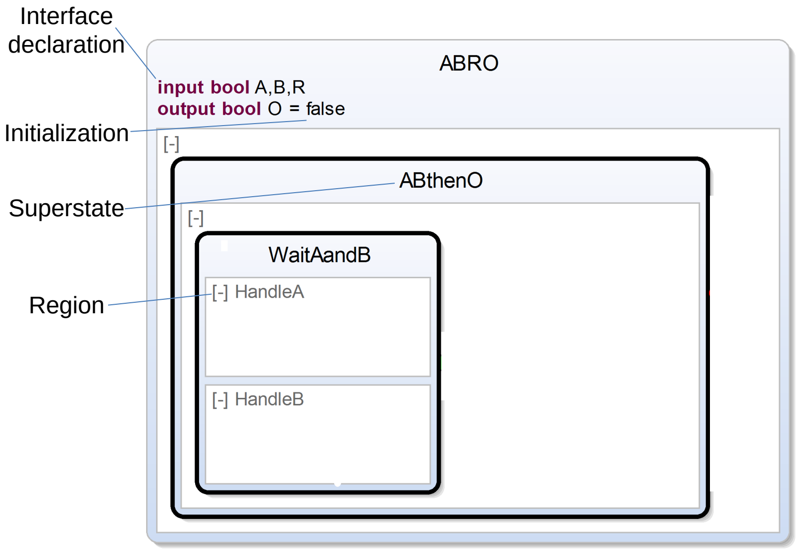
```
input bool A,B,R
output bool O = false
```

Example: ABRO

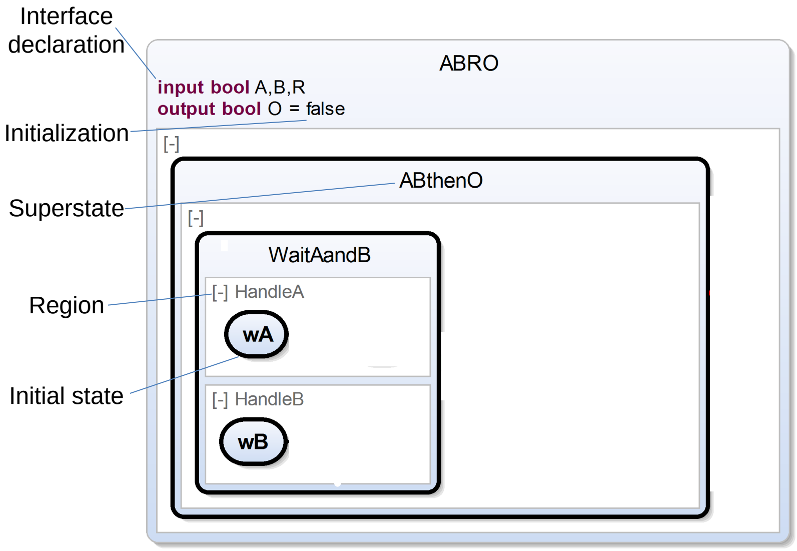


input bool A,B,R
output bool O = false

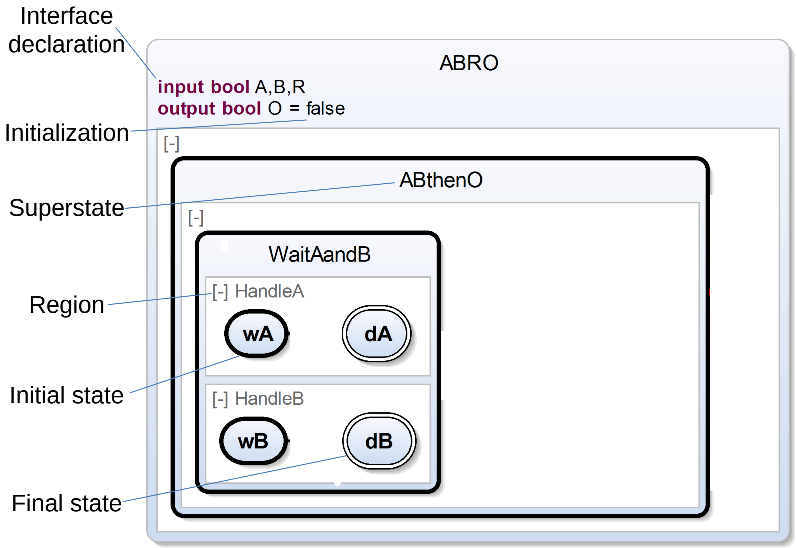
Example: ABRO



Example: ABRO

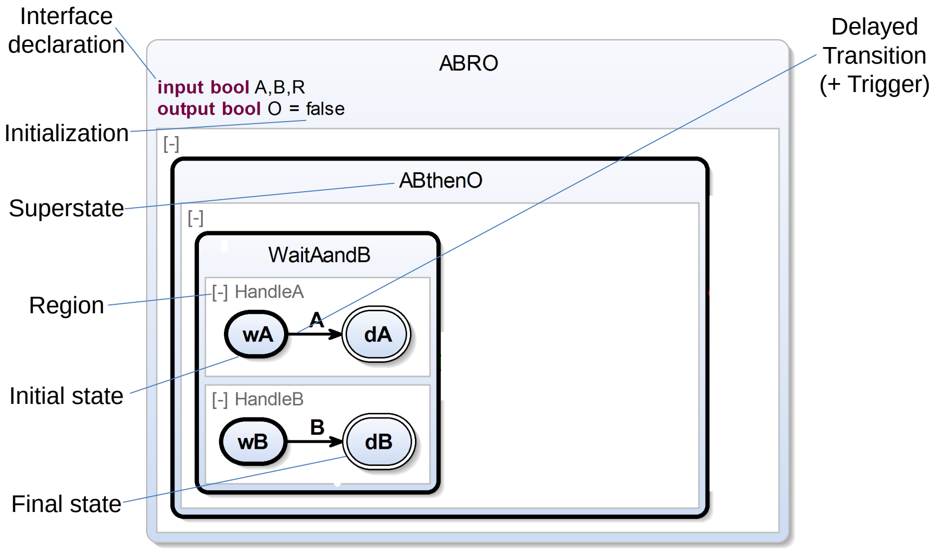


Example: ABRO

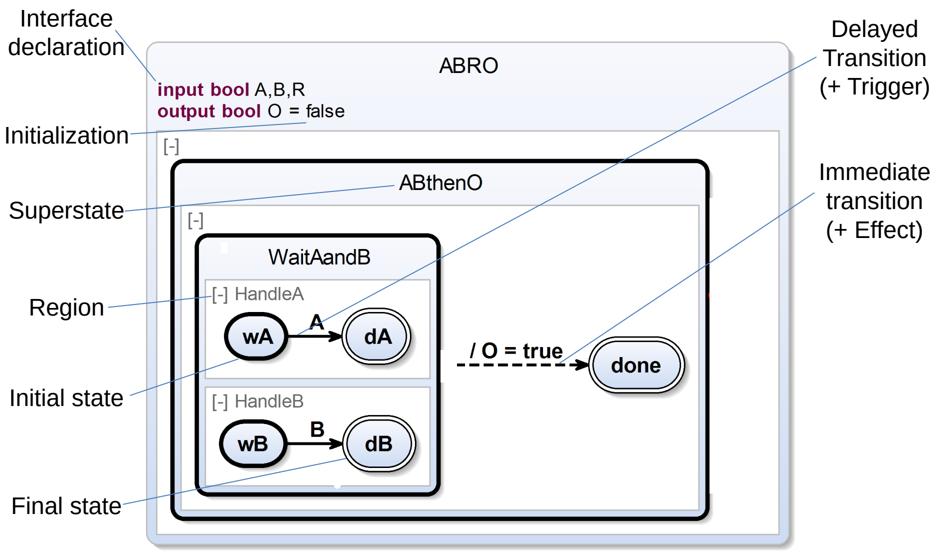


```
input bool A,B,R
output bool O = false
```

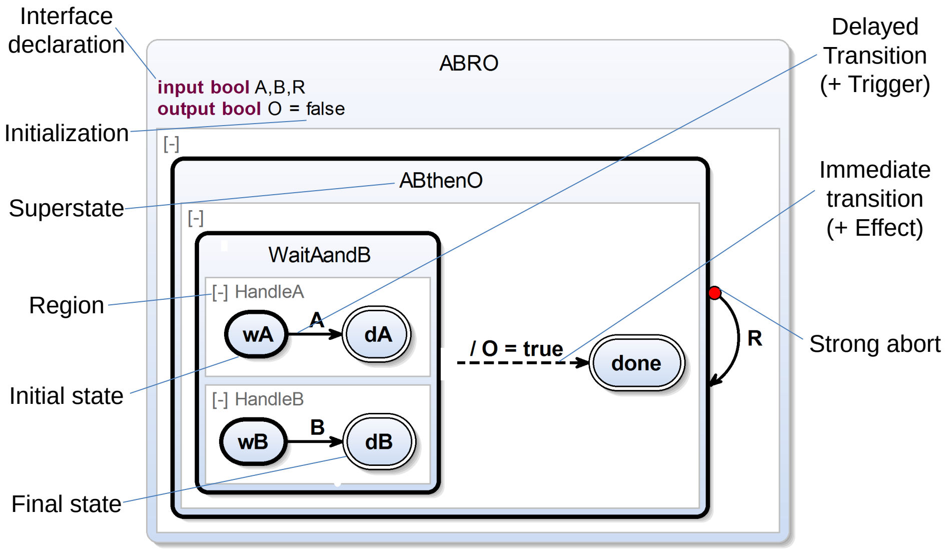
Example: ABRO



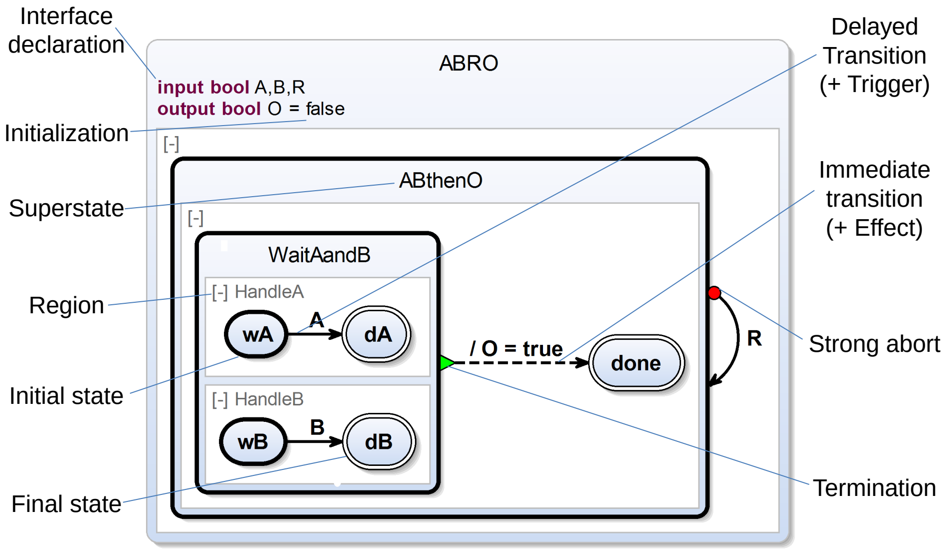
Example: ABRO



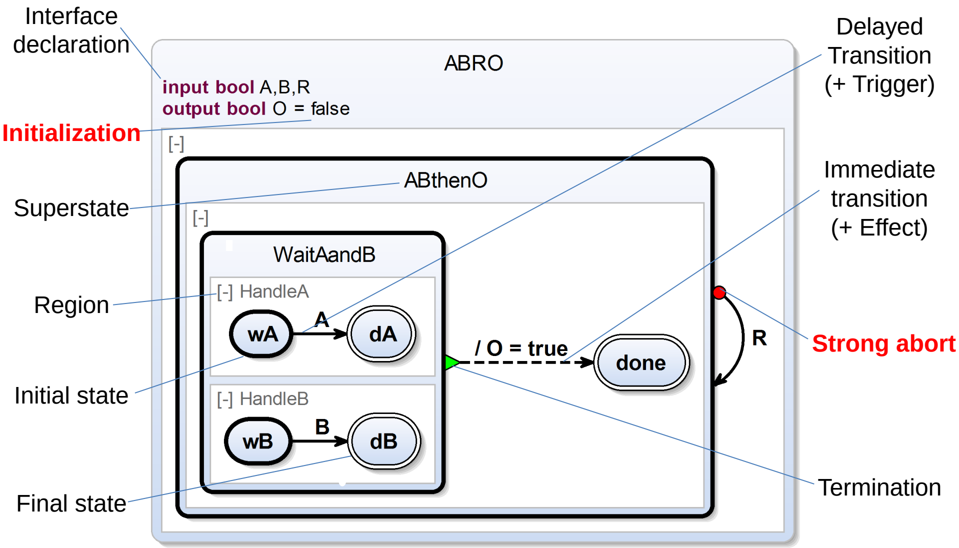
Example: ABRO



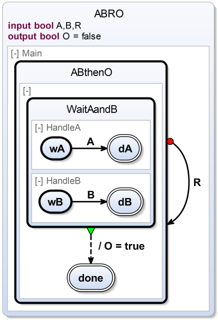
Example: ABRO



Example: ABRO—With Extended SCChart Features

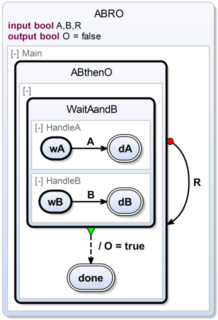


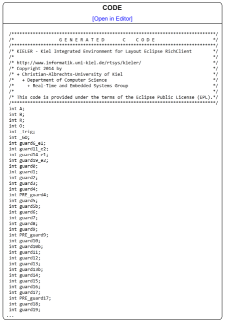
What do we want to do?



Model

What do we want to do?



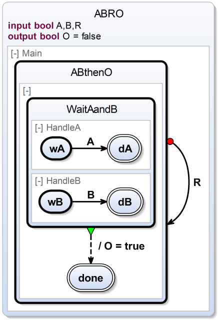


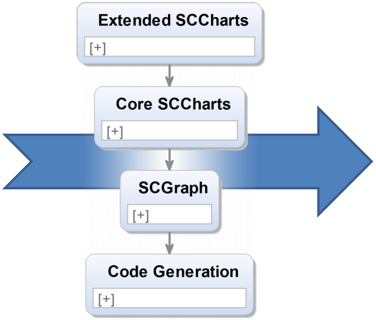
```
/*****************************************************************************/
/*                 G E N E R A T E D     C    C O D E                        */
/*****************************************************************************/
/* KIELER - Kiel Integrated Environment for Layout Eclipse RichClient        */
/*                                                                           */
/* http://www.informatik.uni-kiel.de/rtsys/kieler/                           */
/* Copyright 2014 by                                                         */
/* + Christian-Albrechts-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).*/
/*****************************************************************************/
int A;
int B;
int R;
int O;
int _trig;
int _GO;
int guard6_e1;
int guard11_e2;
int guard14_e1;
int guard19_e2;
int guard0;
int guard1;
int guard2;
int guard3;
int guard4;
int PRE_guard4;
int guard5;
int guard5b;
int guard6;
int guard7;
int guard8;
int guard9;
int PRE_guard9;
int guard10;
int guard10b;
int guard11;
int guard12;
int guard13;
int guard13b;
int guard14;
int guard15;
int guard16;
int guard17;
int PRE_guard17;
int guard18;
int guard19;
...
```

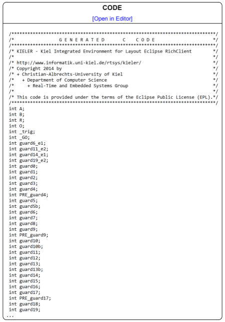
Model

Code

What do we want to do?







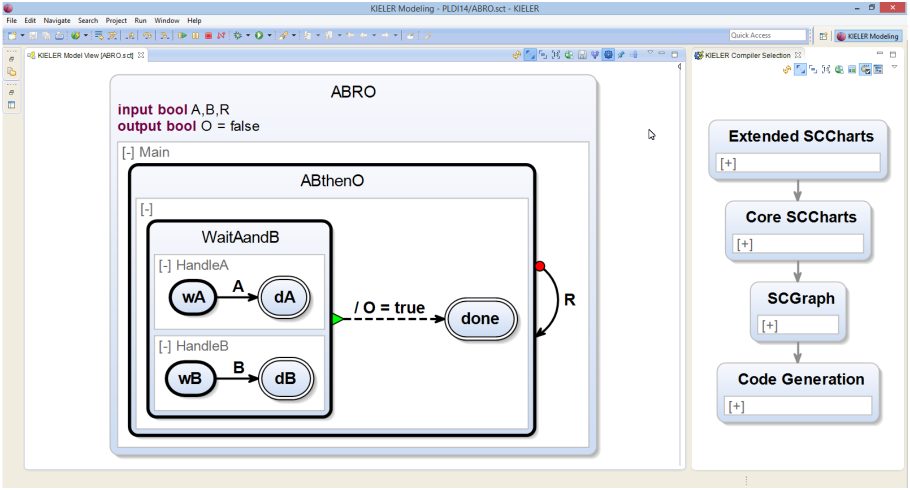
```
/******************************************************************************/
/*                   G E N E R A T E D     C    C O D E                       */
/******************************************************************************/
/* KIELER - Kiel Integrated Environment for Layout Eclipse RichClient         */
/*                                                                            */
/* http://www.informatik.uni-kiel.de/rtsys/kieler/                            */
/* Copyright 2014 by                                                          */
/* + Christian-Albrechts-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).*/
/******************************************************************************/
int A;
int B;
int R;
int O;
int _trig;
int _GO;
int guard6_e1;
int guard11_e2;
int guard14_e1;
int guard19_e2;
int guard0;
int guard1;
int guard2;
int guard3;
int guard4;
int PRE_guard4;
int guard5;
int guard5b;
int guard6;
int guard7;
int guard8;
int guard9;
int PRE_guard9;
int guard10;
int guard10b;
int guard11;
int guard12;
int guard13;
int guard13b;
int guard14;
int guard15;
int guard16;
int guard17;
int PRE_guard17;
int guard18;
int guard19;
...
```

Model

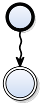
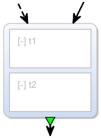
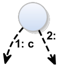
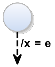
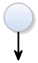
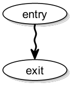
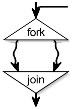
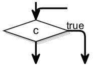
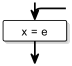
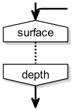
Transformation Stages

Code

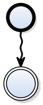
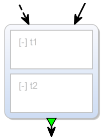
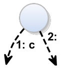
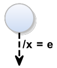
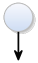
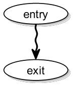
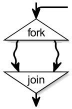
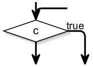
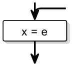
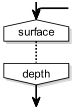
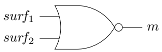
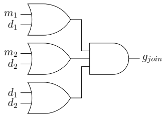
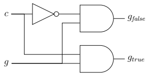
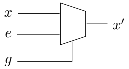
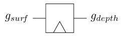
SCCharts Compiler in KIELER Eclipse Richt Client



One Approach to Code Generation: Data-Flow

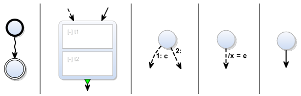
	Region (Thread)	Superstate (Concurrency)	Trigger (Conditional)	Effect (Assignment)	State (Delay)
SCCharts					
SCG					
SCL	t	fork t_1 par t_2 join	if (c) s_1 else s_2	$x = e$	pause

One Approach to Code Generation: Data-Flow

	Region (Thread)	Superstate (Concurrency)	Trigger (Conditional)	Effect (Assignment)	State (Delay)
SCCharts					
SCG					
SCL	t	fork t_1 par t_2 join	if (c) s_1 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		 m_1 d_1 m_2 d_2 d_1 d_2 g_{join}			

Summary

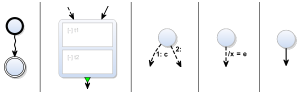
Starting point: Concurrent Statechart
5 Core Constructs +
Smörgåsbord of Extensions

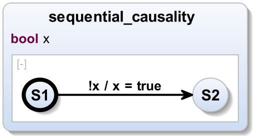


Summary

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

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



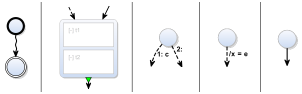


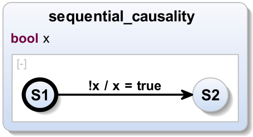
Summary

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





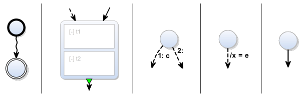
Summary

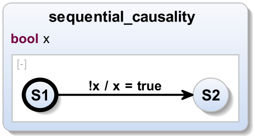
Starting point: Concurrent Statechart
5 Core Constructs +
Smörgåsbord of Extensions

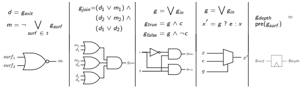
Semantics: Deterministic
Sequentially constructive
; not source of errors,
instead resolves errors

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

Result: SW or HW
Sequential or parallel







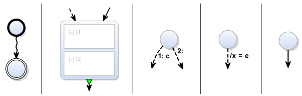
Summary

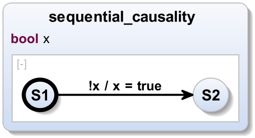
Starting point: Concurrent Statechart
5 Core Constructs +
Smörgåsbord of Extensions

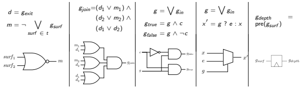
Semantics: Deterministic
Sequentially constructive
; not source of errors,
instead resolves errors

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

Result: SW or HW
Sequential or parallel







$d = g_{exit}$
$m = \neg \bigvee_{surf \in t} g_{surf}$
$g_{join} = (d_1 \vee m_1) \wedge (d_2 \vee m_2) \wedge (d_1 \vee d_2)$
$g = \bigvee g_{in}$
$g_{true} = g \wedge c$
$g_{false} = g \wedge \neg c$
$g = \bigvee g_{in}$
$x' = g\ ?\ e : x$
$g_{depth} = \mathrm{pre}(g_{surf})$

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

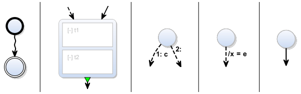
Summary

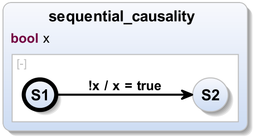
Starting point: Concurrent Statechart
5 Core Constructs +
Smörgåsbord of Extensions

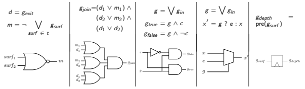
Semantics: Deterministic
Sequentially constructive
; not source of errors,
instead resolves errors

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

Result: SW or HW
Sequential or parallel





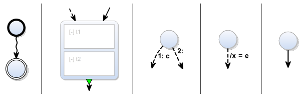

$d = g_{exit}$
$m = \neg \bigvee_{surf \in t} g_{surf}$
$g_{join} = (d_1 \vee m_1) \wedge (d_2 \vee m_2) \wedge (d_1 \vee d_2)$
$g = \bigvee g_{in}$
$g_{true} = g \wedge c$
$g_{false} = g \wedge \neg c$
$g = \bigvee g_{in}$
$x' = g \, ? \, e : x$
$g_{depth} = \mathrm{pre}(g_{surf})$

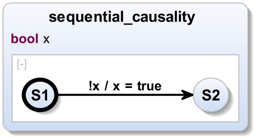
What compiler/optimization machinery can be put to work here?

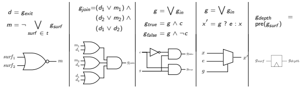
Thanks!

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





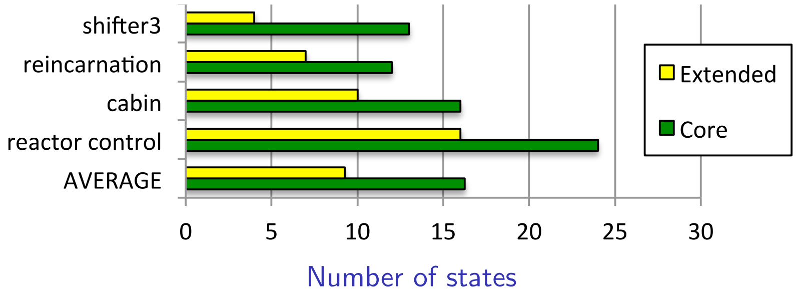

$d = g_{exit}$
$m = \neg \bigvee_{surf \in t} g_{surf}$
$g_{join} = (d_1 \vee m_1) \wedge (d_2 \vee m_2) \wedge (d_1 \vee d_2)$
$g = \bigvee g_{in}$
$g_{true} = g \wedge c$
$g_{false} = g \wedge \neg c$
$g = \bigvee g_{in}$
$x' = g ~?~ e : x$
$g_{depth} = pre(g_{surf})$

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

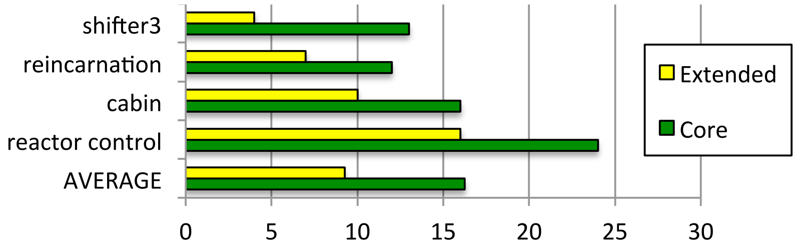
Thanks!

Questions or comments?

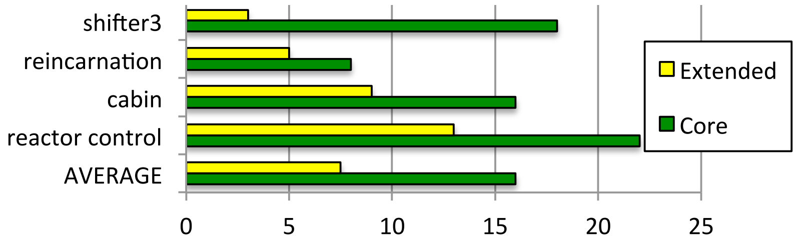
Effects of Expanding Expensions



Effects of Expanding Expensions

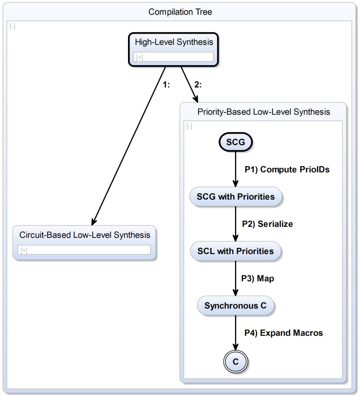


Number of states



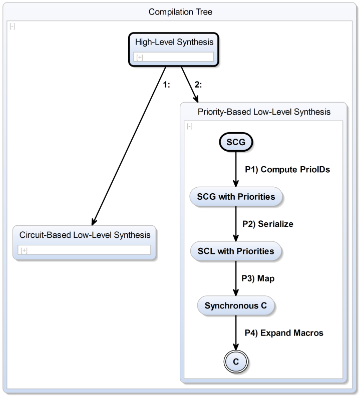
Number of transitions

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

Low-Level Synthesis II: The Priority Approach

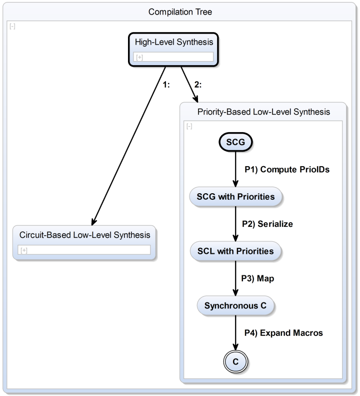


- More software-like
- Don't emulate control flow with guards/basic blocks, but with program counters/threads
- Priority-based thread dispatching
- SCL_P: SCL + PrioIDs
- Implemented as C macros

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 $\Rightarrow$ 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

```
// Boolean type
typedef int bool;
#define false 0
#define true  1

// Enable/disable threads with prioID p
#define _u2b(u)             (1 << u)
#define _enable(p)          _enabled |= _u2b(p);  \
                            active |= _u2b(p)
#define _isEnabled(p)       ((_enabled & _u2b(p)) != 0)
#define _disable(p)         _enabled &= ~_u2b(p)

// Set current thread continuation
#define _setPC(p, label) _pc[p] = &&label
```

SCL$_P$ Macros II

```
#define _pause(label)      _setPC(_cid, label); \
                           goto _L_PAUSE

// Pause, resume at pause
#define _concat_helper(a, b)  a ## b
#define _concat(a, b)      _concat_helper(a, b)
#define _label_            _concat(_L, __LINE__)
#define pause              _pause(_label_); _label_:

// Fork/join sibling thread with prioID p
#define fork1(label, p)    _setPC(p, label); _enable(p);
#define join1(p)           _label_: if (_isEnabled(p)) \
                           { _pause(_label_); }

// Terminate thread at "par"
#define par                goto _L_TERM;
```

ABO SCL$_P$ I

```
int tick()
{
  tickstart(2);
  O1 = false;
  O2 = false;

  fork1(HandleB, 1)
        {
    HandleA:
    if (!A) {
      pause;
      goto HandleA
            ;
    }
    B = true;
    O1 = true;

  } par {
```

$\Longrightarrow$

```
int tick()
{
  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; } ;
  O1 = 0;
  O2 = 0;

  _pc[1] = &&HandleB; enabled |= (1 << 1);
        active |= (1 << 1); {
    HandleA:
    if (!A) {
      _pc[_cid] = &&_L94; goto _L_PAUSE;
            _L94:;
      goto HandleA;
    }
    B = 1;
    O1 = 1;

  } 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 }
```

$\Longrightarrow$

```
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 }
```

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)	+	−