

Synchronous Languages—Lecture 11

Prof. Dr. Reinhard von Hanxleden

Christian-Albrechts Universität Kiel
Department of Computer Science
Real-Time Systems and Embedded Systems Group

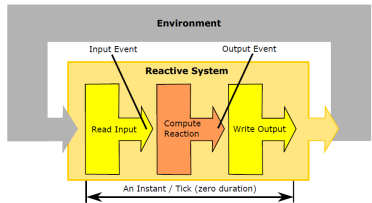
28 May 2020

Last compiled: May 27, 2020, 10:55 hrs



*SCCharts — Sequentially
Constructive Statecharts for
Safety-Critical Applications*

Reactive Embedded Systems

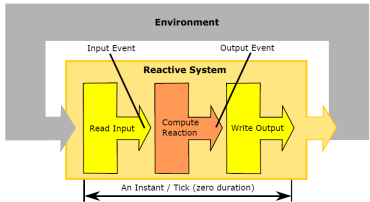


SEUS 2013



- ▶ Embedded systems react to inputs with computed outputs

Reactive Embedded Systems

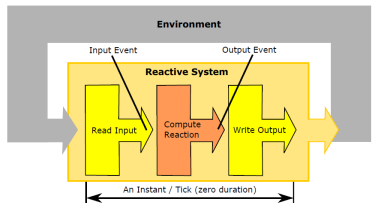


SEUS 2013



- ▶ Embedded systems react to inputs with computed outputs
- ▶ Typically **state based** computations

Reactive Embedded Systems

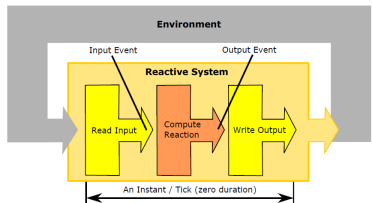


SEUS 2013



- ▶ Embedded systems react to inputs with computed outputs
- ▶ Typically **state based** computations
- ▶ Computations often exploit **concurrency**

Reactive Embedded Systems

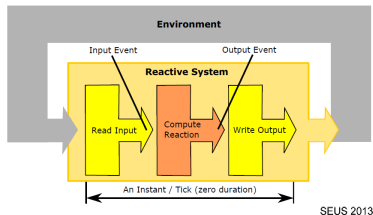


SEUS 2013



- ▶ Embedded systems react to inputs with computed outputs
- ▶ Typically **state based** computations
- ▶ Computations often exploit **concurrency** → Threads

Reactive Embedded Systems



```

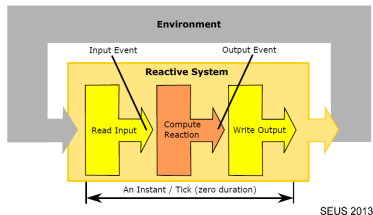
public class ValueHolder {
    private List listeners = new LinkedList();
    private int value;
    public interface Listener {
        public void valueChanged(int newValue);
    }
    public void addListener(Listener listener) {
        listeners.add(listener);
    }
    public void setValue(int newValue) {
        value = newValue;
        Iterator i = listeners.iterator();
        while(i.hasNext()) {
            ((Listener)i.next()).valueChanged(newValue);
        }
    }
}

```

E. A. Lee, The Problem with Threads, 2006

- ▶ Embedded systems react to inputs with computed outputs
- ▶ Typically **state based** computations
- ▶ Computations often exploit **concurrency** → Threads
- ▶ Threads are problematic

Reactive Embedded Systems



```

public class ValueHolder {
    private List listeners = new LinkedList();
    private int value;
    public interface Listener {
        public void valueChanged(int newValue);
    }
    public void addListener(Listener listener) {
        listeners.add(listener);
    }
    public void setValue(int newValue) {
        value = newValue;
        Iterator i = listeners.iterator();
        while(i.hasNext()) {
            ((Listener)i.next()).valueChanged(newValue);
        }
    }
}

```

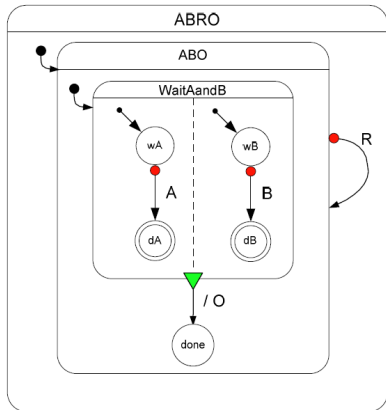
E. A. Lee, The Problem with Threads, 2006

- ▶ Embedded systems react to inputs with computed outputs
- ▶ Typically **state based** computations
- ▶ Computations often exploit **concurrency** → Threads
- ▶ Threads are problematic → **Synchronous languages**: Lustre, Esterel, SCADE, SyncCharts

SyncCharts

- ▶ **Statechart** dialect for specifying **deterministic** & robust **concurrency**

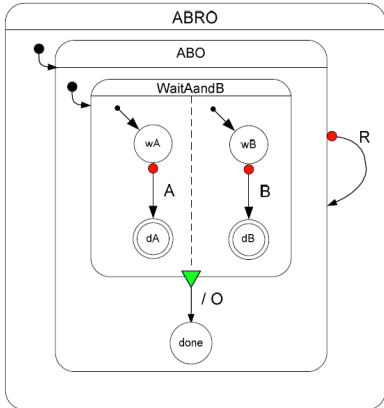
SyncCharts



[Charles André, Semantics of SyncCharts, 2003]

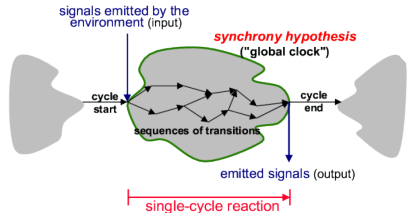
- ▶ **Statechart** dialect for specifying **deterministic** & robust **concurrency**
- ▶ SyncCharts:
 - ▶ Hierarchy, Concurrency, Broadcast

SyncCharts



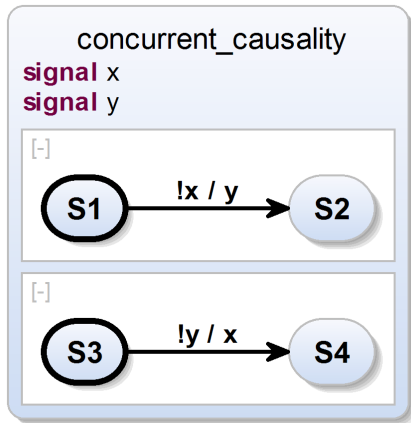
[Charles André, Semantics of SyncCharts, 2003]

- ▶ **Statechart** dialect for specifying **deterministic** & robust **concurrency**
- ▶ SyncCharts:
 - ▶ Hierarchy, Concurrency, Broadcast
 - ▶ Synchrony Hypothesis
 1. Discrete ticks
 2. Computations: Zero time

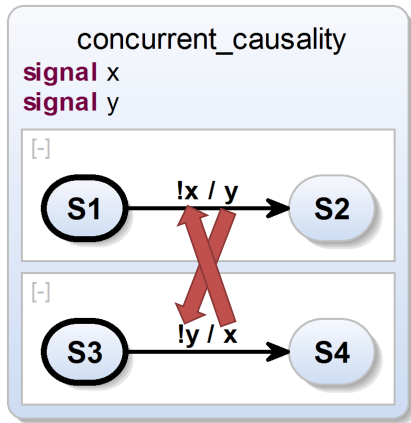


[Gerald Lüttgen, 2001]

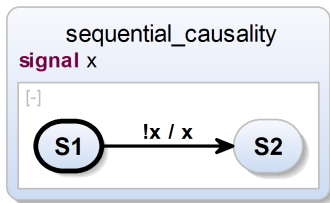
Causality in SyncCharts



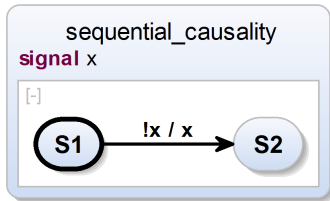
Causality in SyncCharts



Causality in SyncCharts (cont'd)

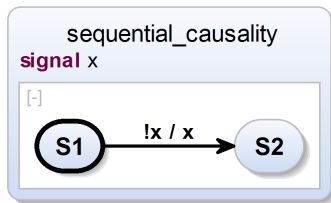


Causality in SyncCharts (cont'd)



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

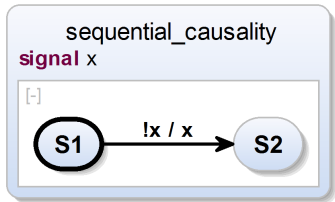
Causality in SyncCharts (cont'd)



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

- ▶ Rejected by SyncCharts compiler

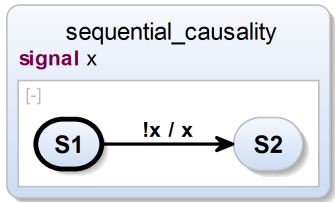
Causality in SyncCharts (cont'd)



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

- ▶ Rejected by SyncCharts compiler
- ▶ *Signal Coherence Rule*

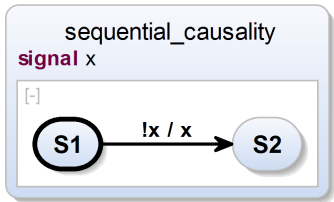
Causality in SyncCharts (cont'd)



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

- ▶ Rejected by SyncCharts compiler
- ▶ *Signal Coherence Rule*
- ▶ May seem awkward from SyncCharts perspective

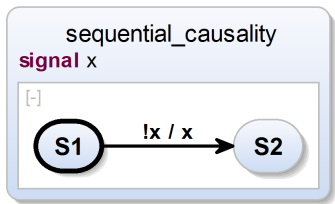
Causality in SyncCharts (cont'd)



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

- ▶ Rejected by SyncCharts compiler
- ▶ *Signal Coherence Rule*
- ▶ May seem awkward from SyncCharts perspective, but common paradigm

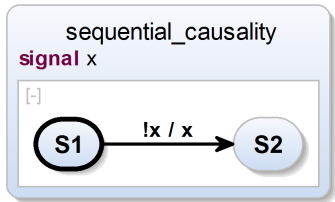
Causality in SyncCharts (cont'd)



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

- ▶ Rejected by SyncCharts compiler
- ▶ *Signal Coherence Rule*
- ▶ May seem awkward from SyncCharts perspective, but common paradigm
- ▶ Deterministic sequential execution possible

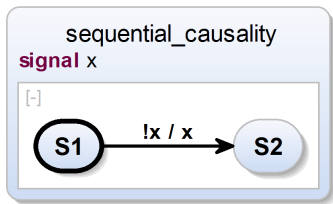
Causality in SyncCharts (cont'd)



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

- ▶ Rejected by SyncCharts compiler
- ▶ *Signal Coherence Rule*
- ▶ May seem awkward from SyncCharts perspective, but common paradigm
- ▶ Deterministic sequential execution possible using *Sequentially Constructive MoC*

Causality in SyncCharts (cont'd)



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

- ▶ Rejected by SyncCharts compiler
- ▶ *Signal Coherence Rule*
- ▶ May seem awkward from SyncCharts perspective, but common paradigm
- ▶ Deterministic sequential execution possible using *Sequentially Constructive MoC*
→ **Sequentially Constructive Charts (SCCharts)**

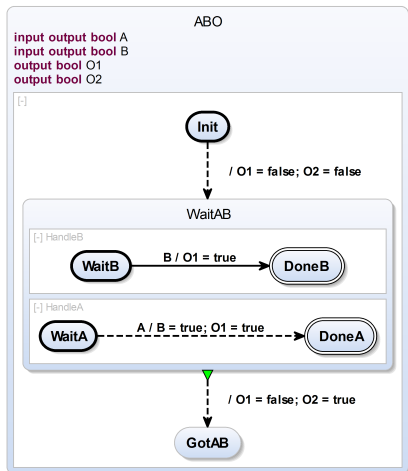
Overview

- ▶ SCCharts Overview
- ▶ Extended SCCharts → Core SCCharts
- ▶ Normalizing Core SCCharts
- ▶ Implementation in KIELER

SCCharts Overview

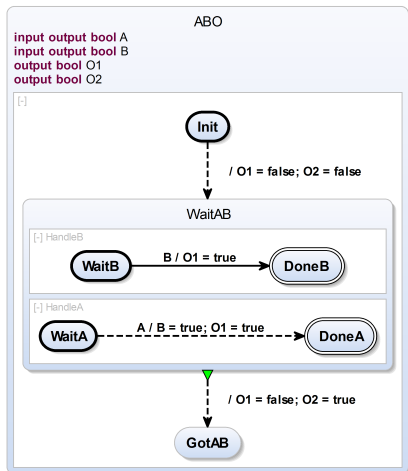
- ▶ SCCharts $\hat{=}$
SyncCharts syntax +
Sequentially Constructive semantics

SCCharts Overview



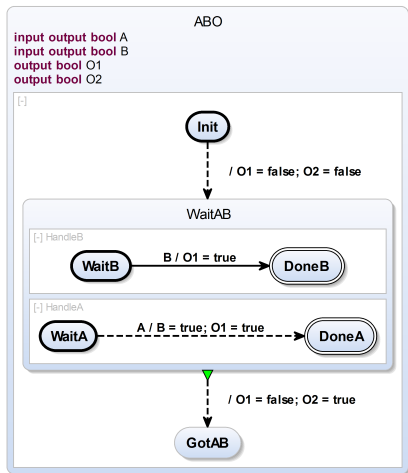
- ▶ SCCharts $\hat{=}$ SyncCharts syntax + Sequentially Constructive semantics
- ▶ *Hello World* of Sequential Constructiveness: **ABO**

SCCharts Overview



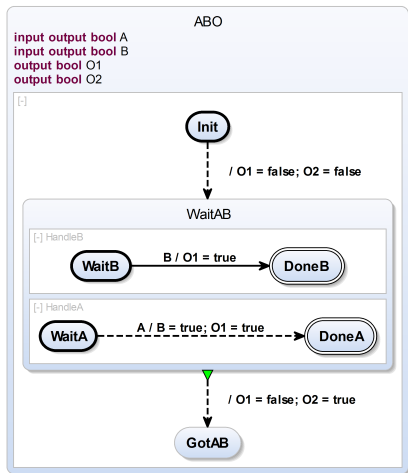
- ▶ SCCharts $\hat{=}$ SyncCharts syntax + Sequentially Constructive semantics
- ▶ *Hello World* of Sequential Constructiveness: **ABO**
 - ▶ Variables instead of signals

SCCharts Overview



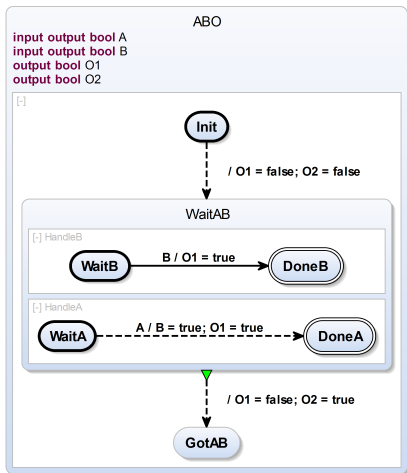
- ▶ SCCharts $\hat{=}$ SyncCharts syntax + Sequentially Constructive semantics
- ▶ *Hello World* of Sequential Constructiveness: **ABO**
 - ▶ Variables instead of signals
 - ▶ Behavior (briefly)

SCCharts Overview



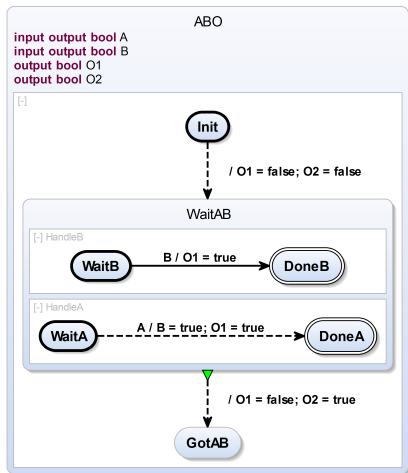
- ▶ SCCharts $\hat{=}$ SyncCharts syntax + Sequentially Constructive semantics
- ▶ *Hello World* of Sequential Constructiveness: **ABO**
 - ▶ Variables instead of signals
 - ▶ Behavior (briefly)
 1. Initialize

SCCharts Overview



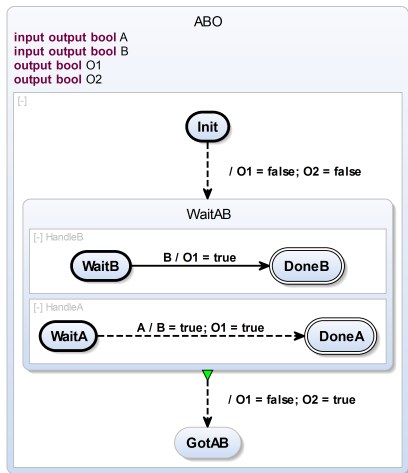
- ▶ SCCharts $\hat{=}$ SyncCharts syntax + Sequentially Constructive semantics
- ▶ *Hello World* of Sequential Constructiveness: **ABO**
 - ▶ Variables instead of signals
 - ▶ Behavior (briefly)
 1. Initialize
 2. Concurrently wait for inputs *A* or *B* to become *true*

SCCharts Overview



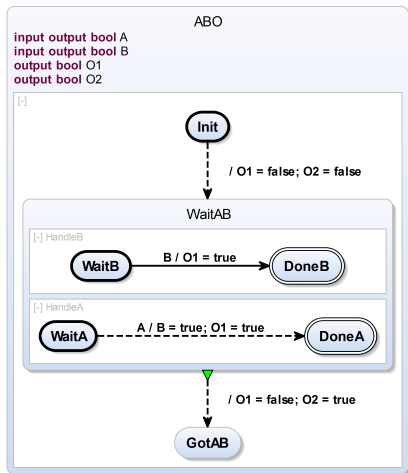
- ▶ SCCharts $\hat{=}$ SyncCharts syntax + Sequentially Constructive semantics
- ▶ *Hello World* of Sequential Constructiveness: **ABO**
 - ▶ Variables instead of signals
 - ▶ Behavior (briefly)
 1. Initialize
 2. Concurrently wait for inputs *A* or *B* to become *true*
 3. Once *A* and *B* are true after the initial tick, take *Termination*

SCCharts Overview



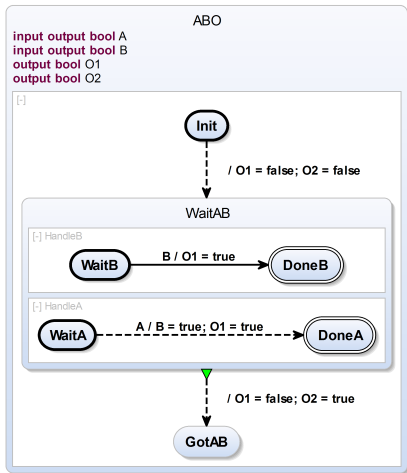
- ▶ SCCharts $\hat{=}$
 - SyncCharts syntax +
 - Sequentially Constructive semantics
- ▶ *Hello World* of Sequential Constructiveness: **ABO**
 - ▶ Variables instead of signals
 - ▶ Behavior (briefly)
 1. Initialize
 2. Concurrently wait for inputs *A* or *B* to become *true*
 3. Once *A* and *B* are true after the initial tick, take *Termination*
 4. Sequentially set *O1* and *O2*

SCCharts Overview

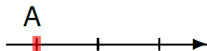


- ▶ SCCharts $\hat{=}$ SyncCharts syntax + Sequentially Constructive semantics
- ▶ *Hello World* of Sequential Constructiveness: **ABO**
 - ▶ Variables instead of signals
 - ▶ Behavior (briefly)
 1. Initialize
 2. Concurrently wait for inputs *A* or *B* to become *true*
 3. Once *A* and *B* are true after the initial tick, take *Termination*
 4. Sequentially set *O1* and *O2*

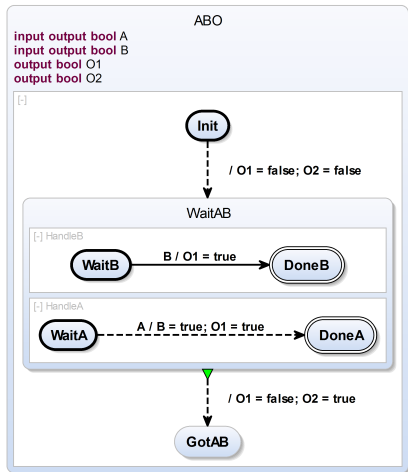
SCCharts Overview



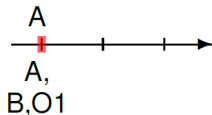
- ▶ SCCharts $\hat{=}$
 - SyncCharts syntax +
 - Sequentially Constructive semantics
- ▶ *Hello World* of Sequential Constructiveness: **ABO**
 - ▶ Variables instead of signals
 - ▶ Behavior (briefly)
 1. Initialize
 2. Concurrently wait for inputs *A* or *B* to become *true*
 3. Once *A* and *B* are true after the initial tick, take *Termination*
 4. Sequentially set *O1* and *O2*



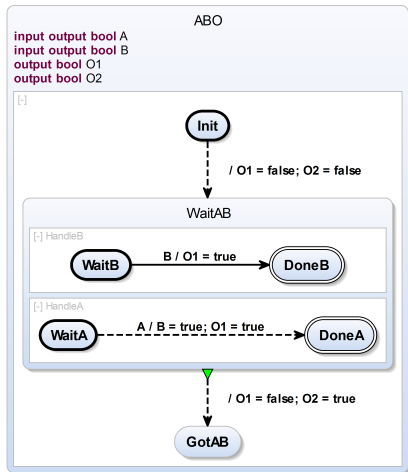
SCCharts Overview



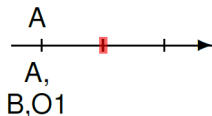
- ▶ SCCharts $\hat{=}$
 - SyncCharts syntax +
 - Sequentially Constructive semantics
- ▶ *Hello World* of Sequential Constructiveness: **ABO**
 - ▶ Variables instead of signals
 - ▶ Behavior (briefly)
 1. Initialize
 2. Concurrently wait for inputs *A* or *B* to become *true*
 3. Once *A* and *B* are true after the initial tick, take *Termination*
 4. Sequentially set *O1* and *O2*



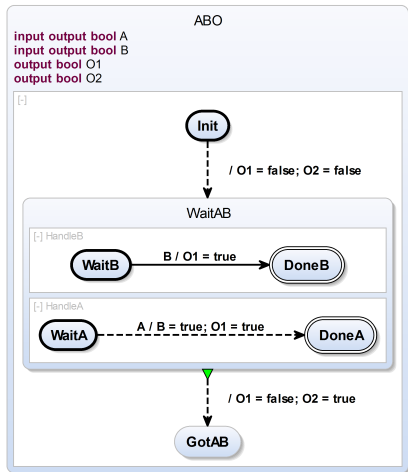
SCCharts Overview



- ▶ SCCharts $\hat{=}$
 - SyncCharts syntax +
 - Sequentially Constructive semantics
- ▶ *Hello World* of Sequential Constructiveness: **ABO**
 - ▶ Variables instead of signals
 - ▶ Behavior (briefly)
 1. Initialize
 2. Concurrently wait for inputs *A* or *B* to become *true*
 3. Once *A* and *B* are true after the initial tick, take *Termination*
 4. Sequentially set *O1* and *O2*



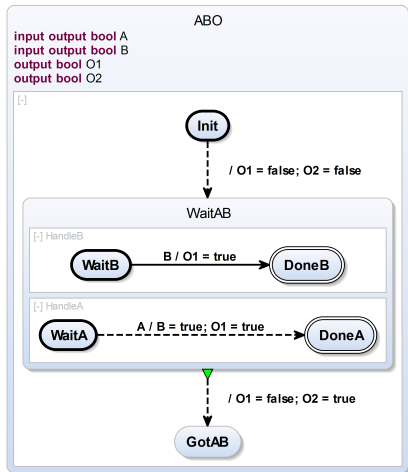
SCCharts Overview



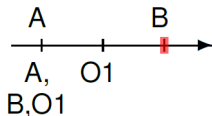
- ▶ SCCharts $\hat{=}$
 - SyncCharts syntax +
 - Sequentially Constructive semantics
- ▶ *Hello World* of Sequential Constructiveness: **ABO**
 - ▶ Variables instead of signals
 - ▶ Behavior (briefly)
 1. Initialize
 2. Concurrently wait for inputs *A* or *B* to become *true*
 3. Once *A* and *B* are true after the initial tick, take *Termination*
 4. Sequentially set *O1* and *O2*



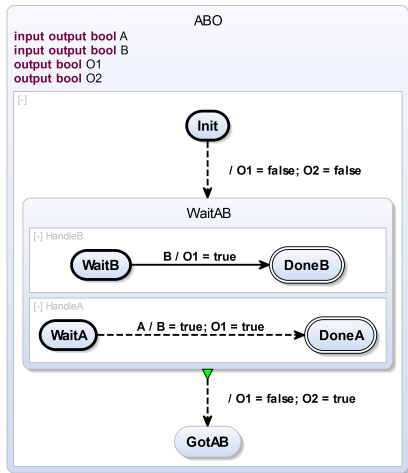
SCCharts Overview



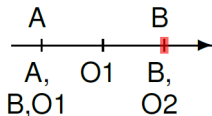
- ▶ SCCharts $\hat{=}$
 - SyncCharts syntax +
 - Sequentially Constructive semantics
- ▶ *Hello World* of Sequential Constructiveness: **ABO**
 - ▶ Variables instead of signals
 - ▶ Behavior (briefly)
 1. Initialize
 2. Concurrently wait for inputs *A* or *B* to become *true*
 3. Once *A* and *B* are true after the initial tick, take *Termination*
 4. Sequentially set *O1* and *O2*



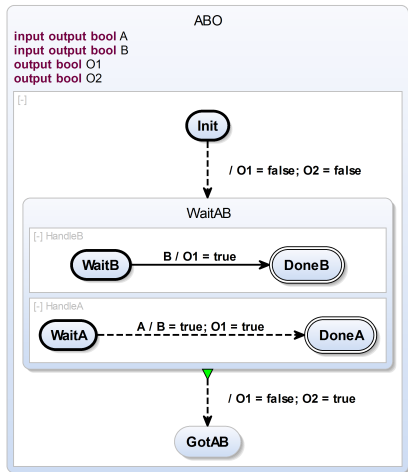
SCCharts Overview



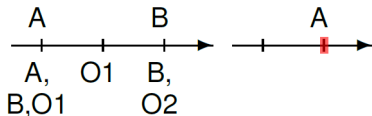
- ▶ SCCharts $\hat{=}$
 - SyncCharts syntax +
 - Sequentially Constructive semantics
- ▶ *Hello World* of Sequential Constructiveness: **ABO**
 - ▶ Variables instead of signals
 - ▶ Behavior (briefly)
 1. Initialize
 2. Concurrently wait for inputs *A* or *B* to become *true*
 3. Once *A* and *B* are true after the initial tick, take *Termination*
 4. Sequentially set *O1* and *O2*



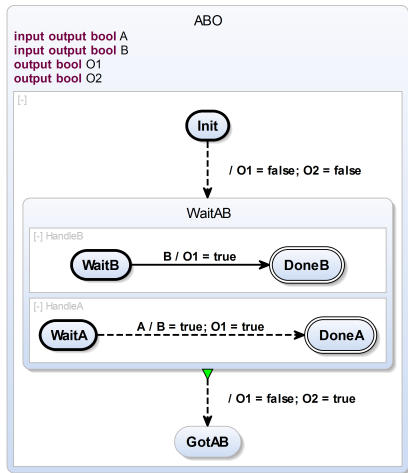
SCCharts Overview



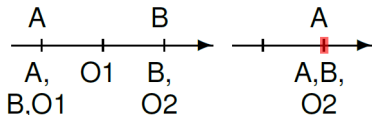
- ▶ SCCharts $\hat{=}$
 - SyncCharts syntax +
 - Sequentially Constructive semantics
- ▶ *Hello World* of Sequential Constructiveness: **ABO**
 - ▶ Variables instead of signals
 - ▶ Behavior (briefly)
 1. Initialize
 2. Concurrently wait for inputs *A* or *B* to become *true*
 3. Once *A* and *B* are true after the initial tick, take *Termination*
 4. Sequentially set *O1* and *O2*



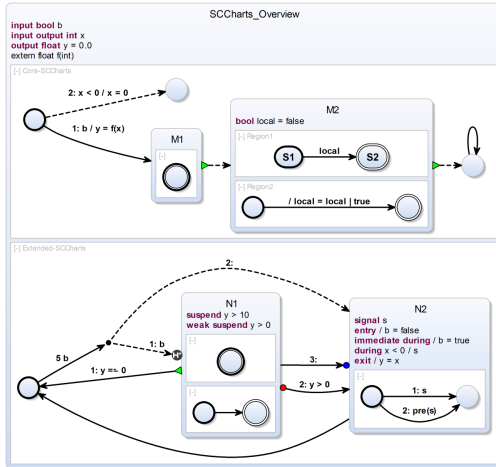
SCCharts Overview



- ▶ SCCharts $\hat{=}$
 - SyncCharts syntax +
 - Sequentially Constructive semantics
- ▶ *Hello World* of Sequential Constructiveness: **ABO**
 - ▶ Variables instead of signals
 - ▶ Behavior (briefly)
 1. Initialize
 2. Concurrently wait for inputs *A* or *B* to become *true*
 3. Once *A* and *B* are true after the initial tick, take *Termination*
 4. Sequentially set *O1* and *O2*



SCCharts — Features

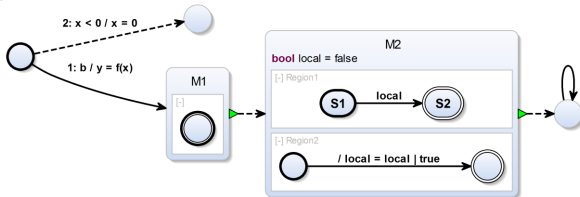


SCCharts_Overview

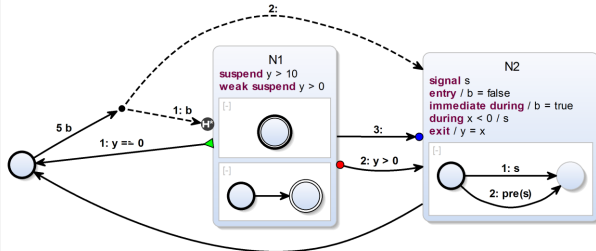
Root state

input bool b
 input output int x
 output float y = 0,0
 extern float f(int)

[-] Core-SCCharts



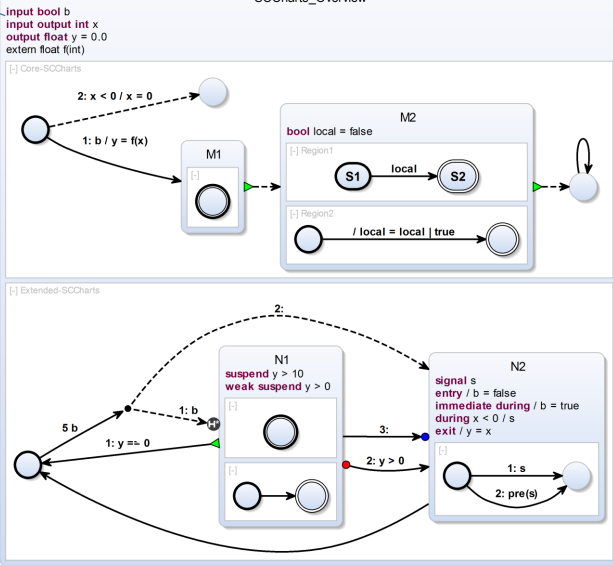
[-] Extended-SCCharts



Interface
declaration

SCCharts_Overview

Root state

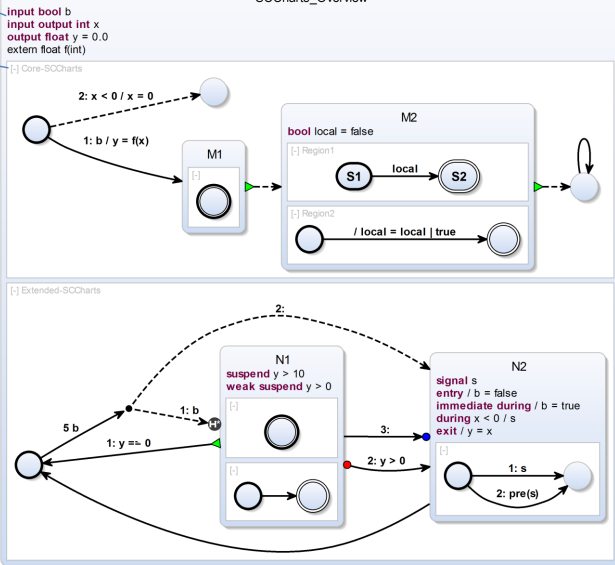


Interface declaration

Region ID

SCCharts_Overview

Root state



Interface declaration

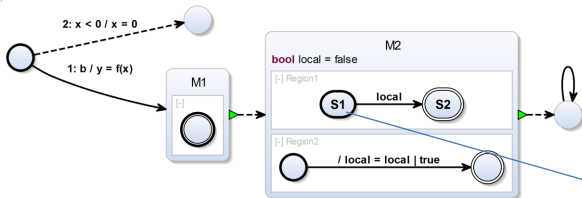
Region ID

SCCharts_Overview

Root state

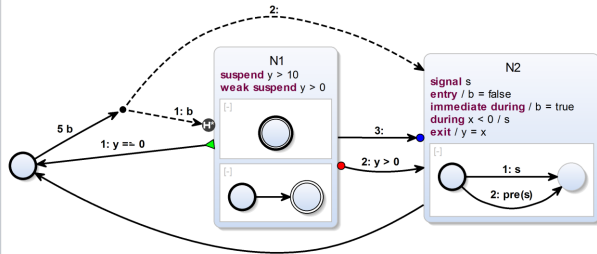
input bool b
input output int x
output float y = 0,0
extern float f(int)

[-] Core-SCCharts



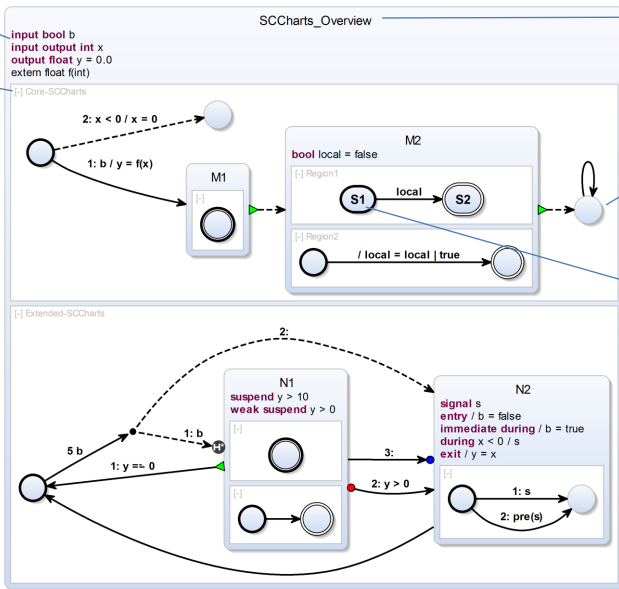
Named simple states

[-] Extended-SCCharts



Interface declaration

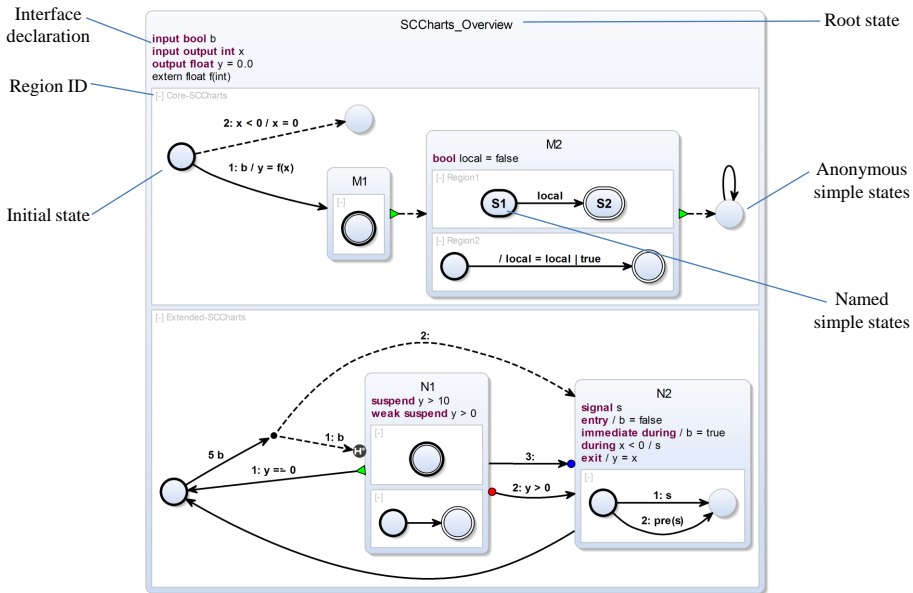
Region ID

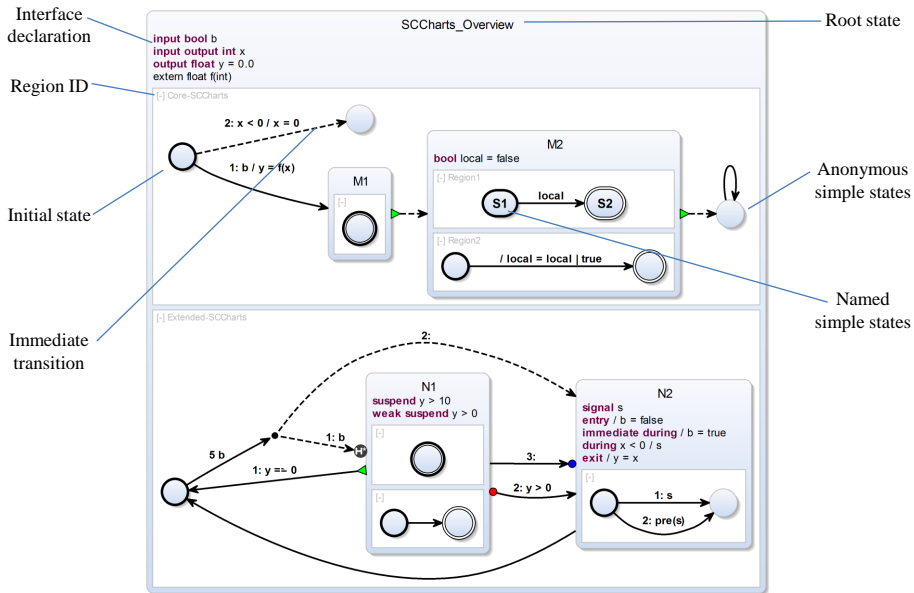


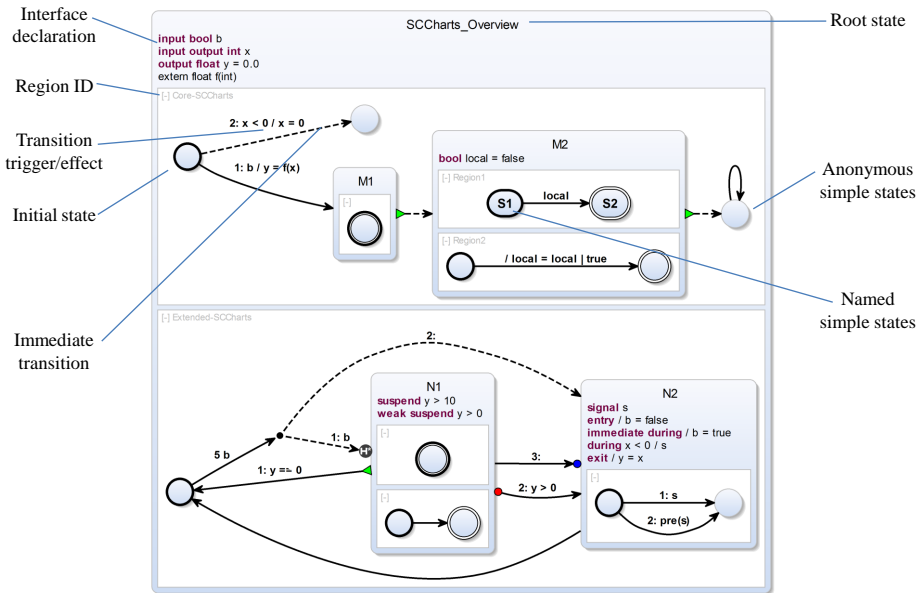
Root state

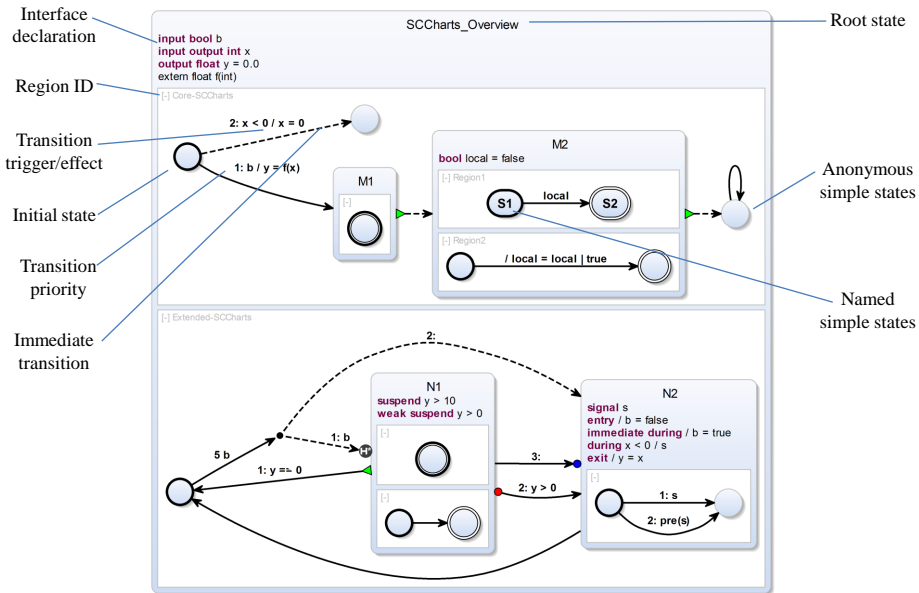
Anonymous simple states

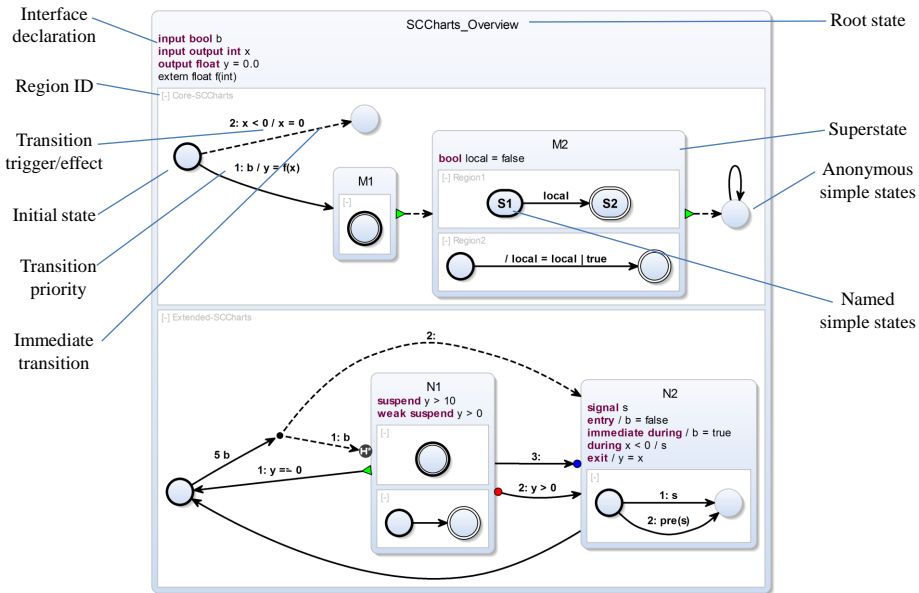
Named simple states

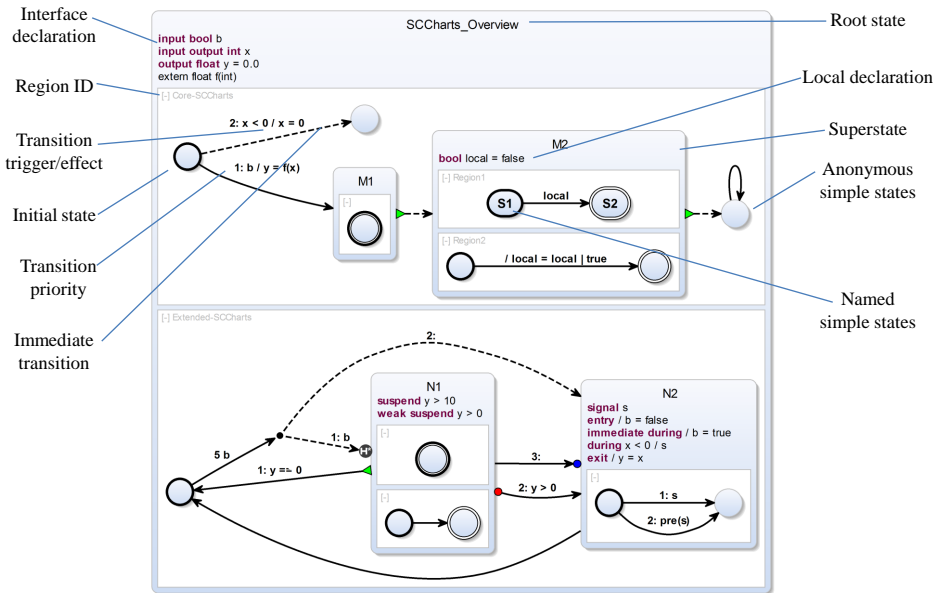


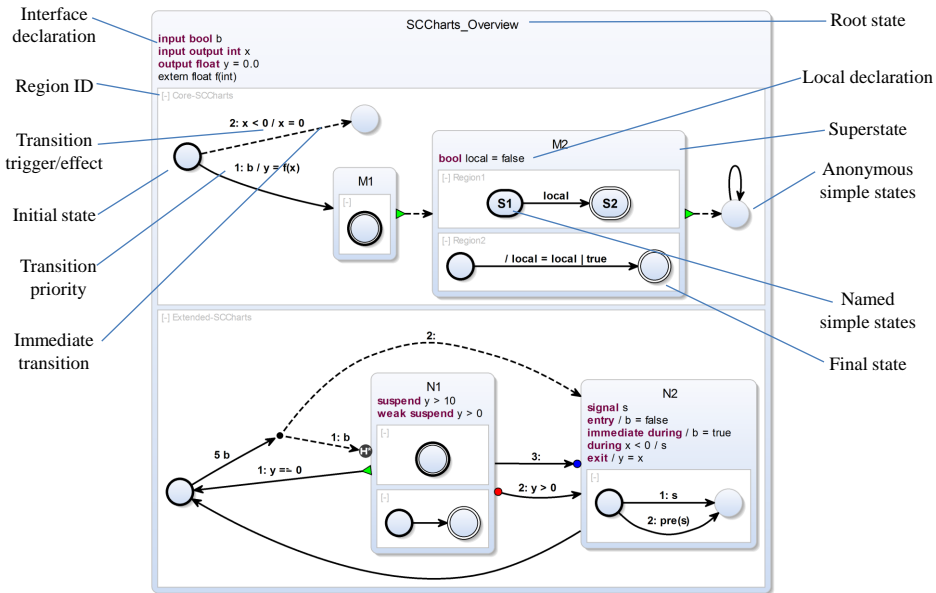












Interface declaration

Region ID

Transition trigger/effect

Initial state

Transition priority

Immediate transition

SCCharts_Overview

Root state

Local declaration

Superstate

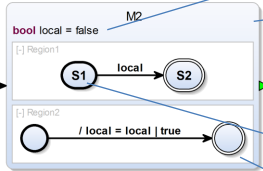
Anonymous simple states

Named simple states

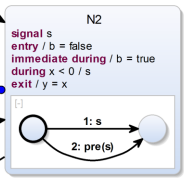
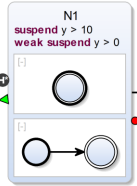
Final state

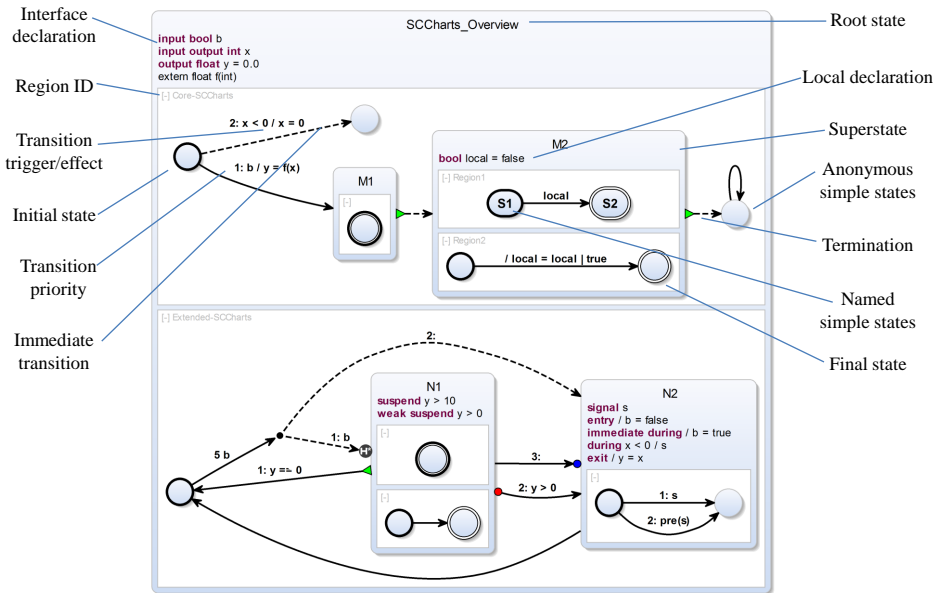
`input bool b`
`input output int x`
`output float y = 0,0`
`extern float f(int)`

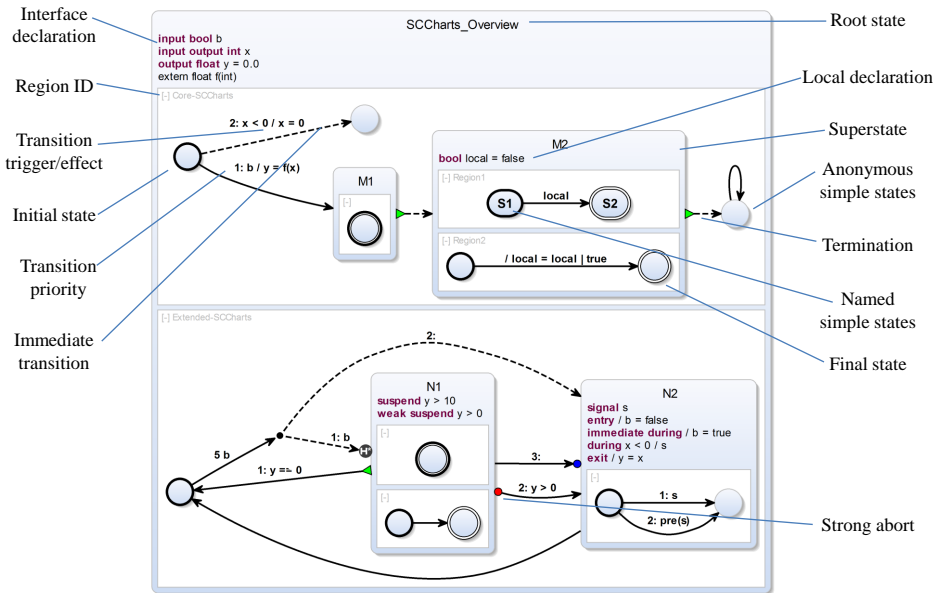
`[-] Core-SCCharts`

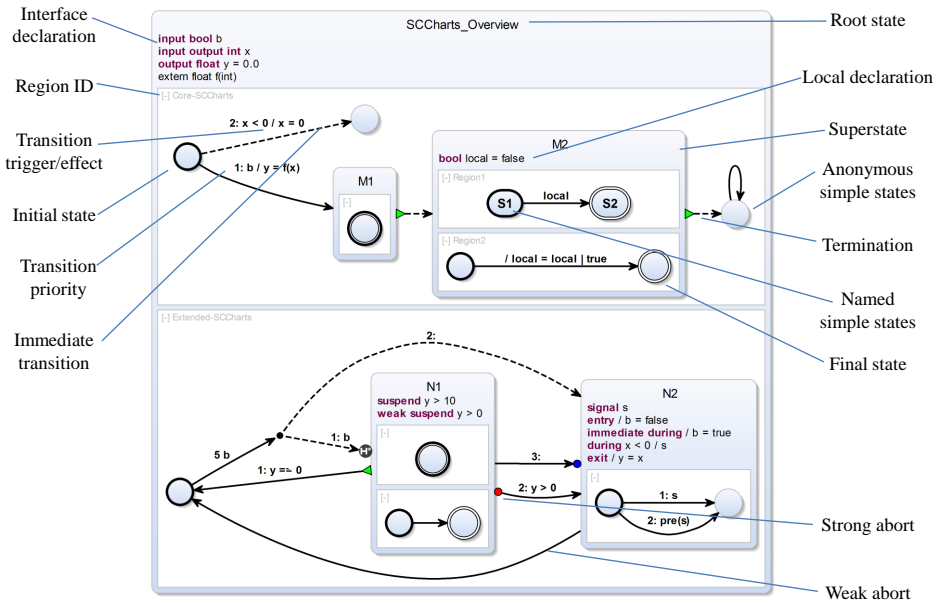


`[-] Extended-SCCharts`









Interface declaration

Region ID

Transition trigger/effect

Initial state

Transition priority

Immediate transition

Root state

Local declaration

Superstate

Anonymous simple states

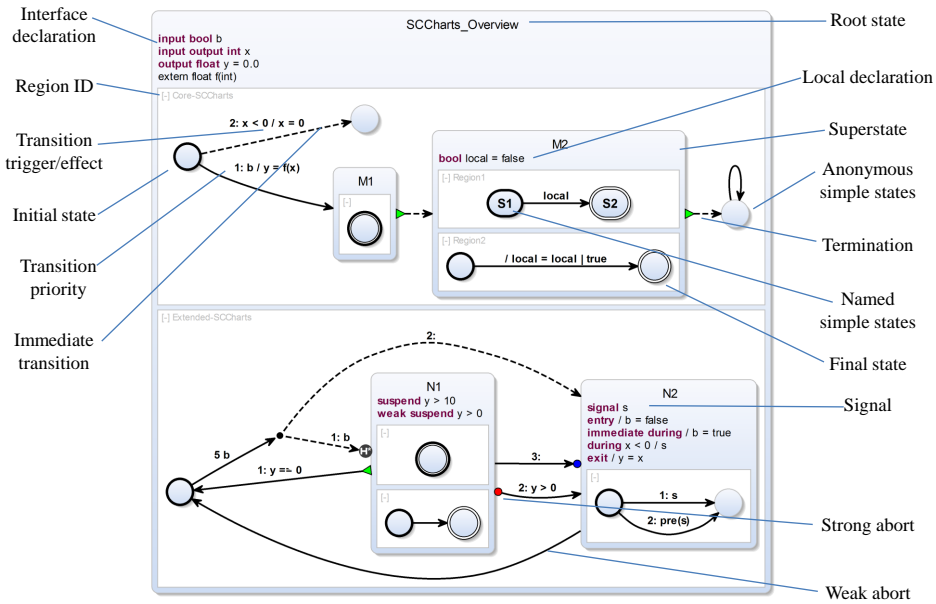
Termination

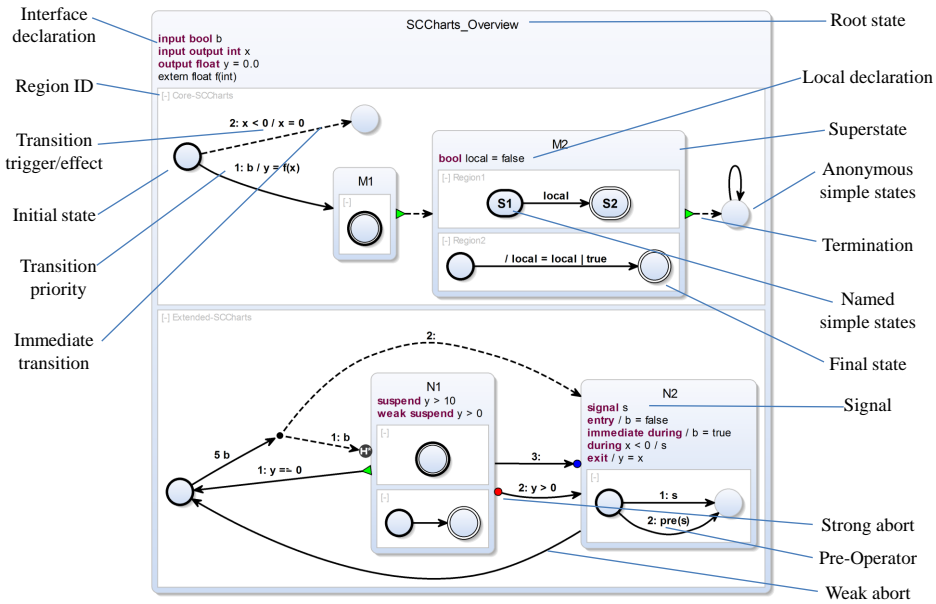
Named simple states

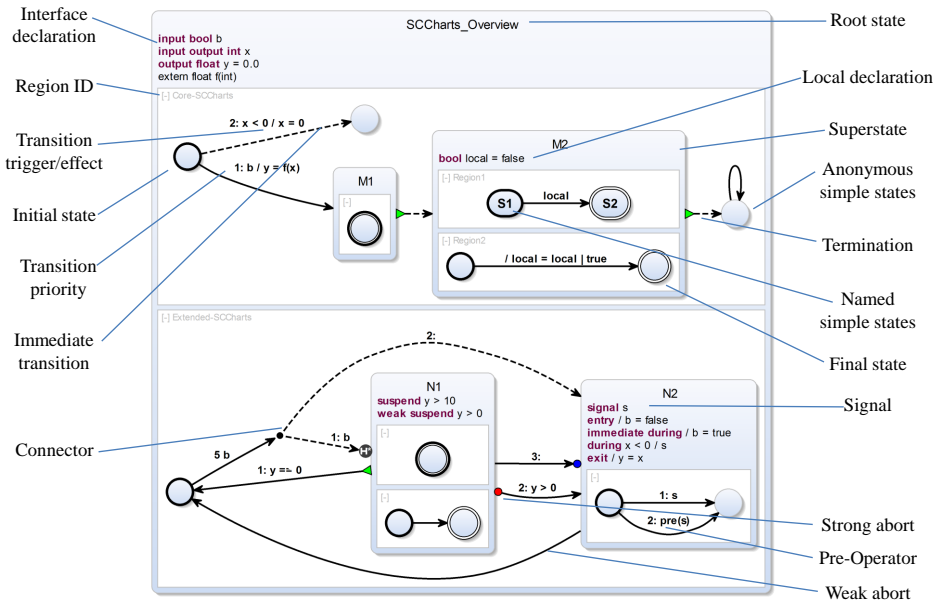
Final state

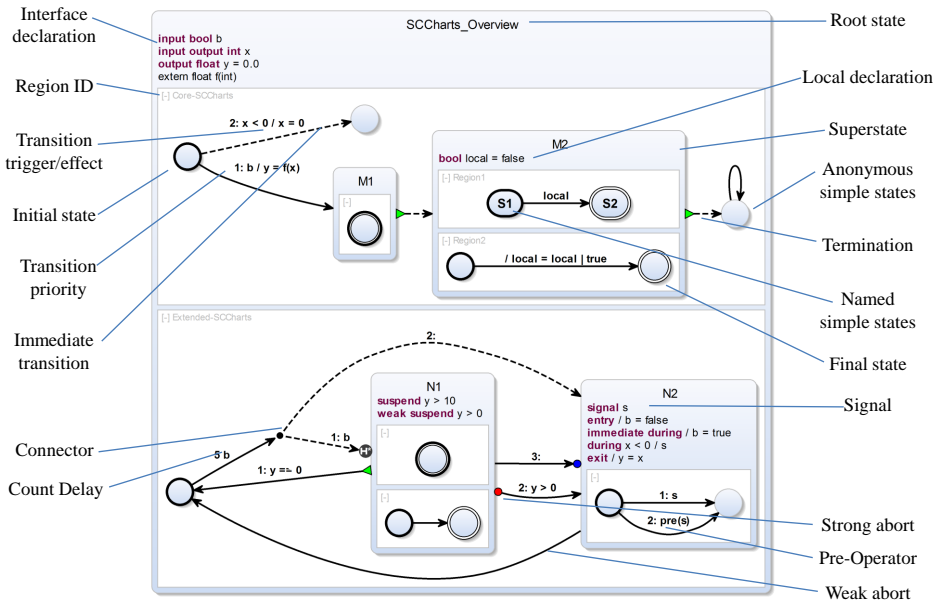
Strong abort

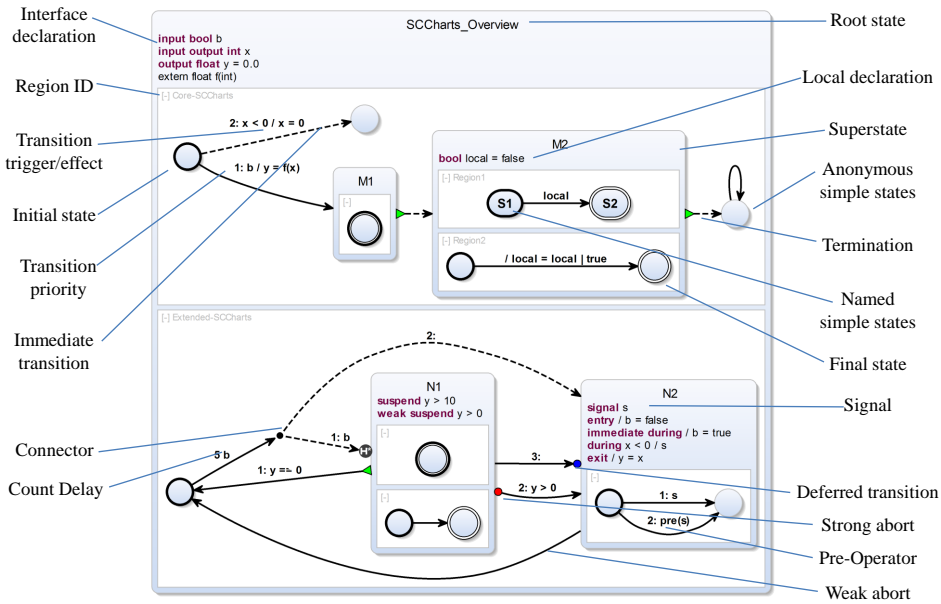
Weak abort

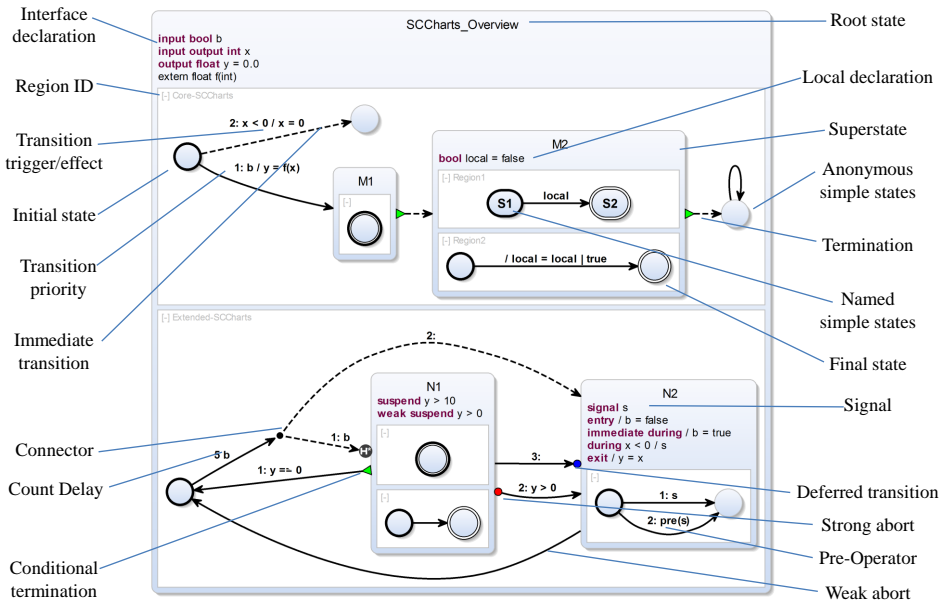


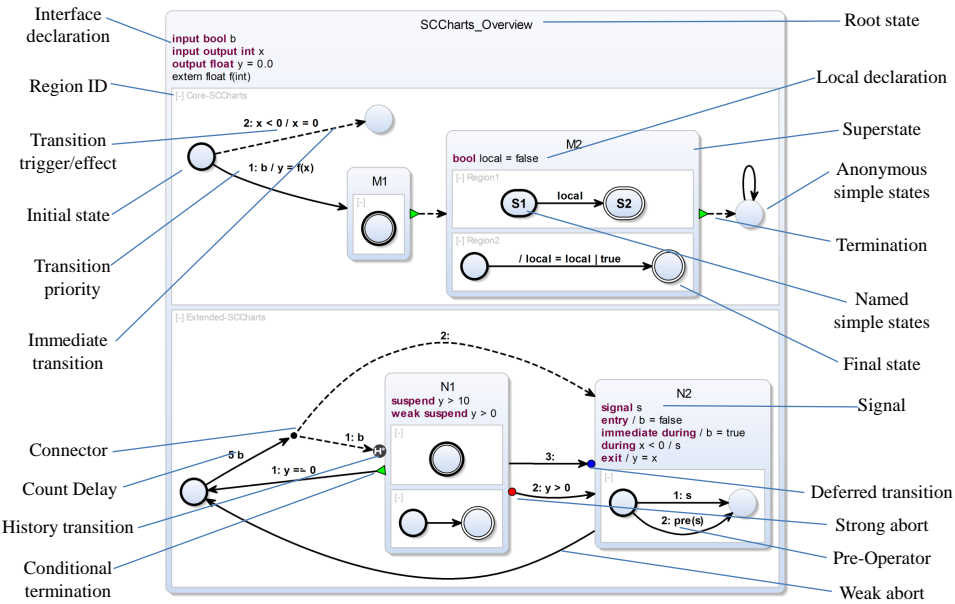


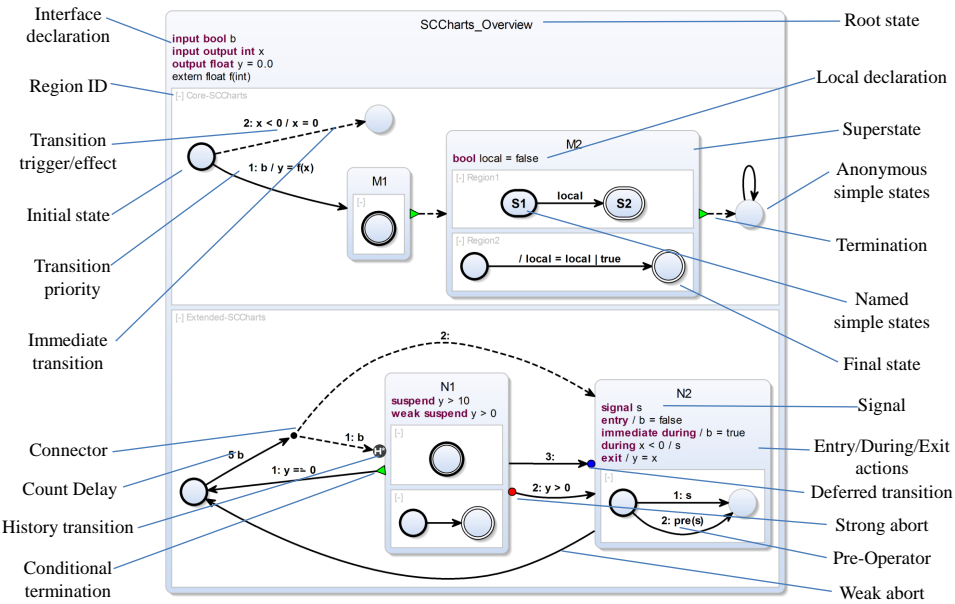


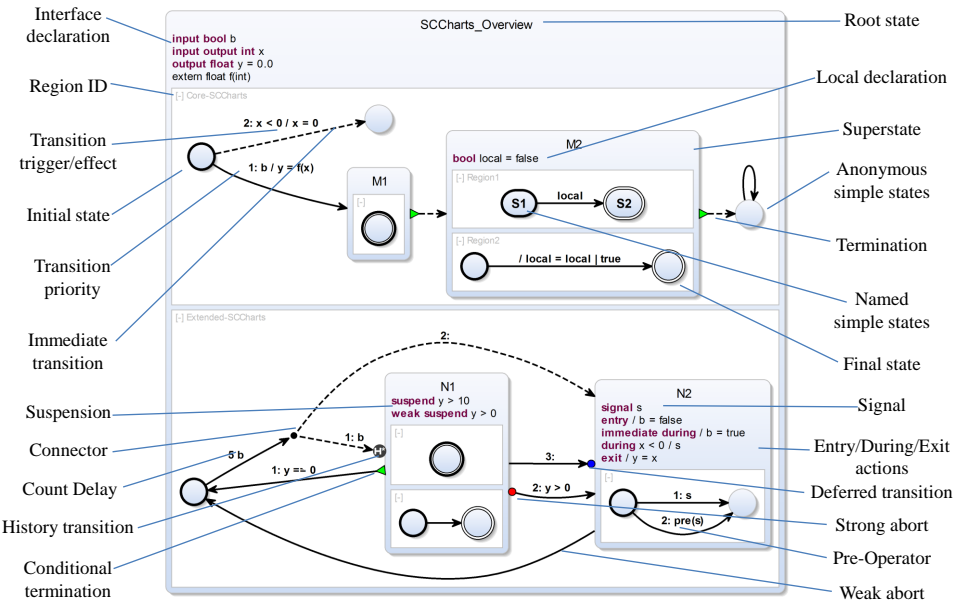




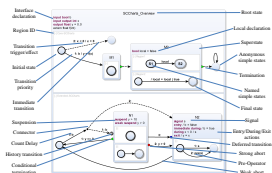






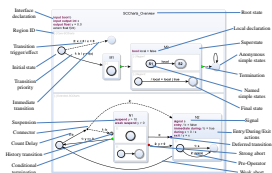


Motivation for Core SCCharts



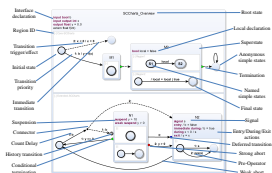
► Observation I

Motivation for Core SCCharts



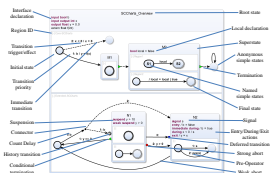
► Observation I: Numerous features

Motivation for Core SCCharts



- ▶ **Observation I: Numerous features**
 - ▶ 😊 Compactness / readability of models

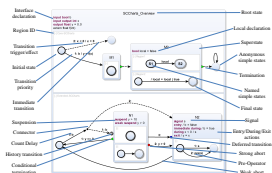
Motivation for Core SCCharts



► Observation I: Numerous features

- 😊 Compactness / readability of models
- 😞 Steeper learning curve

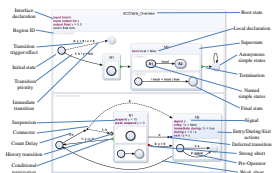
Motivation for Core SCCharts



► Observation I: Numerous features

- 😊 Compactness / readability of models
- 😞 Steeper learning curve
- 😞 Direct compilation & verification more complex

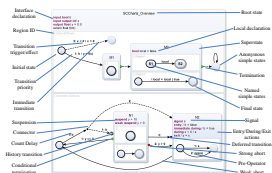
Motivation for Core SCCharts



- ▶ **Observation I: Numerous features**
 - ▶ 😊 Compactness / readability of models
 - ▶ 😞 Steeper learning curve
 - ▶ 😞 Direct compilation & verification more complex

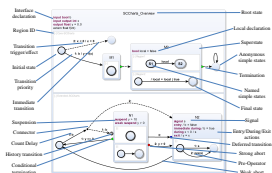
- ▶ **Observation II**

Motivation for Core SCCharts



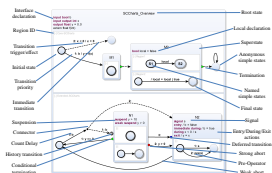
- ▶ **Observation I:** Numerous features
 - ▶ ☺ Compactness / readability of models
 - ▶ ☹ Steeper learning curve
 - ▶ ☹ Direct compilation & verification more complex
- ▶ **Observation II:** Various features can be expressed by other ones

Motivation for Core SCCharts



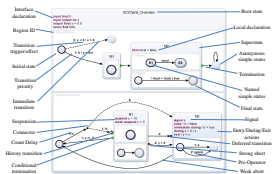
- ▶ **Observation I:** Numerous features
 - ▶ ☺ Compactness / readability of models
 - ▶ ☹ Steeper learning curve
 - ▶ ☹ Direct compilation & verification more complex
- ▶ **Observation II:** Various features can be expressed by other ones
- ▶ **Consequence**

Motivation for Core SCCharts



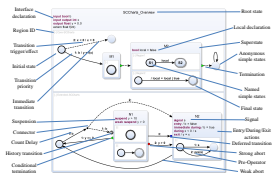
- ▶ **Observation I:** Numerous features
 - ▶ ☺ Compactness / readability of models
 - ▶ ☹ Steeper learning curve
 - ▶ ☹ Direct compilation & verification more complex
- ▶ **Observation II:** Various features can be expressed by other ones
- ▶ **Consequence:** ⇒ Define extended features by means of base features

Motivation (Cont'd)



► Advantages

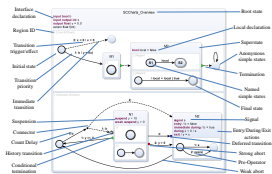
Motivation (Cont'd)



► **Advantages:**

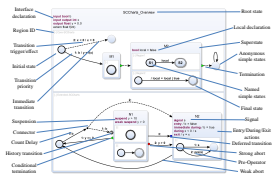
- Minimal base language (Core SCCharts)
+ advanced features (Extended SCCharts)

Motivation (Cont'd)



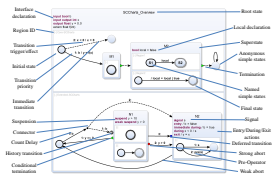
- ▶ **Advantages:**
 - ▶ Minimal base language (Core SCCharts)
 - + advanced features (Extended SCCharts)
 - ▶ Similar to Esterel Kernel Statements & Statement Expansion

Motivation (Cont'd)



- ▶ **Advantages:**
 - ▶ Minimal base language (Core SCCharts)
 - + advanced features (Extended SCCharts)
 - ▶ Similar to Esterel Kernel Statements & Statement Expansion
 - ▶ Advanced features are *syntactic sugar*

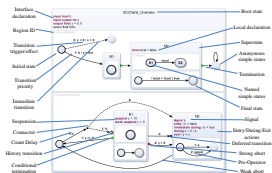
Motivation (Cont'd)



► Advantages:

- Minimal base language (Core SCCharts)
 - + advanced features (Extended SCCharts)
 - Similar to Esterel Kernel Statements & Statement Expansion
- Advanced features are *syntactic sugar*
- Extensible

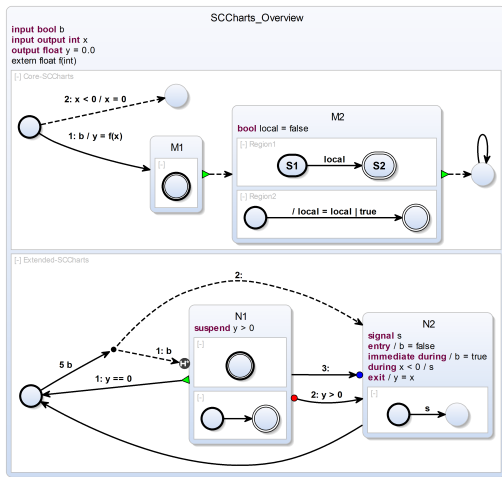
Motivation (Cont'd)



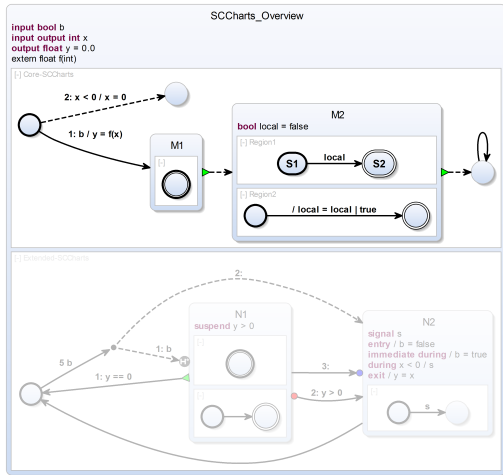
► **Advantages:**

- Minimal base language (Core SCCharts)
 - + advanced features (Extended SCCharts)
 - Similar to Esterel Kernel Statements & Statement Expansion
- Advanced features are *syntactic sugar*
- Extensible
- Compilation (ongoing research)
 - Modular & extensible
 - Less complex
 - Possibly less efficient

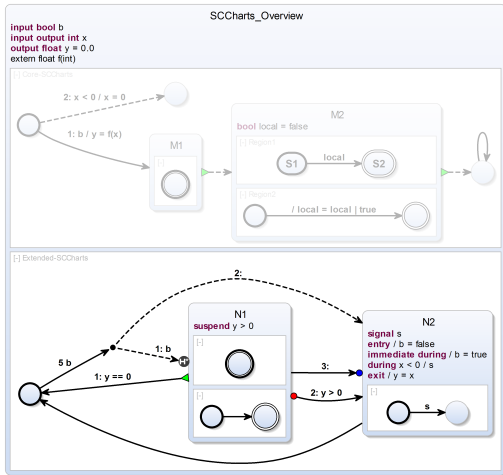
SCCharts — Core & Extended Features



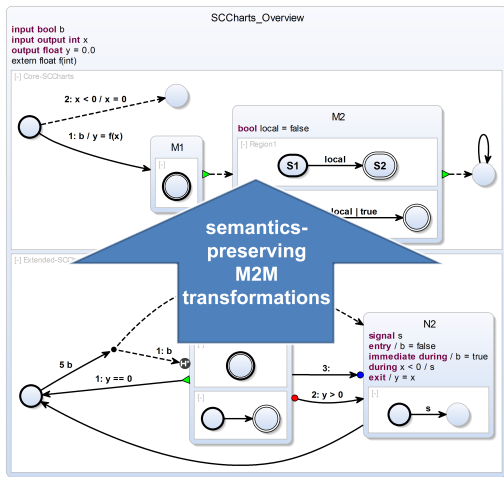
SCCharts — Core & Extended Features



SCCharts — Core & Extended Features



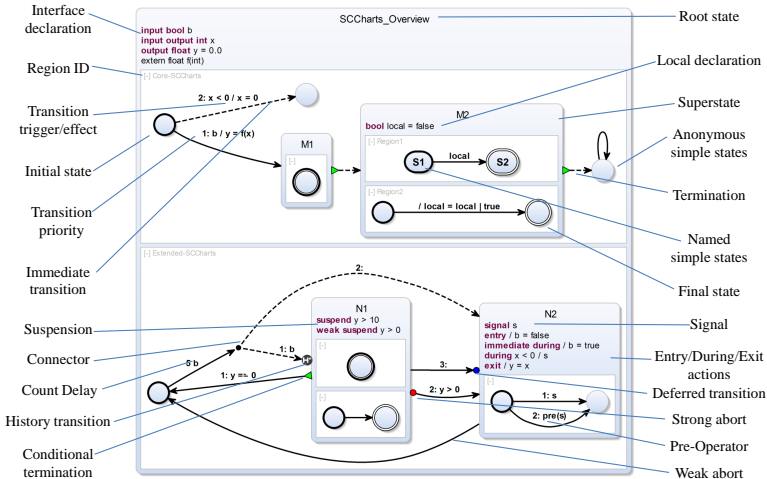
SCCharts — Core & Extended Features



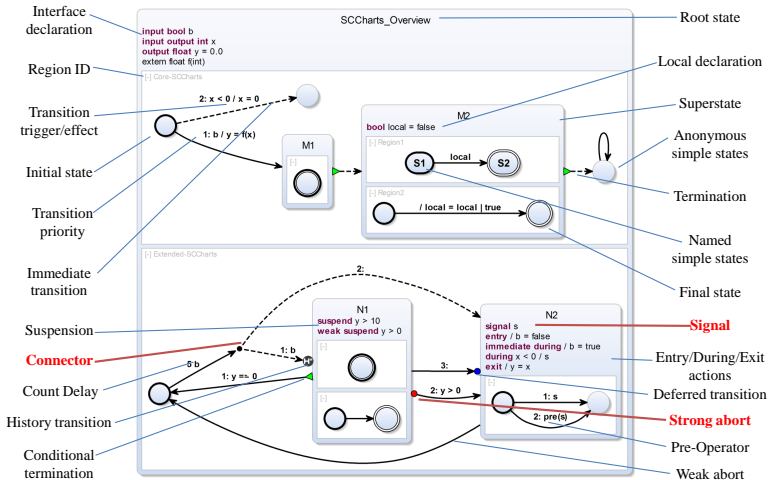
Overview

- ▶ SCCharts Overview
- ▶ Extended SCCharts → Core SCCharts
- ▶ Normalizing Core SCCharts
- ▶ Implementation in KIELER

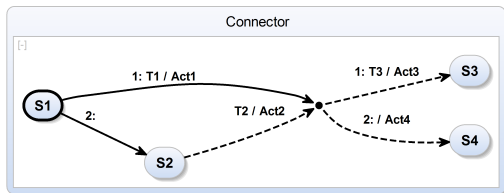
SCCharts — Core Transformations Examples



SCCharts — Core Transformations Examples

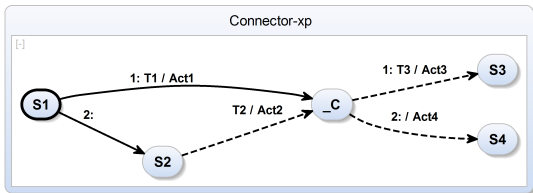
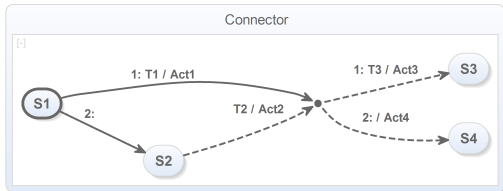


Transforming Connectors

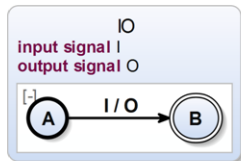


Extended SCCharts with Connectors

Transforming Connectors

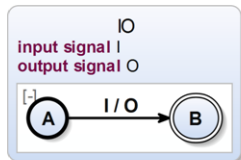


Transforming Signals

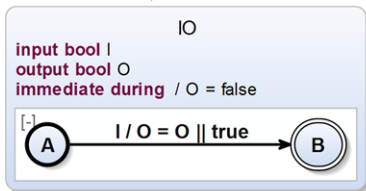
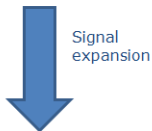


Extended SCCharts with Signals

Transforming Signals

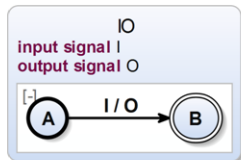


Extended SCCharts with Signals

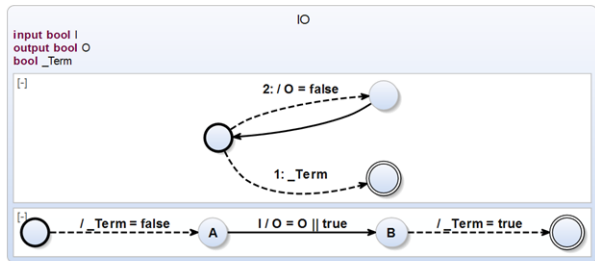


Core SCCharts with During Actions

Transforming Signals

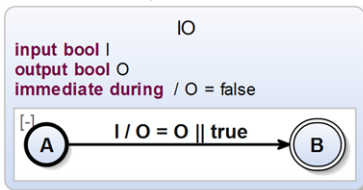


Extended SCCharts with Signals



Core SCCharts only

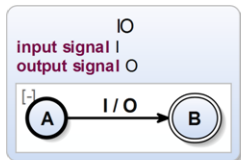
Signal
expansion



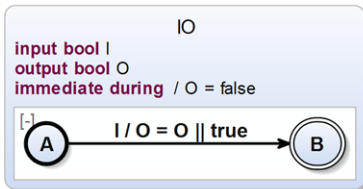
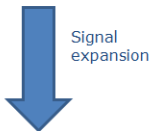
Core SCCharts with During Actions

Action
expansion

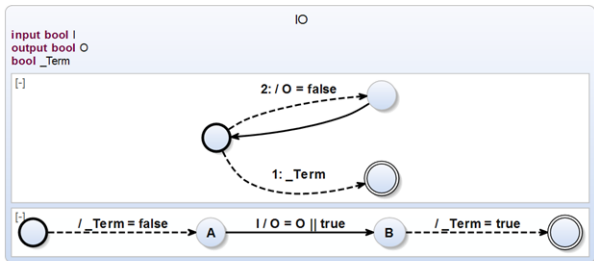
Transforming Signals



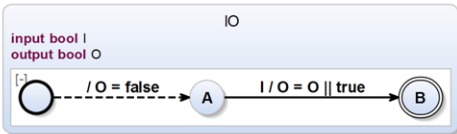
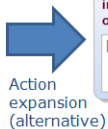
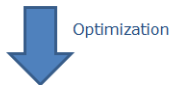
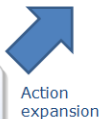
Extended SCCharts with Signals



Core SCCharts with During Actions

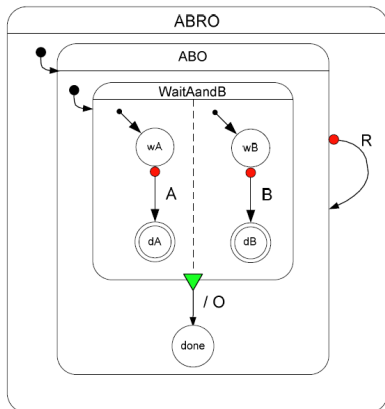


Core SCCharts only



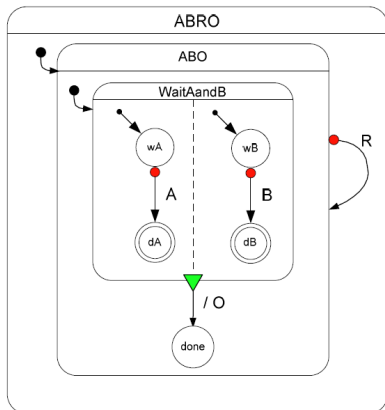
Core SCCharts only (optimized)

SyncChart and SCChart ABRO

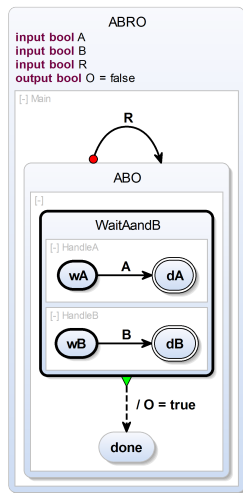


[Charles André, Semantics of SyncCharts, 2003]

SyncChart and SCChart ABRO

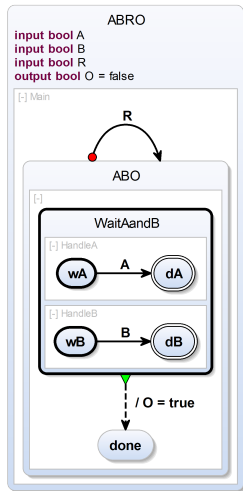


[Charles André, Semantics of SyncCharts, 2003]



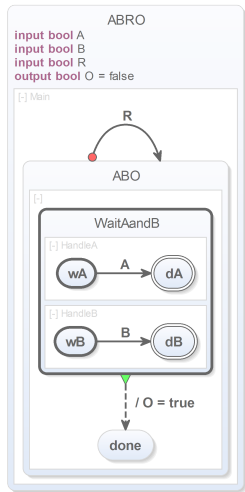
ABRO SCChart

ABRO — Transforming Strong Aborts

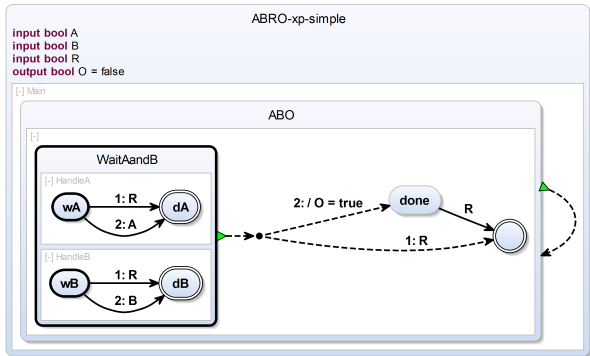


ABRO SCChart with Strong Abort

ABRO — Transforming Strong Aborts

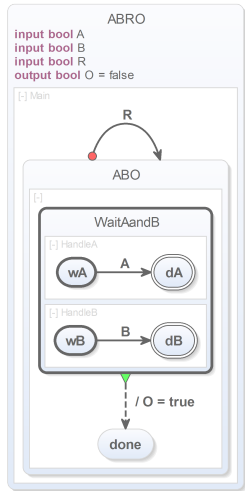


ABRO SCChart with Strong Abort

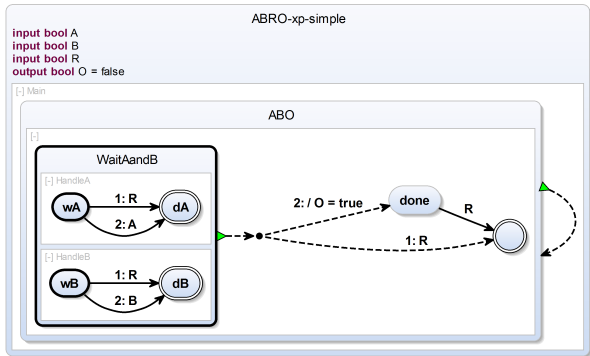


Core SCChart without Strong Abort

ABRO — Transforming Strong Aborts



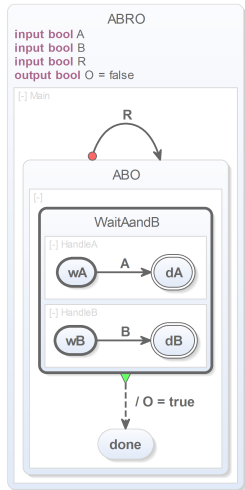
ABRO SCChart with Strong Abort



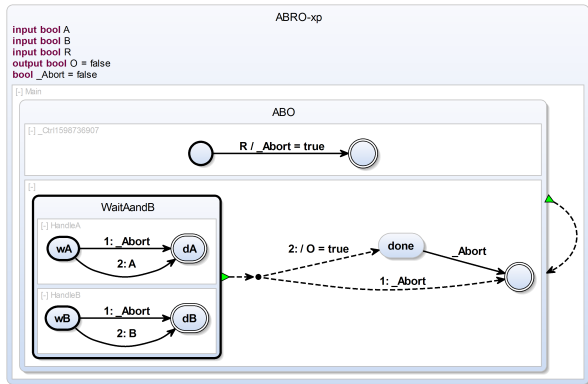
Core SCChart without Strong Abort

→ **Write-Things-Once (WTO) principle violated**

ABRO — Transforming Strong Aborts (cont'd)

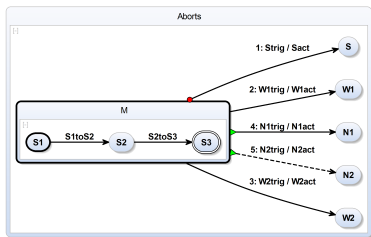


ABRO SCChart with Strong Abort



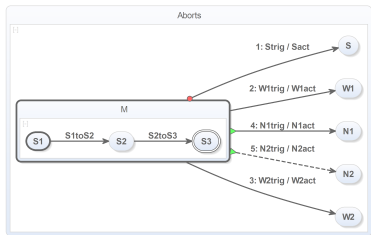
Core SCChart without Strong Abort and WTO

Transforming General Aborts

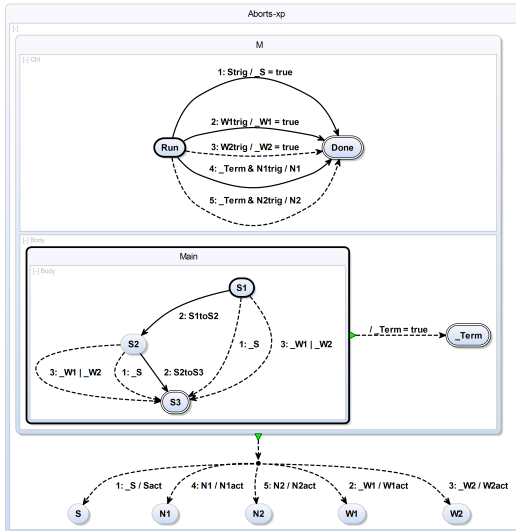


Extended SCCharts with Aborts

Transforming General Aborts



Extended SCCharts with Aborts

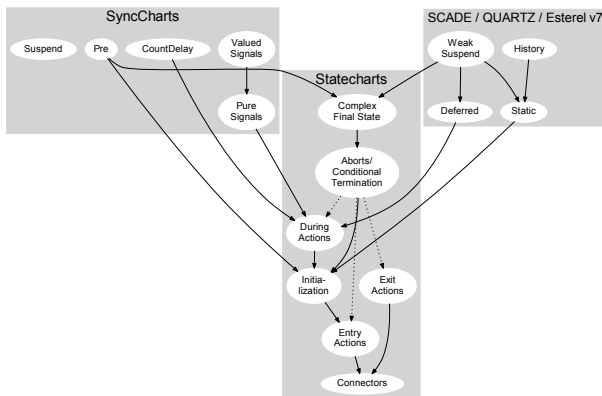


Core SCCharts with one Termination

Overview

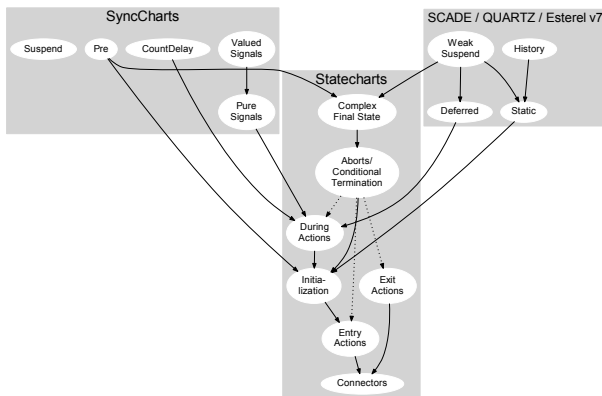
- ▶ SCCharts Overview
- ▶ Extended SCCharts → Core SCCharts
- ▶ Normalizing Core SCCharts
- ▶ Implementation in KIELER

Single-Pass Language-Driven Incremental Compilation (SLIC)



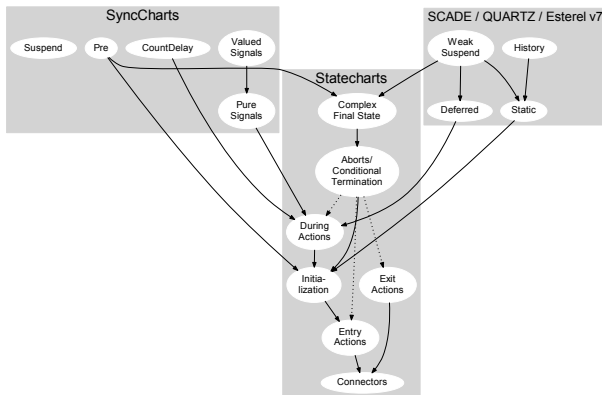
- ▶ Some core transformations will produce (use) some other extended features (solid lines)

Single-Pass Language-Driven Incremental Compilation (SLIC)



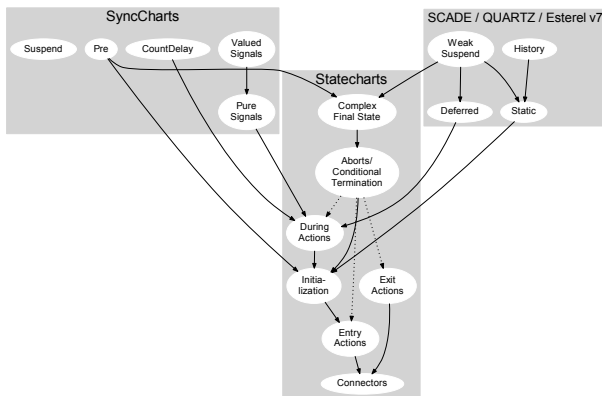
- ▶ Some core transformations will produce (use) some other extended features (solid lines)
- ▶ Other core transformations cannot handle some extended features (dashed lines)

Single-Pass Language-Driven Incremental Compilation (SLIC)



- ▶ Some core transformations will produce (use) some other extended features (solid lines)
- ▶ Other core transformations cannot handle some extended features (dashed lines)
- ▶ → Order in which core transformations are applied is important

Single-Pass Language-Driven Incremental Compilation (SLIC)



- ▶ Some core transformations will produce (use) some other extended features (solid lines)
- ▶ Other core transformations cannot handle some extended features (dashed lines)
- ▶ → Order in which core transformations are applied is important
- ▶ → Dependencies (do not have any cycle, which would be forbidden)

Normalization

- ▶ Further simplify compilation process for Core SCCharts

Normalization

- ▶ Further simplify compilation process for Core SCCharts
- ▶ Allowed patterns:

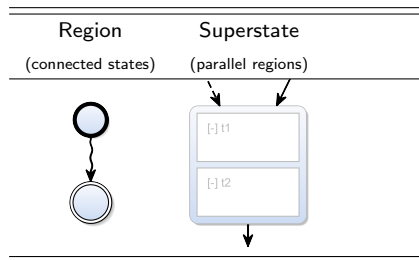
Region

(connected states)



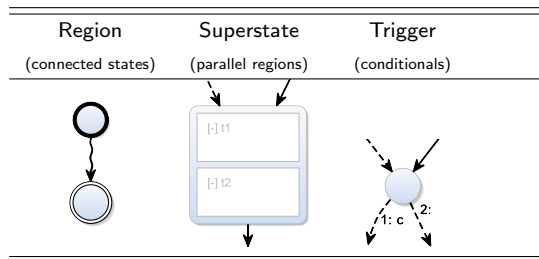
Normalization

- ▶ Further simplify compilation process for Core SCCharts
- ▶ Allowed patterns:



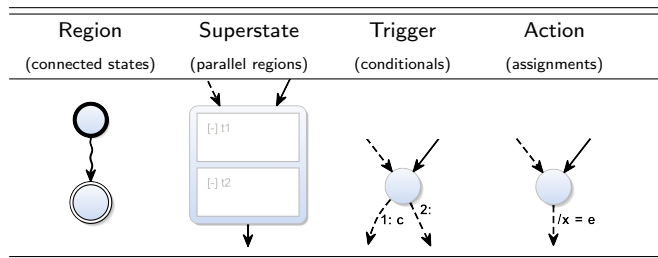
Normalization

- ▶ Further simplify compilation process for Core SCCharts
- ▶ Allowed patterns:



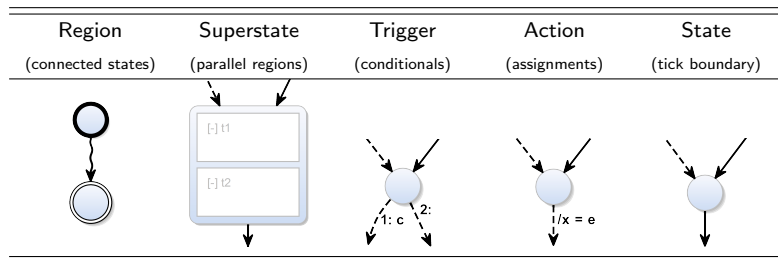
Normalization

- ▶ Further simplify compilation process for Core SCCharts
- ▶ Allowed patterns:

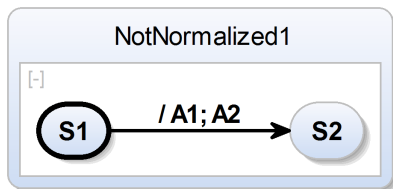


Normalization

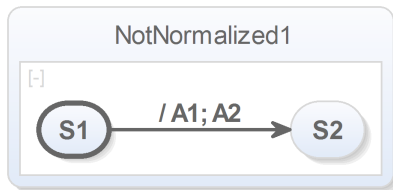
- ▶ Further simplify compilation process for Core SCCharts
- ▶ Allowed patterns:



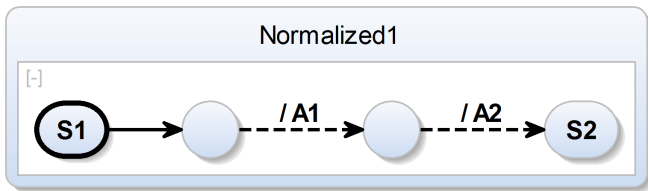
Actions Normalization



Actions Normalization

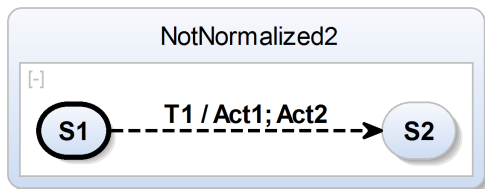


Core SCChart before normalization

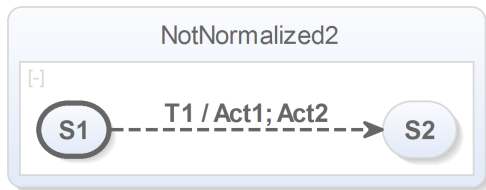


Core SCChart after normalization

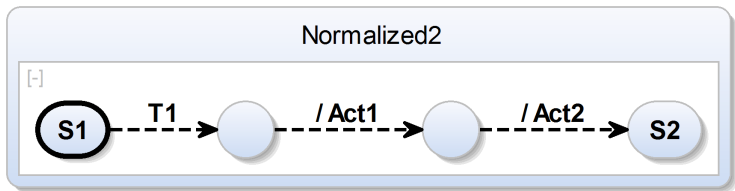
Actions Normalization (cont'd)



Actions Normalization (cont'd)



Core SCChart before normalization

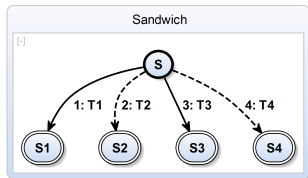


Core SCChart after normalization

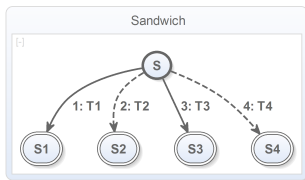
Actions Normalization Implementation Example

```
1  def void transformTriggerActions(Transition transition) {
2      if (((transition.trigger != null || !transition.immediate)
3          && !transition.actions.nullOrEmpty) || transition.actions.size > 1) {
4
5          val targetState = transition.targetState
6          val parentRegion = targetState.parentRegion
7          val transitionOriginalTarget = transition.targetState
8
9          var Transition lastTransition = transition
10
11         for (action : transition.actions.immutableCopy) {
12
13             val actionState = parentRegion.createState(targetState.id + action.id)
14             actionState.setTypeConnector
15
16             val actionTransition = createImmediateTransition.addAction(action)
17             actionTransition.setSourceState(actionState)
18
19             lastTransition.setTargetState(actionState)
20             lastTransition = actionTransition
21         }
22
23         lastTransition.setTargetState(transitionOriginalTarget)
24     }
25 }
```

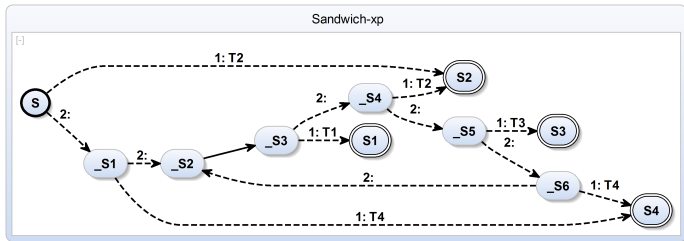
Trigger Normalization



Trigger Normalization

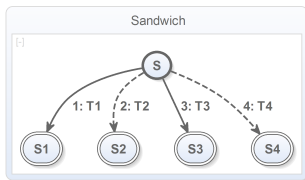


Core SCChart before normalization

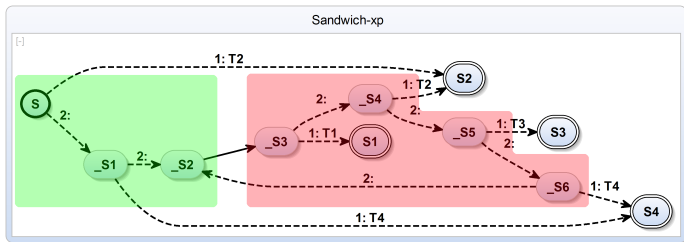


Core SCChart after normalization

Trigger Normalization

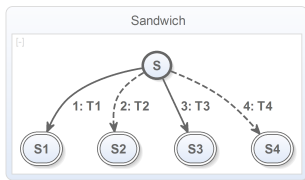


Core SCChart before normalization

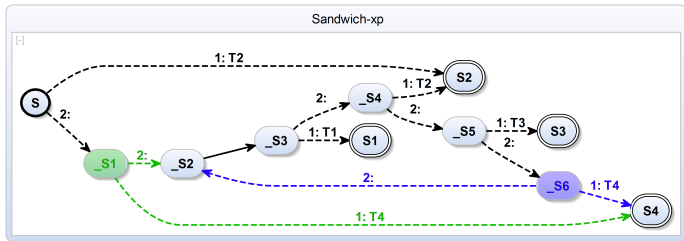


Core SCChart after normalization (Surface & Depth)

Trigger Normalization

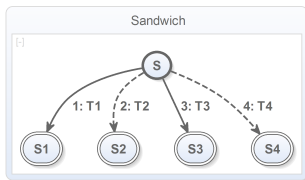


Core SCChart before normalization

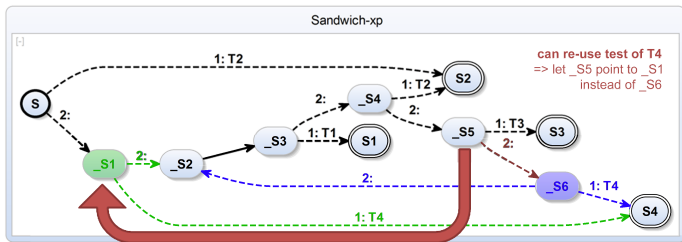


Core SCChart after normalization (potential optimization)

Trigger Normalization

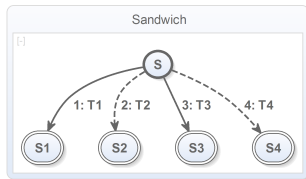


Core SCChart before normalization

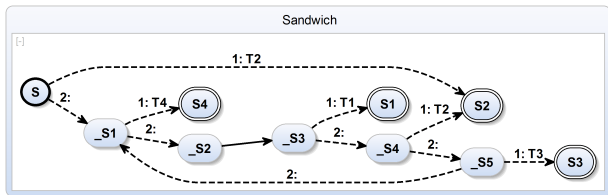


Core SCChart after normalization (potential optimization)

Trigger Normalization (Cont'd)

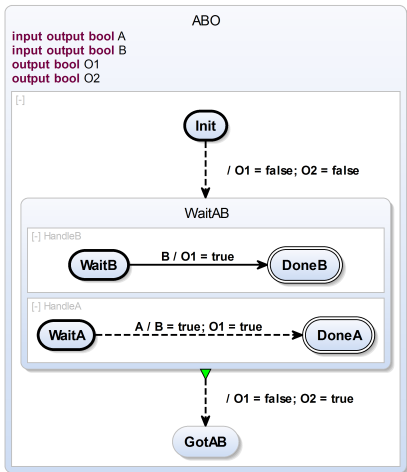


Core SCChart before normalization



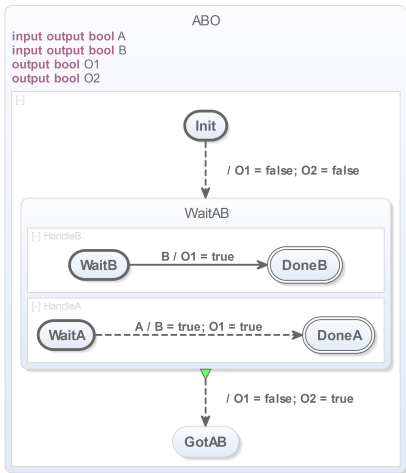
Core SCChart after **optimized** normalization

ABO — Normalization Example (Actions)

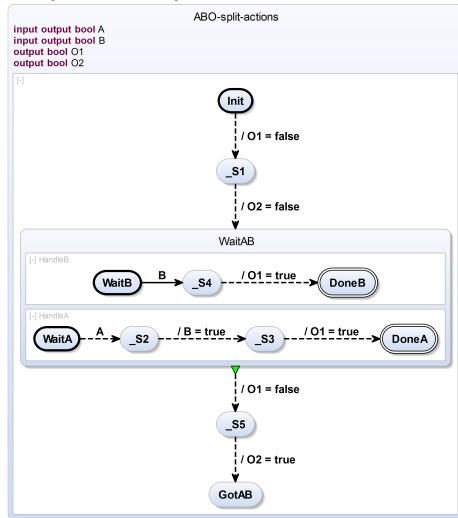


ABO Core SCChart

ABO — Normalization Example (Actions)

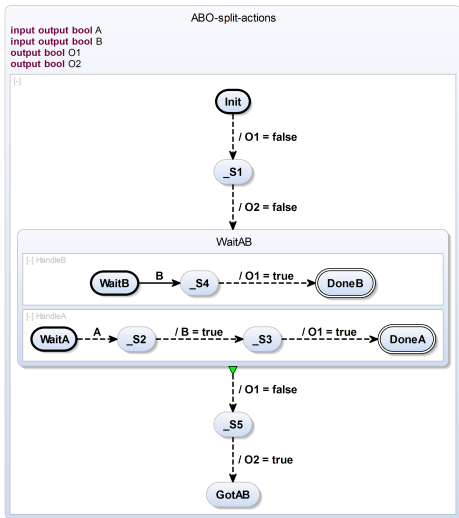


ABO Core SCChart



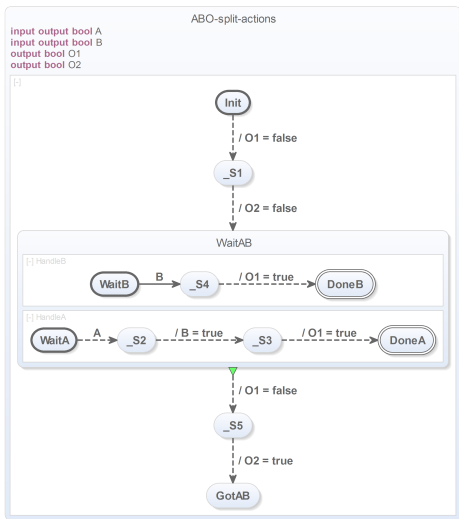
ABO Core SCChart with normalized actions

ABO — Normalization Example (Actions & Trigger)

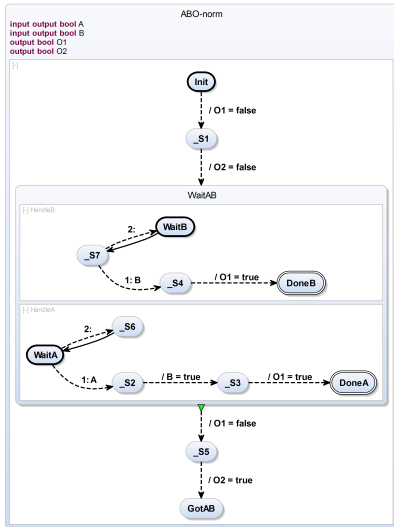


ABO Core SCChart with normalized actions

ABO — Normalization Example (Actions & Trigger)



ABO Core SCChart with normalized actions



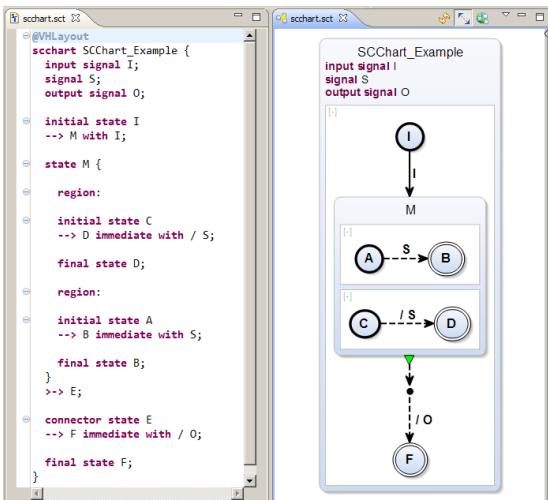
ABO Normalized SCChart

Overview

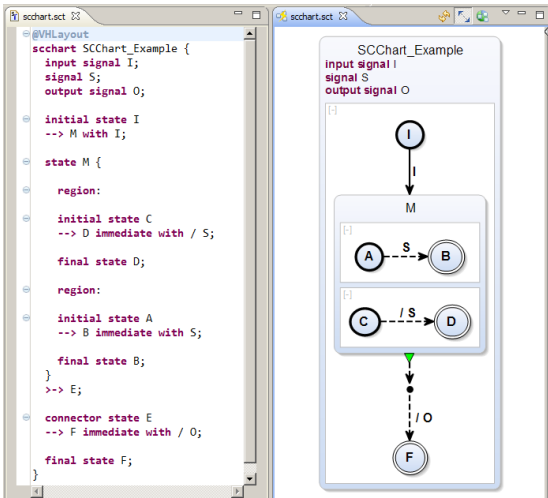
- ▶ SCCharts Overview
- ▶ Extended SCCharts → Core SCCharts
- ▶ Normalizing Core SCCharts
- ▶ Implementation in KIELER

Textual Modeling with KLightD

- ▶ Eclipse based KIELER framework

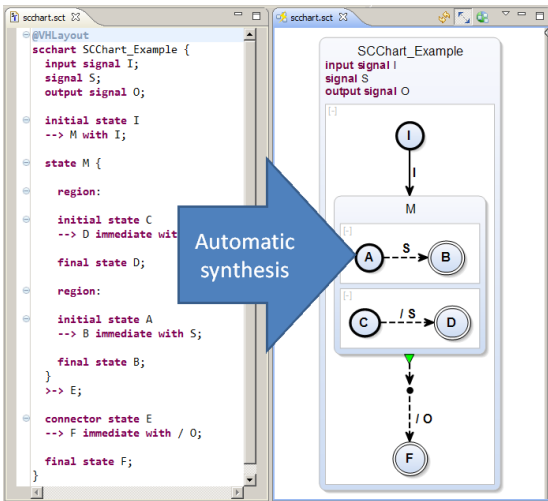


Textual Modeling with KLighD

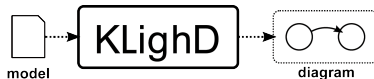


- ▶ Eclipse based KIELER framework
- ▶ Textual modeling based on Xtext
 - ▶ Syntax highlighting
 - ▶ Code completion
 - ▶ Formatter

Textual Modeling with KLightD



- ▶ Eclipse based KIELER framework
- ▶ Textual modeling based on Xtext
 - ▶ Syntax highlighting
 - ▶ Code completion
 - ▶ Formatter
- ▶ Transient view based on KLightD



[C. Schneider et al., VL/HCC'13]

SCCharts Interactive Compilation

Resource - SCCharts/ISOLA14/DEMO/ABRO_ORIGINAL.sct - KIELER

```

@VHLayout
schart ABRO {
  input bool A, B, R;
  output bool O;

  // @direction DOWN
  region Main:
    initial state ABthenO {
      entry / O = false;

      initial state WaitAB {

        region HandleA:

          initial state WA
          --> DA with A;

          final state DA;

        region HandleB:

          initial state WB
          --> DB with B;

          final state DB;
        }
      }
    }
  }
  state Done;
  o-> ABthenO with R;
}
  
```

KIELER Model View [ABRO_ORIGINAL.sct]

ABRO
input bool A, B, R
output bool O

[-] Main

R

ABthenO
entry / O = false

WaitAB

[-] HandleA

WA A DA

[-] HandleB

WB B DB

Done
/ O = true

KIELER Compiler Selection

Progress

Extended SCCharts

SCCharts Expansions

SyncCharts

SCADE / QUARTZ / Esterel v7

Statecharts

Const

Initialization

Entry

Abort

Exit

Connector

During

Complex Final State

Core SCCharts

SCGraph

Code Generation

Diagram Options

- Declarations
- State actions
- Transition labels
- Dependencies & optimized priorities
- Dependencies & priorities
- Reference Expansion
- Adaptive Zoom
- Shadow
- Black/White (Paper)

Layout Options

Directions:

- Down
- Left
- Right
- Up

Spacing:

.....

SCCharts Interactive Compilation

The screenshot displays the KIELER Modeling IDE with the following components:

- Textual Modeling:** A code editor showing the SCChart definition for `ABRO`. A purple box highlights the text "Textual Modeling".


```

@VHLayout
schart ABRO {
  input bool A, B, R;
  output bool O;

  // @direction DOWN
  region Main:
    initial state ABthenO {
      entry / O = false;

      initial state lB
      --> DB with B;

      final state DB;
      --> Done with / O = true;

      state Done;
    }
    o-> ABthenO with R;
  }

```
- Statechart Diagram:** A graphical representation of the SCChart. It shows an entry state `ABthenO` with the condition `/ O = false`. This state leads to a `WaitAB` region containing two parallel handshakes:
 - `HandeA`: `VA` (wait) → `DA` (done)
 - `HandeB`: `WB` (wait) → `DB` (done)
 A transition labeled `R` returns from `ABthenO` to itself. A dashed arrow labeled `/ O = true` leads from `ABthenO` to a `Done` state.
- Extended SCCharts:** A flow diagram showing the compilation pipeline:
 - `SCCharts Expansions` (containing `SyncCharts` and `SCADE / QUARTZ / Esterel v7`) feeds into `Statecharts`.
 - `Statecharts` includes components like `Const`, `Initialization`, `Entry`, `Exit`, `Connector`, `Abort`, `During`, and `Complex Final State`.
 - The pipeline continues to `Core SCCharts`, `SCGraph`, and finally `Code Generation`.

On the right side, there are panels for **Diagram Options** (with checkboxes for Declarations, State actions, Transition labels, etc.) and **Layout Options** (with radio buttons for Direction and Spacing).

SCCharts Interactive Compilation

The screenshot displays the KIELER Modeling IDE with the following components:

- Textual Modeling (Left Pane):** Shows the source SCChart code for `ABRO`. A purple box highlights the text "Textual Modeling".


```

@VHLayout
schart ABRO {
  input bool A, B, R;
  output bool O;

  // @direction DOWN
  region Main:
    initial state ABthenO {
      entry / O = false;

      initial state lB
      --> DB with B;

      final state DB;
    }
    --> Done with / O = true;

    state Done;
  }
  o-> ABthenO with R;
}
      
```
- Modeled Diagram (Middle Pane):** Shows a graphical representation of the SCChart. A yellow box highlights the text "Modeled Diagram". The diagram includes:
 - ABRO**: The main chart with inputs A, B, R and output O.
 - ABthenO**: An inner chart with entry condition `/ O = false`.
 - WaitAB**: A sub-chart containing two handshakes:
 - HandeA**: `VA` (wait) → `DA` (done) on signal `A`.
 - HandeB**: `WB` (wait) → `DB` (done) on signal `B`.
 - Done**: A final state reached from `WaitAB` with the condition `/ O = true`.
 - Transitions**: A self-loop on `ABthenO` labeled `R`, and a transition from `ABthenO` to `Done` labeled `R`.
- Extended SCCharts (Bottom Pane):** Shows a flow diagram of the compilation process:
 - SCCharts Expansions** (containing `SyncCharts` and `SCADE / QUARTZ / Esterel v7`) feeds into **Statecharts**.
 - Statecharts** (containing `Const`, `Initialization`, `Entry`, `Exit`, `Connector`, `Abort`, `Complex Final State`, and `During`) feeds into **Core SCCharts**.
 - Core SCCharts** feeds into **SCGraph**, which finally feeds into **Code Generation**.
- Diagram Options (Right Pane):** A list of checkboxes for customizing the diagram, including:
 - Declarations
 - State actions
 - Transition labels
 - Dependencies & optimized priorities
 - Dependencies & priorities
 - Reference Expansion
 - Adaptive Zoom
 - Shadow
 - Black/White (Paper)
- Layout Options (Right Pane):** Controls for diagram orientation and spacing:
 - Direction:** Down, Left, Right, Up
 - Spacing:** A slider control.

SCCharts Interactive Compilation

Textual Modeling

```

@VHLayout
schart ABRO {
  input bool A, B, R;
  output bool O;

  // @direction DOWN
  region Main:
    initial state ABthenO {
      entry / O = false;

      initial state WB
      --> DB with B;

      final state DB;
      --> Done with / O = true;

      state Done;
    }
    o-> ABthenO with R;
  }
}
  
```

Modeled Diagram

The diagram illustrates the state transitions for the 'ABRO' chart. It starts with an initial state 'ABthenO' where 'O = false'. A transition labeled 'R' leads to a state 'WaitAB'. Inside 'WaitAB', there are two parallel regions: 'HandeA' with states 'WA' and 'DA' connected by transition 'A', and 'HandeB' with states 'WB' and 'DB' connected by transition 'B'. A transition labeled 'O = true' leads from 'WaitAB' to the final state 'Done'.

Extended SCCharts

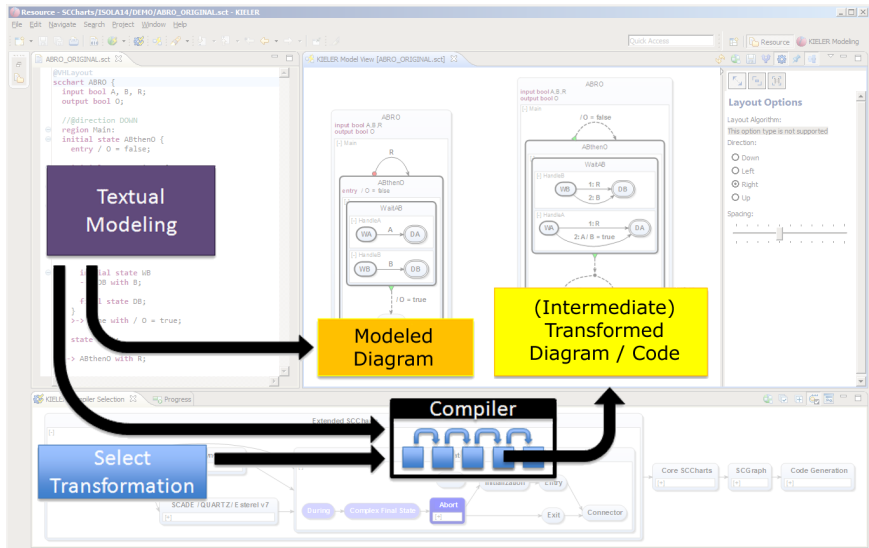
The flowchart shows the compilation pipeline: 'Extended SCCharts' (containing 'Statecharts' and 'During' blocks) leads to 'Core SCCharts', which then leads to 'SCGraph' and finally 'Code Generation'. A 'Select Transformation' block is shown as an input to the 'Extended SCCharts' stage.

SCCharts Interactive Compilation

The screenshot displays the SCCharts IDE interface, illustrating the interactive compilation process. The main window is divided into several panes:

- Textual Modeling:** The left pane shows the source code for the `ABRO` statechart. A purple box highlights the initial state `ABthenO` and its entry condition `/ O = false`. A yellow box highlights the final state `DB` and the completion condition `/ O = true`.
- Modeled Diagram:** The middle-left pane shows a graphical representation of the statechart. A yellow box highlights the `ABthenO` state and its entry condition `/ O = false`.
- (Intermediate) Transformed Diagram / Code:** The middle-right pane shows a more detailed diagram of the statechart, including the `WaitAB` state and its transitions. A yellow box highlights the `ABthenO` state and its entry condition `/ O = false`.
- Layout Options:** The right pane shows the `Layout Options` dialog, which allows users to specify the layout algorithm (Down, Left, Right, Up) and the spacing between elements.
- Extended SCCharts:** The bottom pane shows the `Extended SCCharts` configuration window. A blue box highlights the `Select Transformation` button. The `Statecharts` section shows a flow from `Const` to `Initialization` to `Entry` to `Exit` to `Connector`. The `Abort` button is highlighted in blue.

SCCharts Interactive Compilation



Conclusions

- ▶ SyncCharts **are** a great choice for specifying deterministic control-flow behavior

Conclusions

- ▶ SyncCharts **are** a great choice for specifying deterministic control-flow behavior...
- ▶ ... but do not accept sequentiality
If (!done) { ... ; done = true;}

Conclusions

- ▶ SyncCharts **are** a great choice for specifying deterministic control-flow behavior...
- ▶ ... but do not accept sequentiality
If (!done) { ... ; done = true; }
- ▶ **SCCharts** extend SyncCharts w.r.t. semantics
→ Sequentially Constructive MoC

Conclusions

- ▶ SyncCharts **are** a great choice for specifying deterministic control-flow behavior...
- ▶ ... but do not accept sequentiality
If (!done) { ... ; done = true;}
- ▶ **SCCharts** extend SyncCharts w.r.t. semantics
→ Sequentially Constructive MoC
 - ▶ All valid SyncCharts interpreted as SCCharts **keep** their meaning

Conclusions

- ▶ SyncCharts **are** a great choice for specifying deterministic control-flow behavior...
- ▶ ... but do not accept sequentiality
If (!done) { ... ; done = true; }
- ▶ **SCCharts** extend SyncCharts w.r.t. semantics
→ Sequentially Constructive MoC
 - ▶ All valid SyncCharts interpreted as SCCharts **keep their meaning**
- ▶ **Core** SCCharts: Few basic features for simpler & more robust compilation
- ▶ **Extended** SCCharts: Syntactic sugar, readability, extensible
- ▶ **Normalized** SCCharts: Further ease compilation

Conclusions

- ▶ SyncCharts **are** a great choice for specifying deterministic control-flow behavior...
- ▶ ... but do not accept sequentiality
If (!done) { ... ; done = true; }
- ▶ **SCCharts** extend SyncCharts w.r.t. semantics
→ Sequentially Constructive MoC
 - ▶ All valid SyncCharts interpreted as SCCharts **keep their meaning**
- ▶ **Core** SCCharts: Few basic features for simpler & more robust compilation
- ▶ **Extended** SCCharts: Syntactic sugar, readability, extensible
- ▶ **Normalized** SCCharts: Further ease compilation
→ Details in the next lecture :-)

To Go Further

- ▶ R. von Hanxleden, B. Duderstadt, C. Motika, S. Smyth, M. Mendler, J. Aguado, S. Mercer, and O. O'Brien. *SCCharts: Sequentially Constructive Statecharts for Safety-Critical Applications*. Proc. ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI'14), Edinburgh, UK, June 2014. <https://rtsys.informatik.uni-kiel.de/~biblio/downloads/papers/pldi14.pdf>
- ▶ C. Motika, S. Smyth and R. von Hanxleden, *Compiling SCCharts—A Case-Study on Interactive Model-Based Compilation*, Proc. 6th International Symposium on Leveraging Applications of Formal Methods, Verification and Validation (ISoLA 2014), Corfu, Greece, LNCS 8802, pp. 443–462
<https://rtsys.informatik.uni-kiel.de/~biblio/downloads/papers/isola14.pdf>