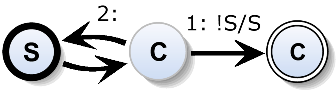
SCCharts

Sequentially Constructive Charts

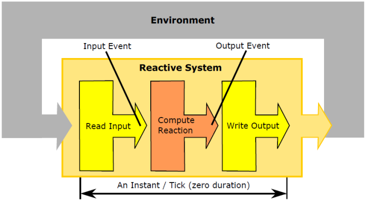
Reinhard von Hanxleden, Björn Duderstadt, Christian Motika,
Steven Smyth, Michael Mendler, Joaquin Aguado, Stephen Mercer, and
Owen O'Brien

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



SYNCHRON'13
Dagstuhl, 19 Nov. 2013

Reactive Embedded Systems



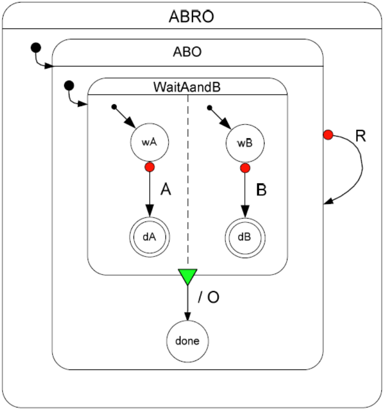
SEUS 2013

```
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

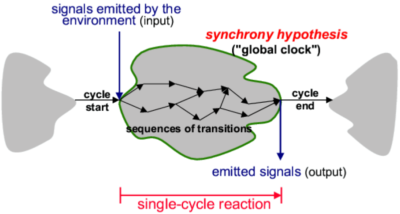
- Embedded systems react to inputs with computed outputs
- Typically state based computations
- Computations often exploit concurrency → Threads
- Threads are problematic → **Synchronous languages**: Lustre, Esterel, SCADE, SyncCharts

SyncCharts



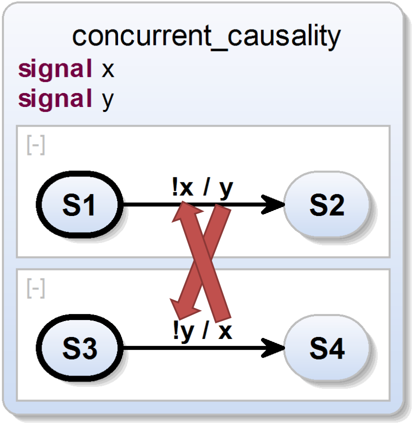
Charles André, Semantics of SyncCharts, 2003

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

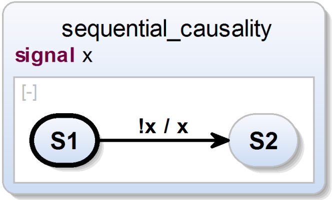


[Gerald Lüttgen, 2001]

Causality in SyncCharts



Causality in SyncCharts (cont'd)



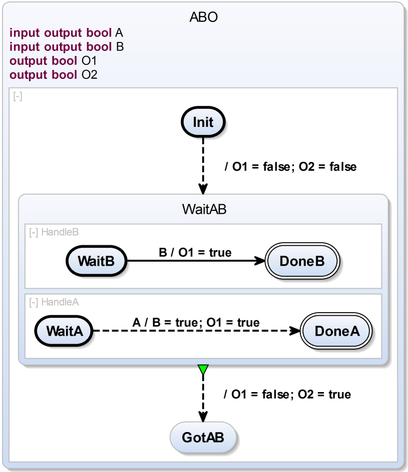

```
if (!done) {
  ...
  done = true;
}
```

- 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)**

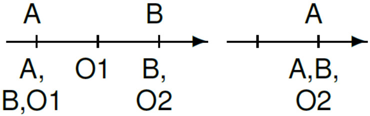
Overview

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

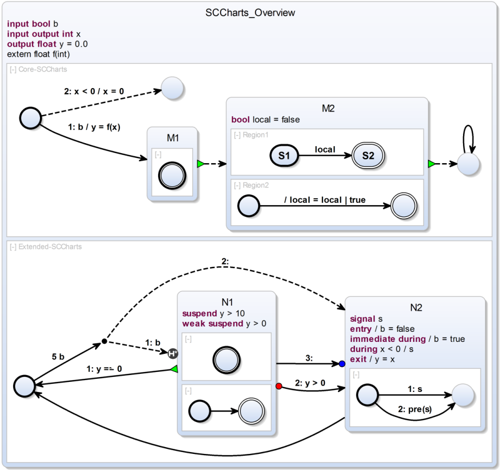
SCCharts Overview

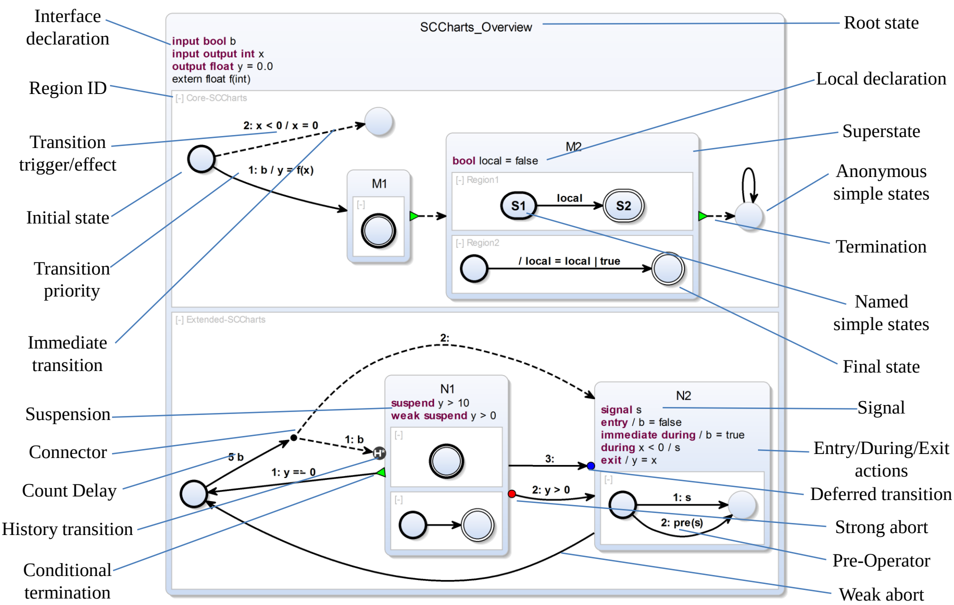


- SCCharts $\widehat{=}$ SyncCharts syntax + Seqentially Constructive semantics
- *Hello World* of Sequential Constructiveness: **ABO**
  - Variables instead of signals
  - Behavior (briefly)
    1. Initialize
    2. Concurrently wait for inputs *A* or *B* to become *true*
    3. Once *A* and *B* are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



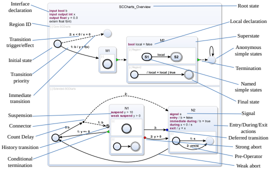
SCCharts - Features





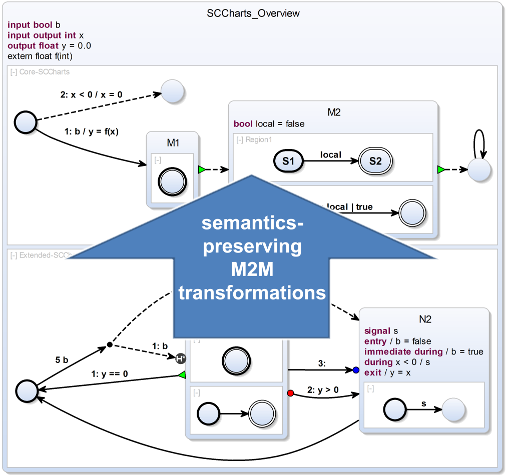
Interface declaration
Region ID
Transition trigger/effect
Initial state
Transition priority
Immediate transition
Suspension
Connector
Count Delay
History transition
Conditional termination
Root state
Local declaration
Superstate
Anonymous simple states
Termination
Named simple states
Final state
Signal
Entry/During/Exit actions
Deferred transition
Strong abort
Pre-Operator
Weak abort
SCCharts_Overview
input bool b
input output int x
output float y = 0.0
extern float f(int)
[-] Core-SCCharts
2: x < 0 / x = 0
1: b / y = f(x)
M1
[-]
M2
bool local = false
[-] Region1
S1
local
S2
[-] Region2
/ local = local | true
[-] Extended-SCCharts
2:
N1
suspend y > 10
weak suspend y > 0
[-]
[-]
N2
signal s
entry / b = false
immediate during / b = true
during x < 0 / s
exit / y = x
[-]
1: b
5 b
1: y == 0
3:
2: y > 0
1: s
2: pre(s)

Motivation



- Numerous features
  - ☺ Readability of models
  - ☹ Compilation & verification more complex
  - ☺ Various features can be expressed by other ones → Syntactic sugar
- ⇒ Minimal base language (Core SCCharts) + advanced features (Extended SCCharts)
  - Define extended features by means of base features
  - Extensible
  - Similar to Esterel Kernel Statements & Statement Expansion

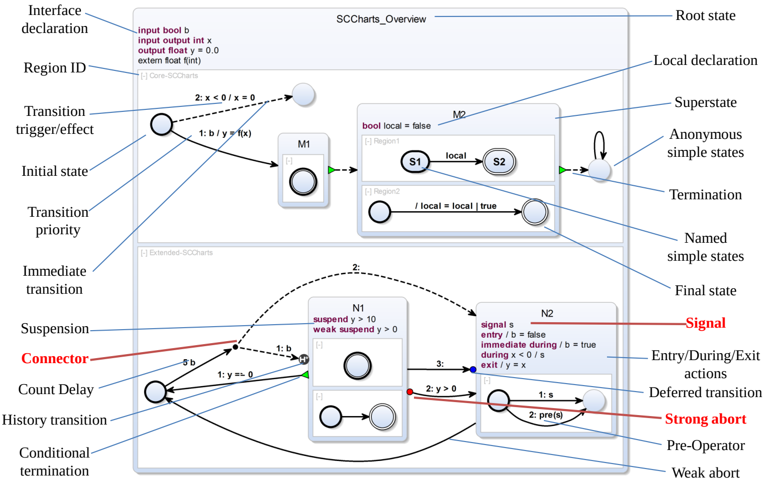
SCCharts - Core & Extended Features



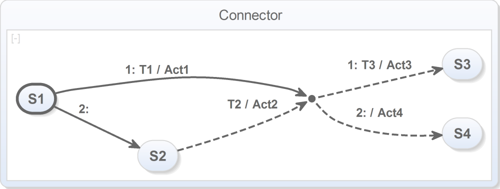
Overview

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

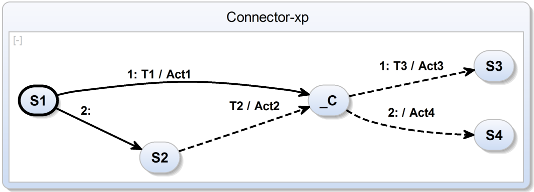
SCCharts - Core Transformations Examples



Transforming Connectors

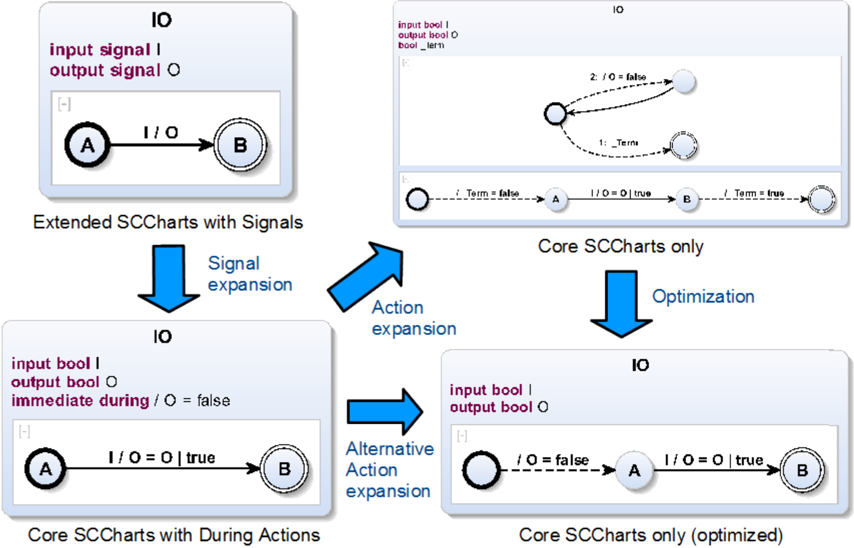


Extended SCCharts with Connectors

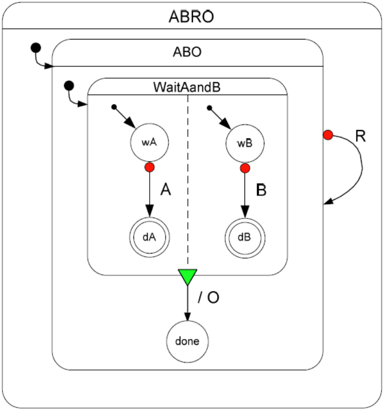


Core SCCharts without Connectors

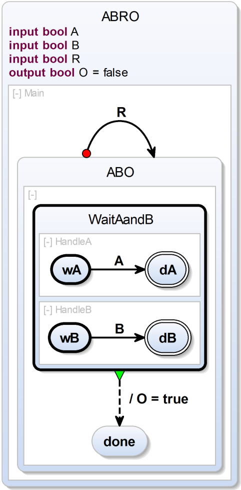
Transforming Signals



SyncChart and SCChart ABRO

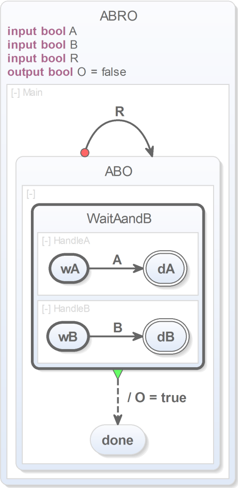


Charles André, Semantics of SyncCharts, 2003

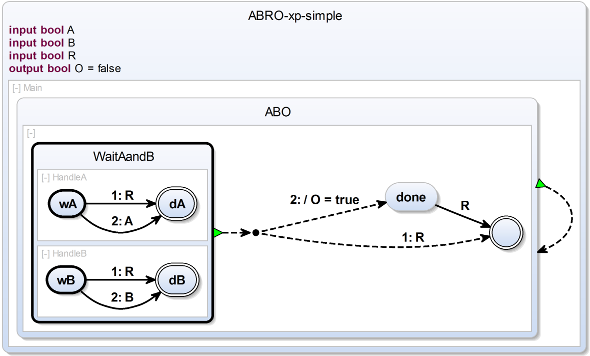


. ABRO SCChart

ABRO - Transforming Strong Aborts



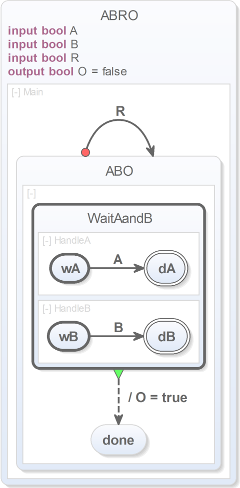
ABRO SCChart with Strong Abort



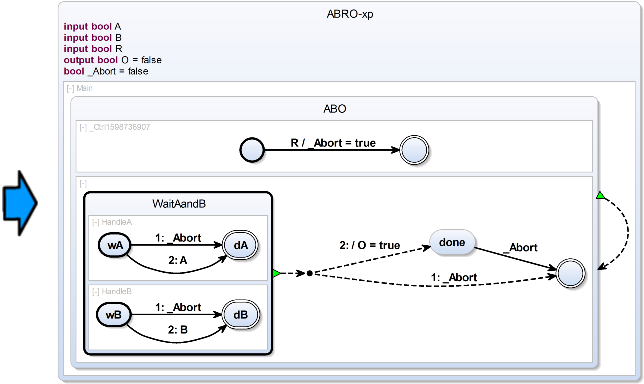
Core SCChart without Strong Abort

→ Write-Things-Once (WTO) principle violated

ABRO - Transforming Strong Aborts (cont'd)

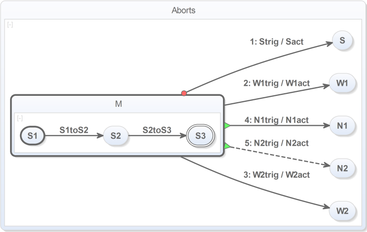


ABRO SCChart with Strong Abort

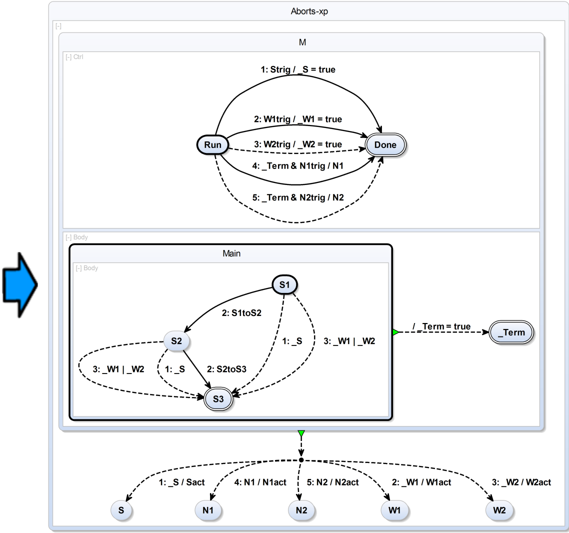


Core SCChart without Strong Abort and WTO

Transforming General Aborts



Extended SCCharts with Aborts



Core SCCharts with one Termination

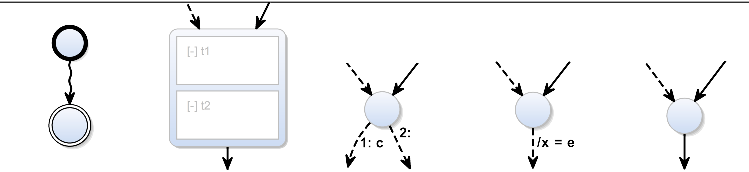
Overview

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

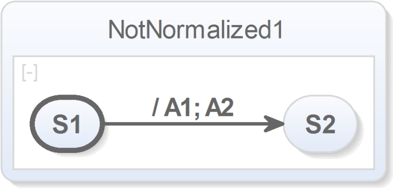
Normalization

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

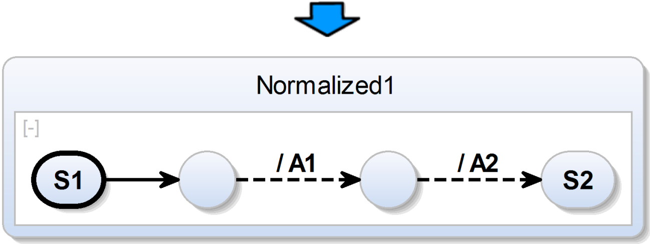
Region	Superstate	Trigger	Action	State
(connected states)	(parallel regions)	(conditionals)	(assignments)	(tick boundary)



Actions Normalization

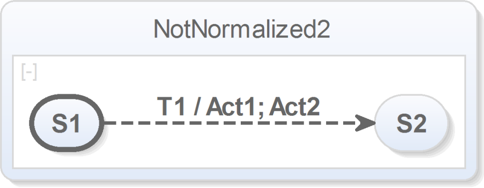


Core SCChart before normalization

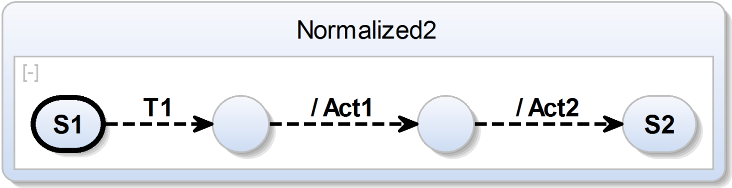


Core SCChart after normalization

Actions Normalization (cont'd)



Core SCChart before normalization

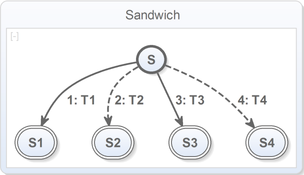


Core SCChart after normalization

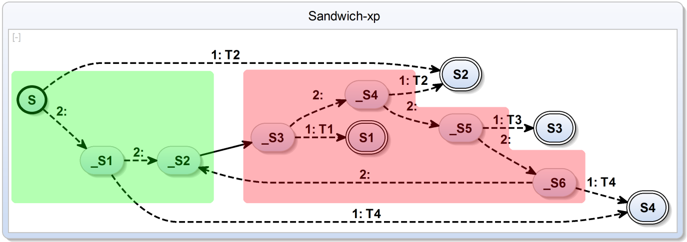
Actions Normalization Implementation Example

```
def void transformTriggerActions(Transition transition) {
   if (((transition.trigger != null || !transition.immediate)
         && !transition.actions.nullOrEmpty) || transition.actions.size > 1) {

      val targetState = transition.targetState
      val parentRegion = targetState.parentRegion
      val transitionOriginalTarget = transition.targetState

      var Transition lastTransition = transition

      for (action : transition.actions.immutableCopy) {

         val actionState = parentRegion.createState(targetState.id + action.id)
         actionState.setTypeConnector

         val actionTransition = createImmediateTransition.addAction(action)
         actionTransition.setSourceState(actionState)

         lastTransition.setTargetState(actionState)
         lastTransition = actionTransition
      }

      lastTransition.setTargetState(transitionOriginalTarget)
   }
}
```

Trigger Normalization

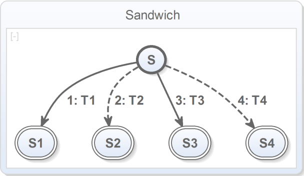


Core SCChart before normalization

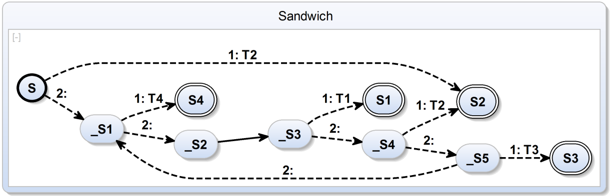


Core SCChart after normalization (Surface & Depth)

Trigger Normalization (Cont'd)

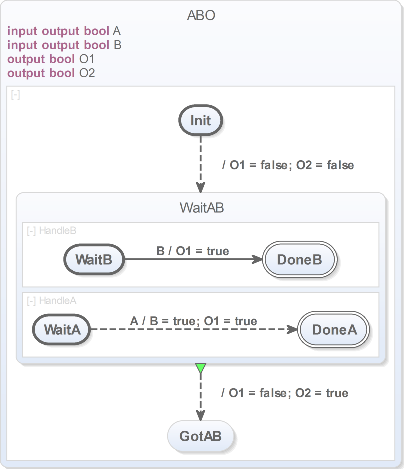


Core SCChart before normalization

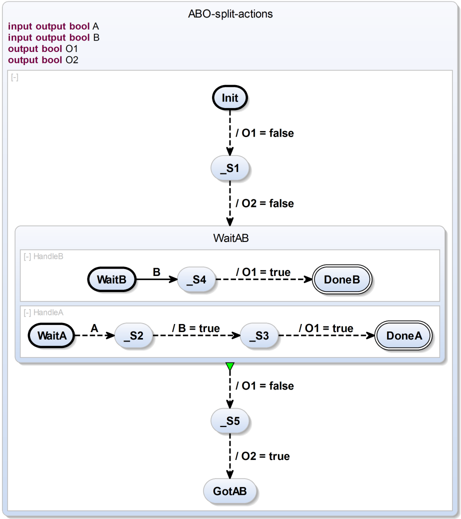


Core SCChart after optimized normalization

ABO - Normalization Example (Actions)

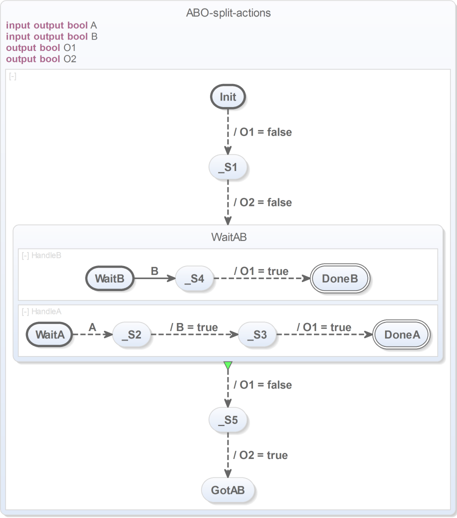


ABO Core SCChart

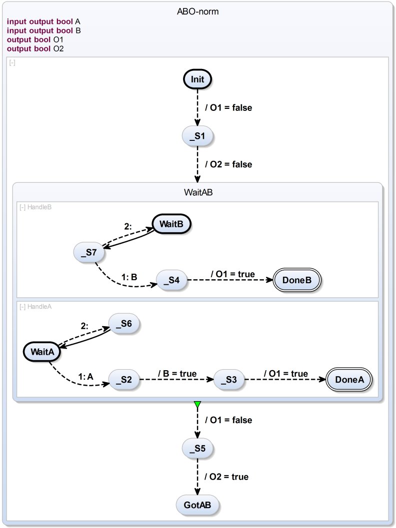


. ABO Core SCChart with normalized actions

ABO - Normalization Example (Actions & Trigger)



ABO Core SCChart with normalized actions

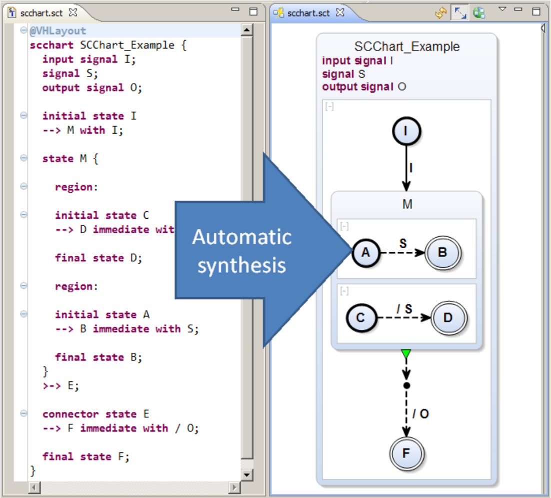


. ABO Normalized SCChart

Overview

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

Textual Modeling with KLighD



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

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

SCCharts Demo

LIVE DEMO

Conclusions

- SyncCharts **are** a great choice for specifying deterministic control-flow behavior…
- … but does 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
  → Reinhard will give details :-)

To Go Further

Charles André.

Semantics of SyncCharts, 2003.

Gérard Berry.

The Esterel v5 Language Primer, 2000.

Schneider, C., Spönemann, M., and von Hanxleden, R.

Just model! – Putting automatic synthesis of node-link-diagrams into practice.

In *Proceedings of the IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC'13)* (San Jose, CA, USA, 15–19 Sept. 2013).

UNI Kiel, Real-Time and Embedded Systems group.

KIELER webpage.

http://www.informatik.uni-kiel.de/en/rtsys/kieler/.

von Hanxleden, R., Lee, E. A., Motika, C., and Fuhrmann, H.

Multi-view modeling and pragmatics in 2020 — position paper on designing complex cyber-physical systems.

In *Proceedings of the 17th International Monterey Workshop on Development, Operation and Management of Large-Scale Complex IT Systems, LNCS* (Oxford, UK, Dec. 2012), vol. 7539.

von Hanxleden, R., Mendler, M., Aguado, J., Duderstadt, B., Fuhrmann, I., Motika, C., Mercer, S., and O'Brien, O.

Sequentially Constructive Concurrency—A conservative extension of the synchronous model of computation.

That's all folks!

Any questions or suggestions?

Sequentially Constructive MoC

- Natural sequencing prescribes deterministic scheduling
  - `stmt1; stmt2`
  - `trigger/effect`
- Only concurrent data dependencies matter
  - Sequential data dependencies do not lead to rejection
- Deterministic concurrent scheduling:
  Distinguish between relative and absolute writes
  - Absolute writes: `x = false`
  - Relative writes: `x = x | true`
  - Reads: `y = x`
  - (1) Absolute writes, (2) relative writes, (3) reads
- Sequentially Constructiveness fully subsumes
  Berry Constructiveness

Concurrency with Threads

- Typical *observer pattern* implemented with Java Threads

```
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

- Not thread safe! E.g., multiple threads call `setValue()`.

Synchronous Program Classes