Synchronous Languages—Lecture 11

Prof. Dr. Reinhard von Hanxleden

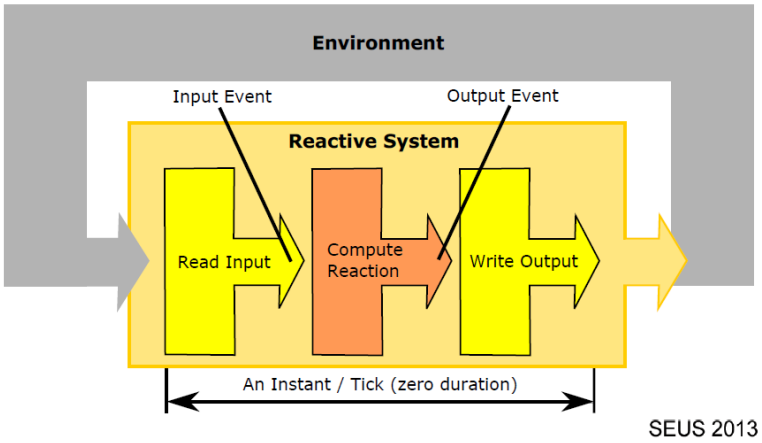
Christian-Albrechts Universität Kiel
Department of Computer Science
Real-Time Systems and Embedded Systems Group

6 Dec. 2016

Last compiled: December 15, 2016, 10:40 hrs

SCCharts — Sequentially Constructive Statecharts for Safety-Critical Applications

Reactive Embedded Systems

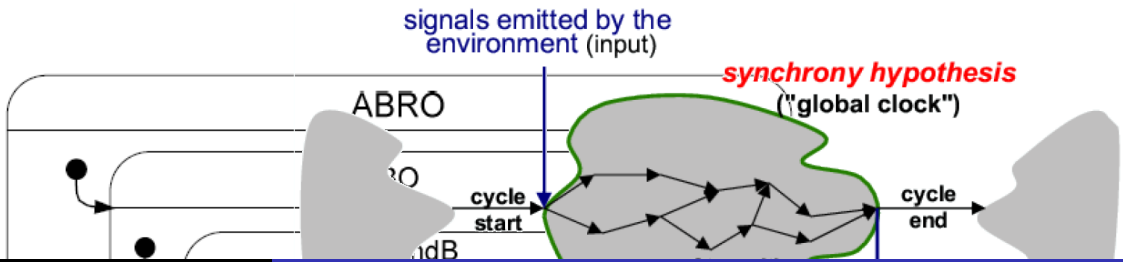


- Embedded systems react to inputs with computed outputs
- Typically state based computations
- Computations often exploit concurrency → Threads

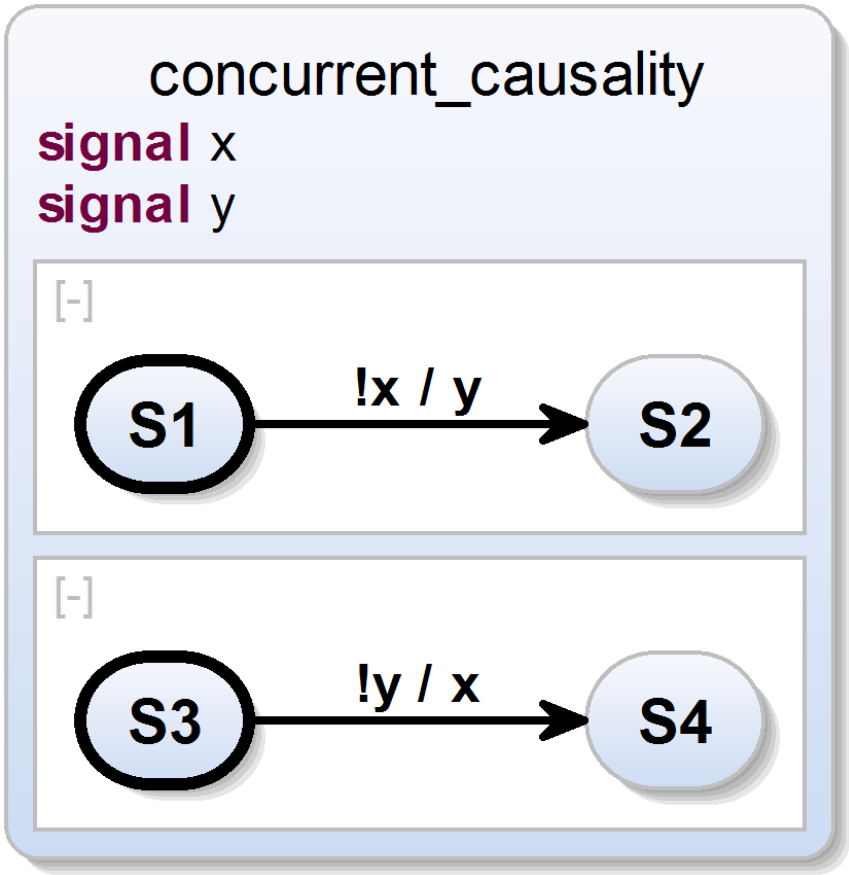
```
public class ValueHolder {
   private List listeners = new LinkedList();
   private int value;
```

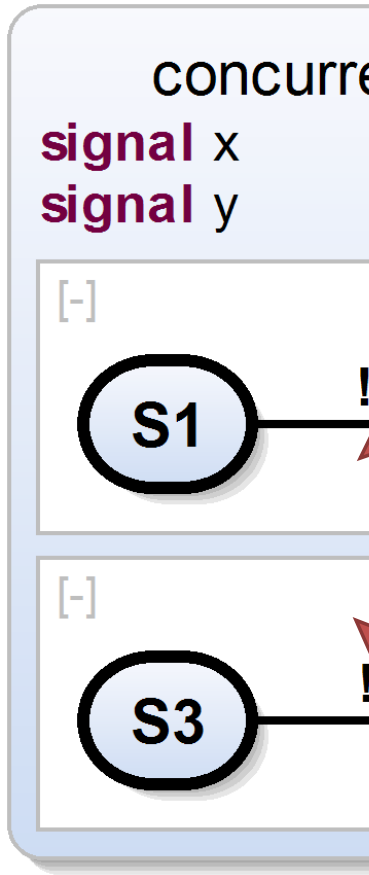
SyncCharts

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

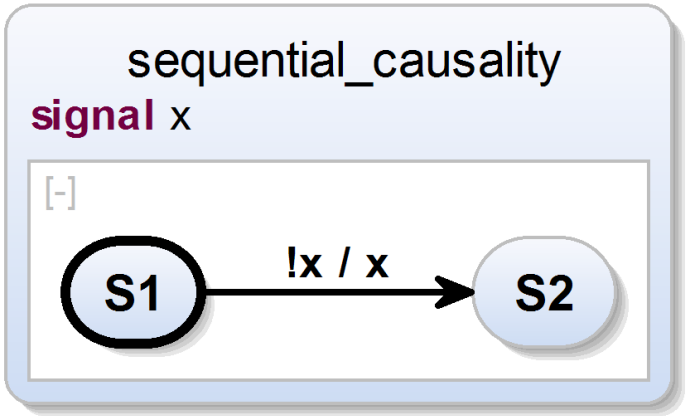


Causality in SyncCharts





Causality in SyncCharts (cont'd)



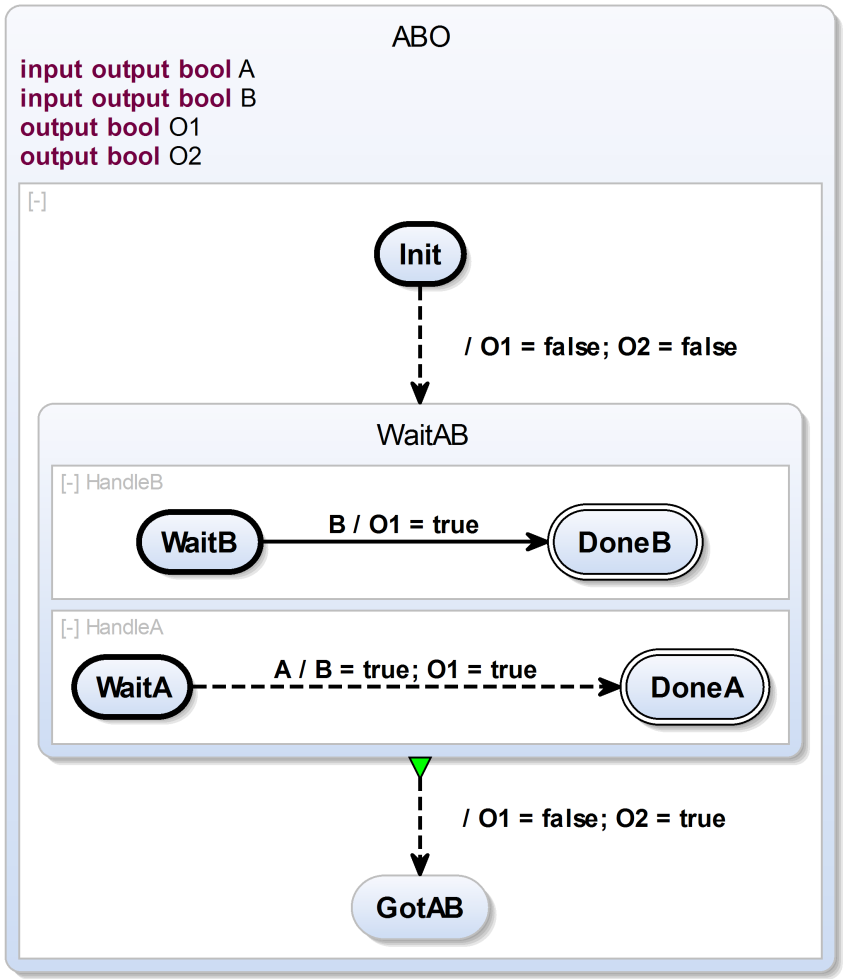

```
if (!done) {

   ...

   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)**

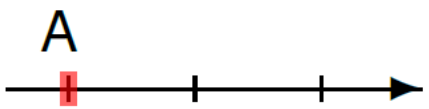
Overview

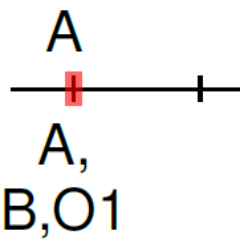
- SCCharts Overview
- Extended SCCharts → Core SCCharts
- Normalizing Core SCCharts
- Implementation in KIELER

SCCharts Overview

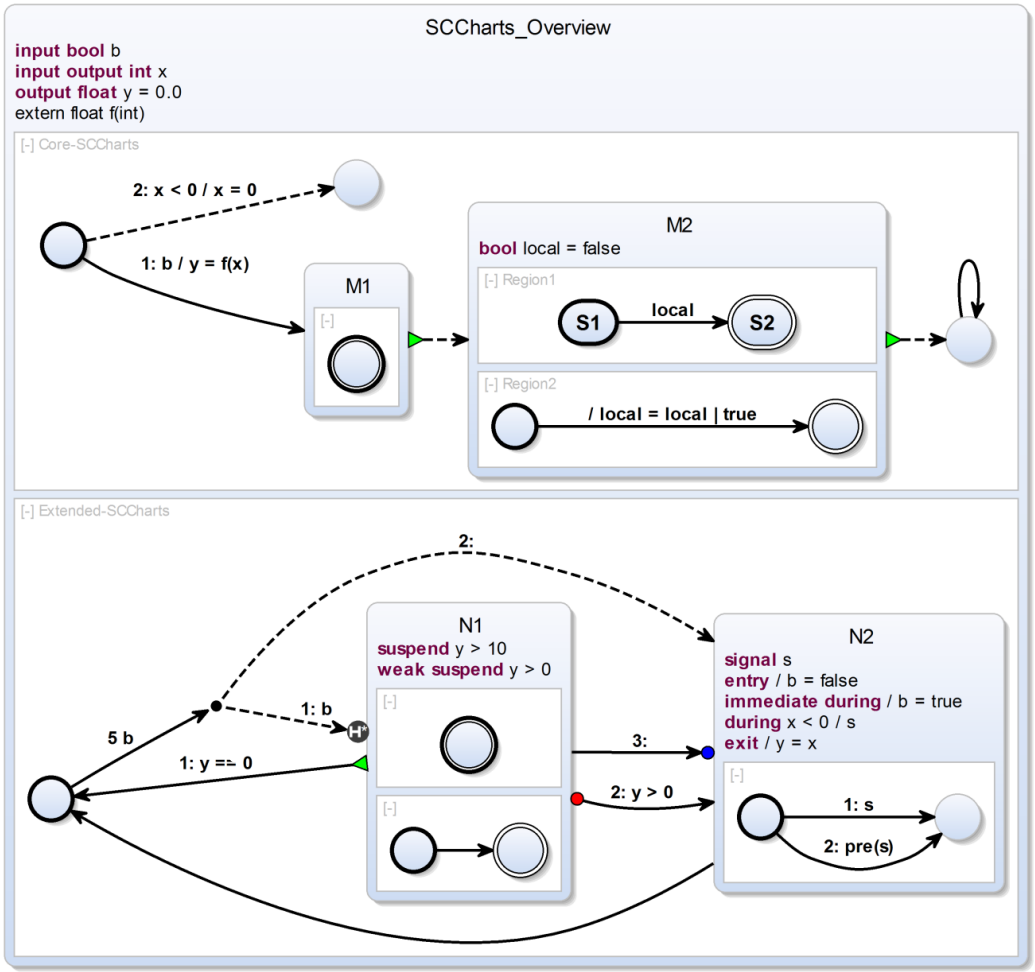


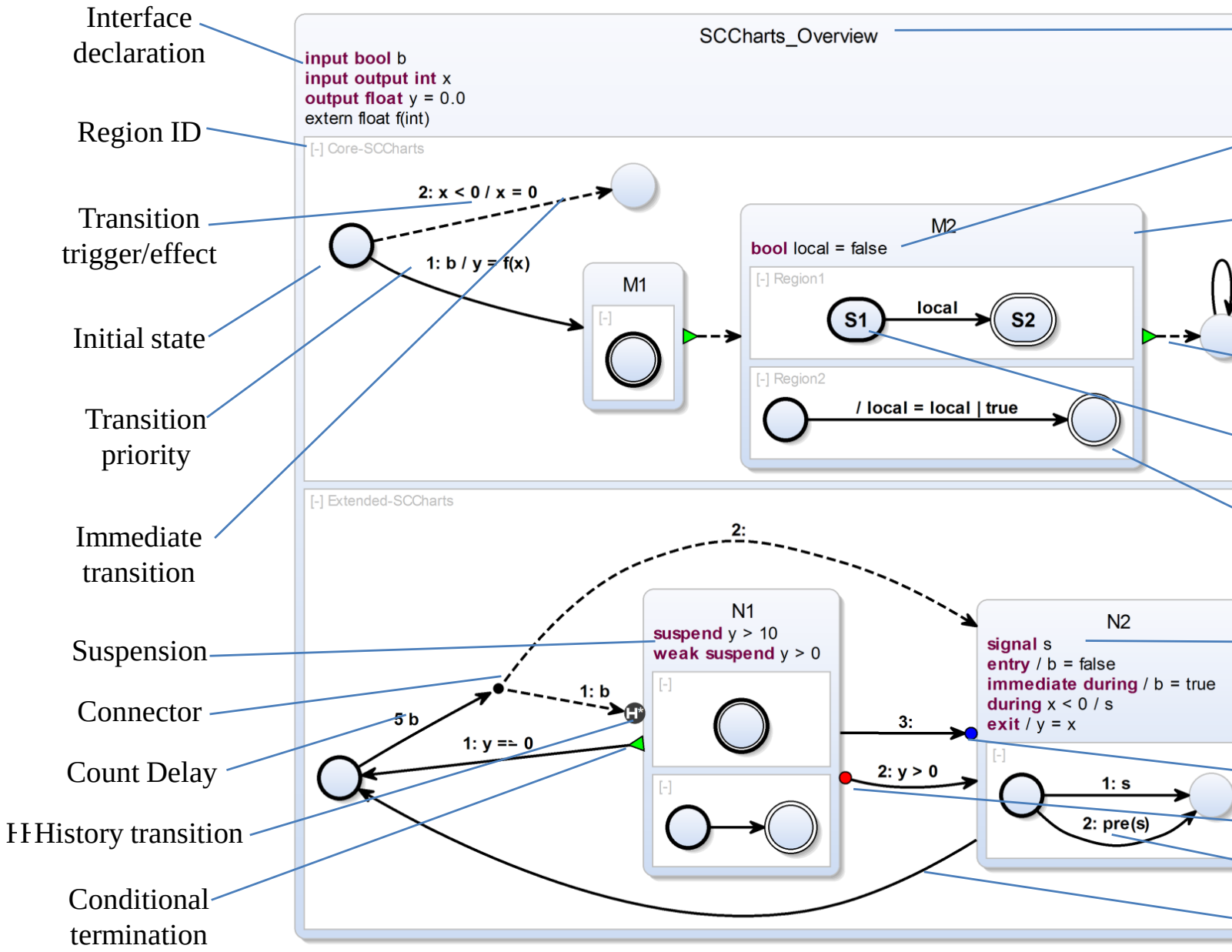
- SCCharts $\widehat{=}$
  SyncCharts syntax +
  Seqentially Constructive semantics
- *Hello World* of Sequential Constructiveness: **ABO**
  - Variables instead of signals
  - Behavior (briefly)
    1. Initialize
    2. Concurrently wait for inputs A or B to become *true*
    3. Once A and B are true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



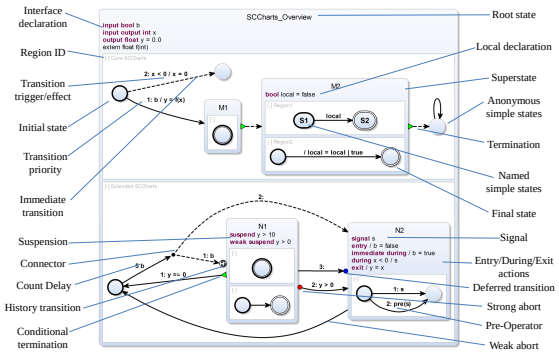


SCCharts — Features



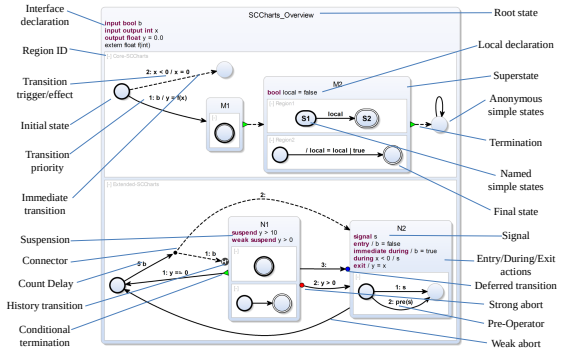

Interface declaration
Region ID
Transition trigger/effect
Initial state
Transition priority
Immediate transition
Suspension
Connector
Count Delay
History transition
Conditional termination
SCCharts_Overview
input bool b
input output int x
output float y = 0.0
extern float f(int)
[-] Core-SCCharts
2: 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
1: b
5 b
1: y == 0
3:
2: y > 0
N2
signal s
entry / b = false
immediate during / b = true
during x < 0 / s
exit / y = x
1: s
2: pre(s)

Motivation for Core SCCharts



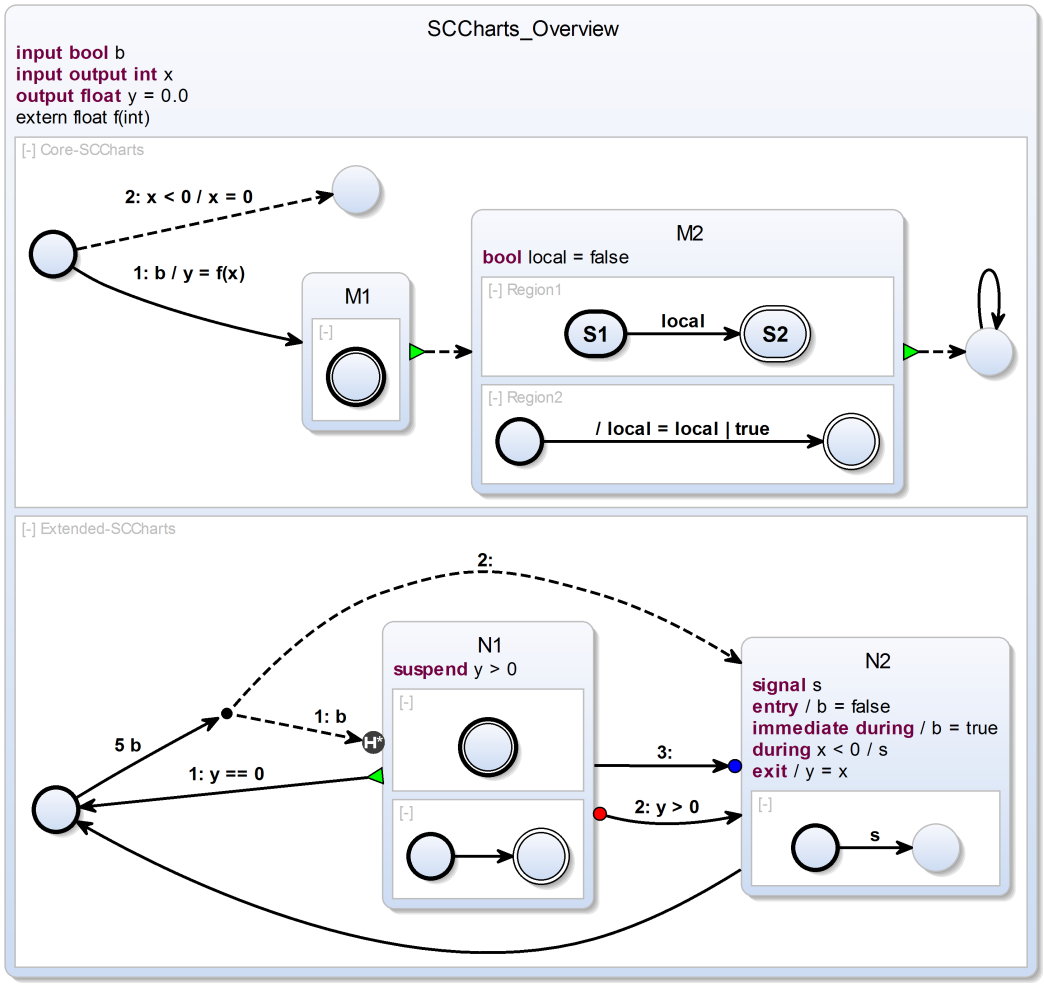
- **Observation I**: Numerous features
  - ☺ Compactness / readability of models
  - ☹ Steeper learning curve
  - ☹ Direct compilation & verification more complex
- **Observation II**: Various features can be expressed by other ones
- **Consequence**: ⇒ Define extended features by means of base features

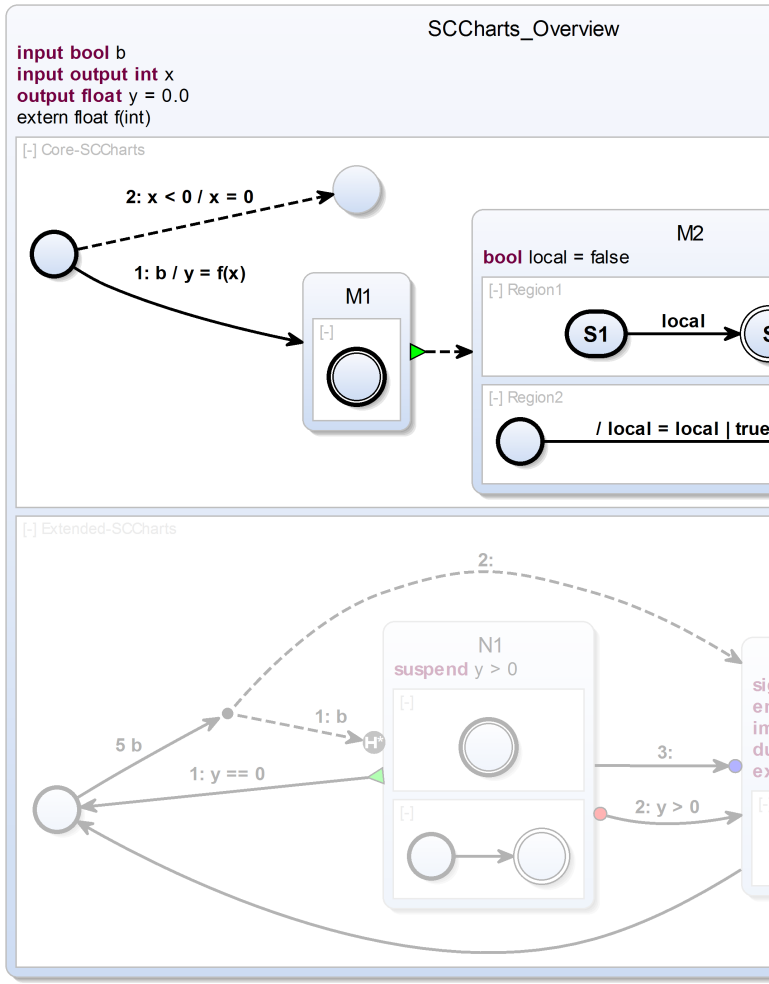
Motivation (Cont'd)



- **Advantages**:
  - 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

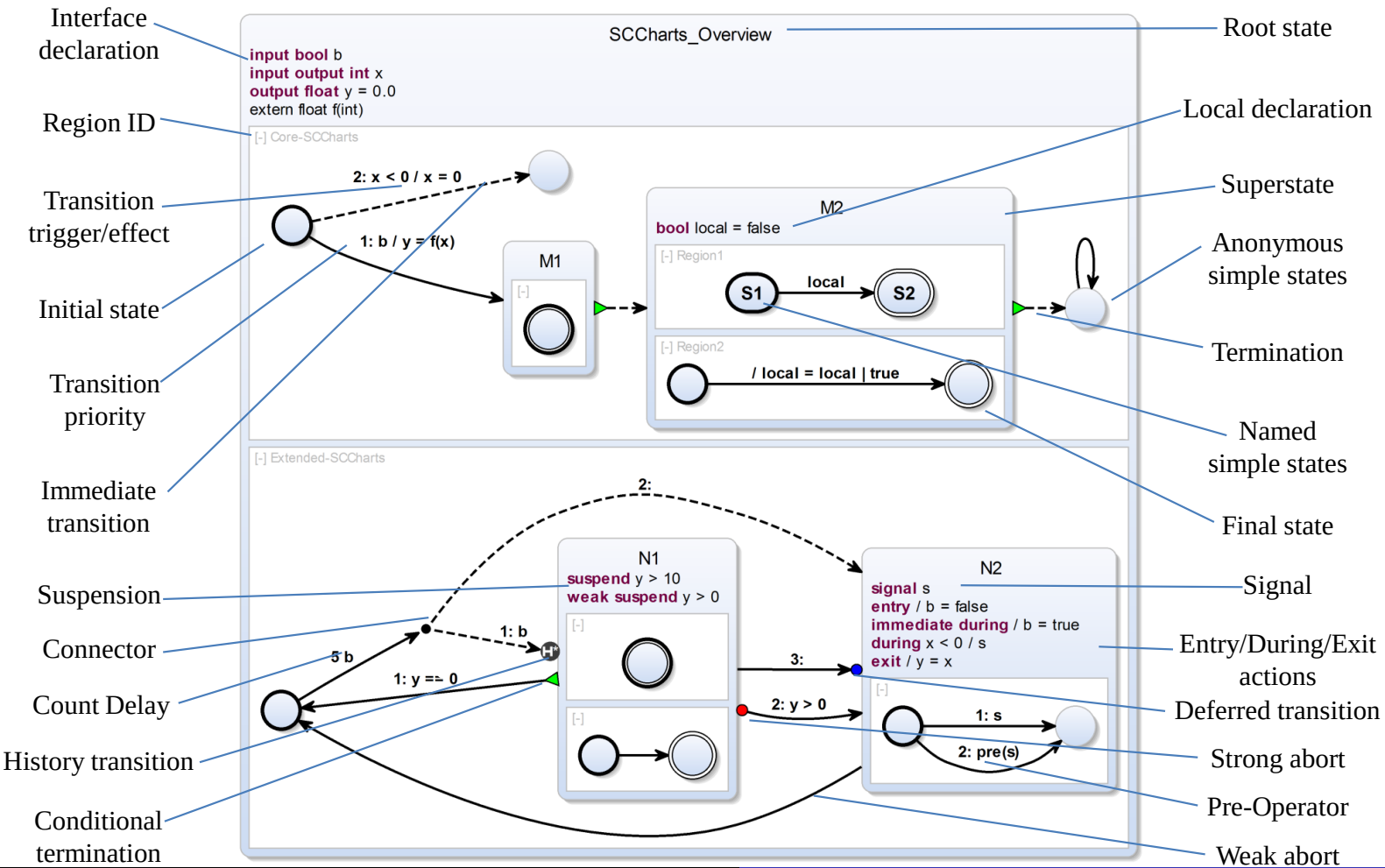




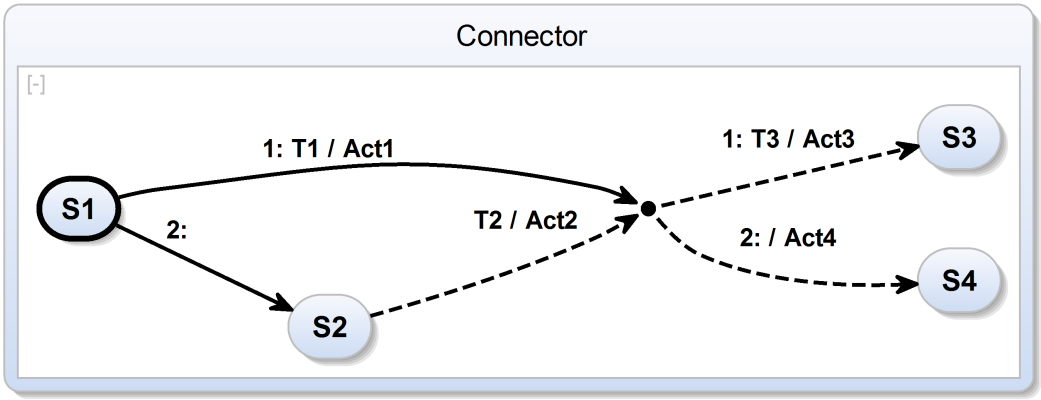
Overview

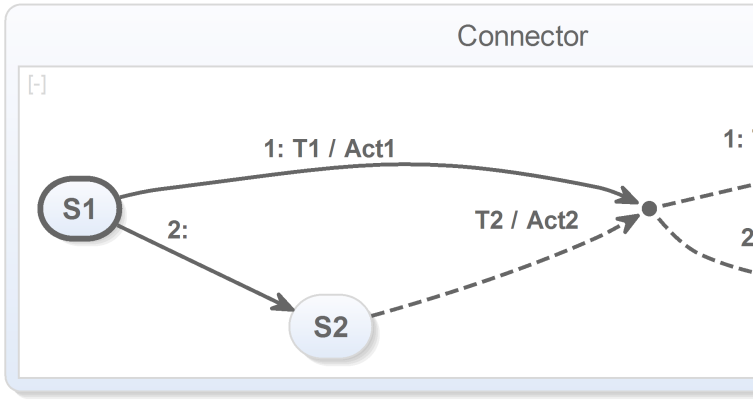
- SCCharts Overview
- Extended SCCharts → Core SCCharts
- Normalizing Core SCCharts
- Implementation in KIELER

SCCharts — Core Transformations Examples

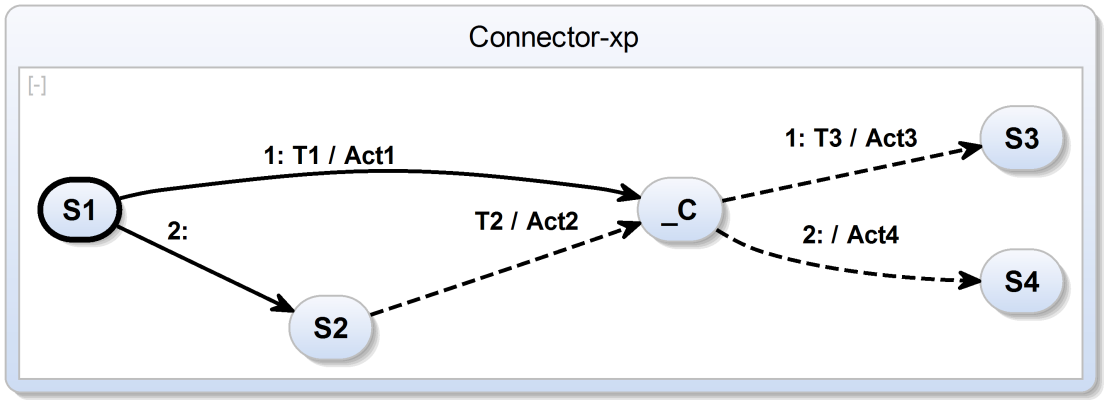


Transforming Connectors



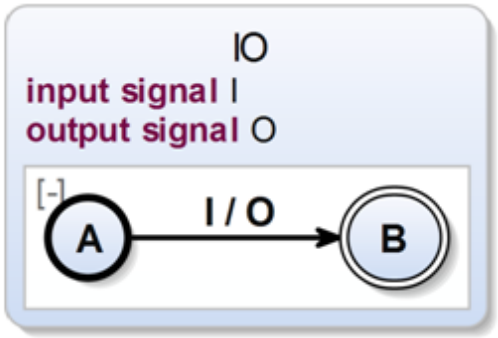


Extended SCCharts with Connectors



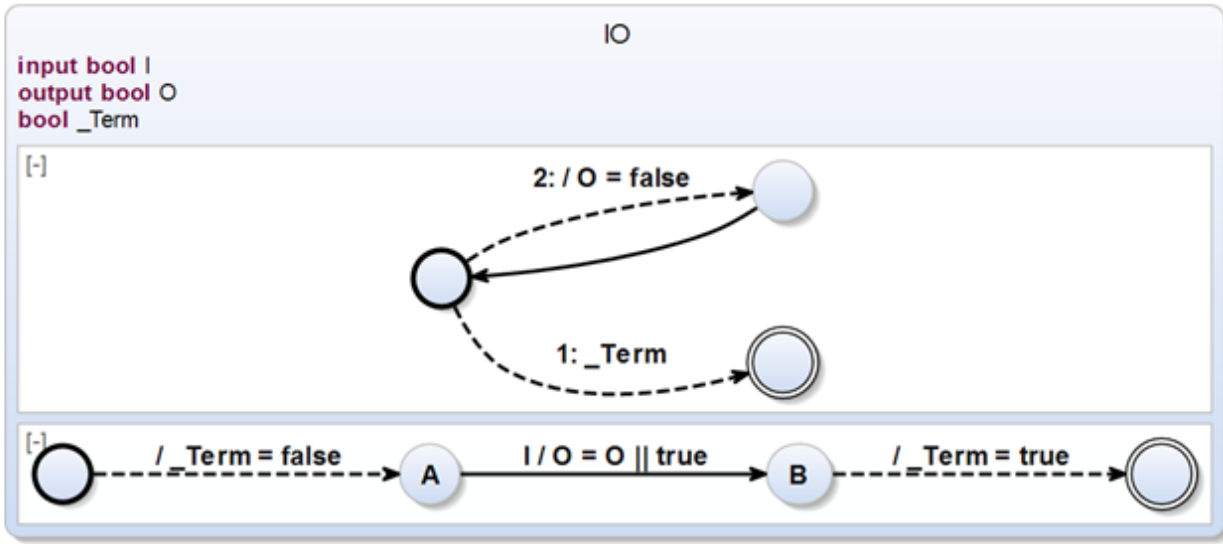
Core SCCharts without Connectors

Transforming Signals



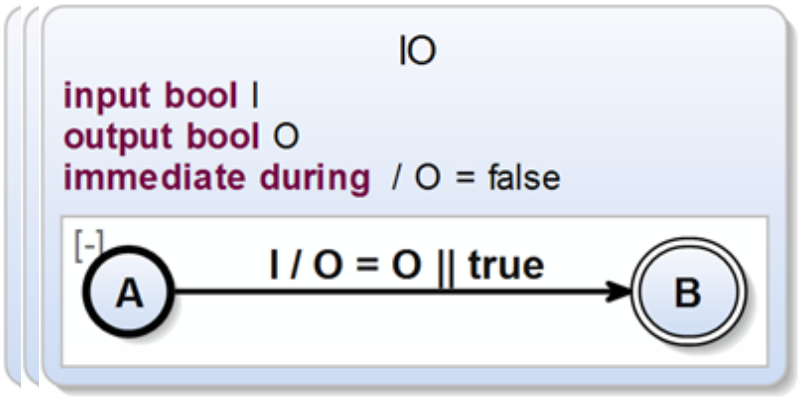
Extended SCCharts with Signals

Signal expansion



Core SCCharts only

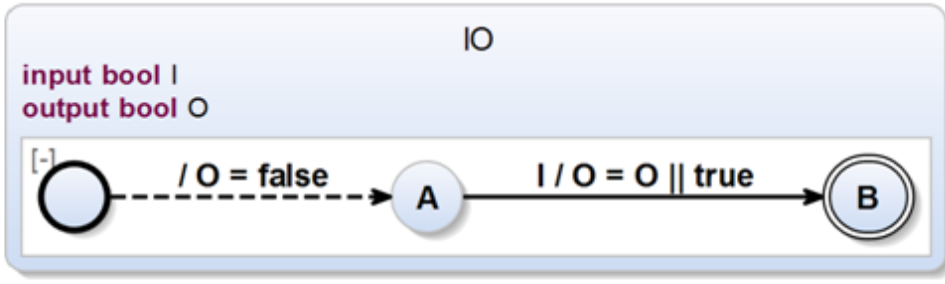
Optimization



Core SCCharts with During Actions

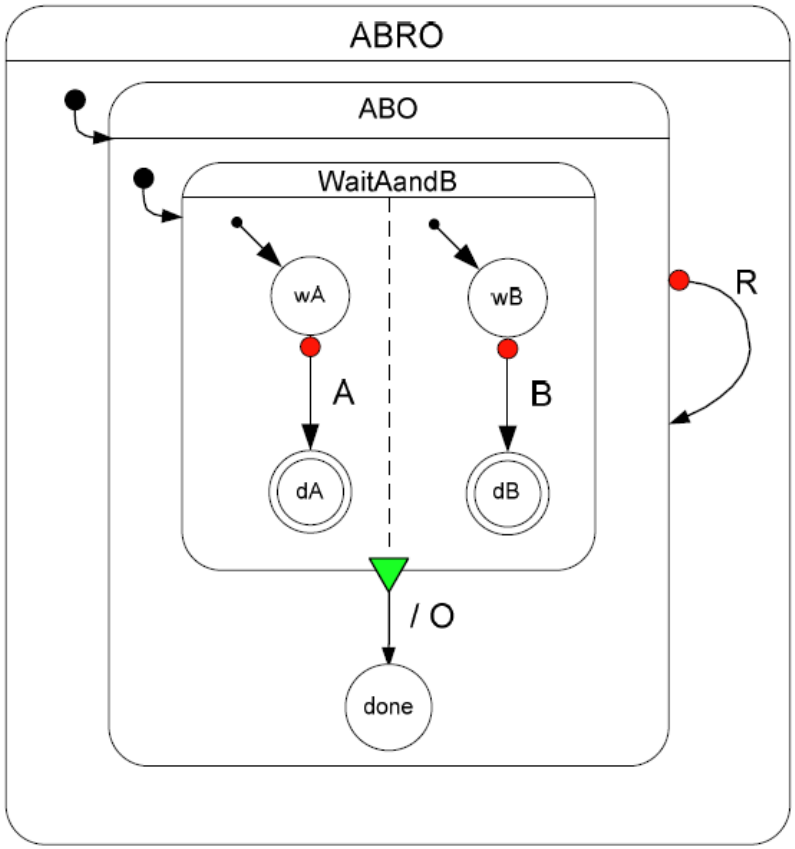
Action expansion

Action expansion (alternative)

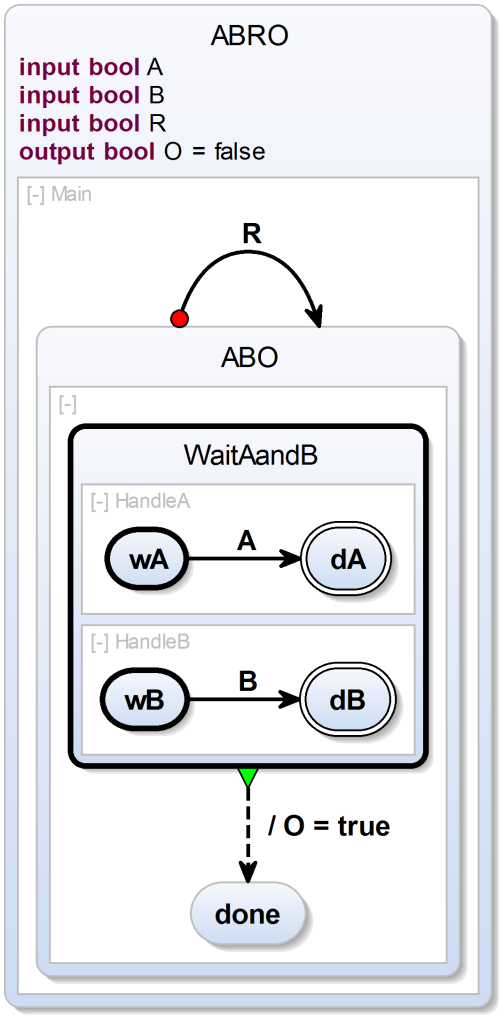


Core SCCharts only (optimized)

SyncChart and SCChart ABRO

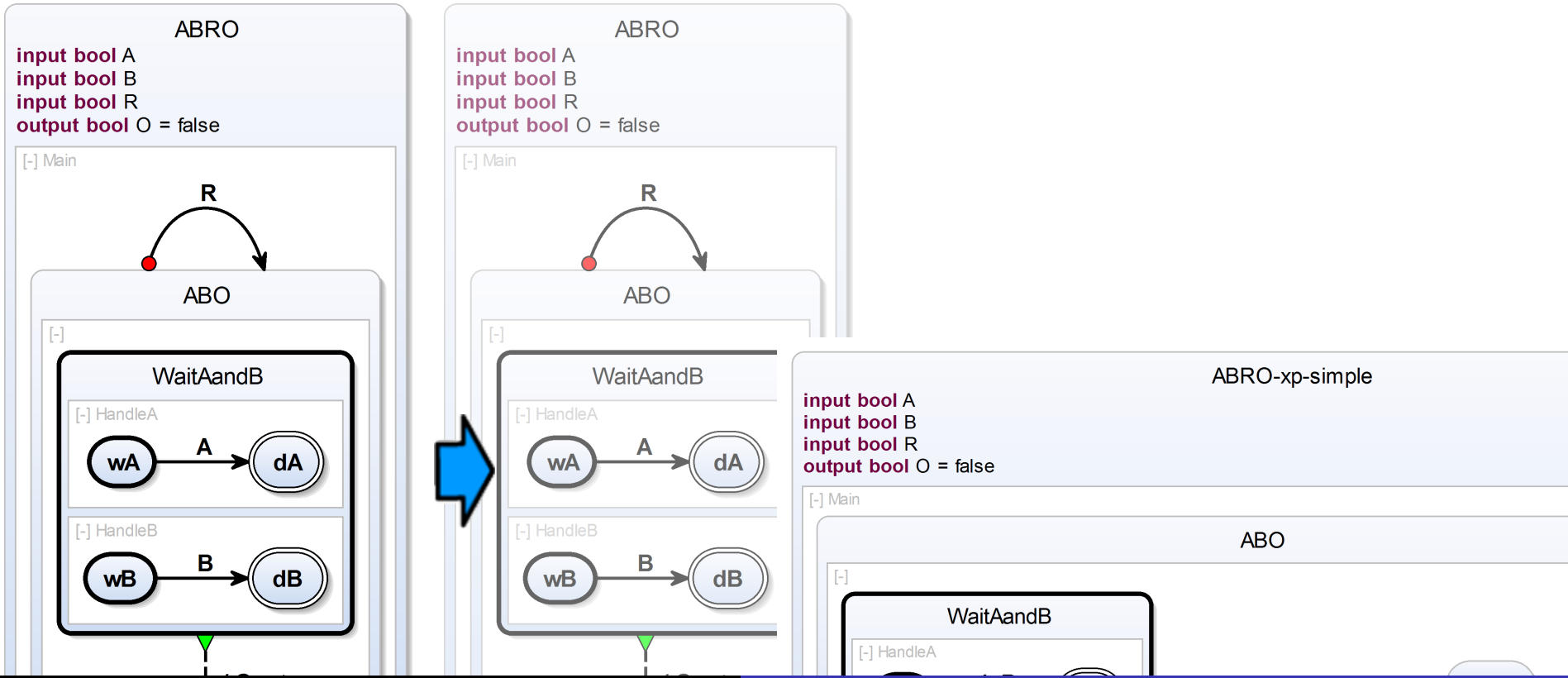


[Charles André, Semantics of SyncCharts, 2003]

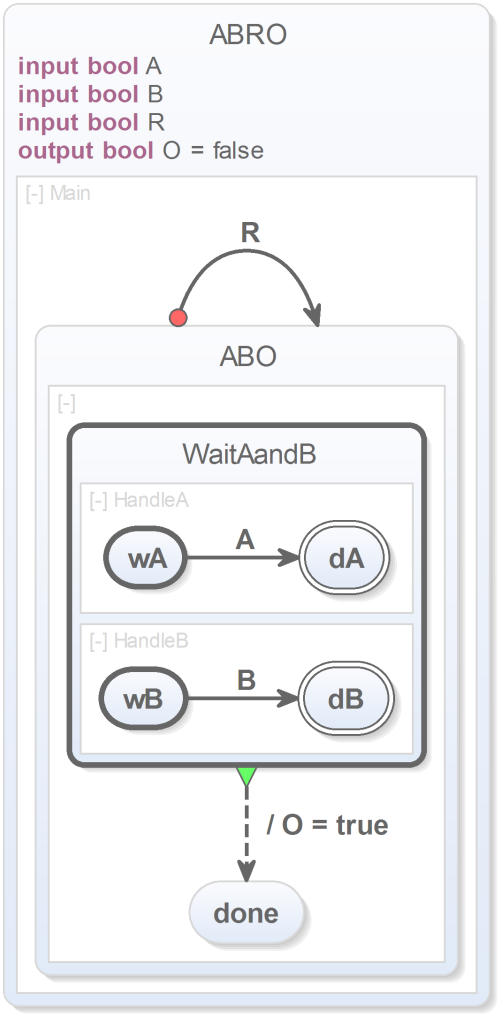


. ABRO SCChart

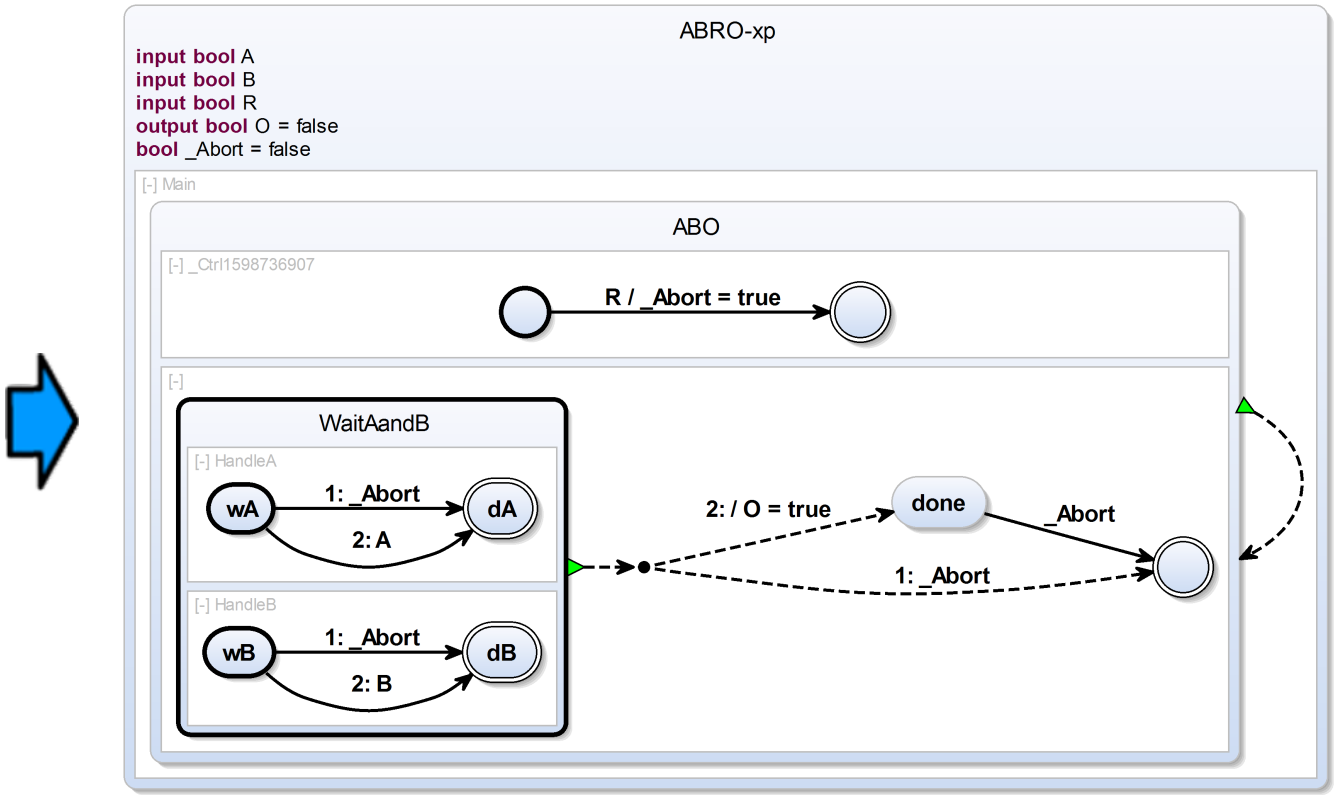
ABRO — Transforming Strong Aborts



ABRO — Transforming Strong Aborts (cont'd)

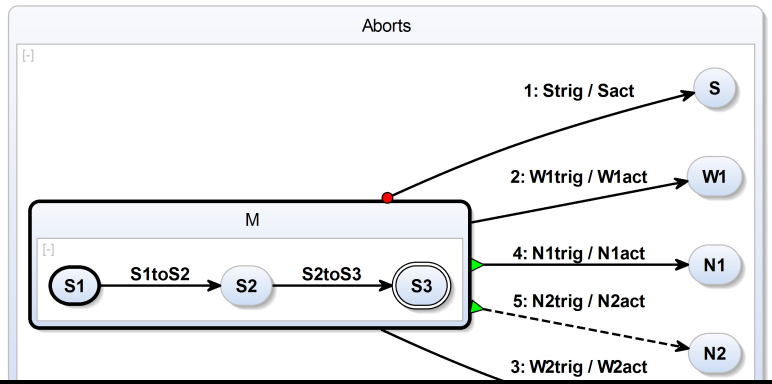


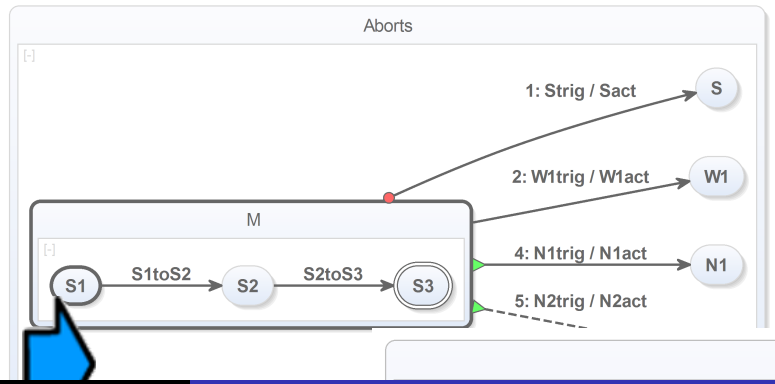
ABRO SCChart with Strong Abort



Core SCChart without Strong Abort and WTO

Transforming General Aborts

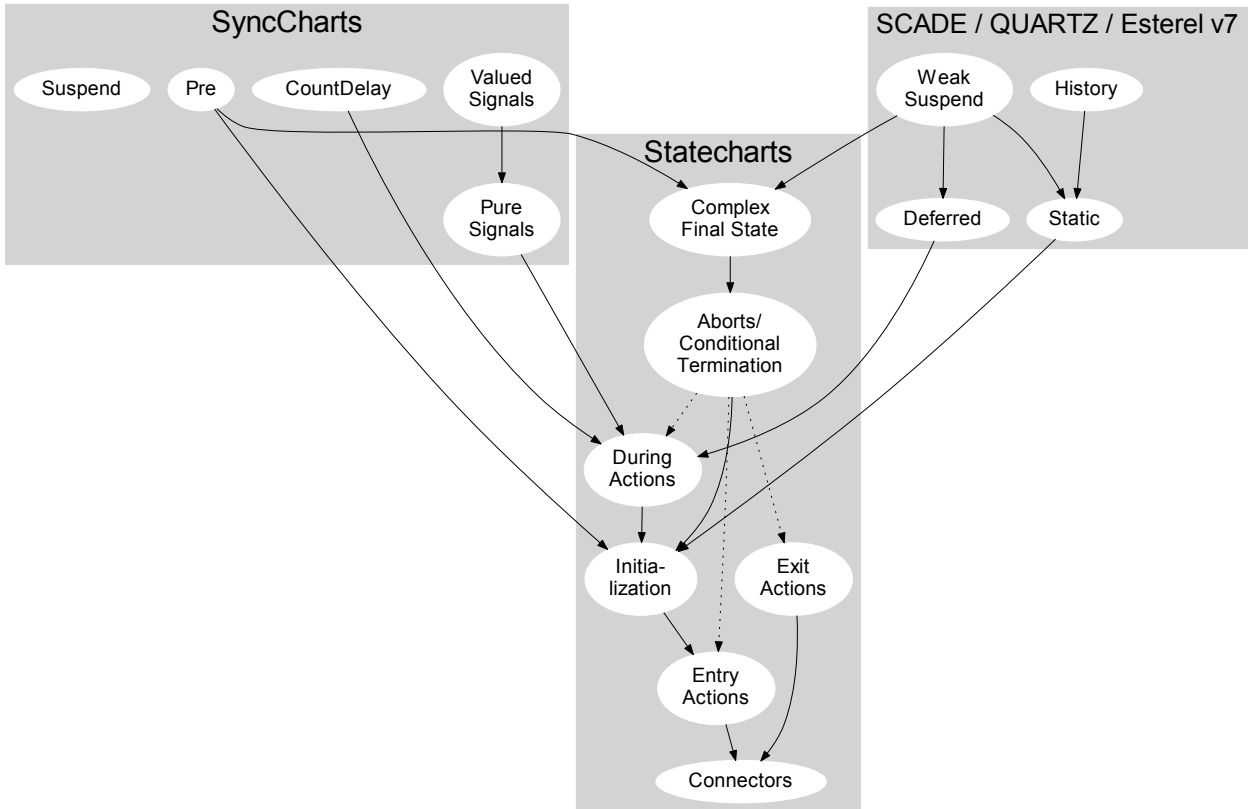




Overview

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

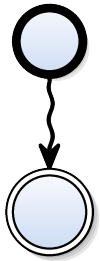
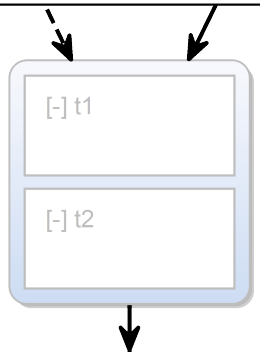
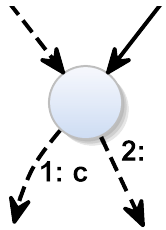
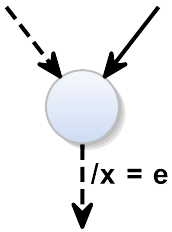
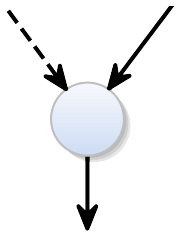
Single-Pass Language-Driven Incremental Compilation (SLIC)



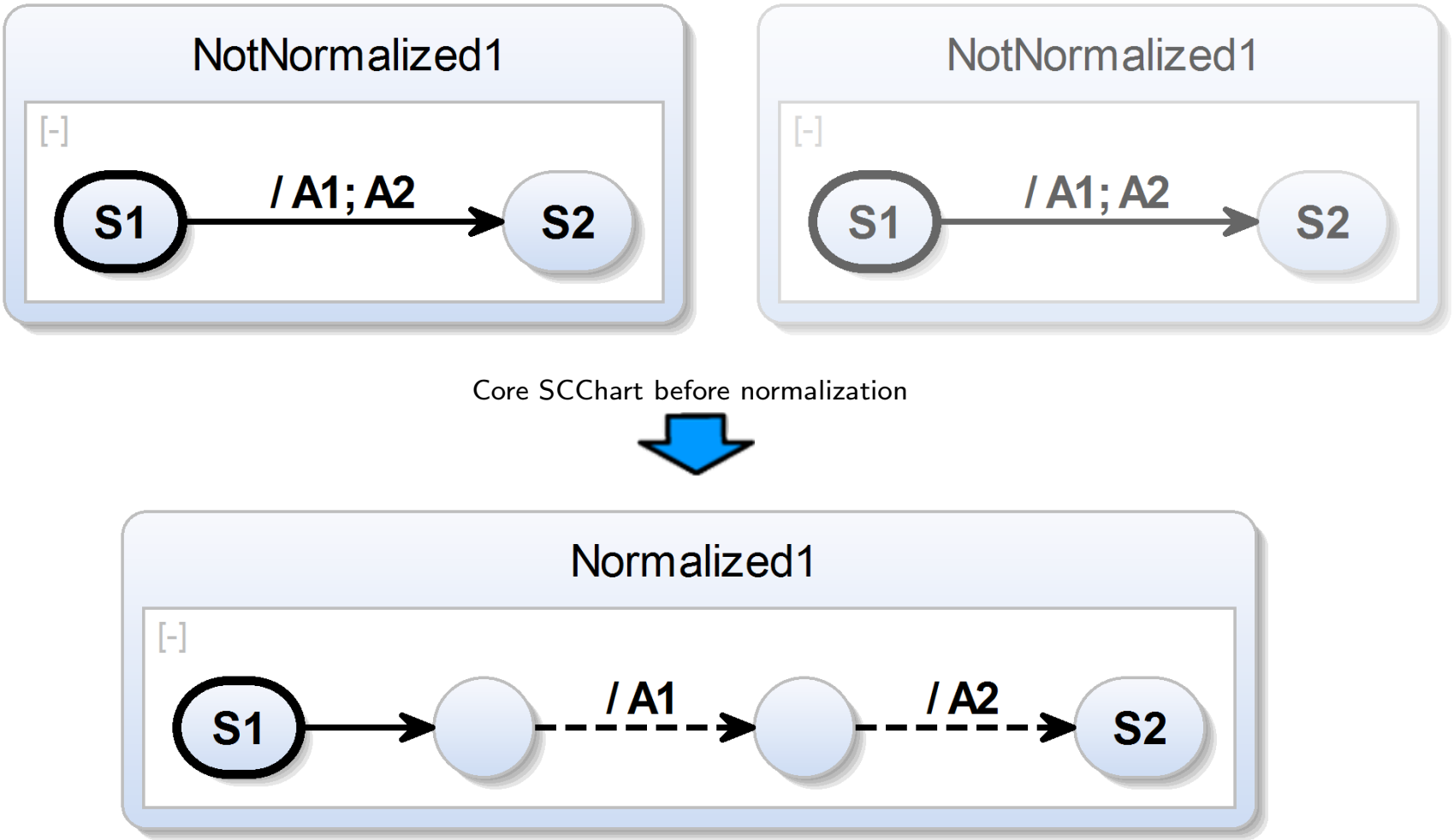
- Some core transformations will produce (use) some other extended features (solid lines)
- Other core transformations cannot handle some extended features (dashed lines)
- $\rightarrow$ Order in which core transformations are applied is important
- $\rightarrow$ Dependencies (do not have any cycle,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

Normalization

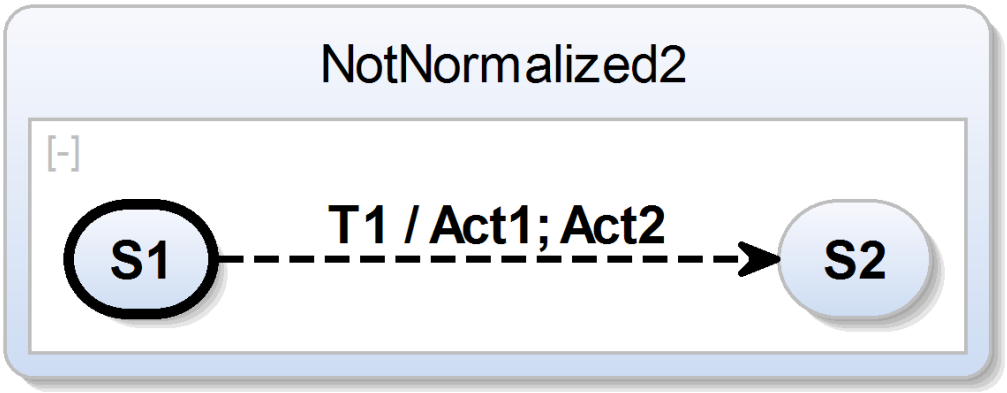
- Further simplify compilation process for Core SCCharts
- Allowed patterns:

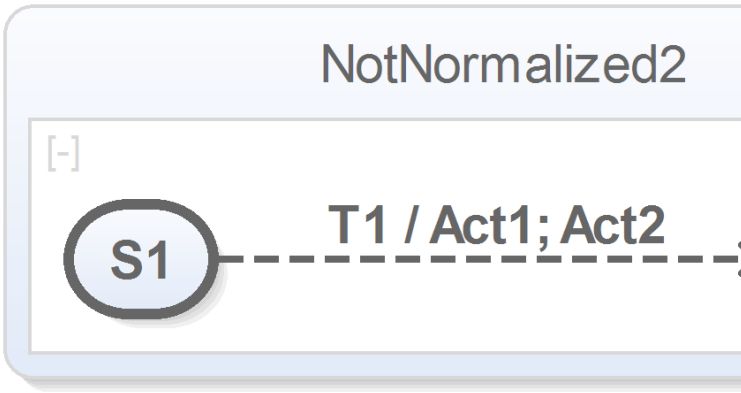
Region	Superstate	Trigger	Action	State
(connected states)	(parallel regions)	(conditionals)	(assignments)	(tick boundary)
				

Actions Normalization

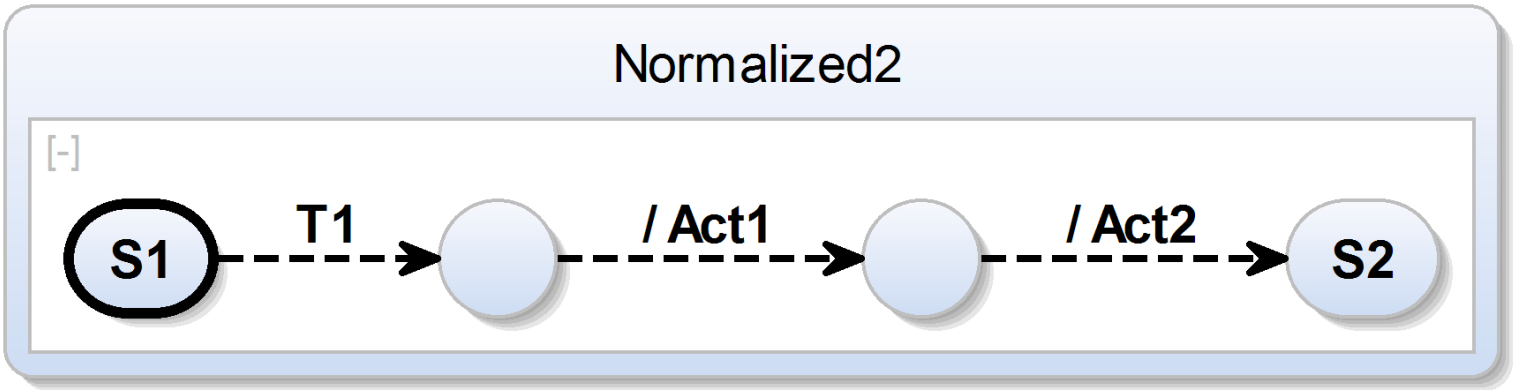


Actions Normalization (cont'd)





Core SCChart before normalization

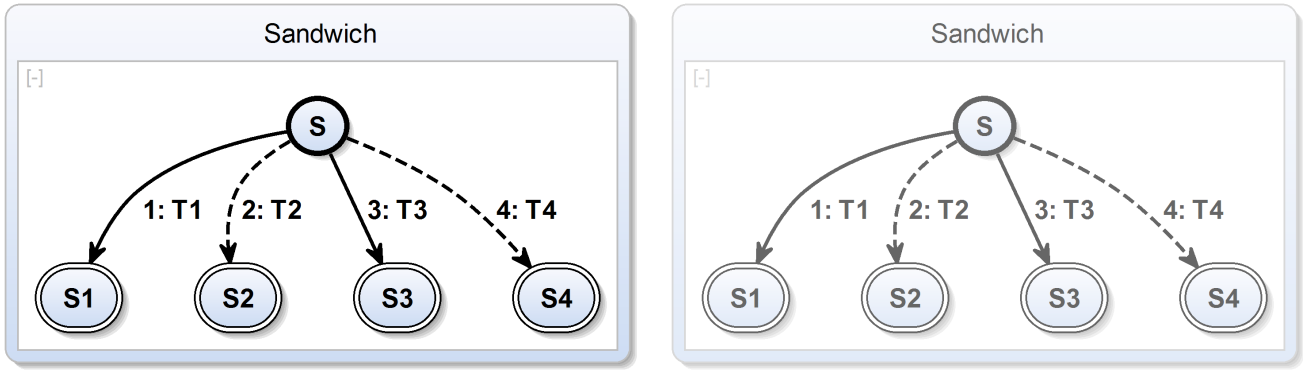


Core SCChart after normalization

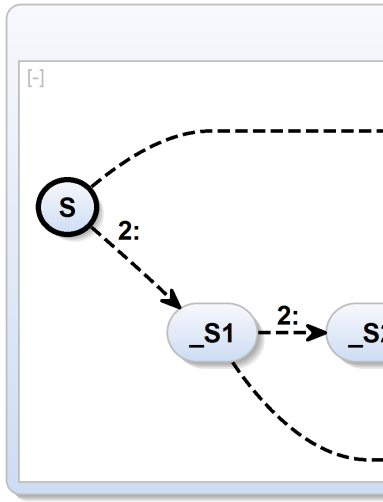
Actions Normalization Implementation Example

```
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

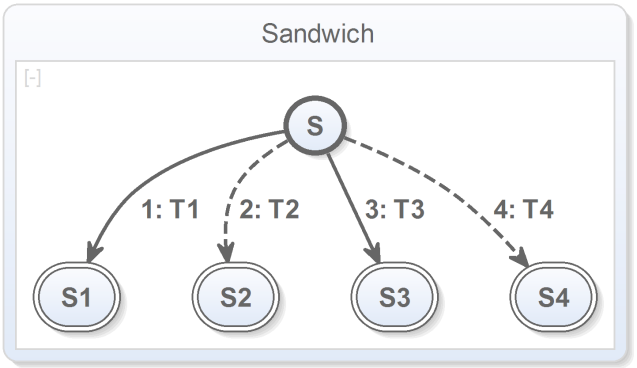


Core SCChart before normalization

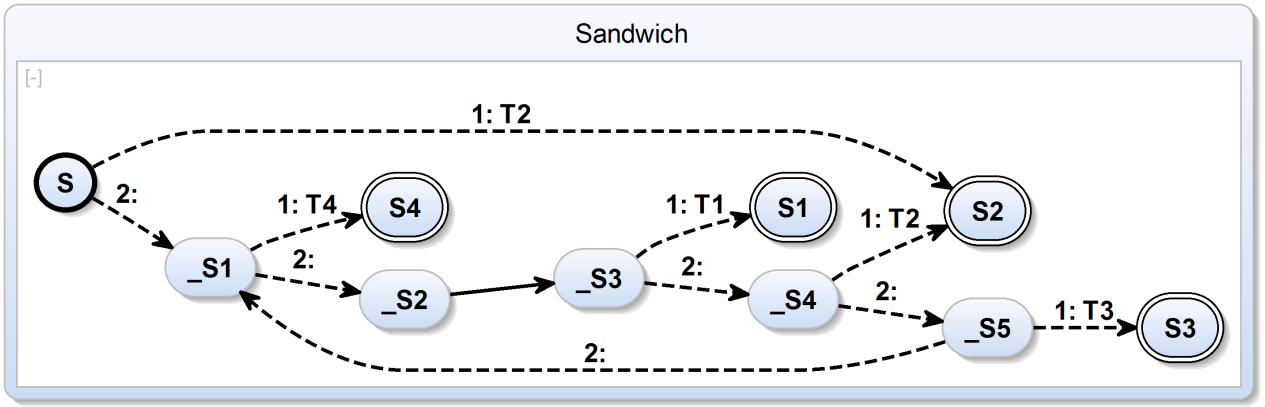


Core SCChart after normalization

Trigger Normalization (Cont'd)

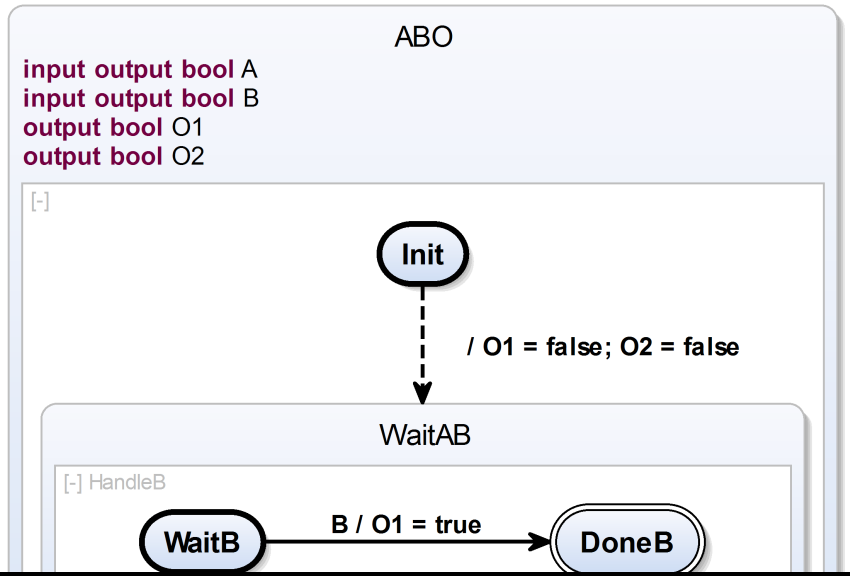


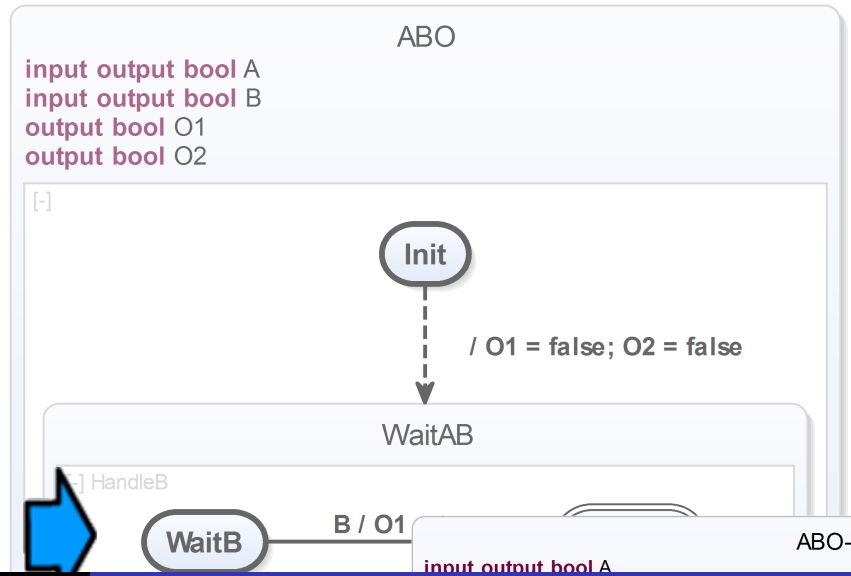
Core SCChart before normalization



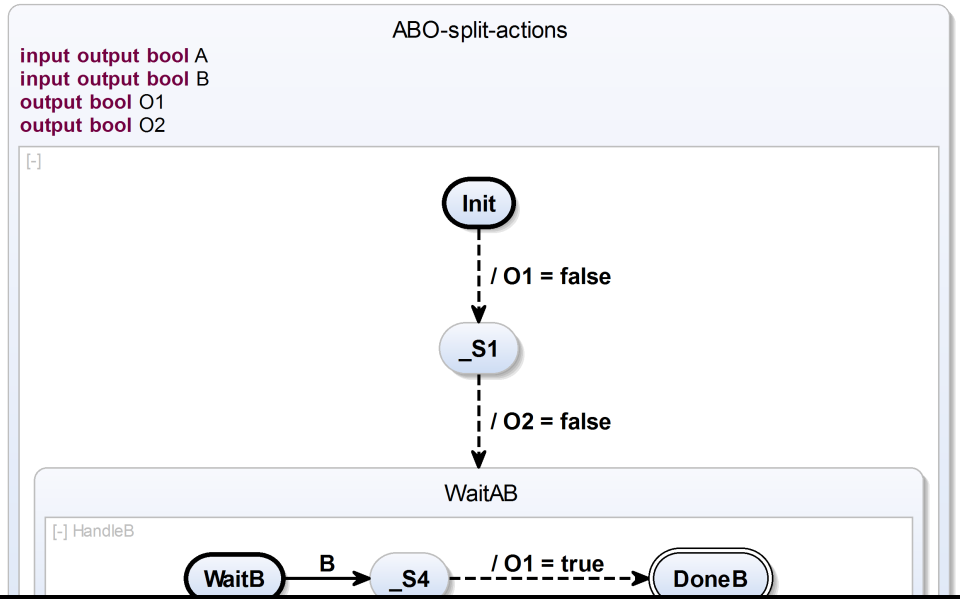
Core SCChart after optimized normalization

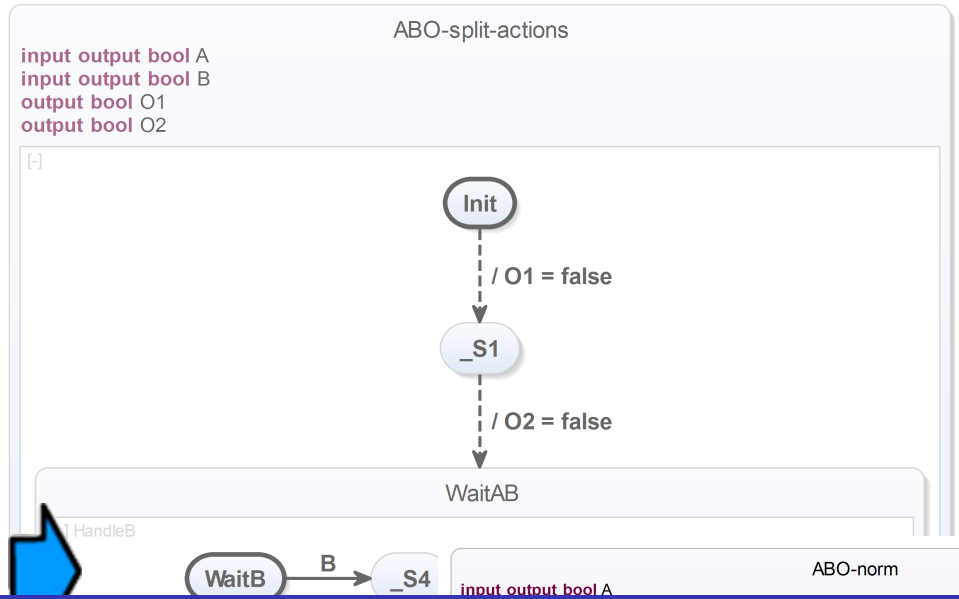
ABO — Normalization Example (Actions)





ABO — Normalization Example (Actions & Trigger)

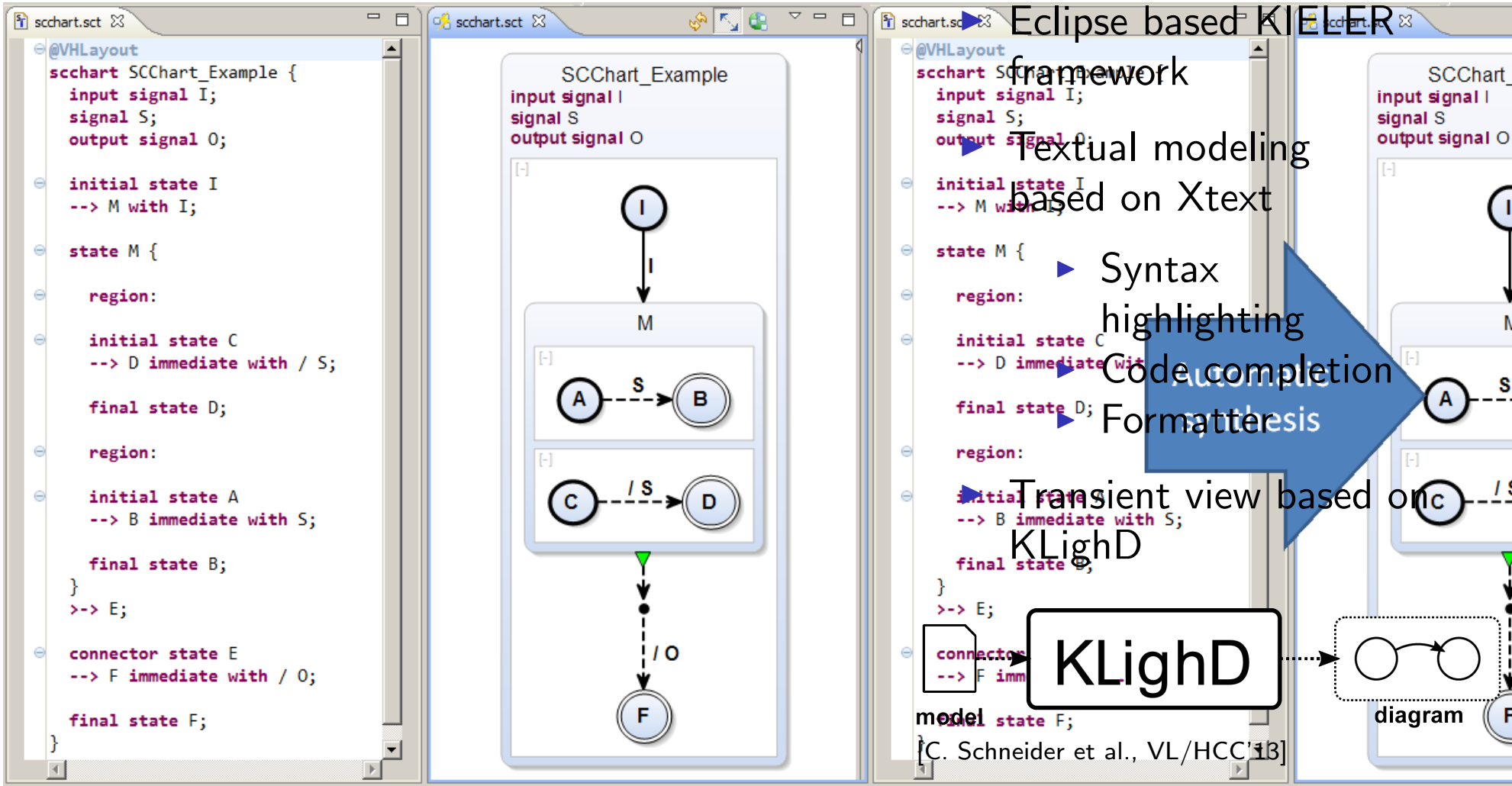




Overview

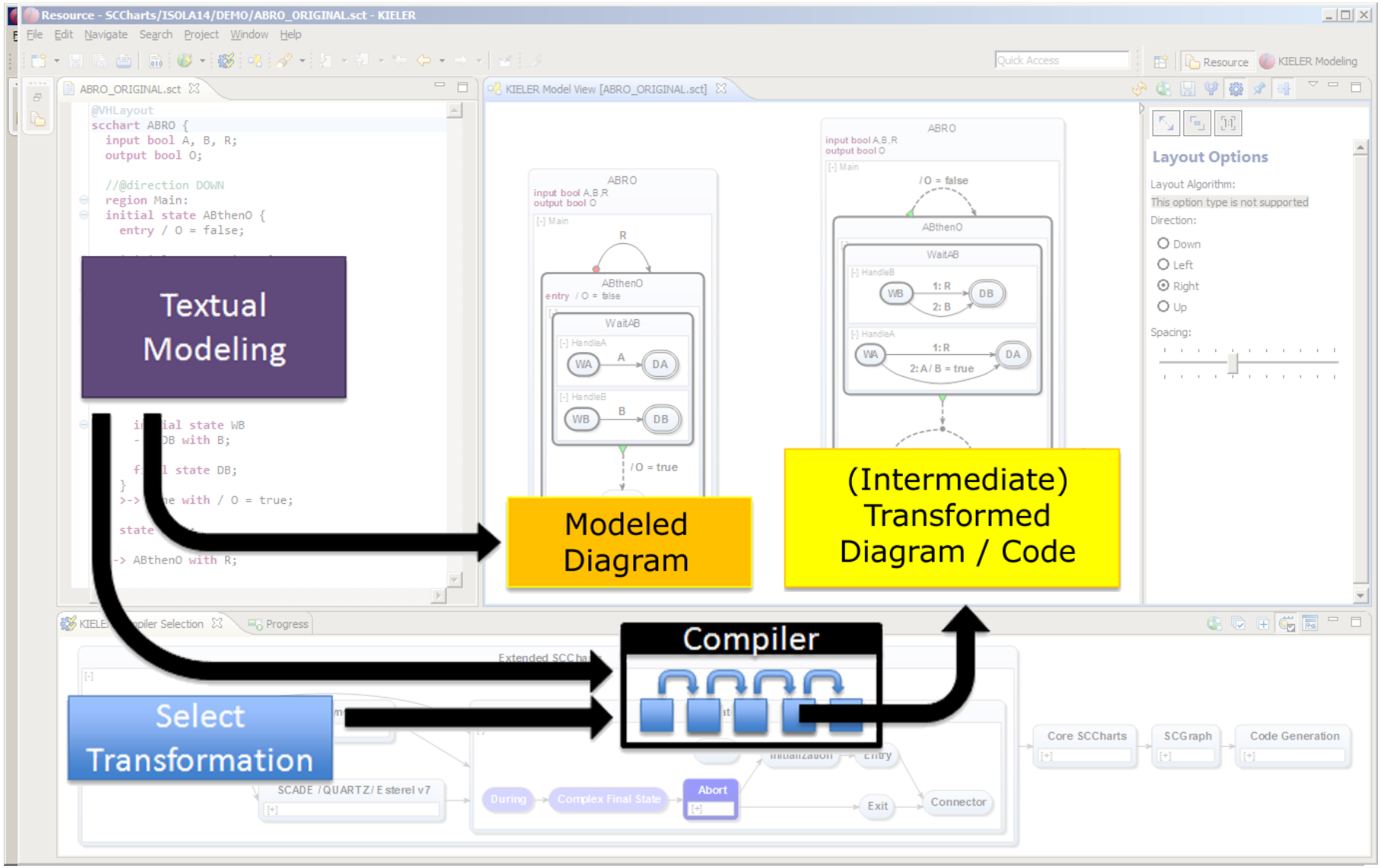
- SCCharts Overview
- Extended SCCharts → Core SCCharts
- Normalizing Core SCCharts
- Implementation in KIELER

Textual Modeling with KLighD



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

SCCharts Interactive 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

- DFG-funded PRETSY Project: `www.pretsy.org`
- 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. `http://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 `http://rtsys.informatik.uni-kiel.de/~biblio/downloads/papers/isola14.pdf`