Synchronous Languages—Lecture 11

Prof. Dr. Reinhard von Hanxleden

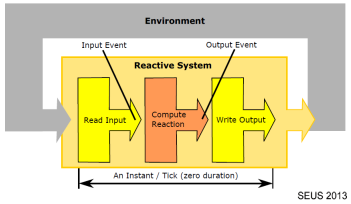
Christian-Albrechts Universität Kiel
Department of Computer Science
Real-Time Systems and Embedded Systems Group

10 Dec. 2018

Last compiled: January 29, 2019, 10:54 hrs

SCCharts — Sequentially Constructive Statecharts for Safety-Critical Applications

Reactive Embedded Systems

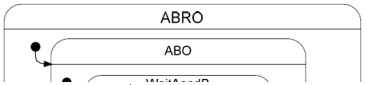


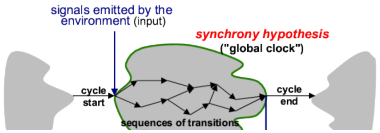
- Embedded systems react to inputs with computed outputs
- Typically state based computations
- Computations often exploit concurrency → Threads
- Threads are problematic → **Synchronous languages**: Lustre, Esterel, SCADE, SyncCharts

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

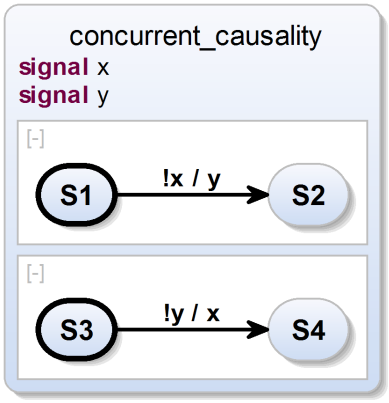

SyncCharts

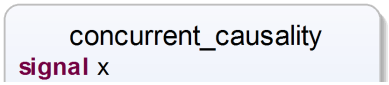
- Statechart dialect for specifying deterministic & robust concurrency
- SyncCharts:
  - Hierarchy, Concurrency, Broadcast
  - Synchrony Hypothesis
    1. Discrete ticks
    2. Computations: Zero time



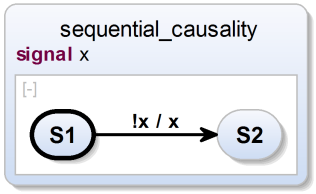


Causality in SyncCharts





Causality in SyncCharts (cont'd)




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

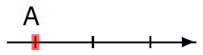
- Rejected by SyncCharts compiler
- *Signal Coherence Rule*
- May seem awkward from SyncCharts perspective, but common paradigm
- Deterministic sequential execution possible using *Sequentially Constructive MoC*
  → **Sequentially Constructive Charts (SCCharts)**

Overview

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

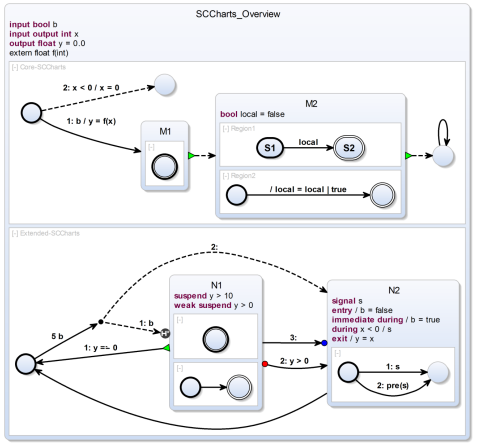
SCCharts Overview

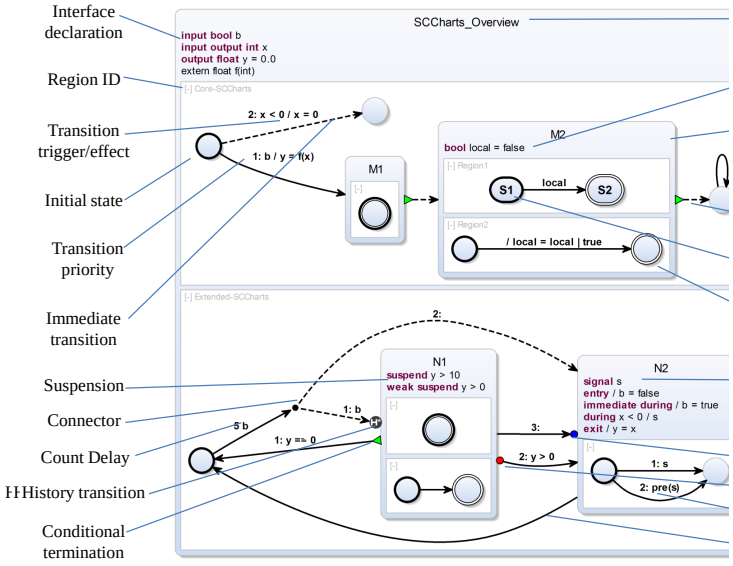
- SCCharts ≙
  SyncCharts syntax +
  Seqentially Constructive semantics
- *Hello World* of Sequential Constructiveness: **ABO**
  - Variables instead of signals
  - Behavior (briefly)
    1. Initialize
    2. Concurrently wait for inputs *A* or *B* to become *true*
    3. Once *A* and *B* are true after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



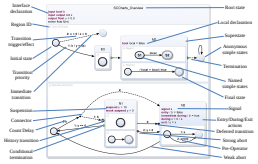
A

SCCharts — Features



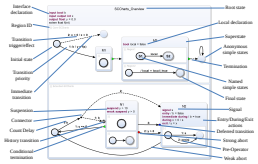


Motivation (Cont'd)



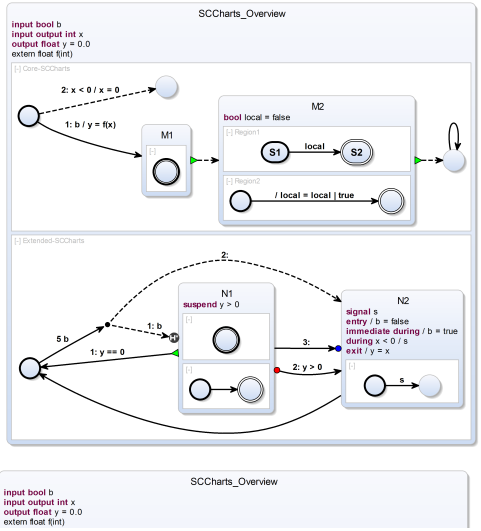
- **Advantages**:
  - Minimal base language (Core SCCharts)
    + advanced features (Extended SCCharts)
    - Similar to Esterel Kernel Statements & Statement Expansion
  - Advanced features are *syntactic sugar*
  - Extensible
  - Compilation (ongoing research)
    - Modular & extensible
    - Less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

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

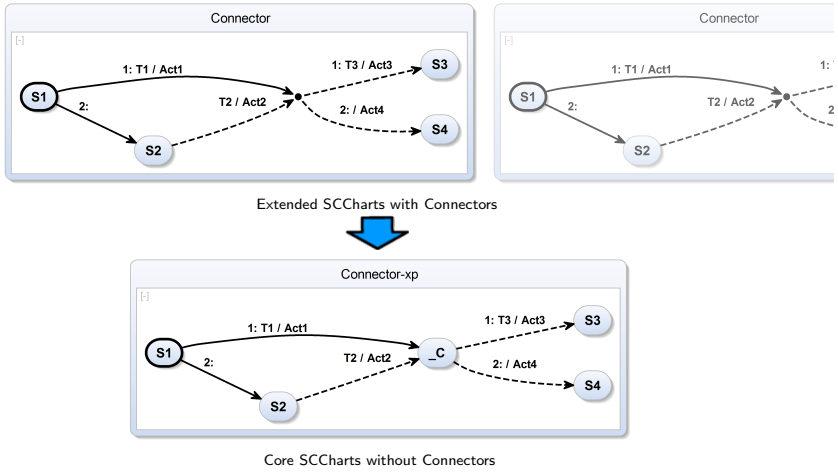
SCCharts — Core & Extended Features



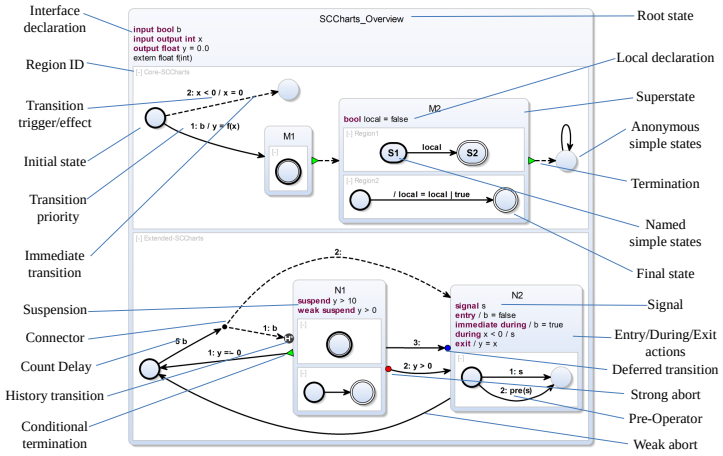
Overview

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

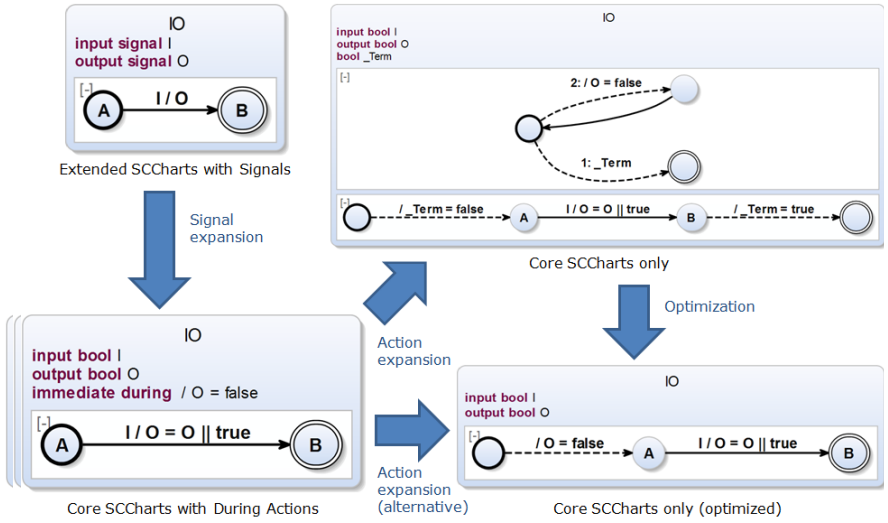
Transforming Connectors



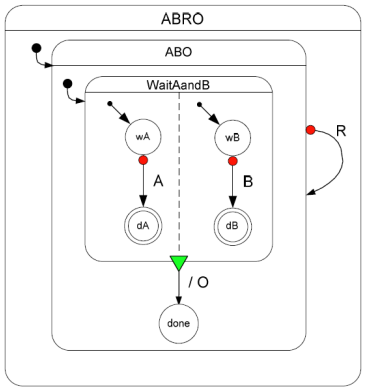
SCCharts — Core Transformations Examples



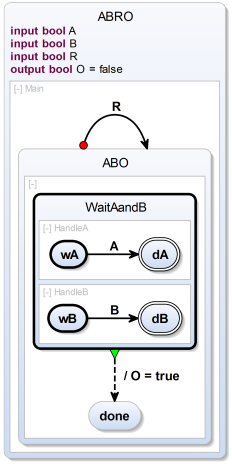
Transforming Signals



SyncChart and SCChart ABRO

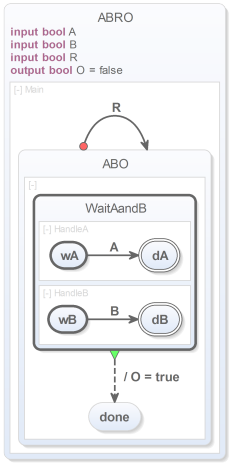


[Charles André, Semantics of SyncCharts, 2003]

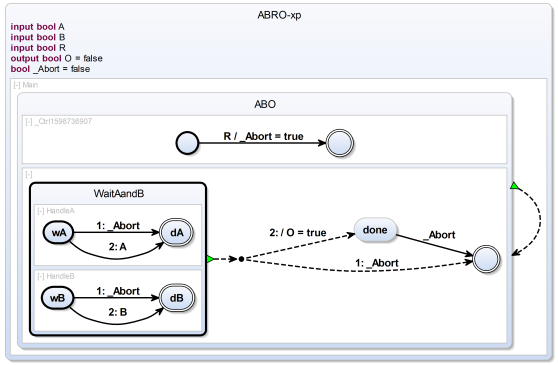


ABRO SCChart

ABRO — Transforming Strong Aborts (cont'd)

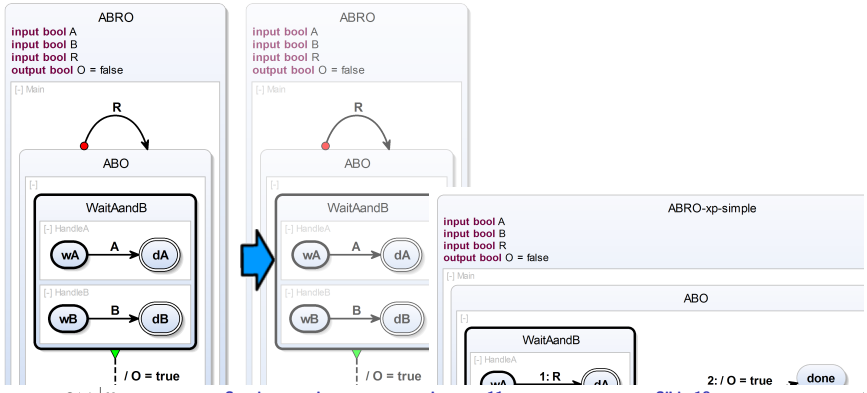


ABRO SCChart with Strong Abort

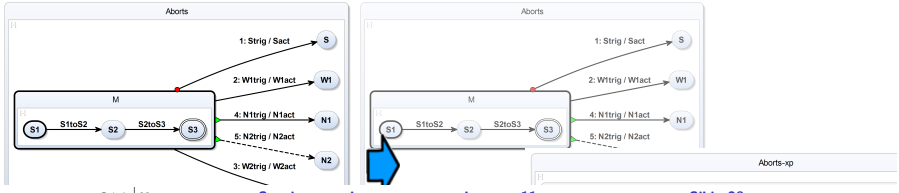


Core SCChart without Strong Abort and WTO

ABRO — Transforming Strong Aborts



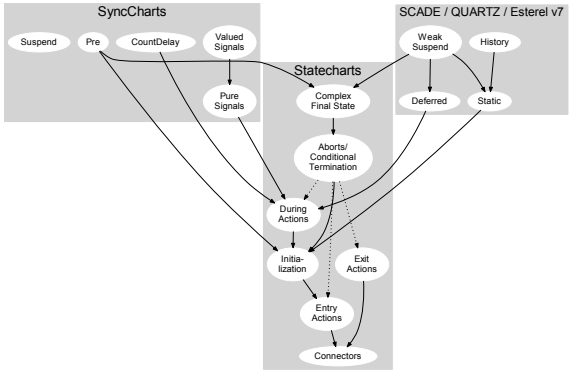
Transforming General Aborts



Overview

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

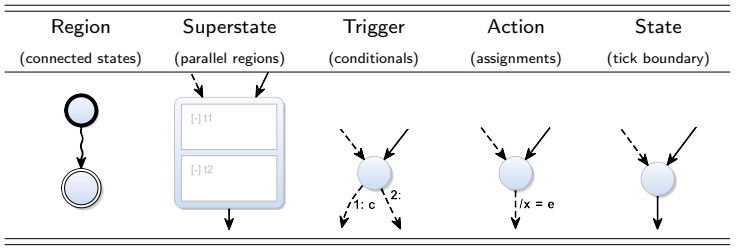
Single-Pass Language-Driven Incremental Compilation (SLIC)



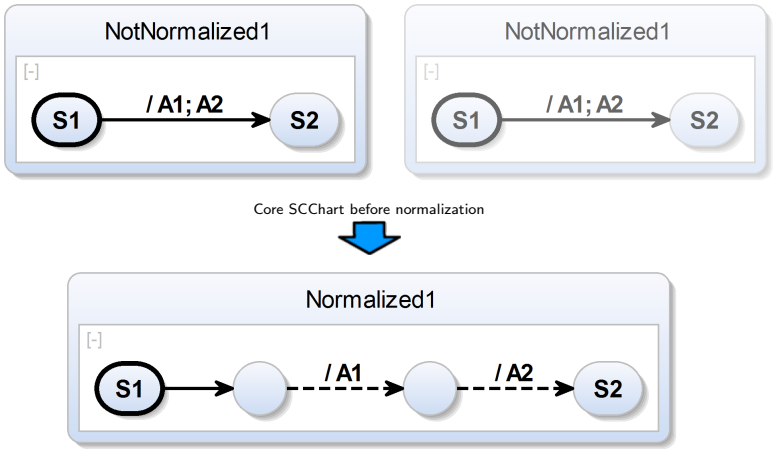
- Some core transformations will produce (use) some other extended features (solid lines)
- Other core transformations cannot handle some extended features (dashed lines)
- → Order in which core transformations are applied is important
- → Dependencies (do not have any cycle, which would be forbidden)

Normalization

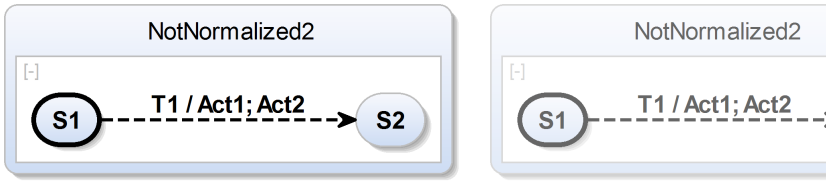
- Further simplify compilation process for Core SCCharts
- Allowed patterns:



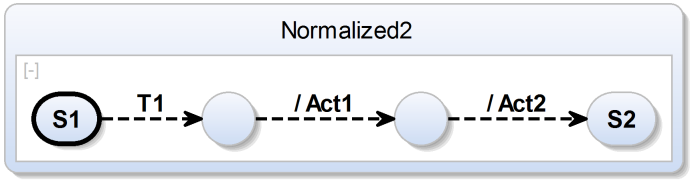
Actions Normalization



Actions Normalization (cont'd)

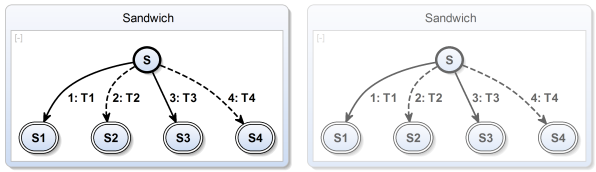


Core SCChart before normalization



Core SCChart after normalization

Trigger Normalization

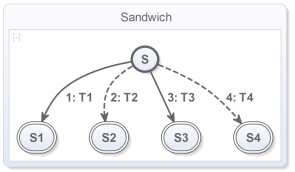


Core SCChart before normalization

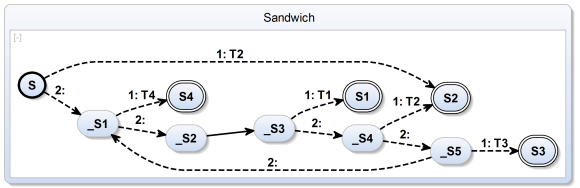
Actions Normalization Implementation Example

```
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 (Cont'd)

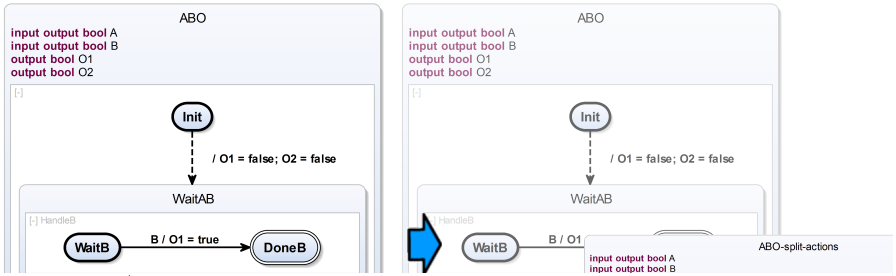


Core SCChart before normalization



Core SCChart after optimized normalization

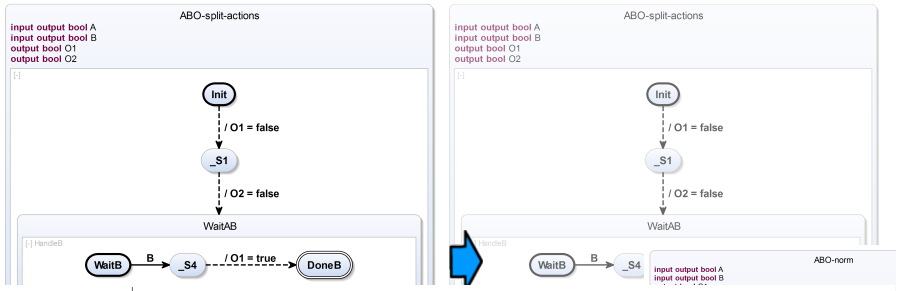
ABO — Normalization Example (Actions)



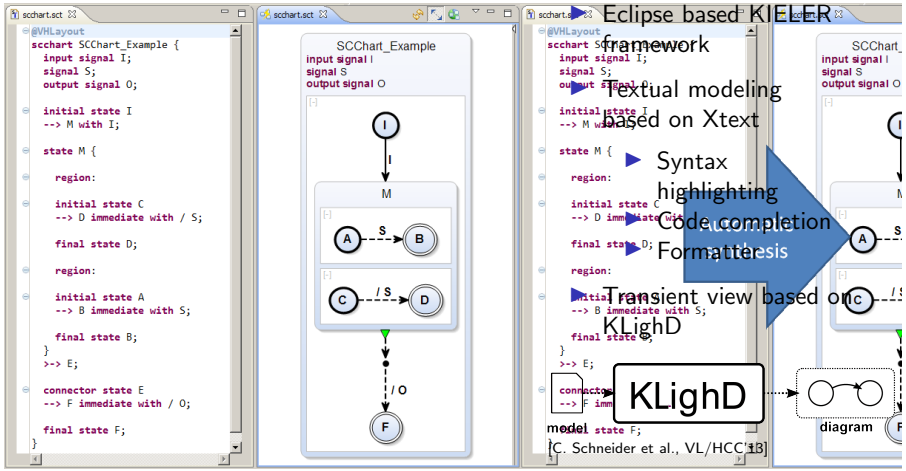
Overview

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

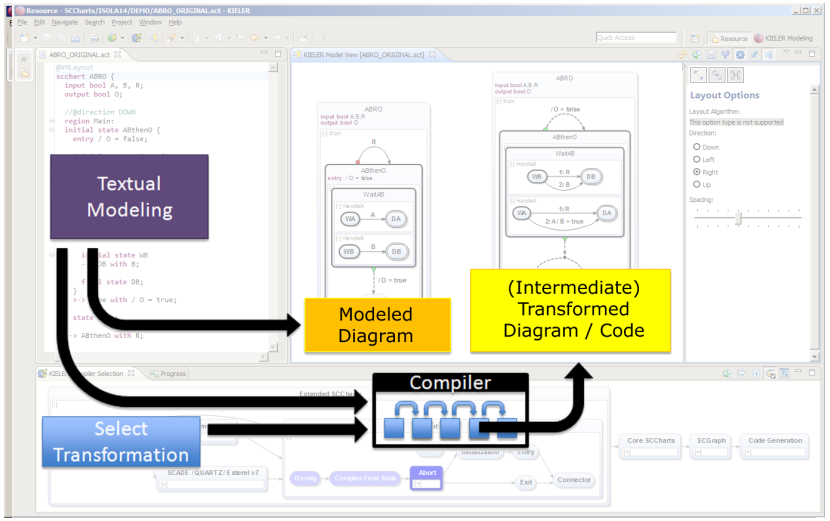
ABO — Normalization Example (Actions & Trigger)



Textual Modeling with KLighD



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. `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`