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 11, 2016, 13:23 hrs

SCCharts — Sequentially
Constructive Statecharts for
Safety-Critical Applications

SyncCharts

Statechart dialect for specifying
deterministic & robust concurrency

SyncCharts:
- Hierarchy, Concurrency, Broadcast
- Synchrony Hypothesis
  1. Discrete ticks
  2. Computations: Zero time

Causality in SyncCharts

Embedded systems react to inputs
with computed outputs
Typically state based computations
Computations often exploit concurrency → Threads
Causality in SyncCharts (cont’d)

```c
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)

SCCharts Overview
Extended SCCharts → Core SCCharts
Normalizing Core SCCharts & Implementation
Overview
Features
Core Transformations
SCCharts — Features
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

**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
Overview

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

SCCharts — Core Transformations Examples

Transforming Connectors

Transforming Signals
SyncChart and SCChart ABRO

ABRO — Transforming Strong Aborts (cont’d)

Transforming General Aborts
Overview

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

Normalization

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

<table>
<thead>
<tr>
<th>Region</th>
<th>Superstate</th>
<th>Trigger</th>
<th>Action</th>
<th>State</th>
</tr>
</thead>
<tbody>
<tr>
<td>(connected states)</td>
<td>(parallel regions)</td>
<td>(conditionals)</td>
<td>(assignments)</td>
<td>(tick boundary)</td>
</tr>
</tbody>
</table>

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)
Actions Normalization (cont’d)

```
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 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 (Cont’d)
SCCharts Overview

Extended SCCharts → Core SCCharts

Normalizing Core SCCharts & Implementation

Compilation / Normalization

Modelling SCharts

Conclusion

Overview

SCCharts Overview

Extended SCCharts → Core SCCharts

Normalizing Core SCCharts

Implementation in KIELER

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 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

To Go Further

- DFG-funded PRETSY Project: www.pretsy.org