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
SyncCharts

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

```
concurrent_causality

signal x
signal y

[ - ]
S1 !x / y S2

[ - ]
S3 !y / x S4
```

```
concurrent_causality

signal x

[ - ]
```
Causality in SyncCharts (cont’d)

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 $\equiv$
  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 $O_2$
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
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

- Interface declaration
- Region ID
- Transition trigger/effect
- Initial state
- Transition priority
- Immediate transition
- Suspension
- Connector
- Count Delay
- History transition
- Conditional termination
- Final state
- Connector
- 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
Transforming Connectors

Extended SCCharts with Connectors

Core SCCharts without Connectors
Transforming Signals

Extended SCCharts with Signals

Signal expansion

Core SCCharts with During Actions

Action expansion

Core SCCharts only (optimized)

Action expansion (alternative)
SyncChart and SCChart ABRO

[Charles André, Semantics of SyncCharts, 2003]
ABRO — Transforming Strong Aborts
ABRO — Transforming Strong Aborts (cont’d)

ABRO SCChart with Strong Abort

Core SCChart without Strong Abort and WTO
SCCharts Overview
Extended SCCharts → Core SCCharts
Normalizing Core SCCharts & Implementation

Transforming General Aborts
Overview

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

<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>

- Normalization further simplifies the compilation process for Core SCCharts.

Allowed patterns include:
- Region
- Superstate
- Trigger
- Action
- State

Diagram showing state transitions and actions.
Actions Normalization

Core SCChart before normalization

Core SCChart after normalization
Actions Normalization (cont’d)

Core SCChart before normalization

Core SCChart after normalization
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
Trigger Normalization (Cont’d)
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

- 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

Textual Modeling → Modeled Diagram → (Intermediate) Transformed Diagram / Code → Compiler

Select Transformation
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