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

- 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

E. A. Lee, The Problem with Threads, 2006
SyncCharts

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

Charles André, Semantics of SyncCharts, 2003

[Christian Motika]

Sequentially Constructive Charts (SCCharts)
Causality in SyncCharts

```
concurrent_causality

signal x
signal y

S1 ─── !x / y ─── S2

S3 ─── !y / x ─── S4
```
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)**

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

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

- SCCharts $\cong$ SyncCharts syntax + Seqeutially 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
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

- Interface declaration
- Region ID
- Transition trigger/effect
- Initial state
- Transition priority
- Immediate transition
- Suspension
- Connector
- Count Delay
- History transition
- Conditional termination
- Final state
- Root state
- Local declaration
- Superstate
- Anonymous simple states
- Termination
- Named simple states
- Signal
- Entry/During/Exit actions
- Deferred transition
- Strong abort
- Pre-Operator
- Weak abort

Christian Motika

Sequentially Constructive Charts (SCCharts)
Transforming Connectors

Extended SCCharts with Connectors

Core SCCharts without Connectors
Transforming Signals

Extended SCCharts with Signals

Core SCCharts with During Actions

Signal expansion

Action expansion

Optimization

Core SCCharts only (optimized)
SyncChart and SCChart ABRO

Charles André, Semantics of SyncCharts, 2003
ABRO - Transforming Strong Aborts

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
Overview

- SCCharts Overview
- Extended SCCharts → Core SCCharts
- Normalizing Core SCCharts
- Implementation in KIELER
- Demo
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)</td>
<td>(parallel)</td>
<td>(conditionals)</td>
<td>(assignments)</td>
<td>(tick boundary)</td>
</tr>
</tbody>
</table>

Diagram showing allowed patterns:

- Region:
  - Connected states

- Superstate:
  - Parallel regions

- Trigger:
  - Conditional

- Action:
  - Assignments

- State:
  - 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

```java
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**

- **Init**: / O1 = false; O2 = false
- **WaitAB**
  - **WaitB**: B / O1 = true → DoneB
  - **WaitA**: A / B = true; O1 = true → DoneA
  - **GotAB**: / O1 = false; O2 = true

**ABO-split-actions**

- **Init**: / O1 = false
- **S1**: / O2 = false
- **WaitAB**
  - **S4**: / O1 = true → DoneB
  - **S3**: / O1 = true → DoneA
  - **S2**: / B = true
  - **S5**: / O2 = true
  - **GotAB**
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

Automatic synthesis

Christian Motika
Sequentially Constructive Charts (SCCharts)
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 :-)

SyncCharts are a great choice for specifying deterministic control-flow behavior... but does not accept sequentiality:

```java
If (!done) { ... ; done = true; }
```
To Go Further

**CHARLES ANDRÉ.**

**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.

**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: \( x = \text{false} \)
  - Relative writes: \( x = x \mid \text{true} \)
  - 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

```java
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

- Sequentially Constructive (S)
- Logically Correct (L)
- Pnueli-Shalev Constructive (P)
- Berry Constructive (B)
- Acyclic SC (A)
- Speculate on absence
- Sequences of values
- Ineffective writes
- Cycle of concurrent dependencies or concurrent writes
- Out-of-order scheduling
- All Programs

Diagram:

0. Synchronous Program Classes
1. Sequentially Constructive (S)
2. Logically Correct (L)
3. Pnueli-Shalev Constructive (P)
4. Berry Constructive (B)
5. Acyclic SC (A)
6. Speculate on absence
7. Sequences of values
8. Ineffective writes
9. Cycle of concurrent dependencies or concurrent writes
10. Out-of-order scheduling
11. All Programs

Christian Motika
Sequentially Constructive Charts (SCCharts)