Sequential Constructiveness, SCL and SCCharts

Incorporating synchrony in conventional languages

Reinhard von Hanxleden (U Kiel)

14 March 2018, Collège de France
Reactive Systems
1980s: Statecharts

[David Harel,  
Statecharts: A Visual Formalism for Complex Systems,  
Science of Computer Programming, 1987]
1990s: Many Statecharts

[Michael von der Beeck,
A Comparison of Statecharts Variants,
Formal Techniques in Real-Time and Fault-Tolerant Systems, LNCS 863, 1994]
1991: Argos

1995: SyncCharts, a.k.a. Safe State Machines

![SyncCharts Diagram]

SCCharts – Motivation
Limitations of Strict Synchrony

```c
if (!x) {
    x = true;
}
```

Good C
Limitations of Strict Synchrony

- Bad SyncChart
- Good SCChart
Limitations of Strict Synchrony

```plaintext
present x else
  emit x
end
```

Bad Esterel

Good SCEst
SCCharts – Motivation

Preserve nice properties of synchronous programming

- Determinacy

- Sound semantic basis

- Efficient synthesis

Reduce the pain

- Make it easy to adapt for mainstream programmer

- Reject only models where determinacy is compromised
Model of Computation
Sequential Constructiveness

Sequential control flow overrides „write before read“

Writes visible only to reads that are

1. sequential successors or

2. concurrent

[v. Hanxleden, Mendler, et al.,
*Sequentially Constructive Concurrency—A Conservative Extension of the Synchronous Model of Computation*,
ACM TECS ’14]
SCCharts – MoC

It’s all about scheduling variable accesses within reaction ...

• Sequential accesses to $x$: unconstrained

• Concurrent accesses to $x$ ("iur protocol"):
  
  \[
  \text{init} \quad \Rightarrow \quad \text{updates} \quad \Rightarrow \quad \text{reads}
  \]
  
  $x = 1 \quad \ldots \quad x += 2 \quad \ldots \quad x += 5 \quad \ldots \quad y = x \quad \ldots \quad z = x$

• Concurrent accesses may lead to causality cycles – compiler must reject those
SCCharts – MoC

SCChart / SCEst program is ...

... **SC**, „is sequentially constructive,“ if

1. there exist runs obeying iur protocol

2. all such runs produce same result

... **SCC**, „corresponds to sequentially constructive circuit,“ if it is SC and does not „speculate“
Program Classes

module XY:
[
    present X then emit Y end
||
    present Y then emit X end
]

module XYesle:
[
    present X then emit Y end
||
    present Y else emit X end
]

module Dynamic:
loop
    emit S;
    present I then pause end;
    present S then emit T end;
    present I else pause end end

module ABBA:
[
    present A and I
    then emit B end
||
    present B and not I
    then emit A end
||
    present A or B
    then emit O end
]

module OffOn:
present S then emit T end;
emit S;
present S then emit U end
SCCharts – The Language(s)
Interface
declaration

input bool A, B, R
output bool O = false

Initialization
Interface declaration

input bool A, B, R
output bool O = false

Initialization

Superstate

ABthenO
input bool A, B, R
output bool O = false

ABthenO

WaitAandB

[-] HandleA

[-] HandleB
Interface declaration

**input bool** A,B,R
**output bool** O = false

Initialization

Superstate

Region

Initial state

```
[-] HandleA
  wA

[-] HandleB
  wB
```
input bool A, B, R
output bool O = false

ABRO

[ ]

ABthenO

[ ]

WaitAandB

[ ] HandleA

wA  dA

[ ] HandleB

wB  dB

Interface declaration
Initialization
Superstate
Region
Initial state
Final state
input bool A, B, R
output bool O = false

Interface declaration

Initialization

Superstate

Region

Initial state

Final state

Delayed Transition (+ Trigger)
Interface declaration

input bool A, B, R
output bool O = false

Initialization

Superstate

Region

Initial state

Final state

Delayed Transition (+ Trigger)

Immediate transition (+ Effect)
input bool A, B, R
output bool O = false

Interface declaration

Initialization

Superstate

Region

Initial state

Final state

Delayed Transition (+ Trigger)

Immediate transition (+ Effect)

Strong abort

ABRO

WaitAandB

[-] HandleA

wA \rightarrow dA

[-] HandleB

wB \rightarrow dB

\neg O = true \rightarrow done

R
SCChart Building Blocks

<table>
<thead>
<tr>
<th>Region</th>
<th>Trigger</th>
<th>Effect</th>
<th>Superstate</th>
<th>State</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image" alt="Region" /></td>
<td><img src="image" alt="Trigger" /></td>
<td><img src="image" alt="Effect" /></td>
<td><img src="image" alt="Superstate" /></td>
<td><img src="image" alt="State" /></td>
</tr>
</tbody>
</table>

**Normalized Core SCCharts**

1: c

/x = e
<table>
<thead>
<tr>
<th>Region</th>
<th>Trigger</th>
<th>Effect</th>
<th>Superstate</th>
<th>State</th>
</tr>
</thead>
<tbody>
<tr>
<td>Normalized Core SCCharts</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### M2M Mappings

<table>
<thead>
<tr>
<th>Thread</th>
<th>Conditional</th>
<th>Assignment</th>
<th>Concurrency</th>
<th>Delay</th>
</tr>
</thead>
<tbody>
<tr>
<td>SCL</td>
<td>t</td>
<td>if (c) s₁ else s₂</td>
<td>x = e</td>
<td>pause</td>
</tr>
<tr>
<td>SCG</td>
<td>entry</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>exit</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>fork t₁ par t₂ join</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Some Syntactic Sugar: Core SCCharts

- Interface declaration
- Region ID
- Transition trigger/effect
- Initial state
- Immediate transition
- Transition priority
- Root state
- Local declaration
- Superstate
- Anonymous simple state
- Termination
- Named simple state
- Final state

### Normalized Core SCCharts

<table>
<thead>
<tr>
<th>Region</th>
<th>Trigger</th>
<th>Effect</th>
<th>Superstate</th>
<th>State</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image" alt="Normalized Core SCCharts" /></td>
<td><img src="image" alt="Trigger" /></td>
<td><img src="image" alt="Effect" /></td>
<td><img src="image" alt="Superstate" /></td>
<td><img src="image" alt="State" /></td>
</tr>
</tbody>
</table>
More Syntactic Sugar: Extended SCCharts
Compilation: Expand Signals
Compilation: Expand Signals
Expand During Action
Core SCChart
Normalized Core SCChart
Normalized Core SCChart
Taking Stock

SCCharts defined by M2M Transformations

- Extended SCCharts
- Core SCCharts
- Normalized Core SCCharts
- SCL/SCG
SCCharts – Pragmatics
Key to Pragmatics: MVC

• A **model** represents knowledge

• A **view** is a (visual) representation of its model. It would ordinarily highlight certain attributes of the model and suppress others. It is thus acting as a **presentation filter**.

• A **controller** is the link between a user and the system. It provides the user with input by arranging for relevant views to present themselves in appropriate places on the screen.

[Trygve Reenskaug, Models – Views – Controllers, Xerox PARC technical note, 1979]
Textual Modeling

SCChart **model** specified in `.sctx`

- Efficient editing
- Facilitates model comparison
- Easy revision control

Graphical **view** automatically synthesized with KIELER (**controller**)

- Customizable view
- Saves developer time
SCCharts
http://www.sccharts.com/

KIELER
The Key to Efficient Modeling
http://www.rtsys.informatik.uni-kiel.de/en/research/kieler

ELK
Eclipse Layout Kernel
https://www.eclipse.org/elk/

All open source, under EPL license
View Synthesis
Adding Region “HandleC”
Back to Original ABRO
Graphics-to-Text Navigation: Select Region

scchart ABRO {
  input signal A, B, R
  output signal 0

  initial state ABthen0 {
    initial state WaitAandB {
      region "HandleA" {
        initial state wA
        if A go to dA

        final state dA
      }

      region "HandleB" {
        initial state wB
        if B go to dB

        final state dB
      }

      join to done do 0

      final state done
    }

    if R abort to ABthen0
  }
}

input signal A, B, R
output signal 0
Graphics-to-Text Navigation: Select Transition
View Filtering: Hide Declarations
View Filtering: Collapse Regions
Direction VH (Vertical-Horizontal)
Change Layout: Direction HV
Change Layout: Direction Down
Change Layout: Direction Right
Simulation – Initialization
Simulation – Tick 1
Simulation – Tick 1
Simulation – Tick 2
Simulation – Tick 3
Simulation – Tick 3
Simulation – Tick 4
Simulation – Tick 5
Simulation – Tick 5
Simulation – Tick 6
SCCharts – Classroom-Tested
SCCharts – Classroom-Tested

- 10,000 / 135,000 SCChart nodes before/after normalization
- 650,000 lines of C-code
- Compiles in about 2 min’s
- 2 ms reaction time
SCCharts – Modeling Dataflow
Dataflow View in ABRO
Dataflow View in ABRO
Dataflow View in ABRO
Dataflow View, with Communication

[Wechselberg, Schulz-Rosengarten, Smyth, von Hanxleden
Augmenting State Models with Data Flow
Principles of Modeling – LNCS Festschrift on Edward Lee's 60th Birthday (to appear)]
[Wechselberg, Schulz-Rosengarten, Smyth, von Hanxleden
Augmenting State Models with Data Flow
Principles of Modeling – LNCS Festschrift on Edward Lee's 60th Birthday (to appear)]]
Dataflow View, with Communication

[Wechselberg, Schulz-Rosengarten, Smyth, von Hanxleden
Augmenting State Models with Data Flow
Principles of Modeling – LNCS Festschrift on Edward Lee's 60th Birthday (to appear)]
Dataflow View, with Communication

[Wechselberg, Schulz-Rosengarten, Smyth, von Hanxleden
Augmenting State Models with Data Flow
Principles of Modeling – LNCS Festschrift on Edward Lee's 60th Birthday (to appear)]
let node chrono (StSt, Rst) = (disp_1, disp_2) where
  automaton
  CHRONO ->
  do automaton
  STOP ->
  do s = 0 -> last s
  and m = 0 -> last m
  and run = false
  unless StSt continue START
  | START ->
  let d = 0 -> (pre d + 1) mod 100 in
  do s = if d < pre d
  then (last s + 1) mod 60
  else last s
  and m = if s < last s
  then (last m + 1) mod 60
  else last m
  and run = true
  unless StSt continue STOP
  end
  until Rst and not run then CHRONO
end and
automaton
  TIME ->
  do disp_1 = s
  and disp_2 = m
  until Rst and run then LAP
  | LAP ->
  do until Rst then TIME
end

Figure 2: A Chronometer

[Colaço, Pagano, Pouzet,
A Conservative Extension of Synchronous Dataflow with State Machines,
EMSOFT '05]
node chrono (StSt:bool, Rst:bool)
returns (disp_1:int, disp_2:int);

var s: int;
var m: int;
var run: bool;
var d: int;

let

automaton

CHRONO ->
  automaton
    STOP ->
    s = 0 -> pre s;
    m = 0 -> pre m;
    run = false;
  unless StSt continue START;
  I START ->
    d = 0 -> (pre d + 1) mod 100;
    s = if d < pre d
        then (pre s + 1) mod 60
        else pre s;
    m = if s < pre s
        then (pre m + 1) mod 60
        else pre m;
    run = true;
  unless StSt continue STOP;
end;
until Rst and not run then CHRONO;
end;

automaton

TIME ->
  disp_1 = s;
  disp_2 = m;
  until Rst and run then LAP;
  I LAP ->
  until Rst then TIME;
end;
node chrono (StSt:bool, Rst:bool)
returns (disp_1:int, disp_2:int);
var s: int;
var m: int;
var run: bool;
var d: int;
let
automaton
  CHRONO ->
    automaton
      STOP ->
        s = 0 -> pre s;
        m = 0 -> pre m;
        run = false;
    unless StSt continue START;
    I START ->
      d = 0 -> (pre d + 1) mod 100;
      s = if d < pre d then (pre s + 1) mod 60 else pre s;
      m = if s < pre s then (pre m + 1) mod 60 else pre m;
      run = true;
    unless StSt continue STOP;
    until Rst and not run then CHRONO;
  end;
automaton
  TIME ->
    disp_1 = s;
    disp_2 = m;
    until Rst and run then LAP;
  I LAP ->
    until Rst then TIME;
end;
Diagram

[Clement Pascutto (ENS), Internship project at Kiel]
From C to SCCharts
Extracting Visual Models from C

int main(int a) {
    int b = 10, c = 6;
    if (a > 4) {
        a = a - 1;
    } else {
        a = c + 3;
    }
    while (a <= b) {
        a = a + 1;
        if (a == c) {
            a = b * 2;
        }
    }
    for (int i = 0; i < b; i = i + 1) {
        a = i * 2;
    }
    return a;
}

[Smyth, Lenga, von Hanxleden, 
Model Extraction of Legacy C Code in SCCharts, ISoLA DS 2016]
int fib(int n) {
    int fl = 0, fh = 1;
    if (n<=1) { fh = n; }
    else {
        for (int i=2; i<=n; i++) {
            int tmp = fh;
            fh += fl;
            fl = tmp;
        }
    }
    return fh;
}
```c
int main(int argc, char** argv) {
    int a, b;
    if (argc>0) {
        a = atoi(argv[0]);
    } else {
        a = 0;
    }
    b = fib(a);
    return b;
}

int fib(int n) {
    int fl = 0, fh = 1;
    if (n<=1) { fh = n; }
    else {
        for (int i=2; i<=n; i++) {
            int tmp = fh;
            fh += fl;
            fl = tmp;
        }
    }
    return fh;
}
```
typedef struct {
    char _GO;
    char g7;
    ...
} TickData1;

void reset1(TickData1 *d) {
    d->pg12 = 0;
    d->GO = 1;
    d->TERM = 0;
}

void tick1(TickData1 *d) {
    tickLogic1(d);
    d->GO = 0;
    d->pg12 = d->g12;
}

void tickLogic1(TickData1 *d) {
    d->g7 = d->GO;
    if (d->g7) {
        d->fl = 0;
        d->fh = 1;
    }
    d->cg7 = d->n <= 1;
    d->g8 = d->g7 && d->cg7;
    if (d->g8) {
        d->fh = d->n;
    }
    d->g13 = d->pg12;
    d->g10 = d->g7 && !d->cg7;
    if (d->g10) {
        d->fib_int_local_i = 2;
    }
    d->g11 = d->g13 || d->g10;
    d->cg11 =
    d->fib_int_local_i = d->n;
    d->g12 = d->g11 && d->cg11;
    if (d->g12) {
        d->fib_int_local_tmp = d->fh;
        d->fh = d->fh + d->fl;
        d->fl = d->fib_int_local_tmp;
        d->fib_int_local_i =
        d->fib_int_local_i + 1;
    }
    d->g9 = d->g11 &&
    !d->cg11 || d->g8;
    if (d->g9) {
        d->ret = d->fh;
        d->TERM = 1;
    }
typedef struct {
    char _GO;
    char g7;
    ...
} TickData1;

void tickLogic1(TickData1 *d) {
    d->g7 = d->_GO;
    if (d->g7) {
        d->fl = 0;
        d->fh = 1;
    }
    d->cg7 = d->n <= 1;
    d->g8 = d->g7 && d->cg7;
    if (d->g8) {
        d->fh = d->n;
    }
    d->g13 = d->pg12;
    d->g10 = d->g7 && ld->cg7;
    if (d->g10) {
        d->fib_int_local_i = 2;
    }
    d->g11 = d->g13 || d->g10;
    d->g12 = d->g11 && d->cg11;
    if (d->g12) {
        d->fib_int_local_tmp = d->fh;
        d->fh = d->fh + d->fl;
        d->fl = d->fib_int_local_tmp;
        d->fib_int_local_i = d->fib_int_local_i + 1;
    }
    d->g9 = d->g11 && ld->cg11 || d->g8;
    if (d->g9) {
        d->ret = d->fh;
        d->TERM = 1;
    }
}

void reset1(TickData1 *d) { 
    d->pg12 = 0;
    d->GO = 1;
    d->TERM = 0;
}
SCCharts Wrap-Up

Language

- 5 core constructs
- Smörgåsboard of extensions

Model of Computation

- Relaxed synchrony
- Still determinate
- Can model C programs

Compilation

- M2M transformations
- Stress-tested in KIELER

Still plenty of things to do: Variants on SC MoC, optimize code generation, pragmatics improvements for schedulability analysis, ...
Code Generation
# Downstream Compilation

So far, two alternative compilation strategies from SCL/SCG to C/VHDL

<table>
<thead>
<tr>
<th></th>
<th>Dataflow</th>
<th>Priority</th>
</tr>
</thead>
<tbody>
<tr>
<td>Accepts instantaneous loops</td>
<td>−</td>
<td>+</td>
</tr>
<tr>
<td>Can synthesize hardware</td>
<td>+</td>
<td>−</td>
</tr>
<tr>
<td>Can synthesize software</td>
<td>+</td>
<td>+</td>
</tr>
<tr>
<td>Size scales well (linear in size of SCChart)</td>
<td>+</td>
<td>+</td>
</tr>
<tr>
<td>Speed scales well (execute only active parts)</td>
<td>−</td>
<td>+</td>
</tr>
<tr>
<td>Instruction-cache friendly (good locality)</td>
<td>+</td>
<td>−</td>
</tr>
<tr>
<td>Pipeline friendly (little/no branching)</td>
<td>+</td>
<td>−</td>
</tr>
<tr>
<td>WCRT predictable (simple control flow)</td>
<td>+</td>
<td>+/−</td>
</tr>
<tr>
<td>Low execution time jitter (simple/fixed flow)</td>
<td>+</td>
<td>−</td>
</tr>
</tbody>
</table>

[von Hanxleden, Duderstadt, Motika, et al.,
SCCharts: Sequentially Constructive Statecharts for Safety-Critical Applications, PLDI'14]
Compilation Option 1: Dataflow
Compilation Option 2: Priority-Based
Compilation Option 2: Priority-Based

```
1 depth
1 O = false
1 surface
```

```
1 entry
1 exit
```

```
0 R
```

```
0 O |= true
0 exit
```

true
int tick() {
    tickstart(7);
    fork1(_region_0, _region_1, 3) {
        _region_0:
        O = 0;
        pause;
        goto _region_0;
    } par {
        _region_1:
        fork1(_region_2, _region_3, 2) {
            _region_2:
            prio(6);
            pause;
            if(R) {
                } else {
                    if(B) {
                        } else {
                            } join2(2, 5);
                            } join2(3, 6);
                            } goto HandleB;
            } par {
                HandleA:
                prio(4);
                pause;
                if(R) {
                    } else {
                        if(A) {
                            } else {
                                } join1(7);
                                } tickreturn();
                            }
                        }
                    }
                }
            }
        }
    }
}
}
SCCharts and Time

- Synchrony Hypothesis: Outputs are synchronous with inputs.
- Computation "does not take time"
- Actual computation time does not influence result
- Sequence of outputs determined by inputs
Synchronous Execution

Initialize Memory
for each input event do
  Compute Outputs
  Update Memory
end

Initialize Memory
for each clock tick do
  Read Inputs
  Compute Outputs
  Update Memory
end

Fig. 1 Two common synchronous execution schemes: event driven (left) and sample driven (right).

[Benveniste et al., *The Synchronous Languages Twelve Years Later*, Proc. IEEE, 2003]
Multiform Notion of Time

Only the simultaneity and precedence of events are considered.

This means that the physical time does not play any special role.

This is called multiform notion of time.

[https://en.wikipedia.org/wiki/Esterel]
Packaging Physical Time as Events

Event "HMS": 100 μsec have passed since last HMS
Event "TMS": 1000 μsec have passed since last TMS

[Timothy Bourke, SYNCHRON 2009]
A Problem With That ...

Fig. 4: Granularity of timing inputs

[Timothy Bourke, SYNCHRON 2009]
Dynamic Ticks

- Recall logical time:

- Physical time, time-triggered:

- Physical time, dynamic ticks:

[von Hanxleden, Bourke, Girault, *Real-Time Ticks for Synchronous Programming*, FDL'17]
/** Controller for stepper motor */

scchart MOTOR {
    output int currentUsec = 0; // [usec] Current simulated time;
        // when deployed, this should be input
    output int wakeUsec;       // [usec] Time for next wake-up

    input bool accel, decel;   // Increase/decrease speed
    input bool stop;          // Emergency stop - sets (angular) speeds to 0

    output bool motor = false; // Motor pulse
    output float v;           // [cm/sec] Robot speed
    output int pMotorUsec;    // [usec] Half period for motor

    int pSetSpeedsMaxUsec = 500000; // [usec] Maximum period of speed control loop
    int pSetSpeedsMinUsec = 400000; // [usec] Minimum period of speed control loop
    output int pUsec;          // [usec] Previous period (delta of wake-up times)
    output int pMinUsec = pSetSpeedsMaxUsec; // [usec] Minimum period

    float dV = 2;             // [cm/sec] Delta v applied during one speed
        // control loop cycle
    float vMax = 20;          // [cm/sec] Max speed of left/right motor
    float cmPerHalfPeriod = 1; // [cm] Distance traveled by motor per half period
        // (duration of true or false)
Now use KEELER to synthesize graphical
SCChart with ELK and simulate...
[Wechselberg, Schulz-Rosengarten, Smyth, von Hanxleden
Augmenting State Models with Data Flow
Principles of Modeling – LNCS Festschrift on Edward Lee's 60th Birthday (to appear)]