SCCD: A Statecharts and Class Diagrams multi-target compiler for a family of Statecharts languages

Joeri Exelmans

21 November 2019

What is SCCD?

- Modeling language combining **S**tate**C**harts and **C**lass **D**iagrams
- Original version by Glenn De Jonghe (2013)
- Extended with semantic options during my Research Internship II (2014-2015)
- Performance improvements by Simon Van Mierlo (2015 – 2017)

An SCCD model is:

- A class diagram (set of classes and associations between them, w/ multiplicities)
- Each class in said diagram has its behavior defined with a statechart

The SCCD project consists of

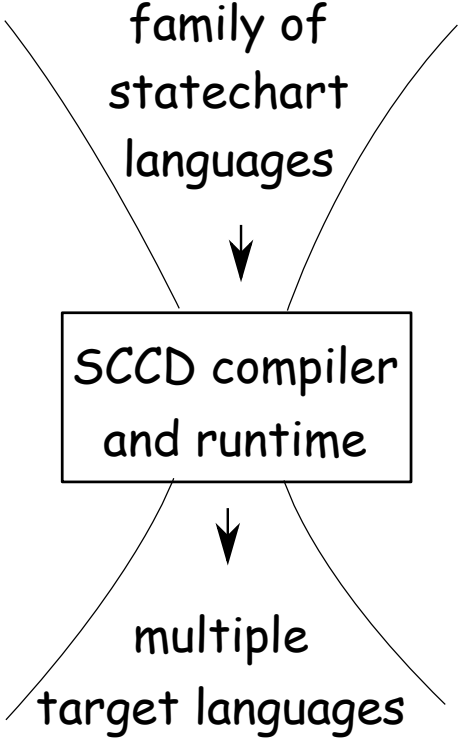
- **Compiler**: takes a SCCD model (in XML) and produces code in Python or JavaScript
- **Runtime**: mostly logic for "stepping" each object, instantiating new objects & (very simple) conformance checks. Also implemented in Python and JavaScript.
- A bunch of black-box **tests**
  (given sequence of inputs → expected sequence of outputs?)

There are many "statechart" variants, and they all differ slightly when it comes to semantics

- David Harel's original paper (1987)
- Standards
  - SCXML, W3 consortium (2015) (already many implementations...)
  - PSSM, part of UML, OMG (May 2019)
- Commercial products:
  - Statemate (1986)
  - Rhapsody (1996)
  - Stateflow (2004)
  - YAKINDU (2008)

SCCD aims to support...

- As input: a **family** of statechart languages
- As output:
    - Multiple target languages, currently:
        - Python
        - JavaScript
    - Multiple types of platforms:
        - Thread (the runtime runs an event loop in its own thread of execution) (if supported in target language)
        - Event loop (for integration with an existing event loop, e.g. in GUI toolkits or in JavaScript: the runtime is called from such eventloop, but may also schedule future events in that eventloop)
        - Game loop (the runtime only runs when requested to "advance logical time by X")



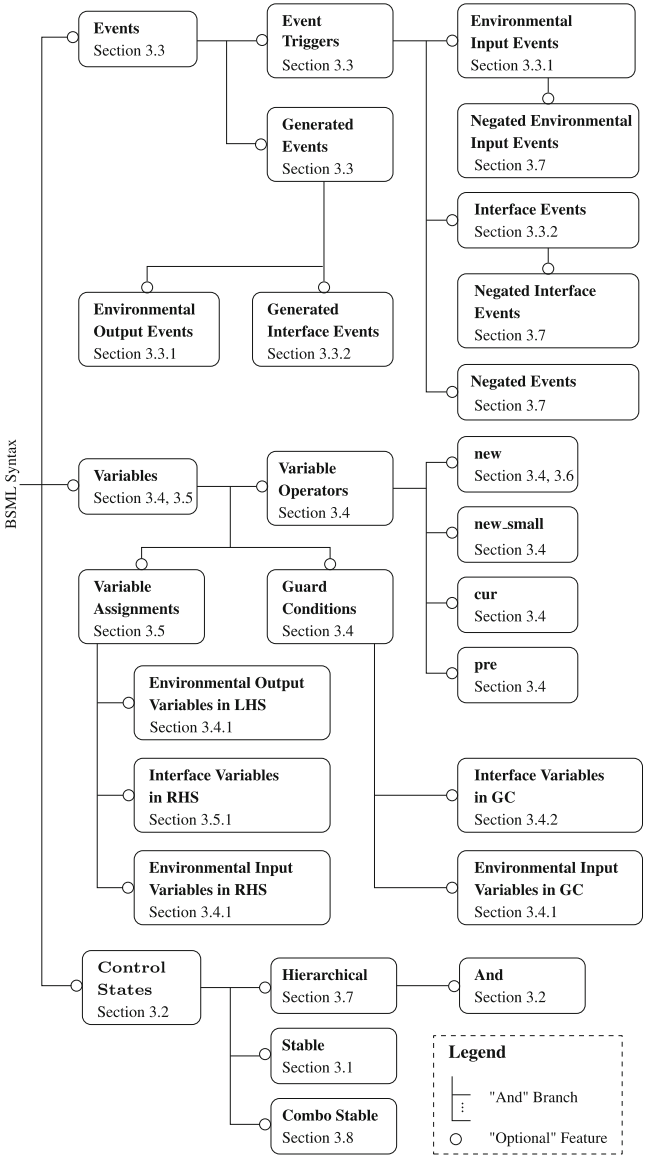
Semantic variation of Statecharts

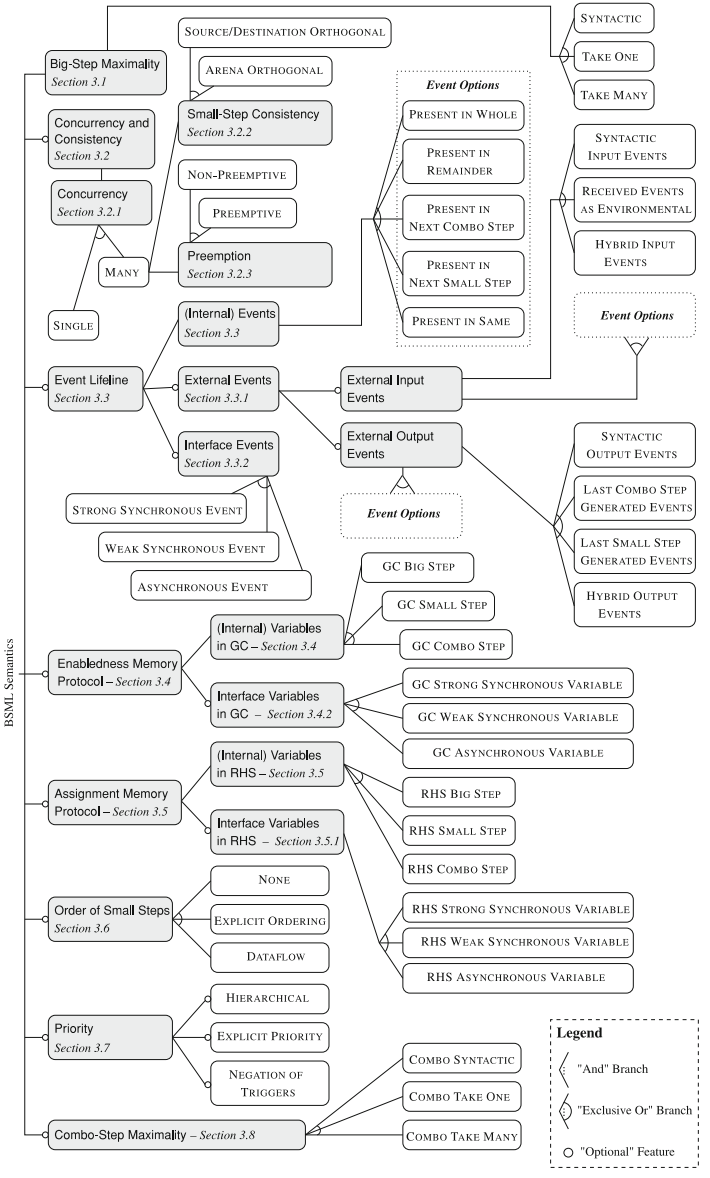
Super cool 2010 paper *"Deconstructing the semantics of big-step modelling languages"* by Nancy Day & Joanne Atlee

= A study/comparison of many languages that can be mapped onto a **common statechart syntax**, such that only the semantics differ on a limited number (8) of variation points defined in the paper

Languages compared all intend to model **reactive & interactive systems**, including:

- Statechart-like (Harel Statecharts, Statemate, Rhapsody, ...)
- Synchronous programming languages (Esterel, Argos, ...)


BSML Syntax
Events
Section 3.3
Event Triggers
Section 3.3
Generated Events
Section 3.3
Environmental Input Events
Section 3.3.1
Negated Environmental Input Events
Section 3.7
Interface Events
Section 3.3.2
Negated Interface Events
Section 3.7
Negated Events
Section 3.7
Environmental Output Events
Section 3.3.1
Generated Interface Events
Section 3.3.2
Variables
Section 3.4, 3.5
Variable Operators
Section 3.4
new
Section 3.4, 3.6
new_small
Section 3.4
cur
Section 3.4
pre
Section 3.4
Variable Assignments
Section 3.5
Guard Conditions
Section 3.4
Environmental Output Variables in LHS
Section 3.4.1
Interface Variables in RHS
Section 3.5.1
Environmental Input Variables in RHS
Section 3.4.1
Interface Variables in GC
Section 3.4.2
Environmental Input Variables in GC
Section 3.4.1
Control States
Section 3.2
Hierarchical
Section 3.7
And
Section 3.2
Stable
Section 3.1
Combo Stable
Section 3.8
Legend
"And" Branch
"Optional" Feature


BSML Semantics
Source/Destination Orthogonal
Arena Orthogonal
Small-Step Consistency
Section 3.2.2
Non-Preemptive
Preemptive
Preemption
Section 3.2.3
Many
Single
Big-Step Maximality
Section 3.1
Concurrency and Consistency
Section 3.2
Concurrency
Section 3.2.1
Event Options
Present in Whole
Present in Remainder
Present in Next Combo Step
Present in Next Small Step
Present in Same
Syntactic
Take One
Take Many
Syntactic Input Events
Received Events as Environmental
Hybrid Input Events
Event Options
(Internal) Events
Section 3.3
Event Lifeline
Section 3.3
External Events
Section 3.3.1
External Input Events
Interface Events
Section 3.3.2
External Output Events
Event Options
Syntactic Output Events
Last Combo Step Generated Events
Last Small Step Generated Events
Hybrid Output Events
Strong Synchronous Event
Weak Synchronous Event
Asynchronous Event
Enabledness Memory Protocol – Section 3.4
(Internal) Variables in GC – Section 3.4
Interface Variables in GC – Section 3.4.2
GC Big Step
GC Small Step
GC Combo Step
GC Strong Synchronous Variable
GC Weak Synchronous Variable
GC Asynchronous Variable
Assignment Memory Protocol – Section 3.5
(Internal) Variables in RHS – Section 3.5
Interface Variables in RHS – Section 3.5.1
RHS Big Step
RHS Small Step
RHS Combo Step
RHS Strong Synchronous Variable
RHS Weak Synchronous Variable
RHS Asynchronous Variable
Order of Small Steps
Section 3.6
None
Explicit Ordering
Dataflow
Priority
Section 3.7
Hierarchical
Explicit Priority
Negation of Triggers
Combo-Step Maximality – Section 3.8
Combo Syntactic
Combo Take One
Combo Take Many
Legend
"And" Branch
"Exclusive Or" Branch
"Optional" Feature

Example BSMLs and their semantic options

Semantic Aspects	Semantic Options	[21]	[42]	[30]	[19]	[6]	[33]	[22]	[3]
Big-Step Maximality	Syntactic					4			
	Take One	4	4				4	4	4
	Take Many			4	4				
Concurrency	Single	4	4	4	4			4	
	Many					4	4		4
Small-Step Consistency	Source/Destination Orthogonal								
	Arena Orthogonal					4	4		4
Preemption	Non-Preemptive					4	4		
	Preemptive								
(Internal) Event Lifeline	Present in Whole					4	4		
	Present in Remainder	4	4						
	Present in Next Combo Step			4	4				
	Present in Next Small Step								
	Present in Same								
Environmental Input Events	Syntactic Input Events			4		4	4		
	Received Events as Environmental	4	4		4				
	Hybrid Input Event								
(Interface) Event Lifeline	Strong Synchronous Event								
	Weak Synchronous Event								
	Asynchronous Event					4			
(Internal Variables) Enabledness Memory Protocol	GC/RHS Big Step							4	4
	GC/RHS Combo Step				4				
	GC/RHS Small Step		4	4		4		4	
(Interface Variables) Memory Protocol	GC/RHS Strong Synchronous Variable								4
	GC/RHS Weak Synchronous Variable								
	GC/RHS Asynchronous Variable								
Combo-Step Maximality	Combo Syntactic								
	Combo Take One			4	4				
	Combo Take Many								
Order of Small Steps	None	4		4	4	4	4		
	Explicit Ordering								
	Dataflow		4					4	4
Priority	Hierarchical				4				
	Explicit Priority								
	Negation of Triggers		4	4	4	4	4	4	4

[21]: Harel statecharts, [42]: Pnueli and Shalev statecharts, [30]: RSML, [19]: Statemate, [6]: Esterel, [33]: Argos, [22]: SCR, and [3]: reactive modules

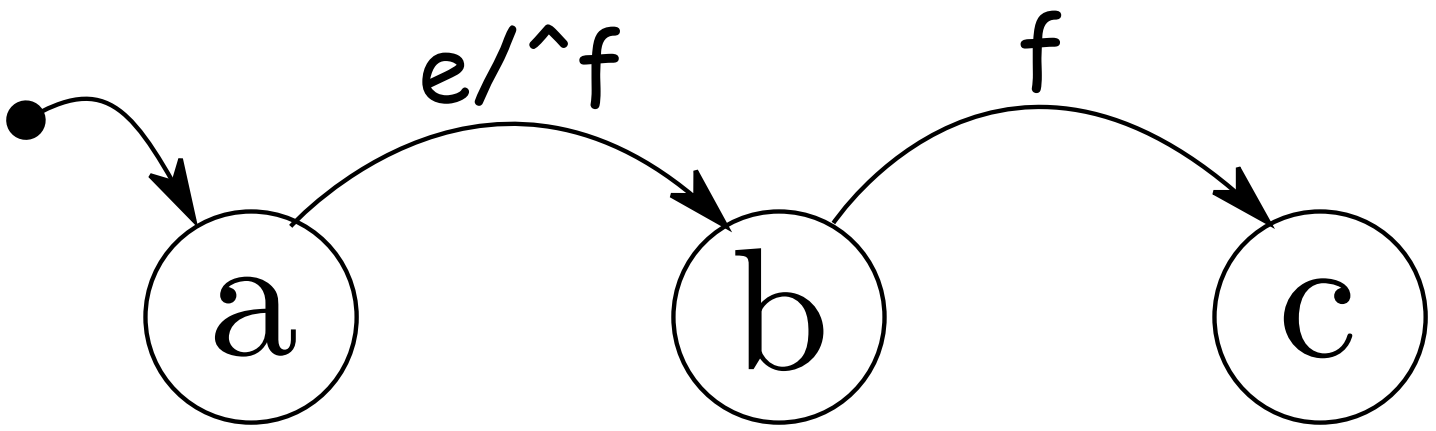
Notion of a "Big Step"

- The execution of a model is a sequence of Big Steps
- A Big Step takes input from the environment, and produces output

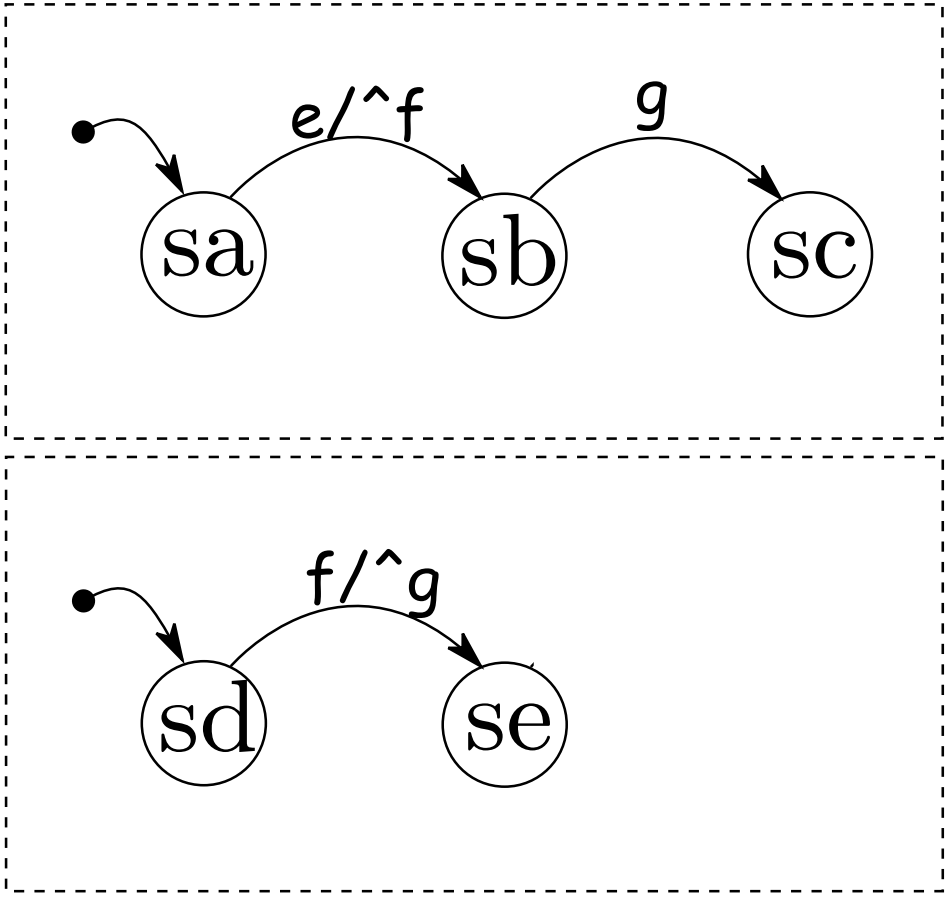
  → Within a Big Step, there's no interaction with the envorinment
- Within a Big Step, multiple transitions **may** occur
- When modeling a reactive system, a Big Step takes 0 logical time to execute

  = "synchrony hypothesis"

Example of a semantic option: When does a Big Step end?



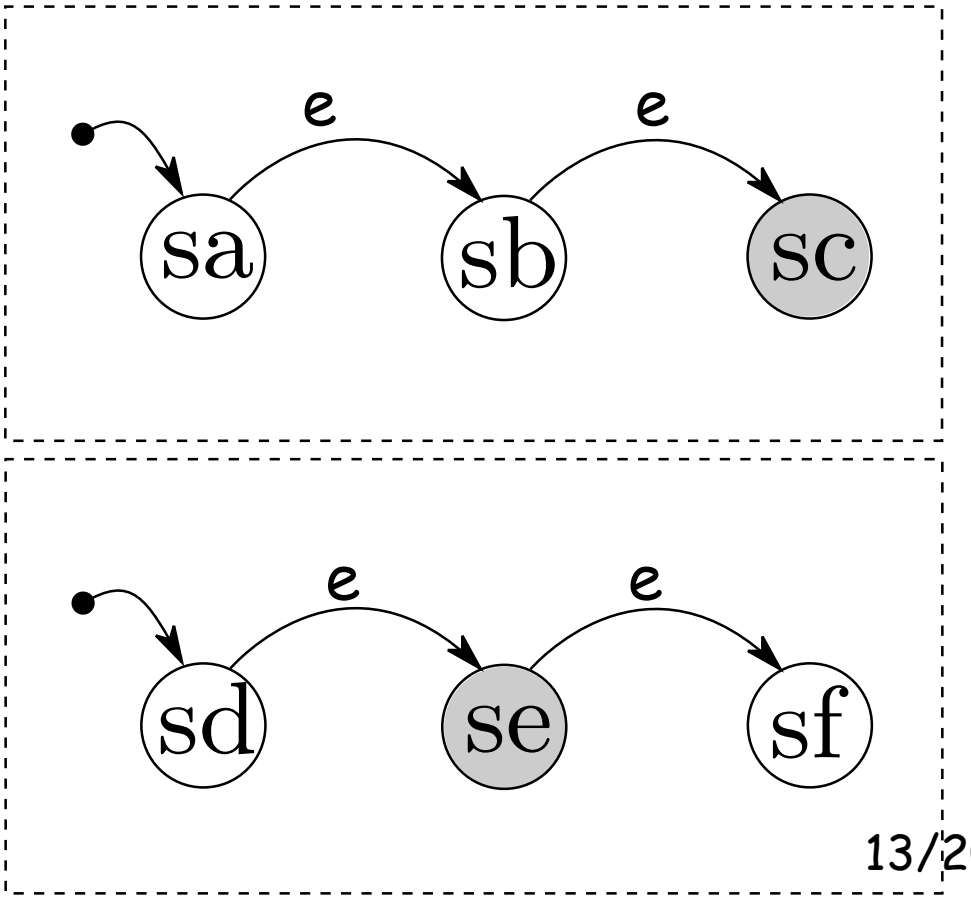
What about this one?



Another possibility: Stable states

- Extend the syntax with notation for "stable state"

- Big Step ends when the entire model is in a stable configuration



Examples of more semantic options

- **Priority**: if multiple transitions can occur, which one to choose? (deterministically)
- Are **internal events** treated differently from input events?
- When do we evaluate our **guard conditions**? (Only at the beginning of a big step? After each transition?)
- Can multiple transitions in orthogonal components occur **concurrently**? (= logical concurrency, meaning: there is no ordering between the transitions)

Master thesis

- Improve SCCD: Offer maximal support for the options in Day & Atlee's paper
- Research:
  - Can we achieve compatibility with new standards (2015: SCXML, 2019: PSSM) as a semantic configuration?
  - **What are useful combinations of semantic options? Are certain combinations useless?**

Combinations of semantic options

- Semantic options and additional constraints from Day & Atlee modeled in **Clafer** (= language for variability) yields millions of combinations
- Can we prune this search space further?

The "class diagrams" part of SCCD

- Not really the focus of my thesis, but still interesting
- To build large complex (possibly distributed) systems, need runtime instantiation/destruction of objects (each with a statechart)
- Use class diagrams to model runtime constraints (multiplicities etc.)
- Possible source of inspiration: Erlang

Erlang

"Success story":

- Developed at Ericsson to solve a real problem: *Making reliable distributed systems in the presence of errors* (also the title of Joe Armstrong's 2003 thesis)
- 1986 – 1991: Grown from a Prolog dialect to a real language
- 1998: First released product (ATM switch) with 1,7 million lines of Erlang code was very reliable
- 1998: Open-sourced

Erlang

- An Erlang system is
  - A collection of processes
  - Async best-effort communication between processes (→ **non-determinism**!)
  - Processes can start new processes
  - Fail-fast error handling: Processes allowed to "just crash"
  - Remote error handling: Crash detected by other process(es)
  - Hot code (re)loading: Update parts of a running system without stopping it

= Implementation of the actor model (Gul Agha '85)

(even though the people behind Erlang had not heard of the actor model)

Other tasks, "TODO":

- Add a neutral action language to SCCD (possibly reuse work that went into HUTN)
- Currently JavaScript runtime is being neglected → bring it up-to-date with Python version