Embedded Systems Design with Statecharts

Gerald Lüttgen

Department of Computer Science

University of York, U.K.

www.cs.york.ac.uk/~luettgen

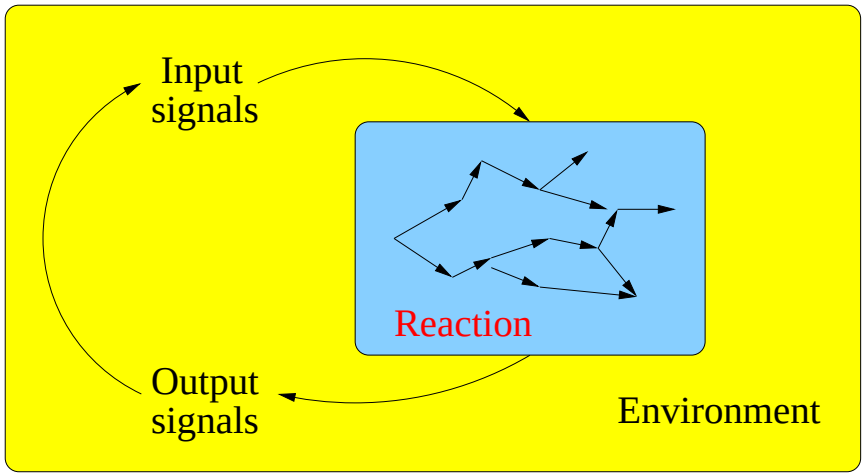
National Institute of Aerospace, Hampton, Virginia, May 2006

Reactive Systems

Are characterized by their ongoing interaction with their environment; they constantly read in statuses of sensor signals and compute statuses of actuator signals

Are at the heart of embedded systems and include controllers in cars, aircraft, telecommunications, medical devices, etc.

Cyclic executive: Cycle–based reaction



- Read statuses of input signals from the environment
- Determine reaction in the form of emitting further signals
- Output signal statuses to the environment

What is Statecharts?

- Statecharts is a visual language for designing and programming reactive, embedded systems.
- Statecharts extends finite state machines by four concepts:
  - Hierarchy (nested state)
  - Concurrency (orthogonal states)
  - Communication (broadcasting of events/signals)
  - Priority and preemption (via negated events, abortion and suspension)
- Statecharts has many applications for embedded systems design, most notably in the automotive and avionics industries.

  Many major avionics companies use Statecharts–centered design tools, including Airbus, EADS, Honeywell and Lockheed Martin.

A Brief History of Statecharts

- Early 1980's: David Harel invents Statecharts in Israel, intended for specifying embedded software for fighter jets.
- At the same time: Gérard Berry develops the textual language Esterel in France.
- 1987: I-Logix is founded in the U.S. and releases the first commercial Statecharts tool, called Statemate.
- Mid 1990's: Charles André develops SyncCharts in France, a purely synchronous variant of Statecharts and a visual front–end for Esterel.
- 1998: MathWorks releases Stateflow for their Matlab/Simulink toolbox.
- 1999: Esterel Technologies is founded in France and releases the design suite Esterel Studio which integrates Esterel and SyncCharts ("Safe State Machines").
- 2001: Esterel Technologies acquires SCADE from Telelogic and integrates Safe State Machines into SCADE.

The SCADE Design Environment

Context of modern embedded systems:

- Control systems, e.g., flight guidance systems, often have modes.
- Each mode is a control law, naturally described as data flow equations.
- A controller switches between modes, and is most naturally described as a finite state machine.

SCADE is a tool supporting the design and programming of such systems:

- A block diagram language for describing operations on data flows, with specialized blocks for describing Safe State Machines
- A simulator and verifier (via SAT solving) for checking design correctness
- An ANSI C and Ada code generator which is DO-178B qualified

Data Flow Equations

Are well adapted to steady process control applications and signal processing

Example: Dynamical system of equations with input signal U, output signal X (with $X_0 = 0$) and internal state variable S (with $S_0 = 1$).

$$\begin{aligned} X_{t+1} &= U_{t+1} * \mathit{sin}(X_t + S_{t+1} - S_t) \\ S_{t+1} &= \mathit{cos}(S_t + U_{t+1}) \end{aligned}$$

In Lustre: The language behind SCADE for describing and computing flows.

```
node Control (U : float) returns (X : float);
var S : float;
let X = 0. -> (U  * sin(pre(X) + S - pre(S)));
    S = 1. -> cos(pre(S) + U)
tel
```

In this example, all signals are sampled at the same clock rate (implicit synchrony).

In general, different flows can be computed at different rates.

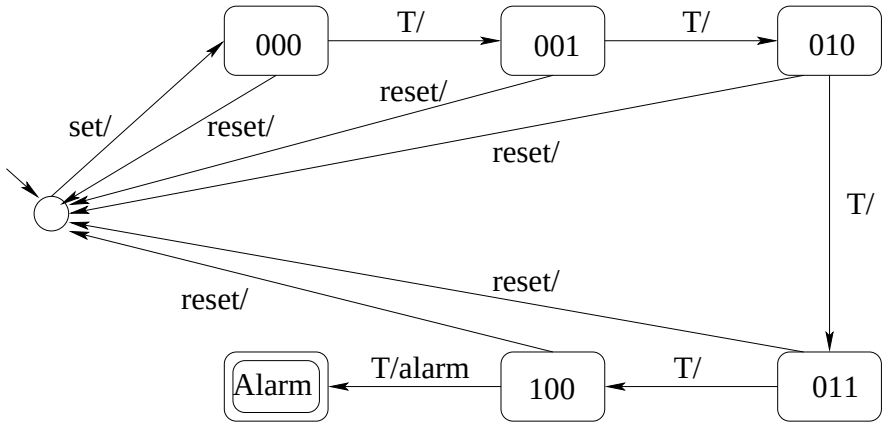
(Example taken from [vHan2004])

Finite State Machines

Are well adapted to modeling discrete controllers

Are well understood by Computer Scientists, Software Engineers and domain experts alike

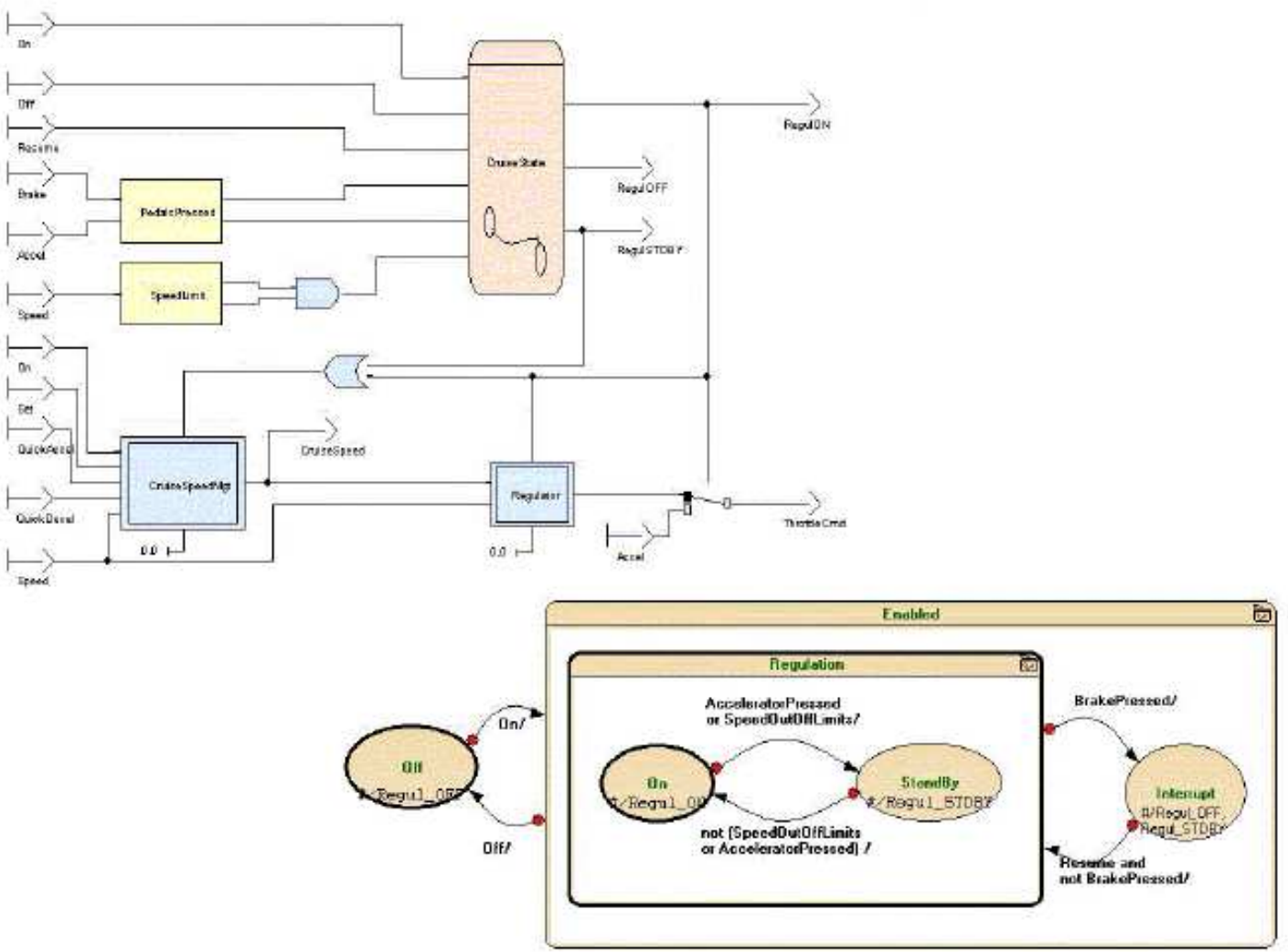
Example in Mealy–automaton notation:



"Raise alarm after 5 clock ticks, if it has been set"

This flat notation does not support the Write Things Once principle, whence the need for a better notation: Statecharts.

A Cruise Control in SCADE v4



(Example taken from [ColPagPou2005])

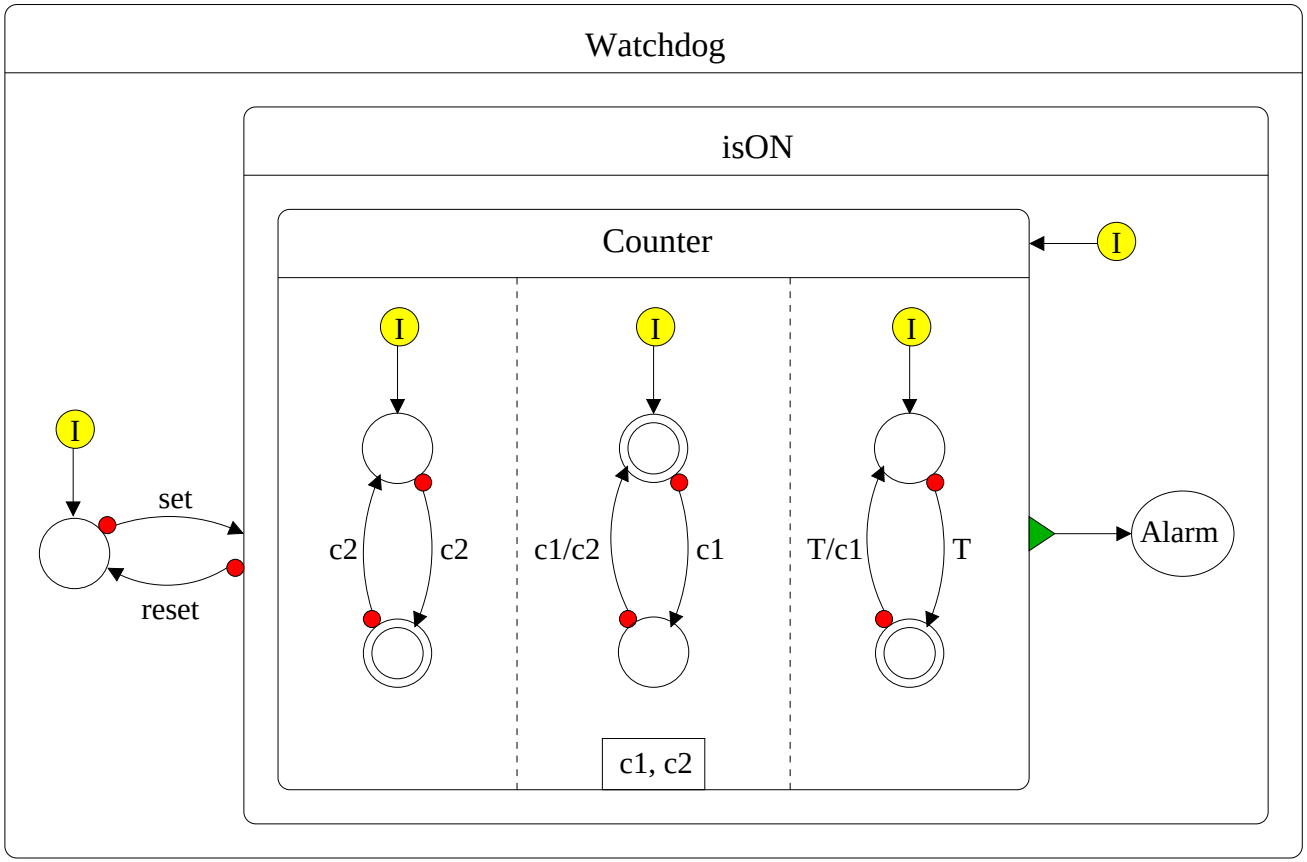
Common Features of Most Statecharts Dialects

- States, including initial and terminal states
- State hierarchy, i.e., nesting of states
- Signals, which may carry data values (signals are also called events)
- Transitions labeled with triggers (e.g., presence of signals and data conditions) and actions (e.g., emitting signals)
- Concurrency, i.e., states and activities executing in parallel
- Modularity, i.e., composition of complex Statecharts from simpler Statecharts
- Base language interface, such as to C or Ada (e.g., for programming state activities and handling data)

This talk focuses on the Statecharts dialect Safe State Machines.

Safe State Machines – Watchdog Example

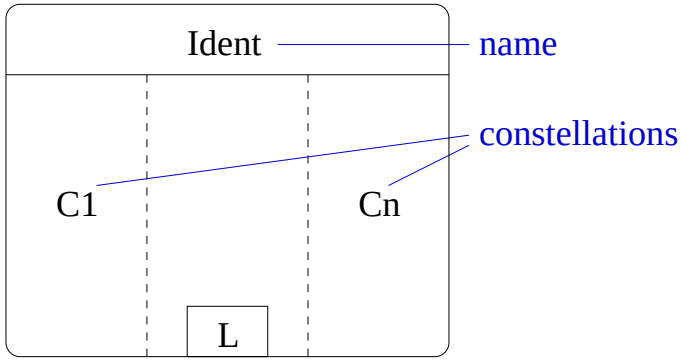
Signal **set** activates the **Counter** which counts the occurrences of signal **T** from 0. If **Counter** reaches 5, then the **Alarm** state is entered. At any time, **reset** disables the alarm.



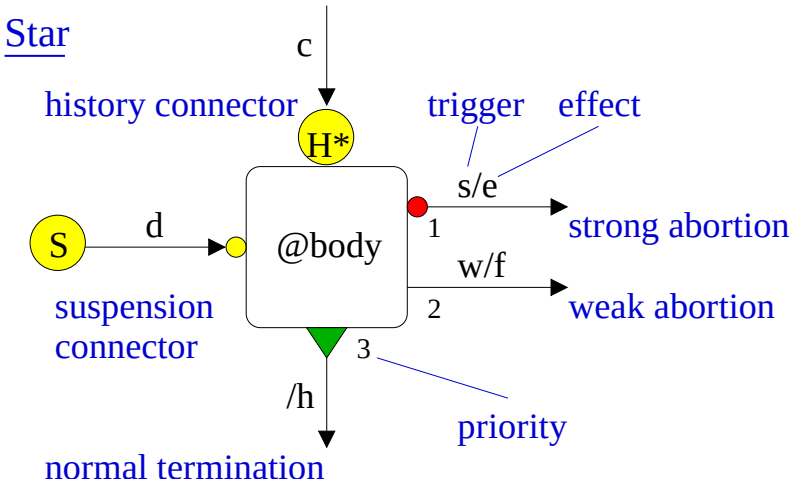
(Example taken from [And96])

Safe State Machines – Notation

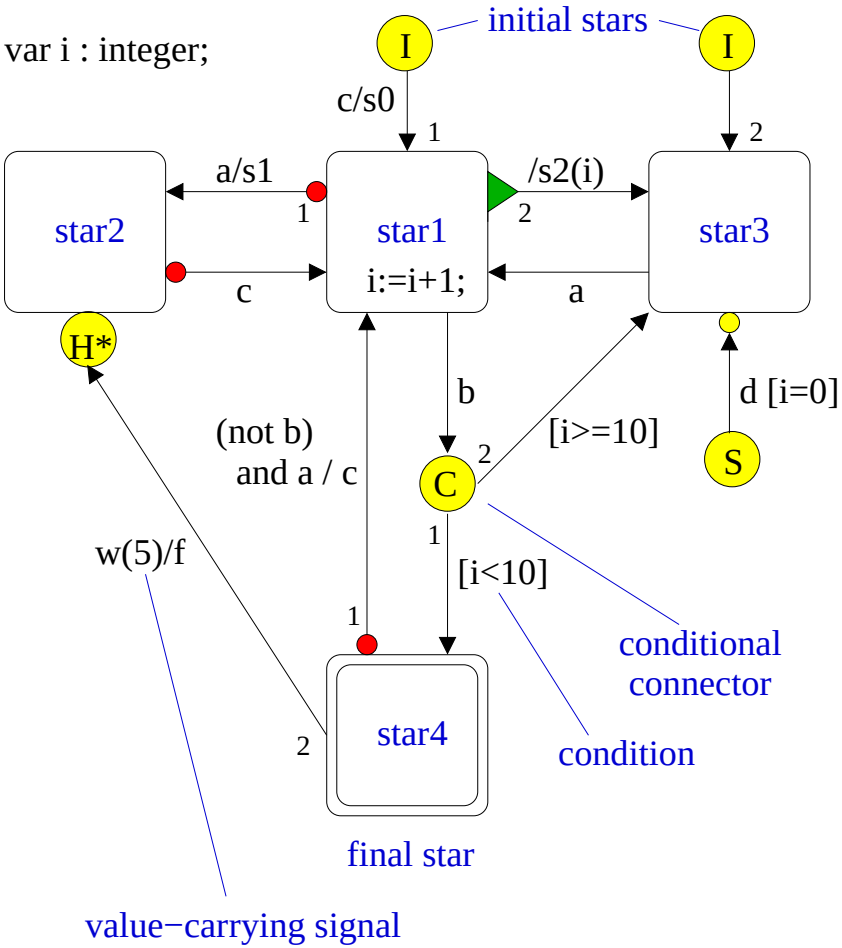
Firmament



Star



Constellation



(Diagrams adapted from [And96])

Safe State Machines – Features

Syntax features:

- No interlevel transitions (i.e., no spaghetti programming via transitions crossing state boundaries)
- Signal scoping (i.e., ability to define certain signals local to a state)
- Explicit priority and suspension constructs

Semantics features:

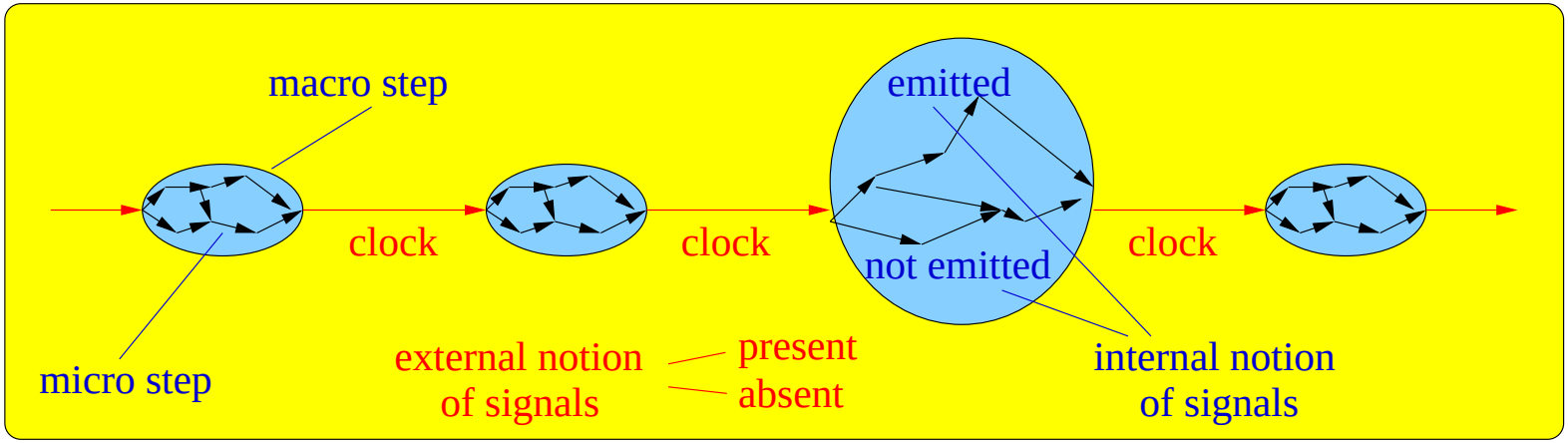
- Synchrony
- Reactiveness & determinism
- Causality & global consistency

Synchrony

The system is much faster than its environment: it reacts in “zero” time!

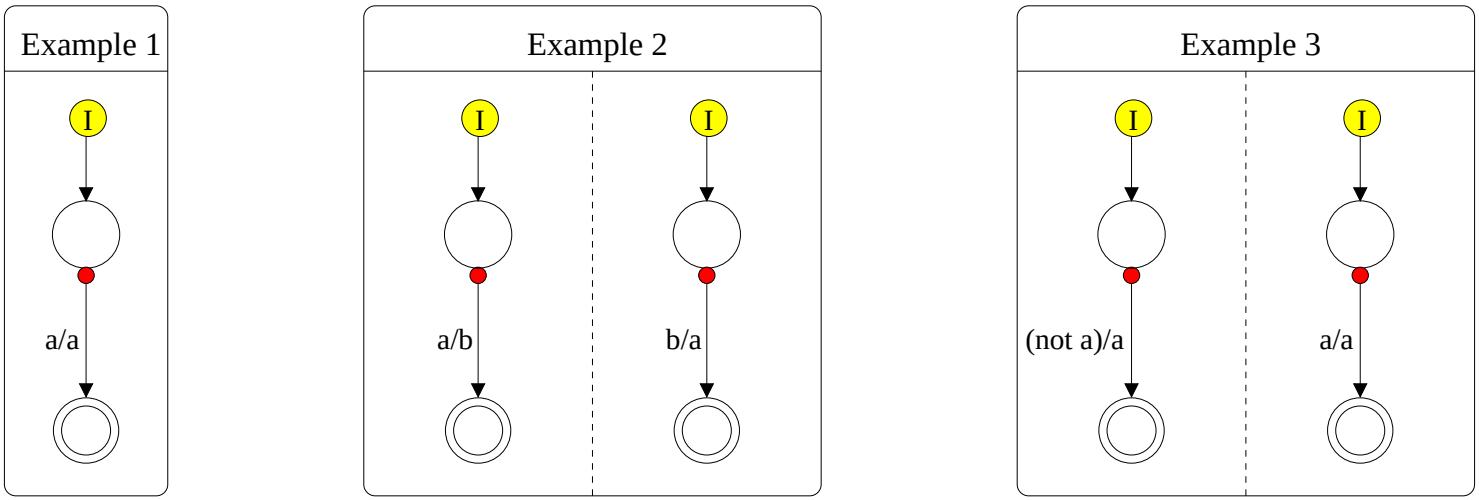
- External (environment) view: Reactions are atomic.
- Internal (system) view: Reactions are non–atomic.

Micro– & Macro–Steps: Abstraction imposed by the synchrony hypothesis



Reactiveness & Determinism

A signal is present within a reaction if and only if it is emitted in this reaction (or if it is asserted by the environment).



- Examples 1 and 2 have two coherent solutions, i.e., the SSMs are reactive but not deterministic.
- Example 3 has a single coherent solution, but ...

Causality & Global Consistency

Causality:

There shall be a causal justification of why signals are or are not emitted, which can be traced back to the statuses of the input signals as provided by the system environment.

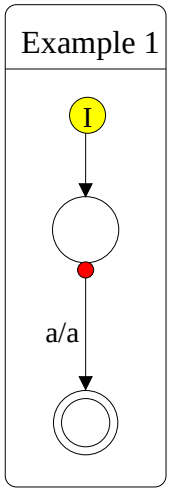
I.e., a rooted chain reaction of triggering and emitting signals.

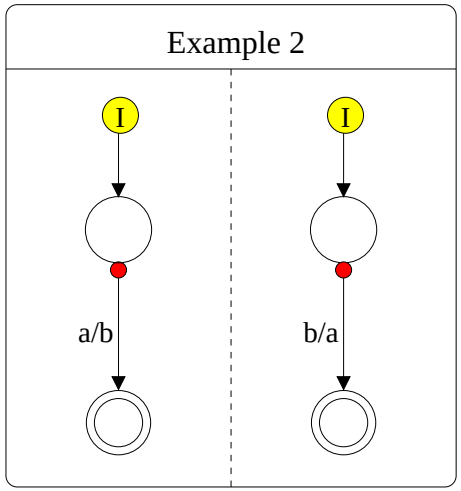
Global consistency:

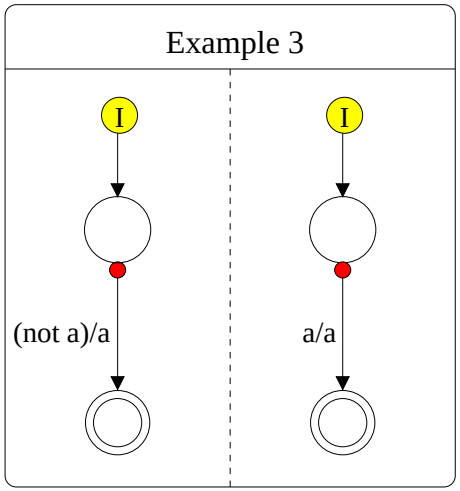
A signal cannot be both present and absent within a given reaction.

I.e., the abstraction made by the synchrony hypothesis is consistent.

Our Pathological Examples Revisited



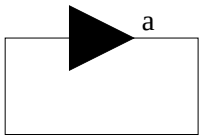


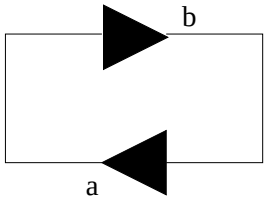


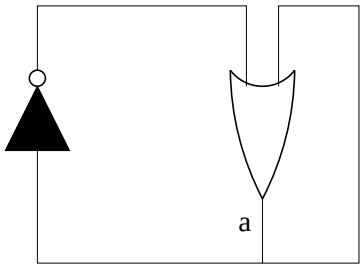
violates causality

violates causality

violates global consistency







Safe State Machines & Circuit Semantics

SSMs are a graphical front–end to the Esterel language; the Esterel compiler rejects non–constructive programs.

Accepted programs correspond one–to–one to stable asynchronous circuits with feedback, i.e., such circuits electrically stabilize for all gate and wire delays, even if the circuits have combinational cycles.

Advantages of the circuit semantics:

- Avoids the state explosion problem which occurs when translating Esterel programs into explicit automata
- Can be directly implemented in hardware
- Can be simulated for software implementation

Statecharts Semantics – Variations

Different Statecharts dialects often employ similar notation but vary significantly in their semantics!

Some important points of departure:

- When may emitted signals be consumed?

  For example, in Simulink/Stateflow or RoseRT, emitted signals may only be consumed in the next macro step/reaction.

- Are negated events allowed?

  UML–based dialects of Statecharts do not support negated events.

- Is negation interpreted constructively?

  For example, in Statemate, transitions with negated triggers may be fired speculatively, which potentially introduces nondeterminism.

Other Statecharts Dialects – Statemate

Statemate:

- Was the first commercial Statecharts tool on the market, developed by I-Logix
- Implements the “original” version of Statecharts, as invented by David Harel
- Is not fully synchronous in the sense of SSMs, but is based on a micro–step semantics instead
- Does not have a constructive semantics regarding negated events, and thus allows nondeterminism to creep into models
- Comes with powerful tools for simulation, verification and code generation
- Was hugely successful in the automotive industry in the 1990’s, but has since been marginalized by Simulink/Stateflow

Other Statecharts Dialects – Simulink/Stateflow

Simulink:

- Is a graphical extension to MathWork's Matlab toolbox
- Simulates designs of dynamic systems
- Has a data–flow look–and–feel, similar to the original SCADE

Stateflow:

- Is a special module embedded in Simulink to model state–based control
- Has a lot of features but a poor semantics (e.g., 12 o'clock rule)
- May be interfaced to the Real–Time Workshop package or the popular dSpace tool for generating code (including C and ADA code)
- Uses a fundamentally different code generation for simulation and target code

Other Statecharts Dialects – Rose RealTime

RoseRT:

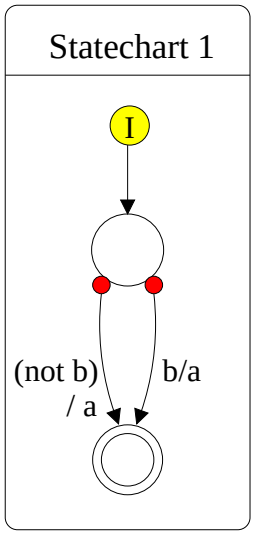
- Is a UML–based tool developed by Rational
- Allows programming via class diagrams, state diagrams and some handwritten code in the target language (e.g., Java, C++, Ada)
- Does not support parallelism, unless the designer writes code for sending and receiving messages
- Can be used for project presentation and documentation, too (as it supports use cases, sequence diagrams, etc.)
- Supports simulation very well, but generates code that is sometimes difficult to understand

Outlook: Achieving Compositionality

Existing Statecharts dialects that employ the synchrony hypothesis, causality and global consistency do not support compositionality!

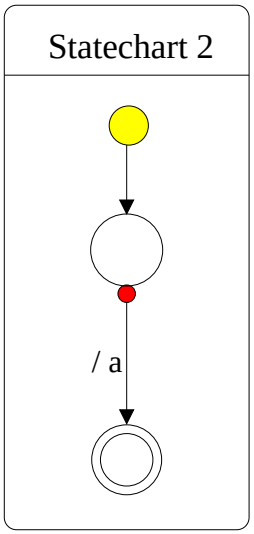
This means that incremental development and compilation are hindered.

Example:



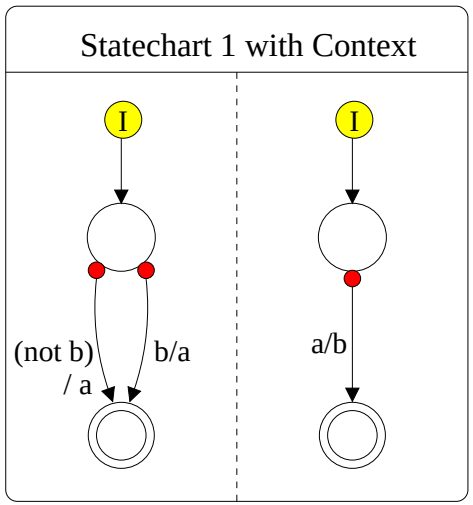
Equivalent reactions:

Input		Reaction
{}	⟼	{a}
{a}	⟼	{a}
{b}	⟼	{a, b}
{a, b}	⟼	{a, b}



Achieving Compositionality (cont'd)

Placing our example in a context:



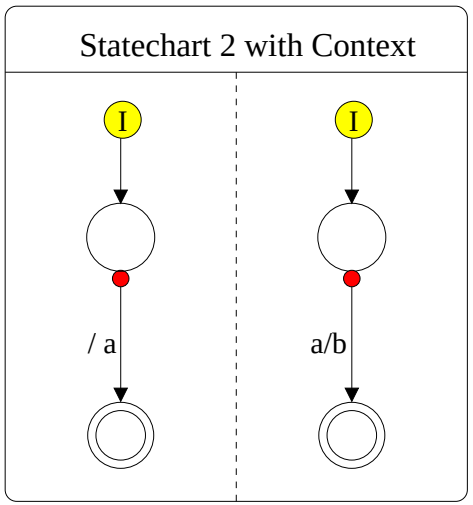
Left:

{} ⟼ "fail"

Violates global consistency!

Right:

{} ⟼ {a, b}



⟹ The Law of the Excluded Middle does not hold!

What is the Intuitive Reason for this Defect?

Logical reading of reactions:

(not b) / a *means* "If b is never present throughout the reaction, then a is asserted."

b / a *means* "If b is present throughout the reaction, then a is asserted."

/ a *means* "a is asserted independently of whether b becomes present at some point during the reaction."

$\Longrightarrow$ Propositional logic excludes events from becoming present "in–between".

Reactions Interpreted in Intuitionistic Logic

"Intuitionistic logic = Propositional logic − Law of the Excluded Middle"

Consider propositional, intuitionistic logic for providing a model–theoretic semantics of Statecharts reactions [LueMen2002]:

- Statecharts reactions can be characterized as so–called response models.
- The model–theoretic semantics is compositional and fully abstract.
- Response models correspond to the stable models studied in the field of answer set programming.

$\Longrightarrow$ Truly modular tool support for verification and code generation is possible – even for quite intricate Statecharts semantics!

Outlook: Mixing Data Flow & State Machines

Modern embedded systems:

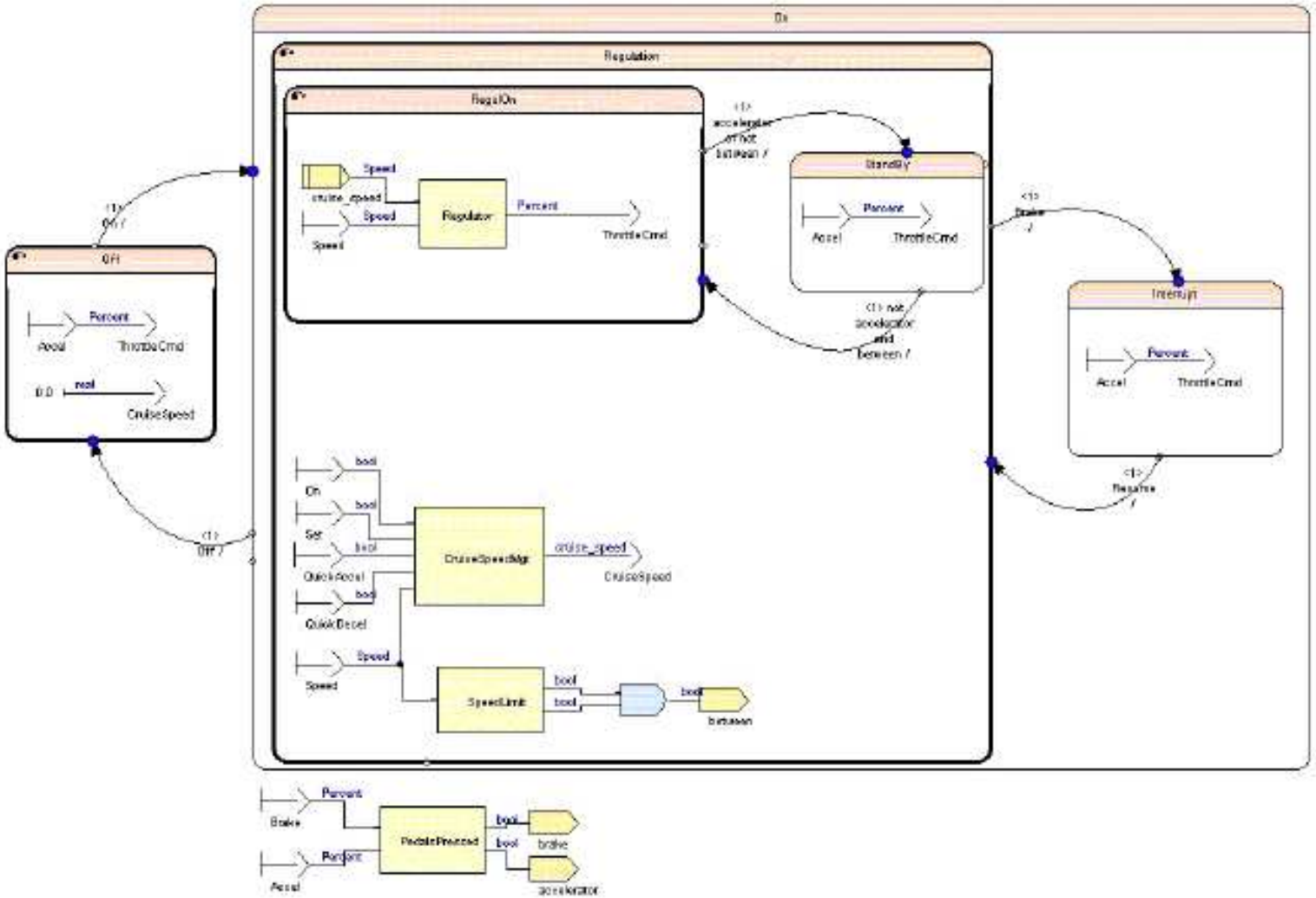
- Process continuous streams of data $\longrightarrow$ data flow
- Adapt functionality according to the operating environment $\longrightarrow$ state machine

Hence, a truly mixed data–flow/state–machine language is needed!

Forthcoming version 6 of SCADE:

- Lifts the restriction that SSMs are only permitted to appear at the leaves of the data flow model
- Allows one to (almost) arbitrarily combine block diagrams and SSMs
- Is based on a uniform extension of the basic clock calculus underlying SCADE

Cruise Control in SCADE v6



(Example taken from [ColPagPou2005])

Conclusions

Statecharts:

- Is a powerful visual language for designing controllers of embedded systems
- Is actually not one language but comes in many dialects
- Has many features, some of which are however quite dangerous to use, at least in certain dialects

A personal note on SCADE:

- Safe State Machines is a good dialect of Statecharts, with an academically well–investigated and “beautiful” semantics.
- SCADE is a tool well worth considering, also because its code generator is DO-178B qualified.
- Version 6 will make the underlying design language even more flexible.

Thank you for your time!

Selected References

- C. André. Representation and analysis of reactive behaviors. Technical report I3S RR 96–28, University of Nice – Sophia Antipolis, France, 1996.
- J.-L. Colaco, B. Pagano, M. Pouzet. A conservative extension of synchronous data-flow with state machines. Presentation at the SYNCHRON workshop, Malta, 2005.
- D. Harel. Statecharts: A visual formalism for complex systems. SCP, 8:231–274, 1987.
- R. von Hanxleden. Modeling reactive systems. Course notes, University of Kiel, Germany, 2004.
- G. Lüttgen, M. Mendler. The intuitionism behind Statecharts steps. ACM TOCL, 3(1):1–41, 2002.

Some Statecharts tools:

- SCADE. Esterel Technologies, *www.esterel-technolgies.com*.
- Simulink/Stateflow. The MathWorks, *www.mathworks.com/products/stateflow/*.
- Statemate. Telelogic/I-Logix, *www.ilogix.com*.
- Rose RealTime. IBM/Rational, *www-306.ibm.com/software/awdtools/developer/technical/*.