17 October 2017

SAND2017-10826PE

Tools for Simple Yet Very High Consequence Controls

Robert Armstrong, Geoffrey Hulette, Karla Morris, Jason Michnovicz, Jon Aytac, Philip Johnson-Freyd, Andrew Smith, Jackson Mayo, Ratish Punnoose

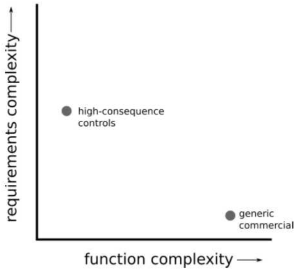
Sandia National Laboratories, Livermore, CA 94551, USA

Sandia National Laboratories is a multi-mission laboratory managed and operated by National Technology and Engineering Solutions of Sandia, LLC., a wholly owned subsidiary of Honeywell International, Inc., for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-NA0003525.

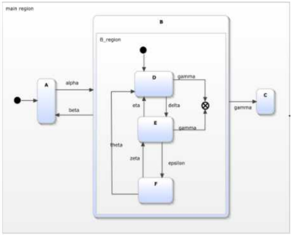
Outline

High-consequence controls: simple function, complex "always/never" requirements

- Our control systems are mostly *low complexity*, relatively *easy to analyze*.
- But, they often have a large number of *complex*, *high-consequence* safety, security, and reliability requirements.
- Low complexity + high consequence + complex requirements = ideal for a formal approach to design and/or verification.



Statecharts is an intuitive design language for simple controllers



- Well-suited to our simple controller domain.
- Mealy machine-based semantics, based on Argos – simple, effective, and approachable for end-users.
- Semantics is easy to express within action systems: Event-B, TLA+, etc.

Statecharts can support refinement-based design

Chart-based constructions in our Statecharts are refinements in the action system sense:

- Parallel and hierarchical composition
- Signal-based synchronization

Extended with a “math language,” our system also supports GCL-style refinement (strengthening guards, weakening actions, etc.)

Statecharts provide "natural" mechanisms for refinement

Hierarchical composition: An abstract parent state is refined by a set of concrete child states and their transitions.

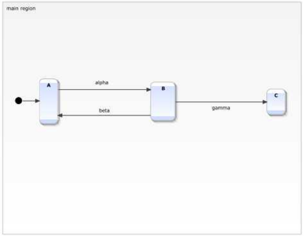


Figure: Abstract model

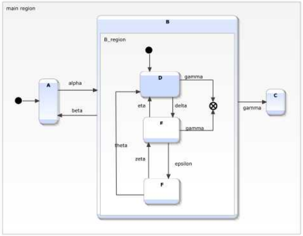


Figure: Refined model

Statecharts provide “natural” mechanisms for refinement

Guard Strengthening: Add guards to previoulsy defined abstract transitions. New guards are based on new concrete variables.

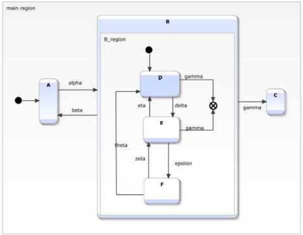


Figure: Abstract model

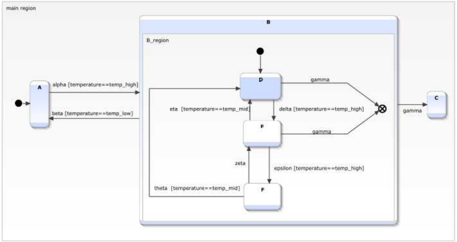
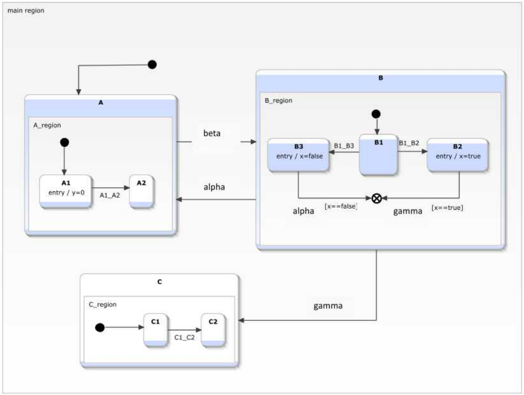

main region
B
B_region
A
alpha [temperature==temp_high]
beta [temperature==temp_low]
D
gamma
eta [temperature==temp_mid]
delta [temperature==temp_high]
F
gamma
zeta
epsilon [temperature==temp_high]
theta [temperature==temp_mid]
F
gamma
C

Figure: Refined model with temperature conditions

Restrictions on the Statechart Semantics

Our version of Statecharts is restricted vis-á-vis Herel's original paper, simplifying the formal semantics:

- Arrows can only go up or down one encapsulation level at a time
- Signals are scoped to the box in which they are created



Research interests

We are broadly interested in research areas related to refinement, action systems, and/or Statecharts:

- "Components" and connections to rely/guarantee reasoning
- Liveness properties
- Mathematical foundations (coalgebraic models, connections to logic, category theory)
- Practical issues, e.g., tractably managing deep hierarchies.

Collaborators interested in refinement-friendly Statecharts:

- Jet Propulsion Laboratory (US)

- University of Southampton (UK)

- Atomic Weapons Establishment (UK)

W3C SCXML Statechart Representation

W3C text representation called SCXML has been modified to accomodate refinement

- XML tools allow new meta-model namespaces to be introduced.
  - Existing SCXML tools will ignore them
- Needed in order to support:
  - Refinement levels (new attribute <iumlb:refinement >)
  - Invariants (new element <iumlb:invariant >)
  - Guards (new element <iumlb:guard >)

SCXML Attribute Extensions

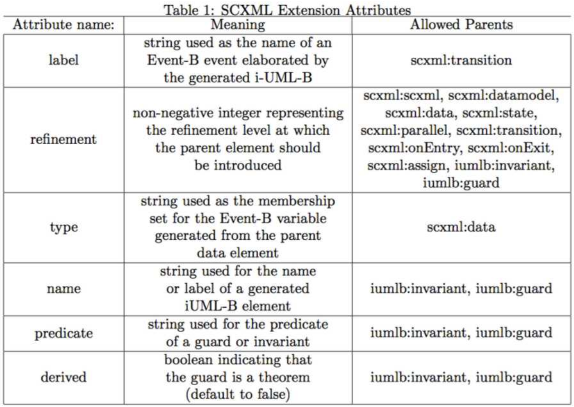
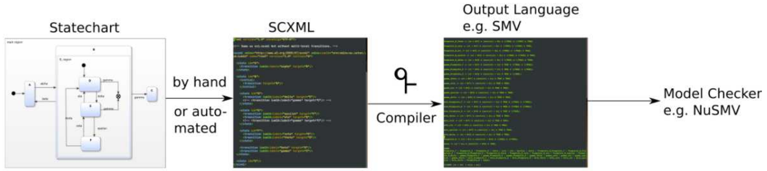


Table 1: SCXML Extension Attributes

Attribute name:	Meaning	Allowed Parents
label	string used as the name of an Event-B event elaborated by the generated i-UML-B	scxml:transition
refinement	non-negative integer representing the refinement level at which the parent element should be introduced	scxml:scxml, scxml:datamodel, scxml:data, scxml:state, scxml:parallel, scxml:transition, scxml:onEntry, scxml:onExit, scxml:assign, iumlb:invariant, iumlb:guard
type	string used as the membership set for the Event-B variable generated from the parent data element	scxml:data
name	string used for the name or label of a generated iUML-B element	iumlb:invariant, iumlb:guard
predicate	string used for the predicate of a guard or invariant	iumlb:invariant, iumlb:guard
derived	boolean indicating that the guard is a theorem (default to false)	iumlb:invariant, iumlb:guard

Q compiler takes SCXML and turns it into various prover languages



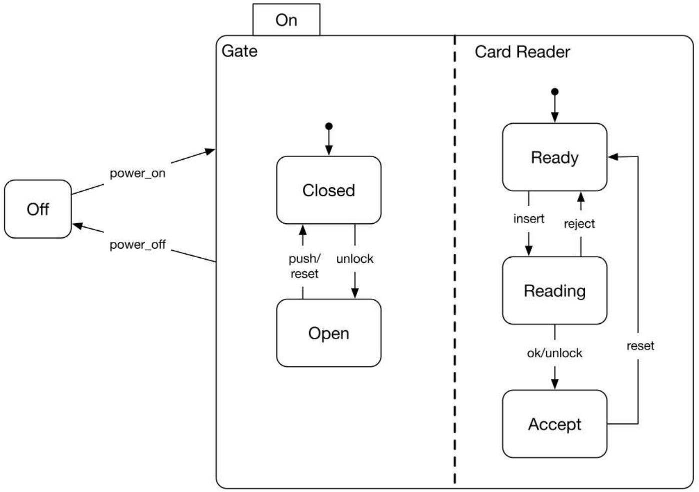
- We have created the Q compiler to take advantage of graphical Statechart-like tools that Engineers are familiar with
- Engineers use Statechart and Statechart-like tools to create specifications and prototypes for their controls
- Some examples
  - Mathworks Stateflow/Simulink
  - Ansys SCADE

Argos semantics "flattens" Statecharts into Mealy machines

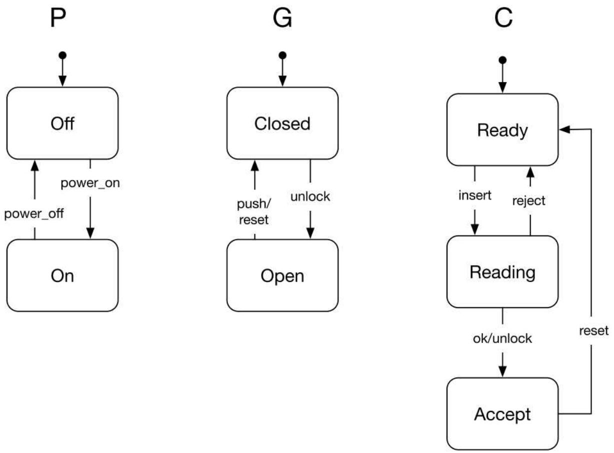
Argos defines a *compositional* semantics for Statecharts.

- Building blocks are Mealy machines, and the operators are *parallel composition*, *hierarchical composition*, and *encapsulation* (synchronization on signals).
  - Argos avoids some problematic constructions in Herel Statecharts, e.g., arrows that cut across the hierarchy.
  - Still has correctness conditions around causality, which are not easy to check for in general.
- Every valid (e.g., causal) Argos chart denotes a Mealy machine.
- We are working on showing that in Argos composed machines are *refinements* of their constituent machines.

Argos example: Statechart representation

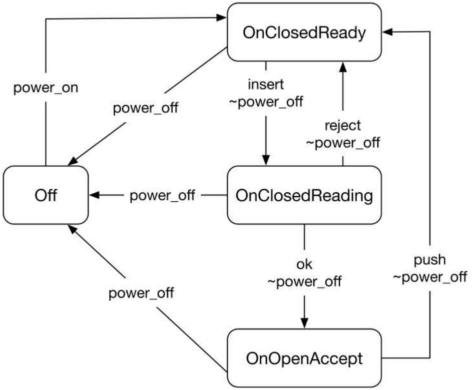


Argos example: Statecharts are compositions of machines



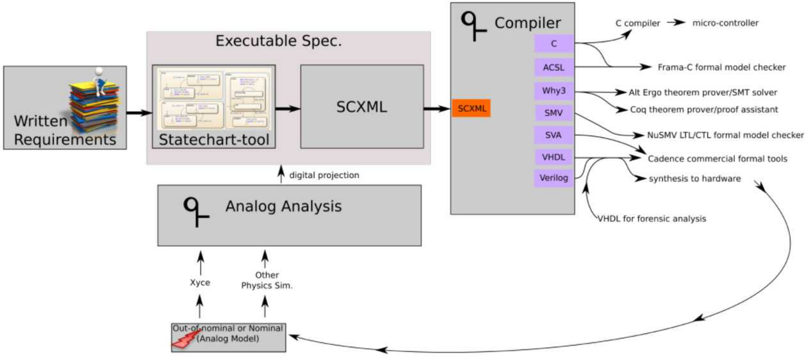
We can express the chart from the last slide as an expression over these Mealy machines: $P \triangleright \{\text{On} \mapsto G \times C\}/\text{unlock}, \text{reset}$.

Argos example: After flattening



Each state also has an explicit self-transition where the condition is the negated disjunction of the pictured outgoing transitions.

ꝗ tool maps Statecharts to many languages for further analysis by various formal tools


Written Requirements
Executable Spec.
Statechart-tool
SCXML
ꝗ Compiler
SCXML
C
ACSL
Why3
SMV
SVA
VHDL
Verilog
C compiler
micro-controller
Frama-C formal model checker
Alt Ergo theorem prover/SMT solver
Coq theorem prover/proof assistant
NuSMV LTL/CTL formal model checker
Cadence commercial formal tools
synthesis to hardware
VHDL for forensic analysis
digital projection
ꝗ Analog Analysis
Xyce
Other Physics Sim.
Out-of-nominal or Nominal (Analog Model)

Refinement and analysis across language boundaries

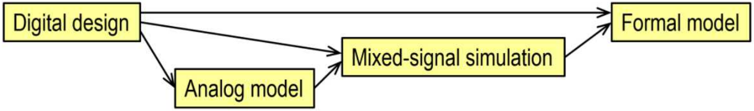
- Refinement from Statecharts to C
  1. Formally requires a mapping of logic and semantics from Statecharts to C (usually obvious)
  2. Obtain a refinement relation between the Statechart program and the C program
  3. Find the “flattened” Statechart transition relation in C by transforming through the refinement relation
  4. Check that the C proves the transition relation with Frama-C
- Current work-flow:
  - all done “by hand”
  - needs tooling for autogenerating refinement relation, proof obligations,etc.
- In principle can do Statechart $\rightarrow$ VHDL/Verilog following the same procedure
  - Substitute Commercial Formal EDA tool for Frama-C

Recover a specification for existing program, forensically

- Start with:
  - Existing program or firmware for which complete understanding is lacking
  - Partial specification (e.g. functional properties are known, safety properties are absent)
- Proceeds similar to correct-by-construciton analysis as before:
  - Reconstruct the specification and prove it against the extant program
  - Likely iterative, similar to CEGAR or abstract interpretation

Systems analysis can incorporate out-of-nominal electrical behavior

- Research is extending digital systems analysis to address physical environments where a device is not fully digital anymore
- Mixed-signal simulation can elucidate the digital imprint (e.g., bit flip pattern) of a physical insult (e.g., radiation) on a circuit
  - Using analog electrical model for the part of the circuit subjected to the insult
- By including digital upsets in a formal or complexity model, effect on rest of the digital state space can be quantified and mitigated
  - Example: Does a digital safety property still hold even in an accident scenario?



Failure modes can be understood via abstractions

- Examples of failures that result in an overapproximation:
  - A logic gate becomes unreliable and nondeterministic
  - A sensor fails, providing random input to a digital control
  - Generally: any malfunction that generates additional behaviors that were not part of the design intent

- Errors induced by environmental physics are common:
  - Radiation (cosmic rays, etc.)
  - Heating (fire, etc.)
  - Physical insult (destruction of sensor, etc.)
- Abstraction techniques can reveal failure modes for which a particular design will be robust
- Abstraction techniques can support designed-for failure modes anticipating likely accidents and faults

Square diagram shows refinement relationships that preserve requirements

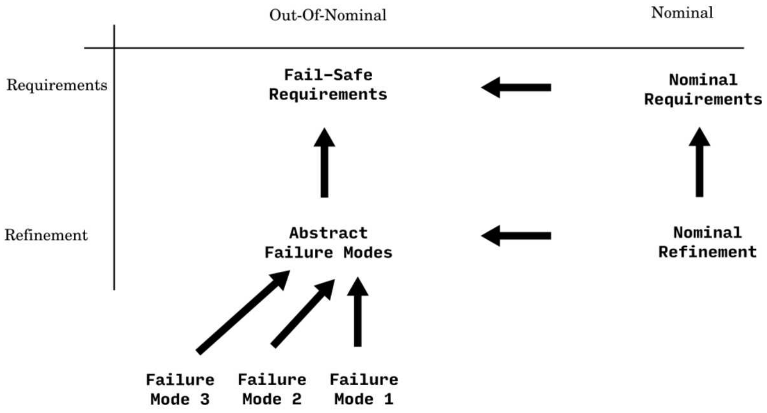


Figure from J. R. Mayo et al., Proc. 4th FTSCS Workshop, CCIS 596, doi:10.1007/978-3-319-29510-7_10. © 2016 Springer.

- Refinement/abstraction conceptual diagram for treating out-of-nominal and nominal models in a unified way
- Arrows point in the direction of abstraction

Broader principles support robustness in complex systems

- Biological and social complex systems typically are *not* formally verified, but show impressive robustness to unforeseen failures
- Why? They have inherent stability constraints from their origins in adaptation and selection

- Our hypothesis: Digital designs constrained by formal methods also exhibit enhanced robustness to unforeseen failures by a similar mechanism

Complex adaptive dynamical systems offer a useful perspective on hardware and software

- As dynamical systems, today's typical digital designs are *chaotic*
- Formal methods, by contrast, enforce *bounded* behavior, similar to that seen in complex systems adapted to their environments
  - To be useful (engineering) or viable (evolution), an adaptive dynamical system must show a coherent response, neither strongly overdamped/inert nor profoundly chaotic/random
  - At the "edge of chaos" (critical) or somewhat below it (subcritical), broad robustness to perturbations is obtained
  - Subcriticality or "smoothness" generalizes the constraints imposed by formal analyzability
- Restricted programming models also extend the power of testing
  - New programming models with intrinsic smoothness could enable more confident generalization of correctness to untested inputs
  - Empirically, incidence of vulnerabilities does differ measurably based on programming language

Boolean networks provide a simple representation of digital logic

- Originally investigated in biology, Boolean networks (BNs) correspond closely to hardware sequential logic gates
  - Each node in the directed graph has two possible states, 0 and 1
  - A node's state transition at each discrete time step is determined from its input connections by a "transfer function"
- Create BNs that add two 1-bit numbers (half-adder function), by random sampling and selection
  - This function is very simple, but we seek BNs representative of more complex implementations
  - BN ensembles differ in average inputs per node (k)
  - Select 20-node BNs that compute the correct result for all inputs when operating *nominally*, and then introduce 1% *bit errors* to evaluate robustness
  - Cascading errors are outlined in red

Boolean network "programs" exhibit quiescence for $k < 2$ and chaos for $k > 2$

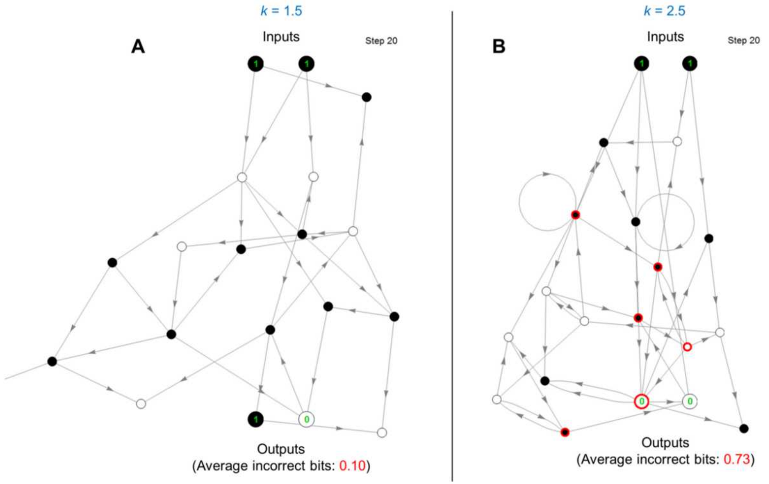


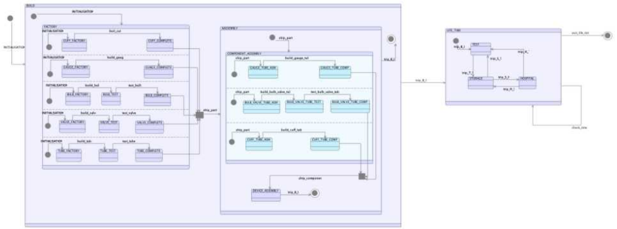
Figure from J. R. Mayo et al., Proc. 9th IEEE Systems Conference, doi:10.1109/SYSCON.2015.7116737. © 2015 IEEE.

Formal verification confirms insights from dynamical systems theory

- While BN stability is relevant well beyond the reach of exhaustive verification, the example half-adder BNs are simple enough to check directly with formal methods
- With the NuSMV model checker, we exhaustively prove/disprove correct function of these two BNs in the presence of bit errors
  - Using a nondeterministic model that allows any single bit error during a range of time steps
  - Example correctness requirement for carry bit:
    `LTLSPEC F ((clock=20) & (n18 = (n00&n01)))`
- NuSMV results: Chaotic BN is susceptible to corruption from *any* time step, whereas quiescent BN can be corrupted *only* in the last 5 of 20 time steps and is self-healing otherwise

Formal Analysis Uncovers Unwelcome Surprises in Complex Systems

- Normally FM work on programs and digital hardware, but here the program is a **representation** of a physical system
  - Find violations of safety, security and reliability **in the model**
  - Draw a correspondence to the physical reality that it represents
  - Big difference: model is an abstraction of reality but also has error bars, digital systems don't have error bars

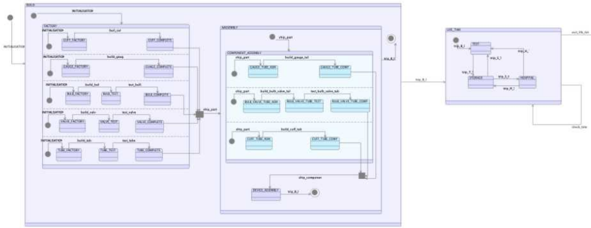


Model factory for building blood pressure monitors

Find evidence of rare but catastrophic failures that mere simulation cannot provide

Our methodology proceeds similar to CEGAR

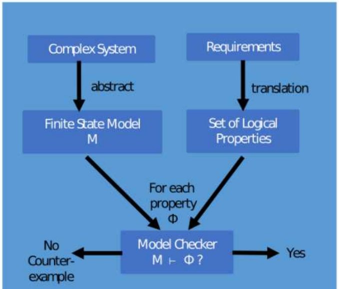
1. Establish a complex system (discrete event) model at some high level of abstraction above the physical system
2. Exhaustively find all counter-examples (may need HPC for this?)
3. Examine counter-examples for concurrence with reality (likely not on first try)
4. Refine model to eliminate unphysical counter-examples
5. Either all counter-examples are physical or loop back to 2



Model factory for building blood pressure monitors

Correct by construction for the design of complex systems

- Imagines a discrete event model constructed to find violations (or proofs of compliance with) safety, security, and reliability properties
  - not necessarily for simulation
- Use the formal analysis as a design tool to eliminate as many counterexamples as possible
  - strategically design-in tests, checks, and inspections to make the design of the physical system more robust


Complex System
Requirements
abstract
translation
Finite State Model M
Set of Logical Properties
For each property Φ
Model Checker M ⊢ Φ ?
No Counter-example
Yes