Semantics of Hybrid Systems

Roberto Passerone
Cadence Berkeley Laboratories

with contributions from E. Lee, A. Pinto, A. Sangiovanni-Vincentelli, H. Zheng

Chess Review
November 18, 2004
Berkeley, CA

Columbus: A Comparative Study

- Considered several existing models and tools
  - Charon, Checkmate, Hysdel, HSIF, HyVisual, Masaccio, Metropolis, Modelica, Scicos, Shift, Simulink
- Systematically compared the models for
  - Expressiveness
  - Concurrency model
  - Discrete/Continuous Communication
  - Hierarchy and Object Oriented design
  - Algebraic loops
- Compared models by running simple, but representative, examples
  - Zeno behavior
  - Level crossing detection
## Summary

<table>
<thead>
<tr>
<th>Language</th>
<th>Nature</th>
<th>Main Features</th>
<th>Featuring Also</th>
</tr>
</thead>
<tbody>
<tr>
<td>CHARON</td>
<td>Modelling Language</td>
<td>Formal semantic for hierarchy, concurrency, refinement</td>
<td>Simulator, Type Checker, Java interface</td>
</tr>
<tr>
<td>CHECKMATE</td>
<td>Verification Toolbox</td>
<td>Formal semantic for simulation, exploration, verification</td>
<td>Based on MATLAB/SIMULINK/STATEFLOW</td>
</tr>
<tr>
<td>HSB</td>
<td>Interchange Format</td>
<td>Modelling of networks of hybrid automata</td>
<td>Simulation through HyVisual</td>
</tr>
<tr>
<td>HYVISUAL</td>
<td>Visual Modeller</td>
<td>Hierarchy support, block diagram editor and simulator</td>
<td>Ptolemy-Il BASED</td>
</tr>
<tr>
<td>MASACCIO</td>
<td>Formal Model</td>
<td>Support for concurrent sequential and timed compositionality</td>
<td>Enables assume/guarantee reasoning</td>
</tr>
<tr>
<td>METROPOLIS</td>
<td>Design Environment</td>
<td>Heterogeneity, formal refinement, mapping</td>
<td>Verification, Simulation</td>
</tr>
<tr>
<td>MODELICA</td>
<td>Modelling Language</td>
<td>Object Oriented, non-causal modelling</td>
<td>Commercial and open simulator available</td>
</tr>
<tr>
<td>SCICOS</td>
<td>Hybrid System Toolbox</td>
<td>Modelling and simulation of hybrid systems</td>
<td>C code generation, interface to Syndex</td>
</tr>
<tr>
<td>SHIFT</td>
<td>Programming Language</td>
<td>Modelling of dynamic networks of hybrid components</td>
<td>C code generation</td>
</tr>
<tr>
<td>SIMULINK</td>
<td>Interactive Tool</td>
<td>Simulator, hierarchy, model discretizer</td>
<td>MATLAB-based, library of predefined blocks</td>
</tr>
</tbody>
</table>

## Summary

<table>
<thead>
<tr>
<th>Language</th>
<th>Hierarchy</th>
<th>Object-Oriented</th>
<th>Non-Causal Modeling</th>
</tr>
</thead>
<tbody>
<tr>
<td>CHECKMATE</td>
<td>N</td>
<td>N</td>
<td>N</td>
</tr>
<tr>
<td>HYSDEL</td>
<td>Y</td>
<td>Y</td>
<td>N</td>
</tr>
<tr>
<td>SCICOS</td>
<td>Y</td>
<td>N</td>
<td>N</td>
</tr>
<tr>
<td>HYVISUAL</td>
<td>N</td>
<td>N</td>
<td>N</td>
</tr>
</tbody>
</table>

## Summary

<table>
<thead>
<tr>
<th>Language</th>
<th>Discrete/Continuous Communication</th>
<th>Algebraic Loops</th>
<th>Dirac Pulses</th>
</tr>
</thead>
<tbody>
<tr>
<td>CHECKMATE</td>
<td>Event Generator and First Order Hold</td>
<td>N</td>
<td>N</td>
</tr>
<tr>
<td>HYSDEL</td>
<td>ToContinuous and ToDiscrete actors</td>
<td>Y</td>
<td>N</td>
</tr>
<tr>
<td>SCICOS</td>
<td>Indirect through when statements</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>HYVISUAL</td>
<td>Interaction between discrete state and continuous state</td>
<td>N</td>
<td>N</td>
</tr>
<tr>
<td>HYVISUAL</td>
<td>Event Generator and First Order Hold. There are no continuous signals</td>
<td>N</td>
<td>N</td>
</tr>
</tbody>
</table>
Conclusions

- Comparative study shows a fragmented landscape
  - Underlying models mostly incompatible
  - Key issues approached differently
- Consolidated view needed to advance the research and rate of adoption
  - Evidence from industry using ad-hoc translators
  - Difficult for engineers and practitioners to choose the right model
- Our activities are complementary ways of providing a consolidated view

Overview

- Solid and clean semantics for hybrid systems
- Interchange Format for hybrid systems
- Approximations for incompatible models
Operational Semantics for Hybrid Systems

- A solid and complete executable semantics for simulation
  - Robust and with no ambiguities
  - Designed to cover embedded software issues
- Focuses on deterministic behavior
  - It is incorrect to choose one trajectory
  - Creating deterministic models must be easy
  - Non-deterministic models must be explored either exhaustively or using Monte Carlo methods
- Avoids continuous time models to represent discrete behaviors
  - Inaccurate for software
  - Truly heterogeneous models are more faithful abstractions

HyVisual: Hybrid Systems as Networks of Automata
Some Semantics Questions

- Expressiveness of model
  - non-deterministic, guard expression language, actions, ...
- Coordination between subsystems (both discrete and continuous)
  - synchronous, time-driven, event-driven, dataflow, ...
  - can outputs and updates be separated?
- What is the meaning of directed cycles?
  - fixed point, error, infinite loop, ...
- What is the meaning of simultaneous events?
  - secondary orderings, such as data precedences, priorities, ...
- Discontinuous signals must have zero transition times
  - Precise transition times
  - Accurate model of Zeno conditions
- Discrete signals should have values only at discrete times
  - Accurately heterogeneous model (vs. continuous approximation)
- Sampling of discontinuous signals must be well-defined
  - Avoid unnecessary nondeterminism
- Transient states must be active for zero time
  - Properly represent glitches

Transient States and Glitches

This model shows that the level crossing detectors detect both the continuous and discontinuous properly.

The three level crossing detectors detect levels 0.5, 1.25, and -0.45 respectively.

The model modal produces a piecewise continuous signal with glitches (produced by the output actions of the transient states).

This finite state machine generates a piecewise-continuous signal with glitches.

The “init” state produces a consistent continuous signal. The states: state1, state2, and state3, are transient states. Their transitions produce glitches with their output actions.
Overview

Solid and clean semantics for hybrid systems

Interchange Format for hybrid systems

Approximations for incompatible models

Interchange Format for Hybrid Systems

- Define a common format that can be used to exchange data between different tools
  - Similar to other standards, such as LEF-DEF, Edif, and more recently OpenAccess
  - Avoid a proliferation of ad-hoc translators
- Flexible model that doesn’t tie you into one particular semantics
  - Unlike HSIF, avoid casting a preferred semantics in stone
  - Instead, a proper IF needs to include a meta model describing the semantics
Metropolis Meta-Model

- Basic elements can be composed to build a domain specific Model of Computation
  - Semantics formally defined as Action Automata
- Flexible infrastructure that supports
  - Heterogeneous modeling
  - Flexible communication semantics
  - Non-determinism
  - Hierarchical, Object Oriented design
  - Explicit causality and scheduling
  - Declarative constraints (invariant, equations)
  - Refinement

Anatomy of a model
Application Scenarios

Towards a Manipulable Semantics

- Action automata determine the semantics of a model through its quantity managers
  - However, if not careful, extracting the automata and analyzing them could be very expensive
  - We are investigating ways of defining quantity managers in simpler and more manageable terms
- When intractable, translations and analysis could be obtained by ignoring certain aspects
  - This is sometimes desirable to decrease the complexity of a model for formal verification
  - This is essential when the information that is ignored is irrelevant
- Models of hybrid systems can be related through approximations
Overview

Solid and clean semantics for hybrid systems

Interchange Format for hybrid systems

Approximations for incompatible models

Conservative Approximations

- Use conservative abstractions to relate different models
  - Why use abstraction A as opposed to abstraction B?
- Study preservation properties of abstractions
  - If a property holds before applying the abstraction, does it also hold after the abstraction?
- What are the properties of interest?
  - Compositionality or commutativity
  - Preservation of the refinement relation
Preservation of refinement

- Each model defines refinement differently
  - A block (actor) implements another when you can replace the former for the latter
  - Refinement denoted by the symbol $\leq$ (partial or pre-order)
- Refinement verification in the abstract is potentially more efficient, but does it hold in the concrete?
  - In other words, does the abstraction preserve refinement?
  - Verification will necessarily be conservative, but is it sound?

Preservation of refinement

- Refinement preserving approximation
  - A function $H$ between two models preserves refinement if and only if $H(p_1) \leq H(p_2)$ implies $p_1 \leq p_2$
  - In other words, $H$ is "inverse" monotonic
  - Analogy (not) for real numbers $r$ and $s$
    - if $[r] \leq [s]$ then not $r \leq s$
- Inverse monotonic functions are not useful
  - Say $H(p_1) = H(p_2)$. Then $p_1 = p_2$
  - In other words, $H$ is injective (not giving up information)
  - Hence $H$ is not an abstraction at all!
- One function does not fit all
Preservation of refinement

- **Conservative approximation**
  - A pair of functions $\Psi = (\Psi_1, \Psi_u)$ is a conservative approximation if and only if
    \[ \Psi_u(p_1) \leq \Psi_1(p_2) \text{ implies } p_1 \leq p_2 \]
  - Analogy: if $[r] \leq [s]$ then $r \leq s$
  - Abstract implies detailed
- **Conservative approximations are useful**
  - Implication going in the right direction
  - $\Psi_1$ and $\Psi_u$ are both abstractions (they need not be injective)
- **Refinement verification is always sound**

Verification problem

- A specification $q$ requires that action "b" always be preceded by action "a"
  - Going from reals to integers, the order between events during the same integer interval is lost
  - Verification unsound when the specification is not represented exactly at the abstract level
- **Conservative approximations detect when the solution would be unsound**
  - But $\Psi(q)$ is not empty!
  - It has all the behaviors for which $a$ and $b$ are separated by at least one time unit
  - Verification possible if the implementation is "slow enough"
- **There is a relation between our sampling frequency and the ability to verify in the abstract**
  - Subtle interaction between implementation and verification strategy
  - Conservative approximations separate those concerns
Inverse of conservative approximation

- The inverse of an abstraction does not necessarily exist
  - $H(p)$ does not determine $p$ uniquely
  - Similarly, $\Psi_u(p)$ and $\Psi_l(p)$ do not determine $p$ uniquely
- Inverse defined when upper and lower bound coincide
  - If $\Psi_u(p) = \Psi_l(p)$, then $p$ can be represented exactly at the abstract level
  - $p$ is uniquely determined in this case

Abstraction and Refinement

- $\Psi_{\text{inv}}$ identifies actors that can be used indifferently in either domain
  - If $Q'$ is an abstraction of $Q$, then $\Psi_{\text{inv}}$ is an injection from $Q'$ to $Q$
  - Actors are "domain polymorphic"
- Other actors are only approximated in the other semantic domain
  - $\Psi_u$ and $\Psi_l$ are different "views"
  - $\Psi_{\text{inv}} \circ \Psi_u$ is a closure operator
  - $\Psi_{\text{inv}} \circ \Psi_l$ is an interior operator
Conclusions

- Comparative study shows a fragmented landscape
  - Underlying models mostly incompatible
  - Key issues approached differently
- Consolidated view needed to advance the research
  - Evidence from industry using ad-hoc translators
  - Difficult for practitioners to choose the right model
- Our activities are complementary ways of providing a consolidated view
  - HyVisual: Solid and clean semantics for hybrid systems
  - Metropolis Interchange Format for hybrid systems
  - Conservative Approximations for incompatible models