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

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

Summary

Language	Continuous/Discrete Signals	Derivative	Automata	State/Dynamics mapping
CHECKMATE	Separation between FSM and dynamical system. Communication through event generator.	YES	STATEFLOW model	Discrete output form FSM to dynamical system.
HYVISUAL	Signal attribute. Automatic detection of type or enforced by user.	Integration	Graphical Editor	State refinement into continuous time models
MODELICA	Defined by a language modifier	YES	Described by algorithm sections.	Different equation sets depending on events.
SCICOS	Defined by port attribute	Integration	Implemented by interconnection of conditional blocks.	Implemented by connection of events selectors.
HYSDEL	Real and Boolean signals	Discrete difference	Implemented by logic functions	Discrete output form FSM to dynamical system.

Summary

Language	Hierarchy	Object Oriented	Non-Causal Modelling
CHECKMATE	N	N	N
HYVISUAL	Y	Y	N
MODELICA	Y	Y	Y
SCICOS	Y	N	N
HYSDEL	N	N	N

Summary

Language	Discrete/Continuous Communication	Algebraic Loops	Dirac Pulses
CHECKMATE	Event Generator and First Order Hold	N	N
HYVISUAL	ToContinuous and ToDiscrete actors	Y	N
MODELICA	Indirect through when statements	Y	Y
SCICOS	Interaction between discrete state and continuous state	N	N
HYSDEL	Event Generator and First Order Hold. There are no continuous signals	N	N

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

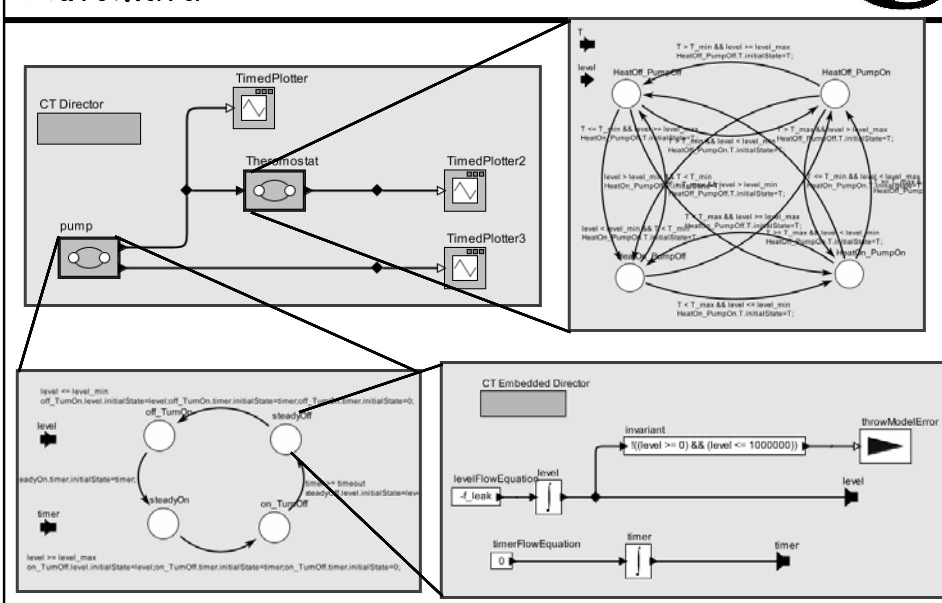
Interchange Format
for hybrid systems

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

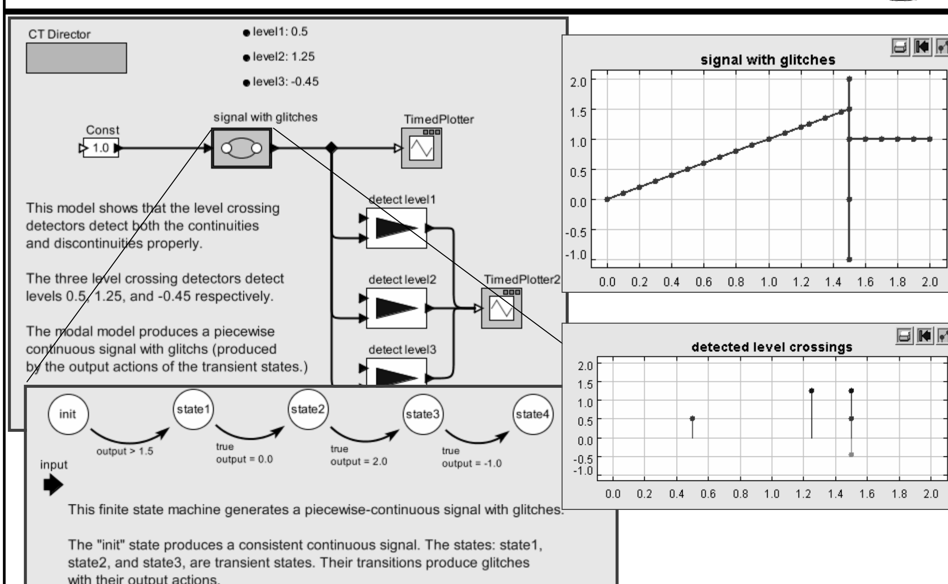
HyVisual: Hybrid Systems as Networks of Automata



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



Overview

Solid and clean semantics
for hybrid systems

Interchange Format
for hybrid systems

Approximations
for incompatible models

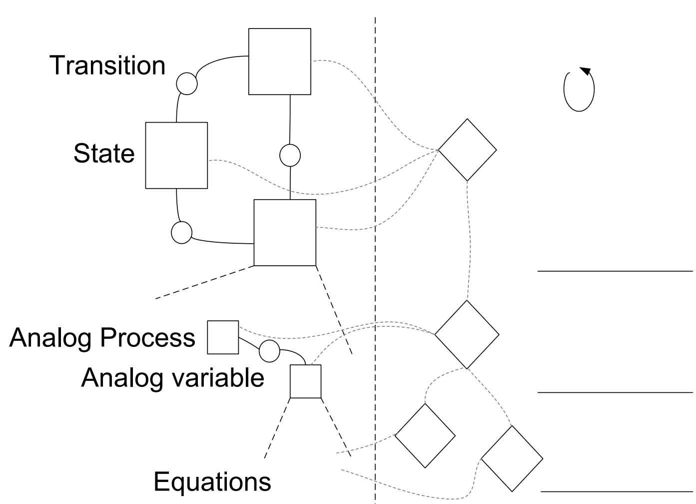
Interchange Format for Hybrid Systems

- 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

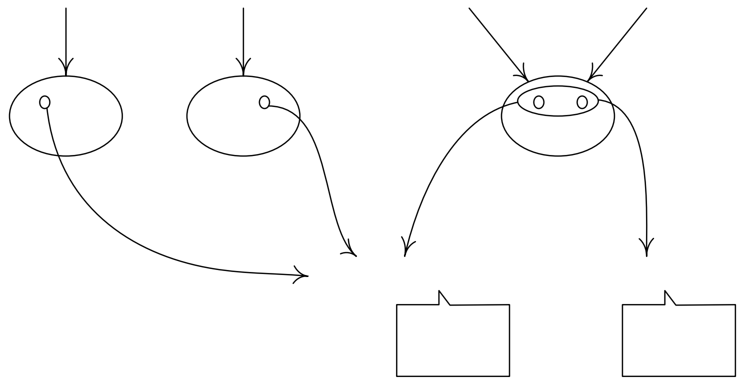
Metropolis Meta-Model

- 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 manager 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

HyVisu

Overview

Solid and clean semantics
for hybrid systems

Interchange Format
for hybrid systems

Approximations
for incompatible models

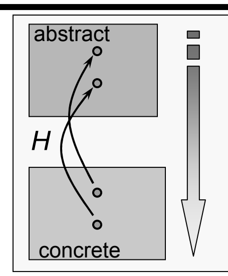
Conservative Approximations

- 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

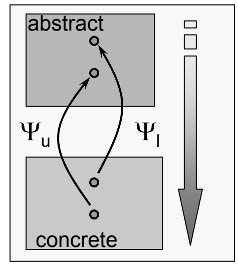
- Each model defines refinement differently
  - A block (actor) implements another when you can replace the former for the latter
  - Refinement denoted by the symbol $\leq$ (partial or pre-order)
- Refinement verification in the abstract is potentially more efficient, but does it hold in the concrete?
  - 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 $\lfloor r \rfloor \leq \lfloor s \rfloor$ 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



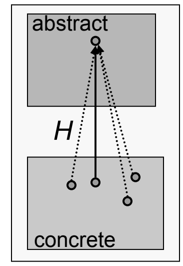
- Conservative approximation
  - A pair of functions $\Psi = (\Psi_l, \Psi_u)$ is a *conservative approximation* if and only if $\Psi_u(p_1) \le \Psi_l(p_2)$ implies $p_1 \le p_2$
  - Analogy: if $\lceil r \rceil \le \lfloor s \rfloor$ then $r \le s$
  - Abstract implies detailed
- Conservative approximations are useful
  - Implication going in the right direction
  - Ψ_l and Ψ_u are both abstractions (they need not be injective)
- Refinement verification is always sound

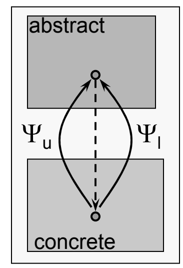
Verification problem

- 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 $\Psi_l(q)$ 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

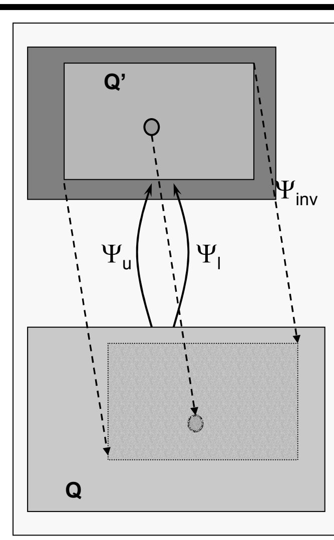
- The inverse of an abstraction does not necessarily exist
  - *H*(*p*) does not determine *p* uniquely
  - Similarly, Ψ_u(*p*) and Ψ_l(*p*) do not determine *p* uniquely
- Inverse defined when upper and lower bound coincide
  - If $\Psi_u(p) = \Psi_l(p)$, then *p* can be represented *exactly* at the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





Abstraction and Refinement

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