Hybrid Systems
Modeling and Verification

Rajeev Alur
University of Pennsylvania
www.cis.upenn.edu/~alur/

Dynamics & Verification Workshop, July 2001
What are Hybrid Systems?

State machines  +  Dynamical systems

\[ \begin{align*}
\text{on} & \quad dx = kx \\
& \quad x < 70
\end{align*} \]

\[ \begin{align*}
\text{off} & \quad dx = -k'x \\
& \quad x > 60
\end{align*} \]
Hybrid Systems in Applications

Computer control systems

- continuous + discrete sensors/actuators
- continuous + discrete control functions
  (>80% of typical feedback control software is logic)
- mode-switching control strategies

Less than 1% of all microprocessors are in PC -- most microprocessors are implementing embedded control.

Embedded software in cars, airplanes, chemical plants, medical devices.
Unified Modeling paradigm

Discrete Control (Software)

Sensors

Physical Plant (Hybrid Dynamics)

Actuators
Motivation

- **Formal foundations for design**
  - Rich modeling constructs
  - Rigorous analysis and controller design
  - Powerful debugging -> improved reliability
  - Model based design -> greater design automation

- **Why now?**
  - UML-based tools -> greater model-based software
  - Confluence of control theory and computer science
  - Software in embedded control systems getting complex
    -> need for better software engineering
  - Advances in formal verification tools and techniques
## Hybrid Dynamic Systems

Dynamic systems with both continuous- & discrete- state variables

<table>
<thead>
<tr>
<th>Models</th>
<th>Continuous-State Systems</th>
<th>Discrete-State Systems</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>differential equations, transfer functions, etc.</td>
<td>automata, Petri nets, Statecharts, etc.</td>
</tr>
<tr>
<td>Analytical Tools</td>
<td>Lyapunov functions, eigenvalue analysis, etc.</td>
<td>Boolean algebra, formal logics, verification, etc.</td>
</tr>
<tr>
<td>Software Tools</td>
<td>Matlab, Matrix(_x), VisSim, etc.,</td>
<td>Statemate, Rational Rose, SMV, etc.</td>
</tr>
</tbody>
</table>
Course Overview

- **Modeling and Semantics**
  - Timed automata, Hybrid automata
  - Modularity, Compositionality, Hierarchy

- **Decidability and Verification**
  - Decidable classes, Quotients, Undecidability

- **Symbolic Reachability**
  - Timed Automata, Linear Hybrid Automata
  - Approximations of reachable sets

- **Applications**
  - Embedded Control Systems
  - Robotics
  - Biological systems
Hybrid Systems Group at Penn

Faculty
- Rajeev Alur (CIS)
- Vijay Kumar (MEAM)
- Insup Lee (CIS)
- George Pappas (EE)
- Harvey Rubin (Medicine)

Research Associates
- Thao Dang
- Rafael Fiero
- Oleg Sokolsky

PhD Students
- Calin Belta
- Joel Esposito
- Yerang Hur
- Franjo Ivancic
- Salvatore La Torre
- Pradumna Mishra
- Jiaxiang Zhou
Acknowledgements

- Thanks for providing powerpoint slides
  - Colleagues at Penn
  - Bruce Krogh at CMU
  - Kim Larsen at Aalborg

- Caution:
  - Only a partial coverage area
  - Mostly computer-science centric
  - Biased towards my current interests
  - Apologies for incomplete references and notational variations
Lecture 1.
Modeling and Semantics
Talk Outline

- Timed Automata
- Hybrid Automata
- CHARON: Hierarchical Specification
- Modular Analysis
Simple Light Control

**WANT:** if press is issued twice *quickly* then the light will get *brighter*; otherwise the light is turned *off.*
Simple Light Control

Solution: Add real-valued clock $x$

Adding continuous variables to state machines
Timed Automata

Clocks: $x, y$

Guard
Boolean combination of comparisons with integer bounds

Reset
Action performed on clocks

State
$(\text{location}, x=v, y=u)$ where $v,u$ are in $\mathbb{R}$

Transitions
\[
\begin{align*}
( n, x=2.4, y=3.1415 ) & \xrightarrow{a} ( m, x=0, y=3.1415 ) \\
( n, x=2.4, y=3.1415 ) & \xrightarrow{e(1.1)} ( n, x=3.5, y=4.2415 )
\end{align*}
\]
Adding Invariants

Clocks: $x, y$

Transitions

1. $(n, x=2.4, y=3.1415)$
2. $(n, x=3.5, y=4.2415)$

Location Invariants

$n, x \leq 5$
$m, y \leq 10$

Invariants ensure progress!!
Clock Constraints

For set $C$ of clocks with $x, y \in C$, the set of clock constraints over $C$, $\Psi(C)$, is defined by

$$\alpha ::= x < c \mid x - y < c \mid \neg \alpha \mid (\alpha \land \alpha)$$

where $c \in \mathbb{N}$ and $\prec \in \{<, \leq\}$.

What can you express:
- Constant lower and upper bounds on delays

Why the restricted syntax:
- slight generalizations (e.g. allowing $x=2y$)
- lead to undecidable model checking problems
Timed Automata

A timed automaton $A$ is a tuple $(L, l_0, E, Label, C, clocks, guard, inv)$ with

- $L$, a non-empty, finite set of locations with initial location $l_0 \in L$
- $E \subseteq L \times L$, a set of edges
- $Label : L \rightarrow 2^{AP}$, a function that assigns to each location $l \in L$ a set $Label(l)$ of atomic propositions
- $C$, a finite set of clocks
- $clocks : E \rightarrow 2^{C}$, a function that assigns to each edge $e \in E$ a set of clocks $clocks(e)$
- $guard : E \rightarrow \Psi(C)$, a function that labels each edge $e \in E$ with a clock constraint $guard(e)$ over $C$, and
- $inv : L \rightarrow \Psi(C)$, a function that assigns to each location an invariant.
Light Switch

- Switch may be turned on whenever at least 2 time units have elapsed since last "turn off".
- Light automatically switches off after 9 time units.
Semantics

- **clock valuations:**  
  \[ V(C) \quad v : C \rightarrow R_{\geq 0} \]

- **state:**  
  \[ (l, v) \text{ where } l \in L \text{ and } v \in V(C) \]

- **Operational semantics of timed automaton is a labeled transition system**  
  \[ (S, \rightarrow) \]
  where \( S \) is the set of all states

  \[ (l, v) \xrightarrow{a} (l', v') \text{ iff } \quad \begin{array}{c}
  \circ \quad g \quad a \quad r \\
  \rightarrow 
  \end{array} \\
  g(v) \text{ and } v' = v[r] \text{ and } Inv(l')(v') \]

- **action transition**

- **delay Transition**  
  \[ (l, v) \xrightarrow{d} (l, v + d) \text{ iff } \quad Inv(l)(v + d') \text{ whenever } d' \leq d \in R_{\geq 0} \]
Semantics: Example

\[(\text{off}, x = y = 0) \xrightarrow{3.5} (\text{off}, x = y = 3.5) \xrightarrow{\text{push}} \]
\[(\text{on}, x = y = 0) \xrightarrow{\pi} (\text{on}, x = y = \pi) \xrightarrow{\text{push}} \]
\[(\text{on}, x = 0, y = \pi) \xrightarrow{3} (\text{on}, x = 3, y = \pi + 3) \xrightarrow{9-(\pi+3)} \]
\[(\text{on}, x = 9-(\pi+3), y = 9) \xrightarrow{\text{click}} (\text{off}, x = 0, y = 9) \ldots \]
Talk Outline

✓ Timed Automata
✓ Hybrid Automata
☐ CHARON: Hierarchical Specification
☐ Compositionality and Refinement
Hybrid Automata

connections or modes (discrete states)
edge guard

\[ \begin{align*}
    X & \in \text{Inv}(l) \\
    dX & \in \text{Flow}(l) \\
    X & \in \text{Init}(l)
\end{align*} \]

\[ e : g(X) \geq 0 \]

\[ J(X, X') \]

invariant: hybrid automaton may remain in \( l \) as long as \( X \in \text{Inv}(l) \)

jump transformation

continuous dynamics
Switched Dynamic Systems

continuous dynamics

jump dynamics

discrete dynamics
Hybrid Automata

- Set $L$ of locations, and set $E$ of edges
- Set $X$ of $k$ continuous variables
- State space: $L \times \mathbb{R}^k$, Region: subset of $\mathbb{R}^k$

For each location $l$,
- Initial states: region $\text{Init}(l)$
- Invariant: region $\text{Inv}(l)$
- Continuous dynamics: $dX$ in $\text{Flow}(l)(X)$

For each edge $e$ from location $l$ to location $l'$
- Guard: region $\text{Guard}(e)$
- Update relation over $\mathbb{R}^k \times \mathbb{R}^k$
- Synchronization labels (communication information)
(Finite) Executions of Hybrid Automata

- State: \((l, x)\) such that \(x\) satisfies \(\text{Inv}(l)\)
- Initialization: \((l,x)\) s.t. \(x\) satisfies \(\text{Init}(l)\)
- Two types of state updates
  - Discrete switches: \((l,x) \rightarrow_a (l',x')\) if there is an a-labeled edge \(e\) from \(l\) to \(l'\) s.t. \(x\) satisfies \(\text{Guard}(e)\) and \((x,x')\) satisfies update relation \(\text{Jump}(e)\)
  - Continuous flows: \((l,x) \rightarrow_f (l,x')\) where \(f\) is a continuous function from \([0,\delta]\) s.t. \(f(0)=x\), \(f(\delta)=x'\), and for all \(t\leq\delta\), \(f(t)\) satisfies \(\text{Inv}(l)\) and \(df(t)\) satisfies \(\text{Flow}(l)(f(t))\)
Refined Modeling

- **Issues coming up**
  - Adding hierarchy for structured modeling
  - Observational semantics
  - Compositionality and refinement

- **Issues not covered**
  - Infinite trajectories, divergence, non-Zenoness
  - Concurrency and synchronization
Talk Outline

✓ Timed Automata
✓ Hybrid Automata
❖ CHARON: Hierarchical Specification
☐ Compositionality and refinement
Trends in Software Design

Emerging notations: UML-RT, Stateflow
- Visual
- Hierarchical modeling of control flow
- Object oriented

Prototyping/modeling but no analysis
- Ad-hoc, informal features
- No support for abstraction

CHARON: Formal, hierarchical, hybrid state-machine based modeling language
CHARON Language Features

- Individual components described as agents
  - Composition, instantiation, and hiding
- Individual behaviors described as modes
  - Encapsulation, instantiation, and Scoping
- Support for concurrency
  - Shared variables as well as message passing
- Support for discrete and continuous behavior
  - Differential as well as algebraic constraints
  - Discrete transitions can call Java routines
Robot Team Approaching a Target
write diff analog position pos₁, pos₂

class position {
    float x;
    float y;
}

Variables Specifiers

Range: discrete/analog

Computation: diff/alg

Access: read/write/local
Architectural Hierarchy
Behavioral Hierarchy

\[
\begin{align*}
\text{pos} & = \text{target} \\
\text{pos.x} & = v \times \cos(\phi) \\
\text{pos.y} & = v \times \sin(\phi)
\end{align*}
\]
Powertrain modeling
agent PowerTrain

\[ P_{\text{atm}} \]

agent Engine

mass of air: \( \dot{m}_a = f_1(m_a, \omega_e, \alpha, P_m) \)

injected fuel rate: \( dMf_i = \frac{1}{\tau_f} (dMf_c - dMf_i) \)

engine speed: \( \dot{\omega}_e = f_2(\omega_e, m_a, T_p, SA) \)

agent AutomaticTransmission

agent TorqueConverter

mode Converter
\[ T_p = \text{alg} \]
\[ T_p = \text{alg} \cdot \exp_2 \]
mode Coupling
\[ T_t = \frac{\omega_t}{\omega_e} < 0.9 \]
\[ T_t = \frac{\omega_t}{\omega_e} \geq 0.9 \]
\[ T_t = \text{alg} \cdot \exp_3 \]

agent Transmission

\( \omega_c, \text{dMai, Pm, Tm} \)

\( \theta_e, \text{sensors} \)

\( \omega_{cr} \)

drive_train

\( \omega \)

gear

\( T \)

\( T_s \)

\( pc_1, pc_4, pb_{12} \)

\( \theta_e, \text{dMai, Pm, Tm} \)

\( \omega_c, \text{sensors} \)

\( \omega_{cr} \)

drive_train

\( \omega \)

gear

\( T \)

\( T_s \)

\( pc_1, pc_4, pb_{12} \)
Charon Summary

- Structured hierarchical modeling
- Formal compositional semantics with a notion of refinement
- Hierarchy can be exploited during analysis (e.g. multi-rate simulation)
- Analysis such as model checking and runtime monitoring to be supported
Exploiting hierarchy: Modular Simulation

1. Get integration time $\delta$ and invariants from the supermode (or the scheduler).

2. While (time $t = 0; t <= \delta$) do:
   - Simplify all invariants.
   - Predict integration step $dt$ based on $\delta$ and the invariants.
   - Execute time round of the active submode and get state $s$ and time elapsed $\epsilon$.
   - Integrate for time $\epsilon$ and get new state $s$.
   - Return $s$ and $t+\epsilon$ if invariants were violated.
   - Increment $t = t+\epsilon$.

3. Return $s$ and $\delta$
- “Slowest-first” order of integration
- Coupling is accommodated by using interpolants for slow variables
- Tight error bound: $O(h^{m+1})$

Use a different time step for each component to exploit multiple time scales, to increasing efficiency.

$\text{error} \approx h^{m+1}C \sum_{j=1}^{i-1} \frac{\partial f_i}{\partial x_j} (R - 1)$

- ratio of largest to smallest step size
- constant
- step size
- coupling

Modular Multi-rate Simulation
Computations for Modular Systems

Efficiency gain increases dramatically when simulating systems with complex right-hand sided or tight error tolerances.
Talk Outline

✓ Timed Automata
✓ Hybrid Automata
✓ CHARON: Hierarchical Specification
凼 Compositionality and Refinement
Motivation

Which properties are preserved?
Can we restrict reasoning to modified parts of design?
Component should have precise interface specification
Components differing only in internal details are equivalent

Theme: Composable Behavioral Interfaces!
Observational Semantics

- Classical programming language concept of denotational semantics: two programs are “equivalent” if they compute the same function

- For reactive systems, ongoing interaction (behavior over time) must be accounted for

- Observational semantics of a hybrid component:
  - Signature (static interface): Set of input/output variables
  - Behavioral interface: Set of traces

- Trace: Projection of an execution onto observable parts (e.g. sequence of input/outputs)
**Compositional Semantics**

- **Traces** should retain all (but no more) information needed to determine interaction of a component with other components.

- **Desired theorems**
  - If $C$ and $C'$ are equivalent, then in any context $C$ must be substitutable by $C'$.
  - Traces of a system with multiple components can be computed from traces of its components.
    
    e.g. $\text{traces}(P|Q) = \text{traces}(P) \cap \text{traces}(Q)$

- Typically, we can project out information about private variables and modes, but not about timing, and even flows, of communication variables.
Global $x$

Local $t$

- **Mode A**
  - $dt = 1$
  - $dx = x$
  - $t \leq 10$
  - $t = 10$
  - $t := 0$

- **Mode B**
  - $dt = 1$
  - $dx = -1$
  - $t \leq 6$
  - $t > 5$
  - $t := 0$

Sample Execution

Sample Trace
Refinement

- Component I refines component S if they have the same static signatures, and every trace of I is also a trace of S.
- Implementation I is more constrained than specification model S.
- Implementation I inherits properties of S.
- Multiple implementations of S possible.
- Desired: Proof calculus for decomposing refinement goals into subgoals.
- Typical rules: Compositionality, Assume-guarantee.
- Foundation for formal top-down design.
- Caution: Details of these general principles are highly sensitive to specifics of a modeling language.
Semantics of Charon modes

- **Semantics of a mode consists of:**
  - entry and exit points
  - global variables
  - traces

- **Key Thm:** Semantics is compositional
  - traces of a mode can be computed from traces of its sub-modes
Refinement

Refinement is trace inclusion

Normal

\[ \{ \hat{t} = 1 \} \{ \text{level} \in [2,10] \} \]

- Compute
e
- Maintain \{t < 10\}
- t := 0
dx
t = 10
de

cpy <

Normal

\[ \{ \hat{t} = 1 \} \{ \text{level} \leq 10 \} \]

- Compute
e
- Maintain \{t < 10\}
- t := 0
de

Normal'

- Same control points and global variables
- Guards and constraints are relaxed
Sub-mode refinement

Controller

Normal'\(\text{de} \quad \text{level} \in [4, 8]\)\quad \text{dx} \quad \text{level} \in [2, 10]\)

Emergency

Controller'

Normal'\(\text{de} \quad \text{level} \in [4, 8]\)\quad \text{dx} \quad \text{level} \in [2, 10]\)

Emergency

Refines
Compositional Reasoning

Sub-mode refinement

Context refinement