Hybrid Systems
Formal Modeling and Verification

Rajeev Alur

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

Tutorial at Illinois Intl Multiconference, Sept 2003
Trends in Model-Based Design

- **Emerging notations:** UML, Stateflow
  - Visual, Hierarchical, Object oriented
  - Simulation, code generation

- **Steady progress in model checking tools**

- **Control design employs tools (Matlab...)**

- **Opportunities to influence design tools**
  - Typically, semantics is not formal
  - Typically, only simulation is supported
  - Code generation available, but ....
Advantages
Automated formal verification, Effective debugging tool

Moderate industrial success
In-house groups: Intel, Microsoft, Lucent, Motorola...
Commercial model checkers: FormalCheck by Cadence

Obstacles
Scalability is still a problem (about 500 state vars)
Effective use requires great expertise

Still, a great success story for CS theory impacting practice, and a vibrant area of research
Hybrid Modeling

State machines + Dynamical systems

\[
\begin{align*}
\text{on} & \quad dx = kx & x < 70 \\
\text{off} & \quad dx = -k'x & x > 60
\end{align*}
\]
Automotive Applications
Coordination Protocols
Interacting Autonomous Robots
Physics-based Animation
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
  - Predicate abstraction
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
Part 1.
Modeling and Semantics
Talk Outline: Part 1

- 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

$(n, x=2.4, y=3.1415) \xrightarrow{a} (m, x=0, y=3.1415)$

$(n, x=2.4, y=3.1415) \xrightarrow{\text{wait}(1.1)} (n, x=3.5, y=4.2415)$
Adding Invariants

Clocks: $x, y$

Transitions

$(n, x=2.4, y=3.1415) \xrightarrow{\text{wait}(3.2)}$

$(n, x=2.4, y=3.1415) \xrightarrow{\text{wait}(1.1)}$

$(n, x=3.5, y=4.2415)$

Location Invariants

$x \leq 5$

$y \leq 10$

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 \( < \in \{<,\leq\} \).

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
- \( Label : L \to 2^{AP} \), a function that assigns to each location \( l \in L \) a set \( Label(l) \) of atomic propositions
- \( C \), a finite set of clocks
- \( clocks : E \to 2^C \), a function that assigns to each edge \( e \in E \) a set of clocks \( clocks(e) \)
- \( guard : E \to \Psi(C) \), a function that labels each edge \( e \in E \) with a clock constraint \( guard(e) \) over \( C \), and
- \( inv : L \to \Psi(C) \), 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**: \( V(C) \ni v : C \rightarrow R \geq 0 \)
- **state**: \( (l, v) \) where \( l \in L \) 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

- **action transition**: \( (l, v) \xrightarrow{a} (l', v') \) iff \( g(v) \) and \( v' = v[r] \) and \( \text{Inv}(l')(v') \)

- **delay transition**: \( (l, v) \xrightarrow{d} (l, v + d) \) iff \( \text{Inv}(l)(v + d') \) whenever \( d' \leq d \in R \geq 0 \)
Semantics: Example

\[
\begin{align*}
(\text{off}, x = y = 0) & \xrightarrow{3.5} (\text{off}, x = y = 3.5) \xrightarrow{\text{push}} \\
(\text{on}, x = y = 0) & \xrightarrow{\pi} (\text{on}, x = y = \pi) \xrightarrow{\text{push}} \\
(\text{on}, x = 0, y = \pi) & \xrightarrow{3} (\text{on}, x = 3, y = \pi + 3) \xrightarrow{9-(\pi+3)} \\
(\text{on}, x = 9-(\pi+3), y = 9) & \xrightarrow{\text{click}} (\text{off}, x = 0, y = 9) \ldots
\end{align*}
\]
Talk Outline: Part 1

✓ Timed Automata
✜ Hybrid Automata
❑ CHARON: Hierarchical Specification
❑ Compositionality and Refinement
Hybrid Automata

- Locations or modes (discrete states)
- Initial condition
- Continuous dynamics
- Jump transformation
- Edge guard
- Invariant: hybrid automaton may remain in $l$ as long as $X \in \text{Inv}(l)$

$X \in \text{Init}(l)$

$X \in \text{Inv}(l)$

$dX \in \text{Flow}(l)$

$e : g(X) \geq 0$

$J(X, X')$

$dX \in \text{Flow}(l')$

$X \in \text{Inv}(l')$
Switched Dynamic Systems

continuous dynamics

discrete dynamics

jump dynamics

mode select

integrator

1/s

F1

F2

F3

flow constraints

xdot(t)

e(t)

m(t)

x(t)

m(t)

cont. state

discrete state

discrete event

x(t)

Je

jump dynamics

jump mapping

initial condition

e(t)
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: \((l,x) \rightarrow (l',x')\) 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: \((l,x) \rightarrow (l,x')\) 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: Part 1

✓ Timed Automata
✓ Hybrid Automata
きっかけ 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
Architectural Hierarchy

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

Robot $1$

\text{dTimer}

\text{pos}

local diff analog timer

\text{r2Est}_1
\text{r2Est}_2
\text{r1Est}_1
\text{r1Est}_2

\begin{align*}
\text{pos} &= \text{target} \\
\text{pos.x} &= v \times \cos(\phi) \\
\text{pos.y} &= v \times \sin(\phi)
\end{align*}
Walking Model: Architecture and Agents

- **Input**
  - touch sensors
- **Output**
  - desired angles of each joint
- **Components**
  - Brain: control four legs
  - Four legs: control servo motors
    - Instantiated from the same pattern
Walking Model: Behavior and Modes

\[
\begin{align*}
\text{dx} &= -v \\
& \quad \text{if } x > \text{stride} / 2 \\
\text{dy} &= kv \\
\text{dx} &= kv \\
& \quad \text{if } x < \text{stride} / 2
\end{align*}
\]
CHARON Toolkit
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 $\varepsilon$.
   - Integrate for time $\varepsilon$ and get new state $s$.
   - Return $s$ and $t+\varepsilon$ if invariants were violated.
   - Increment $t = t+\varepsilon$.

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.

\[ error \approx h^{m+1} C \sum_{j=1}^{i-1} \frac{\partial f_i}{\partial x_j} (R - 1) \]
Efficiency gain increases dramatically when simulating systems with complex right-hand sided or tight error tolerances.

Computations for Modular Systems
Talk Outline: Part 1

✓ Timed Automata
✓ Hybrid Automata
✓ CHARON: Hierarchical Specification
⊕ 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. For example, traces (P || Q) = traces(P) intersect 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
$dt = 1$
$dx = x$
$t <= 10$

Mode B
$dt = 1$
$dx = -1$
$t <= 6$

Sample Execution
Sample Trace
Refinement

- Component I refines component S if they have 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

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

Controller'

Normal'

Controller

Normal

Emergency

Refines

level \in [2, 10]

level \in [4, 8]
Compositional Reasoning

Sub-mode refinement

Context refinement
Part 2.
Decidability and Verification
Model Checking of Hybrid Systems

Is finite state analysis possible?
Is reachability problem decidable?

gives rise to the infinite transition system:

Is finite state analysis possible?
Is reachability problem decidable?
Finite Partitioning

Goal: To partition state-space into finitely many equivalence classes so that equivalent states exhibit similar behaviors.
Talk Outline: Part 2

- Preliminaries: Transition Systems
- Timed Automata and Region Graphs
- Equivalences and Finite Quotients
- Decidable Problems
Labeled Transition System $T$

- Set $Q$ of states
- Set $I$ of initial states
- Set $L$ of labels
- Set $\rightarrow$ of labeled transitions of the form $q \xrightarrow{a} q'$
Let $T=(Q,I,L,\rightarrow)$ be a transition system and $\equiv$ be a partitioning of $Q$ (i.e. an equivalence relation on $Q$).

Quotient $T/\equiv$ is transition system:

1. States are equivalence classes of $\equiv$
2. A state $P$ is initial if it contains a state in $I$
3. Set of labels is $L$
4. Transitions: $P \xrightarrow{a} P'$ if $q \xrightarrow{a} q'$ for some $q$ in $P$ and some $q'$ in $P'$
Language Equivalence

- Language of T: Set of possible finite strings over L that can be generated starting from initial states
- T and T’ are language-equivalent iff they generate the same language
- Roughly speaking, language equivalent systems satisfy the same set of “safety” properties
Bisimulation

- Relation $\cong$ on $Q \times Q'$ is a bisimulation iff whenever $q \cong q'$ then
  
  if $q \xrightarrow{a} u$ then for some $u'$, $u \cong u'$ and $q' \xrightarrow{a} u'$, and
  
  if $q' \xrightarrow{a} u'$ then for some $u$, $u \cong u'$ and $q \xrightarrow{a} u$.

- Transition systems $T$ and $T'$ are bisimilar if there exists bisimulation $\cong$ on $Q \times Q'$ such that
  
  For every $q$ in $I$, there is $q'$ in $I'$, $q \cong q'$ and vice versa

- Many equivalent characterizations (e.g. game-theoretic)

- Roughly speaking, bisimilar systems satisfy the same set of branching-time properties (including safety)
Bisimulation Vs Language equivalence

Language equivalent but not bisimilar
Bisimilarity $\rightarrow$ Language equivalence
Timed Vs Time-Abstract Relations

- Transition system associated with a timed/hybrid automaton:
  - Labels on continuous steps are delays in $\mathbb{R}$
  - Actual delays are suppressed (all continuous steps have same label): Time-abstract

- Two versions of language equivalence and two versions of bisimulation

- Time-abstract relations enough to capture untimed properties (e.g. reachability, safety)
Time-abstract Vs Timed

Time-abstract equivalent but not timed equivalent
Timed equivalence -> Time-abstract equivalence
Talk Outline: Part 2

✓ Preliminaries: Transition Systems
✓ Timed Automata and Region Graphs
☐ Equivalences and Finite Quotients
☐ Decidable Problems
Timed Automata (Recap)

- Only continuous variables are timers
- Invariants and Guards: $x < \text{const}, x = \text{const}$
- Actions: $x := 0$
- Can express lower and upper bounds on delays
Regions
 Finite partitioning of state space

Definition

An equivalence class (i.e. a region) in fact there is only a finite number of regions!!

\[ w \equiv w' \text{ iff they satisfy the same set of constraints of the form } \]
\[ x_i < c, x_i = c, x_i - x_j < c, x_i - x_j = c \text{ for } c \leq \text{largest const relevant to } x_i \]

An equivalence class (i.e. a region) in fact there is only a finite number of regions!!

Alur, Dill, 90
Region Operations

An equivalence class (i.e. a region)

Successor regions, Succ(r)

Reset regions

1 2 3 x

{y}r

{r}r

1 2 y
Properties of Regions

- The region equivalence relation \( \cong \) is a time-abstract bisimulation:
  - Action transitions: If \( w \cong v \) and \( (l,w) \rightarrow_a (l',w') \) for some \( w' \), then \( \exists v' \cong w' \) s.t. \( (l,v) \rightarrow_a (l',v') \)
  - Delay transitions: If \( w \cong v \) then for all real numbers \( d \), there exists \( d' \) s.t. \( w+d \cong v+d' \)

- If \( w \cong v \) then \( (l,w) \) and \( (l,v) \) satisfy the same temporal logic formulas
Region graph of a simple timed automata

(a) $l \xrightarrow{x \geq 2 \{x\}}$

(b)

- A: $l \ x = 0$
- B: $l \ 0 < x < 1$
- C: $l \ x = 1$
- D: $l \ 1 < x < 2$
- E: $l \ x = 2$
- F: $l \ x > 2$
Region Graphs (Summary)

- Finite quotient of timed automaton that is time-abstract bisimilar
- Number of regions: (# of locations) times (product of all constants) times (factorial of number of clocks)
- Precise complexity class of reachability problem: PSPACE (basically, exponential dependence of clocks/constants unavoidable)
Talk Outline: Part 2

✓ Preliminaries: Transition Systems
✓ Timed Automata and Region Graphs
.Angle Equivalences and Finite Quotients
.D Decidable Problems
Multi-rate Automata

- Modest extension of timed automata
  - Dynamics of the form $dx = \text{const}$ (rate of a clock is same in all locations)
  - Guards and invariants: $x < \text{const}, \ x > \text{const}$
  - Resets: $x := \text{const}$

- Simple translation to timed automata that gives time-abstract bisimilar system by scaling

\[ dx = 2 \quad dy = 3 \quad x > 5 \text{ and } y < 1 \]
\[ du = 1 \quad dv = 1 \quad u > 5/2 \text{ and } v < 1/3 \]
Rectangular Automata

- Interesting extension of timed automata
  - Dynamics of the form $dx$ in const interval (rate-bounds of a clock same in all locations)
  - Guards/invariants/resets as before
- Translation to multi-rate automata that gives time-abstract language-equiv system

$dx$ in $[2,3]
\begin{align*}
x > 5 & \quad x < 2 \\
\end{align*}$

$dv = 3$
$du = 2$
$v > 5, u := 5$
$u < 2, v := 2$

Puri, Henzinger, 95
Rectangular Automata may not have finite bisimilar quotients!
Continuous systems

- Given an initial partitioning $P$ of $\mathbb{R}^k$ and continuous dynamics $dX = F(X)$, is there a refinement of $P$ that is time-abstract bisimilar to original system?

- Counter-example: Spiral. Initial partition: $-5 \leq x < 0$ and $0 < x \leq +5$
O-minimal Structures

- A structure over \( \mathbb{R} \) is order-minimal if every definable subset is a finite union of points and open intervals.

- O-minimal structures
  - \( \mathbb{R} \) with \(<, +, -, 0, 1\) (polyhedral sets)
  - \( \mathbb{R} \) with \(<, +, -, *, e^x, 0, 1\) (semialgebraic sets and exponential trajectories)
  - And many more such as sub-analytic
O-minimal Hybrid Systems

- Guards, Flows, Invariants definable in the same o-minimal structure
- Edges reset all variables (to constants or intervals)
- Thm: O-minimal hybrid systems have finite time-abstract bisimilarity quotients

Pappas, Lafferriere, Sastry, 98
Talk Outline: Part 2

✓ Preliminaries: Transition Systems
✓ Timed Automata and Region Graphs
✓ Equivalences and Finite Quotients
✿ Decidable Problems
Decidable Problems

- Model checking branching-time properties of timed automata
- Reachability in rectangular automata
- Timed bisimilarity: are given two timed automata bisimilar?
- Optimization: Compute shortest paths (e.g. minimum time reachability) in timed automata with costs on locations and edges
- Controller synthesis: Computing winning strategies in timed automata with controllable and uncontrollable transitions
Undecidable Reachability Problems

- Timed automata + linear expressions as guards
- Multi-rate automata with comparisons among clocks as guards
- Timed automata + stop-watches (i.e. clocks that can have rates 0 or 1)

Many such results
Proofs by encoding Turing machines/2-counter machines
Sharp boundary for decidability understood
Air-traffic Control Problem as Weighted timed automaton

Start
  c₀

wait1
  w₁ : y < 2
  c₁ : x < 1, y < 1
  y := 0

hold1
  w’₁
  c₃ : x := 0
  y > 1

land2
  c₀ + w₁
  1 < x < 2
  y > 1

x > 1

wait2
  w₂ : x < 2
  c₂ : x < 1, y < 1
  x := 0

hold2
  w’₂
  c₄ : y := 0
  y > 1

Land1
  c₀ + w₂
  1 < y < 2
  x > 1

Done

y > 1

1 < x < 2
Shortest Paths in WTA

- Optimum solution may only be a limit
- Region graph construction not enough
- Algorithm
  1. Reduce to Parametric Shortest Path Problem on graphs (PSP)
  2. Solve PSP
Summary

- Decidability only when simple dynamics or decoupled dynamics
- Theory of equivalences useful in understanding structural properties
Part 3.
Reachability Analysis
Talk Outline: Part 3

- Symbolic Reachability Analysis
- Timed Automata (Kronos, Uppaal)
- Linear Hybrid Automata (HyTech)
- Polyhedral Flow-pipe Approximations (CheckMate)
- Abstraction (Charon, CEGAR, SAL)
Reachability Problem

Model variables $X = \{x_1, \ldots, x_n\}$

Each var is of finite type, say, boolean

Initialization: $I(X)$ condition over $X$

Update: $T(X,X')$

How new vars $X'$ are related to old vars $X$ as a result of executing one step of the program

Target set: $F(X)$

Computational problem:

Can $F$ be satisfied starting with $I$ by repeatedly applying $T$?

Graph Search problem
Symbolic Solution

Data type: region to represent state-sets

$R := I(X)$

Repeat

    If $R$ intersects $F$ report “yes”
    Else if $R$ contains $\text{Post}(R)$ report “no”
    Else $R := R \cup \text{Post}(R)$

$\text{Post}(R)$: Set of successors of states in $R$

Termination may or may not be guaranteed
Symbolic Representations

- **Necessary operations on Regions**
  - Union
  - Intersection
  - Negation
  - Projection
  - Renaming
  - Equality/containment test
  - Emptiness test

- **Different choices for different classes**
  - BDDs for boolean variables in hardware verification
  - Size of representation as opposed to number of states
Ordered Binary Decision Diagrams

Popular representations for Boolean functions

Like a decision graph
No redundant nodes
No isomorphic subgraphs
Variables tested in fixed order

Function: \((a \text{ and } b) \text{ or } (c \text{ and } d)\)

Key properties:
- Canonical!
- Size depends on choice of ordering of variables
- Operations such as union/intersection are efficient
Example: Cache consistency: Gigamax

Real design of a distributed multiprocessor

Global bus

UIC

UIC

M

P

......

Cluster bus

M

P

......

Read-shared/read-owned/write-invalid/write-shared/...

Deadlock found using SMV

Similar successes: IEEE Futurebus+ standard, network RFCs
Reachability for Hybrid Systems

- Same algorithm works in principle
- What’s a suitable representation of regions?
  - Region: subset of $\mathbb{R}^k$
  - Main problem: handling continuous dynamics
- Precise solutions available for restricted continuous dynamics
  - Timed automata
  - Linear hybrid automata
- Even for linear systems, over-approximations of reachable set needed
Talk Outline: Part 3

☑ Symbolic Reachability Analysis

☒ Timed Automata (Kronos, Uppaal)

☐ Linear Hybrid Automata (HyTech)

☐ Polyhedral Flow-pipe Approximations (CheckMate)

☐ Abstraction (Charon, CEGAR, SAL)
Timed Automata

- Only continuous variables are timers
- Invariants and Guards: $x < \text{const}, \ x > = \text{const}$
- Actions: $x := 0$
- Reachability is decidable
- Clustering of regions into zones desirable in practice
- Tools: Uppaal, Kronos, RED ...
- Symbolic representation: matrices
- Techniques to construct timed abstractions of general hybrid systems
Zones
Symbolic computation

State
(n, x=3.2, y=2.5)

Symbolic state (set)
(n, 1≤x≤4, 1≤y≤3)

Zone: conjunction of
x-y<=n, x<=>n

Diagram: Two coordinate axes with a point and a shaded area representing the state and symbolic state respectively.
Symbolic Transitions

Thus \((n, 1 \leq x \leq 4, 1 \leq y \leq 3) \implies (m, 3 < x, y = 0)\)
Canonical Data-structures for Zones

Difference Bounded Matrices

When are two sets of constraints equivalent?

D1
\[
\begin{align*}
x & \leq 1 \\
y-x & \leq 2 \\
z-y & \leq 2 \\
z & \leq 9
\end{align*}
\]

D2
\[
\begin{align*}
x & \leq 1 \\
y-x & \leq 2 \\
y & \leq 3 \\
z-y & \leq 2 \\
z & \leq 7
\end{align*}
\]
Difference Bounds Matrices

- Matrix representation of constraints (bounds on a single clock or difference between 2 clocks)
- Reduced form obtained by running all-pairs shortest path algorithm
- Reduced DBM is canonical
- Operations such as reset, time-successor, inclusion, intersection are efficient
- Popular choice in timed-automata-based tools
Talk Outline: Part 3

- Symbolic Reachability Analysis
- Timed Automata (Kronos, Uppaal)
- Linear Hybrid Automata (HyTech)
- Polyhedral Flow-pipe Approximations (CheckMate)
- Abstraction (Charon, CEGAR, SAL)
Reachability Analysis for Dynamical Systems

- **Goal:** Given an initial region, compute whether a bad state can be reached
- **Key step** is to compute $\text{Reach}(X)$ for a given set $X$ under $\frac{dx}{dt} = f(x)$
Linear Hybrid Automata

- Invariants and guards: linear \((Ax \leq b)\)
- Actions: linear transforms \((x := Ax)\)
- Dynamics: time-invariant, state-independent
  specified by a convex polytope constraining rates
  E.g. \(2 < x \leq 3, \dot{x} = \dot{y}\)

- Tools: HyTech
- Symbolic representation: Polyhedra
- Methodology: abstract dynamics by differential
  inclusions bounding rates
Example LHA
Gate for a railroad controller

- Open: $h = 90$, $dh = 0$
- Lowering: $h \geq 0$, $-10 < dh < -9$
- Raising: $h \leq 90$, $8 < dh < 10$
- Closed: $h = 0$, $dh = 0$
Reachability Computation

Basic element: (location l, polyhedron p)
Set of visited states: a list of (l,p) pairs

Key steps:
- Compute “discrete” successors of (l,p)
- Compute “continuous” successor of (l,p)
- Check if p intersects with “bad” region
- Check if newly found p is covered by already visited polyhedra p1,..., pk (expensive!)
Computing Discrete Successors

- Intersect $p$ with $g$ (result $r$ is a polyhedron)
- Apply linear transformation $a$ to $r$ (result $r'$ is a polyhedron)
- Successor is $(l',r')$
Thm: If initial set $p$, invariant $I$, and rate constraint $r$, are polyhedra, then set of reachable states is a polyhedron (and computable)

- Basically, apply extremal rates to vertices of $p$
Summary: Linear Hybrid Automata

- HyTech implements this strategy
- Core computation: manipulation of polyhedra
- Bottlenecks
  - proliferation of polyhedra (unions)
  - computing with higher dimensional polyhedra

- Many case studies (active structure control, Philips audio control protocol, steam boiler...
Talk Outline: Part 3

- Symbolic Reachability Analysis
- Timed Automata (Kronos, Uppaal)
- Linear Hybrid Automata (HyTech)
- Polyhedral Flow-pipe Approximations (CheckMate)
- Abstraction (Charon, CEGAR, SAL)
Beyond LHA

- Exact computation with polyhedra is limiting.
- If dynamics is $dX=AX$, and $P$ is a polyhedron, Reach($P$) is not a polyhedron

Solutions:

- Approximate Reach($P$) with an enclosing convex polyhedron: Checkmate (Krogh)
- Approximate Reach($P$) with an enclosing (non-convex) orthogonal polyhedron: $d/dt$ (Dang/Maler)
- Level sets method (Greenstreet, Tomlin)
- Use ellipsoids for representation of sets (Kurzhanski)
Polyhedral Flow Pipe Approximations

- divide $R_{[0,T]}(X_0)$ into $[t_k, t_{k+1}]$ segments
- enclose each segment with a convex polytope
- $R^M_{[0,T]}(X_0) = \text{union of polytopes}$
Wrapping Hyperplanes Around a Set

Step 1:
Choose normal vectors, $c_1, \ldots, c_m$
Wrapping Hyperplanes Around a Set

Step 2:
Compute optimal $d$ in $Cx \leq d$,
$C^T = [c_1 \cdots c_m]$:

$$d_i = \max_{x \in S} c_i^T x$$
Wrapping a Flow Pipe Segment

Given normal vectors $c_i$, we wrap $R_{[t_k,t_{k+1}]}(X_0)$ in a polytope by solving for each $i$

$$d_i = \max_{x_0,t} c_i^T x(t,x_0)$$

s.t.
$$x_0 \in X_0$$
$$t \in [t_k,t_{k+1}]$$

Optimization problem is solved by embedding simulation into objective function computation
Improvements for Linear Systems

- $\dot{x} = Ax \Rightarrow x(t, x_0) = e^{At}x_0$
- No longer need to embed simulation into optimization
- Flow pipe segment computation depends only on time step $\Delta t$
- A segment can be obtained by applying $e^{At}$ to another segment of the same $\Delta t$

$$\hat{R}_{[t,t+\Delta t]}(X_0) = e^{At}\hat{R}_{[0,\Delta t]}(X_0)$$
Example: Van der Pol Equation

Van der Pol Equation

\[ \dot{x}_1 = x_2 \]
\[ \dot{x}_2 = -0.2(x_1^2 - 1)x_2 - x_1 \]

Initial Set

\[ X_0 = \{0.8 \leq x_1 \leq 1, x_2 = 0\} \]

Uniform time step

\[ \Delta t_k = 0.5 \]
Summary: Flow Pipe Approximation

- Applies in arbitrary dimensions
- Approximation error doesn't grow with time
- Estimation error (Hausdorff distance) can be made arbitrarily small with $\Delta t < \delta$ and size of $X_0 < \delta$
- Integrated into a complete verification tool (CheckMate)
Talk Outline: Part 3

✓ Symbolic Reachability Analysis
✓ Timed Automata (Kronos, Uppaal)
✓ Linear Hybrid Automata (HyTech)
✓ Polyhedral Flow-pipe Approximations (CheckMate)
 bü Abstraction (Charon, CEGAR, SAL)
Abstraction and Refinement

- Abstraction-based verification
  - Given a model $M$, build an abstraction $A$
  - Check $A$ for violation of properties
  - Either $A$ is safe, or is adequate to indicate a bug in $M$, or gives false negatives (in that case, refine the abstraction and repeat)

- Many projects exploring abstraction-based verification for hybrid systems
  - Predicate abstraction (Charon at Penn)
  - Counter-example guided abstraction refinement (CEGAR at CMU)
  - Qualitative abstraction using symbolic derivatives (SAL at SRI)
Program Abstraction

```c
int x, y;
if x>0 {
    ........
    y:=x+1
    ........
} else {
    ........
    y:=x+1
    ........
}
```

```c
bool bx, by;
if bx {
    ........
    by:=true
    ........
} else {
    ........
    by:={true,false}
    ........
}
```

Predicate Abstraction:
- bx: x>0
- by: y>0
Predicate Abstraction

- Input is a hybrid automaton and a set of $k$ boolean predicates, e.g. $x+y > 5-z$.
- The partitioning of the concrete state space is specified by the user-defined $k$ predicates.
Overview of the Approach

Hybrid system

Boolean predicates

Search in abstract space

Property holds

No! Counter-example

Real counter-example found

Analyze counter-example

Safety property

additional predicates
Thermostat Example

- 2 variables: T (Temperature) and t (timer)
- Initially: t = 0 , 5 <= T <= 10
- Unsafe: Check, T <= 4.5

Heat
\[ dT = 2 \]
\[ T \leq 10, t \leq 3 \]

Cool
\[ dT = -T \]
\[ T \geq 5 \]

Check
\[ dT = -T/2 \]
\[ t \leq 1 \]
Thermostat Abstraction

10 predicates: \( t \leq 0 \), \( t \geq 0.5 \), \( \ldots \), \( T \geq 5 \), \( T \leq 6 \), \( \ldots \)

Only 36 “valid” continuous abstract states
A Sample Abstract Path

35 abstract states reachable.

All states are safe, thus the concrete system is also safe.
What do we analyze?

- **Input hybrid automaton**
  - Guards/updates/invariants are linear expressions
  - Dynamics: \( d\mathbf{x} = A \mathbf{x} + B \mathbf{u} \)
  - \( \mathbf{u} \): uncertain input within a bounded range

- **Initial condition and bad region (both linear)**

- **User provides set of linear predicates for abstraction**

- **Builds on routines for manipulating polyhedra from \( d/dt \)**
Search Algorithm

- Starting with initial abstract states, depth-first search to discover bad state
- On-the-fly search: abstract states analyzed for outgoing transitions only on demand
- Given an abstract state \( s \), first examine outgoing edges and check if they lead to new abstract states
- When discrete transitions do not yield new states, compute continuous successors
  - For abstract state \( s \), compute polyhedral slices at times \( r, 2r, 3r \)
  - Compute bounding polyhedra for reach sets from \( kr \) to \( (k+1)r \)
  - For every intermediate result, check intersection with new abstract states
Why use this approach?

- Reach(X) needs to be computed only for abstract states X and not intermediate regions of unpredictable shapes/complexity
- No need to compute Reach (X). Goal is to find one new abstract state reachable from X, partial results are of great use
  - Simulate vertices
  - Consider time-slices at discrete times
- Our focus is on search strategies to make progress in the abstract state-space
- Initial implementation in C++ with promising results
Counter-example Analysis

- How to efficiently check whether the abstract counter example is a real one?
- Automatic discovery of new predicates
  - Finding a separating hyperplane for two sets of polyhedra
  - Greedy heuristic picks a subset of faces
  - The same sequence of abstract states should be ruled out
Computing Separating Predicates

Given a spurious counter-example and the series of concretely reachable sub-spaces, find a small new set of predicates that will disallow a similar counter-example to reappear.
Thermostat Example

- Remove predicate $t \leq 0$ from predicate set.
- First run: Spurious counter-example is found!
- Separation routine suggest to use 4 predicates:
  
  1. $0.979265 \times T + 0.202584 \times t \leq 9.34423$
  2. $0.872555 \times T + 0.488515 \times t \leq 8.16961$
  3. $0.428587 \times T + 0.9035 \times t \leq 4.11184$
  4. $-0.0680518 \times T + 0.997682 \times t \leq -0.439659$

- Second and third run still find counter-examples. One of 15 suggested predicates:
  
  $0.0139043 \times T + 0.999903 \times t \leq 0.152558$

- 28 predicates are enough to prove safety in fourth iteration with 358 reachable states.
Bounded Completeness

- Simulation can prove unsafe behavior.
- Safety can only be shown using verification, but the problem is undecidable.
- Predicate abstraction introduces errors by
  - approximating reachable sets with polyhedra
  - coarse abstraction using predicates
- Predicate abstraction can prove "bounded safety"
  - upto a specified number $n$ of discrete switches and
  - upto a specified total time flow of $t$ units
  - provided $\text{distance}(\text{Reach},\text{Unsafe}) > \Delta$
Refining the Verifier

- Improved search strategy to guide the search
  - A variety of distance measures to select the next abstract state to continue DFS
- Techniques for eliminating abstract states early from consideration while computing continuous successors
- Binary-space partitioning (BSP) based representation of abstract states
The agent platoon-(i-1) generates a trace for its velocity using its acceleration which is treated as uncertain input to the system: 

\[ v_i \in [a_{\text{min}}, a_{\text{max}}]. \]

The agent controller of platoon-(i) tries to maintain the distance to the previous platoon-(i-1) to some safety distance which depends on its own acceleration and velocity.
Variables:
- distance
- velLeader
- velFollower
- acceleration
  (uncertain input)
Verification Results

- The Predicate Abstraction tool found that the MoBIES Automotive OEP Vehicle-To-Vehicle longitudinal controller satisfies collision-free behavior, if the following initial region is assumed:
  - $20 \leq \text{distance} \leq 1000$
  - $15 \leq \text{velLead} \leq 18$
  - $20 \leq \text{velFollow} \leq 25$

- The tool used 17 predicates, and found 16 reachable abstract states.
- The computation took approximately 4 minutes on a Pentium II with negligible memory resource needs.
Wrap-Up

- Modeling and Analysis in symbiosis
- Progress on safety verification by combining symbolic representations and abstraction
- Many application domains for hybrid systems
- Current Focus: Synthesizing software from hybrid models