Hybrid Systems
Modeling and Verification

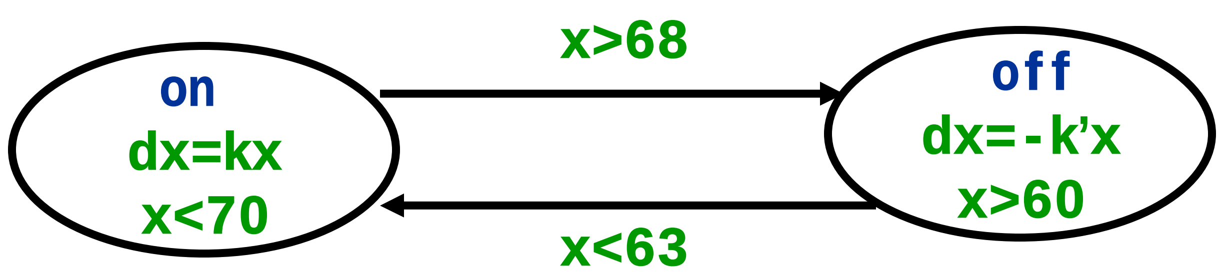
Rajeev Alur

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

Dynamics & Verification Workshop, July 2001

What are Hybrid Systems?

State machines + Dynamical systems



Hybrid Systems in Applications

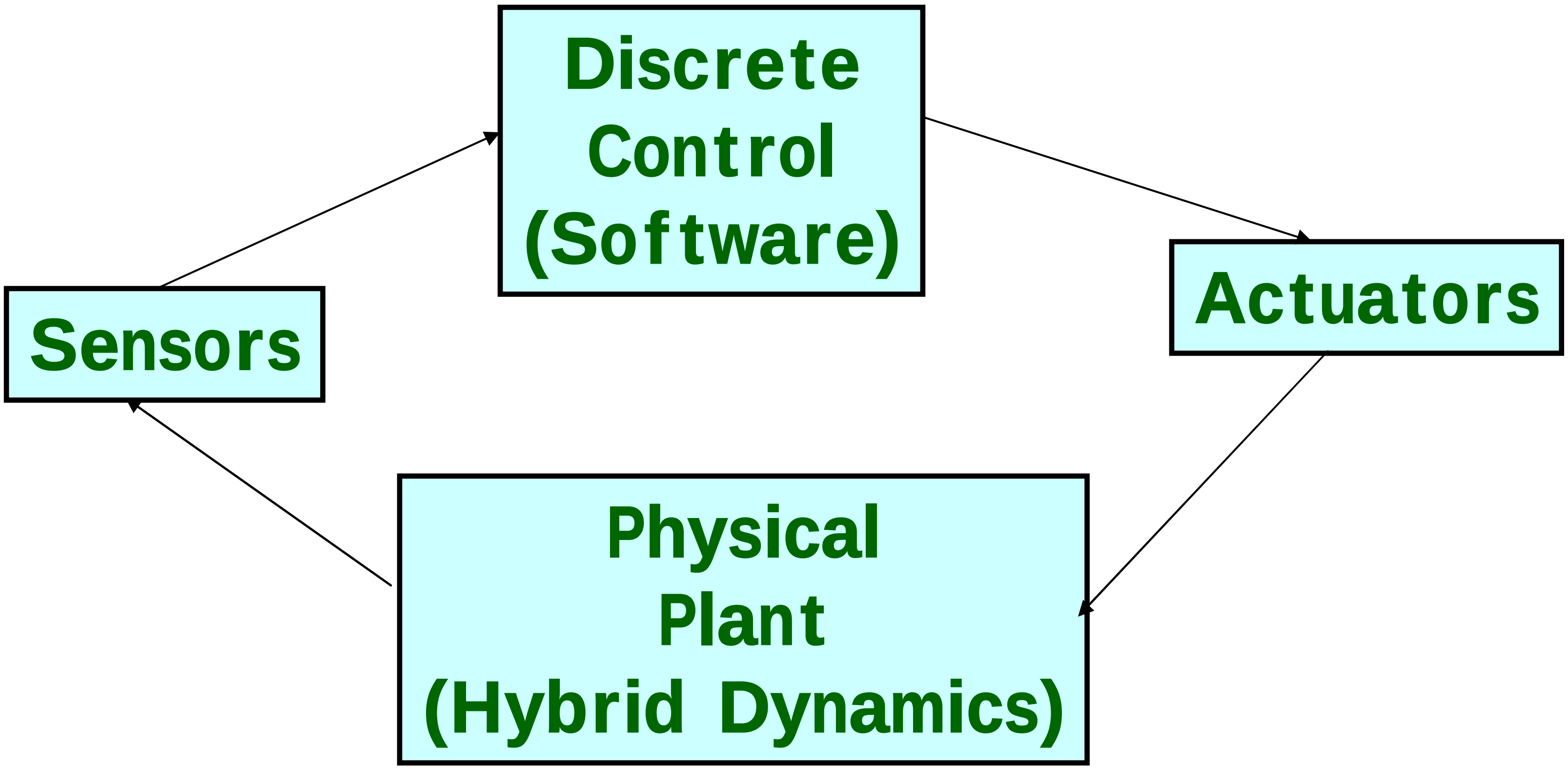
Computer control systems

- 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



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

	Continuous-State Systems	*Discrete-State Systems*
Models	differential equations, transfer functions, etc.	automata, Petri nets, Statecharts, etc.
Analytical Tools	Lyapunov functions, eigenvalue analysis, etc.	Boolean algebra, formal logics, verification, etc.
Software Tools	Matlab, Matrix$_X$, VisSim, etc.,	Statemate, Rational Rose, SMV, etc.

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

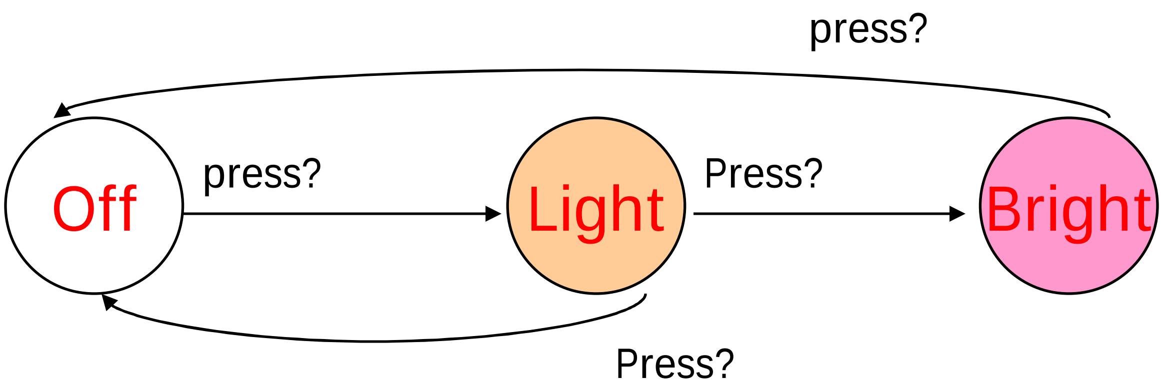
- Thanks for providing powerpoint slides
  - Colleagues at Penn
  - Bruce Krogh at CMU
  - Kim Larsen at Aalborg
- Caution:
  - Only a partial coverage area
  - Mostly computer-science centric
  - Biased towards my current interests
  - Apologies for incomplete references and notational variations

Lecture 1.
Modeling and Semantics

Talk Outline

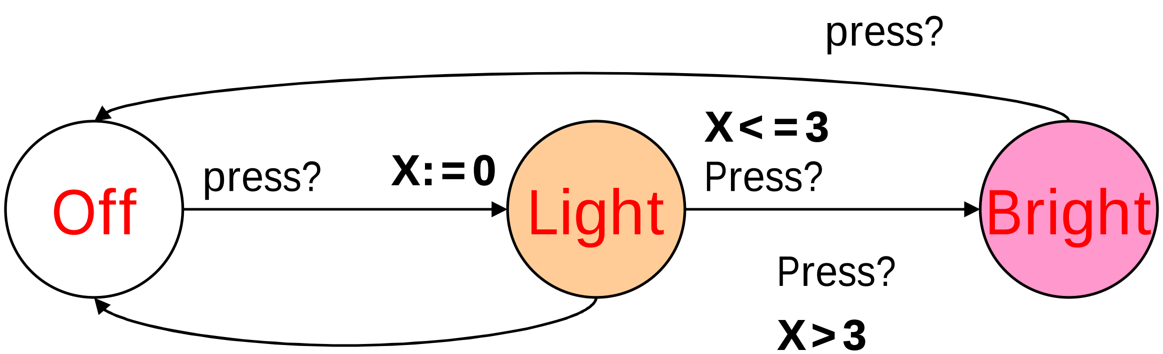
- Timed Automata
- Hybrid Automata
- CHARON: Hierarchical Specification
- Modular Analysis

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

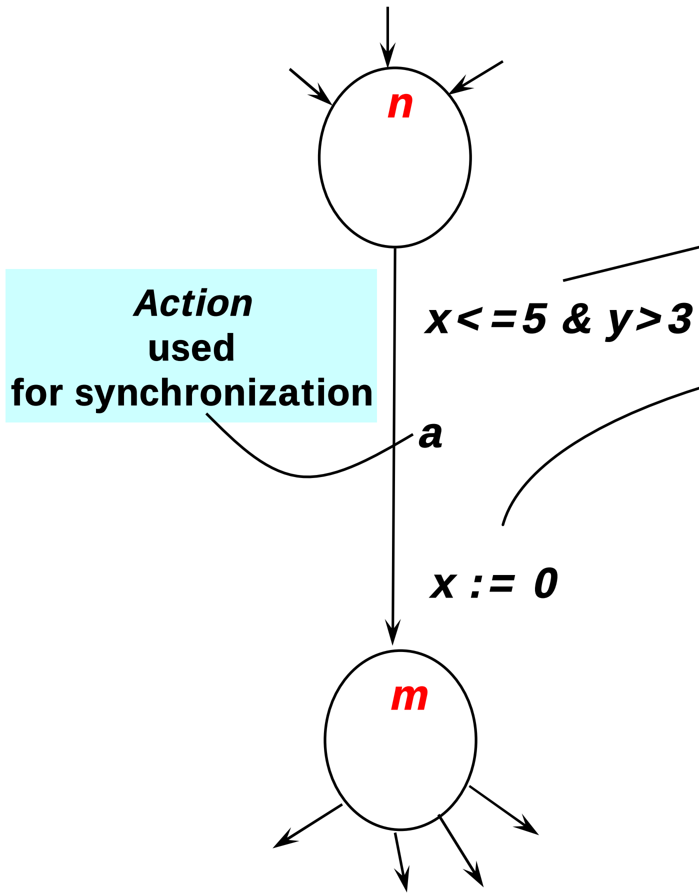


Solution: Add real-valued clock **x**

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

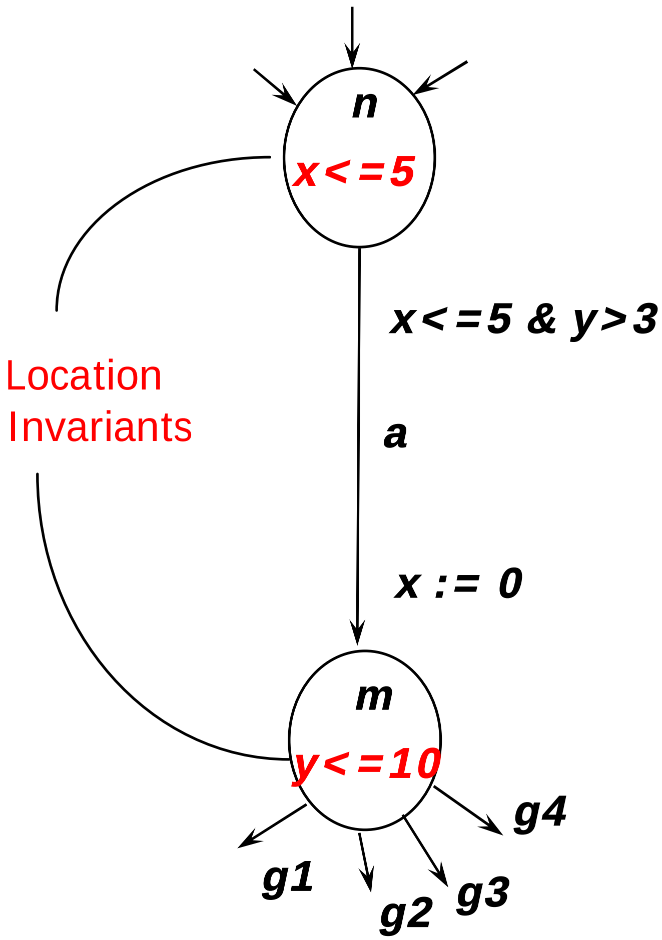
(***location*** , ***x***=v , ***y***=u)   where v,u are in **R**

Transitions

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

$(\, n \,,\, x=2.4 \,,\, y=3.1415 \,) \xrightarrow{e(1.1)} (\, n \,,\, x=3.5 \,,\, y=4.2415 \,)$

Adding Invariants



Clocks: x, y

Transitions

$(\, n \,,\, x=2.4 \,,\, y=3.1415 \,) \xrightarrow{\cancel{e(3.2)}}$

$(\, n \,,\, x=2.4 \,,\, y=3.1415 \,) \xrightarrow{e(1.1)} (\, n \,,\, x=3.5 \,,\, y=4.2415 \,)$

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

$$\alpha ::= x \prec c \;\Big|\; x - y \prec c \;\Big|\; \neg\alpha \;\Big|\; (\alpha \;\wedge\; \alpha)$$

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

What can you express:

Constant lower and upper bounds on delays

Why the restricted syntax:

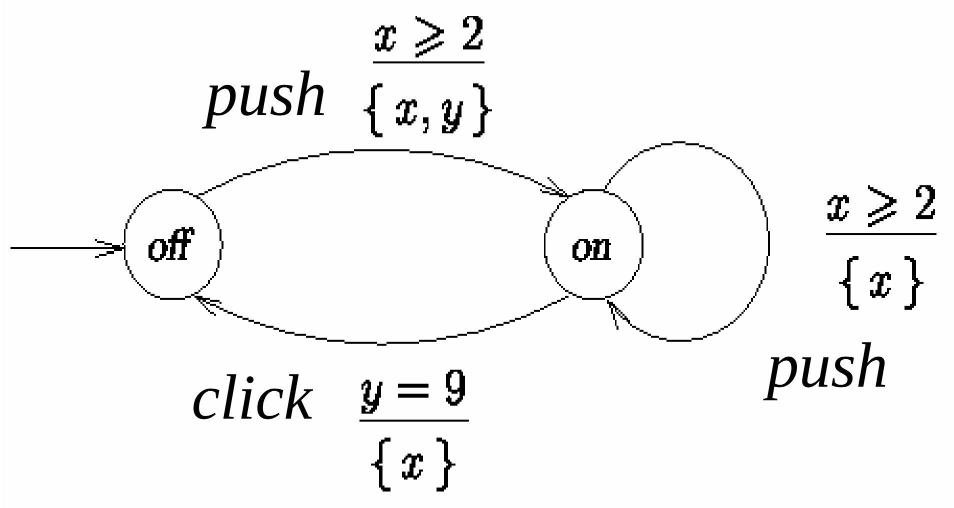
slight generalizations (e.g. allowing x=2y) lead to undecidable model checking problems

Timed Automata

A *timed automaton* $\mathcal{A}$ is a tuple $(L, l_0, E, \mathit{Label}, C, \mathit{clocks}, \mathit{guard}, \mathit{inv})$ 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
- $\mathit{Label} : L \longrightarrow 2^{AP}$, a function that assigns to each location $l \in L$ a set $\mathit{Label}(l)$ of atomic propositions
- C, a finite set of clocks
- $\mathit{clocks} : E \longrightarrow 2^C$, a function that assigns to each edge $e \in E$ a set of clocks $\mathit{clocks}(e)$
- $\mathit{guard} : E \longrightarrow \Psi(C)$, a function that labels each edge $e \in E$ with a clock constraint $\mathit{guard}(e)$ over C, and
- $\mathit{inv} : L \longrightarrow \Psi(C)$, a function that assigns to each location an *invariant*.

Light Switch



- Switch may be turned on whenever at least 2 time units has 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*: $(l, v) \quad \textit{where} \quad l \in L \ \textit{and} \ v \in V(C)$
- Operational semantics of timed automaton is a *labeled transition system* $(S, \rightarrow)$

  where S is the set of all states

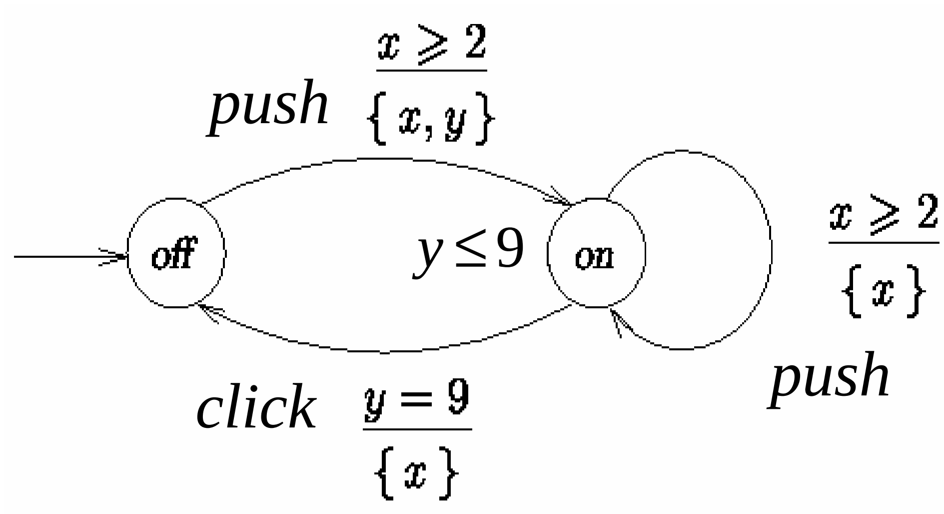
- *action transition* $(l, v) \xrightarrow{a} (l', v') \ \textit{iff}$ $\quad l \xrightarrow{g \quad a \quad r} l'$

  $g(v) \ \textit{and} \ v' = v[r] \ \textit{and} \ Inv(l')(v')$

- *delay Transition* $(l, v) \xrightarrow{d} (l, v + d) \ \textit{iff}$

  $Inv(l)(v + d') \ \textit{whenever} \ d' \leq d \in R_{\geq 0}$

Semantics: Example



$$(off, x = y = 0) \xrightarrow{3.5} (off, x = y = 3.5) \xrightarrow{push}$$

$$(on, x = y = 0) \xrightarrow{\pi} (on, x = y = \pi) \xrightarrow{push}$$

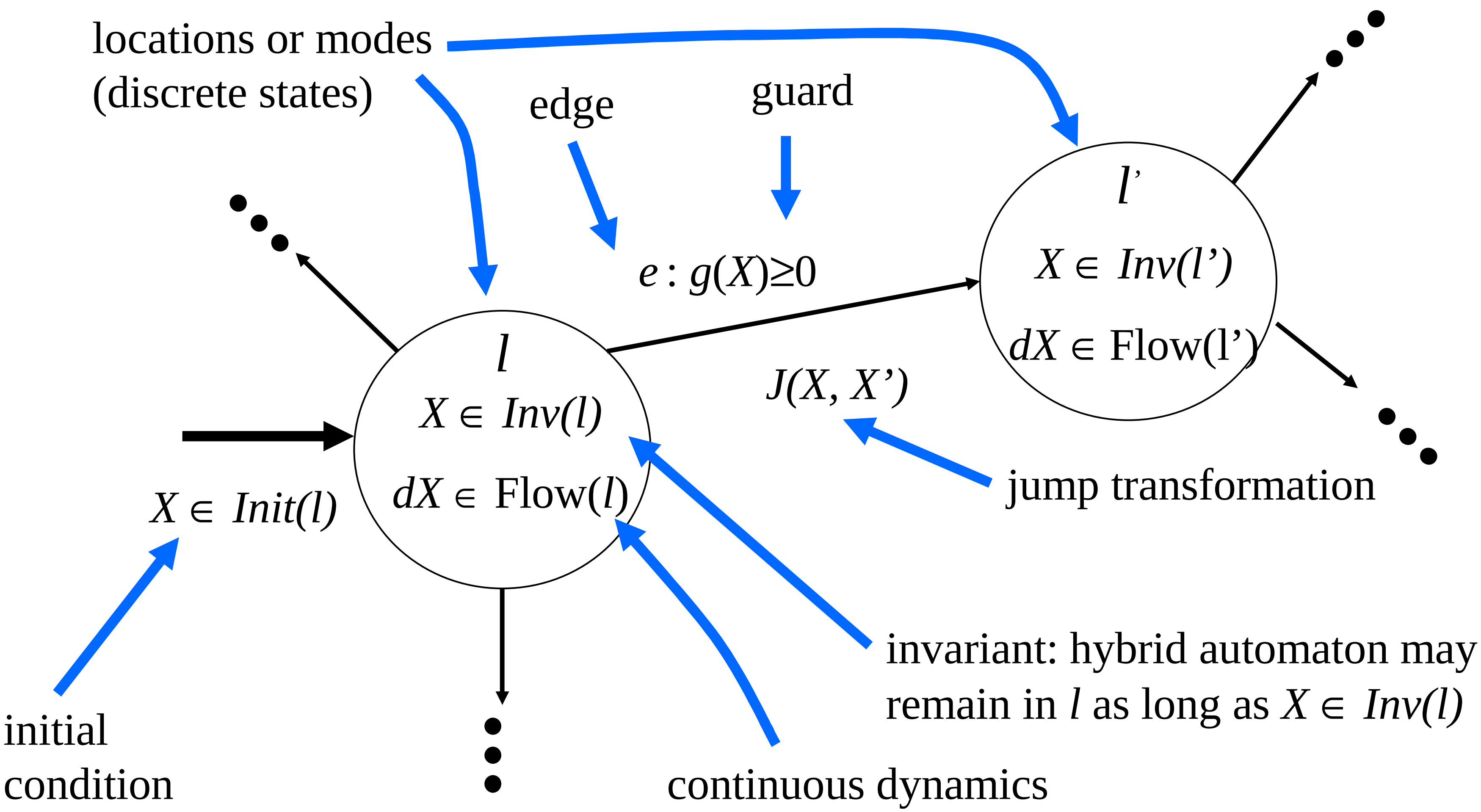
$$(on, x = 0, y = \pi) \xrightarrow{3} (on, x = 3, y = \pi + 3) \xrightarrow{9-(\pi+3)}$$

$$(on, x = 9 - (\pi + 3), y = 9) \xrightarrow{click} (off, x = 0, y = 9) \ldots$$

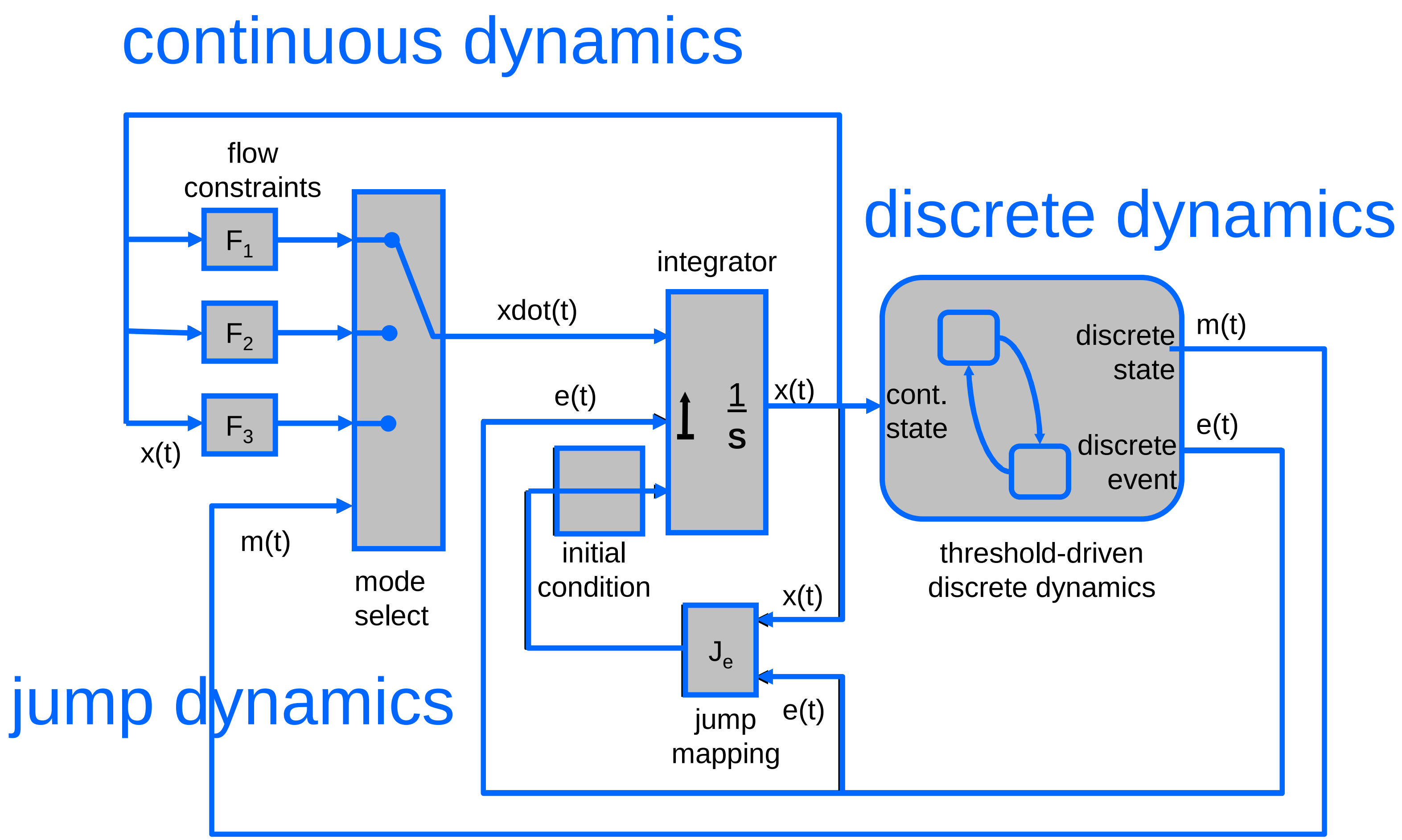
Talk Outline

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

Hybrid Automata



Switched Dynamic Systems



Hybrid Automata

- Set L of of locations, and set E of edges
- Set X of k continuous variables
- State space: L x R^k, Region: subset of R^k
- For each location l,
  - Initial states: region Init(l)
  - Invariant: region Inv(l)
  - Continuous dynamics: dX in Flow(l)(X)
- For each edge e from location l to location l'
  - Guard: region Guard(e)
  - Update relation over R^k x R^k
  - Synchronization labels (communication information)

(Finite) Executions of Hybrid Automata

- State: (l, x) such that x satisfies Inv(l)
- Initialization: (l,x) s.t. x satisfies Init(l)
- Two types of state updates
- Discrete switches: (l,x) –a-> (l',x') if there is an a-labeled edge e from l to l' s.t. x satisfies Guard(e) and (x,x') satisfies update relation Jump(e)
- Continuous flows: (l,x) –f-> (l,x') where f is a continuous function from [0,δ] s.t. f(0)=x, f(δ)=x', and for all t<=δ, f(t) satisfies Inv(l) and df(t) satisfies 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

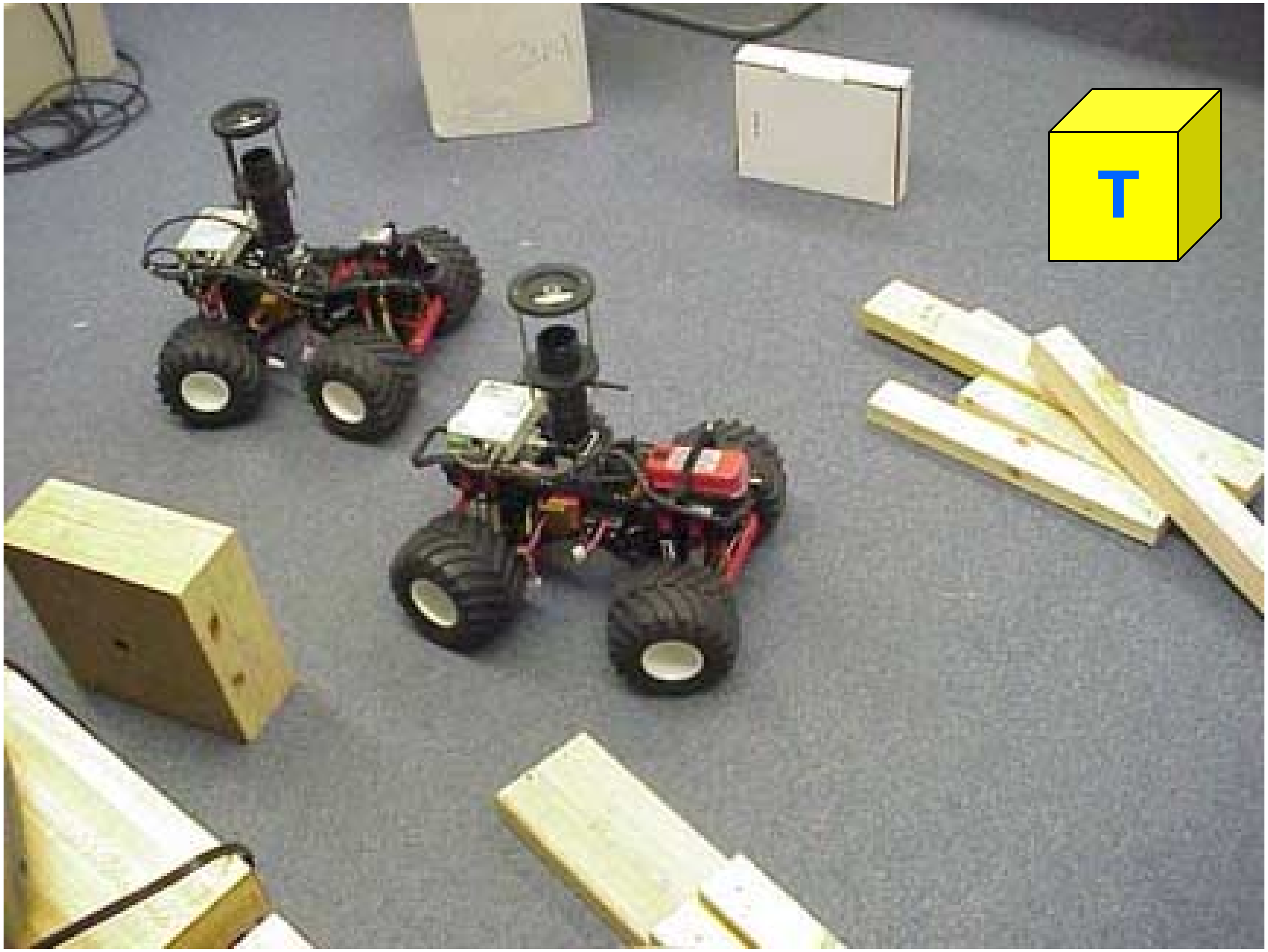
- Emerging notations: UML-RT,Stateflow
  - Visual
  - Hierarchical modeling of control flow
  - Object oriented
- Prototyping/modeling but no analysis
  - Ad-hoc, informal features
  - No support for abstraction

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

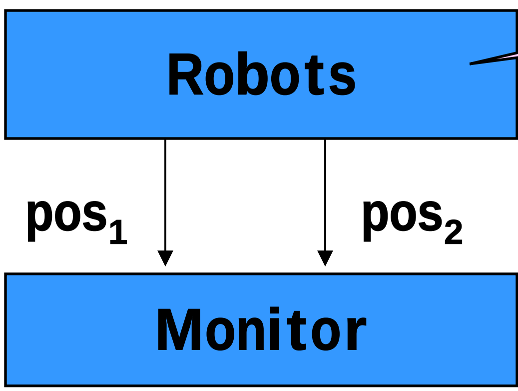
Robot Team Approaching a Target



Architectural Hierarchy

write diff analog position pos_1, pos_2

class position { float x; float y;}



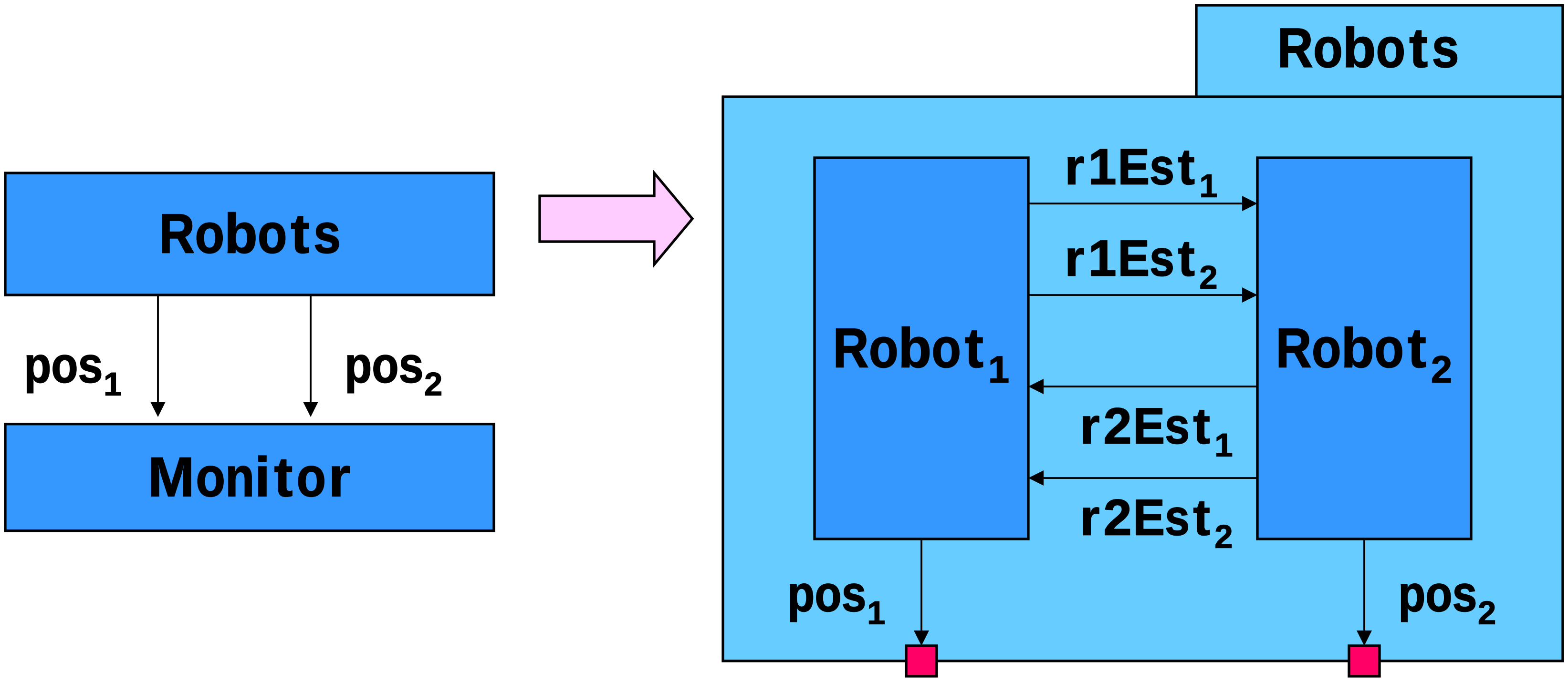
Variables Specifiers

Range: discrete/analog

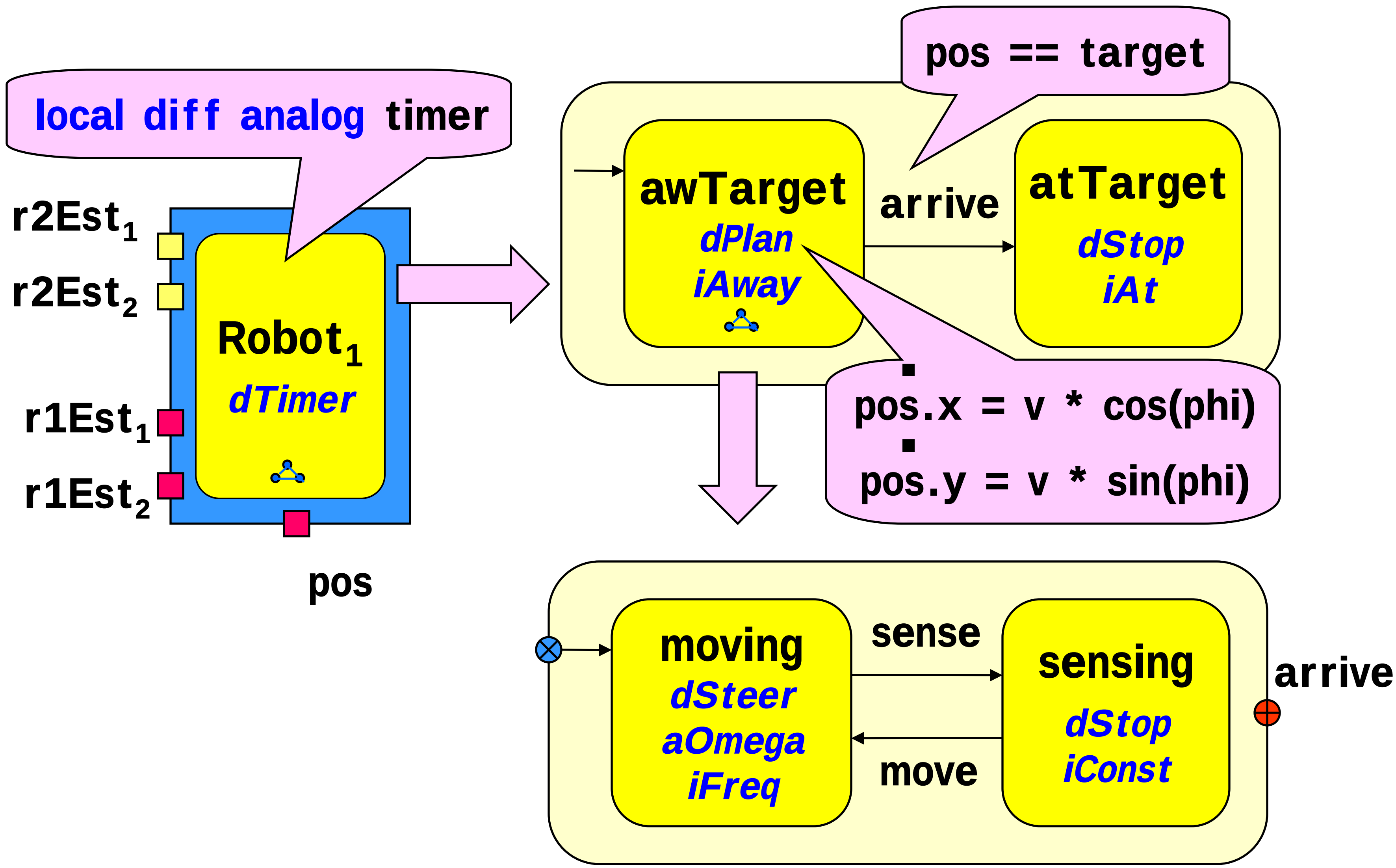
Computation: diff/alg

Access: read/write/local

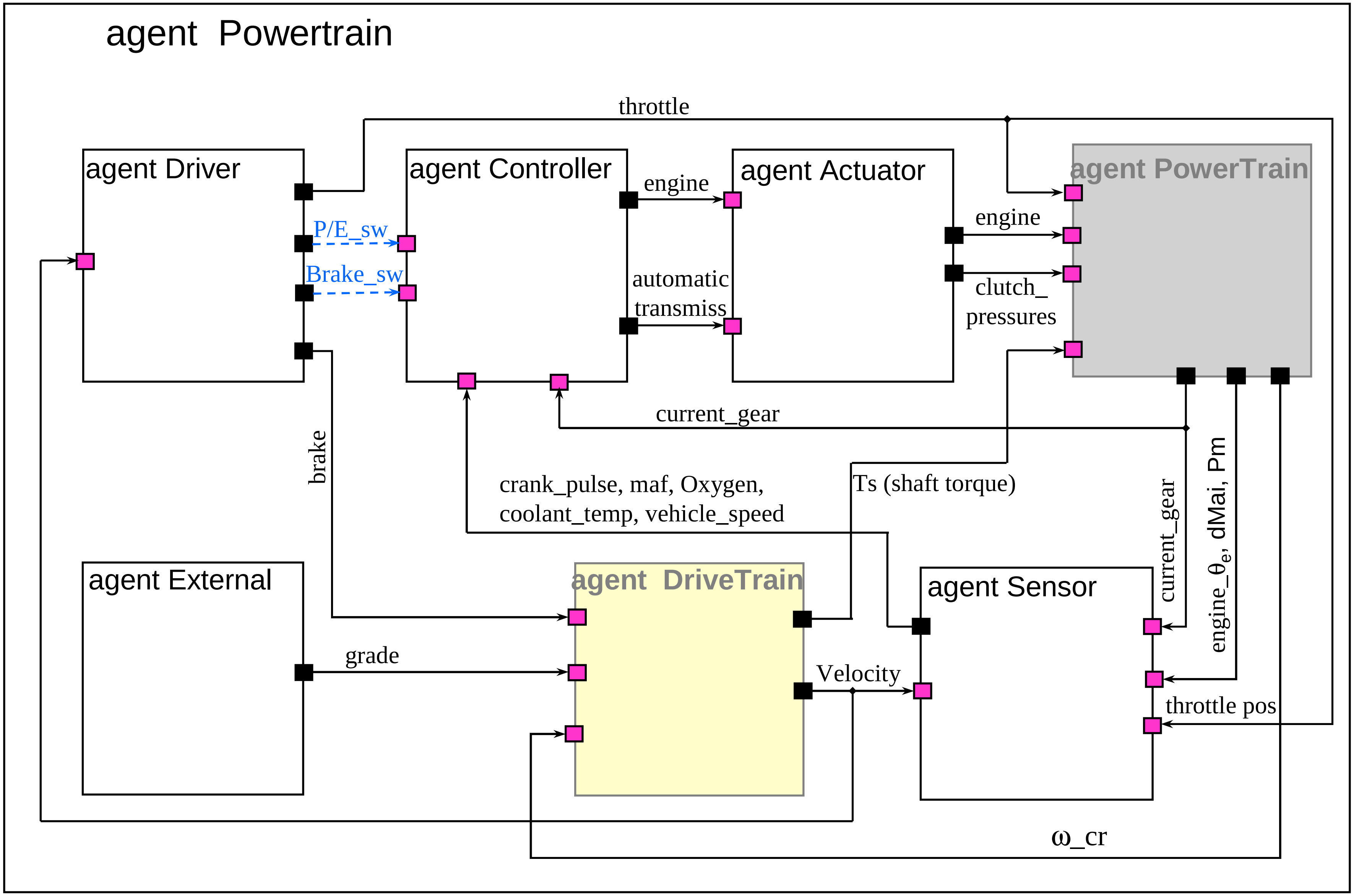
Architectural Hierarchy

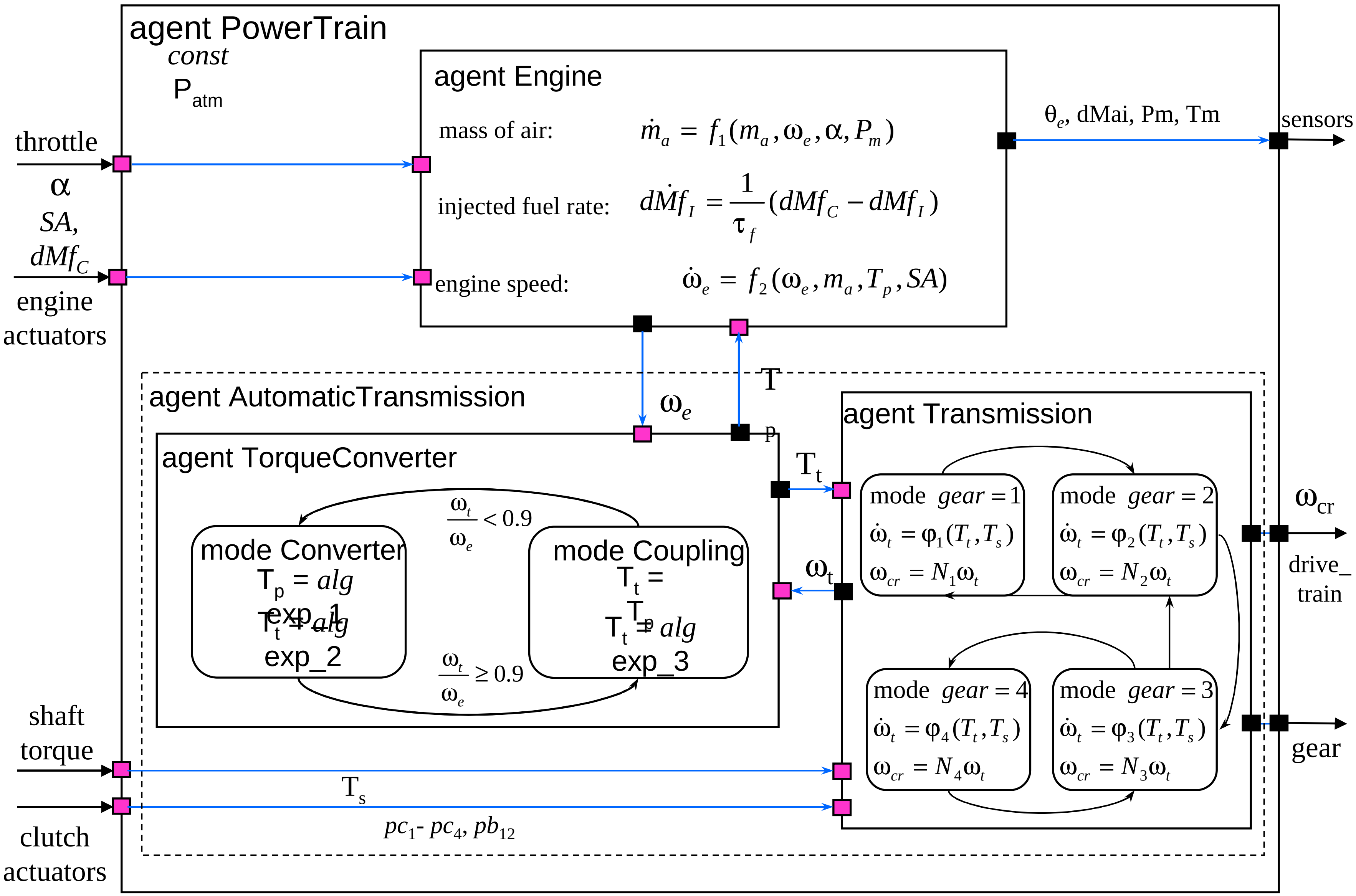


Behavioral Hierarchy



Powertrain modeling



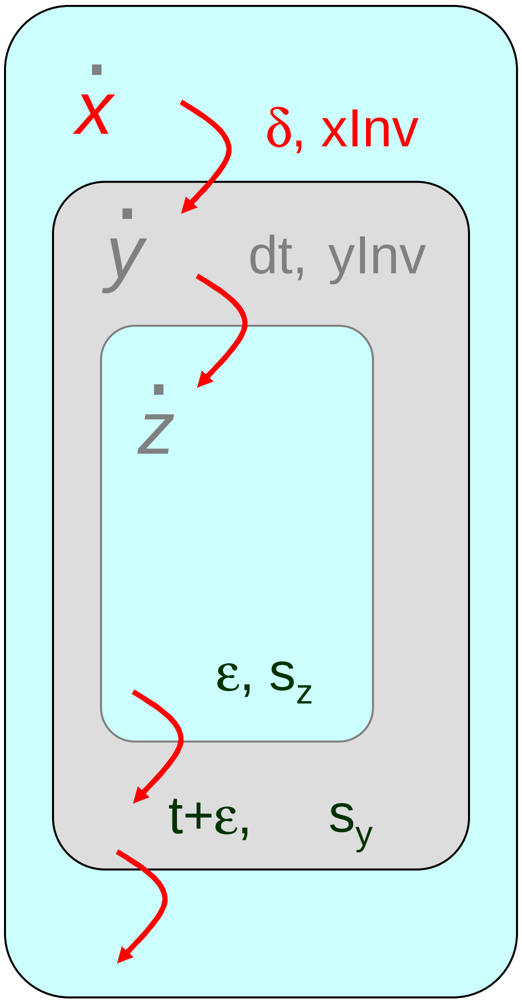


agent PowerTrain
const
P atm
throttle
α
SA,
dMf C
engine
actuators
shaft
torque
clutch
actuators
agent Engine
mass of air:
$\dot{m}_a = f_1(m_a, \omega_e, \alpha, P_m)$
injected fuel rate:
$d\dot{M}f_I = \frac{1}{\tau_f}(dMf_C - dMf_I)$
engine speed:
$\dot{\omega}_e = f_2(\omega_e, m_a, T_p, SA)$
θ_e, dMai, Pm, Tm
sensors
ω_e
T p
agent AutomaticTransmission
agent TorqueConverter
mode Converter
T_p = alg exp_1
T_t = alg exp_2
mode Coupling
T_t = T_p
T_t = alg exp_3
$\frac{\omega_t}{\omega_e} < 0.9$
$\frac{\omega_t}{\omega_e} \geq 0.9$
T_t
ω_t
agent Transmission
mode $gear = 1$
$\dot{\omega}_t = \varphi_1(T_t, T_s)$
$\omega_{cr} = N_1\omega_t$
mode $gear = 2$
$\dot{\omega}_t = \varphi_2(T_t, T_s)$
$\omega_{cr} = N_2\omega_t$
mode $gear = 4$
$\dot{\omega}_t = \varphi_4(T_t, T_s)$
$\omega_{cr} = N_4\omega_t$
mode $gear = 3$
$\dot{\omega}_t = \varphi_3(T_t, T_s)$
$\omega_{cr} = N_3\omega_t$
ω_{cr}
drive_
train
gear
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 δ and invariants from the supermode (or the scheduler).

2. While (time t = 0; t <= δ) do:

   - Simplify all invariants.

   - Predict integration step dt based on δ and the invariants.

   - Execute time round of the active submode and get state s and time elapsed ε.

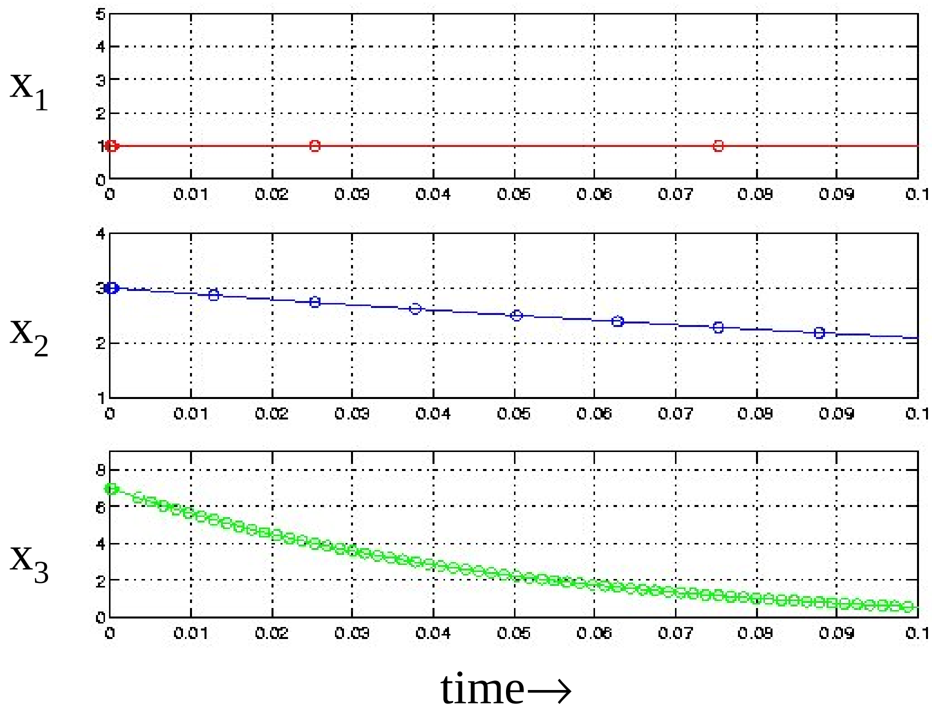
   - Integrate for time ε and get new state s.

   - Return s and t+ε if invariants were violated.

   - Increment t = t+ε.

3. Return s and δ

Modular Multi-rate Simulation



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

ratio of largest to smallest step size

constant

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

step size

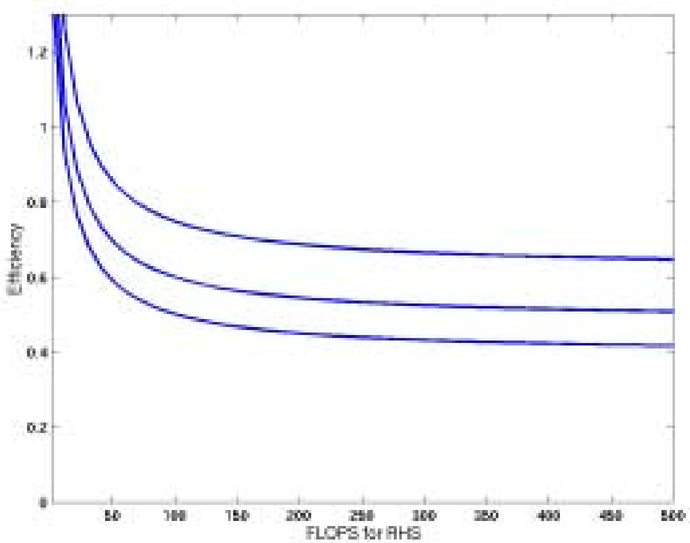
coupling

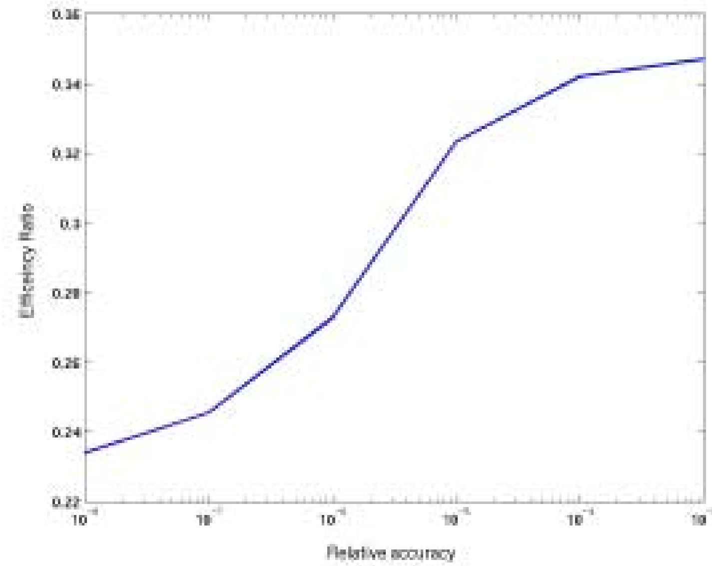
- "Slowest-first" order of integration
- Coupling is accommodated by using interpolants for slow variables
- Tight error bound: **O**(h^{m+1})

Computations for Modular Systems

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

Increasing efficiency





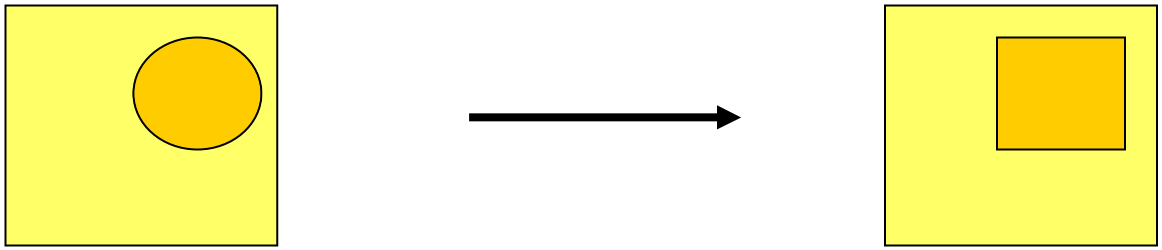
Increasing complexity

Higher tolerances

Talk Outline

- ✓ 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!

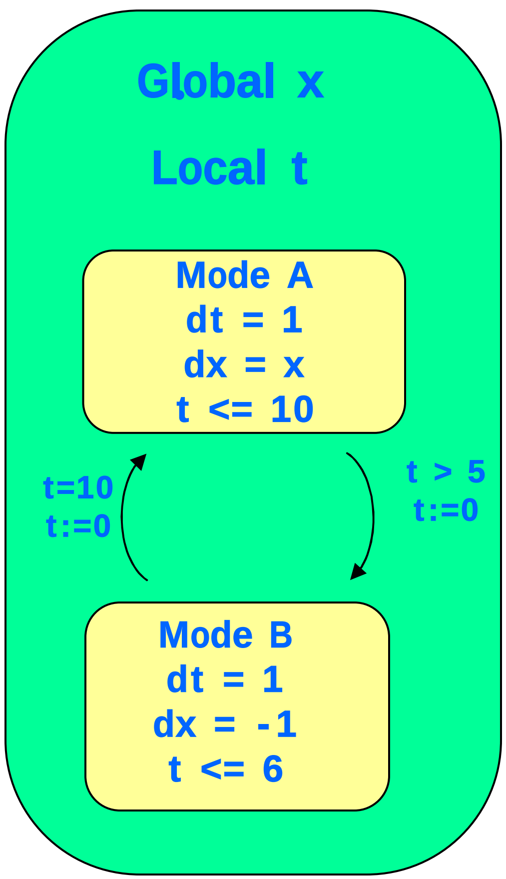
Observational Semantics

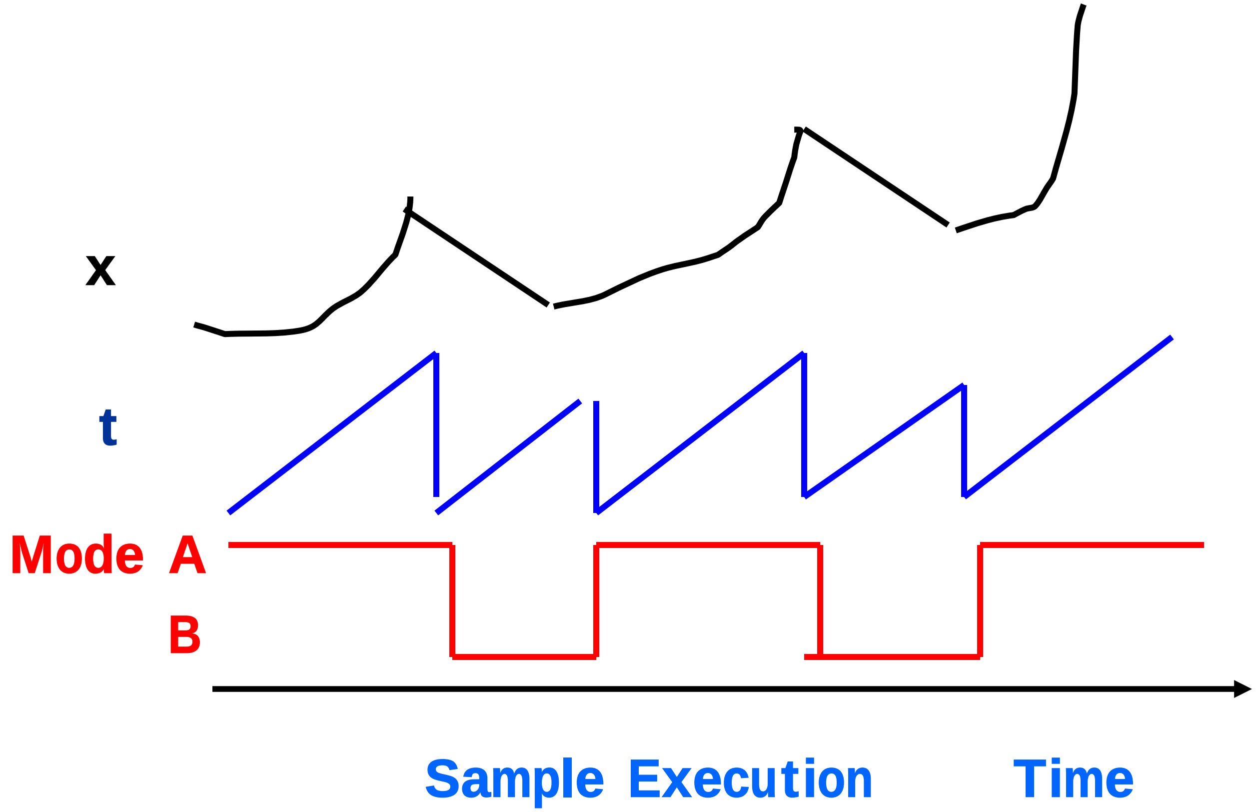
- Classical programming language concept of denotational semantics: two programs are "equivalent" if they compute the same function
- For reactive systems, ongoing interaction (behavior over time) must be accounted for
- Observational semantics of a hybrid component:
  - Signature (static interface): Set of input/output variables
  - Behavioral interface: Set of traces
- Trace: Projection of an execution onto observable parts (e.g. sequence of input/outputs)

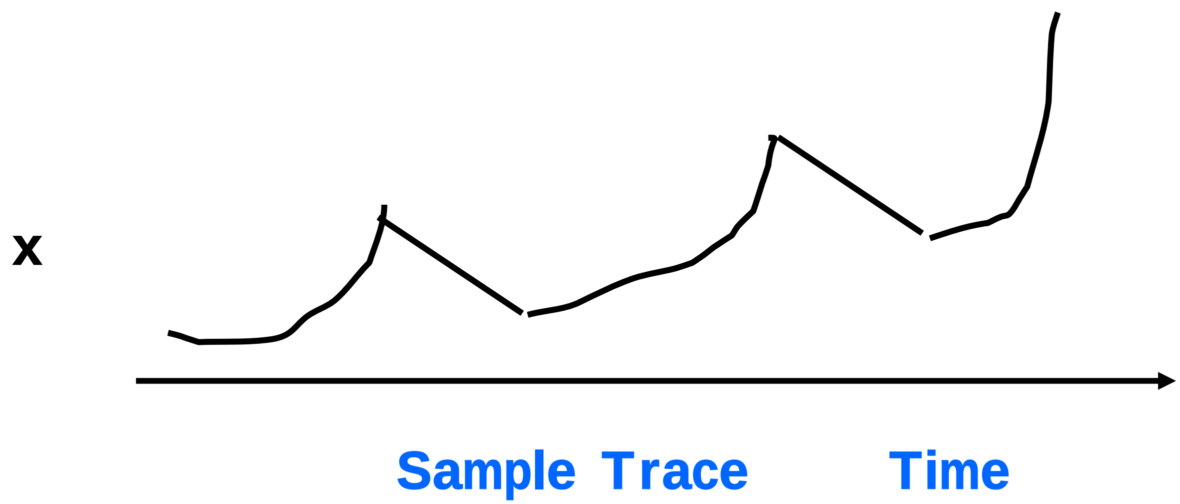
Compositional Semantics

- 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

  e.g. 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
t=10
t:=0
t > 5
t:=0
Mode B
dt = 1
dx = -1
t <= 6


x
t
Mode A
B
Sample Execution
Time


x
Sample Trace
Time

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

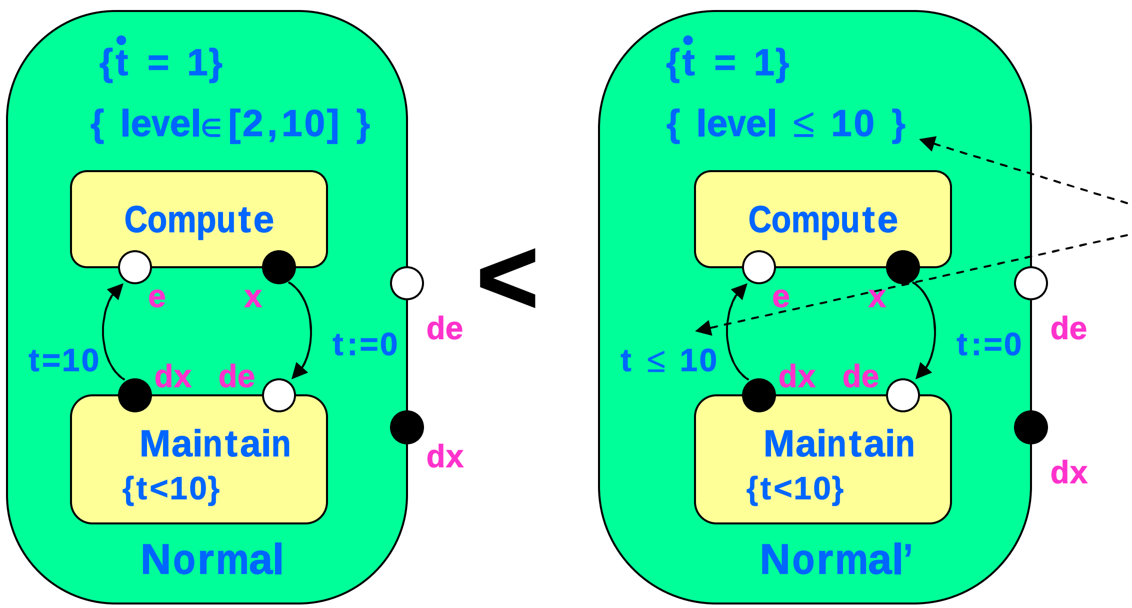
Semantics of Charon modes

- 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

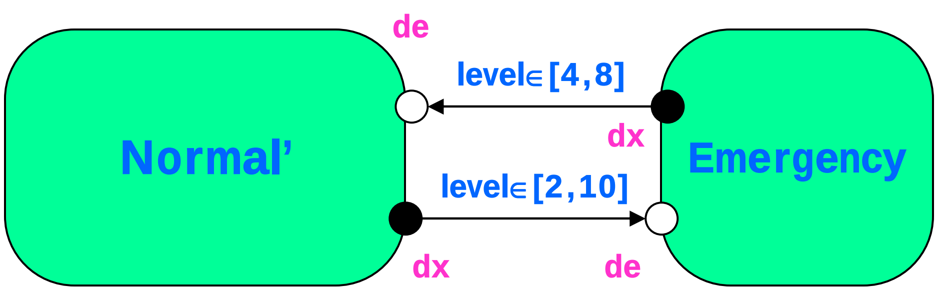
Normal **Normal’**



- **Same control points and global variables**
- **Guards and constraints are relaxed**

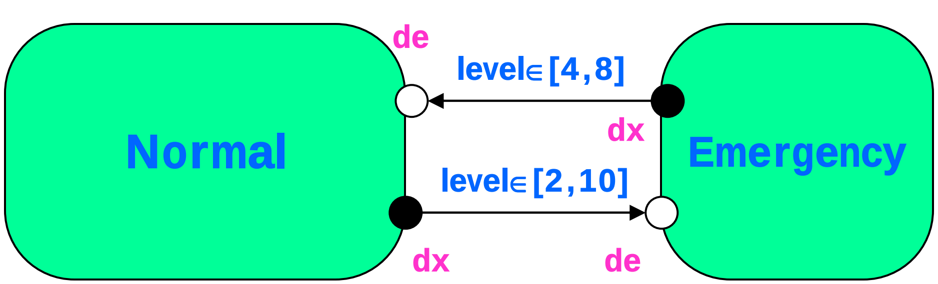
Sub-mode refinement

Controller’

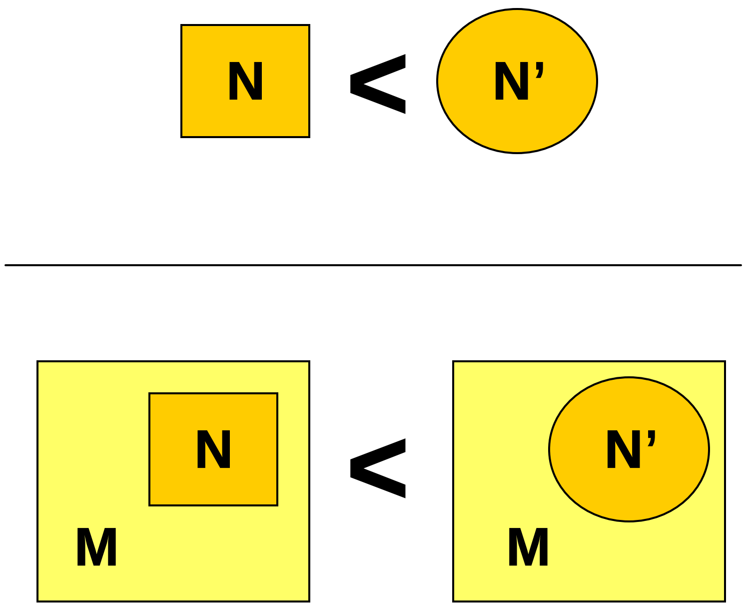


Refines

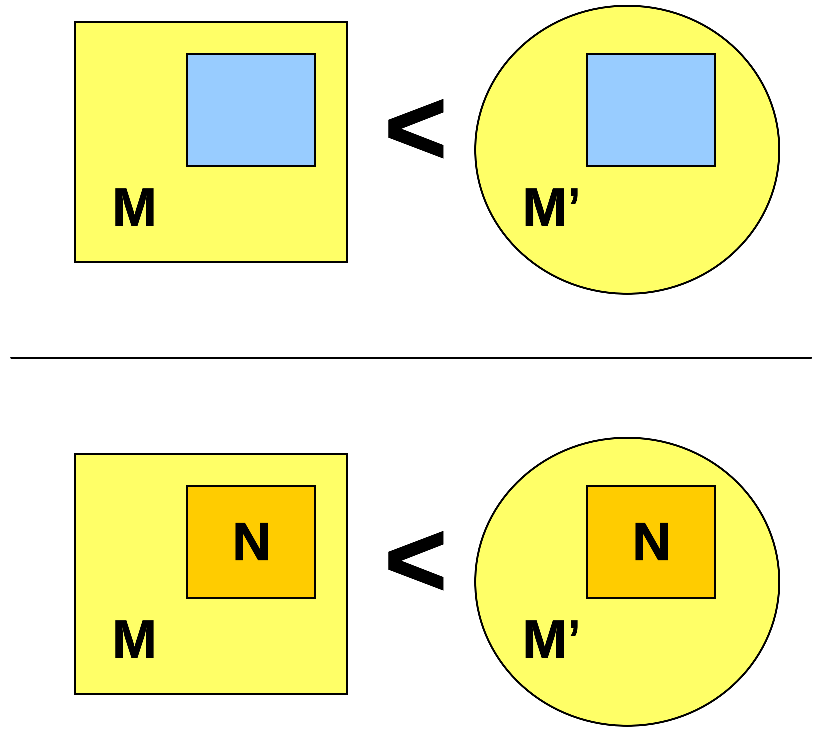
Controller



Compositional Reasoning



Sub-mode refinement



Context refinement