Model-based Code Generation – CHARON Case Study

Jesung Kim

in collaboration with

Rajeev Alur, Yerang Hur, Franjo Ivancic,
Insup Lee, and Oleg Sokolsky

University of Pennsylvania

Contents

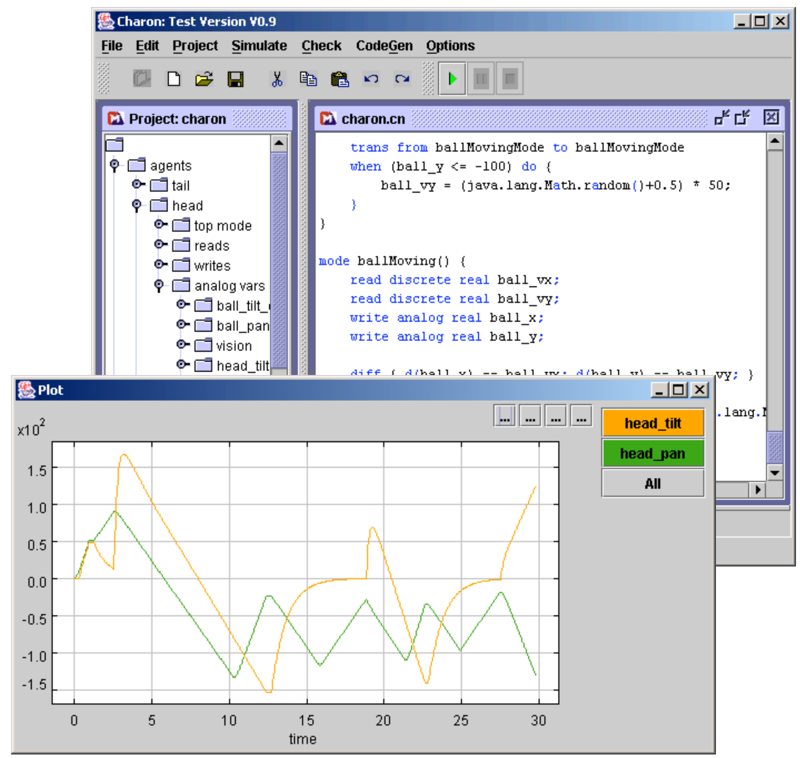
Motivation

- Formal specification of hybrid systems
  - Subject to formal verification
- Automatic generation of the code
  - Elimination of coding errors
- Formulation of the differences between the model and the generated code
  - Bounded difference desired
- Case study in a robotic platform (AIBO)
  - Code generation for fairly complicated systems

Penn
University of Pennsylvania

CHARON Framework

- Language for formal specification of hybrid systems
  - Analog variable
  - Differential/algebraic equation
  - Discrete state transition
- Hierarchical specification
  - Architectural hierarchy
    - agent: communicating entity
  - Behavioral hierarchy
    - mode: hierarchical state machine with continuous dynamics
- Simulation
- Model checking
- <u>Code generation</u>
- Run-time verification



Penn
University of Pennsylvania

Example: Four Legged Robot

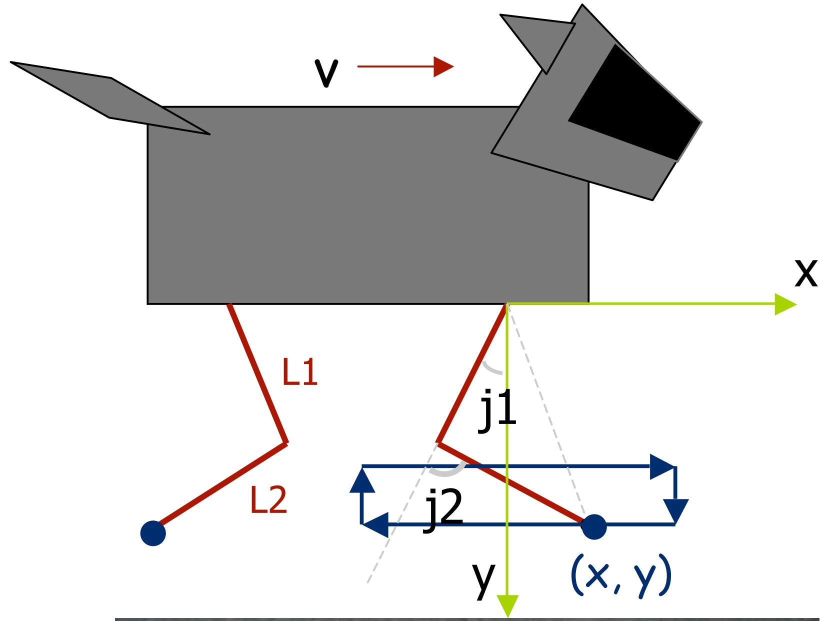
- Control objective
  - $v = c$
- High-level control laws

$$\dot{x} = -v, \quad x \geq -\text{stride}/2 \;\longrightarrow\; \dot{y} = -kv \;\longrightarrow\; \dot{x} = kv, \quad x \leq \text{stride}/2 \;\longrightarrow\; \dot{y} = kv \;\longrightarrow\; \dot{x} = -v, \quad x \geq -\text{stride}/2$$

- Low-level control laws

$$j_1 = \arctan(x/y) - \arccos\left(\frac{x^2 + y^2 + L_1^2 - L_2^2}{2L_1\sqrt{x^2 + y^2}}\right)$$

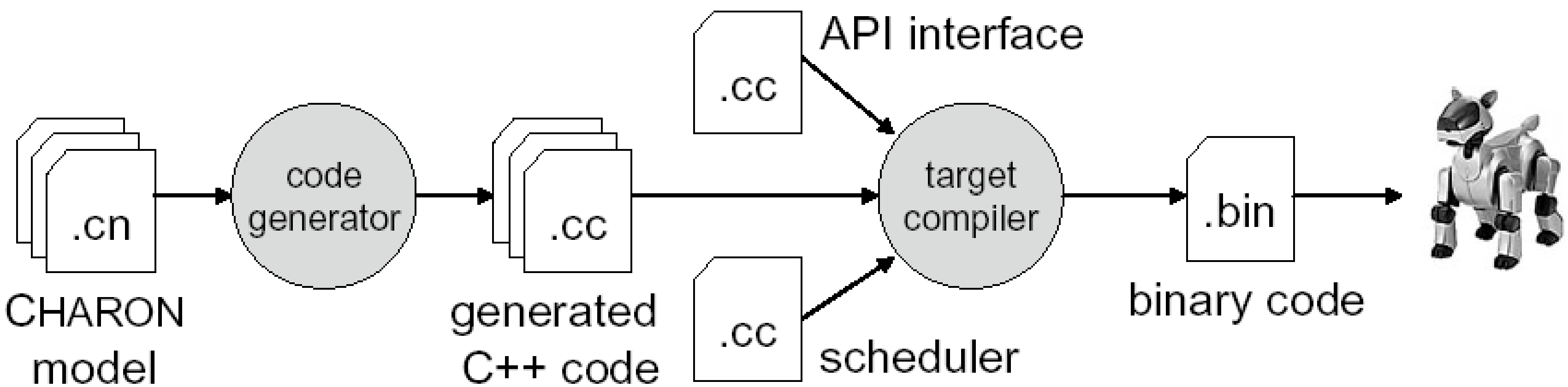
$$j_2 = \arccos\left(\frac{x^2 + y^2 + L_1^2 - L_2^2}{2L_1L_2}\right)$$



*[LCTES 2003] R. Alur, F. Ivancic, J. Kim, I. Lee, and O. Sokolsky. Generating embedded software from hierarchical hybrid models.

CHARON Code Generator

- CHARON code generator translates CHARON models into C++ code
  - Each object of CHARON models is translated into a C++ structure
- Generated C++ code is compiled by the target compiler along with additional code
  - Run-time scheduler: invokes active components periodically
  - API interface routines: associates variables with devices



Translation of CHARON models

- Analog variable
  - C++ class: read and write to variables can be mapped to system API
- Differential equation
  - Euler's method: x' == 10 → x += 10 * h   (h: step size)
  - Runge-Kutta method
- Algebraic equation
  - Assignment statement executed in a data dependency order
    - x == 2y; y == 2z → y = 2z; x = 2y;
- Invariant
  - Assertion statement for runtime verification
  - Instrumented for safer check
- Discrete transition
  - If-then statement
    - urgent transition policy
  - "Instrumented" if-then statement
    - *not-so-urgent* transition policy
- Mode
  - C++ class: collection of variables, equations, transitions, and reference to submodes
- Agent
  - C++ class: interleave execution of each step of subagents

Soundness of Generated Code

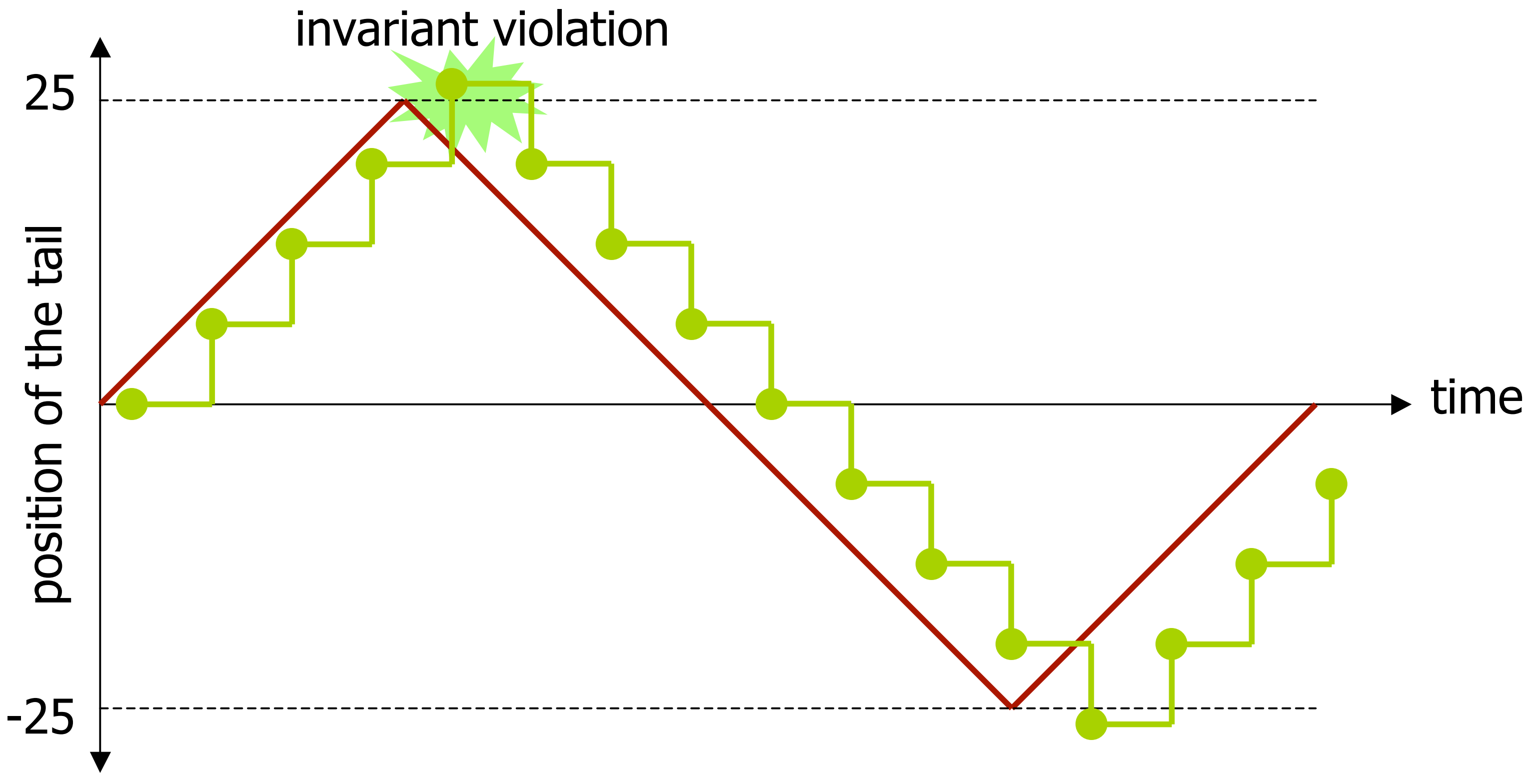
- Code may behave differently from the model:
  - Numerical errors (numerical integration)
  - Floating-point errors (fixed precision arithmetic)
  - Switching errors
    - Missed switching: enabled transition is missed due to discrete testing of switching conditions [LCTES 2003]
    - Invalid switching: disabled transition is evaluated as enabled due to different update frequencies of shared variables [HSCC 2004]

  Properties held in the model may not be guaranteed in the code even if automatically generated!

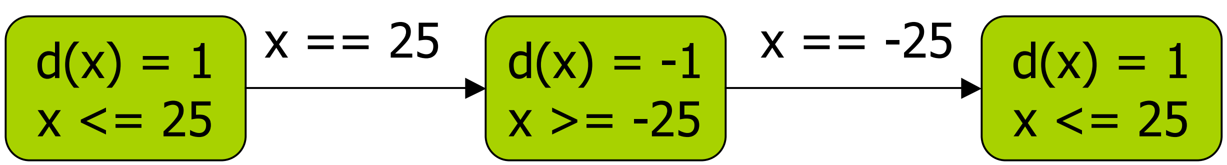
- How to check / prevent switching errors?
  - Exploit non-determinism of the model
    - Check the model if it gives sufficient room for the code to take a switch
  - Design the switching policy
    - Take switching conservatively to prevent an invalid switch

[LCTES 2003] R. Alur, F. Ivancic, J. Kim, I. Lee, and O. Sokolsky. Generating embedded software from hierarchical hybrid models.
[HSCC 2004] Y. Hur, J. Kim, J. Choi, and I. Lee. Sound code generation from communicating hybrid models.

Switch Miss



invariant violation
25
-25
position of the tail
time



d(x) = 1
x <= 25
x == 25
d(x) = -1
x >= -25
x == -25
d(x) = 1
x <= 25

Analysis of Switch Miss

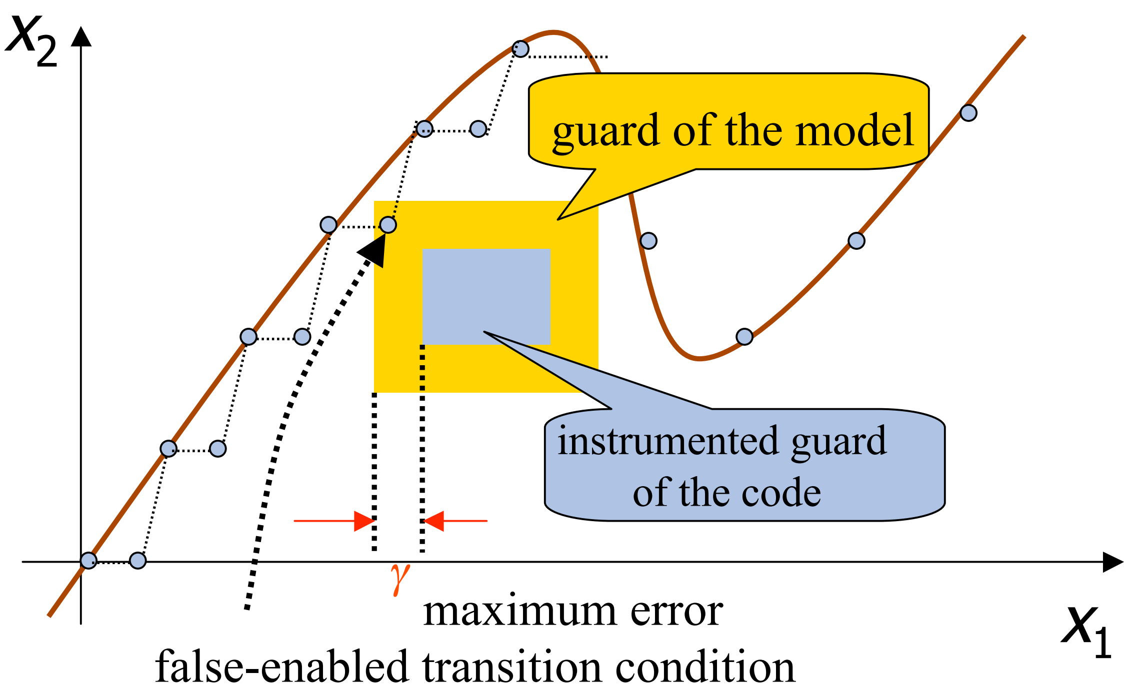
- Given a step size Δ and a CHARON model, check whether for all pairs of invariants I and guards G of switch (= Δ-lookahead agent)

$$Post_{\Phi}(I \setminus G, \Delta) \subseteq I$$

- If so, the generated code will produce a valid execution trace
  - The code performs a switch immediately when the guard is true
- Otherwise, even for the generated code we cannot guarantee a valid execution trace.

Invalid Switch

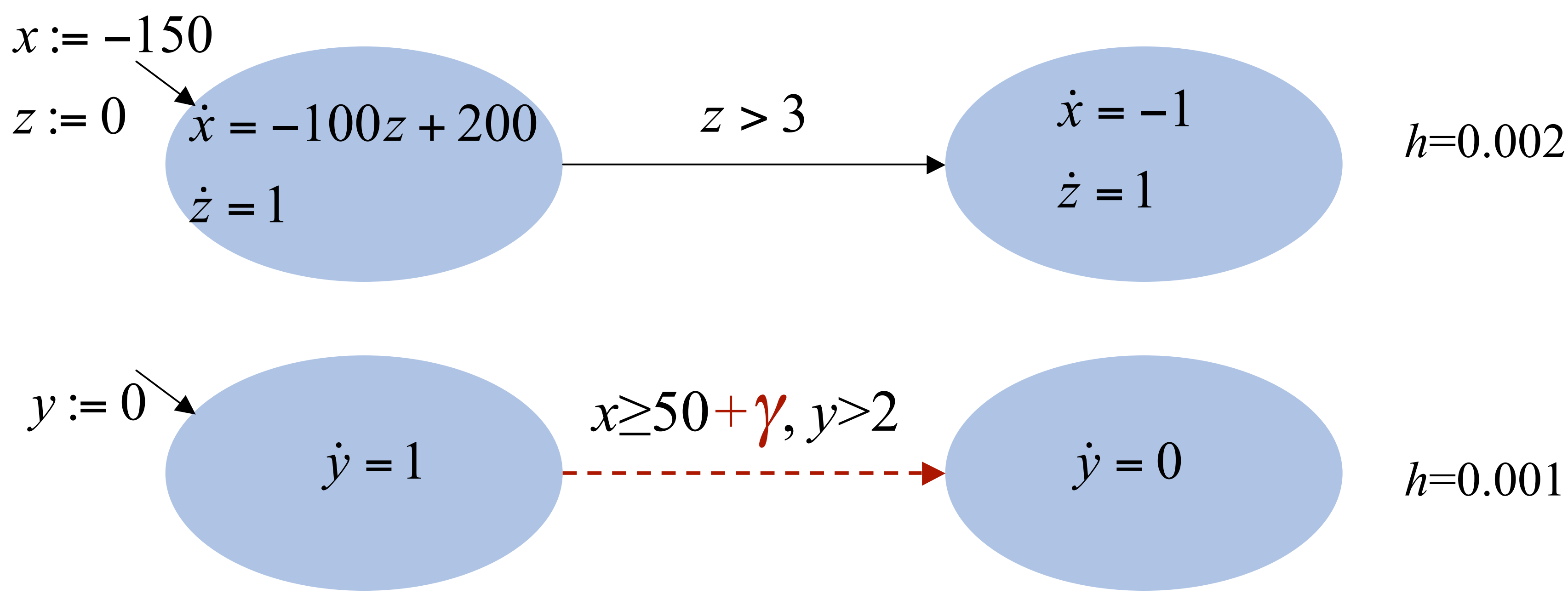
- Variables x_1, x_2, … are updated by different steps Δ_1, Δ_2, …
- Code evaluates switching conditions f(x_1, x_2, …) by referencing $x_1(t_1)$, $x_2(t_2)$, …
  - ($x_1(t_1)$, $x_2(t_2)$, …) may not on the trajectory of (x_1, x_2, …) unless $t_1 == t_2 == …$

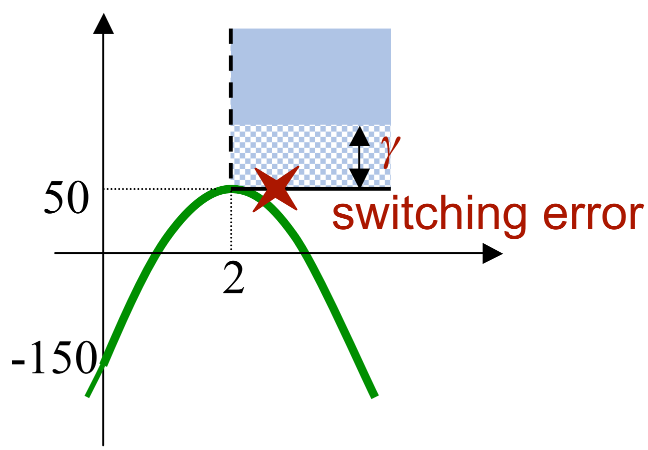


Preventing Invalid Switch through Guard Instrumentation

- Exploit non-determinism in the guard conditions
  - Transition can (but need not immediately) be taken when the guard is enabled
- Compute a maximum error due to asynchrony
  - max(|x'|*Δ)
    - assumption: independent dynamics and rectangular guard sets
- "Tighten" the guards such that transitions are not falsely enabled at the presence of the maximum error
  - Trace of discrete states is equivalent to that of the model

Example

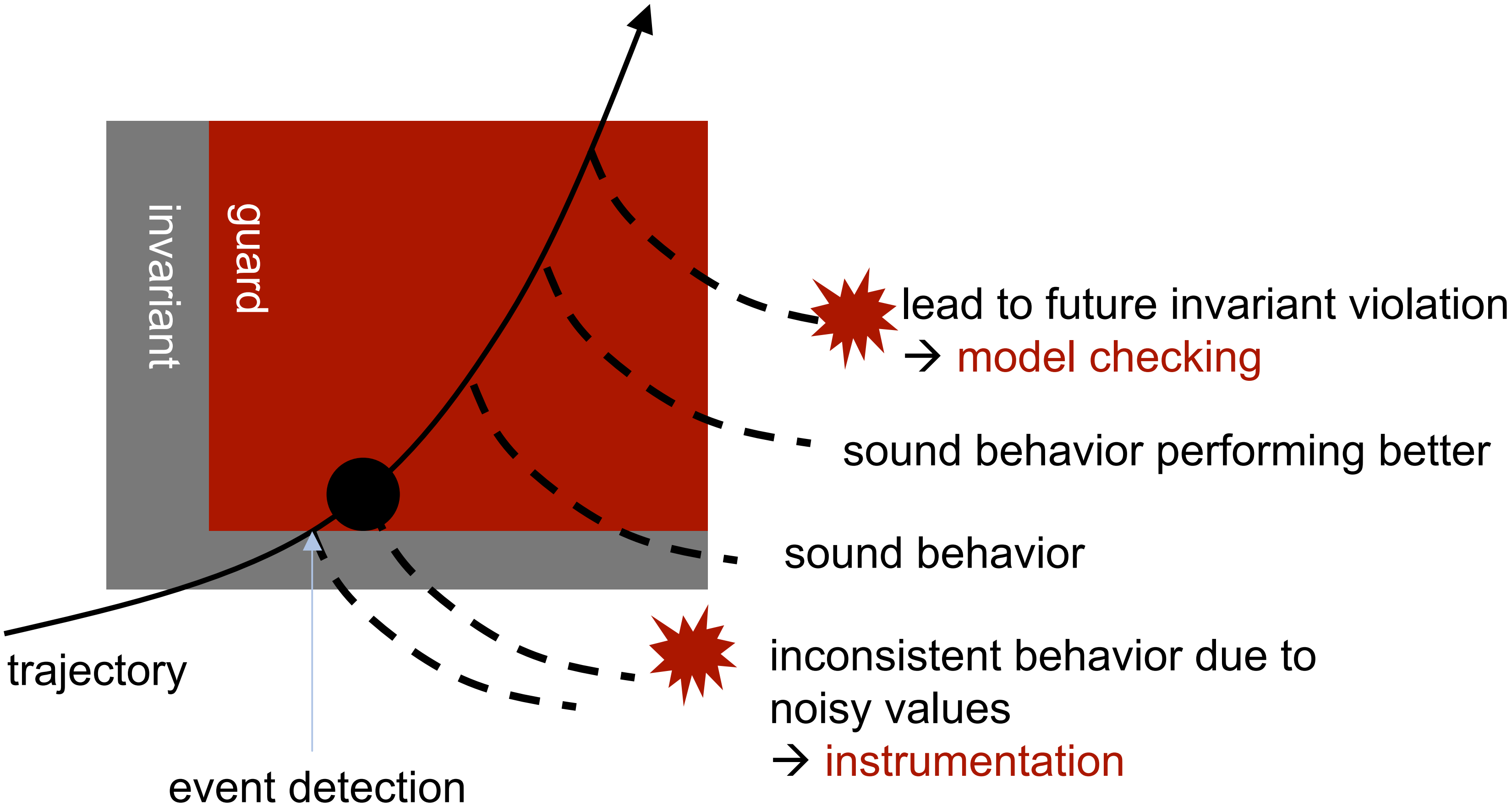




$$\begin{aligned} \gamma &= (\dot{x}(t) \bullet \max(h))_{\max} \\ &= (-100t + 200) \bullet 0.002_{\max} \\ &= 0.4 \end{aligned}$$

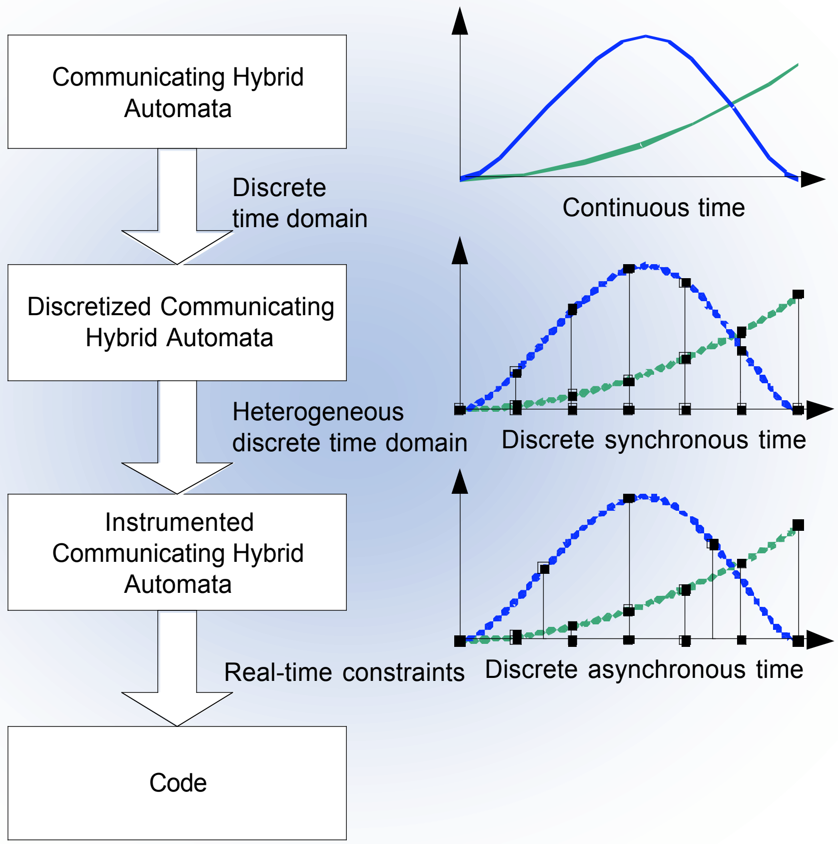
*[HSCC 2004] Y. Hur, J. Kim, J. Choi, and I. Lee.
Sound code generation from Communicating Hybrid models.

Transition Policy



Design Flow of our Framework

- Communicating Hybrid Automata
  - Continuous time domain
- Discretized Communicating Hybrid Automata
  - Single discrete time domain
- Instrumented Communicating Hybrid Automata
  - Each automaton having its own discrete time domain
  - Guard "instrumented" to prevent errors due to different time domains
- Code
  - Machine executable



Summary

- CHARON code generator automates translation of complicated hybrid systems specification into modular C++ code
- Each C++ module can be mapped to a periodic task of RTOS of the target system to approximate continuous update
- Even automatically generated code is *not* semantically equivalent to the original model:
  - Numerical errors, floating-point errors, switching errors…
- Switching errors due to different update rates of shared variables can be prevented through instrumentation of the switching conditions