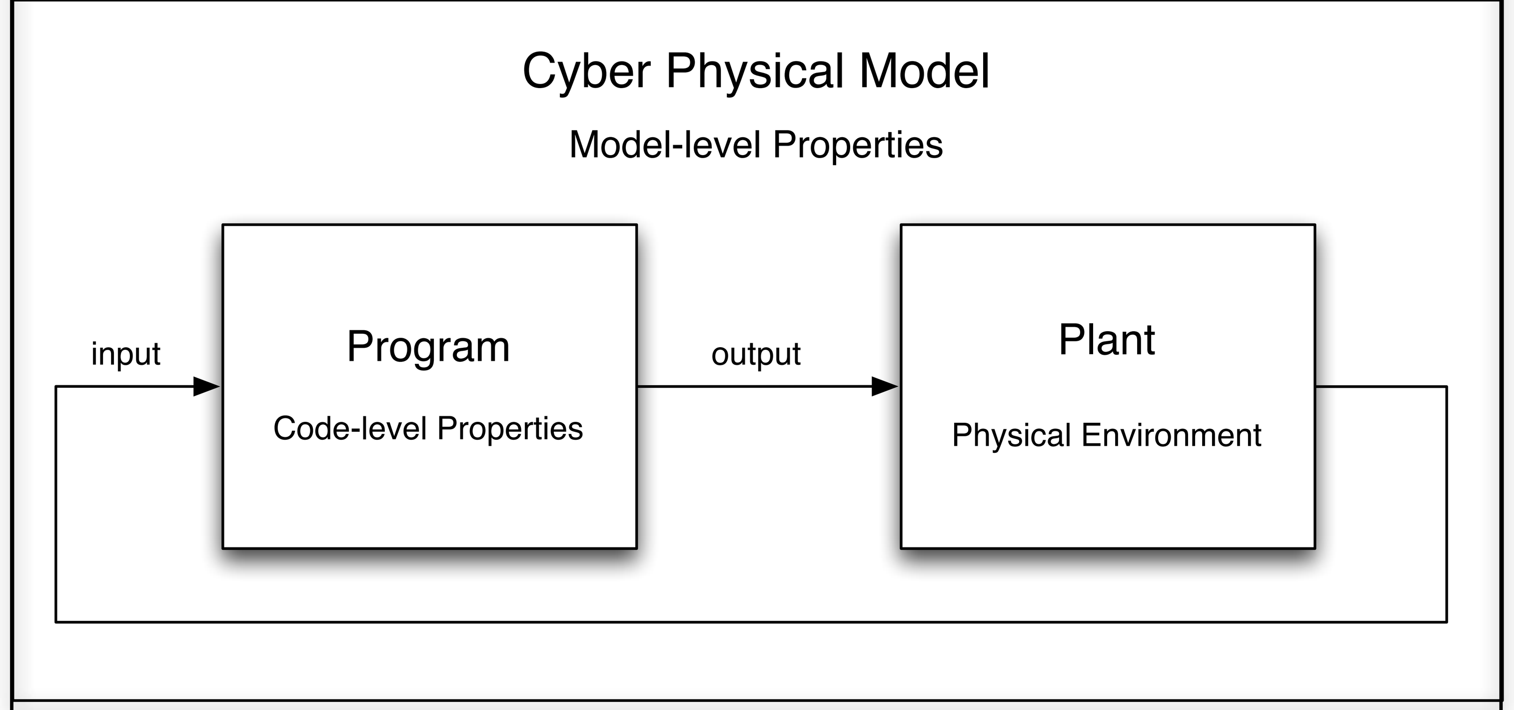
Verifying Model-level Properties by Abstracting C Programs

Carnegie Mellon
Computer Science Department

Soonho Kong Sicun Gao Edmund Clarke

CMACS
NSF Expeditions in Computing

Introduction



- Control software implements hybrid-system models
- Two levels of properties:
  - Code-level : Buffer-overflow, Divide-by-zero
  - Model-level : Functional-safety properties
- **Problem** of verifying model-level properties
  - Model-level properties are not usually specified in terms of program variables.
  - Verification needs to take physical environment into account.

Our Approach

- Reconstruct the high-level model $\mathcal{M}$ from a low-level program $\mathcal{P}$.
- The reconstructed model $\mathcal{M}$ should "$\varepsilon-$ bisimulate" the given program $\mathcal{P}$.
- We can then perform bounded model checking on the reconstructed model with respect to the model-level specification.

$\varepsilon-$ bisimulation Relation

Definition $\mathcal{P} \sim_\epsilon \mathcal{M}$

A program $\mathcal{P}$ ε–bisimulates a model $\mathcal{M}$ with an error bound ϵ if and only if

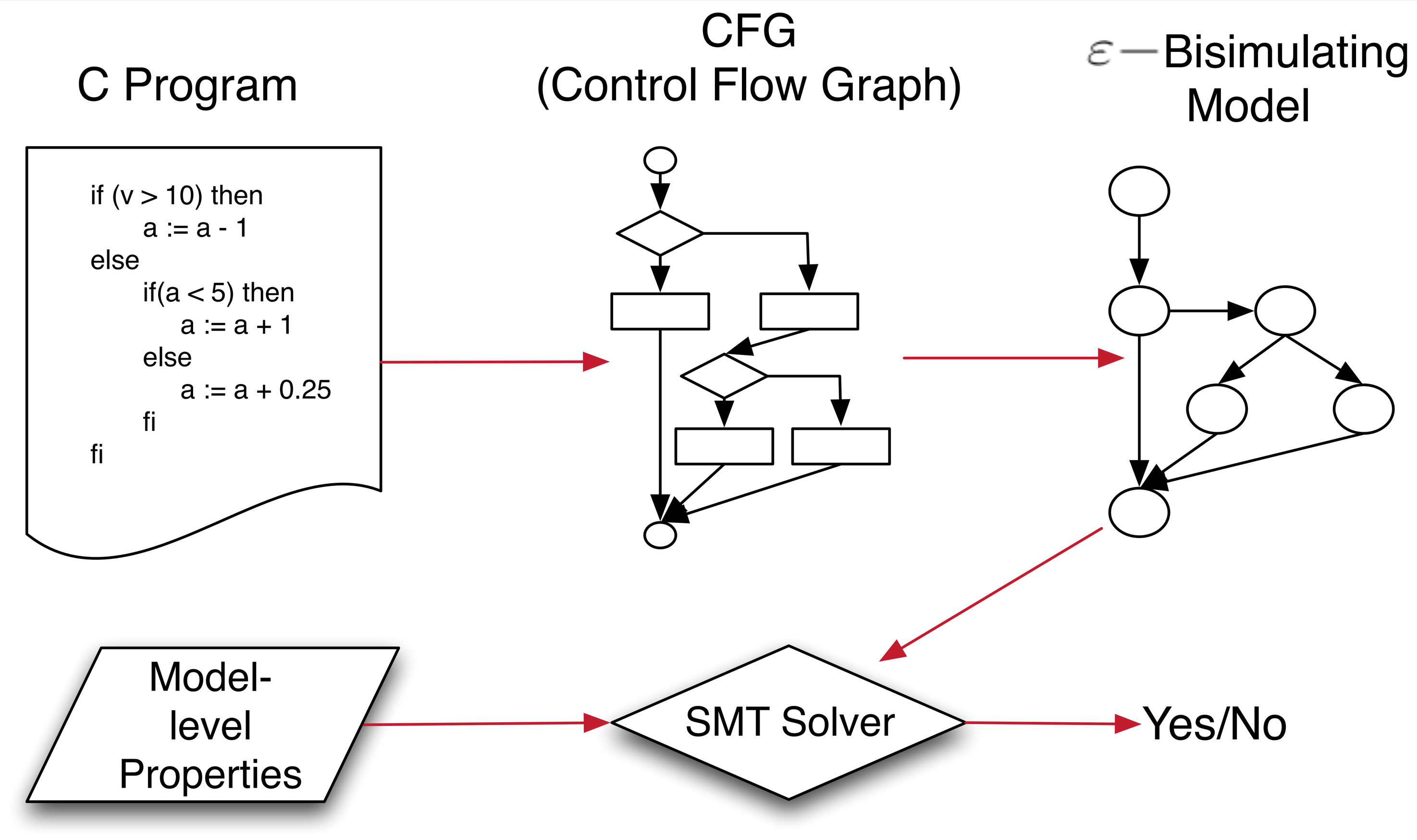
1) Whenever we have a trace from $\vec{x}_1$ to $\vec{x}_2$ in the program, we have a corresponding trajectory from $\vec{x}_1^\epsilon$ to $\vec{x}_2^\epsilon$ in the model where $\vec{x}_1$ and $\vec{x}_2$ are values of program variables.

$$\forall \vec{x}_1, \vec{x}_2 . \mathrm{Trace}_\mathcal{P}(\vec{x}_1, \vec{x}_2) \implies \mathrm{Traj}_\mathcal{M}(\vec{x}_1^\epsilon, \vec{x}_2^\epsilon)$$

2) Whenever we have a trajectory from $\vec{x}_1$ to $\vec{x}_2$ in the model, we have a corresponding trace from $\vec{x}_1^\epsilon$ to $\vec{x}_2^\epsilon$ in the program.

$$\forall \vec{x}_1, \vec{x}_2 . \mathrm{Traj}_\mathcal{M}(\vec{x}_1, \vec{x}_2) \implies \mathrm{Trace}_\mathcal{P}(\vec{x}_1^\epsilon, \vec{x}_2^\epsilon)$$

Framework

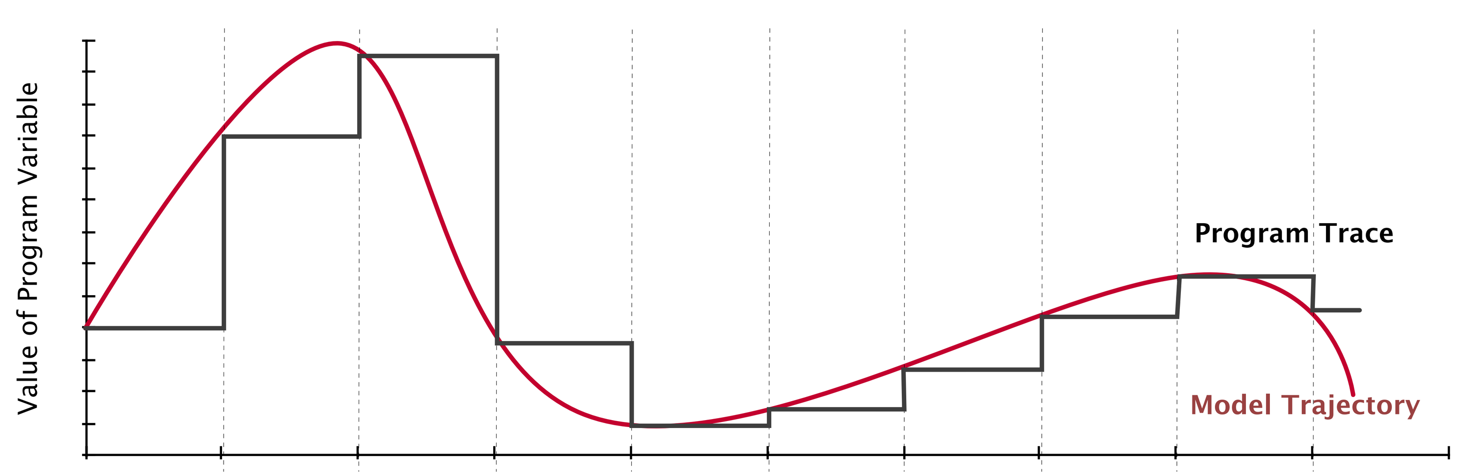


CFG Transformation

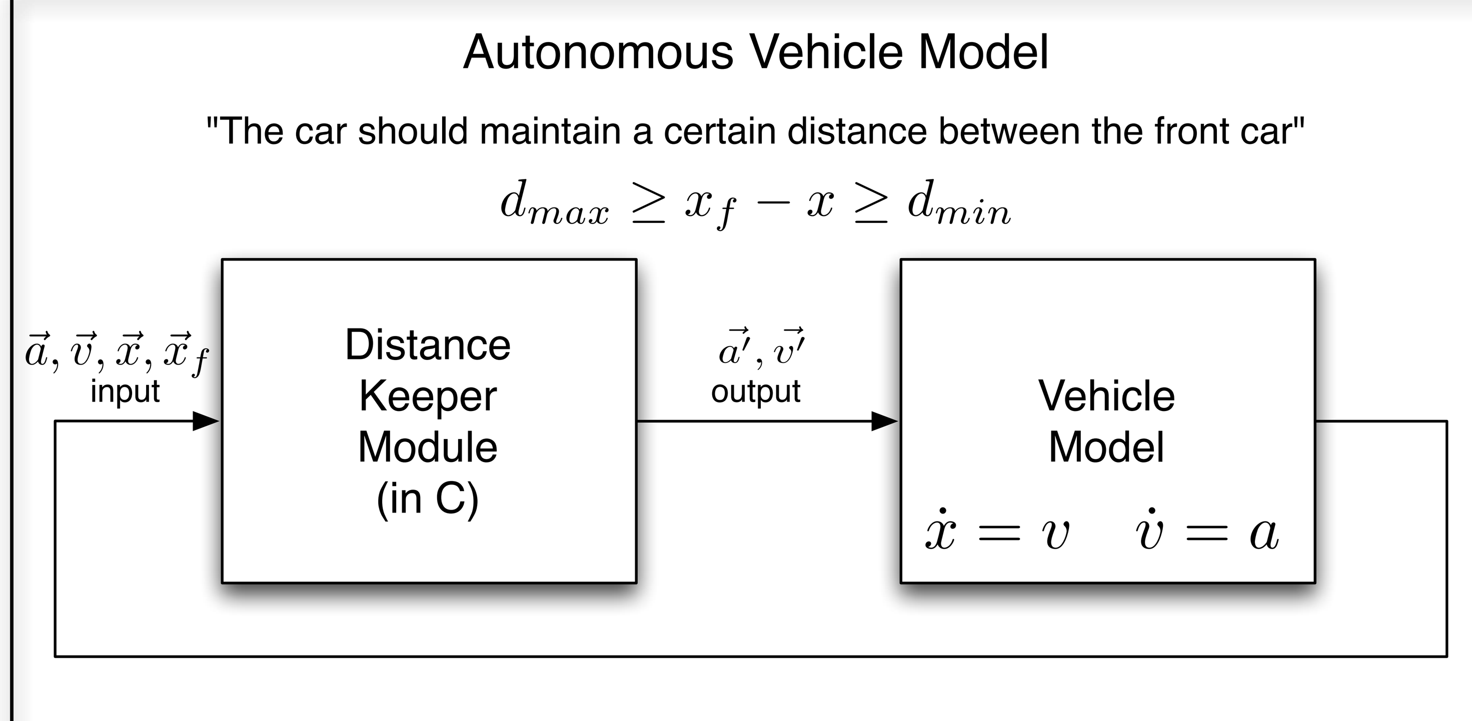
- Input : C program without
  - Pointer Arithmetic, Dynamic Allocations, Function pointers
- Output : Control Flow Graph
- Use CIL(C Intermediate Language) Framework
- Optimization: Program Slicing
  - Given the set of program variables of interest, compute the program slice which may affect the variables of interests.

ε—bisimulation Model Construction

- Input : (optimized) CFG $\Rightarrow$ Output : Hybrid System Model
- Algorithm
  - For each basic block and if-statement in CFG, we create a control location.
  - For each branching edge in CFG, we connect corresponding control locations with the branching condition. For the other edges, we connect corresponding control locations with condition "True". (JUMP part of Hybrid system).
  - Translate each basic block statements into corresponding differential equations. (FLOW part of Hybrid system).
  - Mark corresponding entry block as an initial control location.



Case Study



- TARTAN RACING Project : CMU Robotics Institute + General Motors. The project won 2007 DARPA Urban Challenges ($2M)
- Distance Keeper Module : Core module of the vehicle "BOSS". It maintains the distance between the vehicle and the front one.
  - About 700 LOC C++ code (but very C-like), as starting point. Full Software has 440K LOC.
  - Main method "notify" is called periodically. It takes current information (position, velocity, and acceleration) about the vehicle and the front vehicle. It returns desired acceleration and velocity value to maintain the distance.
  - No dynamic allocation, pointer arithmetic.

Current Progress

- We translated the C++ code into equivalent C code.
- We're implementing CFG Transformer and ε-bisimulation model constructor.