Constraint-based Approach for Analysis of Hybrid Systems

Sumit Gulwani (MSR)
Ashish Tiwari (SRI)

CAV 2008
July 11, 2008

Analysis of Hybrid Systems

Hybrid systems	=	continuous dynamics	+	finite automata
	=	control theory	+	computer science

Analysis techniques combine approaches from the two fields:

- Extension of Lyapunov analysis – for proving stability
- Forward symbolic reachability
- Abstraction and model checking
- CEGAR
- Invariant generation

Inductive Invariant for Safety

The main inference rule for deductive verification:

$$\begin{array}{lrcl} \texttt{Init}: & \forall \vec{x} : Init(\vec{x}) & \Rightarrow & Inv(\vec{x}) \\ \texttt{Ind}: & \forall \vec{x}, \vec{x'} : Inv(\vec{x}) \wedge t(\vec{x}, \vec{x'}) & \Rightarrow & Inv(\vec{x'}) \\ \texttt{Safe}: & \forall \vec{x} : Inv(\vec{x}) & \Rightarrow & Safe(\vec{x}) \\ \hline & & G(Safe(\vec{x})) & \end{array}$$

How to modify this rule to handle continuous dynamics?

How to generate Inv?

How to handle continuous dynamics?

Inductiveness for continuous dynamics:

If the system is in Inv *at time* t*, then it stays in* Inv *at* $t + \epsilon$ *as per the dynamics* $\dot{\vec{x}} = f(\vec{x})$.

Continuity is on our side

If we are in the interior, then there is no fear of going out.

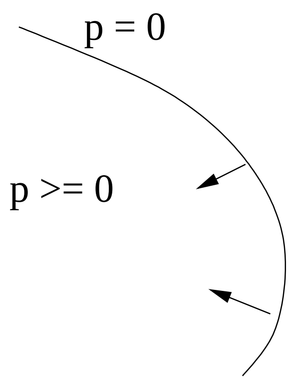
Only need to worry about when we are on the boundary.

If $Inv := (p \geq 0)$

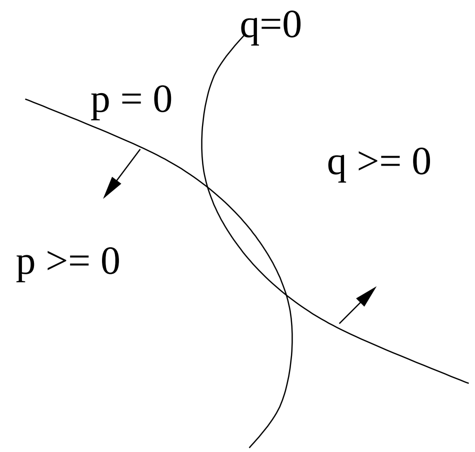
$$\texttt{Ind}_c : \quad \forall \vec{x} : p = 0 \quad \Rightarrow \quad \frac{dp}{dt} \geq 0$$

where $\frac{dp}{dt} := \sum_k (\frac{\partial p}{\partial x_k} \frac{dx_k}{dt})$

Illustrations



$p \geq 0$



$p \geq 0 \ \vee \ q \geq 0$

Inductive Rule for Continuous Dynamics

If $Inv := (p_1 \geq 0 \ \vee \ p_2 \geq 0)$

$$
\begin{array}{ll}
\texttt{Ind}_c : & \forall \vec{x} : p_1 < 0 \ \wedge \ p_2 = 0 \quad \Rightarrow \quad \frac{dp_2}{dt} \geq 0 \\
& \forall \vec{x} : p_1 = 0 \ \wedge \ p_2 < 0 \quad \Rightarrow \quad \frac{dp_1}{dt} \geq 0 \\
& \forall \vec{x} : p_1 = 0 \ \wedge \ p_2 = 0 \quad \Rightarrow \quad \frac{dp_1}{dt} \geq 0 \ \vee \ \frac{dp_2}{dt} \geq 0
\end{array}
$$

If $Inv := \bigwedge_i \bigvee_j (p_{ij} \geq 0)$

$$
\texttt{Ind}_c : \quad \forall \vec{x} : Inv \ \wedge \ \bigwedge_{j' \in J'} p_{ij'} = 0 \ \wedge \ \bigwedge_{j' \notin J'} p_{ij'} < 0 \quad \Rightarrow \quad \bigvee_{j' \in J'} \frac{dp_{ij'}}{dt} \geq 0
$$

for all i and non-empty $J' \subseteq J$

Deductive Verification of Hybrid Systems

Hybrid System is a collection of Q continuous dynamical systems:

Let $Inv := \langle Inv_q \rangle_q \in Q$, where $Inv_q := \bigwedge_i \bigvee_j p_{ij} \geq 0$

$$
\begin{array}{lrcl}
\texttt{Init}_q: & Init_q(\vec{x}) & \Rightarrow & Inv_q(\vec{x}) \\
\texttt{Ind}_{q,q'}: & Inv_q(\vec{x}) \wedge t(\vec{x}, \vec{x'}) & \Rightarrow & Inv_{q'}(\vec{x'}) \\
\texttt{Ind}_q: & Inv_q(\vec{x}) \ \wedge \ \bigwedge_{j' \in J'} p_{ij'} = 0 \ \wedge \ \bigwedge_{j' \notin J'} p_{ij'} < 0 & \Rightarrow & \bigvee_{j' \in J'} (\frac{dp_{ij'}}{dt})_q \geq 0 \forall i, \emptyset \\
\texttt{Safe}_q: & Inv_q(\vec{x}) & \Rightarrow & Safe_q(\vec{x}) \\
\hline
 & & G(Safe(\vec{x})) &
\end{array}
$$

Technical detail: Incorporate state invariants in the antecedents

Background theory is the theory of reals

How to Generate Inv?

Applying the above inference rule

$=$ proving $\exists Inv : \forall \vec{x} : \phi(Inv, \vec{x})$

- Guess a template $\mathcal{I}(\vec{u}, \vec{x})$ for Inv

  $\vec{u}$: template variables, $\vec{x}$: state variables

  Assuming Inv is $\mathcal{I}(\vec{c})$

- Now we need to prove

$$\exists \vec{u} : \forall \vec{x} : \phi(\vec{u}, \vec{x})$$

Bounded Falsification (BMC) vs. Bounded Verification

Solving $\exists\forall$

Restrict to polynomial systems

$\Rightarrow$

ϕ contains only polynomial expressions

$\Rightarrow$

Validity of $\exists\vec{u} : \forall\vec{x} : \phi$ is decidable

More practically, use heuristics to decide $\exists\vec{u} : \forall\vec{x} : \phi$

1. Eliminate $\forall$: $\exists\vec{u} : \forall\vec{x} : \phi \mapsto \exists\vec{u} : \exists\vec{\lambda} : \phi'$
2. Search for $\vec{u}$ and $\vec{\lambda}$ over a finite domain using SMT (bit vector) solver

Step 1: $\exists\forall$ to $\exists$

For linear arithmetic, Farkas' Lemma eliminates $\forall$

$\forall\vec{x} : p_1 \geq 0 \wedge p_2 \geq 0 \Rightarrow p_3 \geq 0$, iff
$\exists\vec{\lambda} : p_3 = \lambda_1 p_1 + \lambda_2 p_2 \ \wedge\ \lambda_1 \geq 0 \ \wedge\ \lambda_2 \geq 0$

For nonlinear, we can still use this and be sound

In theory, we can preserve completeness by using Positivstellensatz

Step 2: $\exists$ to Bit-Vectors

Search for solutions in a finite range using bit-vector decision procedures

$$\exists u \in \mathbb{R} : (u^2 - 2u = 3 \wedge u > 0)$$
$$\Leftarrow \quad \exists u \in \mathbb{Z} : (u^2 - 2u = 3 \wedge u > 0)$$
$$\Leftarrow \quad \exists u \in \mathbb{Z} : (-32 \leq u < 32 \;\wedge\; u^2 - 2u = 3 \wedge u > 0)$$
$$\Leftarrow \quad \exists \vec{b} \in \mathbb{B}^6 : (u * u - 2 * u = 3 \wedge u > 0)$$

We use Yices to search for finite bit length solutions for the original nonlinear constraint

$$\vec{b} = 000011$$

Overall Approach

Given hybrid system HS and optionally property $Safe$:

- Guess a template $\mathcal{I}(\vec{u}, \vec{x})$
- Generate the verification condition: $\exists \vec{u} : \forall \vec{x} : \phi$
- Eliminate $\forall$ using Farkas' Lemma: $\exists \vec{u} : \exists \vec{\lambda} : \psi$
- Guess sizes for $\vec{u}, \vec{\lambda}$: $\exists \vec{bv_u} : \exists \vec{bv_\lambda} : \psi'$
- Ask Yices to search for solutions
- If Yices returns a satisfying assignment, system proved safe

Synthesis

- Approach is oblivious to what is unknown: system or invariant
- The unknown part of the system expressed as a template (first-order unknown variables)
- Existentially quantify the unknowns

$$\exists \vec{v}, \vec{u} : \forall \vec{x} : \phi$$

- Example: switching logic between modes: $x \leq v$
- Enforces safety locally: Can return zeno/ trivial/ systems

Experimental Results

Example	Dim	Vars	Bits	Assertions	Time
disjunction	2	14	6	50	7ms
delta-notch	4	34	8	120	30ms
plankton	3	31	8	110	56ms
thermostat	1	29	20	126	.45s
thermostat synthesis	1	21	20	75	1.2s
ACC	5	28	12	95	1.3s
acc-transmission	4	35	24	122	4.7s
insulin	7	66	18	180	18s

Table 1: The size of the Yices formulas and the time (Time) taken by Yices.

Why is the technique so effective?

- There are only so many templates
  Just one $p \geq 0$ suffices for continuous systems
- Systems have several invariants
- Correct systems have simple invariants
- SMT solvers are fast
- Robust technique does not require any careful tuning or a smart user
- Like **BMC**, SMT solver provides scalability

Conclusions

- Bounded Verification – search for bounded-size inductive invariants
- Effective for safety verification of hybrid system
- Also applicable to synthesis
- Relies on satisfiability of nonlinear constraints
- At present uses an SMT/SAT solver to search for solutions