Automated Certified Hybrid System Safety Verification

Eelis van der Weegen
J.w.w.:
Herman Geuvers
Adam Koprowski
Dan Synek

Radboud University Nijmegen

ITP 2010
Background

- C-CoRN: Coq library of constructive mathematics
- 2000, Milad Niqui: Constructive real numbers
  - Computation impractical 😞
- 2007, Russell O’Connor: Re-implementation
  - Computation practical! 😊
- “Let’s find an application that calls for certified proof-by-computation with reals!”
  → Automated hybrid system safety verification
Hybrid system: Basics

- Model of *software* interacting with *environment*
- Running example: Thermostat
- **Software**: Finite state automaton
  - Thermostat:

```
T >= 9
T <= 6  c := 0
T <= 5  c <= 1
T = -T
```

- **Environment**: Continuous space (typically $\mathbb{R}^n$).
  - Thermostat: $\mathbb{R}^2$ (= Temperature × clock)
- State of hybrid system: software state $\times$ environment state
Hybrid system: Behaviour

System state can change in two ways:

1. Discrete transition:
   - Instantaneous jump to different software state
   - “Guarded” by condition on environment state

2. Continuous transition (‘passage of time’):
   - Environment state (point in continuous space) changes according to flow
   - One flow function per location: solution to differential equations on continuous space:

   \[
   \text{flow} : \text{SoftState} \rightarrow \text{Duration} \rightarrow \text{Point} \rightarrow \text{Point}
   \]

Execution “trace”: sequence of these transitions
Hybrid system: Safety

Given:
1. designated set of *initial* states;
2. designated set of *unsafe* states
   - thermostat: states with temperature $< 4.5$

Safety problem:

*Any unsafe states reachable from initial states?*

- Undecidable in general
- Manual approach: find system invariant
- Better: Do it automatically (using heuristics)!
The predicate abstraction method (Alur, 2006)

Idea:

- Partition continuous space into finite set of \textit{regions}.
  
  \textit{Abstract system state: software state} \times \text{region}

- \textbf{Compute} \textit{abstract} discrete/continuous transitions...

- ... such that resulting graph \textit{respects} original system:
  
  If \(a \rightarrow b\) in concrete system, then \(\text{abs}(a) \rightarrow \text{abs}(b)\) in abstract system

- \textbf{Compute} reachable states in abstract system

- If no unsafe ones among them, system is safe!
Alur’s implementation is pragmatic:

- Nice language for hybrid system specification
- Integration with existing tools
- Modest preconditions on hybrid systems (linear flow/guards/etc)
- Sophisticated optimizations and data structures

But... does not produce fully verified safety proofs:

- Abstract system not provably respectful
- Uncertified implementation
- Floating point approximations of real numbers
Alur’s implementation

Alur’s implementation is pragmatic:

▶ Nice language for hybrid system specification
▶ Integration with existing tools
▶ Modest preconditions on hybrid systems (linear flow/guards/etc)
▶ Sophisticated optimizations and data structures

But... does not produce fully verified safety proofs:
▶ Abstract system not provably respectful
▶ Uncertified implementation
▶ Floating point approximations of real numbers
Our development

Our goal: *do* produce *fully verified* safety proofs.

- Formalize hybrid systems in Coq
- Reimplement abstraction method in Coq
- Keep it simple (for now)
- Different algorithm for abstract transition computation → to make respect provable
- Stronger preconditions on hybrid systems
- Use O’Connor’s “efficient” computable reals in C-CoRN
Abstract system construction: Region partitioning

- Regions in $\mathbb{R}^n$: products of $n$ intervals in $\mathbb{R}$
  - Thermostat: rectangles
- Interval bound selection (Alur):
  1. Start with constants occurring in guards/invariants (e.g. thermostat temperature intervals: 0, 4.5, 5, etc)
  2. Refine if safety unprovable for resulting abstract system
  3. Repeat

In our development:

- Automatic refinement not yet implemented
- For thermostat: refinement needed because constants from guards/invariants don’t immediately work
- Ad-hoc solution: “right” interval bounds given by user
Abstract system construction: Region partitioning

- Regions in $\mathbb{R}^n$: products of $n$ intervals in $\mathbb{R}$
  - Thermostat: rectangles
- Interval bound selection (Alur):
  1. Start with constants occurring in guards/invariants (e.g. thermostat temperature intervals: 0, 4.5, 5, etc)
  2. Refine if safety unprovable for resulting abstract system
  3. Repeat

In our development:
- Automatic refinement not yet implemented
- For thermostat: refinement needed because constants from guards/invariants don’t immediately work
- Ad-hoc solution: “right” interval bounds given by user
Abstract system construction: Continuous transitions

Question:

Given regions A and B and flow function f, is there flow from (a point in) A to (a point in) B?

- If no: no abstract transition
- If yes (or not sure): emit transition

Alur’s heuristic:

- Calculate flow at rectangle corners after $r$, $2r$, $3r$, ..., $nr$
- Use $\frac{d}{dt}$ tool to compute convex hull overapproximation
- Determine intersections with other regions (rectangles)
Abstract system construction: Continuous transitions

We use a different approach:

- Require separability of flow functions:
  \[ f_s(d, (x, y)) = (f_s, X(d, x), f_s, Y(d, y)) \]

- Require flow inverses:
  \[ f_s, X(f_s^{-1}, X(x', x), x) = x' \]

- Decide region-flowability by computing:
  - for each dimension, inverses between region bounds;
  - if no non-negative overlap: omit transition
  - otherwise: emit transition
**Computable reals**

Deciding interval overlap:
- Boils down to deciding if $a < b$ for $a, b \in \mathbb{R}$
- Or equivalently: deciding if $0 < a - b$

Can’t do it for arbitrary *computable* reals!
- Can only observe arbitrarily close $\mathbb{Q}$ approximations of $a - b$

Hence, cannot *decide* overlap in general 😞

But we don’t *need* full decidability!
- We only need “best effort” semi-deciders
Computable reals

Deciding interval overlap:
- Boils down to deciding if $a < b$ for $a, b \in \mathbb{R}$
- Or equivalently: deciding if $0 < a - b$

Can’t do it for arbitrary *computable* reals!
- Can only observe arbitrarily close \( \mathbb{Q} \) approximations of $a - b$

Hence, cannot *decide* overlap in general 😞

But we don’t *need* full decidability!
- We only need “best effort” semi-deciders
Best-effort semi-deciders:

- Underestimation for proposition $P$: term of type $option \ P$
- Naturally gives underestimators for non-overlap and flow absence

Used at higher levels, too, because abstraction method can fail:
- poor partitioning of continuous space;
- epsilon too big;
- unsafe system.

Toplevel result: $option \ TheSystemIsSafe$. 
Local classical reasoning

In Coq’s constructive logic: no PEM for arbitrary propositions 😞

But we do have it under double negation: \( \neg \neg (P \lor \neg P) \)

1. \( DN P := \neg \neg P \) is a monad
2. For some \( P \), \( P \leftrightarrow DN P \)

These stable propositions can escape from \( DN \)!

So we get to use PEM when proving stable propositions 😊

In our development, we:
- introduce strategic \( DN \) annotations and stability req’s;
- ... to make PEM (and e.g. \( a < b \) decisions in \( \mathbb{R} \)) available in their proofs
Local classical reasoning

In Coq’s constructive logic: no PEM for arbitrary propositions 😞

But we do have it under double negation: \( \neg\neg(P \vee \neg P) \)

1. \( DN \ P := \neg\neg P \) is a monad
2. For some \( P \), \( P \leftrightarrow DN \ P \)

These stable propositions can escape from \( DN \)!

So we get to use PEM when proving stable propositions 😊

In our development, we:

- introduce strategic \( DN \) annotations and stability req’s;
- ... to make PEM (and e.g. \( a < b \) decisions in \( \mathbb{R} \)) available in their proofs
Conclusions

▶ It works: we produce *fully certified, formal* proofs of hybrid system safety, in acceptable time
▶ Nice use case for proof-by-computation-with-reals
▶ *Constructive* reals do complicate theory and implementation
▶ ... but this can be dealt with systematically:
  ▶ “estimators” to make “tactics” without dropping to meta-level (Ltac)
  ▶ Double negation monad
Development works, but...

- Still very much a prototype
- No nice interface for defining hybrid system
- Strong restrictions on hybrid systems...
- ... some of which require additional proofs from user (e.g. flow invertibility)
- No automatic refinement
- Less efficient than Alur’s implementation
Future work

Continue work to get best of both worlds:

▶ Ease restrictions on hybrid systems:
  ▶ Better heuristics that don’t require flow separability
  ▶ ODE solver instead of making user provide solution
▶ Nicer user interface / specification language
▶ Implement automatic partitioning refinement
▶ Make C-CoRN reals faster
▶ Conditional guarantees that safety can be determined
▶ Failure traces