Nonlinear Control as Program Synthesis (A Starter)

Sicun Gao

MIT

December 15, 2014
Preliminaries

Definition \( \mathcal{L}_{\mathbb{R}_F} \)

\( \mathcal{L}_{\mathbb{R}_F} \) is the first-order language over the reals that allows arbitrary “numerically computable” functions in \( F \).

- \( \{ \text{polys, exp, sin, ODE}^*, PDE^*, \ldots \} \subseteq F \).

Example (Lyapunov Stability)

\[
\forall \varepsilon \exists \delta \forall t \forall x_0 \forall x. \left( (\|x_0\| < \delta) \land (x = x_0 + \int_0^t f(s)ds) \right) \rightarrow (\|x\| < \varepsilon).
\]
Definition (Delta-Decisions)

Given a bounded first-order sentence $\varphi$ over real numbers and $\delta \in \mathbb{Q}^+$, decide

- $\varphi$ is false
- $\varphi^{-\delta}$ is true.

Theorem (Delta-Decidability)

- Bounded $\mathcal{L}_{R,F}$-sentences are $\delta$-decidable.
- Complexity: $\delta$-decisions of $\Sigma_n$-sentences are in $(\Sigma_n^P)^C$. 
Theoretical Implications

Bounded first-order properties of general nonlinear hybrid systems are mostly $\delta$-decidable with reasonable complexity:

- Bounded Lyapunov stability: $(\Pi_3^P)^C$.
- Bounded reachability of hybrid systems: $\text{NP}^C$ to $(\Sigma_3^P)^C$.
- Invariant checking for hybrid systems: $(\text{coNP})^C$.

All within PSPACE, in sharp contrast to undecidability results.
dReal: nonlinear SMT solving over the reals.

- almost all the nonlinear functions you need: exp, sin, ODEs, ...
- witnesses for delta-SAT, proofs for UNSAT
- an interface for reachability of nonlinear hybrid systems
- solved many challenging benchmarks: Toyota, biological models (cardiac cells, prostate cancer), quadcopters, Kepler conjecture, floating-point implementations, etc.
What’s Next?

- $\mathcal{L}_{\mathbb{R}^+}$-formulas with arbitrary quantification are $\delta$-decidable.
- dReal has shown the $\Sigma_1$/SMT case can be made quite practical.

We are now moving one level up:

**Solve $\exists\forall$-sentences.**

This is the move from *posterior verification* to automating reliable design and implementation.
Control Design

Given a controlled system with some initial conditions:

\[ \frac{dx}{dt} = f(x, t) + g(x, t)u(x, t) \]

The goal of control design is to find a control function \( u \) such that certain conditions are true (stability, safety, ...).
To design a controller $u(x)$ is to solve formula like:

$$
\exists u(x, t) \forall x \forall t \forall x_0. \left( I(x_0) \land x = x_0 + \int_0^t (f + g \cdot u)ds \right) \rightarrow \varphi(x, t)
$$

- Second-order quantification on $u(x)$. Different properties involve different first-order quantification and other variables.
- In the case of linear continuous systems, control theory gives us almost complete algorithms for solving many of these second-order formulas!
To control a continuous system is also to implement $U(x)$ such that a program like the following can run successfully.

```
repeat(dt, [0, T]){  
    assert(loop invariants);
    
    u = U(x);
    x = x + f(x) * dt + g(x) * u * dt;
}
assert(post conditions);
```
Going Hybrid Is Easy

repeat(dt, [0, T])
{
    assert(loop invariants);

    if ... then
        u = U1(x);
    else
        u = U2(x);

    if ... then
        x = x + f1(x) * dt + g1(x) * u * dt;
    else
        x = x + f2(x) * dt + g2(x) * u * dt;
}
assert(post conditions);
Blurring Modelling and Coding

High-level models and low-level code are all programs in different Turing-complete languages.

- The real question is whether there is a right set of languages to express both high-level and low-level “ideas” interconnectively.
- Ideally if we can do everything in one language then what we design is what we implement.

But how can we control the complexity of writing in such a language?
Automate low-level implementation so that engineers can write “high-level” code in a low-level language.

A high-level program is simply sketches [Solar-Lezama PLDI05’] that can abstract away chunks of low-level code, by allowing “holes” in the program.
For a system $\dot{x} = x^2 + \sin(x)$, we want to design a nonlinear controller that stabilizes it.

\[
\begin{align*}
x &= 0; \\
\text{repeat}(dt, [0, T]) \{ \\
\quad u = \ ? * x + \ ? * x^3; \quad //? \text{ is a hole for parameters} \\
\quad x &= x + (x^2 + \sin(x) + u) * dt; \\
\} \\
\text{assert}(\text{Lyapunov\_function}: \text{poly}(x) \text{ of degree 4})
\end{align*}
\]

Synthesize $u$ and a Lyapunov function at the same time.
We should solve:

$$
\exists u(x) \exists V(x, u(x)) \forall x.
$$

$$
V(0, u(0)) = 0 \land (x \neq 0) \rightarrow \left( V(x, u) > 0 \land \frac{dV(x, u)}{dt} \leq 0 \right)
$$

With sketches, this is reduced to solving for the parameters $\vec{p}_u, \vec{p}_v$:

$$
\exists \vec{p}_u \exists \vec{p}_v \forall x.
$$

$$
V(0, u(0)) = 0 \land (x \neq 0) \rightarrow \left( V(x, u) > 0 \land \frac{dV(x, u)}{dt} \leq 0 \right)
$$

which is an $\exists \forall$-sentence.
Solving $\exists \forall$

- Solving $\exists \forall$ is like playing a game between the designer ($\exists$ variables) and the verifier ($\forall$ variables).
- Designer suggests assignments and the verifier suggests a counterexample, both by calling an SMT solver.
- If either one of them is out of choices, we terminate with a definite answer.

It converges with a lot of ideas in control design, but automated reasoning engines can greatly enhance them. At the same time, a lot of program analysis and synthesis techniques come into play.
Benefits

- Engineer writes, in a high-level language, sketches of C-like programs.
- Easy simulation by simply running the code.
- Solvers automate design and implementation at the same time. The result can simply be executable C programs.
- Complete formal proofs can be *automatically* generated from the solving process.
- Familiar GUI to control engineers such as block diagrams can be dynamically generated.
Conclusion

- Moving from posterior verification to the design and implementation phase may produce problems that are algorithmically harder but socially easier.
- We can aim to enhance automation and correctness by construction at the same time.
- Formal methods should learn more from control theory. Control theory itself (not just engineering) may benefit from formal methods.

We should have reasonable solver support for $\exists\forall$-formulas ready in dReal within next year.