Formal methods in analysis

Jeremy Avigad

Department of Philosophy and
Department of Mathematical Sciences
Carnegie Mellon University

May 2015

Sequence of lectures

1. Formal methods in mathematics
2. Automated theorem proving
3. Interactive theorem proving
4. Formal methods in analysis

Recap

Remember: “formal methods” = logical methods in computer science.

Automated methods: SAT solvers, equational reasoning, resolution theorem provers, . . .

- “domain general”: general logical languages
- emphasis on performance
- undecidability, combinatorial explosion set in.

Interactive methods:

- also “domain general”: use a broad foundational framework
- emphasis on precision, rigor
- requires a lot of work on the part of the user

Today

Formal methods in analysis:

- domain specific: reals, integers
- can emphasize either peformance or rigor
- get further in restricted domain

Outline:

- quantifier elimination
- the universal fragment of real closed fields
- combination methods
- numerical methods
- a heuristic symbolic method

Quantifer elimination

A *first-order language* is given by some function and relation symbols.

Example: the language of ordered rings: $0, 1, +, \times, \leq$.

First-order formulas are built up using quantifiers and connectives.

Example: $\forall x, y\ \exists z\ (z * z = x * x + y * y)$.

A *structure* for a language consists of a universe, plus interpretations of the symbols.

Example: $\langle \mathbb{R}, 0, 1, +, \times, \leq \rangle$.

Quantifier elimination

A *theory* T is a set of sentences, closed under logical consequences.

Example: for any structure $\mathcal{M}$, $Th(\mathcal{M}) = \{\varphi \mid \mathcal{M} \vDash \varphi\}$.

Example: for any set of axioms A, $Con(A) = \{\varphi \mid A \vDash \varphi\}$.

A theory T is said to have (effective) *quantifier elimination* if (effectively) every formula φ has a quantifier-free equivalent φ'.

Example: Over $\mathbb{R}$, $\exists x\ (ax^2 + bx + c = 0) \equiv \ldots$

If the quantifier-free sentences are decidable, the theory is then decidable.

Quantifier elimination

Some theories with quantifier elimination:

- linear arithmetic:
$$\mathit{Th}(\langle \mathbb{R}, 0, 1, +, \leq \rangle)$$
- integer linear arithmetic (Presburger 1930):
$$\mathit{Th}(\langle \mathbb{Z}, 0, 1, +, \leq \rangle)$$
- real-closed fields (Tarski c. 1930):
$$\mathit{Th}(\langle \mathbb{R}, 0, 1, +, \times, \leq \rangle)$$
- algebraically closed fields (Tarski c. 1930):
$$\mathit{Th}(\langle \mathbb{C}, 0, 1, +, \times \rangle)$$

The Fourier-Motzkin procedure

Theorem. The theory of $\langle \mathbb{R}, 0, 1, +, < \rangle$ has quantifier-elimination, and so is decidable.

Proof. It suffices to show that if φ is quantifier-free, $\exists x\, \varphi$ is equivalent to a quantifier-free formula.

Note:

- Can put φ in disjunctive normal form.
- $\exists x\, (\theta \vee \eta)$ is equivalent to $\exists x\, \theta \vee \exists x\, \eta$.
- $s \neq t$ is equivalent to $s < t \vee t < s$.
- $s \not< t$ is equivalent to $t < s \vee s = t$.

So, it suffices to assume φ is a conjunction of equalities and strict inequalities.

The Fourier-Motzkin procedure

Expressions that don't involve x can be brought outside the existential quantifier.

Using rational coefficients, can put expressions involving x in *pivot form*:

- $x = s$
- $x < s$
- $s < x$

φ is a conjunction of these.

If any conjunct has the form $x = s$, $\exists x\ \varphi(x)$ is equivalent to $\varphi(s)$, and we're done.

The Fourier-Motzkin procedure

Otherwise, φ is a conjunction of formulas of the form $s_i < x$ and $x < t_j$.

It is not hard to check that $\exists x\ \varphi$ is equivalent to

$$\bigwedge_{i,j} s_i < t_j.$$

Notes:

- Can allow multiplicative coefficients from any computable field.
- The theory is the theory of a divisible ordered group.
- Even the existential fragment is doubly-exponential in principle, but it works well on small problems, in practice.

Presburger arithmetic

Let $T = Th(\langle \mathbb{Z}, 0, 1, +, < \rangle)$.

Add binary relations $s \equiv_n t$, meaning "$t - s$ is divisible by n", for each fixed n.

Theorem (Presburger, 1930). T has elimination of quantifiers, and is hence decidable.

Tarski, Presburger's advisor, did not think it merited a doctorate.

Real closed fields

Let $T = Th(\langle \mathbb{R}, 0, 1, +, \times, < \rangle)$.

Theorem (Tarski, around 1930). T has elimination of quantifiers, and is hence decidable. It is the same as the theory of "real-closed fields" (which is hence complete).

Theorem. T is the theory of "real closed fields."

Corollary. T is decidable.

Corollary. Any two real closed fields are elementarily equivalent.

Real closed fields

Implementations:

- Alfred Tarski proved this around 1930 (finally published in 1948), based on Sturm's theorem.
- Abraham Robinson gave an easier model-theoretic proof in 1956, based on Artin-Schreier.
- George Collins gave a *practical* method in 1975, "cylindrical algebraic decomposition".
- Implementations: Mathematica, Qepcad (now in Sage), Reduce (Redlog), RAHD.
- Sean McLaughlin and John Harrison implemented a proof-producing version for HOL light (it is slow).
- Procedures for RCF are still actively studied, from theoretical and practical perspectives.

Applications

Application #1: For each n, and k, the statement "the kissing number in $\mathbb{R}^n$ is at least k" is a formula in the language of real-closed fields.

Application #2: The first step in Hales' proof of the Kepler conjecture reduces the problem to a finite optimization problem that can, in principle, be expressed in the language of real closed fields.

The bad news: these are way out of reach of the current technology (except for trivial instances of the first).

The best QE procedures are doubly exponential, and this is sharp.

Formal methods in analysis

Outline:

- quantifier elimination
- the universal fragment of RCF
- combination methods
- numerical methods
- a heuristic symbolic method

The universal fragment of RCF

The universal fragment of RCF has special features.

For example, let $p(\vec{x}) \in \mathbb{Z}[x]$.

The Artin-Schreier positive solution to Hilbert's 17th problem (1927) shows that $\forall \vec{x}\ p(\vec{x}) \geq 0$ iff p can be written as a sum of squares of *rational functions*.

A positivstellensatz

In the language of real closed fields, it is sufficient to consider universal formulas of the form:

$$\begin{aligned}\forall \vec{x}\ \neg(& p_1(\vec{x}) = 0 \wedge \ldots \wedge p_n(\vec{x}) = 0 \wedge \\ & q_1(\vec{x}) \geq 0 \wedge \ldots \wedge q_m(\vec{x}) \geq 0 \wedge \\ & r_1(\vec{x}) \neq 0 \wedge \ldots r_p(\vec{x}) \neq 0).\end{aligned}$$

This holds iff there are polynomials P, Q, R such that

- $P + Q + R^2 = 0$
- P is in the ideal generated by $p_1, \ldots, p_n$
- Q is in the "cone" generated by $q_1, \ldots, q_m$
- R is a product of powers of $r_1, \ldots, r_n$

The universal fragment of RCF

Pablo Parillo has introduced the use of semidefinite programming to find such certificates.

John Harrison has developed infrastructure to verify such certificates in the *HOL light* theorem prover.

See John Harrison, "Verifying nonlinear real formulas via sums of squares."

Combination methods

Goal: combine domain-general search procedures with domain specific methods.

Paulson's *MetiTarski* uses

- a resolution theorem prover
- RCF back ends (Qepcad, Z3, Mathematica)

to prove general real-valued inequalities.

Some ideas:

- Replace transcendental functions by bounding rational functions (iteratively).
- Use resolution proof search.
- Split on cases for absolute values, etc.
- Use RCF for "literal deletion," i.e. to infer RCF entailments.

SMT solvers

Remember "satisfiability modulo theories":

- Based on DPLL search for a satisfying propositional assignment.
- Uses CDCL: conflict-driven clauses learning.
- Combines decision procedures / proof procedures for universal fragments.

Strategy:

- A core propositional logic engine tries to assign values of true / false to literals.
- Each "module" inspects the proposals, reports conflicts.
- The core engine "learns," backtracks, and continues the search.

SMT solvers

There are modules based on:

- simplex methods (linear real arithmetic)
- cutting-plane methods (linear integer arithmetic)
- CAD (real closed fields)

Beyond testing feasibility, for SMT, algorithms need to:

- work incrementally
- backtrack efficiently
- produce “explanations”

SMT solvers

There is a large literature; see, for example:

Bruno Dutertre and Leonardo de Moura, “A fast linear-arithmetic solver for DPLL(T),” *CAV*, 2006.

Dejan Jovanović and Leonardo de Moura, “Solving non-linear arithmetic,” *CADE*, 2012.

Formal methods in analysis

Outline:

- quantifier elimination
- the universal fragment of RCF
- combination methods
- numerical methods
- a heuristic symbolic method

Numerical methods

Decision procedures for real closed fields represent a symbolic approach.

Problems:

- Complexity overwhelms.
- Polynomials may not be expressive enough.
- Undecidability sets in quickly.

How can we integrate numeric approaches?

- Calculations are only approximate.
- We want an exact guarantee.

Approximate decidability

Sicun Gao, Ed Clarke, and I proposed a framework that offers:

- More flexibility: arbitrary computable functions
- A restriction: quantification only over bounded domains
- A compromise: approximate decidability (but with an exact guarantee)

This provides a general framework for thinking about verification problems.

Computable analysis

A real number r is *computable* if there is a computable function $\alpha : \mathbb{N} \to \mathbb{Q}$ such that for every i, $|\alpha(i) - r| < 2^{-i}$.

Call such an α a *name*. So r is computable if it has a computable name.

A function $f : \mathbb{R} \to \mathbb{R}$ is computable if, given a name α for r as input (say, as an oracle), it computes a name for $f(r)$.

Fact: a computable function from $\mathbb{R}$ to $\mathbb{R}$ is continuous.

Computable analysis

Most real numbers arising “in nature” are computable:

$$\pi, e, \gamma, \phi, \ldots$$

Similarly, continuous functions arising in nature are computable:

- polynomials
- trigonometric functions
- exp, log
- absolute value, min, and max
- solutions to ordinary differential equations with Lipschitz-continuous computable functions

Computable analysis

Note that the function

$$f(x) = \begin{cases} 1 & \text{if } x \geq 0 \\ 0 & \text{otherwise} \end{cases}$$

is not computable. In other words, we cannot decide $x \geq 0$.

Fix a small δ. We can do the next best thing, that is, decide

- $x \geq 0$
- $x \leq \delta$

Note that there is a “grey area” where either answer is o.k.

(Note also that the procedure cannot be extensional.)

δ-decidability

Choose a language with 0, $+$, $-$, $<$, $\leq$, $|\cdot|$, and symbols for *any* computable functions you want.

Fix a “tolerance” $\delta > 0$. We defined:

- $\varphi^{+\delta}$, a slight strengthening of φ
- $\varphi^{-\delta}$, a slight weakening of φ

such that whenever $\delta' \geq \delta \geq 0$, we have

$$\varphi^{+\delta'} \to \varphi^{+\delta} \to \varphi \to \varphi^{-\delta} \to \varphi^{-\delta'}.$$

δ-decidability

Say a formula φ is *bounded* if every quantifier is of the form $\forall x \in [s, t]$ or $\exists x \in [s, t]$.

Theorem. There is an algorithm which, given any bounded formula φ, correctly returns on of the following two answers:

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

For verification problems, think of the first answer as "the system is safe," and the second as "a small perturbation of the system is unsafe."

Note that there is a grey area where either answer is allowed.

δ-decidability

This is a theoretical result. The practical goal is to implement such an algorithm.

Gao, Clarke, Soonho Kong, and others are developing a tool, *dreal*:

- It focuses on the existential / universal fragment.
- It uses an SMT framework.
- It uses interval constraint propagation.
- It uses numerical methods and CAPA packages for ODE's.

See https://dreal.github.io.

Formal methods in analysis

Outline:

- quantifier elimination
- the universal fragment of RCF
- combination methods
- numerical methods
- a heuristic symbolic method

An example

Consider the following implication:

$$\begin{aligned} &0 < x < y,\ u < v \\ &\qquad\quad \Longrightarrow \\ &\qquad\qquad 2u + \exp(1 + x + x^4) < 2v + \exp(1 + y + y^4) \end{aligned}$$

- This inference is not contained in linear arithmetic or real closed fields.
- This inference is tight: symbolic or numeric approximations are not useful.
- Backchaining using monotonicity properties suggests many equally plausible subgoals.
- But, the inference is completely straightforward.

A new method

Robert Lewis and I (initially with Cody Roux) have developed a new aproach, that:

- verifies inequalities on which other procedures fail
- extends beyond the language of RCF
- is amenable to producing proof terms
- captures natural patterns of inference

But:

- It is not complete.
- It not guaranteed to terminate.

It is designed to complement other procedures.

Polya

We have a prototype Python implementation, *Polya*.

The code is open-source and available online.

- An associated paper.
- Rob's MS thesis.
- Slides from Rob's talks (from which I have borrowed).

We are planning to implement this in Lean.

Overview

Our system verifies inequalities between real variables using:

- operations $+$ and $\cdot$
- multiplication and exponentiation by rational constants
- arbitrary function symbols
- relations $<$ and $=$

As with resolution theorem proving, we establish a theorem by negating the conclusion and deriving a contradiction.

Terms and normal forms

The term

$$3(3y + 5x + 4xy)^2 f(u + v)^{-1}$$

is expressed canonically as

$$75 \cdot \underbrace{(\underbrace{\underbrace{x}_{t_1} + \frac{3}{5} \cdot \underbrace{y}_{t_2} + \frac{4}{5} \cdot \underbrace{xy}_{t_3=t_1 t_2}}_{t_6=t_1+\frac{3}{5}t_2+\frac{4}{5}t_3})^2 \underbrace{f(\underbrace{\underbrace{u}_{t_4} + \underbrace{v}_{t_5}}_{t_7=t_4+t_5})}_{t_8=f(t_7)}{}^{-1}}_{t_9=t_6^2 t_8^{-1}}$$

Modules and database

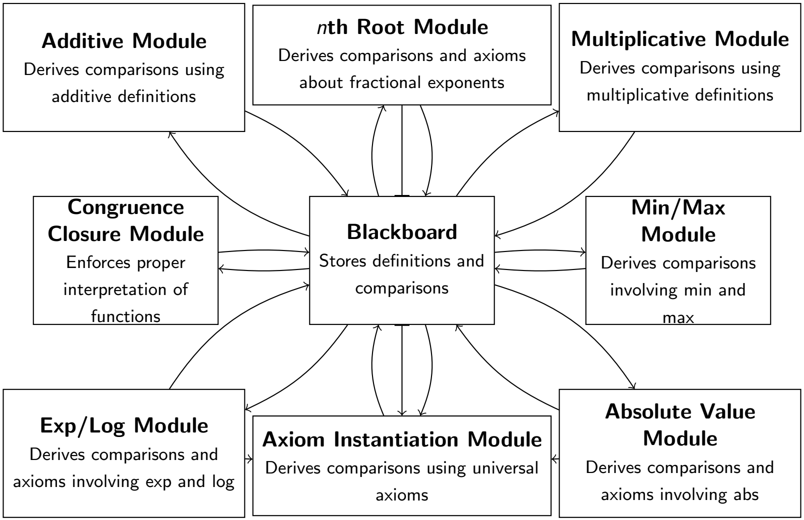
Any comparison between canonical terms can be expressed as $t_i \bowtie 0$ or $t_i \bowtie c \cdot t_j$, where $\bowtie \in \{=, \neq, <, \leq, >, \geq\}$.

A central database (the *blackboard*) stores term definitions and comparisons of this form.

Various modules use this information to learn and assert new comparisons.

The procedure has succeeded in verifying an implication when modules assert contradictory information.

Computational structure


Additive Module
Derives comparisons using additive definitions
*n*th Root Module
Derives comparisons and axioms about fractional exponents
Multiplicative Module
Derives comparisons using multiplicative definitions
Congruence Closure Module
Enforces proper interpretation of functions
Blackboard
Stores definitions and comparisons
Min/Max Module
Derives comparisons involving min and max
Exp/Log Module
Derives comparisons and axioms involving exp and log
Axiom Instantiation Module
Derives comparisons using universal axioms
Absolute Value Module
Derives comparisons and axioms involving abs