Numerical Computations and Formal Methods

Guillaume Melquiond

Proval, Laboratoire de Recherche en Informatique
INRIA Saclay–IdF, Université Paris Sud, CNRS

October 28, 2009

Numerical Computations and Formal Methods

Deductive Program Verification

1. Deductive program verification
   - Floyd-Hoare logic and weakest preconditions
   - A framework for program verification: Why
   - Gappa
2. Computing in a formal system
3. Decision procedures for arithmetic theories
4. Conclusion

Hoare Triple

Definition (Hoare triple)

$$\{\mathit{precondition}\} \quad \mathit{code} \quad \{\mathit{postcondition}\}.$$

Meaning of correctness:
If the precondition holds just before the code is executed,
the postcondition holds just after it has been executed.

Hoare Triple

Definition (Hoare triple)

$$\{precondition\} \quad code \quad \{postcondition\}.$$

Meaning of correctness:
If the precondition holds just before the code is executed, the postcondition holds just after it has been executed.

Note: the definition assumes the code terminates.
If it does not, any postcondition holds, including False.

Hoare Triple

```
{ x >= 0 }
y = floor(sqrt(x))
{ y >= 0 and y*y <= x < (y+1)*(y+1) }
```

Weakest Precondition

Definition (Weakest precondition)

R is the weakest precondition of a code C and a postcondition Q iff any correct triple $\{P\}\ C\ \{Q\}$ satisfies $P \Rightarrow R$.

Weakest Precondition

Definition (Weakest precondition)

R is the weakest precondition of a code C and a postcondition Q iff any correct triple $\{P\}\ C\ \{Q\}$ satisfies $P \Rightarrow R$.

A function behaves correctly (modulo termination)
if its specification can be expressed as a correct triple.

Weakest Precondition

Definition (Weakest precondition)

R is the weakest precondition of a code C and a postcondition Q iff any correct triple $\{P\}\ C\ \{Q\}$ satisfies $P \Rightarrow R$.

A function behaves correctly (modulo termination)
if its specification can be expressed as a correct triple.

How to verify it?

- Compute the weakest precondition (Dijkstra, 1975) from the function and its specified postcondition.
- Prove that the specified precondition implies the weakest one.

A Framework for Program Verification: Why

WHY is a minimal system:

- small ML-like programming language,
- small specification language.

A Framework for Program Verification: Why

WHY is a minimal system:

- small ML-like programming language,
- small specification language.

WHY is an intermediate environment:

- it computes weakest preconditions;
- it generates VCs for provers, interactive or not.

A Framework for Program Verification: Why

WHY is a minimal system:

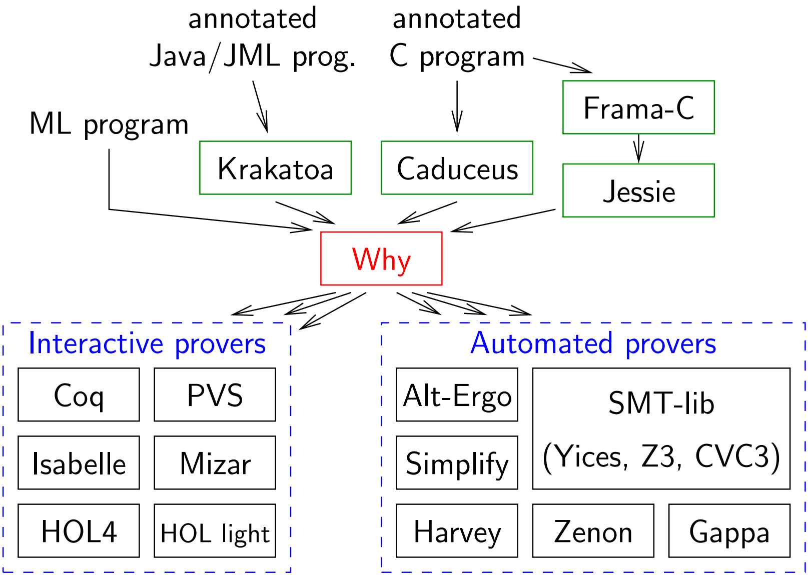
- small ML-like programming language,
- small specification language.

WHY is an intermediate environment:

- it computes weakest preconditions;
- it generates VCs for provers, interactive or not.

Various tools translate programing languages (C, Java) to the ML language.

Environment



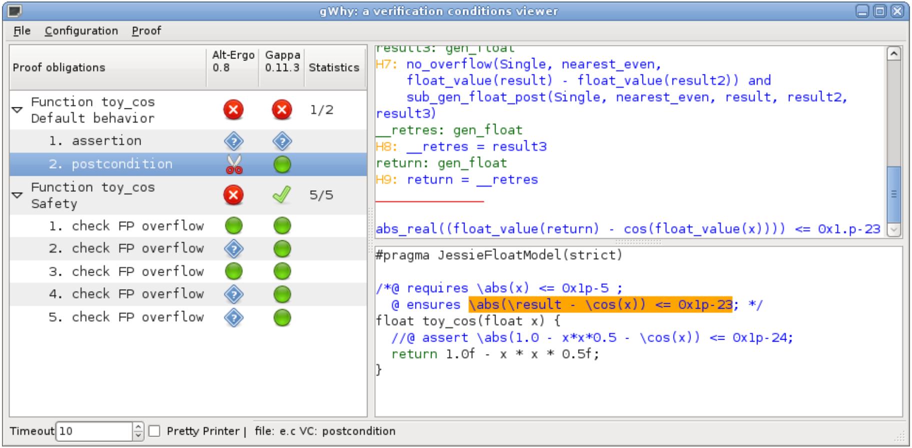
Toy Example: Cosine Around Zero

```
/*@ requires \abs(x) <= 0x1p-5 ;
  @ ensures \abs(\result - \cos(x)) <= 0x1p-23; */
float toy_cos(float x) {
  //@assert \abs(1.0-x*x*0.5 - \cos(x)) <= 0x1p-24;
  return 1.0f - x * x * 0.5f;
}
```

"\result" is the value returned by the function, that is:
$1 - 0.5 \cdot x^2$ with all the operations rounded to nearest binary32.

- Safety: none of the operations overflow nor are invalid.
- Correctness: the result is almost the mathematical cosine.

Frama-C/Jessie/Why + Gappa



Verifying Arithmetic Properties

Kind of properties:

- Precondition validity:
  - no overflow: $\forall \vec{x},\ f(\vec{x}) \in D$;

Verifying Arithmetic Properties

Kind of properties:

- Precondition validity:
  - no overflow: $\forall \vec{x},\ f(\vec{x}) \in D$;
  - no domain error: $\forall \vec{x},\ d(f(\vec{x}), g(\vec{x}), \cdots) \in D$.

Verifying Arithmetic Properties

Kind of properties:

- Precondition validity:
  - no overflow: $\forall \vec{x},\ f(\vec{x}) \in D$;
  - no domain error: $\forall \vec{x},\ d(f(\vec{x}), g(\vec{x}), \cdots) \in D$.
- Accuracy of results:
  - absolute error: $\forall \vec{x},\ f(\vec{x}) - g(\vec{x}) \in E$;

Verifying Arithmetic Properties

Kind of properties:

- Precondition validity:
  - no overflow: $\forall \vec{x},\ f(\vec{x}) \in D$;
  - no domain error: $\forall \vec{x},\ d(f(\vec{x}), g(\vec{x}), \cdots) \in D$.
- Accuracy of results:
  - absolute error: $\forall \vec{x},\ f(\vec{x}) - g(\vec{x}) \in E$;
  - relative error: $\forall \vec{x},\ \exists \varepsilon,\ f(\vec{x}) = g(\vec{x}) \times (1 + \varepsilon)$.

Verifying Arithmetic Properties

Kind of properties:

- Precondition validity:
  - no overflow: $\forall \vec{x},\ f(\vec{x}) \in D$;
  - no domain error: $\forall \vec{x},\ d(f(\vec{x}), g(\vec{x}), \cdots) \in D$.
- Accuracy of results:
  - absolute error: $\forall \vec{x},\ f(\vec{x}) - g(\vec{x}) \in E$;
  - relative error: $\forall \vec{x},\ \exists \varepsilon,\ f(\vec{x}) = g(\vec{x}) \times (1 + \varepsilon)$.

Language of formulas:

- intervals with nonsymbolic bounds,
- expressions with mathematical operators (e.g., $\times$, tan) and rounding operators (e.g., $\lfloor \cdot \rceil$).

Gappa

Input: logical formula about expressions on real numbers.
Output: "Yes" and a formal proof, or "I don't know".

Gappa

Input: logical formula about expressions on real numbers.
Output: "Yes" and a formal proof, or "I don't know".

Method: saturation over a set of theorems.

- Naive interval arithmetic:
  $u \in [\underline{u}, \overline{u}] \wedge v \in [\underline{v}, \overline{v}] \Rightarrow u + v \in [\underline{u} + \underline{v}, \overline{u} + \overline{v}]$.

Gappa

Input: logical formula about expressions on real numbers.
Output: "Yes" and a formal proof, or "I don't know".

Method: saturation over a set of theorems.

- Naive interval arithmetic:
  $u \in [\underline{u}, \overline{u}] \wedge v \in [\underline{v}, \overline{v}] \Rightarrow u + v \in [\underline{u} + \underline{v}, \overline{u} + \overline{v}]$.
- Floating-/fixed-point arithmetic properties:
  $u \in 2^{-1074} \cdot \mathbb{Z} \Rightarrow \exists \varepsilon \in [-2^{-53}, 2^{-53}],\ \circ(u) = u \times (1 + \varepsilon)$.

Gappa

Input: logical formula about expressions on real numbers.
Output: "Yes" and a formal proof, or "I don't know".

Method: saturation over a set of theorems.

- Naive interval arithmetic:
  $u \in [\underline{u}, \overline{u}] \wedge v \in [\underline{v}, \overline{v}] \Rightarrow u + v \in [\underline{u} + \underline{v}, \overline{u} + \overline{v}]$.
- Floating-/fixed-point arithmetic properties:
  $u \in 2^{-1074} \cdot \mathbb{Z} \Rightarrow \exists \varepsilon \in [-2^{-53}, 2^{-53}],\ \circ(u) = u \times (1 + \varepsilon)$.
- Forward error analysis:
  $\tilde{u} \times \tilde{v} - u \times v = (\tilde{u} - u) \times v + u \times (\tilde{v} - v) + (\tilde{u} - u) \times (\tilde{v} - v)$.
- ...

Computing in a Formal System

Example: Peano's Arithmetic

Inductive definition of natural numbers:

```
type nat = O | S of nat    (* 5 = SSSSSO *)
```

Axioms for addition:

add0: $\forall b,\ \mathrm{O} + b = b$
addS: $\forall a\ b,\ (\mathrm{S}\ a) + b = a + (\mathrm{S}\ b)$

Example: Peano's Arithmetic

Deductive proof of $4 + (2 + 3) = 9$: (9 steps)

$$
\dfrac{\dfrac{\dfrac{\begin{array}{c}\dfrac{\dfrac{}{9 = 9}\ \text{reflexivity}}{0 + 9 = 9}\ \mathtt{add0} \\ \vdots\ \mathtt{addS} \times 4 \\ 4 + 5 = 9\end{array}}{4 + (0 + 5) = 9}\ \mathtt{add0}}{4 + (1 + 4) = 9}\ \mathtt{addS}}{4 + (2 + 3) = 9}\ \mathtt{addS}
$$

Introducing Computations into Proofs

Recursive definition of addition:

```
let rec plus x y =
  match x with
  | O -> y
  | S x' -> plus x' (S y)
```

Lemma `plus_xlate`: $\forall a\ b,\ a + b = \texttt{plus}\ a\ b$

Introducing Computations into Proofs

Recursive definition of addition:

```
let rec plus x y =
  match x with
  | O -> y
  | S x' -> plus x' (S y)
```

Lemma `plus_xlate`: $\forall a\ b,\ a + b = \texttt{plus}\ a\ b$

Proof of $4 + (2 + 3) = 9$: (4 steps)

$$\dfrac{\dfrac{\dfrac{\dfrac{}{9 = 9}\ \text{reflexivity}}{\texttt{plus}\ 4\ (\texttt{plus}\ 2\ 3) = 9}\ ???}{4 + (\texttt{plus}\ 2\ 3) = 9}\ \texttt{plus\_xlate}}{4 + (2 + 3) = 9}\ \texttt{plus\_xlate}$$

Type Theory and Conversion

Curry-Howard correspondence and type theory:

1. Proposition A holds if the type A is inhabited.
2. Convertible types have the same inhabitants.

$$\frac{p : A}{p : B}\ A \equiv_\beta B$$

Type Theory and Conversion

Curry-Howard correspondence and type theory:

1. Proposition A holds if the type A is inhabited.
2. Convertible types have the same inhabitants.

$$\frac{p : A}{p : B}\ A \equiv_\beta B$$

Proof of $4 + (2 + 3) = 9$: (4 steps)

$$\cfrac{\cfrac{\cfrac{\overline{p : 9 = 9}\ \text{reflexivity}}{p : \texttt{plus}\ 4\ (\texttt{plus}\ 2\ 3) = 9}\ \beta\text{-reduction}}{4 + (\texttt{plus}\ 2\ 3) = 9}\ \texttt{plus\_xlate}}{4 + (2 + 3) = 9}\ \texttt{plus\_xlate}$$

Encoding Expressions

Inductive definition of expressions on natural numbers:

```
type expr = Nat of nat | Add of expr * expr
let rec interp_expr e =
  match e with
  | Nat n -> n
  | Add (x, y) ->
    (interp_expr x) "+" (interp_expr y)
```

Proof of $4 + (2 + 3) = 9$:

$$\dfrac{\dfrac{???}{\texttt{interp\_expr}\ (\texttt{Add}\ (\texttt{Nat}\ 4, \texttt{Add}\ (\texttt{Nat}\ 2, \texttt{Nat}\ 3))) = 9}}{4 + (2 + 3) = 9}\ \beta\text{-reduction}$$

Evaluating Expressions

Evaluating expressions on natural numbers:

```
let rec eval_expr e =
  match e with
  | Nat n -> n
  | Add (x, y) ->
    plus (eval_expr x) (eval_expr y)
```

Lemma expr_xlate: $\forall e$ interp_expr $e =$ eval_expr e

Evaluating Expressions

Evaluating expressions on natural numbers:

```
let rec eval_expr e =
  match e with
  | Nat n -> n
  | Add (x, y) ->
    plus (eval_expr x) (eval_expr y)
```

Lemma expr_xlate: $\forall e \quad \texttt{interp\_expr}\ e = \texttt{eval\_expr}\ e$

Proof of $4 + (2 + 3) = 9$:

$$\dfrac{\dfrac{\dfrac{\dfrac{}{9 = 9}\ \text{reflexivity}}{\texttt{eval\_expr (Add (Nat 4,} \ldots\texttt{))} = 9}\ \beta\text{-reduction}}{\texttt{interp\_expr (Add (Nat 4,} \ldots\texttt{))} = 9}\ \texttt{expr\_xlate}}{4 + (2 + 3) = 9}\ \beta\text{-reduction}$$

Relational Operators

Equality is usually a native concept, while comparisons are not.

Comparing natural numbers:

```
let rec le x y =
  match x, y with
  | O   , _    -> true
  | S _ , O    -> false
  | S x', S y' -> le x' y'
```

Lemma: $\forall a \forall b \quad \texttt{le}\ a\ b = \mathit{true} \Leftrightarrow a \leq b$

Encoding Comparisons

Inductive definition of relations on natural expressions:

```
type prop = Le of expr * expr
let interp_prop p =
  match p with
  | Le (x, y) ->
    (interp_expr x) "<=" (interp_expr y)
let eval_prop p =
  match p with
  | Le (x, y) -> le (eval_expr x) (eval_expr y)
```

Encoding Comparisons

Inductive definition of relations on natural expressions:

```
type prop = Le of expr * expr
let interp_prop p =
  match p with
  | Le (x, y) ->
    (interp_expr x) "<=" (interp_expr y)
let eval_prop p =
  match p with
  | Le (x, y) -> le (eval_expr x) (eval_expr y)
```

Proof of $4 + (2 + 3) \leq 5 + 6$:

$$\dfrac{\dfrac{\dfrac{}{\mathit{true} = \mathit{true}}\ \text{reflexivity}}{\texttt{eval\_prop (Le (Add ..., Add ...))} = \mathit{true}}\ \beta\text{-reduction}}{\dfrac{\texttt{interp\_prop (Le (Add ..., Add ...))}}{4 + (2 + 3) \leq 5 + 6}\ \beta\text{-reduction}}\ \texttt{prop\_xlate}$$

Some Formalizations of Arithmetic in Coq

- Integers as lists of bits:
  polynomial equality, semi-decision of $(\mathbb{Z}, +, =, <)$.

Some Formalizations of Arithmetic in Coq

- Integers as lists of bits:
  polynomial equality, semi-decision of $(\mathbb{Z}, +, =, <)$.
- Rational numbers and Bernstein polynomials:
  global optimization for Hales' inequalities.

Some Formalizations of Arithmetic in Coq

- Integers as lists of bits:
  polynomial equality, semi-decision of $(\mathbb{Z}, +, =, <)$.
- Rational numbers and Bernstein polynomials:
  global optimization for Hales' inequalities.
- Dyadic numbers and intervals:
  verification of Gappa certificates.

Some Formalizations of Arithmetic in Coq

- Integers as lists of bits:
  polynomial equality, semi-decision of $(\mathbb{Z}, +, =, <)$.
- Rational numbers and Bernstein polynomials:
  global optimization for Hales' inequalities.
- Dyadic numbers and intervals:
  verification of Gappa certificates.
- Integers as binary trees of machine words:
  verification of Pocklington primality certificates.

Some Formalizations of Arithmetic in Coq

- Integers as lists of bits:
  polynomial equality, semi-decision of $(\mathbb{Z}, +, =, <)$.
- Rational numbers and Bernstein polynomials:
  global optimization for Hales' inequalities.
- Dyadic numbers and intervals:
  verification of Gappa certificates.
- Integers as binary trees of machine words:
  verification of Pocklington primality certificates.
- Floating-point numbers and intervals:
  enclosures for expressions of elementary functions.

Some Formalizations of Arithmetic in Coq

- Integers as lists of bits:
  polynomial equality, semi-decision of $(\mathbb{Z}, +, =, <)$.
- Rational numbers and Bernstein polynomials:
  global optimization for Hales' inequalities.
- Dyadic numbers and intervals:
  verification of Gappa certificates.
- Integers as binary trees of machine words:
  verification of Pocklington primality certificates.
- Floating-point numbers and intervals:
  enclosures for expressions of elementary functions.
- Real numbers as streams of integer words.

Enclosures for Expressions of Elementary Functions

Example:

$$\forall x \in [2^{-20}, 1],\ \left|\frac{x \times (1 - 10473 \cdot 2^{-16} \cdot x^2)}{\sin x} - 1\right| \leq 102 \cdot 2^{-16}.$$

Enclosures for Expressions of Elementary Functions

Example:

$$\forall x \in [2^{-20}, 1],\ \left|\frac{x \times (1 - 10473 \cdot 2^{-16} \cdot x^2)}{\sin x} - 1\right| \leq 102 \cdot 2^{-16}.$$

Method: order-1 Taylor interval computations and bisection.

Interval Approaches: Relative Error of a Rounded Sine

Relative error between $\sin x$ and the binary32 Horner evaluation of a degree-3 polynomial for $x \in [2^{-20}, 1]$:

```
Theorem rounded_sine :
  forall x y,
  y = rnd(x * rnd(1 - rnd(rnd(x*x) * (10473/65536)))) ->
  1/1048576 <= x <= 1 ->
  Rabs(y - sin x) <= 103 / 65536 * Rabs(sin x).
Proof.
intros.
set (My := x * (1 - (x*x) * (10473/65536))).
assert (Rabs(My - sin x) <= 102 / 65536 * Rabs(sin x)).
  (* method error *)
  apply helper. admit.
  unfold My.
  abstract interval with
    (i_bisect_diff x, i_depth 40, i_nocheck).
unfold My in H1.
gappa. (* global error *)
Qed.
```

Interval Approaches: Square Root

- Fully computational approach:

$$f([\underline{u},\overline{u}]) = \begin{cases} \left[\triangledown\sqrt{\underline{u}}, \vartriangle\sqrt{\overline{u}}\right] & \text{if } 0 \le u, \\ \perp & \text{otherwise.} \end{cases}$$

Correctness lemma: $\forall x \in [\underline{u},\overline{u}],\ \sqrt{x} \in f([\underline{u},\overline{u}])$.

Interval Approaches: Square Root

- Fully computational approach:

$$f([\underline{u},\overline{u}]) = \begin{cases} \left[\triangledown\sqrt{\underline{u}}, \vartriangle\sqrt{\overline{u}}\right] & \text{if } 0 \le u, \\ \perp & \text{otherwise.} \end{cases}$$

  Correctness lemma: $\forall x \in [\underline{u},\overline{u}],\ \sqrt{x} \in f([\underline{u},\overline{u}])$.

- Oracle-based approach:

$$f([\underline{u},\overline{u}],[\underline{v},\overline{v}]) = 0 \le \overline{v} \land \overline{u} \le \overline{v}^2 \land \begin{cases} 0 \le \underline{u} & \text{if } \underline{v} \le 0 \\ \underline{v}^2 \le \underline{u} & \text{otherwise.} \end{cases}$$

  Correctness lemma:
  $\forall x \in [\underline{u},\overline{u}],\ f([\underline{u},\overline{u}],[\underline{v},\overline{v}]) = \mathit{true} \Rightarrow \sqrt{x} \in [\underline{v},\overline{v}]$.

Computing with (Approximate) Reals: Issues

- Decidability?

Computing with (Approximate) Reals: Issues

- Decidability?
- Semi-decidability?

Decision Procedures for Arithmetic Theories

1. Deductive program verification
2. Computing in a formal system
3. Decision procedures for arithmetic theories
   - Quantifier elimination
   - Theory $(\mathbb{C}, +, \times, =)$
   - Theory $(\mathbb{Q}, +, =, <)$
   - $\forall$-formulas, ideals, and cones
4. Conclusion

Quantifier Elimination

Definition (Quantifier elimination)

A theory T in a first-order language L admits QE if,
for any formula $p \in L$, there is a quantifier-free formula $q \in L$ such that $T \models p \Leftrightarrow q$ and q has no other free variables than p.

Sufficient condition: any formula "$\exists x,\ \alpha_1 \wedge \cdots \wedge \alpha_n$" admits QE.

Property

A formula is decidable in a theory QE if it has no free variables.

Quantifier Elimination

Definition (Quantifier elimination)

A theory T in a first-order language L admits QE if,
for any formula $p \in L$, there is a quantifier-free formula $q \in L$
such that $T \models p \Leftrightarrow q$ and q has no other free variables than p.

Sufficient condition: any formula "$\exists x,\ \alpha_1 \wedge \cdots \wedge \alpha_n$" admits QE.

Property

A formula is decidable in a theory QE if it has no free variables.

Example on $\mathbb{N}$: $\quad \forall x,\ 1 \leq x \Rightarrow \exists y,\ y < x.$

- $\neg(\exists x,\ 1 \leq x \wedge \neg(\exists y,\ y < x))$
- $\neg(\exists x,\ 1 \leq x \wedge \neg(0 < x))$
- $\neg(1 \leq 0)$

Arithmetic Theories and Quantifier Elimination

Decidable theories:

- $(\mathbb{C}, +, \times, =)$ Tarski

Arithmetic Theories and Quantifier Elimination

Decidable theories:

- $(\mathbb{C}, +, \times, =)$ Tarski
- $(\mathbb{R}, +, \times, =, <)$ Collins, Hörmander

Arithmetic Theories and Quantifier Elimination

Decidable theories:

- $(\mathbb{C}, +, \times, =)$ — Tarski
- $(\mathbb{R}, +, \times, =, <)$ — Collins, Hörmander
- $(\mathbb{Q}, +, =, <)$ — Fourier, Motzkin

Arithmetic Theories and Quantifier Elimination

Decidable theories:

- $(\mathbb{C}, +, \times, =)$ — Tarski
- $(\mathbb{R}, +, \times, =, <)$ — Collins, Hörmander
- $(\mathbb{Q}, +, =, <)$ — Fourier, Motzkin
- $(\mathbb{Z}, +, =, <)$ — Presburger, Cooper

Arithmetic Theories and Quantifier Elimination

Decidable theories:

- $(\mathbb{C}, +, \times, =)$ — Tarski
- $(\mathbb{R}, +, \times, =, <)$ — Collins, Hörmander
- $(\mathbb{Q}, +, =, <)$ — Fourier, Motzkin
- $(\mathbb{Z}, +, =, <)$ — Presburger, Cooper
- $(\mathbb{Q}, +, \lfloor\cdot\rfloor, =, <)$ — Weispfenning

Arithmetic Theories and Quantifier Elimination

Decidable theories:

- $(\mathbb{C}, +, \times, =)$ Tarski
- $(\mathbb{R}, +, \times, =, <)$ Collins, Hörmander
- $(\mathbb{Q}, +, =, <)$ Fourier, Motzkin
- $(\mathbb{Z}, +, =, <)$ Presburger, Cooper
- $(\mathbb{Q}, +, \lfloor\cdot\rfloor, =, <)$ Weispfenning

Undecidable theory:

- $(\mathbb{Z}, +, \times, =, <)$ Tarski, Gödel

Theory $(\mathbb{C}, +, \times, =)$

Given $\exists x,\ p_1(x) = 0 \land \cdots \land p_m(x) = 0 \land q_1(x) \neq 0 \land \cdots \land q_n(x) \neq 0.$

Theory $(\mathbb{C}, +, \times, =)$

Given $\exists x,\ p_1(x) = 0 \land \cdots \land p_m(x) = 0 \land q_1(x) \neq 0 \land \cdots \land q_n(x) \neq 0.$

1. Reducing to $\exists x,\ P(x) = 0 \land Q(x) \neq 0$:
   - $q_1(x) \neq 0 \land \cdots \land q_n(x) \neq 0 \Leftrightarrow q_1(x) \times \cdots \times q_n(x) \neq 0.$

Theory $(\mathbb{C}, +, \times, =)$

Given $\exists x,\ p_1(x) = 0 \land \cdots \land p_m(x) = 0 \land q_1(x) \neq 0 \land \cdots \land q_n(x) \neq 0.$

1. Reducing to $\exists x,\ P(x) = 0 \land Q(x) \neq 0$:
   - $q_1(x) \neq 0 \land \cdots \land q_n(x) \neq 0 \Leftrightarrow q_1(x) \times \cdots \times q_n(x) \neq 0.$
   - $c^k \times p_i(x) = p_j(x) \times q(x) + r(x)$, so

     $$p_i(x) = 0 \land p_j(x) = 0 \Leftrightarrow \begin{cases} r(x) = 0 \land p_j(x) = 0 & \text{if } c \neq 0 \\ p_i(x) = 0 \land p_j^*(x) = 0 & \text{if } c = 0 \end{cases}$$

Theory $(\mathbb{C}, +, \times, =)$

Given $\exists x,\ p_1(x) = 0 \wedge \cdots \wedge p_m(x) = 0 \wedge q_1(x) \neq 0 \wedge \cdots \wedge q_n(x) \neq 0.$

1. Reducing to $\exists x,\ P(x) = 0 \wedge Q(x) \neq 0$:
   - $q_1(x) \neq 0 \wedge \cdots \wedge q_n(x) \neq 0 \Leftrightarrow q_1(x) \times \cdots \times q_n(x) \neq 0.$
   - $c^k \times p_i(x) = p_j(x) \times q(x) + r(x)$, so
     $$p_i(x) = 0 \wedge p_j(x) = 0 \Leftrightarrow \begin{cases} r(x) = 0 \wedge p_j(x) = 0 & \text{if } c \neq 0 \\ p_i(x) = 0 \wedge p_j^*(x) = 0 & \text{if } c = 0 \end{cases}$$
2. Cases:
   - $(\exists x,\ Q(x) \neq 0) \Leftrightarrow \neg$(coefs of Q are zero).
   - $(\exists x,\ P(x) = 0) \Leftrightarrow \neg(\ldots)$

Theory $(\mathbb{C}, +, \times, =)$

Given $\exists x,\ p_1(x) = 0 \wedge \cdots \wedge p_m(x) = 0 \wedge q_1(x) \neq 0 \wedge \cdots \wedge q_n(x) \neq 0$.

1. Reducing to $\exists x,\ P(x) = 0 \wedge Q(x) \neq 0$:
   - $q_1(x) \neq 0 \wedge \cdots \wedge q_n(x) \neq 0 \Leftrightarrow q_1(x) \times \cdots \times q_n(x) \neq 0$.
   - $c^k \times p_i(x) = p_j(x) \times q(x) + r(x)$, so
     $$p_i(x) = 0 \wedge p_j(x) = 0 \Leftrightarrow \begin{cases} r(x) = 0 \wedge p_j(x) = 0 & \text{if } c \neq 0 \\ p_i(x) = 0 \wedge p_j^*(x) = 0 & \text{if } c = 0 \end{cases}$$
2. Cases:
   - $(\exists x,\ Q(x) \neq 0) \Leftrightarrow \neg(\text{coefs of } Q \text{ are zero})$.
   - $(\exists x,\ P(x) = 0) \Leftrightarrow \neg(\ldots)$
   - $(\exists x,\ P(x) \neq 0 \Rightarrow Q(x) \neq 0) \Leftrightarrow \neg(P|_x Q^n)$.

Theory $(\mathbb{Q}, +, =, <)$

Quantifier elimination of linear constraints:

- $(\exists x,\ x = \vec{a} \cdot \vec{y} \wedge P[x, \vec{y}]) \Leftrightarrow P[\vec{a} \cdot \vec{y}, \vec{y}]$.

Theory $(\mathbb{Q}, +, =, <)$

Quantifier elimination of linear constraints:

- $(\exists x,\ x = \vec{a} \cdot \vec{y} \wedge P[x, \vec{y}]) \Leftrightarrow P[\vec{a} \cdot \vec{y}, \vec{y}]$.
- $(\exists x,\ \bigwedge_i x < \vec{a}_i \cdot \vec{y} \wedge \bigwedge_j x > \vec{b}_j \cdot \vec{y}) \Leftrightarrow \bigwedge_{i,j} 0 < (\vec{a}_i - \vec{b}_j) \cdot \vec{y}$.

Theory $(\mathbb{Q}, +, =, <)$

Quantifier elimination of linear constraints:

- $(\exists x,\ x = \vec{a} \cdot \vec{y} \wedge P[x, \vec{y}]) \Leftrightarrow P[\vec{a} \cdot \vec{y}, \vec{y}]$.
- $(\exists x,\ \bigwedge_i x < \vec{a}_i \cdot \vec{y} \wedge \bigwedge_j x > \vec{b}_j \cdot \vec{y}) \Leftrightarrow \bigwedge_{i,j} 0 < (\vec{a}_i - \vec{b}_j) \cdot \vec{y}$.

Special case: closed $\exists$-formulas of conjunctions.
Methods: simplex, interior point.

$\forall$-Formulas, Ideals, and Cones

- On $\mathbb{C}$: $\forall \vec{x},\ \bigvee_i p_i(\vec{x}) \neq 0 \vee \bigvee_j q_j(\vec{x}) = 0.$ $\qquad (F)$

$\forall$-Formulas, Ideals, and Cones

- On $\mathbb{C}$: $\forall \vec{x},\ \bigvee_i p_i(\vec{x}) \neq 0 \vee \bigvee_j q_j(\vec{x}) = 0.$ $\quad (F)$

  $$F \Leftrightarrow \forall \vec{x}\vec{z},\ \neg \left(\bigwedge_i p_i(\vec{x}) = 0 \wedge \bigwedge_j z_j \times q_j(\vec{x}) - 1 = 0 \right)$$

$\forall$-Formulas, Ideals, and Cones

- On $\mathbb{C}$: $\forall \vec{x},\ \bigvee_i p_i(\vec{x}) \neq 0 \vee \bigvee_j q_j(\vec{x}) = 0.$ (F)

  $$F \Leftrightarrow \forall \vec{x}\vec{z},\ \neg \left(\bigwedge_i p_i(\vec{x}) = 0 \wedge \bigwedge_j z_j \times q_j(\vec{x}) - 1 = 0 \right)$$
  $$\Leftrightarrow 1 \in \mathrm{Ideal}\left(\cdots, p_i, \cdots, z_j \times q_j - 1, \cdots\right).$$

∀-Formulas, Ideals, and Cones

- On $\mathbb{C}$: $\forall \vec{x},\ \bigvee_i p_i(\vec{x}) \neq 0 \vee \bigvee_j q_j(\vec{x}) = 0.$ $\quad (F)$

  $$F \Leftrightarrow \forall \vec{x}\vec{z},\ \neg \left(\bigwedge_i p_i(\vec{x}) = 0 \wedge \bigwedge_j z_j \times q_j(\vec{x}) - 1 = 0 \right)$$
  $$\Leftrightarrow 1 \in \mathrm{Ideal}(\cdots, p_i, \cdots, z_j \times q_j - 1, \cdots).$$

- On $\mathbb{R}$: $\forall \vec{x},\ \neg \left(\bigwedge_i p_i(\vec{x}) = 0 \wedge \bigwedge_j q_j(\vec{x}) \geq 0 \right).$ $\quad (F)$

∀-Formulas, Ideals, and Cones

- On $\mathbb{C}$: $\forall \vec{x},\ \bigvee_i p_i(\vec{x}) \neq 0 \vee \bigvee_j q_j(\vec{x}) = 0.$ $\quad (F)$

  $$F \Leftrightarrow \forall \vec{x}\vec{z},\ \neg\left(\bigwedge_i p_i(\vec{x}) = 0 \wedge \bigwedge_j z_j \times q_j(\vec{x}) - 1 = 0\right)$$
  $$\Leftrightarrow 1 \in \mathrm{Ideal}(\cdots, p_i, \cdots, z_j \times q_j - 1, \cdots).$$

- On $\mathbb{R}$: $\forall \vec{x},\ \neg\left(\bigwedge_i p_i(\vec{x}) = 0 \wedge \bigwedge_j q_j(\vec{x}) \geq 0\right).$ $\quad (F)$

  $$F \Leftrightarrow -1 \in \mathrm{Ideal}(p_1, \cdots, p_m) + \mathrm{Cone}(q_1, \cdots, q_n).$$

$\forall$-Formulas, Ideals, and Cones

- On $\mathbb{C}$: $\forall \vec{x},\ \bigvee_i p_i(\vec{x}) \neq 0 \vee \bigvee_j q_j(\vec{x}) = 0.$ $\quad (F)$

  $$F \Leftrightarrow \forall \vec{x}\vec{z},\ \neg\left(\bigwedge_i p_i(\vec{x}) = 0 \wedge \bigwedge_j z_j \times q_j(\vec{x}) - 1 = 0\right)$$
  $$\Leftrightarrow 1 \in \mathrm{Ideal}(\cdots, p_i, \cdots, z_j \times q_j - 1, \cdots).$$

- On $\mathbb{R}$: $\forall \vec{x},\ \neg\left(\bigwedge_i p_i(\vec{x}) = 0 \wedge \bigwedge_j q_j(\vec{x}) \geq 0\right).$ $\quad (F)$

  $$F \Leftrightarrow -1 \in \mathrm{Ideal}(p_1, \cdots, p_m) + \mathrm{Cone}(q_1, \cdots, q_n).$$

Methods: Gröbner bases, semi-definite programming, ...

Suitable for oracles: verifying ideal membership ($\Leftarrow$) is just a single polynomial equality.

Conclusion

- Deductive verification allows to certify arbitrary programs.

  But proof obligations lack structure,
  making it difficult for automated provers.

Conclusion

- Deductive verification allows to certify arbitrary programs.

  But proof obligations lack structure,
  making it difficult for automated provers.

- Numerical computations are not incompatible
  with formal systems.

  They can be used to prove mathematical theorems.

Conclusion

- Deductive verification allows to certify arbitrary programs.

  But proof obligations lack structure,
  making it difficult for automated provers.

- Numerical computations are not incompatible
  with formal systems.

  They can be used to prove mathematical theorems.

- There are powerful but slow methods for proving
  some large sets of proof obligations.

  Oracle-based approaches can dramatically increase
  performances on specific subsets.

Questions?