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

1. Deductive program verification
2. Computing in a formal system
3. Decision procedures for arithmetic theories
4. Conclusion
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)

\[
\{\text{precondition}\} \quad \text{code} \quad \{\text{postcondition}\}.
\]

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)

\[ \{ \text{precondition} \} \quad \text{code} \quad \{ \text{postcondition} \} . \]

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

Note: the definition assumes the code terminates. \nIf it does not, any postcondition holds, including False.
Hoare Triple

1 \{ x \geq 0 \}
2 y = \text{floor}(\sqrt{x})
3 \{ y \geq 0 \text{ and } y \times y \leq 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 programming languages (C, Java) to the ML language.
Environment

annotated Java/JML prog.  annotated C program

ML program  Krakatoa  Caduceus  Frama-C  Jessie

Why

Interactive provers
Coq  PVS
Isabelle  Mizar
HOL4  HOL light

Automated provers
Alt-Ergo  Simplify
SMT-lib (Yices, Z3, CVC3)
Harvey  Zenon  Gappa
Toy Example: Cosine Around Zero

```c
/*@ 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

```java
/*@ 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;
}
```
Verifying Arithmetic Properties

Kind of properties:

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

- **Accuracy** of results:
  - absolute error: $\forall \vec{x}, \ f(\vec{x}) - g(\vec{x}) \in E$;
  - relative error: $\forall \vec{x}, \ \exists \epsilon, \ f(\vec{x}) = g(\vec{x}) \times (1 + \epsilon)$.
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., $\lfloor \cdot \rfloor$).
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:
  \[ u \in [u, \bar{u}] \land v \in [v, \bar{v}] \Rightarrow u + v \in [u + v, \bar{u} + \bar{v}]. \]
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:
  \[ u \in [u, \bar{u}] \land v \in [v, \bar{v}] \Rightarrow u + v \in [u + v, \bar{u} + \bar{v}] \].

- 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:**
  \[ u \in [u, \bar{u}] \land v \in [v, \bar{v}] \Rightarrow u + v \in [u + v, \bar{u} + \bar{v}] \].

- **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

1. Deductive program verification

2. Computing in a formal system
   - Type theory and proofs by reflection
   - Some formalizations of arithmetic in Coq

3. Decision procedures for arithmetic theories

4. Conclusion
Example: Peano’s Arithmetic

Inductive definition of natural numbers:

\text{type} \; \text{nat} = \text{O} \mid \text{S} \; \text{of} \; \text{nat} \quad (\ast \; 5 = \text{SSSSSO} \; \ast)

Axioms for addition:

\text{addO:} \quad \forall b, \; \text{O} + b = b

\text{addS:} \quad \forall a \; b, \; (\text{S} \; a) + b = a + (\text{S} \; b)
Example: Peano’s Arithmetic

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

\[
\begin{align*}
9 &= 9 & \text{reflexivity} \\
0 + 9 &= 9 & \text{add0} \\
\vdots & \text{addS} \times 4 \\
4 + 5 &= 9 & \text{add0} \\
4 + (0 + 5) &= 9 & \text{addS} \\
4 + (1 + 4) &= 9 & \text{addS} \\
4 + (2 + 3) &= 9 & \text{addS}
\end{align*}
\]
Introducing Computations into Proofs

Recursive definition of addition:

```ocaml
let rec plus x y =
  match x with
  | 0 -> y
  | S x' -> plus x' (S y)
```

Lemma plus_xlate: \( \forall a\ b, a + b = plus\ a\ b \)
Introducing Computations into Proofs

Recursive definition of addition:

let rec plus x y =
    match x with
    | 0 -> y
    | S x' -> plus x' (S y)

Lemma plus_xlate: \( \forall a \ b, \ a + b = \text{plus} \ a \ b \)

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

\[
\begin{align*}
9 &= 9 & \text{reflexivity} \\
\text{plus} \ 4 \ (\text{plus} \ 2 \ 3) &= 9 \\
4 + (\text{plus} \ 2 \ 3) &= 9 & \text{plus_xlate} \\
4 + (2 + 3) &= 9 & \text{plus_xlate}
\end{align*}
\]
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.

\[
\frac{p : A}{p : B} \quad A \equiv_\beta B
\]
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.

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

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

(4 steps)

\[
\begin{align*}
\frac{p : 9 = 9}{p : \text{plus} \ 4 \ (\text{plus} \ 2 \ 3) = 9} & \quad \text{reflexivity} \\
\frac{p : \text{plus} \ 4 \ (\text{plus} \ 2 \ 3) = 9}{4 + (\text{plus} \ 2 \ 3) = 9} & \quad \beta\text{-reduction} \\
\frac{4 + (2 + 3) = 9}{4 + (2 + 3) = 9} & \quad \text{plus_xlate}
\end{align*}
\]
Encoding Expressions

**Inductive definition** of expressions on natural numbers:

```ocaml
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 $4 + (2 + 3) = 9$:

\[
\begin{align*}
\text{interp\_expr } (\text{Add } (\text{Nat } 4, \text{Add } (\text{Nat } 2, \text{Nat } 3))) &= 9 \\
4 + (2 + 3) &= 9
\end{align*}
\]

$\beta$-reduction
Evaluating Expressions

**Evaluating** expressions on natural numbers:

```ocaml
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:

\[
\text{let rec eval_expr e =} \\
\text{match e with} \\
\text{| Nat n -> n} \\
\text{| Add (x, y) ->} \\
\text{plus (eval_expr x) (eval_expr y)}
\]

Lemma expr_xlate: \( \forall e \) interp_expr e = eval_expr e

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

\[
\begin{align*}
9 & = 9 \quad \text{reflexivity} \\
\text{eval_expr (Add (Nat 4, ...))} & = 9 \\
\text{interp_expr (Add (Nat 4, ...))} & = 9 \\
4 + (2 + 3) & = 9
\end{align*}
\]

\( \beta \)-reduction

expr_xlate

\( \beta \)-reduction
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: \( \forall a \forall b \quad le a b = true \iff a \leq b \)
Encoding Comparisons

Inductive definition of relations on natural expressions:

```ocaml
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:
true = reflexivity
 eval prop (Le (Add ..., Add ...)) = true
 \(\beta\)-reduction
 interp prop (Le (Add ..., Add ...)) prop xlate 4 + (2 + 3) \(\leq\) 5 + 6
 \(\beta\)-reduction

Guillaume Melquiond
Numerical Computations and Formal Methods
Inductive definition of relations on natural expressions:

```ocaml
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 $4 + (2 + 3) \leq 5 + 6$:

\[
\begin{align*}
\text{true} = \text{true} & \quad \text{reflexivity} \\
\text{eval} \text{prop (Le (Add ..., Add ...))} = \text{true} & \quad \text{β-reduction prop_quad}
\end{align*}
\]

\[
\begin{align*}
\text{interp} \text{prop (Le (Add ..., Add ...))} & \quad \text{β-reduction} \\
4 + (2 + 3) \leq 5 + 6 & \quad \text{prop_quad}
\end{align*}
\]
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.
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:

\[
\forall x \in [2^{-20}, 1], \quad \left| \frac{x \times (1 - 10473 \cdot 2^{-16} \cdot x^2)}{\sin x} - 1 \right| \leq 102 \cdot 2^{-16}.
\]
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.
Relative error between \( \sin x \) and the \( \text{binary32} \) Horner evaluation of a degree-3 polynomial for \( x \in [2^{-20}, 1] \):

Theorem rounded_sine :

\[
\forall x, y, \quad y = \text{rnd}(x * \text{rnd}(1 - \text{rnd}(\text{rnd}(x*x) * (10473/65536)))) \rightarrow \\
1/1048576 \leq x \leq 1 \rightarrow \\
\text{Rabs}(y - \sin x) \leq 103 / 65536 * \text{Rabs}(\sin x).
\]

Proof.

intros.

set (My := x * (1 - (x*x) * (10473/65536))).

assert (Rabs(My - \sin x) \leq 102 / 65536 * \text{Rabs}(\sin x)).

(* \text{method error} *)

apply helper. admit.

unfold My.

abstract interval with

(i_bisect_diff x, i_depth 40, i_nocheck).

unfold My in H1.

gappa. (* \text{global error} *)

Qed.
Interval Approaches: Square Root

- Fully computational approach:

\[ f([u, \bar{u}]) = \begin{cases} \left[ \nabla \sqrt{u}, \Delta \sqrt{u} \right] & \text{if } 0 \leq u, \\ \bot & \text{otherwise.} \end{cases} \]

Correctness lemma: \( \forall x \in [u, \bar{u}], \sqrt{x} \in f([u, \bar{u}]). \)
Interval Approaches: Square Root

- **Fully computational** approach:
  \[ f([u, \overline{u}]) = \begin{cases} \left[ \nabla \sqrt{u}, \triangle \sqrt{u} \right] & \text{if } 0 \leq u, \\ \bot & \text{otherwise.} \end{cases} \]

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

- **Oracle-based** approach:
  \[ f([u, \overline{u}], [v, \overline{v}]) = 0 \leq \overline{v} \land u \leq \overline{v}^2 \land \begin{cases} 0 \leq u & \text{if } v \leq 0 \\ \overline{v}^2 \leq u & \text{otherwise.} \end{cases} \]

  Correctness lemma:
  \( \forall x \in [u, \overline{u}], f([u, \overline{u}], [v, \overline{v}]) = true \Rightarrow \sqrt{x} \in [v, \overline{v}]. \)
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 $T \models p \iff q$ and $q$ has no other free variables than $p$.

**Sufficient condition:** any formula “$\exists x, \alpha_1 \land \cdots \land \alpha_n$” 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 $T \models p \iff q$ and $q$ has no other free variables than $p$.

**Sufficient condition:** any formula “$\exists x, \; \alpha_1 \land \cdots \land \alpha_n$” admits QE.

**Property**

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

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

$\neg (\exists x, \; 1 \leq x \land \neg (\exists y, \; y < x))$

$\neg (\exists x, \; 1 \leq x \land \neg (0 < x))$

$\neg (1 \leq 0)$
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:

- \((\mathbb{C}, +, \times, =)\)
- \((\mathbb{R}, +, \times, =, <)\)
- \((\mathbb{Q}, +, =, <)\)
- \((\mathbb{Z}, +, =, <)\)
- \((\mathbb{Q}, +, \lfloor \cdot \rfloor, =, <)\)

Tarski
Collins, Hörmander
Fourier, Motzkin
Presburger, Cooper
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
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 $(\mathbb{C}, +, \times, =)$
Theory \((\mathbb{C}, +, \times, =)\)

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

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

Reducing to \(\exists x, \ P(x) = 0 \land Q(x) \neq 0:\)

- \(q_1(x) \neq 0 \land \cdots \land q_n(x) \neq 0 \iff q_1(x) \times \cdots \times q_n(x) \neq 0.\)

- \(c^k \times p_i(x) = p_j(x) \times q(x) + r(x), \text{ so}\)

\[ p_i(x) = 0 \land p_j(x) = 0 \iff \begin{cases} r(x) = 0 \land p_j(x) = 0 & \text{if } c \neq 0 \\ p_i(x) = 0 \land p^*_j(x) = 0 & \text{if } c = 0 \end{cases} \]
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
     \[
     p_i(x) = 0 \land p_j(x) = 0 \Leftrightarrow \begin{cases} 
     r(x) = 0 \land p_j(x) = 0 & \text{if } c \neq 0 \\
     p_i(x) = 0 \land p_j^*(x) = 0 & \text{if } c = 0
     \end{cases}
     \]

2. Cases:
   - \( (\exists x, \ Q(x) \neq 0) \Leftrightarrow \lnot(\text{coefs of } Q \text{ are zero}). \)
   - \( (\exists x, \ P(x) = 0) \Leftrightarrow \lnot(\ldots) \)
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 \(\exists x, \ P(x) = 0 \land Q(x) \neq 0\):
   - \(q_1(x) \neq 0 \land \cdots \land q_n(x) \neq 0 \iff q_1(x) \times \cdots \times q_n(x) \neq 0\).
   - \(c^k \times p_i(x) = p_j(x) \times q(x) + r(x),\) so
     \[p_i(x) = 0 \land p_j(x) = 0 \iff \begin{cases} r(x) = 0 \land p_j(x) = 0 \quad \text{if } c \neq 0 \\ p_i(x) = 0 \land p_j^*(x) = 0 \quad \text{if } c = 0 \end{cases}\]

2. Cases:
   - \((\exists x, \ Q(x) \neq 0) \iff \neg(\text{coefs of } Q \text{ are zero}).\)
   - \((\exists x, \ P(x) = 0) \iff \neg(\ldots)\)
   - \((\exists x, \ P(x) \neq 0 \Rightarrow Q(x) \neq 0) \iff \neg(\forall x Qn).\)
Theory \((\mathbb{Q}, +, =, <)\)

Quantifier elimination of linear constraints:

\[
(\exists x, \ x = \bar{a} \cdot \bar{y} \land P[x, \bar{y}]) \iff P[\bar{a} \cdot \bar{y}, \bar{y}].
\]
Theory \(( \mathbb{Q}, +, =, < )\)

Quantifier elimination of linear constraints:

- \((\exists x, \ x = \bar{a} \cdot \bar{y} \land P[x, \bar{y}]) \iff P[\bar{a} \cdot \bar{y}, \bar{y}]\).

- \((\exists x, \ \land_i x < \bar{a}_i \cdot \bar{y} \land \land_j x > \bar{b}_j \cdot \bar{y}) \iff \land_{i,j} 0 < (\bar{a}_i - \bar{b}_j) \cdot \bar{y} \).

Special case: closed \(\exists\)-formulas of conjunctions.

Methods: simplex, interior point.
Theory \((\mathbb{Q}, +, =, <)\)

Quantifier elimination of linear constraints:
- \((\exists x, \ x = \bar{a} \cdot \bar{y} \land P[x, \bar{y}]) \iff P[\bar{a} \cdot \bar{y}, \bar{y}].\)
- \((\exists x, \ \land_i x < \bar{a}_i \cdot \bar{y} \land \land_j x > \bar{b}_j \cdot \bar{y}) \iff \land_{i,j} 0 < (\bar{a}_i - \bar{b}_j) \cdot \bar{y}.\)

Special case: closed \(\exists\)-formulas of conjunctions.
Methods: simplex, interior point.
∀-Formulas, Ideals, and Cones

- On $\mathbb{C}$:
  \[ \forall x, \bigvee_i p_i(x) \neq 0 \lor \bigvee_j q_j(x) = 0. \]  
  \( (F) \)
\( \forall \)-Formulas, Ideals, and Cones

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

\[ F \iff \forall \vec{x} \vec{z}, \neg \left( \bigwedge_i p_i(\vec{x}) = 0 \land \bigwedge_j z_j \times q_j(\vec{x}) - 1 = 0 \right) \]
∀-Formulas, Ideals, and Cones

On \( \mathbb{C} \): \( \forall \vec{x}, \bigvee_i p_i(\vec{x}) \neq 0 \lor \bigvee_j q_j(\vec{x}) = 0. \) \hspace{1cm} (F)

\[
F \iff \forall \vec{x} \vec{z}, \neg \left( \bigwedge_i p_i(\vec{x}) = 0 \land \bigwedge_j z_j \times q_j(\vec{x}) - 1 = 0 \right)
\]
\[
\Leftrightarrow 1 \in \text{Ideal}(\cdots, p_i, \cdots, z_j \times q_j - 1, \cdots).
\]
∀-Formulas, Ideals, and Cones

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

\[
F \iff \forall \vec{x} \vec{z}, \neg \left( \bigwedge_i p_i(\vec{x}) = 0 \land \bigwedge_j z_j \times q_j(\vec{x}) - 1 = 0 \right) \iff 1 \in \text{Ideal}(\cdots, p_i, \cdots, z_j \times q - 1, \cdots).
\]

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

\[
\]
∀-Formulas, Ideals, and Cones

- On $\mathbb{C}$: $\forall x, \bigvee_i p_i(x) \neq 0 \lor \bigvee_j q_j(x) = 0$. \hspace{1cm} (F)

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

  $\iff 1 \in \text{Ideal}(\cdots, p_i, \cdots, z_j \times q_j - 1, \cdots)$.

- On $\mathbb{R}$: $\forall x, \neg \left( \bigwedge_i p_i(x) = 0 \land \bigwedge_j q_j(x) \geq 0 \right)$. \hspace{1cm} (F)

  $F \iff -1 \in \text{Ideal}(p_1, \cdots, p_m) + \text{Cone}(q_1, \cdots, q_n)$. 
∀-Formulas, Ideals, and Cones

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

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

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

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

Methods: Gröbner bases, semi-definite programming, \ldots

Suitable for oracles: verifying ideal membership ($\iff$) 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?