Formal proofs and certified computation in Coq

Érik Martin-Dorel

http://erik.martin-dorel.org

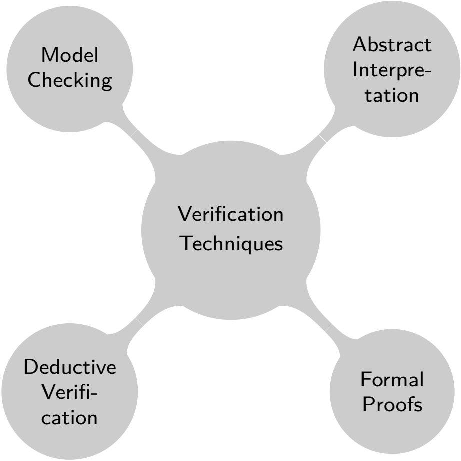
Équipe ACADIE, Laboratoire IRIT
Université Toulouse III - Paul Sabatier

French Symposium on Games
26–30 May 2015
Université Paris Diderot

ACADIE

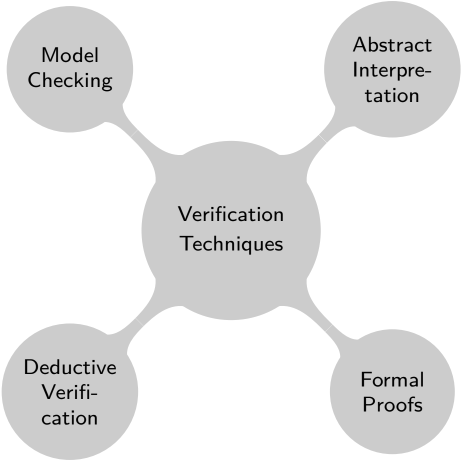
Formal Methods

- Gather
  - a set of mathematically-based techniques
  - designed to specify and verify computer systems



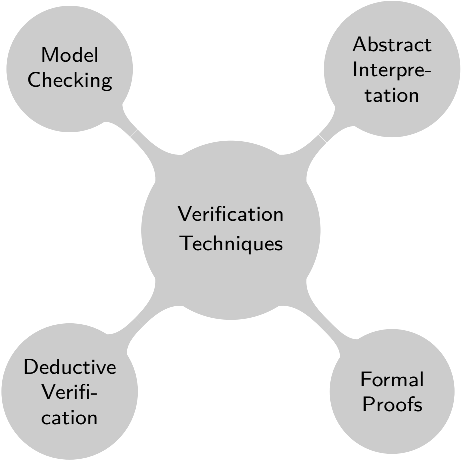
Formal Methods

- Gather
  - a set of mathematically-based techniques
  - designed to specify and verify computer systems
- Used in areas where
  - errors can cause loss of life
  - errors can cause significant financial damage



Formal Methods

- Gather
  - a set of mathematically-based techniques
  - designed to specify and verify computer systems
- Used in areas where
  - errors can cause loss of life
  - errors can cause significant financial damage
- For instance
  - for the Paris Métro Line 14
  - at Intel, AMD, ...



Formal Proofs

- needs a proof assistant (= proof checker (= theorem prover))
  - specify algorithms and theorems
  - develop proofs interactively
  - check proofs
  - but also perform computations, develop automatic tactics...
- various tools: ACL2, Agda, Coq, HOL Light, Isabelle, Mizar, PVS...

Formal Proofs

- needs a proof assistant (= proof checker (= theorem prover))
  - specify algorithms and theorems
  - develop proofs interactively
  - check proofs
  - but also perform computations, develop automatic tactics...
- various tools: ACL2, Agda, Coq, HOL Light, Isabelle, Mizar, PVS...
- main criteria to classify them:
  - the kind of underlying logic (FOL/HOL, classical/intuitionistic...)
  - the presence of a proof kernel (De Bruijn's criterion)
  - the degree of automation
  - the availability of large libraries of formalized results
- see also [Freek Wiedijk (2006): The Seventeen Provers of the World]

How to Believe a Machine-Checked Proof [R. Pollack 1998]

How to Believe a Machine-Checked Proof [R. Pollack 1998]

Two sub-problems:

1. decide if the putative formal proof is really a derivation in the given formal system

How to Believe a Machine-Checked Proof [R. Pollack 1998]

Two sub-problems:

1. decide if the putative formal proof is really a derivation in the given formal system

2. decide if what it proves really has the informal meaning claimed for it

How to Believe a Machine-Checked Proof [R. Pollack 1998]

Two sub-problems:

1. decide if the putative formal proof is really a derivation in the given formal system
   - → this question can be answered by a machine
   - → need to trust the hardware, the OS... and the proof checker (but it is a simple program: it just need to check the proof, not to "discover" it !)
2. decide if what it proves really has the informal meaning claimed for it

How to Believe a Machine-Checked Proof [R. Pollack 1998]

Two sub-problems:

1. decide if the putative formal proof is really a derivation in the given formal system
   - → this question can be answered by a machine
   - → need to trust the hardware, the OS... and the proof checker (but it is a simple program: it just need to check the proof, not to "discover" it !)
2. decide if what it proves really has the informal meaning claimed for it
   - → this is an informal question
   - → well surveyable: check that the formalized definitions indeed correspond to the usual mathematical ones
     (no need to dive into proof details: they're fully handled by the checker)

ACADIE

Focus on the Coq proof assistant

- Written in OCaml

Focus on the Coq proof assistant

- Written in OCaml
- Initiated by Thierry Coquand and Gérard Huet, and developed by Inria since 1984 (the latest stable release being version 8.4pl6)

Focus on the Coq proof assistant

- Written in OCaml
- Initiated by Thierry Coquand and Gérard Huet, and developed by Inria since 1984 (the latest stable release being version 8.4pl6)
- Provides a strongly-typed functional programming language and proof framework, based on the *Calculus of Inductive Constructions*, a higher-order logic that is constructive (= intuitionistic) and very expressive

Focus on the Coq proof assistant

- Written in OCaml
- Initiated by Thierry Coquand and Gérard Huet, and developed by Inria since 1984 (the latest stable release being version 8.4pl6)
- Provides a strongly-typed functional programming language and proof framework, based on the *Calculus of Inductive Constructions*, a higher-order logic that is constructive (= intuitionistic) and very expressive
- [Yves Bertot, Pierre Castéran (2004): Coq'Art: The Calculus of Inductive Constructions]

Focus on the Coq proof assistant

- Written in OCaml
- Initiated by Thierry Coquand and Gérard Huet, and developed by Inria since 1984 (the latest stable release being version 8.4pl6)
- Provides a strongly-typed functional programming language and proof framework, based on the *Calculus of Inductive Constructions*, a higher-order logic that is constructive (= intuitionistic) and very expressive
- [Yves Bertot, Pierre Castéran (2004): Coq'Art: The Calculus of Inductive Constructions]
- Coq has been awarded the 2013 ACM Software System Award, and the 2013 SIGPLAN Programming Languages Software Award.

Recap the role of Coq's kernel

The Curry–Howard correspondence

A proposition is . . . . a type

Recap the role of Coq's kernel

The Curry–Howard correspondence

A proposition is . . . . a type

A proof of a proposition is . . . a program that inhabits this type

Recap the role of Coq's kernel

The Curry–Howard correspondence

A proposition is . . . . a type
A proof of a proposition is . . . a program that inhabits this type
A false proposition is . . . . an empty type

Recap the role of Coq's kernel

The Curry–Howard correspondence

A proposition is	a type
A proof of a proposition is ...	a program that inhabits this type
A false proposition is	an empty type
A proof of P *implies* Q is ...	a program p turning any proof of P into a proof of Q; denoted by $p : P \to Q$

Recap the role of Coq's kernel

The Curry–Howard correspondence

A proposition is	a type
A proof of a proposition is ...	a program that inhabits this type
A false proposition is	an empty type
A proof of P *implies* Q is ...	a program p turning any proof of P into a proof of Q; denoted by $p : P \to Q$

Checking that p is a proof of a theorem T (in a proof environment E) amounts to calculating the type of p (w.r.t. E) and comparing it with T.

Recap the role of Coq's kernel

The Curry–Howard correspondence

A proposition is	a type
A proof of a proposition is . . .	a program that inhabits this type
A false proposition is	an empty type
A proof of P *implies* Q is . . .	a program p turning any proof of P into a proof of Q; denoted by $p : P \to Q$

Checking that p is a proof of a theorem T (in a proof environment E) amounts to calculating the type of p (w.r.t. E) and comparing it with T. We say that it is a type judgement $E \vdash p : T$.

Coq, proofs and computation

Coq comes with a primitive notion of computation, called conversion.

Key feature of Coq's logic: the convertibility rule

In environment E, if $p : A$ and if A and B are convertible, then $p : B$.

Coq, proofs and computation

Coq comes with a primitive notion of computation, called conversion.

Key feature of Coq's logic: the convertibility rule

In environment E, if $p : A$ and if A and B are convertible, then $p : B$.

So roughly speaking, typing is performed "modulo computation".

Coq, proofs and computation

Coq comes with a primitive notion of computation, called conversion.

Key feature of Coq's logic: the convertibility rule

In environment E, if $p : A$ and if A and B are convertible, then $p : B$.

So roughly speaking, typing is performed "modulo computation".

Toy example of proof based on computation

- Assume we want to prove $4 \leqslant 8$, not using the axiomatic approach[a]

[a] i.e. without relying on $\forall n : \mathbb{N},\ n \leqslant n$ and $\forall m, n : \mathbb{N},\ m \leqslant n \Rightarrow m \leqslant n+1$

Coq, proofs and computation

Coq comes with a primitive notion of computation, called conversion.

Key feature of Coq's logic: the convertibility rule

In environment E, if $p : A$ and if A and B are convertible, then $p : B$.

So roughly speaking, typing is performed "modulo computation".

Toy example of proof based on computation

- Assume we want to prove $4 \leqslant 8$, not using the axiomatic approach[a]
- We define $\ominus$ as the subtraction over $\mathbb{N}$, i.e. $m \ominus n := \max(0, m - n)$.

[a] i.e. without relying on $\forall n : \mathbb{N},\ n \leqslant n$ and $\forall m, n : \mathbb{N},\ m \leqslant n \Rightarrow m \leqslant n+1$

Coq, proofs and computation

Coq comes with a primitive notion of computation, called conversion.

Key feature of Coq's logic: the convertibility rule

In environment E, if $p : A$ and if A and B are convertible, then $p : B$.

So roughly speaking, typing is performed "modulo computation".

Toy example of proof based on computation

- Assume we want to prove $4 \leqslant 8$, not using the axiomatic approach[a]
- We define $\ominus$ as the subtraction over $\mathbb{N}$, i.e. $m \ominus n := \max(0, m - n)$.
- We rewrite $4 \leqslant 8$ as $4 \ominus 8 = 0$.

[a] i.e. without relying on $\forall n : \mathbb{N},\ n \leqslant n$ and $\forall m, n : \mathbb{N},\ m \leqslant n \Rightarrow m \leqslant n + 1$

Coq, proofs and computation

Coq comes with a primitive notion of computation, called conversion.

Key feature of Coq's logic: the convertibility rule

In environment E, if $p : A$ and if A and B are convertible, then $p : B$.

So roughly speaking, typing is performed "modulo computation".

Toy example of proof based on computation

- Assume we want to prove $4 \leqslant 8$, not using the axiomatic approach[a]
- We define $\ominus$ as the subtraction over $\mathbb{N}$, i.e. $m \ominus n := \max(0, m - n)$.
- We rewrite $4 \leqslant 8$ as $4 \ominus 8 = 0$.
- We compute and get $0 = 0$, which trivially holds ($\mathrm{refl} : 0 = 0$)

[a] i.e. without relying on $\forall n : \mathbb{N},\ n \leqslant n$ and $\forall m, n : \mathbb{N},\ m \leqslant n \Rightarrow m \leqslant n + 1$

Coq, proofs and computation

Coq comes with a primitive notion of computation, called conversion.

Key feature of Coq's logic: the convertibility rule

In environment E, if $p : A$ and if A and B are convertible, then $p : B$.

So roughly speaking, typing is performed "modulo computation".

Toy example of proof based on computation

- Assume we want to prove $4 \leqslant 8$, not using the axiomatic approach[a]
- We define $\ominus$ as the subtraction over $\mathbb{N}$, i.e. $m \ominus n := \max(0, m - n)$.
- We rewrite $4 \leqslant 8$ as $4 \ominus 8 = 0$.
- We compute and get $0 = 0$, which trivially holds ($\mathrm{refl} : 0 = 0$)
- As $0 = 0$ and $4 \ominus 8 = 0$ are convertible, we also have $\mathrm{refl} : 4 \ominus 8 = 0$, hence the result.

[a] i.e. without relying on $\forall n : \mathbb{N},\ n \leqslant n$ and $\forall m, n : \mathbb{N},\ m \leqslant n \Rightarrow m \leqslant n + 1$

Approaches to certify computation with a Proof Assistant

Borrowing [G. Barthe, G. Ruys, H. Barendregt, 1995]'s terminology

"autarkic approach": perform all calculations inside the proof assistant

"skeptical approach": rely on certificates that are produced by a given tool, external to the proof assistant, then checked

Approaches to certify computation with a Proof Assistant

Borrowing [G. Barthe, G. Ruys, H. Barendregt, 1995]'s terminology

"autarkic approach": perform all calculations inside the proof assistant

"skeptical approach": rely on certificates that are produced by a given tool, external to the proof assistant, then checked

extraction of programs: generate compilable source code (e.g. in OCaml) correct by construction, from the formalized algorithm: e.g., the CompCert C compiler has been designed this way [X. Leroy (2009): A Formally Verified Compiler Back-end].

Approaches to certify computation with a Proof Assistant

Borrowing [G. Barthe, G. Ruys, H. Barendregt, 1995]'s terminology

"autarkic approach": perform all calculations inside the proof assistant

"skeptical approach": rely on certificates that are produced by a given tool, external to the proof assistant, then checked

extraction of programs: generate compilable source code (e.g. in OCaml) correct by construction, from the formalized algorithm: e.g., the CompCert C compiler has been designed this way [X. Leroy (2009): A Formally Verified Compiler Back-end].

deductive verification: annotate the (imperative) program code and use dedicated tools, such as Frama-C/Jessie/Why3, to generate proof obligations (to be discharged by automated provers or proof assistants as back-ends)

Overview of the Reals library (included in Coq's stdlib)

- originated in the Coq formalization of the Three Gap Theorem (Steinhaus' conjecture), cf. [Micaela Mayero's PhD thesis, 2001]

Overview of the Reals library (included in Coq's stdlib)

- originated in the Coq formalization of the Three Gap Theorem (Steinhaus' conjecture), cf. [Micaela Mayero's PhD thesis, 2001]
- classical axiomatization of $\mathbb{R}$ as a complete Archimedean ordered field

Overview of the Reals library (included in Coq's stdlib)

- originated in the Coq formalization of the Three Gap Theorem (Steinhaus' conjecture), cf. [Micaela Mayero's PhD thesis, 2001]
- classical axiomatization of $\mathbb{R}$ as a complete Archimedean ordered field
- the classical flavor of this formalization is due to the trichotomy axiom (named `total_order_T` in the code)

Overview of the Reals library (included in Coq's stdlib)

- originated in the Coq formalization of the Three Gap Theorem (Steinhaus' conjecture), cf. [Micaela Mayero's PhD thesis, 2001]
- classical axiomatization of $\mathbb{R}$ as a complete Archimedean ordered field
- the classical flavor of this formalization is due to the trichotomy axiom (named `total_order_T` in the code)
- part of the Coq standard library

Overview of the Reals library (included in Coq's stdlib)

- originated in the Coq formalization of the Three Gap Theorem (Steinhaus' conjecture), cf. [Micaela Mayero's PhD thesis, 2001]
- classical axiomatization of $\mathbb{R}$ as a complete Archimedean ordered field
- the classical flavor of this formalization is due to the trichotomy axiom (named `total_order_T` in the code)
- part of the Coq standard library
- technicalities: the division is a total function

Overview of the Reals library (included in Coq's stdlib)

- originated in the Coq formalization of the Three Gap Theorem (Steinhaus' conjecture), cf. [Micaela Mayero's PhD thesis, 2001]
- classical axiomatization of $\mathbb{R}$ as a complete Archimedean ordered field
- the classical flavor of this formalization is due to the trichotomy axiom (named `total_order_T` in the code)
- part of the Coq standard library
- technicalities: the division is a total function
- gathers support results on derivability, Riemann integral (both defined with dependent types) and reference functions

Overview of the Coquelicot library

- a new library of real analysis for Coq

Overview of the Coquelicot library

- a new library of real analysis for Coq
- conservative extension of the Reals standard library

Overview of the Coquelicot library

- a new library of real analysis for Coq
- conservative extension of the Reals standard library
- cf. [Sylvie Boldo, Catherine Lelay, Guillaume Melquiond (2015): Coquelicot: A User-Friendly Library of Real Analysis for Coq]

Overview of the Coquelicot library

- a new library of real analysis for Coq
- conservative extension of the Reals standard library
- cf. [Sylvie Boldo, Catherine Lelay, Guillaume Melquiond (2015): Coquelicot: A User-Friendly Library of Real Analysis for Coq]
- new features:

Overview of the Coquelicot library

- a new library of real analysis for Coq
- conservative extension of the Reals standard library
- cf. [Sylvie Boldo, Catherine Lelay, Guillaume Melquiond (2015): Coquelicot: A User-Friendly Library of Real Analysis for Coq]
- new features:
  - user-friendly definitions of limits, derivatives, integrals. . . (with total functions in place of dependent types)

ACADIE

Overview of the Coquelicot library

- a new library of real analysis for Coq
- conservative extension of the Reals standard library
- cf. [Sylvie Boldo, Catherine Lelay, Guillaume Melquiond (2015): Coquelicot: A User-Friendly Library of Real Analysis for Coq]
- new features:
  - user-friendly definitions of limits, derivatives, integrals... (with total functions in place of dependent types)
  - comprehensive set of theorems on these notions, up to power series, parametric integrals, two-dimensional differentiability, asymptotic behaviors

Overview of the Coquelicot library

- a new library of real analysis for Coq
- conservative extension of the Reals standard library
- cf. [Sylvie Boldo, Catherine Lelay, Guillaume Melquiond (2015): Coquelicot: A User-Friendly Library of Real Analysis for Coq]
- new features:
  - user-friendly definitions of limits, derivatives, integrals. . . (with total functions in place of dependent types)
  - comprehensive set of theorems on these notions, up to power series, parametric integrals, two-dimensional differentiability, asymptotic behaviors
  - tactics to automate proofs on derivatives

Overview of the C-CoRN library

- C-CoRN = Constructive Coq Repository at Nijmegen

Overview of the C-CoRN library

- C-CoRN = Constructive Coq Repository at Nijmegen
- originated in the FTA project for formalizing the Fundamental Theorem of Algebra constructively

Overview of the C-CoRN library

- C-CoRN = Constructive Coq Repository at Nijmegen
- originated in the FTA project for formalizing the Fundamental Theorem of Algebra constructively
- intuitionistic axiomatization *via* an algebraic hierarchy built upon constructive setoids + construction of a real number structure *via* Cauchy sequences, cf. [Milad Niqui's PhD thesis, 2004].

Overview of the C-CoRN library

- C-CoRN = Constructive Coq Repository at Nijmegen
- originated in the FTA project for formalizing the Fundamental Theorem of Algebra constructively
- intuitionistic axiomatization *via* an algebraic hierarchy built upon constructive setoids + construction of a real number structure *via* Cauchy sequences, cf. [Milad Niqui's PhD thesis, 2004].
- features:

Overview of the C-CoRN library

- C-CoRN = Constructive Coq Repository at Nijmegen
- originated in the FTA project for formalizing the Fundamental Theorem of Algebra constructively
- intuitionistic axiomatization *via* an algebraic hierarchy built upon constructive setoids + construction of a real number structure *via* Cauchy sequences, cf. [Milad Niqui's PhD thesis, 2004].
- features:
  - large and generic library in the spirit of E. Bishop's constructive analysis

Overview of the C-CoRN library

- C-CoRN = Constructive Coq Repository at Nijmegen
- originated in the FTA project for formalizing the Fundamental Theorem of Algebra constructively
- intuitionistic axiomatization *via* an algebraic hierarchy built upon constructive setoids + construction of a real number structure *via* Cauchy sequences, cf. [Milad Niqui's PhD thesis, 2004].
- features:
  - large and generic library in the spirit of E. Bishop's constructive analysis
  - "computational real numbers" $\rightsquigarrow$ "proof by computation" is possible

Overview of the C-CoRN library

- C-CoRN = Constructive Coq Repository at Nijmegen
- originated in the FTA project for formalizing the Fundamental Theorem of Algebra constructively
- intuitionistic axiomatization *via* an algebraic hierarchy built upon constructive setoids + construction of a real number structure *via* Cauchy sequences, cf. [Milad Niqui's PhD thesis, 2004].
- features:
  - large and generic library in the spirit of E. Bishop's constructive analysis
  - "computational real numbers" $\rightsquigarrow$ "proof by computation" is possible
  - by construction, all functions overs the constructive reals are continuous
    $\rightsquigarrow$ hinders the applicability to proofs in standard/numerical analysis

Overview of the SSReflect/MathComp libraries

- SSReflect was born during the formal verification of the Four Color Theorem by Georges Gonthier collaborating with Benjamin Werner

Overview of the SSReflect/MathComp libraries

- SSReflect was born during the formal verification of the Four Color Theorem by Georges Gonthier collaborating with Benjamin Werner
- SSReflect: extension of the Coq proof language that promotes the Small Scale Reflection: use reflection ($\approx$ proof by computation) whenever possible, even for low-level reasoning

ACADIE

Overview of the SSReflect/MathComp libraries

- SSReflect was born during the formal verification of the Four Color Theorem by Georges Gonthier collaborating with Benjamin Werner
- SSReflect: extension of the Coq proof language that promotes the Small Scale Reflection: use reflection ($\approx$ proof by computation) whenever possible, even for low-level reasoning
- the Mathematical Components project, led by G. Gonthier, culminated in the formalization of the Feit–Thompson theorem in Sept. 2012 (more than 300 textbook pages and a 6-year formalization effort):

```
Theorem Feit_Thompson (gT: finGroupType)(G: {group gT}):
  odd #|G| → solvable G.
```

Overview of the SSReflect/MathComp libraries

- SSReflect was born during the formal verification of the Four Color Theorem by Georges Gonthier collaborating with Benjamin Werner
- SSReflect: extension of the Coq proof language that promotes the Small Scale Reflection: use reflection ($\approx$ proof by computation) whenever possible, even for low-level reasoning
- the Mathematical Components project, led by G. Gonthier, culminated in the formalization of the Feit–Thompson theorem in Sept. 2012 (more than 300 textbook pages and a 6-year formalization effort):

```
Theorem Feit_Thompson (gT: finGroupType)(G: {group gT}):
  odd #|G| → solvable G.
```

- $\leadsto$ MathComp: comprehensive library of algebra, based on SSReflect

Overview of the CoqEAL library

- CoqEAL = the Coq Effective Algebra Library

Overview of the CoqEAL library

- CoqEAL = the Coq Effective Algebra Library
- originated in the ForMath project

Overview of the CoqEAL library

- CoqEAL = the Coq Effective Algebra Library
- originated in the ForMath project
- aim: facilitate the verification of effective symbolic computation algorithms in Coq

Overview of the CoqEAL library

- CoqEAL = the Coq Effective Algebra Library
- originated in the ForMath project
- aim: facilitate the verification of effective symbolic computation algorithms in Coq
- idea: prove a high-level version of the algorithm (e.g. by relying on SSReflect/MathComp) then proceed by refinement

Overview of the CoqEAL library

- CoqEAL = the Coq Effective Algebra Library
- originated in the ForMath project
- aim: facilitate the verification of effective symbolic computation algorithms in Coq
- idea: prove a high-level version of the algorithm (e.g. by relying on SSReflect/MathComp) then proceed by refinement
- CoqEAL has been specifically designed to reduce the "bookkeeping" that occurs in the refinement proofs

Overview of the CoqEAL library

- CoqEAL = the Coq Effective Algebra Library
- originated in the ForMath project
- aim: facilitate the verification of effective symbolic computation algorithms in Coq
- idea: prove a high-level version of the algorithm (e.g. by relying on SSReflect/MathComp) then proceed by refinement
- CoqEAL has been specifically designed to reduce the "bookkeeping" that occurs in the refinement proofs
- [C. Cohen, M. Dénès, A. Mörtberg (2013): Refinements for Free!]

Overview of the CoqInterval library — Issues and methods

- aim: (automatically) prove in Coq that the distance between $f(x)$ and some approximation $P(x)$ is bounded by some $\epsilon > 0$ for all $x \in \boldsymbol{I}$.

ACADIE

Overview of the CoqInterval library — Issues and methods

- aim: (automatically) prove in Coq that the distance between $f(x)$ and some approximation $P(x)$ is bounded by some $\epsilon > 0$ for all $x \in \boldsymbol{I}$.
- [G. Melquiond (2008): Proving bounds on real-valued functions with computations]

Overview of the CoqInterval library — Issues and methods

- aim: (automatically) prove in Coq that the distance between $f(x)$ and some approximation $P(x)$ is bounded by some $\epsilon > 0$ for all $x \in \boldsymbol{I}$.
- [G. Melquiond (2008): Proving bounds on real-valued functions with computations]
- main datatype: intervals with floating-point numbers bounds
  e.g., we'll consider an interval such as $[3.1415, 3.1416]$ in place of π

Overview of the CoqInterval library — Issues and methods

- aim: (automatically) prove in Coq that the distance between $f(x)$ and some approximation $P(x)$ is bounded by some $\epsilon > 0$ for all $x \in \boldsymbol{I}$.
- [G. Melquiond (2008): Proving bounds on real-valued functions with computations]
- main datatype: intervals with floating-point numbers bounds
  e.g., we'll consider an interval such as $[3.1415, 3.1416]$ in place of π
- dependency problem: when a variable occur several times, it typically leads to an overestimation of the range
  e.g., for $f(x) = x \cdot (1 - x)$ and $\boldsymbol{x} = [0, 1]$, we get $\mathrm{eval}_{\mathrm{IA}}(f, \boldsymbol{x}) = [0, 1]$, while the exact range is $f(\boldsymbol{x}) = [0, \frac{1}{4}]$

ACADIE

Overview of the CoqInterval library — Issues and methods

- aim: (automatically) prove in Coq that the distance between $f(x)$ and some approximation $P(x)$ is bounded by some $\epsilon > 0$ for all $x \in \boldsymbol{I}$.
- [G. Melquiond (2008): Proving bounds on real-valued functions with computations]
- main datatype: intervals with floating-point numbers bounds
  e.g., we'll consider an interval such as $[3.1415, 3.1416]$ in place of π
- dependency problem: when a variable occur several times, it typically leads to an overestimation of the range
  e.g., for $f(x) = x \cdot (1 - x)$ and $\boldsymbol{x} = [0, 1]$, we get $\mathrm{eval}_{\mathrm{IA}}(f, \boldsymbol{x}) = [0, 1]$, while the exact range is $f(\boldsymbol{x}) = [0, \frac{1}{4}]$
- solutions: bisection, automatic differentiation. . .

Overview of the CoqInterval library — Issues and methods

- aim: (automatically) prove in Coq that the distance between $f(x)$ and some approximation $P(x)$ is bounded by some $\epsilon > 0$ for all $x \in \boldsymbol{I}$.
- [G. Melquiond (2008): Proving bounds on real-valued functions with computations]
- main datatype: intervals with floating-point numbers bounds
  e.g., we'll consider an interval such as $[3.1415, 3.1416]$ in place of π
- dependency problem: when a variable occur several times, it typically leads to an overestimation of the range
  e.g., for $f(x) = x \cdot (1 - x)$ and $\boldsymbol{x} = [0, 1]$, we get $\mathrm{eval}_{\mathrm{IA}}(f, \boldsymbol{x}) = [0, 1]$, while the exact range is $f(\boldsymbol{x}) = [0, \frac{1}{4}]$
- solutions: bisection, automatic differentiation. . . or Taylor Models:
  [N. Brisebarre, M. Joldeș, EMD, M. Mayero, J-M. Muller, I. Pașca, L. Rideau, and L. Théry (2012): Rigorous Polynomial Approximation Using Taylor Models in Coq]

ACADIE

Overview of the CoqInterval library — Proof example #1

Example taken from [John Harrison (1997): Verifying the Accuracy of Polynomial Approximations in HOL]

```
Require Import Reals Interval_tactic.
Local Open Scope R_scope.
```

`Theorem Harrison97 :` $\forall x : \mathbb{R}, -\frac{10831}{1000000} \leqslant x \leqslant \frac{10831}{1000000} \implies$

$$\left|(e^x - 1) - \left(x + \frac{8388676}{2^{24}}x^2 + \frac{11184876}{2^{26}}x^3\right)\right| \leqslant \frac{23}{27} \times \frac{1}{2^{33}}.$$

Overview of the CoqInterval library — Proof example #1

Example taken from [John Harrison (1997): Verifying the Accuracy of Polynomial Approximations in HOL]

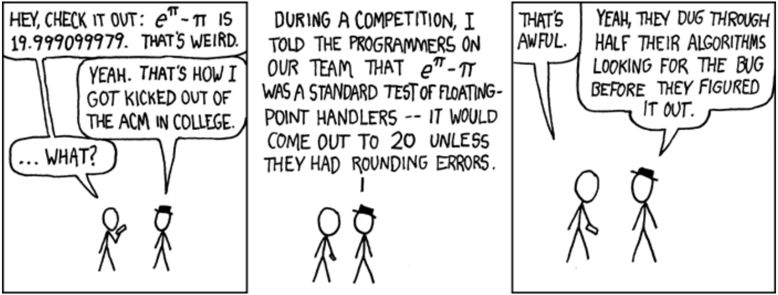
```
Require Import Reals Interval_tactic.
Local Open Scope R_scope.
```

Theorem Harrison97 : $\forall x : \mathbb{R}, -\frac{10831}{1000000} \leqslant x \leqslant \frac{10831}{1000000} \implies$

$$\left|(e^x - 1) - \left(x + \frac{8388676}{2^{24}}x^2 + \frac{11184876}{2^{26}}x^3\right)\right| \leqslant \frac{23}{27} \times \frac{1}{2^{33}}.$$

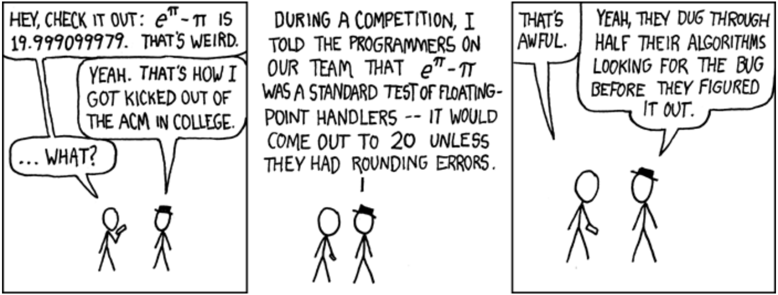
```
Proof.
intros x H.
interval with (i_bisect_taylor x 3, i_prec 50). (* in 0.5s *)
Qed.
```

Overview of the CoqInterval library — Proof example #2



(xkcd.com/217)

Overview of the CoqInterval library — Proof example #2



(xkcd.com/217)

```
Lemma xkcd217 : 19 999 099 979/10^9 < e^π − π < 19 999 099 980/10^9.
Proof.
split; interval with (i_prec 40). (* in 0.15s *)
Qed.
```

Related works on formalized game theory

- [René Vestergaard (2005): A constructive approach to sequential Nash equilibria]

$\leadsto$ proof, formalized in Coq, that all non-cooperative, sequential games have a Nash equilibrium

- [Stéphane Le Roux' PhD thesis, 2008]

$\leadsto$ generalizes and formalizes in Coq the notions of strategic game and Nash equilibrium (notably, not requiring payoffs to be real numbers)

- [Evgeny Dantsin, Jan-Georg Smaus, Sergei Soloviev (2012): Algorithms in Games Evolving in Time: Winning Strategies Based on Testing]

$\leadsto$ formalizes in Isabelle/HOL sufficient conditions for the computability of a winning strategy function (for two-player games evolving in time)

ACADIE

Perspectives

- **Motivation:** results of game theory have a key role in decision making and numerous applications $\Rightarrow$ providing a formal certificate would facilitate the audit of such decisions by independent experts.

Perspectives

- **Motivation:** results of game theory have a key role in decision making and numerous applications $\Rightarrow$ providing a formal certificate would facilitate the audit of such decisions by independent experts.
- **Aim:** identify key problems in game theory that are amenable to formal proof.

Perspectives

- **Motivation:** results of game theory have a key role in decision making and numerous applications $\Rightarrow$ providing a formal certificate would facilitate the audit of such decisions by independent experts.
- **Aim:** identify key problems in game theory that are amenable to formal proof.
- **Long-term goal:** obtain some game-theoretic and formally-certified components that may be extended, combined, and reused.

Perspectives

- **Motivation:** results of game theory have a key role in decision making and numerous applications $\Rightarrow$ providing a formal certificate would facilitate the audit of such decisions by independent experts.
- **Aim:** identify key problems in game theory that are amenable to formal proof.
- **Long-term goal:** obtain some game-theoretic and formally-certified components that may be extended, combined, and reused.

Thank you for your attention!