Proof assistants as a routine tool?

Neil Strickland

The Goal

The Goal

- Computer checkable proofs as a routine tool in ongoing mathematical research.

The Goal

- Computer checkable proofs as a routine tool in ongoing mathematical research.
- There are active research areas where reliability is a real concern.

The Goal

- Computer checkable proofs as a routine tool in ongoing mathematical research.
- There are active research areas where reliability is a real concern.
- Routine formalisation would create other cultural changes.

The Goal

- Computer checkable proofs as a routine tool in ongoing mathematical research.
- There are active research areas where reliability is a real concern.
- Routine formalisation would create other cultural changes.
- What is happening? Textbooks, landmarks, computer science.

The Goal

- Computer checkable proofs as a routine tool in ongoing mathematical research.
- There are active research areas where reliability is a real concern.
- Routine formalisation would create other cultural changes.
- What is happening? Textbooks, landmarks, computer science.
- Homotopy type theory.

The Goal

- Computer checkable proofs as a routine tool in ongoing mathematical research.
- There are active research areas where reliability is a real concern.
- Routine formalisation would create other cultural changes.
- What is happening? Textbooks, landmarks, computer science.
- Homotopy type theory.
- Experimentation and semi-formal verification in new mathematics; but not formal verification.

The Goal

- Computer checkable proofs as a routine tool in ongoing mathematical research.
- There are active research areas where reliability is a real concern.
- Routine formalisation would create other cultural changes.
- What is happening? Textbooks, landmarks, computer science.
- Homotopy type theory.
- Experimentation and semi-formal verification in new mathematics; but not formal verification.
- New interest creates new opportunities.

A plug for Freek Wiedijk

`http://www.cs.kun.nl/~freek/`

This is the personal home page of Freek Wiedijk.

In case you're looking for a way to reach me:

my snail-mail addresses are: Zandstraat 28-1, 1011 HL Amsterdam (home) and: Postbus 9010, 6500 GL Nijmegen, or: Room 01.17, Mercator 1, Toernooiveld 212, 6525 EC Nijmegen (work)
my telephone numbers are 06-20422671 (mobile), 020-4289648 (home) and 024-3652649 (work) and the fax number of my work is 024-3652728
my e-mail is freek@cs.ru.nl (if you want to make sure that your mail won't be eaten by my spam filter, mention free ultrafilters in the subject line of your message)

For my American friends: the name Freek is pronounced like "Phrake". It's a perfectly ordinary Dutch name (from Frederic), no reference to freak was ever intended. And "Wiedijk" is pronounced like "Weedike".

(Catalogs, comparisons, history, overview.)

Which framework?

Which framework?

- I will mostly discuss Coq + standard library.

Which framework?

- I will mostly discuss Coq + standard library.
- Coq + ssreflect: used for Four Colour Theorem, Odd Order Theorem. Large library. No compiled distribution. Complex compilation procedure fails. Little documentation.

Which framework?

- I will mostly discuss Coq + standard library.
- Coq + ssreflect: used for Four Colour Theorem, Odd Order Theorem. Large library. No compiled distribution. Complex compilation procedure fails. Little documentation.
- Coq + CoRN + MathClasses: Large library. Not well advertised or documented. No compiled distribution. Compilation needs extra tools, is very slow, and has some errors. No user documentation.

Which framework?

- I will mostly discuss Coq + standard library.
- Coq + ssreflect: used for Four Colour Theorem, Odd Order Theorem. Large library. No compiled distribution. Complex compilation procedure fails. Little documentation.
- Coq + CoRN + MathClasses: Large library. Not well advertised or documented. No compiled distribution. Compilation needs extra tools, is very slow, and has some errors. No user documentation.
- Coq + UniMath: Fairly large library, but not well modularised. Many theoretical insights; clearly effective for current investigations in HoTT.

Which framework?

- I will mostly discuss Coq + standard library.
- Coq + ssreflect: used for Four Colour Theorem, Odd Order Theorem. Large library. No compiled distribution. Complex compilation procedure fails. Little documentation.
- Coq + CoRN + MathClasses: Large library. Not well advertised or documented. No compiled distribution. Compilation needs extra tools, is very slow, and has some errors. No user documentation.
- Coq + UniMath: Fairly large library, but not well modularised. Many theoretical insights; clearly effective for current investigations in HoTT.
- Compatibility of CoRN and HoTT (for example) is unclear.

Which framework?

- I will mostly discuss Coq + standard library.
- Coq + ssreflect: used for Four Colour Theorem, Odd Order Theorem. Large library. No compiled distribution. Complex compilation procedure fails. Little documentation.
- Coq + CoRN + MathClasses: Large library. Not well advertised or documented. No compiled distribution. Compilation needs extra tools, is very slow, and has some errors. No user documentation.
- Coq + UniMath: Fairly large library, but not well modularised. Many theoretical insights; clearly effective for current investigations in HoTT.
- Compatibility of CoRN and HoTT (for example) is unclear.
- Agda: library is smaller than for Coq, organised more like CoRN. Interaction style is different, but not a major issue.

Which framework?

- I will mostly discuss Coq + standard library.
- Coq + ssreflect: used for Four Colour Theorem, Odd Order Theorem. Large library. No compiled distribution. Complex compilation procedure fails. Little documentation.
- Coq + CoRN + MathClasses: Large library. Not well advertised or documented. No compiled distribution. Compilation needs extra tools, is very slow, and has some errors. No user documentation.
- Coq + UniMath: Fairly large library, but not well modularised. Many theoretical insights; clearly effective for current investigations in HoTT.
- Compatibility of CoRN and HoTT (for example) is unclear.
- Agda: library is smaller than for Coq, organised more like CoRN. Interaction style is different, but not a major issue.
- Isabelle etc: mentioned for completeness.

Exercise: there are infinitely many primes

Exercise: there are infinitely many primes

- More specifically, if $n \in \mathbb{N}$ then the smallest nontrivial divisor of $n! + 1$ is a prime that is larger than n.

Exercise: there are infinitely many primes

- More specifically, if $n \in \mathbb{N}$ then the smallest nontrivial divisor of $n! + 1$ is a prime that is larger than n.
- This is first semester, first lecture material.

Exercise: there are infinitely many primes

- More specifically, if $n \in \mathbb{N}$ then the smallest nontrivial divisor of $n! + 1$ is a prime that is larger than n.
- This is first semester, first lecture material.
- Proof assistants will not be used routinely unless this exercise is straightforward.

Exercise: there are infinitely many primes

- More specifically, if $n \in \mathbb{N}$ then the smallest nontrivial divisor of $n! + 1$ is a prime that is larger than n.
- This is first semester, first lecture material.
- Proof assistants will not be used routinely unless this exercise is straightforward.
- Maple : `largerprime := (n) -> min(numtheory[divisors](n!+1) minus {1});`

Exercise: there are infinitely many primes

- More specifically, if $n \in \mathbb{N}$ then the smallest nontrivial divisor of $n! + 1$ is a prime that is larger than n.
- This is first semester, first lecture material.
- Proof assistants will not be used routinely unless this exercise is straightforward.
- Maple : `largerprime := (n) -> min(numtheory[divisors](n!+1) minus {1});`
- LaTeX:

```latex
\documentclass{amsart}
\newtheorem{theorem}{Theorem}

\begin{document}

\begin{theorem}
 For every natural number $n$, there is a prime $p$ with $p>n$.
\end{theorem}
\begin{proof}
 Put $D=\{d : d \mid n!+1 \text{ and } d > 1\}$.  This
 contains $ n!+1$ itself, so it is a nonempty set of natural numbers,
 so it has a smallest element, say $p$.  If there were a number $d$
 with $1<d<p$ such that $d\mid p$, then we would have
 $d\mid p\mid n!+1$, so $d\in D$, but also $d<p$, which is impossible
 as $p$ was defined to be the smallest element in $D$.  Thus, there
 cannot be any such number $d$, which means that $p$ is prime.  Next,
 note that the numbers $1,2,\dotsc,n$ all divide $n!$ and so do not
 divide $n!+1$, so $p$ cannot be any of these numbers, so $p>n$.
\end{proof}

\end{document}
```

Documentation I would have liked to see

Documentation I would have liked to see

- General survey of the library: here is where you find $\mathbb{N}$, $\mathbb{Z}$ and $\mathbb{Q}$. Here is how to load up the basic definitions about primes.

Documentation I would have liked to see

- General survey of the library: here is where you find $\mathbb{N}$, $\mathbb{Z}$ and $\mathbb{Q}$. Here is how to load up the basic definitions about primes.
- Conversion of number types.

Documentation I would have liked to see

- General survey of the library: here is where you find $\mathbb{N}$, $\mathbb{Z}$ and $\mathbb{Q}$. Here is how to load up the basic definitions about primes.
- Conversion of number types.
- Explanation for mathematicians of propositions as types, proofs as terms, passing proofs as arguments, (in)equality defined inductively.

Documentation I would have liked to see

- General survey of the library: here is where you find $\mathbb{N}$, $\mathbb{Z}$ and $\mathbb{Q}$. Here is how to load up the basic definitions about primes.
- Conversion of number types.
- Explanation for mathematicians of propositions as types, proofs as terms, passing proofs as arguments, (in)equality defined inductively.
- Practical advice for `Prop` vs `Set` vs `Type`, from user perspective.

Documentation I would have liked to see

- General survey of the library: here is where you find $\mathbb{N}$, $\mathbb{Z}$ and $\mathbb{Q}$. Here is how to load up the basic definitions about primes.
- Conversion of number types.
- Explanation for mathematicians of propositions as types, proofs as terms, passing proofs as arguments, (in)equality defined inductively.
- Practical advice for `Prop` vs `Set` vs `Type`, from user perspective.
- Examples of common proof patterns: chains of (in)equalities, cases, cases that cannot arise, contradiction, (structural) induction.

Documentation I would have liked to see

- General survey of the library: here is where you find $\mathbb{N}$, $\mathbb{Z}$ and $\mathbb{Q}$. Here is how to load up the basic definitions about primes.
- Conversion of number types.
- Explanation for mathematicians of propositions as types, proofs as terms, passing proofs as arguments, (in)equality defined inductively.
- Practical advice for `Prop` vs `Set` vs `Type`, from user perspective.
- Examples of common proof patterns: chains of (in)equalities, cases, cases that cannot arise, contradiction, (structural) induction.
- The tactics `omega` and `ring`, explained early.

Documentation I would have liked to see

- General survey of the library: here is where you find $\mathbb{N}$, $\mathbb{Z}$ and $\mathbb{Q}$. Here is how to load up the basic definitions about primes.
- Conversion of number types.
- Explanation for mathematicians of propositions as types, proofs as terms, passing proofs as arguments, (in)equality defined inductively.
- Practical advice for `Prop` vs `Set` vs `Type`, from user perspective.
- Examples of common proof patterns: chains of (in)equalities, cases, cases that cannot arise, contradiction, (structural) induction.
- The tactics `omega` and `ring`, explained early.
- Decidable propositions. $P \vee \neg P$ vs $\{P\} + \{\neg P\}$ and $\exists x\, Px$ vs $\{x \mid Px\}$.

Documentation I would have liked to see

- General survey of the library: here is where you find $\mathbb{N}$, $\mathbb{Z}$ and $\mathbb{Q}$. Here is how to load up the basic definitions about primes.
- Conversion of number types.
- Explanation for mathematicians of propositions as types, proofs as terms, passing proofs as arguments, (in)equality defined inductively.
- Practical advice for `Prop` vs `Set` vs `Type`, from user perspective.
- Examples of common proof patterns: chains of (in)equalities, cases, cases that cannot arise, contradiction, (structural) induction.
- The tactics `omega` and `ring`, explained early.
- Decidable propositions. $P \vee \neg P$ vs $\{P\} + \{\neg P\}$ and $\exists x\, Px$ vs $\{x \mid Px\}$.
- A collection of undergraduate level proofs with detailed, line by line commentary.

Documentation I actually read

Documentation I actually read

- Coq d'Art (in French; English version is not online).

Documentation I actually read

- Coq d'Art (in French; English version is not online).
- The reference manual.

Documentation I actually read

- Coq d'Art (in French; English version is not online).
- The reference manual.
- Various tutorials. They look quite good for applications in computer science, but mathematical content is thin.

Documentation I actually read

- Coq d'Art (in French; English version is not online).
- The reference manual.
- Various tutorials. They look quite good for applications in computer science, but mathematical content is thin.
- Library source code.

Documentation I actually read

- Coq d'Art (in French; English version is not online).
- The reference manual.
- Various tutorials. They look quite good for applications in computer science, but mathematical content is thin.
- Library source code.
- Other background: several large scale software systems in a wide variety of languages; extensive semi-formal verification in Maple and Mathematica.

Documentation I actually read

- Coq d'Art (in French; English version is not online).
- The reference manual.
- Various tutorials. They look quite good for applications in computer science, but mathematical content is thin.
- Library source code.
- Other background: several large scale software systems in a wide variety of languages; extensive semi-formal verification in Maple and Mathematica.
- I proved in Agda and Coq that there are infinitely many primes. Both were extremely painful.

Has it been done already?

Has it been done already?

- There is a Coq proof of the infinitude of primes written in 2006 by Russell O'Connor.

Has it been done already?

- There is a Coq proof of the infinitude of primes written in 2006 by Russell O'Connor.
- Google could not find me any other proof.

Has it been done already?

- There is a Coq proof of the infinitude of primes written in 2006 by Russell O'Connor.
- Google could not find me any other proof.
- O'Connor's proof appears in the Cocorico wiki, not in the standard Coq library or the usual collection of user contributions.

Has it been done already?

- There is a Coq proof of the infinitude of primes written in 2006 by Russell O'Connor.
- Google could not find me any other proof.
- O'Connor's proof appears in the Cocorico wiki, not in the standard Coq library or the usual collection of user contributions.
- It was written for Coq 7.3, and no longer works in the current Coq 8.4 because of changes in syntax rules and changes in the organisation of the standard library.

Has it been done already?

- There is a Coq proof of the infinitude of primes written in 2006 by Russell O'Connor.
- Google could not find me any other proof.
- O'Connor's proof appears in the Cocorico wiki, not in the standard Coq library or the usual collection of user contributions.
- It was written for Coq 7.3, and no longer works in the current Coq 8.4 because of changes in syntax rules and changes in the organisation of the standard library.
- It is largely self-contained, and so includes many basic facts (like $\forall a, b \in \mathbb{N}\ (a > b \rightarrow a \neq 0)$) as well as the definition and basic properties of divisibility and prime numbers.

Has it been done already?

- There is a Coq proof of the infinitude of primes written in 2006 by Russell O'Connor.
- Google could not find me any other proof.
- O'Connor's proof appears in the Cocorico wiki, not in the standard Coq library or the usual collection of user contributions.
- It was written for Coq 7.3, and no longer works in the current Coq 8.4 because of changes in syntax rules and changes in the organisation of the standard library.
- It is largely self-contained, and so includes many basic facts (like $\forall a, b \in \mathbb{N}\ (a > b \to a \neq 0)$) as well as the definition and basic properties of divisibility and prime numbers.
- The lemmas are just named P1 to P29, so any other script that referenced them would not be readable.

Has it been done already?

- There is a Coq proof of the infinitude of primes written in 2006 by Russell O'Connor.
- Google could not find me any other proof.
- O'Connor's proof appears in the Cocorico wiki, not in the standard Coq library or the usual collection of user contributions.
- It was written for Coq 7.3, and no longer works in the current Coq 8.4 because of changes in syntax rules and changes in the organisation of the standard library.
- It is largely self-contained, and so includes many basic facts (like $\forall a, b \in \mathbb{N}\ (a > b \to a \neq 0)$) as well as the definition and basic properties of divisibility and prime numbers.
- The lemmas are just named P1 to P29, so any other script that referenced them would not be readable.
- The whole proof is 826 lines long. There are no comments.

Has it been done already?

- There is a Coq proof of the infinitude of primes written in 2006 by Russell O'Connor.
- Google could not find me any other proof.
- O'Connor's proof appears in the Cocorico wiki, not in the standard Coq library or the usual collection of user contributions.
- It was written for Coq 7.3, and no longer works in the current Coq 8.4 because of changes in syntax rules and changes in the organisation of the standard library.
- It is largely self-contained, and so includes many basic facts (like $\forall a, b \in \mathbb{N}\ (a > b \to a \neq 0)$) as well as the definition and basic properties of divisibility and prime numbers.
- The lemmas are just named P1 to P29, so any other script that referenced them would not be readable.
- The whole proof is 826 lines long. There are no comments.
- As is typical with Coq proof scripts, one cannot easily see how the proof works without stepping through it in CoqIde.

Which numbers?

Which numbers?

- `Coq.Init.Datatypes.nat`

Which numbers?

- `Coq.Init.Datatypes.nat`
- `Coq.Arith.BinNat.N`

Which numbers?

- `Coq.Init.Datatypes.nat`
- `Coq.Arith.BinNat.N`
- `Coq.Numbers.Natural.Abstract.NBase`; some kind of abstraction layer?

Which numbers?

- `Coq.Init.Datatypes.nat`
- `Coq.Arith.BinNat.N`
- `Coq.Numbers.Natural.Abstract.NBase`; some kind of abstraction layer?
- Natural numbers in CoRN? Another abstraction layer?

Which numbers?

- `Coq.Init.Datatypes.nat`
- `Coq.Arith.BinNat.N`
- `Coq.Numbers.Natural.Abstract.NBase`; some kind of abstraction layer?
- Natural numbers in CoRN? Another abstraction layer?
- ssreflect and the HoTT library also have their own natural numbers.

Which numbers?

- `Coq.Init.Datatypes.nat`
- `Coq.Arith.BinNat.N`
- `Coq.Numbers.Natural.Abstract.NBase`; some kind of abstraction layer?
- Natural numbers in CoRN? Another abstraction layer?
- ssreflect and the HoTT library also have their own natural numbers.
- There are also several variants of $\mathbb{Z}$.

Which numbers?

- `Coq.Init.Datatypes.nat`
- `Coq.Arith.BinNat.N`
- `Coq.Numbers.Natural.Abstract.NBase`; some kind of abstraction layer?
- Natural numbers in CoRN? Another abstraction layer?
- ssreflect and the HoTT library also have their own natural numbers.
- There are also several variants of $\mathbb{Z}$.
- Many questions about compatibility and conversion.

Divisibility and primes?

Divisibility and primes?

- `FundamentalArithmetics` in user contribs

Divisibility and primes?

- `FundamentalArithmetics` in user contribs
- `Maths` in user contribs

Divisibility and primes?

- `FundamentalArithmetics` in user contribs
- `Maths` in user contribs
- Embedded in O'Connor's proof of infinitude of primes

Divisibility and primes?

- `FundamentalArithmetics` in user contribs
- `Maths` in user contribs
- Embedded in O'Connor's proof of infinitude of primes
- `Coq.ZArith.Znumtheory` in the standard library

Divisibility and primes?

- `FundamentalArithmetics` in user contribs
- `Maths` in user contribs
- Embedded in O'Connor's proof of infinitude of primes
- `Coq.ZArith.Znumtheory` in the standard library
- Building user contributions requires additional tools, fiddling with environment variables.

Smallest elements

Smallest elements

- Every nonempty, decidable subset of $\mathbb{N}$ has a smallest element.

Smallest elements

- Every nonempty, decidable subset of $\mathbb{N}$ has a smallest element.
- One needs to understand the framework for decidability.

Smallest elements

- Every nonempty, decidable subset of $\mathbb{N}$ has a smallest element.
- One needs to understand the framework for decidability.
- There is a version in `Coq.Arith.Wf_nat`; initially, I did not find it.

Smallest elements

- Every nonempty, decidable subset of $\mathbb{N}$ has a smallest element.
- One needs to understand the framework for decidability.
- There is a version in `Coq.Arith.Wf_nat`; initially, I did not find it.
- I could not use it because of `Prop` vs `Set` issues.

Smallest elements

- Every nonempty, decidable subset of $\mathbb{N}$ has a smallest element.
- One needs to understand the framework for decidability.
- There is a version in `Coq.Arith.Wf_nat`; initially, I did not find it.
- I could not use it because of `Prop` vs `Set` issues.
- From a constructive proof that $\exists! \, n \, P(n)$, you can "obviously" extract the value of n. You have to spend a lot of time reading non-obvious parts of the reference manual to understand why this does not work, and how to reorganise to avoid the problem.

Smallest elements

- Every nonempty, decidable subset of $\mathbb{N}$ has a smallest element.
- One needs to understand the framework for decidability.
- There is a version in `Coq.Arith.Wf_nat`; initially, I did not find it.
- I could not use it because of `Prop` vs `Set` issues.
- From a constructive proof that $\exists! \, n \, P(n)$, you can "obviously" extract the value of n. You have to spend a lot of time reading non-obvious parts of the reference manual to understand why this does not work, and how to reorganise to avoid the problem.
- I wrote my own proof in a rather different style from `Coq.Arith.Wf_nat`. It took 76 lines and was painful.

Smallest elements

- Every nonempty, decidable subset of $\mathbb{N}$ has a smallest element.
- One needs to understand the framework for decidability.
- There is a version in `Coq.Arith.Wf_nat`; initially, I did not find it.
- I could not use it because of `Prop` vs `Set` issues.
- From a constructive proof that $\exists! \, n \, P(n)$, you can "obviously" extract the value of n. You have to spend a lot of time reading non-obvious parts of the reference manual to understand why this does not work, and how to reorganise to avoid the problem.
- I wrote my own proof in a rather different style from `Coq.Arith.Wf_nat`. It took 76 lines and was painful.
- The main thing that would have reduced the pain: comparable examples, heavily annotated.

Smallest elements

- Every nonempty, decidable subset of $\mathbb{N}$ has a smallest element.
- One needs to understand the framework for decidability.
- There is a version in `Coq.Arith.Wf_nat`; initially, I did not find it.
- I could not use it because of `Prop` vs `Set` issues.
- From a constructive proof that $\exists! \, n \, P(n)$, you can "obviously" extract the value of n. You have to spend a lot of time reading non-obvious parts of the reference manual to understand why this does not work, and how to reorganise to avoid the problem.
- I wrote my own proof in a rather different style from `Coq.Arith.Wf_nat`. It took 76 lines and was painful.
- The main thing that would have reduced the pain: comparable examples, heavily annotated.
- I have started writing an extractable proof in the style of `Coq.Arith.Wf_nat`, but have not finished.

Remaining steps

Remaining steps

- Find the smallest element p of $\{d : d > 1,\ d|m\}$ or $\{d' : (d' + 2)|m\}$. Check that it is prime.

Remaining steps

- Find the smallest element p of $\{d : d > 1,\ d|m\}$ or $\{d' : (d' + 2)|m\}$. Check that it is prime.
- My proof is 102 lines.

Remaining steps

- Find the smallest element p of $\{d : d > 1,\ d|m\}$ or $\{d' : (d'+2)|m\}$. Check that it is prime.
- My proof is 102 lines.
- It is written in Agda-like style, with (reasonably efficient) proof terms, and semi-meaningful names for intermediate terms.

Remaining steps

- Find the smallest element p of $\{d : d > 1,\ d|m\}$ or $\{d' : (d'+2)|m\}$. Check that it is prime.
- My proof is 102 lines.
- It is written in Agda-like style, with (reasonably efficient) proof terms, and semi-meaningful names for intermediate terms.
- The output is a record, packaging p with a proof of its properties.

Remaining steps

- Find the smallest element p of $\{d : d > 1,\ d|m\}$ or $\{d' : (d' + 2)|m\}$. Check that it is prime.
- My proof is 102 lines.
- It is written in Agda-like style, with (reasonably efficient) proof terms, and semi-meaningful names for intermediate terms.
- The output is a record, packaging p with a proof of its properties.
- An irritatingly large portion deals with exceptional cases 0 and 1.

Remaining steps

- Find the smallest element p of $\{d : d > 1,\ d|m\}$ or $\{d' : (d' + 2)|m\}$. Check that it is prime.
- My proof is 102 lines.
- It is written in Agda-like style, with (reasonably efficient) proof terms, and semi-meaningful names for intermediate terms.
- The output is a record, packaging p with a proof of its properties.
- An irritatingly large portion deals with exceptional cases 0 and 1.
- The main thing that would have reduced the pain: comparable examples, heavily annotated.

Remaining steps

- Find the smallest element p of $\{d : d > 1,\ d|m\}$ or $\{d' : (d' + 2)|m\}$. Check that it is prime.
- My proof is 102 lines.
- It is written in Agda-like style, with (reasonably efficient) proof terms, and semi-meaningful names for intermediate terms.
- The output is a record, packaging p with a proof of its properties.
- An irritatingly large portion deals with exceptional cases 0 and 1.
- The main thing that would have reduced the pain: comparable examples, heavily annotated.
- Final step: apply the above with $m = n! + 1$.

Remaining steps

- Find the smallest element p of $\{d : d > 1,\ d|m\}$ or $\{d' : (d'+2)|m\}$. Check that it is prime.
- My proof is 102 lines.
- It is written in Agda-like style, with (reasonably efficient) proof terms, and semi-meaningful names for intermediate terms.
- The output is a record, packaging p with a proof of its properties.
- An irritatingly large portion deals with exceptional cases 0 and 1.
- The main thing that would have reduced the pain: comparable examples, heavily annotated.
- Final step: apply the above with $m = n! + 1$.
- One needs basic facts like $k!|n!$ when $0 \leq k \leq n$, and $k|n!$ when $0 < k \leq n!$. I spent 78 lines on these. It was not too painful, but it would be better if these facts were in `Coq.Arith.Factorial`.

What I would like to see

What I would like to see

- Documentation for mathematicians.

What I would like to see

- Documentation for mathematicians.
- A maintained collection of heavily documented proofs.

What I would like to see

- Documentation for mathematicians.
- A maintained collection of heavily documented proofs.
- A much larger standard library, incorporating CoRN.

What I would like to see

- Documentation for mathematicians.
- A maintained collection of heavily documented proofs.
- A much larger standard library, incorporating CoRN.
  - Consistent coding conventions.

What I would like to see

- Documentation for mathematicians.
- A maintained collection of heavily documented proofs.
- A much larger standard library, incorporating CoRN.
  - Consistent coding conventions.
  - Systematic namespaces.

What I would like to see

- Documentation for mathematicians.
- A maintained collection of heavily documented proofs.
- A much larger standard library, incorporating CoRN.
  - Consistent coding conventions.
  - Systematic namespaces.
  - Meaningful documentation for end users.

What I would like to see

- Documentation for mathematicians.
- A maintained collection of heavily documented proofs.
- A much larger standard library, incorporating CoRN.
  - Consistent coding conventions.
  - Systematic namespaces.
  - Meaningful documentation for end users.
- User contributions:

What I would like to see

- Documentation for mathematicians.
- A maintained collection of heavily documented proofs.
- A much larger standard library, incorporating CoRN.
  - Consistent coding conventions.
  - Systematic namespaces.
  - Meaningful documentation for end users.
- User contributions:
  - Regression tests.

What I would like to see

- Documentation for mathematicians.
- A maintained collection of heavily documented proofs.
- A much larger standard library, incorporating CoRN.
  - Consistent coding conventions.
  - Systematic namespaces.
  - Meaningful documentation for end users.
- User contributions:
  - Regression tests.
  - A promotion process.

What I would like to see

- Documentation for mathematicians.
- A maintained collection of heavily documented proofs.
- A much larger standard library, incorporating CoRN.
  - Consistent coding conventions.
  - Systematic namespaces.
  - Meaningful documentation for end users.
- User contributions:
  - Regression tests.
  - A promotion process.
  - Distribution in compiled form.

What I would like to see

- Documentation for mathematicians.
- A maintained collection of heavily documented proofs.
- A much larger standard library, incorporating CoRN.
  - Consistent coding conventions.
  - Systematic namespaces.
  - Meaningful documentation for end users.
- User contributions:
  - Regression tests.
  - A promotion process.
  - Distribution in compiled form.
  - Download, unpack and build (if necessary) from CoqIde.

What I would like to see

- Documentation for mathematicians.
- A maintained collection of heavily documented proofs.
- A much larger standard library, incorporating CoRN.
    - Consistent coding conventions.
    - Systematic namespaces.
    - Meaningful documentation for end users.
- User contributions:
    - Regression tests.
    - A promotion process.
    - Distribution in compiled form.
    - Download, unpack and build (if necessary) from CoqIde.
- Crowd-sourced annotation.

What I would like to see

- Documentation for mathematicians.
- A maintained collection of heavily documented proofs.
- A much larger standard library, incorporating CoRN.
  - Consistent coding conventions.
  - Systematic namespaces.
  - Meaningful documentation for end users.
- User contributions:
  - Regression tests.
  - A promotion process.
  - Distribution in compiled form.
  - Download, unpack and build (if necessary) from CoqIde.
- Crowd-sourced annotation.
- Intelligent search with incremental autocompletion from a central index of the library and user contributions.

What I would like to see

- Documentation for mathematicians.
- A maintained collection of heavily documented proofs.
- A much larger standard library, incorporating CoRN.
  - Consistent coding conventions.
  - Systematic namespaces.
  - Meaningful documentation for end users.
- User contributions:
  - Regression tests.
  - A promotion process.
  - Distribution in compiled form.
  - Download, unpack and build (if necessary) from CoqIde.
- Crowd-sourced annotation.
- Intelligent search with incremental autocompletion from a central index of the library and user contributions.
- Notes and examples to assist with porting from other frameworks.

What I would like to see

- Documentation for mathematicians.
- A maintained collection of heavily documented proofs.
- A much larger standard library, incorporating CoRN.
  - Consistent coding conventions.
  - Systematic namespaces.
  - Meaningful documentation for end users.
- User contributions:
  - Regression tests.
  - A promotion process.
  - Distribution in compiled form.
  - Download, unpack and build (if necessary) from CoqIde.
- Crowd-sourced annotation.
- Intelligent search with incremental autocompletion from a central index of the library and user contributions.
- Notes and examples to assist with porting from other frameworks.
- Frameworks for using or trusting external engines.

What I would like to see

- Documentation for mathematicians.
- A maintained collection of heavily documented proofs.
- A much larger standard library, incorporating CoRN.
  - Consistent coding conventions.
  - Systematic namespaces.
  - Meaningful documentation for end users.
- User contributions:
  - Regression tests.
  - A promotion process.
  - Distribution in compiled form.
  - Download, unpack and build (if necessary) from CoqIde.
- Crowd-sourced annotation.
- Intelligent search with incremental autocompletion from a central index of the library and user contributions.
- Notes and examples to assist with porting from other frameworks.
- Frameworks for using or trusting external engines.
- Frameworks for trusting the literature.