Formal Mathematics and Application to Software Safety and Internet Security

Jean-Pierre Jouannaud
École Polytechnique
91400 Palaiseau, France

email: jouannaud@lix.polytechnique.fr
http: //w³.lix.polytechnique.fr/Labo/Jean-Pierre.Jouannaud

Project LogiCal, Pôle Commun de Recherche en Informatique du Plateau de Saclay, CNRS, École Polytechnique, INRIA, Université Paris-Sud.

November 5, 2004
Outline

1. Theorems and proofs in mathematics
2. Four celebrated examples
   - Examples from mathematics
   - Examples from computer science
3. Deductions and Computations
   - Foundations from mathematical logic
   - Integrating deductions and computations
4. Proof Assistants
5. Coq
6. Conclusion
A theorem is a mathematical statement whose proof consists in a succession of deductions following the rules of logic.

One rule allows using any existing theorem.

To be sure a proof is correct, mathematicians require that it can be entirely read and understood by other mathematicians.

Some proofs do not follow this schema: they involve computations that cannot be done or followed by a mathematician in a lifetime.
A theorem is a mathematical statement whose proof consists in a succession of deductions following the rules of logic.

One rule allows using any existing theorem.

To be sure a proof is correct, mathematicians require that it can be entirely read and understood by other mathematicians.

Some proofs do not follow this schema: they involve computations that cannot be done or followed by a mathematician in a lifetime.
A theorem is a mathematical statement whose proof consists in a succession of deductions following the rules of logic.

One rule allows using any existing theorem.

To be sure a proof is correct, mathematicians require that it can be entirely read and understood by other mathematicians.

Some proofs do not follow this schema: they involve computations that cannot be done or followed by a mathematician in a lifetime.
A theorem is a mathematical statement whose proof consists in a succession of deductions following the rules of logic.

One rule allows using any existing theorem.

To be sure a proof is correct, mathematicians require that it can be entirely read and understood by other mathematicians.

Some proofs do not follow this schema: they involve computations that cannot be done or followed by a mathematician in a lifetime.
Four examples from mathematics and computer science

- **Four colors theorem:** 1200 hours of computations by Appel and Haken in 1976.
- **Kepler’s conjecture:** over ten years of computations with more than $10^5$ polynomials having over 100 variables and over 1000 constants by Hales in 1998.
- **Primality:** $4405^{2638} + 2638^{4405}$ is the biggest (15071 digits) proved “ordinary prime”: 720 days of computation by Morain et al in 2003.
- **Authentication:** Needham-Schröder protocol shown wrong by machine in 1996 by Lowe.
Four examples from mathematics and computer science

- **Four colors theorem**: 1200 hours of computations by Appel and Haken in 1976.
- **Kepler’s conjecture**: over ten years of computations with more than $10^5$ polynomials having over 100 variables and over 1000 constants by Hales in 1998.
- **Primality**: $4405^{2638} + 2638^{4405}$ is the biggest (15071 digits) proved “ordinary prime”: 720 days of computation by Morain at al in 2003.
- **Authentication**: Needham-Schröder protocol shown wrong by machine in 1996 by Lowe.
Four examples from mathematics and computer science

- **Four colors theorem:** 1200 hours of computations by Appel and Haken in 1976.
- **Kepler’s conjecture:** over ten years of computations with more than $10^5$ polynomials having over 100 variables and over 1000 constants by Hales in 1998.
- **Primality:** $4405^{2638} + 2638^{4405}$ is the biggest (15071 digits) proved “ordinary prime”: 720 days of computation by Morain at al in 2003.
- **Authentication:** Needham-Schröder protocol shown wrong by machine in 1996 by Lowe.
Four examples from mathematics and computer science

- **Four colors theorem**: 1200 hours of computations by Appel and Haken in 1976.
- **Kepler’s conjecture**: over ten years of computations with more than $10^5$ polynomials having over 100 variables and over 1000 constants by Hales in 1998.
- **Primality**: $4405^{2638} + 2638^{4405}$ is the biggest (15071 digits) proved “ordinary prime”: 720 days of computation by Morain et al in 2003.
- **Authentication**: Needham-Schröder protocol shown wrong by machine in 1996 by Lowe.
1852 Guthrie remarks that 4 colors suffice to draw the county map of England.

1878 Conjecture published by Cayley.

1879 First proof by Kempe, but unfortunately ...

1890 it only shows that five colors suffice.

1913 Birkhoff: reducible configurations

1969 Heesch: finding irreducible configurations

1976 Appel and Haken: enumerate and check the 1478 irreducible configurations on computer.

1852 Guthrie remarks that 4 colors suffice to draw the county map of England.

1878 Conjecture published by Cayley.

1879 First proof by Kempe, but unfortunately ...

1890 it only shows that five colors suffice.

1913 Birkhoff: reducible configurations

1969 Heesch: finding irreducible configurations

1976 Appel and Haken: enumerate and check the 1478 irreducible configurations on computer.

1852  Guthrie remarks that 4 colors suffice to draw the county map of England.
1878  Conjecture published by Cayley.
1879  First proof by Kempe, but unfortunately ...
1890  it only shows that five colors suffice.
1913  Birkhoff: reducible configurations
1969  Heesch: finding irreducible configurations
1976  Appel and Haken: enumerate and check the 1478 irreducible configurations on computer.
1852 Guthrie remarks that 4 colors suffice to draw the county map of England.
1878 Conjecture published by Cayley.
1879 First proof by Kempe, but unfortunately ...
1890 it only shows that five colors suffice.
1913 Birkhoff: reducible configurations
1969 Heesch: finding irreducible configurations
1976 Appel and Haken: enumerate and check the 1478 irreducible configurations on computer.
1852 Guthrie remarks that 4 colors suffice to draw the county map of England.
1878 Conjecture published by Cayley.
1879 First proof by Kempe, but unfortunately ...
1890 it only shows that five colors suffice.
1913 Birkhoff: reducible configurations
1969 Heesch: finding irreducible configurations
1976 Appel and Haken: enumerate and check the 1478 irreducible configurations on computer.
1852 Guthrie remarks that 4 colors suffice to draw the county map of England.

1878 Conjecture published by Cayley.

1879 First proof by Kempe, but unfortunately ...

1890 it only shows that five colors suffice.

1913 Birkhoff: reducible configurations

1969 Heesch: finding irreducible configurations

1976 Appel and Haken: enumerate and check the 1478 irreducible configurations on computer.

1852 Guthrie remarks that 4 colors suffice to draw the county map of England.
1878 Conjecture published by Cayley.
1879 First proof by Kempe, but unfortunately ...
1890 it only shows that five colors suffice.
1913 Birkhoff: reducible configurations
1969 Heesch: finding irreducible configurations
1976 Appel and Haken: enumerate and check the 1478 irreducible configurations on computer.
1852 Guthrie remarks that 4 colors suffice to draw the county map of England.
1878 Conjecture published by Cayley.
1879 First proof by Kempe, but unfortunately ...
1890 it only shows that five colors suffice.
1913 Birkhoff: reducible configurations
1969 Heesch: finding irreducible configurations
1976 Appel and Haken: enumerate and check the 1478 irreducible configurations on computer.
Face-centered cubic packing
1610 Sir Raleigh asks Harriot how to compute the number of cannon balls in a stack.

1610 Harriot solves it, wonders which packing is best in space, and writes to Kepler.

1611 Kepler conjectures that best is “face centred cubic packing” ... used daily by fruit sellers.

1910 Thue solves the circles packing problem.

... After numerous wrong proofs in 388 years, 1998 Hales solves the spheres packing problem.

- Dimension 4: networks of crystals.
- Higher dimensions: error correcting codes.
Packing spheres

1610 Sir Raleigh asks Harriot how to compute the number of cannon balls in a stack.

1610 Harriot solves it, wonders which packing is best in space, and writes to Kepler.

1611 Kepler conjectures that best is “face centred cubic packing” ... used daily by fruit sellers.

1910 Thue solves the circles packing problem.

... After numerous wrong proofs in 388 years,

1998 Hales solves the spheres packing problem.

• Dimension 4: networks of cristals.
• Higher dimensions: error correcting codes.
1610 Sir Raleigh asks Harriot how to compute the number of cannon balls in a stack.

1610 Harriot solves it, wonders which packing is best in space, and writes to Kepler.

1611 Kepler conjectures that best is “face centred cubic packing” ... used daily by fruit sellers.

1910 Thue solves the circles packing problem.

... After numerous wrong proofs in 388 years,

1998 Hales solves the spheres packing problem.

- Dimension 4: networks of crystals.
- Higher dimensions: error correcting codes.
Packing spheres

1610 Sir Raleigh asks Harriot how to compute the number of canon balls in a stack.

1610 Harriot solves it, wonders which packing is best in space, and writes to Kepler.

1611 Kepler conjectures that best is “face centred cubic packing” ... used daily by fruit sellers.

1910 Thue solves the circles packing problem.

... After numerous wrong proofs in 388 years,

1998 Hales solves the spheres packing problem.

- Dimension 4: networks of crystals.
- Higher dimensions: error correcting codes.
1610 **Sir Raleigh** asks Harriot how to compute the number of cannon balls in a stack.

1610 **Harriot** solves it, wonders which packing is best in space, and writes to Kepler.

1611 **Kepler** conjectures that best is “face centred cubic packing” ... used daily by fruit sellers.

1910 **Thue** solves the circles packing problem.

... After numerous wrong proofs in 388 years,

1998 **Hales** solves the spheres packing problem.

- Dimension 4: networks of crystals.
- Higher dimensions: error correcting codes.
1610 Sir Raleigh asks Harriot how to compute the number of canon balls in a stack.

1610 Harriot solves it, wonders which packing is best in space, and writes to Kepler.

1611 Kepler conjectures that best is “face centred cubic packing” ... used daily by fruit sellers.

1910 Thue solves the circles packing problem.

... After numerous wrong proofs in 388 years,

1998 Hales solves the spheres packing problem.

- Dimension 4: networks of cristals.
- Higher dimensions: error correcting codes.
1610 Sir Raleigh asks Harriot how to compute the number of canon balls in a stack.

1610 Harriot solves it, wonders which packing is best in space, and writes to Kepler.

1611 Kepler conjectures that best is “face centred cubic packing” ... used daily by fruit sellers.

1910 Thue solves the circles packing problem.

... After numerous wrong proofs in 388 years,

1998 Hales solves the spheres packing problem.

- Dimension 4: networks of cristals.

- Higher dimensions: error correcting codes.
1610 Sir Raleigh asks Harriot how to compute the number of canon balls in a stack.

1610 Harriot solves it, wonders which packing is best in space, and writes to Kepler.

1611 Kepler conjectures that best is “face centred cubic packing” ... used daily by fruit sellers.

1910 Thue solves the circles packing problem.

... After numerous wrong proofs in 388 years,

1998 Hales solves the spheres packing problem.

- Dimension 4: networks of crystals.
- Higher dimensions: error correcting codes.
**Encryption:** Given message $m$ and public key $K$, compute message $m' = K(m)$.

**Decryption:** Given message $m'$ and private key $K^{-1}$ compute $m = K^{-1}(m')$.

**Requirements:**
- Encryption and decryption must be fast.
- Computing $K^{-1}$ from $K$ should be unfeasible.

**RSA private key:** pair $(p, q)$ of two primes.

**RSA public key:** product $pq$ of these primes.

**Primality testing:** can be made fast enough and bug free (certificate).

**Factoring:** computing $p, q$ from key $pq$ is very hard for large enough keys.
RSA

- **Encryption:** Given message \( m \) and public key \( K \), compute message \( m' = K(m) \).

- **Decryption:** Given message \( m' \) and private key \( K^{-1} \) compute \( m = K^{-1}(m') \).

- **Requirements:**
  Encryption and decryption must be fast.
  Computing \( K^{-1} \) from \( K \) should be unfeasible.

- **RSA private key:** pair \((p, q)\) of two primes.
- **RSA public key:** product \( pq \) of these primes.
- **Primality testing:** can be made fast enough and bug free (certificate).
- **Factoring:** computing \( p, q \) from key \( pq \) is very hard for large enough keys.
RSA

- **Encryption**: Given message $m$ and public key $K$, compute message $m' = K(m)$.
- **Decryption**: Given message $m'$ and private key $K^{-1}$ compute $m = K^{-1}(m')$.
- **Requirements**: Encryption and decryption must be fast. Computing $K^{-1}$ from $K$ should be unfeasible.
  - RSA private key: pair $(p, q)$ of two primes.
  - RSA public key: product $pq$ of these primes.
  - Primality testing: can be made fast enough and bug free (certificate).
  - Factoring: computing $p, q$ from key $= pq$ is very hard for large enough keys.
RSA

- **Encryption**: Given message $m$ and public key $K$, compute message $m' = K(m)$.
- **Decryption**: Given message $m'$ and private key $K^{-1}$ compute $m = K^{-1}(m')$.
- **Requirements**: Encryption and decryption must be fast. Computing $K^{-1}$ from $K$ should be unfeasible.
- **RSA private key**: pair $(p, q)$ of two primes.
- **RSA public key**: product $pq$ of these primes.
- **Primality testing**: can be made fast enough and bug free (certificate).
- **Factoring**: computing $p, q$ from key $pq$ is very hard for large enough keys.
RSA

- **Encryption:** Given message $m$ and public key $K$, compute message $m' = K(m)$.
- **Decryption:** Given message $m'$ and private key $K^{-1}$ compute $m = K^{-1}(m')$.

**Requirements:**
Encryption and decryption must be *fast*. Computing $K^{-1}$ from $K$ should be *unfeasible*.

- **RSA private key:** pair $(p, q)$ of two primes.
- **RSA public key:** product $pq$ of these primes.

- **Primality testing:** can be made fast enough and bug free (certificate).
- **Factoring:** computing $p, q$ from key $pq$ is very hard for large enough keys.
RSA

- **Encryption:** Given message $m$ and public key $K$, compute message $m' = K(m)$.
- **Decryption:** Given message $m'$ and private key $K^{-1}$ compute $m = K^{-1}(m')$.

**Requirements:**
- Encryption and decryption must be fast.
- Computing $K^{-1}$ from $K$ should be unfeasible.

- **RSA private key:** pair $(p, q)$ of two primes.
- **RSA public key:** product $pq$ of these primes.
- **Primality testing:** can be made fast enough and bug free (certificate).
- **Factoring:** computing $p, q$ from key $pq$ is very hard for large enough keys.
RSA

- **Encryption:** Given message $m$ and public key $K$, compute message $m' = K(m)$.
- **Decryption:** Given message $m'$ and private key $K^{-1}$ compute $m = K^{-1}(m')$.
- **Requirements:**
  Encryption and decryption must be **fast**.
  Computing $K^{-1}$ from $K$ should be **unfeasible**.
- **RSA private key:** pair $(p, q)$ of two primes.
- **RSA public key:** product $pq$ of these primes.
- **Primality testing:** can be made fast enough and bug free (certificate).
- **Factoring:** computing $p, q$ from key $= pq$ is very hard for large enough keys.
**Primality and Factoring**

- **Erathostenes**: First algorithm for primality.

1975 Pratt: Primality is in NP.

1985 Rivest, Shamir, Addleman propose the use of primes for public key cryptosystems.

2002 Agrawal, Kayal, Saxena: Primality is in P.

2003 Morain: Primality is in $n^3$ under a conjecture about the density of prime numbers.

- Factoring is subexponential, but not (yet) polynomial.

2003 A 155 digits RSA key broken by Morain.
Primality and Factoring

- **Erathostenes:** First algorithm for primality.

1975 **Pratt** Primality is in NP.

1985 **Rivest, Shamir, Addleman** propose the use of primes for public key cryptosystems.

2002 **Agrawal, Kayal, Saxena:** primality is in P.

2003 **Morain:** primality is in $n^3$ under a conjecture about the density of prime numbers.

- Factoring is subexponential, but not (yet) polynomial.

2003 A 155 digits RSA key broken by **Morain**.
Primality and Factoring

- **Erathosthenes**: First algorithm for primality.

1975 **Pratt** Primality is in NP.

1985 **Rivest, Shamir, Addleman** propose the use of primes for public key cryptosystems.

2002 **Agrawal, Kayal, Saxena**: primality is in P.

2003 **Morain**: primality is in $n^3$ under a conjecture about the density of prime numbers.

Factoring is subexponential, but not (yet) polynomial.

2003 A 155 digits RSA key broken by Morain.
- **Eratosthenes**: First algorithm for primality.

1975 **Pratt** Primality is in NP.

1985 **Rivest, Shamir, Addleman** propose the use of primes for public key cryptosystems.

2002 **Agrawal, Kayal, Saxena**: primality is in P.

2003 **Morain**: primality is in $n^3$ under a conjecture about the density of prime numbers.

- Factoring is subexponential, but not (yet) polynomial.

2003 A 155 digits RSA key broken by **Morain**.
- **Erathostenes**: First algorithm for primality.

1975 **Pratt** Primality is in NP.

1985 **Rivest, Shamir, Addleman** propose the use of primes for public key cryptosystems.

2002 **Agrawal, Kayal, Saxena**: primality is in P.

2003 **Morain**: primality is in \( n^3 \) under a conjecture about the density of prime numbers.

- Factoring is subexponential, but not (yet) polynomial.

2003 A 155 digits RSA key broken by **Morain**.
- Eratosthenes: First algorithm for primality.

1975 Pratt Primality is in NP.

1985 Rivest, Shamir, Addleman propose the use of primes for public key crytosystems.

2002 Agrawal, Kayal, Saxena: primality is in P.

2003 Morain: primality is in $n^3$ under a conjecture about the density of prime numbers.

- Factoring is subexponential, but not (yet) polynomial.

2003 A 155 digits RSA key broken by Morain.
Erathostenes: First algorithm for primality.

1975 Pratt Primality is in NP.

1985 Rivest, Shamir, Addleman propose the use of primes for public key cryptosystems.

2002 Agrawal, Kayal, Saxena: primality is in P.

2003 Morain: primality is in $n^3$ under a conjecture about the density of prime numbers.

Factoring is subexponential, but not (yet) polynomial.

2003 A 155 digits RSA key broken by Morain.
1978 Publication by Needham-Schröder of a protocol for mutual authentication. Used over 15 years ...

1996 A “middle man” attack is found by Lowe who gave a modification of the protocol.

- The protocol had been proved correct under implicit hypotheses not satisfied in practice.
- The new version has been proved correct for the Dolev-Yao model.
1978 Publication by Needham-Schröder of a protocol for mutual authentication. Used over 15 years ...

1996 A “middle man” attack is found by Lowe who gave a modification of the protocol.

- The protocol had been proved correct under implicit hypotheses not satisfied in practice.
- The new version has been proved correct for the Dolev-Yao model.
1978 Publication by Needham-Schröder of a protocol for mutual authentication. Used over 15 years …

1996 A “middle man” attack is found by Lowe who gave a modification of the protocol.
- The protocol had been proved correct under implicit hypotheses not satisfied in practice.
- The new version has been proved correct for the Dolev-Yao model.
1978 Publication by Needham-Schröder of a protocol for mutual authentication. Used over 15 years ...

1996 A “middle man” attack is found by Lowe who gave a modification of the protocol.

- The protocol had been proved correct under implicit hypotheses not satisfied in practice.
- The new version has been proved correct for the Dolev-Yao model.
Protocol

Agents $A, B, I$

Emails $A, B, I$

Nonce $N_x$ is a fresh random number

Public encryption keys: $K_A, K_B, K_I$

Secret decryption keys: $K_A^{-1}, K_B^{-1}, K_I^{-1}$

Run: sequence of 3 authentication messages

$$A \rightarrow B : A, B, \{N_A, A\}_{K_B}$$

$$B \rightarrow A : B, A, \{N_A, N_B\}_{K_A}$$

$$A \rightarrow B : A, B, \{N_B\}_{K_B}$$
Protocol

Agents $A, B, I$

Emails $A, B, I$

Nonce $N_x$ is a fresh random number

Public encryption keys: $K_A$, $K_B$, $K_I$

Secret decryption keys: $K_A^{-1}$, $K_B^{-1}$, $K_I^{-1}$

Run: sequence of 3 authentication messages

$A \rightarrow B : A, B, \{N_A, A\}_{K_B}$

$B \rightarrow A : B, A, \{N_A, N_B\}_{K_A}$

$A \rightarrow B : A, B, \{N_B\}_{K_B}$
Protocol

Agents $A, B, I$

Emails $A, B, I$

Nonce $N_x$ is a fresh random number

Public encryption keys: $K_A$, $K_B$, $K_I$

Secret decryption keys: $K_A^{-1}$, $K_B^{-1}$, $K_I^{-1}$

Run: sequence of 3 authentication messages

- $A \rightarrow B : A, B, \{N_A, A\}_{K_B}$
- $B \rightarrow A : B, A, \{N_A, N_B\}_{K_A}$
- $A \rightarrow B : A, B, \{N_B\}_{K_B}$
Agents $A, B, I$

Emails $A, B, I$

Nonce $N_x$ is a fresh random number

Public encryption keys: $K_A, K_B, K_I$

Secret decryption keys: $K_A^{-1}, K_B^{-1}, K_I^{-1}$

Run: sequence of 3 authentication messages

$$A \rightarrow B : A, B, \{N_A, A\}_{K_B}$$

$$B \rightarrow A : B, A, \{N_A, N_B\}_{K_A}$$

$$A \rightarrow B : A, B, \{N_B\}_{K_B}$$
Protocol

Agents $A, B, I$

Emails $A, B, I$

Nonce $N_x$ is a fresh random number

Public encryption keys: $K_A, K_B, K_I$

Secret decryption keys: $K_A^{-1}, K_B^{-1}, K_I^{-1}$

Run: sequence of 3 authentication messages

$A \rightarrow B : A, B, \{N_A, A\}_K_B$

$B \rightarrow A : B, A, \{N_A, N_B\}_K_A$

$A \rightarrow B : A, B, \{N_B\}_K_B$
Protocol

Agents $A, B, I$

Emails $A, B, I$

Nonce $N_x$ is a fresh random number

Public encryption keys: $K_A, K_B, K_I$

Secret decryption keys: $K_A^{-1}, K_B^{-1}, K_I^{-1}$

Run: sequence of 3 authentication messages

$A \rightarrow B : A, B, \{N_A, A\}_{K_B}$

$B \rightarrow A : B, A, \{N_A, N_B\}_{K_A}$

$A \rightarrow B : A, B, \{N_B\}_{K_B}$
Protocol

Agents $A, B, I$

Emails $A, B, I$

Nonce $N_x$ is a fresh random number

Public encryption keys: $K_A, K_B, K_I$

Secret decryption keys: $K_A^{-1}, K_B^{-1}, K_I^{-1}$

Run: sequence of 3 authentication messages

$$A \rightarrow B : A, B, \{N_A, A\}_{K_B}$$

$$B \rightarrow A : B, A, \{N_A, N_B\}_{K_A}$$

$$A \rightarrow B : A, B, \{N_B\}_{K_B}$$
Agents $A, B, I$

Emails $A, B, I$

Nonce $N_x$ is a fresh random number

Public encryption keys: $K_A, K_B, K_I$

Secret decryption keys: $K_A^{-1}, K_B^{-1}, K_I^{-1}$

Run: sequence of 3 authentication messages

\[
A \rightarrow B : A, B, \{N_A, A\}_{K_B}
\]

\[
B \rightarrow A : B, A, \{N_A, N_B\}_{K_A}
\]

\[
A \rightarrow B : A, B, \{N_B\}_{K_B}
\]
Agents $A, B, I$

Emails $A, B, I$

Nonce $N_x$ is a fresh random number

Public encryption keys: $K_A, K_B, K_I$

Secret decryption keys: $K^{-1}_A, K^{-1}_B, K^{-1}_I$

Run: sequence of 3 authentication messages

$A \rightarrow B : A, B, \{N_A, A\}_{K_B}$

$B \rightarrow A : B, A, \{N_A, N_B\}_{K_A}$

$A \rightarrow B : A, B, \{N_B\}_{K_B}$
Protocol

Agents $A, B, I$

Emails $A, B, I$

Nonce $N_x$ is a fresh random number

Public encryption keys: $K_A, K_B, K_I$

Secret decryption keys: $K_A^{-1}, K_B^{-1}, K_I^{-1}$

Run: sequence of 3 authentication messages

$A \rightarrow B : A, B, \{N_A, A\}^{K_B}$

$B \rightarrow A : B, A, \{N_A, N_B\}^{K_A}$

$A \rightarrow B : A, B, \{N_B\}^{K_B}$
Attack: man in the middle

\[ \alpha - 1 \quad \text{A} \rightarrow \text{I} : \text{A, I, } \{N_A, A\}_{K_I} \]

\[ \beta - 1 \quad \text{I} \rightarrow \text{B} : \text{I, B, } \{N_A, A\}_{K_B} \]

\[ \beta - 2 \quad \text{B} \rightarrow \text{I} : \text{B, I, } \{N_A, N_B\}_{K_A} \]

\[ \alpha - 2 \quad \text{I} \rightarrow \text{A} : \text{I, A, } \{N_A, N_B\}_{K_A} \]

\[ \alpha - 3 \quad \text{A} \rightarrow \text{I} : \text{A, I, } \{N_B\}_{K_I} \]

\[ \beta - 3 \quad \text{I} \rightarrow \text{B} : \text{I, B, } \{N_B\}_{K_B} \]

B believes he has carried out a run with A.
Attack: man in the middle

\[\alpha - 1 \quad A \rightarrow I : \ A, I, \{N_A, A\}_{K_I}\]

\[\beta - 1 \quad I \rightarrow B : \ I, B, \{N_A, A\}_{K_B}\]

\[\beta - 2 \quad B \rightarrow I : \ B, I, \{N_A, N_B\}_{K_A}\]

\[\alpha - 2 \quad I \rightarrow A : \ I, A, \{N_A, N_B\}_{K_A}\]

\[\alpha - 3 \quad A \rightarrow I : \ A, I, \{N_B\}_{K_I}\]

\[\beta - 3 \quad I \rightarrow B : \ I, B, \{N_B\}_{K_B}\]

B believes he has carried out a run with A.
Attack: man in the middle

\[
\begin{align*}
\alpha - 1 & \quad A \rightarrow I : A, I, \{ N_A, A \} K_I \\
\beta - 1 & \quad I \rightarrow B : I, B, \{ N_A, A \} K_B \\
\beta - 2 & \quad B \rightarrow I : B, I, \{ N_A, N_B \} K_A \\
\alpha - 2 & \quad I \rightarrow A : I, A, \{ N_A, N_B \} K_A \\
\alpha - 3 & \quad A \rightarrow I : A, I, \{ N_B \} K_I \\
\beta - 3 & \quad I \rightarrow B : I, B, \{ N_B \} K_B \\
\end{align*}
\]

B believes he has carried out a run with A.
Attack: man in the middle

\[\begin{align*}
\alpha - 1 & \quad A \rightarrow I : A, I, \{N_A, A\}_{K_I} \\
\beta - 1 & \quad I \rightarrow B : I, B, \{N_A, A\}_{K_B} \\
\beta - 2 & \quad B \rightarrow I : B, I, \{N_A, N_B\}_{K_A} \\
\alpha - 2 & \quad I \rightarrow A : I, A, \{N_A, N_B\}_{K_A} \\
\alpha - 3 & \quad A \rightarrow I : A, I, \{N_B\}_{K_I} \\
\beta - 3 & \quad I \rightarrow B : I, B, \{N_B\}_{K_B}
\end{align*}\]

*B believes he has carried out a run with A.*
Attack: man in the middle

\[ \alpha - 1 \quad A \rightarrow I : A, I, \{ N_A, A \}_K I \]

\[ \beta - 1 \quad I \rightarrow B : I, B, \{ N_A, A \}_K B \]

\[ \beta - 2 \quad B \rightarrow I : B, I, \{ N_A, N_B \}_K A \]

\[ \alpha - 2 \quad I \rightarrow A : I, A, \{ N_A, N_B \}_K A \]

\[ \alpha - 3 \quad A \rightarrow I : A, I, \{ N_B \}_K I \]

\[ \beta - 3 \quad I \rightarrow B : I, B, \{ N_B \}_K B \]

B believes he has carried out a run with A.
Attack: man in the middle

\[ \alpha - 1 \quad A \rightarrow I : A, I, \{N_A, A\}_K_i \]

\[ \beta - 1 \quad I \rightarrow B : I, B, \{N_A, A\}_K_B \]

\[ \beta - 2 \quad B \rightarrow I : B, I, \{N_A, N_B\}_K_A \]

\[ \alpha - 2 \quad I \rightarrow A : I, A, \{N_A, N_B\}_K_A \]

\[ \alpha - 3 \quad A \rightarrow I : A, I, \{N_B\}_K_i \]

\[ \beta - 3 \quad I \rightarrow B : I, B, \{N_B\}_K_B \]

B believes he has carried out a run with A.
$\alpha - 1 \quad A \rightarrow I : A, I, \{N_A, A\}_{K_I}$

$\beta - 1 \quad I \rightarrow B : I, B, \{N_A, A\}_{K_B}$

$\beta - 2 \quad B \rightarrow I : B, I, \{N_A, N_B\}_{K_A}$

$\alpha - 2 \quad I \rightarrow A : I, A, \{N_A, N_B\}_{K_A}$

$\alpha - 3 \quad A \rightarrow I : A, I, \{N_B\}_{K_I}$

$\beta - 3 \quad I \rightarrow B : I, B, \{N_B\}_{K_B}$

*B believes he has carried out a run with* $A$.  


Mathematicians attack the encryption algorithm
Computer scientists attack the cryptographic protocol
Physicists attack the transmission material
Thieves attack the man-machine interface
Mathematicians attack the encryption algorithm

Computer scientists attack the cryptographic protocol

Physicists attack the transmission material

Thieves attack the man-machine interface
- Mathematicians attack the encryption algorithm
- Computer scientists attack the cryptographic protocol
- Physicists attack the transmission material
- Thieves attack the man-machine interface
Mathematicians attack the encryption algorithm
Computer scientists attack the cryptographic protocol
Physicists attack the transmission material
Thieves attack the man-machine interface
*Given:* a statement about arithmetic.

*Question:* is it a theorem?

*Hilbert’s program:* finding an algorithm to answer this question is the most important task for a mathematician.

*Gödel’s answer:* no program can answer this question.
Given: a statement about arithmetic.

Question: is it a theorem?

Hilbert’s program: finding an algorithm to answer this question is the most important task for a mathematician.

Gödel’s answer: no program can answer this question.
Given: a statement about arithmetic.

Question: is it a theorem?

Hilbert’s program: finding an algorithm to answer this question is the most important task for a mathematician.

Gödel’s answer: no program can answer this question.
Given: a statement about arithmetic.

Question: is it a theorem?

Hilbert’s program: finding an algorithm to answer this question is the most important task for a mathematician.

Gödel’s answer: no program can answer this question.
Decision procedures are programs able to answer specific instances of the question.

For example, reachability is decidable in \( PSPACE \) for finite state systems.

Shostak: combine decision procedures.
Decision procedures are programs able to answer specific instances of the question. For example, reachability is decidable in $PSPACE$ for finite state systems. Shostak: combine decision procedures.
Decision procedures are programs able to answer specific instances of the question. For example, reachability is decidable in $PSPACE$ for finite state systems. Shostak: combine decision procedures.
**Given**: a statement $S$ about arithmetic and a proof $P$ of $S$.

**Question**: is the proof correct?

**Gentzen**: There is a program able to answer this question.

Such a program is called a *proof assistant*.

**Our target**: a proof assistant which
- is guaranteed to construct correct proofs,
- performs automatically in case of a decidable verification problem.
**Decidability of Proof-Checking**

- **Given:** a statement $S$ about arithmetic and a proof $P$ of $S$.
- **Question:** is the proof correct?
- **Gentzen:** There is a program able to answer this question.
- Such a program is called a *proof assistant*.
- Our target: a proof assistant which
  - is guaranteed to construct correct proofs,
  - performs automatically in case of a decidable verification problem.
**Given:** a statement $S$ about arithmetic and a proof $P$ of $S$.

**Question:** is the proof correct?

**Gentzen:** There is a program able to answer this question.

Such a program is called a *proof assistant*.

**Our target:** a proof assistant which
- is guaranteed to construct correct proofs,
- performs automatically in case of a decidable verification problem.
Decidability of Proof-Checking

- **Given:** a statement $S$ about arithmetic and a proof $P$ of $S$.

- **Question:** is the proof correct?

- **Gentzen:** There is a program able to answer this question.

- **Such a program is called a proof assistant.**

- **Our target:** a proof assistant which
  - is guaranteed to construct correct proofs,
  - performs automatically in case of a decidable verification problem.
Given: a statement S about arithmetic and a proof P of S.

Question: is the proof correct?

Gentzen: There is a program able to answer this question.

Such a program is called a *proof assistant*.

Our target: a proof assistant which
- is guaranteed to construct correct proofs,
- performs automatically in case of a decidable verification problem.
Integrating deductions and computations
In general, a proof requires deduction as well as computation steps:

- A proof of Even(2+2) is made of
  - the computation of $2 + 2$ resulting in 4
  - a proof of Even(4)
  - a mechanism to integrate both

Three ingredients are needed in proofs:

- **Deductions**: $\Gamma \vdash p : P$
- **Computations**: $\Gamma \vdash P \rightarrow Q$
- **Conversion**: $\frac{\Gamma \vdash p : P \quad \Gamma \vdash P \rightarrow Q}{\Gamma \vdash p : Q}$
In general, a proof requires deduction as well as computation steps:

A proof of Even(2+2) is made of
- the computation of $2 + 2$ resulting in 4
- a proof of Even(4)
- a mechanism to integrate both

Three ingredients are needed in proofs:

- **Deductions:** $\Gamma \vdash p : P$
- **Computations:** $\Gamma \vdash P \rightarrow Q$
- **Conversion:** \[
\frac{\Gamma \vdash p : P \quad \Gamma \vdash P \rightarrow Q}{\Gamma \vdash p : Q}
\]
Deductions and computations

In general, a proof requires deduction as well as computation steps:

A proof of Even(2+2) is made of
- the computation of $2 + 2$ resulting in 4
- a proof of Even(4)
- a mechanism to integrate both

Three ingredients are needed in proofs:

**deductions**: $\Gamma \vdash p : P$

**computations**: $\Gamma \vdash P \rightarrow Q$

**conversion**: $\Gamma \vdash p : P \quad \Gamma \vdash P \rightarrow Q \quad \Gamma \vdash p : Q$
Example: $2 + 2$ is even

Representing natural numbers in Peano notation with 0 and $s$, 4 is $s(s(s(s(0))))$.

$\Gamma = \{p : E(0), q : \forall x. E(x) \implies E(s(s(x))), \forall xy. x + s(y) \rightarrow s(x) + y, \forall x. x + 0 \rightarrow x\}$

Computation:
$\Gamma \vdash E(2+2) \rightarrow E(3+1) \rightarrow E(4+0) \rightarrow E(4)$

Conversion:

$\Gamma \vdash ?? : E(4)$
$\Gamma \vdash E(2+2) \rightarrow E(4)$

$\Gamma \vdash ?? : E(2+2)$
Example: $2 + 2$ is even

- Representing natural numbers in Peano notation with 0 and $s$, $4$ is $s(s(s(s(0))))$.
- $\Gamma = \{ p : E(0), q : \forall x. E(x) \rightarrow E(s(s(x))), \forall xy. x + s(y) \rightarrow s(x) + y, \forall x. x + 0 \rightarrow x \}$
- Computation:
  $\Gamma \vdash E(2 + 2) \rightarrow E(3 + 1) \rightarrow E(4 + 0) \rightarrow E(4)$
- Conversion:
  $\Gamma \vdash ??? : E(4)$
  $\Gamma \vdash E(2 + 2) \rightarrow E(4)$
  $\Gamma \vdash ??? : E(2 + 2)$
Example: $2 + 2$ is even

- Representing natural numbers in Peano notation with 0 and $s$, 4 is $s(s(s(s(0))))$.

  $$\Gamma = \{ p : E(0), q : \forall x. E(x) \implies E(s(s(x))),$$
  $$\forall xy. x + s(y) \rightarrow s(x) + y, \ \forall x. x + 0 \rightarrow x \}$$

- Computation:

  $$\Gamma \vdash E(2 + 2) \rightarrow E(3 + 1) \rightarrow E(4 + 0) \rightarrow E(4)$$

- Conversion:

  $$\Gamma \vdash ?? : E(4) \quad \Gamma \vdash E(2 + 2) \rightarrow E(4)$$

  $\Gamma \vdash ?? : E(2 + 2)$
Example: 2 + 2 is even

- Representing natural numbers in Peano notation with 0 and s, 4 is \( s(s(s(s(0)))) \).
- \( \Gamma = \{ p : E(0), q : \forall x. E(x) \Rightarrow E(s(s(x))), \\
\forall xy. x + s(y) \rightarrow s(x) + y, \forall x. x + 0 \rightarrow x \} \)

- **Computation:**
  \( \Gamma \vdash E(2 + 2) \rightarrow E(3 + 1) \rightarrow E(4 + 0) \rightarrow E(4) \)

- **Conversion:**
  \[
  \Gamma \vdash ?? : E(4) \quad \Gamma \vdash E(2 + 2) \rightarrow E(4) \\
  \hline
  \Gamma \vdash ?? : E(2 + 2)
  \]
Deduction:

\[
\vdash q(0, p) : E(2) \quad \vdash q : \forall x. E(x) \implies E(s(s(x))) \\
\vdash q(2) : E(2) \implies E(4) \\
\vdash q(2, q(0, p)) : E(4)
\]

\[
\vdash p : E(0) \quad \vdash q(0) : E(0) \implies E(2) \\
\vdash q(0, p) : E(2)
\]
Assuming computations terminate, then it becomes possible to check if a given proof $p$ of the proposition $A$ is correct or not.

The algorithm works by induction on the size of $A$, except for the conversion rule, where it must verify that $A \rightarrow B$.

This algorithm constitutes the kernel of a proof assistant.
Assuming computations terminate, then it becomes possible to check if a given proof $p$ of the proposition $A$ is correct or not.

The algorithm works by induction on the size of $A$, except for the conversion rule, where it must verify that $A \rightarrow B$.

This algorithm constitutes the kernel of a proof assistant.
Assuming computations terminate, then it becomes possible to check if a given proof \( p \) of the proposition \( A \) is correct or not.

The algorithm works by induction on the size of \( A \), except for the conversion rule, where it must verify that \( A \rightarrow B \).

This algorithm constitutes the kernel of a proof assistant.
Proof assistant

- A **logic programming language** dedicated to processing mathematics
- A set of deduction and computation rules which characterize the chosen logic.
- An proof-checking algorithm, **kernel** of the proof assistant.
- **Proof tactics** helping the user building proofs.
- **A tactic language** for writing new tactics.
- **Libraries** of proved theorems.
A **logic programming language** dedicated to processing mathematics

A set of deduction and computation rules which characterize the chosen logic.

An proof-checking algorithm, **kernel** of the proof assistant.

**Proof tactics** helping the user building proofs.

A **tactic language** for writing new tactics.

**Libraries** of proved theorems.
A logic programming language dedicated to processing mathematics

A set of deduction and computation rules which characterize the chosen logic.

An proof-checking algorithm, kernel of the proof assistant.

Proof tactics helping the user building proofs.

A tactic language for writing new tactics.

Libraries of proved theorems.
Proof assistant

- A logic programming language dedicated to processing mathematics
- A set of deduction and computation rules which characterize the chosen logic.
- An proof-checking algorithm, kernel of the proof assistant.
- Proof tactics helping the user building proofs.
- A tactic language for writing new tactics.
- Libraries of proved theorems.
Proof assistant

- A logic programming language dedicated to processing mathematics
- A set of deduction and computation rules which characterize the chosen logic.
- An proof-checking algorithm, kernel of the proof assistant.
- Proof tactics helping the user building proofs.
- A tactic language for writing new tactics.
- Libraries of proved theorems.
• A **logic programming language** dedicated to processing mathematics
• A set of deduction and computation rules which characterize the chosen **logic**.
• An proof-checking algorithm, **kernel** of the proof assistant.
• **Proof tactics** helping the user building proofs.
• A **tactic language** for writing new tactics.
• **Libraries** of proved theorems.
Major proof assistants

- Coq, PCRI, France.
- PVS, Stanford Research Institute, California.
- HOL, UK, and Isabelle, Germany.
- NuPRL (Cornell University), SVC, (Stanford), ACL2 (Arg. Nat. Lab.), LEGO (Edinburgh), Twelf (Carnegie-Mellon), Alf (Sweden), Mizar (Poland), B (Abrial’s company in France), ...
Major proof assistants

- **Coq**, PCRI, France.
- **PVS**, Stanford Research Institute, California.
- **HOL**, UK, and **Isabelle**, Germany.
- **NuPRL** (Cornell University), **SVC** (Stanford), **ACL2** (Arg. Nat. Lab.), **LEGO** (Edinburgh), **Twelf** (Carnegie-Mellon), **Alf** (Sweden), **Mizar** (Poland), **B** (Abrial’s company in France), ...
Major proof assistants

- **Coq**, PCRI, France.
- **PVS**, Stanford Research Institute, California.
- **HOL**, UK, and **Isabelle**, Germany.
- **NuPRL** (Cornell University), **SVC** (Stanford), **ACL2** (Arg. Nat. Lab.), **LEGO** (Edinburgh), **Twelf** (Carnegie-Mellon), **Alf** (Sweden), **Mizar** (Poland), **B** (Abrial’s company in France), ...
Major proof assistants

- Coq, PCRI, France.
- PVS, Stanford Research Institute, California.
- HOL, UK, and Isabelle, Germany.
- NuPRL (Cornell University), SVC, (Stanford), ACL2 (Arg. Nat. Lab.), LEGO (Edinburgh), Twelf (Carnegie-Mellon), Alf (Sweden), Mizar (Poland), B (Abrial’s company in France), ...
The proof assistant Coq
Coq’s logical foundations

- Kernel based on
  the *Calculus of Inductive Constructions* of Coquand and Paulin
  Interactive Modules and Fonctors of Chrzaszcz
  Compiler of Grégoire

- Comes with
  a code extractor by Letouzey
  a tactic language of Delahaye
  a graphic proof interface of Monate

- Prototype version includes
  A rewriting engine by Blanqui
  small proof engines by Strub
Coq’s logical foundations

- Kernel based on the Calculus of Inductive Constructions of Coquand and Paulin
- Interactive Modules and Fonctors of Chrzaszcz
- Compiler of Grégoire

- Comes with
  - a code extractor by Letouzey
  - a tactic language of Delahaye
  - a graphic proof interface of Monate

- Prototype version includes
  - A rewriting engine by Blanqui
  - small proof engines by Strub
Coq’s logical foundations

- Kernel based on the Calculus of Inductive Constructions of Coquand and Paulin
- Interactive Modules and Fonctors of Chrzaszcz
- Compiler of Grégoire

- Comes with a code extractor by Letouzey
- A tactic language of Delahaye
- A graphic proof interface of Monate

- Prototype version includes A rewriting engine by Blanqui
- Small proof engines by Strub
Module OrderedTypeFacts [O : OrderedType].
Lemma lt_not_gt : (x,y:O.t)(O.lt y y) → ¬ (O.lt y x).
Proof. Intros; Intro; Absurd (O.eq x x); EAuto.
Qed.

... many other lemmas...

End OrderedTypeFacts.
Module Type OrderedType.
Parameter t : Set.
Parameter eq : t → t → Prop.
Parameter eq_refl : (x:t)(eq x x).
Parameter eq_sym : (x,y:t) (eq x y) → (eq y x).
Parameter eq_trans : (x,y,z:t) (eq x y) → (eq y z) → (eq x z).
Parameter lt_trans : (x,y,z:t) (lt x y) → (lt y z) → (lt x z).
Parameter lt_not_eq : (x,y:t) (lt x y) → ¬ (eq x y).
Parameter compare : (x,y:t) (Comp lt eq x y).
End OrderedType.
Inductive Comp [X:Set; lt,eq:X→ X → Prop; x,y:X] : Set :=
   Lt : (lt x y) → (Comp lt eq x y)
   Eq : (eq x y) → (Comp lt eq x y)
   Gt : (lt y x) → (Comp lt eq x y).
The proof assistant Coq

- **Kernel**: 10K lines of Objective Caml
- **Tactics**: 100K lines of Objective Caml and Coq tactic language, outputing a proof term.
- **Libraries**: of checked proof developments and tactics,
- **Academic** as well as **industrial users**.
- **User’s group**, hotline, website, LGPL licence.
The proof assistant Coq

- Kernel: 10K lines of Objective Caml
- Tactics: 100K lines of Objective Caml and Coq tactic language, outputing a proof term.
- Libraries of checked proof developments and tactics,
- Academic as well as industrial users.
- User’s group, hotline, website, LGPL licence.
The proof assistant Coq

- **Kernel**: 10K lines of Objective Caml
- **Tactics**: 100K lines of Objective Caml and Coq tactic language, outputing a proof term.
- **Libraries of checked proof developments and tactics**, Academic as well as industrial users.
- **User’s group, hotline, website, LGPL licence.**
The proof assistant Coq

- Kernel: 10K lines of Objective Caml
- Tactics: 100K lines of Objective Caml and Coq tactic language, outputing a proof term.
- Libraries of checked proof developments and tactics,
- Academic as well as industrial users.
- User’s group, hotline, website, LGPL licence.
The proof assistant Coq

- Kernel: 10K lines of Objective Caml
- Tactics: 100K lines of Objective Caml and Coq tactic language, outputing a proof term.
- Libraries of checked proof developments and tactics,
- Academic as well as industrial users.
- User’s group, hotline, website, LGPL licence.

Read the Coq primer and user’s manual

Load the platform suited to your application

- **Calife:** timed automata (telecommunications)
- **Why:** annotated imperative programs translated into functional programs + verification conditions
- **Krakatoa:** JAVA/JAVACARDS programs
- **Caduceus:** prototype platform for C programs

Build your own platform otherwise
Using Coq

- Load Coq from [http://coq.inria.fr](http://coq.inria.fr)
- Read the Coq primer and user’s manual
- Load the platform suited to your application
  - **Calife**: timed automata (telecommunications)
  - **Why**: annotated imperative programs translated into functional programs + verification conditions
  - **Krakatoa**: JAVA/JAVACARDS programs
  - **Caduceus**: prototype platform for C programs
- Build your own platform otherwise
Using Coq

- Read the Coq primer and user’s manual
- Load the platform suited to your application
  - **Calife**: timed automata (telecommunications)
  - **Why**: annotated imperative programs translated into functional programs + verification conditions
  - **Krakatoa**: JAVA/JAVACARDS programs
  - **Caduceus**: prototype platform for C programs
- Build your own platform otherwise
Using Coq

- Read the Coq primer and user’s manual
- Load the platform suited to your application
- **Calife**: timed automata (telecommunications)
  - Why: annotated imperative programs translated into functional programs + verification conditions
- **Krakatoa**: JAVA/JAVACARDS programs
- **Caduceus**: prototype platform for C programs
- Build your own platform otherwise
Using Coq

- Read the Coq primer and user’s manual
- Load the platform suited to your application
  - **Calife:** timed automata (telecommunications)
  - **Why:** annotated imperative programs translated into functional programs + verification conditions
  - **Krakatoa:** JAVA/JAVACARDS programs
  - **Caduceus:** prototype platform for C programs
- Build your own platform otherwise
Using Coq

- Read the Coq primer and user’s manual
- Load the platform suited to your application
  - **Calife**: timed automata (telecommunications)
  - **Why**: annotated imperative programs translated into functional programs + verification conditions
  - **Krakatoa**: JAVA/JAVACARDS programs
  - **Caduceus**: prototype platform for C programs
- Build your own platform otherwise
Using Coq

- Load Coq from [http://coq.inria.fr](http://coq.inria.fr)
- Read the Coq primer and user’s manual
- Load the platform suited to your application
  - **Calife**: timed automata (telecommunications)
  - **Why**: annotated imperative programs translated into functional programs + verification conditions
  - **Krakatoa**: JAVA/JAVACARDS programs
  - **Caduceus**: prototype platform for C programs
- Build your own platform otherwise
Using Coq

- Load Coq from [http://coq.inria.fr](http://coq.inria.fr)
- Read the Coq primer and user’s manual
- Load the platform suited to your application
  - **Calife:** timed automata (telecommunications)
  - **Why:** annotated imperative programs translated into functional programs + verification conditions
  - **Krakatoa:** JAVA/JAVACARDS programs
  - **Caduceus:** prototype platform for C programs
- Build your own platform otherwise
XML-based input format for timed automata

- Interactive graphic support
- Graphic simulation tools
- Testing tools
- Code generators for Coq, Chronos, Hytech, and Prism
- Applications to telecommunication protocols: ABR, PGM, PIM, CSMA/CA
- Funded by RNRT, RNTL and France Telecom
- XML-based input format for timed automata
- Interactive graphic support
  - Graphic simulation tools
  - Testing tools
  - Code generators for Coq, Chronos, Hytech, and Prism
- Applications to telecommunication protocols: ABR, PGM, PIM, CSMA/CA
- Funded by RNRT, RNTL and France Telecom
Calife

- XML-based input format for timed automata
- Interactive graphic support
- Graphic simulation tools
  - Testing tools
  - Code generators for Coq, Chronos, Hytech, and Prism
  - Applications to telecommunication protocols: ABR, PGM, PIM, CSMA/CA
  - Funded by RNRT, RNTL, and France Telecom
XML-based input format for timed automata
Interactive graphic support
Graphic simulation tools
Testing tools
Code generators for Coq, Chronos, Hytech, and Prism
Applications to telecommunication protocols: ABR, PGM, PIM, CSMA/CA
Funded by RNRT, RNTL and France Telecom
Calife

- XML-based input format for timed automata
- Interactive graphic support
- Graphic simulation tools
- Testing tools
- Code generators for Coq, Chronos, Hytech, and Prism
- Applications to telecommunication protocols: ABR, PGM, PIM, CSMA/CA
- Funded by RNRT, RNTL and France Telecom
XML-based input format for timed automata
Interactive graphic support
Graphic simulation tools
Testing tools
Code generators for Coq, Chronos, Hytech, and Prism
Applications to telecommunication protocols: ABR, PGM, PIM, CSMA/CA
Funded by RNRT, RNTL and France Telecom
XML-based input format for timed automata
Interactive graphic support
Graphic simulation tools
Testing tools
Code generators for Coq, Chronos, Hytech, and Prism
Applications to telecommunication protocols: ABR, PGM, PIM, CSMA/CA
Funded by RNRT, RNTL and France Telecom
For JAVA/JAVACARDS programs

- Trusted Logics: security properties of cryptographic protocols: highest level of security for their methodology
- Schlumberger: security properties of their ATM, an entire model proved in Coq, over 500K lines of Coq
- Few interactions with both companies
For JAVA/JAVACARDS programs

- Trusted Logics: security properties of cryptographic protocols: highest level of security for their methodology

- Schlumberger: security properties of their ATM, an entire model proved in Coq, over 500K lines of Coq

- Few interactions with both companies
For JAVA/JAVACARDS programs

- Trusted Logics: security properties of cryptographic protocols: highest level of security for their methodology

- Schlumberger: security properties of their ATM, an entire model proved in Coq, over 500K lines of Coq

- Few interactions with both companies
For JAVA/JAVACARDS programs

- Trusted Logics: security properties of cryptographic protocols: highest level of security for their methodology
- Schlumberger: security properties of their ATM, an entire model proved in Coq, over 500K lines of Coq
- Few interactions with both companies
Current developments

- Verification of probabilistic statements about deterministic processes
- Specification and verification of probabilistic protocols
- Extend Grégoire’s abstract machine for handling rewriting
- Small proof engines and their combination
- Extraction of complexity information from proofs
- More experiments
Current developments

- Verification of probabilistic statements about deterministic processes
- Specification and verification of probabilistic protocols
- Extend Grégoire’s abstract machine for handling rewriting
- Small proof engines and their combination
- Extraction of complexity information from proofs
- More experiments
Current developments

- Verification of probabilistic statements about deterministic processes
- Specification and verification of probabilistic protocols
- Extend Grégoire’s abstract machine for handling rewriting
- Small proof engines and their combination
- Extraction of complexity information from proofs
- More experiments
Verification of probabilistic statements about deterministic processes

Specification and verification of probabilistic protocols

Extend Grégoire’s abstract machine for handling rewriting

Small proof engines and their combination

Extraction of complexity information from proofs

More experiments
Current developments

- Verification of probabilistic statements about deterministic processes
- Specification and verification of probabilistic protocols
- Extend Grégoire’s abstract machine for handling rewriting
- Small proof engines and their combination
- Extraction of complexity information from proofs
- More experiments
Current developments

- Verification of probabilistic statements about deterministic processes
- Specification and verification of probabilistic protocols
- Extend Grégoire’s abstract machine for handling rewriting
- Small proof engines and their combination
- Extraction of complexity information from proofs
- More experiments
Conclusion

- Proof assistants are very powerful specification languages
- Proof assistants should be at the heart of any verification tool
- Proof assistants should incorporate decision procedures in a transparent way
- Proof assistants are hard to use without dedicated platforms
- Software, unlike theorems, has a short life time, but may involve human’s life, money, or image.
- Current market is very small (electronic commerce), but will grow slowly (critical software).
Conclusion

- Proof assistants are very powerful specification languages
- Proof assistants should be at the heart of any verification tool
- Proof assistants should incorporate decision procedures in a transparent way
- Proof assistants are hard to use without dedicated platforms
- Software, unlike theorems, has a short lifetime, but may involve human’s life, money, or image.
- Current market is very small (electronic commerce), but will grow slowly (critical software).
Conclusion

- Proof assistants are very powerful specification languages
- Proof assistants should be at the heart of any verification tool
- Proof assistants should incorporate decision procedures in a transparent way
- Proof assistants are hard to use without dedicated platforms
- Software, unlike theorems, has a short lifetime, but may involve human’s life, money, or image.
- Current market is very small (electronic commerce), but will grow slowly (critical software).
Conclusion

- Proof assistants are very powerful specification languages
- Proof assistants should be at the heart of any verification tool
- Proof assistants should incorporate decision procedures in a transparent way
- Proof assistants are hard to use without dedicated platforms
- Software, unlike theorems, has a short lifetime, but may involve human’s life, money, or image.
- Current market is very small (electronic commerce), but will grow slowly (critical software).
Conclusion

- Proof assistants are very powerful specification languages
- Proof assistants should be at the heart of any verification tool
- Proof assistants should incorporate decision procedures in a transparent way
- Proof assistants are hard to use without dedicated platforms
- Software, unlike theorems, has a short life time, but may involve human’s life, money, or image.
- Current market is very small (electronic commerce), but will grow slowly (critical software).
Conclusion

- Proof assistants are very powerful specification languages
- Proof assistants should be at the heart of any verification tool
- Proof assistants should incorporate decision procedures in a transparent way
- Proof assistants are hard to use without dedicated platforms
- Software, unlike theorems, has a short life time, but may involve human’s life, money, or image.
- Current market is very small (electronic commerce), but will grow slowly (critical software).
Acknowledgments to

G. Huet, T. Coquand, C. Paulin, G. Dowek
for their vision and early implementations;
Barras, Filliatre, Grégoire, Herbelin,
Blanqui, Chrząszcz, Monate, Strub
for their theoretical and software contributions;
LogiCal for its extreme dedication to Coq;
Trusted Logics for putting forward their use of
Coq and Why;
France-Telecom, EADS, Thalès for funding us;
INRIA, CNRS for their continuous support.
Outline
Theorems and proofs in mathematics
Four celebrated examples
Deductions and Computations
Proof Assistants
Coq
Conclusion

Jean-Pierre Jouannaud École Polytechnique 91400 Palaiseau, France