Theorem Proving languages for Verification

Jean-Pierre Jouannaud
École Polytechnique
91400 Palaiseau, France

email: jouannaud@lix.polytechnique.fr
http: //w^3.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.

ÉCOLE
POLYTECHNIQUE

Outline

1. Verification = Specification + Deduction + Computation + Abstraction
2. Logical foundations
3. Proof Assistants
4. Coq
5. Current developments and Conclusions

Verification

- Given a system to be analyzed,
- 1. elaborate a model of the system.
- 2. Test some liveness property
- 2. Verify some safety property
- 2. Prove a general logical property
- 3. When available ressources do not suffice, abstract the model and try again
- 4. When the task is finally completed, prove the abstractions used.

Real need of powerful, secure, interactive tools

Verification

- Given a system to be analyzed,
- 1. elaborate a model of the system.
- 2. Test some liveness property
- 2. Verify some safety property
- 2. Prove a general logical property
- 3. When available ressources do not suffice, abstract the model and try again
- 4. When the task is finally completed, prove the abstractions used.

Real need of powerful, secure, interactive tools

Verification

- Given a system to be analyzed,
- 1. elaborate a model of the system.
- 2. Test some liveness property
- 2. Verify some safety property
- 2. Prove a general logical property
- 3. When available ressources do not suffice, abstract the model and try again
- 4. When the task is finally completed, prove the abstractions used.

Real need of powerful, secure, interactive tools

Verification

- Given a system to be analyzed,
- 1. elaborate a model of the system.
- 2. Test some liveness property
- 2. Verify some safety property
- 2. Prove a general logical property
- 3. When available ressources do not suffice, abstract the model and try again
- 4. When the task is finally completed, prove the abstractions used.

Real need of powerful, secure, interactive tools

Verification

- Given a system to be analyzed,
- 1. elaborate a model of the system.
- 2. Test some liveness property
- 2. Verify some safety property
- 2. Prove a general logical property
- 3. When available ressources do not suffice, abstract the model and try again
- 4. When the task is finally completed, prove the abstractions used.

Real need of powerful, secure, interactive tools

Verification

- Given a system to be analyzed,
- 1. elaborate a model of the system.
- 2. Test some liveness property
- 2. Verify some safety property
- 2. Prove a general logical property
- 3. When available ressources do not suffice, abstract the model and try again
- 4. When the task is finally completed, prove the abstractions used.

Real need of powerful, secure, interactive tools

Verification

- Given a system to be analyzed,
- 1. elaborate a model of the system.
- 2. Test some liveness property
- 2. Verify some safety property
- 2. Prove a general logical property
- 3. When available ressources do not suffice, abstract the model and try again
- 4. When the task is finally completed, prove the abstractions used.

Real need of powerful, secure, interactive tools

Verification

- Given a system to be analyzed,
- 1. elaborate a model of the system.
- 2. Test some liveness property
- 2. Verify some safety property
- 2. Prove a general logical property
- 3. When available ressources do not suffice, abstract the model and try again
- 4. When the task is finally completed, prove the abstractions used.

Real need of powerful, secure, interactive tools

Verification

- Given a system to be analyzed,
- 1. elaborate a model of the system.
- 2. Test some liveness property
- 2. Verify some safety property
- 2. Prove a general logical property
- 3. When available ressources do not suffice, abstract the model and try again
- 4. When the task is finally completed, prove the abstractions used.

Real need of powerful, secure, interactive tools

Logical Foundations

Hilbert's program:
automate mathematical reasonning

Undecidability of Proof-Search

- Given: a logical statement.
- Question: is it a theorem?
- Gödel: there is no program able to answer this question.

Undecidability of Proof-Search

- Given: a logical statement.
- Question: is it a theorem?
- Gödel: there is no program able to answer this question.

Undecidability of Proof-Search

- Given: a logical statement.
- Question: is it a theorem?
- Gödel: there is no program able to answer this question.

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

- 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

- 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.

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 garanteed 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 garanteed 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 garanteed 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 garanteed 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 garanteed to construct correct proofs,
  - performs automatically in case of a decidable verification problem.

Computations and Deductions

- 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: $\Gamma \vdash P \longrightarrow Q$

conversion: $$\frac{\Gamma \vdash p : P \qquad \Gamma \vdash P \longrightarrow Q}{\Gamma \vdash p : Q}$$

Computations and Deductions

- 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: $\dfrac{\Gamma \vdash p : P \qquad \Gamma \vdash P \rightarrow Q}{\Gamma \vdash p : Q}$

- 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: $\Gamma \vdash P \longrightarrow Q$

conversion: $$\frac{\Gamma \vdash p : P \qquad \Gamma \vdash P \longrightarrow Q}{\Gamma \vdash p : Q}$$

Example: $2 + 2$ is even

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

$$\frac{\Gamma \vdash ?? : E(4) \qquad \Gamma \vdash E(2+2) \longrightarrow E(4)}{\Gamma \vdash ?? : E(2+2)}$$

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:

$$\frac{\Gamma \vdash ?? : E(4) \qquad \Gamma \vdash E(2+2) \longrightarrow E(4)}{\Gamma \vdash ?? : E(2+2)}$$

Example: $2+2$ is even

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

$$\frac{\Gamma \vdash ?? : E(4) \qquad \Gamma \vdash E(2+2) \longrightarrow E(4)}{\Gamma \vdash ?? : E(2+2)}$$

Example: $2+2$ is even

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

$$\frac{\Gamma \vdash \ ?? : E(4) \qquad \Gamma \vdash E(2+2) \longrightarrow E(4)}{\Gamma \vdash \ ?? : E(2+2)}$$

Deduction:

$$\dfrac{\dfrac{\cdots}{\vdash\ q(0,p) : E(2)} \qquad \dfrac{\overline{\vdash\ q : \forall x.E(x) \implies E(s(s(x)))}}{\vdash\ q(2) : E(2) \implies E(4)}}{\vdash\ q(2, q(0,p)) : E(4)}$$

$$\dfrac{\dfrac{}{\vdash\ p : E(0)} \qquad \dfrac{\overline{q : \ \vdash\ \forall x.E(x) \implies E(s(s(x)))}}{\vdash\ q(0) : E(0) \implies E(2)}}{\vdash\ q(0,p) : E(2)}$$

Proof checking

- 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 $A \longrightarrow B$.
- This algorithm constitutes the kernel of a proof assistant.

Proof checking

- 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 $A \longrightarrow B$.
- This algorithm constitutes the kernel of a proof assistant.

Proof checking

- 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 $A \longrightarrow B$.
- This algorithm constitutes the kernel of a proof assistant.

Proof Assistants

De Bruijn's program

What is a 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.

What is a 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.

What is a 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.

What is a 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.

What is a 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.

What is a 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.

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
  Chrzaczsz
  Compiler of Grégoire
- Comes with
  a code extracter by Letouzey
  a tactic language of Delahaye
  a graphic proof interface of Monate
- Prototype version includes
  rewriting 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
  Chrzaczsz
  Compiler of Grégoire
- Comes with
  a code extracter by Letouzey
  a tactic language of Delahaye
  a graphic proof interface of Monate
- Prototype version includes
  rewriting 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
  Chrzaczsz
  Compiler of Grégoire
- Comes with
  a code extracter by Letouzey
  a tactic language of Delahaye
  a graphic proof interface of Monate
- Prototype version includes
  rewriting by Blanqui
  small proof engines by Strub

Coq's interactive module system

```
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.
```

Coq's interactive module system

```
Module Type Orderedtype.
Parameter t : Set.
Parameter eq : t → t → Prop.
Paremeter eq_refl : (x:t)(eq x x ).
Paremeter eq_sym : (x,y:t) (eq x y) → (eq y x).
Paremeter eq_trans : (x,y,z:t) (eq x y) → (eq y z) → (
Paremeter lt_trans : (x,y,z:t) (lt x y) → (lt y z) → (lt x
Paremeter lt_not_eq : (x,y:t) (lt x y) → ¬ (eq x y).
Parameter compare : (x,y:t) (Comp lt eq x y).
End OrderedType.
```

Coq's interactive module system

Inductice Comp [X:Set; lt,eq:X$\longrightarrow$ X $\longrightarrow$ Prop; x,y:X] :

| Lt : (lt x y) $\longrightarrow$ (Comp lt eq x y)

| Eq : (eq x y) $\longrightarrow$ (Comp lt eq x y)

| Gt : (lt y x) $\longrightarrow$ (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.

Using Coq

- Load Coq from 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
- 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
- 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
- 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
- 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
- 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
- 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
- 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

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

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

Krakatoa

- For JAVA/JAVACARDS programs
- Trusted Logics: security properties of crytographic 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

Krakatoa

- For JAVA/JAVACARDS programs
- Trusted Logics: security properties of crytographic 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

Krakatoa

- For JAVA/JAVACARDS programs
- Trusted Logics: security properties of crytographic 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

Krakatoa

- For JAVA/JAVACARDS programs
- Trusted Logics: security properties of crytographic 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 and Conclusions

Current developments

- Verification of probabilistic statements about deterministic processes
- Specification and verification of probabilistic protocols
- Compiler for 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
- Compiler for 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
- Compiler for 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
- Compiler for 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
- Compiler for 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
- Compiler for 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 incoporate decision procedures in a transparent way
- Proof assistants are hard to use without dedicated platforms
- Market is very small (electronic commerce)

Conclusion

- Proof assistants are very powerful specification languages
- Proof assistants should be at the heart of any verification tool
- Proof assistants should incoporate decision procedures in a transparent way
- Proof assistants are hard to use without dedicated platforms
- Market is very small (electronic commerce)

Conclusion

- Proof assistants are very powerful specification languages
- Proof assistants should be at the heart of any verification tool
- Proof assistants should incoporate decision procedures in a transparent way
- Proof assistants are hard to use without dedicated platforms
- Market is very small (electronic commerce)

Conclusion

- Proof assistants are very powerful specification languages
- Proof assistants should be at the heart of any verification tool
- Proof assistants should incoporate decision procedures in a transparent way
- Proof assistants are hard to use without dedicated platforms
- Market is very small (electronic commerce)

Conclusion

- Proof assistants are very powerful specification languages
- Proof assistants should be at the heart of any verification tool
- Proof assistants should incoporate decision procedures in a transparent way
- Proof assistants are hard to use without dedicated platforms
- Market is very small (electronic commerce)

Acknowledgments to

G. Huet, T. Coquand, C. Paulin, G. Dowek
for their vision and early implementations;
Barras, Filliatre, Grégoire, Herbelin,
Blanqui, Chrzaczsz, 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.