Formal Methods Applied to the Implementation of Secure Software/Hardware using PVS

Mauricio Ayala-Rincón
Grupo de Teoria da Computação, Universidade de Brasília (UnB)
Short Course at ITIV - KIT

10-12 May 2010, Karlsruhe
Talk’s Plan

Motivation: generation of simple pieces of secure software/hardware
  PVS
  Case study: KB2D an algorithm for Detection and Resolution of Air Traffic Conflicts
  Case study: Formalisation of the Security of Cryptographic Protocols

Formal proofs
  Type Inference and Deductions
  Curry-Howard isomorphism - programs as proofs
  Proofs in the Prototype Verification System - PVS
  Programs versus demonstrations in PVS
  Formalisation of reconfigurable hardware - a simple example

Conclusions and Future Work
What is PVS?

The Prototype Verification System (PVS), developed by SRI International Computer Science Laboratory, is a interactive theorem prover which consists of

1. a specification language:
   - based on higher-order logic;
   - a type system based on Church’s simple theory of types augmented with subtypes and dependent types.

2. an interactive theorem prover:
   - based on sequent calculus; that is, goals in PVS are sequents of the form $\Gamma \vdash \Delta$, where $\Gamma$ and $\Delta$ are finite sequences of formulae, with the usual Gentzen semantics.
Motivation: generation of simple pieces of secure software/hardware

Case study: KB2D an algorithm for Detection and Resolution of Air Traffic Conflicts

KB2D [GnAR07] improves NIA/NASA’s KB3D
The Problem: Basic Definition and concepts

- **Avoidance Region**: circle centered in the aircraft.
- **Conflict**: two aircraft are said to be in conflict when their avoidance regions overlap.
The Problem: Basic definitions and concepts

Protected Zone: circle twice as big as the avoidance region.
The Problem: Basic definitions and concepts

- A conflict is the incursion of the \textit{ownship} in the \textit{intruder’s} protected zone.
Conflict Detection and Resolution Algorithm

- **KB3D** (Gilles Dowek, César Muñoz, and Alfons Geser)
  3-Dimensional conflict detection and resolution algorithm (CD&R) which allows either changes of
  - vertical speed only
  - horizontal speed only
  - heading only

  - KB2D combines changes of horizontal speed and of heading

- KB2D is a 2-Dimensional CD&R.
Conflict Detection and Resolution Algorithm

- **KB3D** (Gilles Dowek, César Muñoz, and Alfons Geser)

  3-Dimensional conflict detection and resolution algorithm (CD&R) which allows either changes of

  - vertical speed only

  - horizontal speed only

  - heading only

- KB2D combines changes of horizontal speed and of heading

- **KB2D** is a 2-Dimensional CD&R.
**KB2D: Inputs**

- **s**: *ownership’s relative position*
- **vo**: *ownership’s velocity*
- **vi**: *intruder’s velocity*
- **tpp**: Required Time of Arrival
Motivation: generation of simple pieces of secure software/hardware

Case study: KB2D an algorithm for Detection and Resolution of Air Traffic Conflicts

KB2D: Outputs

- **vpo**: Resolution velocity
- **vppo**: Recovery velocity
- **tp**: Time of switch
The Algorithm (Geometric Solution)
The Algorithm (Geometric Solution)

1. Ownship’s relative velocity: \( v \)
The Algorithm (Geometric Solution)

1. Ownship’s relative velocity: \( v \)
2. Tangent points: \( Q_1 \) and \( Q_{-1} \)
The Algorithm (Geometric Solution)

1. Ownship’s relative velocity: \( v \)
2. Tangent points: \( Q_1 \) and \( Q_{-1} \)
3. Relative resolution velocities: \( v_{p1} \) and \( v_{p-1} \)
The Algorithm (Geometric Solution)

1. Ownship’s relative velocity: \( v \)
2. Tangent points: \( Q_1 \) and \( Q_{-1} \)
3. Relative resolution velocities: \( v_{p1} \) and \( v_{p-1} \)
4. Absolute resolution velocities: \( v_{po1} \) and \( v_{po-1} \)

\[ DO = (0,0) \]
The Algorithm (Geometric Solution)

1. Ownship’s relative velocity: $v$
2. Tangent points: $Q_1$ and $Q_{-1}$
3. Relative resolution velocities: $vp_1$ and $vp_{-1}$
4. Absolute resolution velocities: $vpo_1$ and $vpo_{-1}$
Computing the tangent points

\[
\begin{align*}
    sx \cdot Qx + sy \cdot Qy &= D^2 \\
    Qx^2 + Qy^2 &= D^2
\end{align*}
\]
Computing the relative resolution velocities

\[\begin{align*}
vp & = k \cdot (Q - s) \\
vp \cdot (vp - v) & = 0
\end{align*}\]
Geometric and Analytic Solution (Recovery)

\[ s + tp \cdot vp + (tpp - tp) \cdot vpp = sp = s + tpp \cdot v \]

\[ \Rightarrow vpp = \frac{1}{tpp - tp}(tpp \cdot v - tp \cdot vp) \]
Optimality (2D)

Theorem

The relative resolution velocity is optimal; i.e., it requires the least effort, among all vectors on the whole universe of possible solutions on the same side of the circle.
Coordination

Let $A$ and $B$ be two conflicting aircrafts.
Motivation: generation of simple pieces of secure software/hardware

Case study: KB2D an algorithm for Detection and Resolution of Air Traffic Conflicts

Coordination

- The relative positions computed by each aircraft are opposite.
- The time of loss of separation is the same for both aircrafts.
Motivation: generation of simple pieces of secure software/hardware

Case study: KB2D an algorithm for Detection and Resolution of Air Traffic Conflicts

Coordination

aircraft A

vpa

va

aircraft B

Protected zone
Cooperation

Lemma

For all $\epsilon = \pm 1$, $v_{pa}$ and $v_{pb}$ are parallel.
Motivation: generation of simple pieces of secure software/hardware

Case study: KB2D an algorithm for Detection and Resolution of Air Traffic Conflicts

Coordination

Lemma

For all $\epsilon = \pm 1$, $v_{pa}$ and $v_{pb}$ are parallel.
Formal Verification (An Example)

**Theorem (kb2d_correct)**

For all \( s, v = vo - vi, T > 0, D > 0, vp, vpo, \epsilon s = \pm 1, \)
conflict\((s, v, T) \) and
\( s_x^2 + s_y^2 > D^2 \) and
\( vpo = kb2d(sx, sy, vox, voy, vix, viy, \epsilon s) \) and
\( vp = vpo - vi \) and \( vpo \neq 0 \)
implies
separation\((s, vp)\).
Formal methods in cryptography

- Why proving mathematically security requirements?
- Authentication protocol of Needham-Schroeder
  - was considered during 17 years to be secure.
  - but Lowe detected a “man-in-the-middle” vulnerability in this protocol [Low95, Low96].

- Example: formalisation of the security of the Dolev-Yao two-party cascade protocol [DY83].
  - To be published 6th Computability in Europe [NNdMAR10].
Cryptographic operations over monoids

- Any user $u \in U$ owns $E_u$ and $D_u$.
  - $E = \{E_u \mid u \in U\}$
  - $D = \{D_u \mid u \in U\}$
- $\Sigma = E \cup D$
- $\Sigma^*$ set of words over $\Sigma$.
- Monoid freely generated by $\Sigma$ and congruences:
  \[ E_u D_u = \lambda \quad D_u E_u = \lambda, \quad \forall u \in U \]  
  \[ (1) \]
- $E_u(D_u(M)) = D_u(E_u(M)) = M, \forall M$ plain text.
Formalisation: normalisation property

- Rewriting rules:
  \[
  E_u D_u \rightarrow \lambda \quad D_u E_u \rightarrow \lambda, \quad \forall u \in U
  \]  
  (2)

- Canonical form: \( \forall \delta \in \Sigma^*, \bar{\delta} \) is such that
  \[
  \delta \rightarrow^* \bar{\delta}
  \]
  and \( \bar{\delta} \) is irreducible.

- \( \forall u \in U, E_u^c = D_u \) e \( D_u^c = E_u \).
Specification of the *Protocol Step*

Definition (Protocol Step: $\alpha\beta : U \times U \to \Sigma^*$)

\[\forall x, y \in U \mid x \neq y : \]

1. $\alpha\beta(x, y) \neq \lambda$
2. $\alpha\beta(x, y) = \alpha\beta(x, y)$
3. $\alpha\beta(x, y) \in \Phi(x, y)^*$\[\Phi(x, y) = \{D_x, E_x, E_y\}\]
4. $\forall u, v \in U :$
   4.1. $|\alpha\beta(x, y)| = |\alpha\beta(u, v)|$
   4.2. $\forall 0 \leq j < |\alpha\beta(x, y)| :$
      4.2.1. $\alpha\beta(x, y)[j] = E_x \iff \alpha\beta(u, v)[j] = E_u$
      4.2.2. $\alpha\beta(x, y)[j] = E_y \iff \alpha\beta(u, v)[j] = E_v$
      4.2.3. $\alpha\beta(x, y)[j] = D_x \iff \alpha\beta(u, v)[j] = D_u$
      4.2.4. $\alpha\beta(x, y)[j] = D_y \iff \alpha\beta(u, v)[j] = D_v$
PVS specification of the Protocol Step

PVS Protocol Step

alphabeta_welldef?(ab : alphabeta, x, y : U) : bool =
ab(x,y)'length > 0 AND
normalseq?(ab(x,y)) AND
( FORALL(j : nat | j < ab(x,y)'length) :
  member(ab(x,y)(j),validSetxy(x,y)) ) AND
abUsers?(ab, x, y)

Protocol Step is the same for each pair of users

abUsers?(ab : alphabeta, x, y : U) : bool =
FORALL(u, v : U):
  ab(x,y)'length = ab(u,v)'length AND
  FORALL(i : nat | i < ab(x,y)'length) :
    (user(ab(x,y)(i)) = x OR user(ab(x,y)(i)) = y) AND
    (crTyp(ab(x,y)(i)) = crTyp(ab(u,v)(i))) AND
    (user(ab(x,y)(i)) = x IFF user(ab(u,v)(i)) = u) AND
    (user(ab(x,y)(i)) = y IFF user(ab(u,v)(i)) = v)
Specification of *Cascade Protocols*

- Nonempty sequence of protocol steps, \( \forall x, y \in U \).
- Protocol steps alternate between \( x \) and \( y \).

**Definition (Cascade Protocol)**

\[ \forall 0 \leq i < |P| \ e \ \forall x, y \in U: \]

1. \( P_i(x, y) \), *for i even*
2. \( P_i(y, x) \), *for i odd*
Functionality - *Cascade Protocol*

- $x \rightarrow y$ represents submission of message from $x$ to $y$, $x, y \in U$.

**Communication between users $x, y \in U$**

- $x \rightarrow y : P_0 M = \alpha \beta_0(x, y) M$
- $y \rightarrow x : P_1 P_0 M = \alpha \beta_1(y, x) \alpha \beta_0(x, y) M$
- $\vdots$
- $x \rightarrow y : P_{|P| - 1} \ldots P_0 M = \alpha \beta_{|P| - 1}(x, y) \ldots \alpha \beta_0(x, y) M$, if $|P| > 2$ odd
  - or
- $y \rightarrow x : P_{|P| - 1} \ldots P_0 M = \alpha \beta_{|P| - 1}(y, x) \ldots \alpha \beta_0(x, y) M$, if $|P| > 2$ even
Specification of the adversary Admissible Language

Definition (Adversary Admissible Language)

\[(\Sigma_1^*(z) \cup \Sigma_2)^*, \text{ where:} \]

\[\Sigma_1(z) = E \cup \{D_z\}, \text{ and} \]
\[\Sigma_2 = \{P_i(x, y) \mid 1 \leq i < |P| \text{ and } x, y \in U, x \neq y\}\]

- An adversary \(z\) can:
  - Observe all the traffic in the communication net;
  - Do all things an honest user can do;
  - Create, intercept, destroy and modify messages.
  - Supplant other users.

- But \(z\) is limited by cryptographic primitives.
Definition *secure cascade protocol*

**Definition (Secure Cascade Protocol)**

*P* is secure whenever for all *x*, *y*, *z* ∈ *U*, ∀ *γ* ∈ (*Σ₁*(*z*) ∪ *Σ₂)* and 0 ≤ *i* < |*P*|, it holds:

\[
\gamma P_i \ldots P_0 \neq \lambda
\]
Security characterisation: *Initial Condition of Security*

**Definition (Initial Condition of Security)**

\[ \forall x, y \in U:\]

\[ P_0(x, y) \cap \{E_x, E_y\} \neq \phi \]

Without this condition, \( P_0(x, y) = D_x^k \) \((k \in \mathbb{N}^*)\).
Security characterisation: *Balancing Property*

**Definition (Balancing Property (BP))**

Let $\delta \in \Sigma^*$. $\delta$ satisfies BP w.r.t. $z \in U$, whenever:

$$\exists 0 \leq i < |\delta| : \delta_i = D_z \implies \exists 0 \leq j < |\delta| : \delta_j = E_z$$
Balancing Property for a cascade protocol $P$

Definition (BP Cascade Protocol)

- A cascade protocol $P$ is balanced whenever:
  
  $\forall x, y \in U$ and $\forall 0 < i < |P|$: 
  
  $P_i(x, y)$ satisfies BP w.r.t. $x$, if $i$ even 
  $P_i(y, x)$ satisfies BP w.r.t. $y$, if $i$ odd 

- Example:
  
  Let $P_2$ the third step of a cascade protocol $P$, such that $P_2(x, y) = E_y D_x E_y$, then, $P$ is not balanced.
Formalisation of security for cascade protocols

Theorem (Characterisation of security)
A cascade protocol $P$ is secure iff,

(i) it satisfies the initial security property and
(ii) it is balanced.

Formalisation in PVS

```
theorem1 : THEOREM FORALL (prot : welldefined_protocol,
                      x : U, y : U | x /= y, z : U | z /= x AND z /= y) :
      secure_protocol?(prot, x, y, z) IFF
      ( alpha0ContainsE?(prot, x, y) AND balanced_cascade_protocol?(prot) )
```
**Sketch of the formalisation**

- Let $P$ be a cascade protocol.

- **Necessity**, by *contraposition*:
  
  $$\neg(i) \lor \neg(ii) \implies P \text{ insecure}.$$ 

- **Sufficiency**, by *contradiction*:
  
  $$(i) \land (ii) \land P \text{ insecure} \implies P \text{ secure}.$$ 

**Theorem of Security**

A cascade protocol $P$ is secure iff

- (i) it satisfies the security initial condition
- (ii) it is balanced.

- **Sufficiency**: one assumes, by contradiction, that $P$ is insecure.
- PVS formalisation divided in 9 sub-theories.
Motivation: generation of simple pieces of secure software/hardware

Case study: Formalisation of the Security of Cryptographic Protocols

Structure of the PVS formalisation
Necessity

- **A)** \(\neg(i) \implies P \text{ insecure} \)
  - \(P_0(x, y) = D_x^k (k \in \mathbb{N}^*)\).
  - \(\gamma = E_x^k\), so that \(\gamma P_0 = \lambda\)

- **B)** \(\neg(ii) \implies P \text{ insecure} \)
  - By lemma of *extraction of private operator*:
    - \(u, v \in U \mid u \neq v\)
    - Step protocol \(\alpha\beta(u, v)\) unbalanced.
    - \(\exists \tau_1, \tau_2 \in \Sigma_1^*(v)\), such that \(\tau_1 \alpha \beta(u, v) \tau_2 = D_u\).
  - By induction in the length of \(P_0(x, y) = \{D_x, E_x, E_y\}P_0(x, y)_{[1,|P_0|−1]}\)
    - **Induction step**: eliminate \(D_x\) applying \(E_x \in \Sigma_1^*(z)\) and eliminate \(\{E_x, E_y\}\) applying lemma above.
Sufficiency

\[ (i) \land (ii) \land P \text{ insecure} \implies P \text{ secure} \]

Lemma (Admissible language is balanced)

Let \( P \) be a balanced cascade protocol. For any \( z \in U \), \( \forall \gamma \in (\Sigma_1^*(z) \cup \Sigma_2)^* \) and \( \forall a \in U \mid a \neq z \), it holds: \( \widetilde{\gamma} \) satisfies BP w.r.t. \( a \).
Sufficiency

- Since $P$ is insecure, $\exists \gamma \in (\Sigma_1^*(z) \cup \Sigma_2)^*$ such that $\overline{\gamma}^c = P_0(x, y)$.

- Contradiction is obtained considering $\overline{\gamma}^c = P_0(x, y)$.

  - $E_y \in P_0(x, y)$:
    - Since $\overline{\gamma}^c = P_0(x, y)$, then $D_y \in \gamma$.
    - $\overline{\gamma}$ is balanced: $E_y \in \overline{\gamma}$
    - Thus, $D_y \in P_0(x, y)$. CONTRADICTION.

  - $E_y \notin P_0(x, y)$:
    - Since $P_0(x, y)$ balanced, then $D_y \notin P_0(x, y)$.
    - $P_0(x, y) = E_x^k (k \in \mathbb{N}^*)$
    - Thus, $\overline{\gamma} = D_x^k$. CONTRADICTION, since $\overline{\gamma}$ satisfies BP w.r.t. $x$. 
Types

- Discrimination of classes of objects
  - Implicitly used in intuitive systems
    - Euclid *Elements*
  - Necessity of an explicit definition for abstract systems
Types

- Discrimination of classes of objects
- **Implicitly** used in intuitive systems
  - Euclid *Elements*
- Necessity of an *explicit* definition for abstract systems
Types

- Discrimination of classes of objects
- Implicitly used in intuitive systems
  - Euclid *Elements*
- Necessity of an *explicit* definition for abstract systems
History of types

- Treatment of paradoxes and inconsistencies in the formalization of mathematics:
  - Auto-reference, auto-reproduction
- Simple Types in the $\lambda$-calculus [Alonzo Church 1940]
- Implicit Types [Haskell Curry 1958]
- Type-free languages: LISP [John McCarthy 1956-9]
- Typed languages: Fortran, Algol,…
- Languages with types à la Curry: ML [Robin Milner 1980]
Simple Types

SYNTAX

TYPES
\[ A ::= K \mid A \to B \]

TERMS
\[ a ::= x \mid (a \ a) \mid \lambda x : B . a \]

- A \( \lambda \)-term \( a \) has type \( B \), denoted \( a : B \)

- Context \( \Gamma = \{ x_1 : A_1 , x_2 : A_2 , \ldots , x_n : A_n \} \)

- A \( \lambda \)-term \( a \) has type \( B \) under context \( \Gamma \)

\[ \Gamma \vdash a : B \]

Type Judgment
Simple Types

SYNTAX

TYPES  \( A ::= K \mid A \rightarrow B \)
TERMS  \( a ::= x \mid (a\ a) \mid \lambda x : B. a \)

- A \( \lambda \)-term \( a \) has type \( B \), denoted \( a : B \)
- Context \( \Gamma = \{ x_1 : A_1, x_2 : A_2, \ldots, x_n : A_n \} \)
- A \( \lambda \)-term \( a \) has type \( B \) under context \( \Gamma \)

\[ \Gamma \vdash a : B \]

Type Judgment
Simple Types

SYNTAX

TYPES

\[ A ::= K \mid A \rightarrow B \]

TERMS

\[ a ::= x \mid (a \ a) \mid \lambda x : B . a \]

- A \( \lambda \)-term \( a \) has type \( B \), denoted \( a : B \)

- Context \( \Gamma = \{ x_1 : A_1, x_2 : A_2, \ldots, x_n : A_n \} \)

- A \( \lambda \)-term \( a \) has type \( B \) under context \( \Gamma \)

\[ \Gamma \vdash a : B \]

Type Judgment
Simple Types

SYNTAX

TYPES \[ A ::= K \ | \ A \rightarrow B \]
TERMS \[ a ::= x \ | \ (a \ a) \ | \ \lambda x : B \ a \]

- A \( \lambda \)-term \( a \) has type \( B \), denoted \( a : B \)
- Context \( \Gamma = \left\{ x_1 : A_1 , x_2 : A_2 , \ldots , x_n : A_n \right\} \)
- A \( \lambda \)-term \( a \) has type \( B \) under context \( \Gamma \)

\[ \Gamma \vdash a : B \]

Type Judgment
Simple Types

Examples

\[
\begin{align*}
(\lambda_x.x \quad \lambda_x.x) & \rightarrow_\beta \lambda_x.x \\
(\lambda_x.(x \ x) \quad \lambda_x.(x \ x)) & \rightarrow_\beta (\lambda_x.(x \ x) \quad \lambda_x.(x \ x))
\end{align*}
\]

Paradoxal Argumentation

Auto-aplication makes sense:

\[
(\lambda_x:A \rightarrow A.x \quad \lambda_x:A.x) \rightarrow_\beta \lambda_x:A.x
\]

Polymorphism!
Simple Types

Examples

\[
\begin{cases}
\lambda_x \cdot x \; \lambda_x \cdot x \rightarrow_\beta \lambda_x \cdot x & \text{auto-aplication} \\
\lambda_x \cdot (x \; x) \; \lambda_x \cdot (x \; x) \rightarrow_\beta \lambda_x \cdot (x \; x) \; \lambda_x \cdot (x \; x) & \text{auto-reproduction}
\end{cases}
\]

Paradoxal Argumentation

Auto-reproduction doesn’t make sense:

\[
\big((\lambda_{x: \tau_1} \cdot (x \; x) \; \lambda_{x: \tau_2} \cdot (x \; x)) \rightarrow_\beta \big(\lambda_{x: \tau_3} \cdot (x \; x) \; \lambda_{x: \tau_4} \cdot (x \; x)\big) \big)
\]

Acceptable term, but non typable!
$TA_\lambda$: the simply typed $\lambda$-calculus

\[
\begin{align*}
\frac{x \notin \Gamma}{x : A, \Gamma \vdash x : A} \quad (Start) & & \quad \frac{x \notin \Gamma \quad \Gamma \vdash a : B}{x : A, \Gamma \vdash a : B} \quad (Weak) \\
\frac{x : A, \Gamma \vdash a : B}{\Gamma \vdash \lambda x : A . a : A \rightarrow B} \quad (Abs) & & \quad \frac{\Gamma \vdash a : B \rightarrow A \quad \Gamma \vdash b : B}{\Gamma \vdash (a \; b) : A} \quad (App)
\end{align*}
\]

Table: $TA_\lambda$
Example: type inference (auto-aplication)

Example (Type inference (auto-aplication))

\[
\begin{align*}
\Gamma & \vdash (\lambda_{x:A \rightarrow A}.x \lambda_{x:A .x}) : A \rightarrow A \\
\vdash & \lambda_{x:A}.x : A \rightarrow A \\
\vdash & x : A \rightarrow A \vdash x : A \rightarrow A \\
\vdash & \lambda_{x:A}.x : A \rightarrow A \\
\vdash & x : A \vdash x : A \\
\vdash & \Gamma \vdash (\lambda_{x:A \rightarrow A}.x \lambda_{x:A .x}) : A \rightarrow A
\end{align*}
\]
Relevant problems in type theory

- **Verification**: given $M$ and $A$ determine whether there exists $\Gamma$ s.t. $\Gamma \vdash M : A$.
- **Inference**: given $M$ determine $\Gamma$ and $A$ s.t. $\Gamma \vdash M : A$.
- **Inhabitation**: given a type $A$. There exist *inhabitants* inside the context $\Gamma$ iff there exists a $\lambda$-term $M$ s.t. $\Gamma \vdash M : A$.
- **Subject reduction**: do preserve types all computations?
- **Principal Typing**: for all term $M$ there exists a *more general* typing $(\Gamma, A)$, s.t. $\Gamma \vdash M : A$. 
Revisiting relevant problems in type theory

- Type verification: are correct the designed types for the program?
- Type inference: Is the program correct?
- Existence of inhabitants: extraction of a program from a proof.
Relation between proofs and programs was detected by Haskell Curry [1934-1942], but was only applied until the 1960s by N.G. de Bruijn and William Howard.

**Type Theory** versus **Intuitionistic Logic**

Luitzen Egbertus Jan Brouwer [1920]

*Typing* rules from the simple typed \( \lambda \)-calculus correspond 1-1 to the deductive rules of the minimal intuitionistic logic: *typing* rules are logical rules decorated with typed \( \lambda \)-terms.
proofs as programs - Curry-Howard isomorphism

**Implicational intuitionistic logic**

**Implicational formulas** are built from *propositional variables* (denoted by $A, B, C, \ldots$) using only implication $\rightarrow$:

Thus, if $\sigma$ and $\tau$ are implicational formulas, then $(\sigma \rightarrow \tau)$ is also an implicational formula.
A judgment in the intuitionistic logic, written as $\Omega \vdash I A$, means that “$A$ is a logic consequence of $\Omega$”.

Deduction rules of the minimal intuitionistic logic

A formula $A$ is a \textit{tautology} if, and only if the judgment $\vdash I A$ is provable.
proofs as programs - Curry-Howard isomorphism

Example \((A \to ((A \to B) \to B))\) is a tautology

\[
\frac{A, A \to B \vdash A \to B}{(Axiom)} \quad \frac{A, A \to B \vdash A}{(Axiom)} \quad \frac{A, A \to B \vdash B}{(Elim)}
\]

\[
\frac{A, A \to B \vdash B}{(Intro)} \quad \frac{A \vdash (A \to B) \to B}{(Intro)} \quad \frac{\vdash (A \to B) \to B}{(Intro)}
\]

\[\vdash \lambda x : A. \lambda y : A \to B. (y \ x) : A \to ((A \to B) \to B)\]

In the context of \(\lambda\)-calculos it holds:
Example. **Peirce’s Law**: \((PL) \ ((A \rightarrow B) \rightarrow A) \rightarrow A\)

Holds in the classical logic, but not in the intuitionistic logic!
Isomorphism (Curry-Howard)

\( \Omega \vdash A \) is provable in the minimal intuitionistic logic if, and only if \( \Gamma \vdash M : A \) is a valid type judgment in the simple typed \( \lambda \)-calculus, where \( \Gamma \) is a list of declarations for propositional variables, \( s \) in \( \Omega \). The term \( M \) is a \( \lambda \)-term that represents the derivation of the proof.

References: [Hin97], [Sim00], ...
## Natural deduction

### Table: Natural deduction: inference rules

<table>
<thead>
<tr>
<th>Introduction</th>
<th>Elimination</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\frac{\varphi}{\varphi \land \psi}$ ($\land i$)</td>
<td>$\frac{\varphi \land \psi}{\varphi}$ ($\land e_r$) $\frac{\varphi \land \psi}{\psi}$ ($\land e_l$)</td>
</tr>
<tr>
<td>$\frac{\psi}{\varphi \land \psi}$</td>
<td>($\land i$)</td>
</tr>
<tr>
<td>$\varphi \lor \psi$ ($\lor i_r$)</td>
<td>$\frac{\varphi \lor \psi}{\varphi}$ ($\lor i_l$)</td>
</tr>
<tr>
<td>$\frac{\varphi}{\varphi \lor \psi}$ ($\lor i_l$)</td>
<td>$\frac{\varphi \lor \psi}{\chi}$ ($\lor e$), $u$, $v$</td>
</tr>
<tr>
<td>$\frac{[\varphi]^u}{\varphi \lor \psi}$</td>
<td>($\lor e$), $u$, $v$</td>
</tr>
<tr>
<td>$\vdots$</td>
<td>$\vdots$</td>
</tr>
<tr>
<td>$\frac{\psi}{\varphi \rightarrow \psi}$ ($\rightarrow i$), $u$</td>
<td>$\frac{\varphi \varphi \rightarrow \psi}{\psi}$ ($\rightarrow e$)</td>
</tr>
</tbody>
</table>
Natural deduction

Table: **Natural deduction: inference rules**

<table>
<thead>
<tr>
<th>Introduction</th>
<th>Elimination</th>
</tr>
</thead>
<tbody>
<tr>
<td>$[\varphi]^u$</td>
<td>$\bot$</td>
</tr>
<tr>
<td>$\vdash \neg \varphi$ ($\neg i$), $u$</td>
<td>$\varphi \vdash \neg \varphi$ ($\neg e$)</td>
</tr>
<tr>
<td>$\vdash \bot$ ($\bot e$)</td>
<td>$\varphi \vdash \neg \neg \varphi$ ($\neg \neg$)</td>
</tr>
<tr>
<td>$t = t$ ($= i$)</td>
<td>$t_1 = t_2 \quad \varphi[x/t_1]$ $\varphi[x/t_2]$ ($= e$)</td>
</tr>
</tbody>
</table>
## Natural deduction

**Table:** **Natural deduction: inference rules**

<table>
<thead>
<tr>
<th>Introduction</th>
<th>Elimination</th>
</tr>
</thead>
<tbody>
<tr>
<td>$y$ independent</td>
<td>$\forall x \varphi$</td>
</tr>
<tr>
<td>$\vdash \varphi[x/y]$</td>
<td>$\frac{\forall x \varphi}{\varphi[x/t]}$ (\textit{\forall e})</td>
</tr>
<tr>
<td>$\forall x \varphi$</td>
<td>$\frac{\varphi[x/t]}{\exists x \varphi}$ (\textit{\exists i})</td>
</tr>
<tr>
<td>$\exists x \varphi$</td>
<td>$\frac{\forall x \varphi}{\forall x \varphi}$ (\textit{\forall i})</td>
</tr>
<tr>
<td>$\forall x \varphi$</td>
<td>$\frac{\varphi[x/t]}{\exists x \varphi}$ (\textit{\exists i})</td>
</tr>
<tr>
<td>$\exists x \varphi$</td>
<td>$\frac{\forall x \varphi}{\forall x \varphi}$ (\textit{\forall i})</td>
</tr>
<tr>
<td>$\exists x \varphi$</td>
<td>$\frac{\forall x \varphi}{\forall x \varphi}$ (\textit{\forall i})</td>
</tr>
</tbody>
</table>
An example of natural deduction

1. $\Delta_1$:

\[
\frac{\neg \varphi[x/y]^\nu}{\exists x \neg \varphi} \quad \frac{\exists i}{\neg \exists x \neg \varphi}^u \quad (\neg e)
\]

\[
\frac{\bot}{\varphi[x/y]} \quad (PBC), \nu
\]

\[
\frac{\varphi[x/y]}{\forall x \varphi} \quad (\forall i)
\]

\[
\frac{\bot}{\neg \forall x \varphi} \quad (PBC), w
\]

2. $\Delta_2$:

\[
\frac{\forall x \varphi^\nu}{\varphi[x/y]} \quad (\forall e)
\]

\[
\frac{\neg \varphi[x/y]^w}{\bot} \quad (\neg e)
\]

\[
\frac{\bot}{\exists x \neg \varphi}^u \quad (\exists e), w
\]

\[
\frac{\bot}{\neg \forall x \varphi} \quad (\neg i), \nu
\]

\[
\frac{\exists x \neg \varphi \to \neg \forall x \varphi}{\bot} \quad (\rightarrow i), u
\]
## Gentzen Systems

**Table: Gentzen Systems: Inference Rules**

<table>
<thead>
<tr>
<th>Left rules</th>
<th>Right rules</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Axioms</strong></td>
<td></td>
</tr>
<tr>
<td>$A \vdash A$</td>
<td>$\bot \vdash \bot$</td>
</tr>
<tr>
<td>(Ax)</td>
<td>(L⊥)</td>
</tr>
<tr>
<td><strong>Structural rules</strong></td>
<td></td>
</tr>
<tr>
<td>$\Gamma \vdash \Delta$</td>
<td>$\Gamma \vdash \Delta$</td>
</tr>
<tr>
<td>$A, \Gamma \vdash \Delta$</td>
<td>$\Gamma, \Delta, A$</td>
</tr>
<tr>
<td>(LW)</td>
<td>(RW)</td>
</tr>
<tr>
<td>$A, A, \Gamma \vdash \Delta$</td>
<td>$\Gamma \vdash \Delta, A, A$</td>
</tr>
<tr>
<td>(LC)</td>
<td>(RC)</td>
</tr>
</tbody>
</table>
Gentzen Systems

**Table: Gentzen Systems: inference rules**

<table>
<thead>
<tr>
<th>Left rules</th>
<th>Right rules</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Logical rules</strong></td>
<td></td>
</tr>
<tr>
<td>$A_i, \Gamma \vdash \Delta \quad A_0 \land A_1, \Gamma \vdash \Delta \quad (L\land), \ (i = 0, 1)$</td>
<td>$\Gamma \vdash \Delta, A \quad \Gamma \vdash \Delta, B \quad (R\land)$</td>
</tr>
<tr>
<td>$A, \Gamma \vdash \Delta \quad B, \Gamma \vdash \Delta \quad A \lor B, \Gamma \vdash \Delta \quad (L\lor)$</td>
<td>$\Gamma \vdash \Delta, A_i \quad \Gamma \vdash \Delta, A_0 \lor A_1 \quad (R\lor), \ (i = 0, 1)$</td>
</tr>
<tr>
<td>$\Gamma \vdash \Delta, A \quad B, \Gamma \vdash \Delta \quad A \rightarrow B, \Gamma \vdash \Delta \quad (L\rightarrow)$</td>
<td>$A, \Gamma \vdash \Delta, B \quad \Gamma \vdash \Delta, A \rightarrow B \quad (R\rightarrow)$</td>
</tr>
</tbody>
</table>
# Gentzen Systems

## Table: Gentzen Systems: Inference Rules

<table>
<thead>
<tr>
<th>Left rules</th>
<th>Right rules</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Logical rules</strong></td>
<td></td>
</tr>
<tr>
<td>$A[x/t], \Gamma \vdash \Delta$</td>
<td>$\Gamma \vdash \Delta, A[x/y]$ $(R\forall), y \not\in FV(\Gamma, \Delta)$</td>
</tr>
<tr>
<td>$\forall x A, \Gamma \vdash \Delta$</td>
<td>$\Gamma \vdash \Delta, \forall x A$ $(L\forall)$</td>
</tr>
<tr>
<td>$A[x/y], \Gamma \vdash \Delta$</td>
<td>$\Gamma \vdash \Delta, A[x/t]$ $(R\exists)$</td>
</tr>
<tr>
<td>$\exists x A, \Gamma \vdash \Delta$</td>
<td></td>
</tr>
</tbody>
</table>
An example of deduction à la Gentzen

\[
\frac{\neg \varphi}{\varphi \vdash \neg \neg \varphi} \quad (\text{Ax})
\]

\[
\frac{\exists x \neg \varphi}{
\exists x \neg \varphi, \neg \varphi \vdash \exists x \neg 
\varphi} \quad (\text{R}\exists)
\]

\[
\frac{\neg \exists x \neg \varphi}{
\neg \exists x \neg \varphi, \neg \varphi \vdash \neg \exists
x \neg \varphi} \quad (\text{LW})
\]

\[
\frac{\neg \exists x \neg \varphi}{
\neg \exists x \neg \varphi, \neg \varphi \vdash \exists x \neg
\varphi} \quad (\text{Ax})
\]

\[
\frac{\exists x \neg \varphi \vdash \exists x \neg \varphi}{
\exists x \neg \varphi, \neg \varphi \vdash \exists x \neg \varphi
\neg \varphi} \quad (\text{R}\wedge)
\]

\[
\frac{\exists x \neg \varphi \vdash \exists x \neg \varphi}{
\exists x \neg \varphi, \neg \varphi \vdash \exists x \neg \varphi
\neg \varphi} \quad (\text{LW})
\]

\[
\frac{\neg \exists x \neg \varphi}{
\neg \exists x \neg \varphi, \neg \varphi \vdash \exists x \neg
\varphi} \quad (\text{Ax})
\]

\[
\frac{\exists x \neg \varphi \vdash \exists x \neg \varphi}{
\exists x \neg \varphi, \neg \varphi \vdash \exists x \neg \varphi
\neg \varphi} \quad (\text{R}\wedge)
\]

\[
\frac{\neg \exists x \neg \varphi}{
\neg \exists x \neg \varphi, \neg \varphi \vdash \exists x \neg
\varphi} \quad (\text{Ax})
\]

\[
\frac{\exists x \neg \varphi \vdash \exists x \neg \varphi}{
\exists x \neg \varphi, \neg \varphi \vdash \exists x \neg \varphi
\neg \varphi} \quad (\text{R}\wedge)
\]

\[
\frac{\neg \exists x \neg \varphi}{
\neg \exists x \neg \varphi, \neg \varphi \vdash \exists x \neg
\varphi} \quad (\text{Ax})
\]

\[
\frac{\exists x \neg \varphi \vdash \exists x \neg \varphi}{
\exists x \neg \varphi, \neg \varphi \vdash \exists x \neg \varphi
\neg \varphi} \quad (\text{R}\wedge)
\]

\[
\frac{\neg \exists x \neg \varphi}{
\neg \exists x \neg \varphi, \neg \varphi \vdash \exists x \neg
\varphi} \quad (\text{Ax})
\]

\[
\frac{\exists x \neg \varphi \vdash \exists x \neg \varphi}{
\exists x \neg \varphi, \neg \varphi \vdash \exists x \neg \varphi
\neg \varphi} \quad (\text{R}\wedge)
\]

\[
\frac{\neg \exists x \neg \varphi}{
\neg \exists x \neg \varphi, \neg \varphi \vdash \exists x \neg
\varphi} \quad (\text{Ax})
\]

\[
\frac{\exists x \neg \varphi \vdash \exists x \neg \varphi}{
\exists x \neg \varphi, \neg \varphi \vdash \exists x \neg \varphi
\neg \varphi} \quad (\text{R}\wedge)
\]

\[
\frac{\neg \exists x \neg \varphi}{
\neg \exists x \neg \varphi, \neg \varphi \vdash \exists x \neg
\varphi} \quad (\text{Ax})
\]

\[
\frac{\exists x \neg \varphi \vdash \exists x \neg \varphi}{
\exists x \neg \varphi, \neg \varphi \vdash \exists x \neg \varphi
\neg \varphi} \quad (\text{R}\wedge)
\]

\[
\frac{\neg \exists x \neg \varphi}{
\neg \exists x \neg \varphi, \neg \varphi \vdash \exists x \neg
\varphi} \quad (\text{Ax})
\]

\[
\frac{\exists x \neg \varphi \vdash \exists x \neg \varphi}{
\exists x \neg \varphi, \neg \varphi \vdash \exists x \neg \varphi
\neg \varphi} \quad (\text{R}\wedge)
\]

\[
\frac{\neg \exists x \neg \varphi}{
\neg \exists x \neg \varphi, \neg \varphi \vdash \exists x \neg
\varphi} \quad (\text{Ax})
\]

\[
\frac{\exists x \neg \varphi \vdash \exists x \neg \varphi}{
\exists x \neg \varphi, \neg \varphi \vdash \exists x \neg \varphi
\neg \varphi} \quad (\text{R}\wedge)
\]

\[
\frac{\neg \exists x \neg \varphi}{
\neg \exists x \neg \varphi, \neg \varphi \vdash \exists x \neg
\varphi} \quad (\text{Ax})
\]

\[
\frac{\exists x \neg \varphi \vdash \exists x \neg \varphi}{
\exists x \neg \varphi, \neg \varphi \vdash \exists x \neg \varphi
\neg \varphi} \quad (\text{R}\wedge)
\]

\[
\frac{\neg \exists x \neg \varphi}{
\neg \exists x \neg \varphi, \neg \varphi \vdash \exists x \neg
\varphi} \quad (\text{Ax})
\]

\[
\frac{\exists x \neg \varphi \vdash \exists x \neg \varphi}{
\exists x \neg \varphi, \neg \varphi \vdash \exists x \neg \varphi
\neg \varphi} \quad (\text{R}\wedge)
\]

\[
\frac{\neg \exists x \neg \varphi}{
\neg \exists x \neg \varphi, \neg \varphi \vdash \exists x \neg
\varphi} \quad (\text{Ax})
\]

\[
\frac{\exists x \neg \varphi \vdash \exists x \neg \varphi}{
\exists x \neg \varphi, \neg \varphi \vdash \exists x \neg \varphi
\neg \varphi} \quad (\text{R}\wedge)
\]
The Prototype Verification System - PVS

PVS is a verification system, developed by the SRI International Computer Science Laboratory, which consists of

1. a specification language:
   - based on higher-order logic;
   - a type system based on Church’s simple theory of types augmented with subtypes and dependent types.

2. an interactive theorem prover:
   - based on sequent calculus; that is, goals in PVS are sequents of the form $\Gamma \vdash \Delta$, where $\Gamma$ and $\Delta$ are finite sequences of formulae, with the usual Gentzen semantics.
Sequent calculus

- Sequents of the form: $\Gamma \vdash \Delta$.
  - Assuming $\Gamma$ and $\Delta$ derivable.
  - $A_1, A_2, ..., A_n \vdash B_1, B_2, ..., B_m$ interpreted as
    $A_1 \land A_2 \land ... \land A_n \vdash B_1 \lor B_2 \lor ... \lor B_m$.

- Inference rules
  - Premises and conclusions are simultaneously constructed.
  - Example: $\frac{\Gamma \vdash \Delta}{\Gamma_1 \vdash \Delta_1}$

- Goal: $\vdash \Delta$. 
Sequent calculus in PVS

- Representation of $A_1, A_2, \ldots, A_n \vdash B_1, B_2, \ldots, B_m$:
  
  $\vdash 1 \quad A_1$
  
  $\vdash n \quad A_n$
  
  $\vdash 1 \quad B_1$
  
  $\vdash n \quad B_n$

- Proof tree: each node is labelled by a sequent.
- A PVS proof command corresponds to the application of an inference rule.
- In general: $\frac{\Gamma_1 \vdash \Delta_1 \ldots \Gamma_n \vdash \Delta_n}{\Gamma \vdash \Delta}$
Some inference rules in PVS

- **Structural:**

  \[
  \frac{\Gamma_1 \vdash \Delta_1}{\Gamma_2 \vdash \Delta_2} \text{ W, if } \Gamma_1 \subseteq \Gamma_2 \text{ e } \Delta_1 \subseteq \Delta_2
  \]

- **Propositional:**

  \[
  \frac{\Gamma, A \vdash A, \Delta}{\text{Ax}} \quad \frac{\Gamma, \text{FALSE} \vdash \Delta}{\text{FALSE}\vdash} \quad \frac{\Gamma \vdash \text{TRUE}, \Delta}{\text{TRUE}}
  \]
Some inference rules in PVS

- **Cut:**
  - Corresponds to the case proof command.
  
  \[
  \frac{\Gamma, A \vdash \Delta}{\Gamma \vdash \Delta} \quad \frac{\Gamma \vdash A, \Delta}{\Gamma \vdash \Delta} \quad \text{Cut}
  \]

- **Conditional:** \texttt{IF-THEN-ELSE}.

  \[
  \frac{\Gamma, A, B \vdash \Delta \quad \Gamma, C \vdash A, \Delta}{\Gamma, \text{IF}(A, B, C) \vdash \Delta} \quad \text{IF} \vdash
  \]
  \[
  \frac{\Gamma, A \vdash B, \Delta \quad \Gamma \vdash A, C, \Delta}{\Gamma \vdash \text{IF}(A, B, C)\Delta} \quad \vdash \text{IF}
  \]
Programs versus demonstrations

Example: greatest common divisor \( gcd \)

\[ \textbf{Theorem} \ [\text{Euclid 320-275 BC}] \forall n \geq 0, m > 0, gcd(n, m) = gcd(m, n \ MOD \ m) \]

idea

(Detail: “\( n \ MOD \ m \)” is computed as “\( (n - m) \ MOD \ m \)”)

procedure \( gcd(m, n) \)
\quad if \( m < n \) then \( gcd(n, m) \)
\quad else \( (m \geq n) \)
\quad \quad \quad \quad gcd(m - n, n)
End procedure

algorithm
Programs versus demonstrations

\[
gcd(6, 4) \rightarrow gcd(2, 4) \rightarrow gcd(4, 2) \rightarrow gcd(2, 2) \rightarrow gcd(0, 2) \rightarrow gcd(2, 0) \rightarrow \cdots
\]

problem: infinite loop

Proof of totality: Domain \( \mathbb{N} \) (Type of the objects)
BI: \( gcd(0, n) \) undefined! Define \( gcd(0, n) = n \).
PI: Suppose \( gcd(k, n) \) well-defined for all \( n \) and \( k < m \), with \( m > 0 \).
\[ \Rightarrow \ gcd(m, n) \text{ well-defined:} \]
Case 1: \( m > n \). \( gcd(m, n) = gcd(m - n, n) \) Apply IH only if \( n > 0 \)! Define \( gcd(m, 0) = m \).
Case 2: \( m \leq n \). \( gcd(m, n) = gcd(n, m) \) that is well-defined by IH.
Programs versus demonstrations

procedure \texttt{gcd}(m, n)
\begin{align*}
&\text{if } m = 0 \text{ then } n \\
&\text{else } (** \quad m > 0 \ast \ast) \\
&&\text{if } m < n \text{ then } \texttt{gcd}(n, m) \\
&&\text{else } (** \quad m > 0 \& \ m \geq n \ast \ast) \\
&&&&\text{if } n = 0 \text{ then } m \\
&&&&\text{else } (** \quad m > 0 \& \ n > 0 \& \ m \geq n \ast \ast) \\
&&&&\quad \texttt{gcd}(m - n, n)
\end{align*}
End procedure

Program extracted from the proved correct specification
Example in PVS: $gcd$ extended to $\mathbb{Z} \times \mathbb{Z}$

**Theorem** [Euclid 320-275 BC] $\forall n \geq 0, m > 0, gcd(n, m) = gcd(m, n \mod m)$

**Theorem** [Euclid $\mathbb{Z}^2$] $\forall m, n \neq 0 \in \mathbb{Z}, gcd(m, n) = gcd(m, m \mod n)$

(Detail: “$n \mod m$” is computed as “$(n - m) \mod m$”)
Example in PVS: \textit{gcd} extended to $\mathbb{Z} \times \mathbb{Z}$

\begin{verbatim}
procedure gcd(m, n)
  if |m| = |n| then |m|
  else, if (m = 0 or n = 0) then |m + n|
    else, if |n| > |m| then gcd(|n| - |m|, |m|)
    else gcd(|m| - |n|, |n|)
End procedure
\end{verbatim}

algorithm extended to $\mathbb{Z}^2$
Example in PVS: $gcd : \mathbb{Z} \times \mathbb{Z} \rightarrow \mathbb{N}$ Executable code

- Specification & verification in PVS
- Executable code extracted from the proved correct specification - Muñoz’s system PVSWhy
Formalisation of the correctness of $gcd$

Quantitative Information

<table>
<thead>
<tr>
<th>Theory</th>
<th>L. Specification</th>
<th>L. Proof</th>
<th>Theorems</th>
<th>TCCs</th>
<th>S. Specification</th>
<th>S. Proof</th>
</tr>
</thead>
<tbody>
<tr>
<td>gcd</td>
<td>94</td>
<td>1665</td>
<td>21</td>
<td>6</td>
<td>3.2K</td>
<td>74K</td>
</tr>
<tr>
<td></td>
<td>94</td>
<td>1665</td>
<td>21</td>
<td>6</td>
<td>3.2K</td>
<td>74K</td>
</tr>
</tbody>
</table>
Executable code for \texttt{gcd} in $\mathbb{Z} \times \mathbb{Z}$ extracted with PVSWhy

```java
/* File: gcd.java
 * Automatically generated from PVS theory gcd (gcd.pvs)
 * By: PVS2Why-0.1 (10/31/07)
 * Date: 11:45:52 11/1/2007
 */

import PVS2Java.*;

public class gcd {

    public int gcd(final int n,
                    final int m) {
        final int abs_n = Math.abs(n);
        final int abs_m = Math.abs(m);
        if (abs_n == abs_m) {
            return abs_n;
        } else {
            if (n == 0 || m == 0) {
                return abs_n + m;
            } else {
                if (abs_n > abs_m) {
                    return gcd(abs_n - abs_m, abs_m);
                } else {
                    return gcd(abs_m - abs_n, abs_n);
                }
            }
        }
    }

    // Higher order function gcd
    public Lambda<Integer> gcd = new Lambda<Integer>() {
        public Integer apply(Object... obj) {
            int n = (Integer)obj[0];
            int m = (Integer)obj[1];
            return gcd(n, m);
        }
    };
```
Formalisation of the logical correctness of a simple 2D convolution

Figure: Wong, Jasiunas & Kearney 2D convolution [WJK05]
Formalisation of the logical correctness of a simple 2D convolution

Implementation of WJK-Convolution in FPGAs
Departamento Engenharia Mecatrônica/UnB
Formalisation of the logical correctness of an improved 2D convolution

Implementation Y-Convolution in FPGAs
(J.Yudi) Departamento Engenharia Mecatrônica/UnB
Formalisation of the logical correctness of a simple 2D convolution

Quantitative Information

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>image_masks</td>
<td>194</td>
<td>3788</td>
<td>75</td>
<td>64</td>
<td>7.8K</td>
<td>78K</td>
</tr>
<tr>
<td>fin_seq_extra</td>
<td>162</td>
<td>1612</td>
<td>62</td>
<td>29</td>
<td>7K</td>
<td>179k</td>
</tr>
<tr>
<td></td>
<td>356</td>
<td>5400</td>
<td>137</td>
<td>93</td>
<td>14.8K</td>
<td>257K</td>
</tr>
</tbody>
</table>
Conclusions and Future Work

- Nowadays formalising computational objects is essential in order to produce certified and robust products.
- Each piece of software/hardware deserves a formal mathematical treatment.
- Advances in formal methods includes:
  - specification and formalisation of mathematical theories and proof technologies that can be applied to a particular style of design (e.g. trs theory [GAR10]);
  - application of particular formalisation styles to the design and production of specific technological tools: such as cryptographic protocols (e.g. [SAR10]) and reconfigurable hardware implementations (e.g. [ARLJH06]).
Conclusions and Future Work

References

Prototyping time- and space-efficient computations of algebraic operations over dynamically reconfigurable systems modeled by rewriting-logic.

D. Dolev and A. C. Yao.

A. L. Galdino and M. Ayala-Rincón.
A Formalization of the Knuth-Bendix(-Huet) Critical Pair Theorem.
Springer Online First doi 10.1007/s10817-010-9165-2.

A.L. Galdino, C. Muñoz, and M. Ayala-Rincón.
Formal Verification of an Optimal Air Traffic Conflict Detection and Recovery Algorithm.

J.R. Hindley.
*Basic Simple Type Theory*.

G. Lowe.
References

G. Lowe.
Breaking and Fixing the Needham-Schroeder Public-Key Protocol Using FDR.

Formalization of Security Proofs Using PVS in the Dolev-Yao Model.

D.N. Sobrinho and M. Ayala-Rincón.

H. Simmons.
*Derivation and Computation: taking the Curry-Howard correspondence seriously.*

S.C. Wong, M. Jasiunas, and D. Kearney.