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

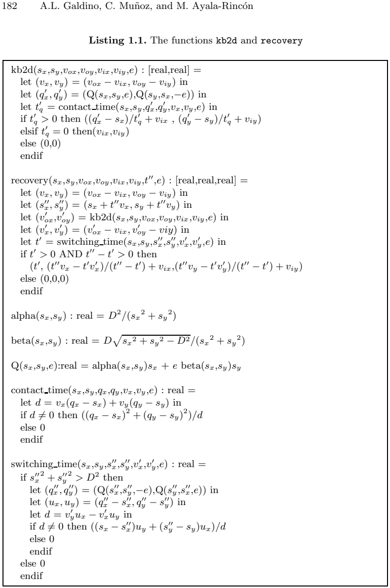
Talk's Plan

- 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 Γ and Δ are finite sequences of formulae, with the usual Gentzen semantics.



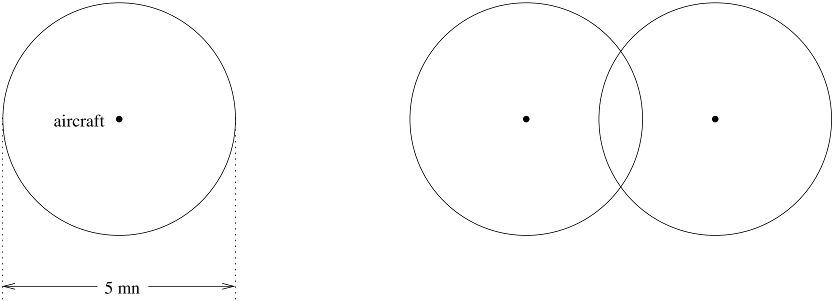
182 A.L. Galdino, C. Muñoz, and M. Ayala-Rincón

Listing 1.1. The functions `kb2d` and `recovery`

```
kb2d($s_x$,$s_y$,$v_{ox}$,$v_{oy}$,$v_{ix}$,$v_{iy}$,$e$) : [real,real] =
  let $(v_x, v_y) = (v_{ox} - v_{ix}, v_{oy} - v_{iy})$ in
  let $(q'_x, q'_y)$ = (Q($s_x$,$s_y$,$e$),Q($s_y$,$s_x$,$-e$)) in
  let $t'_q$ = contact_time($s_x$,$s_y$,$q'_x$,$q'_y$,$v_x$,$v_y$,$e$) in
  if $t'_q > 0$ then $((q'_x - s_x)/t'_q + v_{ix}\ ,\ (q'_y - s_y)/t'_q + v_{iy})$
  elsif $t'_q = 0$ then($v_{ix}$,$v_{iy}$)
  else (0,0)
  endif

recovery($s_x$,$s_y$,$v_{ox}$,$v_{oy}$,$v_{ix}$,$v_{iy}$,$t''$,$e$) : [real,real,real] =
  let $(v_x, v_y) = (v_{ox} - v_{ix}, v_{oy} - v_{iy})$ in
  let $(s''_x, s''_y) = (s_x + t''v_x, s_y + t''v_y)$ in
  let $(v'_{ox}, v'_{oy})$ = kb2d($s_x$,$s_y$,$v_{ox}$,$v_{oy}$,$v_{ix}$,$v_{iy}$,$e$) in
  let $(v'_x, v'_y) = (v'_{ox} - v_{ix}, v'_{oy} - viy)$ in
  let $t'$ = switching_time($s_x$,$s_y$,$s''_x$,$s''_y$,$v'_x$,$v'_y$,$e$) in
  if $t' > 0$ AND $t'' - t' > 0$ then
    $(t',\ (t''v_x - t'v'_x)/(t'' - t') + v_{ix},(t''v_y - t'v'_y)/(t'' - t') + v_{iy})$
  else (0,0,0)
  endif

alpha($s_x$,$s_y$) : real = $D^2/({s_x}^2 + {s_y}^2)$

beta($s_x$,$s_y$) : real = $D\sqrt{{s_x}^2 + {s_y}^2 - D^2}/({s_x}^2 + {s_y}^2)$

Q($s_x$,$s_y$,$e$):real = alpha($s_x$,$s_y$)$s_x$ + $e$ beta($s_x$,$s_y$)$s_y$

contact_time($s_x$,$s_y$,$q_x$,$q_y$,$v_x$,$v_y$,$e$) : real =
  let $d = v_x(q_x - s_x) + v_y(q_y - s_y)$ in
  if $d \neq 0$ then $((q_x - s_x)^2 + (q_y - s_y)^2)/d$
  else 0
  endif

switching_time($s_x$,$s_y$,$s''_x$,$s''_y$,$v'_x$,$v'_y$,$e$) : real =
  if ${s''_x}^2 + {s''_y}^2 > D^2$ then
    let $(q''_x, q''_y)$ = (Q($s''_x$,$s''_y$,$-e$),Q($s''_y$,$s''_x$,$e$)) in
    let $(u_x, u_y) = (q''_x - s''_x, q''_y - s''_y)$ in
    let $d = v'_y u_x - v'_x u_y$ in
    if $d \neq 0$ then $((s_x - s''_x)u_y + (s''_y - s_y)u_x)/d$
    else 0
    endif
  else 0
  endif
```

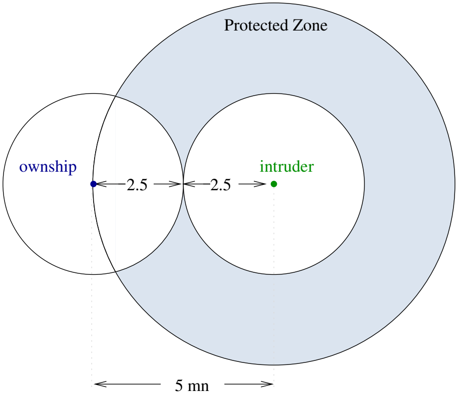
KB2D [GnAR07] improves NIA/NASA's KB3D

The Problem: Basic Definition and concepts



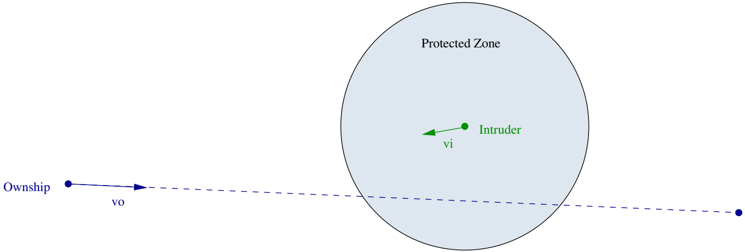
- **Avoidance Region:** circle centered in the aircraft.
- **Conflict:** two aircraft are said to be in conflict when their avoidance regions overlap.

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 *ownship* in the *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

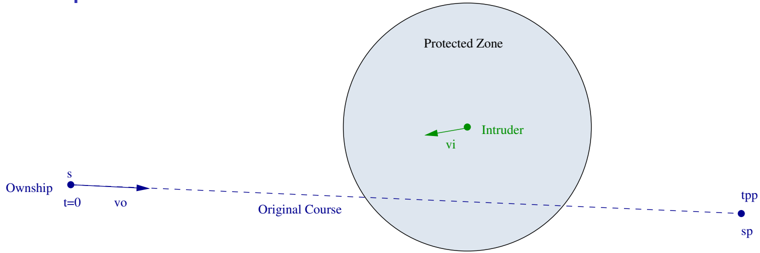
  - horizontal speed only

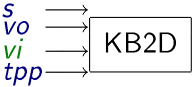
  - heading only

  - KB2D combines changes of horizontal speed **and** of heading

- KB2D is a 2-Dimensional CD&R.

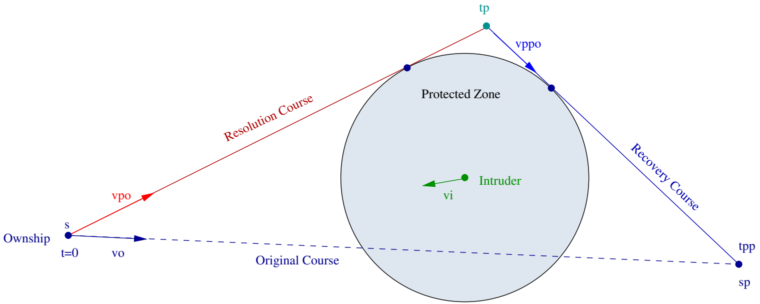
KB2D: Inputs

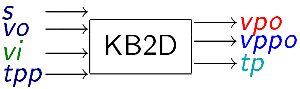




- s: *ownship's* relative position
- vo: *ownship's* velocity
- vi: *intruder's* velocity
- tpp: Required Time of Arrival

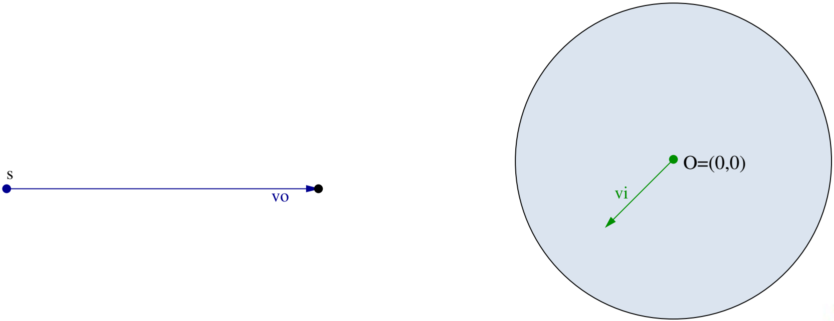
KB2D: Outputs





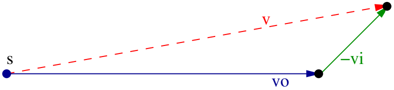
- vpo: Resolution velocity
- vppo: Recovery velocity
- tp: Time of switch

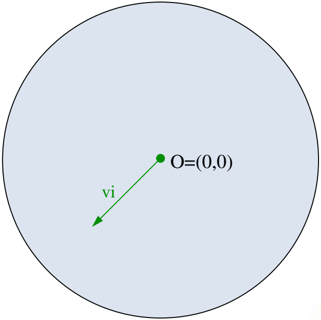
The Algorithm (Geometric Solution)



The Algorithm (Geometric Solution)

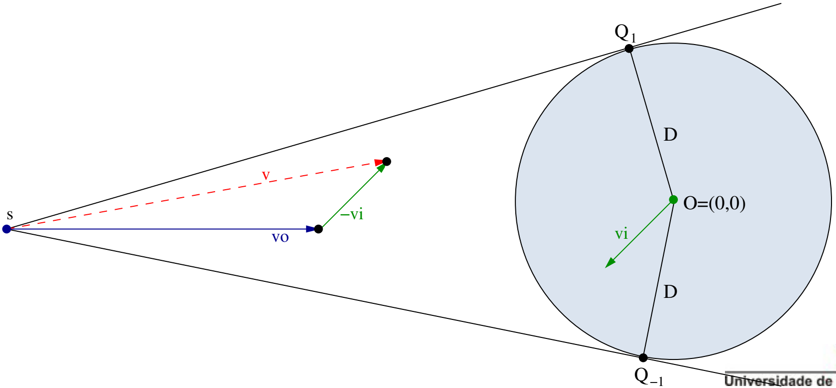
1. Ownship's relative velocity: v





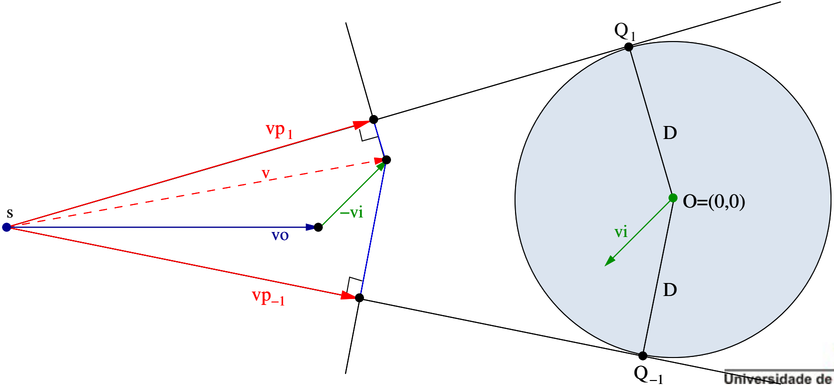
The Algorithm (Geometric Solution)

1. Ownship's relative velocity: v
2. Tangent points: Q_1 and Q_{-1}



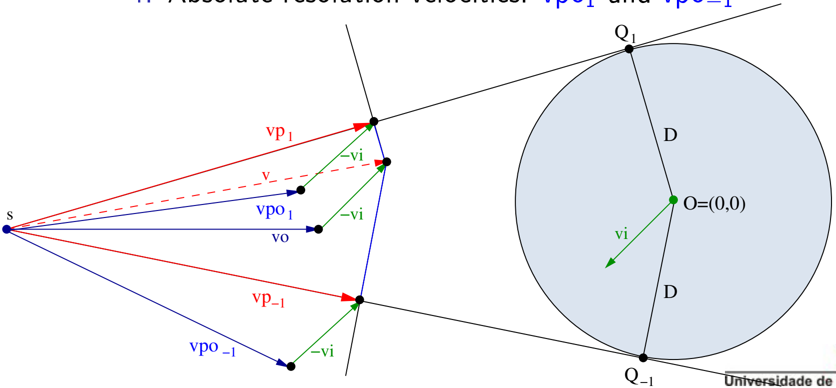
The Algorithm (Geometric Solution)

1. Ownship's relative velocity: v
2. Tangent points: Q_1 and Q_{-1}
3. Relative resolution velocities: vp_1 and vp_{-1}



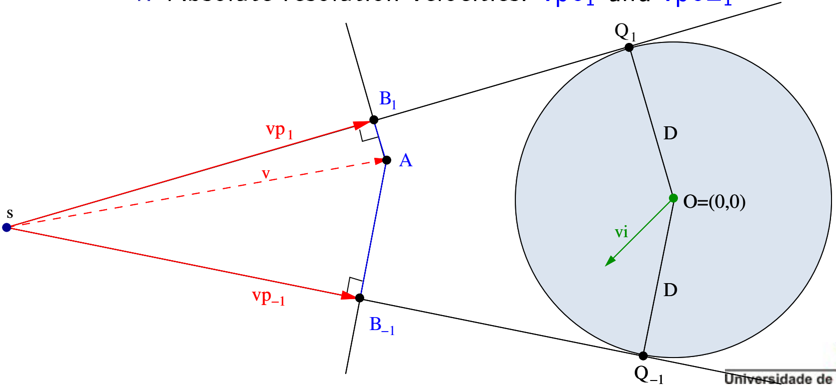
The Algorithm (Geometric Solution)

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}

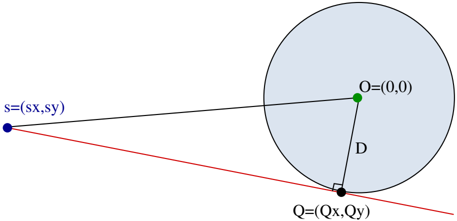


The Algorithm (Geometric Solution)

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}

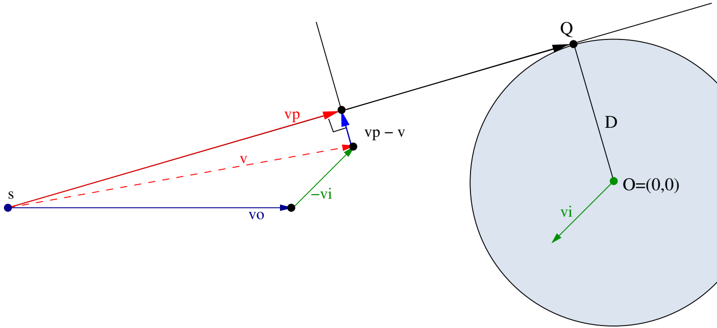


Computing the tangent points



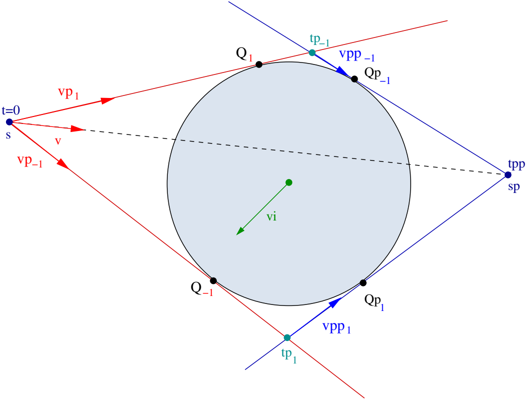
$$\begin{cases} sx.Qx + sy.Qy & = & D^2 \\ Qx^2 + Qy^2 & = & D^2 \end{cases}$$

Computing the relative resolution velocities



$$\begin{cases} vp & = & k \cdot (Q - s) \\ vp \cdot (vp - v) & = & 0 \end{cases}$$

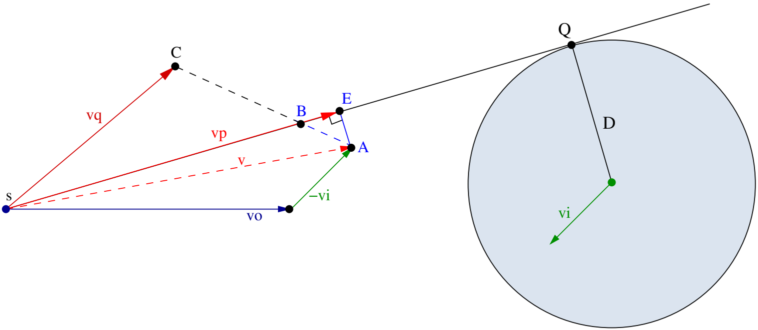
Geometric and Analytic Solution (Recovery)



$$s + tp\,vp + (tpp - tp)vpp = \color{blue}{sp} = s + tpp\,v$$

$$\Longrightarrow \qquad vpp = \frac{1}{tpp - tp}(tpp\,v - tp\,vp)$$

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.

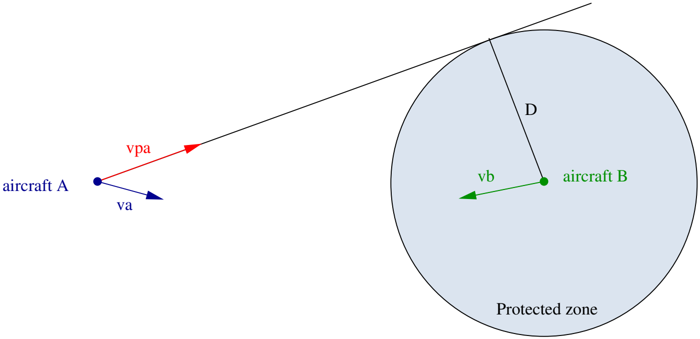
Coordination

- Let A and B be two conflicting aircrafts.

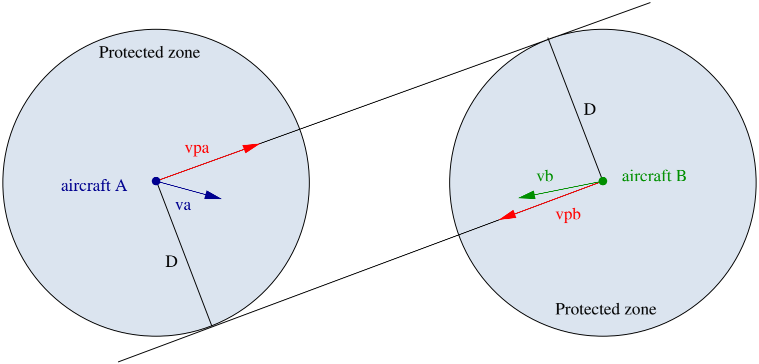
Coordination

- The relative positions computed by each aircraft are opposite.
- The time of loss of separation is the same for both aircrafts.

Coordination



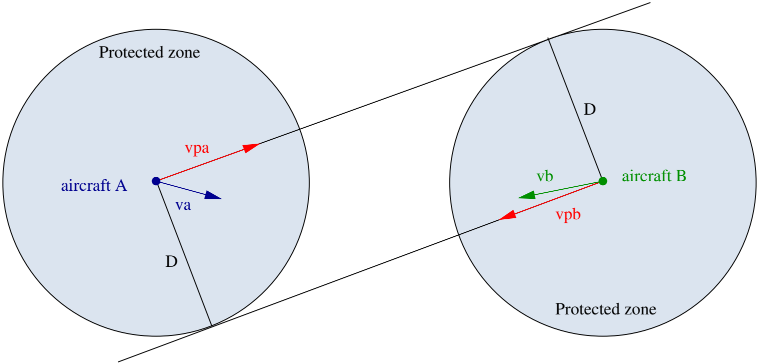
Coordination



Lemma

For all eps = ± 1, vpa and vpb are parallel.

Coordination



Lemma

For all eps = ± 1, vpa *and* vpb *are parallel.*

Formal Verification (An Example)

Theorem (kb2d_correct)

For all $s,\ v = vo - vi,\ T > 0,\ D > 0,\ vp,\ vpo,\ eps = \pm 1,$
$conflict?(s,\ v,\ T)$ *and*
$s_x^2 + s_y^2 > D^2$ *and*
$vpo\ = kb2d(sx,sy,vox,voy,vix,viy,eps)$ *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.
    - $E = \{E_u \mid u \in U\}$
    - $D = \{D_u \mid u \in U\}$
- $\Sigma = E \cup D$
- Σ^* set of words over Σ.
- Monoid freely generated by Σ and congruences:

$$E_u D_u = \lambda \qquad D_u E_u = \lambda, \quad \forall u \in U \tag{1}$$

- $E_u(D_u(M)) = D_u(E_u(M)) = M, \forall M$ plain text.

Formalisation: normalisation property

- Rewriting rules:

$$E_u D_u \rightarrow \lambda \qquad D_u E_u \rightarrow \lambda, \quad \forall u \in U \tag{2}$$

- Canonical form: $\forall \delta \in \Sigma^*$, $\overline{\delta}$ is such that

$$\delta \rightarrow^* \overline{\delta}$$

  and $\overline{\delta}$ 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: $\alpha\beta : U \times U \to \Sigma^*$)

$$\forall x, y \in U \mid x \neq y :$$

$$\left\{ \begin{array}{l}
1.\ \alpha\beta(x,y) \neq \lambda \\
2.\ \alpha\beta(x,y) = \overline{\alpha\beta(x,y)} \\
3.\ \alpha\beta(x,y) \in \Phi(x,y)^* \\
4.\ \forall u, v \in U : \\
\quad 4.1.\ |\alpha\beta(x,y)| = |\alpha\beta(u,v)| \\
\quad 4.2.\ \forall 0 \leq j < |\alpha\beta(x,y)| : \\
\quad\quad 4.2.1.\ \ \alpha\beta(x,y)_{[j]} = E_x \textit{ iff } \alpha\beta(u,v)_{[j]} = E_u \\
\quad\quad 4.2.2.\ \ \alpha\beta(x,y)_{[j]} = E_y \textit{ iff } \alpha\beta(u,v)_{[j]} = E_v \\
\quad\quad 4.2.3.\ \ \alpha\beta(x,y)_{[j]} = D_x \textit{ iff } \alpha\beta(u,v)_{[j]} = D_u \\
\quad\quad 4.2.4.\ \ \alpha\beta(x,y)_{[j]} = D_y \textit{ iff } \alpha\beta(u,v)_{[j]} = D_v
\end{array} \right.$$

$$\boxed{\Phi(x,y) = \{D_x, E_x, E_y\}}$$

Universidade de Brasília

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 $x, y \in U$

$$x \rightarrow y: \; P_0M = \alpha\beta_0(x,y)M$$
$$y \rightarrow x: \; P_1P_0M = \alpha\beta_1(y,x)\alpha\beta_0(x,y)M$$
$$\vdots$$
$$x \rightarrow y: \; P_{|P|-1}...P_0M = \alpha\beta_{|P|-1}(x,y)...\alpha\beta_0(x,y)M, \text{ if } |P| > 2 \text{ odd}$$

or

$$y \rightarrow x: \; P_{|P|-1}...P_0M = \alpha\beta_{|P|-1}(y,x)...\alpha\beta_0(x,y)M, \text{ if } |P| > 2 \text{ even}$$

Specification of the adversary *Admissible Language*

Definition (Adversary *Admissible Language*)

$(\Sigma_1^*(z) \cup \Sigma_2)^*$, *where:*

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

- 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 $x, y, z \in U$, $\forall \gamma \in (\Sigma_1^*(z) \cup \Sigma_2)^*$ *and* $0 \leq i < |P|$, *it holds:*

$$\overline{\gamma P_i ... P_0} \neq \lambda$$

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 $\delta \in \Sigma^*$. δ *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* $\forall 0 < i < |P|$:

  $$\begin{array}{l} P_i(x,y) \textit{ satisfies BP w.r.t. } x, \textit{ if } i \textit{ even} \\ P_i(y,x) \textit{ satisfies BP w.r.t. } y, \textit{ if } i \textit{ odd} \end{array}$$

- Example:

  Let P_2 the third step of a cascade protocol P, such that $\boxed{P_2(x,y) = E_y D_x E_y}$, 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**: $\neg(i) \vee \neg(ii) \implies P$ insecure.
- *Sufficiency*, by **contradiction**: $(i) \wedge (ii) \wedge P$ insecure $\implies$ P secure.

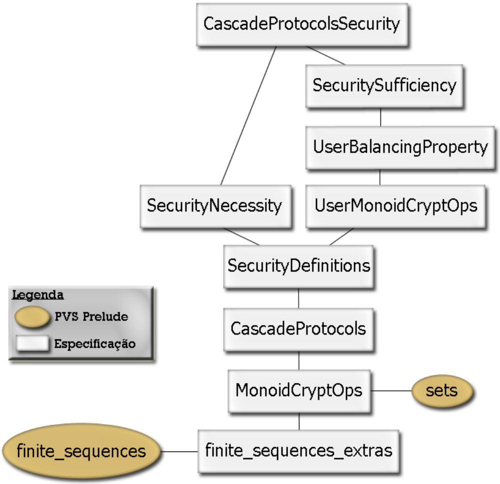
Theorem of Security

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.

Structure of the PVS formalisation



CascadeProtocolsSecurity

SecuritySufficiency

UserBalancingProperty

SecurityNecessity

UserMonoidCryptOps

SecurityDefinitions

CascadeProtocols

MonoidCryptOps

sets

finite_sequences

finite_sequences_extras

Legenda

PVS Prelude

Especificação

Necessity

- $A)\ \neg(i) \implies P$ insecure
  - $P_0(x,y) = D_x^k\ (k \in \mathbb{N}^*)$.
  - $\gamma = E_x^k$, so that $\overline{\gamma P_0} = \lambda$
- $B)\ \neg(ii) \implies P$ 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 $\overline{\tau_1 \alpha\beta(u,v)\tau_2} = D_u$.
  - 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

- $(i) \wedge (ii) \wedge P$ insecure $\implies$ P 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: $\overline{\gamma}$ 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 $\boxed{\overline{\gamma}^c = P_0(x,y)}$.
  - $E_y \in P_0(x,y)$:
    - Since $\overline{\gamma}^c = P_0(x,y)$, then $D_y \in \overline{\gamma}$.
    - $\overline{\gamma}$ 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*
- Neccesity of an explicit definition for abstract systems

Types

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

Types

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

History of types

- Treatment of paradoxes an inconsistencies in the formalization of mathematics:
  - Auto-reference, auto-reproduction
- Simple Types in the λ-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	$A ::= K \mid A \rightarrow B$
TERMS	$a ::= x \mid (a\ a) \mid \lambda x{:}B.a$

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

$$\underbrace{\Gamma \vdash a : B}_{\text{Type Judgment}}$$

Simple Types

SYNTAX

TYPES	$A ::= K \mid A \to B$
TERMS	$a ::= x \mid (a\ a) \mid \lambda x{:}B.a$

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

$$\underbrace{\Gamma \vdash a : B}_{\text{Type Judgment}}$$

Simple Types

SYNTAX

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

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

$$\underbrace{\Gamma \vdash a : B}_{\text{Type Judgment}}$$

Simple Types

SYNTAX

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

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

$$\underbrace{\Gamma \vdash a : B}_{\text{Type Judgment}}$$

Simple Types

$$\underbrace{\text{Examples}\begin{cases}(\lambda_x.x\ \ \lambda_x.x) \rightarrow_\beta \lambda_x.x & \text{auto-aplication}\\ (\lambda_x.(x\ \ x)\ \ \lambda_x.(x\ \ x)) \rightarrow_\beta (\lambda_x.(x\ \ x)\ \ \lambda_x.(x\ \ x)) & \text{auto-reproduction}\end{cases}}_{\text{Paradoxal Argumentation}}$$

Auto-aplication makes sense:

$$(\ \overbrace{\lambda_{x:A\rightarrow A}.x}^{(A\rightarrow A)\rightarrow A\rightarrow A}\ \ \overbrace{\lambda_{x:A}.x}^{A\rightarrow A}) \rightarrow_\beta \overbrace{\lambda_{x:A}.x}^{A\rightarrow A}$$

Polymorphism!

Simple Types

$$\underbrace{\text{Examples}\begin{cases} (\lambda_x.x\ \ \lambda_x.x) \rightarrow_\beta \lambda_x.x & \text{auto-aplication}\\ (\lambda_x.(x\ \ x)\ \ \lambda_x.(x\ \ x)) \rightarrow_\beta (\lambda_x.(x\ \ x)\ \ \lambda_x.(x\ \ x)) & \text{auto-reproduction}\end{cases}}_{\text{Paradoxal Argumentation}}$$

Auto-reproduction doesn't make sense:

$$(\lambda_{x:\tau_1}.(x\ \ x)\ \ \lambda_{x:\tau_2}.(x\ \ x)) \rightarrow_\beta (\lambda_{x:\tau_3}.(x\ \ x)\ \ \lambda_{x:\tau_4}.(x\ \ x))$$

Acceptable term, but non typable!

TA_λ: the simply typed λ-calculus

$$\frac{x \notin \Gamma}{x : A, \Gamma \vdash x : A}\ (Start) \qquad \frac{x \notin \Gamma \quad \Gamma \vdash a : B}{x : A, \Gamma \vdash a : B}\ (Weak)$$

$$\frac{x : A, \Gamma \vdash a : B}{\Gamma \vdash \lambda_{x:A}.a : A \rightarrow B}\ (Abs) \qquad \frac{\Gamma \vdash a : B \rightarrow A \quad \Gamma \vdash b : B}{\Gamma \vdash (a\ b) : A}\ (App)$$

Table: TA_λ

Example: type inference (auto-aplication)

Example (Type inference (auto-aplication))

$$\cfrac{\cfrac{\overline{x : A \vdash x : A}\ (Start)}{\vdash \lambda_{x:A}.x : A \to A}\ (Abs) \quad \cfrac{\overline{x : A \to A \vdash x : A \to A}\ (Start)}{\vdash \lambda_{x:A \to A}.x : (A \to A) \to (A \to A)}\ (Abs)}{\Gamma \vdash (\lambda_{x:A \to A}.x\ \lambda_{x:A}.x) : A \to A}\ (App)$$

Relevant problems in type theory

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

Revisiting relevant problems in type theory

$$\boxed{\underbrace{\Gamma}_{\text{variable declarations}} \vdash \underbrace{M}_{\lambda\text{-term or program}} : \underbrace{A}_{\text{type}}}$$

- 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.

proofs as programs - Curry-Howard isomorphism

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.

$\boxed{\text{Type Theory}}$ versus $\underbrace{\boxed{\text{Intiutionistic Logic}}}_{\text{Luitzen Egbertus Jan Brouwer [1920]}}$

Typing rules from the simple typed λ-calculus correspond 1-1 to the deductive rules of the minimal intuitionistic logic: *typing* rules are logical rules decorated with typed λ-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 σ and τ are implicational formulas, then $(\sigma \rightarrow \tau)$ is also an implicational formula.

proofs as programs - Curry-Howard isomorphism

A judgment in the intuitionistic logic, written as $\boxed{\Omega \vdash_I A}$, means that "$A$ is a logic consequence of Ω".

$$\boxed{\frac{}{\Omega, A \vdash_I A}(Axiom) \quad \frac{\Omega, A \vdash_I B}{\Omega \vdash_I A \to B}(Intro) \quad \frac{\Omega \vdash_I A \to B \quad \Omega \vdash_I A}{\Omega \vdash_I B}(Elim)}$$

Deduction rules of the minimal intuitionistic logic

A formel A is a *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)

$$\dfrac{\dfrac{\dfrac{\dfrac{}{A, A \to B \vdash_I A \to B}(Axiom) \qquad \dfrac{}{A, A \to B \vdash_I A}(Axiom)}{A, A \to B \vdash_I B}(Elim)}{A \vdash_I (A \to B) \to B}(Intro)}{\vdash_I A \to ((A \to B) \to B)}(Intro)$$

In the context of λ-calculos it holds:

$$\vdash \lambda_{x:A}.\lambda_{y:A \to B}.(y \ \ x) : A \to ((A \to B) \to B)$$

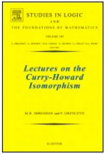
proofs as programs - Curry-Howard isomorphism

Example. **Peirce's Law**: (PL) $((A \rightarrow B) \rightarrow A) \rightarrow A$
Holds in the classical logic, but not in the intuitionistic logic!

proofs as programs - Curry-Howard isomorphism

Isomorphism (Curry-Howard)

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

References: [Hin97], , [Sim00], ...

STUDIES IN LOGIC AND THE FOUNDATIONS OF MATHEMATICS

Lectures on the Curry-Howard Isomorphism

Natural deduction

Table: Natural deduction: inference rules

introduction	elimination
$\dfrac{\varphi \quad \psi}{\varphi \wedge \psi}\ (\wedge i)$	$\dfrac{\varphi \wedge \psi}{\varphi}\ (\wedge e_r) \qquad \dfrac{\varphi \wedge \psi}{\psi}\ (\wedge e_l)$
$\dfrac{\varphi}{\varphi \vee \psi}\ (\vee i_r) \qquad \dfrac{\psi}{\varphi \vee \psi}\ (\vee i_l)$	$\dfrac{\varphi \vee \psi \quad \begin{matrix}[\varphi]^u \\ \vdots \\ \chi\end{matrix} \quad \begin{matrix}[\psi]^v \\ \vdots \\ \chi\end{matrix}}{\chi}\ (\vee e), u, v$
$\dfrac{\begin{matrix}[\varphi]^u \\ \vdots \\ \psi\end{matrix}}{\varphi \to \psi}\ (\to i), u$	$\dfrac{\varphi \quad \varphi \to \psi}{\psi}\ (\to e)$

Natural deduction

Table: Natural deduction: inference rules

introduction	elimination
$\dfrac{\begin{matrix}[\varphi]^u \\ \vdots \\ \bot\end{matrix}}{\neg\varphi}\ (\neg i), u$	$\dfrac{\varphi \quad \neg\varphi}{\bot}\ (\neg e)$ $\dfrac{\bot}{\varphi}\ (\bot e)$ $\dfrac{\neg\neg\varphi}{\varphi}\ (\neg\neg)$
$\dfrac{}{t = t}\ (= i)$	$\dfrac{t_1 = t_2 \quad \varphi[x/t_1]}{\varphi[x/t_2]}\ (= e)$

Natural deduction

Table: NATURAL DEDUCTION: INFERENCE RULES

introduction	elimination
$\dfrac{\boxed{\begin{array}{c} y \text{ independente} \\ \vdots \\ \varphi[x/y] \end{array}}}{\forall x\ \varphi}\ (\forall i)$	$\dfrac{\forall x\ \varphi}{\varphi[x/t]}\ (\forall e)$
$\dfrac{\varphi[x/t]}{\exists x\ \varphi}\ (\exists i)$	$\dfrac{\exists x\ \varphi \quad \boxed{\begin{array}{c} [\varphi[x/y]]^u \\ y \text{ indep.} \vdots \\ \chi \end{array}}}{\chi}\ (\exists e), u$

An example of natural deduction

1. Δ_1:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{[\neg\varphi[x/y]]^v}{\exists x\neg\varphi}\,(\exists i)\quad[\neg\exists x\neg\varphi]^u}{\bot}\,(\neg e)}{\varphi[x/y]}\,(PBC),v}{\forall x\,\varphi}\,(\forall i)\qquad[\neg\forall x\,\varphi]^w}{\bot}\,(\neg e)}{\exists x\,\neg\varphi}\,(PBC),u}{\neg\forall x\,\varphi\rightarrow\exists x\,\neg\varphi}\,(\rightarrow i),w$$

2. Δ_2:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{[\forall x\,\varphi]^v}{\varphi[x/y]}\,(\forall e)\quad[\neg\varphi[x/y]]^w}{\bot}\,(\neg e)\quad[\exists x\,\neg\varphi]^u}{\bot}\,(\exists e),w}{\neg\forall x\,\varphi}\,(\neg i),v}{\exists x\,\neg\varphi\rightarrow\neg\forall x\,\varphi}\,(\rightarrow i),u$$

Gentzen Systems

Table: GENTZEN SYSTEMS: INFERENCE RULES

Left rules	Right rules
Axioms	
$\dfrac{}{A \vdash A}\ (Ax)$	$\dfrac{}{\bot \vdash}\ (L\bot)$
Structural rules	
$\dfrac{\Gamma \vdash \Delta}{A, \Gamma \vdash \Delta}\ (LW)$	$\dfrac{\Gamma \vdash \Delta}{\Gamma \vdash \Delta, A}\ (RW)$
$\dfrac{A, A, \Gamma \vdash \Delta}{A, \Gamma \vdash \Delta}\ (LC)$	$\dfrac{\Gamma \vdash \Delta, A, A}{\Gamma \vdash \Delta, A}\ (RC)$

Gentzen Systems

Table: GENTZEN SYSTEMS: INFERENCE RULES

Left rules	Right rules
Logical rules	
$\dfrac{A_i, \Gamma \vdash \Delta}{A_0 \wedge A_1, \Gamma \vdash \Delta}\ (L\wedge),\ (i = 0, 1)$	$\dfrac{\Gamma \vdash \Delta, A \qquad \Gamma \vdash \Delta, B}{\Gamma \vdash \Delta, A \wedge B}\ (R\wedge)$
$\dfrac{A, \Gamma \vdash \Delta \qquad B, \Gamma \vdash \Delta}{A \vee B, \Gamma \vdash \Delta}\ (L\vee)$	$\dfrac{\Gamma \vdash \Delta, A_i}{\Gamma \vdash \Delta, A_0 \vee A_1}\ (R\vee),\ (i = 0, 1)$
$\dfrac{\Gamma \vdash \Delta, A \qquad B, \Gamma \vdash \Delta}{A \rightarrow B, \Gamma \vdash \Delta}\ (L \rightarrow)$	$\dfrac{A, \Gamma \vdash \Delta, B}{\Gamma \vdash \Delta, A \rightarrow B}\ (R \rightarrow)$

Gentzen Systems

Table: GENTZEN SYSTEMS: INFERENCE RULES

Left rules	Right rules
Logical rules	
$\dfrac{A[x/t], \Gamma \vdash \Delta}{\forall x A, \Gamma \vdash \Delta}\ (L\forall)$	$\dfrac{\Gamma \vdash \Delta, A[x/y]}{\Gamma \vdash \Delta, \forall x A}\ (R\forall),\ y \notin FV(\Gamma, \Delta)$
$\dfrac{A[x/y], \Gamma \vdash \Delta}{\exists x A, \Gamma \vdash \Delta}\ (L\exists),\ y \notin FV(\Gamma, \Delta)$	$\dfrac{\Gamma \vdash \Delta, A[x/t]}{\Gamma \vdash \Delta, \exists x A}\ (R\exists)$

An example of deduction à la Gentzen

$$
\begin{prooftree}
\AxiomC{}
\RightLabel{(Ax)}
\UnaryInfC{$\neg\varphi[x/y] \vdash \neg\varphi[x/y]$}
\RightLabel{$(R\exists)$}
\UnaryInfC{$\neg\varphi[x/y] \vdash \exists x\, \neg\varphi$}
\RightLabel{LW}
\UnaryInfC{$\neg\exists x\, \neg\varphi,\ \ \neg\varphi[x/y] \vdash \exists x\, \neg\varphi$}
\AxiomC{}
\RightLabel{(Ax)}
\UnaryInfC{$\neg\exists x\, \neg\varphi \vdash \neg\exists x\, \neg\varphi$}
\RightLabel{(LW)}
\UnaryInfC{$\neg\exists x\, \neg\varphi,\ \ \neg\varphi[x/y] \vdash \neg\exists x\, \neg\varphi$}
\RightLabel{$(R\wedge)$}
\BinaryInfC{$\neg\exists x\, \neg\varphi,\ \ \neg\varphi[x/y] \vdash \exists x\, \neg\varphi \wedge \neg\exists x\, \neg\varphi$}
\RightLabel{$(R \rightarrow)$}
\UnaryInfC{$\neg\exists x\, \neg\varphi \vdash \neg\varphi[x/y] \rightarrow \bot$}
\RightLabel{$(R\forall)$}
\UnaryInfC{$\neg\exists x\, \neg\varphi \vdash \forall x\, \varphi$}
\RightLabel{(LW)}
\UnaryInfC{$\neg\forall x\, \varphi,\ \ \neg\exists x\, \neg\varphi \vdash \forall x\, \varphi$}
\AxiomC{}
\RightLabel{(Ax)}
\UnaryInfC{$\neg\forall x\, \varphi \vdash \neg\forall x\, \varphi$}
\RightLabel{(LW)}
\UnaryInfC{$\neg\forall x\, \varphi,\ \ \neg\exists x\, \neg\varphi \vdash \neg\forall x\, \varphi$}
\RightLabel{$(R\wedge)$}
\BinaryInfC{$\neg\forall x\, \varphi,\ \ \neg\exists x\, \neg\varphi \vdash \forall x\, \varphi \wedge \neg\forall x\, \varphi$}
\RightLabel{$(R \rightarrow)$}
\UnaryInfC{$\neg\forall x\, \varphi \vdash \neg\exists x\, \neg\varphi \rightarrow \bot$}
\RightLabel{$(R \rightarrow)$}
\UnaryInfC{$\vdash \neg\forall x\, \varphi \rightarrow \exists x\, \neg\varphi$}
\end{prooftree}
$$

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 Γ and Δ are finite sequences of formulae, with the usual Gentzen semantics.

Sequent calculus

- Sequents of the form: $\Gamma \vdash \Delta$.
  - Assuming Γ and Δ *derivable*.
  - $A_1, A_2, \dots, A_n \vdash B_1, B_2, \dots, B_m$ interpreted as $A_1 \wedge A_2 \wedge \dots \wedge A_n \vdash B_1 \vee B_2 \vee \dots \vee B_m$.
- Inference rules
  - Premises and conclusions are simultaneously constructed.
  - Example: $\dfrac{\Gamma \vdash \Delta}{\Gamma_1 \vdash \Delta_1}$
- Goal: $\vdash \Delta$.

Sequent calculus in PVS

- Representation of $A_1, A_2, \ldots, A_n \vdash B_1, B_2, \ldots, B_m$:

$$\begin{array}{l}
\texttt{[-1]}\;\; A_1 \\
\quad\vdots \\
\texttt{[-n]}\;\; A_n \\
\texttt{|----------} \\
\texttt{[1]}\;\; B_1 \\
\quad\vdots \\
\texttt{[n]}\;\; B_n
\end{array}$$

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

Some inference rules in PVS

- Structural:

$$\boxed{\frac{\Gamma_1 \vdash \Delta_1}{\Gamma_2 \vdash \Delta_2}\mathbf{W},\ \text{if } \Gamma_1 \subseteq \Gamma_2 \text{ e } \Delta_1 \subseteq \Delta_2}$$

- Propositional:

$$\boxed{\frac{}{\Gamma, A \vdash A, \Delta}\mathbf{Ax}} \quad \boxed{\frac{}{\Gamma, FALSE \vdash \Delta}\mathbf{FALSE}\vdash}$$

$$\boxed{\frac{}{\Gamma \vdash TRUE, \Delta}\vdash\mathbf{TRUE}}$$

Some inference rules in PVS

- Cut:
  - Corresponds to the case proof command.

$$\boxed{\dfrac{\Gamma, A \vdash \Delta \qquad \Gamma \vdash A, \Delta}{\Gamma \vdash \Delta}\,\textbf{Cut}}$$

- Conditional: `IF-THEN-ELSE`.

$$\boxed{\dfrac{\Gamma, A, B \vdash \Delta \qquad \Gamma, C \vdash A, \Delta}{\Gamma, \mathtt{IF}(A, B, C) \vdash \Delta}\,\mathtt{IF}\vdash}$$

$$\boxed{\dfrac{\Gamma, A \vdash B, \Delta \qquad \Gamma \vdash A, C, \Delta}{\Gamma \vdash \mathtt{IF}(A, B, C)\Delta}\,\vdash \mathtt{IF}}$$

Programs versus demonstrations

Example: greatest common divisor *gcd*

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

(Detail: "n MOD m" is computed as "$(n - m)$ MOD m)

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

algorithm

Programs versus demonstrations

$$\underbrace{gcd(6,4) \rightarrow gcd(2,4) \rightarrow gcd(4,2) \rightarrow gcd(2,2) \rightarrow gcd(0,2) \rightarrow gcd(2,0) \rightarrow \cdots}_{\text{problem: infinite loop}}$$

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 $m > 0$.
$\Rightarrow gcd(m,n)$ 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 $gcd(m, n)$
    if $m = 0$ then $n$
    else $(** m > 0 **)$
        if $m < n$ then $gcd(n, m)$
        else $(** m > 0 \& m \geq n **)$
            if $n = 0$ then $m$
            else $(** m > 0 \& n > 0 \& m \geq n **)$
                $gcd(m - n, n)$
End procedure
```

Program extracted from the proved correct specification

Example in PVS: *gcd* extended to $\mathbb{Z} \times \mathbb{Z}$

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

$$\underbrace{\textbf{Theorem } \text{[Euclid } \mathbb{Z}^2\text{]} \forall m, n \neq 0 \in \mathbb{Z}, gcd(m, n) = gcd(m, m \text{ MOD } n)}_{\text{extension' idea}}$$

(Detail: "n MOD m" is computed as "$(n - m)$ MOD m)

Example in PVS: *gcd* extended to $\mathbb{Z} \times \mathbb{Z}$

```
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
```

$\underbrace{\qquad\qquad\qquad\qquad}_{\text{algorithm extended to } \mathbb{Z}^2}$

Example in PVS: *gcd* $: \mathbb{Z} \times \mathbb{Z} \rightarrow \mathbb{N}$ Executable code

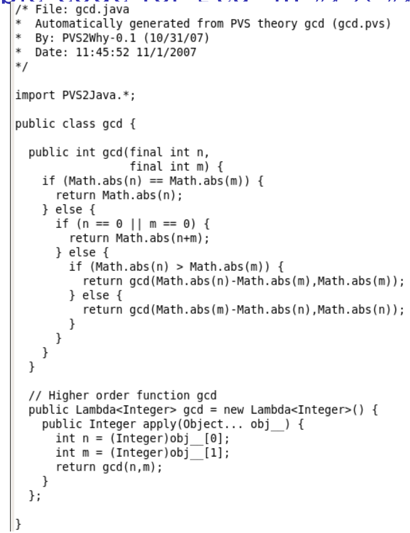
- Specification & verification in PVS
- Executable code extracted from the proved correct specification - Muñoz's system `PVSWhy`

Formalisation of the correctness of *gcd*

Quantitative Information

Theory	L. Specification	L. Proof	Theorems	TCCs	S. Specification	S. Proof
`gcd`	94	1665	21	6	3.2K	74k
	94	1665	21	6	3.2K	74K

Executable code for *gcd* in $\mathbb{Z} \times \mathbb{Z}$ extracted with `PVSWhy`



```
/* 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) {
    if (Math.abs(n) == Math.abs(m)) {
      return Math.abs(n);
    } else {
      if (n == 0 || m == 0) {
        return Math.abs(n+m);
      } else {
        if (Math.abs(n) > Math.abs(m)) {
          return gcd(Math.abs(n)-Math.abs(m),Math.abs(m));
        } else {
          return gcd(Math.abs(m)-Math.abs(n),Math.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

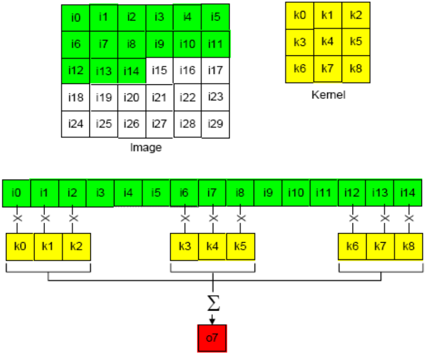
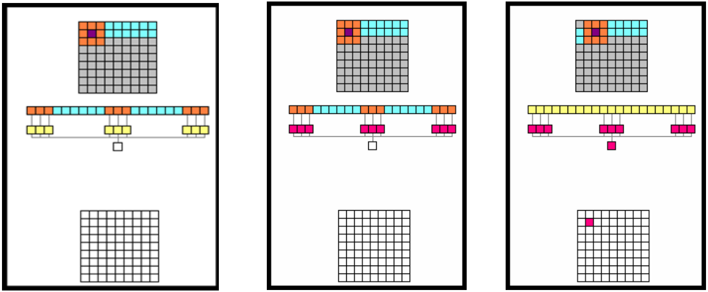


Figure: Wong, Jasiunas & Kearney 2D convolution [WJK05]

Formalisation of the logical correctness of a simple 2D convolution



Implementation of WJK-Convolution in FPGAs
Departamento Engenharia Mecatrônica/UnB

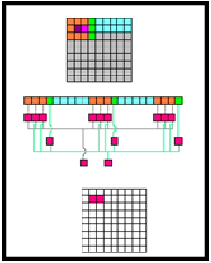
Formalisation of the logical correctness of an improved 2D convolution

a	a	a
b	b	b
c	c	c

a	a	a	a	a
b	b	b	b	b
c	c	c	c	c
d	d	d	d	d
e	e	e	e	e

a	a	a	a	a	a	a
b	b	b	b	b	b	b
c	c	c	c	c	c	c
d	d	d	d	d	d	d
e	e	e	e	e	e	e
f	f	f	f	f	f	f
g	g	g	g	g	g	g

Máscaras



Implementation Y-Convolution in FPGAs
(J.Yudi) Departamento Engenharia Mecatrônica/UnB

Formalisation of the logical correctness of a simple 2D convolution

Quantitative Information

Theory	L. Specification	L. Proof	Theorems	TCCs	T. Specification	T. Proof
`image_masks`	194	3788	75	64	7.8K	78K
`fin_seq_extra`	162	1612	62	29	7K	179k
	356	5400	137	93	14.8K	257K

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]);
  - aplication 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]).

References

M. Ayala-Rincón, C. H. Llanos, R. P. Jacobi, and R. W. Hartenstein.
Prototyping time- and space-efficient computations of algebraic operations over dynamically reconfigurable systems modeled by rewriting-logic.
ACM Trans. Design Autom. Electr. Syst., 11(2):251–281, 2006.

D. Dolev and A. C. Yao.
On the Security of Public Key Protocols.
IEEE. T. on Information Theory, 29(2):198–208, 1983.

A. L. Galdino and M. Ayala-Rincón.
A Formalization of the Knuth-Bendix(-Huet) Critical Pair Theorem.
J. of Automated Reasoning, 2010.
Springer Online First doi 10.1007/s10817-010-9165-2.

A.L. Galdino, C. Mu noz, and M. Ayala-Rincón.
Formal Verification of an Optimal Air Traffic Conflict Detection and Recovery Algorithm.
In *Proc. WoLLIC 2007*, volume 4576 of *Lecture Notes in Computer Science*, pages 177–188, 2007.

J.R. Hindley.
Basic Simple Type Theory.
Number 42 in Cambridge Tracts in Theoretical Computer Science. Cambridge, 1997.

G. Lowe.
An Attack on the Needham-Schroeder Public-Key Authentication Protocol.
Information Processing Letters, 56(3):131–133, 1995.

References

G. Lowe.
Breaking and Fixing the Needham-Schroeder Public-Key Protocol Using FDR.
Software - Concepts and Tools, 17(3):93–102, 1996.

R.B. Nogueira, A. Nascimento, F. L.C. de Moura, and M. Ayala-Rincón.
Formalization of Security Proofs Using PVS in the Dolev-Yao Model.
In *Proc. 6th Computability in Europe - Algorithms, Proofs and Processes*, 2010.

D.N. Sobrinho and M. Ayala-Rincón.
Reduction of the Intruder Deduction Problem into Equational Elementary Deduction for Electronic Purse Protocols with Blind Signatures.
In *Proc. WoLLIC 2010*, volume 6188 of *Lecture Notes in Computer Science*, pages 218–231, 2010.

H. Simmons.
Derivation and Computation: taking the Curry-Howard correspondence seriously.
Number 51 in Cambridge Tracts in Theoretical Computer Science. Cambridge, 2000.

S.C. Wong, M. Jasiunas, and D. Kearney.
Fast 2D Convolution Using Reconfigurable Computing.
In *Proceedings of the Eighth International Symposium on Signal Processing and Its Applications*, pages 791–794, 2005.