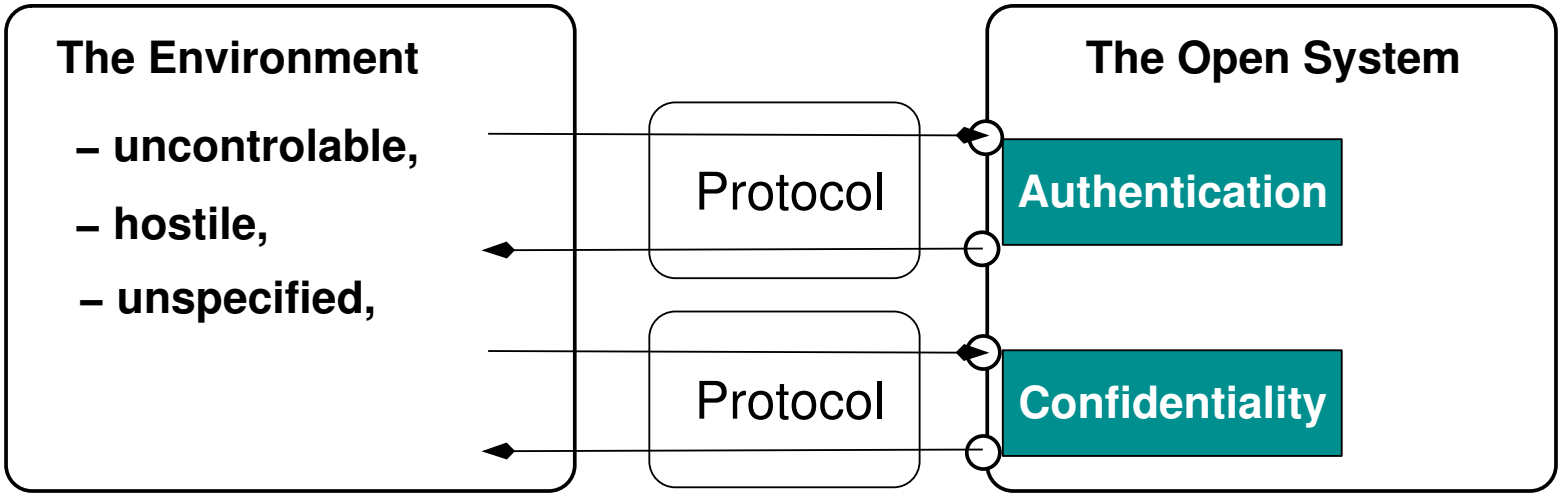
Certification of Cryptographic Protocols by Model-checking and Proof Conretization

R. Janvier, Y. Lakhnech, **M. Périn**

VERIMAG Lab.
Grenoble, France

4 April 2006
Workshop on Innovative Techniques for Certification of Embedded Systems

Applications of cryptographic protocols



The open system *(e.g., a smart card)* ensures security based on the assumptions provided by the cryptographic protocols used in exchange with the environment *(e.g., the smart card interface)*.

Cryptographic protocols

are used to ensure

authentication, non-repudiation, anonymity,...
almost all rely on the **secrecy property**

in a hostile environment

a powerful intruder controls the network: he can listen, intercept, replay, forge fake messages from existing ones

Dolev and Yao's inference rules define the deduction **capacities of the intruder**, *e.g, he can only decipher an encrypted message if he knows the decryption key*

We assume perfect cryptography and we are interested in **logical flaws** of protocols.

We do not consider attack on cryptographic algorithms (the mathematical level).

Cryptographic protocols

are used to ensure

authentication, non-repudiation, anonymity,...
almost all rely on the **secrecy property**

in a hostile environment

a powerful intruder controls the network: he can listen, intercept, replay, forge fake messages from existing ones

Dolev and Yao's inference rules define the deduction **capacities of the intruder**, *e.g, he can only decipher an encrypted message if he knows the decryption key*

We assume perfect cryptography and we are interested in **logical flaws** of protocols.

We do not consider attack on cryptographic algorithms (the mathematical level).

Cryptographic protocols

are used to ensure

authentication, non-repudiation, anonymity,...
almost all rely on the **secrecy property**

in a hostile environment

a powerful intruder controls the network: he can listen, intercept, replay, forge fake messages from existing ones

Dolev and Yao's inference rules define the deduction **capacities of the intruder**, *e.g, he can only decipher an encrypted message if he knows the decryption key*

We assume perfect cryptography and we are interested in **logical flaws** of protocols.

We do not consider attack on cryptographic algorithms (the mathematical level).

Needham-Schroeder's authentication protocol

The goal of this protocol

Mutual authentication of principals A and B based on exchange of a shared secret: the randomly generated number N_B

Needham-Schroeder	Needham-Schroeder-Lowe	Bad correction

1	$A \rightarrow B$:			$!\{A, N_A\}_{k_B}$	
2	$B \rightarrow A$:	$?\{x, n_a\}_{k_B}$	$\rightarrow$	$!\{n_a, N_B\}_{k_x}$	*attack*
2	$B \rightarrow A$:	$?\{x, n_a\}_{k_B}$	$\rightarrow$	$!\{B, (n_a, N_B)\}_{k_x}$	*safe*
2	$B \rightarrow A$:	$?\{x, n_a\}_{k_B}$	$\rightarrow$	$!\{n_a, (N_B, B)\}_{k_x}$	*attack*
3	$A \rightarrow B$:	$?\{N_A, n_b\}_{k_A}$	$\rightarrow$	$!\{n_b\}_{k_B}$	Man in the middle
3	$A \rightarrow B$:	$?\{B, (N_A, n_b)\}_{k_A}$	$\rightarrow$	$!\{n_b\}_{k_B}$	
3	$A \rightarrow B$:	$?\{N_A, \underbrace{(n_b, B)}_{n_b}\}_{k_A}$	$\rightarrow$	$!\{n_b\}_{k_B}$	Type attack + m.i.m

Needham-Schroeder's authentication protocol

The goal of this protocol

Mutual authentication of principals A and B based on exchange of a shared secret: the randomly generated number N_B

Needham-Schroeder | Needham-Schroeder-Lowe | Bad correction

1	$A \rightarrow B$:			$!\{A, N_A\}_{k_B}$	
2	$B \rightarrow A$:	$?\{x, n_a\}_{k_B}$	$\rightarrow$	$!\{n_a, N_B\}_{k_x}$	*attack*
2	$B \rightarrow A$:	$?\{x, n_a\}_{k_B}$	$\rightarrow$	$!\{B, (n_a, N_B)\}_{k_x}$	*safe*
2	$B \rightarrow A$:	$?\{x, n_a\}_{k_B}$	$\rightarrow$	$!\{n_a, (N_B, B)\}_{k_x}$	*attack*
3	$A \rightarrow B$:	$?\{N_A, n_b\}_{k_A}$	$\rightarrow$	$!\{n_b\}_{k_B}$	Man in the middle
3	$A \rightarrow B$:	$?\{B, (N_A, n_b)\}_{k_A}$	$\rightarrow$	$!\{n_b\}_{k_B}$	
3	$A \rightarrow B$:	$?\{N_A, \underbrace{(n_b, B)}_{n_b}\}_{k_A}$	$\rightarrow$	$!\{n_b\}_{k_B}$	Type attack + m.i.m

Needham-Schroeder's authentication protocol

The goal of this protocol

Mutual authentication of principals A and B based on exchange of a shared secret: the randomly generated number N_B

Needham-Schroeder	Needham-Schroeder-Lowe	Bad correction

1	$A \rightarrow B$:			$!\{A, N_A\}_{k_B}$	
2	$B \rightarrow A$:	$?\{x, n_a\}_{k_B}$	$\rightarrow$	$!\{n_a, N_B\}_{k_x}$	*attack*
2	$B \rightarrow A$:	$?\{x, n_a\}_{k_B}$	$\rightarrow$	$!\{B, (n_a, N_B)\}_{k_x}$	*safe*
2	$B \rightarrow A$:	$?\{x, n_a\}_{k_B}$	$\rightarrow$	$!\{n_a, (N_B, B)\}_{k_x}$	*attack*
3	$A \rightarrow B$:	$?\{N_A, n_b\}_{k_A}$	$\rightarrow$	$!\{n_b\}_{k_B}$	Man in the middle
3	$A \rightarrow B$:	$?\{B, (N_A, n_b)\}_{k_A}$	$\rightarrow$	$!\{n_b\}_{k_B}$	
3	$A \rightarrow B$:	$?\{N_A, \underbrace{(n_b, B)}_{n_b}\}_{k_A}$	$\rightarrow$	$!\{n_b\}_{k_B}$	Type attack + m.i.m

Needham-Schroeder's authentication protocol

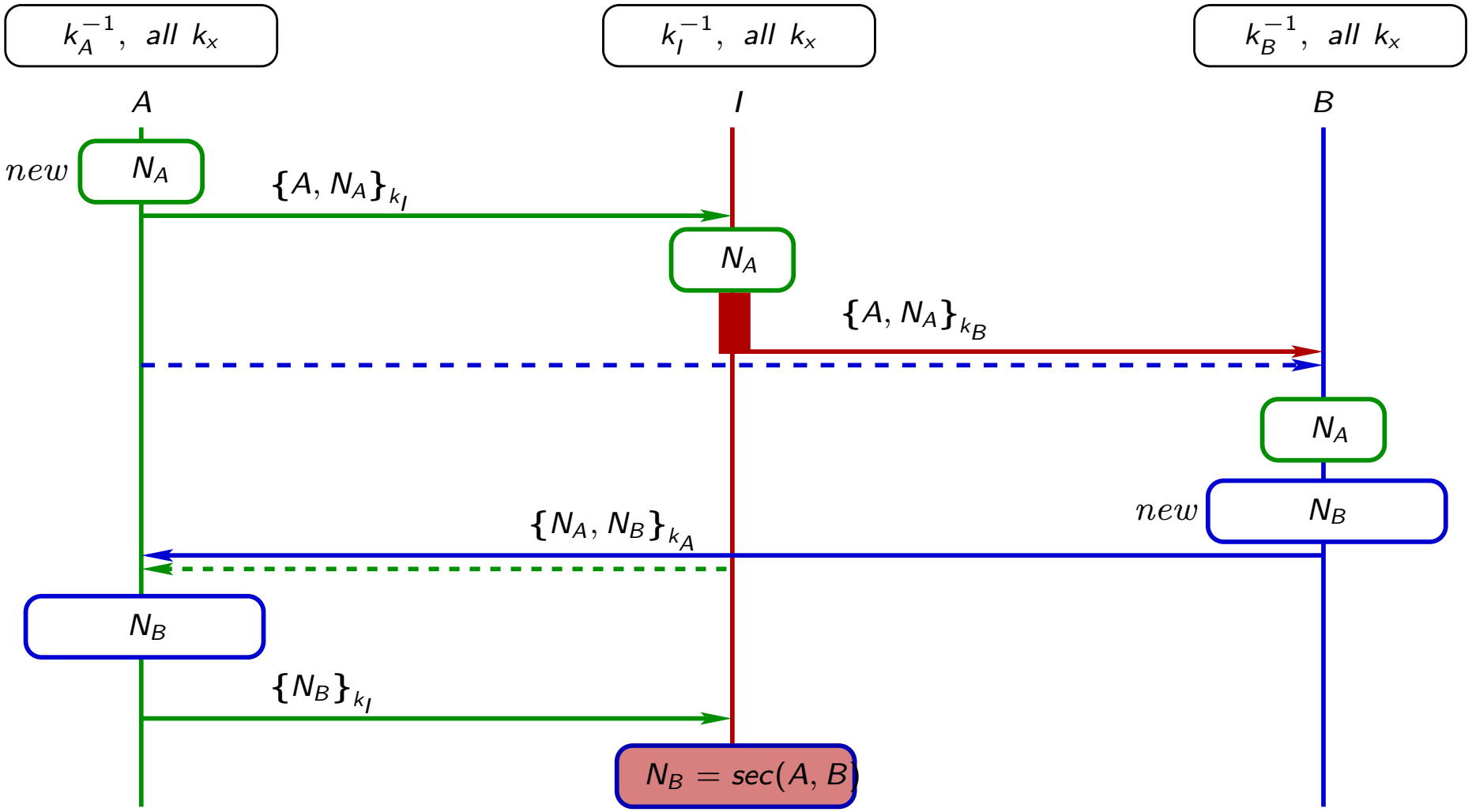
The goal of this protocol

Mutual authentication of principals A and B based on exchange of a shared secret: the randomly generated number N_B

Needham-Schroeder	Needham-Schroeder-Lowe	Bad correction

1	$A \rightarrow B$:			$!\{A, N_A\}_{k_B}$	
2	$B \rightarrow A$:	$?\{x, n_a\}_{k_B}$	$\rightarrow$	$!\{n_a, N_B\}_{k_x}$	*attack*
2	$B \rightarrow A$:	$?\{x, n_a\}_{k_B}$	$\rightarrow$	$!\{B, (n_a, N_B)\}_{k_x}$	*safe*
2	$B \rightarrow A$:	$?\{x, n_a\}_{k_B}$	$\rightarrow$	$!\{n_a, (N_B, B)\}_{k_x}$	*attack*
3	$A \rightarrow B$:	$?\{N_A, n_b\}_{k_A}$	$\rightarrow$	$!\{n_b\}_{k_B}$	Man in the middle
3	$A \rightarrow B$:	$?\{B, (N_A, n_b)\}_{k_A}$	$\rightarrow$	$!\{n_b\}_{k_B}$	
3	$A \rightarrow B$:	$?\{N_A, \underbrace{(n_b, B)}_{n_b}\}_{k_A}$	$\rightarrow$	$!\{n_b\}_{k_B}$	Type attack + m.i.m

The man in the middle attack on NS's protocol



Needs for **Verification** of Cryptography Protocols

- **Basic components for security functionalities**
- **Very sensitive to minor changes**
- **Error prone and difficult to prove due to the underlying semantics:**
  - A proof requires to consider an **unbounded number of sessions of protocol** running in parallel **against Dolev and Yao's intruder**
  - The Needham-Schroeder's authentication protocol has been used during 17 years before G.Lowe discovered the "man-in-the-middle" attack by model-checking two sessions of the protocol.

Needs for **Verification** of Cryptography Protocols

- **Basic components for security functionalities**
- **Very sensitive to minor changes**
- **Error prone and difficult to prove due to the underlying semantics:**
  - A proof requires to consider an **unbounded number of sessions of protocol** running in parallel **against Dolev and Yao's intruder**
  - The Needham-Schroeder's authentication protocol has been used during 17 years before G.Lowe discovered the "man-in-the-middle" attack by model-checking two sessions of the protocol.

Needs for **Verification** of Cryptography Protocols

- **Basic components for security functionalities**
- **Very sensitive to minor changes**
- **Error prone and difficult to prove due to the underlying semantics:**
  - A proof requires to consider an **unbounded number of sessions of protocol** running in parallel **against Dolev and Yao's intruder**
  - The Needham-Schroeder's authentication protocol has been used during 17 years before G.Lowe discovered the "man-in-the-middle" attack by model-checking two sessions of the protocol.

Needs for **Verification** of Cryptography Protocols

- **Basic components for security functionalities**
- **Very sensitive to minor changes**
- **Error prone and difficult to prove due to the underlying semantics:**
  - A proof requires to consider an **unbounded number of sessions of protocol** running in parallel **against Dolev and Yao's intruder**
  - The Needham-Schroeder's authentication protocol has been used during 17 years before G.Lowe discovered the "man-in-the-middle" attack by model-checking two sessions of the protocol.

Needs for **Verification** of Cryptography Protocols

- **Basic components for security functionalities**
- **Very sensitive to minor changes**
- **Error prone and difficult to prove due to the underlying semantics:**
  - A proof requires to consider an **unbounded number of sessions of protocol** running in parallel **against Dolev and Yao's intruder**
  - The Needham-Schroeder's authentication protocol has been used during 17 years before G.Lowe discovered the "man-in-the-middle" attack by model-checking two sessions of the protocol.

Needs for **Certification** of Cryptography Protocols

Certification aims at reducing the Trusted Computing Base

Don't trust the verification tool

- Verification by (possibly erroneous) model-checker
- using (possibly unsafe) abstractions

Check its verdict

- instrument the verification tool
- produce a verdict that can be checked independently of the verification tool
- verdict = a proof in a formal setting where proof checking is just type checking the proof term
- need only to trust the proof-checker
- HERMES uses the COQ proof-engine and proof-checker

Needs for **Certification** of Cryptography Protocols

Certification aims at reducing the Trusted Computing Base

Don't trust the verification tool

- Verification by (possibly erroneous) model-checker
- using (possibly unsafe) abstractions

Check its verdict

- instrument the verification tool
- produce a verdict that can be checked independently of the verification tool
- verdict = a proof in a formal setting where proof checking is just type checking the proof term
- need only to trust the proof-checker
- HERMES uses the COQ proof-engine and proof-checker

Needs for **Certification** of Cryptography Protocols

Certification aims at reducing the Trusted Computing Base

Don't trust the verification tool

- Verification by (possibly erroneous) model-checker
- using (possibly unsafe) abstractions

Check its verdict

- instrument the verification tool
- produce a verdict that can be checked independently of the verification tool
- verdict = a proof in a formal setting where proof checking is just type checking the proof term
- need only to trust the proof-checker
- HERMES uses the COQ proof-engine and proof-checker

Needs for **Certification** of Cryptography Protocols

Certification aims at reducing the Trusted Computing Base

Don't trust the verification tool

- Verification by (possibly erroneous) model-checker
- using (possibly unsafe) abstractions

Check its verdict

- instrument the verification tool
- produce a verdict that can be checked independently of the verification tool
- verdict = a proof in a formal setting where proof checking is just type checking the proof term
- need only to trust the proof-checker
- HERMES uses the COQ proof-engine and proof-checker

Needs for **Certification** of Cryptography Protocols

Certification aims at reducing the Trusted Computing Base

Don't trust the verification tool

- Verification by (possibly erroneous) model-checker
- using (possibly unsafe) abstractions

Check its verdict

- instrument the verification tool
- produce a verdict that can be checked independently of the verification tool
- verdict = a proof in a formal setting where proof checking is just type checking the proof term
- need only to trust the proof-checker
- HERMES uses the COQ proof-engine and proof-checker

Needs for **Certification** of Cryptography Protocols

Certification aims at reducing the Trusted Computing Base

Don't trust the verification tool

- Verification by (possibly erroneous) model-checker
- using (possibly unsafe) abstractions

Check its verdict

- instrument the verification tool
- produce a verdict that can be checked independently of the verification tool
- verdict = a proof in a formal setting where proof checking is just type checking the proof term
- need only to trust the proof-checker
- HERMES uses the COQ proof-engine and proof-checker

Needs for **Certification** of Cryptography Protocols

Certification aims at reducing the Trusted Computing Base

Don't trust the verification tool

- Verification by (possibly erroneous) model-checker
- using (possibly unsafe) abstractions

Check its verdict

- instrument the verification tool
- produce a verdict that can be checked independently of the verification tool
- verdict = a proof in a formal setting where proof checking is just type checking the proof term
- need only to trust the proof-checker
- HERMES uses the COQ proof-engine and proof-checker

Needs for **Certification** of Cryptography Protocols

Certification aims at reducing the Trusted Computing Base

Don't trust the verification tool

- Verification by (possibly erroneous) model-checker
- using (possibly unsafe) abstractions

Check its verdict

- instrument the verification tool
- produce a verdict that can be checked independently of the verification tool
- verdict = a proof in a formal setting where proof checking is just type checking the proof term
- need only to trust the proof-checker
- HERMES uses the COQ proof-engine and proof-checker

Needs for **Certification** of Cryptography Protocols

Certification aims at reducing the Trusted Computing Base

Don't trust the verification tool

- Verification by (possibly erroneous) model-checker
- using (possibly unsafe) abstractions

Check its verdict

- instrument the verification tool
- produce a verdict that can be checked independently of the verification tool
- verdict = a proof in a formal setting where proof checking is just type checking the proof term
- need only to trust the proof-checker
- HERMES uses the COQ proof-engine and proof-checker

The principle of the HERMES verification tool

Transition 3 of a session between A and I of NS protocol

$$A \rightarrow I : \quad ?\{N_A, n_b\}_{k_A} \rightarrow !\{n_b\}_{k_I} \qquad S$$

Although the key k_A is safe, the message $\{N_A, S\}_{k_A}$ must be secret for it reveals the secret S on transitions of $R = \mathscr{P} \cup DY$.

HERMES' operator $pre(R, secrets)$ returns new secrets that reveal the given $secrets$.

The principle of the HERMES verification tool

Transition 3 of a session between A and I of NS protocol

$$A \rightarrow I : \quad ?\{N_A, n_b\}_{k_A} \rightarrow !\{n_b\}_{k_I} \qquad S$$

$$A \rightarrow I : \quad ?\{N_A, S\}_{k_A} \rightarrow \underbrace{!\{S\}_{k_I} \vdash_{DY} S}_{I \ knows \ k_I^{-1}}$$

Although the key k_A is safe, the message $\{N_A, S\}_{k_A}$ must be secret for it reveals the secret S on transitions of $R = \mathscr{P} \cup DY$.

HERMES' operator $pre(R, secrets)$ returns new secrets that reveal the given $secrets$.

The principle of the HERMES verification tool

Transition 3 of a session between A and I of NS protocol

$$A \rightarrow I : \quad ?\{N_A, n_b\}_{k_A} \rightarrow !\{n_b\}_{k_I} \qquad S$$

$$A \rightarrow I : \quad \underbrace{?\{N_A, S\}_{k_A} \rightarrow !\{S\}_{k_I}}_{\text{instance of transition 3 with } n_b = S} \quad \vdash_{DY} \; S$$

Although the key k_A is safe, the message $\{N_A, S\}_{k_A}$ must be secret for it reveals the secret S on transitions of $R = \mathscr{P} \cup DY$.

HERMES' operator $pre(R, secrets)$ returns new secrets that reveal the given $secrets$.

The principle of the HERMES verification tool

Transition 3 of a session between A and I of NS protocol

$$A \rightarrow I : \quad ?\{N_A, n_b\}_{k_A} \rightarrow !\{n_b\}_{k_I} \qquad S$$

$$A \rightarrow I : \quad \underbrace{?\{N_A, S\}_{k_A} \rightarrow !\{S\}_{k_I}}_{\text{instance of transition 3 with } n_b = S} \quad \vdash_{DY} \; S$$

Although the key k_A is safe, the message $\{N_A, S\}_{k_A}$ must be secret for it reveals the secret S on transitions of $R = \mathscr{P} \cup DY$.

HERMES' operator $pre(R, secrets)$ returns new secrets that reveal the given $secrets$.

The principle of the HERMES verification tool

Transition 3 of a session between A and I of NS protocol

$$A \rightarrow I : \quad ?\{N_A, n_b\}_{k_A} \rightarrow !\{n_b\}_{k_I} \qquad S$$

$$A \rightarrow I : \quad \underbrace{?\{N_A, S\}_{k_A} \rightarrow !\{S\}_{k_I}}_{\text{instance of transition 3 with } n_b = S} \quad \vdash_{DY} \; S$$

Although the key k_A is safe, the message $\{N_A, S\}_{k_A}$ must be secret for it reveals the secret S on transitions of $R = \mathscr{P} \cup DY$.

HERMES' operator $pre(R, secrets)$ returns new secrets that reveal the given *secrets*.

The HERMES verification tool

Reachability analysis based on the operator *pre*

1. Backward computation of secrets starting with S until reaching a set S_{HERMES} of secrets stable for *pre*.
2. $S_{HERMES} \cap$ Initial knowledge of $I = \emptyset$

Abstractions used in HERMES

1. abstraction on principals: A, B are honest, the others aren't
   $Others \simeq$ the intruder I
   That induces the abstraction on keys k_A, k_B, k_I
2. unbounded number of sessions abstracted in sessions
   $\mathscr{P}(A,B) \parallel \mathscr{P}(A,I) \parallel \mathscr{P}(I,A) \parallel \mathscr{P}(I,B) \parallel \mathscr{P}(B,I) \parallel \mathscr{P}(I,I)$
3. abstraction on random numbers $N^{AB}, N^{AI}, N^{IA}, N^{IB}, N^{BI}, N^{II}$
4. abstract representation of the infinite set of dangerous messages as patterns of messages

The HERMES verification tool

Reachability analysis based on the operator *pre*

1. Backward computation of secrets starting with S until reaching a set S_{HERMES} of secrets stable for *pre*.
2. $S_{HERMES} \cap$ Initial knowledge of $I = \emptyset$

Abstractions used in HERMES

1. abstraction on principals: A, B are honest, the others aren't
   $Others \simeq$ the intruder I
   That induces the abstraction on keys k_A, k_B, k_I
2. unbounded number of sessions abstracted in sessions
   $\mathscr{P}(A,B) \parallel \mathscr{P}(A,I) \parallel \mathscr{P}(I,A) \parallel \mathscr{P}(I,B) \parallel \mathscr{P}(B,I) \parallel \mathscr{P}(I,I)$
3. abstraction on random numbers $N^{AB}, N^{AI}, N^{IA}, N^{IB}, N^{BI}, N^{II}$
4. abstract representation of the infinite set of dangerous messages as patterns of messages

The HERMES verification tool

Reachability analysis based on the operator *pre*

1. Backward computation of secrets starting with S until reaching a set S_{HERMES} of secrets stable for *pre*.
2. $S_{HERMES} \cap$ Initial knowledge of $I = \emptyset$

Abstractions used in HERMES

1. abstraction on principals: A, B are honest, the others aren't
   $Others \simeq$ the intruder I
   That induces the abstraction on keys k_A, k_B, k_I
2. unbounded number of sessions abstracted in sessions
   $\mathscr{P}(A,B) \parallel \mathscr{P}(A,I) \parallel \mathscr{P}(I,A) \parallel \mathscr{P}(I,B) \parallel \mathscr{P}(B,I) \parallel \mathscr{P}(I,I)$
3. abstraction on random numbers $N^{AB}, N^{AI}, N^{IA}, N^{IB}, N^{BI}, N^{II}$
4. abstract representation of the infinite set of dangerous messages as patterns of messages

The HERMES verification tool

Reachability analysis based on the operator *pre*

1. Backward computation of secrets starting with S until reaching a set S_{HERMES} of secrets stable for *pre*.
2. $S_{HERMES} \cap$ Initial knowledge of $I = \emptyset$

Abstractions used in HERMES

1. abstraction on principals: A, B are honest, the others aren't
   $Others \simeq$ the intruder I
   That induces the abstraction on keys k_A, k_B, k_I
2. unbounded number of sessions abstracted in sessions
   $\mathscr{P}(A,B) \parallel \mathscr{P}(A,I) \parallel \mathscr{P}(I,A) \parallel \mathscr{P}(I,B) \parallel \mathscr{P}(B,I) \parallel \mathscr{P}(I,I)$
3. abstraction on random numbers $N^{AB}, N^{AI}, N^{IA}, N^{IB}, N^{BI}, N^{II}$
4. abstract representation of the infinite set of dangerous messages as patterns of messages

The HERMES verification tool

Reachability analysis based on the operator *pre*

1. Backward computation of secrets starting with S until reaching a set S_{HERMES} of secrets stable for *pre*.
2. $S_{HERMES} \cap$ Initial knowledge of $I = \emptyset$

Abstractions used in HERMES

1. abstraction on principals: A, B are honest, the others aren't
   Others $\simeq$ the intruder I
   That induces the abstraction on keys k_A, k_B, k_I
2. unbounded number of sessions abstracted in sessions
   $\mathscr{P}(A,B) \parallel \mathscr{P}(A,I) \parallel \mathscr{P}(I,A) \parallel \mathscr{P}(I,B) \parallel \mathscr{P}(B,I) \parallel \mathscr{P}(I,I)$
3. abstraction on random numbers $N^{AB}, N^{AI}, N^{IA}, N^{IB}, N^{BI}, N^{II}$
4. abstract representation of the infinite set of dangerous messages as patterns of messages

The HERMES verification tool

Reachability analysis based on the operator *pre*

1. Backward computation of secrets starting with S until reaching a set S_{HERMES} of secrets stable for *pre*.
2. $S_{HERMES} \cap$ Initial knowledge of $I = \emptyset$

Abstractions used in HERMES

1. abstraction on principals: A, B are honest, the others aren't
   $Others \simeq$ the intruder I
   That induces the abstraction on keys k_A, k_B, k_I
2. unbounded number of sessions abstracted in sessions
   $\mathscr{P}(A,B) \parallel \mathscr{P}(A,I) \parallel \mathscr{P}(I,A) \parallel \mathscr{P}(I,B) \parallel \mathscr{P}(B,I) \parallel \mathscr{P}(I,I)$
3. abstraction on random numbers $N^{AB}, N^{AI}, N^{IA}, N^{IB}, N^{BI}, N^{II}$
4. abstract representation of the infinite set of dangerous messages as patterns of messages

The HERMES verification tool

Reachability analysis based on the operator *pre*

1. Backward computation of secrets starting with S until reaching a set S_{HERMES} of secrets stable for *pre*.
2. $S_{HERMES} \cap$ Initial knowledge of $I = \emptyset$

Abstractions used in HERMES

1. abstraction on principals: A, B are honest, the others aren't
   $Others \simeq$ the intruder I
   That induces the abstraction on keys k_A, k_B, k_I
2. unbounded number of sessions abstracted in sessions
   $\mathscr{P}(A,B) \parallel \mathscr{P}(A,I) \parallel \mathscr{P}(I,A) \parallel \mathscr{P}(I,B) \parallel \mathscr{P}(B,I) \parallel \mathscr{P}(I,I)$
3. abstraction on random numbers $N^{AB}, N^{AI}, N^{IA}, N^{IB}, N^{BI}, N^{II}$
4. abstract representation of the infinite set of dangerous messages as patterns of messages

The HERMES verification tool

Reachability analysis based on the operator *pre*

1. Backward computation of secrets starting with S until reaching a set S_{HERMES} of secrets stable for *pre*.
2. $S_{HERMES} \cap$ Initial knowledge of $I = \emptyset$

Abstractions used in HERMES

1. abstraction on principals: A, B are honest, the others aren't
   $Others \simeq$ the intruder I
   That induces the abstraction on keys k_A, k_B, k_I
2. unbounded number of sessions abstracted in sessions
   $\mathscr{P}(A,B) \parallel \mathscr{P}(A,I) \parallel \mathscr{P}(I,A) \parallel \mathscr{P}(I,B) \parallel \mathscr{P}(B,I) \parallel \mathscr{P}(I,I)$
3. abstraction on random numbers $N^{AB}, N^{AI}, N^{IA}, N^{IB}, N^{BI}, N^{II}$
4. abstract representation of the infinite set of dangerous messages as patterns of messages

Certifying the *abstract verdict* of HERMES

Does the abstract system (R^α, I_0^α) satisfies a property $Secret(S^\alpha)$?

Negative verdict

HERMES produces a counter-example. It is a proof of $(R^\alpha, I_0^\alpha) \not\models Secret(S^\alpha)$.

Positive verdict

HERMES generates a **checkable proof** based on the **principle of induction for *pre*-reachability analysis**

$$\frac{\overbrace{S^\alpha \subseteq S^\alpha_{HERMES}}^{\text{trivial proof}} \quad \overbrace{S^\alpha_{HERMES} \cap I_0^\alpha = \emptyset}^{\text{Usage conditions of the protocol}} \quad \overbrace{pre(R^\alpha, S^\alpha_{HERMES}) \subseteq S^\alpha_{HERMES}}^{\text{Correctness of HERMES}}}{pre^*(R^\alpha, S^\alpha) \cap I_0^\alpha = \emptyset \quad \equiv \quad R^{\alpha *}(I_0^\alpha) \cap S^\alpha = \emptyset}$$

Certifying the *abstract verdict* of HERMES

Does the abstract system (R^α, I_0^α) satisfies a property $Secret(S^\alpha)$?

Negative verdict

HERMES produces a counter-example. It is a proof of $(R^\alpha, I_0^\alpha) \not\models Secret(S^\alpha)$.

Positive verdict

HERMES generates a **checkable proof** based on the **principle of induction for *pre*-reachability analysis**

$$\frac{\overbrace{S^\alpha \subseteq S^\alpha_{HERMES}}^{\text{trivial proof}} \qquad \overbrace{S^\alpha_{HERMES} \cap I_0^\alpha = \emptyset}^{\text{Usage conditions of the protocol}} \qquad \overbrace{pre(R^\alpha, S^\alpha_{HERMES}) \subseteq S^\alpha_{HERMES}}^{\text{Correctness of HERMES}}}{pre^*(R^\alpha, S^\alpha) \cap I_0^\alpha = \emptyset \quad \equiv \quad R^{\alpha *}(I_0^\alpha) \cap S^\alpha = \emptyset}$$

Certifying the *abstract verdict* of HERMES

Does the abstract system (R^α, I_0^α) satisfies a property $Secret(S^\alpha)$?

Negative verdict

HERMES produces a counter-example. It is a proof of $(R^\alpha, I_0^\alpha) \not\models Secret(S^\alpha)$.

Positive verdict

HERMES generates a **checkable proof** based on the **principle of induction for *pre*-reachability analysis**

$$\frac{\overbrace{S^\alpha \subseteq S^\alpha_{HERMES}}^{\text{trivial proof}} \quad \overbrace{S^\alpha_{HERMES} \cap I_0^\alpha = \emptyset}^{\text{Usage conditions of the protocol}} \quad \overbrace{pre(R^\alpha, S^\alpha_{HERMES}) \subseteq S^\alpha_{HERMES}}^{\text{Correctness of HERMES}}}{pre^*(R^\alpha, S^\alpha) \cap I_0^\alpha = \emptyset \quad \equiv \quad R^{\alpha *}(I_0^\alpha) \cap S^\alpha = \emptyset}$$

Certifying the *abstract verdict* of HERMES

Does the abstract system (R^α, I_0^α) satisfies a property $Secret(S^\alpha)$?

Negative verdict

HERMES produces a counter-example. It is a proof of $(R^\alpha, I_0^\alpha) \not\models Secret(S^\alpha)$.

Positive verdict

HERMES generates a **checkable proof** based on the **principle of induction for *pre*-reachability analysis**

$$\frac{\overbrace{S^\alpha \subseteq S^\alpha_{HERMES}}^{\text{trivial proof}} \qquad \overbrace{S^\alpha_{HERMES} \cap I_0^\alpha = \emptyset}^{\text{Usage conditions of the protocol}} \qquad \overbrace{pre(R^\alpha, S^\alpha_{HERMES}) \subseteq S^\alpha_{HERMES}}^{\text{Correctness of HERMES}}}{pre^*(R^\alpha, S^\alpha) \cap I_0^\alpha = \emptyset \quad \equiv \quad R^{\alpha *}(I_0^\alpha) \cap S^\alpha = \emptyset}$$

Certifying the *abstract verdict* of HERMES

Does the abstract system (R^α, I_0^α) satisfies a property $Secret(S^\alpha)$?

Negative verdict

HERMES produces a counter-example. It is a proof of $(R^\alpha, I_0^\alpha) \not\models Secret(S^\alpha)$.

Positive verdict

HERMES generates a **checkable proof** based on the **principle of induction for *pre*-reachability analysis**

$$\frac{\overbrace{S^\alpha \subseteq S^\alpha_{HERMES}}^{\text{trivial proof}} \qquad \overbrace{S^\alpha_{HERMES} \cap I_0^\alpha = \emptyset}^{\text{Usage conditions of the protocol}} \qquad \overbrace{pre(R^\alpha, S^\alpha_{HERMES}) \subseteq S^\alpha_{HERMES}}^{\textbf{Correctness of HERMES}}}{pre^*(R^\alpha, S^\alpha) \cap I_0^\alpha = \emptyset \quad \equiv \quad R^{\alpha *}(I_0^\alpha) \cap S^\alpha = \emptyset}$$

Abstract proof of stability

For each transition τ of R^α, for each secret s in $pre(\tau, S^\alpha_{HERMES})$, we have to prove s $\in S^\alpha_{HERMES}$.

Run an instrumented version of HERMES on S^α_{HERMES}

All the attempt to add a new secret s fails, it already belongs to S^α_{HERMES}. The execution path of HERMES that leads to that conclusion gives the argument for proving s $\in S^\alpha_{HERMES}$.

HERMES computations drives the proof

- **Proof of a disjunction:** The evaluation of branching conditions that control the execution path of HERMES indicates which part makes the disjunction true.
- **Proof of a existential property:** requires to exhibit a witness of the property. HERMES records them.
- The reminder of the proof is managed by general proof-tactics.

Abstract proof of stability

For each transition τ of R^α, for each secret $\mathbf{s}$ in $pre(\tau, S^\alpha_{HERMES})$, we have to prove $\mathbf{s} \in S^\alpha_{HERMES}$.

Run an instrumented version of HERMES on S^α_{HERMES}

All the attempt to add a new secret s fails, it already belongs to S^α_{HERMES}. The execution path of HERMES that leads to that conclusion gives the argument for proving $\mathbf{s} \in S^\alpha_{HERMES}$.

HERMES computations drives the proof

- **Proof of a disjunction:** The evaluation of branching conditions that control the execution path of HERMES indicates which part makes the disjunction true.
- **Proof of a existential property:** requires to exhibit a witness of the property. HERMES records them.
- The reminder of the proof is managed by general proof-tactics.

Abstract proof of stability

For each transition τ of R^α, for each secret $\mathbf{s}$ in $pre(\tau, S^\alpha_{HERMES})$, we have to prove $\mathbf{s} \in S^\alpha_{HERMES}$.

Run an instrumented version of HERMES on S^α_{HERMES}

All the attempt to add a new secret s fails, it already belongs to S^α_{HERMES}. The execution path of HERMES that leads to that conclusion gives the argument for proving $\mathbf{s} \in S^\alpha_{HERMES}$.

HERMES computations drives the proof

- **Proof of a disjunction:** The evaluation of branching conditions that control the execution path of HERMES indicates which part makes the disjunction true.
- **Proof of a existential property:** requires to exhibit a witness of the property. HERMES records them.
- The reminder of the proof is managed by general proof-tactics.

Abstract proof of stability

For each transition τ of R^α, for each secret $\mathbf{s}$ in $pre(\tau, S^\alpha_{HERMES})$, we have to prove $\mathbf{s} \in S^\alpha_{HERMES}$.

Run an instrumented version of HERMES on S^α_{HERMES}

All the attempt to add a new secret s fails, it already belongs to S^α_{HERMES}. The execution path of HERMES that leads to that conclusion gives the argument for proving $\mathbf{s} \in S^\alpha_{HERMES}$.

HERMES computations drives the proof

- **Proof of a disjunction:** The evaluation of branching conditions that control the execution path of HERMES indicates which part makes the disjunction true.
- **Proof of a existential property:** requires to exhibit a witness of the property. HERMES records them.
- The reminder of the proof is managed by general proof-tactics.

Abstract proof of stability

For each transition τ of R^α, for each secret $\mathbf{s}$ in $pre(\tau, S^\alpha_{HERMES})$, we have to prove $\mathbf{s} \in S^\alpha_{HERMES}$.

Run an instrumented version of HERMES on S^α_{HERMES}

All the attempt to add a new secret s fails, it already belongs to S^α_{HERMES}. The execution path of HERMES that leads to that conclusion gives the argument for proving $\mathbf{s} \in S^\alpha_{HERMES}$.

HERMES computations drives the proof

- **Proof of a disjunction:** The evaluation of branching conditions that control the execution path of HERMES indicates which part makes the disjunction true.
- **Proof of a existential property:** requires to exhibit a witness of the property. HERMES records them.
- The reminder of the proof is managed by general proof-tactics.

Abstract proof of stability

For each transition τ of R^α, for each secret $\mathbf{s}$ in $pre(\tau, S^\alpha_{HERMES})$, we have to prove $\mathbf{s} \in S^\alpha_{HERMES}$.

Run an instrumented version of HERMES on S^α_{HERMES}

All the attempt to add a new secret s fails, it already belongs to S^α_{HERMES}. The execution path of HERMES that leads to that conclusion gives the argument for proving $\mathbf{s} \in S^\alpha_{HERMES}$.

HERMES computations drives the proof

- **Proof of a disjunction:** The evaluation of branching conditions that control the execution path of HERMES indicates which part makes the disjunction true.
- **Proof of a existential property:** requires to exhibit a witness of the property. HERMES records them.
- The reminder of the proof is managed by general proof-tactics.

Proof based on safeness of the abstraction

$(R,\ S_{HERMES}),\ \alpha,\ (R^{\alpha}, S^{\alpha}_{HERMES})$

(PO_1)

$\alpha(R,\ S_{HERMES}) = (R^{\alpha}, S^{\alpha}_{HERMES})$

$R^{\alpha}, S^{\alpha}_{HERMES}$

∇^{α}

$pre(R^{\alpha}, S^{\alpha}_{HERMES}) \subseteq S^{\alpha}_{HERMES}$

(PO_2)

$pre(R, S_{HERMES}) \subseteq S_{HERMES}$

(PO_1) correct computation of the abstract image of $R,\ S_{HERMES}$

(PO_2) safeness of the abstraction α

$$pre(R^{\alpha}, S^{\alpha}_{HERMES}) \subseteq S^{\alpha}_{HERMES} \Rightarrow pre(R, S_{HERMES}) \subseteq S_{HERMES}$$

Proof based on safeness of the abstraction

$(R,\ S_{HERMES}),\ \alpha,\ (R^{\alpha}, S^{\alpha}_{HERMES})$

(PO_1)

$\alpha(R,\ S_{HERMES}) = (R^{\alpha}, S^{\alpha}_{HERMES})$

$R^{\alpha}, S^{\alpha}_{HERMES}$

∇^{α}

$pre(R^{\alpha}, S^{\alpha}_{HERMES}) \subseteq S^{\alpha}_{HERMES}$

(PO_2)

$pre(R, S_{HERMES}) \subseteq S_{HERMES}$

(PO$_1$) correct computation of the abstract image of $R,\ S_{HERMES}$

(PO$_2$) safeness of the abstraction α

$$pre(R^{\alpha}, S^{\alpha}_{HERMES}) \subseteq S^{\alpha}_{HERMES} \Rightarrow pre(R, S_{HERMES}) \subseteq S_{HERMES}$$

Proof based on safeness of the abstraction

$(R,\ S_{HERMES}),\ \alpha,\ (R^{\alpha}, S^{\alpha}_{HERMES})$

(PO_1)

$\alpha(R,\ S_{HERMES}) = (R^{\alpha}, S^{\alpha}_{HERMES})$

$R^{\alpha}, S^{\alpha}_{HERMES}$

∇^{α}

$pre(R^{\alpha}, S^{\alpha}_{HERMES}) \subseteq S^{\alpha}_{HERMES}$

(PO_2)

$pre(R, S_{HERMES}) \subseteq S_{HERMES}$

(PO$_1$) correct computation of the abstract image of $R,\ S_{HERMES}$

(PO$_2$) safeness of the abstraction α

$$pre(R^{\alpha}, S^{\alpha}_{HERMES}) \subseteq S^{\alpha}_{HERMES} \Rightarrow pre(R, S_{HERMES}) \subseteq S_{HERMES}$$

Concretization of the ∇^α proof provided by HERMES

It avoids proving safeness of the abstraction

by **removing** all references to **the abstraction relation** and abstract domains **from the proof**

Principle of the concretization function $[\![.]\!] : \nabla \to \nabla$

- Abstract constants are replaced by **constant symbols** c **with additional hypothesis:** $[\![C^\alpha]\!] \rightsquigarrow c$ such that $\underbrace{\alpha(c, C^\alpha)}_{\text{concrete predicate}}$
- Abstract equalities are **equivalence relation** on the concrete domain: $[\![=^\alpha]\!] \rightsquigarrow \sim$ with $x \sim y \stackrel{def}{=} \underbrace{\exists C^\alpha \ \alpha(x, C^\alpha) \wedge \alpha(y, C^\alpha)}_{\text{concrete predicate}}$
- Abstract predicates are replaced by **concrete predicates with uniform valuation on the equivalence classes of** $\sim$ $[\![P^\alpha]\!] \rightsquigarrow$ *$\sim$-uniform version of P*

Concretization of the ∇^α proof provided by HERMES

It avoids proving safeness of the abstraction

by **removing** all references to **the abstraction relation** and abstract domains **from the proof**

Principle of the concretization function $[\![.]\!] : \nabla \to \nabla$

- Abstract constants are replaced by **constant symbols** c **with additional hypothesis:** $[\![C^\alpha]\!] \rightsquigarrow c$ such that $\underbrace{\alpha(c, C^\alpha)}_{\text{concrete predicate}}$
- Abstract equalities are **equivalence relation** on the concrete domain: $[\![=^\alpha]\!] \rightsquigarrow \sim$ with $x \sim y \stackrel{def}{=} \underbrace{\exists C^\alpha\ \alpha(x, C^\alpha) \wedge \alpha(y, C^\alpha)}_{\text{concrete predicate}}$
- Abstract predicates are replaced by **concrete predicates with uniform valuation on the equivalence classes of** $\sim$
  $[\![P^\alpha]\!] \rightsquigarrow$ *$\sim$-uniform version of P*

Concretization of the ∇^α proof provided by HERMES

It avoids proving safeness of the abstraction

by **removing** all references to **the abstraction relation** and abstract domains **from the proof**

Principle of the concretization function $[\![.]\!] : \nabla \to \nabla$

- Abstract constants are replaced by **constant symbols** c **with additional hypothesis:** $[\![C^\alpha]\!] \rightsquigarrow c$ such that $\underbrace{\alpha(c, C^\alpha)}_{\text{concrete predicate}}$
- Abstract equalities are **equivalence relation** on the concrete domain: $[\![=^\alpha]\!] \rightsquigarrow \sim$ with $x \sim y \stackrel{def}{=} \underbrace{\exists C^\alpha\ \alpha(x, C^\alpha) \wedge \alpha(y, C^\alpha)}_{\text{concrete predicate}}$
- Abstract predicates are replaced by **concrete predicates with uniform valuation on the equivalence classes of** $\sim$
  $[\![P^\alpha]\!] \rightsquigarrow$ *$\sim$-uniform version of P*

Concretization of the ∇^α proof provided by HERMES

It avoids proving safeness of the abstraction

by **removing** all references to **the abstraction relation** and abstract domains **from the proof**

Principle of the concretization function $[\![.]\!] : \nabla \to \nabla$

- Abstract constants are replaced by **constant symbols** c **with additional hypothesis:** $[\![C^\alpha]\!] \rightsquigarrow c$ such that $\underbrace{\alpha(c, C^\alpha)}_{\textbf{concrete predicate}}$
- Abstract equalities are **equivalence relation** on the concrete domain: $[\![=^\alpha]\!] \rightsquigarrow \sim$ with $x \sim y \stackrel{def}{=} \underbrace{\exists C^\alpha\ \alpha(x, C^\alpha) \wedge \alpha(y, C^\alpha)}_{\textbf{concrete predicate}}$
- Abstract predicates are replaced by **concrete predicates with uniform valuation on the equivalence classes of** $\sim$
  $[\![P^\alpha]\!] \rightsquigarrow$ *$\sim$-uniform version of P*

Concretization of the ∇^α proof provided by HERMES

It avoids proving safeness of the abstraction

by **removing** all references to **the abstraction relation** and abstract domains **from the proof**

Principle of the concretization function $[\![.]\!] : \nabla \to \nabla$

- Abstract constants are replaced by **constant symbols** c **with additional hypothesis:** $[\![C^\alpha]\!] \rightsquigarrow c$ such that $\underbrace{\alpha(c, C^\alpha)}_{\textbf{concrete predicate}}$
- Abstract equalities are **equivalence relation** on the concrete domain: $[\![=^\alpha]\!] \rightsquigarrow \sim$ with $x \sim y \stackrel{def}{=} \underbrace{\exists C^\alpha\ \alpha(x, C^\alpha) \wedge \alpha(y, C^\alpha)}_{\textbf{concrete predicate}}$
- Abstract predicates are replaced by **concrete predicates with uniform valuation on the equivalence classes of** $\sim$
  $[\![P^\alpha]\!] \rightsquigarrow$ *$\sim$-uniform version of P*

Concretization of the ∇^α proof provided by HERMES

It avoids proving safeness of the abstraction

by **removing** all references to **the abstraction relation** and abstract domains **from the proof**

Principle of the concretization function $[\![.]\!] : \nabla \to \nabla$

- Abstract constants are replaced by **constant symbols** c **with additional hypothesis:** $[\![C^\alpha]\!] \rightsquigarrow c$ such that $\underbrace{\alpha(c, C^\alpha)}_{\textbf{concrete predicate}}$
- Abstract equalities are **equivalence relation** on the concrete domain: $[\![=^\alpha]\!] \rightsquigarrow \sim$ with $x \sim y \stackrel{def}{=} \underbrace{\exists C^\alpha\ \alpha(x, C^\alpha) \wedge \alpha(y, C^\alpha)}_{\textbf{concrete predicate}}$
- Abstract predicates are replaced by **concrete predicates with uniform valuation on the equivalence classes of** $\sim$
  $[\![P^\alpha]\!] \rightsquigarrow \sim$*-uniform version of* P

Concretization of the ∇^α proof provided by HERMES

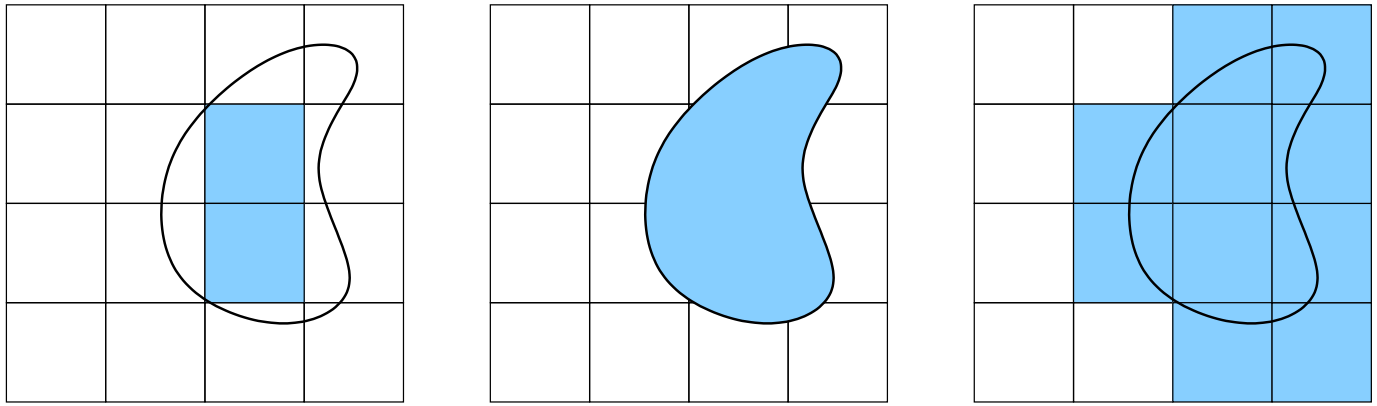
It avoids proving safeness of the abstraction

by **removing** all references to **the abstraction relation** and abstract domains **from the proof**

Principle of the concretization function $[\![.]\!] : \nabla \to \nabla$

- Abstract constants are replaced by **constant symbols** c **with additional hypothesis:** $[\![C^\alpha]\!] \rightsquigarrow c$ such that $\underbrace{\alpha(c, C^\alpha)}_{\textbf{concrete predicate}}$
- Abstract equalities are **equivalence relation** on the concrete domain: $[\![=^\alpha]\!] \rightsquigarrow \sim$ with $x \sim y \stackrel{def}{=} \underbrace{\exists C^\alpha\ \alpha(x, C^\alpha) \wedge \alpha(y, C^\alpha)}_{\textbf{concrete predicate}}$
- Abstract predicates are replaced by **concrete predicates with uniform valuation on the equivalence classes of** $\sim$
  $[\![P^\alpha]\!] \rightsquigarrow$ *$\sim$-uniform version of P*

Uniform valuation of a predicate on equivalence classes



P^- P P^+

uniform-restriction of P

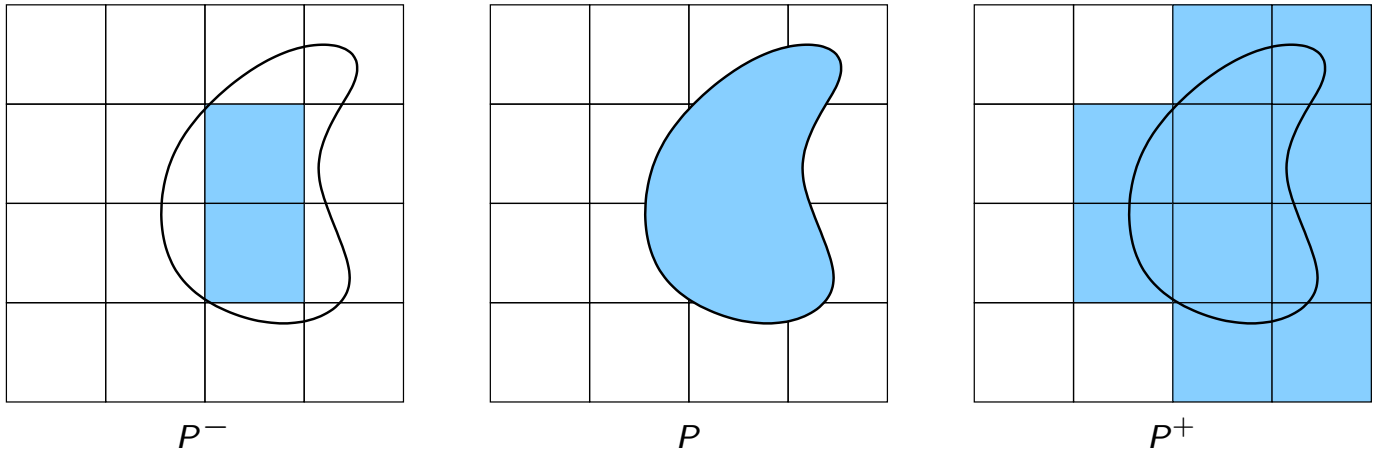
$$P^-(x) \stackrel{def}{=} \underbrace{\forall y,\ y \sim x \Rightarrow P(y)}_{\text{concrete predicate}}$$

uniform-extension of P

$$P^+(x) \stackrel{def}{=} \underbrace{\exists y,\ y \sim x \wedge P(y)}_{\text{concrete predicate}}$$

Remark: $P \Rightarrow P^+$

Uniform valuation of a predicate on equivalence classes



uniform-restriction of P

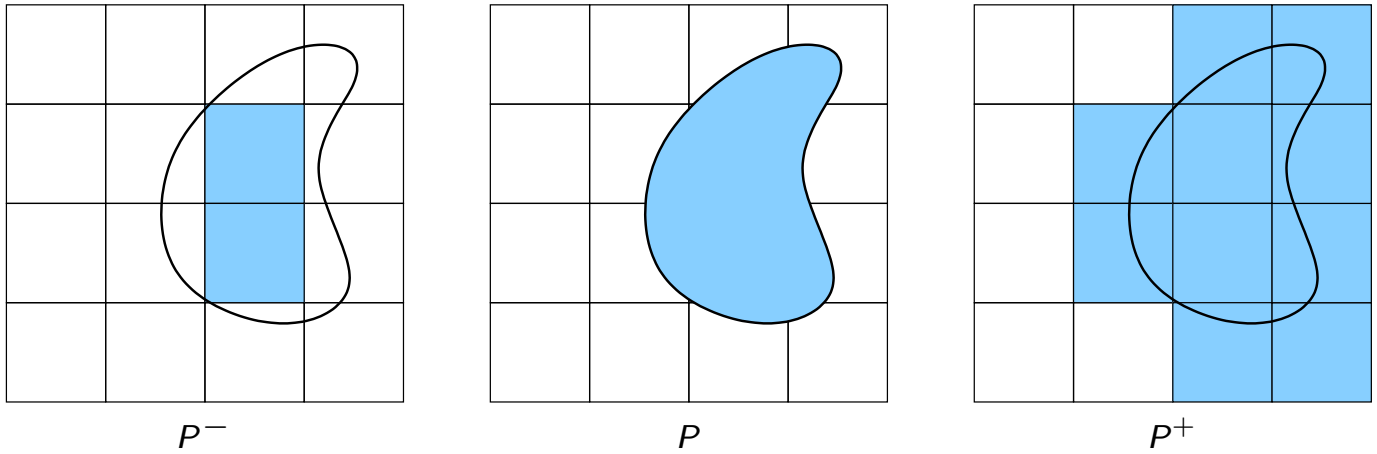
$$P^-(x) \stackrel{def}{=} \underbrace{\forall y,\ y \sim x \Rightarrow P(y)}_{\textbf{concrete predicate}}$$

uniform-extension of P

$$P^+(x) \stackrel{def}{=} \underbrace{\exists y,\ y \sim x \wedge P(y)}_{\textbf{concrete predicate}}$$

Remark: $P \Rightarrow P^+$

Uniform valuation of a predicate on equivalence classes



uniform-restriction of P

$$P^-(x) \stackrel{def}{=} \underbrace{\forall y,\ y \sim x \Rightarrow P(y)}_{\textbf{concrete predicate}}$$

uniform-extension of P

$$P^+(x) \stackrel{def}{=} \underbrace{\exists y,\ y \sim x \wedge P(y)}_{\textbf{concrete predicate}}$$

Remark: $P \Rightarrow P^+$

Proof by concretization

$$R,\ S_{HERMES} = R\ ,\ [\![def.\ S^{\alpha}_{HERMES}]\!]$$

(PO_1) *easy but not fully automatic*

$$R \overset{\ell}{\Rightarrow} expanded\ def.R^{+}$$

$$[\![def.\ R^{\alpha}]\!],\ [\![def.\ S^{\alpha}_{HERMES}]\!] = \overbrace{expanded\ def.\ R^{+}}, [\![def.\ S^{\alpha}_{HERMES}]\!]$$

$[\![\nabla^{\alpha}]\!]$

$$[\![pre(R^{\alpha}, S^{\alpha}_{HERMES}) \subseteq S^{\alpha}_{HERMES}]\!] = R^{+}(s, s') \wedge s' \in S_{HERMES} \Rightarrow s \in S_{HERMES}$$

(PO_2) *trivial and automatic for* $(R \Rightarrow R^{+})$

$$pre(R, S_{HERMES}) \subseteq S_{HERMES} = R(s, s') \wedge s' \in S_{HERMES} \Rightarrow s \in S_{HERMES}$$

Proof by concretization

$$R,\ S_{HERMES} = R\ ,\ [\![def.\ S^{\alpha}_{HERMES}]\!]$$

(PO_1) — *easy but not fully automatic*

$$R \overset{\ell}{\Rightarrow} \text{expanded def.} R^{+}$$

$$[\![def.\ R^{\alpha}]\!],\ [\![def.\ S^{\alpha}_{HERMES}]\!] = \overbrace{\text{expanded def.}\ R^{+}}^{R \overset{\ell}{\Rightarrow} \text{expanded def.} R^{+}},\ [\![def.\ S^{\alpha}_{HERMES}]\!]$$

$[\![\nabla^{\alpha}]\!]$

$$[\![pre(R^{\alpha}, S^{\alpha}_{HERMES}) \subseteq S^{\alpha}_{HERMES}]\!] = R^{+}(s, s') \wedge s' \in S_{HERMES} \Rightarrow s \in S_{HERMES}$$

(PO_2) — *trivial and automatic for* $(R \Rightarrow R^{+})$

$$pre(R, S_{HERMES}) \subseteq S_{HERMES} = R(s, s') \wedge s' \in S_{HERMES} \Rightarrow s \in S_{HERMES}$$

Proof by concretization

$$R,\ S_{HERMES} = R\ ,\ [\![def.\ S^{\alpha}_{HERMES}]\!]$$

(PO_1) *easy but not fully automatic*

$$R \overset{?}{\Rightarrow} expanded\ def.R^{+}$$

$$[\![def.\ R^{\alpha}]\!],\ [\![def.\ S^{\alpha}_{HERMES}]\!] = \overbrace{expanded\ def.\ R^{+}}, [\![def.\ S^{\alpha}_{HERMES}]\!]$$

$[\![\nabla^{\alpha}]\!]$

$$[\![pre(R^{\alpha}, S^{\alpha}_{HERMES}) \subseteq S^{\alpha}_{HERMES}]\!] = R^{+}(s,s') \wedge s' \in S_{HERMES} \Rightarrow s \in S_{HERMES}$$

(PO_2) *trivial and automatic for* $(R \Rightarrow R^{+})$

$$pre(R, S_{HERMES}) \subseteq S_{HERMES} = R(s,s') \wedge s' \in S_{HERMES} \Rightarrow s \in S_{HERMES}$$

Proof by concretization

$$R,\ S_{HERMES} = R\ ,\ [\![def.\ S^{\alpha}_{HERMES}]\!]$$

(PO_1) — *easy but not fully automatic*

$$R \overset{?}{\Rightarrow} expanded\ def.R^{+}$$

$$[\![def.\ R^{\alpha}]\!],\ [\![def.\ S^{\alpha}_{HERMES}]\!] = \overbrace{expanded\ def.\ R^{+}}^{R \overset{?}{\Rightarrow} expanded\ def.R^{+}},\ [\![def.\ S^{\alpha}_{HERMES}]\!]$$

$[\![\nabla^{\alpha}]\!]$

$$[\![pre(R^{\alpha}, S^{\alpha}_{HERMES}) \subseteq S^{\alpha}_{HERMES}]\!] = R^{+}(s,s') \wedge s' \in S_{HERMES} \Rightarrow s \in S_{HERMES}$$

(PO_2) — *trivial and automatic for* $(R \Rightarrow R^{+})$

$$pre(R, S_{HERMES}) \subseteq S_{HERMES} = R(s,s') \wedge s' \in S_{HERMES} \Rightarrow s \in S_{HERMES}$$

Proof by concretization

$$R,\ S_{HERMES} = R\ ,\ [\![def.\ S^{\alpha}_{HERMES}]\!]$$

(PO_1) — *easy but not fully automatic*

$$R \overset{?}{\Rightarrow} expanded\ def.R^{+}$$

$$[\![def.\ R^{\alpha}]\!],\ [\![def.\ S^{\alpha}_{HERMES}]\!] = \overbrace{expanded\ def.\ R^{+}}^{R \overset{?}{\Rightarrow} expanded\ def.R^{+}},\ [\![def.\ S^{\alpha}_{HERMES}]\!]$$

$[\![\nabla^{\alpha}]\!]$

$$[\![pre(R^{\alpha}, S^{\alpha}_{HERMES}) \subseteq S^{\alpha}_{HERMES}]\!] = R^{+}(s,s') \wedge s' \in S_{HERMES} \Rightarrow s \in S_{HERMES}$$

(PO_2) — *trivial and automatic for* $(R \Rightarrow R^{+})$

$$pre(R, S_{HERMES}) \subseteq S_{HERMES} = R(s,s') \wedge s' \in S_{HERMES} \Rightarrow s \in S_{HERMES}$$

Proof by concretization

$$R,\ S_{HERMES} \quad = \quad R \quad , \ [\![def.\ S^{\alpha}_{HERMES}]\!]$$

(PO_1) — *easy but not fully automatic*

$$R \overset{?}{\Rightarrow} expanded\ def.R^{+}$$

$$[\![def.\ R^{\alpha}]\!],\ [\![def.\ S^{\alpha}_{HERMES}]\!] = \overbrace{expanded\ def.\ R^{+}}, \ [\![def.\ S^{\alpha}_{HERMES}]\!]$$

$[\![\nabla^{\alpha}]\!]$

$$[\![pre(R^{\alpha}, S^{\alpha}_{HERMES}) \subseteq S^{\alpha}_{HERMES}]\!] = R^{+}(s, s') \wedge s' \in S_{HERMES} \Rightarrow s \in S_{HERMES}$$

(PO_2) — *trivial and automatic for* $(R \Rightarrow R^{+})$

$$pre(R, S_{HERMES}) \subseteq S_{HERMES} \quad = \quad R(s, s') \wedge s' \in S_{HERMES} \Rightarrow s \in S_{HERMES}$$

Proof by concretization

$$R,\ S_{HERMES} \quad = \quad R \quad , \ [\![def.\ S^{\alpha}_{HERMES}]\!]$$

(PO_1) ***easy but not fully automatic***

$$[\![def.\ R^{\alpha}]\!],\ [\![def.\ S^{\alpha}_{HERMES}]\!] = \overbrace{expanded\ def.\ R^{+}}^{R \overset{?}{\Rightarrow} expanded\ def.R^{+}},\ [\![def.\ S^{\alpha}_{HERMES}]\!]$$

$[\![\nabla^{\alpha}]\!]$

$$[\![pre(R^{\alpha}, S^{\alpha}_{HERMES}) \subseteq S^{\alpha}_{HERMES}]\!] = R^{+}(s, s') \wedge s' \in S_{HERMES} \Rightarrow s \in S_{HERMES}$$

(PO_2) ***trivial and automatic for*** $(R \Rightarrow R^{+})$

$$pre(R, S_{HERMES}) \subseteq S_{HERMES} \quad = \quad R(s, s') \wedge s' \in S_{HERMES} \Rightarrow s \in S_{HERMES}$$

Certification versus Proof-Carrying Code

- **Challenge of PCC: reducing the size of proof and complexity of proof-checking** (not so important for certification)
- **Challenge of certification: reducing the Trusted Computing Base to the proof-checker**
  - HERMES's uses the COQ-engine where proof-checking is type checking
  - PPC places the VCG to the TCB
- Both establish a **checkable relation between the system S, the property φ and the proof ∇** through
  - the Verification Condition Generator in PCC : $VCG(S, \varphi)$
  - extraction of the protocol and the property from HERMES's proof (No need for VCG)

Certification versus Proof-Carrying Code

- **Challenge of PCC: reducing the size of proof and complexity of proof-checking** (not so important for certification)
- **Challenge of certification: reducing the Trusted Computing Base to the proof-checker**
  - HERMES's uses the COQ-engine where proof-checking is type checking
  - PPC places the VCG to the TCB
- Both establish a **checkable relation between the system S, the property φ and the proof ∇** through
  - the Verification Condition Generator in PCC : $VCG(S, \varphi)$
  - extraction of the protocol and the property from HERMES's proof (No need for VCG)

Certification versus Proof-Carrying Code

- **Challenge of PCC: reducing the size of proof and complexity of proof-checking** (not so important for certification)
- **Challenge of certification: reducing the Trusted Computing Base to the proof-checker**
  - HERMES's uses the COQ-engine where proof-checking is type checking
  - PPC places the VCG to the TCB
- Both establish a **checkable relation between the system S, the property φ and the proof ∇** through
  - the Verification Condition Generator in PCC : $VCG(S, \varphi)$
  - extraction of the protocol and the property from HERMES's proof (No need for VCG)

Certification versus Proof-Carrying Code

- **Challenge of PCC: reducing the size of proof and complexity of proof-checking** (not so important for certification)
- **Challenge of certification: reducing the Trusted Computing Base to the proof-checker**
  - HERMES's uses the COQ-engine where proof-checking is type checking
  - PPC places the VCG to the TCB
- Both establish a **checkable relation between the system S, the property φ and the proof ∇** through
  - the Verification Condition Generator in PCC : $VCG(S, \varphi)$
  - extraction of the protocol and the property from HERMES's proof (No need for VCG)

Certification versus Proof-Carrying Code

- **Challenge of PCC: reducing the size of proof and complexity of proof-checking** (not so important for certification)
- **Challenge of certification: reducing the Trusted Computing Base to the proof-checker**
  - HERMES's uses the COQ-engine where proof-checking is type checking
  - PPC places the VCG to the TCB
- Both establish a **checkable relation between the system S, the property φ and the proof ∇** through
  - the Verification Condition Generator in PCC : $VCG(S, \varphi)$
  - extraction of the protocol and the property from HERMES's proof (No need for VCG)

Certification versus Proof-Carrying Code

- **Challenge of PCC: reducing the size of proof and complexity of proof-checking** (not so important for certification)
- **Challenge of certification: reducing the Trusted Computing Base to the proof-checker**
  - HERMES's uses the COQ-engine where proof-checking is type checking
  - PPC places the VCG to the TCB
- Both establish a **checkable relation between the system S, the property φ and the proof ∇** through
  - the Verification Condition Generator in PCC : $VCG(S, \varphi)$
  - extraction of the protocol and the property from HERMES's proof (No need for VCG)

Certification versus Proof-Carrying Code

- **Challenge of PCC: reducing the size of proof and complexity of proof-checking** (not so important for certification)
- **Challenge of certification: reducing the Trusted Computing Base to the proof-checker**
  - HERMES's uses the COQ-engine where proof-checking is type checking
  - PPC places the VCG to the TCB
- Both establish a **checkable relation between the system S, the property φ and the proof ∇** through
  - the Verification Condition Generator in PCC : $VCG(S, \varphi)$
  - extraction of the protocol and the property from HERMES's proof (No need for VCG)

Related work

1. BLAST [T.Henziger *et al.*, 2002]
   - concrete proofs invariants properties of C program,
   - for predicate abstraction
   - uses a VCG
2. [K.Namjoshi, 2003]
   generation of concrete proof of mu-calculus properties
   - in a specific deduction system
   - level of automatization?
3. [Many people in MC community]
   generation of checkable witness of the computations of a model-checker
   - requires to trust the checker
   - is the checker simpler than the model-checker?

Related work

1. BLAST [T.Henziger *et al.*, 2002]
   - concrete proofs invariants properties of C program,
   - for predicate abstraction
   - uses a VCG
2. [K.Namjoshi, 2003]
   generation of concrete proof of mu-calculus properties
   - in a specific deduction system
   - level of automatization?
3. [Many people in MC community]
   generation of checkable witness of the computations of a model-checker
   - requires to trust the checker
   - is the checker simpler than the model-checker?

Related work

1. BLAST [T.Henziger *et al.*, 2002]
   - concrete proofs invariants properties of C program,
   - for predicate abstraction
   - uses a VCG
2. [K.Namjoshi, 2003]
   generation of concrete proof of mu-calculus properties
   - in a specific deduction system
   - level of automatization?
3. [Many people in MC community]
   generation of checkable witness of the computations of a model-checker
   - requires to trust the checker
   - is the checker simpler than the model-checker?

Related work

1. BLAST [T.Henziger *et al.*, 2002]
   - concrete proofs invariants properties of C program,
   - for predicate abstraction
   - uses a VCG
2. [K.Namjoshi, 2003]
   generation of concrete proof of mu-calculus properties
   - in a specific deduction system
   - level of automatization?
3. [Many people in MC community]
   generation of checkable witness of the computations of a model-checker
   - requires to trust the checker
   - is the checker simpler than the model-checker?

Related work

1. BLAST [T.Henziger *et al.*, 2002]
   - concrete proofs invariants properties of C program,
   - for predicate abstraction
   - uses a VCG
2. [K.Namjoshi, 2003]
   generation of concrete proof of mu-calculus properties
   - in a specific deduction system
   - level of automatization?
3. [Many people in MC community]
   generation of checkable witness of the computations of a model-checker
   - requires to trust the checker
   - is the checker simpler than the model-checker?

Related work

1. BLAST [T.Henziger *et al.*, 2002]
   - concrete proofs invariants properties of C program,
   - for predicate abstraction
   - uses a VCG
2. [K.Namjoshi, 2003]
   generation of concrete proof of mu-calculus properties
   - in a specific deduction system
   - level of automatization?
3. [Many people in MC community]
   generation of checkable witness of the computations of a model-checker
   - requires to trust the checker
   - is the checker simpler than the model-checker?

Related work

1. BLAST [T.Henziger *et al.*, 2002]
   - concrete proofs invariants properties of C program,
   - for predicate abstraction
   - uses a VCG
2. [K.Namjoshi, 2003]
   generation of concrete proof of mu-calculus properties
   - in a specific deduction system
   - level of automatization?
3. [Many people in MC community]
   generation of checkable witness of the computations of a model-checker
   - requires to trust the checker
   - is the checker simpler than the model-checker?

Related work

1. BLAST [T.Henziger *et al.*, 2002]
   - concrete proofs invariants properties of C program,
   - for predicate abstraction
   - uses a VCG
2. [K.Namjoshi, 2003]
   generation of concrete proof of mu-calculus properties
   - in a specific deduction system
   - level of automatization?
3. [Many people in MC community]
   generation of checkable witness of the computations of a model-checker
   - requires to trust the checker
   - is the checker simpler than the model-checker?

Related work

1. BLAST [T.Henziger *et al.*, 2002]
   - concrete proofs invariants properties of C program,
   - for predicate abstraction
   - uses a VCG
2. [K.Namjoshi, 2003]
   generation of concrete proof of mu-calculus properties
   - in a specific deduction system
   - level of automatization?
3. [Many people in MC community]
   generation of checkable witness of the computations of a model-checker
   - requires to trust the checker
   - is the checker simpler than the model-checker?

Related work

1. BLAST [T.Henziger *et al.*, 2002]
   - concrete proofs invariants properties of C program,
   - for predicate abstraction
   - uses a VCG
2. [K.Namjoshi, 2003]
   generation of concrete proof of mu-calculus properties
   - in a specific deduction system
   - level of automatization?
3. [Many people in MC community]
   generation of checkable witness of the computations of a model-checker
   - requires to trust the checker
   - is the checker simpler than the model-checker?

HERMES is available on-line

`http://www.verimag.imag.fr/` → DCS → software → HERMES