Convincing proofs for program certification

Michaël Périn

joint work with Manuel Garnacho

Verimag - Univerisité de Grenoble

SafeCert'08

Formal certification

Context

- Working group on certification of EADS, RATP and the french certification authority for security
- We consider the highest level of formal certification: formal evidence of correctness of critical applications

Motivation

- Verification tools are used in the design of critical applications
- The verdict of a verification tool is not recognised by evaluators unless the VT has been certified

Long term goal

Using VTs during development and getting a high certification at low cost

Certifying the result of a tools (Namjoshi'01 ... Leroy'06)

- Too difficult to certify the VT itself but ...

$$\textit{System} - \boxed{\text{VT}} \rightarrow \textit{Property}$$

$$\downarrow$$

checkable certificate of the statement $\textit{System} \models \textit{Property}$

$$\downarrow$$

$$\boxed{\text{Verifier}} \rightarrow \textit{yes}/\textit{no}$$

Certifying the verifier is simpler than certifying the VT

The approach of Foundational Proof-Carrying Code

- Certificate = formal checkable proofs in mathematical logic
- Verifier = proof-checker that must be certified

Certifying the result of a tools (Namjoshi'01 ... Leroy'06)

- Too difficult to certify the VT itself but ...

$$\textit{System} - \boxed{\text{VT}} \rightarrow \textit{Property}$$

$$\downarrow$$

checkable certificate of the statement $\quad \textit{System} \models \textit{Property}$

$$\downarrow$$

$$\boxed{\text{Verifier}} \rightarrow \textit{yes}/\textit{no}$$

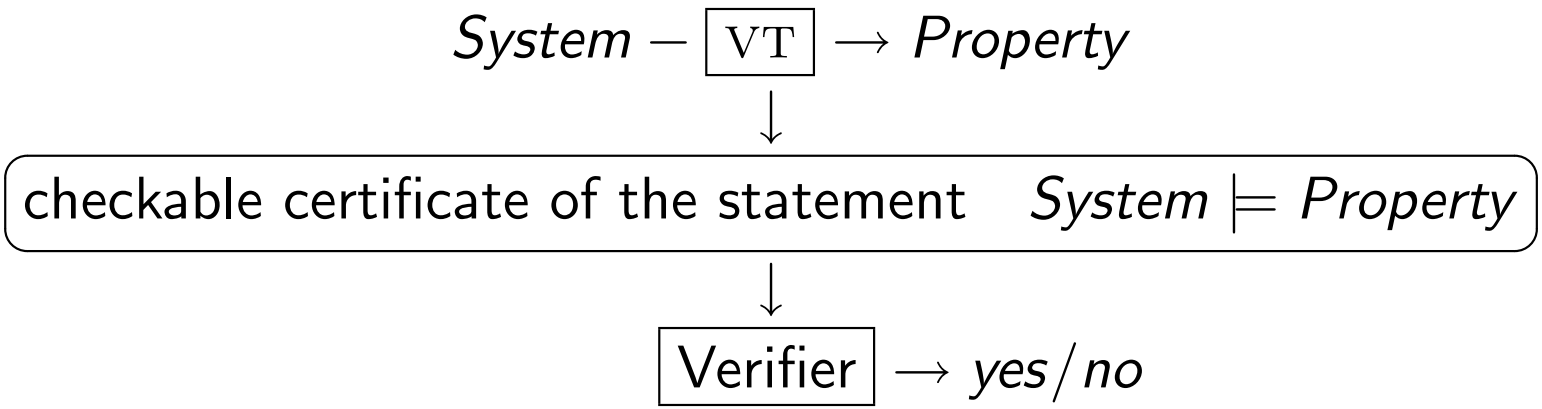
Certifying the verifier is simpler than certifying the VT

The approach of Foundational Proof-Carrying Code

- Certificate = formal checkable proofs in mathematical logic
- Verifier = proof-checker that must be certified

Certifying the result of a tools (Namjoshi'01 ... Leroy'06)

- Too difficult to certify the VT itself but ...



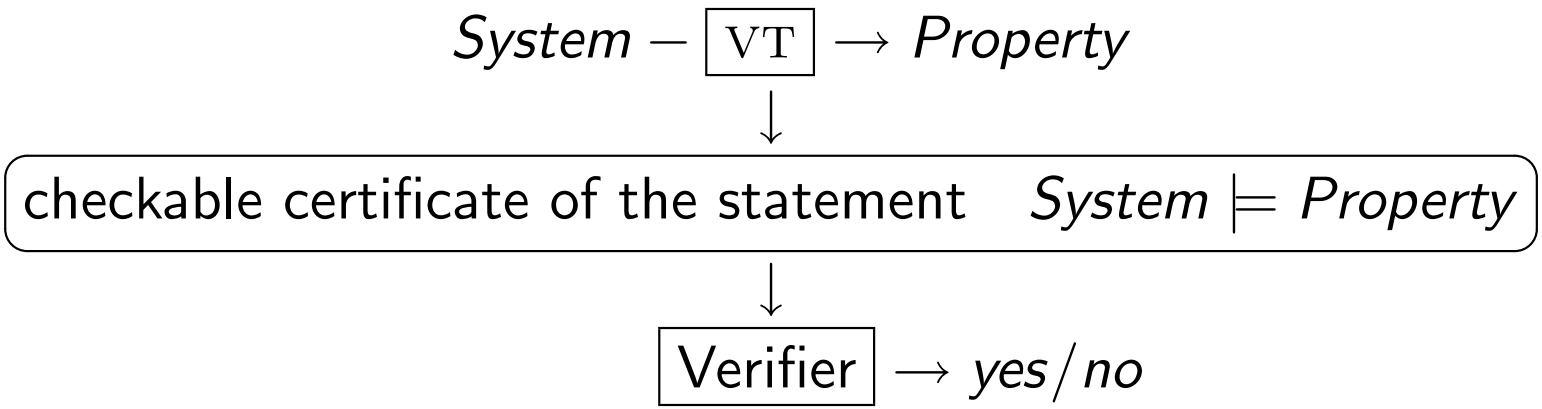
Certifying the verifier is simpler than certifying the VT

The approach of Foundational Proof-Carrying Code

- Certificate = formal checkable proofs in mathematical logic
- Verifier = proof-checker that must be certified

Certifying the result of a tools (Namjoshi'01 ... Leroy'06)

- Too difficult to certify the VT itself but ...



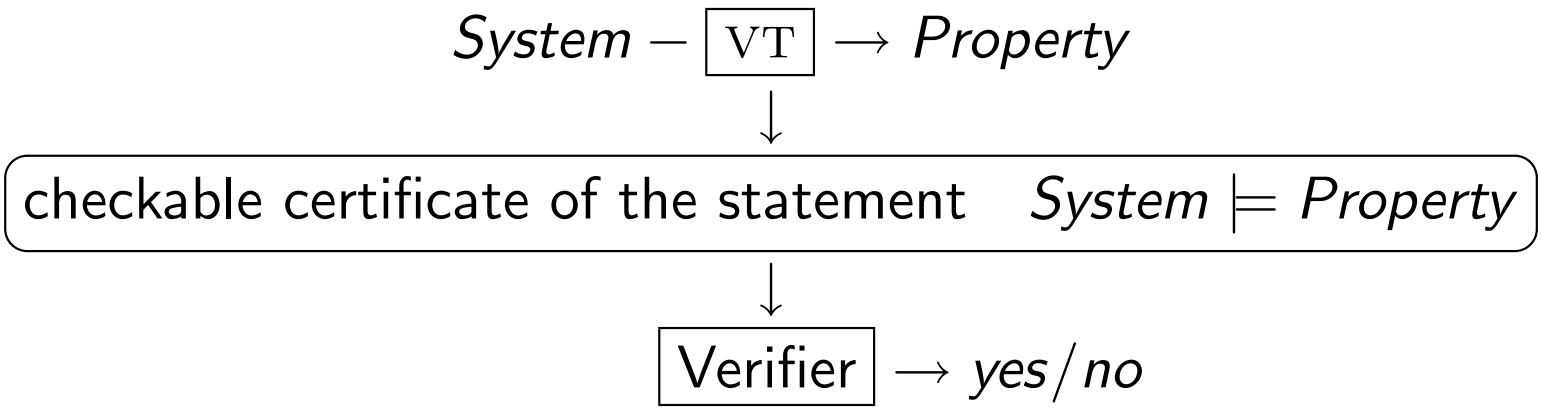
Certifying the verifier is simpler than certifying the VT

The approach of Foundational Proof-Carrying Code

- Certificate = formal checkable proofs in mathematical logic
- Verifier = proof-checker that must be certified

Certifying the result of a tools (Namjoshi'01 ... Leroy'06)

- Too difficult to certify the VT itself but ...



Certifying the verifier is simpler than certifying the VT

The approach of Foundational Proof-Carrying Code

- Certificate = formal checkable proofs in mathematical logic
- Verifier = proof-checker that must be certified

Formal checkable proofs

Given a set of derivation rules a proof is the application of these rules that ends with the statement to prove.

A derivation

$$\dfrac{\dfrac{\Phi_1 \quad \Phi_2}{\Phi_4}\, r_1 \qquad \dfrac{\Phi_3}{\Phi_5}\, r_2}{\Phi_4 \wedge \Phi_5}\, \wedge$$

corresponds to a term

$$\textsc{apply}\,(\wedge,\, [\,\textsc{apply}(r_1, [\Phi_1, \Phi_2], \Phi_4)\;;\;\textsc{apply}(r_2, [\Phi_3], \Phi_5)\,],\; \Phi_4 \wedge \Phi_5)$$

Verifying the proof-term:

- check that each step is a correct instantiation of a derivation rule
- needs only recursive traversal of the proof-term and matching

Formal checkable proofs

Given a set of derivation rules a proof is the application of these rules that ends with the statement to prove.

A derivation

$$\dfrac{\dfrac{\dfrac{\Phi_1 \quad \Phi_2}{\Phi_4}\, r_1 \qquad \dfrac{\Phi_3}{\Phi_5}\, r_2}{\Phi_4 \ \wedge \ \Phi_5}\, \wedge}{}$$

corresponds to a term

$$\textsc{apply}\,(\wedge,\, [\ \textsc{apply}(r_1, [\Phi_1, \Phi_2], \Phi_4)\ ;\ \textsc{apply}(r_2, [\Phi_3], \Phi_5)\ ],\ \Phi_4 \ \wedge \ \Phi_5)$$

Verifying the proof-term:

- check that each step is a correct instantiation of a derivation rule
- needs only recursive traversal of the proof-term and matching

Formal checkable proofs

Given a set of derivation rules a proof is the application of these rules that ends with the statement to prove.

A derivation

$$\dfrac{\dfrac{\dfrac{\Phi_1 \quad \Phi_2}{\Phi_4}\, r_1 \qquad \dfrac{\Phi_3}{\Phi_5}\, r_2}{\Phi_4 \ \wedge \ \Phi_5}\, \wedge}{}$$

corresponds to a term

$$\textsc{apply}\,(\wedge,\ [\ \textsc{apply}(r_1, [\Phi_1, \Phi_2], \Phi_4)\ ;\ \textsc{apply}(r_2, [\Phi_3], \Phi_5)\ ],\ \Phi_4 \ \wedge \ \Phi_5)$$

Verifying the proof-term:

- check that each step is a correct instantiation of a derivation rule
- needs only recursive traversal of the proof-term and matching

Building the proof-checker of a proof ∇

... requires recursive function and matching

Implementation in PROLOG by direct translation of each rule

The $(\wedge_i)$ derivation rule:

$$\frac{\Phi_1 \qquad \Phi_2}{\Phi_1 \wedge \Phi_2}\ \wedge_i$$

and the corresponding PROLOG clause of the checker:

```
check(∇, Φ1 ∧ Φ2):-
    ∇ = apply(∧i, [∇1, ∇2], Φ1 ∧ Φ2),
    check(∇1, Φ1),
    check(∇2, Φ2).
```

6 criteria for a convincing proof

requirements of skeptical evaluators for accepting a formal proof as a certificate

(i) The verifier must have been certified by the evaluators

- proof-carrying code : verifier $>$ 23 000 lines of C
- no proof-checker has already been certified
- a trusted proof-checker is built in collaboration with the evaluators during validation of the rules

(ii) The proof must addressed the actual program to certified

- VTs produce apply many transformations before verification
- the proof is done on the abstract syntax tree of the program

(iii) Evaluators must agree on a logical framework in which the correctness property can be stated

- specific logics (eg. temporal logic) are difficult to grasp
- We rely on the standard background of computer scientists in mathematics: FOL and definition of predicates specific to the problem

6 criteria for a convincing proof

requirements of skeptical evaluators for accepting a formal proof as a certificate

(i) The verifier must have been certified by the evaluators

- proof-carrying code : verifier $>$ 23 000 lines of C
- no proof-checker has already been certified
- a trusted proof-checker is built in collaboration with the evaluators during validation of the rules

(ii) The proof must addressed the actual program to certified

- VTs produce apply many transformations before verification
- the proof is done on the abstract syntax tree of the program

(iii) Evaluators must agree on a logical framework in which the correctness property can be stated

- specific logics (eg. temporal logic) are difficult to grasp
- We rely on the standard background of computer scientists in mathematics: FOL and definition of predicates specific to the problem

6 criteria for a convincing proof

requirements of skeptical evaluators for accepting a formal proof as a certificate

(i) The verifier must have been certified by the evaluators

- proof-carrying code : verifier $>$ 23 000 lines of C
- no proof-checker has already been certified
- a trusted proof-checker is built in collaboration with the evaluators during validation of the rules

(ii) The proof must addressed the actual program to certified

- VTs produce apply many transformations before verification
- the proof is done on the abstract syntax tree of the program

(iii) Evaluators must agree on a logical framework in which the correctness property can be stated

- specific logics (eg. temporal logic) are difficult to grasp
- We rely on the standard background of computer scientists in mathematics: FOL and definition of predicates specific to the problem

6 criteria for a convincing proof

requirements of skeptical evaluators for accepting a formal proof as a certificate

(i) The verifier must have been certified by the evaluators

- proof-carrying code : verifier $>$ 23 000 lines of C
- no proof-checker has already been certified
- a trusted proof-checker is built in collaboration with the evaluators during validation of the rules

(ii) The proof must addressed the actual program to certified

- VTs produce apply many transformations before verification
- the proof is done on the abstract syntax tree of the program

(iii) Evaluators must agree on a logical framework in which the correctness property can be stated

- specific logics (eg. temporal logic) are difficult to grasp
- We rely on the standard background of computer scientists in mathematics: FOL and definition of predicates specific to the problem

6 criteria for a convincing proof

requirements of skeptical evaluators for accepting a formal proof as a certificate

(i) The verifier must have been certified by the evaluators

- proof-carrying code : verifier $>$ 23 000 lines of C
- no proof-checker has already been certified
- a trusted proof-checker is built in collaboration with the evaluators during validation of the rules

(ii) The proof must addressed the actual program to certified

- VTs produce apply many transformations before verification
- the proof is done on the abstract syntax tree of the program

(iii) Evaluators must agree on a logical framework in which the correctness property can be stated

- specific logics (eg. temporal logic) are difficult to grasp
- We rely on the standard background of computer scientists in mathematics: FOL and definition of predicates specific to the problem

6 criteria for a convincing proof

requirements of skeptical evaluators for accepting a formal proof as a certificate

(i) The verifier must have been certified by the evaluators

- proof-carrying code : verifier $>$ 23 000 lines of C
- no proof-checker has already been certified
- a trusted proof-checker is built in collaboration with the evaluators during validation of the rules

(ii) The proof must addressed the actual program to certified

- VTs produce apply many transformations before verification
- the proof is done on the abstract syntax tree of the program

(iii) Evaluators must agree on a logical framework in which the correctness property can be stated

- specific logics (eg. temporal logic) are difficult to grasp
- We rely on the standard background of computer scientists in mathematics: FOL and definition of predicates specific to the problem

(iv) Evaluators must validate all logical and semantic rules

- in VTs, the semantics is hard coded, not available
- all rules are examined during the validation of proof-checker

(v) Few and obvious derivation rules

- general purpose minimal theorem provers: large and over detailed description
- semantics of the instructions used in the program and a specific logic

(vi) The proof must address the exact verification problem

- VTs carry the problem into their framework
- the actual statement $program \models property$ appears explicitly at the root of the proof

(iv) Evaluators must validate all logical and semantic rules

- in VTs, the semantics is hard coded, not available
- all rules are examined during the validation of proof-checker

(v) Few and obvious derivation rules

- general purpose minimal theorem provers: large and over detailed description
- semantics of the instructions used in the program and a specific logic

(vi) The proof must address the exact verification problem

- VTs carry the problem into their framework
- the actual statement $program \models property$ appears explicitly at the root of the proof

(iv) Evaluators must validate all logical and semantic rules

- in VTs, the semantics is hard coded, not available
- all rules are examined during the validation of proof-checker

(v) Few and obvious derivation rules

- general purpose minimal theorem provers: large and over detailed description
- semantics of the instructions used in the program and a specific logic

(vi) The proof must address the exact verification problem

- VTs carry the problem into their framework
- the actual statement *program* $\models$ *property* appears explicitly at the root of the proof

(iv) Evaluators must validate all logical and semantic rules

- in VTs, the semantics is hard coded, not available
- all rules are examined during the validation of proof-checker

(v) Few and obvious derivation rules

- general purpose minimal theorem provers: large and over detailed description
- semantics of the instructions used in the program and a specific logic

(vi) The proof must address the exact verification problem

- VTs carry the problem into their framework
- the actual statement *program* $\models$ *property* appears explicitly at the root of the proof

(iv) Evaluators must validate all logical and semantic rules

- in VTs, the semantics is hard coded, not available
- all rules are examined during the validation of proof-checker

(v) Few and obvious derivation rules

- general purpose minimal theorem provers: large and over detailed description
- semantics of the instructions used in the program and a specific logic

(vi) The proof must address the exact verification problem

- VTs carry the problem into their framework
- the actual statement $program \models property$ appears explicitly at the root of the proof

(iv) Evaluators must validate all logical and semantic rules

- in VTs, the semantics is hard coded, not available
- all rules are examined during the validation of proof-checker

(v) Few and obvious derivation rules

- general purpose minimal theorem provers: large and over detailed description
- semantics of the instructions used in the program and a specific logic

(vi) The proof must address the exact verification problem

- VTs carry the problem into their framework
- the actual statement $program \models property$ appears explicitly at the root of the proof

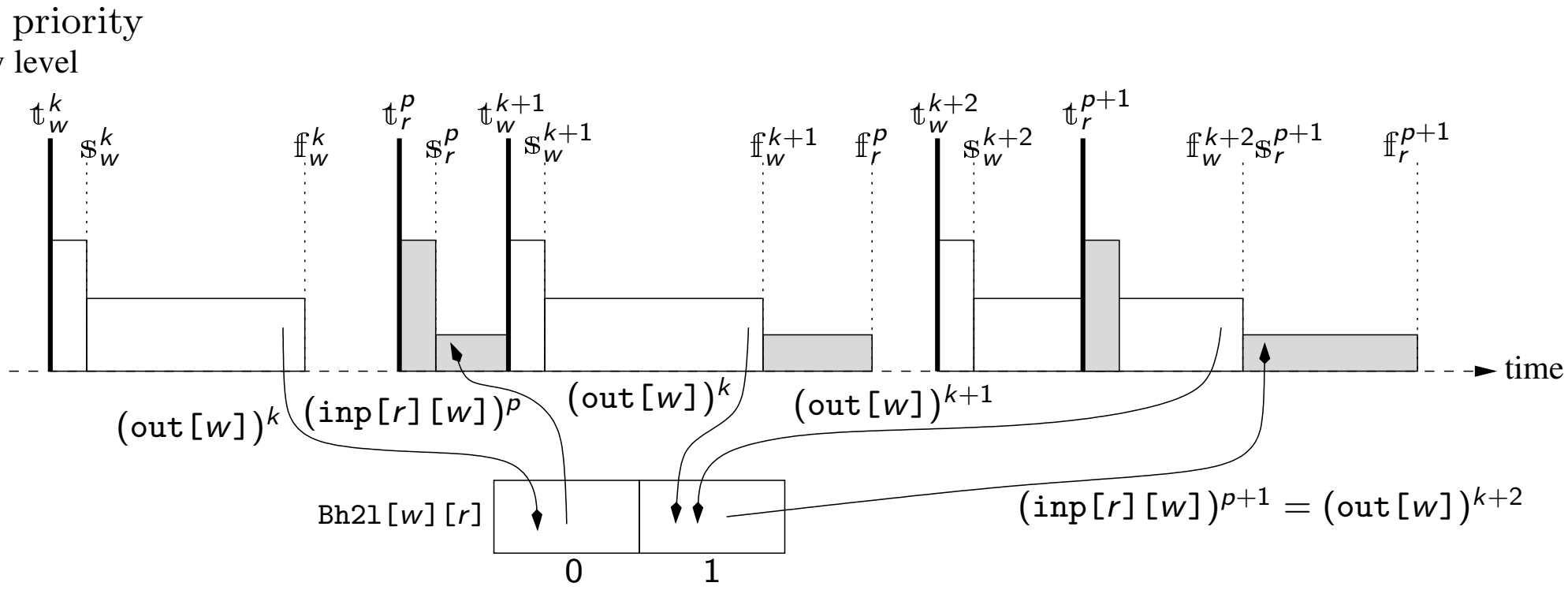
Application to a communication protocol

- developed for avionic systems
- for implementing multi-task data-flow real-time system
- onto an event-based operating system with preemption and priority
- written in C
- presented at EMSOFT'2005

  S.Tripakis, C.Sofronis, N.Scaife, P.Caspi

  "semantics-preserving and memory-efficient implementation of inter-task communication under static priority or EDF schedulers"

Buffering protocol using arrays



Correctness property: always get the latest output

$$\mathbb{t}_w^k .. \mathbb{t}_r^p .. \mathbb{t}_w^{k+1} \subseteq \sigma \Rightarrow (\texttt{inp}[r][w])^p = (\texttt{out}[w])^k$$

Proof based on equalities on symbolic value

The semantics of the C instructions

$$t = x \ ; \ x = y \ ; \ y = t$$

is the conjunction of equalities

$$t^1 = x^0 \wedge x^1 = y^0 \wedge y^1 = t^1$$

which implies

$$x^1 = y^0 \wedge y^1 = x^0$$

Semantics rules for the C instruction (excerpts)

$$\frac{\langle \mathcal{V}, v{=}e, \mathcal{V}' \rangle}{\text{EVAL}(\mathcal{V}', v) = \text{EVAL}(\mathcal{V}, e)}\ asg_1$$

$$\frac{\langle \mathcal{V}, v{=}e, \mathcal{V}' \rangle}{\text{AC}(\mathcal{V}', v) = \text{AC}(\mathcal{V}, v) + 1}\ asg_3$$

$$\frac{\langle \mathcal{V},\ \texttt{for(i=0; i<}n\texttt{; i=i+1)\{}P\texttt{(i)\}}\ , \mathcal{V}' \rangle \quad n > 0}{\langle \mathcal{V},\ \texttt{for(i=0; i<}n\texttt{-1; i=i+1)\{}P\texttt{(i)\}}\ ;\ P(n)\ , \mathcal{V}' \rangle}\ for_2$$

Validation of the derivation rules

- natural deduction for FOL + equality + induction: 22 standard rules
- definitions of predicates on traces: 9 rules
- C semantics: 11 rules
- definition of the system semantics and priority mechanisms: 8 rules
- relating events and task triggering: 8 rules
- mathematical property of $=, <, \leq, +$ on naturals numbers: 12 rules

The proof-checking function is the direct translation of those 70 rules into 70 PROLOG clauses.

Proof sketch

The proof consists in

- inductions on sequence of events and on the number of occurrence of the triggering event $\mathbb{t}_w^k$
- followed by a case study
- reduced using non-interference lemma to 6 possible interleavings of events

$$\mathbb{t}_w^k .. \mathbb{s}_w^k .. \mathbb{f}_w^k \quad .. \quad \mathbb{t}_w^{k+1} .. \mathbb{s}_w^{k+1} .. \mathbb{f}_w^{k+1} \quad \textit{with} \quad \mathbb{t}_r^p .. \mathbb{s}_r^p .. \mathbb{f}_r^p$$

- These proofs done with the help of an instrumented symbolic interpreter.

Future work

Proof generation by instrumentation of a verification tool must solve the gap problem:

- VTs reason on an abstraction of the system
- whereas proofs deals with the actual system
- the proof does not follow the VT computations

Reduction of the size of proof-terms is needed

- using lemmata
- compact proof representation using reversible proof transformations

Position

Goals

1. confronting the evaluators with evidence they cannot reject
2. reducing the Trusted Computing Base
3. relying on standard mathematical knowledge of Bachelors in Computer science ... because the proof activity is a social discipline: *do we agree on the proof steps and axioms ?*

Lobbying

- Evaluators don't (shouldn't) care how you produced the proof
- Evaluators don't have to read the proofs (proofs are huge!)

if

1. they validate the derivations rules of the proof system
2. they validate the proof-checker