Automated verification of termination certificates

F. Blanqui (INRIA)

LIAMA, Tsinghua University, Beijing, China

http://www.loria.fr/~blanqui/

FORMES project

http://liama.ia.ac.cn/wiki/projects:formes:home

Talk on 03/12/08 at East China Normal University, Shanghai, China

Outline

Why certifying (critical) software?

- software have bugs, sometimes difficult to detect
- bugs cost a lot (in money or lives)
- tests are necessary but cannot cover all cases
- static analysis is powerful but cannot check all properties
- formal certification is required by laws or contracts

How to certify a software?

approaches:

- make a mathematical proof on paper
  - long, fastidious, error-prone

  e.g. "Proof of a program: Find", by C.A.R. Hoare, 1971
- use a proof assistant
  - long, fastidious
- use a dedicated tool on top of a proof assistant

  e.g. "Formal Proof of a Program: Find", by J.-C. Filliâtre, 2006

How to certify a software?

approaches:

- make a mathematical proof on paper
  - long, fastidious, error-prone

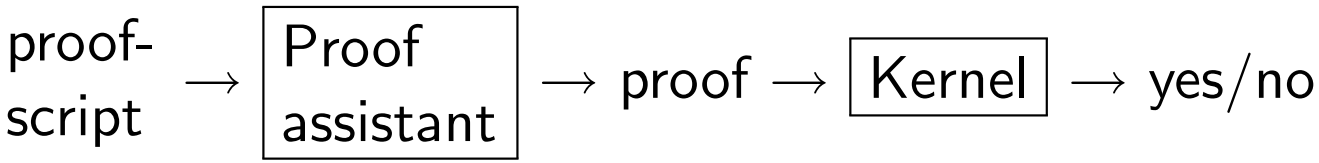
  e.g. "Proof of a program: Find", by C.A.R. Hoare, 1971
- use a proof assistant
  - long, fastidious
- use a dedicated tool on top of a proof assistant

  e.g. "Formal Proof of a Program: Find", by J.-C. Filliâtre, 2006

problem: a proof assistant may have bugs too!...

How to certify the certifier?

- logicians have shown that proofs can be carried out by using a small number of deduction rules
- proof assistants like Coq, Isabelle, etc. are based on this idea:
  - a kernel checks the correctness of proofs
    it is small enough to be checked by hand
  - a proof development environment provides tools for building proofs
    it does not matter too much that these tools have bugs:
    their results are checked by the kernel



Proof assistants

generally provide:

- a programming language for defining functions
- a specification language for defining predicates
- a script language for building proofs
- libraries with usual functions, predicates and tactics
- a module system
- ...

What to certify?

approaches:

- prove once and for all that the software has no bug
  - hard and time consuming task
  - must be redone every time the code is changed
- make the software output some certificate that can be used to check the correctness of its computations
  - does not depend on the way the tool is implemented

Outline

Why certifying termination?

- termination is an important property
- it is undecidable
- more and more complex techniques are developped
- every year in the termination competition, some tools are disqualified because someone found some wrong answer

How to certify termination?

CoLoR project: build a dedicated tool on top of a proof assistant

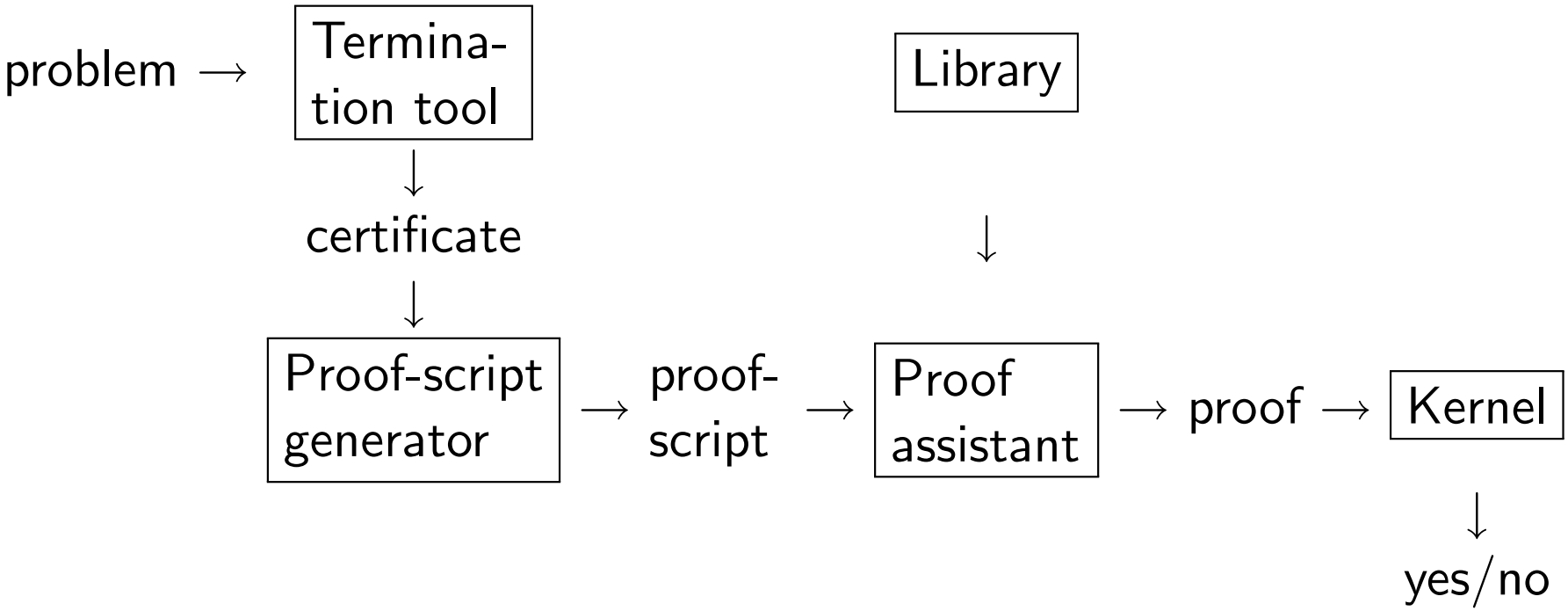
constraints:

- developpers of termination tools do not know proof assistants
- it is hard and time consuming to develop, and thus generate, complex proofs

Our approach

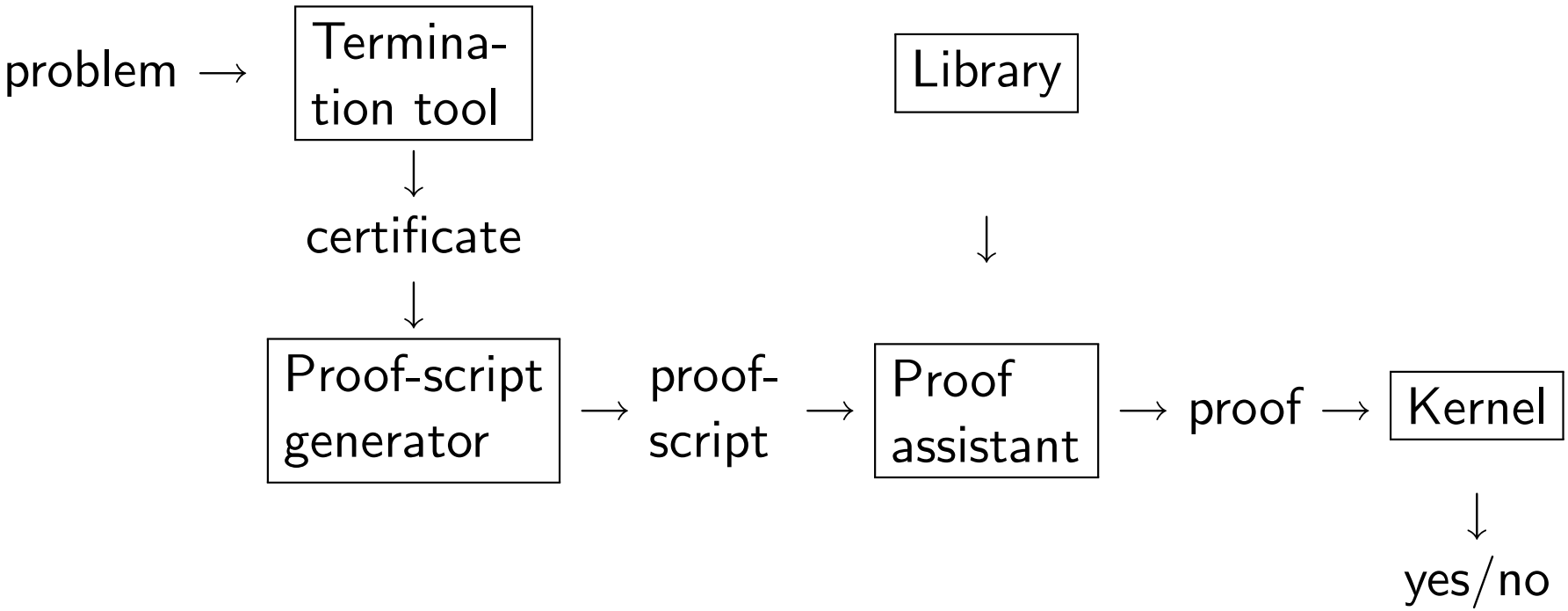
- provide a simple language for termination certificates
- develop a library in a proof assistant providing:
  - a formalization of programs and their operational semantics
  - a formalization of theorems the certificates rely on
  - tactics for checking the conditions of theorems
- provide a tool generating a proof-script from a certificate

Our approach



CoLoR

Proof assistant = Coq
Library = CoLoR
Proof-script generator = Rainbow



What programs can CoLoR currently handle?

string and term rewrite systems

- rewriting is a general and powerful programming paradigm
- logic and functional programs can be encoded with rewriting
- data are represented as strings or terms (trees)
- programs are transformation rules applied as long as possible
  e.g. $x + (-x) \to 0$
  $\mathit{insert}\ x\ (y :: \mathit{ys}) \to \mathit{if}\ x \leq y\ \mathit{then}\ x :: y :: \mathit{ys}\ \mathit{else}\ y :: \mathit{insert}\ x\ \mathit{ys}$

Outline

Example of criterion: polynomial interpretations

- for each function symbol f of arity n, we assume given an integer polynomial P_f with n variables
- a term t can then be interpreted by a polynomial P_t by composing the interpretations of each symbol occuring in t
- then a program defined by a set $\mathcal{R}$ of rules terminates if:
  - each P_f is monotone in all its variables
  - for every rule $l \to r \in \mathcal{R}$, $P_l > P_r$

Certification of polynomial interpretations

- What is provided by the CoLoR library?
  - the formalization of the termination criterion itself (PI)
  - a tactic to try to prove monotony (mon)
  - a tactic to try to prove $P_l > P_r$ (decr)
- What is the certificate required by Rainbow?
  - the polynomials P_f
- What is done by Rainbow?
  - translation in Coq of the set of rules $\mathcal{R}$ and the polynomials P_f
  - generation of the proof-script: `apply TC; [mon|decr]`
- What is done by Coq?
  - run the proof-script to generate a proof and check its correctness

XML Schema for termination certificates

```
...

<complexType name="ReductionOrdering">
  <choice>
    <element name="poly_int" type="prf:PolynomialInterpretation"
    ...
  </choice>
</complexType>

<complexType name="Proof">
  <choice>
    <element name="manna_ness" type="prf:ReductionOrdering"/>
    ...
  </choice>
</complexType>
```

Example of termination problem I

```
<?xml version="1.0" encoding="ISO-8859-1"?>

<problem
  xmlns="urn:rainbow.problem.format"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="urn:rainbow.problem.format problem.xsd">

<algebra>
  <signature>
    <mapping><fun>plus</fun><arity>2</arity></mapping>
    <mapping><fun>succ</fun><arity>1</arity></mapping>
    <mapping><fun>zero</fun><arity>0</arity></mapping>
  </signature>
</algebra>

<theory/>
```

Example of termination problem II

```
<strategy><none/></strategy>

<rules>
  <rule> <!-- plus zero v0 = v0 -->
    <lhs>
      <app>
        <fun>plus</fun>
        <arg><app><fun>zero</fun></app></arg>
        <arg><var>0</var></arg>
      </app>
    </lhs>
    <rhs>
      <var>0</var>
    </rhs>
  </rule>
  ...
```

An example of termination certificate

```
<?xml version="1.0" encoding="ISO-8859-1"?>

<proof
  xmlns="urn:rainbow.proof.format"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="urn:rainbow.proof.format proof.xsd">

<manna_ness>
  <poly_int>
    <mapping>
      <fun>plus</fun>
      <polynomial> <!-- 2.x_1 + 1.x_2 + 2 -->
        <monomial>
          <coef>2</coef>
          <var>1</var>
          <var>0</var>
```

Coq file generated by Rainbow I

```
Inductive symbol : Set :=
  | plus : symbol
  | succ : symbol
  | zero : symbol.

Lemma eq_symbol_dec : forall f g : symbol, {f=g}+{~f=g}.

Proof. decide equality. Qed.

Definition arity (s : symbol) :=
  match s with
  | plus => 2
  | succ => 1
  | zero => 0
end.
```

Coq file generated by Rainbow II

```
Require Import ASignature.

Definition sig := mkSignature arity eq_symbol_dec.

Definition S_plus x2 x1 := (@Fun sig plus (Vcons x2 (Vcons x1 Vn
Definition S_succ x1 := (@Fun sig succ (Vcons x1 Vnil)).
Definition S_zero := (@Fun sig zero Vnil).

Require Import ATrs.

Definition rules :=
   mkRule (S_plus S_zero (Var 0))
          (Var 0)
:: mkRule (S_plus (S_succ (Var 1)) (Var 0))
          (S_succ (S_plus (Var 1) (Var 0)))
:: nil.
```

Coq file generated by Rainbow III

```
Require Import Polynom.

Definition poly (f : symbol) : poly (arity f) :=
  match f as f return poly (arity f) with
    | plus => (2%Z, (Vcons 1 (Vcons 0 Vnil)))
              :: (1%Z, (Vcons 0 (Vcons 1 Vnil)))
              :: (2%Z, (Vcons 0 (Vcons 0 Vnil)))
              :: nil
    | succ => (1%Z, (Vcons 1 Vnil))
              :: (1%Z, (Vcons 0 Vnil))
              :: nil
    | zero => (1%Z, Vnil)
              :: nil
end.
```

Coq file generated by Rainbow IV

```
Require Import MonotonePolynom.

Lemma monotony : forall f, pmonotone (poly f).

Proof.
pmonotone.
Qed.

Require Import APolyInt.

Definition PI := mkPolyInterpretation sig poly monotony.

Lemma termination : wf (red rules).

Proof.
poly_int PI.
```

Outline

CoLoR

CoLoR library:

Number of Coq lines:	53687
Inductive predicates:	43
Recursive functions:	176
Definitions:	807
Tactics:	126
Theorems:	2803

Rainbow program:

Number of OCaml lines:	3162

Formalized termination techniques

- integer polynomial interpretations
- integer matrix interpretations
- first and higher order recursive path ordering
- argument filtering
- dependancy pair transformation
- dependency graph decomposition

Supported class of programs

rewrite systems on:

- strings
- first-order terms with symbols of fixed arity
- first-order terms with varyadic symbols
- simply-typed λ-terms

Libraries of general interest

- vectors
- integer polynomials
- matrices
- (ordered) semi-rings
- multisets
- first-order term algebra

Termination competition

every year is organized an international competition on termination

`http://termination-portal.org/`

Rainbow and CoLoR are currently used as a certification backend by 4 automated termination tools:

- AProVE, best TRS termination tool
- Matchbox, best SRS termination tool
- TPA
- TTT2

Termination competition results in 2007

Category	Rank	Tool	Score	N	%	NC-Score	%
TRS	1	tpa-color	354	975	36.3		
	2	cime-a3pat	317		32.5		
	3	ttt2-color	289		29.6	643	44.9

NC = Non Certified

Termination competition results in 2008

Category	Rank	Tool	Score	N	%	NC-Score	%
SRS	1	matchbox-color	466	732	63.7		
	2	aprove-cert	420		57.4		
	3	aprove-color	415		56.7	523	79.3
	4	aprove-a3pat	114		15.6		
TRS	1	aprove-cert	594	1391	42.7		
	2	aprove-color	580		41.7	1226	47.3
	3	aprove-a3pat	532		38.2		
	4	cime-a3pat	531		38.2		
	5	matchbox-color	458		32.9		

aprove-cert runs aprove-color and aprove-a3pat in parallel

Future work

- formalize more termination techniques
- handle more programming paradigms:
  logic programs, functional programs, etc.
- certify non-termination too

Thank you !

you can freely download Rainbow and CoLoR on

`http://color.loria.fr/`