CoLoR

a Coq Library on Rewriting and Termination

Frédéric Blanqui (INRIA & LORIA)
Solange Coupet-Grimal (Université de Provence)
William Delobel (Université de Provence)
Sébastien Hinderer (LORIA)
Adam Koprowski (Eindhoven University of Technology)

Why certifying (termination) tools ?

- techniques are more and more advanced
- tools have bugs
- problems are bigger and bigger
- certification is required in proof assistants

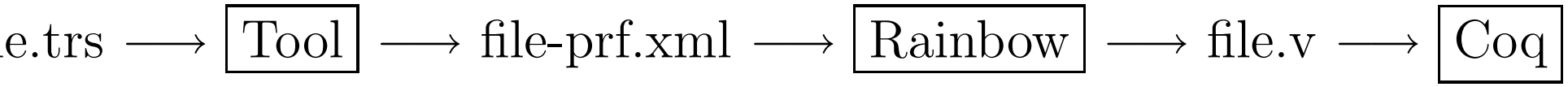
How to certify your (termination) tool ?

approaches:

- prove once and for all in a proof assistant that your tool has no bug
  - hard and time consuming task
  - must be redone every time the code is changed
- produce proofs and check them in some proof assistant
  + does not depend on the way the tool is implemented

owever, in both cases, you need to certify the theorems you rely on

Our approach



oq is a powerful and largely used proof assistant/checker

ermination proof grammar:

```
roof = MANNA_NESS red_ord | DP proof | ...
ed_ord = POLY_INT poly_int | RPO prec | LEX red_ord+ ...
oly_int = | symbol polynom poly_int
olynom = ...
```

XML Schema for termination proofs

```
.

omplexType name="ReductionOrdering">
<choice>
  <element name="rpo" type="prf:RPO"/>
  <element name="poly_int" type="prf:PolynomialInterpretation"/>
  <element name="lex" type="prf:LexicographicCombination"/>
</choice>
complexType>

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

lement name="proof" type="prf:Proof"/>
```

Example

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

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

lgebra>
<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>

heory/>

trategy><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>
...
/rules>

/problem>
```

A possible termination proof

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

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

anna_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>
      </monomial>
      <monomial>
```

```
        <coef>1</coef>
        <var>0</var>
        <var>1</var>
      </monomial>
      <monomial>
        <coef>2</coef>
        <var>0</var>
        <var>0</var>
      </monomial>
    </polynomial>
  </mapping>
  ...
</poly_int>
manna_ness>

proof>
```

Coq file generated by Rainbow

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

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

oof. decide equality. Qed.

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

```
equire Import ASignature.

efinition sig := mkSignature arity eq_symbol_dec.

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

equire Import ATrs.

efinition 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.
```

```
equire Import Polynom.

efinition 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
d.
```

```
equire Import MonotonePolynom.

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

roof.
onotone.
d.

equire Import APolyInt.

efinition PI := mkPolyInterpretation sig poly monotony.

emma termination : wf (red rules).

roof.
ly_int PI.
d.
```

CoLoR

braries on:

- data structures: lists, vectors, polynomials, multisets, ...
- term structures: varyadic, with arity, simply typed
- termination criteria: term conversion, arguments filtering, dependency pairs, polynomial interpretations, RPO, HORPO

tactics for automatically checking conditions of theorems

r simply typed λ-terms, multisets and HORPO:

. Adam's RTA paper

Term structures

- varyadic terms:

```
Inductive term : Set :=
  | Var : nat -> term
  | Fun : symbol -> list term -> term.
```

- terms with arity:

```
Inductive term : Set :=
  | Var : nat -> term
  | Fun : forall f : symbol, vector term (arity f) -> term.
```

```
emma manna_ness :
reduction_ordering succ -> compatible succ R -> wf (red R).

emma wf_vred_imp_wf_ared :
wf (vred (vrules_of_arules R)) -> wf (ared R).
```

Polynomials

```
* n = number of variables *)
efinition poly n := list (Z * vector nat n).
```

xample: $x^2 + 2y = 1 \cdot x^2y^0 + 2 \cdot x^0y^1$

```
1%Z, Vcons 2 (Vcons 0 Vnil))
  (2%Z, Vcons 0 (Vcons 1 Vnil))
  nil
```

```
emma polyInterpretationTermination : forall R,
 lforall (fun r => coef_pos (rulePoly r)) R -> wf (red R).
```

$ulePoly(l \to r) = [\![l]\!] - [\![r]\!] + 1$

Dependency pairs

```
finition chain t u := exists t' : term,
clos_refl_trans (int_red R) t t' /\ hd_red dp t' u.

mma wf_chain : wf chain -> wf (red R).

mma dp1 : forall (succ succ_eq : relation term)
(succ_eq_refl : reflexive succ_eq) (succ_eq_trans : transitive succ_eq)
(hsucc_eq : rewrite_ordering succ_eq) (succ_eq_incl : inclusion succ succ_eq)
(compat : inclusion (compose succ_eq succ) succ)
(hwf : wf succ) (hsucc : substitution_closed succ)
(comp_succ_eq : compatible succ_eq R) (comp_succ : compatible succ dp),
wf (red R).

mma dp1rc : forall (succ_trans : transitive succ)
(hsucc : weak_reduction_ordering succ (clos_refl succ))
(comp_succ_eq : compatible (clos_refl succ) R) (comp_succ : compatible succ dp),
wf (red R).
```

CoLoR size

3200 lines of Coq code (including blank lines and comments)

Util	**10800**	**Terms**	**17200**	**Termination**	**5200**
List	2900	Varyadic	400	Conversion	200
Vector	1500	WithArity	2800	Filter	300
Polynom	625	SimpleType	14000	PolyInt	250
Multiset	4400			DP	250
Others	1375			RPO	1200
				HORPO	3000

Thank you !

you can freely download Rainbow and CoLoR on

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