Automated verification of termination certificates

Frédéric Blanqui and Kim Quyen Ly

Outline

1. Software certification
2. Termination of rewriting and its certification
3. Our approach
4. Conclusion

Why/how to certify software?

- Software have bugs, sometimes difficult to detect
- Bugs are merely annoying and inconvenient
  but some can have extremely serious consequences

Why/how to certify software?

- Software have bugs, sometimes difficult to detect
- Bugs are merely annoying and inconvenient
  but some can have extremely serious consequences

Solutions:

- Tests are necessary but cannot cover all cases

Why/how to certify software?

- Software have bugs, sometimes difficult to detect
- Bugs are merely annoying and inconvenient
  but some can have extremely serious consequences

Solutions:

- Tests are necessary but cannot cover all cases
- Static analysis is powerful but cannot check all properties

Why/how to certify software?

- Software have bugs, sometimes difficult to detect
- Bugs are merely annoying and inconvenient
  but some can have extremely serious consequences

Solutions:

- Tests are necessary but cannot cover all cases
- Static analysis is powerful but cannot check all properties
- Formal certification (sometimes required by contract)

Why/how to certify software?

- Software have bugs, sometimes difficult to detect
- Bugs are merely annoying and inconvenient
  but some can have extremely serious consequences

Solutions:

- Tests are necessary but cannot cover all cases
- Static analysis is powerful but cannot check all properties
- Formal certification (sometimes required by contract)
- Use of certificates

Use of certificates

instead of proving that a source code is correct for every possible input

- Has to be redone each time the source code is changed
- Difficult when the tool uses complex heuristics

Use of certificates

instead of proving that a source code is correct for every possible input

- Has to be redone each time the source code is changed
- Difficult when the tool uses complex heuristics

check that its result is correct each time it is run
by providing a certificate and verifying it

- Does not depend on the source code
- Finding a solution to a problem is generally more difficult than checking that a solution is correct (P≠NP)

How to certify a software?

Proof on paper? long, difficult, error-prone
(e.g. “Proof of a program: Find”, Hoare, 1971)

How to certify a software?

Proof on paper? long, difficult, error-prone
(e.g. "Proof of a program: Find", Hoare, 1971)

$\Rightarrow$ Use a proof assistant!

Generally provides:

- A language for defining functions and properties
- Libraries of definitions and theorems
- Basic proof tactics and decision procedures
- A language for defining advanced proof tactics

Examples of works done in a proof assistant:

- 4-color theorem (2005), odd-order theorem (Gonthier et al, 2012)
- Formal verification of a realistic C compiler (Leroy 2009)
- Formal verification of an OS kernel (Klein et al, 2009)

The Coq proof assistant

- Main features:
  - Interactive theorem proving
  - Powerful specification language (including dependent types and inductive definitions)
  - Tactic language to build proofs
  - Type-checking algorithm to check proofs
- Coq has a large standard library including: Integers, Reals, Sets, etc.
- Extraction
  - Automatic generation of functional code from Coq proofs, in order to produce certified programs
  - Actually from Coq to ML or Haskell

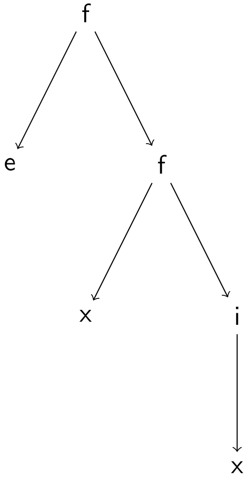
Outline

1. Software certification
2. Termination of rewriting and its certification
3. Our approach
4. Conclusion

First-order terms/trees

Symbols: $f \in \mathcal{F}$ n-ary
Variables: $x \in \mathcal{V}$
Terms: $x \mid f(t_1, \ldots, t_n)$

Example: $f(e, f(x, i(x)))$



Term rewriting

Introduced by Knuth in 1967:

Dershowitz-Jouannaud 1990

> “Rewrite systems are directed equations used to compute by repeatedly replacing subterms of a given formula with equal terms until the simplest form possible is obtained.”

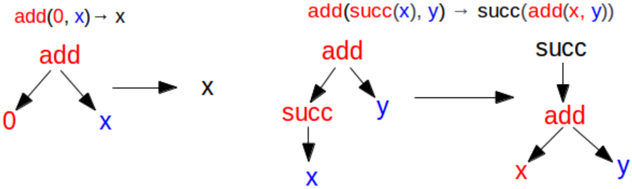
- Particular case: first-order functional programs
- It is Turing-complete (termination is undecidable even with one rule only)
- Programming languages based on rewriting: CafeOBJ, ELAN, Maude

Example of term rewriting system (TRS)

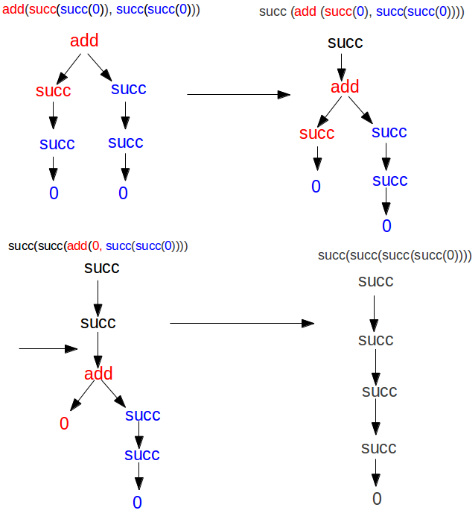
for solving the word problem in group theory:

$$\begin{array}{rcl}
(x+y)+z & \rightarrow & x+(y+z)\\
x+0 & \rightarrow & x\\
0+x & \rightarrow & x\\
x+(-x) & \rightarrow & 0\\
(-x)+x & \rightarrow & 0\\
(-x)+(x+y) & \rightarrow & y\\
x+((-x)+y) & \rightarrow & y\\
-0 & \rightarrow & 0
\end{array}$$

Example of rewriting sequence



Example of term rewriting system



How to prove termination of TRSs?

Many techniques and tools have been developed over the years (AProVE, TTT2, . . .)

Example: polynomial interpretations (Lankford, 1975)

- Interpret each function symbol f of arity n by a polynomial $\mathcal{P}_f$ with n variables on some well-founded domain (e.g. non-negative integers)
- Then, by composition, any term with n variables can be interpreted by a polynomial with n variables

Theorem

A program defined by a set $\mathcal{R}$ of rules terminates if:

- Each $\mathcal{P}_f$ is monotone in each variable
- For every rule $l \to r$, we have $\mathcal{P}_l > \mathcal{P}_r$

Example of polynomial interpretation on $\mathbb{N}$

$$\begin{array}{rcl} \mathrm{add(zero,x)} & \to & \mathrm{x} \\ \mathrm{add(succ(x),y)} & \to & \mathrm{succ(add(x,y))} \end{array}$$

polynomial interpretation:

$$\begin{array}{rl} \mathcal{P}_{add}(X,Y) & = 2X + Y \\ \mathcal{P}_{succ}(X) & = X + 1 \\ \mathcal{P}_{zero} & = 1 \end{array}$$

then:

$$\begin{array}{c} 2(1) + X >_{\mathbb{N}} X \\ 2(X+1) + Y >_{\mathbb{N}} (2X+Y) + 1 \end{array}$$

whatever are the values of $X, Y \in \mathbb{N}$

Certificate for polynomial interpretation on $\mathbb{N}$

- The certificate require: the polynomials $\mathcal{P}_f$
- How to verify its correctness?
  - Check that each $\mathcal{P}_f$ is monotone in each variable
  - Check that, for every rule $l \rightarrow r \in \mathcal{R}$, we have $\mathcal{P}_l > \mathcal{P}_r$

Certificate for polynomial interpretation on $\mathbb{N}$

- The certificate require: the polynomials $\mathcal{P}_f$
- How to verify its correctness?
  - Check that each $\mathcal{P}_f$ is monotone in each variable
  - Check that, for every rule $l \rightarrow r \in \mathcal{R}$, we have $\mathcal{P}_l > \mathcal{P}_r$

CPF: termination certificate grammar (XML Schema)

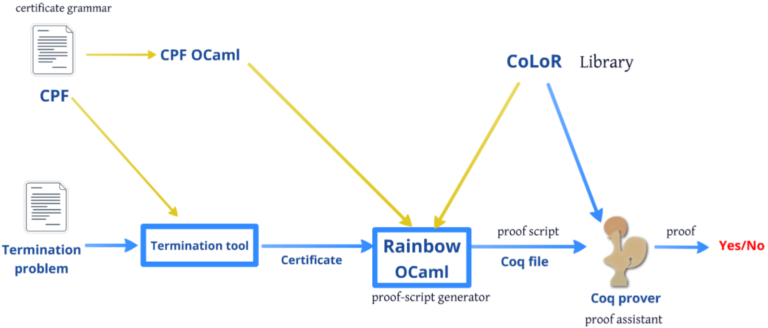
TPDB: termination problems data base

TermComp: annual international termination competition

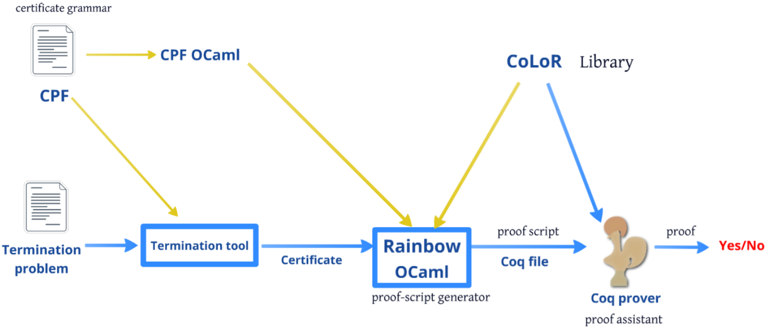
Outline

1. Software certification
2. Termination of rewriting and its certification
3. **Our approach**
4. Conclusion

Old Rainbow architecture: generate a Coq script

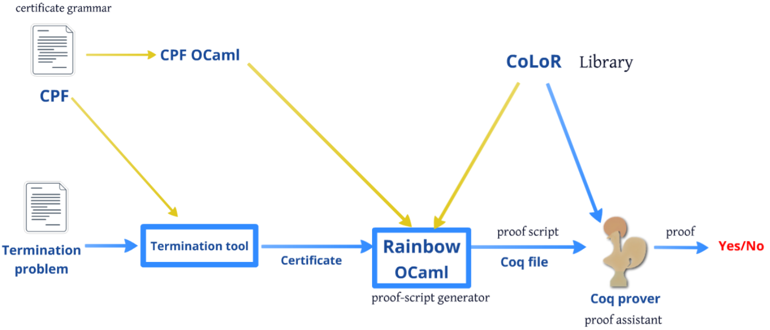


Old Rainbow architecture: generate a Coq script



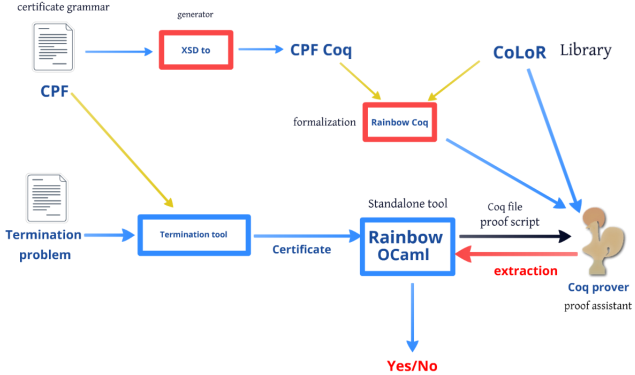
Advantages	Termination proofs can be re-used in Coq

Old Rainbow architecture: generate a Coq script

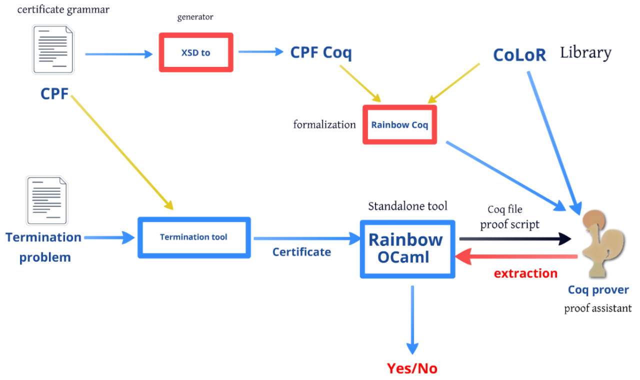


Advantages	Termination proofs can be re-used in Coq
Disadvantages	Coq is too slow Rainbow is not certified

New Rainbow architecture: formalize Rainbow itself



New Rainbow architecture: formalize Rainbow itself



Advantages	~~Termination proofs can be re-used in Coq~~
Disadvantages	~~Coq is too slow~~ ~~Rainbow is not certified~~

Outline

1. Software certification
2. Termination of rewriting and its certification
3. Our approach
4. Conclusion

Conclusion

- Developed a tool that generates from an XML Schema S:
    - An OCaml/Coq data type for representing XML files valid wrt S
    - An OCaml parsing function for XML files valid wrt S
- Defined and formally proved in Coq a termination certificate verifier for:
    - Polynomial interpretations
    - Dependency pairs and dependency graph decomposition

Conclusion

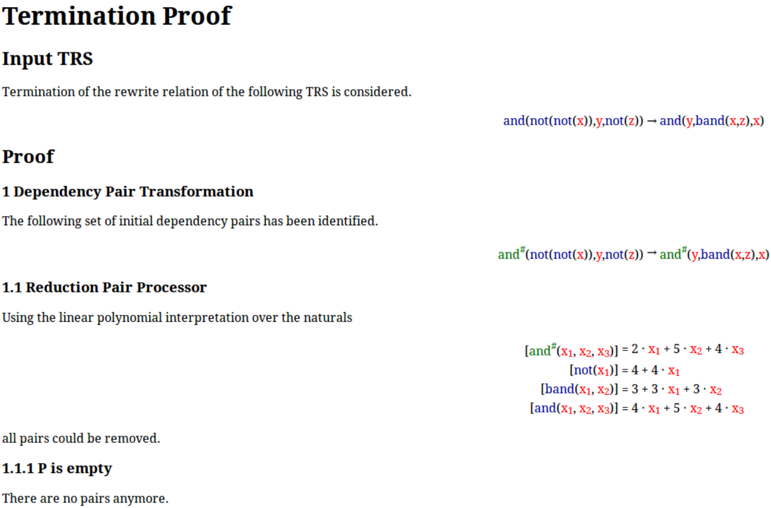
- Developed a tool that generates from an XML Schema S:
  - An OCaml/Coq data type for representing XML files valid wrt S
  - An OCaml parsing function for XML files valid wrt S
- Defined and formally proved in Coq a termination certificate verifier for:
  - Polynomial interpretations
  - Dependency pairs and dependency graph decomposition
- Future work: handle other termination techniques (matrix interpretations, arguments filtering, . . .)

Conclusion

- Developed a tool that generates from an XML Schema S:
  - An OCaml/Coq data type for representing XML files valid wrt S
  - An OCaml parsing function for XML files valid wrt S
- Defined and formally proved in Coq a termination certificate verifier for:
  - Polynomial interpretations
  - Dependency pairs and dependency graph decomposition
- Future work: handle other termination techniques (matrix interpretations, arguments filtering, . . .)

Thank you for your attention!

CPF:Termination proof example



Termination Proof

Input TRS

Termination of the rewrite relation of the following TRS is considered.

$$\mathrm{and}(\mathrm{not}(\mathrm{not}(x)),y,\mathrm{not}(z)) \to \mathrm{and}(y,\mathrm{band}(x,z),x)$$

Proof

1 Dependency Pair Transformation

The following set of initial dependency pairs has been identified.

$$\mathrm{and}^{\#}(\mathrm{not}(\mathrm{not}(x)),y,\mathrm{not}(z)) \to \mathrm{and}^{\#}(y,\mathrm{band}(x,z),x)$$

1.1 Reduction Pair Processor

Using the linear polynomial interpretation over the naturals

$$\begin{aligned}
[\mathrm{and}^{\#}(x_1, x_2, x_3)] &= 2 \cdot x_1 + 5 \cdot x_2 + 4 \cdot x_3\\
[\mathrm{not}(x_1)] &= 4 + 4 \cdot x_1\\
[\mathrm{band}(x_1, x_2)] &= 3 + 3 \cdot x_1 + 3 \cdot x_2\\
[\mathrm{and}(x_1, x_2, x_3)] &= 4 \cdot x_1 + 5 \cdot x_2 + 4 \cdot x_3
\end{aligned}$$

all pairs could be removed.

1.1.1 P is empty

There are no pairs anymore.

Tools

Rainbow

library: CoLoR
proof assistant: Coq
approach: deep embedding + extraction

CeTA

library: IsaFoR
proof assistant: Isabelle/HOL
approach: deep embedding + extraction

CiME3

library: Coccinelle
proof assistant: Coq
approach: shallow embedding + script generation