# Automated verification of termination certificates

Kim Quyen LY

University Joseph Fourier

17 November 2011

Supervisor: Frédéric BLANQUI

# Outline

# Motivation

- **Termination** is an important and difficult problem.
- In general this problem is undecidable.
- Many methods and criteria have been developed, and they are being used in various programs (termination provers)
- These programs are more and more complex, and their result difficult to check by hand.
- Every year in the **termination competition**, some tools are **disqualified** because of mistakes found in their proofs.
- For these tools to be used in the certification of critical systems and proof assistants, their results must be **certified**.

$\Rightarrow$ How to certify termination proofs generated by termination provers?

# 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 an integer polynomial $P_t$ by recursively composing the polynomials interpreting the function symbols occurring in the term $t$
- then a program defined by a set $R$ of rules terminates if:
  - each $P_f$ is monotone in each variable
  - for every rule $l \to r \in R, P_l > P_r$ on $\mathbb{N}$

certificate: polynomials $P_f$
such a certificate is correct if the above conditions are satisfied

# Goal

develop a safe, efficient and modular termination certificate verifier

our solution:

- write a verifier in Coq
- prove its correctness using the CoLoR library
- extract it to OCaml

# XML

termination certificates are given as XML files

```
<root>
  <child1 attribute1="value1">
    <subchild> .... </subchild>
  </child1>
  <child2>
    <subchild> .... </subchild>
  </child2>
</root>
```

abstractly, an XML file is a tree whose nodes are tagged and may have attributes (leaves are strings)

# XML Schema (XSD)

an XSD file describes a class of XML files by defining the possible tags and attributes, and how tagged elements can be composed

XSD type = set of XML elements

It is itself defined as an XML file! with the following tags:

- `<sequence>XSD_type1 XSD_type2 ...</sequence>`: product type
- `<choice>XSD_type1 XSD_type2 ...</choice>`: union type
- `<group name ="<name>">XSD_type</group>`: names a type
- `<element name="<tag>">XSD_type</element>`: declares a tag, its attributes and its possible sons

remark: XSD definitions need not be ordered and can be forward or backward referenced

# XSD example (part 1/2)

```
<xs:group name="symbol">
  <xs:annotation>
    <xs:documentation>is used as a function symbol in terms, orderings, ....</xs:documentation>
  </xs:annotation>
  <xs:choice>
    <xs:element ref="name"/>
    <xs:element name="sharp">
      <xs:complexType>
        <xs:sequence>
          <xs:group ref="symbol"/>
        </xs:sequence>
      </xs:complexType>
    </xs:element>
    <xs:element name="labeledSymbol">
      <xs:complexType>
        <xs:sequence>
          <xs:group ref="symbol"/>
          <xs:group ref="label"/>
        </xs:sequence>
      </xs:complexType>
    </xs:element>
  </xs:choice>
</xs:group>
```

# CPF

the CPF format is regularly modified and extended with new features, it is useful to have a tool that can automatically generate in OCaml and Coq:

- data structures
- parsers
- pretty-printers

for that format

# XSD example (part 2/2)

OCaml type corresponding to previous XSD definition:

```
type label =
| Label_numberLabel of nonNegativeInteger list
| Label_symbolLabel of symbol list
and symbol =
| Symbol_name of name
| Symbol_sharp of symbol
| Symbol_labeledSymbol of symbol * label
```

problem in Coq: detect mutually inductive types

# Dependence relation

type expressions: $T = C | T \Rightarrow T$
type definition for a type constant $C$: a type for each constructor

$C$ *def-depends* on $D$, written $C \leadsto D$, if:
$C$ has a constructor in the type of which $D$ occurs

let $\geq$ ($\simeq$) be the transitive (reflexive and symmetric) closure of $\leadsto$

- two types $C$ and $D$ *depend* on each other if $C \simeq D$
- a type $C$ can be defined *before* $D$ if $C < D$

the transitive closure and then the computation and the ordering of equivalence classes can be done using computations on boolean matrices

# General picture

- define a boolean function:

  `Fixpoint check : cpf -> bool := ...`

- prove that it is correct:

  `Lemma check_ok : forall c, check c = true -> WF (red c)`

  where `(red c)` is the rewrite relation defined in `c`

for returning useful information in case of failure, instead of `bool` we use:

```
Inductive result (A : Type) : Type :=
| Ok : A -> result A
| Ko : error -> result A.
```

# Translation of CPF types to CoLoR types

problem: CoLoR terms are defined wrt some arity function

```
Record Signature : Type := mkSignature {
  symbol :> Type;
  arity : symbol -> nat;
  beq_symb : symbol -> symbol -> bool;
  beq_symb_ok : forall x y, beq_symb x y = true <-> x = y
}.
Inductive term : Type :=
  | Var : variable -> term
  | Fun : forall f : Sig, vector term (arity f) -> term.
```

- the arity can be computed by examining rules
- the translation of terms may fail

# Remarks

- check requires many auxiliary functions for testing equality on CPF types (symbols, terms, etc.)
- the induction principles automatically generated by Coq for some types is too weak and needs to be redefined

this is currently done by hand $\Rightarrow$ generate this automatically?

# Problem with CoLoR

problem: CoLoR uses modules and functors

they need to be instantiated

- modules cannot be defined inside sections
- certificates are recursive

modules need to be first-class object

solution: change CoLoR to use records instead.

# What did I do?

- generator of Coq type definitions from XSD
- generator of OCaml type definitions and parsing functions from XSD
- replaced modules by records in some CoLoR files
- translation of CPF types into CoLoR types
- definition of a certificate verifier for polynomial interpretations
- correctness proof almost finished for polynomial interpretations

# Some figures

cpf.xsd: 1400 lines of XML

ml_of_xsd.ml: 400 lines of OCaml

cpf.ml: 1200 lines of generated OCaml

coq_of_xsd.ml: 200 lines of OCaml

cpf.v: 250 lines of generated Coq

rainbow.v: 1200 lines of Coq

modified CoLoR files: 650 lines of Coq

# What did I learn?

- XML and XML Schema
- OCaml
- more on Coq

remark: the use of dependent types in CoLoR makes definitions and proofs more difficult

# Work plan for 2012

- November 2011: finish the new implementation of the OCaml and Coq type definitions generator from XSD
- December 2011: finish the correctness proof for polynomial interpretations on integers
- January 2012: extraction to OCaml, linking with CPF parser and testing on the Termination Problem Data Base (TPDB)
- February 2012 - October 2012: extension to other termination techniques

**Thank you for your attention!!!**