Interoperability between proof systems: The triumvirate of automation, expressivity, and safety

Chantal Keller

November, 28th 2016
Trust automatic devices

Pas mal, hein! ce machin-là…
Trust automatic devices
The triumvirate

SMT/FO provers

proof producing SMT/FO provers
Beagle
Why3 method B

F*
Psyche
metis

PVS
Isabelle/HOL

Coq/Agda/Beluga
Dedukti/MMT
The triumvirate of automation, expressivity, and safety.
Why so many (general) provers?

A wide range of applications, such as:

- deductive verification
- proofs of programs
- “mathematical” proofs
- formalizing metatheory
- induction/coinduction
- reasoning on/with computation
- . . .
Interoperability: get the best of everything

But:

- at what cost/effort?
- how agnostic can the systems be?
- portability?
- automation?
Interoperability: get the best of everything

But:

- at what cost/effort?
- how agnostic can the systems be?
- portability?
- automation?

In this talk: three examples of interoperability between two systems A and B
1. Autarkic approach (certified provers)

system A
1. Autarkic approach (certified provers)

- System A
- System B

Interoperability between proof systems: The triumvirate of automation, expressivity, and safety Chantal Keller 6 / 24
1. Autarkic approach (certified provers)
1. Autarkic approach (certified provers)
Example: Ergo (A = Coq; B = subset of Alt-Ergo)

In Coq:

the prover:

\[
\text{ergo} : \text{formula} \rightarrow \text{bool}
\]
certified:

\[
\text{ergo\_correct} : \forall f. \text{ergo} f = \text{true} \Rightarrow \text{valid} f
\]

Show that a formula \( f_0 \) is valid:

(\text{ergo\_correct})

\[
\forall f. \text{ergo} f = \text{true} \Rightarrow \text{valid} f
\]

\[
\text{ergo} f_0 = \text{true}
\]

valid \( f_0 \)
Example: Ergo (A = Coq; B = subset of Alt-Ergo)
Example: Ergo (A = Coq; B = subset of Alt-Ergo)

In Coq:

- the prover:
  \[\text{ergo} : \text{formula} \rightarrow \text{bool}\]
- certified:
  \[\text{ergo\_correct} : \forall f. \text{ergo} f = \text{true} \Rightarrow \text{valid} f\]

Show that a formula \(f_0\) is valid:

\[
\frac{(\text{ergo\_correct})}{\forall f. \text{ergo} f = \text{true} \Rightarrow \text{valid} f} \quad \text{ergo} f_0 = \text{true} \quad \text{valid} f_0
\]
Example: Ergo \( (A = \text{Coq}; B = \text{subset of Alt-Ergo}) \)

In Coq:

- the prover:
  
  \[
  \text{ergo} : \text{formula} \rightarrow \text{bool}
  \]

- certified:
  
  \[
  \text{ergo\_correct} : \forall f. \text{ergo\_correct} f = \text{true} \Rightarrow \text{valid} f
  \]

Show that a formula \( f_0 \) is valid:

\[
\begin{align*}
(\text{ergo\_correct}) \\
\forall f. \text{ergo}\ f = \text{true} \Rightarrow \text{valid} f \\
\text{ergo}\ f_0 = \text{true} \\
\text{valid}\ f_0
\end{align*}
\]

\(\rightarrow\) computational reflection
Advantages and limitations

+ shared representation of formulas
+ correctness established once and for all
+ formal correctness of the algorithms

- really hard to prove
- really hard to maintain or improve ( fixes the implementation)
- not always possible
Our criteria:

- at what cost/effort?
  - statements? none
  - proofs? huge
- how agnostic can the systems be? not at all
- portability? none
- automation? medium
2. Skeptical approach (certifying provers)
2. Skeptical approach (certifying provers)

system A

system B
2. Skeptical approach (certifying provers)
2. Skeptical approach (certifying provers)

![Diagram showing interoperability between proof systems: The triumvirate of automation, expressivity, and safety.]

Chantal Keller
2. Skeptical approach (certifying provers)

Interoperability between proof systems: The triumvirate of automation, expressivity, and safety
 Chantal Keller 10 / 24
2. Skeptical approach (certifying provers)
2. Skeptical approach (certifying provers)

Interoperability between proof systems: The triumvirate of automation, expressivity, and safety Chantal Keller 10 / 24
### Examples (non-exhaustive list)

<table>
<thead>
<tr>
<th>Tool</th>
<th>System A</th>
<th>System B</th>
<th>Reconstruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>metis</td>
<td>HOL systems</td>
<td>metis</td>
<td>HOL p.t.</td>
</tr>
<tr>
<td>sledgehammer</td>
<td>Isabelle/HOL Coq</td>
<td>SAT/SMT/FO Zenon</td>
<td>lsa. tactics Coq p.t./tactics</td>
</tr>
<tr>
<td>Zenon</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CeTA</td>
<td>Isabelle/HOL Coq</td>
<td>termination SAT/SMT HOL Light</td>
<td>checker reflexive checker reflexive checker</td>
</tr>
<tr>
<td>SMTCoq</td>
<td>HOL Light &amp; Coq</td>
<td></td>
<td></td>
</tr>
<tr>
<td>HOL Light &amp; Coq</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Advantages and limitations

+ correctness easier to establish
+ System B may evolve independently
+ pre-processing ⇒ efficiency and various provers
+ caching

- no shared representation of formulas
- if the (transformed) certificate is false?
- System B needs instrumentation
Our criteria:

- at what cost/effort?
  - statements? from small to huge
  - proofs? small

- how agnostic can the systems be? only certificates

- portability? great

- automation? medium to good
About embeddings

Different ways to embed the terms of System B into System A
Deep embedding

Different ways to embed the terms of System B into System A

- How to proceed:
  Data-type in A to represent the objects of B

- Example of the simply typed $\lambda$-calculus in Coq:
  Inductive typeD : Type :=
  | o : typeD
  | a : typeD → typeD → typeD.

  Inductive termD : Type :=
  | Var: string → termD
  | Lam: string → typeD → termD → termD
  | App: termD → termD → termD.

- Representation of $\lambda x : bool . x$:
  Lam "x" o (Var "x")
Shallow embedding

Different ways to embed the terms of System B into System A

How to proceed:
- Objects of A to represent the objects of B
- Example of the simply typed $\lambda$-calculus in Coq:
  
  \[
  \text{Definition typeS := Type.}
  \]

- Representation of $\lambda x : bool. x$:
  
  \[
  \text{fun x: bool => x}
  \]
Comparison between embeddings

Summary:

<table>
<thead>
<tr>
<th>Deep</th>
<th>Shallow</th>
</tr>
</thead>
<tbody>
<tr>
<td>Inductive typeD := ...</td>
<td>Definition typeS := Type</td>
</tr>
<tr>
<td>Inductive termD := ...</td>
<td>Coq proof term</td>
</tr>
<tr>
<td>A proof in A that the theorem is provable in B</td>
<td>The theorem is proved in A in the fragment corresponding to B</td>
</tr>
<tr>
<td>Access the structure of terms</td>
<td>Coq theorems</td>
</tr>
<tr>
<td>- induction</td>
<td></td>
</tr>
<tr>
<td>- computational reflection</td>
<td></td>
</tr>
<tr>
<td>API to the external world</td>
<td></td>
</tr>
</tbody>
</table>

Interoperability between proof systems: The triumvirate of automation, expressivity, and safety Chantal Keller 15 / 24
Comparison between embeddings

Summary:

<table>
<thead>
<tr>
<th>Deep</th>
<th>Shallow</th>
</tr>
</thead>
<tbody>
<tr>
<td>Inductive typeD := ...</td>
<td>Definition typeS := Type</td>
</tr>
<tr>
<td>Inductive termD := ...</td>
<td>Coq proof term</td>
</tr>
<tr>
<td>A proof in A that the theorem is provable in B</td>
<td>The theorem is proved in A in the fragment corresponding to B</td>
</tr>
<tr>
<td>Access the structure of terms</td>
<td>Concrete Coq terms</td>
</tr>
<tr>
<td>- induction</td>
<td></td>
</tr>
<tr>
<td>- computational reflection</td>
<td></td>
</tr>
<tr>
<td>API to the external world</td>
<td>Coq theorems</td>
</tr>
</tbody>
</table>

→ deeply check, then interpret into shallow
What is a certificate?

How to prove that 5 divides 45?

- Lemma: \( \forall d \ n, d|n \iff (n = 0) \lor (n \geq d \land d|(n - d)) \)
- Apply this lemma 10 times:
  - 5 divides 0
  - thus 5 divides 5
  - thus ...
  - thus 5 divides 45

↕️ linear time and space
↕️ lemma easy to prove
What is a certificate?

How to prove that 5 divides 45?

- Implement the function modulo
- Lemma: $\forall d \ n, d|n \iff n \equiv 0 \ (d)$
- Apply this lemma once and compute:
  - $45 \equiv 0(5)$ (computation)
  - thus 5 divides 45

$\rightarrow$ linear time but constant space!
$\rightarrow$ lemma still easy to prove
What is a certificate?

How to prove that 5 divides 45?

- Implement the divisibility rule of 5
- Lemma: \( \forall n, 5 \mid n \iff \text{rule5}(n) = \text{true} \)
- Apply this lemma once and compute:
  - \( \text{rule5}(45) = \text{true} \) (computation)
  - thus 5 divides 45

\( \rightarrow \) constant time and space!
\( \leftrightarrow \) lemma slightly more difficult to establish
A good format of certificate

Good certificate:
- easy to check
- fast to generate

Good checkers:
- efficient
- modular: accept various formats!
A universal format of certificate?

Propositions of standards:

- LFSC
- veriT/SMTCoq
- TPTP
- Open Theory
- dedukti
- ProofCert
- flexiformal proofs (OpenMath)
- . . . ?
3. *A priori* approaches

**Built-in interoperability:**

- decide in advance the interoperability you want with System B
- build System A around it
Example: F*

- impure functional programming language
- rich type system: dependent and refined types (to express various properties on programs)
- type checking: designed to use the Z3 SMT solver
- Curry-Howard: programs are proofs
Example: F*

- impure functional programming language
- rich type system: dependent and refined types
  (to express various properties on programs)
- type checking: designed to use the Z3 SMT solver
- Curry-Howard: programs are proofs

```
val u : nat -> Tot nat
let rec u n = if n = 0 then 0 else u (n-1)
```
Example: F*

- impure functional programming language
- rich type system: dependent and refined types
  (to express various properties on programs)
- type checking: designed to use the Z3 SMT solver
- Curry-Howard: programs are proofs

```ml
val u : nat -> Tot nat
let rec u n = if n = 0 then 0 else u (n-1)

val induction : n:nat -> Lemma (ensures (u n = 0))
let rec induction n =
  if n = 0 then () else induction (n-1)
```

Interoperability between proof systems: The triumvirate of automation, expressivity, and safety Chantal Keller 20 / 24
Example: Lean

- interactive theorem prover
- based on proof-irrelevant Type Theory
- elaboration mechanism with built-in automation
Example: Lean

- interactive theorem prover
- based on proof-irrelevant Type Theory
- elaboration mechanism with built-in automation

```lean
definition u : nat → nat
| u 0 := 0
| u (n+1) := u n
```

Interoperability between proof systems: The triumvirate of automation, expressivity, and safety  Chantal Keller 21 / 24
Example: Lean

- interactive theorem prover
- based on proof-irrelevant Type Theory
- elaboration mechanism with built-in automation

```lean
definition u : nat → nat
| u 0 := 0
| u (n+1) := u n

theorem induction : ∀ n, u n = 0
| induction 0 := rfl
| induction (n+1) := induction n
```
Our criteria:

- At what cost/effort?
  - Statements? From small to huge
  - Proofs? From small to huge
- How agnostic can the systems be? Good
- Portability? Bad
- Automation? Really good
Summary

<table>
<thead>
<tr>
<th>Criterion</th>
<th>Autarkic</th>
<th>Skeptical</th>
<th>A priori</th>
</tr>
</thead>
<tbody>
<tr>
<td>effort (statements)</td>
<td>++</td>
<td>−</td>
<td>−</td>
</tr>
<tr>
<td>effort (proofs)</td>
<td>−</td>
<td>++</td>
<td>−</td>
</tr>
<tr>
<td>agnostic</td>
<td>−</td>
<td>++</td>
<td>+</td>
</tr>
<tr>
<td>portability</td>
<td>−</td>
<td>++</td>
<td>−</td>
</tr>
<tr>
<td>automation</td>
<td>+</td>
<td>+</td>
<td>++</td>
</tr>
</tbody>
</table>
Takeaway

Some lessons for new systems:

- interoperability is hard!
- think from the very beginning that people may want to use your system differently
- certificates (possibly in a standard), API, ...