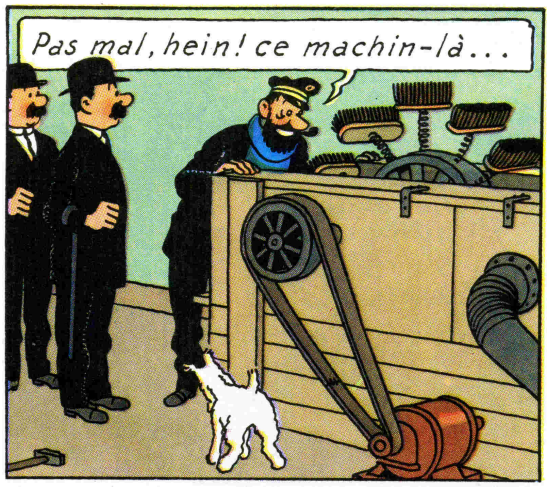
Interoperability between proof systems: The triumvirate of automation, expressivity, and safety

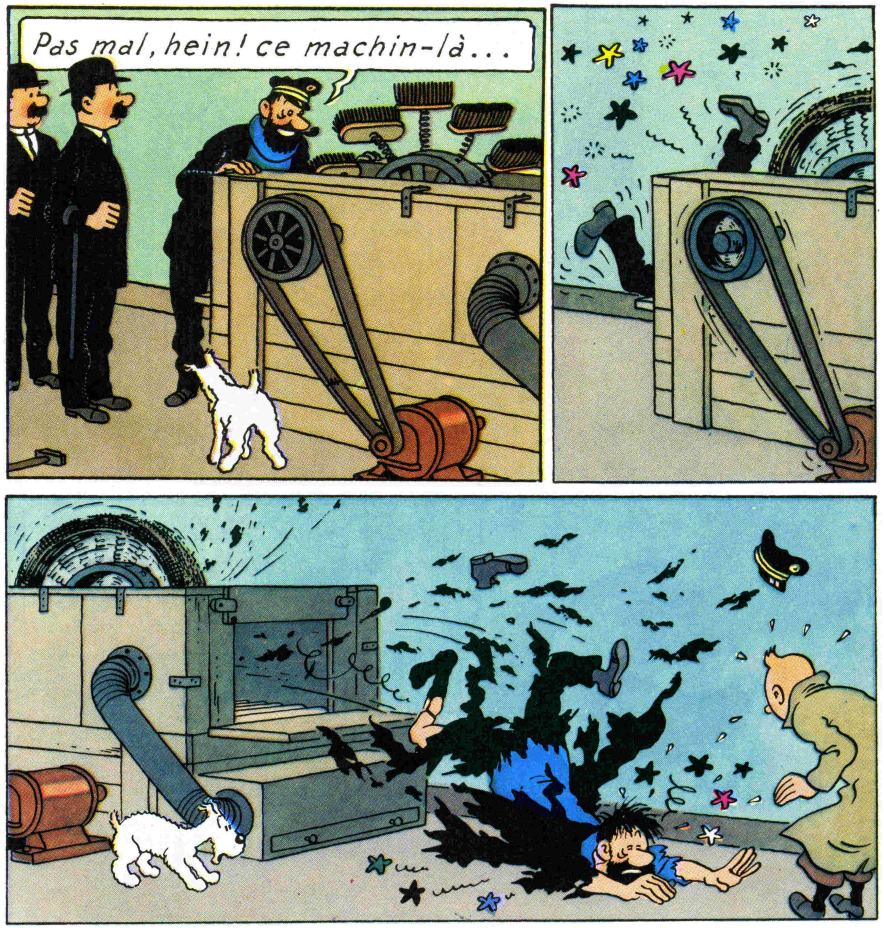
Chantal Keller

November, 28th 2016

Trust automatic devices



Trust automatic devices



The triumvirate

SMT/FO provers

proof producing SMT/FO provers

Beagle

Why3 method B

F*

Psyche

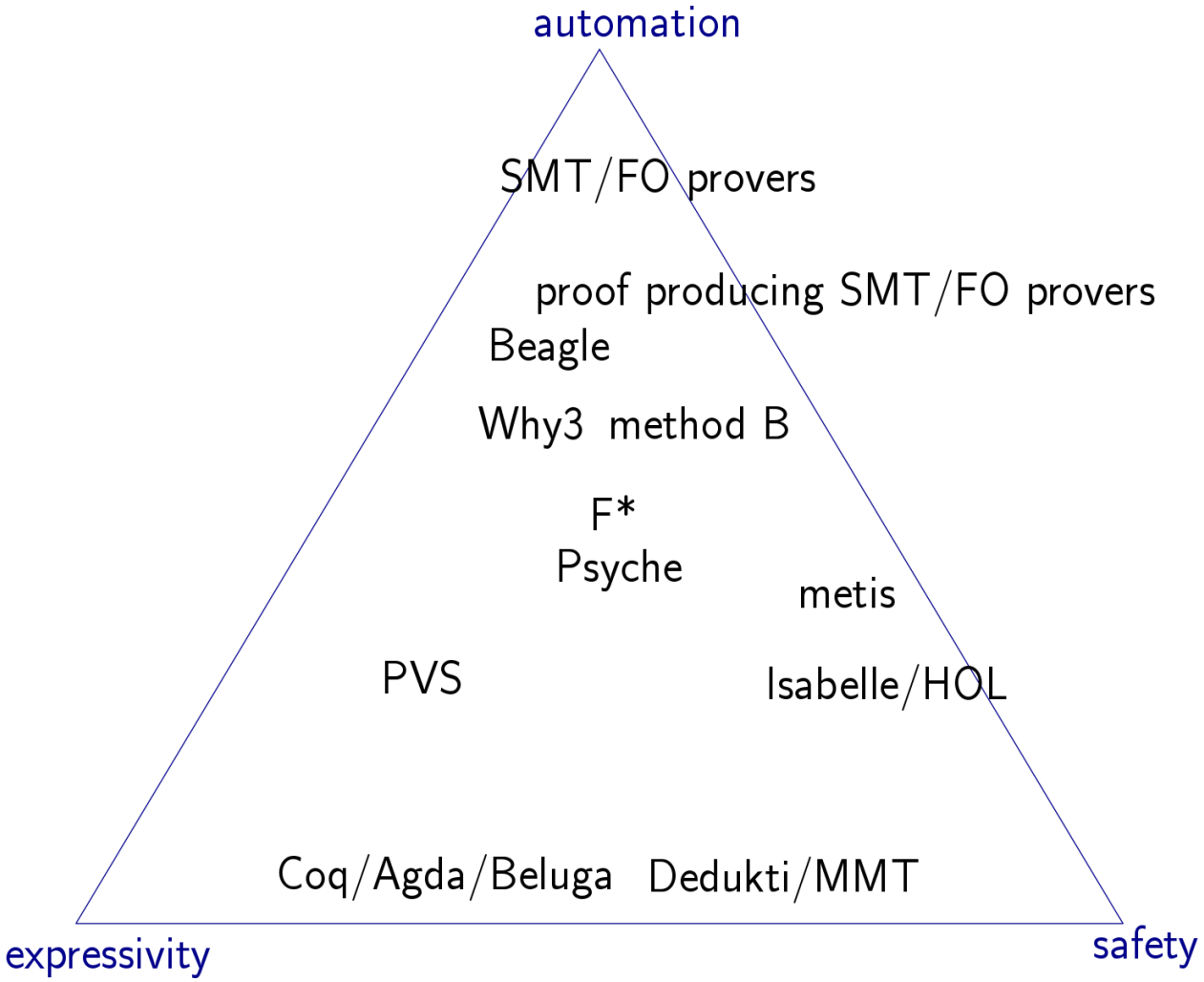
metis

PVS

Isabelle/HOL

Coq/Agda/Beluga Dedukti/MMT

The triumvirate



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?

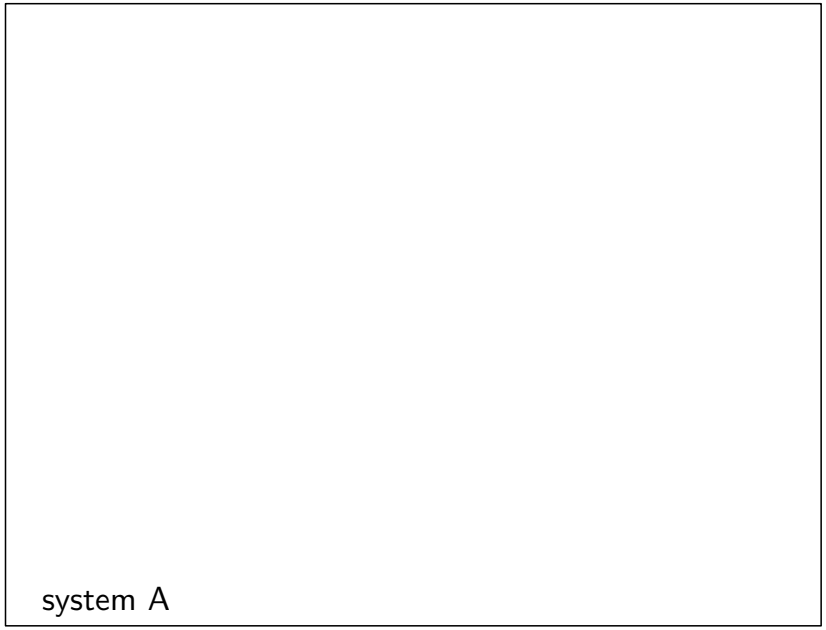
Interoperability: get the best of everything

But:

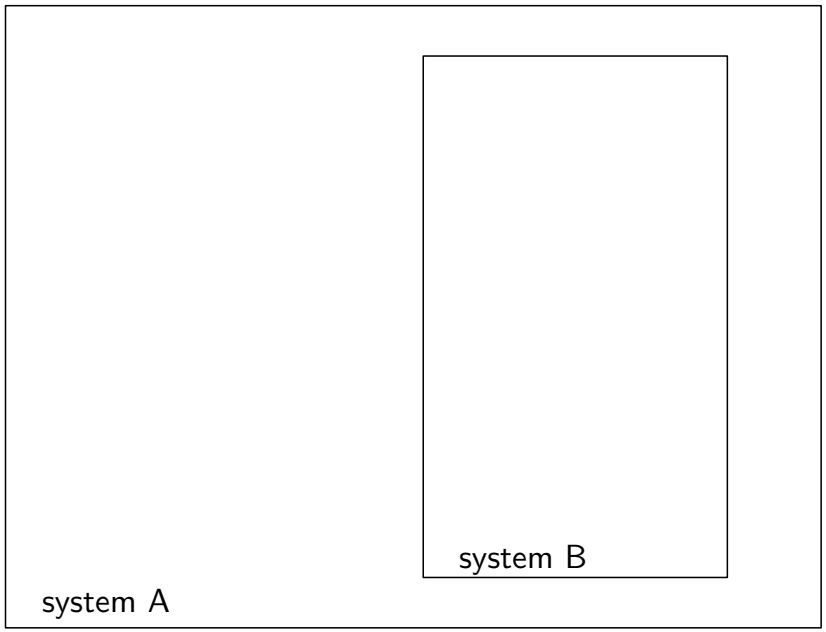
- at what cost/effort?
- how agnostic can the systems be?
- portability?
- automation?

In this talk: three examples of interoperability between two systems A and B

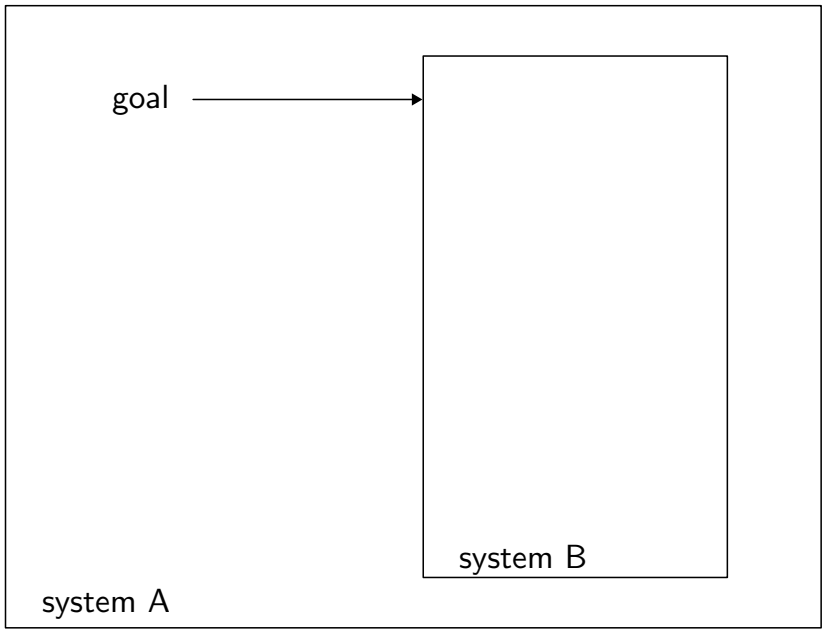
1. Autarkic approach (certified provers)



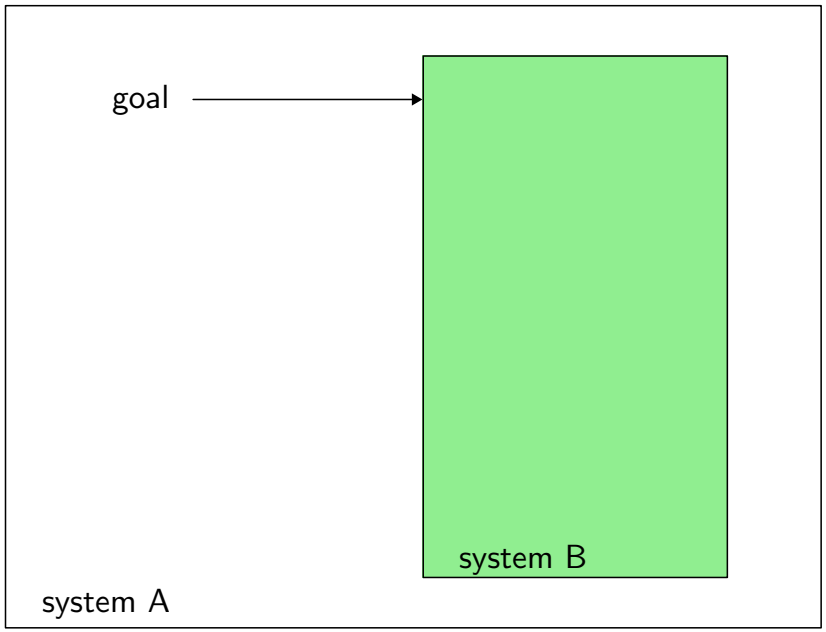
1. Autarkic approach (certified provers)



1. Autarkic approach (certified provers)



1. Autarkic approach (certified provers)



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

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

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

In Coq:

- the prover:
  `ergo : formula → bool`
- certified:
  `ergo_correct : ∀f. ergo f = true ⇒ valid f`

Show that a formula $\mathtt{f}_0$ is valid:

$$\dfrac{\dfrac{\text{(ergo\_correct)}}{\forall \mathtt{f}.\ \ \mathtt{ergo\ f = true} \Rightarrow \mathtt{valid\ f}} \qquad \dfrac{}{\mathtt{ergo\ f_0 = true}}}{\mathtt{valid\ f_0}}$$

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

In Coq:

- the prover:
  `ergo : formula → bool`
- certified:
  `ergo_correct : ∀f. ergo f = true ⇒ valid f`

Show that a formula $\mathtt{f}_0$ is valid:

$$\dfrac{\dfrac{\text{(ergo\_correct)}}{\forall \mathtt{f}.\ \ \mathtt{ergo\ f = true} \Rightarrow \mathtt{valid\ f}} \qquad \dfrac{}{\mathtt{ergo\ f_0 = true}}}{\mathtt{valid\ f_0}}$$

↪ **computational reflection**

Advantages and limitations

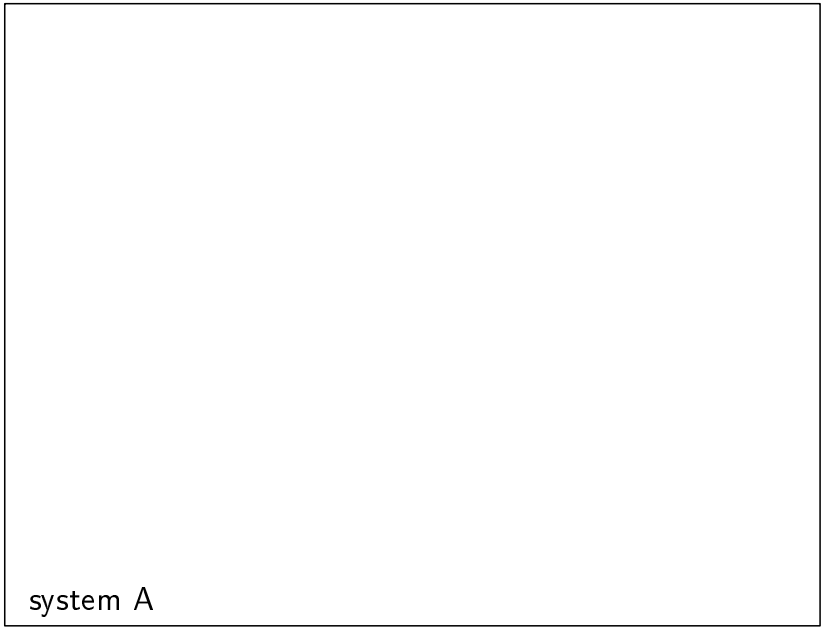
+ shared representation of formulas
+ correctness established once and for all
+ formal correctness of the algorithms

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

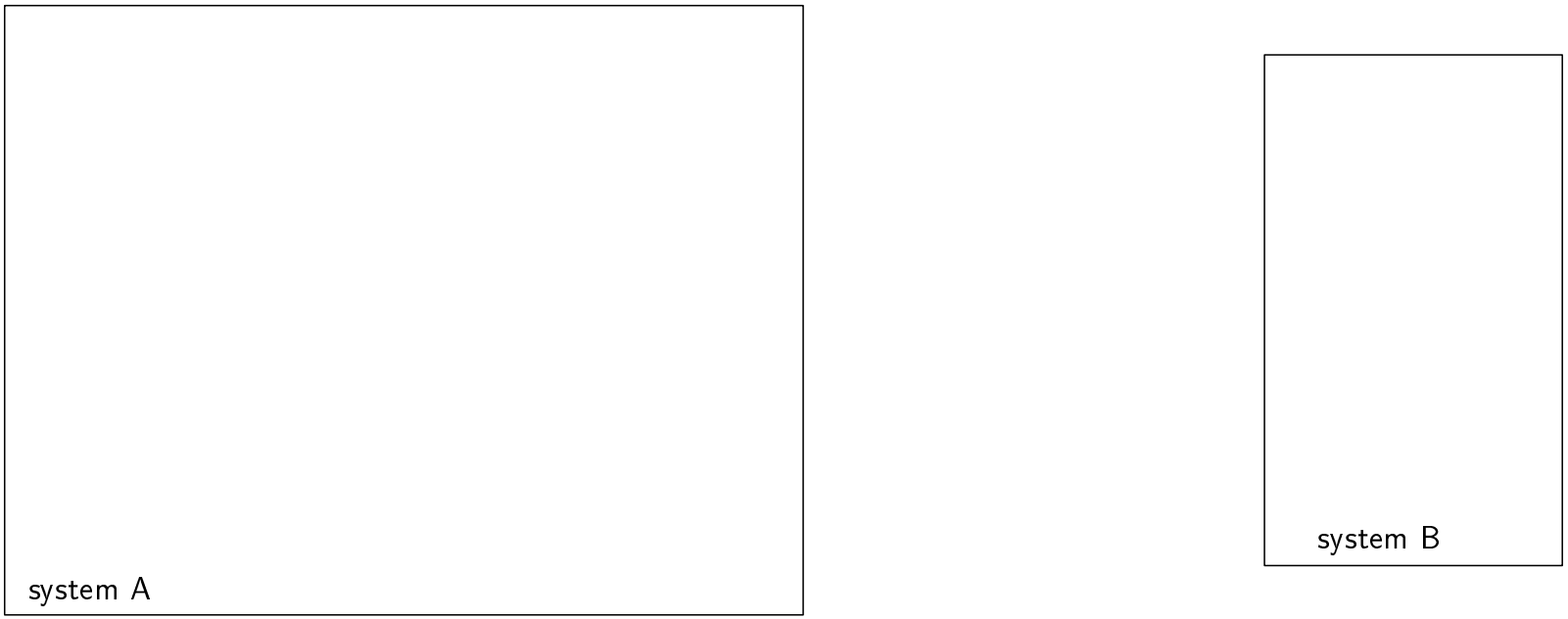
Our criteria:

- at what cost/effort?
  - statements? none
  - proofs? huge
- how agnostic can the systems be? not at all
- portability? none
- automation? medium

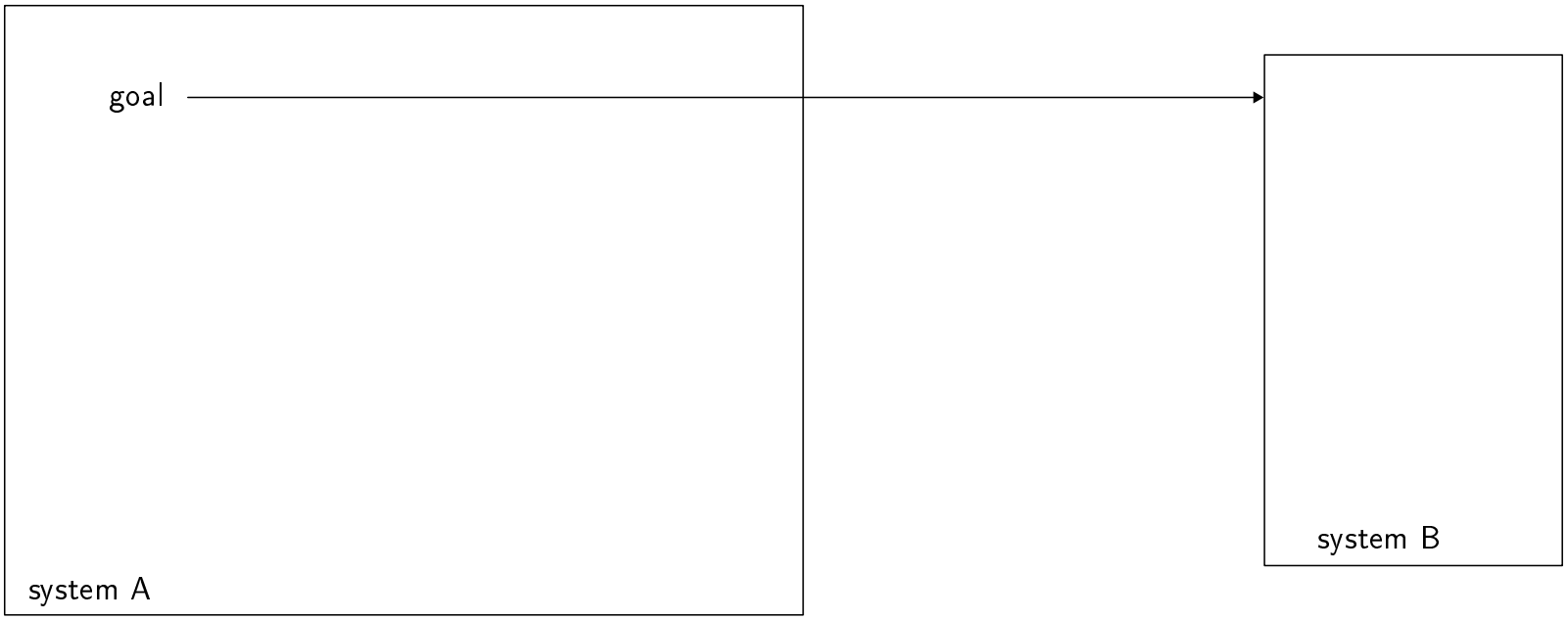
2. Skeptical approach (certifying provers)



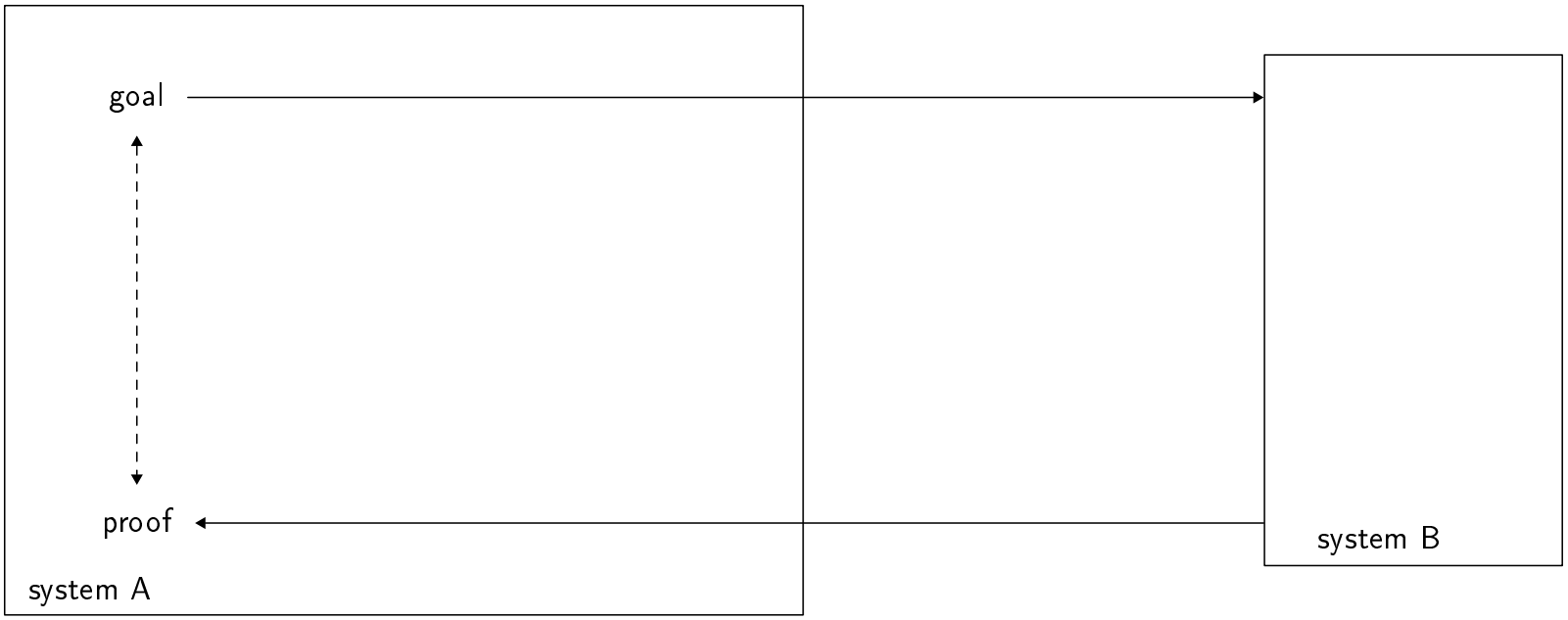
2. Skeptical approach (certifying provers)



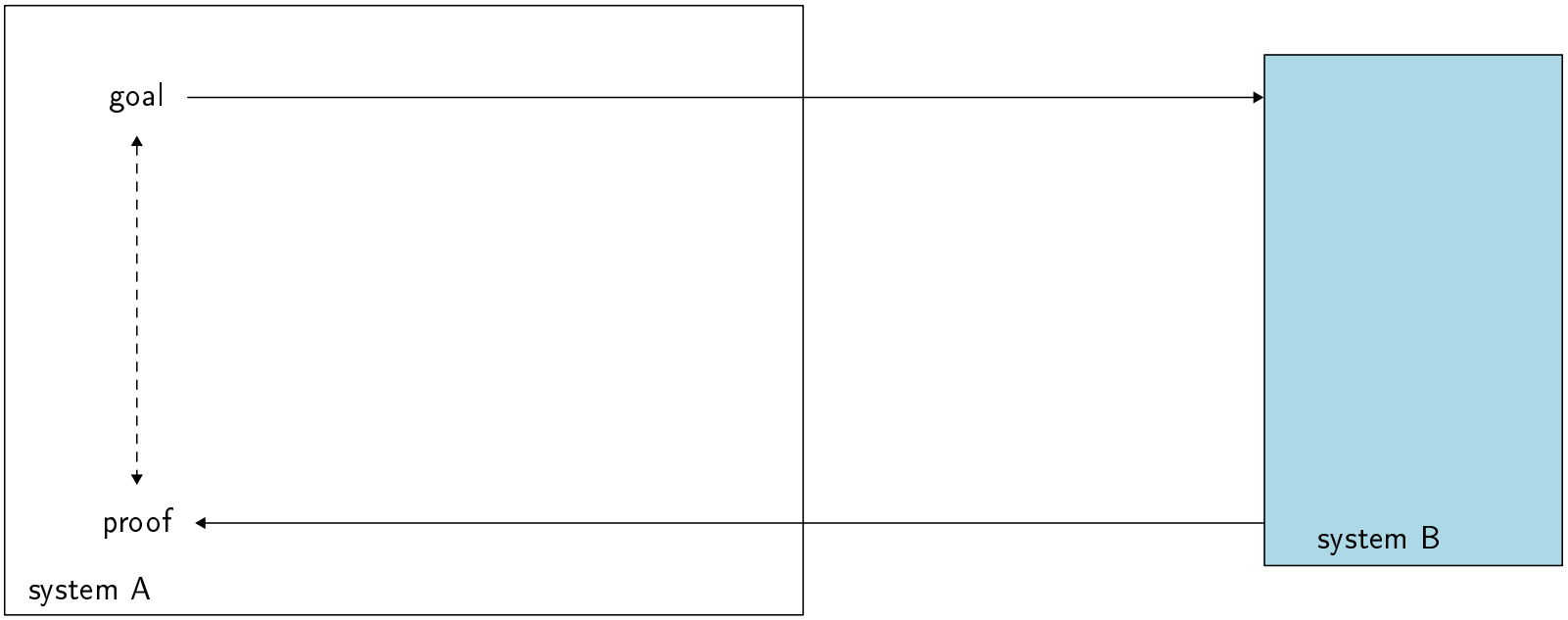
2. Skeptical approach (certifying provers)



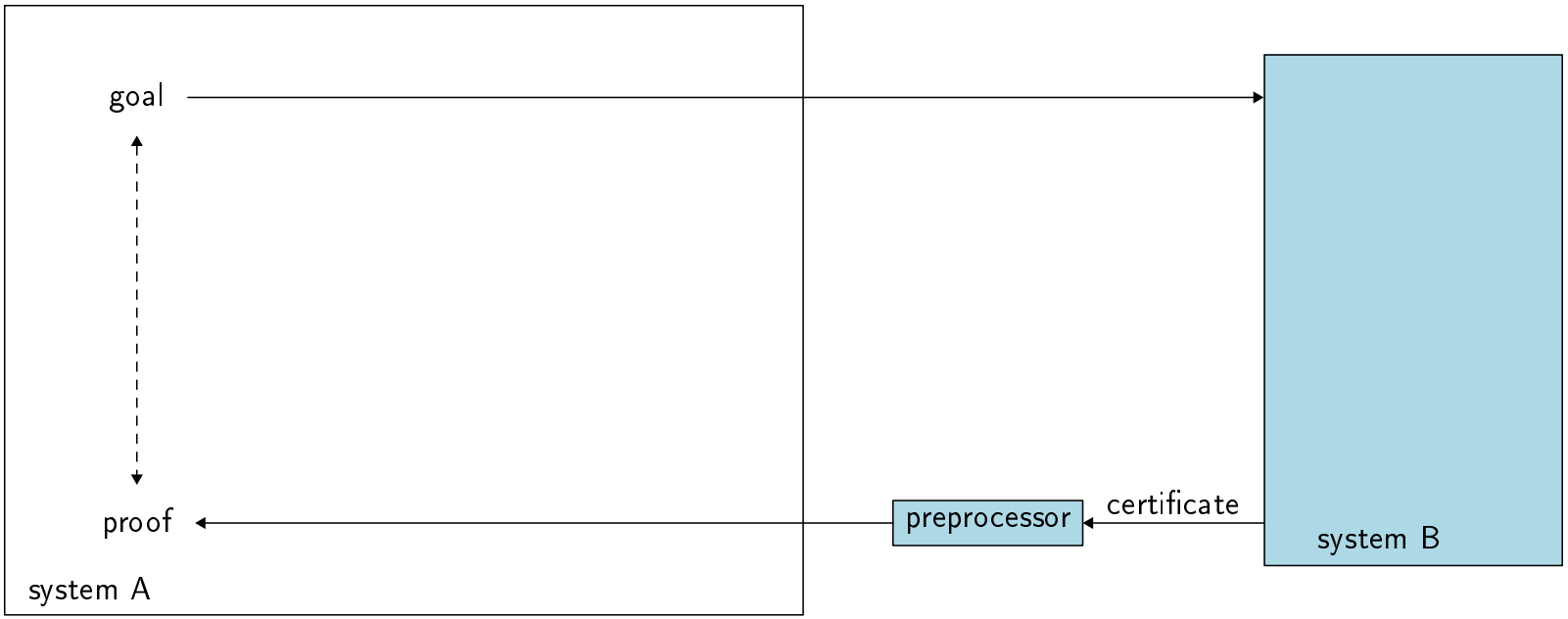
2. Skeptical approach (certifying provers)



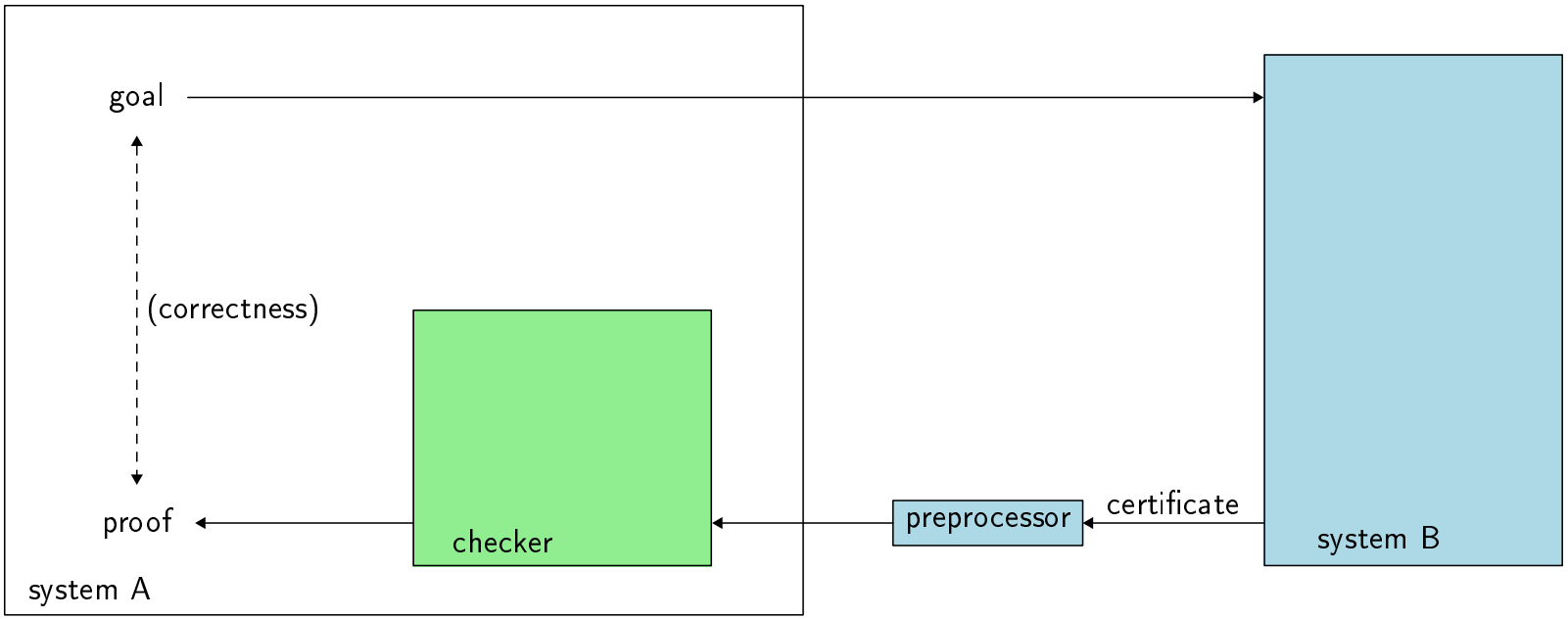
2. Skeptical approach (certifying provers)



2. Skeptical approach (certifying provers)



2. Skeptical approach (certifying provers)



Examples (non-exhaustive list)

Tool	System A	System B	Reconstruction
metis	HOL systems	metis	HOL p.t.
sledgehammer	Isabelle/HOL	SAT/SMT/FO	Isa. tactics
Zenon	Coq	Zenon	Coq p.t./tactics
CeTA	Isabelle/HOL	termination	checker
SMTCoq	Coq	SAT/SMT	reflexive checker
HOL Light & Coq	Coq	HOL Light	reflexive checker

Advantages and limitations

+ correctness easier to establish
+ System B may evolve independently
+ pre-processing $\Rightarrow$ 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 λ-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 λ-calculus in Coq:

```
Definition typeS := Type.
```

- Representation of $\lambda x : bool.x$:
  `fun x:  bool => x`

Comparison between embeddings

Summary:

Deep	Shallow
`Inductive typeD := ...` `Inductive termD := ...`	`Definition typeS := Type`
`Inductive derivD := ...` A proof in A that the theorem is provable in B	Coq proof term The theorem is proved in A in the fragment corresponding to B
Access the structure of terms - induction - computational reflection	Concrete Coq terms
API to the external world	Coq theorems

Comparison between embeddings

Summary:

Deep	Shallow
`Inductive typeD := ...` `Inductive termD := ...`	`Definition typeS := Type`
`Inductive derivD := ...` A proof in A that the theorem is provable in B	Coq proof term The theorem is proved in A in the fragment corresponding to B
Access the structure of terms - induction - computational reflection	Concrete Coq terms
API to the external world	Coq theorems

$\hookrightarrow$ deeply check, then interpret into shallow

What is a certificate?

How to prove that 5 divides 45?

- Lemma: $\forall d\ n, d|n \Leftrightarrow (n = 0) \vee (n \geqslant d \wedge d|(n - d))$
- Apply this lemma 10 times:
  - 5 divides 0
  - thus 5 divides 5
  - thus ...
  - thus 5 divides 45

$\hookrightarrow$ linear time and space

$\hookrightarrow$ lemma easy to prove

What is a certificate?

How to prove that 5 divides 45?

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

$\hookrightarrow$ linear time but constant space!
$\hookrightarrow$ lemma still easy to prove

What is a certificate?

How to prove that 5 divides 45?

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

$\hookrightarrow$ constant time and space!
$\hookrightarrow$ 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

```
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)
```

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

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

Example: Lean

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

```
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

Criterion	Autarkic	Skeptical	*A priori*
effort (statements)	++	–	-
effort (proofs)	–	++	-
agnostic	–	++	+
portability	–	++	-
automation	+	+	++

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, ...