Pollack-inconsistency

Freek Wiedijk

Radboud University Nijmegen

2010 07 15 , 12 : 00

UITP 2010

Pollack-inconsistency

Freek Wiedijk

Radboud University Nijmegen

2010 07 15 , 12 : 00

UITP 2010

when can one trust a proof assistant?

How to Believe a Machine-Checked Proof[1]

Robert Pollack
BRICS, [2] *Computer Science Dept., Aarhus University*
DK-8000 Aarhus C, Denmark

1 Introduction

Suppose I say "Here is a machine-checked proof of Fermat's last theorem (FLT)". How can you use my putative machine-checked proof as evidence for belief in FLT? I start from the position that you must have some personal experience of understanding to attain belief, and to have this experience you must engage your intuition and other mental processes which are impossible to formalise.

By machine-checked proof I mean a formal derivation in some given formal system; I am talking about derivability, not about truth. Further, I want to talk about *actually* believing an *actual* formal proof, not about formal proofs in principle; to be interesting, any approach to this problem must be feasible. You might try to read my proof, just as you would a proof in a journal; however, with the current state of the art, this proof will surely be too long for you to have confidence that you have understood it. This paper presents a technological approach for reducing the problem of believing a formal proof to the same psychological and philosophical issues as believing a conventional proof in a mathematics journal. The approach is not entirely successful philosophically as there seems to be a fundamental difference between machine checked mathematics, which depends on empirical knowledge about the physical world, and informal mathematics, which needs no such knowledge (see section 3.2.2).

In the rest of this introduction I outline the approach and mention related work. In following sections I discuss what we expect from a proof, add details to the approach, pointing out problems that arise, and concentrate on what I believe is the primary technical problem: expressiveness and feasibility for checking of formal systems and representations of mathematical notions.

1.1 Outline of the approach

The problem is how to believe FLT when given only a putative proof formalised in a given logic. Assume it is a logic that you believe is consistent, and appropriate for FLT. The "thing" I give you is some computer files. There may be questions about the physical and abstract representations of the files (how to physically

[1] A version of this paper appears in Sambin and Smith (editors) *Twenty Five Years of Constructive Type Theory*, Oxford University Press.

[2] Basic Research in Computer Science, Centre of the Danish National Research Foundation. The author also thanks Edinburgh University and Chalmers University.

Randy Pollack and how to believe a machine-checked proof

How to Believe a Machine-Checked Proof[1]

Robert Pollack
BRICS,[2] *Computer Science Dept., Aarhus University*
DK-8000 Aarhus C, Denmark

1 Introduction

Suppose I say "Here is a machine-checked proof of Fermat's last theorem (FLT)". How can you use my putative machine-checked proof as evidence for belief in FLT? I start from the position that you must have some personal experience of understanding to attain belief, and to have this experience you must engage your intuition and other mental processes which are impossible to formalise.

By machine-checked proof I mean a formal derivation in some given formal system; I am talking about derivability, not about truth. Further, I want to talk about *actually* believing an *actual* formal proof, not about formal proofs in principle; to be interesting, any approach to this problem must be feasible. You might try to read my proof, just as you would a proof in a journal; however, with the current state of the art, this proof will surely be too long for you to have confidence that you have understood it. This paper presents a technological approach for reducing the problem of believing a formal proof to the same psychological and philosophical issues as believing a conventional proof in a mathematics journal. The approach is not entirely successful philosophically as there seems to be a fundamental difference between machine checked mathematics, which depends on empirical knowledge about the physical world, and informal mathematics, which needs no such knowledge (see section 3.2.2).

In the rest of this introduction I outline the approach and mention related work. In following sections I discuss what we expect from a proof, add details to the approach, pointing out problems that arise, and concentrate on what I believe is the primary technical problem: expressiveness and feasibility for checking of formal systems and representations of mathematical notions.

1.1 Outline of the approach

The problem is how to believe FLT when given only a putative proof formalised in a given logic. Assume it is a logic that you believe is consistent, and appropriate for FLT. The "thing" I give you is some computer files. There may be questions about the physical and abstract representations of the files (how to physically

[1] A version of this paper appears in Sambin and Smith (editors) *Twenty Five Years of Constructive Type Theory*, Oxford University Press.

[2] Basic Research in Computer Science, Centre of the Danish National Research Foundation. The author also thanks Edinburgh University and Chalmers University.

formal proof versus correctness

the computer has checked a formalization without finding errors

formal proof versus correctness

the computer has checked a formalization without finding errors
is it now certain that there are no errors?

formal proof versus correctness

the computer has checked a formalization without finding errors
is it now certain that there are no errors?

- **philosophical issues**

  'certainty without any doubt is impossible'

formal proof versus correctness

the computer has checked a formalization without finding errors
is it now certain that there are no errors?

- **philosophical issues**

  'certainty without any doubt is impossible'

- **software issues**

  'all programs have bugs'

formal proof versus correctness

the computer has checked a formalization without finding errors
is it now certain that there are no errors?

- **philosophical issues**

  'certainty without any doubt is impossible'

- **software issues**

  'all programs have bugs'

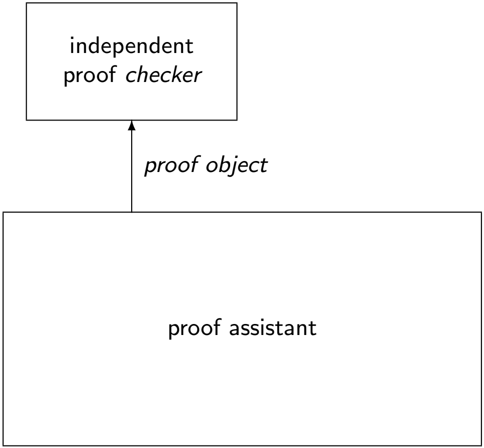
- **the real problem**

  certain that the proofs are correct
  not certain that the definitions are 'correct'

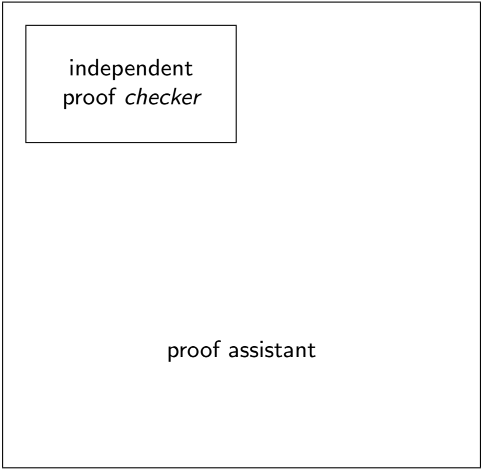
the de Bruijn criterion

proof assistant

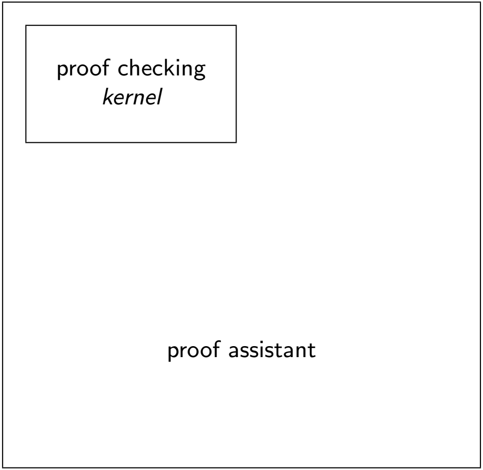
the de Bruijn criterion



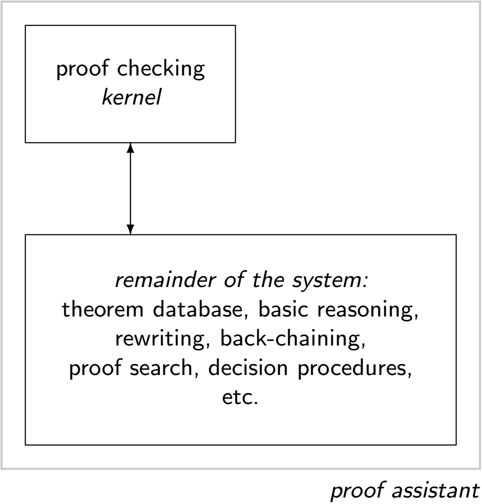
the de Bruijn criterion



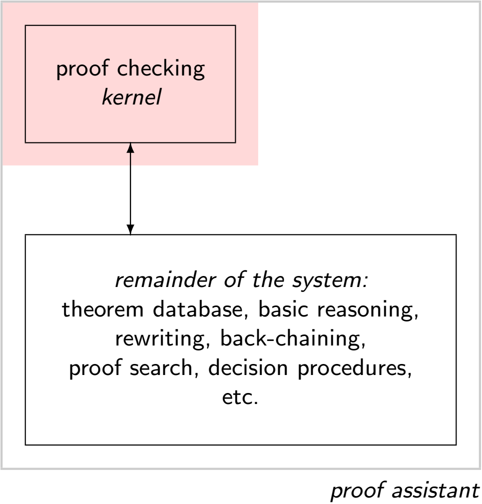
the de Bruijn criterion



the de Bruijn criterion



the de Bruijn criterion



a student's remark

a proof assistant for propositional logic

course **proof assistants** at Radboud University Nijmegen

a proof assistant for propositional logic

course **proof assistants** at Radboud University Nijmegen

- implement small proof assistant

  LCF-style proof assistant
  minimal propositional logic (= only implication)

$$\frac{}{\Gamma \cup \{A\} \vdash A} \qquad \frac{\Gamma \cup \{A\} \vdash B}{\Gamma \vdash A \to B} \qquad \frac{\Gamma \vdash A \to B \quad \Gamma \vdash A}{\Gamma \vdash B}$$

a proof assistant for propositional logic

course **proof assistants** at Radboud University Nijmegen

- implement small proof assistant

  LCF-style proof assistant
  minimal propositional logic (= only implication)

$$\frac{}{\{A\} \vdash A} \qquad \frac{\Gamma \vdash B}{\Gamma - \{A\} \vdash A \to B} \qquad \frac{\Gamma \vdash A \to B \quad \Delta \vdash A}{\Gamma \cup \Delta \vdash B}$$

a proof assistant for propositional logic

course **proof assistants** at Radboud University Nijmegen

- implement small proof assistant

  LCF-style proof assistant
  minimal propositional logic (= only implication)

$$\frac{}{\{A\} \vdash A} \qquad \frac{\Gamma \vdash B}{\Gamma - \{A\} \vdash A \to B} \qquad \frac{\Gamma \vdash A \to B \quad \Delta \vdash A}{\Gamma \cup \Delta \vdash B}$$

student **Marc Schoolderman**:

- let's add the other propositional connectives

a proof assistant for propositional logic

course **proof assistants** at Radboud University Nijmegen

- implement small proof assistant

  LCF-style proof assistant
  minimal propositional logic (= only implication)

$$\frac{}{\{A\} \vdash A} \qquad \frac{\Gamma \vdash B}{\Gamma - \{A\} \vdash A \to B} \qquad \frac{\Gamma \vdash A \to B \quad \Delta \vdash A}{\Gamma \cup \Delta \vdash B}$$

student **Marc Schoolderman**:

- let's add the other propositional connectives ...
  ... *in the parser and pretty-printer!*

a proof assistant for propositional logic

course **proof assistants** at Radboud University Nijmegen

- implement small proof assistant

  LCF-style proof assistant
  minimal propositional logic (= only implication)

$$\frac{}{\{A\} \vdash A} \qquad \frac{\Gamma \vdash B}{\Gamma - \{A\} \vdash A \rightarrow B} \qquad \frac{\Gamma \vdash A \rightarrow B \quad \Delta \vdash A}{\Gamma \cup \Delta \vdash B}$$

student **Marc Schoolderman**:

- let's add the other propositional connectives . . .
  . . . *in the parser and pretty-printer!*

  parser and pretty-printer have to know about logic

the digits of hundred factorial

the kernel of HOL Light

proof assistant that best satisfies the de Bruijn criterion:

HOL Light, John Harrison, 1998–*today*

the kernel of HOL Light

proof assistant that best satisfies the de Bruijn criterion:

HOL Light, John Harrison, 1998–*today*

proof checking kernel = `fusion.ml`

671 lines $\approx$ 10 printed pages $\approx$ 0.2 % of the system
only 395 lines of actual code

the kernel of HOL Light

proof assistant that best satisfies the de Bruijn criterion:

HOL Light, John Harrison, 1998–*today*

proof checking kernel = `fusion.ml`

671 lines $\approx$ 10 printed pages $\approx$ 0.2 % of the system
only 395 lines of actual code

self-verification of HOL Light: John Harrison, 2006

the kernel of HOL Light

proof assistant that best satisfies the de Bruijn criterion:

HOL Light, John Harrison, 1998–*today*

proof checking kernel = `fusion.ml`

671 lines ≈ 10 printed pages ≈ 0.2 % of the system
only 395 lines of actual code

self-verification of HOL Light: John Harrison, 2006

```
let rec type_of tm =
  match tm with
    Var(_,ty) -> ty
  | Const(_,ty) -> ty
  | Comb(s,_) -> hd(tl(snd(dest_type(type_of s))))
  | Abs(Var(_,ty),t) -> Tyapp("fun",[ty;type_of t])
```

the kernel of HOL Light

proof assistant that best satisfies the de Bruijn criterion:

HOL Light, John Harrison, 1998–*today*

proof checking kernel = `fusion.ml`

671 lines ≈ 10 printed pages ≈ 0.2 % of the system
only 395 lines of actual code

self-verification of HOL Light: John Harrison, 2006

```
let typeof = define
 `(typeof (Var n ty) = ty) /\
  (typeof (Equal ty) = Fun ty (Fun ty Bool)) /\
  (typeof (Select ty) = Fun (Fun ty Bool) ty) /\
  (typeof (Comb s t) = codomain (typeof s)) /\
  (typeof (Abs n ty t) = Fun ty (typeof t))`;;
```

```
#
```

numerical calculations in HOL Light

```
# `1 + 1`;;
```

numerical calculations in HOL Light

```
# `1 + 1`;;
val it : term = `1 + 1`
#
```

numerical calculations in HOL Light

```
# `1 + 1`;;
val it : term = `1 + 1`
# NUM_REDUCE_CONV it;;
val it : thm = |- 1 + 1 = 2
#
```

numerical calculations in HOL Light

```
# `1 + 1`;;
val it : term = `1 + 1`
# NUM_REDUCE_CONV it;;
val it : thm = |- 1 + 1 = 2
# rhs (concl it);;
val it : term = `2`
#
```

numerical calculations in HOL Light

```
# `1 + 1`;;
val it : term = `1 + 1`
# NUM_REDUCE_CONV it;;
val it : thm = |- 1 + 1 = 2
# rhs (concl it);;
val it : term = `2`
# #remove_printer print_qterm;;
# it;;
val it : term =
  Comb (Const ("NUMERAL", `:num->num`),
   Comb (Const ("BIT0", `:num->num`),
    Comb (Const ("BIT1", `:num->num`), Const ("_0", `:num`))))
#
```

numerical calculations in HOL Light

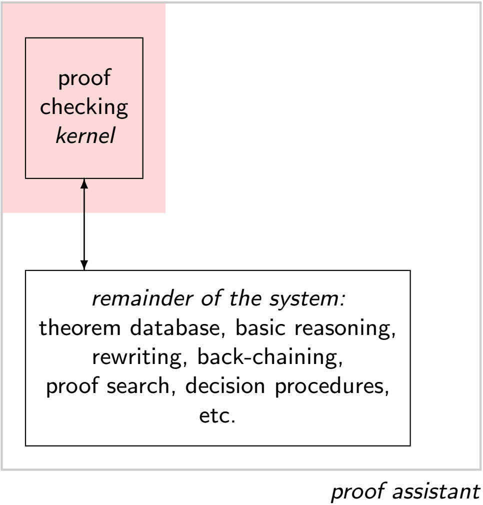
```
# `1 + 1`;;
val it : term = `1 + 1`
# NUM_REDUCE_CONV it;;
val it : thm = |- 1 + 1 = 2
# rhs (concl it);;
val it : term = `2`
# #remove_printer print_qterm;;
# it;;
val it : term =
  Comb (Const ("NUMERAL", `:num->num`),
   Comb (Const ("BIT0", `:num->num`),
    Comb (Const ("BIT1", `:num->num`), Const ("_0", `:num`))))
# #install_printer print_qterm;;
# `NUMERAL (BIT0 (BIT1 _0))`;;
val it : term = `2`
#
```

numerical calculations in HOL Light

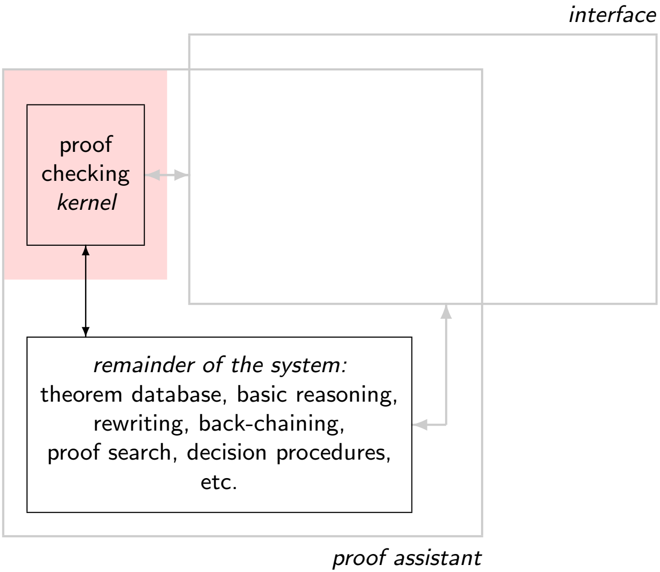
```
# `1 + 1`;;
val it : term = `1 + 1`
# NUM_REDUCE_CONV it;;
val it : thm = |- 1 + 1 = 2
# rhs (concl it);;
val it : term = `2`
# #remove_printer print_qterm;;
# it;;
val it : term =
  Comb (Const ("NUMERAL", `:num->num`),
   Comb (Const ("BIT0", `:num->num`),
    Comb (Const ("BIT1", `:num->num`), Const ("_0", `:num`))))
# #install_printer print_qterm;;
# `NUMERAL (BIT0 (BIT1 _0))`;;
val it : term = `2`

# rhs (concl (NUM_REDUCE_CONV `FACT 100`));;
val it : term =
  `93326215443944152681699238856266700490715968264381621468592963895217
#
```

Pollack-inconsistency

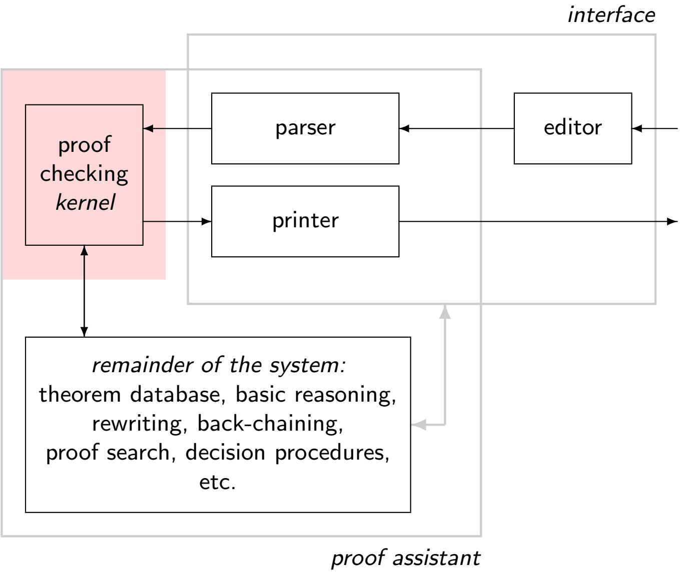
the structure of a proof assistant



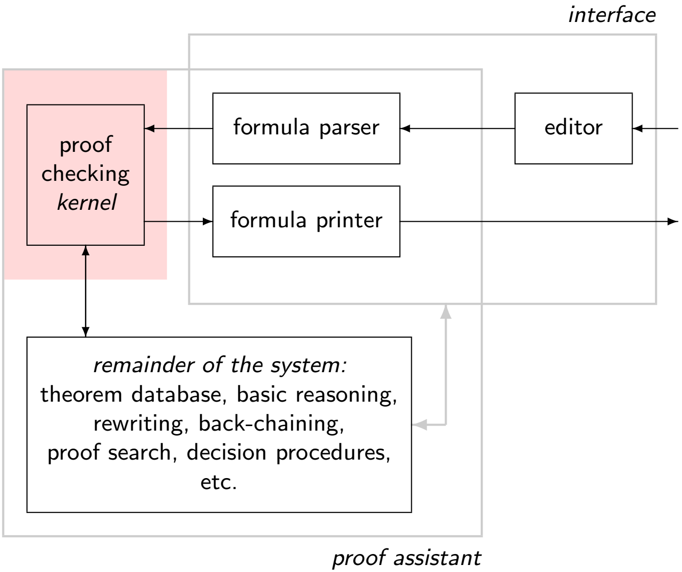
the structure of a proof assistant



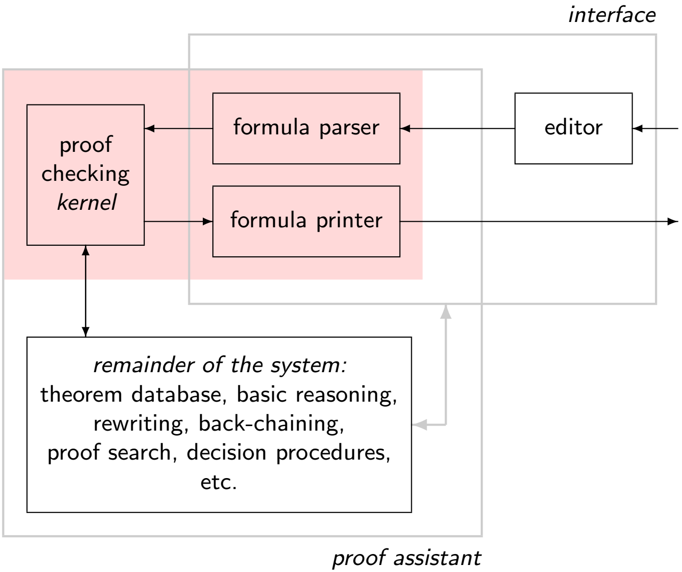
the structure of a proof assistant



the structure of a proof assistant



the structure of a proof assistant



is parsing the left inverse of pretty-printing?

formula parser and pretty-printer:

$$\begin{aligned} \mathsf{parse_f} &: \mathsf{string} \longrightarrow \mathsf{formula} \\ \mathsf{print_f} &: \mathsf{formula} \longrightarrow \mathsf{string} \end{aligned}$$

is parsing the left inverse of pretty-printing?

formula parser and pretty-printer:

$$\begin{aligned} \mathsf{parse}_\mathsf{f} &: \mathsf{string} \longrightarrow \mathsf{formula} \\ \mathsf{print}_\mathsf{f} &: \mathsf{formula} \longrightarrow \mathsf{string} \end{aligned}$$

generally $\mathsf{print}_\mathsf{f}$ is total while $\mathsf{parse}_\mathsf{f}$ is not

is parsing the left inverse of pretty-printing?

formula parser and pretty-printer:

$$\begin{aligned} \mathsf{parse_f} &: \mathsf{string} \longrightarrow \mathsf{formula} \\ \mathsf{print_f} &: \mathsf{formula} \longrightarrow \mathsf{string} \end{aligned}$$

generally $\mathsf{print_f}$ is total while $\mathsf{parse_f}$ is not

'well-behaved': $\mathsf{parse_f}(\mathsf{print_f}(\phi)) = \phi$

is parsing the left inverse of pretty-printing?

formula parser and pretty-printer:

$$\begin{aligned} \mathsf{parse_f} &: \mathsf{string} \longrightarrow \mathsf{formula} \\ \mathsf{print_f} &: \mathsf{formula} \longrightarrow \mathsf{string} \end{aligned}$$

generally $\mathsf{print_f}$ is total while $\mathsf{parse_f}$ is not

'well-behaved': $\mathsf{parse_f}(\mathsf{print_f}(\phi)) = \phi$

in practice well-behavedness occasionally breaks

is parsing the left inverse of pretty-printing?

formula parser and pretty-printer:

$$\begin{aligned} \mathsf{parse_f} &: \mathsf{string} \to \mathsf{formula} \\ \mathsf{print_f} &: \mathsf{formula} \to \mathsf{string} \end{aligned}$$

generally $\mathsf{print_f}$ is total while $\mathsf{parse_f}$ is not

'well-behaved': $\mathsf{parse_f}(\mathsf{print_f}(\phi)) = \phi$

in practice well-behavedness occasionally breaks

$$\mathsf{print_f}(\mathsf{parse_f}(s)) \neq s$$

$\mathsf{parse_f}(\texttt{"1 + 1 = 2"}) = \mathsf{parse_f}(\texttt{"1+1=2"}) = \mathsf{parse_f}(\texttt{"(1 + 1) = 2"})$

Pollack-axioms and Pollack-inconsistency

Pollack-inconsistency:

'$\perp$ is provable from Pollack-axioms'

Pollack-axioms and Pollack-inconsistency

Pollack-inconsistency:

'$\perp$ is provable from Pollack-axioms'

Pollack-axioms:

$$\text{`}\phi_1 \Leftrightarrow \phi_2\text{'} \quad \text{when} \quad \mathsf{print}_\mathsf{f}(\phi_1) = \mathsf{print}_\mathsf{f}(\phi_2)$$

Pollack-axioms and Pollack-inconsistency

Pollack-inconsistency:

'$\perp$ is provable from Pollack-axioms'

Pollack-axioms:

$$\begin{array}{lcl} `\phi_1 \Leftrightarrow \phi_2\text{'} & \text{when} & \mathsf{print}_\mathsf{f}(\phi_1) = \mathsf{print}_\mathsf{f}(\phi_2) \\ `t_1 = t_2\text{'} & \text{when} & \mathsf{print}_\mathsf{t}(t_1) = \mathsf{print}_\mathsf{t}(t_2) \end{array}$$

Pollack-axioms and Pollack-inconsistency

Pollack-inconsistency:

'$\perp$ is provable from Pollack-axioms'

Pollack-axioms:

$$
\begin{array}{ccc}
`\phi_1 \Leftrightarrow \phi_2' & \text{when} & \mathsf{print}_{\mathsf{f}}(\phi_1) = \mathsf{print}_{\mathsf{f}}(\phi_2) \\
`t_1 = t_2' & \text{when} & \mathsf{print}_{\mathsf{t}}(t_1) = \mathsf{print}_{\mathsf{t}}(t_2)
\end{array}
$$

- default printer with default settings
- default equality

  Coq: 'Leibniz equality'
- only t_1 and t_2 for which $t_1 = t_2$ is well-typed

weak versus strong Pollack-inconsistency

strong Pollack-inconsistency =

. . . without adding definitions

weak Pollack-inconsistency =

. . . extra definitions allowed

- . . . on top of the standard library of the system
- only **conservative** definitions

  same provable formulas not involving the new definition
- notations considered a form of definition

  Coq: coercions

Pollack-super-inconsistency

Pollack-super-inconsistency =

$$\phi \text{ is provable} \quad \text{with} \quad \mathsf{print}_{\mathsf{f}}(\phi) = \mathsf{print}_{\mathsf{f}}(\bot)$$

where

$$\bot = \left\{ \begin{array}{ll} \texttt{F} & \text{HOL} \\ \texttt{False} & \text{Isabelle} \\ \texttt{False} & \text{Coq} \\ \texttt{contradiction} & \text{Mizar} \end{array} \right\}$$

Pollack-super-inconsistency

Pollack-super-inconsistency =

$$\phi \text{ is provable} \quad \text{with} \quad \mathsf{print}_\mathsf{f}(\phi) = \mathsf{print}_\mathsf{f}(\perp)$$

where

$$\perp = \left\{ \begin{array}{ll} \texttt{F} & \text{HOL} \\ \texttt{False} & \text{Isabelle} \\ \texttt{False} & \text{Coq} \\ \texttt{contradiction} & \text{Mizar} \end{array} \right\}$$

not only does the system appear inconsistent
it even looks like one has proved a trivial false in the system

some of the best proof assistants are Pollack-inconsistent

HOL Light

```
#
```

HOL Light (and Isabelle)

#

HOL Light (and Isabelle)

```
# `?!x:1. T`;;
```

HOL Light (and Isabelle)

```
# `?!x:1. T`;;
val it : term = `?!x. T`
#
```

HOL Light (and Isabelle)

```
# `?!x:1. T`;;
val it : term = `?!x. T`
# `?!x:bool. T`;;
val it : term = `?!x. T`
#
```

HOL Light (and Isabelle)

```
# `?!x:1. T`;;
val it : term = `?!x. T`
# `?!x:bool. T`;;
val it : term = `?!x. T`

# mk_eq(mk_var("0",`:1`),mk_var("1",`:1`));;
```

HOL Light (and Isabelle)

```
# `?!x:1. T`;;
val it : term = `?!x. T`
# `?!x:bool. T`;;
val it : term = `?!x. T`

# mk_eq(mk_var("0",`:1`),mk_var("1",`:1`));;
val it : term = `0 = 1`
#
```

HOL Light (and Isabelle)

```
# `?!x:1. T`;;
val it : term = `?!x. T`
# `?!x:bool. T`;;
val it : term = `?!x. T`

# mk_eq(mk_var("0",`:1`),mk_var("1",`:1`));;
val it : term = `0 = 1`
# prove(it, ONCE_REWRITE_TAC[one] THEN REFL_TAC);;
val it : thm = |- 0 = 1
#
```

HOL Light (and Isabelle)

```
# `?!x:1. T`;;
val it : term = `?!x. T`
# `?!x:bool. T`;;
val it : term = `?!x. T`

# mk_eq(mk_var("0",`:1`),mk_var("1",`:1`));;
val it : term = `0 = 1`
# prove(it, ONCE_REWRITE_TAC[one] THEN REFL_TAC);;
val it : thm = |- 0 = 1

# override_interface("F",`T`);;
val it : unit = ()
#
```

HOL Light (and Isabelle)

```
# `?!x:1. T`;;
val it : term = `?!x. T`
# `?!x:bool. T`;;
val it : term = `?!x. T`

# mk_eq(mk_var("0",`:1`),mk_var("1",`:1`));;
val it : term = `0 = 1`
# prove(it, ONCE_REWRITE_TAC[one] THEN REFL_TAC);;
val it : thm = |- 0 = 1

# override_interface("F",`T`);;
val it : unit = ()
# mk_const("F",[]);;
val it : term = `F`
# `T`;;
val it : term = `F`
#
```

HOL Light (and Isabelle)

```
# `?!x:1. T`;;
val it : term = `?!x. T`
# `?!x:bool. T`;;
val it : term = `?!x. T`

# mk_eq(mk_var("0",`:1`),mk_var("1",`:1`));;
val it : term = `0 = 1`
# prove(it, ONCE_REWRITE_TAC[one] THEN REFL_TAC);;
val it : thm = |- 0 = 1

# override_interface("F",`T`);;
val it : unit = ()
# mk_const("F",[]);;
val it : term = `F`
# `T`;;
val it : term = `F`
# prove(`F`, ACCEPT_TAC TRUTH);;
val it : thm = |- F
#
```

Coq

```
Coq &lt;
```

Coq


```
Coq < Coercion S : nat >-> nat.
S is now a coercion

Coq <
```

Coq

```
Coq < Coercion S : nat >-> nat.
S is now a coercion

Coq < Check 0.
0
     : nat

Coq <
```

Coq

```
Coq < Coercion S : nat >-> nat.
S is now a coercion

Coq < Check 0.
0
     : nat

Coq < Check 1.
0
     : nat

Coq <
```

Coq

```
Coq < Coercion S : nat >-> nat.
S is now a coercion

Coq < Check 0.
0
     : nat

Coq < Check 1.
0
     : nat

Coq < Definition _Prop := Prop.
_Prop is defined

Coq < Definition _not : _Prop -> Prop := not.
_not is defined

Coq < Coercion _not : _Prop >-> Sortclass.
_not is now a coercion

Coq <
```

Coq (continued)

```
Coq < Coercion _not : _Prop >-> Sortclass.
_not is now a coercion

Coq < Lemma _I : _not False.
1 subgoal

  ============================
   False

_I <
```

Coq (continued)

```
Coq < Coercion _not : _Prop >-> Sortclass.
_not is now a coercion

Coq < Lemma _I : _not False.
1 subgoal

  ============================
   False

_I < exact (fun x => x).
Proof completed.

_I < Qed.
exact (fun x => x).

_I is defined

Coq <
```

Coq (continued)

```
Coq < Coercion _not : _Prop >-> Sortclass.
_not is now a coercion

Coq < Lemma _I : _not False.
1 subgoal

  ============================
   False

_I < exact (fun x => x).
Proof completed.

_I < Qed.
exact (fun x => x).

_I is defined

Coq < Check _I.
_I
     : False

Coq <
```

Mizar

```
definition let x be real number;
  func [x] equals 1; coherence;
end;

definition let x be natural number;
  func [x] equals 0; coherence;
end;

theorem [0] <> [0 qua real number];
```

Mizar

```
definition let x be real number;
  func [x] equals 1; coherence;
end;

definition let x be natural number;
  func [x] equals 0; coherence;
end;

theorem [0] <> [0 qua real number];
```

Mizar

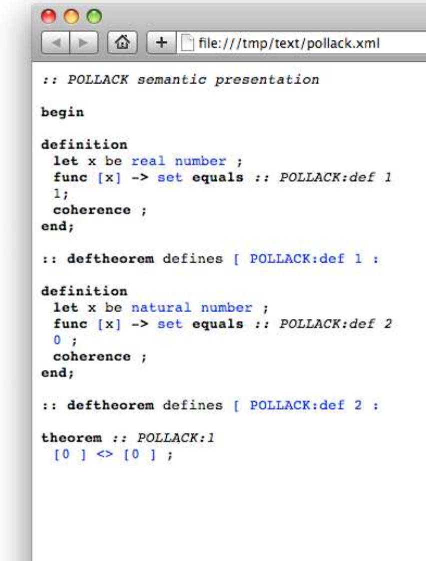
```
definition let x be real number;
  func [x] equals 1; coherence;
end;

definition let x be natural number;
  func [x] equals 0; coherence;
end;

theorem [0] <> [0 qua real number];
```

Mizar

```
definition let x be real number;
  func [x] equals 1; coherence;
end;

definition let x be natural number;
  func [x] equals 0; coherence;
end;

theorem [0] <> [0 qua real number];
```

Mizar

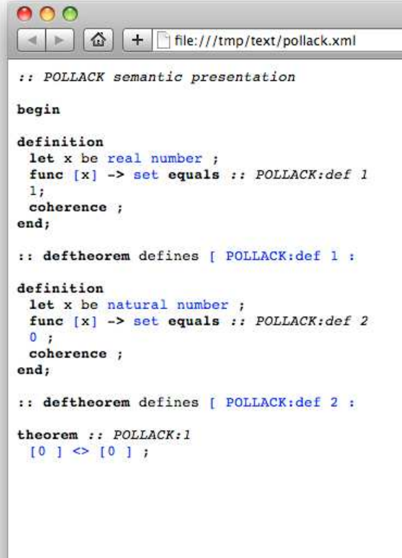
```
definition let x be real number;
  func [x] equals 1; coherence;
end;

definition let x be natural number;
  func [x] equals 0; coherence;
end;

theorem [0] <> [0 qua real number];
```



Mizar

```
definition let x be real number;
  func [x] equals 1; coherence;
end;

definition let x be natural number;
  func [x] equals 0; coherence;
end;

theorem [0] <> [0 qua real number];
```

```
theorem :: POLLACK:1
  [0 ] <> [0 ] ;
```



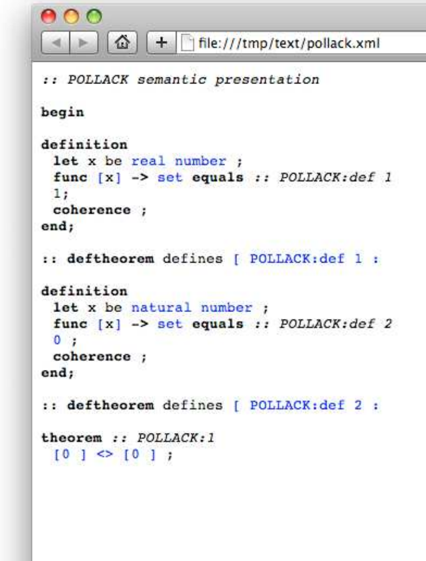
file:///tmp/text/pollack.xml

```
:: POLLACK semantic presentation

begin

definition
 let x be real number ;
 func [x] -> set equals :: POLLACK:def 1
 1;
 coherence ;
end;

:: deftheorem defines [ POLLACK:def 1 :

definition
 let x be natural number ;
 func [x] -> set equals :: POLLACK:def 2
 0 ;
 coherence ;
end;

:: deftheorem defines [ POLLACK:def 2 :

theorem :: POLLACK:1
 [0 ] <> [0 ] ;
```

Mizar

```
definition let x be real number;
  func [x] equals 1; coherence;
end;

definition let x be natural number;
  func [x] equals 0; coherence;
end;

theorem [0] <> [0 qua real number];
```

```
theorem :: POLLACK:1
  [0 ] <> [0 ] ;
```



```
file:///tmp/text/pollack.xml

:: POLLACK semantic presentation

begin

definition
 let x be real number ;
 func [x] -> set equals :: POLLACK:def 1
 1;
 coherence ;
end;

:: deftheorem defines [ POLLACK:def 1 :

definition
 let x be natural number ;
 func [x] -> set equals :: POLLACK:def 2
 0 ;
 coherence ;
end;

:: deftheorem defines [ POLLACK:def 2 :

theorem :: POLLACK:1
 [0 ] <> [0 ] ;
```

Pollack-consistency on the cheap

checking the pretty-printer at runtime

'hack' for making it easier to prove Pollack-consistency

checking the pretty-printer at runtime

'hack' for making it easier to prove Pollack-consistency

- $\mathrm{print}_{\mathrm{normal}}$

  good, complicated

- $\mathrm{print}_{\mathrm{failsafe}}$

  failsafe, trivial

checking the pretty-printer at runtime

'hack' for making it easier to prove Pollack-consistency

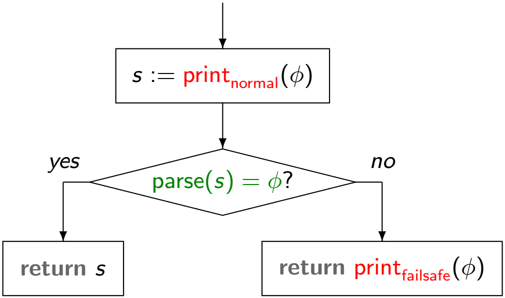
- $\text{print}_{\text{normal}}$

  good, complicated

- $\text{print}_{\text{failsafe}}$

  failsafe, trivial

combine into $\text{print}_{\text{combined}}$:



checking the pretty-printer at runtime

'hack' for making it easier to prove Pollack-consistency

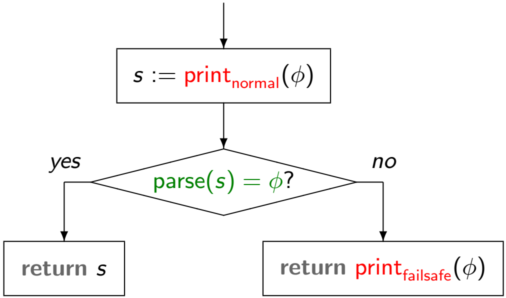
- $\text{print}_{\text{normal}}$

  good, complicated

- $\text{print}_{\text{failsafe}}$

  failsafe, trivial

combine into $\text{print}_{\text{combined}}$:



$$\text{parse}(\text{print}_{\text{failsafe}}(\phi)) = \phi$$
$$\Downarrow$$
$$\text{parse}(\text{print}_{\text{combined}}(\phi)) = \phi$$

does Pollack-inconsistency matter?

the attitude of the proof assistants community

computer algebra users

a little inconsistency does not matter much . . .

the attitude of the proof assistants community

computer algebra users

a little inconsistency does not matter much ...

```
> 1/(1-x) = simplify(1/(1-x))
```

$$\frac{1}{1-x} = -\frac{1}{-1+x}$$

computer algebra users

a little inconsistency does not matter much . . .

```
> int(1/(1-x),x) = int(simplify(1/(1-x)),x)
```

$$-\ln(1-x) = -\ln(-1+x)$$

the attitude of the proof assistants community

computer algebra users

a little inconsistency does not matter much . . .

```
> int(1/(1-x),x) = int(simplify(1/(1-x)),x)
```

$$-\ln(1-x) = -\ln(-1+x)$$

```
> evalf(subs(x=-1, %));
```

$$-0.6931471806 = -0.6931471806 - 3.141592654i$$

the attitude of the proof assistants community

computer algebra users

a little inconsistency does not matter much ...

```
> int(1/(1-x),x) = int(simplify(1/(1-x)),x)
```

$$-\ln(1-x) = -\ln(-1+x)$$

```
> evalf(subs(x=-1, %));
```

$$-0.6931471806 = -0.6931471806 - 3.141592654i$$

proof assistant users

consistency is very important!

the attitude of the proof assistants community

computer algebra users

a little inconsistency does not matter much . . .

```
> int(1/(1-x),x) = int(simplify(1/(1-x)),x)
```

$$-\ln(1-x) = -\ln(-1+x)$$

```
> evalf(subs(x=-1, %));
```

$$-0.6931471806 = -0.6931471806 - 3.141592654i$$

proof assistant users

consistency is very important!
a little Pollack-inconsistency does not matter much . . .

the attitude of the proof assistants community

computer algebra users

a little inconsistency does not matter much . . .

```
> int(1/(1-x),x) = int(simplify(1/(1-x)),x)
```

$$-\ln(1-x) = -\ln(-1+x)$$

```
> evalf(subs(x=-1, %));
```

$$-0.6931471806 = -0.6931471806 - 3.141592654i$$

proof assistant users

apart from principled people like Randy Pollack and Mark Adams

consistency is very important!

a little Pollack-inconsistency does not matter much . . .