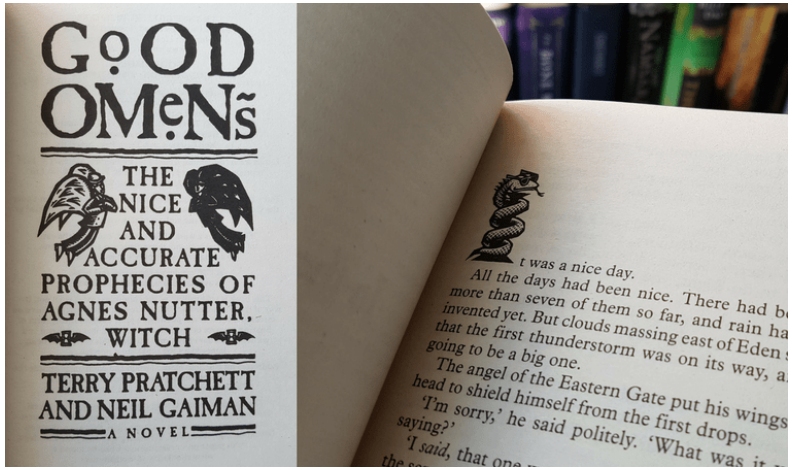
yet another nice and accurate interactive theorem prover

(sufficiently similar, sufficiently different)

Freek Wiedijk

Radboud University Nijmegen

2018-06-25



nice

combining the nicest aspects of state-of-the-art provers:

- document centric (Isabelle/Mizar)
- small (stateless) LCF kernel (HOL/Isabelle/Coq)
- saving state per source file (Coq)
- first order logic with schemes (Mizar)
- ZFC set theory (Mizar)
- (soft) dependent types (Mizar/Coq)
- subtyping, intersection types (Mizar)
- declarative (structured) proofs (Mizar/Isar)
- procedural proofs (SSReflect/HOL)
- declarative interface (Isabelle/Mizar)
- user proof automation (HOL/Coq)
- proper treatment of partiality (PVS/B)

accurate

- accurate for mathematics
  - classical mathematics
  - full strength of set theory
  - declarative proofs
  - first class binders ($\sum$, $\frac{d}{dx}$, $\int$, $\lim$, etc)
  - proper treatment of partiality
- accurate for computer science
  - implemented/programmable in a functional language (ML)
  - mathematical language contains a functional language
  - algebraic datatypes
  - recursive functions (with general recursion)
  - inductively defined predicates/relations
  - Poincaré principle = computations do not need proofs

three interfaces

- ML read-eval-print loop (HOL)
- batch compiler (Coq/Mizar)
- web based IDE (Isabelle-style, but in web browser)
  - light weight
  - text only, just a few panes

three interfaces

- ML read-eval-print loop (HOL)
- batch compiler (Coq/Mizar)
- web based IDE (Isabelle-style, but in web browser)
  - light weight
  - text only, just a few panes
  - later maybe: diagrams corresponding to proof state
  - later maybe: proof display as formal proof sketches
  - later maybe: cross referencing with informal math

some further plans

- first class version management
- integrated social network (global name space)
- Pollack-consistent (= parsing is left inverse of printing)
- grow declarative proofs using procedural steps (miz3)
- automatically convert procedural proofs to declarative (miz3)
- computer algebra using filter calculus (FEAR)
- `thm` keeps track of axioms and lamps used
- the only state in the implementation: global `ref` variables
- only plain text (outside 7-bit ASCII = syntax error)

target trial formalizations

- the real numbers are a field
  - Bert Jutting (Automath)
  - John Harrison (HOL)
  - Milad Niqui (Coq)
  - Russell O'Connor (Coq)
  - Georges Gonthier (Coq)

    etc
- big step semantics $\longleftrightarrow$ small step semantics
  of small while language (IMP)
  - Tobias Nipkow, Gerwin Klein: concrete semantics (Isabelle)
  - Benjamin Pierce: software foundations (Coq)

    etc

two kernels

- **inner kernel**
  - untyped
  - total
  - de Bruijn indices
  - no contexts
  - small
  - abstract datatypes protected by a module
  - ‘magic lamps’
- **outer kernel**
  - typed
  - partial
  - named variables
  - contexts with typed variables
  - large
  - data creation not protected by an abstract interface

kernel data structures isomorphic apart from ‘tags’

the logic

$$\frac{\neg A \in \Delta \quad A \in \Delta}{\vdash \Delta}$$

$$\frac{\neg\neg A \in \Delta \quad \vdash \Delta, A}{\vdash \Delta}$$

$$\frac{\forall \vec{x}\, \mathbb{W}\vec{A} \in \Delta \quad \vec{x} \notin \mathsf{FV}(\Delta) \quad \vdash \Delta, \vec{A}}{\vdash \Delta}$$

$$\frac{\neg\, \forall \vec{x}\, \mathbb{W}\vec{A} \in \Delta \quad \cdots \quad \vdash \Delta, \neg A_i[\vec{x} := \vec{t}] \quad \cdots}{\vdash \Delta}$$

$$\frac{t = t \in \Delta}{\vdash \Delta}$$

$$\frac{\neg(t = t') \in \Delta \quad \vdash \Delta, A[x := t] \quad \vdash \Delta, \neg A[x := t']}{\vdash \Delta}$$

$\ldots$ + instantiation + scheme instantiation + cut rule

soft types

typing is theorem proving
slowest theorem prover ever

built into the logic/kernel:

	Coq	HOL	here
types	+	+	−
(co) inductive definitions	+	−	−

soft types imply proper treatment of partiality

choice: no predicate subtypes
(I want type checking to be non-interactive)

soft types imply proper treatment of partiality

choice: no predicate subtypes
(I want type checking to be non-interactive)

```
type_of (term "x") = type "real"
type_of (term "1/x") = type "real"
```

soft types imply proper treatment of partiality

choice: no predicate subtypes
(I want type checking to be non-interactive)

```
type_of (term "x") = type "real"
type_of (term "1/x") = type "real"
```

```
real x |- real x
real x |- real (1/x)
```

soft types imply proper treatment of partiality

choice: no predicate subtypes
(I want type checking to be non-interactive)

```
type_of (term "x") = type "real"
type_of (term "1/x") = type "real"
```

```
real x |- real x
real x |- real (1/x)
```

`1/0` is defined (inner kernel is total) but why should it be `real`?

soft types imply proper treatment of partiality

choice: no predicate subtypes
(I want type checking to be non-interactive)

```
type_of (term "x") = type "real"
type_of (term "1/x") = type "real"
```

```
real x |- real x
real x |- real (1/x)
```

`1/0` is defined (inner kernel is total) but why should it be `real`?
types do not even need to be inhabited outside the domain
(I want dependent types to have the possibility to be empty)

soft types imply proper treatment of partiality

choice: no predicate subtypes
(I want type checking to be non-interactive)

```
type_of (term "x") = type "real"
type_of (term "1/x") = type "real"
```

```
real x |- real x
real x |- x != 0 => real (1/x)
```

1/0 is defined (inner kernel is total) but why should it be `real`?
types do not even need to be inhabited outside the domain
(I want dependent types to have the possibility to be empty)

soft types imply proper treatment of partiality

choice: no predicate subtypes
(I want type checking to be non-interactive)

```
type_of (term "x") = type "real"
type_of (term "1/x") = type "real"

cc_of (term "x") = []
cc_of (term "1/x") = [form "x != 0"]

real x |- real x
real x |- x != 0 => real (1/x)
```

`1/0` is defined (inner kernel is total) but why should it be `real`?
types do not even need to be inhabited outside the domain
(I want dependent types to have the possibility to be empty)

every term has a type and a list of correctness conditions

soft types imply proper treatment of partiality

choice: no predicate subtypes
(I want type checking to be non-interactive)

```
type_of (term "x") = type "real"
type_of (term "1/x") = type "real"

cc_of (term "x") = []
cc_of (term "1/x") = [form "x != 0"]

real x |- real x
real x |- x != 0 => real (1/x)
```

`1/0` is defined (inner kernel is total) but why should it be `real`?
types do not even need to be inhabited outside the domain
(I want dependent types to have the possibility to be empty)

every term has a type and a list of correctness conditions

having to deal with partiality is a natural consequence of soft typing
(. . . but the logic is total!)

similar projects



- **Cezary Kaliszyk**: Mizar-like system on top of Isabelle
  - still sufficiently different?
  - heavier
  - partiality?
- **John Harrison**: let's make set theory great again!
  - no soft types?
- **Harvey Friedman**
  - not an implementer
  - free (partial) logic is weird: $\vdash \exists x.\, \neg(\frac{1}{x} = 0) \wedge \neg(\frac{1}{x} \neq 0)$?
- several other ZFC based systems

what do I have already?

- not much yet

what do I have already?

- not much yet
- plans

what do I have already?

- not much yet
- plans
- several false starts

what do I have already?

- not much yet
- plans
- several false starts

- implementation of the inner kernel (currently 266 $<$ 448 lines)
- basic version of typed terms
- first next step: simple parser/printer (both inner/outer kernels)

what do I have already?

- not much yet
- plans
- several false starts

- implementation of the inner kernel (currently 266 $<$ 448 lines)
- basic version of typed terms
- first next step: simple parser/printer (both inner/outer kernels)
  - recursive descent parser
    (CakeML has no parser generator)
  - Pollack-consistent?

    ```
    x + 0
    #[+].1# real x (#0# real)
    ```

questions?

formalization, like sex, and like innovation, is one of those things where the people who talk about it the most tend to be those who do it the least — *John Harrison*