Formale Mathematik

AG 1 des Natur- und Ingenieurwiss. Kolleg VI

Bad Honnef, 21. März 2016

Peter Koepke, University of Bonn, Germany

Mathematical formalism

(M. Magidor: *How large is the first strongly compact ...*)

F is well-defined in $V[G \restriction \beta]$ because the first two cases in the definition are exclusive. If both cases hold, we get $Q_1, Q_2 \in G \restriction \beta, B_{\gamma,1}, B_{\gamma,2}$ for $\gamma \in E - \beta$ such that if we define

$$S_1 = Q_1 \cup \{\langle p_\beta \frown \langle \delta_1, \ldots, \delta_n \rangle, B_{\beta,1} \rangle\} \cup \{\langle p_\gamma, B_{\gamma,1} \rangle\}_{\gamma \in E-(\beta+1)},$$

$$S_2 = Q_2 \cup \{\langle p_\beta \frown \langle \delta_1, \ldots, \delta_n \rangle, B_{\beta,2} \rangle\} \cup \{\langle p_\gamma, B_{\gamma,2} \rangle\}_{\gamma \in E-(\beta+1)},$$

then $S_1 \Vdash \Phi$ and $S_2 \Vdash \neg\Phi$. Let Q be a common extension of Q_1 and Q_2 in $\mathcal{P}_\beta$ which exists since Q_1 and Q_2 are members of the same $\mathcal{P}_\beta$ generic filter. $B_{\gamma,1}$ and $B_{\gamma,2}$ for $\gamma \in E - \beta$ are terms appropriate for $\mathcal{P}_\gamma$, which are forced by every member of $\mathcal{P}_\gamma$ to be in $\widetilde{U}_\gamma$, which is forced to be an ultrafilter on γ. Hence if D_γ is the term which canonically denotes the intersection of $B_{\gamma,1}$ and $B_{\gamma,2}$, then D_γ is forced by every condition to be in $\widetilde{U}_\gamma$. Define

$$S = Q \cup \{\langle p_\beta \frown \langle \delta_1, \ldots, \delta_n \rangle, D_\beta \rangle\} \cup \{\langle p_\gamma, D_\gamma \rangle\}_{\gamma \in E-(\beta+1)}.$$

S is clearly in $\mathcal{P}_\alpha$ and a common extension of S_1 and S_2, hence $S \Vdash \Phi$ and $S \Vdash \neg\Phi$, we hence derive a contradiction and F is well-defined partition.

Can mathematics be fully formalized?

A.N.Whitehead, B.Russell, *Principia Mathematica*:

∗54·56. $\vdash : \alpha \sim\in 0 \cup 1 \cup 2 \,.\equiv.\, (\exists x, y, z) \,.\, x, y, z \in \alpha \,.\, x \neq y \,.\, x \neq z \,.\, y \neq z$

Dem.

$\vdash \,.\, {*}54{\cdot}55 \,.\, {*}11{\cdot}52 \,.\, \supset$

$\vdash :. \alpha \sim\in 0 \cup 1 \cup 2 \,.\equiv:\, (\exists x, y) \,.\, x, y \in \alpha \,.\, x \neq y \,.\, \alpha \neq \iota`x \cup \iota`y :$

$[{*}51{\cdot}2.{*}22{\cdot}59] \quad \equiv : (\exists x, y) \,.\, \iota`x \cup \iota`y \subset \alpha \,.\, x \neq y \,.\, \alpha \neq \iota`x \cup \iota`y :$

$[{*}24{\cdot}6] \quad \equiv : (\exists x, y) \,.\, \iota`x \cup \iota`y \subset \alpha \,.\, x \neq y \,.\, \exists ! \, \alpha - (\iota`x \cup \iota`y) :$

$[{*}51{\cdot}232.\mathrm{Transp}] \quad \equiv : (\exists x, y) : \iota`x \cup \iota`y \subset \alpha \,.\, x \neq y : (\exists z) \,.\, z \in \alpha \,.\, z \neq x \,.\, z \neq y :$

$[{*}51{\cdot}2.{*}22{\cdot}59] \quad \equiv : (\exists x, y, z) \,.\, x, y, z \in \alpha \,.\, x \neq y \,.\, x \neq z \,.\, y \neq z :. \supset \vdash . \mathrm{Prop}$

In virtue of this proposition, a class which is neither null nor a unit class nor a couple contains at least three distinct members. Hence it will follow that any cardinal number other than 0 or 1 or 2 is equal to or greater than 3. The above proposition is used in ∗104·43, which is an existence-theorem of considerable importance in cardinal arithmetic.

The Gödel completeness theorem

K. Gödel, *Die Vollständigkeit der Axiome des logischen Funktionenkalküls*

Formal axioms:

1. $X \vee X \rightarrow X$,
2. $X \rightarrow X \vee Y$,
3. $X \vee Y \rightarrow Y \vee X$,
4. $(X \rightarrow Y) \rightarrow (Z \vee X \rightarrow Z \vee Y)$,
5. $(x)F(x) \rightarrow F(y)$,
6. $(x)[X \vee F(x)] \rightarrow X \vee (x)F(x)$.

Rules of inference:[6]

1. The inferential schema: From A and $A \rightarrow B$, B may be inferred;
2. The rule of substitution for propositional and functional variables;
3. From $A(x)$, $(x)A(x)$ may be inferred;
4. Individual variables (free or bound) may be replaced by any others, so long as this does not cause overlapping of the scopes of variables denoted by the same sign.

The Gödel completeness theorem

K. Gödel, *Die Vollständigkeit der Axiome des logischen Funktionenkalküls*:

Every valid formula of the restricted functional calculus is provable.

K. Gödel, *Über formal unentscheidbare Sätze der Principia mathematica ...*:

The development of mathematics towards greater precision has led, as is well known, to the formalization of large tracts of it, so that one can prove any theorem using nothing but a few mechanical rules. The most comprehensive formal systems that have been set up hitherto are the system of *Principia mathematica* (PM) on the one hand and the Zermelo-Fraenkel axiom system of set theory. These two systems are so comprehensive that in them all methods of proof today used in mathematics are formalized, that is, reduced to a few axioms and rules of inference.

On the complexity of formal proofs

N. Bourbaki: *Theory of Sets*

If formalized mathematics were as simple as the game of chess, then once our chosen formalized language had been described there would remain only the task of writing out our proofs in this language, [...] But the matter is far from being as simple as that, and no great experience is necessary to perceive that such a project is absolutely unrealizable: the tiniest proof at the beginnings of the Theory of Sets would already require several hundreds of signs for its complete formalization. [...] formalized mathematics cannot in practice be written down in full, [...] We shall therefore very quickly abandon formalized mathematics, [...]

On the complexity of formal proofs

K. Gödel, *Über formal unentscheidbare Sätze der Principia mathematica ...*:

22. $FR(x) \equiv (n)\{0 < n \leq l(x) \rightarrow Elf\,(n\,Gl\,x) \vee (Ep, q)\,[0 < p, q < n \,\&\, Op\,(n\,Gl\,x, p\,Gl\,x, q\,Gl\,x)]\} \,\&\, l(x) > 0$

x ist eine Reihe von *Formeln*, deren jede entweder *Elementarformel* ist oder aus den vorhergehenden durch die Operationen der *Negation*, *Disjunktion*, *Generalisation* hervorgeht.

23. $\mathrm{Form}\,(x) \equiv (En)\,\{n \leq (Pr\,[l\,(x)^2])^{x \cdot [l\,(x)]^2} \,\&\, FR\,(n) \,\&\, x = [l\,(n)]\;Gl\,n\}$ [35]

x ist *Formel* (d. h. letztes Glied einer *Formelreihe* n).

45. $x\,B\,y \equiv Bw\,(x) \,\&\, [l\,(x)]\;Gl\,x = y$

x ist ein *Beweis* für die *Formel* y.

46. $\mathrm{Bew}\,(x) \equiv (Ey)\;y\,B\,x$

x ist eine *beweisbare Formel*. [Bew (x) ist der einzige unter den Begriffen 1—46, von dem nicht behauptet werden kann, er sei rekursiv.]

Computer-supported formal mathematics

J. McCarthy: *Computer Programs for Checking Mathematical Proofs*

Checking mathematical proofs is potentially one of the most interesting and useful applications of automatic computers. ... Proofs to be checked by computer may be briefer and easier to write than the informal proofs acceptable to mathematicians. This is because the computer can be asked to do much more work to check each step than a human is willing to do, and this permits longer and fewer steps.

Computer-supported formal proofs

J. Harrison, *Handbook of Practical Logic and Automated Reasoning*

The inductive data type `formula`

```
type ('a)formula = False
                 | True
                 | Atom of 'a
                 | Not of ('a)formula
                 | And of ('a)formula * ('a)formula
                 | Or of ('a)formula * ('a)formula
                 | Imp of ('a)formula * ('a)formula
                 | Iff of ('a)formula * ('a)formula
                 | Forall of string * ('a)formula
                 | Exists of string * ('a)formula;;
```

Computer-supported formal proofs

Recursively defined substitution functions `subst` and `substq`

```
let rec subst subfn fm =
  match fm with
    False -> False
  | True -> True
  | Atom(R(p,args)) -> Atom(R(p,map (tsubst subfn) args))
  | Not(p) -> Not(subst subfn p)
  | And(p,q) -> And(subst subfn p,subst subfn q)
  | Or(p,q) -> Or(subst subfn p,subst subfn q)
  | Imp(p,q) -> Imp(subst subfn p,subst subfn q)
  | Iff(p,q) -> Iff(subst subfn p,subst subfn q)
  | Forall(x,p) -> substq subfn mk_forall x p
  | Exists(x,p) -> substq subfn mk_exists x p

and substq subfn quant x p =
  let x' = if exists (fun y -> mem x (fvt(tryapplyd subfn y (Var y))))
                      (subtract (fv p) [x])
           then variant x (fv(subst (undefine x subfn) p)) else x in
  quant x' (subst ((x |-> Var x') subfn) p);;
```

Computer-supported formal proofs ...

The Prolog-like prover `meson`

```
let puremeson fm =
  let cls = simpcnf(specialize(pnf fm)) in
  let rules = itlist ((@) ** contrapositives) cls [] in
  deepen (fun n ->
     mexpand rules [] False (fun x -> x) (undefined,n,0); n) 0;;
```

```
let meson fm =
  let fm1 = askolemize(Not(generalize fm)) in
  map (puremeson ** list_conj) (simpdnf fm1);;
```

... proof of the Kepler conjecture in HOL Light

```
|- the_kepler_conjecture <=>
        (!V. packing V
            ==> (?c. !r. &1 <= r
                ==> &(CARD(V INTER ball(vec 0,r))) <=
                       pi * r pow 3 / sqrt(&18) + c * r pow 2))

|- the_nonlinear_inequalities /\
   import_tame_classification
   ==> the_kepler_conjecture
```

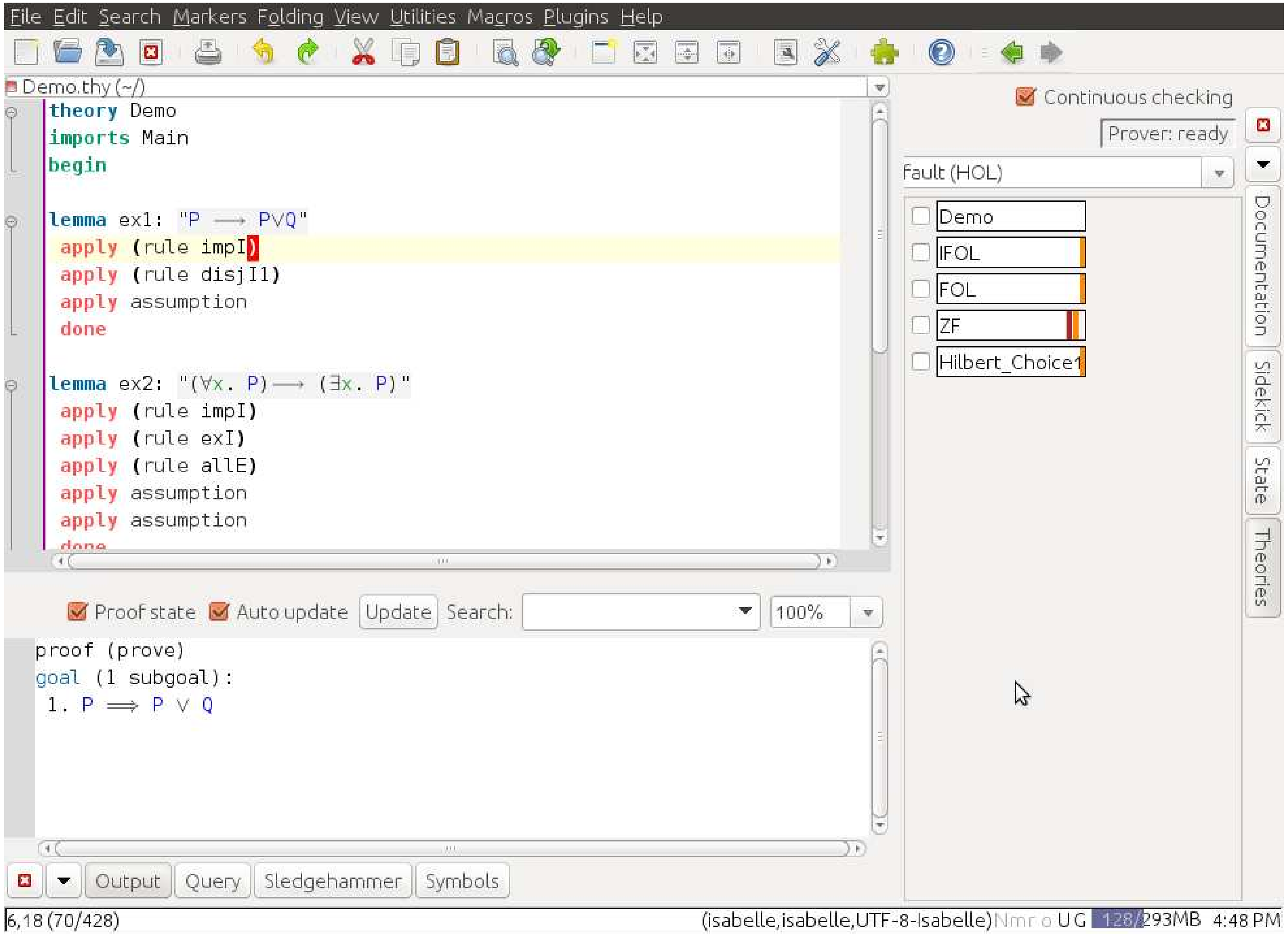
The Isabelle system

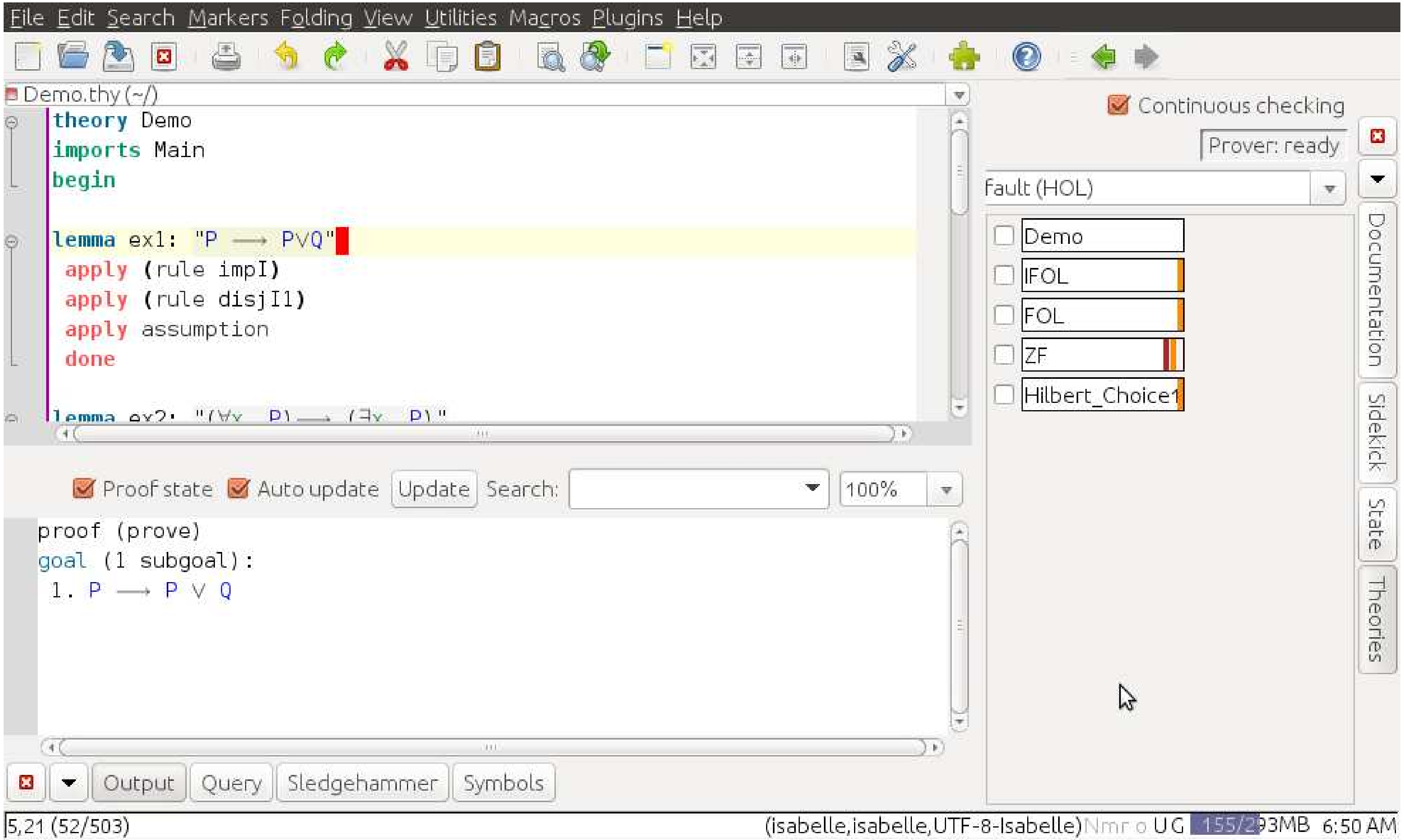
Developed by L. Paulson and others (since 1980s)

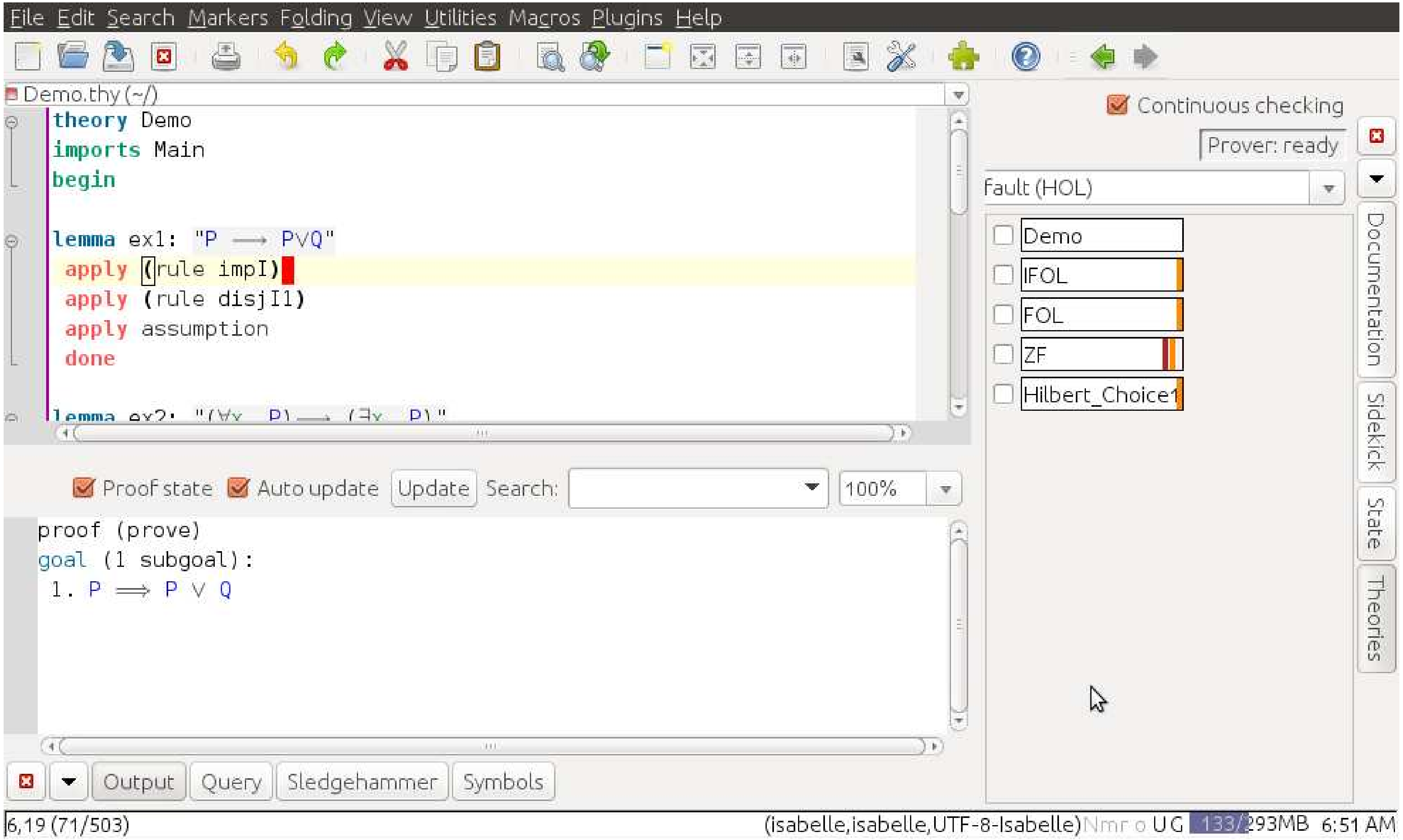
Interactive and programable system for the development of proofs

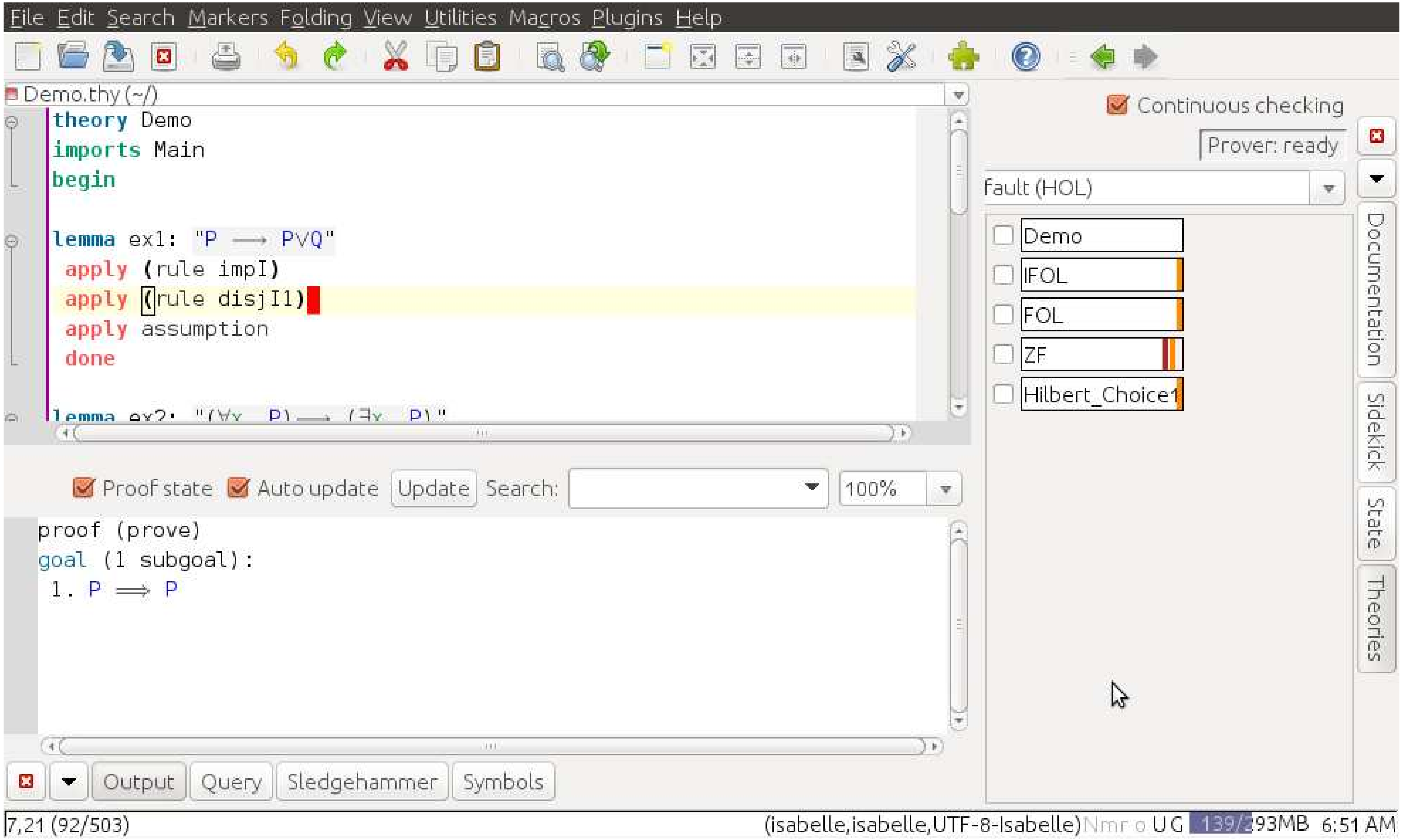
Generic system allowing various logics, e.g., first-order logic (FOL) and higher-order logic (HOL)

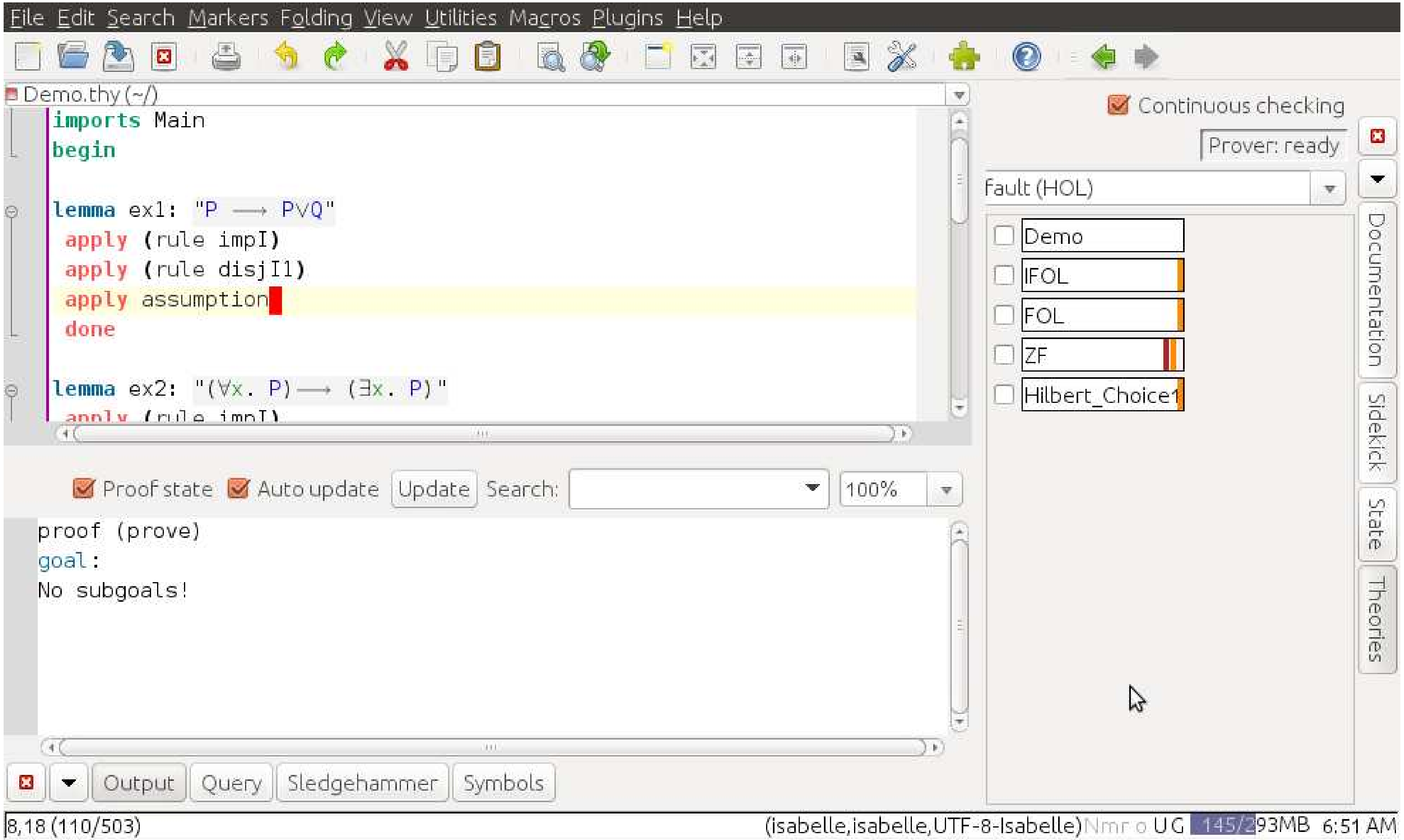
Large scale formalizations: part of Kepler conjecture project

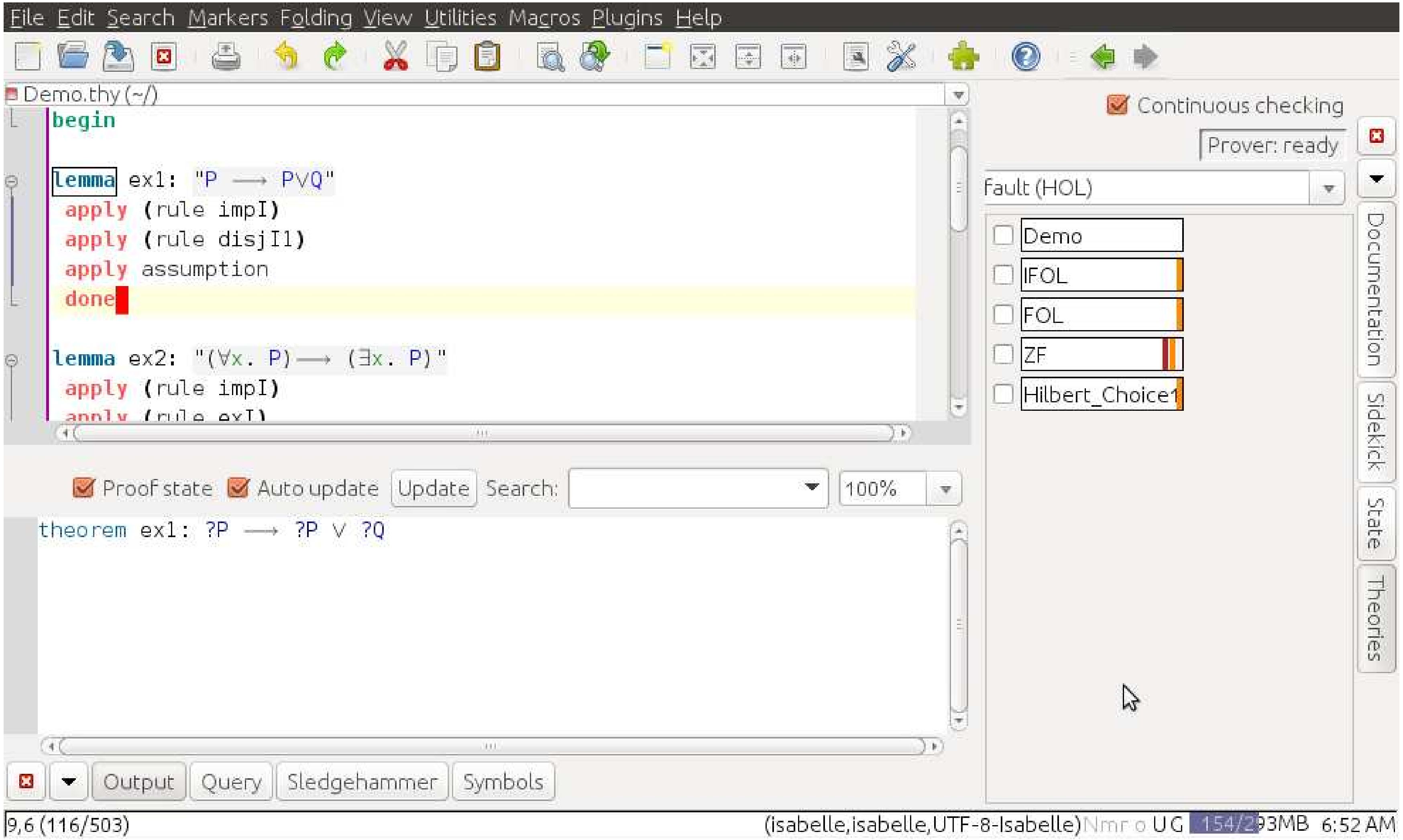

File Edit Search Markers Folding View Utilities Macros Plugins Help
Demo.thy (~/)
theory Demo
imports Main
begin
lemma ex1: "P ⟶ P∨Q"
apply (rule impI)
apply (rule disjI1)
apply assumption
done
lemma ex2: "(∀x. P) ⟶ (∃x. P)"
apply (rule impI)
apply (rule exI)
apply (rule allE)
apply assumption
apply assumption
done
Proof state
Auto update
Update
Search:
100%
proof (prove)
goal (1 subgoal):
1. P ⟹ P ∨ Q
Output
Query
Sledgehammer
Symbols
Continuous checking
Prover: ready
fault (HOL)
Demo
IFOL
FOL
ZF
Hilbert_Choice1
Documentation
Sidekick
State
Theories
6,18 (70/428)
(isabelle,isabelle,UTF-8-Isabelle) Nmr o UG 128/293MB 4:48 PM

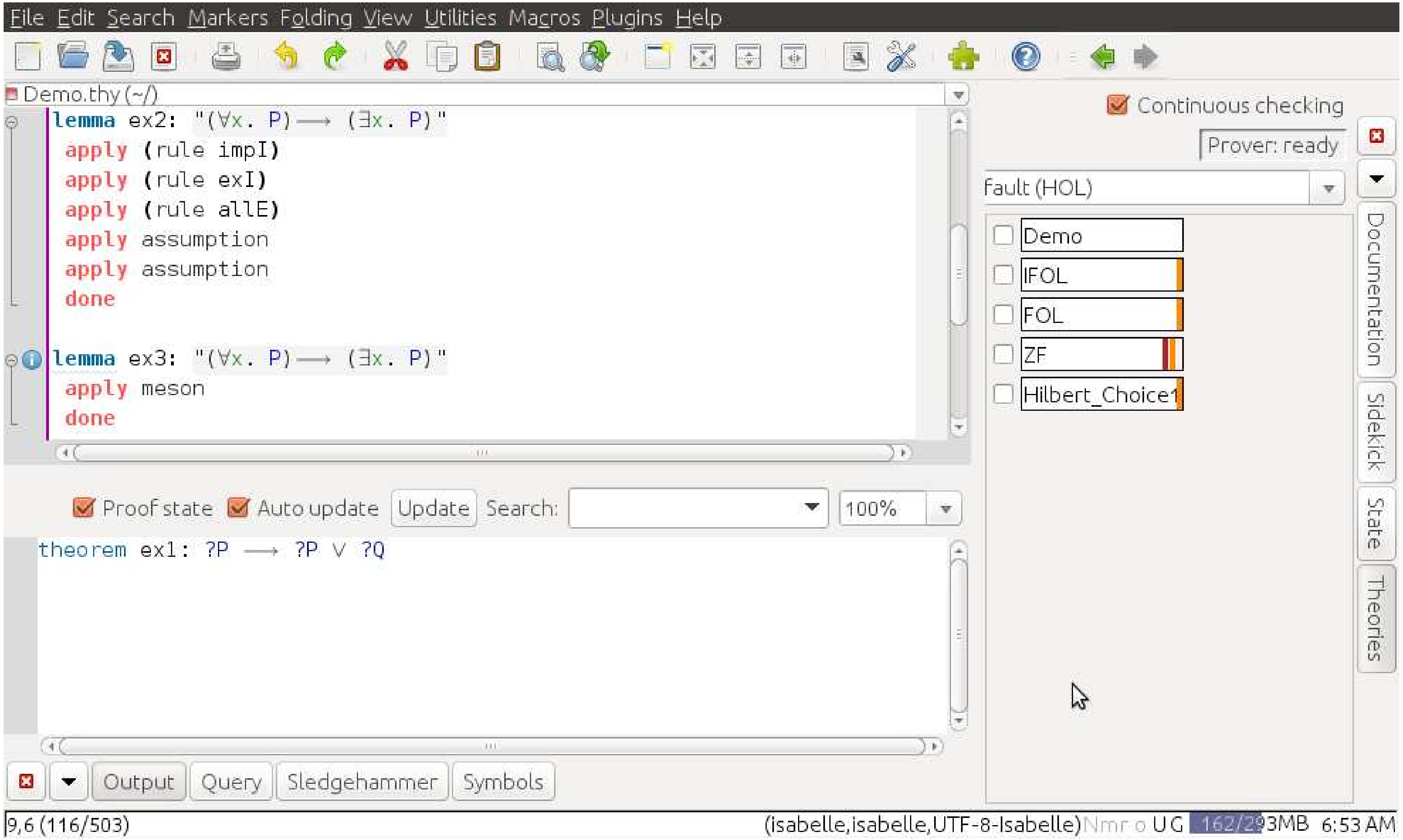

File Edit Search Markers Folding View Utilities Macros Plugins Help
Demo.thy (~/)
theory Demo
imports Main
begin
lemma ex1: "P ⟶ P∨Q"
apply (rule impI)
apply (rule disjI1)
apply assumption
done
Continuous checking
Prover: ready
fault (HOL)
Demo
IFOL
FOL
ZF
Hilbert_Choice1
Documentation
Sidekick
State
Theories
Proof state
Auto update
Update
Search:
100%
proof (prove)
goal (1 subgoal):
1. P ⟶ P ∨ Q
Output
Query
Sledgehammer
Symbols
5,21 (52/503)
(isabelle,isabelle,UTF-8-Isabelle)
6:50 AM

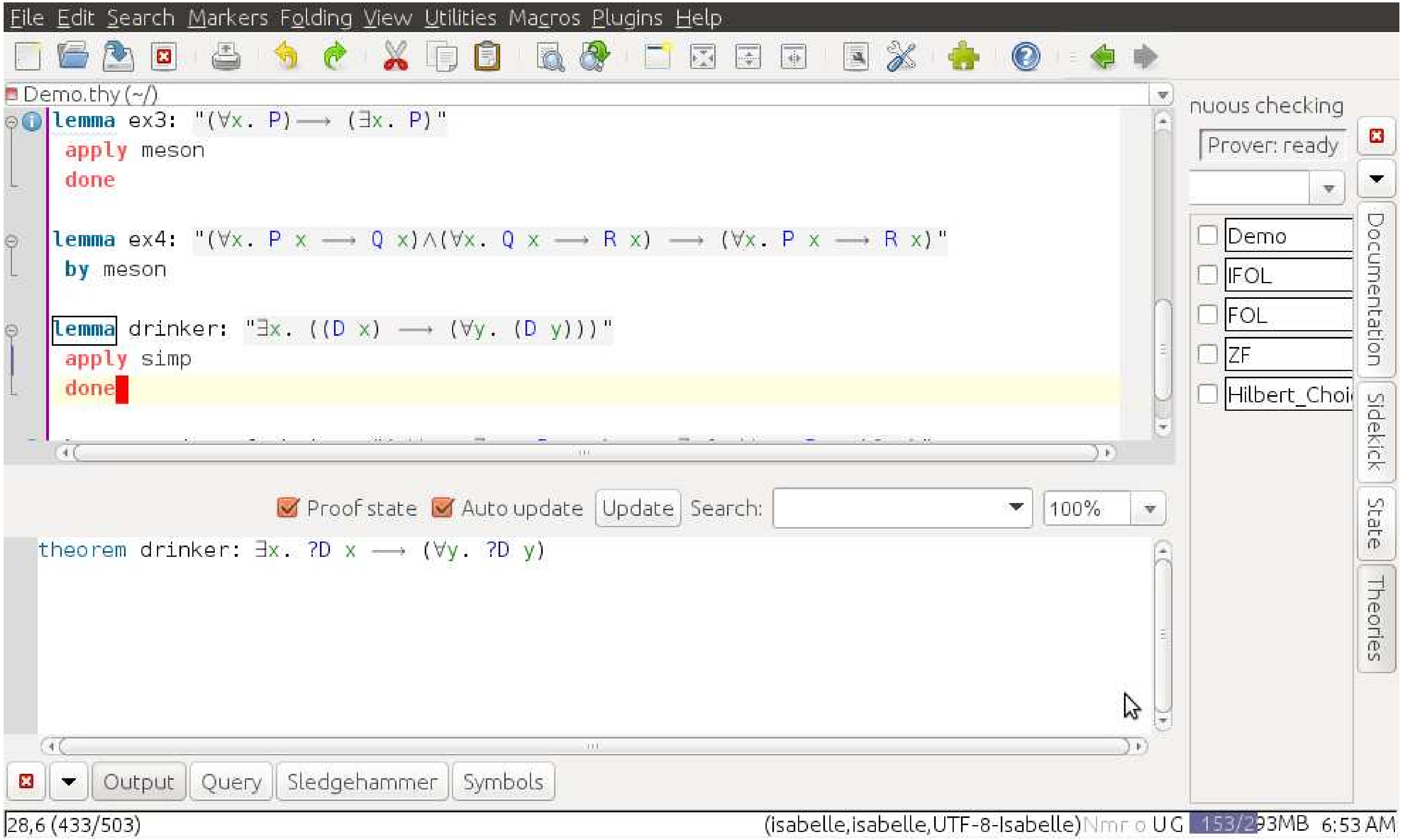

File Edit Search Markers Folding View Utilities Macros Plugins Help
Demo.thy (~/)
theory Demo
imports Main
begin
lemma ex1: "P ⟶ P∨Q"
apply (rule impI)
apply (rule disjI1)
apply assumption
done
Continuous checking
Prover: ready
fault (HOL)
Demo
IFOL
FOL
ZF
Hilbert_Choice1
Documentation
Sidekick
State
Theories
Proof state
Auto update
Update
Search:
100%
proof (prove)
goal (1 subgoal):
1. P ⟹ P ∨ Q
Output
Query
Sledgehammer
Symbols
6,19 (71/503)
(isabelle,isabelle,UTF-8-Isabelle)
293MB
6:51 AM

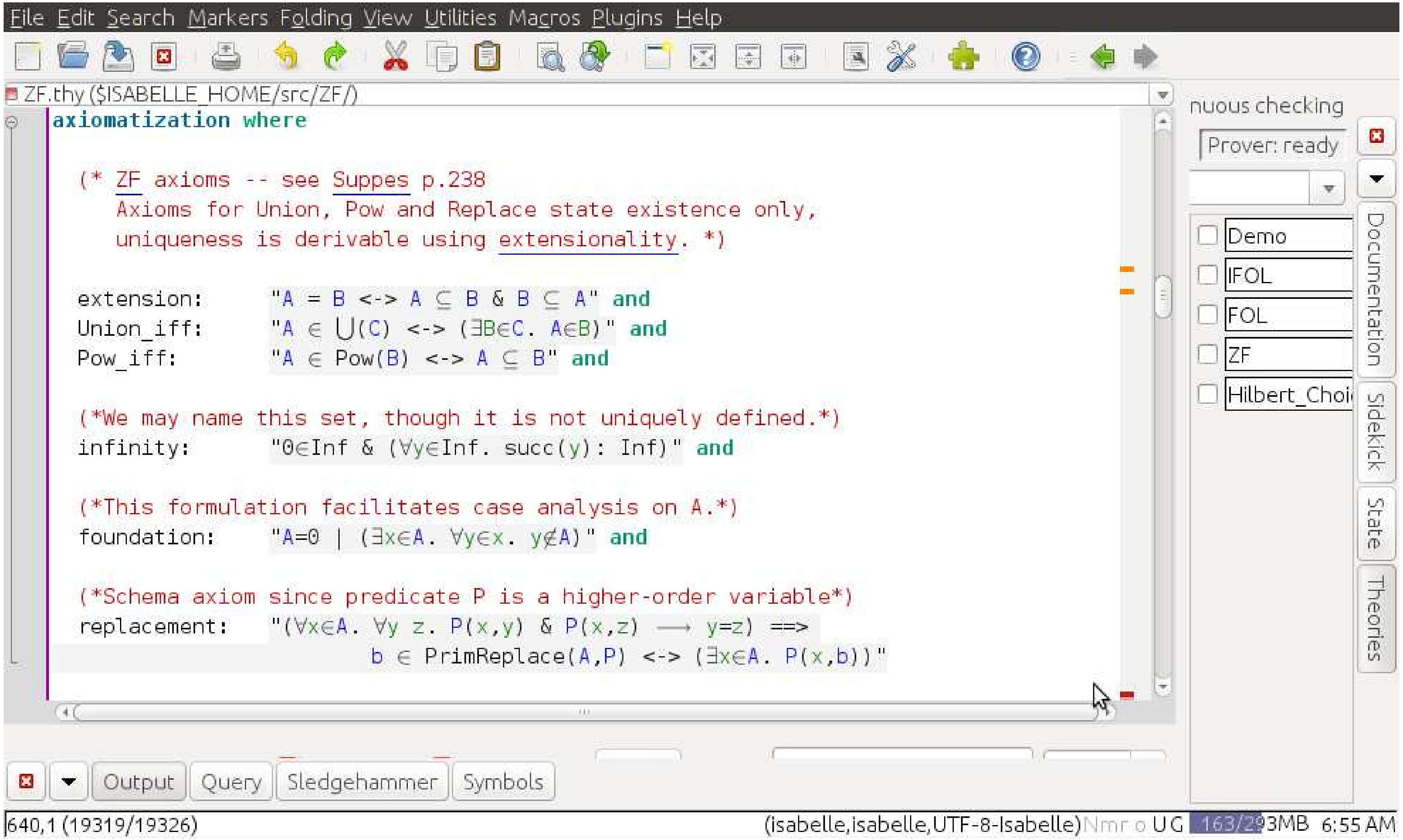

File Edit Search Markers Folding View Utilities Macros Plugins Help
Demo.thy (~/)
theory Demo
imports Main
begin
lemma ex1: "P ⟶ P∨Q"
apply (rule impI)
apply (rule disjI1)
apply assumption
done
Continuous checking
Prover: ready
fault (HOL)
Demo
IFOL
FOL
ZF
Hilbert_Choice1
Documentation
Sidekick
State
Theories
Proof state
Auto update
Update
Search:
100%
proof (prove)
goal (1 subgoal):
1. P ⟹ P
Output
Query
Sledgehammer
Symbols
7,21 (92/503)
(isabelle,isabelle,UTF-8-Isabelle)
6:51 AM

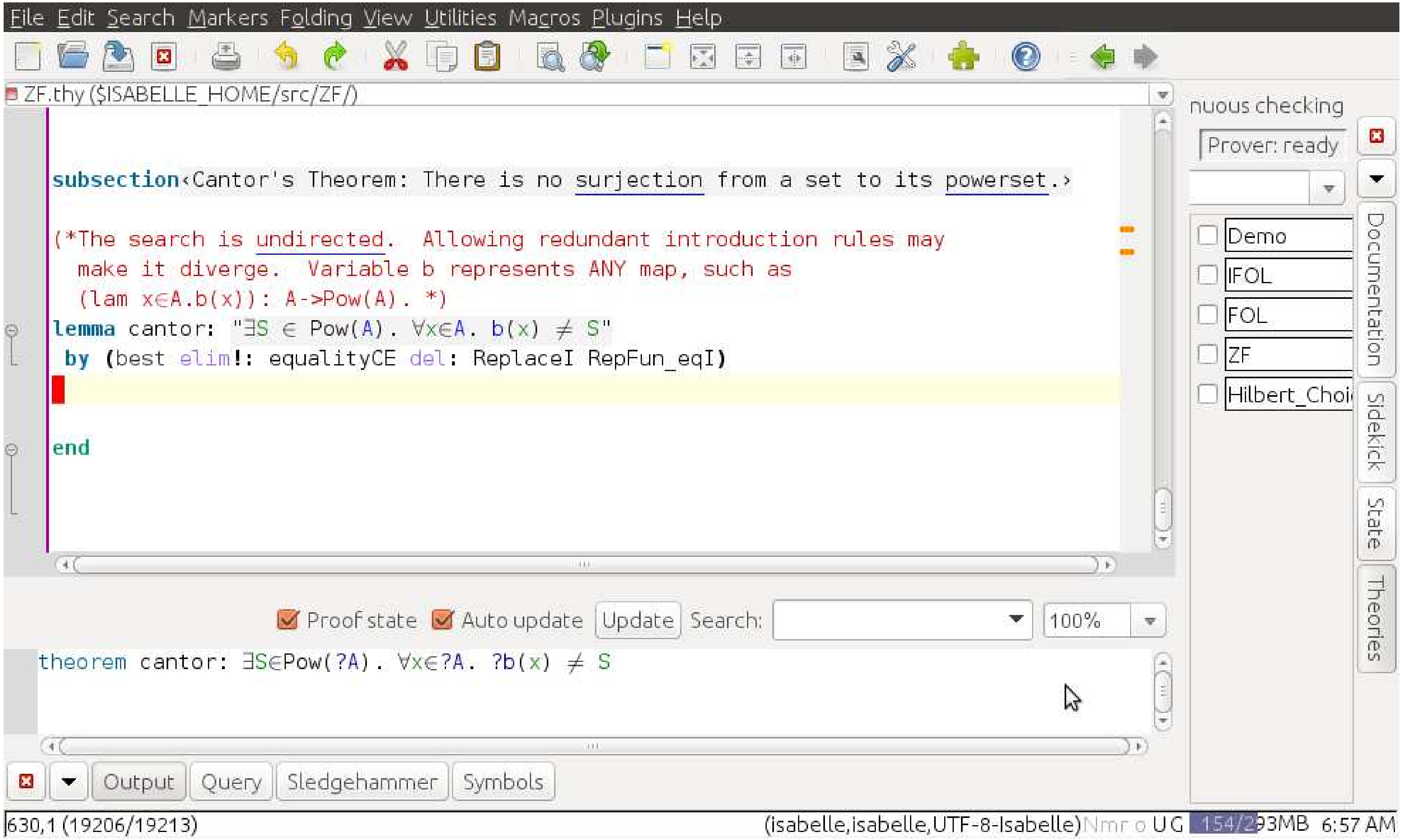

File Edit Search Markers Folding View Utilities Macros Plugins Help
Demo.thy (~/)
imports Main
begin
lemma ex1: "P ⟶ P∨Q"
apply (rule impI)
apply (rule disjI1)
apply assumption
done
lemma ex2: "(∀x. P) ⟶ (∃x. P)"
Proof state
Auto update
Update
Search:
100%
proof (prove)
goal:
No subgoals!
Output
Query
Sledgehammer
Symbols
Continuous checking
Prover: ready
fault (HOL)
Demo
IFOL
FOL
ZF
Hilbert_Choice1
Documentation
Sidekick
State
Theories
8,18 (110/503)
(isabelle,isabelle,UTF-8-Isabelle)
6:51 AM

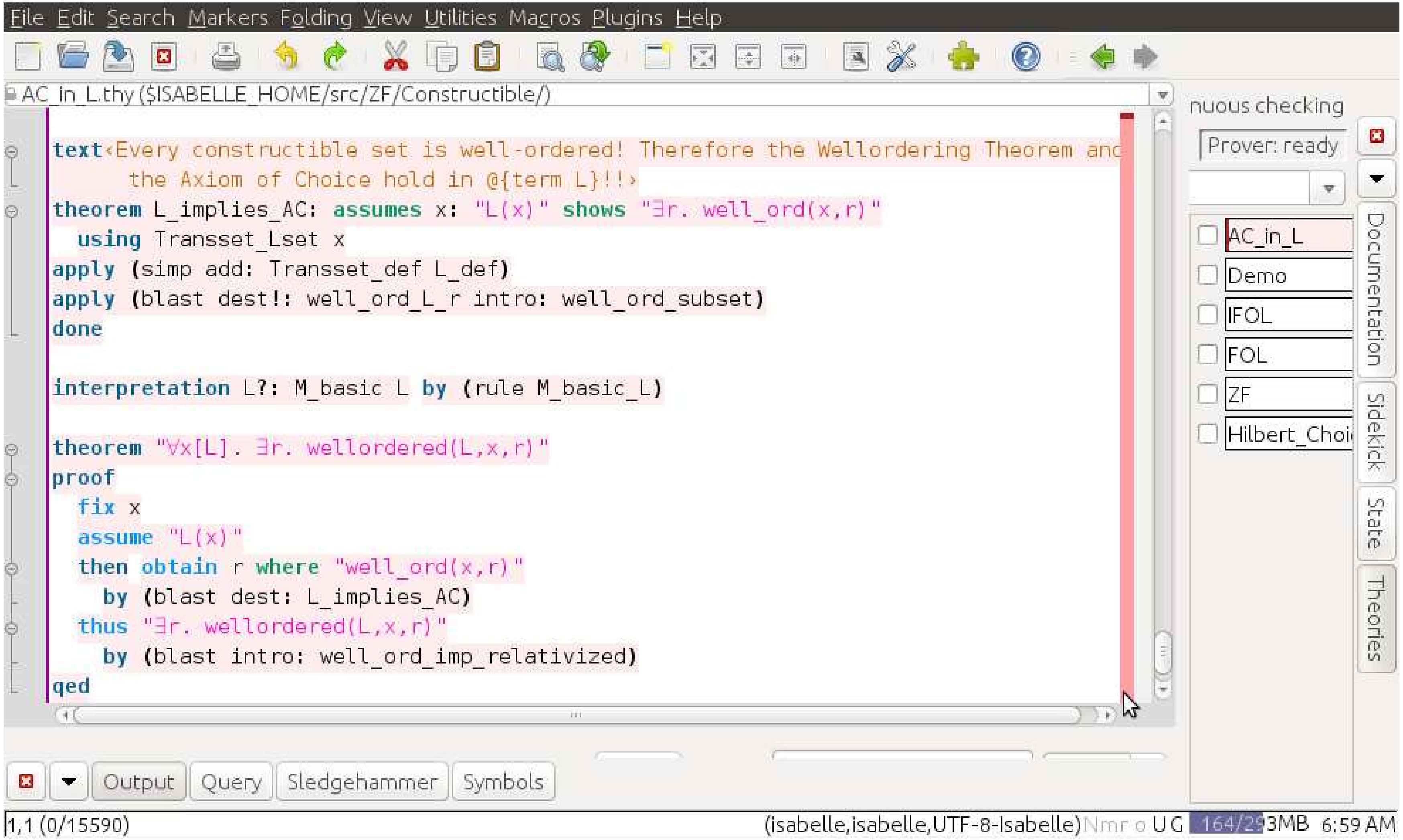

File Edit Search Markers Folding View Utilities Macros Plugins Help
Demo.thy (~/)
begin
lemma ex1: "P ⟶ P∨Q"
apply (rule impI)
apply (rule disjI1)
apply assumption
done
lemma ex2: "(∀x. P) ⟶ (∃x. P)"
apply (rule impI)
Continuous checking
Prover: ready
fault (HOL)
Demo
IFOL
FOL
ZF
Hilbert_Choice1
Documentation
Sidekick
State
Theories
Proof state
Auto update
Update
Search:
100%
theorem ex1: ?P ⟶ ?P ∨ ?Q
Output
Query
Sledgehammer
Symbols
9,6 (116/503)
(isabelle,isabelle,UTF-8-Isabelle)
6:52 AM

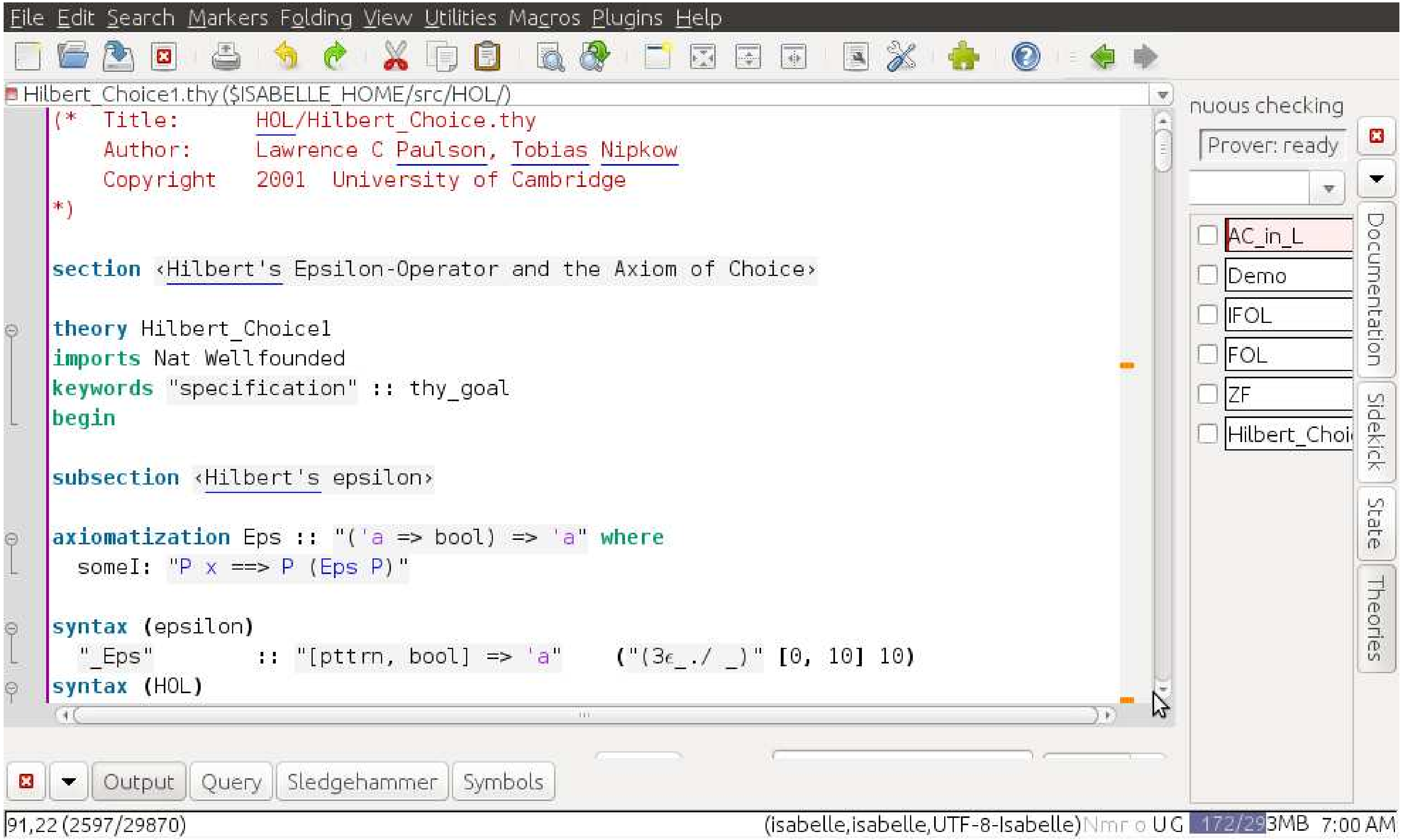

File Edit Search Markers Folding View Utilities Macros Plugins Help
Demo.thy (~/)
lemma ex2: "(∀x. P) ⟶ (∃x. P)"
apply (rule impI)
apply (rule exI)
apply (rule allE)
apply assumption
apply assumption
done
lemma ex3: "(∀x. P) ⟶ (∃x. P)"
apply meson
done
Continuous checking
Prover: ready
fault (HOL)
Demo
IFOL
FOL
ZF
Hilbert_Choice1
Documentation
Sidekick
State
Theories
Proof state
Auto update
Update
Search:
100%
theorem ex1: ?P ⟶ ?P ∨ ?Q
Output
Query
Sledgehammer
Symbols
9,6 (116/503)
(isabelle,isabelle,UTF-8-Isabelle)
162/293MB
6:53 AM

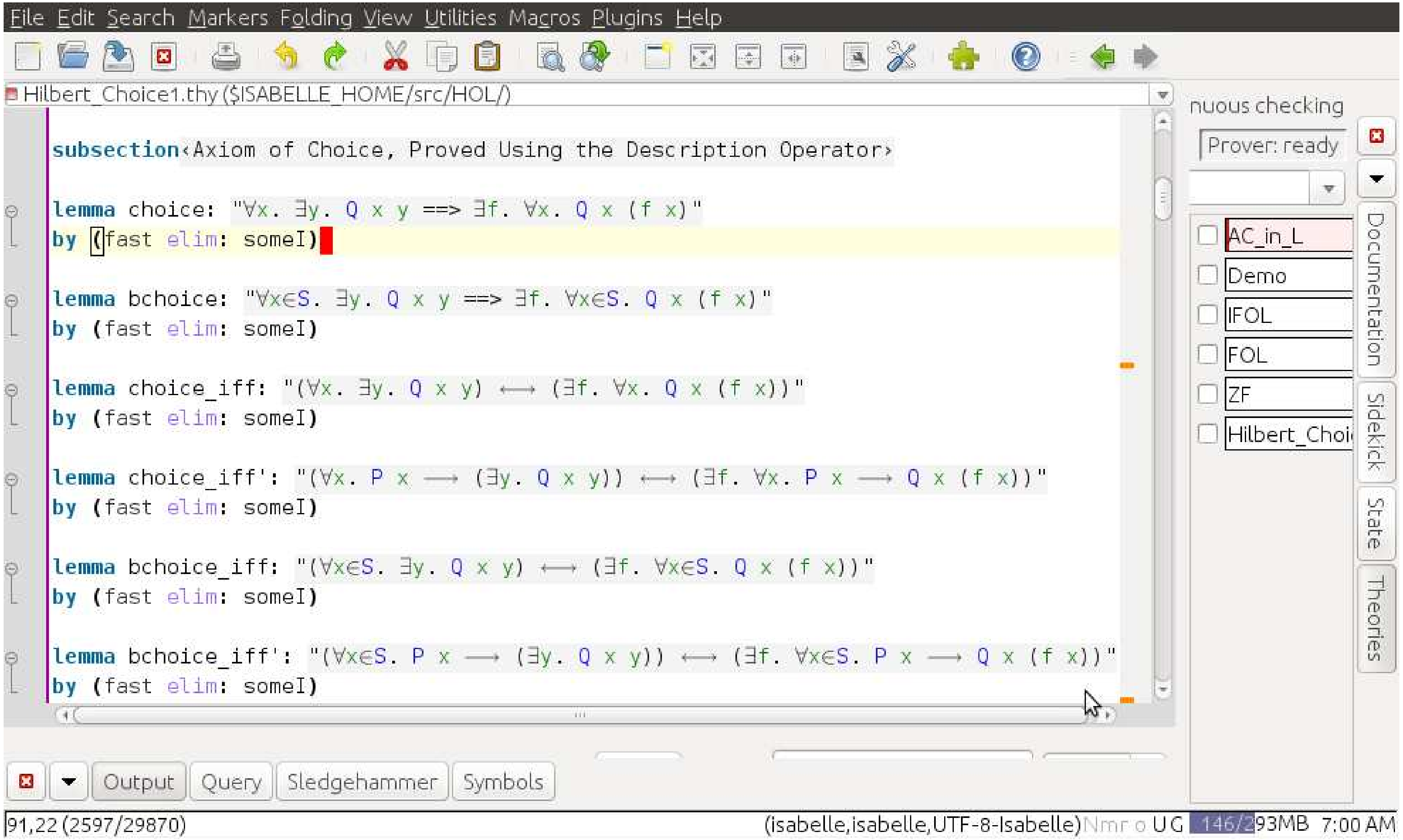

File Edit Search Markers Folding View Utilities Macros Plugins Help
Demo.thy (~/)
lemma ex3: "(∀x. P) ⟶ (∃x. P)"
apply meson
done
lemma ex4: "(∀x. P x ⟶ Q x)∧(∀x. Q x ⟶ R x) ⟶ (∀x. P x ⟶ R x)"
by meson
lemma drinker: "∃x. ((D x) ⟶ (∀y. (D y)))"
apply simp
done
nuous checking
Prover: ready
Demo
IFOL
FOL
ZF
Hilbert_Choi
Documentation
Sidekick
State
Theories
Proof state
Auto update
Update
Search:
100%
theorem drinker: ∃x. ?D x ⟶ (∀y. ?D y)
Output
Query
Sledgehammer
Symbols
28,6 (433/503)
(isabelle,isabelle,UTF-8-Isabelle)
153/293MB
6:53 AM


File Edit Search Markers Folding View Utilities Macros Plugins Help
ZF.thy ($ISABELLE_HOME/src/ZF/)
axiomatization where
(* ZF axioms -- see Suppes p.238
Axioms for Union, Pow and Replace state existence only,
uniqueness is derivable using extensionality. *)
extension: "A = B <-> A ⊆ B & B ⊆ A" and
Union_iff: "A ∈ ⋃(C) <-> (∃B∈C. A∈B)" and
Pow_iff: "A ∈ Pow(B) <-> A ⊆ B" and
(*We may name this set, though it is not uniquely defined.*)
infinity: "0∈Inf & (∀y∈Inf. succ(y): Inf)" and
(*This formulation facilitates case analysis on A.*)
foundation: "A=0 | (∃x∈A. ∀y∈x. y∉A)" and
(*Schema axiom since predicate P is a higher-order variable*)
replacement: "(∀x∈A. ∀y z. P(x,y) & P(x,z) ⟶ y=z) ==>
b ∈ PrimReplace(A,P) <-> (∃x∈A. P(x,b))"
nuous checking
Prover: ready
Demo
IFOL
FOL
ZF
Hilbert_Choi
Documentation
Sidekick
State
Theories
Output
Query
Sledgehammer
Symbols
640,1 (19319/19326)
(isabelle,isabelle,UTF-8-Isabelle)
163/293MB 6:55 AM


File Edit Search Markers Folding View Utilities Macros Plugins Help
ZF.thy ($ISABELLE_HOME/src/ZF/)
subsection‹Cantor's Theorem: There is no surjection from a set to its powerset.›
(*The search is undirected. Allowing redundant introduction rules may
make it diverge. Variable b represents ANY map, such as
(lam x∈A.b(x)): A->Pow(A). *)
lemma cantor: "∃S ∈ Pow(A). ∀x∈A. b(x) ≠ S"
by (best elim!: equalityCE del: ReplaceI RepFun_eqI)
end
nuous checking
Prover: ready
Demo
IFOL
FOL
ZF
Hilbert_Choi
Documentation
Sidekick
State
Theories
Proof state
Auto update
Update
Search:
100%
theorem cantor: ∃S∈Pow(?A). ∀x∈?A. ?b(x) ≠ S
Output
Query
Sledgehammer
Symbols
630,1 (19206/19213)
(isabelle,isabelle,UTF-8-Isabelle)
154/293MB
6:57 AM


File Edit Search Markers Folding View Utilities Macros Plugins Help
AC_in_L.thy ($ISABELLE_HOME/src/ZF/Constructible/)
text‹Every constructible set is well-ordered! Therefore the Wellordering Theorem and
the Axiom of Choice hold in @{term L}!!›
theorem L_implies_AC: assumes x: "L(x)" shows "∃r. well_ord(x,r)"
using Transset_Lset x
apply (simp add: Transset_def L_def)
apply (blast dest!: well_ord_L_r intro: well_ord_subset)
done
interpretation L?: M_basic L by (rule M_basic_L)
theorem "∀x[L]. ∃r. wellordered(L,x,r)"
proof
fix x
assume "L(x)"
then obtain r where "well_ord(x,r)"
by (blast dest: L_implies_AC)
thus "∃r. wellordered(L,x,r)"
by (blast intro: well_ord_imp_relativized)
qed
nuous checking
Prover: ready
AC_in_L
Demo
IFOL
FOL
ZF
Hilbert_Choi
Documentation
Sidekick
State
Theories
Output
Query
Sledgehammer
Symbols
1,1 (0/15590)
(isabelle,isabelle,UTF-8-Isabelle) Nmr o UG 164/293MB 6:59 AM


File Edit Search Markers Folding View Utilities Macros Plugins Help
Hilbert_Choice1.thy ($ISABELLE_HOME/src/HOL/)
(* Title: HOL/Hilbert_Choice.thy
Author: Lawrence C Paulson, Tobias Nipkow
Copyright 2001 University of Cambridge
*)
section ‹Hilbert's Epsilon-Operator and the Axiom of Choice›
theory Hilbert_Choice1
imports Nat Wellfounded
keywords "specification" :: thy_goal
begin
subsection ‹Hilbert's epsilon›
axiomatization Eps :: "('a => bool) => 'a" where
someI: "P x ==> P (Eps P)"
syntax (epsilon)
"_Eps" :: "[pttrn, bool] => 'a" ("(3ϵ_./ _)" [0, 10] 10)
syntax (HOL)
nuous checking
Prover: ready
AC_in_L
Demo
IFOL
FOL
ZF
Hilbert_Choi
Documentation
Sidekick
State
Theories
Output
Query
Sledgehammer
Symbols
91,22 (2597/29870)
(isabelle,isabelle,UTF-8-Isabelle)Nmr o UG 172/293MB 7:00 AM


File Edit Search Markers Folding View Utilities Macros Plugins Help
Hilbert_Choice1.thy ($ISABELLE_HOME/src/HOL/)
subsection‹Axiom of Choice, Proved Using the Description Operator›
lemma choice: "∀x. ∃y. Q x y ==> ∃f. ∀x. Q x (f x)"
by (fast elim: someI)
lemma bchoice: "∀x∈S. ∃y. Q x y ==> ∃f. ∀x∈S. Q x (f x)"
by (fast elim: someI)
lemma choice_iff: "(∀x. ∃y. Q x y) ⟷ (∃f. ∀x. Q x (f x))"
by (fast elim: someI)
lemma choice_iff': "(∀x. P x ⟶ (∃y. Q x y)) ⟷ (∃f. ∀x. P x ⟶ Q x (f x))"
by (fast elim: someI)
lemma bchoice_iff: "(∀x∈S. ∃y. Q x y) ⟷ (∃f. ∀x∈S. Q x (f x))"
by (fast elim: someI)
lemma bchoice_iff': "(∀x∈S. P x ⟶ (∃y. Q x y)) ⟷ (∃f. ∀x∈S. P x ⟶ Q x (f x))"
by (fast elim: someI)
nuous checking
Prover: ready
AC_in_L
Demo
IFOL
FOL
ZF
Hilbert_Choi
Documentation
Sidekick
State
Theories
Output
Query
Sledgehammer
Symbols
91,22 (2597/29870)
(isabelle,isabelle,UTF-8-Isabelle) Nmr o UG 146/293MB 7:00 AM

The Isabelle system

Backward proving through application of proof methods

Reducing goal to subgoals and eventually to empty list of subgoals

Limited insight in the “real proof”

Forward proving through **Isar** proof language

Isabelle and set theory

Advanced set theory has been formalized in Isabelle

Set theory can be formalized in several ways: ZF / NGB in FOL / HOL

Inference between Isabelle's logic / type theory with set theoretic axioms

Requires further analysis, especially for axiomatic investigations

(Un-)Naturality of formal mathematics

Freek Wiedijk, *The QED manifesto revisited*

The other reason that there has not been much progress on the vision [...] is that currently formalized mathematics does not resemble real mathematics at all. Formal proofs look like computer program source code. For people who do like reading program source code that is nice, but most mathematicians [...] do not fall in that class.

Apply-style Isabelle

```
lemma iterates_omega_fixedpoint:
     "[| Normal(F); Ord(a) |] ==> F(F^\<omega> (a)) = F^\<omega> (a)"
apply (frule Normal_increasing, assumption)
apply (erule leE)
apply (simp_all add: iterates_omega_triv [OF sym])  (*for subgoal 2*)
apply (simp add:  iterates_omega_def Normal_Union)
apply (rule equalityI, force simp add: nat_succI)
apply clarify
apply (rule UN_I, assumption)
apply (frule iterates_Normal_increasing, assumption, assumption, simp)
apply (blast intro: Ord_trans ltD Ord_iterates_Normal Normal_imp_Ord [of F])
done
```

Forward proving in Isar

```
lemma UNIV_is_not_in_ZF: "UNIV \<noteq> explode R"
proof
  let ?Russell = "{ x. Not(Elem x x) }"
  have "?Russell = UNIV" by (simp add: irreflexiv_Elem)
  moreover assume "UNIV = explode R"
  ultimately have russell: "?Russell = explode R" by simp
  then show "False"
  proof(cases "Elem R R")
    case True
    then show ?thesis
      by (insert irreflexiv_Elem, auto)
  next
    case False
    then have "R \<in> ?Russell" by auto
    then have "Elem R R" by (simp add: russell explode_def)
    with False show ?thesis by auto
  qed
qed
```

The Naproche project: Natural language proof checking

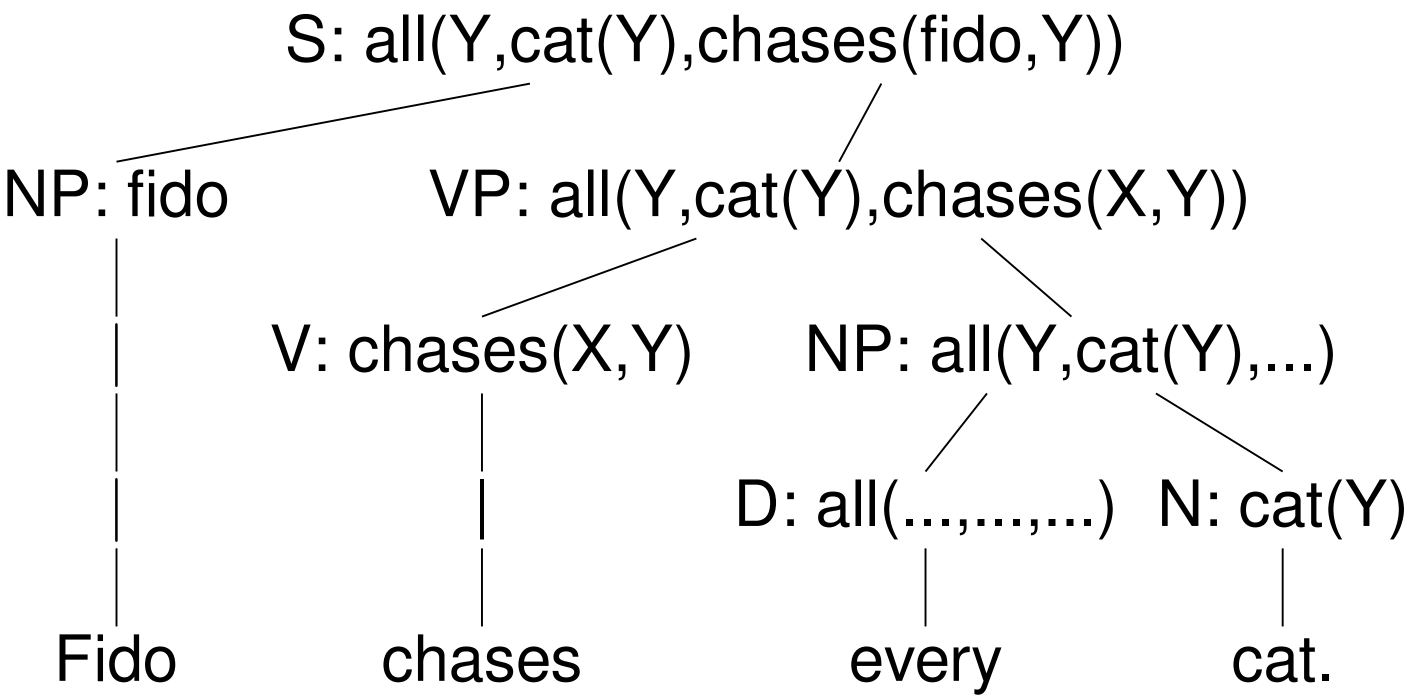
- combining formal mathematics with computer linguistics
- joint with M. Cramer and B. Schröder
- development of a mathematical authoring system with a $\LaTeX$-quality graphical interface

Mathematical statements

"V contains every set." $\longleftrightarrow$ "Fido chases every cat."

Linguistic analysis

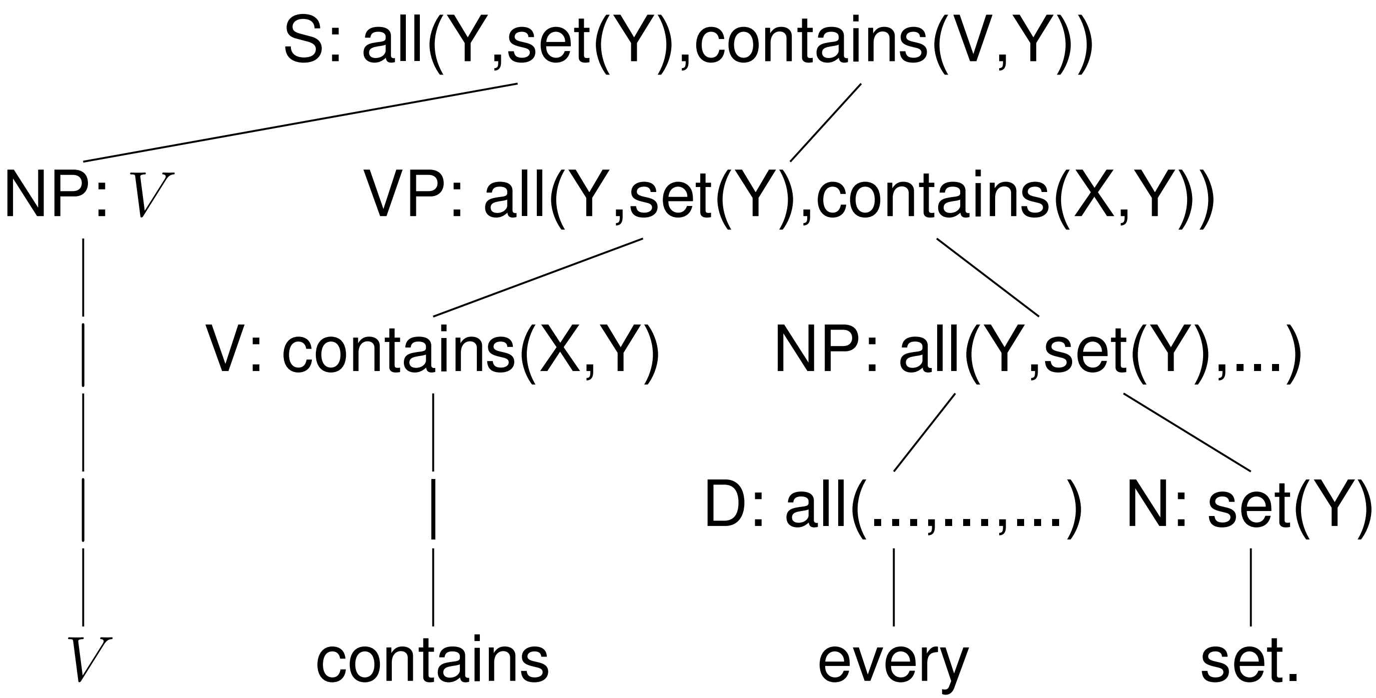
“Fido chases every cat.”



$\forall Y\,(\mathrm{cat}(Y) \rightarrow \mathrm{chases}(\mathrm{fido}, Y))$.

Linguistic analysis

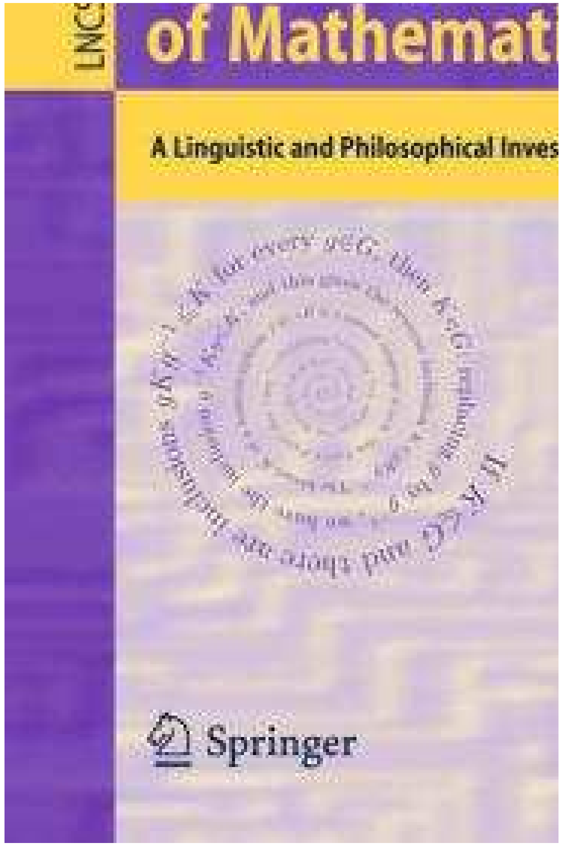
"V contains every set."



$\forall Y(\mathrm{set}(Y) \to V \supseteq Y)$.

The Language of Mathematics

- Mohan Ganesalingam: *The Language of Mathematics*,



Andrei Paskevich’ System of Automatic Deduction (SAD)

- started by Victor Glushkov, continued with Alexander Lyaletski and Konstantin Verchinine

- simple phrase structure grammar

- http://nevidal.org/sad.en.html

Linguistically improved SAD example: Cantor's theorem

The power set of A is the set of subsets of A. Let $\mathcal{P}(A)$ denote the power set of A.

Theorem 1. *There is no surjection from A onto the power set of A.*

Proof. Assume F is a surjection from A onto $\mathcal{P}(A)$. Let

$$B = \{x \in A \mid x \notin F(x)\}.$$

$B \in \mathcal{P}(A)$. Take $a \in A$ such that $B = F(a)$.

$$a \in B \text{ iff } a \notin F(a) \text{ iff } a \notin B.$$

Contradiction. $\square$

Outlook

- Combine techniques from various formal mathematics systems to obtain power and naturalness

- Will this lead to acceptance by mathematical practioneers?

- J. Avigad: On a personal note, I am entirely convinced that formal verification of mathematics will eventually become commonplace.

- D. Scott: Big Proofs will soon show that computers and logic have to be used TOGETHER to make progress in certain areas of mathematics. That is, we need to show convincingly how
  COMPUTER-ASSISTED PROOFS APPLY TO MATHEMATICS.
  We are almost there [...].

Auf eine erfolgreiche AG “Formale Mathematik!