Undecidability of Peano Arithmetic

Marc Hermes

2. July 2020

SAARLAND UNIVERSITY

COMPUTER SCIENCE

SIC Saarland Informatics Campus

Main Statement (Informally)

Main Statement (Informally)

Considering logical entailment for the first-order theory of Peano Arithmetic (PA), we can show

Main Statement (Informally)

Considering logical entailment for the first-order theory of Peano Arithmetic (PA), we can show

> There is no algorithm which can tell us for every formula φ if it holds in every model of PA.

Main Statement (Informally)

Considering logical entailment for the first-order theory of Peano Arithmetic (PA), we can show

> There is no algorithm which can tell us for every formula φ if it holds in every model of PA.

The proof of this is fully mechanised Coq.

The Coq Proof Assistant

The Coq Proof Assistant

- Coq is an interactive proof assistant [The Coq Proof Assistant, 2020]
- Based on the *calculus of constructions* by Thierry Coquand

The Coq Proof Assistant

- Coq is an interactive proof assistant [The Coq Proof Assistant, 2020]
- Based on the *calculus of constructions* by Thierry Coquand
- Work started in 1984 by Coquand and Gérard Huet
- Is still actively developed and supported

The Coq Proof Assistant

- Coq is an interactive proof assistant [The Coq Proof Assistant, 2020]
- Based on the *calculus of constructions* by Thierry Coquand
- Work started in 1984 by Coquand and Gérard Huet
- Is still actively developed and supported

Noteworthy proofs that have been mechanised in Coq:

The Coq Proof Assistant

- Coq is an interactive proof assistant [The Coq Proof Assistant, 2020]
- Based on the *calculus of constructions* by Thierry Coquand
- Work started in 1984 by Coquand and Gérard Huet
- Is still actively developed and supported

Noteworthy proofs that have been mechanised in Coq:

- Four Colour Theorem [Gonthier, 2008]
- Feit-Thompson Theorem [Gonthier et al., 2013]
- CompCert Compiler [Leroy et al., 2012]

The Coq Proof Assistant

- Coq is an interactive proof assistant [The Coq Proof Assistant, 2020]
- Based on the *calculus of constructions* by Thierry Coquand
- Work started in 1984 by Coquand and Gérard Huet
- Is still actively developed and supported

Noteworthy proofs that have been mechanised in Coq:

- Four Colour Theorem [Gonthier, 2008]
- Feit-Thompson Theorem [Gonthier et al., 2013]
- CompCert Compiler [Leroy et al., 2012]

and most relevant for this talk:

The Coq Proof Assistant

- Coq is an interactive proof assistant [The Coq Proof Assistant, 2020]
- Based on the *calculus of constructions* by Thierry Coquand
- Work started in 1984 by Coquand and Gérard Huet
- Is still actively developed and supported

Noteworthy proofs that have been mechanised in Coq:

- Four Colour Theorem [Gonthier, 2008]
- Feit-Thompson Theorem [Gonthier et al., 2013]
- CompCert Compiler [Leroy et al., 2012]

and most relevant for this talk:

- Hilbert's 10th Problem [Larchey-Wendling and Forster, 2019]

Coq and Mathematics

Mathematics is in the most part implicitly framed in set theory. Coq is based on a different kind of foundational theory. (dependent type theory)

Coq and Mathematics

Mathematics is in the most part implicitly framed in set theory. Coq is based on a different kind of foundational theory. (dependent type theory)

There are a lot of intuitions mathematicians have, which are not justified in set theory, but *are* when using a type theory.

Coq and Mathematics

Mathematics is in the most part implicitly framed in set theory. Coq is based on a different kind of foundational theory. (dependent type theory)

There are a lot of intuitions mathematicians have, which are not justified in set theory, but *are* when using a type theory.

- $2 = (0, 0)$
- $\emptyset + 1 = \{\emptyset\}$
- $\sin(\cos) \in \pi$

Coq and Mathematics

Mathematics is in the most part implicitly framed in set theory. Coq is based on a different kind of foundational theory. (dependent type theory)

There are a lot of intuitions mathematicians have, which are not justified in set theory, but *are* when using a type theory.

- $2 = \{\emptyset, \{\emptyset\}\} = (\emptyset, \emptyset) = (0, 0)$
- $0 + 1 = 1$
- most likely $\sin(\cos) \notin \pi$

Coq and Mathematics

Mathematics is in the most part implicitly framed in set theory. Coq is based on a different kind of foundational theory. (dependent type theory)

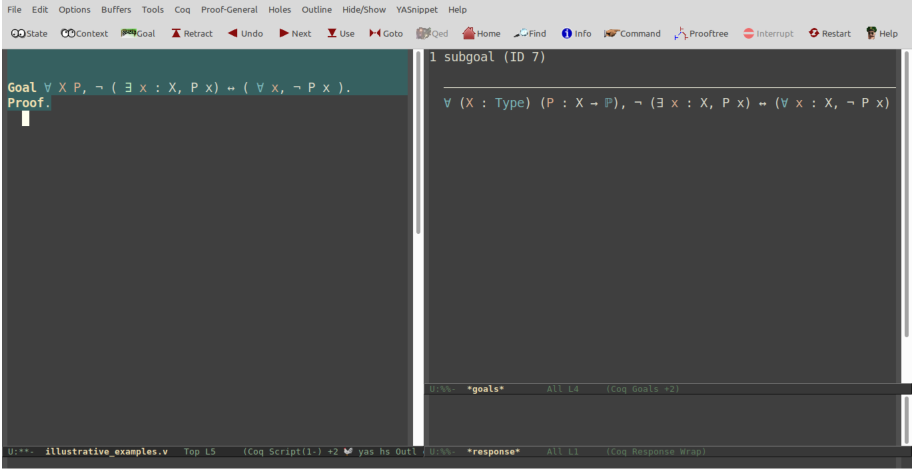
There are a lot of intuitions mathematicians have, which are not justified in set theory, but *are* when using a type theory.

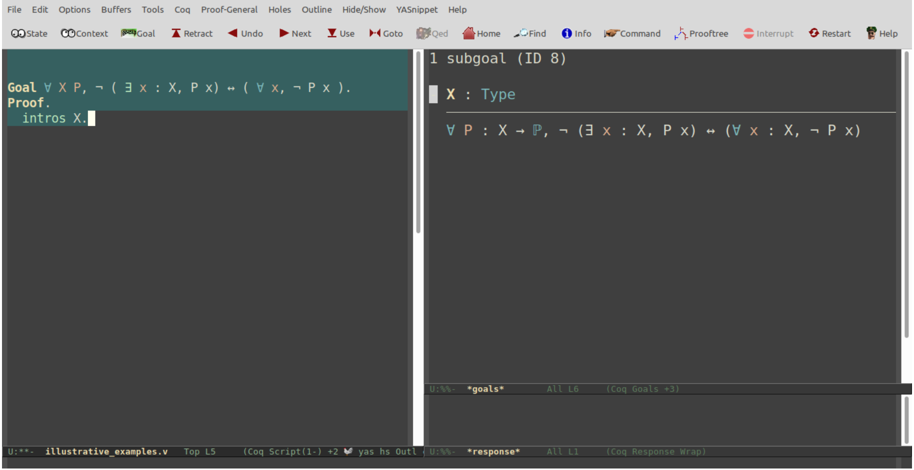
- $2 = (0, 0)$
- $\emptyset + 1 = \{\emptyset\}$
- $\text{sin}(\text{cos}) \in \pi$

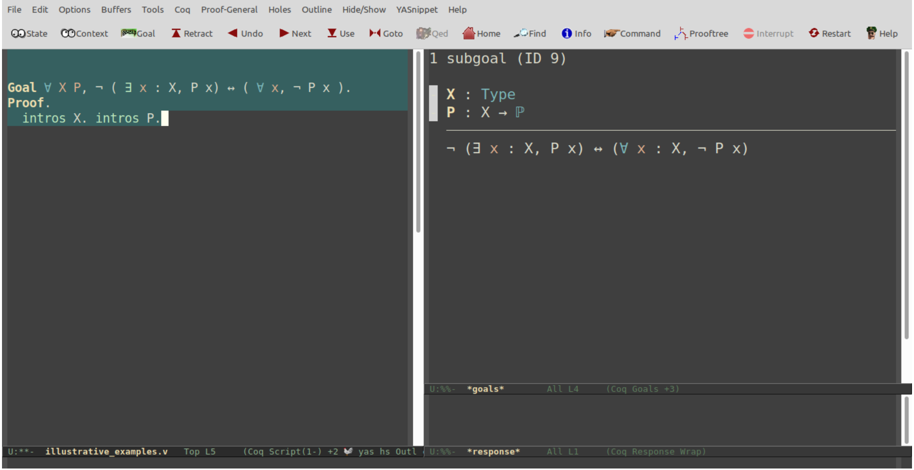
In agreement with intuition, the above statements do not make sense in type theory!

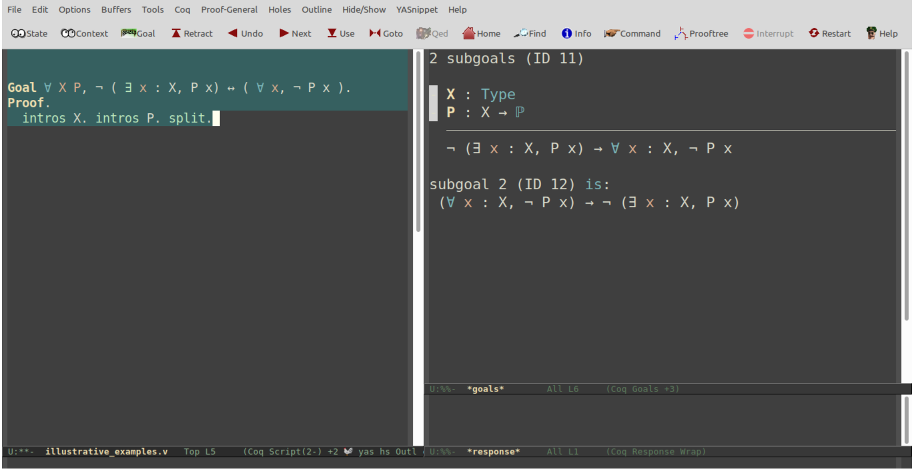
Proofs in Coq

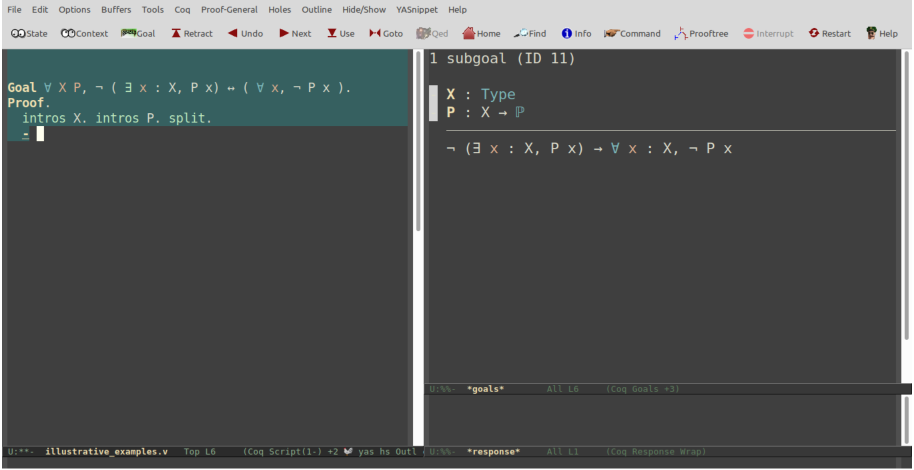
Let's look at some proofs inside of Coq!

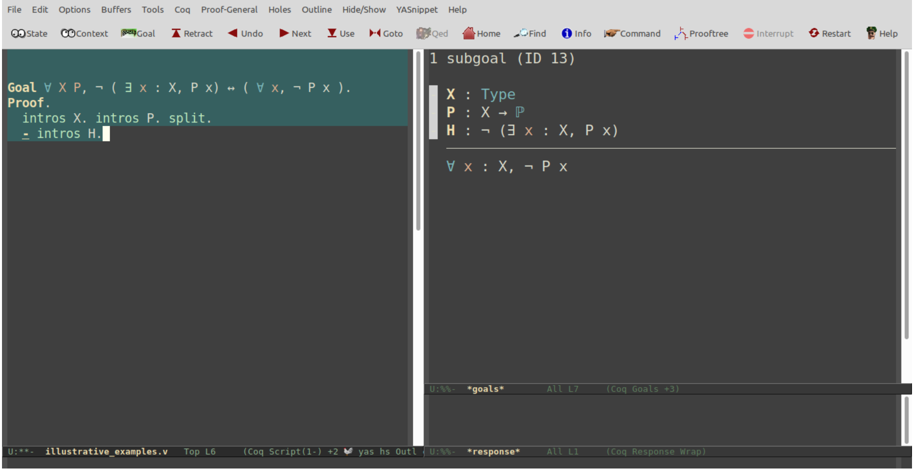

File Edit Options Buffers Tools Coq Proof-General Holes Outline Hide/Show YASnippet Help
State Context Goal Retract Undo Next Use Goto Qed Home Find Info Command Prooftree Interrupt Restart Help
Goal ∀ X P, ¬ (∃ x : X, P x) ↔ (∀ x, ¬ P x).
Proof.
1 subgoal (ID 7)
∀ (X : Type) (P : X → ℙ), ¬ (∃ x : X, P x) ↔ (∀ x : X, ¬ P x)
U:%%- *goals* All L4 (Coq Goals +2)
U:**- illustrative_examples.v Top L5 (Coq Script(1-) +2 yas hs Outl
U:%%- *response* All L1 (Coq Response Wrap)

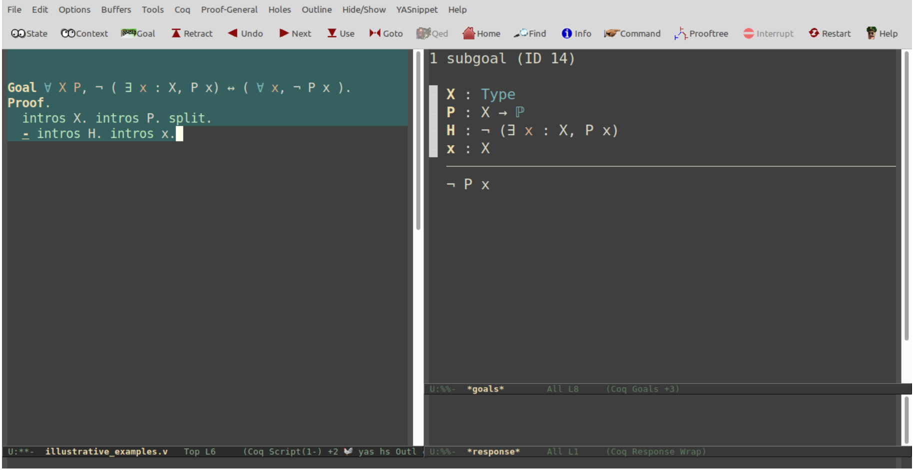

File Edit Options Buffers Tools Coq Proof-General Holes Outline Hide/Show YASnippet Help
State Context Goal Retract Undo Next Use Goto Qed Home Find Info Command Prooftree Interrupt Restart Help
Goal ∀ X P, ¬ (∃ x : X, P x) ↔ (∀ x, ¬ P x).
Proof.
intros X.
1 subgoal (ID 8)
X : Type
∀ P : X → ℙ, ¬ (∃ x : X, P x) ↔ (∀ x : X, ¬ P x)
U:%%- *goals* All L6 (Coq Goals +3)
U:**- illustrative_examples.v Top L5 (Coq Script(1-) +2 yas hs Outl
U:%%- *response* All L1 (Coq Response Wrap)

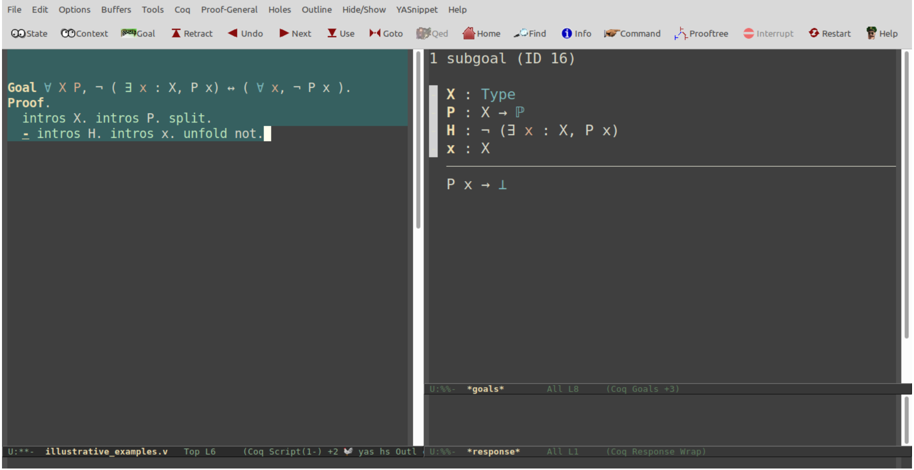

File Edit Options Buffers Tools Coq Proof-General Holes Outline Hide/Show YASnippet Help
State Context Goal Retract Undo Next Use Goto Qed Home Find Info Command Prooftree Interrupt Restart Help
Goal ∀ X P, ¬ (∃ x : X, P x) ↔ (∀ x, ¬ P x).
Proof.
intros X. intros P.
1 subgoal (ID 9)
X : Type
P : X → ℙ
¬ (∃ x : X, P x) ↔ (∀ x : X, ¬ P x)
U:%%- *goals* All L4 (Coq Goals +3)
U:**- illustrative_examples.v Top L5 (Coq Script(1-) +2 yas hs Outl
U:%%- *response* All L1 (Coq Response Wrap)

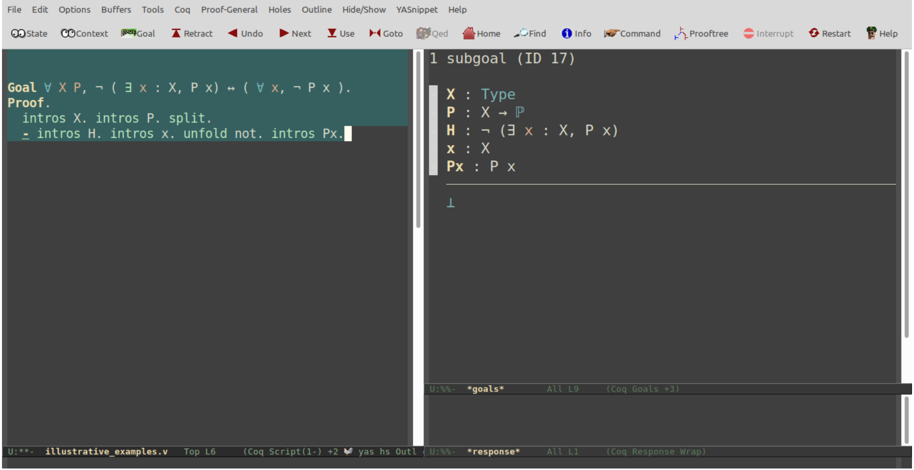

File Edit Options Buffers Tools Coq Proof-General Holes Outline Hide/Show YASnippet Help
State Context Goal Retract Undo Next Use Goto Qed Home Find Info Command Prooftree Interrupt Restart Help
Goal ∀ X P, ¬ (∃ x : X, P x) ↔ (∀ x, ¬ P x).
Proof.
intros X. intros P. split.
2 subgoals (ID 11)
X : Type
P : X → ℙ
¬ (∃ x : X, P x) → ∀ x : X, ¬ P x
subgoal 2 (ID 12) is:
(∀ x : X, ¬ P x) → ¬ (∃ x : X, P x)
U:%%- *goals* All L6 (Coq Goals +3)
U:**- illustrative_examples.v Top L5 (Coq Script(2-) +2 yas hs Outl
U:%%- *response* All L1 (Coq Response Wrap)

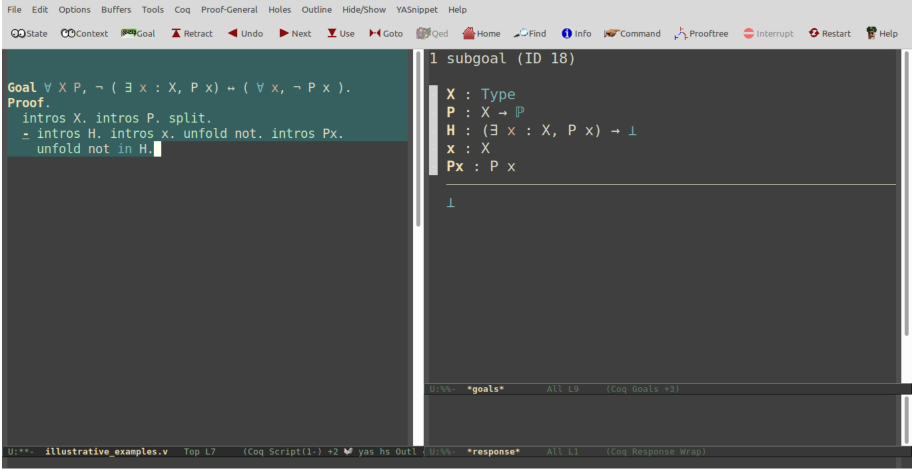

File Edit Options Buffers Tools Coq Proof-General Holes Outline Hide/Show YASnippet Help
State Context Goal Retract Undo Next Use Goto Qed Home Find Info Command Prooftree Interrupt Restart Help
Goal ∀ X P, ¬ (∃ x : X, P x) ↔ (∀ x, ¬ P x).
Proof.
intros X. intros P. split.
-
1 subgoal (ID 11)
X : Type
P : X → ℙ
¬ (∃ x : X, P x) → ∀ x : X, ¬ P x
U:%%- *goals* All L6 (Coq Goals +3)
U:**- illustrative_examples.v Top L6 (Coq Script(1-) +2 yas hs Outl
U:%%- *response* All L1 (Coq Response Wrap)

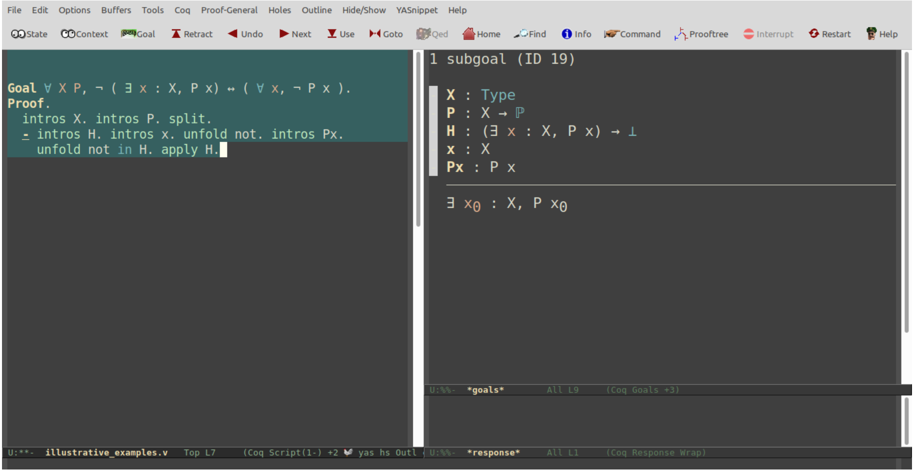

File Edit Options Buffers Tools Coq Proof-General Holes Outline Hide/Show YASnippet Help
State Context Goal Retract Undo Next Use Goto Qed Home Find Info Command Prooftree Interrupt Restart Help
Goal ∀ X P, ¬ (∃ x : X, P x) ↔ (∀ x, ¬ P x).
Proof.
intros X. intros P. split.
- intros H.
1 subgoal (ID 13)
X : Type
P : X → ℙ
H : ¬ (∃ x : X, P x)
∀ x : X, ¬ P x
U:%%- *goals* All L7 (Coq Goals +3)
U:**- illustrative_examples.v Top L6 (Coq Script(1-) +2 yas hs Outl
U:%%- *response* All L1 (Coq Response Wrap)

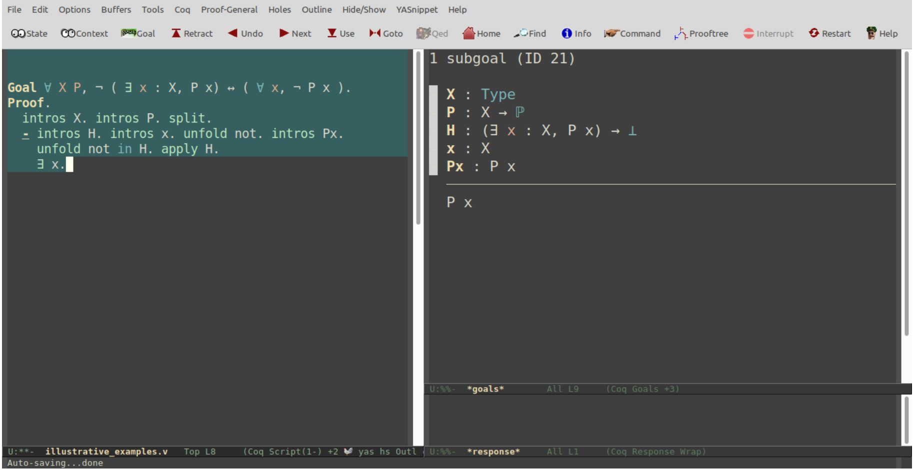

File Edit Options Buffers Tools Coq Proof-General Holes Outline Hide/Show YASnippet Help
State Context Goal Retract Undo Next Use Goto Qed Home Find Info Command Prooftree Interrupt Restart Help
Goal ∀ X P, ¬ (∃ x : X, P x) ↔ (∀ x, ¬ P x).
Proof.
intros X. intros P. split.
- intros H. intros x.
1 subgoal (ID 14)
X : Type
P : X → ℙ
H : ¬ (∃ x : X, P x)
x : X
¬ P x
U:%%- *goals* All L8 (Coq Goals +3)
U:**- illustrative_examples.v Top L6 (Coq Script(1-) +2 yas hs Outl
U:%%- *response* All L1 (Coq Response Wrap)

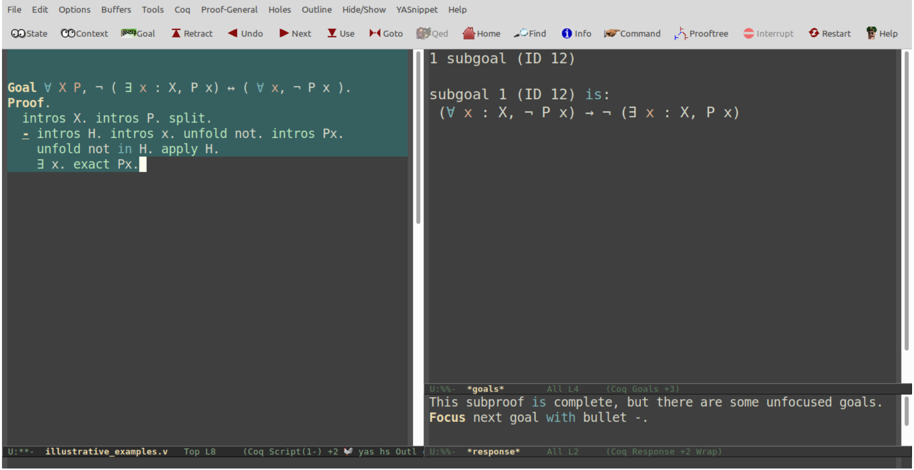

File Edit Options Buffers Tools Coq Proof-General Holes Outline Hide/Show YASnippet Help
State Context Goal Retract Undo Next Use Goto Qed Home Find Info Command Prooftree Interrupt Restart Help
Goal ∀ X P, ¬ (∃ x : X, P x) ↔ (∀ x, ¬ P x).
Proof.
intros X. intros P. split.
- intros H. intros x. unfold not.
1 subgoal (ID 16)
X : Type
P : X → ℙ
H : ¬ (∃ x : X, P x)
x : X
P x → ⊥
U:%%- *goals* All L8 (Coq Goals +3)
U:**- illustrative_examples.v Top L6 (Coq Script(1-) +2 yas hs Outl
U:%%- *response* All L1 (Coq Response Wrap)

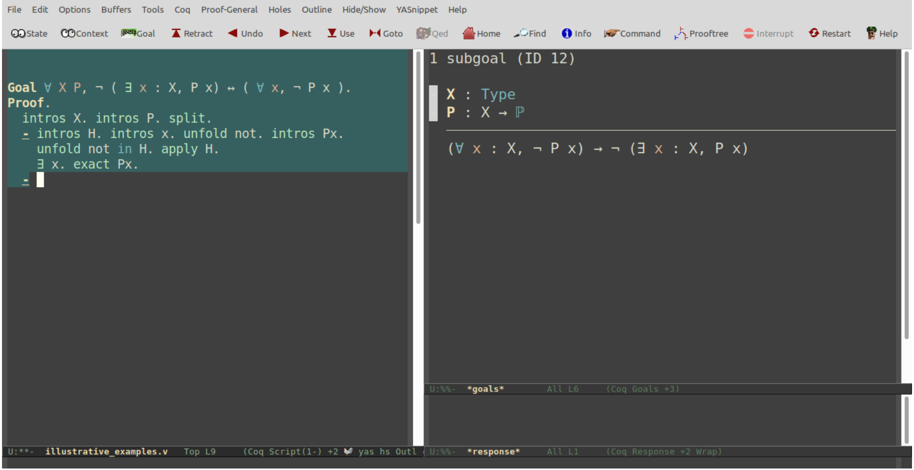

File Edit Options Buffers Tools Coq Proof-General Holes Outline Hide/Show YASnippet Help
State Context Goal Retract Undo Next Use Goto Qed Home Find Info Command Prooftree Interrupt Restart Help
Goal ∀ X P, ¬ (∃ x : X, P x) ↔ (∀ x, ¬ P x).
Proof.
intros X. intros P. split.
- intros H. intros x. unfold not. intros Px.
1 subgoal (ID 17)
X : Type
P : X → ℙ
H : ¬ (∃ x : X, P x)
x : X
Px : P x
⊥
U:%%- *goals* All L9 (Coq Goals +3)
U:**- illustrative_examples.v Top L6 (Coq Script(1-) +2 yas hs Outl
U:%%- *response* All L1 (Coq Response Wrap)

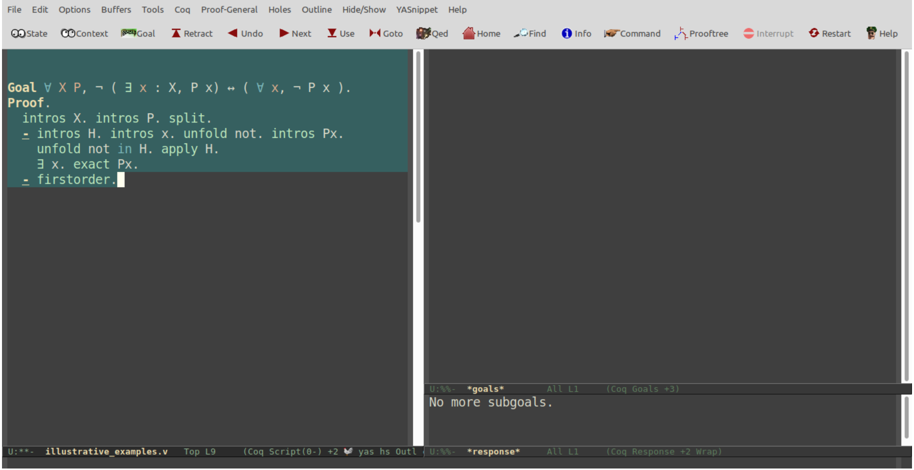

File Edit Options Buffers Tools Coq Proof-General Holes Outline Hide/Show YASnippet Help
State Context Goal Retract Undo Next Use Goto Qed Home Find Info Command Prooftree Interrupt Restart Help
Goal ∀ X P, ¬ (∃ x : X, P x) ↔ (∀ x, ¬ P x).
Proof.
intros X. intros P. split.
- intros H. intros x. unfold not. intros Px.
unfold not in H.
1 subgoal (ID 18)
X : Type
P : X → ℙ
H : (∃ x : X, P x) → ⊥
x : X
Px : P x
⊥
U:%%- *goals* All L9 (Coq Goals +3)
U:**- illustrative_examples.v Top L7 (Coq Script(1-) +2 yas hs Outl
U:%%- *response* All L1 (Coq Response Wrap)

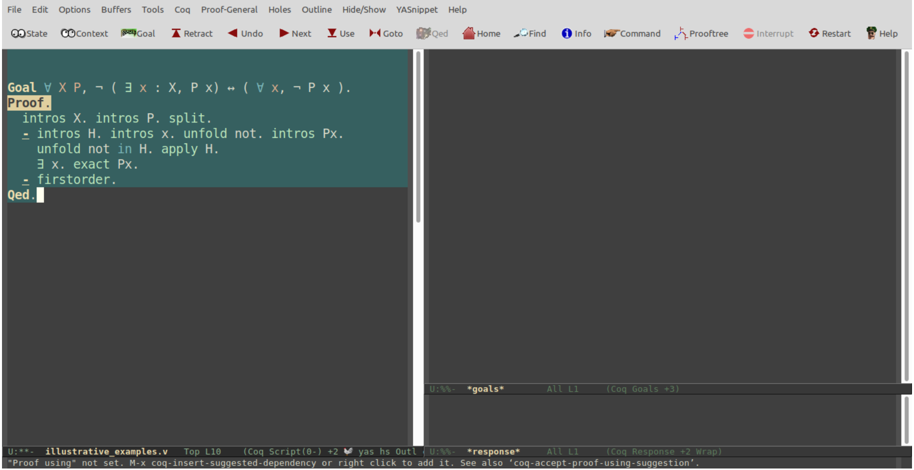

File Edit Options Buffers Tools Coq Proof-General Holes Outline Hide/Show YASnippet Help
State Context Goal Retract Undo Next Use Goto Qed Home Find Info Command Prooftree Interrupt Restart Help
Goal ∀ X P, ¬ (∃ x : X, P x) ↔ (∀ x, ¬ P x).
Proof.
intros X. intros P. split.
- intros H. intros x. unfold not. intros Px.
unfold not in H. apply H.
U:**- illustrative_examples.v Top L7 (Coq Script(1-) +2 yas hs Outl
1 subgoal (ID 19)
X : Type
P : X → ℙ
H : (∃ x : X, P x) → ⊥
x : X
Px : P x
∃ x0 : X, P x0
U:%%- *goals* All L9 (Coq Goals +3)
U:%%- *response* All L1 (Coq Response Wrap)

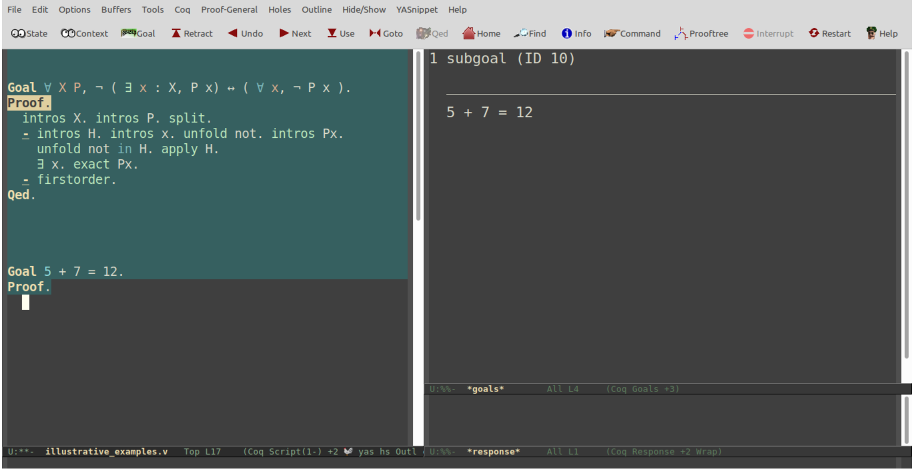

File Edit Options Buffers Tools Coq Proof-General Holes Outline Hide/Show YASnippet Help
State Context Goal Retract Undo Next Use Goto Qed Home Find Info Command Prooftree Interrupt Restart Help
Goal ∀ X P, ¬ (∃ x : X, P x) ↔ (∀ x, ¬ P x).
Proof.
intros X. intros P. split.
- intros H. intros x. unfold not. intros Px.
unfold not in H. apply H.
∃ x.
1 subgoal (ID 21)
X : Type
P : X → ℙ
H : (∃ x : X, P x) → ⊥
x : X
Px : P x
P x
U:%%- *goals* All L9 (Coq Goals +3)
U:**- illustrative_examples.v Top L8 (Coq Script(1-) +2 yas hs Outl
U:%%- *response* All L1 (Coq Response Wrap)
Auto-saving...done

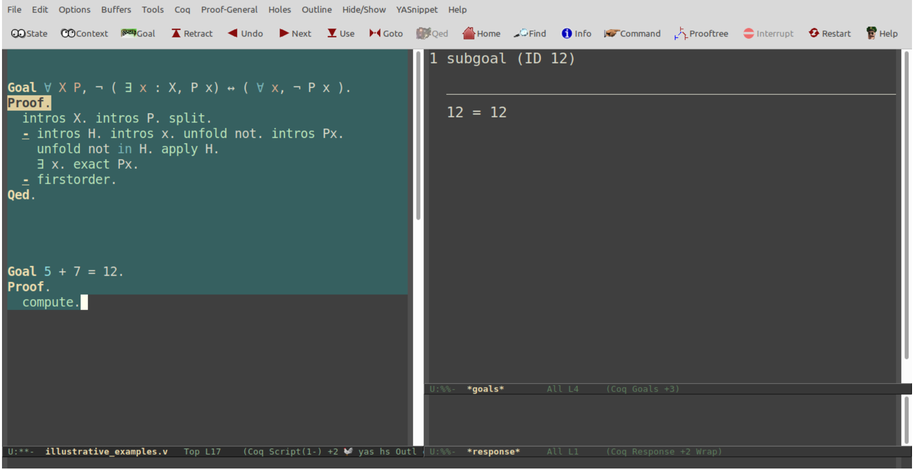

File Edit Options Buffers Tools Coq Proof-General Holes Outline Hide/Show YASnippet Help
State Context Goal Retract Undo Next Use Goto Qed Home Find Info Command Prooftree Interrupt Restart Help
Goal ∀ X P, ¬ (∃ x : X, P x) ↔ (∀ x, ¬ P x).
Proof.
intros X. intros P. split.
- intros H. intros x. unfold not. intros Px.
unfold not in H. apply H.
∃ x. exact Px.
U:**- illustrative_examples.v Top L8 (Coq Script(1-) +2 yas hs Outl
1 subgoal (ID 12)
subgoal 1 (ID 12) is:
(∀ x : X, ¬ P x) → ¬ (∃ x : X, P x)
U:%%- *goals* All L4 (Coq Goals +3)
This subproof is complete, but there are some unfocused goals.
Focus next goal with bullet -.
U:%%- *response* All L2 (Coq Response +2 Wrap)

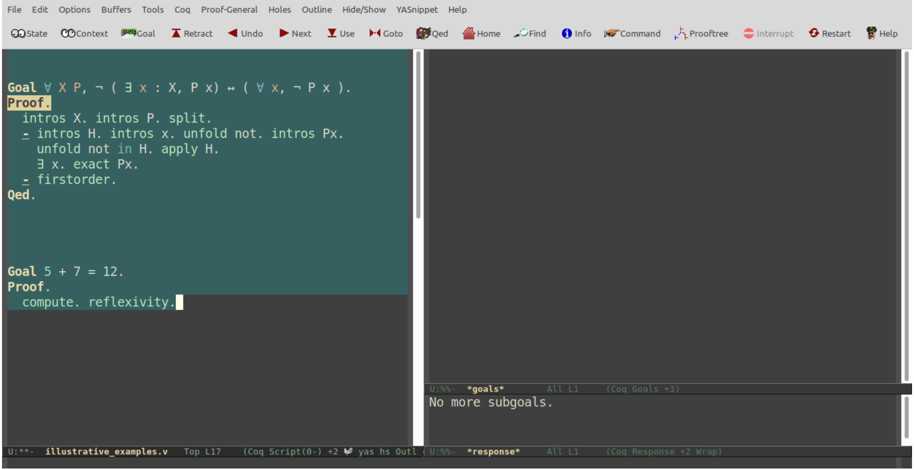

File Edit Options Buffers Tools Coq Proof-General Holes Outline Hide/Show YASnippet Help
State Context Goal Retract Undo Next Use Goto Qed Home Find Info Command Prooftree Interrupt Restart Help
Goal ∀ X P, ¬ (∃ x : X, P x) ↔ (∀ x, ¬ P x).
Proof.
intros X. intros P. split.
- intros H. intros x. unfold not. intros Px.
unfold not in H. apply H.
∃ x. exact Px.
-
1 subgoal (ID 12)
X : Type
P : X → ℙ
(∀ x : X, ¬ P x) → ¬ (∃ x : X, P x)
U:%%- *goals* All L6 (Coq Goals +3)
U:**- illustrative_examples.v Top L9 (Coq Script(1-) +2 yas hs Outl
U:%%- *response* All L1 (Coq Response +2 Wrap)

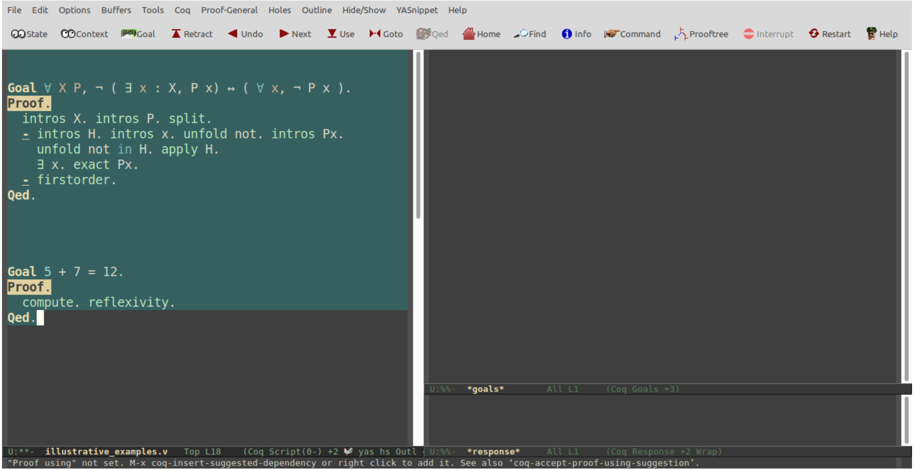

File Edit Options Buffers Tools Coq Proof-General Holes Outline Hide/Show YASnippet Help
State Context Goal Retract Undo Next Use Goto Qed Home Find Info Command Prooftree Interrupt Restart Help
Goal ∀ X P, ¬ (∃ x : X, P x) ↔ (∀ x, ¬ P x).
Proof.
intros X. intros P. split.
- intros H. intros x. unfold not. intros Px.
unfold not in H. apply H.
∃ x. exact Px.
- firstorder.
No more subgoals.
U:**- illustrative_examples.v Top L9 (Coq Script(0-) +2 yas hs Outl
U:%%- *goals* All L1 (Coq Goals +3)
U:%%- *response* All L1 (Coq Response +2 Wrap)


File Edit Options Buffers Tools Coq Proof-General Holes Outline Hide/Show YASnippet Help
State Context Goal Retract Undo Next Use Goto Qed Home Find Info Command Prooftree Interrupt Restart Help
Goal ∀ X P, ¬ (∃ x : X, P x) ↔ (∀ x, ¬ P x).
Proof.
intros X. intros P. split.
- intros H. intros x. unfold not. intros Px.
unfold not in H. apply H.
∃ x. exact Px.
- firstorder.
Qed.
U:%%- *goals* All L1 (Coq Goals +3)
U:**- illustrative_examples.v Top L10 (Coq Script(0-) +2 yas hs Outl
U:%%- *response* All L1 (Coq Response +2 Wrap)
"Proof using" not set. M-x coq-insert-suggested-dependency or right click to add it. See also ‘coq-accept-proof-using-suggestion’.


File Edit Options Buffers Tools Coq Proof-General Holes Outline Hide/Show YASnippet Help
State Context Goal Retract Undo Next Use Goto Qed Home Find Info Command Prooftree Interrupt Restart Help
Goal ∀ X P, ¬ (∃ x : X, P x) ↔ (∀ x, ¬ P x).
Proof.
intros X. intros P. split.
- intros H. intros x. unfold not. intros Px.
unfold not in H. apply H.
∃ x. exact Px.
- firstorder.
Qed.
Goal 5 + 7 = 12.
Proof.
1 subgoal (ID 10)
5 + 7 = 12
U:%%- *goals* All L4 (Coq Goals +3)
U:**- illustrative_examples.v Top L17 (Coq Script(1-) +2 yas hs Outl
U:%%- *response* All L1 (Coq Response +2 Wrap)


File Edit Options Buffers Tools Coq Proof-General Holes Outline Hide/Show YASnippet Help
State Context Goal Retract Undo Next Use Goto Qed Home Find Info Command Prooftree Interrupt Restart Help
Goal ∀ X P, ¬ (∃ x : X, P x) ↔ (∀ x, ¬ P x).
Proof.
intros X. intros P. split.
- intros H. intros x. unfold not. intros Px.
unfold not in H. apply H.
∃ x. exact Px.
- firstorder.
Qed.
Goal 5 + 7 = 12.
Proof.
compute.
1 subgoal (ID 12)
12 = 12
U:%%- *goals* All L4 (Coq Goals +3)
U:**- illustrative_examples.v Top L17 (Coq Script(1-) +2 yas hs Outl
U:%%- *response* All L1 (Coq Response +2 Wrap)


File Edit Options Buffers Tools Coq Proof-General Holes Outline Hide/Show YASnippet Help
State Context Goal Retract Undo Next Use Goto Qed Home Find Info Command Prooftree Interrupt Restart Help
Goal ∀ X P, ¬ (∃ x : X, P x) ↔ (∀ x, ¬ P x).
Proof.
intros X. intros P. split.
- intros H. intros x. unfold not. intros Px.
unfold not in H. apply H.
∃ x. exact Px.
- firstorder.
Qed.
Goal 5 + 7 = 12.
Proof.
compute. reflexivity.
U:**- illustrative_examples.v Top L17 (Coq Script(0-) +2 yas hs Outl
U:%%- *goals* All L1 (Coq Goals +3)
No more subgoals.
U:%%- *response* All L1 (Coq Response +2 Wrap)


File Edit Options Buffers Tools Coq Proof-General Holes Outline Hide/Show YASnippet Help
State Context Goal Retract Undo Next Use Goto Qed Home Find Info Command Prooftree Interrupt Restart Help
Goal ∀ X P, ¬ (∃ x : X, P x) ↔ (∀ x, ¬ P x).
Proof.
intros X. intros P. split.
- intros H. intros x. unfold not. intros Px.
unfold not in H. apply H.
∃ x. exact Px.
- firstorder.
Qed.
Goal 5 + 7 = 12.
Proof.
compute. reflexivity.
Qed.
U:%%- *goals* All L1 (Coq Goals +3)
U:**- illustrative_examples.v Top L18 (Coq Script(0-) +2 yas hs Outl
U:%%- *response* All L1 (Coq Response +2 Wrap)
"Proof using" not set. M-x coq-insert-suggested-dependency or right click to add it. See also 'coq-accept-proof-using-suggestion'.

Undecidability along Reductions

Undecidable Predicate (informally)

A predicate which has no algorithmic decision procedure.

Undecidability along Reductions

Undecidable Predicate (informally)

A predicate which has no algorithmic decision procedure.

Let α be some **undecidable** predicate on a type A and β a predicate on B. If we have a **computable** function $f : A \to B$ with

$$\forall x : A.\ \ \alpha(x) \ \leftrightarrow \ \beta(f(x))$$

then β is also undecidable.

Undecidability along Reductions

Undecidable Predicate (informally)

A predicate which has no algorithmic decision procedure.

Let α be some **undecidable** predicate on a type A and β a predicate on B. If we have a **computable** function $f : A \to B$ with

$$\forall x : A.\ \ \alpha(x) \ \leftrightarrow \ \beta(f(x))$$

then β is also undecidable.

Intuition

β decidable by algorithm and f computable $\to$ $(\beta \circ f \leftrightarrow \alpha)$ decidable. $\lightning$

Reductions

Definition

Let α be some predicate on a type A and β a predicate on B. Then we call $f : A \to B$ a reduction from α to β iff

$$\forall x : A.\ \ \alpha(x) \ \leftrightarrow \ \beta(f(x))$$

and f is **computable**.

Reductions

Definition

Let α be some predicate on a type A and β a predicate on B. Then we call $f : A \to B$ a reduction from α to β iff

$$\forall x : A.\ \ \alpha(x) \ \leftrightarrow \ \beta(f(x))$$

~~and f is computable~~.

Reductions

Definition

Let α be some predicate on a type A and β a predicate on B. Then we call $f : A \to B$ a reduction from α to β iff

$$\forall x : A.\ \ \alpha(x) \ \leftrightarrow\ \beta(f(x))$$

~~and f is computable~~.

The above gives a synthetic notion for reductions, which is justified by noting that from the outside we can recognise:

Reductions

Definition

Let α be some predicate on a type A and β a predicate on B. Then we call $f : A \to B$ a reduction from α to β iff

$$\forall x : A.\ \ \alpha(x) \ \leftrightarrow \ \beta(f(x))$$

~~and f is computable~~.

The above gives a synthetic notion for reductions, which is justified by noting that from the outside we can recognise:

- Coq's internal logic is constructive

Reductions

Definition

Let α be some predicate on a type A and β a predicate on B. Then we call $f : A \to B$ a reduction from α to β iff

$$\forall x : A.\ \ \alpha(x)\ \leftrightarrow\ \beta(f(x))$$

~~and f is computable~~.

The above gives a synthetic notion for reductions, which is justified by noting that from the outside we can recognise:

- Coq's internal logic is constructive
- Every function definable in Coq is computable

Reductions

Relevant for us: What are A, B, α, β and f in our case?

$$f : A \to B \quad \text{s.t.} \quad \forall x.\ \ \alpha(x) \ \leftrightarrow \ \beta(f(x))$$

Fragment FA of Peano Arithmetic

The first-order theory of PA has the following symbols:

Function Symbols : $0 \quad S \quad + \quad \cdot$ **Predicate Symbols :** $\equiv$

Logical Symbols : $\bot \quad \land \quad \lor \quad \rightarrow$ **Quantifiers :** $\forall \quad \exists$

$$f : A \to B \quad \text{s.t.} \quad \forall x.\ \alpha(x) \leftrightarrow \beta(f(x))$$

Fragment FA of Peano Arithmetic

The first-order theory of PA has the following symbols:

Function Symbols : $0 \quad S \quad + \quad \cdot$ **Predicate Symbols :** $\equiv$

Logical Symbols : $\bot \quad \wedge \quad \vee \quad \rightarrow$ **Quantifiers :** $\forall \quad \exists$

We don't assume all axioms, but only the following fragment

$$
\begin{aligned}
\textbf{Zero addition} &: \forall x.\ 0 + x \equiv x \\
\textbf{Recursion for addition} &: \forall xy.\ (Sx) + y \equiv S(x + y) \\
\textbf{Zero multiplication} &: \forall x.\ 0 \cdot x \equiv 0 \\
\textbf{Recursion for multiplication} &: \forall xy.\ (Sx) \cdot y \equiv y + x \cdot y
\end{aligned}
$$

$$f : A \rightarrow B \quad \text{s.t.} \quad \forall x.\ \ \alpha(x) \ \leftrightarrow \ \beta(f(x))$$

Fragment FA of Peano Arithmetic

The first-order theory of PA has the following symbols:

Function Symbols : $0 \quad S \quad + \quad \cdot$ **Predicate Symbols :** $\equiv$

Logical Symbols : $\perp \quad \wedge \quad \vee \quad \rightarrow$ **Quantifiers :** $\forall \quad \exists$

We don't assume all axioms, but only the following fragment

Zero addition : $\forall x.\ 0 + x \equiv x$

Recursion for addition : $\forall xy.\ (Sx) + y \equiv S(x + y)$

Zero multiplication : $\forall x.\ 0 \cdot x \equiv 0$

Recursion for multiplication : $\forall xy.\ (Sx) \cdot y \equiv y + x \cdot y$

$f : A \rightarrow$ FA formulas s.t. $\forall x.\ \alpha(x) \leftrightarrow \beta(f(x))$

Diophantine constraints

We define expressions containing variables, we call

atomic equations

$$x_i = 1 \quad | \quad x_i + x_j = x_k \quad | \quad x_i \cdot x_j = x_k$$

Diophantine constraints

We define expressions containing variables, we call

atomic equations

$$x_i = 1 \quad | \quad x_i + x_j = x_k \quad | \quad x_i \cdot x_j = x_k$$

And evaluations of these expressions for given $\sigma : \mathbb{N} \to \mathbb{N}$

- $[x_i + x_j = x_k]_\sigma := \ \sigma(i) + \sigma(j) = \sigma(k)$

Diophantine constraints

We define expressions containing variables, we call

atomic equations

$$x_i = 1 \quad | \quad x_i + x_j = x_k \quad | \quad x_i \cdot x_j = x_k$$

And evaluations of these expressions for given $\sigma : \mathbb{N} \to \mathbb{N}$

- $[x_i = 1]_\sigma := \ \sigma(i) = 1$
- $[x_i + x_j = x_k]_\sigma := \ \sigma(i) + \sigma(j) = \sigma(k)$
- $[x_i \cdot x_j = x_k]_\sigma := \ \sigma(i) \cdot \sigma(j) = \sigma(k)$

Diophantine constraints

We define expressions containing variables, we call

atomic equations

$$x_i = 1 \quad | \quad x_i + x_j = x_k \quad | \quad x_i \cdot x_j = x_k$$

And evaluations of these expressions for given $\sigma : \mathbb{N} \to \mathbb{N}$

- $[x_i = 1]_\sigma := \ \sigma(i) = 1$
- $[x_i + x_j = x_k]_\sigma := \ \sigma(i) + \sigma(j) = \sigma(k)$
- $[x_i \cdot x_j = x_k]_\sigma := \ \sigma(i) \cdot \sigma(j) = \sigma(k)$

We call a list $L = [e_1, \ldots, e_n]$ of atomic equations e_j a H10 problem and extend $[\ \ ]_\sigma$ to problems by $[L]_\sigma := [e_1]_\sigma \land \ldots \land [e_n]_\sigma$.

Satisfiabiliy of diophantine constraints

$$f : A \to \text{FA formulas} \quad \text{s.t.} \quad \forall x.\ \ \alpha(x) \ \leftrightarrow \ \beta(f(x))$$

Satisfiabiliy of diophantine constraints

Given a H10 problem L, we can now ask the question:

Satisfiability

Can L be satisfied? $\leftrightarrow$ Can we show $\exists\sigma.\ [L]_\sigma$?

$$f : A \to \text{FA formulas} \quad \text{s.t.} \quad \forall x.\ \ \alpha(x) \ \leftrightarrow\ \beta(f(x))$$

Satisfiabiliy of diophantine constraints

Given a H10 problem L, we can now ask the question:

Satisfiability

Can L be satisfied? $\leftrightarrow$ Can we show $\exists\sigma.\ [L]_\sigma$?

This question is equivalent to asking if some diophantine equation has a solution. The latter is known to be **undecidable** [Matijasevič, 1970] [Larchey-Wendling and Forster, 2019].

$$f : A \to \text{FA formulas} \quad \text{s.t.} \quad \forall x.\ \ \alpha(x) \ \leftrightarrow\ \beta(f(x))$$

Satisfiabiliy of diophantine constraints

Given a H10 problem L, we can now ask the question:

Satisfiability

Can L be satisfied? $\leftrightarrow$ Can we show $\exists\sigma.\ [L]_\sigma$?

This question is equivalent to asking if some diophantine equation has a solution. The latter is known to be **undecidable** [Matijasevič, 1970] [Larchey-Wendling and Forster, 2019].

$$f : \text{H10 problems} \to \text{FA formulas} \quad \text{s.t.} \quad \forall L.\ \ \text{sat}(L) \ \leftrightarrow \ \beta(f(L))$$

Embedding H10 problems into FA

f : H10 problems $\rightarrow$ FA formulas s.t. $\forall L.\ \text{sat}(L) \leftrightarrow \beta(f(L))$

Embedding H10 problems into FA

Let's look at the following example of an H10 problem

$$L = [x + x = y \ , \ y \cdot y = x]$$

We want to send this to a formula in FA which intuitively expresses the satisfiability of L.

$$f : \text{H10 problems} \to \text{FA formulas} \quad \text{s.t.} \quad \forall L. \ \ \text{sat}(L) \ \leftrightarrow \ \beta(f(L))$$

Embedding H10 problems into FA

Let's look at the following example of an H10 problem

$$L = [x + x = y \;,\; y \cdot y = x]$$

We want to send this to a formula in FA which intuitively expresses the satisfiability of L.

The choice is canonical:

$$\exists x \, \exists y \quad x + x \equiv y \;\wedge\; y \cdot y \equiv x$$

$$f : \text{H10 problems} \to \text{FA formulas} \quad \text{s.t.} \quad \forall L.\;\; \text{sat}(L) \;\leftrightarrow\; \beta(f(L))$$

Embedding H10 problems into FA

Let's look at the following example of an H10 problem

$$L = [x + x = y \ , \ y \cdot y = x]$$

We want to send this to a formula in FA which intuitively expresses the satisfiability of L.

The choice is canonical:

$$\exists x \ \exists y \quad \underbrace{x + x \equiv y \ \wedge \ y \cdot y \equiv x}_{\varepsilon^*(L)}$$

$$f : \text{H10 problems} \to \text{FA formulas} \quad \text{s.t.} \quad \forall L. \ \ \mathrm{sat}(L) \ \leftrightarrow \ \beta(f(L))$$

Embedding H10 problems into FA

Let's look at the following example of an H10 problem

$$L = [x + x = y\ ,\ y \cdot y = x]$$

We want to send this to a formula in FA which intuitively expresses the satisfiability of L.

The choice is canonical:

$$\underbrace{\exists x\ \exists y \quad \underbrace{x + x \equiv y\ \wedge\ y \cdot y \equiv x}_{\varepsilon^*(L)}}_{\varepsilon(L)}$$

$$f : \text{H10 problems} \to \text{FA formulas} \quad \text{s.t.} \quad \forall L.\ \ \text{sat}(L)\ \leftrightarrow\ \beta(f(L))$$

Embedding H10 problems into FA

Let's look at the following example of an H10 problem

$$L = [x + x = y \ , \ y \cdot y = x]$$

We want to send this to a formula in FA which intuitively expresses the satisfiability of L.

The choice is canonical:

$$\underbrace{\exists x \ \exists y \quad \underbrace{x + x \equiv y \ \wedge \ y \cdot y \equiv x}_{\varepsilon^*(L)}}_{\varepsilon(L)}$$

$$\varepsilon : \text{H10 problems} \to \text{FA formulas} \quad \text{s.t.} \quad \forall L. \ \ \mathrm{sat}(L) \ \leftrightarrow \ \beta(\varepsilon(L))$$

Tarski Semantics

We can interpret sentences from our first-order language of arithmetic in the standard model $(\mathbb{N}, 0, S, +, \cdot)$.

Tarski Semantics

We can interpret sentences from our first-order language of arithmetic in the standard model $(\mathbb{N}, 0, S, +, \cdot)$.

Given an environment $\rho : \mathbb{N} \to \mathbb{N}$ we can evaluate terms. We can then use this to define truth of formulas φ in $\mathbb{N}$, for which we write $\mathbb{N} \vDash \varphi$.

Tarski Semantics

We can interpret sentences from our first-order language of arithmetic in the standard model $(\mathbb{N}, 0, S, +, \cdot)$.

Given an environment $\rho : \mathbb{N} \to \mathbb{N}$ we can evaluate terms. We can then use this to define truth of formulas φ in $\mathbb{N}$, for which we write $\mathbb{N} \vDash \varphi$.

Examples

- $\mathbb{N} \vDash (x_1 + x_2 \equiv x_3) \;=\; \forall \rho.\;\; \rho(1) + \rho(2) = \rho(3)$
- $\mathbb{N} \vDash (\forall x.\, 0 + x \equiv x) \;=\; \forall n : \mathbb{N}.\, 0 + n = n$

Tarski Semantics

If we replace $\mathbb{N}$ with some other domain D providing

- $\mathbb{O} : D$
- $\mathbb{S} : D \to D$
- $\oplus : D \times D \to D$
- $\otimes : D \times D \to D$

we get the more general notion of a model $(D, \mathbb{O}, \mathbb{S}, \oplus, \otimes)$ for arithmetic.

ε : H10 problems $\to$ FA formulas s.t. $\forall L.$ sat$(L) \leftrightarrow \beta(\varepsilon(L))$

Tarski Semantics

If we replace $\mathbb{N}$ with some other domain D providing

- $\mathbb{O} : D$
- $\mathbb{S} : D \to D$
- $\oplus : D \times D \to D$
- $\otimes : D \times D \to D$

we get the more general notion of a model $(D, \mathbb{O}, \mathbb{S}, \oplus, \otimes)$ for arithmetic.

Example

$$D \vDash (\forall x.\ 0 + x \equiv 0) \ = \ \forall d : D.\ \mathbb{O} \oplus d = d$$

$$\varepsilon : \text{H10 problems} \to \text{FA formulas} \quad \text{s.t.} \quad \forall L.\ \ \text{sat}(L) \ \leftrightarrow \ \beta(\varepsilon(L))$$

Tarski Semantics

If we replace $\mathbb{N}$ with some other domain D providing

- $\mathbb{O} : D$
- $\mathbb{S} : D \to D$
- $\oplus : D \times D \to D$
- $\otimes : D \times D \to D$

we get the more general notion of a model $(D, \mathbb{O}, \mathbb{S}, \oplus, \otimes)$ for arithmetic.

Example

$$D \vDash (\forall x.\ 0 + x \equiv 0) \ = \ \forall d : D.\ \mathbb{O} \oplus d = d$$

We call φ valid in FA and write FA $\vDash \varphi$ iff

$$\forall D \text{ model of FA} \quad \forall \rho.\ \ D \vDash_\rho \varphi$$

$$\varepsilon : \text{H10 problems} \to \text{FA formulas} \quad \text{s.t.} \quad \forall L.\ \ \text{sat}(L) \ \leftrightarrow \ \beta(\varepsilon(L))$$

Tarski Semantics

If we replace $\mathbb{N}$ with some other domain D providing

- $\mathbb{O} : D$
- $\mathbb{S} : D \to D$
- $\oplus : D \times D \to D$
- $\otimes : D \times D \to D$

we get the more general notion of a model $(D, \mathbb{O}, \mathbb{S}, \oplus, \otimes)$ for arithmetic.

Example

$$D \vDash (\forall x.\ 0 + x \equiv 0) \ = \ \forall d : D.\ \mathbb{O} \oplus d = d$$

We call φ valid in FA and write FA $\vDash \varphi$ iff

$$\forall D \text{ model of FA} \quad \forall \rho.\ \ D \vDash_\rho \varphi$$

$$\varepsilon : \text{H10 problems} \to \text{FA formulas} \quad \text{s.t.} \quad \forall L.\ \ \mathsf{sat}(L) \ \leftrightarrow \ \text{FA} \vDash \varepsilon(L)$$

Canonical Model Homomorphism

If we have some FA model D, we can recursively define a function $\nu : \mathbb{N} \to D$ by

Definition

$$\nu(0) := \mathbb{O} \quad , \quad \nu(x+1) := \nu(x) \oplus \mathbb{S}\ \mathbb{O}$$

Giving us an embedding of $\mathbb{N}$ into any FA model.

Canonical Model Homomorphism

If we have some FA model D, we can recursively define a function $\nu : \mathbb{N} \to D$ by

Definition

$$\nu(0) := \mathbb{O} \quad , \quad \nu(x+1) := \nu(x) \oplus \mathbb{S}\,\mathbb{O}$$

Giving us an embedding of $\mathbb{N}$ into any FA model.

By induction over $x : \mathbb{N}$ we can show that ν is a homomorphism:

Homorphism Lemma

$$\nu(x+y) = \nu(x) \oplus \nu(y) \qquad \nu(x \cdot y) = \nu(x) \otimes \nu(y)$$

Canonical Model Homomorphism

If we have some FA model D, we can recursively define a function $\nu : \mathbb{N} \to D$ by

Definition

$$\nu(0) := \mathbb{O} \quad , \quad \nu(x+1) := \nu(x) \oplus \mathbb{S}\ \mathbb{O}$$

Giving us an embedding of $\mathbb{N}$ into any FA model.

By induction over $x : \mathbb{N}$ we can show that ν is a homomorphism:

Homorphism Lemma

$$\nu(x+y) = \nu(x) \oplus \nu(y) \qquad \qquad \nu(x \cdot y) = \nu(x) \otimes \nu(y)$$

For the proof of these equations we need the axioms we assumed for FA.

Verification of Reduction

Verification of Reduction

To verify the reduction, we now need to show

Theorem

$$\forall L.\ \mathsf{sat}(L) \ \leftrightarrow\ \mathsf{FA} \vDash \varepsilon(L)$$

Verification of Reduction

To verify the reduction, we now need to show

Theorem

$$\forall L.\ \mathsf{sat}(L) \ \leftrightarrow \ \mathsf{FA} \vDash \varepsilon(L)$$

Proof.

$\leftarrow$ We use that $\mathbb{N} \vDash \exists^N \varepsilon^*(L)$. Providing us N elements in $\mathbb{N}$ that give us a solution for L.

Verification of Reduction

To verify the reduction, we now need to show

Theorem

$$\forall L.\ \mathsf{sat}(L) \ \leftrightarrow\ \mathsf{FA} \vDash \varepsilon(L)$$

Proof.

$\leftarrow$ We use that $\mathbb{N} \vDash \exists^N \varepsilon^*(L)$. Providing us N elements in $\mathbb{N}$ that give us a solution for L.

$\rightarrow$ By $\mathsf{sat}(L)$ we have a solution σ for L, which we can transport to any model D via the homomorphism ν.

Verification of Reduction

To verify the reduction, we now need to show

Theorem

$$\forall L.\ \mathsf{sat}(L) \ \leftrightarrow\ \mathsf{FA} \vDash \varepsilon(L)$$

Proof.

$\leftarrow$ We use that $\mathbb{N} \vDash \exists^N \varepsilon^*(L)$. Providing us N elements in $\mathbb{N}$ that give us a solution for L.

$\rightarrow$ By $\mathsf{sat}(L)$ we have a solution σ for L, which we can transport to any model D via the homomorphism ν. $\square$

Closing Remarks

Since the proof works for the fragment FA, it also works for PA. This was very easy to check with Coq.

Closing Remarks

Since the proof works for the fragment FA, it also works for PA. This was very easy to check with Coq.

Advantages of working with Coq

- Definitions can easily be modified; broken proofs will be pointed out

Closing Remarks

Since the proof works for the fragment FA, it also works for PA. This was very easy to check with Coq.

Advantages of working with Coq

- Definitions can easily be modified; broken proofs will be pointed out
- Admitting proof goals

Closing Remarks

Since the proof works for the fragment FA, it also works for PA. This was very easy to check with Coq.

Advantages of working with Coq

- Definitions can easily be modified; broken proofs will be pointed out
- Admitting proof goals
- Looking up definitions is a matter of seconds

Closing Remarks

Since the proof works for the fragment FA, it also works for PA. This was very easy to check with Coq.

Advantages of working with Coq

- Definitions can easily be modified; broken proofs will be pointed out
- Admitting proof goals
- Looking up definitions is a matter of seconds
- Standard library with many theorems

Closing Remarks

Since the proof works for the fragment FA, it also works for PA. This was very easy to check with Coq.

Advantages of working with Coq

- Definitions can easily be modified; broken proofs will be pointed out
- Admitting proof goals
- Looking up definitions is a matter of seconds
- Standard library with many theorems
- Book-keeping

Closing Remarks

Since the proof works for the fragment FA, it also works for PA. This was very easy to check with Coq.

Advantages of working with Coq

- Definitions can easily be modified; broken proofs will be pointed out
- Admitting proof goals
- Looking up definitions is a matter of seconds
- Standard library with many theorems
- Book-keeping
- Automation

Closing Remarks

Since the proof works for the fragment FA, it also works for PA. This was very easy to check with Coq.

Disadvantages

Closing Remarks

Since the proof works for the fragment FA, it also works for PA. This was very easy to check with Coq.

Disadvantages

- Seemingly trivial things can become hard

Closing Remarks

Since the proof works for the fragment FA, it also works for PA. This was very easy to check with Coq.

Disadvantages

- ~~Seemingly~~ trivial things can become hard

Closing Remarks

Since the proof works for the fragment FA, it also works for PA. This was very easy to check with Coq.

Disadvantages

- ~~Seemingly~~ trivial things can become hard
-

Closing Remarks

Since the proof works for the fragment FA, it also works for PA. This was very easy to check with Coq.

Disadvantages

- ~~Seemingly~~ trivial things can become hard
-
-

Closing Remarks

Since the proof works for the fragment FA, it also works for PA. This was very easy to check with Coq.

Disadvantages

- ~~Seemingly~~ trivial things can become hard
-
-
-

Closing Remarks

Since the proof works for the fragment FA, it also works for PA. This was very easy to check with Coq.

Disadvantages

- ~~Seemingly~~ trivial things can become hard
-
-
-
- Nothing else came to my mind

More Work...

I did

- Some results on finite PA models.
- Failed Attempt of an undecidability proof.

More Work...

I did

- Some results on finite PA models.
- Failed Attempt of an undecidability proof.

In progress right now

- replacing FA $\vDash$ by FA $\vdash$

More Work...

I did

- Some results on finite PA models.
- Failed Attempt of an undecidability proof.

In progress right now

- replacing FA $\vDash$ by FA $\vdash$

Possible next goals

- Tennenbaum's Theorem
- Self-verifying Theories
- Getting PA $\vdash \varphi$ from $\mathbb{N} \vDash \varphi$

More Work...

I did

- Some results on finite PA models.
- Failed Attempt of an undecidability proof.

In progress right now

- replacing FA $\vDash$ by FA $\vdash$

Possible next goals

- Tennenbaum's Theorem
- Self-verifying Theories
- Getting PA $\vdash \varphi$ from $\mathbb{N} \vDash \varphi$

Thank you for your attention!

Bibliography

Gonthier, G. (2008).
Formal proof–the four-color theorem.
Notices of the AMS, 55(11):1382–1393.

Gonthier, G., Asperti, A., Avigad, J., Bertot, Y., Cohen, C., Garillot, F., Le Roux, S., Mahboubi, A., O'Connor, R., Biha, S. O., et al. (2013).
A machine-checked proof of the odd order theorem.
In *International Conference on Interactive Theorem Proving*, pages 163–179. Springer.

Larchey-Wendling, D. and Forster, Y. (2019).
Hilbert's Tenth Problem in Coq.
In Geuvers, H., editor, *4th International Conference on Formal Structures for Computation and Deduction (FSCD 2019)*, volume 131 of *LIPIcs*, pages 27:1–27:20.

Leroy, X. et al. (2012).
The compcert verified compiler.

Matijasevič, Y. V. (1970).
Enumerable sets are diophantine.
Soviet Math. Dokl., 11:354–358.

Smith, P. (2013).
An introduction to Gödel's theorems.
Cambridge University Press.

The Coq Proof Assistant (2020).
`http://coq.inria.fr`.