Formalizing and Proving Theorems in Coq — Lecture 4

Tobias Kappé

Today's lecture

We will start with...

- some housekeeping
- a review of inductive proofs on propositions

Today's lecture

We will start with...

- some housekeeping
- a review of inductive proofs on propositions

You will learn about...

- using Coq's `Setoid` library for rewriting tasks.
- advanced topics involving recursion and inductives.

Induction on propositions: a review

Let $\leq$ and $\preceq$ be the smallest relations on $\mathbb{N}$ satisfying, for all $n, m, k \in \mathbb{N}$:

$$\frac{}{n \leq n} \qquad \frac{n \leq m}{n \leq S(m)} \qquad \Big| \qquad \frac{}{n \preceq n} \qquad \frac{}{n \preceq S(n)} \qquad \frac{n \preceq k \qquad k \preceq m}{n \preceq m}$$

Induction on propositions: a review

Let $\leq$ and $\preceq$ be the smallest relations on $\mathbb{N}$ satisfying, for all $n, m, k \in \mathbb{N}$:

$$\frac{}{n \leq n} \qquad \frac{n \leq m}{n \leq S(m)} \qquad \Big| \qquad \frac{}{n \preceq n} \qquad \frac{}{n \preceq S(n)} \qquad \frac{n \preceq k \qquad k \preceq m}{n \preceq m}$$

Claim

For all $n, m \in \mathbb{N}$, if $n \leq m$ then also $n \preceq m$.

Proof.

Suppose we have a *proof* of $n \leq m$. We proceed by induction on the height of the proof tree. If $n \leq m$ by the first rule, then $n = m$, and so $n \preceq m$ by the third rule. If $n \leq m$ by the second rule, then $m = S(m')$ for some m' with $n \leq m'$. By induction, $n \preceq m'$. By the fourth rule, $m' \preceq S(m')$; by the last rule, $n \preceq S(m') = m$. $\square$

Induction on propositions: some reflection

Induction on propositions: some reflection

- We showed how to transform a *proof* of $n \leq m$ into a proof of $n \preceq m$.

Induction on propositions: some reflection

- We showed how to transform a *proof* of $n \leq m$ into a proof of $n \preceq m$.
- This type of strategy really only makes sense in a constructive setting.

Induction on propositions: some reflection

- We showed how to transform a *proof* of $n \leq m$ into a proof of $n \preceq m$.
- This type of strategy really only makes sense in a constructive setting.
- In Coq, *proofs are data*, whose *types are propositions*.

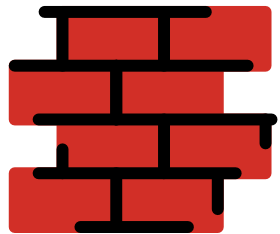
Group terms

Let X be a set of variables.

Group terms over X are built as follows:

- The symbol 0 is a group term.
- If x is a variable, then x is a term.
- If u and v are group terms, then so is $u + v$.
- If u is a group term, then so is the term $-u$.

NB: these are expressions, not numbers!

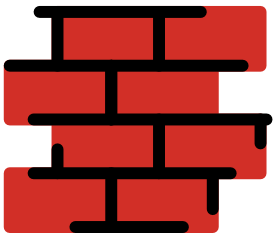


Group terms, continued

Alternatively, group terms over X form the smallest set $\mathbb{G}(X)$ satisfying the following rules:

$$\frac{}{0 \in \mathbb{G}(X)} \qquad \frac{x \in X}{x \in \mathbb{G}(X)}$$

$$\frac{u, v \in \mathbb{G}(X)}{u + v \in \mathbb{G}(X)} \qquad \frac{u \in \mathbb{G}(X)}{-u \in \mathbb{G}(X)}$$



Group equivalence

Group terms are considered "equivalent" modulo

- Associativity of $+$, i.e.,

$$u + (v + w) \equiv (u + v) + w$$

- 0 is a unit *on the right*, i.e., $u + 0 \equiv u$
- $-u$ is the inverse of u *on the right*, i.e.,

$$u + -u \equiv 0$$

- Equivalence is a congruence, i.e.,

$$u \equiv v \wedge u' \equiv v \implies u + u' \equiv v + v'$$

Groups — demonstration

We are going to

1. encode group terms
2. encode group equivalence
3. prove some interesting facts

Groups — reflection

That was not ideal:

- Lots of applications of `GTrans`
- Very deeply nested proofs.
- Hard to get a good overview.

Groups — reflection

That was not ideal:

- Lots of applications of `GTrans`
- Very deeply nested proofs.
- Hard to get a good overview.

We can simplify by using *setoids*

Setoids — reflection

Using `Setoids` we can

- teach Coq about properties of relations
- use those properties through `rewrite`
- encode pen-and-paper proofs more nicely.

Propositional Dynamic Logic — Syntax

Let P and A be sets of *propositions* and *actions* respectively.

The set of *programs* Π is generated by the following grammar:

$$\alpha, \beta ::= a \; (a \in A) \mid \alpha + \beta \mid \alpha; \beta \mid \alpha^*$$

The set of *formulas* Φ is generated by the following grammar:

$$\phi, \psi ::= p, \overline{p} \; (p \in P) \mid \phi \vee \psi \mid \phi \wedge \psi \mid \langle \alpha \rangle \phi \mid [\alpha] \phi$$

Propositional Dynamic Logic — Semantics

A *frame* is a triple (W, σ, π) where

- W is a set of *worlds*;
- $\sigma : A \to \mathcal{P}(W \times W)$ assigns a relation to each action;
- $\pi : P \to \mathcal{P}(W)$ assigns a set of worlds to each proposition.

Propositional Dynamic Logic — Semantics

We can extend $\sigma : A \to \mathcal{P}(W \times W)$ to act on Π:

$$\sigma(\alpha + \beta) = \sigma(\alpha) \cup \sigma(\beta) \qquad \sigma(\alpha; \beta) = \sigma(\alpha) \circ \sigma(\beta) \qquad \sigma(\alpha^*) = \sigma(\alpha)^*$$

Propositional Dynamic Logic — Semantics

We can extend $\sigma : A \to \mathcal{P}(W \times W)$ to act on Π:

$$\sigma(\alpha + \beta) = \sigma(\alpha) \cup \sigma(\beta) \qquad \sigma(\alpha; \beta) = \sigma(\alpha) \circ \sigma(\beta) \qquad \sigma(\alpha^*) = \sigma(\alpha)^*$$

We can also extend $\pi : P \to \mathcal{P}(W)$ to act on Φ:

$$\pi(\overline{p}) = W \setminus \pi(p) \qquad \pi(\phi \vee \psi) = \pi(\phi) \cup \pi(\psi) \qquad \pi(\phi \wedge \psi) = \pi(\phi) \cap \pi(\psi)$$

$$\pi(\langle\alpha\rangle\phi) = \{w \in W : \exists w' \in W.\ w' \in \pi(\phi) \wedge w\ \sigma(\alpha)\ w'\}$$

$$\pi([\alpha]\phi) = \{w \in W : \forall w' \in W.\ w\ \sigma(\alpha)\ w' \implies w' \in \pi(\phi)\}$$

Propositional Dynamic Logic — demonstration

We are going to

1. encode our definition of a frame
2. encode the syntax of programs
3. assign a semantics to programs
4. encode the syntax of formulas
5. assign a semantics to formulas
6. prove some elementary properties of PDL

Propositional Dynamic Logic II — Syntax

Let P and A be sets of *propositions* and *actions* respectively.

The set of *programs* Π is generated by the following grammar:

$$\alpha, \beta ::= a \ (a \in A) \mid \alpha + \beta \mid \alpha; \beta \mid \alpha^*$$

The set of *formulas* Φ is generated by the following grammar:

$$\phi, \psi ::= p \ (p \in P) \mid \phi \vee \psi \mid \neg\phi \mid \langle\alpha\rangle\phi$$

Propositional Dynamic Logic II — Syntax

Let P and A be sets of *propositions* and *actions* respectively.

The set of *programs* Π is generated by the following grammar:

$$\alpha, \beta ::= a \ (a \in A) \mid \alpha + \beta \mid \alpha; \beta \mid \alpha^*$$

The set of *formulas* Φ is generated by the following grammar:

$$\phi, \psi ::= p \ (p \in P) \mid \phi \vee \psi \mid \neg\phi \mid \langle\alpha\rangle\phi$$

NB: no more negative propositions, conjunctions, or boxes!

Propositional Dynamic Logic II — Semantics

We can extend $\sigma : A \to \mathcal{P}(W \times W)$ to act on Π:

$$\sigma(\alpha + \beta) = \sigma(\alpha) \cup \sigma(\beta) \qquad \sigma(\alpha; \beta) = \sigma(\alpha) \circ \sigma(\beta) \qquad \sigma(\alpha^*) = \sigma(\alpha)^*$$

Propositional Dynamic Logic II — Semantics

We can extend $\sigma : A \to \mathcal{P}(W \times W)$ to act on Π:

$$\sigma(\alpha + \beta) = \sigma(\alpha) \cup \sigma(\beta) \qquad \sigma(\alpha; \beta) = \sigma(\alpha) \circ \sigma(\beta) \qquad \sigma(\alpha^*) = \sigma(\alpha)^*$$

We can also extend $\pi : P \to \mathcal{P}(W)$ to act on Φ:

$$\pi(\overline{p}) = W \setminus \pi(p) \qquad \pi(\phi \vee \psi) = \pi(\phi) \cup \pi(\psi) \qquad \pi(\neg\phi) = W \setminus \pi(\phi)$$

$$\pi(\langle\alpha\rangle\phi) = \{w \in W : \exists w' \in W.\ w' \in \pi(\phi) \wedge w\ \sigma(\alpha)\ w'\}$$

Propositional Dynamic Logic II — demonstration

We are going to

1. reuse our definition of a frame
2. resue the syntax/semantics of programs
3. encode the syntax of formulas
4. assign a semantics to formulas
5. prove some elementary properties of PDL

Propositional Dynamic Logic III — Syntax

Let P and A be sets of *propositions* and *actions* respectively.

The set of *programs* Π is generated by the following grammar:

$$\alpha, \beta ::= a \ (a \in A) \mid \phi? \ (\phi \in \Phi) \mid \alpha + \beta \mid \alpha; \beta \mid \alpha^*$$

The set of *formulas* Φ is generated by the following grammar:

$$\phi, \psi ::= p \ (p \in P) \mid \phi \vee \psi \mid \neg\phi \mid \langle\alpha\rangle\phi$$

Propositional Dynamic Logic III — Syntax

Let P and A be sets of *propositions* and *actions* respectively.

The set of *programs* Π is generated by the following grammar:

$$\alpha, \beta ::= a \ (a \in A) \mid \phi? \ (\phi \in \Phi) \mid \alpha + \beta \mid \alpha; \beta \mid \alpha^*$$

The set of *formulas* Φ is generated by the following grammar:

$$\phi, \psi ::= p \ (p \in P) \mid \phi \vee \psi \mid \neg\phi \mid \langle\alpha\rangle\phi$$

NB: mutual recursion!

Propositional Dynamic Logic III — Semantics

We can extend $\sigma : A \to \mathcal{P}(W \times W)$ to act on Π:

$$\sigma(\alpha + \beta) = \sigma(\alpha) \cup \sigma(\beta) \qquad \sigma(\phi?) = \{(w, w) : w \in \pi(\phi)\}$$

$$\sigma(\alpha; \beta) = \sigma(\alpha) \circ \sigma(\beta) \qquad \sigma(\alpha^*) = \sigma(\alpha)^*$$

Propositional Dynamic Logic III — Semantics

We can extend $\sigma : A \to \mathcal{P}(W \times W)$ to act on Π:

$$\sigma(\alpha + \beta) = \sigma(\alpha) \cup \sigma(\beta) \qquad \sigma(\phi?) = \{(w, w) : w \in \pi(\phi)\}$$

$$\sigma(\alpha; \beta) = \sigma(\alpha) \circ \sigma(\beta) \qquad \sigma(\alpha^*) = \sigma(\alpha)^*$$

We can also extend $\pi : P \to \mathcal{P}(W)$ to act on Φ:

$$\pi(\overline{p}) = W \setminus \pi(p) \qquad \pi(\phi \vee \psi) = \pi(\phi) \cup \pi(\psi) \qquad \pi(\neg\phi) = W \setminus \pi(\phi)$$

$$\pi(\langle\alpha\rangle\phi) = \{w \in W : \exists w' \in W.\ w' \in \pi(\phi) \wedge w\ \sigma(\alpha)\ w'\}$$

Propositional Dynamic Logic III — demonstration

We are going to

1. reuse our definition of a frame
2. encode the syntax of programs/formulas
3. encode the semantics of programs/formulas
4. prove some elementary properties of PDL

Demo

Next lecture

- learn about the Curry-Howard isomorphism
- synthesize programs, run proofs

Emojis designed by OpenMoji (CC BY-SA 4.0)

Next lecture

- learn about the Curry-Howard isomorphism
- synthesize programs, run proofs

Homework:

- Complete the proofs in `lecture-4*.v`.

Emojis designed by OpenMoji (CC BY-SA 4.0)