Mathematics for Computer Scientists 2 (G52MC2)

L03 : More Coq, Classical Logic

Thorsten Altenkirch

School of Computer Science
University of Nottingham

October 8, 2009

The cut rule

- Sometimes we want to prove a goal `Q` via an intermediate goal `P`.
- In this case we say:

  `cut P.`
- `cut` then generates two subgoals: $P \to Q$ and P.
- Proof rule:

$$\frac{\Gamma \vdash P \to Q \qquad \Gamma \vdash P}{\Gamma \vdash Q}\ \texttt{cut P}$$

- See l03.v for an example.

Shorthands for intro

- Instead of repeatedly using `intro` we can use `intros`.

  ```
  intro H1.
  intro H2.
  ```

  can be abbreviated as:

  ```
  intros H1 H2.
  ```

  This works for any number of hypotheses.
- We can omit the name of the hypothesis. If we say:

  ```
  intro.
  ```

  Coq will choose a name for us.
- This also works for `intros`.

  ```
  intros.
  ```

  repatedly applies intro — as long as the goal is an implication.
- **Warning:** Letting Coq generate names makes the proof less readabile and less flexible.

`assumption`

- `assumption` is an alternative to `exact H`.
- `assumption` searches through the hypotheses until it finds one that matches.
- `assumption` is preferable,
  if we use names generated by Coq.

Tactics and proof objects

- `intro(s)`, `apply`, ... are **tactics**.
- They are scripts that generate **proof objects**.
- E.g. after proving `I` (l01.v) we can say:

  ```
  Print I.
  ```

  and Coq replies with:

  ```
  I = fun H : P => H
    : P -> P
  ```

- Proof objects are expressions in the **Calculus of Inductive Constructions** (CIC).
- CIC is a functional programming language — like Haskell but with a more sophisticated type system.

Tactics and proof objects

- Instead of using tactics we could have said:

```
Definition I :  P -> P
             := fun H : P => H.
```

- It is better to use tactics to generate proofs.
- We can define new tactics using Coq's **tactic definition language** *(beyond the scope of this course)*.

Automatic tactics

- Proving theorems in propositional logic is a very *mechanic* activity.
- We can use automatic tactics, e.g. `auto`:

```
Lemma C : (P -> Q -> R) -> (Q -> P -> R).
auto.
Qed.
```

- **Warning:** You are not supposed to use automatic tactics in coursework or exam, unless you explicitely told so.

How does `auto` work?

- `auto` first does `intros`.
- Then `auto` tries to `apply` any matching hypothesis.
- `auto` then recursively repeats the two steps until it has completed the proof.
- To ensure termination, `auto` only repeats a fixed number of times.
- The default is 5, we can change this, e.g.

  ```
  auto 10.
  ```

  but this may be quite slow!

Limitations of `auto`

- `auto` doesn't know about $\wedge$,$\vee$ or $\neg$, e.g. it cannot prove:

```
Goal (P1 \/ P1) -> P1.
auto. (* fails *)
```

- The search depth may be insufficent, e.g. `auto` cannot prove:

```
Lemma hard : (P1 -> P2) -> (P2 -> P3)
          -> (P3 -> P4) -> (P4 -> P5)
          -> (P5 -> P6) -> P1 -> P6.
auto. (* fails *)
```

- However `auto 6.` can solve this goal.

tauto

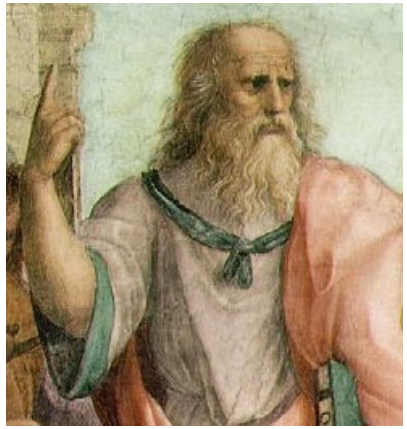
- `tauto` is a **complete** tactic for propositional logic.
- It can solve any goal in propositional logic that is provable!
- `tauto` uses the fact that propositional logic is **decidable**.
- `tauto` can be quite slow.
  The time grows exponential with the size of the goal.
- `auto` is more popular, because it is:
  - faster,
  - it also works for predicate logic,
  - it can be taught new tricks (using `Hint`).

To be or not to be . . .

- The proposition $P \vee \neg P$ is called the *principle of the excluded middle* (PEM).
- PEM is not accepted in *intuitionistic logic* but is valid in *classical logic*.
- PEM is not provable in Coq, but it can be added as an axiom.

```
Axiom classic : P \/ ~P.
```

Platonism and classical logic



Plato (423 - 348 BC)

- Classical logic is often associated with *Platonism*.
- In Platonism we believe that there is a perfect world of ideas (or forms).
- Every proposition is true or false with respect to this perfect world.
- Even, if we don't know it!

$A \wedge (B \vee C) \rightarrow (A \wedge B) \vee (A \wedge C)$, classically

A	B	C	$l = A \wedge (B \vee C)$	$r = A \wedge B \vee A \wedge C$	$l \rightarrow r$
False	False	False	False	False	True
False	False	True	False	False	True
False	True	False	False	False	True
False	True	True	False	False	True
True	False	False	False	False	True
True	False	True	True	True	True
True	True	False	True	True	True
True	True	True	True	True	True

- A proposition is a *classical tautology*, if its truthtable assigns always true.
- The same truth table shows that $A \wedge (B \vee C) \leftrightarrow (A \wedge B) \vee (A \wedge C)$

Intuitionism

Brouwer (1881-1966)

- The Dutch Mathematician Brouwer developed *Intuitionism* in the first half of the 20th century.
- Intuitionism doesn't require a perfect world of ideas.
- It is based on what humans can accept.
- Instead of truthtables we use the BHK interpretation.
- BHK = Brouwer, Heyting, Kolmogorov.
- *Programs are evidence!*

BHK in Haskell

- Evidence for $A \wedge B$ is given by pairs:
```
type And a b = (a,b)
```
- Evidence for $A \vee B$ is tagged evidence for A or B.
```
data Or a b = Inl a | Inr b
```
- Evidence for $A \longrightarrow B$ is a program computing evidence for B from evidence for A.

$A \wedge (B \vee C) \rightarrow (A \wedge B) \vee (A \wedge C)$, constructively

```
f :: And a (Or b c) -> Or (And a b) (And a c)

f (a,Inl b) = Inl (a,b)
f (a,Inr c) = Inr (a,c)
```

- A proposition is an intuionistic tautology, if we can write a (terminating) program of the corresponding type.
- Can you construct a program proving the other direction?

Evidence for PEM?

- PEM is justified by a truthtable:

P	$\neg P$	$P \vee \neg P$
True	False	True
False	True	True

- However, there is no (terminating) program:

```
pem :: Or a (a -> False)
```

  where False is a datatype with no constructors.

- Indeed, such a program would have to *decide* any proposition, e.g.

twin prime conjecture

There are infinitely many primes p such that p + 2 is also prime.