Sometimes we want to prove a goal $Q$ via an intermediate goal $P$.

In this case we say:

\texttt{cut} $P$.

\texttt{cut} then generates two subgoals: $P \rightarrow Q$ and $P$.

Proof rule:

\[
\begin{array}{c}
\Gamma \vdash P \rightarrow Q \\
\Gamma \vdash P \\
\hline
\Gamma \vdash Q \\
\end{array}
\quad \text{cut} P
\]

See l03.v for an example.
Instead of repeatedly using `intro` we can use `intros`.

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

can be abbreviated as:

```plaintext
intros H1 H2.
```

This works for any number of hypotheses.

We can omit the name of the hypothesis. If we say:

```plaintext
intro.
```

Coq will choose a name for us.

This also works for `intros`.

```plaintext
intros.
```

repeatedly applies intro — as long as the goal is an implication.

**Warning:** Letting Coq generate names makes the proof less readable and less flexible.
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.
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:

\[ I = \text{fun } H : P \Rightarrow H : P \Rightarrow P \]

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.
Instead of using tactics we could have said:

\[
\text{Definition } I : \ P \rightarrow P \\
:= \text{ fun } H : P \Rightarrow H.
\]

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).
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 explicitly told so.
How does \texttt{auto} work?

- \texttt{auto} first does intros.
- Then \texttt{auto} tries to apply any matching hypothesis.
- \texttt{auto} then recursively repeats the two steps until it has completed the proof.
- To ensure termination, \texttt{auto} only repeats a fixed number of times.
- The default is 5, we can change this, e.g. \texttt{auto 10}.
- but this may be quite slow!
• auto doesn’t know about ∧, ∨ or ¬, e.g. it cannot prove:
  Goal (P1 \/ P1) \rightarrow P1.
avto. (* fails *)

• The search depth may be insufficient, e.g. auto cannot prove:
  Lemma hard : (P1 \rightarrow P2) \rightarrow (P2 \rightarrow P3)
               \rightarrow (P3 \rightarrow P4) \rightarrow (P4 \rightarrow P5)
               \rightarrow (P5 \rightarrow P6) \rightarrow P1 \rightarrow P6.
avto. (* fails *)

• However auto 6. can solve this goal.
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).
The proposition \( P \lor \neg P \) 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 : \( P \lor \neg P \).
Platonism and classical logic

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!

Plato (423 - 348 BC)
\[ A \land (B \lor C) \rightarrow (A \land B) \lor (A \land C) \text{, classically} \]

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>( l = A \land (B \lor C) )</th>
<th>( r = A \land B \lor A \land C )</th>
<th>( l \rightarrow r )</th>
</tr>
</thead>
<tbody>
<tr>
<td>False</td>
<td>False</td>
<td>False</td>
<td>False</td>
<td>False</td>
<td>True</td>
</tr>
<tr>
<td>False</td>
<td>False</td>
<td>True</td>
<td>False</td>
<td>False</td>
<td>True</td>
</tr>
<tr>
<td>False</td>
<td>True</td>
<td>False</td>
<td>False</td>
<td>False</td>
<td>True</td>
</tr>
<tr>
<td>False</td>
<td>True</td>
<td>True</td>
<td>False</td>
<td>False</td>
<td>True</td>
</tr>
<tr>
<td>True</td>
<td>False</td>
<td>False</td>
<td>False</td>
<td>False</td>
<td>True</td>
</tr>
<tr>
<td>True</td>
<td>False</td>
<td>True</td>
<td>True</td>
<td>True</td>
<td>True</td>
</tr>
<tr>
<td>True</td>
<td>True</td>
<td>False</td>
<td>True</td>
<td>True</td>
<td>True</td>
</tr>
<tr>
<td>True</td>
<td>True</td>
<td>True</td>
<td>True</td>
<td>True</td>
<td>True</td>
</tr>
</tbody>
</table>

- A proposition is a **classical tautology**, if its truthtable assigns always true.
- The same truth table shows that
  \[ A \land (B \lor C) \leftrightarrow (A \land B) \lor (A \land C) \]
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 truth-tables we use the BHK interpretation.
- BHK = Brouwer, Heyting, Kolmogorov.
- *Programs are evidence!*
Evidence for $A \land B$ is given by pairs:

```
type And a b = (a, b)
```

Evidence for $A \lor B$ is tagged evidence for $A$ or $B$.

```
data Or a b = Inl a | Inr b
```

Evidence for $A \rightarrow B$ is a program computing evidence for $B$ from evidence for $A$. 
\( A \land (B \lor C) \rightarrow (A \land B) \lor (A \land C) \), constructively

\[
\begin{align*}
f &: \text{And} \ a \ (\text{Or} \ b \ c) \rightarrow \text{Or} \ (\text{And} \ a \ b) \ (\text{And} \ a \ c) \\
f (a, \text{Inl} \ b) &= \text{Inl} \ (a, b) \\
f (a, \text{Inr} \ c) &= \text{Inr} \ (a, c)
\end{align*}
\]

- A proposition is an intuitionistic 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 truth table:

<table>
<thead>
<tr>
<th>$P$</th>
<th>$\neg P$</th>
<th>$P \lor \neg P$</th>
</tr>
</thead>
<tbody>
<tr>
<td>True</td>
<td>False</td>
<td>True</td>
</tr>
<tr>
<td>False</td>
<td>True</td>
<td>True</td>
</tr>
</tbody>
</table>

- However, there is no (terminating) program:

```haskell
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.