Coq Proof Assistant: Propositions and Proofs

Mirco Kocher

Logic and Theory Group
Institute of Computer Science and Applied Mathematics
Universität Bern

2012

Overview

Overview

Overview

Overview

Overview

Overview

Overview

Overview

Overview

Propositional logic

$$(P \Rightarrow Q) \Rightarrow ((Q \Rightarrow R) \Rightarrow (P \Rightarrow R))$$

- P, Q, R are propositions
- Goal is to prove the implication

Propositional logic

$(P \Rightarrow Q) \Rightarrow ((Q \Rightarrow R) \Rightarrow (P \Rightarrow R))$

- P, Q, R are propositions
- Goal is to prove the implication

Tarski approach

$(P \Rightarrow Q)$

- Question "is the proposition P true?"
- Classical logic
- Assign to every variable a denotation true or false
- Truth table
- Formula is valid iff true in all cases

Tarski approach

$(P \Rightarrow Q)$

- Question "is the proposition P true?"
- Classical logic
- Assign to every variable a denotation true or false
- Truth table
- Formula is valid iff true in all cases

Tarski approach

$(P \Rightarrow Q)$

- Question "is the proposition P true?"
- Classical logic
- Assign to every variable a denotation true or false
- Truth table
- Formula is valid iff true in all cases

Tarski approach

$(P \Rightarrow Q)$

- Question "is the proposition P true?"
- Classical logic
- Assign to every variable a denotation true or false
- Truth table
- Formula is valid iff true in all cases

Tarski approach

$(P \Rightarrow Q)$

- Question "is the proposition P true?"
- Classical logic
- Assign to every variable a denotation true or false
- Truth table
- Formula is valid iff true in all cases

Heyting approach

$(P \Rightarrow Q)$

- Question "what are the proofs of P (if any)?"
- Intuitionistic logic
- Obtain a proof of Q from a proof of P
- Arbitrary proof of P constructs a proof of Q
- Coq system follows this approach

Heyting approach

$(P \Rightarrow Q)$

- Question "what are the proofs of P (if any)?"
- Intuitionistic logic
- Obtain a proof of Q from a proof of P
- Arbitrary proof of P constructs a proof of Q
- Coq system follows this approach

Heyting approach

$(P \Rightarrow Q)$

- Question "what are the proofs of P (if any)?"
- Intuitionistic logic
- Obtain a proof of Q from a proof of P
- Arbitrary proof of P constructs a proof of Q
- Coq system follows this approach

Heyting approach

$(P \Rightarrow Q)$

- Question "what are the proofs of P (if any)?"
- Intuitionistic logic
- Obtain a proof of Q from a proof of P
- Arbitrary proof of P constructs a proof of Q
- Coq system follows this approach

Heyting approach

$(P \Rightarrow Q)$

- Question "what are the proofs of P (if any)?"
- Intuitionistic logic
- Obtain a proof of Q from a proof of P
- Arbitrary proof of P constructs a proof of Q
- Coq system follows this approach

Translate from Heyting into Coq

$(P \Rightarrow Q)$

- Curry-Howard isomorphism is the direct relationship between computer programs and proofs
- Proofs are seen as programs
- Statements are treated as specifications
- The "$\Rightarrow$" becomes "$\rightarrow$"

$(P \rightarrow Q)$

Translate from Heyting into Coq

$(P \Rightarrow Q)$

- Curry-Howard isomorphism is the direct relationship between computer programs and proofs
- Proofs are seen as programs
- Statements are treated as specifications
- The "$\Rightarrow$" becomes "$\rightarrow$"

$(P \rightarrow Q)$

Translate from Heyting into Coq

$(P \Rightarrow Q)$

- Curry-Howard isomorphism is the direct relationship between computer programs and proofs
- Proofs are seen as programs
- Statements are treated as specifications
- The "$\Rightarrow$" becomes "$\rightarrow$"

$(P \rightarrow Q)$

Translate from Heyting into Coq

$(P \Rightarrow Q)$

- Curry-Howard isomorphism is the direct relationship between computer programs and proofs
- Proofs are seen as programs
- Statements are treated as specifications
- The "$\Rightarrow$" becomes "$\rightarrow$"

$(P \rightarrow Q)$

Overview

- Present types and tools used to reason in Coq
- Use tactics to obtain proofs in a semi-automatic way
- Control the flow of an interactive proof
- More details on tactics and corrresponting typing rules
- Composing tactics, tacticals
- Proof irrelevance

Overview

- Present types and tools used to reason in Coq
- Use tactics to obtain proofs in a semi-automatic way
- Control the flow of an interactive proof
- More details on tactics and corrresponting typing rules
- Composing tactics, tacticals
- Proof irrelevance

Overview

- Present types and tools used to reason in Coq
- Use tactics to obtain proofs in a semi-automatic way
- Control the flow of an interactive proof
- More details on tactics and corrresponting typing rules
- Composing tactics, tacticals
- Proof irrelevance

Overview

- Present types and tools used to reason in Coq
- Use tactics to obtain proofs in a semi-automatic way
- Control the flow of an interactive proof
- More details on tactics and corrresponting typing rules
- Composing tactics, tacticals
- Proof irrelevance

Overview

- Present types and tools used to reason in Coq
- Use tactics to obtain proofs in a semi-automatic way
- Control the flow of an interactive proof
- More details on tactics and corrresponding typing rules
- Composing tactics, tacticals
- Proof irrelevance

Overview

- Present types and tools used to reason in Coq
- Use tactics to obtain proofs in a semi-automatic way
- Control the flow of an interactive proof
- More details on tactics and corrresponding typing rules
- Composing tactics, tacticals
- Proof irrelevance

Sorts

Prop is one of the predefined sorts

- Plays a similar role as Set
- Set is for programs and specifications (`nat -> nat`, `bool`)
  Prop is for proofs and Propositions (P, Q, R)

Definition: Every type P whose type is the sort `Prop` is called a proposition. Any term t whose type is a propostion is called a proof term (=proof).

Sorts

Prop is one of the predefined sorts

- Plays a similar role as Set
- Set is for programs and specifications (`nat -> nat`, `bool`)
  Prop is for proofs and Propositions (P, Q, R)

Definition: Every type P whose type is the sort `Prop` is called a proposition. Any term t whose type is a propostion is called a proof term (=proof).

Sorts

Prop is one of the predefined sorts

- Plays a similar role as Set
- Set is for programs and specifications (`nat -> nat`, `bool`)
  Prop is for proofs and Propositions (P, Q, R)

Definition: Every type P whose type is the sort `Prop` is called a proposition. Any term t whose type is a propostion is called a proof term (=proof).

Hypothesis

`Hypothesis h:P`

- Local declaration
- *h* is the name of the hypothesis
- P is its statement
- Synonymous to

  `Variable h:P`
- Use

  `Hypotheses`

  or

  `Variables`

  to declare several at a time

Hypothesis

`Hypothesis h:P`

- Local declaration
- *h* is the name of the hypothesis
- P is its statement
- Synonymous to

  `Variable h:P`
- Use

  `Hypotheses`

  or

  `Variables`

  to declare several at a time

Hypothesis

`Hypothesis h:P`

- Local declaration
- *h* is the name of the hypothesis
- P is its statement
- Synonymous to
  `Variable h:P`
- Use
  `Hypotheses`
  or
  `Variables`
  to declare several at a time

Hypothesis

`Hypothesis h:P`

- Local declaration
- *h* is the name of the hypothesis
- P is its statement
- Synonymous to

  `Variable h:P`
- Use

  `Hypotheses`

  or

  `Variables`

  to declare several at a time

Hypothesis

`Hypothesis h:P`

- Local declaration
- *h* is the name of the hypothesis
- P is its statement
- Synonymous to

  `Variable h:P`
- Use

  `Hypotheses`

  or

  `Variables`

  to declare several at a time

Axiom

`Axiom x:P`

- Global declaration
- Synonymous to `Parameter x:P`

Environment contains axioms

Context contains hypotheses

$E, \Gamma \vdash \pi : P$

Axiom

```
Axiom x:P
```

- Global declaration
- Synonymous to
  `Parameter x:P`

Environment contains axioms

Context contains hypotheses

$E, \Gamma \vdash \pi : P$

Axiom

`Axiom x:P`

- Global declaration
- Synonymous to
  `Parameter x:P`

Environment contains axioms

Context contains hypotheses

$E, \Gamma \vdash \pi : P$

Axiom

```
Axiom x:P
```

- Global declaration
- Synonymous to
  `Parameter x:P`

Environment contains axioms

Context contains hypotheses

$E, \Gamma \vdash \pi : P$

Axiom

`Axiom x:P`

- Global declaration
- Synonymous to
  `Parameter x:P`

Environment contains axioms

Context contains hypotheses

$E, \Gamma \vdash \pi : P$

Goals and Tactics

A goal is the pairing of two pieces of information: a local context Γ and a type P that is well-formed in this context.

Goal: $E, \Gamma \vdash^{?} P$
Term t of type P is called a *solution*

Tactics:
Can be applied to a goal.
Decompose this goal into simpler goals and create new goals

If g is input goal and g_1, g_2, ..., g_k are output goals, then it's possible to construct a solution of g from the solutions of goals g_i.

Goals and Tactics

A goal is the pairing of two pieces of information: a local context Γ and a type P that is well-formed in this context.

Goal: $E, \Gamma \vdash^{?} P$
Term t of type P is called a *solution*

Tactics:
Can be applied to a goal.
Decompose this goal into simpler goals and create new goals

If g is input goal and g_1, g_2, ..., g_k are output goals, then it's possible to construct a solution of g from the solutions of goals g_i.

Goals and Tactics

A goal is the pairing of two pieces of information: a local context Γ and a type P that is well-formed in this context.

Goal: $E, \Gamma \vdash^{?} P$
Term t of type P is called a *solution*

Tactics:
Can be applied to a goal.
Decompose this goal into simpler goals and create new goals

If g is input goal and g_1, g_2, ..., g_k are output goals, then it's possible to construct a solution of g from the solutions of goals g_i.

Goals and Tactics

A goal is the pairing of two pieces of information: a local context Γ and a type P that is well-formed in this context.

Goal: $E, \Gamma \vdash^{?} P$
Term t of type P is called a *solution*

Tactics:
Can be applied to a goal.
Decompose this goal into simpler goals and create new goals

If g is input goal and g_1, g_2, ..., g_k are output goals, then it's possible to construct a solution of g from the solutions of goals g_i.

Goals and Tactics

A goal is the pairing of two pieces of information: a local context Γ and a type P that is well-formed in this context.

Goal: $E, \Gamma \vdash^{?} P$
Term t of type P is called a *solution*

Tactics:
Can be applied to a goal.
Decompose this goal into simpler goals and create new goals

If g is input goal and g_1, g_2, ..., g_k are output goals, then it's possible to construct a solution of g from the solutions of goals g_i.

Transitivity

Proove this implication by introducing tactics as needed

$$(P \to Q) \to (Q \to R) \to (P \to R)$$

Example 1

$$(P \to Q) \to (Q \to R) \to (P \to R)$$

```
Section Example1.
Variables P Q R T : Prop.
```

```
P is assumed
Q is assumed
R is assumed
T is assumed
```

Example 1

$(P \to Q) \to (Q \to R) \to (P \to R)$

```
Section Example1.
Variables P Q R T : Prop.
```

```
P is assumed
Q is assumed
R is assumed
T is assumed
```

Example 1

$(P \to Q) \to (Q \to R) \to (P \to R)$

```
Section Example1.
Variables P Q R T : Prop.
```

```
P is assumed
Q is assumed
R is assumed
T is assumed
```

Example 1

$$(P \to Q) \to (Q \to R) \to (P \to R)$$

```
Theorem imp_trans : (P->Q) -> (Q->R) -> P -> R.

1 subgoal

P : Prop
Q : Prop
R : Prop
T : Prop
===============
 (P->Q) -> (Q->R) -> P -> R

Proof.
```

Example 1

$(P \to Q) \to (Q \to R) \to (P \to R)$

```
Theorem imp_trans : (P->Q) -> (Q->R) -> P -> R.

1 subgoal

P : Prop
Q : Prop
R : Prop
T : Prop
===============
 (P->Q) -> (Q->R) -> P -> R

Proof.
```

Example 1

$$(P \to Q) \to (Q \to R) \to (P \to R)$$

```
Theorem imp_trans : (P->Q) -> (Q->R) -> P -> R.

1 subgoal

P : Prop
Q : Prop
R : Prop
T : Prop
===============
 (P->Q) -> (Q->R) -> P -> R

Proof.
```

Example 1

$$(P \to Q) \to (Q \to R) \to (P \to R)$$

```
Theorem imp_trans : (P->Q) -> (Q->R) -> P -> R.
```

```
1 subgoal

P : Prop
Q : Prop
R : Prop
T : Prop
===============
 (P->Q) -> (Q->R) -> P -> R
```

```
Proof.
```

intros

Goal: $(P \to Q) \to (Q \to R) \to (P \to R)$

```
intros H H' p.
```

- Transform the task of constructing a proof into proving R with those hypotheses added
- $H : P \to Q$
- $H' : Q \to R$ and
- $p : P$
- New subgoal: R

Simplifies the statement to prove and increases the resources available

intros

Goal: $(P \to Q) \to (Q \to R) \to (P \to R)$

```
intros H H' p.
```

- Transform the task of constructing a proof into proving R with those hypotheses added
- $H : P \to Q$
- $H' : Q \to R$ and
- $p : P$
- New subgoal: R

Simplifies the statement to prove and increases the resources available

intros

Goal: $(P \to Q) \to (Q \to R) \to (P \to R)$

```
intros H H' p.
```

- Transform the task of constructing a proof into proving R with those hypotheses added
- $H : P \to Q$
- $H' : Q \to R$ and
- $p : P$
- New subgoal: R

Simplifies the statement to prove and increases the resources available

intros

Goal: $(P \to Q) \to (Q \to R) \to (P \to R)$

```
intros H H' p.
```

- Transform the task of constructing a proof into proving R with those hypotheses added
- $H : P \to Q$
- $H' : Q \to R$ and
- $p : P$
- New subgoal: R

Simplifies the statement to prove and increases the resources available

intros

Goal: $(P \to Q) \to (Q \to R) \to (P \to R)$

```
intros H H' p.
```

- Transform the task of constructing a proof into proving R with those hypotheses added
- $H : P \to Q$
- $H' : Q \to R$ and
- $p : P$
- New subgoal: R

Simplifies the statement to prove and increases the resources available

intros

Goal: $(P \to Q) \to (Q \to R) \to (P \to R)$

```
intros H H' p.
```

- Transform the task of constructing a proof into proving R with those hypotheses added
- $H : P \to Q$
- $H' : Q \to R$ and
- $p : P$
- New subgoal: R

Simplifies the statement to prove and increases the resources available

intros

Goal: $(P \to Q) \to (Q \to R) \to (P \to R)$

```
intros H H' p.
```

- Transform the task of constructing a proof into proving R with those hypotheses added
- $H : P \to Q$
- $H' : Q \to R$ and
- $p : P$
- New subgoal: R

Simplifies the statement to prove and increases the resources available

intros

Goal: $(P \to Q) \to (Q \to R) \to (P \to R)$

```
intros H H' p.
```

- Transform the task of constructing a proof into proving R with those hypotheses added
- $H : P \to Q$
- $H' : Q \to R$ and
- $p : P$
- New subgoal: R

Simplifies the statement to prove and increases the resources available

Example 1

$$(P \to Q) \to (Q \to R) \to (P \to R)$$

```
intros H H' p.

1 subgoal

P : Prop
Q : Prop
R : Prop
T : Prop
H : P->Q
H' : Q->R
p : P
===============
 R
```

Example 1

$$(P \to Q) \to (Q \to R) \to (P \to R)$$

```
intros H H' p.
```

```
1 subgoal

P : Prop
Q : Prop
R : Prop
T : Prop
H : P->Q
H' : Q->R
p : P
===============
 R
```

Example 1

$(P \rightarrow Q) \rightarrow (Q \rightarrow R) \rightarrow (P \rightarrow R)$

```
intros H H' p.
```

```
1 subgoal

P : Prop
Q : Prop
R : Prop
T : Prop
H : P->Q
H' : Q->R
p : P
===============
 R
```

apply

Subgoal: R

Hypothesis $H' : Q \to R$ and $H : P \to Q$

`apply H'.`

- Use the hypothesis H' to advance our proof
- Argument has to be a premise and a conclusion
- Creates new goal for the premise
- New subgoal: Q

Applying hypothesis H gives the new subgoal P

apply

Subgoal: R

Hypothesis $H' : Q \to R$ and $H : P \to Q$

```
apply H'.
```

- Use the hypothesis H' to advance our proof
- Argument has to be a premise and a conclusion
- Creates new goal for the premise
- New subgoal: Q

Applying hypothesis H gives the new subgoal P

apply

Subgoal: R

Hypothesis $H' : Q \to R$ and $H : P \to Q$

`apply H'.`

- Use the hypothesis H' to advance our proof
- Argument has to be a premise and a conclusion
- Creates new goal for the premise
- New subgoal: Q

Applying hypothesis H gives the new subgoal P

apply

Subgoal: R

Hypothesis $H' : Q \to R$ and $H : P \to Q$

```
apply H'.
```

- Use the hypothesis H' to advance our proof
- Argument has to be a premise and a conclusion
- Creates new goal for the premise
- New subgoal: Q

Applying hypothesis H gives the new subgoal P

apply

Subgoal: R

Hypothesis $H' : Q \to R$ and $H : P \to Q$

```
apply H'.
```

- Use the hypothesis H' to advance our proof
- Argument has to be a premise and a conclusion
- Creates new goal for the premise
- New subgoal: Q

Applying hypothesis H gives the new subgoal P

apply

Subgoal: R

Hypothesis $H' : Q \to R$ and $H : P \to Q$

```
apply H'.
```

- Use the hypothesis H' to advance our proof
- Argument has to be a premise and a conclusion
- Creates new goal for the premise
- New subgoal: Q

Applying hypothesis H gives the new subgoal P

Example 1

$$(P \to Q) \to (Q \to R) \to (P \to R)$$

```
apply H'.

1 subgoal

P : Prop
Q : Prop
R : Prop
T : Prop
H : P->Q
H' : Q->R
p : P
===============
 Q
```

Example 1

$(P \to Q) \to (Q \to R) \to (P \to R)$

```
apply H'.
```

```
1 subgoal

P : Prop
Q : Prop
R : Prop
T : Prop
H : P->Q
H' : Q->R
p : P
===============
 Q
```

Example 1

$(P \to Q) \to (Q \to R) \to (P \to R)$

```
apply H'.
```

```
1 subgoal

P : Prop
Q : Prop
R : Prop
T : Prop
H : P->Q
H' : Q->R
p : P
===============
 Q
```

Example 1

$(P \to Q) \to (Q \to R) \to (P \to R)$

```
apply H.

1 subgoal

P : Prop
Q : Prop
R : Prop
T : Prop
H : P->Q
H' : Q->R
p : P
===============
 P
```

Example 1

$$(P \to Q) \to (Q \to R) \to (P \to R)$$

```
apply H.
```

```
1 subgoal

P : Prop
Q : Prop
R : Prop
T : Prop
H : P->Q
H' : Q->R
p : P
===============
 P
```

Example 1

$(P \to Q) \to (Q \to R) \to (P \to R)$

```
apply H.
```

```
1 subgoal

P : Prop
Q : Prop
R : Prop
T : Prop
H : P->Q
H' : Q->R
p : P
===============
 P
```

Tactic assumption

Subgoal: P

Hypothesis $p : P$

`assumption.`

- Statement to proof is exactly statement of hypothesis p
- Succeeds without generating any new goal

`No more subgoals`

Proof complete

Tactic assumption

Subgoal: P

Hypothesis $p : P$

`assumption.`

- Statement to proof is exactly statement of hypothesis p
- Succeeds without generating any new goal

`No more subgoals`

Proof complete

Tactic assumption

Subgoal: P

Hypothesis $p : P$

```
assumption.
```

- Statement to proof is exactly statement of hypothesis p
- Succeeds without generating any new goal

```
No more subgoals
```

Proof complete

Tactic assumption

Subgoal: P

Hypothesis $p : P$

`assumption.`

- Statement to proof is exactly statement of hypothesis p
- Succeeds without generating any new goal

`No more subgoals`

Proof complete

Tactic assumption

Subgoal: P

Hypothesis $p : P$

`assumption.`

- Statement to proof is exactly statement of hypothesis p
- Succeeds without generating any new goal

`No more subgoals`

Proof complete

Finish

```
Qed.
```

- Every proof should be terminated by the Qed command
- Saves the theorem's name, statement and proof term
- Displays the sequence of tactics.

```
intros H H' p.
apply H'.
apply H.
assumption.
```

```
Print imp_trans.
```

- Shows the proof like any *Gallina* definition

```
imp-trans = fun (H:P->Q)(H':Q->R)(p:P) => H' (H p)
 : (P->Q) -> (Q->R) -> P -> R
```

Finish

```
Qed.
```

- Every proof should be terminated by the Qed command
- Saves the theorem's name, statement and proof term
- Displays the sequence of tactics.

```
intros H H' p.
apply H'.
apply H.
assumption.
```

```
Print imp_trans.
```

- Shows the proof like any *Gallina* definition

```
imp-trans = fun (H:P->Q)(H':Q->R)(p:P) => H' (H p)
 : (P->Q) -> (Q->R) -> P -> R
```

Finish

```
Qed.
```

- Every proof should be terminated by the Qed command
- Saves the theorem's name, statement and proof term
- Displays the sequence of tactics.

```
intros H H' p.
apply H'.
apply H.
assumption.
```

```
Print imp_trans.
```

- Shows the proof like any *Gallina* definition

```
imp-trans = fun (H:P->Q)(H':Q->R)(p:P) => H' (H p)
 : (P->Q) -> (Q->R) -> P -> R
```

Finish

```
Qed.
```

- Every proof should be terminated by the Qed command
- Saves the theorem's name, statement and proof term
- Displays the sequence of tactics.

```
intros H H' p.
apply H'.
apply H.
assumption.
```

```
Print imp_trans.
```

- Shows the proof like any *Gallina* definition

```
imp-trans = fun (H:P->Q)(H':Q->R)(p:P) => H' (H p)
 : (P->Q) -> (Q->R) -> P -> R
```

Finish

```
Qed.
```

- Every proof should be terminated by the Qed command
- Saves the theorem's name, statement and proof term
- Displays the sequence of tactics.

```
intros H H' p.
apply H'.
apply H.
assumption.
```

```
Print imp_trans.
```

- Shows the proof like any *Gallina* definition

```
imp-trans = fun (H:P->Q)(H':Q->R)(p:P) => H' (H p)
 : (P->Q) -> (Q->R) -> P -> R
```

Finish

```
Qed.
```

- Every proof should be terminated by the Qed command
- Saves the theorem's name, statement and proof term
- Displays the sequence of tactics.

```
intros H H' p.
apply H'.
apply H.
assumption.
```

```
Print imp_trans.
```

- Shows the proof like any *Gallina* definition

```
imp-trans = fun (H:P->Q)(H':Q->R)(p:P) => H' (H p)
 : (P->Q) -> (Q->R) -> P -> R
```

Reading the proof term

```
Print imp_trans.

imp-trans = fun (H:P->Q)(H':Q->R)(p:P) => H' (H p)
 : (P->Q) -> (Q->R) -> P -> R
```

(H) : Assume P $\to$ Q
(H') : Assume Q $\to$ R
(p) : Assume P
(1) : By using (p) we get P
(2) : By applyig (H) we get Q
(3) : By applying (H') we get R

Discharging H, H' and p, we get $(P \to Q) \to (Q \to R) \to P \to R$

Reading the proof term

```
Print imp_trans.
```

```
imp-trans = fun (H:P->Q)(H':Q->R)(p:P) => H' (H p)
 : (P->Q) -> (Q->R) -> P -> R
```

(H) : Assume P $\rightarrow$ Q
(H') : Assume Q $\rightarrow$ R
(p) : Assume P
(1) : By using (p) we get P
(2) : By applyig (H) we get Q
(3) : By applying (H') we get R

Discharging H, H' and p, we get $(P \rightarrow Q) \rightarrow (Q \rightarrow R) \rightarrow P \rightarrow R$

Reading the proof term

```
Print imp_trans.
```

```
imp-trans = fun (H:P->Q)(H':Q->R)(p:P) => H' (H p)
 : (P->Q) -> (Q->R) -> P -> R
```

(H) : Assume P $\rightarrow$ Q
(H') : Assume Q $\rightarrow$ R
(p) : Assume P
(1) : By using (p) we get P
(2) : By applyig (H) we get Q
(3) : By applying (H') we get R

Discharging H, H' and p, we get $(P \rightarrow Q) \rightarrow (Q \rightarrow R) \rightarrow P \rightarrow R$

Tactic auto

Not all the details for the proof are needed

Transitivity is a rather trivial statement
A few automatic tactics are able to solve this goal

```
Theorem transitivity : (P->Q) -> (Q->R) -> P -> R.
Proof.

auto.
Qed.
```

Tactic auto

Not all the details for the proof are needed

Transitivity is a rather trivial statement
A few automatic tactics are able to solve this goal

```
Theorem transitivity : (P->Q) -> (Q->R) -> P -> R.
Proof.

auto.
Qed.
```

Tactic auto

Not all the details for the proof are needed

Transitivity is a rather trivial statement
A few automatic tactics are able to solve this goal

```
Theorem transitivity : (P->Q) -> (Q->R) -> P -> R.
Proof.

auto.
Qed.
```

Tactic auto

Not all the details for the proof are needed

Transitivity is a rather trivial statement
A few automatic tactics are able to solve this goal

```
Theorem transitivity : (P->Q) -> (Q->R) -> P -> R.
Proof.

auto.
Qed.
```

Building Rules

Same kind of typing rules that control the construction of proposition as those used to build simple specifications.

Difference is in the use of *Prop* instead of *Set*.

$E, \Gamma \vdash id : Prop$

```
Check id.
```

```
id : Prop
```

Building Rules

Same kind of typing rules that control the construction of proposition as those used to build simple specifications.

Difference is in the use of *Prop* instead of *Set*.

$E, \Gamma \vdash id : Prop$

```
Check id.
```

```
id : Prop
```

Building Rules

Same kind of typing rules that control the construction of proposition as those used to build simple specifications.

Difference is in the use of *Prop* instead of *Set*.

$E, \Gamma \vdash id : Prop$

```
Check id.
```

```
id : Prop
```

Building Rules

Same kind of typing rules that control the construction of proposition as those used to build simple specifications.

Difference is in the use of *Prop* instead of *Set*.

$E, \Gamma \vdash id : Prop$

```
Check id.
```

```
id : Prop
```

Building Rules

Same kind of typing rules that control the construction of proposition as those used to build simple specifications.

Difference is in the use of *Prop* instead of *Set*.

$E, \Gamma \vdash id : Prop$

```
Check id.
```

```
id : Prop
```

General Prod

$$\textbf{Prod-Prop}\ \frac{E, \Gamma \vdash P : Prop \qquad E, \Gamma \vdash Q : Prop}{E, \Gamma \vdash P \to Q : Prop}$$

As the rules Prod-Set and Prod-Prop only differ in their use of sorts they can be presented as a single rule with parameter sort s

$$\textbf{Prod}\ \frac{E, \Gamma \vdash A : s \qquad E, \Gamma \vdash B : s}{E, \Gamma \vdash A \to B : s}$$

With $s \in \{\text{Set}, \text{Prop}\}$

General Prod

$$\textbf{Prod-Prop}\ \frac{E, \Gamma \vdash P : Prop \qquad E, \Gamma \vdash Q : Prop}{E, \Gamma \vdash P \rightarrow Q : Prop}$$

As the rules Prod-Set and Prod-Prop only differ in their use of sorts they can be presented as a single rule with parameter sort s

$$\textbf{Prod}\ \frac{E, \Gamma \vdash A : s \qquad E, \Gamma \vdash B : s}{E, \Gamma \vdash A \rightarrow B : s}$$

With $s \in \{\text{Set}, \text{Prop}\}$

General Prod

$$\textbf{Prod-Prop}\ \frac{E, \Gamma \vdash P : Prop \qquad E, \Gamma \vdash Q : Prop}{E, \Gamma \vdash P \rightarrow Q : Prop}$$

As the rules Prod-Set and Prod-Prop only differ in their use of sorts they can be presented as a single rule with parameter sort s

$$\textbf{Prod}\ \frac{E, \Gamma \vdash A : s \qquad E, \Gamma \vdash B : s}{E, \Gamma \vdash A \rightarrow B : s}$$

With $s \in \{\text{Set}, \text{Prop}\}$

Var rule

$$\textbf{Var}\ \frac{(x : P) \in E \cup \Gamma}{E, \Gamma \vdash x : P}$$

This corresponds to the tactic `assumption` or, if x is not in the context, `exact x`.

Var rule

$$\textbf{Var}\ \frac{(x : P) \in E \cup \Gamma}{E, \Gamma \vdash x : P}$$

This corresponds to the tactic `assumption` or, if x is not in the context, `exact x`.

exact

The exact tactic takes any term (with right type) as argument

The whole proof could be given as an argument

```
Theorem delta : (P->P->Q) -> P -> Q.
Proof.

exact (fun (H:P->P->Q)(p:P) => H p p).
Qed.
```

Or even shorter:

```
Theorem delta : (P->P->Q) -> P -> Q.
Proof (fun (H:P->P->Q)(p:P) => H p p).
```

Uses the context hypotheses

exact

The exact tactic takes any term (with right type) as argument

The whole proof could be given as an argument

```
Theorem delta : (P->P->Q) -> P -> Q.
Proof.
exact (fun (H:P->P->Q)(p:P) => H p p).
Qed.
```

Or even shorter:

```
Theorem delta : (P->P->Q) -> P -> Q.
Proof (fun (H:P->P->Q)(p:P) => H p p).
```

Uses the context hypotheses

exact

The exact tactic takes any term (with right type) as argument

The whole proof could be given as an argument

```
Theorem delta : (P->P->Q) -> P -> Q.
Proof.
```

```
exact (fun (H:P->P->Q)(p:P) => H p p).
Qed.
```

Or even shorter:

```
Theorem delta : (P->P->Q) -> P -> Q.
Proof (fun (H:P->P->Q)(p:P) => H p p).
```

Uses the context hypotheses

exact

The exact tactic takes any term (with right type) as argument

The whole proof could be given as an argument

```
Theorem delta : (P->P->Q) -> P -> Q.
Proof.
exact (fun (H:P->P->Q)(p:P) => H p p).
Qed.
```

Or even shorter:

```
Theorem delta : (P->P->Q) -> P -> Q.
Proof (fun (H:P->P->Q)(p:P) => H p p).
```

Uses the context hypotheses

exact

The exact tactic takes any term (with right type) as argument

The whole proof could be given as an argument

```
Theorem delta : (P->P->Q) -> P -> Q.
Proof.

exact (fun (H:P->P->Q)(p:P) => H p p).
Qed.
```

Or even shorter:

```
Theorem delta : (P->P->Q) -> P -> Q.
Proof (fun (H:P->P->Q)(p:P) => H p p).
```

Uses the context hypotheses

exact

The exact tactic takes any term (with right type) as argument

The whole proof could be given as an argument

```
Theorem delta : (P->P->Q) -> P -> Q.
Proof.
exact (fun (H:P->P->Q)(p:P) => H p p).
Qed.
```

Or even shorter:

```
Theorem delta : (P->P->Q) -> P -> Q.
Proof (fun (H:P->P->Q)(p:P) => H p p).
```

Uses the context hypotheses

Modus Ponens

Apply tactic uses the Modus Ponens

$$\textbf{App}\ \frac{E, \Gamma \vdash t : P \rightarrow Q \qquad E, \Gamma \vdash t' : P}{E, \Gamma \vdash tt' : Q}$$

Term t : $P_1 \rightarrow P_2 \rightarrow ... \rightarrow P_n \rightarrow Q$
Goal : $P_k \rightarrow P_{k+1} \rightarrow ... \rightarrow P_n \rightarrow Q$, also called "head type" of rank k
The term Q is called "final type" if it's not an arrow type.
Head and final types play a role in later tactics

```
apply t
```

generates k-1 goals with statements $P_1, ..., P_{k-1}$

if new goals have solution $t_1, t_2, ..., t_{k-1}$
solution is $t\ t_1\ t_2\ ...\ t_{k-1}$ for initial goal

Modus Ponens

Apply tactic uses the Modus Ponens

$$\textbf{App}\ \frac{E, \Gamma \vdash t : P \to Q \qquad E, \Gamma \vdash t' : P}{E, \Gamma \vdash tt' : Q}$$

Term $t: P_1 \to P_2 \to ... \to P_n \to Q$
Goal : $P_k \to P_{k+1} \to ... \to P_n \to Q$, also called "head type" of rank k
The term Q is called "final type" if it's not an arrow type.
Head and final types play a role in later tactics

```
apply t
```

generates k-1 goals with statements $P_1, ..., P_{k-1}$

if new goals have solution $t_1, t_2, ..., t_{k-1}$
solution is $t\ t_1\ t_2\ ...\ t_{k-1}$ for initial goal

Modus Ponens

Apply tactic uses the Modus Ponens

$$\textbf{App}\ \frac{E, \Gamma \vdash t : P \rightarrow Q \qquad E, \Gamma \vdash t' : P}{E, \Gamma \vdash tt' : Q}$$

Term $t : P_1 \rightarrow P_2 \rightarrow ... \rightarrow P_n \rightarrow Q$
Goal : $P_k \rightarrow P_{k+1} \rightarrow ... \rightarrow P_n \rightarrow Q$, also called "head type" of rank k
The term Q is called "final type" if it's not an arrow type.
Head and final types play a role in later tactics

`apply t`

generates k-1 goals with statements $P_1, ..., P_{k-1}$

if new goals have solution $t_1, t_2, ..., t_{k-1}$
solution is $t\ t_1\ t_2\ ...\ t_{k-1}$ for initial goal

Modus Ponens

Apply tactic uses the Modus Ponens

$$\textbf{App}\ \frac{E, \Gamma \vdash t : P \to Q \qquad E, \Gamma \vdash t' : P}{E, \Gamma \vdash tt' : Q}$$

Term $t : P_1 \to P_2 \to \ldots \to P_n \to Q$
Goal : $P_k \to P_{k+1} \to \ldots \to P_n \to Q$, also called "head type" of rank k
The term Q is called "final type" if it's not an arrow type.
Head and final types play a role in later tactics

```
apply t
```

generates k-1 goals with statements $P_1, \ldots, P_{k-1}$

if new goals have solution $t_1, t_2, \ldots, t_{k-1}$
solution is $t\ t_1\ t_2\ \ldots\ t_{k-1}$ for initial goal

Modus Ponens

Apply tactic uses the Modus Ponens

$$\textbf{App}\ \frac{E, \Gamma \vdash t : P \rightarrow Q \qquad E, \Gamma \vdash t' : P}{E, \Gamma \vdash tt' : Q}$$

Term t : $P_1 \rightarrow P_2 \rightarrow \ldots \rightarrow P_n \rightarrow Q$
Goal : $P_k \rightarrow P_{k+1} \rightarrow \ldots \rightarrow P_n \rightarrow Q$, also called "head type" of rank k
The term Q is called "final type" if it's not an arrow type.
Head and final types play a role in later tactics

```
apply t
```

generates k-1 goals with statements $P_1, \ldots, P_{k-1}$

if new goals have solution $t_1, t_2, \ldots, t_{k-1}$
solution is $t\ t_1\ t_2\ \ldots\ t_{k-1}$ for initial goal

Lam rule

Proof of $P \rightarrow Q$ is a function mapping any proof of P to a proof of Q

$$\textbf{Lam}\ \frac{E, \Gamma :: (H : P) \vdash t : Q}{E, \Gamma \vdash \mathrm{fun}\ H : P \Rightarrow t : P \rightarrow Q}$$

$\Gamma \vdash^{?} P \rightarrow Q$

$\Gamma :: (H : P) \vdash^{?} Q$

If term t is a solution to the subgoal then "fun $H : P \Rightarrow t$" is a solution to the initial goal.

This corresponds to the tactic intro

Lam rule

Proof of $P \to Q$ is a function mapping any proof of P to a proof of Q

$$\textbf{Lam}\ \frac{E, \Gamma :: (H : P) \vdash t : Q}{E, \Gamma \vdash \text{fun } H : P \Rightarrow t : P \to Q}$$

$\Gamma \vdash^? P \to Q$

$\Gamma :: (H : P) \vdash^? Q$

If term t is a solution to the subgoal then "fun $H : P \Rightarrow t$" is a solution to the initial goal.

This corresponds to the tactic intro

Lam rule

Proof of $P \rightarrow Q$ is a function mapping any proof of P to a proof of Q

$$\textbf{Lam}\ \frac{E, \Gamma :: (H : P) \vdash t : Q}{E, \Gamma \vdash \text{fun } H : P \Rightarrow t : P \rightarrow Q}$$

$\Gamma \vdash^{?} P \rightarrow Q$

$\Gamma :: (H : P) \vdash^{?} Q$

If term t is a solution to the subgoal then "fun $H : P \Rightarrow t$" is a solution to the initial goal.

This corresponds to the tactic intro

Lam rule

Proof of $P \to Q$ is a function mapping any proof of P to a proof of Q

$$\textbf{Lam}\ \frac{E, \Gamma :: (H : P) \vdash t : Q}{E, \Gamma \vdash \text{fun } H : P \Rightarrow t : P \to Q}$$

$\Gamma \vdash^? P \to Q$

$\Gamma :: (H : P) \vdash^? Q$

If term t is a solution to the subgoal then "fun $H : P \Rightarrow t$" is a solution to the initial goal.

This corresponds to the tactic intro

Lam rule

Proof of $P \to Q$ is a function mapping any proof of P to a proof of Q

$$\textbf{Lam}\ \frac{E, \Gamma :: (H : P) \vdash t : Q}{E, \Gamma \vdash \mathsf{fun}\ H : P \Rightarrow t : P \to Q}$$

$\Gamma \vdash^{?} P \to Q$

$\Gamma :: (H : P) \vdash^{?} Q$

If term t is a solution to the subgoal then "fun $H : P \Rightarrow t$" is a solution to the initial goal.

This corresponds to the tactic intro

Lam rule

Proof of $P \to Q$ is a function mapping any proof of P to a proof of Q

$$\textbf{Lam}\ \frac{E, \Gamma :: (H : P) \vdash t : Q}{E, \Gamma \vdash \text{fun } H : P \Rightarrow t : P \to Q}$$

$\Gamma \vdash^? P \to Q$

$\Gamma :: (H : P) \vdash^? Q$

If term t is a solution to the subgoal then "fun $H : P \Rightarrow t$" is a solution to the initial goal.

This corresponds to the tactic intro

intros

`intros v_1, v_2, ..., v_n`

is the same as

`intro v_1, intro v_2, ..., intro v_n`

intro v_1 takes the first implication as a hypothesis called v_1

Proof Theorem K : $P \to Q \to P$
Goal is $P \to Q \to P$

`intro p.`

Hypothesis $p : P$ and new Goal $Q \to P$

Without a name intro (resp intros) generates exactly one hypothese (resp as many hypotheses as possible) and automatically names them.

intros

```
intros v_1, v_2, ..., v_n
```

is the same as

```
intro v_1, intro v_2, ..., intro v_n
```

intro v_1 takes the first implication as a hypothesis called v_1

Proof Theorem K : $P \to Q \to P$
Goal is $P \to Q \to P$

```
intro p.
```

Hypothesis $p : P$ and new Goal $Q \to P$

Without a name intro (resp intros) generates exactly one hypothese (resp as many hypotheses as possible) and automatically names them.

intros

`intros v_1, v_2, ..., v_n`

is the same as

`intro v_1, intro v_2, ..., intro v_n`

intro v_1 takes the first implication as a hypothesis called v_1

Proof Theorem K : $P \to Q \to P$
Goal is $P \to Q \to P$

`intro p.`

Hypothesis $p : P$ and new Goal $Q \to P$

Without a name intro (resp intros) generates exactly one hypothese (resp as many hypotheses as possible) and automatically names them.

intros

`intros v_1, v_2, ..., v_n`

is the same as

`intro v_1, intro v_2, ..., intro v_n`

intro v_1 takes the first implication as a hypothesis called v_1

Proof Theorem K : $P \to Q \to P$
Goal is $P \to Q \to P$

`intro p.`

Hypothesis $p : P$ and new Goal $Q \to P$

Without a name intro (resp intros) generates exactly one hypothese (resp as many hypotheses as possible) and automatically names them.

intros

`intros v_1, v_2, ..., v_n`

is the same as

`intro v_1, intro v_2, ..., intro v_n`

intro v_1 takes the first implication as a hypothesis called v_1

Proof Theorem K : $P \to Q \to P$
Goal is $P \to Q \to P$

`intro p.`

Hypothesis $p : P$ and new Goal $Q \to P$

Without a name intro (resp intros) generates exactly one hypothese (resp as many hypotheses as possible) and automatically names them.

intros

`intros v_1, v_2, ..., v_n`

is the same as

`intro v_1, intro v_2, ..., intro v_n`

intro v_1 takes the first implication as a hypothesis called v_1

Proof Theorem K : $P \to Q \to P$
Goal is $P \to Q \to P$

`intro p.`

Hypothesis $p : P$ and new Goal $Q \to P$

Without a name intro (resp intros) generates exactly one hypothese (resp as many hypotheses as possible) and automatically names them.

Handling

`Show i`

Display goal *i* with complete context
Coq displays the current goal after each proof step

`Undo n`

Go back *n* steps and try an alternative if goal can not be solved

`Focus n`

Focus the attention on the *n*th subgoal to prove

`Restart`

Go back to the beginning of the proof

`Abort`

Abandon the proof

Handling

`Show i`

Display goal *i* with complete context
Coq displays the current goal after each proof step

`Undo n`

Go back *n* steps and try an alternative if goal can not be solved

`Focus n`

Focus the attention on the *n*th subgoal to prove

`Restart`

Go back to the beginning of the proof

`Abort`

Abandon the proof

Handling

`Show i`

Display goal *i* with complete context
Coq displays the current goal after each proof step

`Undo n`

Go back *n* steps and try an alternative if goal can not be solved

`Focus n`

Focus the attention on the *n*th subgoal to prove

`Restart`

Go back to the beginning of the proof

`Abort`

Abandon the proof

Handling

`Show i`

Display goal *i* with complete context
Coq displays the current goal after each proof step

`Undo n`

Go back *n* steps and try an alternative if goal can not be solved

`Focus n`

Focus the attention on the *n*th subgoal to prove

`Restart`

Go back to the beginning of the proof

`Abort`

Abandon the proof

Handling

`Show i`

Display goal *i* with complete context
Coq displays the current goal after each proof step

`Undo n`

Go back *n* steps and try an alternative if goal can not be solved

`Focus n`

Focus the attention on the *n*th subgoal to prove

`Restart`

Go back to the beginning of the proof

`Abort`

Abandon the proof

Simple composing

Combine tactics without stopping at intermediary subgoals

Goal: $P \rightarrow Q \rightarrow (P \rightarrow Q \rightarrow R) \rightarrow R$

```
intros p q H; apply H; assumption.
```

Like chess: forsee results of tactics

If any tatcic fails, then the whole combination fails

Simple composing

Combine tactics without stopping at intermediary subgoals

Goal: $P \to Q \to (P \to Q \to R) \to R$

```
intros p q H; apply H; assumption.
```

Like chess: forsee results of tactics

If any tatcic fails, then the whole combination fails

Simple composing

Combine tactics without stopping at intermediary subgoals

Goal: $P \to Q \to (P \to Q \to R) \to R$

```
intros p q H; apply H; assumption.
```

Like chess: forsee results of tactics

If any tatcic fails, then the whole combination fails

Simple composing

Combine tactics without stopping at intermediary subgoals

Goal: $P \to Q \to (P \to Q \to R) \to R$

`intros p q H; apply H; assumption.`

Like chess: forsee results of tactics

If any tatcic fails, then the whole combination fails

Simple composing

Combine tactics without stopping at intermediary subgoals

Goal: $P \rightarrow Q \rightarrow (P \rightarrow Q \rightarrow R) \rightarrow R$

```
intros p q H; apply H; assumption.
```

Like chess: forsee results of tactics

If any tatcic fails, then the whole combination fails

General composing

Tactics can generate multiple subgoals

Goal: $(P \to Q \to R) \to (P \to Q) \to (P \to R)$

```
intros H H' p; apply H; [assumption | apply H'; assumption].
```

Two subgoals P and Q

First is solved with assumption
Other one first has to apply H' and then use assumption

General composing

Tactics can generate multiple subgoals

Goal: $(P \to Q \to R) \to (P \to Q) \to (P \to R)$

```
intros H H' p; apply H; [assumption | apply H'; assumption].
```

Two subgoals P and Q

First is solved with assumption
Other one first has to apply H' and then use assumption

General composing

Tactics can generate multiple subgoals

Goal: $(P \to Q \to R) \to (P \to Q) \to (P \to R)$

```
intros H H' p; apply H; [assumption | apply H'; assumption].
```

Two subgoals P and Q

First is solved with assumption
Other one first has to apply H' and then use assumption

General composing

Tactics can generate multiple subgoals

Goal: $(P \to Q \to R) \to (P \to Q) \to (P \to R)$

```
intros H H' p; apply H; [assumption | apply H'; assumption].
```

Two subgoals P and Q

First is solved with assumption
Other one first has to apply H' and then use assumption

General composing

Tactics can generate multiple subgoals

Goal: $(P \to Q \to R) \to (P \to Q) \to (P \to R)$

```
intros H H' p; apply H; [assumption | apply H'; assumption].
```

Two subgoals P and Q

First is solved with assumption

Other one first has to apply H' and then use assumption

General composing

Tactics can generate multiple subgoals

Goal: $(P \to Q \to R) \to (P \to Q) \to (P \to R)$

```
intros H H' p; apply H; [assumption | apply H'; assumption].
```

Two subgoals P and Q

First is solved with assumption
Other one first has to apply H' and then use assumption

More composing

If a tactic fails automatically use another tactic

```
intros H p; apply H; (assumption || intro H').
```

If assumption fails, then intro H' is used

A tactic can be left unchanged to finish every subgoal in one go

Goal: $(P \to Q) \to (P \to R) \to (P \to Q \to R \to T) \to P \to T$

```
intros H H0 H1 p.
apply H1; [idtac | apply H | apply H0]; assumption
```

More composing

If a tactic fails automatically use another tactic

```
intros H p; apply H; (assumption || intro H').
```

If assumption fails, then intro H' is used

A tactic can be left unchanged to finish every subgoal in one go

Goal: $(P \to Q) \to (P \to R) \to (P \to Q \to R \to T) \to P \to T$

```
intros H H0 H1 p.
apply H1; [idtac | apply H | apply H0]; assumption
```

More composing

If a tactic fails automatically use another tactic

`intros H p; apply H; (assumption || intro H').`

If assumption fails, then intro H' is used

A tactic can be left unchanged to finish every subgoal in one go

Goal: $(P \to Q) \to (P \to R) \to (P \to Q \to R \to T) \to P \to T$

```
intros H H0 H1 p.
apply H1; [idtac | apply H | apply H0]; assumption
```

More composing

If a tactic fails automatically use another tactic

```
intros H p; apply H; (assumption || intro H').
```

If assumption fails, then intro H' is used

A tactic can be left unchanged to finish every subgoal in one go

Goal: $(P \to Q) \to (P \to R) \to (P \to Q \to R \to T) \to P \to T$

```
intros H H0 H1 p.
apply H1; [idtac | apply H | apply H0]; assumption
```

More composing

If a tactic fails automatically use another tactic

```
intros H p; apply H; (assumption || intro H').
```

If assumption fails, then intro H' is used

A tactic can be left unchanged to finish every subgoal in one go

Goal: $(P \to Q) \to (P \to R) \to (P \to Q \to R \to T) \to P \to T$

```
intros H H0 H1 p.
apply H1; [idtac | apply H | apply H0]; assumption
```

More composing

If a tactic fails automatically use another tactic

```
intros H p; apply H; (assumption || intro H').
```

If assumption fails, then intro H' is used

A tactic can be left unchanged to finish every subgoal in one go

Goal: $(P \to Q) \to (P \to R) \to (P \to Q \to R \to T) \to P \to T$

```
intros H H0 H1 p.
apply H1; [idtac | apply H | apply H0]; assumption
```

Fail

Tactic that always fails

Goal: $(P \to Q) \to (P \to Q)$

`intro X; apply X; fail.`

This combination succeeds; there are no more subgoals after "apply X"

Goal: $((P \to P) \to (Q \to Q) \to R) \to R$

`intro X; apply X; fail.`

This combination fails; there are subgoals left after "apply X"

Fail

Tactic that always fails

Goal: $(P \to Q) \to (P \to Q)$

`intro X; apply X; fail.`

This combination succeeds; there are no more subgoals after "apply X"

Goal: $((P \to P) \to (Q \to Q) \to R) \to R$

`intro X; apply X; fail.`

This combination fails; there are subgoals left after "apply X"

Fail

Tactic that always fails

Goal: $(P \to Q) \to (P \to Q)$

`intro X; apply X; fail.`

This combination succeeds; there are no more subgoals after "apply X"

Goal: $((P \to P) \to (Q \to Q) \to R) \to R$

`intro X; apply X; fail.`

This combination fails; there are subgoals left after "apply X"

Fail

Tactic that always fails

Goal: $(P \to Q) \to (P \to Q)$

```
intro X; apply X; fail.
```

This combination succeeds; there are no more subgoals after "apply X"

Goal: $((P \to P) \to (Q \to Q) \to R) \to R$

```
intro X; apply X; fail.
```

This combination fails; there are subgoals left after "apply X"

Fail

Tactic that always fails

Goal: $(P \to Q) \to (P \to Q)$

`intro X; apply X; fail.`

This combination succeeds; there are no more subgoals after "apply X"

Goal: $((P \to P) \to (Q \to Q) \to R) \to R$

`intro X; apply X; fail.`

This combination fails; there are subgoals left after "apply X"

Fail

Tactic that always fails

Goal: $(P \rightarrow Q) \rightarrow (P \rightarrow Q)$

`intro X; apply X; fail.`

This combination succeeds; there are no more subgoals after "apply X"

Goal: $((P \rightarrow P) \rightarrow (Q \rightarrow Q) \rightarrow R) \rightarrow R$

`intro X; apply X; fail.`

This combination fails; there are subgoals left after "apply X"

Try

Combination of tactics that never fail

Goal: $(P \to Q \to R \to T) \to (P \to Q) \to (P \to R \to T)$

```
intros H H' p r.
apply H; try assumption; apply H'; assumption.
```

"try tac" behaves like "tac || idtac"

tac is either applied or the subgoal is left unchanged

Try

Combination of tactics that never fail

Goal: $(P \to Q \to R \to T) \to (P \to Q) \to (P \to R \to T)$

```
intros H H' p r.
apply H; try assumption; apply H'; assumption.
```

"try tac" behaves like "tac || idtac"

tac is either applied or the subgoal is left unchanged

Try

Combination of tactics that never fail

Goal: $(P \to Q \to R \to T) \to (P \to Q) \to (P \to R \to T)$

```
intros H H' p r.
apply H; try assumption; apply H'; assumption.
```

"try tac" behaves like "tac || idtac"

tac is either applied or the subgoal is left unchanged

Try

Combination of tactics that never fail

Goal: $(P \to Q \to R \to T) \to (P \to Q) \to (P \to R \to T)$

```
intros H H' p r.
apply H; try assumption; apply H'; assumption.
```

"try tac" behaves like "tac || idtac"

tac is either applied or the subgoal is left unchanged

Try

Combination of tactics that never fail

Goal: $(P \to Q \to R \to T) \to (P \to Q) \to (P \to R \to T)$

```
intros H H' p r.
apply H; try assumption; apply H'; assumption.
```

"try tac" behaves like "tac || idtac"

tac is either applied or the subgoal is left unchanged

Unprovalbe Propositions

There are goals with no solution at all

Even though they are valid in classical logic

Peirce's formula: $(((P \to Q) \to P) \to P)$

Truth table shows it is a valid formula

Unprovalbe Propositions

There are goals with no solution at all

Even though they are valid in classical logic

Peirce's formula: $(((P \to Q) \to P) \to P)$

Truth table shows it is a valid formula

Unprovalbe Propositions

There are goals with no solution at all

Even though they are valid in classical logic

Peirce's formula: $(((P \rightarrow Q) \rightarrow P) \rightarrow P)$

Truth table shows it is a valid formula

Unprovalbe Propositions

There are goals with no solution at all

Even though they are valid in classical logic

Peirce's formula: $(((P \to Q) \to P) \to P)$

Truth table shows it is a valid formula

Overview

Next Week:

- finish typing rules
- More details on tactics
- Composing
- Proof irrelevance

Overview

Next Week:

- finish typing rules
- More details on tactics
- Composing
- Proof irrelevance

Overview

Next Week:

- finish typing rules
- More details on tactics
- Composing
- Proof irrelevance

Overview

Next Week:

- finish typing rules
- More details on tactics
- Composing
- Proof irrelevance