

# What is an implication?

- Two propositions, P and Q, can form a new proposition  $P \rightarrow Q$ .
  - Read this as "P implies Q"
- A proof of the proposition  $P \to Q$  converts a proof of P into a proof of Q.
- Consider the following two premises:
  - 1. If it's raining, then the streets are wet.
  - 2. It's raining.
- From the above premises we conclude that "the streets are wet"
  - Here, "it's raining" is P and "the streets are wet" is Q.
  - It's raining implies the streets are wet.
- From this implication, we know whenever it rains, the streets are wet.
  - Remember that the truth of our propositions depends on our domain

# What an implication isn't

- It's raining *implies* the streets are wet.
- What does this implication allow us to conclude when it's not raining?
  - Nothing
- Or in the language of logic:  $(P \rightarrow Q) \nrightarrow (Q \rightarrow P)$ 
  - (*P* implies *Q*) does *not* imply (*Q* implies *P*).

### Modus Ponens

• Fancy phrase meaning: if we know  $P \rightarrow Q$  is true and we know P is true, then we know Q is true

```
{ P Q : Prop }, pfPtoQ : P \rightarrow Q, pfP : P \rightarrow Q = Prop }, pfPtoQ : P \rightarrow Q, pfP : P \rightarrow Q = Prop } (\rightarrow-elim)
```

As a Lean function:

# Analysis of Lean function (Elimination "rule")

• Lean function:

```
def arrow elim 
 (R \overline{W}: Prop) (pfRtopfW : R \rightarrow W) (pfR : R) 
 : W := pfRtopfW pfR
```

- This function takes two propositions (R and W), a proof of the implication  $R \to W$ , a proof of R and then provides a proof of W.
  - The implication  $R \to W$  is itself a program that converts a proof of R into a proof of W!
- I.e., "If you have a function that can turn any proof of R into a proof of W, and if you have a proof of R, then you obtain a proof of W, and you do it in particular by applying the function to that value."

## Modus Tollens

• Fancy phrase meaning: if we know  $P \rightarrow Q$  is true and we know Q is not true, then we know P cannot be true

```
{ P Q : Prop }, pfPtoQ : P \rightarrow Q, pfnQ : \negQ 
----- (modus-tollens) 
pfnP: \negP
```

- If we know that "if it's raining, then the streets are wet", and we know that "the streets are not wet", then we know "it's not raining"
- This relies on proof by contradiction, which requires classical logic

# Creating an implication program (Introduction "rule")

Recall our tactic and commutes:

```
def and_commutes { P Q: Prop } (paq: P \ Q) :=
   and.intro
   (and.elim_right paq)
   (and.elim_left paq)
```

- This is a program for converting  $P \wedge Q$  into  $Q \wedge P$ .
  - I.e., this program is (approximately) of type  $P \land Q \rightarrow Q \land P$ .

# Quick-and-dirty declaration of implication

• We can use the Lean keyword variable to introduce variables of whatever type we choose:

```
variables P Q : Prop variable impl : P \rightarrow Q variable pfP : P #check impl pfP
```

Note that we should be careful!

```
variable fimpt : true \rightarrow false theorem zeqo : (0 = 1) := false.elim (fimpt true.intro) #check zeqo
```

# What implications can be proved?

- Another way to read  $P \to Q$  is "if P (is true) then Q (is true)."
- We now ask which of the following implications can be proved?
  - true → true
    - if true (is true) then true (is true)
  - true → false
    - if true (is true) then false (is true)
  - false → true
    - if false (is true) then true (is true)
  - false → false
    - if false (is true) then false (is true)
- What does your intuition tell you?

## Proof of true → true

```
def timpt ( pf_true: true ) : true := pf_true
#check timpt
```

## Proof of true → false

## Proof of false → true

```
def fimpt ( pf_false: false ) : true := true.intro
#check fimpt
```

## Proof of false → false

```
def fimpf (pf_false: false ) : true := pf_false
#check timpt
```

## Truth table for implication

• We summarize our findings in the following table for implication.

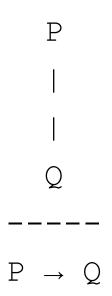
Implication	Proof	Implication value
true → true	Return passed-in proof	true
true → false	No valid proof	false
false → true	true.intro	true
false → false	Return passed-in proof	true

- Remember: implications have truth values
  - Implications are propositions themselves
- Important to note: a proposition implies false iff if the proposition is false, but implying true does not mean the proposition is true

#### → introduction rules

- The  $\rightarrow$  introduction rules say that if assuming that there is proof of P allows you to derive a proof of Q, then one can derive a proof of  $P \rightarrow Q$ , discharging the assumption.
- To represent this rule as an inference rule, we need a notation to represent the idea that from an assumption that there is a proof of P one can derive a proof of Q. If one has such a derivation then one can conclude  $P \rightarrow Q$ .
  - The derivation is in essence a program
  - The program is the proof of the proposition, which is of the type,  $P \rightarrow Q$ .

## → introduction notation



• The proof of a proposition,  $P \to Q$ , in Lean, is a program that takes an argument of type P and returns a result of type Q.

# Alternate formulation (Classical logic)

Alternate view of implication truth table

$P \rightarrow Q$	P = true	P = false
Q = true	true	true
Q = false	false	true

• What is the truth table for  $P \lor Q$  ("P or Q")?

PVQ	P = true	P = false
Q = true	true	true
Q = false	true	false

• What is the truth table for  $\neg P \lor Q$  ("(not P) or Q")?

$\neg P \lor Q$	P = true	P = false
Q = true	true	true
Q = false	false	true

