ECE 108: Discrete Math and Logic I
Chapter 3: Predicate Logic
(Slides)

Mark Aagaard
maagaard@uwaterloo.ca

January 19, 2025

3.1 Introduction

Motivation

- Every person drinks water.
- Amir is a person.
- Therefore, Amir drinks water.

In propositional logic, the best we can do to express this is:

- A = Every person drinks water.
- B = Amir is a person.
- C = Amir drinks water.

The top-level proposition is $A \wedge B \rightarrow C$.
To prove this proposition, we need information about A, B, and C.
We need to be able to reason about:

- objects
- groups of objects
- attributes of objects
- relationships between objects

Motivation

We need to be able to reason about:

• **individual objects**	Waterloo, Toronto
• **groups of objects**	cities, vehicles
• **attributes/properties** of objects	"Waterloo is boring"
• **relations** between objects	"Waterloo is west of Toronto"
• **all objects**	"All cities are boring"
• **some object(s)**	"Some city is exciting"

Predicate logic adds: **types**, **functions** and **quantifiers** to capture these concepts.

> **Note:** A predicate is a function whose return type is proposition.

Think of typed predicate logic as a programming language for reasoning about objects.

Overview

Quantifiers Types Functions

- Types: *person*, *liquid*
- Function: $drinks : (person, liquid)) \rightarrow Prop$
- **Universal quantification**
  - **Everyone drinks water.**
  - every $x : person$. drinks(x, water)
  - $\forall\ (x : person),\ drinks(x, water)$
- **Existential quantification**
  - **Someone drinks tea.**
  - some $y : person$. drinks(y, tea)
  - $\exists\ (x : person),\ drinks(x, tea)$

3.1.1 Types

Do the following both make sense?

- $drinks(Bill, tea)$
- $drinks(Vancouver, helium)$

Programming languages use types to distinguish between different kinds of objects, which helps catch errors.
We can do the same thing in logic.

Type Expressions

In a formula, $(expr : T_1)$ means that the expression $expr$ has type T_1.
Categories of types:

- **basic type** : Atomic types (doesn't decompose into simpler types.
  - Examples of basic types: Booleans, integers, strings, cities, vehicles.
  - Example type assertions: $(3 : \mathbb{Z})$ and $(blue : colour)$
- **product type** : Types of pairs, triples, and tuples in general.
  Example: $(volvo, green) : (vehicle \times colour)$
  a pair consisting of a vehicle and colour.
- **function type** : A function goes from one type to another.
  Example: a function that
  - takes a string and an integer as arguments
  - returns a color

  has type: $(string \times \mathbb{Z}) \rightarrow colour$

Uninterpreted Types

- In logic, we often do not need to give full definitions of types.
- An **uninterpreted type**: we declare that the type exists, and may use it functions, but we do **not** define how the type is implemented.
- In Lean4, types are just another category of value.

We declare uninterpreted types as variables of type *Type*.

Example 2.1:
Declare a type for students and a type for courses.

Var (*student* : *Type*)

Var (*course* : *Type*)

Declare a variable of type *student*:

Var (*Xi* : *student*)

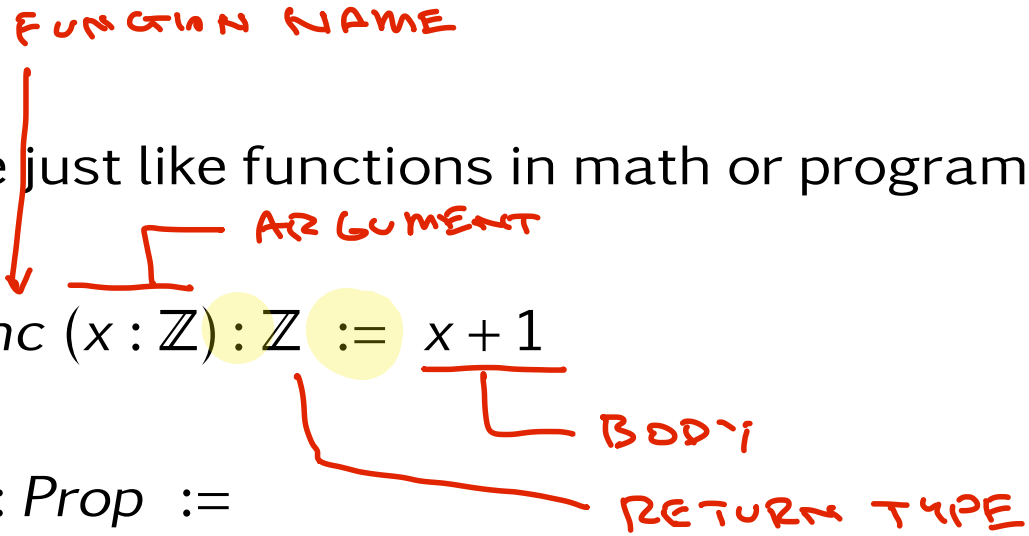
Rules for Types

1. Each symbol, object, function, *etc.* has exactly one type within an individual formula or proof.
   - In one formula, 1 may be an integer and in another formula, 1 may be a natural number.
   - Within a single formula, 1 **may not** be both a natural number an integer.
   - If you need both naturals and integers (or reals and integers) in a single proof, use predicates for the smaller types.
2. Each type must be non-empty.
   - This is a technical requirement that is needed so that things we think should be true are in fact true in the logic.

     For example:

     - If P is true for all elements in type Ty, then there is an element of type Ty for which P is true.
     - $(\forall(x : Ty),\ P(x)) \rightarrow (\exists(x : Ty),\ P(x))$
     - If Ty could be empty, then this statement is false.

3.1.2 Functions



Functions in logic are just like functions in math or programming.

Examples

- Increment: **Def** $inc\ (x : \mathbb{Z}) : \mathbb{Z}\ :=\ x+1$
- Invert:

  Def $g\ (x : Prop) : Prop\ :=$

  $\texttt{if}\ x\ \texttt{then}\ \textbf{false}\ \texttt{else}\ \textbf{true}$
- Fibonacci:

  Def $fib\ (x : \mathbb{N}) : \mathbb{N}\ :=$

  $\texttt{if}\ \ x = 0 \vee x = 1\ \ \texttt{then}\ \ 1\ \texttt{else}\ \ fib\ (x-2) + fib\ (x-1)$

Notation

Note: In functional programming, we do **not** need parentheses around arguments to functions Good: $f\ (x)$ Good: $f\ x$

Note: In functional programming, we do **usually** pass arguments individually:

$f\ x\ y$ A function with **two** arguments

$f\ (x, y)$ A function with **one** argument that is a pair of values.

Note: In Lean4, there must be a **space** between a function and its arguments. BAD: $f(x)$ Good: $f\ x$

f ␣ (x)

Two options to define the type of a function:

- Show the arguments with their types and the return type: $f\ (x : \mathbb{N}) : \mathit{Prop}$
- Show the function's type expression: $f : \mathbb{N} \to \mathit{Prop}$

Pattern Matching

- Lean4 has powerful **pattern matching** features.
- A simple example:

Def *fib* : $\mathbb{N} \to \mathbb{N}$
| $0 \Rightarrow 1$
| $1 \Rightarrow 1$
| $x + 2 \Rightarrow fib\ (x-2) + fib\ (x-1)$

BODY (RESULT EXPRESSION)

PATTERN MATCHED AGAINST ARGUMENT

- There are more features described in the Lean4 documentation.
- The focus of this course is on concepts, not how to drive Lean4. We'll generally stick with the minimal features of Lean4.

3.1.3 Predicates

A **predicate** describes:

- an attribute (property) of an object
- a relationship between two or more objects

Example 2.2:

English	Logic
"Bill" is a person	$Bill : person$
"child" is a predicate about a person	$child : person \rightarrow Prop$
"Bill is a child"	$child\ Bill$
"apples" are a food	$apples : food$
"eats" is a predicate about a person and a food	$eats : person \rightarrow food \rightarrow Prop$
"Bill eats apples"	$eats\ Bill\ apples$

3.1.4 Quantifiers

Motivation

We started this topic by saying predicate logic could be used to describe valid arguments such as:

- Every person drinks water.
- Amir is a person.
- Therefore, Amir drinks water.

How do we formalize "Every person drinks water"?

Quantifiers

A quantifier is always followed by a variable with a type and then a predicate

Name	Symbol	Description
Universal quantification	$\forall (x : Ty),\ P\ x$	$P\ x$ is true for all objects of type Ty ("every", "all", "for all")
Existential quantification	$\exists (x : Ty),\ P\ x$	$P\ x$ is true for at least one object of type Ty ("some", "there exists")

The formula: $\exists (x : Ty),\ P\ x$ is read as:
"There exists x of type Ty, such that $P\ x$".

Examples: Quantifiers

Example 2.3: Everyone is a tall adult.

$\forall\ (x : \text{person}),\ \text{tall}\ x \wedge \text{adult}\ x$

Example 2.4: Someone is happy or hungry.

$\exists\ (x : \text{person}),\ \text{happy}\ x \vee \text{hungry}\ x$

Example 2.5: Every child likes Mickey Mouse.

① $\forall\ (p : \text{person}),\ \text{child}\ p \to \text{likes}\ p\ \text{Mickey}$

Note: To make this fit on one slide, we haven't written the declarations of the predicates. On assessments, you must include the declarations.

② $\forall\ (p : \text{child}),\ \text{likes}\ p\ \text{Mickey}$

Every class that a student takes is taught by an instructor.

OPTION 1:

$\forall$ (c: class), $\forall$ (s: stu),

takes s c $\rightarrow$

$\exists$ (i: instr), teaches i c

SEPARATE TYPES FOR stu AND instr,

$\circ$ MARK CANNOT BE BOTH AN instr AND A stu.

Var (Mark: stu)

X Var (Mark: instr) ← ERROR

OPTION 2: ATTRIBUTES / PREDICATES

$\forall$ (c: class), $\forall$ (s: person)

isStu s $\wedge$ takes s c $\rightarrow$

$\exists$ (i: person), isInstr i $\wedge$ teaches i c

MARK CAN BE BOTH A STU AND AN INSTR

Var (Mark: person)

isStu Mark $\wedge$ isInstr Mark

Lec 9 01-23

Examples

Example 2.6: Given the environment:

$older\ x\ y : person) : Prop$ – – means x is older than y

Are the two formulas below equivalent?

1. $\forall(x : person),\ \exists(y : person),\ older\ x\ y$
2. $\forall(y : person),\ \exists(x : person),\ older\ x\ y$

Everyone is older than someone.

Everyone is younger than someone.

- For everyone person, there is someone who is older.

3. $\forall (y : \text{person}),\ \exists (x : \text{person}),\ \text{older}\ y\ x$

1 & 3 ARE EQUIVALENT.

4. There is someone who is older than everyone else.

$\exists x, \forall y,\ \text{older}\ x\ y$ ↩

DOES NOT INCLUDE "else"

$\exists x, \forall y,\ y \neq x \rightarrow \text{older}\ x\ y$

↳ "Everyone else"

3.1.5 Technicalities

Binding of Variables

Given the environment:

$talks\_to\ (x\ y : person) : Prop$ – means x talks to y

Do the following formulas have the same meaning?

1. $\forall (x : person),\ \exists (y : person),\ talks\_to\ x\ y$
2. $\forall (x : person),\ \exists (x : person),\ talks\_to\ x\ x$

where

1. For everyone, there is someone that they talk to.
   (Everybody talks to someone.)
2. **No**
   ~~For everyone,~~ there is someone who talks to themselves.

It is important which quantifier is associated with a variable.

Alpha Renaming

Alpha renaming: changing **only** the variables in an expression.

Example 2.7: Which ~~two~~ one the expressions below ~~have the same~~ has a different meaning?

- $\forall(x : person),\ \exists(y : person),\ older\ x\ y$
- $\forall(a : person),\ \exists(b : person),\ older\ a\ b$
- $\forall(p : person),\ \exists(q : person),\ older\ q\ p$ ✓

A variable name is just a placeholder.
Replacing a variable

- throughout an entire formula
- with a variable that does not already appear in the formula

will not change the meaning of the formula.

Note: Technically, the new variable **may** be used in the **formula**, so long as it is **not** be used within the **scope** of the variable that it is replacing.

Scope of Quantifiers

Definition 2.1: The **scope of a quantifier** is the subformula over which the quantifier applies in the given formula.
The scope of the quantifier extends to the right end of the formula unless an unmatched right parenthesis stops it.

Bound and Free Variables

- A variable is **bound** to the closest quantifier to the left of its name whose scope it is within. All variable occurrences that are bound to the same quantifier represent the same object.
- An occurrence of a variable is bound if it falls within the scope of a quantifier for that variable.
- An occurrence of a variable is **free** if it does not fall within the scope of a quantifier for that variable.
- A wff is **closed** if it contains no free variables.
- We will be working with closed wff.
- Whether a variable is free or bound depends on the occurrence of the variable in the formula, not on its name. The same variable name can be free or bound.

VALUE OF FREE VAR COMES FROM ENVIRONMENT, JUST LIKE IN PROPOSITIONAL LOGIC.

Parenthesization

- Minimal parentheses:
  $\forall (y : Ty_1),\ P\ y \wedge \forall (z : Ty_2),\ Q\ z \rightarrow R\ z$
- Add parentheses around $\forall$:
  $\forall (y : Ty_1),\ P\ y \wedge (\forall (z : Ty_2),\ Q\ z) \rightarrow R\ z$
  This changes the meaning to:
  $\forall (y : Ty_1),\ ((P\ y \wedge (\forall (z : Ty_2),\ Q\ z)) \rightarrow R\ z)$
  The variable z is now free.

Syntactic Conventions

- Two or more variables with the same quantifier may be written together with a single quantifier.
  - $\forall(x : person),\ \forall(y : person), \ldots$

    may be written as:
  - $\forall(x : person)\ (y : person), \ldots$

    may be written as:
  - $\forall(x\ y : person), \ldots$

3.2 Predicate Logic Proofs

- All of the rules from propositional logic may be used in predicate logic.
- Add elimination rules for quantifiers.
- Add rules to make proofs shorter.
  These rules could be defined in terms of the existing rules.

3.2.1 Quantifier Elimination Rules

Forall Elimination on Goal

The first steps in a proof usually use this rule,
because theorems usually have the form $\forall(x : Ty),\ \ldots \to \ldots$.

$$\begin{array}{rl} \vdash & \forall(x : Ty),\ P\ x \\ \textbf{By} & \text{ForallElim Goal} \\ \text{Asm 1:} & \textbf{Var}\ (x : Ty) \\ \vdash & P\ x \end{array}$$

- Goal is to prove P is true for all values x.
- We prove $P\ x$ for an arbitrary value x
  (We know nothing about x other than it's type.)
- x must be a fresh variable (prevent variable capture)

Forall Elimination on Goal (cont'd)

$$\vdash \quad \forall (x : Ty),\ P\ x$$

By ForallElim Goal

Asm 1: **Var** $(x : Ty)$

$$\vdash \quad P\ x$$

- Justification:
  - We constructed a proof that P is true for a variable about which we know nothing other than its type.
  - If we are given any value of of type Ty, we can use the same proof to prove that P is true for that value.
  - Thus, we have constructed a proof that P is true for all values of type Ty.

▷ To prove that P is true for all values of x, it is sufficient to prove that P is true for a value that we don't know anything about.

Forall Elimination on Assumption

Asm 1: $\forall (x : \mathbb{Z}),\ P\ x$

By ForallElim Asm 1 Using $x = 4$

Asm 2: $P\ 4$

- We know that $P\ x$ is true for **every** x of type $\mathbb{Z}$.
- We get to **choose** how to **instantiate** x when we create Asm 2.
- The instantiation says to use 4 in place of x.
- To prevent variable capture within $P\ x$, we might have to rename some quantified variables in $P\ x$ (Section ??).

Example: Proof

Example 2.8: Given the environment:

Var $(Ty : Type)$

Var $(p\ \ q : Ty \to \mathbb{B})$ Prop

Prove:

$(\forall (x : Ty),\ p\ x \to q\ x)$

$\to$ ← OUTERMOST

$(\ (\forall (x : Ty),\ p\ x) \to (\forall (y : Ty),\ q(y))\ )$

p = is tall

q = drinks O.J.

everyone who is tall drinks O.J.

if everyone is tall then everyone drinks O.J.

By ImplIn Goal

Asm1 : ∀(x:Ty), p x → q x

$\vdash (\forall\ (x : T_y),\ p\ x) \Rightarrow (\forall (y : T_y),\ q\ y)$

By ImplElim Goal

Asm 2: $\forall\ (x : T_y),\ p\ x$

$\vdash\ \forall (y : T_y),\ q\ y$

By ForallElim Goal

Asm 3: Var $(y : T_y)$

$\vdash\ q\ y$

Asm 3: Var $(z : T_y)$

$\vdash\ q\ z$

Asm 3: Var $(x : T_y)$

$\vdash\ q\ x$

```
By FonclElim Asm 2 Using x=y
Asm 4: p y
By FonallElim Asm 1 Using x=y
Asm 5: p y → q y
  By ImplElim Asm 5 Using Asm 4
 Asm 6: q y
  QED Asm 6
```

Lec 10: 01-28

Sketch Out a Proof Strategy

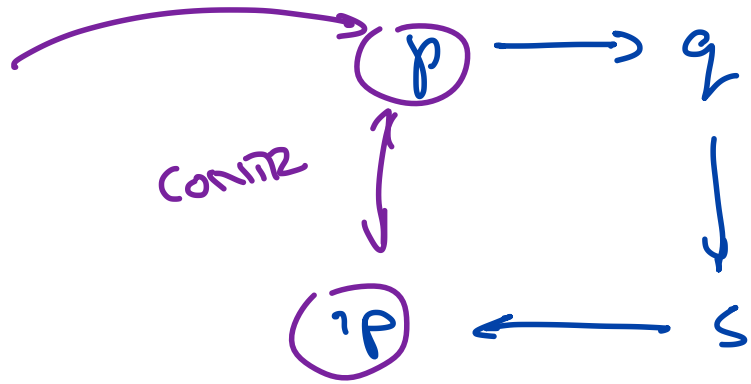
Asm 1: $p \to q$

Asm 2: $p \vee s$,

Asm 3: $s \to \neg p$

Asm 4: $q \to s$

⊢ $s \vee r$



By OrElim Asm 2

{ Subproof 1: "lhs"

Asm 1/1: p

By ImplElim Asm 1 Using Asm 1/1

Asm 1/2: q

By ImplElim Asm 4 Using Asm 1/2

```
    Asm 1/3 : s
    By OrElimLeft Goal
    ⊢ s
    QED Asm 1/3

Subproof 2: "rhs"
    Asm 2/1 : s
    By OrElimLeft Goal
    ⊢ s
    QED Asm 2/1
}
```

ALTERNATIVE

```
By OrIntroLeft
    Asm 1/3 using r
Asm 1/4 s ∨ r
QED Asm 1/4
```

	Asm	Goal
AND	Asm1: p ∧ q By AndElim Asm1	⊢ p ∧ q By AndElim Goal { Subproof 1: "lhs" ⊢ p Subproof 2: "rhs" ⊢ q }
OR	Asm1: p ∨ q By OrElim Asm1 { Subproof 1: "lhs" Asm1/1: p Subproof 2: "rhs" Asm2/1: q	⊢ p ∨ q By OrElimLeft Goal ⊢ p
∀	Asm1: ∀ (x:ℕ), p x By ForallElim Asm1 Using x = 17 Asm2: p 17	⊢ ∀ (x:ℕ), p x By ForallElim Goal Asm1: Var (x:ℕ) ⊢ p x






∃

Asm 1 : ∃ (x : ℕ), p x

By ExistsElim Asm 1

Asm 2 : Var (x : ℕ)

Asm 3 : p x

⊢ ∃ (x : ℕ), p x

By ExistsElim Goal

Using x := 16

⊢ p 16

Example Proof 3

Example 2.9: Prove the reverse direction of the previous theorem:

$\vdash ((\forall(x : Ty),\ p\ x) \to (\forall(y : Ty),\ q(y)))$

$\to$

$(\forall(x : Ty),\ p\ x \to q\ x)$

By ImplElim Goal

Asm1 : $(\forall\ (x:T_y),\ p\ x) \to (\forall\ (y:T_y),\ q\ y)$

$\vdash \forall\ (x:T_y),\ p\ x \to q\ x$

By ImplElim Asm1

X Asm 2: Uh oh, Don't know lhs of Asm 1 is true, so would need subproof.

Try ImplElim Goal

By ForallElim Goal

Asm 2 : Var $(x : T_x)$

$\vdash$ p x → q x

By ImplElim Goal

Asm 3 : p x

$\vdash$ q x

Asm1 : $(\forall\ (x : T_x),\ p\ x) \rightarrow (\forall\ (y : T_y),\ q\ y)$

WE ARE STUCK

CONCRETE EXAMPLE

$p\ x := x > 2 \wedge \mathrm{isPrime}\ x$

$q\ x := \mathrm{isOdd}\ x$

$$(\forall x, (x > 2 \wedge \mathrm{isPrime}\ x) \to \mathrm{isOdd}\ x)$$

$$\to$$

$$((\forall x, x > 2 \wedge \mathrm{isPrime}\ x) \to (\forall x, \mathrm{isOdd}\ x))$$

BAD EXAMPLE, BOTH SIDES OF $\to$ ARE TRUE

Concrete Ex #2

p x := does hw

q x := is Engr

Asm1 : (∀ (x : Stu), does hw x)
→ (∀ (x : Stu), is Engr x)

⊢ ∀ (x : Stu), does hw x → is Engr x

We can't prove this,

to prove that a stu is an engineer,
we must prove that ALL stus are engineers

Suboptimal example, because doesn't match real world.

Concrete Ex #3

$p\ x$:= helps clean the kitchen

$q\ x$:= spends less than 10 minutes cleaning the kitchen

$$(\forall\ x,\ p\ x) \rightarrow (\forall\ x,\ q\ x)$$

This is true: If everyone cleans, then everyone cleans for a short time.

$$(\forall\ x,\ p\ x \rightarrow q\ x)$$

This is false: If only one person cleans, then they must clean for a long time.

WE CAN'T PROVE

$$(\forall x, p\,x) \rightarrow (\forall x, q\,x)$$

$$\rightarrow$$

$$(\forall x, p\,x \rightarrow q\,x)$$

CONCRETE EXAMPLE #3, FIRST DIRECTION

$$(\forall x, p\,x \to q\,x) \to (\forall x, p\,x) \to (\forall x, q\,x)$$

THIS SAYS:

IF IT IS TRUE THAT:

IF SOMEONE CLEANS, THEN THEY CLEAN FOR LESS THAN 10 MINS.

THEN IT IS TRUE THAT:

IF EVERYONE CLEANS, THEN EVERYONE CLEANS FOR $\leq$ 10 mins.

THIS STATEMENT IS <u>TRUE</u> IN THE REAL WORLD BECAUSE THE ANTECEDENT IS <u>FALSE</u>.

3.2.1.1 Example: Syllogism Revisited

Example 2.10:

Complete the proof below.

Asm 1: $\forall(x : Ty),\ p\ x \to q\ x$

Asm 2: **Var** $(y : Ty)$

Asm 3: $p\ y$

$\vdash$ $q\ y$

SKIP

Lec-11 01-29

Exists Elimination on Assumption

Asm 1: $\exists(x : Ty),\ P\ x$

By ExistsElim Asm 1

Asm 2: **Var** $(x : Ty)$

Asm 3: $P\ x$

BECAUSE WE KNOW THAT THERE EXISTS A VALUE OF x FOR WHICH P IS TRUE, WE CAN ASSUME THAT P IS TRUE FOR A VARIABLE x.

- Assumption 1 says that there is some x such that $P\ x$.
- In Asm 2, x represents an arbitrary element of type Ty.
- We don't know anything about which x satisfies $P\ x$, so we get no control over x in Asm 2.
- In Asm 2, x must be a fresh variable that does not appear in any assumption or the goal.
- We might have to rename x to prevent variable capture.

Exists Elimination on Goal

$\vdash \exists (x : \mathbb{Z}),\ P\ x$

By ExistsElim Goal Using 4

$\vdash P\ 4$

$x =$

- To prove that there exists an x such that $P\ x$ is true, we provide a **witness** to instantiate x.
- This proof rule is the dual of ForallElim on an assumption.
- To prevent variable capture within $P\ x$, we might have to rename some quantified variables in $P\ x$.

Example: ForallElim and ExistsElim

Example 2.11:

~~Are we stuck?~~ Can we complete the proof?

Asm 1: $\forall(x : Ty),\ p\,x \to q\,x$

$\vdash\ \exists(x : Ty),\ p\,x \wedge q\,x$

WE ARE STUCK.

WE NEED A VARIABLE OR A VALUE TO INSTANTIATE $\forall$ IN ASM OR GOAL.

VARIABLE OR VALUE MUST BE DECLARED IN PROOF BEFORE WE CAN USE IT.

Example: ForallElim and ExistsElim Swapped

Swap the goal and assumption from the previous example.

Example 2.12:

Can we complete complete the proof below?

Asm 1: $\exists(x : Ty),\ p\ x \wedge q\ x$

$\vdash$ $\forall(x : Ty),\ p\ x \to q\ x$

Are we stuck?

By ExistsElim Asm 1

Asm 2: Var $(x : T_y)$

Asm 3: $p\ x \wedge q\ x$

By ForallElim Goal

Asm 4: Var (x1 : T_T)

⊢ p x1 → q x1

MUST BE "FRESH" (NOT YET USED IN PROOF)

By AndElim Asm 3

Asm 5: p x

Asm 6: q x

By ImplElim Goal

Asm 7: p x1

⊢ q x1

WE ARE STUCK.

FROM Asm 6, WE KNOW q x. OUR GOAL IS q x1. x & x1 ARE DIFFERENT VARIABLES.

- WE DON'T KNOW ANYTHING ABOUT THE RELATIONSHIP BETWEEN x & x1.
- IT MIGHT BE x=x1 OR IT MIGHT BE x≠x1.

Can we complete complete the p

Asm 1: $\exists (x : Ty),\ p\ x \wedge q\ x$

$\vdash\ \forall (x : Ty),\ p\ x \rightarrow q\ x$

Asm1 : ∀ (x : T_y), ¬ p x → q x

Asm2 : ∀ (x : T_y), p x ∨ ¬ p x

⊢ ∀ (x : T_y), p x ∨ q x

By ForallElim Goal -- ONLY RULE THAT WE -- CAN APPLY.

Asm3 : ∀ an (x : T_y)

⊢ p x ∨ q x

By ForallElim Asm 2 Using x := x

Asm 4 : p x ∨ ¬ p x

By OrElim Asm 4

{ Subproof 1 : "lhs"

```
Asm 1/1: p x
By OrElim Left Goal
⊢ p x
QED Asm 1/1
Subproof 2: "rhs"
Asm 2/1: ¬ p x
By ForallElim Asm 1 Using x := x
Asm 2/2: ¬p x → q x
By ImplElim Asm 2/2 Using Asm 2/1
Asm 2/3: q x
QED Asm 2/3.
```

Can we complete complete the p

Asm 1: $\exists (x : Ty),\ p\ x \wedge q\ x$

$\vdash\ \forall (x : Ty),\ p\ x \rightarrow q\ x$

Asm1 : $\forall\ (x : T_y),\ p\ x \rightarrow q\ x$

Asm2 : $\forall\ (x : T_y),\ p\ x \vee \neg\ p\ x$

$\vdash\ \forall\ (x : T_y),\ p\ x \vee q\ x$

By ForallElim Goal -- ONLY RULE THAT WE -- CAN APPLY.

Asm3 : Var $(x : T_y)$

$\vdash\ p\ x \vee q\ x$

ALTERNATIVE STRATEGY

By OrElimLeft Goal

$\vdash\ p\ x$

① ForallElim, OrElim Asm 2

② FIRST SUBPROOF: Asm 1/1 : $p\ x$

QED Asm 1/1 

(3) SECOND SUBPROOF: Asm 2/1: $\neg p x$

$\vdash p x$ 

3.2.2 Substitution

- Substitution is one of the most flexible and powerful proof rules.
- $P[B/A]$ means to replace each occurence of A in P with B.
- Examples of $P[B/A]$:

Substitute 4 in for b:

$$\begin{aligned} & (a+3 > b+1-2)[4/b] \\ = \quad & (a+3 > 4+1-2) \end{aligned}$$

Substitute $z-3$ in for $b+1$:

$$\begin{aligned} & (a+3 > b+1-2)[(z-3)/(b+1)] \\ = \quad & (a+3 > z-3-2) \end{aligned}$$

Asm 1: $A = B$

Asm 2: P

$\vdash$ Q

By Subst Asm 1 Into Asm 2

Asm 3: $P[B/A]$

Asm 1: $A = B$

Asm 2: P

$\vdash$ Q

By Subst Asm 1 Into Goal

$\vdash$ $Q[B/A]$

- Bound variables in the assumptions or goal might need to be renamed to prevent variable capture.

Substitute a Definition from the Environment

rename "f" as "g"

The environment defines a function f.

Def $f\ (x : \mathbb{N})\ :\ \mathbb{N}\ :=\ x+1$

Asm 1:	$\ldots f\ (z+3)\ldots$
$\vdash$	Q
By	Subst f Into Asm 1
Asm 2:	$\ldots z+3+1\ldots$

Asm 1:	P
$\vdash$	$\ldots f(z+3)\ldots$
By	Subst f Into Goal
$\vdash$	$\ldots z+3+1\ldots$

Substitute an Expression

Asm 1: $\ldots x \ldots$

$\vdash$ Q

By Subst $x = y$ Into Asm 1 By *justification*

Asm 2: $\ldots y \ldots$

Asm 1: $\ldots z + 3 + h \ldots$

$\vdash$ Q

By Subst $3 = 2 + 1$ Into Asm 1 By Arith

Asm 2: $\ldots z + 2 + 1 + h \ldots$

Reverse Substitution

All variations of Subst can also be used with SubstRev, which replaces the rhs with the lhs.

3.2.3 Assert

This proof rule allows us to add a proposition to the assumptions, provided that we justify the proposition.

$\vdash P$

By Assert Q by *justification*

Asm 1: Q

$\vdash P$

$\vdash P$

By Assert Q

Sub proof 1: Prove Q

$\vdash Q$

Sub proof 2: Assume Q

2/1 Q

$\vdash P$

Justification must be one of:

- "definition": Q is a definition in the environment.
- "theorem": Q is a previously proved theorem.
- "Arith": Q is an "obvious" arithmetic fact

Assert (cont'd)

- A shortcut, to avoid writing the expression that is asserted twice, assert the assumption that is added to the proof.
- To assert Q without having to write Q in the proof rule, use the templates below.

$\vdash \quad P$

By Assert Asm 1 by *justification*

Asm 1: Q

$\vdash \quad P$

$\vdash \quad P$

By Assert Asm 2/1

Sub proof 1/ Prove the assertion

$\vdash \quad Q$

Sub proof 2/ Assume Asm 2/1

2/1 Q

$\vdash \quad P$

3.2.4 Probably Helpful Rules

Elims

Apply as many ForallElim and/or ExistsElim elimination rules as you wish to an assumption or the goal.

$\vdash \quad \forall (x : \mathbb{N}),\ p\ x \to q\ x$

By Elims Goal

Asm 1: **Var** $(x : \mathbb{N})$

Asm 2: $p\ x$

$\vdash \quad q\ x$

Asm 1: $\forall (x : \mathbb{N}),\ p\ x \to q\ x$

Asm 2: $p\ 5$

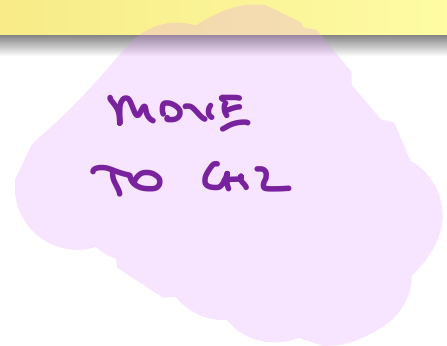
$\vdash \quad R$

By Elims Asm 1 Using x=5, Asm 2

Asm 3: $q\ 5$

$\vdash \quad R$

ImplElim on Assumption Using the Goal



Work backwards from the goal toward the assumptions.

Asm 1: $P \to Q$

$\vdash$ Q

By ImplElim Asm 1 Using Goal

$\vdash$ P

Iff Elimination Assumption

move to Ch 2

IffElim is the same as ImplElim, just apply to $\leftrightarrow$

Asm 1: P

Asm 2: $P \leftrightarrow Q$

By IffElim Asm 2 Using Asm 1

Asm 3: Q

IffElimRev goes the reverse direction.

Asm 1: Q

Asm 2: $P \leftrightarrow Q$

By IffElimRev Asm 2 Using Asm 1

Asm 3: P

Apply IffElim to Iff in Assumption (cont'd)

IffElim applied to an if-and-only-if in an assumption with "Using Goal".

Asm 1: $P \leftrightarrow Q$

$\vdash$ Q

By IffElim Asm 1 Using Goal

$\vdash$ P

IffElimRev goes the reverse direction.

Asm 1: $P \leftrightarrow Q$

$\vdash$ P

By IffElimRev Asm 1 Using Goal

$\vdash$ Q

IffElim on Goal

IffElim on a goal gives two subgoals, one for each direction of implication.

$\vdash \quad P \leftrightarrow Q$

By IffElim Goal

Sub proof 1/ "$\rightarrow$"

$\vdash \quad P \rightarrow Q$

Sub proof 2/ "$\leftarrow$"

$\vdash \quad Q \rightarrow P$

Notice the difference between this slide and the previous slide:

- Previous: $\leftrightarrow$ in assumption: produces one goal
- This: $\leftrightarrow$ in goal: produces two goals

OrElimLeft/Right on an Assumption

When we have a disjunction and know that one of the two disjuncts is false, we can conclude that the other disjunct must be true.

Asm 1: $\neg P$
Asm 2: $P \vee Q$
By OrElimRight Asm 2 Using Asm 1
Asm 3: Q

Asm 1: $\neg Q$
Asm 2: $P \vee Q$
By OrElimLeft Asm 2 Using Asm 1
Asm 3: P

3.2.5 Probably Irrelevant Rules

Let

$$\begin{array}{rl} \vdash & P \\ \textbf{By} & \text{Let } (x : Ty) := Q \\ \text{Asm 1:} & \textbf{Var } (x : Ty) := Q \\ \vdash & P \end{array}$$

- x must be a fresh variable
- Q may be any expression
- Q may include variables that are already in an assumption or the goal