



# CS 3110

## Proofs are Programs

Prof. Clarkson  
Fall 2019

Today's music: *Proof* by Paul Simon

# **CLICKER QUESTION 1**

# Review

Previously in 3110:

- Proofs about programs

Today: Proofs **are** programs

Types = Propositions

**ACT I**

# Three innocent functions

```
let apply f x = f x
```

```
let const x = fun _ -> x
```

```
let subst x y z = x z (y z)
```

# Three innocent functions

```
let apply f x = f x
```

```
: ('a -> 'b) -> 'a -> 'b
```

```
let const x = fun _ -> x
```

```
: 'a -> 'b -> 'a
```

```
let subst x y z = x z (y z)
```

```
: ('a -> 'b -> 'c)
```

```
-> ('a -> 'b) -> 'a -> 'c
```

# Three innocent functions

```
let apply f x = f x
```

```
: ('a -> 'b) -> 'a -> 'b
```

```
let const x = fun _ -> x
```

```
: 'a -> 'b -> 'a
```

```
let subst x y z = x z (y z)
```

```
: ('a -> 'b -> 'c)
```

```
-> ('a -> 'b) -> 'a -> 'c
```

# Three innocent ~~functions~~ propositions

```
let apply f x = f x
```

```
: ('a  $\Rightarrow$  'b)  $\Rightarrow$  'a  $\Rightarrow$  'b
```

```
let const x = fun _ -> x
```

```
: 'a  $\Rightarrow$  'b  $\Rightarrow$  'a
```

```
let subst x y z = x z (y z)
```

```
: ('a  $\Rightarrow$  'b  $\Rightarrow$  'c)
```

```
 $\Rightarrow$  ('a  $\Rightarrow$  'b)  $\Rightarrow$  'a  $\Rightarrow$  'c
```



# Three innocent ~~functions~~ propositions

`let apply f x = f x`

`: ( A  $\Rightarrow$  B )  $\Rightarrow$  A  $\Rightarrow$  B`

`let const x = fun _ -> x`

`: A  $\Rightarrow$  B  $\Rightarrow$  A`

`let subst x y z = x z (y z)`

`: ( A  $\Rightarrow$  B  $\Rightarrow$  C )`

`$\Rightarrow$  ( A  $\Rightarrow$  B )  $\Rightarrow$  A  $\Rightarrow$  C`

# Three innocent ~~functions~~ propositions

```
let apply f x = f x
```

```
: ( A  $\Rightarrow$  B )  $\Rightarrow$  A  $\Rightarrow$  B
```

```
let const x = fun _ -> x
```

```
: A  $\Rightarrow$  ( B  $\Rightarrow$  A )
```

```
let subst x y z = x z (y z)
```

```
: ( A  $\Rightarrow$  ( B  $\Rightarrow$  C ) )
```

```
 $\Rightarrow$  ( ( A  $\Rightarrow$  B )  $\Rightarrow$  ( A  $\Rightarrow$  C ) )
```

Do you recognize these propositions?

# A Sound and Complete Axiomatization for Propositional Logic

Consider the following axiom schemes:

$$\text{A1. } A \Rightarrow (B \Rightarrow A)$$

$$\text{A2. } (A \Rightarrow (B \Rightarrow C)) \Rightarrow ((A \Rightarrow B) \Rightarrow (A \Rightarrow C))$$

$$\text{A3. } ((A \Rightarrow B) \Rightarrow ((A \Rightarrow \neg B) \Rightarrow \neg A))$$

These are axioms schemes; each one encodes an infinite set of axioms:

- ▶  $P \Rightarrow (Q \Rightarrow P)$ ,  $(P \Rightarrow R) \Rightarrow (Q \Rightarrow (P \Rightarrow R))$  are instances of A1.

**Theorem:** A1, A2, A3 + **modus ponens** give a sound and complete axiomatization for formulas in propositional logic involving only  $\Rightarrow$  and  $\neg$ .

# *Modus Ponens*

$A \Rightarrow B$

$A$

---

$B$

# Three innocent functions/propositions

**let** apply f x = f x

MP as axiom

: ( A  $\Rightarrow$  B )  $\Rightarrow$  A  $\Rightarrow$  B

**let** const x = **fun** \_ -> x

: A  $\Rightarrow$  ( B  $\Rightarrow$  A )

A1

**let** subst x y z = x z (y z)

: ( A  $\Rightarrow$  ( B  $\Rightarrow$  C ) )

$\Rightarrow$  ( ( A  $\Rightarrow$  B )  $\Rightarrow$  ( A  $\Rightarrow$  C ) )

A2

# Types and propositions

Logical propositions can be read as program types, and vice versa

Type	Proposition
Type variable 'a	Atomic proposition A
Function type $\rightarrow$	Implication $\Rightarrow$

# Conjunction and truth

```
let fst (a,b) = a
```

```
: 'a * 'b -> 'a
```

```
let snd (a,b) = b
```

```
: 'a * 'b -> 'b
```

```
let pair a b = (a,b)
```

```
: 'a -> 'b -> 'a * 'b
```

```
let tt = ()
```

```
: unit
```

# Conjunction and truth

```
let fst (a,b) = a
```

```
  : (A ∧ B) ⇒ A
```

```
let snd (a,b) = b
```

```
  : (A ∧ B) ⇒ B
```

```
let pair a b = (a,b)
```

```
  : A ⇒ (B ⇒ (A ∧ B))
```

```
let tt = ()
```

```
  : true
```



# Types and propositions

Logical propositions can be read as program types, and vice versa

Type	Proposition
Type variable ' a	Atomic proposition A
Function type $\rightarrow$	Implication $\Rightarrow$
Product type *	Conjunction $\wedge$
<code>unit</code>	True

*Disjunction, false, and negation also possible; see notes*

Program types

and

logical propositions

are fundamentally the same idea

Programs = Proofs

**ACT II**

# Innocent typing rule

- Recall [lec19]
  - Static environment is a map from identifiers to types
  - Typing relation  $\text{env} \vdash e : t$  says that  $e$  has type  $t$  in environment  $\text{env}$
- Typing rule for function application:  
if  $\text{env} \vdash e1 : t \rightarrow u$   
and  $\text{env} \vdash e2 : t$   
then  $\text{env} \vdash e1\ e2 : u$

# Innocent typing rule

if  $\text{env} \vdash e1 : t \rightarrow u$

and  $\text{env} \vdash e2 : t$

then  $\text{env} \vdash e1 \ e2 : u$

# Innocent typing rule

if  $\text{env} \vdash e1 : t \rightarrow u$

and  $\text{env} \vdash e2 : t$

then  $\text{env} \vdash e1\ e2 : u$

# Innocent typing rule

if  $\text{env} \vdash e1 : t \rightarrow u$

and  $\text{env} \vdash e2 : t$

then  $\text{env} \vdash e1 \ e2 : u$

# Innocent typing rule

if  $\text{env} \vdash e1 : t \Rightarrow u$   
and  $\text{env} \vdash e2 : t$   
then  $\text{env} \vdash e1 \ e2 : u$

Do you recognize this rule?

*Modus Ponens*

$A \Rightarrow B$

$A$

---

$B$



**INTERMISSION**

# Logical proof systems

- Ways of formalizing what is *provable*
- Which may differ from what is *true* or *decidable*
- Two styles:
  - Hilbert:
    - lots of axioms
    - few inference rules (maybe just modus ponens)
  - Gentzen:
    - lots of inference rules (a couple for each operator)
    - few axioms

# Inference rules

$$\frac{P_1 \quad P_2 \quad \dots P_n}{Q}$$

- From *premises*  $P_1, P_2, \dots, P_n$
- Infer *conclusion*  $Q$
- Express allowed means of *inference* or *deductive reasoning*
- *Axiom* is an inference rule with zero premises

# Judgments

$$A_1, A_2, \dots, A_n \vdash B$$

- From *assumptions*  $A_1, A_2, \dots, A_n$ 
  - traditional to write  $\Gamma$  for set of assumptions
- Judge that B is *derivable* or *provable*
- Express allowed means of *hypothetical reasoning*
- $\Gamma, A \vdash A$  is an axiom

# Inference rules for $\Rightarrow$ and $\wedge$

$$\frac{\Gamma, A \vdash B}{\Gamma \vdash A \Rightarrow B} \Rightarrow \text{intro}$$

$$\frac{\Gamma \vdash A \Rightarrow B \quad \Gamma \vdash A}{\Gamma \vdash B} \Rightarrow \text{elim}$$

$$\frac{\Gamma \vdash A \quad \Gamma \vdash B}{\Gamma \vdash A \wedge B} \wedge \text{intro}$$

$$\frac{\Gamma \vdash A \wedge B}{\Gamma \vdash A} \wedge \text{elim 1}$$

$$\frac{\Gamma \vdash A \wedge B}{\Gamma \vdash B} \wedge \text{elim 2}$$

# Introduction and elimination

- Introduction rules say how to *define* an operator
- Elimination rules say how to *use* an operator
- Gentzen's insight: every operator should come with intro and elim rules

**BACK TO THE SHOW**

# Innocent typing rule

if  $\text{env} \vdash e1 : t \rightarrow u$

and  $\text{env} \vdash e2 : t$

then  $\text{env} \vdash e1 \ e2 : u$

$$\frac{\text{env} \vdash e1 : t \rightarrow u \quad \text{env} \vdash e2 : t}{\text{env} \vdash e1 \ e2 : u}$$



# Innocent typing rule

if  $\text{env} \vdash e_1 : t \rightarrow u$

and  $\text{env} \vdash e_2 : t$

then  $\text{env} \vdash e_1 e_2 : u$

$$\frac{\text{env} \vdash e_1 : t \rightarrow u \quad \text{env} \vdash e_2 : t}{\text{env} \vdash e_1 e_2 : u}$$

# Innocent typing rule

if  $\text{env} \vdash e1 : t \rightarrow u$

and  $\text{env} \vdash e2 : t$

then  $\text{env} \vdash e1\ e2 : u$

$$\frac{\text{env} \vdash e1 : t \Rightarrow u \quad \text{env} \vdash e2 : t}{\text{env} \vdash e1\ e2 : u} \Rightarrow \text{elim}$$

Modus ponens is function application

# Computing with evidence

- Modus ponens (aka  $\Rightarrow$  elim) is a way of computing with evidence
  - Given evidence  $e_2$  that  $t$  holds
  - And given a way  $e_1$  of transforming evidence for  $t$  into evidence for  $u$
  - MP produces evidence for  $u$  by applying  $e_1$  to  $e_2$
- So  $e_1 \ e_2$  is a program... and a proof!

$$\text{env} \vdash e_1 : t \rightarrow u \quad \text{env} \vdash e_2 : t$$

---

$$\text{env} \vdash e_1 \ e_2 : u$$

# Typing rules for pairs

$$\text{env} \vdash e : t_1 * t_2$$

---

$$\text{env} \vdash \text{fst } e : t_1$$
$$\text{env} \vdash e : t_1 * t_2$$

---

$$\text{env} \vdash \text{snd } e : t_2$$

# Proof rules for $\wedge$

$$\frac{\text{env} \vdash e : t_1 \wedge t_2}{\text{env} \vdash \text{fst } e : t_1} \wedge \text{elim } 1$$

$$\frac{\text{env} \vdash e : t_1 \wedge t_2}{\text{env} \vdash \text{snd } e : t_2} \wedge \text{elim } 2$$

computing with evidence:

given evidence  $e$  for both  $t_i$ , project out the evidence for one of them

# Programs and proofs

- A well-typed program demonstrates that there is at least one value for that type
  - i.e. the that type is **inhabited**
  - a program is a proof that the type is inhabited
- A proof demonstrates that there is at least one way of deriving a formula
  - i.e. that the formula is provable by manipulating assumptions and doing inference
  - a proof is a program that manipulates evidence
- **Proofs are programs, and programs are proofs**

# Programs

and

# Proofs

are fundamentally the same idea

Evaluation = Simplification

## **ACT III**



# Many proofs/programs

A given proposition/type could have many proofs/programs.

Proposition/type:

- $A \Rightarrow (B \Rightarrow (A \wedge B))$
- `'a -> ('b -> ('a * 'b))`

Proofs/programs:

- `fun x -> fun y ->  
 (fun z -> (snd z, fst z)) (y,x)`
- `fun x -> fun y -> (snd (y,x), fst  
 (y,x))`
- `fun x -> fun y -> (x,y)`

# Many proofs/programs

Body of each proof/program:

- `(fun z -> (snd z, fst z)) (y,x)`
- `(snd (y,x), fst (y,x))`
- `(x,y)`

Each is the result of small-stepping the previous  
...and in each case, the proof/program gets simpler

Taking an evaluation step corresponds to simplifying  
the proof

Program evaluation

and

proof simplification

are fundamentally the same idea

# CONCLUSION

# These are all the same ideas

Programming	Logic
Types	Propositions
Programs	Proofs
Evaluation	Simplification

Computation is reasoning  
Functional programming is fundamental

# Upcoming events

- MS2 deliverables due tonight, **no late submissions**
- MS3 due in last week of classes

*This is fundamental.*

**THIS IS 3110**