CS 320: Concepts of Programming Languages

Lecture 3: OCaml Functions

Ankush Das

Nathan Mull

Sep 10, 2024

Office Hours

- **Nathan Mull**
  - *Tue, 6:30pm — 8pm: CGS 323*
  - *Wed, 6:30pm — 8pm: CGS 311*
  - *Thu, 6:30pm — 8pm: CGS 311*
- **Ankush Das** *(Wed, 11am — 1pm: CDS 1001)*
- **Qiancheng (Robin) Fu** *(Fri, 2:15pm — 4:15pm: PSY B33)*
- **june wunder** *(Thu, 2pm — 4pm: CDS 1001)*
- **Zach Casey** *(Mon, 2pm — 4pm: CGS 113)*

Installation Issues?

- Are you still facing installation issues?
- Please see the note about *GitHub Codespaces* from Instructor Nathan Mull on Piazza
- Windows Users? You can try *installing Ubuntu alongside Windows* by creating necessary partitions on your disk
- Some helpful links: https://www.tecmint.com/install-ubuntu-alongside-with-windows-dual-boot/ https://ubuntu.com/tutorials/install-ubuntu-desktop#1-overview
- Can run OCaml in the browser: https://ocaml.org/play

Goals for Today's Lecture

- Writing and Compiling OCaml Code Files
- If Expressions
- Anonymous Functions
- Function Definitions with Pattern Matching
- Function Applications
- Polymorphism

OCaml Code Files

- So far, we have written all our programs in UTop; you will lose all your work when you close UTop
- To save your work, OCaml programs can also be written in files with a "`.ml`" extension (e.g., `lec3.ml`)
- Compile this program using `ocamlc lec3.ml`
- This will create an `a.out` file which can then be executed using `./a.out`
- You can also compile to a specific output file using `ocamlc lec3.ml -o lec3`

Loading Files in UTop

- Instead of compiling files directly, you can also load them in Top
- Use the following command in UTop:
  `#use “lec3.ml”;;`
- Interact with the code, i.e., call functions, assign variables, etc. Assume that all functions in the file are now in scope.
- Close UTop using Control-D or `#quit;;`

If Expressions

- Before we delve into today's abstraction (i.e., functions), let's do another abstraction, namely *if expressions*
- As always, we will define it formally
  - *Syntax*: `if <expr> then <expr> else <expr>`
  - *Type System*: for expression `if e then` e_1 `else` e_2
    - `e` must have type bool
    - e_1 and e_2 must have the same type, say τ
    - Then the expression has type τ

Formal Typing Rule for If Expression

$$\frac{\Gamma \vdash e : \mathsf{bool} \qquad \Gamma \vdash e_1 : \tau \qquad \Gamma \vdash e_2 : \tau}{\Gamma \vdash \mathsf{if}\ e\ \mathsf{then}\ e_1\ \mathsf{else}\ e_2 : \tau}$$

- First premise: e has type bool
- Second premise: e_1 has type τ
- Third premise: e_2 has same type τ
- Conclusion: if-expression has same type τ

An Example Typing Derivation

- How do we determine the type of an expression? Build derivation tree

```
let x = true in
if x then 3 else 4
```

An Example Typing Derivation

- How do we determine the type of an expression? Build derivation tree

```
let x = true in
if x then 3 else 4
```

$$\cdot \vdash \text{let } x = \text{true in if } x \text{ then } 3 \text{ else } 4$$

$$\frac{\Gamma \vdash e_1 : \tau \qquad \Gamma, x : \tau \vdash e_2 : \tau'}{\Gamma \vdash \text{let } x = e_1 \text{ in } e_2 : \tau'}$$

$$\frac{\Gamma \vdash e : \text{bool} \qquad \Gamma \vdash e_1 : \tau \qquad \Gamma \vdash e_2 : \tau}{\Gamma \vdash \text{if } e \text{ then } e_1 \text{ else } e_2 : \tau}$$

An Example Typing Derivation

- How do we determine the type of an expression? Build derivation tree

```
let x = true in
if x then 3 else 4
```

$$\cdot \vdash \text{let } x = \text{true in if } x \text{ then } 3 \text{ else } 4 : \text{int}$$

$$\frac{\Gamma \vdash e_1 : \tau \qquad \Gamma, x : \tau \vdash e_2 : \tau'}{\Gamma \vdash \text{let } x = e_1 \text{ in } e_2 : \tau'}$$

$$\frac{\Gamma \vdash e : \text{bool} \qquad \Gamma \vdash e_1 : \tau \qquad \Gamma \vdash e_2 : \tau}{\Gamma \vdash \text{if } e \text{ then } e_1 \text{ else } e_2 : \tau}$$

An Example Typing Derivation

- How do we determine the type of an expression? Build derivation tree

```
let x = true in
if x then 3 else 4
```

$$\dfrac{}{\cdot \vdash \text{let } x = \text{true in if } x \text{ then } 3 \text{ else } 4 : \text{int}}$$

$$\dfrac{\Gamma \vdash e_1 : \tau \qquad \Gamma, x : \tau \vdash e_2 : \tau'}{\Gamma \vdash \text{let } x = e_1 \text{ in } e_2 : \tau'}$$

$$\dfrac{\Gamma \vdash e : \text{bool} \qquad \Gamma \vdash e_1 : \tau \qquad \Gamma \vdash e_2 : \tau}{\Gamma \vdash \text{if } e \text{ then } e_1 \text{ else } e_2 : \tau}$$

An Example Typing Derivation

- How do we determine the type of an expression? Build derivation tree

```
let x = true in
if x then 3 else 4
```

$$\frac{\cdot \vdash \text{true} : \text{bool}}{\cdot \vdash \text{let } x = \text{true in if } x \text{ then } 3 \text{ else } 4 : \text{int}}$$

$$\frac{\Gamma \vdash e_1 : \tau \qquad \Gamma, x : \tau \vdash e_2 : \tau'}{\Gamma \vdash \text{let } x = e_1 \text{ in } e_2 : \tau'} \qquad \frac{\Gamma \vdash e : \text{bool} \qquad \Gamma \vdash e_1 : \tau \qquad \Gamma \vdash e_2 : \tau}{\Gamma \vdash \text{if } e \text{ then } e_1 \text{ else } e_2 : \tau}$$

An Example Typing Derivation

- How do we determine the type of an expression? Build derivation tree

```
let x = true in
if x then 3 else 4
```

$$\frac{\cdot \vdash \text{true} : \text{bool} \qquad \{x : \text{bool}\} \vdash \text{if } x \text{ then } 3 \text{ else } 4 : \text{int}}{\cdot \vdash \text{let } x = \text{true in if } x \text{ then } 3 \text{ else } 4 : \text{int}}$$

$$\frac{\Gamma \vdash e_1 : \tau \qquad \Gamma, x : \tau \vdash e_2 : \tau'}{\Gamma \vdash \text{let } x = e_1 \text{ in } e_2 : \tau'} \qquad \frac{\Gamma \vdash e : \text{bool} \qquad \Gamma \vdash e_1 : \tau \qquad \Gamma \vdash e_2 : \tau}{\Gamma \vdash \text{if } e \text{ then } e_1 \text{ else } e_2 : \tau}$$

An Example Typing Derivation

- **How do we determine the type of an expression? Build derivation tree**

```
let x = true in
if x then 3 else 4
```

$$\dfrac{\cdot \vdash \mathsf{true} : \mathsf{bool} \qquad \dfrac{}{\{x : \mathsf{bool}\} \vdash \mathsf{if}\ x\ \mathsf{then}\ 3\ \mathsf{else}\ 4 : \mathsf{int}}}{\cdot \vdash \mathsf{let}\ x = \mathsf{true}\ \mathsf{in}\ \mathsf{if}\ x\ \mathsf{then}\ 3\ \mathsf{else}\ 4 : \mathsf{int}}$$

$$\dfrac{\Gamma \vdash e_1 : \tau \qquad \Gamma, x : \tau \vdash e_2 : \tau'}{\Gamma \vdash \mathsf{let}\ x = e_1\ \mathsf{in}\ e_2 : \tau'} \qquad \dfrac{\Gamma \vdash e : \mathsf{bool} \qquad \Gamma \vdash e_1 : \tau \qquad \Gamma \vdash e_2 : \tau}{\Gamma \vdash \mathsf{if}\ e\ \mathsf{then}\ e_1\ \mathsf{else}\ e_2 : \tau}$$

An Example Typing Derivation

- How do we determine the type of an expression? Build derivation tree

```
let x = true in
if x then 3 else 4
```

$$\dfrac{\cdot \vdash \text{true} : \text{bool} \qquad \dfrac{\{x : \text{bool}\} \vdash x : \text{bool}}{\{x : \text{bool}\} \vdash \text{if } x \text{ then } 3 \text{ else } 4 : \text{int}}}{\cdot \vdash \text{let } x = \text{true in if } x \text{ then } 3 \text{ else } 4 : \text{int}}$$

$$\dfrac{\Gamma \vdash e_1 : \tau \qquad \Gamma, x : \tau \vdash e_2 : \tau'}{\Gamma \vdash \text{let } x = e_1 \text{ in } e_2 : \tau'} \qquad \dfrac{\Gamma \vdash e : \text{bool} \qquad \Gamma \vdash e_1 : \tau \qquad \Gamma \vdash e_2 : \tau}{\Gamma \vdash \text{if } e \text{ then } e_1 \text{ else } e_2 : \tau}$$

An Example Typing Derivation

- How do we determine the type of an expression? Build derivation tree

```
let x = true in
if x then 3 else 4
```

$$\dfrac{\cdot \vdash \mathsf{true} : \mathsf{bool} \qquad \dfrac{\{x : \mathsf{bool}\} \vdash x : \mathsf{bool} \qquad \{x : \mathsf{bool}\} \vdash 3 : \mathsf{int}}{\{x : \mathsf{bool}\} \vdash \mathsf{if}\ x\ \mathsf{then}\ 3\ \mathsf{else}\ 4 : \mathsf{int}}}{\cdot \vdash \mathsf{let}\ x = \mathsf{true}\ \mathsf{in}\ \mathsf{if}\ x\ \mathsf{then}\ 3\ \mathsf{else}\ 4 : \mathsf{int}}$$

$$\dfrac{\Gamma \vdash e_1 : \tau \qquad \Gamma, x : \tau \vdash e_2 : \tau'}{\Gamma \vdash \mathsf{let}\ x = e_1\ \mathsf{in}\ e_2 : \tau'} \qquad \dfrac{\Gamma \vdash e : \mathsf{bool} \qquad \Gamma \vdash e_1 : \tau \qquad \Gamma \vdash e_2 : \tau}{\Gamma \vdash \mathsf{if}\ e\ \mathsf{then}\ e_1\ \mathsf{else}\ e_2 : \tau}$$

An Example Typing Derivation

- **How do we determine the type of an expression? Build derivation tree**

```
let x = true in
if x then 3 else 4
```

$$\dfrac{\cdot \vdash \mathsf{true} : \mathsf{bool} \qquad \dfrac{\{x : \mathsf{bool}\} \vdash x : \mathsf{bool} \qquad \{x : \mathsf{bool}\} \vdash 3 : \mathsf{int} \qquad \{x : \mathsf{bool}\} \vdash 4 : \mathsf{int}}{\{x : \mathsf{bool}\} \vdash \mathsf{if}\ x\ \mathsf{then}\ 3\ \mathsf{else}\ 4 : \mathsf{int}}}{\cdot \vdash \mathsf{let}\ x = \mathsf{true}\ \mathsf{in}\ \mathsf{if}\ x\ \mathsf{then}\ 3\ \mathsf{else}\ 4 : \mathsf{int}}$$

$$\dfrac{\Gamma \vdash e_1 : \tau \qquad \Gamma, x : \tau \vdash e_2 : \tau'}{\Gamma \vdash \mathsf{let}\ x = e_1\ \mathsf{in}\ e_2 : \tau'} \qquad \dfrac{\Gamma \vdash e : \mathsf{bool} \qquad \Gamma \vdash e_1 : \tau \qquad \Gamma \vdash e_2 : \tau}{\Gamma \vdash \mathsf{if}\ e\ \mathsf{then}\ e_1\ \mathsf{else}\ e_2 : \tau}$$

An Example Typing Derivation

- How do we determine the type of an expression? Build derivation tree

```
let x = true in
if y then 3 else 4
```

An Example Typing Derivation

- How do we determine the type of an expression? Build derivation tree

```
let x = true in
if y then 3 else 4
```

$$\cdot \vdash \text{let } x = \text{true in if } y \text{ then } 3 \text{ else } 4$$

$$\frac{\Gamma \vdash e_1 : \tau \qquad \Gamma, x : \tau \vdash e_2 : \tau'}{\Gamma \vdash \text{let } x = e_1 \text{ in } e_2 : \tau'} \qquad \frac{\Gamma \vdash e : \text{bool} \qquad \Gamma \vdash e_1 : \tau \qquad \Gamma \vdash e_2 : \tau}{\Gamma \vdash \text{if } e \text{ then } e_1 \text{ else } e_2 : \tau}$$

An Example Typing Derivation

- How do we determine the type of an expression? Build derivation tree

```
let x = true in
if y then 3 else 4
```

$$\cdot \vdash \text{let } x = \text{true in if } y \text{ then } 3 \text{ else } 4 : \text{int}$$

$$\frac{\Gamma \vdash e_1 : \tau \qquad \Gamma, x : \tau \vdash e_2 : \tau'}{\Gamma \vdash \text{let } x = e_1 \text{ in } e_2 : \tau'} \qquad \frac{\Gamma \vdash e : \text{bool} \qquad \Gamma \vdash e_1 : \tau \qquad \Gamma \vdash e_2 : \tau}{\Gamma \vdash \text{if } e \text{ then } e_1 \text{ else } e_2 : \tau}$$

An Example Typing Derivation

- **How do we determine the type of an expression? Build derivation tree**

```
let x = true in
if y then 3 else 4
```

$$\frac{}{\cdot \vdash \text{let } x = \text{true in if } y \text{ then } 3 \text{ else } 4 : \text{int}}$$

$$\frac{\Gamma \vdash e_1 : \tau \qquad \Gamma, x : \tau \vdash e_2 : \tau'}{\Gamma \vdash \text{let } x = e_1 \text{ in } e_2 : \tau'}$$

$$\frac{\Gamma \vdash e : \text{bool} \qquad \Gamma \vdash e_1 : \tau \qquad \Gamma \vdash e_2 : \tau}{\Gamma \vdash \text{if } e \text{ then } e_1 \text{ else } e_2 : \tau}$$

An Example Typing Derivation

- How do we determine the type of an expression? Build derivation tree

```
let x = true in
if y then 3 else 4
```

$$\frac{\cdot \vdash \text{true} : \text{bool}}{\cdot \vdash \text{let } x = \text{true in if } y \text{ then } 3 \text{ else } 4 : \text{int}}$$

$$\frac{\Gamma \vdash e_1 : \tau \qquad \Gamma, x : \tau \vdash e_2 : \tau'}{\Gamma \vdash \text{let } x = e_1 \text{ in } e_2 : \tau'} \qquad \frac{\Gamma \vdash e : \text{bool} \qquad \Gamma \vdash e_1 : \tau \qquad \Gamma \vdash e_2 : \tau}{\Gamma \vdash \text{if } e \text{ then } e_1 \text{ else } e_2 : \tau}$$

An Example Typing Derivation

- How do we determine the type of an expression? Build derivation tree

```
let x = true in
if y then 3 else 4
```

$$\frac{\cdot \vdash \mathsf{true} : \mathsf{bool} \qquad \{x : \mathsf{bool}\} \vdash \mathsf{if}\ y\ \mathsf{then}\ 3\ \mathsf{else}\ 4 : \mathsf{int}}{\cdot \vdash \mathsf{let}\ x = \mathsf{true}\ \mathsf{in}\ \mathsf{if}\ y\ \mathsf{then}\ 3\ \mathsf{else}\ 4 : \mathsf{int}}$$

$$\frac{\Gamma \vdash e_1 : \tau \qquad \Gamma, x : \tau \vdash e_2 : \tau'}{\Gamma \vdash \mathsf{let}\ x = e_1\ \mathsf{in}\ e_2 : \tau'}$$

$$\frac{\Gamma \vdash e : \mathsf{bool} \qquad \Gamma \vdash e_1 : \tau \qquad \Gamma \vdash e_2 : \tau}{\Gamma \vdash \mathsf{if}\ e\ \mathsf{then}\ e_1\ \mathsf{else}\ e_2 : \tau}$$

An Example Typing Derivation

- How do we determine the type of an expression? Build derivation tree

```
let x = true in
if y then 3 else 4
```

$$\dfrac{\cdot \vdash \text{true} : \text{bool} \qquad \dfrac{}{\{x : \text{bool}\} \vdash \text{if } y \text{ then } 3 \text{ else } 4 : \text{int}}}{\cdot \vdash \text{let } x = \text{true in if } y \text{ then } 3 \text{ else } 4 : \text{int}}$$

$$\frac{\Gamma \vdash e_1 : \tau \qquad \Gamma, x : \tau \vdash e_2 : \tau'}{\Gamma \vdash \text{let } x = e_1 \text{ in } e_2 : \tau'} \qquad \frac{\Gamma \vdash e : \text{bool} \qquad \Gamma \vdash e_1 : \tau \qquad \Gamma \vdash e_2 : \tau}{\Gamma \vdash \text{if } e \text{ then } e_1 \text{ else } e_2 : \tau}$$

An Example Typing Derivation

- How do we determine the type of an expression? Build derivation tree

```
let x = true in
if y then 3 else 4
```

$$\dfrac{\cdot \vdash \text{true} : \text{bool} \qquad \dfrac{\{x : \text{bool}\} \vdash y : \text{bool}}{\{x : \text{bool}\} \vdash \text{if } y \text{ then } 3 \text{ else } 4 : \text{int}}}{\cdot \vdash \text{let } x = \text{true in if } y \text{ then } 3 \text{ else } 4 : \text{int}}$$

$$\dfrac{\Gamma \vdash e_1 : \tau \qquad \Gamma, x : \tau \vdash e_2 : \tau'}{\Gamma \vdash \text{let } x = e_1 \text{ in } e_2 : \tau'} \qquad \dfrac{\Gamma \vdash e : \text{bool} \qquad \Gamma \vdash e_1 : \tau \qquad \Gamma \vdash e_2 : \tau}{\Gamma \vdash \text{if } e \text{ then } e_1 \text{ else } e_2 : \tau}$$

An Example Typing Derivation

- How do we determine the type of an expression? Build derivation tree

```
let x = true in
if y then 3 else 4
```

$$\dfrac{\cdot \vdash \text{true} : \text{bool} \qquad \dfrac{\{x : \text{bool}\} \vdash y : \text{bool} \qquad \{x : \text{bool}\} \vdash 3 : \text{int}}{\{x : \text{bool}\} \vdash \text{if } y \text{ then } 3 \text{ else } 4 : \text{int}}}{\cdot \vdash \text{let } x = \text{true in if } y \text{ then } 3 \text{ else } 4 : \text{int}}$$

$$\frac{\Gamma \vdash e_1 : \tau \qquad \Gamma, x : \tau \vdash e_2 : \tau'}{\Gamma \vdash \text{let } x = e_1 \text{ in } e_2 : \tau'} \qquad \frac{\Gamma \vdash e : \text{bool} \qquad \Gamma \vdash e_1 : \tau \qquad \Gamma \vdash e_2 : \tau}{\Gamma \vdash \text{if } e \text{ then } e_1 \text{ else } e_2 : \tau}$$

An Example Typing Derivation

- **How do we determine the type of an expression? Build derivation tree**

```
let x = true in
if y then 3 else 4
```

$$\dfrac{\cdot \vdash \mathsf{true} : \mathsf{bool} \qquad \dfrac{\{x : \mathsf{bool}\} \vdash y : \mathsf{bool} \qquad \{x : \mathsf{bool}\} \vdash 3 : \mathsf{int} \qquad \{x : \mathsf{bool}\} \vdash 4 : \mathsf{int}}{\{x : \mathsf{bool}\} \vdash \mathsf{if}\ y\ \mathsf{then}\ 3\ \mathsf{else}\ 4 : \mathsf{int}}}{\cdot \vdash \mathsf{let}\ x = \mathsf{true}\ \mathsf{in}\ \mathsf{if}\ y\ \mathsf{then}\ 3\ \mathsf{else}\ 4 : \mathsf{int}}$$

$$\dfrac{\Gamma \vdash e_1 : \tau \qquad \Gamma, x : \tau \vdash e_2 : \tau'}{\Gamma \vdash \mathsf{let}\ x = e_1\ \mathsf{in}\ e_2 : \tau'}$$

$$\dfrac{\Gamma \vdash e : \mathsf{bool} \qquad \Gamma \vdash e_1 : \tau \qquad \Gamma \vdash e_2 : \tau}{\Gamma \vdash \mathsf{if}\ e\ \mathsf{then}\ e_1\ \mathsf{else}\ e_2 : \tau}$$

An Example Typing Derivation

- **How do we determine the type of an expression? Build derivation tree**

```
let x = true in
if y then 3 else 4
```

Unbound value y

$$\dfrac{\cdot \vdash \text{true} : \text{bool} \qquad \dfrac{\boxed{\{x : \text{bool}\} \vdash y : \text{bool}} \qquad \{x : \text{bool}\} \vdash 3 : \text{int} \qquad \{x : \text{bool}\} \vdash 4 : \text{int}}{\{x : \text{bool}\} \vdash \text{if } y \text{ then } 3 \text{ else } 4 : \text{int}}}{\cdot \vdash \text{let } x = \text{true in if } y \text{ then } 3 \text{ else } 4 : \text{int}}$$

$$\dfrac{\Gamma \vdash e_1 : \tau \qquad \Gamma, x : \tau \vdash e_2 : \tau'}{\Gamma \vdash \text{let } x = e_1 \text{ in } e_2 : \tau'} \qquad \dfrac{\Gamma \vdash e : \text{bool} \qquad \Gamma \vdash e_1 : \tau \qquad \Gamma \vdash e_2 : \tau}{\Gamma \vdash \text{if } e \text{ then } e_1 \text{ else } e_2 : \tau}$$

An Example Typing Derivation

- How do we determine the type of an expression? Build derivation tree

```
let x = true in
if x then x else 4
```

An Example Typing Derivation

- How do we determine the type of an expression? Build derivation tree

```
let x = true in
if x then x else 4
```

$$\cdot \vdash \text{let } x = \text{true in if } x \text{ then } x \text{ else } 4$$

$$\frac{\Gamma \vdash e_1 : \tau \qquad \Gamma, x : \tau \vdash e_2 : \tau'}{\Gamma \vdash \text{let } x = e_1 \text{ in } e_2 : \tau'}$$

$$\frac{\Gamma \vdash e : \text{bool} \qquad \Gamma \vdash e_1 : \tau \qquad \Gamma \vdash e_2 : \tau}{\Gamma \vdash \text{if } e \text{ then } e_1 \text{ else } e_2 : \tau}$$

An Example Typing Derivation

- **How do we determine the type of an expression? Build derivation tree**

```
let x = true in
if x then x else 4
```

$$\cdot \vdash \text{let } x = \text{true in if } x \text{ then } x \text{ else } 4 : \text{int}$$

$$\frac{\Gamma \vdash e_1 : \tau \qquad \Gamma, x : \tau \vdash e_2 : \tau'}{\Gamma \vdash \text{let } x = e_1 \text{ in } e_2 : \tau'}$$

$$\frac{\Gamma \vdash e : \text{bool} \qquad \Gamma \vdash e_1 : \tau \qquad \Gamma \vdash e_2 : \tau}{\Gamma \vdash \text{if } e \text{ then } e_1 \text{ else } e_2 : \tau}$$

An Example Typing Derivation

- How do we determine the type of an expression? Build derivation tree

```
let x = true in
if x then x else 4
```

$$\frac{}{\cdot \vdash \text{let } x = \text{true in if } x \text{ then } x \text{ else } 4 : \text{int}}$$

$$\frac{\Gamma \vdash e_1 : \tau \qquad \Gamma, x : \tau \vdash e_2 : \tau'}{\Gamma \vdash \text{let } x = e_1 \text{ in } e_2 : \tau'}$$

$$\frac{\Gamma \vdash e : \text{bool} \qquad \Gamma \vdash e_1 : \tau \qquad \Gamma \vdash e_2 : \tau}{\Gamma \vdash \text{if } e \text{ then } e_1 \text{ else } e_2 : \tau}$$

An Example Typing Derivation

- How do we determine the type of an expression? Build derivation tree

```
let x = true in
if x then x else 4
```

$$\frac{\cdot \vdash \text{true} : \text{bool}}{\cdot \vdash \text{let } x = \text{true in if } x \text{ then } x \text{ else } 4 : \text{int}}$$

$$\frac{\Gamma \vdash e_1 : \tau \qquad \Gamma, x : \tau \vdash e_2 : \tau'}{\Gamma \vdash \text{let } x = e_1 \text{ in } e_2 : \tau'} \qquad \frac{\Gamma \vdash e : \text{bool} \qquad \Gamma \vdash e_1 : \tau \qquad \Gamma \vdash e_2 : \tau}{\Gamma \vdash \text{if } e \text{ then } e_1 \text{ else } e_2 : \tau}$$

An Example Typing Derivation

- How do we determine the type of an expression? Build derivation tree

```
let x = true in
if x then x else 4
```

$$\frac{\cdot \vdash \mathsf{true} : \mathsf{bool} \qquad \{x : \mathsf{bool}\} \vdash \mathsf{if}\ x\ \mathsf{then}\ x\ \mathsf{else}\ 4 : \mathsf{int}}{\cdot \vdash \mathsf{let}\ x = \mathsf{true}\ \mathsf{in}\ \mathsf{if}\ x\ \mathsf{then}\ x\ \mathsf{else}\ 4 : \mathsf{int}}$$

$$\frac{\Gamma \vdash e_1 : \tau \qquad \Gamma, x : \tau \vdash e_2 : \tau'}{\Gamma \vdash \mathsf{let}\ x = e_1\ \mathsf{in}\ e_2 : \tau'} \qquad \frac{\Gamma \vdash e : \mathsf{bool} \qquad \Gamma \vdash e_1 : \tau \qquad \Gamma \vdash e_2 : \tau}{\Gamma \vdash \mathsf{if}\ e\ \mathsf{then}\ e_1\ \mathsf{else}\ e_2 : \tau}$$

An Example Typing Derivation

- How do we determine the type of an expression? Build derivation tree

```
let x = true in
if x then x else 4
```

$$\dfrac{\cdot \vdash \text{true} : \text{bool} \qquad \dfrac{}{\{x : \text{bool}\} \vdash \text{if } x \text{ then } x \text{ else } 4 : \text{int}}}{\cdot \vdash \text{let } x = \text{true in if } x \text{ then } x \text{ else } 4 : \text{int}}$$

$$\frac{\Gamma \vdash e_1 : \tau \qquad \Gamma, x : \tau \vdash e_2 : \tau'}{\Gamma \vdash \text{let } x = e_1 \text{ in } e_2 : \tau'} \qquad \frac{\Gamma \vdash e : \text{bool} \qquad \Gamma \vdash e_1 : \tau \qquad \Gamma \vdash e_2 : \tau}{\Gamma \vdash \text{if } e \text{ then } e_1 \text{ else } e_2 : \tau}$$

An Example Typing Derivation

- How do we determine the type of an expression? Build derivation tree

```
let x = true in
if x then x else 4
```

$$\dfrac{\cdot \vdash \mathsf{true} : \mathsf{bool} \qquad \dfrac{\{x : \mathsf{bool}\} \vdash x : \mathsf{bool}}{\{x : \mathsf{bool}\} \vdash \mathsf{if}\ x\ \mathsf{then}\ x\ \mathsf{else}\ 4 : \mathsf{int}}}{\cdot \vdash \mathsf{let}\ x = \mathsf{true}\ \mathsf{in}\ \mathsf{if}\ x\ \mathsf{then}\ x\ \mathsf{else}\ 4 : \mathsf{int}}$$

$$\dfrac{\Gamma \vdash e_1 : \tau \qquad \Gamma, x : \tau \vdash e_2 : \tau'}{\Gamma \vdash \mathsf{let}\ x = e_1\ \mathsf{in}\ e_2 : \tau'} \qquad \dfrac{\Gamma \vdash e : \mathsf{bool} \qquad \Gamma \vdash e_1 : \tau \qquad \Gamma \vdash e_2 : \tau}{\Gamma \vdash \mathsf{if}\ e\ \mathsf{then}\ e_1\ \mathsf{else}\ e_2 : \tau}$$

An Example Typing Derivation

- How do we determine the type of an expression? Build derivation tree

```
let x = true in
if x then x else 4
```

$$\dfrac{\cdot \vdash \text{true} : \text{bool} \qquad \dfrac{\{x : \text{bool}\} \vdash x : \text{bool} \qquad \{x : \text{bool}\} \vdash x : \text{int}}{\{x : \text{bool}\} \vdash \text{if } x \text{ then } x \text{ else } 4 : \text{int}}}{\cdot \vdash \text{let } x = \text{true in if } x \text{ then } x \text{ else } 4 : \text{int}}$$

$$\dfrac{\Gamma \vdash e_1 : \tau \qquad \Gamma, x : \tau \vdash e_2 : \tau'}{\Gamma \vdash \text{let } x = e_1 \text{ in } e_2 : \tau'} \qquad \dfrac{\Gamma \vdash e : \text{bool} \qquad \Gamma \vdash e_1 : \tau \qquad \Gamma \vdash e_2 : \tau}{\Gamma \vdash \text{if } e \text{ then } e_1 \text{ else } e_2 : \tau}$$

An Example Typing Derivation

- **How do we determine the type of an expression? Build derivation tree**

```
let x = true in
if x then x else 4
```

$$\dfrac{\cdot \vdash \text{true} : \text{bool} \qquad \dfrac{\{x : \text{bool}\} \vdash x : \text{bool} \qquad \{x : \text{bool}\} \vdash x : \text{int} \qquad \{x : \text{bool}\} \vdash 4 : \text{int}}{\{x : \text{bool}\} \vdash \text{if } x \text{ then } x \text{ else } 4 : \text{int}}}{\cdot \vdash \text{let } x = \text{true in if } x \text{ then } x \text{ else } 4 : \text{int}}$$

$$\dfrac{\Gamma \vdash e_1 : \tau \qquad \Gamma, x : \tau \vdash e_2 : \tau'}{\Gamma \vdash \text{let } x = e_1 \text{ in } e_2 : \tau'} \qquad \dfrac{\Gamma \vdash e : \text{bool} \qquad \Gamma \vdash e_1 : \tau \qquad \Gamma \vdash e_2 : \tau}{\Gamma \vdash \text{if } e \text{ then } e_1 \text{ else } e_2 : \tau}$$

An Example Typing Derivation

- How do we determine the type of an expression? Build derivation tree

```
let x = true in
if x then x else 4
```

This expression has type bool
but an expression was expected of type int

$$\dfrac{\cdot \vdash \text{true} : \text{bool} \qquad \dfrac{\{x : \text{bool}\} \vdash x : \text{bool} \qquad \boxed{\{x : \text{bool}\} \vdash x : \text{int}} \qquad \{x : \text{bool}\} \vdash 4 : \text{int}}{\{x : \text{bool}\} \vdash \text{if } x \text{ then } x \text{ else } 4 : \text{int}}}{\cdot \vdash \text{let } x = \text{true in if } x \text{ then } x \text{ else } 4 : \text{int}}$$

$$\frac{\Gamma \vdash e_1 : \tau \qquad \Gamma, x : \tau \vdash e_2 : \tau'}{\Gamma \vdash \text{let } x = e_1 \text{ in } e_2 : \tau'} \qquad \frac{\Gamma \vdash e : \text{bool} \qquad \Gamma \vdash e_1 : \tau \qquad \Gamma \vdash e_2 : \tau}{\Gamma \vdash \text{if } e \text{ then } e_1 \text{ else } e_2 : \tau}$$

What Happens if They Have Different Types?

$$\frac{\Gamma \vdash e : \mathsf{bool} \qquad \Gamma \vdash e_1 : \tau_1 \qquad \Gamma \vdash e_2 : \tau_2}{\Gamma \vdash \mathsf{if}\ e\ \mathsf{then}\ e_1\ \mathsf{else}\ e_2 : ?}$$

- **What if e_1 has type τ_1 and e_2 has same type τ_2?**
- **Depending on the value of e, the let-expression has different types**
- **But type needs to be determined at compile-time, not runtime, hence this is not possible!**
- **This violates an important theorem called *preservation* (Yes! PLs have to satisfy important mathematical theorems!!)**

Semantics of If Expressions

- *Semantics*: for expression `if e then e₁ else e₂`
- As always, we will define it formally
  - If `e` evaluates to true
    - Then we evaluate e_1. If e_1 evaluates to v_1, then the expression evaluates to v_1
  - Otherwise, if `e` evaluates to false
    - Then we evaluate e_2. If e_2 evaluates to v_2, then the expression evaluates to v_2

Formal Semantics Rule for If Expressions

- **Two rules (*true* case and *false* case)**

$$\frac{e \Downarrow \mathsf{true} \qquad e_1 \Downarrow v_1}{\mathsf{if}\ e\ \mathsf{then}\ e_1\ \mathsf{else}\ e_2 \Downarrow v_1}$$

$$\frac{e \Downarrow \mathsf{false} \qquad e_2 \Downarrow v_2}{\mathsf{if}\ e\ \mathsf{then}\ e_1\ \mathsf{else}\ e_2 \Downarrow v_2}$$

Goals for Today's Lecture

- Writing and Compiling OCaml Code Files
- If Expressions
- Anonymous Functions
- Function Definitions
- Function Applications
- Polymorphism

Anonymous Functions

- Recall how we defined functions: `let f x = e`
- We can define the same function using the fun-expression
- `let f = fun x -> e`
- e.g., `let f x = x + 2` can also be written as
  `let f = fun x -> x + 2`
- Why define it this way? Because not every function needs a name; we can define anonymous (no-name) functions

Anonymous Functions

- Suppose we define a simple function and immediately apply it

- ```
  let f x = x + 2 in
  let y = f 3 …
  ```

- This can instead be written as
  ```
  let y = (fun x -> x + 2) 3
  ```

- We will cover more examples of anonymous functions later when we do higher-order programming
  ```

Pattern Matching

- Recall factorial function from the last lecture
- We can define factorial as follows also:
-
```
let rec factorial n =
match n with
| 0 -> 1
| n -> n * factorial (n-1)
```
- Pattern matching is quite powerful and expressive as we will see in the upcoming lectures! It can be used against booleans, integers, floats, etc.

More Pattern Matching

- We can pattern match on multiple arguments also:

```
let and x y =
match x, y with
| true, true -> true
| true, false -> false
| false, true -> false
| false, false -> false
```

- We can even use wildcards (_) in patterns:

```
let and x y =
match x, y with
| true, true -> true
| _, _ -> false
```

Formal Typing Rule for Function Definitions

$$\frac{\Gamma, x : \tau \vdash e_1 : \tau_1 \qquad \Gamma, f : \tau \to \tau_1 \vdash e_2 : \tau_2}{\Gamma \vdash \mathsf{let}\ f\ x = e_1\ \mathsf{in}\ e_2 : \tau_2}$$

- We assume `x` has type τ and use that to infer the type of e_1, say τ_1
- Then, we know `f` has type τ `->` τ_1
- Then, we add the type of `f` to scope by adding it to context Γ and infer the type of e_2 to be τ_2
- This can be extended to functions with multiple arguments *(Homework)*

Let’s Do An Example

- `let f x = x + 1`
- Assume `x` has some type τ
- In function body, `x` is added using the ‘`+`’ operator, so τ `= int`
- And the function returns an integer
- Therefore, `f : int -> int`
- *Homework: Please do the typing derivation*

Another Example


- ```
  let and x y =
  match x, y with
  | true, true -> true
  | _, _ -> false
  ```
- Assume `x` has some type τ_1 and `y` has some type τ_2
- In function body, `x` and `y` are matched with bools, so $\tau_1 = \tau_2 =$ `bool`
- And the function returns bool in all branches
- Therefore, `and : bool -> bool -> bool`
- *Homework: Please do the typing derivation*
  ```

Function Applications

- `let rec factorial n =`
  `if n = 0 then 1 else n * factorial (n-1)`
- Suppose we have expression: `f e` , i.e., function `f` is applied to expression `e`
- Suppose the type of `f` is τ `->` τ_1
- What should be the type of `e`? It has to τ; anything else would be a type error
- Then, `f e :` τ_1

Typing Rule for Function Applications

$$\frac{\Gamma \vdash f : \tau \to \tau_1 \qquad \Gamma \vdash e : \tau}{\Gamma \vdash f\ e : \tau_1}$$

- First premise: `f` has type τ `->` τ_1
- Second premise: `e` has type τ
- Conclusion: `f e` has type τ_1

Examples: Are These Right or Wrong?


- ```
  let rec factorial n =
  if n = 0. then 1 else n * factorial (n -. 1.)
  ```
- ```
  let rec factorial n =
  if n = 0 then 1. else n * factorial (n - 1)
  ```
- ```
  let rec factorial n =
  if n = 0 then 1. else n *. factorial (n - 1)
  ```
- ```
  let rec factorial n =
  if n = 0. then 1. else n * factorial (n -. 1)
  ```
- ```
  let rec factorial n =
  if n = 0. then 1. else n *. factorial (n -. 1.)
  ```
  ```

Examples: Are These Right or Wrong?


- ```
  let rec factorial n =
  if n = 0. then 1 else n * factorial (n -. 1.)
  ```
- ```
  let rec factorial n =
  if n = 0 then 1. else n * factorial (n - 1)
  ```
- ```
  let rec factorial n =
  if n = 0 then 1. else n *. factorial (n - 1)
  ```
- ```
  let rec factorial n =
  if n = 0. then 1. else n * factorial (n -. 1)
  ```
- ```
  let rec factorial n =
  if n = 0. then 1. else n *. factorial (n -. 1.)
  ```
  ```

Examples: Are These Right or Wrong?


- ```
  let rec factorial n =
  if n = 0. then 1 else n * factorial (n -. 1.)
  ```
- ```
  let rec factorial n =
  if n = 0 then 1. else n * factorial (n - 1)
  ```
- ```
  let rec factorial n =
  if n = 0 then 1. else n *. factorial (n - 1)
  ```
- ```
  let rec factorial n =
  if n = 0. then 1. else n * factorial (n -. 1)
  ```
- ```
  let rec factorial n =
  if n = 0. then 1. else n *. factorial (n -. 1.)
  ```
  ```

Examples: Are These Right or Wrong?

- `let rec factorial n =`
  `if n = 0. then 1 else n * factorial (n -. 1.)` ✗
- `let rec factorial n =`
  `if n = 0 then 1. else n * factorial (n - 1)` ✗
- `let rec factorial n =`
  `if n = 0 then 1. else n *. factorial (n - 1)` ✗
- `let rec factorial n =`
  `if n = 0. then 1. else n * factorial (n -. 1)`
- `let rec factorial n =`
  `if n = 0. then 1. else n *. factorial (n -. 1.)`

Examples: Are These Right or Wrong?


- ```
  let rec factorial n =
  if n = 0. then 1 else n * factorial (n -. 1.)
  ```
  ✗
- ```
  let rec factorial n =
  if n = 0 then 1. else n * factorial (n - 1)
  ```
  ✗
- ```
  let rec factorial n =
  if n = 0 then 1. else n *. factorial (n - 1)
  ```
  ✗
- ```
  let rec factorial n =
  if n = 0. then 1. else n * factorial (n -. 1)
  ```
  ✗
- ```
  let rec factorial n =
  if n = 0. then 1. else n *. factorial (n -. 1.)
  ```
  ```

Examples: Are These Right or Wrong?


- ```
  let rec factorial n =
  if n = 0. then 1 else n * factorial (n -. 1.)
  ```
  ✗

- ```
  let rec factorial n =
  if n = 0 then 1. else n * factorial (n - 1)
  ```
  ✗

- ```
  let rec factorial n =
  if n = 0 then 1. else n *. factorial (n - 1)
  ```
  ✗

- ```
  let rec factorial n =
  if n = 0. then 1. else n * factorial (n -. 1)
  ```
  ✗

- ```
  let rec factorial n =
  if n = 0. then 1. else n *. factorial (n -. 1.)
  ```
  ✓
  ```

Evaluating Function Applications


- ```
  let rec factorial n =
  if n = 0 then 1 else n * factorial (n-1)
  ```

- Let's compute: `factorial 2`

- Idea: Apply substitution; in the above case, substitute `2` for function body of `factorial`

- And keep doing it until the expression becomes a value
  ```

Evaluating 2!

- `let rec factorial n =`
  `if n = 0 then 1 else n * factorial (n-1)`

- `factorial 2`

```
= [2/n]if n = 0 then 1 else n * factorial (n-1)
= if 2 = 0 then 1 else 2 * factorial (2-1)
= 2 * factorial 1
= 2 * [1/n]if n = 0 then 1 else n * factorial (n-1)
= 2 * if 1 = 0 then 1 else 1 * factorial 0
= 2 * 1 * factorial 0
= 2 * 1 * [0/n]if n = 0 then 1 else n * factorial (n-1)
= 2 * 1 * if 0 = 0 then 1 else 0 * factorial -1
= 2 * 1 * 1
= 2
```

Semantics Rule for Function Applications

$$\frac{e \Downarrow v \qquad \text{let } f\ x = e' \qquad [v/x]e' \Downarrow v'}{f\ e \Downarrow v'}$$

- First premise: evaluate argument `e` to value `v`
- Second premise: find the definition of `f` suppose it is defined with parameter `x` and body `e`
- Third premise: substitute `v` for `x` `e` and evaluate it to `v′`
- Conclusion: `f e` evaluates to `v′`

Evaluating 2! Formally

factorial 2

$$\frac{e \Downarrow v \qquad \text{let } f\ x = e' \qquad [v/x]e' \Downarrow v'}{f\ e \Downarrow v'}$$

$$\frac{e_1 \Downarrow v_1 \qquad e_2 \Downarrow v_2 \qquad v = v_1 \times v_2}{e_1 \times e_2 \Downarrow v}$$

$$\frac{e \Downarrow \text{false} \qquad e_2 \Downarrow v_2}{\text{if } e \text{ then } e_1 \text{ else } e_2 \Downarrow v_2}$$

Evaluating 2! Formally

$$\text{factorial } 2 \Downarrow 2$$

$$\frac{e \Downarrow v \qquad \text{let } f\ x = e' \qquad [v/x]e' \Downarrow v'}{f\ e \Downarrow v'} \qquad \frac{e_1 \Downarrow v_1 \qquad e_2 \Downarrow v_2 \qquad v = v_1 \times v_2}{e_1 \times e_2 \Downarrow v}$$

$$\frac{e \Downarrow \text{false} \qquad e_2 \Downarrow v_2}{\text{if } e \text{ then } e_1 \text{ else } e_2 \Downarrow v_2}$$

Evaluating 2! Formally

$$\overline{\text{factorial } 2 \Downarrow 2}$$

$$\frac{e \Downarrow v \qquad \text{let } f\ x = e' \qquad [v/x]e' \Downarrow v'}{f\ e \Downarrow v'} \qquad\qquad \frac{e_1 \Downarrow v_1 \qquad e_2 \Downarrow v_2 \qquad v = v_1 \times v_2}{e_1 \times e_2 \Downarrow v}$$

$$\frac{e \Downarrow \text{false} \qquad e_2 \Downarrow v_2}{\text{if } e \text{ then } e_1 \text{ else } e_2 \Downarrow v_2}$$

Evaluating 2! Formally

$$\frac{2 \Downarrow 2}{\text{factorial } 2 \Downarrow 2}$$

$$\frac{e \Downarrow v \qquad \text{let } f\ x = e' \qquad [v/x]e' \Downarrow v'}{f\ e \Downarrow v'} \qquad \frac{e_1 \Downarrow v_1 \qquad e_2 \Downarrow v_2 \qquad v = v_1 \times v_2}{e_1 \times e_2 \Downarrow v}$$

$$\frac{e \Downarrow \text{false} \qquad e_2 \Downarrow v_2}{\text{if } e \text{ then } e_1 \text{ else } e_2 \Downarrow v_2}$$

Evaluating 2! Formally

$$\frac{2 \Downarrow 2 \qquad [2/n]\text{if } n = 0 \text{ then } 1 \text{ else } n * \text{factorial } (n-1) \Downarrow 2}{\text{factorial } 2 \Downarrow 2}$$

$$\frac{e \Downarrow v \qquad \text{let } f\ x = e' \qquad [v/x]e' \Downarrow v'}{f\ e \Downarrow v'} \qquad \frac{e_1 \Downarrow v_1 \qquad e_2 \Downarrow v_2 \qquad v = v_1 \times v_2}{e_1 \times e_2 \Downarrow v}$$

$$\frac{e \Downarrow \text{false} \qquad e_2 \Downarrow v_2}{\text{if } e \text{ then } e_1 \text{ else } e_2 \Downarrow v_2}$$

Evaluating 2! Formally

$$\dfrac{2 \Downarrow 2 \qquad \dfrac{}{[2/n]\text{if } n = 0 \text{ then } 1 \text{ else } n * \text{factorial } (n-1) \Downarrow 2}}{\text{factorial } 2 \Downarrow 2}$$

$$\dfrac{e \Downarrow v \qquad \text{let } f\ x = e' \qquad [v/x]e' \Downarrow v'}{f\ e \Downarrow v'} \qquad\qquad \dfrac{e_1 \Downarrow v_1 \qquad e_2 \Downarrow v_2 \qquad v = v_1 \times v_2}{e_1 \times e_2 \Downarrow v}$$

$$\dfrac{e \Downarrow \text{false} \qquad e_2 \Downarrow v_2}{\text{if } e \text{ then } e_1 \text{ else } e_2 \Downarrow v_2}$$

Evaluating 2! Formally

$$\dfrac{2 \Downarrow 2 \qquad \dfrac{\text{if } 2 = 0 \text{ then } 1 \text{ else } 2 * \text{factorial } (2-1) \Downarrow 2}{[2/n]\text{if } n = 0 \text{ then } 1 \text{ else } n * \text{factorial } (n-1) \Downarrow 2}}{\text{factorial } 2 \Downarrow 2}$$

$$\dfrac{e \Downarrow v \qquad \text{let } f\ x = e' \qquad [v/x]e' \Downarrow v'}{f\ e \Downarrow v'} \qquad \dfrac{e_1 \Downarrow v_1 \qquad e_2 \Downarrow v_2 \qquad v = v_1 \times v_2}{e_1 \times e_2 \Downarrow v}$$

$$\dfrac{e \Downarrow \text{false} \qquad e_2 \Downarrow v_2}{\text{if } e \text{ then } e_1 \text{ else } e_2 \Downarrow v_2}$$

Evaluating 2! Formally

$$\dfrac{2 \Downarrow 2 \qquad \dfrac{\overline{\text{if } 2 = 0 \text{ then } 1 \text{ else } 2 * \text{factorial } (2 - 1) \Downarrow 2}}{[2/n]\text{if } n = 0 \text{ then } 1 \text{ else } n * \text{factorial } (n - 1) \Downarrow 2}}{\text{factorial } 2 \Downarrow 2}$$

$$\frac{e \Downarrow v \qquad \text{let } f\ x = e' \qquad [v/x]e' \Downarrow v'}{f\ e \Downarrow v'} \qquad \frac{e_1 \Downarrow v_1 \qquad e_2 \Downarrow v_2 \qquad v = v_1 \times v_2}{e_1 \times e_2 \Downarrow v}$$

$$\frac{e \Downarrow \text{false} \qquad e_2 \Downarrow v_2}{\text{if } e \text{ then } e_1 \text{ else } e_2 \Downarrow v_2}$$

Evaluating 2! Formally

$$
\cfrac{2 \Downarrow 2 \qquad \cfrac{\cfrac{2 = 0 \Downarrow \text{false}}{\text{if } 2 = 0 \text{ then } 1 \text{ else } 2 * \text{factorial } (2 - 1) \Downarrow 2}}{[2/n]\text{if } n = 0 \text{ then } 1 \text{ else } n * \text{factorial } (n - 1) \Downarrow 2}}{\text{factorial } 2 \Downarrow 2}
$$

$$
\frac{e \Downarrow v \qquad \text{let } f\ x = e' \qquad [v/x]e' \Downarrow v'}{f\ e \Downarrow v'}
\qquad
\frac{e_1 \Downarrow v_1 \qquad e_2 \Downarrow v_2 \qquad v = v_1 \times v_2}{e_1 \times e_2 \Downarrow v}
$$

$$
\frac{e \Downarrow \text{false} \qquad e_2 \Downarrow v_2}{\text{if } e \text{ then } e_1 \text{ else } e_2 \Downarrow v_2}
$$

Evaluating 2! Formally

$$\cfrac{2 \Downarrow 2 \qquad \cfrac{\cfrac{2 = 0 \Downarrow \mathsf{false} \qquad 2 * \mathsf{factorial}\ (2-1) \Downarrow 2}{\mathsf{if}\ 2 = 0\ \mathsf{then}\ 1\ \mathsf{else}\ 2 * \mathsf{factorial}\ (2-1) \Downarrow 2}}{[2/n]\mathsf{if}\ n = 0\ \mathsf{then}\ 1\ \mathsf{else}\ n * \mathsf{factorial}\ (n-1) \Downarrow 2}}{\mathsf{factorial}\ 2 \Downarrow 2}$$

$$\frac{e \Downarrow v \qquad \mathsf{let}\ f\ x = e' \qquad [v/x]e' \Downarrow v'}{f\ e \Downarrow v'} \qquad \frac{e_1 \Downarrow v_1 \qquad e_2 \Downarrow v_2 \qquad v = v_1 \times v_2}{e_1 \times e_2 \Downarrow v}$$

$$\frac{e \Downarrow \mathsf{false} \qquad e_2 \Downarrow v_2}{\mathsf{if}\ e\ \mathsf{then}\ e_1\ \mathsf{else}\ e_2 \Downarrow v_2}$$

Evaluating 2! Formally

$$\dfrac{2 \Downarrow 2 \qquad \dfrac{\dfrac{2 = 0 \Downarrow \mathsf{false} \qquad \dfrac{}{2 * \mathsf{factorial}\ (2 - 1) \Downarrow 2}}{\mathsf{if}\ 2 = 0\ \mathsf{then}\ 1\ \mathsf{else}\ 2 * \mathsf{factorial}\ (2 - 1) \Downarrow 2}}{[2/n]\mathsf{if}\ n = 0\ \mathsf{then}\ 1\ \mathsf{else}\ n * \mathsf{factorial}\ (n - 1) \Downarrow 2}}{\mathsf{factorial}\ 2 \Downarrow 2}$$

$$\dfrac{e \Downarrow v \qquad \mathsf{let}\ f\ x = e' \qquad [v/x]e' \Downarrow v'}{f\ e \Downarrow v'} \qquad \dfrac{e_1 \Downarrow v_1 \qquad e_2 \Downarrow v_2 \qquad v = v_1 \times v_2}{e_1 \times e_2 \Downarrow v}$$

$$\dfrac{e \Downarrow \mathsf{false} \qquad e_2 \Downarrow v_2}{\mathsf{if}\ e\ \mathsf{then}\ e_1\ \mathsf{else}\ e_2 \Downarrow v_2}$$

Evaluating 2! Formally

$$\dfrac{2 \Downarrow 2 \qquad \dfrac{2 = 0 \Downarrow \text{false} \qquad \dfrac{2 \Downarrow 2}{2 * \text{factorial } (2-1) \Downarrow 2}}{\dfrac{\text{if } 2 = 0 \text{ then } 1 \text{ else } 2 * \text{factorial } (2-1) \Downarrow 2}{[2/n]\text{if } n = 0 \text{ then } 1 \text{ else } n * \text{factorial } (n-1) \Downarrow 2}}}{\text{factorial } 2 \Downarrow 2}$$

$$\frac{e \Downarrow v \qquad \text{let } f\ x = e' \qquad [v/x]e' \Downarrow v'}{f\ e \Downarrow v'} \qquad \frac{e_1 \Downarrow v_1 \qquad e_2 \Downarrow v_2 \qquad v = v_1 \times v_2}{e_1 \times e_2 \Downarrow v}$$

$$\frac{e \Downarrow \text{false} \qquad e_2 \Downarrow v_2}{\text{if } e \text{ then } e_1 \text{ else } e_2 \Downarrow v_2}$$

Evaluating 2! Formally

$$\dfrac{2 \Downarrow 2 \qquad \dfrac{2 = 0 \Downarrow \mathsf{false} \qquad \dfrac{2 \Downarrow 2 \qquad \mathsf{factorial}\ (2-1) \Downarrow 1}{2 * \mathsf{factorial}\ (2-1) \Downarrow 2}}{\dfrac{\mathsf{if}\ 2 = 0\ \mathsf{then}\ 1\ \mathsf{else}\ 2 * \mathsf{factorial}\ (2-1) \Downarrow 2}{[2/n]\mathsf{if}\ n = 0\ \mathsf{then}\ 1\ \mathsf{else}\ n * \mathsf{factorial}\ (n-1) \Downarrow 2}}}{\mathsf{factorial}\ 2 \Downarrow 2}$$

$$\frac{e \Downarrow v \qquad \mathsf{let}\ f\ x = e' \qquad [v/x]e' \Downarrow v'}{f\ e \Downarrow v'} \qquad \frac{e_1 \Downarrow v_1 \qquad e_2 \Downarrow v_2 \qquad v = v_1 \times v_2}{e_1 \times e_2 \Downarrow v}$$

$$\frac{e \Downarrow \mathsf{false} \qquad e_2 \Downarrow v_2}{\mathsf{if}\ e\ \mathsf{then}\ e_1\ \mathsf{else}\ e_2 \Downarrow v_2}$$

Evaluating 2! Formally

$$\dfrac{\dfrac{2 = 0 \Downarrow \text{false} \qquad \dfrac{2 \Downarrow 2 \qquad \dfrac{}{\text{factorial } (2-1) \Downarrow 1}}{2 * \text{factorial } (2-1) \Downarrow 2}}{\text{if } 2 = 0 \text{ then } 1 \text{ else } 2 * \text{factorial } (2-1) \Downarrow 2}}{\dfrac{2 \Downarrow 2 \qquad [2/n]\text{if } n = 0 \text{ then } 1 \text{ else } n * \text{factorial } (n-1) \Downarrow 2}{\text{factorial } 2 \Downarrow 2}}$$

$$\frac{e \Downarrow v \qquad \text{let } f\ x = e' \qquad [v/x]e' \Downarrow v'}{f\ e \Downarrow v'} \qquad \frac{e_1 \Downarrow v_1 \qquad e_2 \Downarrow v_2 \qquad v = v_1 \times v_2}{e_1 \times e_2 \Downarrow v}$$

$$\frac{e \Downarrow \text{false} \qquad e_2 \Downarrow v_2}{\text{if } e \text{ then } e_1 \text{ else } e_2 \Downarrow v_2}$$

Evaluating 2! Formally

$$
\dfrac{2 \Downarrow 2 \qquad \dfrac{\dfrac{2 = 0 \Downarrow \text{false} \qquad \dfrac{2 \Downarrow 2 \qquad \dfrac{2 - 1 \Downarrow 1}{\text{factorial } (2 - 1) \Downarrow 1}}{2 * \text{factorial } (2 - 1) \Downarrow 2}}{\text{if } 2 = 0 \text{ then } 1 \text{ else } 2 * \text{factorial } (2 - 1) \Downarrow 2}}{[2/n]\text{if } n = 0 \text{ then } 1 \text{ else } n * \text{factorial } (n - 1) \Downarrow 2}}{\text{factorial } 2 \Downarrow 2}
$$

$$
\frac{e \Downarrow v \qquad \text{let } f\ x = e' \qquad [v/x]e' \Downarrow v'}{f\ e \Downarrow v'}
\qquad\qquad
\frac{e_1 \Downarrow v_1 \qquad e_2 \Downarrow v_2 \qquad v = v_1 \times v_2}{e_1 \times e_2 \Downarrow v}
$$

$$
\frac{e \Downarrow \text{false} \qquad e_2 \Downarrow v_2}{\text{if } e \text{ then } e_1 \text{ else } e_2 \Downarrow v_2}
$$

Evaluating 2! Formally

$$\dfrac{2 \Downarrow 2 \qquad \dfrac{2 = 0 \Downarrow \mathsf{false} \qquad \dfrac{2 \Downarrow 2 \qquad \dfrac{2 - 1 \Downarrow 1 \qquad [1/n]\mathsf{if}\ n = 0\ \mathsf{then}\ 1\ \mathsf{else}\ n * \mathsf{factorial}\ (n-1) \Downarrow 2}{\mathsf{factorial}\ (2-1) \Downarrow 1}}{2 * \mathsf{factorial}\ (2-1) \Downarrow 2}}{\dfrac{\mathsf{if}\ 2 = 0\ \mathsf{then}\ 1\ \mathsf{else}\ 2 * \mathsf{factorial}\ (2-1) \Downarrow 2}{[2/n]\mathsf{if}\ n = 0\ \mathsf{then}\ 1\ \mathsf{else}\ n * \mathsf{factorial}\ (n-1) \Downarrow 2}}}{\mathsf{factorial}\ 2 \Downarrow 2}$$

$$\dfrac{e \Downarrow v \qquad \mathsf{let}\ f\ x = e' \qquad [v/x]e' \Downarrow v'}{f\ e \Downarrow v'} \qquad \dfrac{e_1 \Downarrow v_1 \qquad e_2 \Downarrow v_2 \qquad v = v_1 \times v_2}{e_1 \times e_2 \Downarrow v}$$

$$\dfrac{e \Downarrow \mathsf{false} \qquad e_2 \Downarrow v_2}{\mathsf{if}\ e\ \mathsf{then}\ e_1\ \mathsf{else}\ e_2 \Downarrow v_2}$$

Evaluating 2! Formally

$$
\dfrac{2 \Downarrow 2 \qquad \dfrac{\dfrac{2 = 0 \Downarrow \text{false} \qquad \dfrac{2 \Downarrow 2 \qquad \dfrac{2 - 1 \Downarrow 1 \qquad \dfrac{}{[1/n]\text{if } n = 0 \text{ then } 1 \text{ else } n * \text{factorial } (n - 1) \Downarrow 2}}{\text{factorial } (2 - 1) \Downarrow 1}}{2 * \text{factorial } (2 - 1) \Downarrow 2}}{\text{if } 2 = 0 \text{ then } 1 \text{ else } 2 * \text{factorial } (2 - 1) \Downarrow 2}}{[2/n]\text{if } n = 0 \text{ then } 1 \text{ else } n * \text{factorial } (n - 1) \Downarrow 2}}{\text{factorial } 2 \Downarrow 2}
$$

$$
\dfrac{e \Downarrow v \qquad \text{let } f\ x = e' \qquad [v/x]e' \Downarrow v'}{f\ e \Downarrow v'}
\qquad\qquad
\dfrac{e_1 \Downarrow v_1 \qquad e_2 \Downarrow v_2 \qquad v = v_1 \times v_2}{e_1 \times e_2 \Downarrow v}
$$

$$
\dfrac{e \Downarrow \text{false} \qquad e_2 \Downarrow v_2}{\text{if } e \text{ then } e_1 \text{ else } e_2 \Downarrow v_2}
$$

Evaluating 2! Formally

$$
\dfrac{2 \Downarrow 2 \qquad \dfrac{2 = 0 \Downarrow \text{false} \qquad \dfrac{2 \Downarrow 2 \qquad \dfrac{2 - 1 \Downarrow 1 \qquad \dfrac{\text{if } 1 = 0 \text{ then } 1 \text{ else } 1 * \text{factorial } (1 - 1) \Downarrow 1}{[1/n]\text{if } n = 0 \text{ then } 1 \text{ else } n * \text{factorial } (n - 1) \Downarrow 2}}{\text{factorial } (2 - 1) \Downarrow 1}}{2 * \text{factorial } (2 - 1) \Downarrow 2}}{\dfrac{\text{if } 2 = 0 \text{ then } 1 \text{ else } 2 * \text{factorial } (2 - 1) \Downarrow 2}{[2/n]\text{if } n = 0 \text{ then } 1 \text{ else } n * \text{factorial } (n - 1) \Downarrow 2}}}{\text{factorial } 2 \Downarrow 2}
$$

$$
\dfrac{e \Downarrow v \qquad \text{let } f\ x = e' \qquad [v/x]e' \Downarrow v'}{f\ e \Downarrow v'}
\qquad\qquad
\dfrac{e_1 \Downarrow v_1 \qquad e_2 \Downarrow v_2 \qquad v = v_1 \times v_2}{e_1 \times e_2 \Downarrow v}
$$

$$
\dfrac{e \Downarrow \text{false} \qquad e_2 \Downarrow v_2}{\text{if } e \text{ then } e_1 \text{ else } e_2 \Downarrow v_2}
$$

Evaluating 2! Formally

$$
\dfrac{2 \Downarrow 2 \qquad \dfrac{2 = 0 \Downarrow \text{false} \qquad \dfrac{2 \Downarrow 2 \qquad \dfrac{2 - 1 \Downarrow 1 \qquad \dfrac{\dfrac{}{\text{if } 1 = 0 \text{ then } 1 \text{ else } 1 * \text{factorial } (1 - 1) \Downarrow 1}}{[1/n]\text{if } n = 0 \text{ then } 1 \text{ else } n * \text{factorial } (n - 1) \Downarrow 2}}{\text{factorial } (2 - 1) \Downarrow 1}}{2 * \text{factorial } (2 - 1) \Downarrow 2}}{\text{if } 2 = 0 \text{ then } 1 \text{ else } 2 * \text{factorial } (2 - 1) \Downarrow 2}}{\dfrac{[2/n]\text{if } n = 0 \text{ then } 1 \text{ else } n * \text{factorial } (n - 1) \Downarrow 2}{\text{factorial } 2 \Downarrow 2}}
$$

$$
\frac{e \Downarrow v \qquad \text{let } f\ x = e' \qquad [v/x]e' \Downarrow v'}{f\ e \Downarrow v'}
\qquad\qquad
\frac{e_1 \Downarrow v_1 \qquad e_2 \Downarrow v_2 \qquad v = v_1 \times v_2}{e_1 \times e_2 \Downarrow v}
$$

$$
\frac{e \Downarrow \text{false} \qquad e_2 \Downarrow v_2}{\text{if } e \text{ then } e_1 \text{ else } e_2 \Downarrow v_2}
$$

Evaluating 2! Formally

$$
\frac{2 \Downarrow 2 \qquad \dfrac{\dfrac{2 = 0 \Downarrow \text{false} \qquad \dfrac{2 \Downarrow 2 \qquad \dfrac{2 - 1 \Downarrow 1 \qquad \dfrac{\dfrac{1 = 0 \Downarrow \text{false}}{\text{if } 1 = 0 \text{ then } 1 \text{ else } 1 * \text{factorial } (1 - 1) \Downarrow 1}}{[1/n]\text{if } n = 0 \text{ then } 1 \text{ else } n * \text{factorial } (n - 1) \Downarrow 2}}{\text{factorial } (2 - 1) \Downarrow 1}}{2 * \text{factorial } (2 - 1) \Downarrow 2}}{\text{if } 2 = 0 \text{ then } 1 \text{ else } 2 * \text{factorial } (2 - 1) \Downarrow 2}}{[2/n]\text{if } n = 0 \text{ then } 1 \text{ else } n * \text{factorial } (n - 1) \Downarrow 2}}{\text{factorial } 2 \Downarrow 2}
$$

$$
\frac{e \Downarrow v \qquad \text{let } f\ x = e' \qquad [v/x]e' \Downarrow v'}{f\ e \Downarrow v'}
\qquad\qquad
\frac{e_1 \Downarrow v_1 \qquad e_2 \Downarrow v_2 \qquad v = v_1 \times v_2}{e_1 \times e_2 \Downarrow v}
$$

$$
\frac{e \Downarrow \text{false} \qquad e_2 \Downarrow v_2}{\text{if } e \text{ then } e_1 \text{ else } e_2 \Downarrow v_2}
$$

Evaluating 2! Formally

$$\dfrac{2 \Downarrow 2 \qquad \dfrac{2 = 0 \Downarrow \text{false} \qquad \dfrac{2 \Downarrow 2 \qquad \dfrac{2 - 1 \Downarrow 1 \qquad \dfrac{\dfrac{1 = 0 \Downarrow \text{false} \qquad \text{factorial } (1-1) \Downarrow 1}{\text{if } 1 = 0 \text{ then } 1 \text{ else } 1 * \text{factorial } (1-1) \Downarrow 1}}{[1/n]\text{if } n = 0 \text{ then } 1 \text{ else } n * \text{factorial } (n-1) \Downarrow 2}}{\text{factorial } (2-1) \Downarrow 1}}{2 * \text{factorial } (2-1) \Downarrow 2}}{\dfrac{\text{if } 2 = 0 \text{ then } 1 \text{ else } 2 * \text{factorial } (2-1) \Downarrow 2}{[2/n]\text{if } n = 0 \text{ then } 1 \text{ else } n * \text{factorial } (n-1) \Downarrow 2}}}{\text{factorial } 2 \Downarrow 2}$$

$$\dfrac{e \Downarrow v \qquad \text{let } f\ x = e' \qquad [v/x]e' \Downarrow v'}{f\ e \Downarrow v'} \qquad\qquad \dfrac{e_1 \Downarrow v_1 \qquad e_2 \Downarrow v_2 \qquad v = v_1 \times v_2}{e_1 \times e_2 \Downarrow v}$$

$$\dfrac{e \Downarrow \text{false} \qquad e_2 \Downarrow v_2}{\text{if } e \text{ then } e_1 \text{ else } e_2 \Downarrow v_2}$$

One Final Note On Evaluation

- Recall the factorial function on floats

```
let rec factorial n =
if n = 0. then 1. else n *. factorial (n -. 1.)
```

- What happens if we call

```
factorial 1.5
= 1.5 * factorial 0.5
= 1.5 * 0.5 * factorial -0.5
= 1.5 * 0.5 * -0.5 * factorial -1.5
= 1.5 * 0.5 * -0.5 * -1.5 * factorial -2.5
= ……………………………………………
```

- This will continue forever!

Another Example of Weird Evaluation

- `let x = 2 / 0`
- Try this in UTop!
- What will `x` evaluate to?
- We know that division by 0 is impossible!
- In OCaml, this raises an exception called `Exception: Division_by_zero.`

Outcome of Semantics

- Therefore, semantics has 3 outcomes:
  - Value: The expression evaluates to a value (most examples)
  - Exception: The evaluation raises an exception (e.g., stack overflow, division by 0)
  - Infinite Loop: The evaluation keeps going forever! (e.g., factorial)
- You should try all of these in UTop!

Partial Function Applications

- Suppose we have a function `f : int -> int -> int`
  `let f x y = x + y`

- We can apply this function partially by defining:
  `let g = f 2`

- Now `g` becomes a function `g : int -> int` with the definition
  `let g y = 2 + y`

- Simply a result of substitution: `2` for `x` in the definition of `f`

Advantages of Partial Application

- Helps us define functions compactly promoting reuse
- Suppose we have a function `pow : int -> int -> int`
```
let pow x y =
if y = 0 then 1 else x * pow x (y-1)
```
- Suppose we want to define a power of 2 function. We can just do
```
let pow2 = pow 2
```
- Suppose we want to define a power of 3 function. We can just do
```
let pow3 = pow 3
```
- We can do this for any value of `x`

Can You Infer the Type?

- Take this function for example
  `let f x = x`
- What should be the type of `f`?
- Should it be `int -> int`? Or `bool -> bool`? Or something else?
- Technically, it can be any type `α -> α` for any type `α`
- These functions are called polymorphic functions (similar to generics in Java, etc.)
- These functions are defined for all types and can be used at any type.

Polymorphic Functions

- This identity function can be applied to any argument
  `let f x = x`

- In OCaml, type is written as `‘a -> ‘a` (`‘a` is a polymorphic variable)

- ```
  f 3 = 3
  f true = true
  f “cs320” = “cs320”
  ```

- In fact, they only need to be defined once and can be called with different types without defining them individually for each type

- This makes them powerful! They are crucial for data structures like stacks, queues, trees that can store arbitrary data
  ```

More on Polymorphism

- What should be the type of this function?

```
let f x y =
if x = y then "Equal" else "Unequal"
```

- The type of `f : α -> α -> string`

- What about this function?should be hat should be the type of `f`

```
let g x y = x
```

- The type of `g : α -> β -> string`

Conclusions

- Compiling OCaml Code Files using `ocamlc` and `#use`
- Rules for Syntax, Typing, and Semantics for If Expressions
- Anonymous Functions using `fun` abstraction
- Pattern Matching using `match` abstraction
- Rules for Syntax, Typing, and Semantics for Function Applications
- Partial Function Applications
- Polymorphism