

Derivations

Concepts of Programming Languages Lecture 5

Practice Problem

What is the typing rule for let expression?

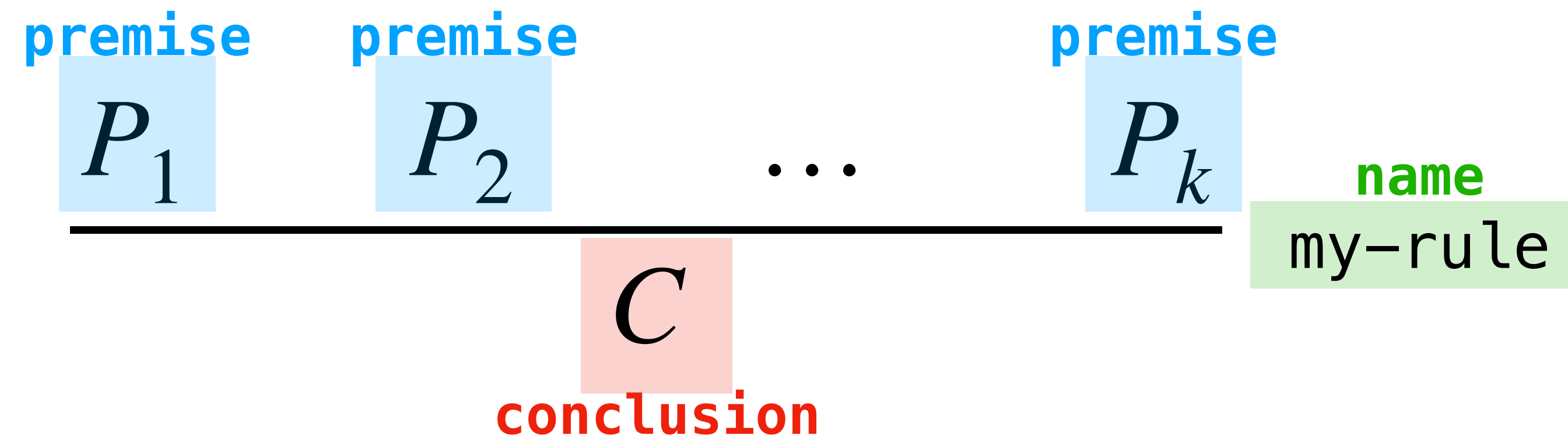
What are the semantics rules for if expression?

Outline

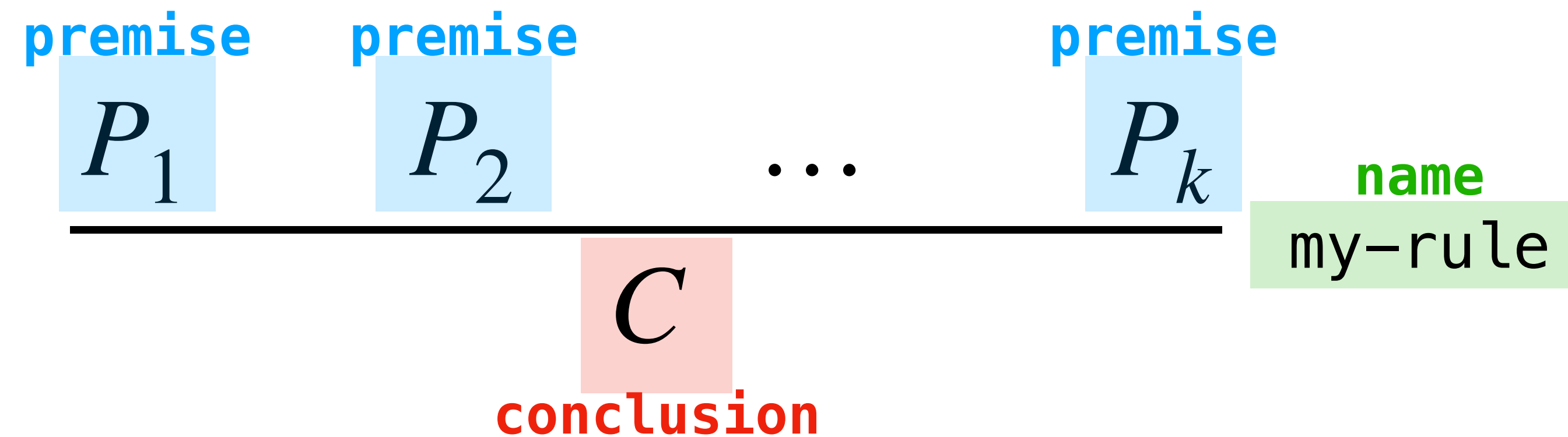
- » Discuss derivations in general
- » See how to read and write derivations
- » Go through a couple examples

Recap

Recall: Inference Rules

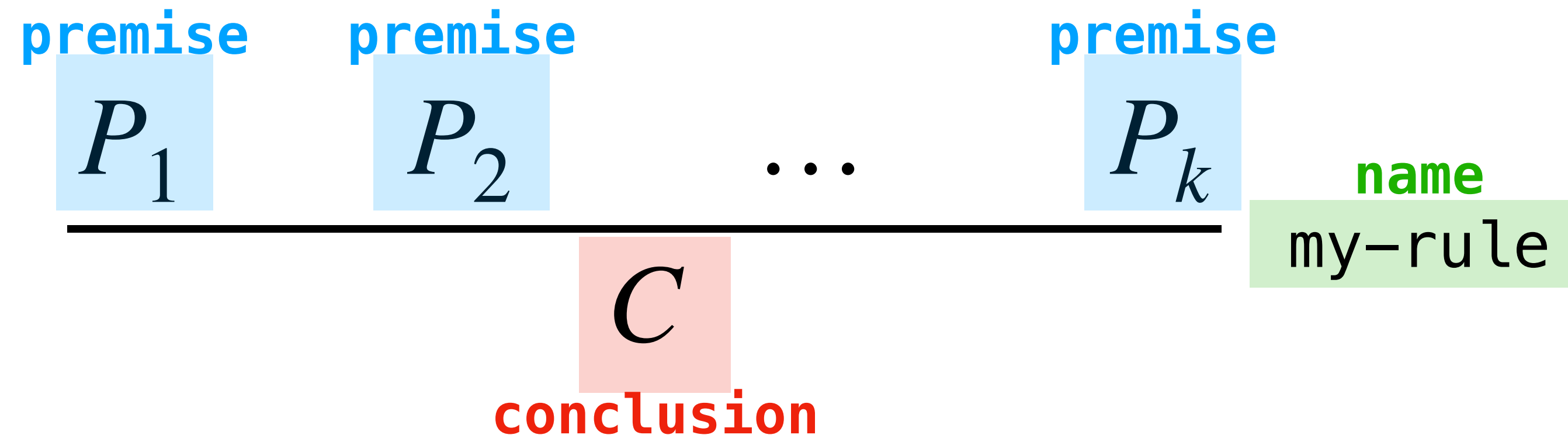


Recall: Inference Rules



Then general form of an inference rule has a collection of **premises** and a **conclusion**. Read as

Recall: Inference Rules



Then general form of an inference rule has a collection of **premises** and a **conclusion**. Read as

*If P_1 holds and P_2 holds and ... P_k holds, then C holds (by **my-rule**)*

Recall: Typing Judgments

$$\begin{array}{c} \text{context} \\ \Gamma \end{array} \vdash \begin{array}{c} \text{expression} \\ e \end{array} : \begin{array}{c} \text{type} \\ \tau \end{array}$$

A typing judgment is a compact way of representing the statement:

e is of type τ in/under the context Γ

Recall: Contexts

$$\Gamma = \{ x : \text{int}, y : \text{string}, z : \text{int} \rightarrow \text{string} \}$$

Recall: Contexts

$$\Gamma = \{ x : \text{int}, y : \text{string}, z : \text{int} \rightarrow \text{string} \}$$

A **context** is a set of **variable declarations**

Recall: Contexts

$$\Gamma = \{ x : \text{int}, y : \text{string}, z : \text{int} \rightarrow \text{string} \}$$

A **context** is a set of **variable declarations**

A variable declaration $(x : \tau)$ says: "I declare that the variable x is of type τ "

Recall: Contexts

$$\Gamma = \{ x : \text{int}, y : \text{string}, z : \text{int} \rightarrow \text{string} \}$$

A **context** is a set of **variable declarations**

A variable declaration $(x : \tau)$ says: "I declare that the variable x is of type τ "

A context keeps track of all the types of variables in the "environment"

Derivations

High Level

$$\frac{}{\{\} \vdash 2 : \text{int}} \text{(intLit)} \quad \frac{}{\{y : \text{int}\} \vdash y : \text{int}} \text{(var)} \quad \frac{}{\{y : \text{int}\} \vdash y : \text{int}} \text{(var)} \quad \frac{}{\{y : \text{int}\} \vdash y + y : \text{int}} \text{(intAdd)}$$
$$\frac{}{\{\} \vdash \text{let } y = 2 \text{ in } y + y : \text{int}} \text{(let)}$$

High Level

$$\frac{}{\{\} \vdash 2 : \text{int}} (\text{intLit}) \quad \frac{}{\{y : \text{int}\} \vdash y : \text{int}} (\text{var}) \quad \frac{}{\{y : \text{int}\} \vdash y : \text{int}} (\text{var}) \quad \frac{}{\{y : \text{int}\} \vdash y + y : \text{int}} (\text{intAdd})$$
$$\frac{}{\{\} \vdash \text{let } y = 2 \text{ in } y + y : \text{int}} (\text{let})$$

Derivations allow us to *prove* that a typing judgment holds with respect to a collection of inference rules

High Level

$$\frac{\frac{}{\{\} \vdash 2 : \text{int}} \text{(intLit)} \quad \frac{\frac{}{\{y : \text{int}\} \vdash y : \text{int}} \text{(var)} \quad \frac{\frac{}{\{y : \text{int}\} \vdash y : \text{int}} \text{(var)}}{\{y : \text{int}\} \vdash y + y : \text{int}} \text{(intAdd)}}{\{\} \vdash \text{let } y = 2 \text{ in } y + y : \text{int}} \text{(let)}$$

Derivations allow us to *prove* that a typing judgment holds with respect to a collection of inference rules

Formally, a **derivation** is a tree in which:

High Level

$$\frac{\frac{}{\{\} \vdash 2 : \text{int}} \text{(intLit)} \quad \frac{\frac{}{\{y : \text{int}\} \vdash y : \text{int}} \text{(var)} \quad \frac{\frac{}{\{y : \text{int}\} \vdash y : \text{int}} \text{(var)}}{\{y : \text{int}\} \vdash y + y : \text{int}} \text{(intAdd)}}{\{\} \vdash \text{let } y = 2 \text{ in } y + y : \text{int}} \text{(let)}$$

Derivations allow us to *prove* that a typing judgment holds with respect to a collection of inference rules

Formally, a **derivation** is a tree in which:

» each node is labeled with a typing judgment

High Level

$$\frac{\frac{}{\{\} \vdash 2 : \text{int}} \text{(intLit)} \quad \frac{\frac{}{\{y : \text{int}\} \vdash y : \text{int}} \text{(var)} \quad \frac{\frac{}{\{y : \text{int}\} \vdash y : \text{int}} \text{(var)}}{\{y : \text{int}\} \vdash y + y : \text{int}} \text{(intAdd)}}{\{\} \vdash \text{let } y = 2 \text{ in } y + y : \text{int}} \text{(let)}$$

Derivations allow us to *prove* that a typing judgment holds with respect to a collection of inference rules

Formally, a **derivation** is a tree in which:

- » each node is labeled with a typing judgment
- » and typing judgment *follows* from the typing judgments at it's children by an inference rule

Applying Rules

$$\frac{\Gamma \vdash e_1 : \tau_1 \quad \Gamma, x : \tau_1 \vdash e_2 : \tau}{\Gamma \vdash \text{let } x = e_1 \text{ in } e_2 : \tau} \text{ (let)} \quad \frac{\Gamma \vdash e_1 : \text{int} \quad \Gamma \vdash e_2 : \text{int}}{\Gamma \vdash e_1 + e_2 : \text{int}} \text{ (intAdd)}$$

$$\frac{\frac{}{\{\} \vdash 2 : \text{int}} \text{ (intLit)} \quad \frac{\frac{}{\{y : \text{int}\} \vdash y : \text{int}} \text{ (var)} \quad \frac{\frac{}{\{y : \text{int}\} \vdash y : \text{int}} \text{ (var)}}{\{y : \text{int}\} \vdash y + y : \text{int}} \text{ (intAdd)}}{\{\} \vdash \text{let } y = 2 \text{ in } y + y : \text{int}} \text{ (let)}$$

Applying Rules

$$\frac{\Gamma \vdash e_1 : \tau_1 \quad \Gamma, x : \tau_1 \vdash e_2 : \tau}{\Gamma \vdash \text{let } x = e_1 \text{ in } e_2 : \tau} \text{ (let)} \quad \frac{\Gamma \vdash e_1 : \text{int} \quad \Gamma \vdash e_2 : \text{int}}{\Gamma \vdash e_1 + e_2 : \text{int}} \text{ (intAdd)}$$

$$\frac{\frac{}{\{\} \vdash 2 : \text{int}} \text{ (intLit)} \quad \frac{\frac{}{\{y : \text{int}\} \vdash y : \text{int}} \text{ (var)} \quad \frac{\frac{}{\{y : \text{int}\} \vdash y : \text{int}} \text{ (var)} \text{ (intAdd)}}{\{y : \text{int}\} \vdash y + y : \text{int}} \text{ (let)}}{\{\} \vdash \text{let } y = 2 \text{ in } y + y : \text{int}}$$

So far, we've used rules as ways of describing the behavior of a PL

Applying Rules

$$\frac{\Gamma \vdash e_1 : \tau_1 \quad \Gamma, x : \tau_1 \vdash e_2 : \tau}{\Gamma \vdash \text{let } x = e_1 \text{ in } e_2 : \tau} \text{ (let)} \quad \frac{\Gamma \vdash e_1 : \text{int} \quad \Gamma \vdash e_2 : \text{int}}{\Gamma \vdash e_1 + e_2 : \text{int}} \text{ (intAdd)}$$

$$\frac{\frac{}{\{\} \vdash 2 : \text{int}} \text{ (intLit)} \quad \frac{\frac{}{\{y : \text{int}\} \vdash y : \text{int}} \text{ (var)} \quad \frac{\frac{}{\{y : \text{int}\} \vdash y : \text{int}} \text{ (var)}}{\{y : \text{int}\} \vdash y + y : \text{int}} \text{ (intAdd)}}{\{\} \vdash \text{let } y = 2 \text{ in } y + y : \text{int}} \text{ (let)}$$

So far, we've used rules as ways of describing the behavior of a PL

When we build typing derivations, we *instantiate* the meta-variables in the rule at *particular* expressions, contexts, etc.

Building from the Ground Up

$$\frac{}{\{\} \vdash 2 : \text{int}} \text{(intLit)} \quad \frac{}{\{y : \text{int}\} \vdash y : \text{int}} \text{(var)} \quad \frac{}{\{y : \text{int}\} \vdash y : \text{int}} \text{(var)} \quad \frac{}{\{y : \text{int}\} \vdash y + y : \text{int}} \text{(intAdd)}$$
$$\frac{}{\{\} \vdash \text{let } y = 2 \text{ in } y + y : \text{int}} \text{(let)}$$

Building from the Ground Up

$$\frac{\frac{}{\{\} \vdash 2 : \text{int}} \text{(intLit)} \quad \frac{\frac{}{\{y : \text{int}\} \vdash y : \text{int}} \text{(var)} \quad \frac{\frac{}{\{y : \text{int}\} \vdash y : \text{int}} \text{(var)}}{\{y : \text{int}\} \vdash y + y : \text{int}} \text{(intAdd)}}{\{\} \vdash \text{let } y = 2 \text{ in } y + y : \text{int}} \text{(let)}$$

1. Start from the bottom!

Building from the Ground Up

$$\frac{\frac{}{\{\} \vdash 2 : \text{int}} \text{(intLit)} \quad \frac{\frac{}{\{y : \text{int}\} \vdash y : \text{int}} \text{(var)} \quad \frac{\frac{}{\{y : \text{int}\} \vdash y : \text{int}} \text{(var)}}{\{y : \text{int}\} \vdash y + y : \text{int}} \text{(intAdd)}}{\{\} \vdash \text{let } y = 2 \text{ in } y + y : \text{int}} \text{(let)}$$

1. Start from the bottom!
2. Apply the rule based on the expression

Building from the Ground Up

$$\frac{\frac{}{\{\} \vdash 2 : \text{int}} \text{(intLit)} \quad \frac{\frac{}{\{y : \text{int}\} \vdash y : \text{int}} \text{(var)} \quad \frac{\frac{}{\{y : \text{int}\} \vdash y : \text{int}} \text{(var)}}{\{y : \text{int}\} \vdash y + y : \text{int}} \text{(intAdd)}}{\{\} \vdash \text{let } y = 2 \text{ in } y + y : \text{int}} \text{(let)}$$

1. Start from the bottom!
2. Apply the rule based on the expression

Repeat for the premises

Axioms (When are we done?)

$$\frac{}{\{\} \vdash 2 : \text{int}} \text{(intLit)} \quad \frac{}{\{y : \text{int}\} \vdash y : \text{int}} \text{(var)} \quad \frac{}{\{y : \text{int}\} \vdash y : \text{int}} \text{(var)} \quad \frac{}{\{y : \text{int}\} \vdash y + y : \text{int}} \text{(intAdd)}$$
$$\frac{}{\{\} \vdash \text{let } y = 2 \text{ in } y + y : \text{int}} \text{(let)}$$

Axioms (When are we done?)

$$\frac{}{\{\} \vdash 2 : \text{int}} \text{(intLit)} \quad \frac{}{\{y : \text{int}\} \vdash y : \text{int}} \text{(var)} \quad \frac{}{\{y : \text{int}\} \vdash y : \text{int}} \text{(var)} \quad \frac{}{\{y : \text{int}\} \vdash y + y : \text{int}} \text{(intAdd)}$$
$$\frac{}{\{\} \vdash \text{let } y = 2 \text{ in } y + y : \text{int}} \text{(let)}$$

We know that we can stop building a derivation once we need to derive a premise with an **axiom**, i.e., a rule with no premises

Axioms (When are we done?)

$$\frac{}{\{\} \vdash 2 : \text{int}} \text{(intLit)} \quad \frac{}{\{y : \text{int}\} \vdash y : \text{int}} \text{(var)} \quad \frac{}{\{y : \text{int}\} \vdash y : \text{int}} \text{(var)} \quad \frac{}{\{y : \text{int}\} \vdash y + y : \text{int}} \text{(intAdd)}$$
$$\frac{}{\{\} \vdash \text{let } y = 2 \text{ in } y + y : \text{int}} \text{(let)}$$

We know that we can stop building a derivation once we need to derive a premise with an **axiom**, i.e., a rule with no premises

In our case, this will almost always be "literal" or "variable" rules

Steps for a Typing Derivation

$$\frac{\frac{}{\{\} \vdash 2 : \text{int}} \text{(intLit)} \quad \frac{\frac{}{\{y : \text{int}\} \vdash y : \text{int}} \text{(var)} \quad \frac{\frac{}{\{y : \text{int}\} \vdash y : \text{int}} \text{(var)}}{\{y : \text{int}\} \vdash y + y : \text{int}} \text{(intAdd)}}{\{\} \vdash \text{let } y = 2 \text{ in } y + y : \text{int}} \text{(let)}$$

Steps for a Typing Derivation

$$\frac{\frac{}{\{\} \vdash 2 : \text{int}} \text{(intLit)} \quad \frac{\frac{}{\{y : \text{int}\} \vdash y : \text{int}} \text{(var)} \quad \frac{\frac{}{\{y : \text{int}\} \vdash y : \text{int}} \text{(var)}}{\{y : \text{int}\} \vdash y + y : \text{int}} \text{(intAdd)}}{\{\} \vdash \text{let } y = 2 \text{ in } y + y : \text{int}} \text{(let)}$$

1. Start from the bottom!

Steps for a Typing Derivation

$$\frac{\frac{}{\{\} \vdash 2 : \text{int}} \text{(intLit)} \quad \frac{\frac{}{\{y : \text{int}\} \vdash y : \text{int}} \text{(var)} \quad \frac{\frac{}{\{y : \text{int}\} \vdash y : \text{int}} \text{(var)}}{\{y : \text{int}\} \vdash y + y : \text{int}} \text{(intAdd)}}{\{\} \vdash \text{let } y = 2 \text{ in } y + y : \text{int}} \text{(let)}$$

1. Start from the bottom!
2. Apply the rule based on the expression

Steps for a Typing Derivation

$$\frac{\frac{}{\{\} \vdash 2 : \text{int}} \text{(intLit)} \quad \frac{\frac{}{\{y : \text{int}\} \vdash y : \text{int}} \text{(var)} \quad \frac{\frac{}{\{y : \text{int}\} \vdash y : \text{int}} \text{(var)}}{\{y : \text{int}\} \vdash y + y : \text{int}} \text{(intAdd)}}{\{\} \vdash \text{let } y = 2 \text{ in } y + y : \text{int}} \text{(let)}$$

1. Start from the bottom!

2. Apply the rule based on the expression

Repeat for premises until there are none left

How To Know Which Rule To Apply?

$$\frac{}{\{\} \vdash 2 : \text{int}} \text{(intLit)} \quad \frac{}{\{y : \text{int}\} \vdash y : \text{int}} \text{(var)} \quad \frac{}{\{y : \text{int}\} \vdash y : \text{int}} \text{(var)} \quad \frac{}{\{y : \text{int}\} \vdash y + y : \text{int}} \text{(intAdd)}$$
$$\frac{}{\{\} \vdash \text{let } y = 2 \text{ in } y + y : \text{int}} \text{(let)}$$

How To Know Which Rule To Apply?

$$\frac{}{\{\} \vdash 2 : \text{int}} \text{(intLit)} \quad \frac{}{\{y : \text{int}\} \vdash y : \text{int}} \text{(var)} \quad \frac{}{\{y : \text{int}\} \vdash y : \text{int}} \text{(var)} \quad \frac{}{\{y : \text{int}\} \vdash y + y : \text{int}} \text{(intAdd)}$$
$$\frac{}{\{\} \vdash \text{let } y = 2 \text{ in } y + y : \text{int}} \text{(let)}$$

Easy because of a very cool feature of OCaml:

How To Know Which Rule To Apply?

$$\frac{}{\{\} \vdash 2 : \text{int}} \text{(intLit)} \quad \frac{}{\{y : \text{int}\} \vdash y : \text{int}} \text{(var)} \quad \frac{}{\{y : \text{int}\} \vdash y : \text{int}} \text{(var)} \quad \frac{}{\{y : \text{int}\} \vdash y + y : \text{int}} \text{(intAdd)} \quad \frac{}{\{\} \vdash \text{let } y = 2 \text{ in } y + y : \text{int}} \text{(let)}$$

Easy because of a very cool feature of OCaml:

Syntax Directed Type System

How To Know Which Rule To Apply?

$$\frac{}{\{\} \vdash 2 : \text{int}} \text{(intLit)} \quad \frac{}{\{y : \text{int}\} \vdash y : \text{int}} \text{(var)} \quad \frac{}{\{y : \text{int}\} \vdash y : \text{int}} \text{(var)} \quad \frac{}{\{y : \text{int}\} \vdash y + y : \text{int}} \text{(intAdd)} \quad \frac{}{\{y : \text{int}\} \vdash y + y : \text{int}} \text{(let)} \\ \{\} \vdash \text{let } y = 2 \text{ in } y + y : \text{int}$$

Easy because of a very cool feature of OCaml:

Syntax Directed Type System

Only one typing rule per expression.

Integer Literals

1.

$$\frac{n \text{ is an int lit}}{\Gamma \vdash n : \text{int}} \quad (\text{intLit})$$

2.

$$\frac{n \text{ is an int lit}}{n \Downarrow n} \quad (\text{intLitEval})$$

1. If n is an integer literal, then it is of type int in any context
2. If n is an integer literal, then it evaluates to the number it represents

A Note about Side Conditions

we don't write "2 is an integer literal"

$$\frac{\boxed{\phantom{\text{side condition}}} \text{ (intLit)} \quad \frac{}{\{y : \text{int}\} \vdash y : \text{int}} \text{ (var)} \quad \frac{}{\{y : \text{int}\} \vdash y : \text{int}} \text{ (var)}}{\{y : \text{int}\} \vdash y + y : \text{int}} \text{ (intAdd)} \\ \frac{\{y : \text{int}\} \vdash y + y : \text{int}}{\{\} \vdash \text{let } y = 2 \text{ in } y + y : \text{int}} \text{ (let)}$$

If a premise is a side-condition this *it is not included in the derivation*

Side conditions need to hold in order to apply the rule, but they don't appear in the derivation itself

We will **try** to always write side conditions in **green**

Float Literals

1.

$$\frac{n \text{ is a float lit}}{\Gamma \vdash n : \text{float}} \text{ (floatLit)}$$

2.

$$\frac{n \text{ is a float lit}}{n \Downarrow n} \text{ (floatLitEval)}$$

1. If n is an float literal, then it is of type float in any context
2. If n is an float literal, then it evaluates to the number it represents

Boolean Literals

1.
$$\frac{}{\Gamma \vdash \text{true} : \text{bool}} \text{ (trueLit)}$$

2.
$$\frac{}{\Gamma \vdash \text{false} : \text{bool}} \text{ (falseLit)}$$

3.
$$\frac{}{\text{true} \Downarrow \top} \text{ (trueLitEval)}$$

4.
$$\frac{}{\text{false} \Downarrow \perp} \text{ (falseLitEval)}$$

1. `true` is of type `bool` in any context
2. `false` is of type `bool` in any context
3. `true` evaluates to the value \top
4. `false` evaluates to the value \perp

Variables

$$\frac{(v : \tau) \in \Gamma}{\Gamma \vdash v : \tau} \text{ (intLit)}$$

If v is declared to be of type τ in the context Γ , then v is of type τ in Γ

Variables cannot be evaluated (more on this when we talk about substitution and well-scopedness)

Okay, that was a lot,
let's do some examples

Back to the Example

$$\frac{}{\{\} \vdash 2 : \text{int}} \text{(intLit)} \quad \frac{}{\{y : \text{int}\} \vdash y : \text{int}} \text{(var)} \quad \frac{}{\{y : \text{int}\} \vdash y : \text{int}} \text{(var)} \quad \frac{}{\{y : \text{int}\} \vdash y + y : \text{int}} \text{(intAdd)}$$
$$\frac{}{\{\} \vdash \text{let } y = 2 \text{ in } y + y : \text{int}} \text{(let)}$$

Back to the Example

$$\frac{\frac{}{\{\} \vdash 2 : \text{int}} \text{(intLit)} \quad \frac{\frac{}{\{y : \text{int}\} \vdash y : \text{int}} \text{(var)} \quad \frac{\frac{}{\{y : \text{int}\} \vdash y : \text{int}} \text{(var)}}{\{y : \text{int}\} \vdash y + y : \text{int}} \text{(intAdd)}}{\{\} \vdash \text{let } y = 2 \text{ in } y + y : \text{int}} \text{(let)}$$

A derivation is just a mathy way of writing a natural language prove that a typing derivation holds

Back to the Example

$$\frac{\frac{}{\{\} \vdash 2 : \text{int}} \text{(intLit)} \quad \frac{\frac{}{\{y : \text{int}\} \vdash y : \text{int}} \text{(var)} \quad \frac{\frac{}{\{y : \text{int}\} \vdash y : \text{int}} \text{(var)}}{\{y : \text{int}\} \vdash y + y : \text{int}} \text{(intAdd)}}{\{\} \vdash \text{let } y = 2 \text{ in } y + y : \text{int}} \text{(let)}$$

A derivation is just a mathy way of writing a natural language prove that a typing derivation holds

*(In fact, most mathematical arguments can be represented formally as derivation trees, this is the called **proof theory**)*

Derivations Encode Natural Language Arguments

$$\frac{\frac{}{\{\} \vdash 2 : \text{int}} \text{(intLit)} \quad \frac{\frac{}{\{y : \text{int}\} \vdash y : \text{int}} \text{(var)} \quad \frac{\frac{}{\{y : \text{int}\} \vdash y : \text{int}} \text{(var)}}{\{y : \text{int}\} \vdash y + y : \text{int}} \text{(intAdd)}}{\{\} \vdash \text{let } y = 2 \text{ in } y + y : \text{int}} \text{(let)}$$

Derivations Encode Natural Language Arguments

$$\begin{array}{c}
 \frac{}{\{\} \vdash 2 : \text{int}} \text{(intLit)} \quad \frac{}{\{y : \text{int}\} \vdash y : \text{int}} \text{(var)} \quad \frac{}{\{y : \text{int}\} \vdash y : \text{int}} \text{(var)} \\
 \frac{}{\{y : \text{int}\} \vdash y + y : \text{int}} \text{(intAdd)} \\
 \frac{}{\{\} \vdash \text{let } y = 2 \text{ in } y + y : \text{int}} \text{(let)}
 \end{array}$$

$$\frac{\Gamma \vdash e_1 : \tau_1 \quad \Gamma, x : \tau_1 \vdash e_2 : \tau_2}{\Gamma \vdash \text{let } x = e_1 \text{ in } e_2 : \tau_2} \text{(let)}$$

The expression **let y = 2 in y + y** is an **int** *because*

Derivations Encode Natural Language Arguments

$$\begin{array}{c}
 \frac{}{\{\} \vdash 2 : \text{int}} \text{(intLit)} \quad \frac{}{\{y : \text{int}\} \vdash y : \text{int}} \text{(var)} \quad \frac{}{\{y : \text{int}\} \vdash y : \text{int}} \text{(var)} \\
 \hline
 \frac{}{\{y : \text{int}\} \vdash y + y : \text{int}} \text{(intAdd)} \\
 \hline
 \frac{}{\{\} \vdash \text{let } y = 2 \text{ in } y + y : \text{int}} \text{(let)}
 \end{array}$$

$$\frac{\Gamma \vdash e_1 : \tau_1 \quad \Gamma, x : \tau_1 \vdash e_2 : \tau_2}{\Gamma \vdash \text{let } x = e_1 \text{ in } e_2 : \tau_2} \text{(let)}$$

The expression **let y = 2 in y + y** is an **int** *because*

» **2** is an **int** by fiat (and so **y** is being assigned to a well-typed expression)

Derivations Encode Natural Language Arguments

$$\begin{array}{c}
 \frac{}{\{\} \vdash 2 : \text{int}} \text{(intLit)} \quad \frac{}{\{y : \text{int}\} \vdash y : \text{int}} \text{(var)} \quad \frac{}{\{y : \text{int}\} \vdash y : \text{int}} \text{(var)} \\
 \frac{}{\{y : \text{int}\} \vdash y + y : \text{int}} \text{(intAdd)} \\
 \hline
 \{\} \vdash \text{let } y = 2 \text{ in } y + y : \text{int} \text{ (let)}
 \end{array}$$

$$\frac{\Gamma \vdash e_1 : \tau_1 \quad \Gamma, x : \tau_1 \vdash e_2 : \tau_2}{\Gamma \vdash \text{let } x = e_1 \text{ in } e_2 : \tau_2} \text{(let)}$$

$$\frac{\text{n is an integer literal}}{\Gamma \vdash \text{n} : \text{int}} \text{(intLit)}$$

The expression `let y = 2 in y + y` is an `int` because

» `2` is an `int` by fiat (and so `y` is being assigned to a well-typed expression)

Derivations Encode Natural Language Arguments

$$\begin{array}{c}
 \frac{}{\{\} \vdash 2 : \text{int}} \text{(intLit)} \quad \frac{}{\{y : \text{int}\} \vdash y : \text{int}} \text{(var)} \quad \frac{}{\{y : \text{int}\} \vdash y : \text{int}} \text{(var)} \\
 \hline
 \frac{}{\{y : \text{int}\} \vdash y + y : \text{int}} \text{(intAdd)} \\
 \hline
 \frac{}{\{\} \vdash \text{let } y = 2 \text{ in } y + y : \text{int}} \text{(let)}
 \end{array}$$

$$\frac{\Gamma \vdash e_1 : \tau_1 \quad \Gamma, x : \tau_1 \vdash e_2 : \tau_2}{\Gamma \vdash \text{let } x = e_1 \text{ in } e_2 : \tau_2} \text{(let)}$$

$$\frac{\text{n is an integer literal}}{\Gamma \vdash n : \text{int}} \text{(intLit)}$$

The expression `let y = 2 in y + y` is an `int` because

» `2` is an `int` by fiat (and so `y` is being assigned to a well-typed

» and, assuming `y` is an `int`, `y + y` is an `int` because

$$\frac{\Gamma \vdash e_1 : \text{int} \quad \Gamma \vdash e_2 : \text{int}}{\Gamma \vdash e_1 + e_2 : \text{int}} \text{(addInt)}$$

Derivations Encode Natural Language Arguments

$$\frac{\frac{\frac{\checkmark}{\{\} \vdash 2 : \text{int}}{\{\} \vdash \text{let } y = 2 \text{ in } y + y : \text{int}} \text{ (intLit)}}{\frac{\frac{\frac{\{\} \vdash 2 : \text{int}}{\{\} \vdash \text{let } y = 2 \text{ in } y + y : \text{int}} \text{ (intLit)}}{\{\} \vdash \text{let } y = 2 \text{ in } y + y : \text{int}} \text{ (let)}} \text{ (var)}}{\frac{\frac{\frac{\{\} \vdash 2 : \text{int}}{\{\} \vdash \text{let } y = 2 \text{ in } y + y : \text{int}} \text{ (intLit)}}{\{\} \vdash \text{let } y = 2 \text{ in } y + y : \text{int}} \text{ (let)}} \text{ (var)}} \text{ (intAdd)}$$

$$\frac{\Gamma \vdash e_1 : \tau_1 \quad \Gamma, x : \tau_1 \vdash e_2 : \tau_2}{\Gamma \vdash \text{let } x = e_1 \text{ in } e_2 : \tau_2} \text{ (let)}$$

$$\frac{\text{n is an integer literal}}{\Gamma \vdash \mathbf{n} : \mathbf{int}} \quad (\text{intLit})$$

The expression **let y = 2 in y + y** is an **int** *because*

» **2** is an **int** by fiat (and so **y** is being assigned to a well-typed

» and, assuming **y** is an int, **y + y** is an **int** *because*

- **y** is an **int** (by assumption)

$$\frac{\Gamma \vdash e_1 : \text{int} \quad \Gamma \vdash e_2 : \text{int}}{\Gamma \vdash e_1 + e_2 : \text{int}} \text{ (addInt)}$$

Derivations Encode Natural Language Arguments

$$\begin{array}{c}
 \frac{}{\{\} \vdash 2 : \text{int}} \text{(intLit)} \quad \frac{\frac{}{\{\} \vdash 2 : \text{int}} \text{(intLit)} \quad \frac{}{\{\} \vdash 2 : \text{int}} \text{(intLit)}}{\{\} \vdash \text{let } y = 2 \text{ in } y + y : \text{int}} \text{(let)} \\
 \frac{}{\{\} \vdash 2 : \text{int}} \text{(intLit)} \quad \frac{\frac{}{\{\} \vdash 2 : \text{int}} \text{(intLit)} \quad \frac{}{\{\} \vdash 2 : \text{int}} \text{(intLit)}}{\{\} \vdash \text{let } y = 2 \text{ in } y + y : \text{int}} \text{(let)}
 \end{array}$$

$$\frac{\Gamma \vdash e_1 : \tau_1 \quad \Gamma, x : \tau_1 \vdash e_2 : \tau_2}{\Gamma \vdash \text{let } x = e_1 \text{ in } e_2 : \tau_2} \text{(let)}$$

$$\frac{\text{n is an integer literal}}{\Gamma \vdash \text{n} : \text{int}} \text{(intLit)}$$

The expression `let y = 2 in y + y` is an `int` because

» `2` is an `int` by fiat (and so `y` is being assigned to a well-typed

» and, assuming `y` is an `int`, `y + y` is an `int` because

$$\frac{\Gamma \vdash e_1 : \text{int} \quad \Gamma \vdash e_2 : \text{int}}{\Gamma \vdash e_1 + e_2 : \text{int}} \text{(addInt)}$$

- `y` is an `int` (by assumption)

$$\frac{(x : \tau) \in \Gamma}{\Gamma \vdash x : \tau} \text{(var)}$$

Derivations Encode Natural Language Arguments

$$\begin{array}{c}
 \frac{}{\{\} \vdash 2 : \text{int}} \text{(intLit)} \quad \frac{}{\{y : \text{int}\} \vdash y : \text{int}} \text{(var)} \quad \frac{}{\{y : \text{int}\} \vdash y : \text{int}} \text{(var)} \\
 \frac{}{\{y : \text{int}\} \vdash y + y : \text{int}} \text{(let)} \quad \frac{}{\{y : \text{int}\} \vdash y : \text{int}} \text{(intAdd)} \\
 \hline
 \{\} \vdash \text{let } y = 2 \text{ in } y + y : \text{int}
 \end{array}$$

$$\frac{\Gamma \vdash e_1 : \tau_1 \quad \Gamma, x : \tau_1 \vdash e_2 : \tau_2}{\Gamma \vdash \text{let } x = e_1 \text{ in } e_2 : \tau_2} \text{(let)}$$

$$\frac{\text{n is an integer literal}}{\Gamma \vdash \text{n} : \text{int}} \text{(intLit)}$$

The expression `let y = 2 in y + y` is an `int` because

» `2` is an `int` by fiat (and so `y` is being assigned to a well-typed

» and, assuming `y` is an `int`, `y + y` is an `int` because

$$\frac{\Gamma \vdash e_1 : \text{int} \quad \Gamma \vdash e_2 : \text{int}}{\Gamma \vdash e_1 + e_2 : \text{int}} \text{(addInt)}$$

$$\frac{(x : \tau) \in \Gamma}{\Gamma \vdash x : \tau} \text{(var)}$$

- `y` is an `int` (by assumption)
- and so is `y` (by assumption)

Derivations Encode Natural Language Arguments

$$\begin{array}{c}
 \frac{}{\{\} \vdash 2 : \text{int}} \text{(intLit)} \quad \frac{}{\{y : \text{int}\} \vdash y : \text{int}} \text{(var)} \quad \frac{}{\{y : \text{int}\} \vdash y : \text{int}} \text{(var)} \\
 \frac{}{\{y : \text{int}\} \vdash y + y : \text{int}} \text{(intAdd)} \\
 \hline
 \{\} \vdash \text{let } y = 2 \text{ in } y + y : \text{int} \text{ (let)}
 \end{array}$$

$$\frac{\Gamma \vdash e_1 : \tau_1 \quad \Gamma, x : \tau_1 \vdash e_2 : \tau_2}{\Gamma \vdash \text{let } x = e_1 \text{ in } e_2 : \tau_2} \text{(let)}$$

$$\frac{\text{n is an integer literal}}{\Gamma \vdash \text{n} : \text{int}} \text{(intLit)}$$

The expression `let y = 2 in y + y` is an `int` because

- » `2` is an `int` by fiat (and so `y` is being assigned to a well-typed
- » and, assuming `y` is an `int`, `y + y` is an `int` because

$$\frac{\Gamma \vdash e_1 : \text{int} \quad \Gamma \vdash e_2 : \text{int}}{\Gamma \vdash e_1 + e_2 : \text{int}} \text{(addInt)}$$

- `y` is an `int` (by assumption)
- and so is `y` (by assumption)

$$\frac{(x : \tau) \in \Gamma}{\Gamma \vdash x : \tau} \text{(var)}$$

Derivations Encode Natural Language Arguments

$$\begin{array}{c}
 \frac{}{\{\} \vdash 2 : \text{int}} \text{(intLit)} \quad \frac{}{\{y : \text{int}\} \vdash y : \text{int}} \text{(var)} \quad \frac{}{\{y : \text{int}\} \vdash y : \text{int}} \text{(var)} \\
 \hline
 \frac{}{\{y : \text{int}\} \vdash y + y : \text{int}} \text{(intAdd)} \\
 \hline
 \{\} \vdash \text{let } y = 2 \text{ in } y + y : \text{int} \text{(let)}
 \end{array}$$

$$\frac{\Gamma \vdash e_1 : \tau_1 \quad \Gamma, x : \tau_1 \vdash e_2 : \tau_2}{\Gamma \vdash \text{let } x = e_1 \text{ in } e_2 : \tau_2} \text{(let)}$$

$$\frac{\text{n is an integer literal}}{\Gamma \vdash \text{n} : \text{int}} \text{(intLit)}$$

The expression `let y = 2 in y + y` is an `int` because

» `2` is an `int` by fiat (and so `y` is being assigned to a well-typed

» and, assuming `y` is an `int`, `y + y` is an `int` because

$$\frac{\Gamma \vdash e_1 : \text{int} \quad \Gamma \vdash e_2 : \text{int}}{\Gamma \vdash e_1 + e_2 : \text{int}} \text{(addInt)}$$

- `y` is an `int` (by assumption)
- and so is `y` (by assumption)

$$\frac{(x : \tau) \in \Gamma}{\Gamma \vdash x : \tau} \text{(var)}$$

and so integer-adding these two expressions (`y` and `y`) yields an `int`

Derivations Encode Natural Language Arguments

$$\begin{array}{c}
 \frac{}{\{\} \vdash 2 : \text{int}} \text{(intLit)} \quad \frac{}{\{y : \text{int}\} \vdash y : \text{int}} \text{(var)} \quad \frac{}{\{y : \text{int}\} \vdash y : \text{int}} \text{(var)} \\
 \frac{}{\{y : \text{int}\} \vdash y + y : \text{int}} \text{(intAdd)} \\
 \hline
 \{\} \vdash \text{let } y = 2 \text{ in } y + y : \text{int} \text{ (let)}
 \end{array}$$

$$\frac{\Gamma \vdash e_1 : \tau_1 \quad \Gamma, x : \tau_1 \vdash e_2 : \tau_2}{\Gamma \vdash \text{let } x = e_1 \text{ in } e_2 : \tau_2} \text{(let)}$$

$$\frac{\text{n is an integer literal}}{\Gamma \vdash \text{n} : \text{int}} \text{(intLit)}$$

The expression `let y = 2 in y + y` is an `int` because

» `2` is an `int` by fiat (and so `y` is being assigned to a well-typed

» and, assuming `y` is an `int`, `y + y` is an `int` because

$$\frac{\Gamma \vdash e_1 : \text{int} \quad \Gamma \vdash e_2 : \text{int}}{\Gamma \vdash e_1 + e_2 : \text{int}} \text{(addInt)}$$

- `y` is an `int` (by assumption)
- and so is `y` (by assumption)

and so integer-adding these two expressions (`y` and `y`) yields an `int`

and so assigning `y` to `2` in `y + y` yields an `int`

$$\frac{(x : \tau) \in \Gamma}{\Gamma \vdash x : \tau} \text{(var)}$$

Derivations Encode Natural Language Arguments

$$\begin{array}{c}
 \frac{}{\{\} \vdash 2 : \text{int}} \text{(intLit)} \quad \frac{}{\{y : \text{int}\} \vdash y : \text{int}} \text{(var)} \quad \frac{}{\{y : \text{int}\} \vdash y : \text{int}} \text{(var)} \\
 \frac{}{\{y : \text{int}\} \vdash y + y : \text{int}} \text{(intAdd)} \\
 \hline
 \{\} \vdash \text{let } y = 2 \text{ in } y + y : \text{int} \text{(let)}
 \end{array}$$

$$\frac{\Gamma \vdash e_1 : \tau_1 \quad \Gamma, x : \tau_1 \vdash e_2 : \tau_2}{\Gamma \vdash \text{let } x = e_1 \text{ in } e_2 : \tau_2} \text{(let)}$$

$$\frac{\text{n is an integer literal}}{\Gamma \vdash \text{n} : \text{int}} \text{(intLit)}$$

The expression `let y = 2 in y + y` is an `int` because

» `2` is an `int` by fiat (and so `y` is being assigned to a well-typed

» and, assuming `y` is an `int`, `y + y` is an `int` because

$$\frac{\Gamma \vdash e_1 : \text{int} \quad \Gamma \vdash e_2 : \text{int}}{\Gamma \vdash e_1 + e_2 : \text{int}} \text{(addInt)}$$

- `y` is an `int` (by assumption)
- and so is `y` (by assumption)

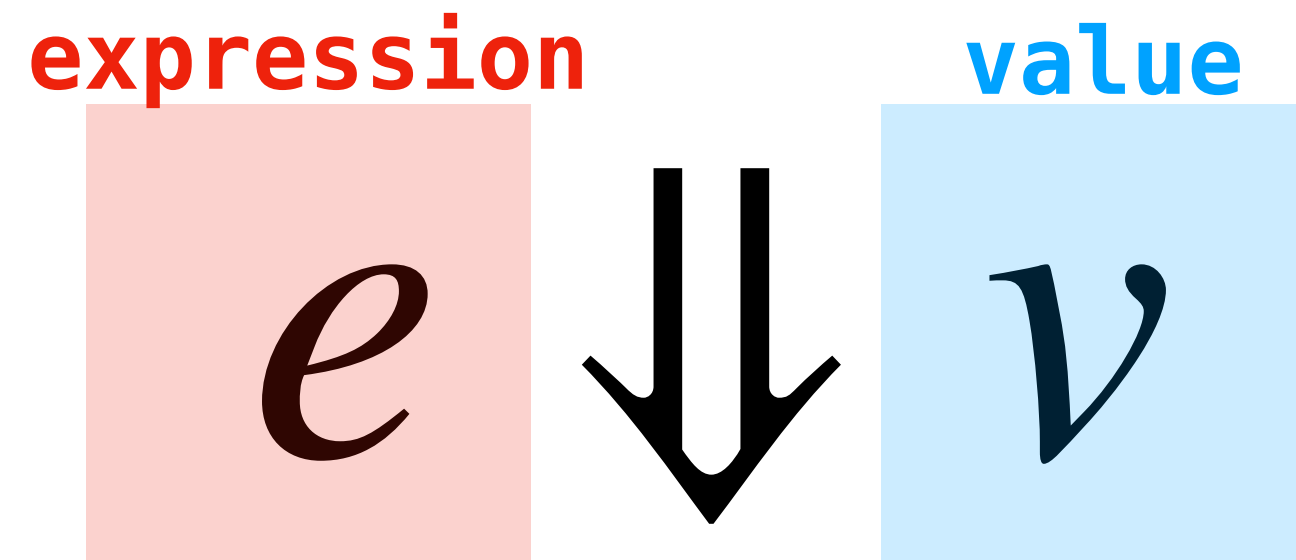
$$\frac{(x : \tau) \in \Gamma}{\Gamma \vdash x : \tau} \text{(var)}$$

and so integer-adding these two expressions (`y` and `y`) yields an `int`

and so assigning `y` to `2` in `y + y` yields an `int`

And all this works for
semantics judgements as well

Recall: Semantic Judgements



A semantic judgment is a compact way of representing the statement:

The expression e evaluates to the value v

A **semantic rule** is an inference rule with semantic judgments

Recall: Integer Addition Semantic Rule

$$\frac{e_1 \Downarrow v_1 \quad e_2 \Downarrow v_2 \quad v_1 + v_2 = v}{e_1 + e_2 \Downarrow v} \text{ (evalInt)}$$

If e_1 evaluates to the (integer) v_1 and e_2 evaluates to the (integer) v_2 , and $v_1 + v_2 = v$, then $e_1 + e_2$ evaluates to the (integer) v

Semantic Derivations

$$\frac{\frac{}{\text{true} \Downarrow \top} \text{ (trueEval)} \quad \frac{}{2 \Downarrow 2} \text{ (intEval)}}{\text{if true then 2 else 3} \Downarrow 2} \text{ (ifEval)}$$

We can also write derivations to prove semantic judgments

The principle is the same, except that the judgments are semantic judgments instead of typing judgments

Examples

Example (Typing)

$\{\} \vdash \text{if true then 2 else 5} : \text{int}$

Example (Evaluation)

if true then 2 else 5 \Downarrow 2

Example (Typing)

$\{x : \text{int}\} \vdash 3 + x \triangleleft 4 : \text{bool}$

Example (Evaluation)

let $x = 2$ in $3 + x \Downarrow 5$

Summary

Derivations are **tree-like proofs** that judgments hold with respect to a collection of inference rules

Derivations are **compact mathematical representations** of English language arguments

Learning to write derivations takes *practice*