Derivations

Concepts of Programming Languages
Lecture 6

CAS CS 320

Practice Problem

Suppose introduced an xor operator into OCaml. Write down (to the best of your ability) the syntax, typing, and semantic rules for xor

Outline

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

Recap

Recall: Inference Rules

$$\dfrac{\overset{\text{premise}}{P_1} \qquad \overset{\text{premise}}{P_2} \qquad \cdots \qquad \overset{\text{premise}}{P_k}}{\underset{\text{conclusion}}{C}}\;\overset{\text{name}}{\texttt{my-rule}}$$

Recall: Inference Rules

$$\frac{\overset{\text{premise}}{P_1} \qquad \overset{\text{premise}}{P_2} \qquad \ldots \qquad \overset{\text{premise}}{P_k}}{\underset{\text{conclusion}}{C}}\ \overset{\text{name}}{\texttt{my-rule}}$$

The general form of an inference rule has a collection of **premises** and a **conclusion** all of which are **judgments**

Recall: Inference Rules

$$\frac{\overset{\text{premise}}{P_1} \qquad \overset{\text{premise}}{P_2} \qquad \cdots \qquad \overset{\text{premise}}{P_k}}{\underset{\text{conclusion}}{C}}\ \overset{\text{name}}{\texttt{my-rule}}$$

The general form of an inference rule has a collection of **premises** and a **conclusion** all of which are **judgments**

There may be no premises, this is called an **axiom**

Recall: Inference Rules

$$\frac{\overset{\text{premise}}{P_1} \qquad \overset{\text{premise}}{P_2} \qquad \cdots \qquad \overset{\text{premise}}{P_k}}{\underset{\text{conclusion}}{C}}\ \overset{\text{name}}{\texttt{my-rule}}$$

<u>We can read this as:</u>

If the judgments P_1 through P_k hold, then the judgment C holds (by ***my-rule****)*

Recall: Typing Judgments

$$\underbrace{\Gamma}_{\text{context}} \vdash \underbrace{e}_{\text{expression}} : \underbrace{\tau}_{\text{type}}$$

A typing judgment a compact way of representing the statement:

e is of type τ in the context Γ

A **typing rule** is an inference rule whose premises and conclusion are typing judgments

Recall: Contexts

$\Gamma = \{$ `x : int, y : string, z : int -> string` $\}$

Recall: Contexts

$\Gamma = \{$ `x : int, y : string, z : int -> string` $\}$

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

Recall: Contexts

$$\Gamma = \{ \texttt{x : int, y : string, z : int -> string} \}$$

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

$$\Gamma = \{\ \texttt{x : int, y : string, z : int -> string}\ \}$$

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

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

High Level

$$\dfrac{\dfrac{}{\{\} \vdash \texttt{2} : \texttt{int}}\,(\text{intLit}) \qquad \dfrac{\dfrac{}{\{\texttt{y} : \texttt{int}\} \vdash \texttt{y} : \texttt{int}}\,(\text{var}) \qquad \dfrac{}{\{\texttt{y} : \texttt{int}\} \vdash \texttt{y} : \texttt{int}}\,(\text{var})}{\{\texttt{y} : \texttt{int}\} \vdash \texttt{y} + \texttt{y} : \texttt{int}}\,(\text{intAdd})}{\{\} \vdash \texttt{let y = 2 in } \texttt{y} + \texttt{y} : \texttt{int}}\,(\text{let})$$

Derivations *prove* that a judgment holds w.r.t some rules

High Level

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

Derivations *prove* that a judgment holds w.r.t some rules

A **derivation** is a tree in which:

High Level

$$\dfrac{\dfrac{}{\{\} \vdash \texttt{2} : \texttt{int}}\,(\text{intLit}) \qquad \dfrac{\dfrac{}{\{\texttt{y} : \texttt{int}\} \vdash \texttt{y} : \texttt{int}}\,(\text{var}) \qquad \dfrac{}{\{\texttt{y} : \texttt{int}\} \vdash \texttt{y} : \texttt{int}}\,(\text{var})}{\{\texttt{y} : \texttt{int}\} \vdash \texttt{y} + \texttt{y} : \texttt{int}}\,(\text{intAdd})}{\{\} \vdash \texttt{let y = 2 in y} + \texttt{y} : \texttt{int}}\,(\text{let})$$

Derivations *prove* that a judgment holds w.r.t some rules

A **derivation** is a tree in which:

» each node is labeled with a judgment

High Level

$$\dfrac{\dfrac{}{\{\} \vdash \texttt{2} : \texttt{int}}\ (\text{intLit}) \qquad \dfrac{\dfrac{}{\{\texttt{y} : \texttt{int}\} \vdash \texttt{y} : \texttt{int}}\ (\text{var}) \qquad \dfrac{}{\{\texttt{y} : \texttt{int}\} \vdash \texttt{y} : \texttt{int}}\ (\text{var})}{\{\texttt{y} : \texttt{int}\} \vdash \texttt{y} + \texttt{y} : \texttt{int}}\ (\text{intAdd})}{\{\} \vdash \texttt{let y = 2 in y} + \texttt{y} : \texttt{int}}\ (\text{let})$$

Derivations *prove* that a judgment holds w.r.t some rules

A **derivation** is a tree in which:

» each node is labeled with a judgment

» and judgment *follows* from the judgments at it's children by an inference rule

Applying Rules

$$\frac{}{\Gamma \vdash \texttt{[]} : \tau\ \texttt{list}}\ \text{(nil)} \qquad \frac{\Gamma \vdash e_1 : \tau \qquad \Gamma \vdash e_2 : \tau\ \texttt{list}}{\Gamma \vdash e_1\ \texttt{::}\ e_2 : \tau\ \texttt{list}}\ \text{(cons)}$$

$$\frac{\{\texttt{x} : \texttt{int}\} \vdash \texttt{x + 1} : \texttt{int} \qquad \{\texttt{x} : \texttt{int}\} \vdash \texttt{[]} : \texttt{int list}}{\{\texttt{x} : \texttt{int}\} \vdash \texttt{(x + 1) :: []} : \texttt{int list}}\ \text{(cons)}$$

Applying Rules

$$\frac{}{\Gamma \vdash \texttt{[]} : \tau\ \texttt{list}}\ (\text{nil}) \qquad \frac{\Gamma \vdash e_1 : \tau \qquad \Gamma \vdash e_2 : \tau\ \texttt{list}}{\Gamma \vdash e_1\ \texttt{::}\ e_2 : \tau\ \texttt{list}}\ (\text{cons})$$

$$\frac{\{\texttt{x} : \texttt{int}\} \vdash \texttt{x + 1} : \texttt{int} \qquad \{\texttt{x} : \texttt{int}\} \vdash \texttt{[]} : \texttt{int list}}{\{\texttt{x} : \texttt{int}\} \vdash \texttt{(x + 1) :: []} : \texttt{int list}}\ (\text{cons})$$

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

Applying Rules

$$\frac{}{\Gamma \vdash \texttt{[]} : \tau\ \texttt{list}}\ \text{(nil)} \qquad \frac{\Gamma \vdash e_1 : \tau \qquad \Gamma \vdash e_2 : \tau\ \texttt{list}}{\Gamma \vdash e_1\ \texttt{::}\ e_2 : \tau\ \texttt{list}}\ \text{(cons)}$$

$$\frac{\{\mathtt{x} : \mathtt{int}\} \vdash \mathtt{x + 1} : \mathtt{int} \qquad \{\mathtt{x} : \mathtt{int}\} \vdash \mathtt{[]} : \mathtt{int\ list}}{\{\mathtt{x} : \mathtt{int}\} \vdash \mathtt{(x + 1)\ ::\ []} : \mathtt{int\ list}}\ \text{(cons)}$$

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

$$
\dfrac{\dfrac{\dfrac{}{\{\mathtt{x} : \mathtt{int}\} \vdash \mathtt{x} : \mathtt{int}}\ (\mathsf{var}) \qquad \dfrac{}{\{\mathtt{x} : \mathtt{int}\} \vdash \mathtt{1} : \mathtt{int}}\ (\mathsf{intLit})}{\{\mathtt{x} : \mathtt{int}\} \vdash \mathtt{x + 1} : \mathtt{int}}\ (\mathsf{intAdd}) \qquad \dfrac{}{\{\mathtt{x} : \mathtt{int}\} \vdash \mathtt{[]} : \mathtt{int\ list}}\ (\mathsf{nil})}{\{\mathtt{x} : \mathtt{int}\} \vdash \mathtt{(x + 1)\ ::\ []} : \mathtt{int\ list}}\ (\mathsf{cons})
$$

Building from the Ground Up

$$\dfrac{\dfrac{\dfrac{}{\{\texttt{x} : \texttt{int}\} \vdash \texttt{x} : \texttt{int}}\,(\text{var}) \qquad \dfrac{}{\{\texttt{x} : \texttt{int}\} \vdash \texttt{1} : \texttt{int}}\,(\text{intLit})}{\{\texttt{x} : \texttt{int}\} \vdash \texttt{x} + \texttt{1} : \texttt{int}}\,(\text{intAdd}) \qquad \dfrac{}{\{\texttt{x} : \texttt{int}\} \vdash \texttt{[]} : \texttt{int list}}\,(\text{nil})}{\{\texttt{x} : \texttt{int}\} \vdash \texttt{(x + 1) :: []} : \texttt{int list}}\,(\text{cons})$$

But we can't *just* apply rules, because it's possible that the premises of a rule **also need to be demonstrated**

Building from the Ground Up

$$\dfrac{\dfrac{\dfrac{}{\{\mathtt{x} : \mathtt{int}\} \vdash \mathtt{x} : \mathtt{int}}\,(\text{var}) \qquad \dfrac{}{\{\mathtt{x} : \mathtt{int}\} \vdash \mathtt{1} : \mathtt{int}}\,(\text{intLit})}{\{\mathtt{x} : \mathtt{int}\} \vdash \mathtt{x} + \mathtt{1} : \mathtt{int}}\,(\text{intAdd}) \qquad \dfrac{}{\{\mathtt{x} : \mathtt{int}\} \vdash \mathtt{[]} : \mathtt{int\ list}}\,(\text{nil})}{\{\mathtt{x} : \mathtt{int}\} \vdash \mathtt{(x + 1)\ ::\ []} : \mathtt{int\ list}}\,(\text{cons})$$

But we can't *just* apply rules, because it's possible that the premises of a rule **also need to be demonstrated**

This is how we get our tree structure: we apply rules from the ground up

Axioms (When are we done?)

$$\dfrac{\dfrac{\dfrac{}{\{\texttt{x} : \texttt{int}\} \vdash \texttt{x} : \texttt{int}}\,(\text{var}) \qquad \dfrac{}{\{\texttt{x} : \texttt{int}\} \vdash \texttt{1} : \texttt{int}}\,(\text{intLit})}{\{\texttt{x} : \texttt{int}\} \vdash \texttt{x + 1} : \texttt{int}}\,(\text{intAdd}) \qquad \dfrac{}{\{\texttt{x} : \texttt{int}\} \vdash \texttt{[]} : \texttt{int list}}\,(\text{nil})}{\{\texttt{x} : \texttt{int}\} \vdash \texttt{(x + 1) :: []} : \texttt{int list}}\,(\text{cons})$$

Axioms (When are we done?)

$$\dfrac{\dfrac{\dfrac{}{\{\texttt{x} : \texttt{int}\} \vdash \texttt{x} : \texttt{int}}\,(\text{var}) \qquad \dfrac{}{\{\texttt{x} : \texttt{int}\} \vdash \texttt{1} : \texttt{int}}\,(\text{intLit})}{\{\texttt{x} : \texttt{int}\} \vdash \texttt{x + 1} : \texttt{int}}\,(\text{intAdd}) \qquad \dfrac{}{\{\texttt{x} : \texttt{int}\} \vdash \texttt{[]} : \texttt{int list}}\,(\text{nil})}{\{\texttt{x} : \texttt{int}\} \vdash \texttt{(x + 1) :: []} : \texttt{int list}}\,(\text{cons})$$

The leaves of the tree are **axioms**, i.e., a rules with no premises

Axioms (When are we done?)

$$\dfrac{\dfrac{\dfrac{}{\{\texttt{x} : \texttt{int}\} \vdash \texttt{x} : \texttt{int}}\,\text{(var)} \qquad \dfrac{}{\{\texttt{x} : \texttt{int}\} \vdash \texttt{1} : \texttt{int}}\,\text{(intLit)}}{\{\texttt{x} : \texttt{int}\} \vdash \texttt{x + 1} : \texttt{int}}\,\text{(intAdd)} \qquad \dfrac{}{\{\texttt{x} : \texttt{int}\} \vdash \texttt{[]} : \texttt{int list}}\,\text{(nil)}}{\{\texttt{x} : \texttt{int}\} \vdash \texttt{(x + 1) :: []} : \texttt{int list}}\,\text{(cons)}$$

The leaves of the tree are **axioms**, i.e., a rules with no premises

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

Integer Literals

(1)

$$\frac{\texttt{n is an int lit}}{\Gamma \vdash \mathsf{n} : \texttt{int}}\ (\mathrm{intLit})$$

(2)

$$\frac{\texttt{n is an int lit}}{\mathsf{n} \Downarrow n}\ (\mathrm{intLitEval})$$

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 "1 is an integer literal"

$$\dfrac{\dfrac{\dfrac{}{\{\texttt{x} : \texttt{int}\} \vdash \texttt{x} : \texttt{int}}\,(\text{var}) \qquad \dfrac{}{\{\texttt{x} : \texttt{int}\} \vdash \texttt{1} : \texttt{int}}\,(\text{intLit})}{\{\texttt{x} : \texttt{int}\} \vdash \texttt{x + 1} : \texttt{int}}\,(\text{intAdd}) \qquad \dfrac{}{\{\texttt{x} : \texttt{int}\} \vdash \texttt{[]} : \texttt{int list}}\,(\text{nil})}{\{\texttt{x} : \texttt{int}\} \vdash \texttt{(x + 1) :: []} : \texttt{int list}}\,(\text{cons})$$

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 always make side conditions clear

Float Literals

(1)

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

(2)

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

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

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

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

(4)
$$\frac{}{\texttt{false} \Downarrow \bot}\ (\text{falseLitEval})$$

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

Variables

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

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)

Back to the Example

$$\dfrac{\dfrac{}{\{\} \vdash \texttt{2} : \texttt{int}}\ (\mathsf{intLit}) \qquad \dfrac{\dfrac{}{\{\texttt{y} : \texttt{int}\} \vdash \texttt{y} : \texttt{int}}\ (\mathsf{var}) \qquad \dfrac{}{\{\texttt{y} : \texttt{int}\} \vdash \texttt{y} : \texttt{int}}\ (\mathsf{var})}{\{\texttt{y} : \texttt{int}\} \vdash \texttt{y} + \texttt{y} : \texttt{int}}\ (\mathsf{intAdd})}{\{\} \vdash \texttt{let y = 2 in } \texttt{y} + \texttt{y} : \texttt{int}}\ (\mathsf{let})$$

Back to the Example

$$
\dfrac{\dfrac{}{\{\} \vdash \texttt{2} : \texttt{int}}\;(\text{intLit}) \qquad \dfrac{\dfrac{}{\{\texttt{y} : \texttt{int}\} \vdash \texttt{y} : \texttt{int}}\;(\text{var}) \qquad \dfrac{}{\{\texttt{y} : \texttt{int}\} \vdash \texttt{y} : \texttt{int}}\;(\text{var})}{\{\texttt{y} : \texttt{int}\} \vdash \texttt{y} + \texttt{y} : \texttt{int}}\;(\text{intAdd})}{\{\} \vdash \texttt{let y = 2 in } \texttt{y} + \texttt{y} : \texttt{int}}\;(\text{let})
$$

We need $\{\} \vdash 2 : \texttt{int}$ in order to proof that the bottom typing judgment holds

Back to the Example

$$\dfrac{\dfrac{}{\{\} \vdash \mathtt{2} : \mathtt{int}}\;(\mathsf{intLit}) \qquad \dfrac{\dfrac{}{\{\mathtt{y} : \mathtt{int}\} \vdash \mathtt{y} : \mathtt{int}}\;(\mathsf{var}) \qquad \dfrac{}{\{\mathtt{y} : \mathtt{int}\} \vdash \mathtt{y} : \mathtt{int}}\;(\mathsf{var})}{\{\mathtt{y} : \mathtt{int}\} \vdash \mathtt{y} + \mathtt{y} : \mathtt{int}}\;(\mathsf{intAdd})}{\{\} \vdash \mathtt{let\ y\ =\ 2\ in\ y} + \mathtt{y} : \mathtt{int}}\;(\mathsf{let})$$

We need $\{\} \vdash \mathtt{2} : \mathtt{int}$ in order to proof that the bottom typing judgment holds

Now we know that this follows from the **intLit** rule, which says that 2 is always an int, *by fiat*

Okay, I know that was a lot, let's take a step back

Derivations Encode Natural Language Arguments

$$
\begin{prooftree}
\infer0[(intLit)]{\{\} \vdash \texttt{2} : \texttt{int}}
\infer0[(var)]{\{\texttt{y} : \texttt{int}\} \vdash \texttt{y} : \texttt{int}}
\infer0[(var)]{\{\texttt{y} : \texttt{int}\} \vdash \texttt{y} : \texttt{int}}
\infer2[(intAdd)]{\{\texttt{y} : \texttt{int}\} \vdash \texttt{y} + \texttt{y} : \texttt{int}}
\infer2[(let)]{\{\} \vdash \texttt{let y = 2 in } \texttt{y} + \texttt{y} : \texttt{int}}
\end{prooftree}
$$

Derivations Encode Natural Language Arguments

$$\dfrac{\dfrac{}{\{\} \vdash \texttt{2} : \texttt{int}}\;(\mathsf{intLit}) \qquad \dfrac{\dfrac{}{\{\texttt{y} : \texttt{int}\} \vdash \texttt{y} : \texttt{int}}\;(\mathsf{var}) \qquad \dfrac{}{\{\texttt{y} : \texttt{int}\} \vdash \texttt{y} : \texttt{int}}\;(\mathsf{var})}{\{\texttt{y} : \texttt{int}\} \vdash \texttt{y} + \texttt{y} : \texttt{int}}\;(\mathsf{intAdd})}{\{\} \vdash \texttt{let y = 2 in } \texttt{y} + \texttt{y} : \texttt{int}}\;(\mathsf{let})$$

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

Derivations Encode Natural Language Arguments

$$\dfrac{\dfrac{}{\{\} \vdash \texttt{2} : \texttt{int}}\ (\text{intLit}) \qquad \dfrac{\dfrac{}{\{\texttt{y} : \texttt{int}\} \vdash \texttt{y} : \texttt{int}}\ (\text{var}) \qquad \dfrac{}{\{\texttt{y} : \texttt{int}\} \vdash \texttt{y} : \texttt{int}}\ (\text{var})}{\{\texttt{y} : \texttt{int}\} \vdash \texttt{y} + \texttt{y} : \texttt{int}}\ (\text{intAdd})}{\{\} \vdash \texttt{let y = 2 in y} + \texttt{y} : \texttt{int}}\ (\text{let})$$

A derivation is just a mathy way of writing a natural language proof 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

$$
\dfrac{\dfrac{}{\{\} \vdash \texttt{2} : \texttt{int}}\;(\mathsf{intLit}) \qquad \dfrac{\dfrac{}{\{\texttt{y} : \texttt{int}\} \vdash \texttt{y} : \texttt{int}}\;(\mathsf{var}) \qquad \dfrac{}{\{\texttt{y} : \texttt{int}\} \vdash \texttt{y} : \texttt{int}}\;(\mathsf{var})}{\{\texttt{y} : \texttt{int}\} \vdash \texttt{y} + \texttt{y} : \texttt{int}}\;(\mathsf{intAdd})}{\{\} \vdash \texttt{let y = 2 in } \texttt{y} + \texttt{y} : \texttt{int}}\;(\mathsf{let})
$$

Derivations Encode Natural Language Arguments

$$\dfrac{\dfrac{}{\{\} \vdash \texttt{2} : \texttt{int}}\ (\text{intLit}) \qquad \dfrac{\dfrac{}{\{\texttt{y} : \texttt{int}\} \vdash \texttt{y} : \texttt{int}}\ (\text{var}) \qquad \dfrac{}{\{\texttt{y} : \texttt{int}\} \vdash \texttt{y} : \texttt{int}}\ (\text{var})}{\{\texttt{y} : \texttt{int}\} \vdash \texttt{y} + \texttt{y} : \texttt{int}}\ (\text{intAdd})}{\{\} \vdash \texttt{let y = 2 in y} + \texttt{y} : \texttt{int}}\ (\text{let})$$

$$\dfrac{\Gamma \vdash e_1 : \tau_1 \qquad \Gamma, x : \tau_1 \vdash e_2 : \tau_2}{\Gamma \vdash \texttt{let}\ x\ \texttt{=}\ e_1\ \texttt{in}\ e_2 : \tau_2}\ (\text{let})$$

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

Derivations Encode Natural Language Arguments

$$\dfrac{\dfrac{}{\{\} \vdash \texttt{2} : \texttt{int}}\,(\text{intLit}) \qquad \dfrac{\dfrac{}{\{\texttt{y} : \texttt{int}\} \vdash \texttt{y} : \texttt{int}}\,(\text{var}) \qquad \dfrac{}{\{\texttt{y} : \texttt{int}\} \vdash \texttt{y} : \texttt{int}}\,(\text{var})}{\{\texttt{y} : \texttt{int}\} \vdash \texttt{y} + \texttt{y} : \texttt{int}}\,(\text{intAdd})}{\{\} \vdash \texttt{let y = 2 in } \texttt{y} + \texttt{y} : \texttt{int}}\,(\text{let})$$

$$\dfrac{\Gamma \vdash e_1 : \tau_1 \qquad \Gamma, x : \tau_1 \vdash e_2 : \tau_2}{\Gamma \vdash \texttt{let } x \texttt{ = } e_1 \texttt{ in } e_2 : \tau_2}\,(\text{let})$$

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

$$\dfrac{\dfrac{}{\{\}\vdash \texttt{2}:\texttt{int}}\,(\text{intLit}) \qquad \dfrac{\dfrac{}{\{\texttt{y}:\texttt{int}\}\vdash \texttt{y}:\texttt{int}}\,(\text{var}) \qquad \dfrac{}{\{\texttt{y}:\texttt{int}\}\vdash \texttt{y}:\texttt{int}}\,(\text{var})}{\{\texttt{y}:\texttt{int}\}\vdash \texttt{y}+\texttt{y}:\texttt{int}}\,(\text{intAdd})}{\{\}\vdash \texttt{let y = 2 in y}+\texttt{y}:\texttt{int}}\,(\text{let})$$

$$\dfrac{\Gamma\vdash e_1:\tau_1 \qquad \Gamma, x:\tau_1\vdash e_2:\tau_2}{\Gamma\vdash \texttt{let}\ x\ \texttt{=}\ e_1\ \texttt{in}\ e_2:\tau_2}\,(\text{let})$$

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

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

$$\dfrac{\dfrac{}{\{\} \vdash \mathtt{2} : \mathtt{int}}\ (\text{intLit}) \qquad \dfrac{\dfrac{}{\{\mathtt{y} : \mathtt{int}\} \vdash \mathtt{y} : \mathtt{int}}\ (\text{var}) \qquad \dfrac{}{\{\mathtt{y} : \mathtt{int}\} \vdash \mathtt{y} : \mathtt{int}}\ (\text{var})}{\{\mathtt{y} : \mathtt{int}\} \vdash \mathtt{y} + \mathtt{y} : \mathtt{int}}\ (\text{intAdd})}{\{\} \vdash \mathtt{let}\ \mathtt{y}\ \mathtt{=}\ \mathtt{2}\ \mathtt{in}\ \mathtt{y} + \mathtt{y} : \mathtt{int}}\ (\text{let})$$

$$\dfrac{\Gamma \vdash e_1 : \tau_1 \qquad \Gamma, x : \tau_1 \vdash e_2 : \tau_2}{\Gamma \vdash \mathtt{let}\ x\ \mathtt{=}\ e_1\ \mathtt{in}\ e_2 : \tau_2}\ (\text{let})$$

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

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*

$$\dfrac{\Gamma \vdash e_1 : \mathtt{int} \qquad \Gamma \vdash e_2 : \mathtt{int}}{\Gamma \vdash e_1 + e_2 : \mathtt{int}}\ (\text{addInt})$$

Derivations Encode Natural Language Arguments

$$\dfrac{\dfrac{}{\{\} \vdash \texttt{2} : \texttt{int}}\,(\text{intLit}) \qquad \dfrac{\dfrac{}{\{\texttt{y} : \texttt{int}\} \vdash \texttt{y} : \texttt{int}}\,(\text{var}) \qquad \dfrac{}{\{\texttt{y} : \texttt{int}\} \vdash \texttt{y} : \texttt{int}}\,(\text{var})}{\{\texttt{y} : \texttt{int}\} \vdash \texttt{y} + \texttt{y} : \texttt{int}}\,(\text{intAdd})}{\{\} \vdash \texttt{let y = 2 in } \texttt{y} + \texttt{y} : \texttt{int}}\,(\text{let})$$

$$\dfrac{\Gamma \vdash e_1 : \tau_1 \qquad \Gamma, x : \tau_1 \vdash e_2 : \tau_2}{\Gamma \vdash \texttt{let } x \texttt{ = } e_1 \texttt{ in } e_2 : \tau_2}\,(\text{let})$$

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

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*

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

$$\dfrac{\Gamma \vdash e_1 : \texttt{int} \qquad \Gamma \vdash e_2 : \texttt{int}}{\Gamma \vdash e_1 + e_2 : \texttt{int}}\,(\text{addInt})$$

Derivations Encode Natural Language Arguments

$$\dfrac{\dfrac{}{\{\} \vdash \texttt{2} : \texttt{int}}\,(\text{intLit}) \qquad \dfrac{\dfrac{}{\{\texttt{y} : \texttt{int}\} \vdash \texttt{y} : \texttt{int}}\,(\text{var}) \qquad \dfrac{}{\{\texttt{y} : \texttt{int}\} \vdash \texttt{y} : \texttt{int}}\,(\text{var})}{\{\texttt{y} : \texttt{int}\} \vdash \texttt{y} + \texttt{y} : \texttt{int}}\,(\text{intAdd})}{\{\} \vdash \texttt{let y = 2 in y} + \texttt{y} : \texttt{int}}\,(\text{let})$$

$$\dfrac{\Gamma \vdash e_1 : \tau_1 \qquad \Gamma, x : \tau_1 \vdash e_2 : \tau_2}{\Gamma \vdash \texttt{let}\ x\ \texttt{=}\ e_1\ \texttt{in}\ e_2 : \tau_2}\,(\text{let})$$

$$\dfrac{\mathbf{n}\ \text{is an integer literal}}{\Gamma \vdash \mathbf{n} : \texttt{int}}\,(\text{intLit})$$

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*

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

$$\dfrac{\Gamma \vdash e_1 : \texttt{int} \qquad \Gamma \vdash e_2 : \texttt{int}}{\Gamma \vdash e_1 + e_2 : \texttt{int}}\,(\text{addInt})$$

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

Derivations Encode Natural Language Arguments

$$\dfrac{\dfrac{}{\{\} \vdash \texttt{2} : \texttt{int}}\ (\text{intLit}) \qquad \dfrac{\dfrac{}{\{\texttt{y} : \texttt{int}\} \vdash \texttt{y} : \texttt{int}}\ (\text{var}) \qquad \dfrac{}{\{\texttt{y} : \texttt{int}\} \vdash \texttt{y} : \texttt{int}}\ (\text{var})}{\{\texttt{y} : \texttt{int}\} \vdash \texttt{y} + \texttt{y} : \texttt{int}}\ (\text{intAdd})}{\{\} \vdash \texttt{let y = 2 in y} + \texttt{y} : \texttt{int}}\ (\text{let})$$

$$\dfrac{\Gamma \vdash e_1 : \tau_1 \qquad \Gamma, x : \tau_1 \vdash e_2 : \tau_2}{\Gamma \vdash \texttt{let}\ x\ \texttt{=}\ e_1\ \texttt{in}\ e_2 : \tau_2}\ (\text{let})$$

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

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

$$\dfrac{\Gamma \vdash e_1 : \texttt{int} \qquad \Gamma \vdash e_2 : \texttt{int}}{\Gamma \vdash e_1 \texttt{+} e_2 : \texttt{int}}\ (\text{addInt})$$

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

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

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

Derivations Encode Natural Language Arguments

$$\dfrac{\dfrac{}{\{\} \vdash \texttt{2} : \texttt{int}}\,(\text{intLit}) \qquad \dfrac{\dfrac{}{\{\texttt{y} : \texttt{int}\} \vdash \texttt{y} : \texttt{int}}\,(\text{var}) \qquad \dfrac{}{\{\texttt{y} : \texttt{int}\} \vdash \texttt{y} : \texttt{int}}\,(\text{var})}{\{\texttt{y} : \texttt{int}\} \vdash \texttt{y} + \texttt{y} : \texttt{int}}\,(\text{intAdd})}{\{\} \vdash \texttt{let y = 2 in } \texttt{y} + \texttt{y} : \texttt{int}}\,(\text{let})$$

$$\dfrac{\Gamma \vdash e_1 : \tau_1 \qquad \Gamma, x : \tau_1 \vdash e_2 : \tau_2}{\Gamma \vdash \texttt{let } x \texttt{ = } e_1 \texttt{ in } e_2 : \tau_2}\,(\text{let})$$

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

$$\dfrac{\Gamma \vdash e_1 : \texttt{int} \qquad \Gamma \vdash e_2 : \texttt{int}}{\Gamma \vdash e_1 \texttt{ + } e_2 : \texttt{int}}\,(\text{addInt})$$

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

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)
- and so is **y** (by assumption)

Derivations Encode Natural Language Arguments

$$\dfrac{\dfrac{}{\{\}\vdash 2:\texttt{int}}\,(\text{intLit}) \qquad \dfrac{\dfrac{}{\{\texttt{y}:\texttt{int}\}\vdash \texttt{y}:\texttt{int}}\,(\text{var}) \qquad \dfrac{}{\{\texttt{y}:\texttt{int}\}\vdash \texttt{y}:\texttt{int}}\,(\text{var})}{\{\texttt{y}:\texttt{int}\}\vdash \texttt{y}+\texttt{y}:\texttt{int}}\,(\text{intAdd})}{\{\}\vdash \texttt{let y = 2 in y}+\texttt{y}:\texttt{int}}\,(\text{let})$$

$$\dfrac{\Gamma\vdash e_1:\tau_1 \qquad \Gamma, x:\tau_1\vdash e_2:\tau_2}{\Gamma\vdash \texttt{let}\ x\ \texttt{=}\ e_1\ \texttt{in}\ e_2:\tau_2}\,(\text{let})$$

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

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*

- **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**

$$\dfrac{\Gamma\vdash e_1:\texttt{int} \qquad \Gamma\vdash e_2:\texttt{int}}{\Gamma\vdash e_1 + e_2:\texttt{int}}\,(\text{addInt})$$

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

Derivations Encode Natural Language Arguments

$$\dfrac{\dfrac{}{\{\} \vdash 2 : \texttt{int}}\ (\text{intLit}) \qquad \dfrac{\dfrac{}{\{\texttt{y} : \texttt{int}\} \vdash \texttt{y} : \texttt{int}}\ (\text{var}) \qquad \dfrac{}{\{\texttt{y} : \texttt{int}\} \vdash \texttt{y} : \texttt{int}}\ (\text{var})}{\{\texttt{y} : \texttt{int}\} \vdash \texttt{y} + \texttt{y} : \texttt{int}}\ (\text{intAdd})}{\{\} \vdash \texttt{let y = 2 in y + y} : \texttt{int}}\ (\text{let})$$

$$\dfrac{\Gamma \vdash e_1 : \tau_1 \qquad \Gamma, x : \tau_1 \vdash e_2 : \tau_2}{\Gamma \vdash \texttt{let}\ x\ \texttt{=}\ e_1\ \texttt{in}\ e_2 : \tau_2}\ (\text{let})$$

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

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*

- `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`

$$\dfrac{\Gamma \vdash e_1 : \texttt{int} \qquad \Gamma \vdash e_2 : \texttt{int}}{\Gamma \vdash e_1 \texttt{ + } e_2 : \texttt{int}}\ (\text{addInt})$$

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

Derivations Encode Natural Language Arguments

$$\dfrac{\dfrac{}{\{\} \vdash \texttt{2} : \texttt{int}}\,(\text{intLit}) \qquad \dfrac{\dfrac{}{\{\texttt{y} : \texttt{int}\} \vdash \texttt{y} : \texttt{int}}\,(\text{var}) \qquad \dfrac{}{\{\texttt{y} : \texttt{int}\} \vdash \texttt{y} : \texttt{int}}\,(\text{var})}{\{\texttt{y} : \texttt{int}\} \vdash \texttt{y} + \texttt{y} : \texttt{int}}\,(\text{intAdd})}{\{\} \vdash \texttt{let y = 2 in } \texttt{y} + \texttt{y} : \texttt{int}}\,(\text{let})$$

$$\dfrac{\Gamma \vdash e_1 : \tau_1 \qquad \Gamma, x : \tau_1 \vdash e_2 : \tau_2}{\Gamma \vdash \texttt{let } x \texttt{ = } e_1 \texttt{ in } e_2 : \tau_2}\,(\text{let})$$

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

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*

- `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`

$$\dfrac{\Gamma \vdash e_1 : \texttt{int} \qquad \Gamma \vdash e_2 : \texttt{int}}{\Gamma \vdash e_1 \texttt{ + } e_2 : \texttt{int}}\,(\text{addInt})$$

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

And all this works for semantics judgements as well

Recall: Semantic Judgements

$$\overset{\text{expression}}{e} \Downarrow \overset{\text{value}}{v}$$

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

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

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

$$\dfrac{\dfrac{}{\texttt{true} \Downarrow \top}\,(\mathsf{trueEval}) \qquad \dfrac{}{\texttt{2} \Downarrow 2}\,(\mathsf{intEval})}{\texttt{if true then 2 else 3} \Downarrow 2}\,(\mathsf{ifEval})$$

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)

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

Example (Evaluation)

`if true then 2 else 5` $\Downarrow$ `2`

Example (Typing)

$\{\} \vdash$ `2 + 3 <> 4 : bool`

Example (Evaluation)

`2 + 3 <> 4` $\Downarrow$ `true`

Example (Evaluation)

$\texttt{let x = 2 in x + x} \Downarrow \texttt{4}$

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*