Derivations

Concepts of Programming Languages
Lecture 5

CAS CS 320

Practice Problem

What is the typing rule for let expression?

What are the semantics rules for if expression?

$$\frac{\Gamma \vdash e_1 : \tau_1 \qquad \overset{\{\},\, y:\mathrm{int}}{\boxed{\Gamma, x : \tau_1}} \vdash e_2 : \tau}{\{\}\ \Gamma \vdash \mathrm{let}\ x = e_1\ \mathrm{in}\ e_2 : \tau}\ (\mathrm{let})$$

$$\frac{e \Downarrow \bot \qquad e_2 \Downarrow v_2}{\mathrm{if}\ e\ \mathrm{then}\ e_1\ \mathrm{else}\ e_2 \Downarrow v_2}\ (\mathrm{IfF})$$

$$\frac{e \Downarrow T \qquad e_1 \Downarrow v_1}{\mathrm{if}\ e\ \mathrm{then}\ e_1\ \mathrm{else}\ e_2 \Downarrow v_1}\ (\mathrm{IfT})$$

Outline

» Discuss derivations in general

» See how to read and write derivations

» Go through a couple examples

Recap

Recall: Inference Rules

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

Recall: Inference Rules

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

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

Recall: Inference Rules

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

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

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

A <u>typing judgment</u> is a compact way of representing the statement:

e is of type τ in/under the context Γ

Recall: Contexts

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

Recall: Contexts

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

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

Recall: Contexts

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

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

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

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

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

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

High Level

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

Derivations 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

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

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

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

Derivations 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

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

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

Applying Rules

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

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

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

Applying Rules

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

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

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

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

Building from the Ground Up

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

1. Start from the bottom!

Building from the Ground Up

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

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

Building from the Ground Up

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

1. Start from the bottom!

2. Apply the rule based on the expression

Repeat for the premises

Axioms (When are we done?)

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

Axioms (When are we done?)

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

We 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?)

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

We 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

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

Steps for a Typing Derivation

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

1. Start from the bottom!

Steps for a Typing Derivation

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

1. Start from the bottom!

2. Apply the rule based on the expression

Steps for a Typing Derivation

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

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?

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

How To Know Which Rule To Apply?

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

Easy because of a very cool feature of OCaml:

How To Know Which Rule To Apply?

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

Easy because of a very cool feature of OCaml:

Syntax Directed Type System

How To Know Which Rule To Apply?

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

Easy because of a very cool feature of OCaml:

Syntax Directed Type System

Only one typing rule per expression.

Integer Literals

1.

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

2.

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

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"

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

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.

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

2.

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

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

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

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

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

1. `true` is of type `bool` in any context
2. `false` if of type `bool` in any context
3. `true` evaluates to the value ⊤
4. `false` evaluates to the value ⊥

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)

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

Back to the Example

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

Back to the Example

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

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

Back to the Example

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

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

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

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

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

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

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

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

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

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

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 expression)

Derivations Encode Natural Language Arguments

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

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

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

`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`*

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

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

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

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

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

Derivations Encode Natural Language Arguments

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

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

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

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)

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

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

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

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

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

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

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

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

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

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

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

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

Derivations Encode Natural Language Arguments

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

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

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

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

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

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

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

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

» 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`

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

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

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

» 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

$$\underset{\text{expression}}{\boxed{e}} \Downarrow \underset{\text{value}}{\boxed{v}}$$

A <u>semantic judgment</u> 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

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

If e_1 *evaluates to the (integer)* v_1 *and* e_2 *evaluates to the (integer)* v_2*, and* $\boxed{v_1 + v_2 = v}$*, then* $e_1 + e_2$ *evaluates to the (integer)* v

Semantic Derivations

$$\dfrac{\dfrac{}{\texttt{true} \Downarrow \bot}\,(\cancel{\text{trueLitEval}})\,(\text{falseLitEval}) \qquad \dfrac{}{3 \Downarrow 3}\,(\text{intLitEval})}{\texttt{if true then 2 else 3} \Downarrow 3}\,(\text{ifF})$$

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

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)

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

$$\frac{}{\Gamma \vdash true : bool}\ (trueLit) \qquad \frac{n \text{ is an int lit}}{\Gamma \vdash n : int}\ (intLit)$$

$$\frac{\dfrac{}{\{\} \vdash true : bool}\ (trueLit) \quad \dfrac{}{\{\} \vdash 2 : int}\ (intLit) \quad \dfrac{}{\{\} \vdash 5 : int}\ (intLit)}{\{\} \vdash \textbf{if true then 2 else 5} : int}\ (if)$$

Example (Evaluation)

$$\dfrac{\dfrac{}{\text{true} \Downarrow T}\;(\text{trueLitEval}) \qquad \dfrac{}{2 \Downarrow 2}\;(\text{intEval})}{\textbf{if true then 2 else 5} \Downarrow 2}\;(\text{if}\,T)$$

Example (Typing)

$$\frac{\Gamma \vdash e_1 : \tau \qquad \Gamma \vdash e_2 : \tau}{\Gamma \vdash e_1 <> e_2 : \text{bool}} (<>)$$

$$\frac{\Gamma \vdash e_1 : \text{int} \qquad \Gamma \vdash e_2 : \text{int}}{\Gamma \vdash e_1 + e_2 : \text{int}}$$

$$\dfrac{\dfrac{\dfrac{}{\{x : \text{int}\} \vdash 3 : \text{int}}\,(\text{intLit}) \qquad \dfrac{}{\{x : \text{int}\} \vdash x : \text{int}}\,\text{var}}{\{x : \text{int}\} \vdash 3 + x : \text{int}}\,\text{intAdd} \qquad \dfrac{}{\{x : \text{int}\} \vdash 4 : \text{int}}\,(\text{intLit})}{\{x : \text{int}\} \vdash 3 + x <> 4 : \text{bool}}\,(<>)$$

Example (Evaluation)

$$\frac{e_1 \Downarrow v_1 \qquad \boxed{[v_1/x]e_2} \Downarrow v}{\text{let } x = e_1 \text{ in } e_2 \Downarrow v}\ (\text{letEval})$$

$$\dfrac{\dfrac{}{2 \Downarrow 2}\ (\text{intLit Eval}) \qquad \dfrac{3 \Downarrow 3 \qquad 2 \Downarrow 2}{3 + 2 \Downarrow 5}\ (\text{intAdd Eval})}{\textbf{let } x = \underline{2} \textbf{ in } 3 + x \Downarrow 5}\ (\text{letEval})$$

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*