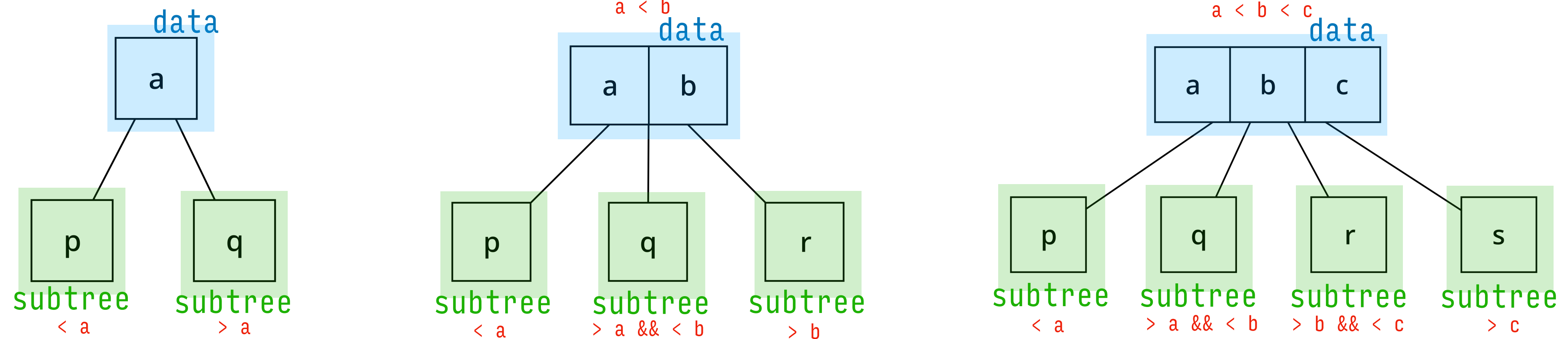


Derivations

Concepts of Programming Languages
Lecture 6

Practice Problem



A **2-3-4 tree** is a self-balancing tree structure with three possible kinds of nodes, shown above. Write an ADT to represent 2-3-4 trees

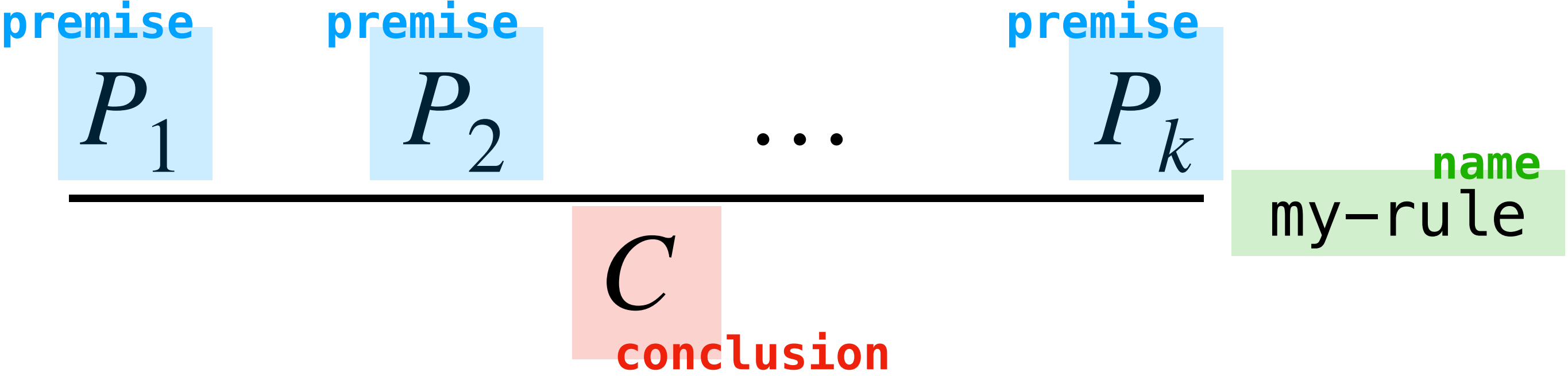
(If you have extra time, try implementing search for 2-3-4 trees)

Outline

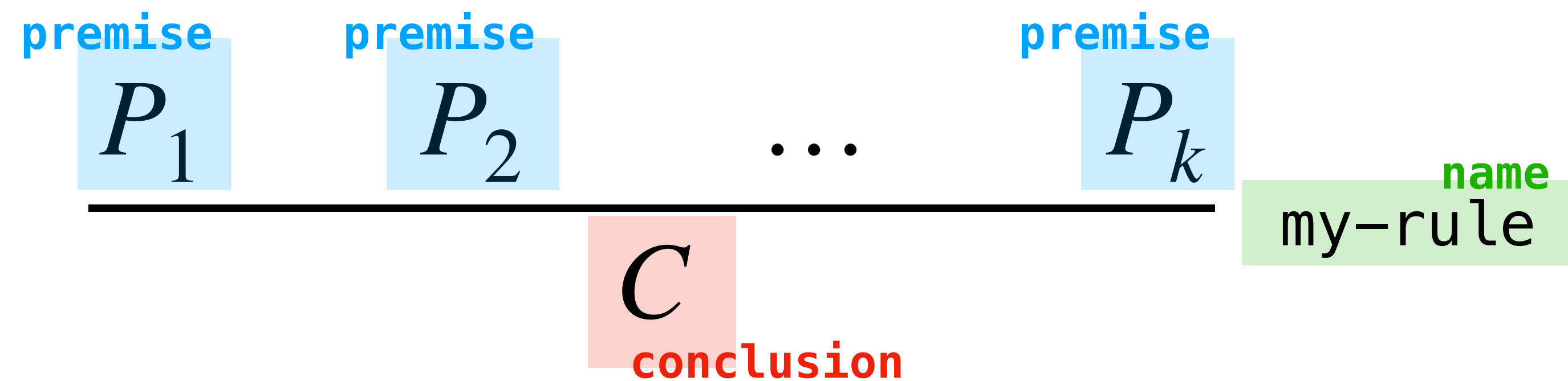
- » Discuss derivations in general
- » See how to read and write derivations
- » Go through a couple examples

Recap

Recall: Inference Rules

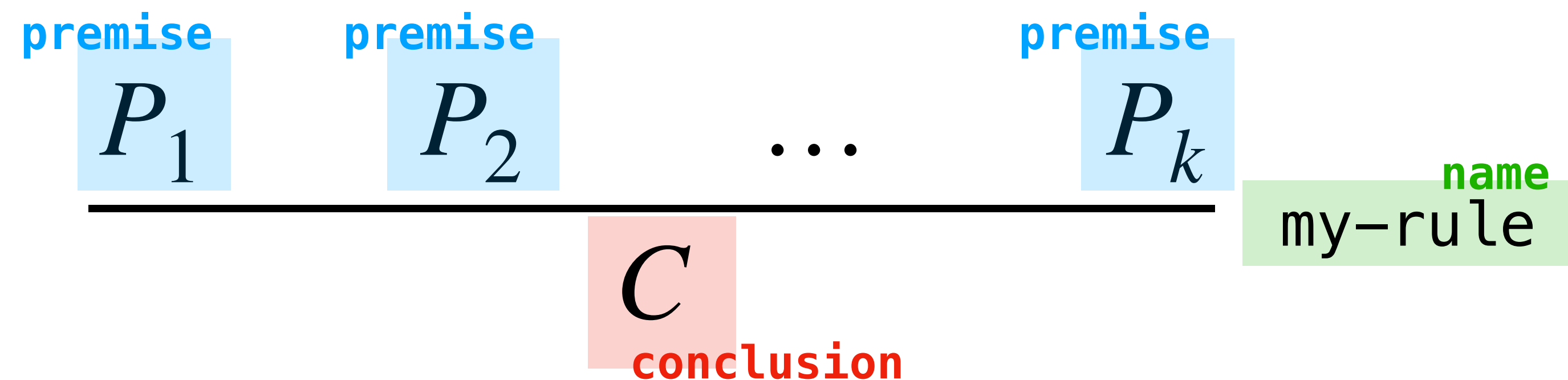


Recall: Inference Rules



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

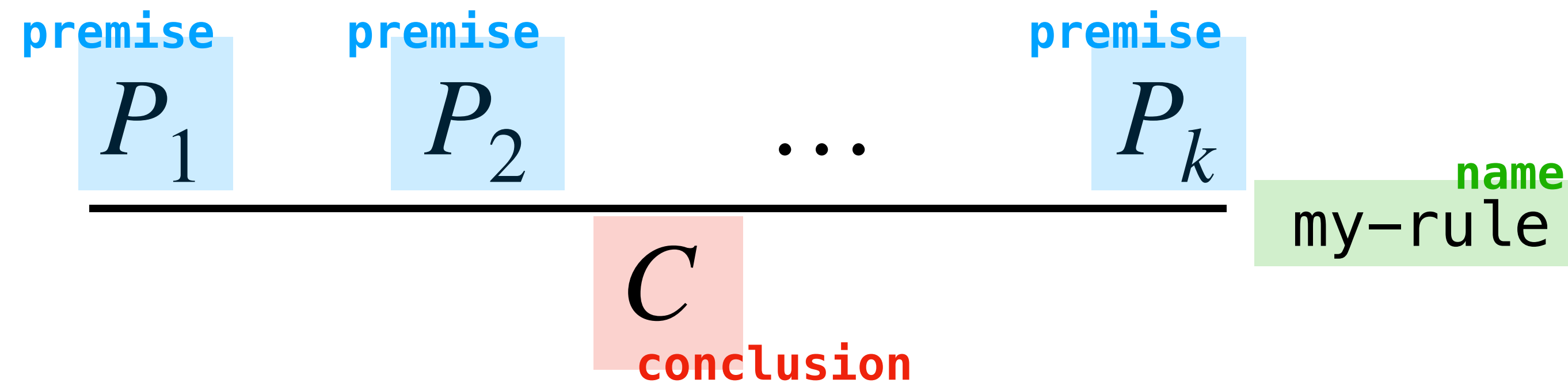
Recall: Inference Rules



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



We can read this as:

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

Typing Judgments

The diagram illustrates a typing judgment $\Gamma \vdash e : \tau$. The components are highlighted with colored boxes and labels: the context Γ is in a light blue box labeled "context" in blue; the expression e is in a light red box labeled "expression" in red; and the type τ is in a light green box labeled "type" in green. The symbols \vdash and $:$ are placed between the boxes.

A **typing judgment** is 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

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

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"

Recall: Reading Typing Judgements

$\{b : \text{bool}\} \vdash \text{if } b \text{ then } 2 \text{ else } 3 : \text{int}$

Recall: Reading Typing Judgements

$$\{b : \text{bool}\} \vdash \text{if } b \text{ then } 2 \text{ else } 3 : \text{int}$$

In English: *Given that b is a bool , the expression $\text{if } b \text{ then } 2 \text{ else } 3$ is an int*

Recall: Reading Typing Judgements

$$\{b : \text{bool}\} \vdash \text{if } b \text{ then } 2 \text{ else } 3 : \text{int}$$

In English: *Given that b is a bool , the expression $\text{if } b \text{ then } 2 \text{ else } 3$ is an int*

The context allows us to determine the type of an expression *relative to the types of variables*

Recall: Integer Addition Typing Rule

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

If e_1 is an *int* (in any context Γ) and e_2 is an *int* then (in any context Γ) $e_1 + e_2$ is an *int* (in any context Γ)

Recall: Judgements are Statements

`{b : bool} ⊢ if b then 2 else 3 : string`

Recall: Judgements are Statements

```
{b : bool} ⊢ if b then 2 else 3 : string
```

A judgement is a *statement* in the same way that "there are infinitely many twin primes" or "pigs fly" is a statement

Recall: Judgements are Statements

```
{b : bool} ⊢ if b then 2 else 3 : string
```

A judgement is a *statement* in the same way that "there are infinitely many twin primes" or "pigs fly" is a statement

We haven't *proved* anything by writing down a typing judgment

Recall: Judgements are Statements

$\{b : \text{bool}\} \vdash \text{if } b \text{ then } 2 \text{ else } 3 : \text{string}$

A judgement is a *statement* in the same way that "there are infinitely many twin primes" or "pigs fly" is a statement

We haven't *proved* anything by writing down a typing judgment

Today: We will talk about **typing derivations**, which are used to demonstrate that expressions *actually* have their expected types in our PL

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

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

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

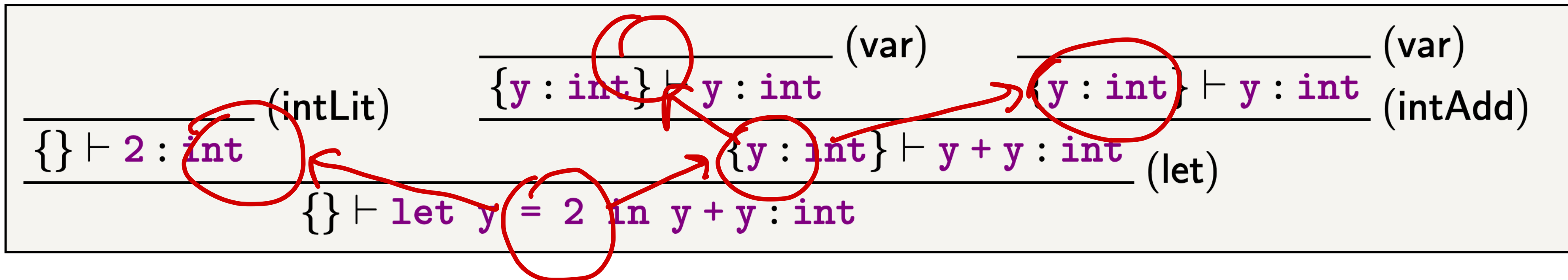
$$\frac{\frac{}{\{\} \vdash 2 : \text{int}} \text{(intLit)} \quad \frac{\frac{}{\{y : \text{int}\} \vdash y : \text{int}} \text{(var)} \quad \frac{\frac{}{\{y : \text{int}\} \vdash y : \text{int}} \text{(var)}}{\{y : \text{int}\} \vdash y + y : \text{int}} \text{(intAdd)}}{\{\} \vdash \text{let } y =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

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



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

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

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

Applying Rules

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

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

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

Applying Rules

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

$\Gamma \quad e_1 \quad \tau$ $\{x : \text{int}\} \vdash x + 1 : \text{int}$	$\Gamma \quad e_2 \quad \tau$ $\{x : \text{int}\} \vdash [] : \text{int list}$
$\{x : \text{int}\} \vdash (x + 1) :: [] : \text{int list}$	

$\Gamma \quad e_1 \quad e_2 \quad \tau$

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

$\frac{}{\{x : \text{int}\} \vdash x : \text{int}} \text{ (var)}$	$\frac{}{\{x : \text{int}\} \vdash 1 : \text{int}} \text{ (intLit)}$	
$\frac{}{\{x : \text{int}\} \vdash x + 1 : \text{int}} \text{ (intAdd)}$	$\frac{}{\{x : \text{int}\} \vdash [] : \text{int list}} \text{ (nil)}$	
$\frac{\frac{}{\{x : \text{int}\} \vdash x + 1 : \text{int}} \text{ (intAdd)} \quad \frac{}{\{x : \text{int}\} \vdash [] : \text{int list}} \text{ (nil)}}{\{x : \text{int}\} \vdash (x + 1) :: [] : \text{int list}} \text{ (cons)}$		

Building from the Ground Up

$\frac{\overbrace{\{x : \text{int}\} \vdash x : \text{int}}^{e_1} \text{ (var)}}{\{x : \text{int}\} \vdash x : \text{int}}$	$\frac{\overbrace{\{x : \text{int}\} \vdash 1 : \text{int}}^{e_2} \text{ (intLit)}}{\{x : \text{int}\} \vdash 1 : \text{int}}$	$\frac{}{\{x : \text{int}\} \vdash [] : \text{int list}} \text{ (nil)}$
$\frac{\{x : \text{int}\} \vdash x + 1 : \text{int}}{\{x : \text{int}\} \vdash (x + 1) :: [] : \text{int list}} \text{ (intAdd)}$	$\frac{\{x : \text{int}\} \vdash [] : \text{int list}}{\{x : \text{int}\} \vdash (x + 1) :: [] : \text{int list}} \text{ (cons)}$	

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

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

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?)

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

Axioms (When are we done?)

$$\begin{array}{c} \frac{}{\{x : \text{int}\} \vdash x : \text{int}} \text{(var)} \quad \frac{}{\{x : \text{int}\} \vdash 1 : \text{int}} \text{(intLit)} \\ \hline \frac{}{\{x : \text{int}\} \vdash x + 1 : \text{int}} \text{(intAdd)} \quad \frac{}{\{x : \text{int}\} \vdash [] : \text{int list}} \text{(nil)} \\ \hline \frac{}{\{x : \text{int}\} \vdash (x + 1) :: [] : \text{int list}} \text{(cons)} \end{array}$$

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?)

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

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

Integer Literals

~~$\frac{}{\vdash true : int}$~~

(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 "1 is an integer literal"

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

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

$$\begin{array}{ll} (1) & (2) \\ \frac{n \text{ is an float lit}}{\Gamma \vdash n : \text{float}} & \frac{n \text{ is an float lit}}{n \Downarrow n} \\ \text{(floatLit)} & \text{(floatLitEval)} \end{array}$$

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

$$\begin{array}{ll} (1) \quad \frac{}{\Gamma \vdash \text{true} : \text{bool}} \text{ (trueLit)} & (2) \quad \frac{}{\Gamma \vdash \text{false} : \text{bool}} \text{ (falseLit)} \\ (3) \quad \frac{}{\text{true} \Downarrow \top} \text{ (trueLitEval)} & (4) \quad \frac{}{\text{false} \Downarrow \perp} \text{ (falseLitEval)} \end{array}$$

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)

Back to the Example

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

We need $\{\} \vdash 2 : \text{int}$ 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

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

Derivations Encode Natural Language Arguments

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

A derivation is just a math-y way of writing a natural language prove that a typing derivation holds

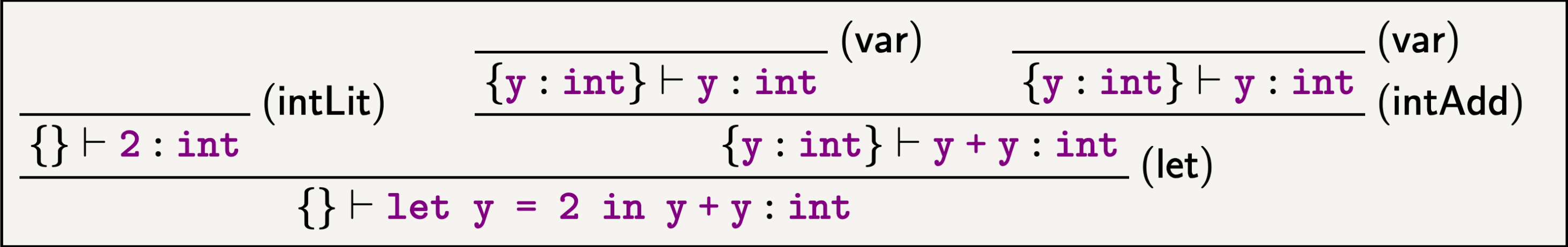
Derivations Encode Natural Language Arguments

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

A derivation is just a math-y 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



Derivations Encode Natural Language Arguments

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

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

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

$$\begin{array}{c}
 \frac{e_1 \quad \tau_1 \quad \checkmark}{\{\} \vdash 2 : \text{int}} \text{(intLit)} \quad \frac{}{\{y : \text{int}\} \vdash y : \text{int}} \text{(var)} \quad \frac{}{\{y : \text{int}\} \vdash y : \text{int}} \text{(var)} \\
 \frac{}{\{y : \text{int}\} \vdash y + y : \text{int}} \text{(intAdd)} \\
 \hline
 \{\} \vdash \text{let } y = 2 \text{ in } y + y : \text{int} \text{ (let)}
 \end{array}$$

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

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

The expression $\Gamma \quad x \quad e_1 \quad e_2 \quad \tau_2$ `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

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

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)

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

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

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

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

Derivations Encode Natural Language Arguments

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

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

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

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 expression)

>> 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

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

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

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

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)

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

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

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

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

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 expression)

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

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

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

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

Derivations Encode Natural Language Arguments

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

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

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

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 expression)

» 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 expression)

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

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

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

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

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 expression)

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

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

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

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

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**

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)

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

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

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

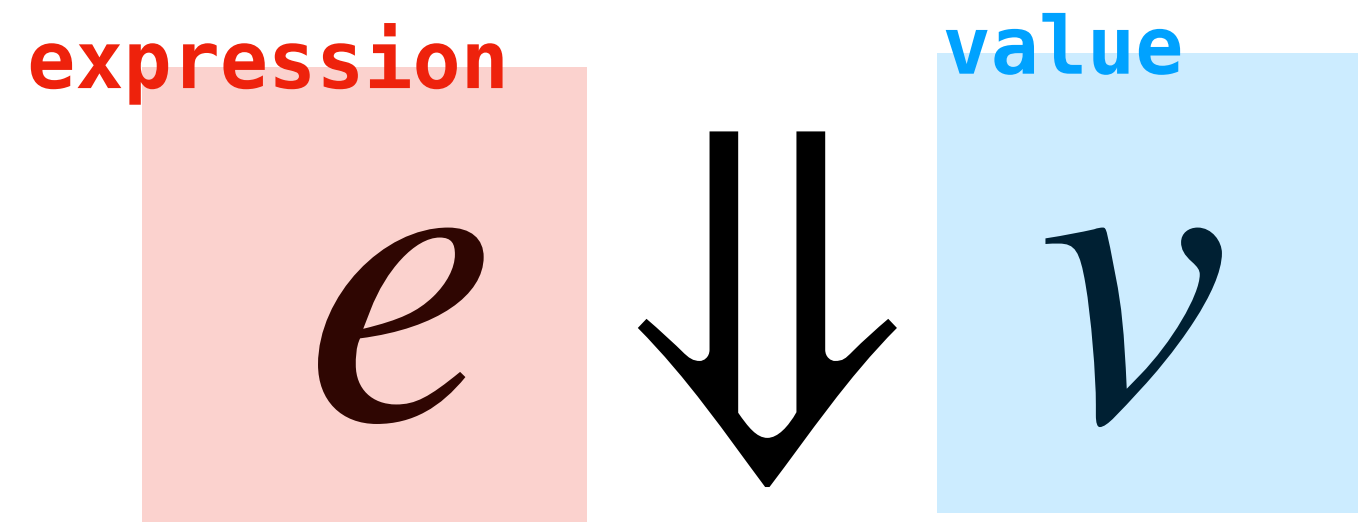
$$\frac{(x : \tau) \in \Gamma}{\Gamma \vdash x : \tau} \text{(var)}$$

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

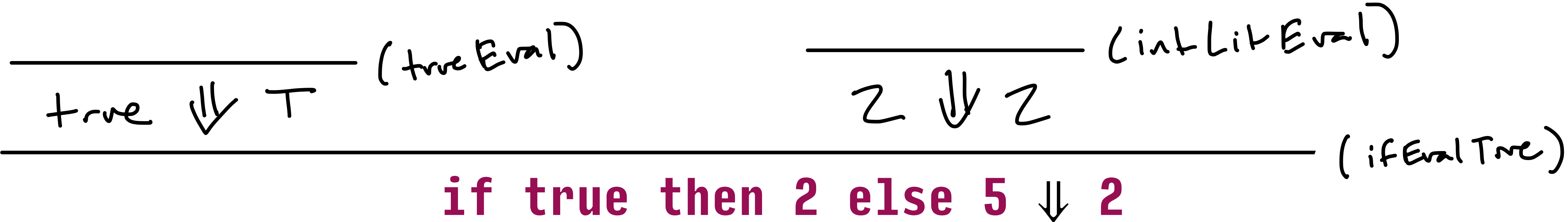
The principle is the same, except that the judgments are semantic judgments instead of typing judgments

Examples

Example (Typing)

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

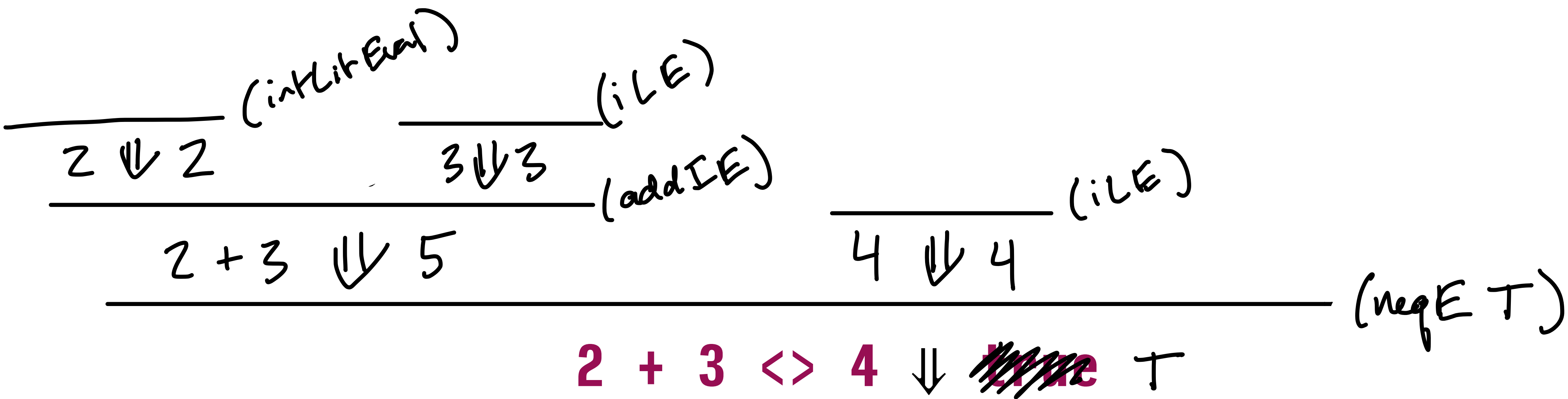
Example (Evaluation)



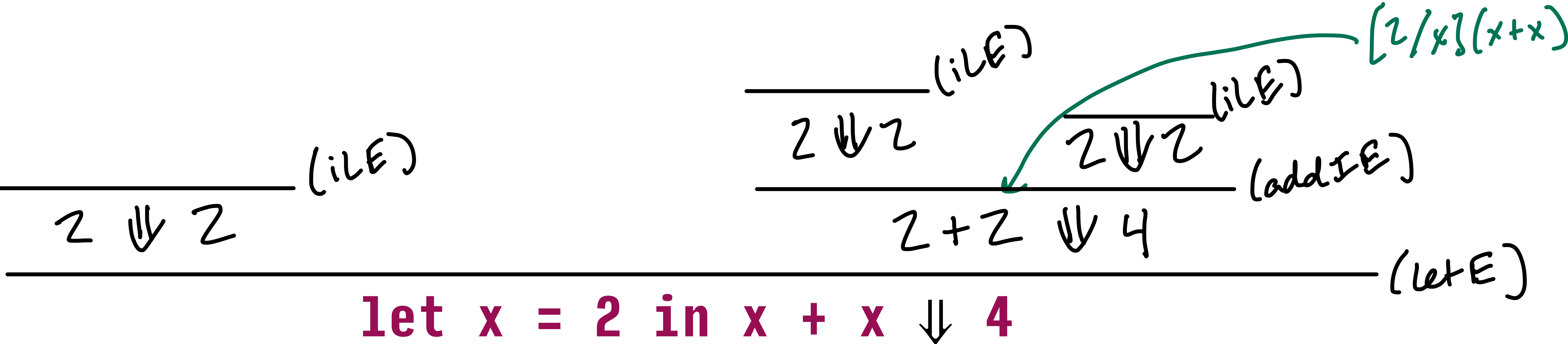
Example (Typing)

$$\frac{\frac{\frac{}{\{ \} \vdash 2 : \text{int}} \text{(intLit)}}{\{ \} \vdash 2 + 3 : \text{int}} \quad \frac{\frac{\frac{}{\{ \} \vdash 3 : \text{int}} \text{(intLit)}}{\{ \} \vdash 3 : \text{int}} \text{(addInt)}}{\{ \} \vdash 4 : \text{int}} \text{(intLit)}}{\{ \} \vdash (2 + 3) <> 4 : \text{bool}} \text{(Neg)}$$

Example (Evaluation)



Example (Evaluation)



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*