What are type rules?

 $\Gamma \vdash \mathbf{e}_1 : num \qquad \Gamma \vdash \mathbf{e}_2 : num$ $\Gamma \vdash \{+ \mathbf{e}_1 \ \mathbf{e}_2\} : num$

An example - the rule for +

What are type rules?

 $\Gamma \vdash \mathbf{e}_{1} : num \qquad \Gamma \vdash \mathbf{e}_{2} : num$ $\Gamma \vdash \{+ | \mathbf{e}_{1} | \mathbf{e}_{2}\} : num$

An example - the rule for +

- This is just one of a set of inference rules.
- Together the set of rules define the type judgment, which is a relation that assigns types to expressions.

What are type rules?

 $\Gamma \vdash \mathbf{e}_{1} : num \qquad \Gamma \vdash \mathbf{e}_{2} : num$ $\Gamma \vdash \{+ | \mathbf{e}_{1} | \mathbf{e}_{2}\} : num$

An example - the rule for +

- This is just one of a set of inference rules.
- Together the set of rules define the type judgment, which is a relation that assigns types to expressions.
- Fine, but what does that mean...

A B C

The general form of a inference rule

A B C

The general form of a inference rule

- A and B are *premises* (not necessarily two of them)
- C is the conclusion

A B C

The general form of a inference rule

- A and B are premises (not necessarily two of them)
- C is the conclusion
- This is a *rule*, which says:
- If I know A and B, then I can conclude C

A B A∧B

 A
 B

 A
 A

An example — logical "and"

• If I know A, and I know B, then I can conclude A \land B

 A
 B

 A
 A

- If I know A, and I know B, then I can conclude A \land B
- How would I know A and B?

 A
 B

 A
 Λ
 B

- If I know A, and I know B, then I can conclude A \land B
- How would I know A and B?
- I used some rule to conclude they were true

 A
 B

 A
 Λ
 B

- If I know A, and I know B, then I can conclude A \land B
- Some other Λ rules:

A ∧ B	A ∧ B
Α	В

 A
 B

 A
 Λ
 B

An example — logical "and"

• If I know A, and I know B, then I can conclude A \land B

A \lambda B	A	٨	B
A		В	

• Add some more rules and you have a system for deciding the truth of logical sentences

 $\Gamma \vdash \mathbf{e}_{1} : num \qquad \Gamma \vdash \mathbf{e}_{2} : num$ $\Gamma \vdash \{\mathbf{+} \mid \mathbf{e}_{1} \mid \mathbf{e}_{2}\} : num$

 $\Gamma \vdash \mathbf{e}_1 : num \qquad \Gamma \vdash \mathbf{e}_2 : num$ $\Gamma \vdash \{\mathbf{+} \mid \mathbf{e}_1 \mid \mathbf{e}_2\} : num$

One of a set of rules that define the type judgment

 $\Gamma \vdash \mathbf{e}_{1} : num \qquad \Gamma \vdash \mathbf{e}_{2} : num$ $\Gamma \vdash \{+ \mathbf{e}_{1} \ \mathbf{e}_{2}\} : num$

One of a set of rules that define the type judgment

- Which means...
- With the type bindings in $\Gamma,$ I can conclude that e has the type τ

 $\Gamma \vdash \mathbf{e}_{1} : num \qquad \Gamma \vdash \mathbf{e}_{2} : num$ $\Gamma \vdash \{+ \mathbf{e}_{1} \ \mathbf{e}_{2}\} : num$

One of a set of rules that define the type judgment

- Which means...
- With the type bindings in $\Gamma,$ I can conclude that e has the type τ
- Γ is the type environment and is just a map from <id> to τ (type)

TFAE Rules

$\Gamma \vdash \langle num \rangle : num [\langle id \rangle \leftarrow \tau] \vdash \langle id \rangle : \tau$
$\Gamma \vdash \texttt{true} : \texttt{bool}$ $\Gamma \vdash \texttt{false} : \texttt{bool}$
$\Gamma \vdash \mathbf{e}_1 : num \qquad \Gamma \vdash \mathbf{e}_2 : num$
$\Gamma \vdash \{+ e_1 e_2\} : num$
$\Gamma \vdash \mathbf{e}_1 : \textit{bool} \qquad \Gamma \vdash \mathbf{e}_2 : \tau_0 \qquad \Gamma \vdash \mathbf{e}_3 : \tau_0$
$\Gamma \vdash \{ if e_1 e_2 e_3 \} : \tau_0$
$\Gamma[\langle \mathbf{id} \rangle \leftarrow \tau_1] \vdash \mathbf{e} : \tau_0$
$\Gamma \vdash \{\texttt{fun } \{\texttt{} : \tau_1\} \ \texttt{e}\} \ : \ (\tau_1 \rightarrow \tau_0)$
$\Gamma \vdash \mathbf{e}_0 : (\tau_1 \rightarrow \tau_0) \qquad \Gamma \vdash \mathbf{e}_1 : \tau_1$
$\Gamma \vdash \{\mathbf{e}_0 \ \mathbf{e}_1\} : \tau_0$

Type derivations

1 : <i>num</i>	2 : <i>num</i>			
{ + 1 2}	: num	3	•	num
{+ {+	· 1 2} 3} :	num		

• We can conclude that an expression has some type if we can come up with a derivation using the type rules.

Type derivations

1 : <i>num</i>	2 : <i>num</i>			
{ + 1 2}	: num	3	:	num
{+ {+	· 1 2} 3} :	num		

- We can conclude that an expression has some type if we can come up with a derivation using the type rules.
- Great, but given some expression, how can we find the right derivation, and its type

Type derivations

1 : <i>num</i>	2 : <i>num</i>			
{ + 1 2}	: num	3	:	num
+ + +	- 1 2} 3} :	num		

- We can conclude that an expression has some type if we can come up with a derivation using the type rules.
- Great, but given some expression, how can we find the right derivation, and its type
- And what if it doesn't have a type...

 $\Gamma \vdash \mathbf{e} : \tau$

Let's try to find a type for this expression [] \vdash {if true {+ 1 2} 3} : τ ?

 $\Gamma \vdash \mathbf{e} : \tau$

Let's try to find a type for this expression

[] \vdash {if true {+ 1 2} 3} : τ ?

- What is the type of {if true {+ 1 2} 3}?
- Is there some τ that will satisfy the type judgment?

 $\Gamma \vdash \mathbf{e} : \tau$

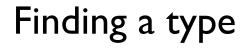
Let's try to find a type for this expression

[] \vdash {if true {+ 1 2} 3} : τ ?

Let's try a rule:

 $\Gamma \vdash \mathbf{e}_1 : num \qquad \Gamma \vdash \mathbf{e}_2 : num$

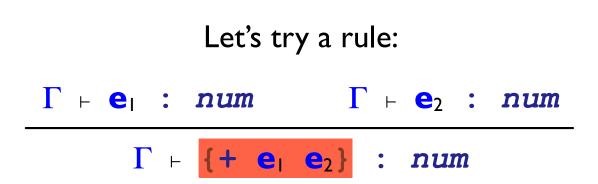
 $\Gamma \vdash \{+ e_1 e_2\} : num$



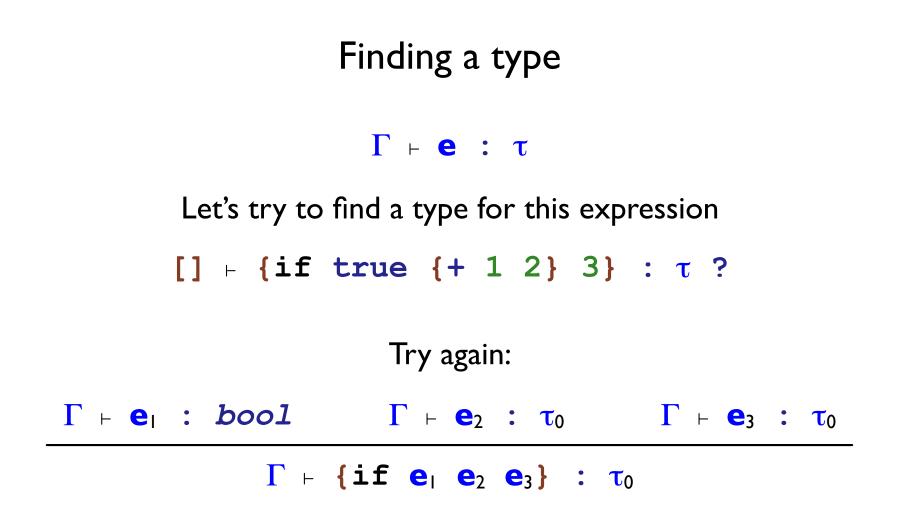
 $\Gamma \vdash \mathbf{e} : \tau$

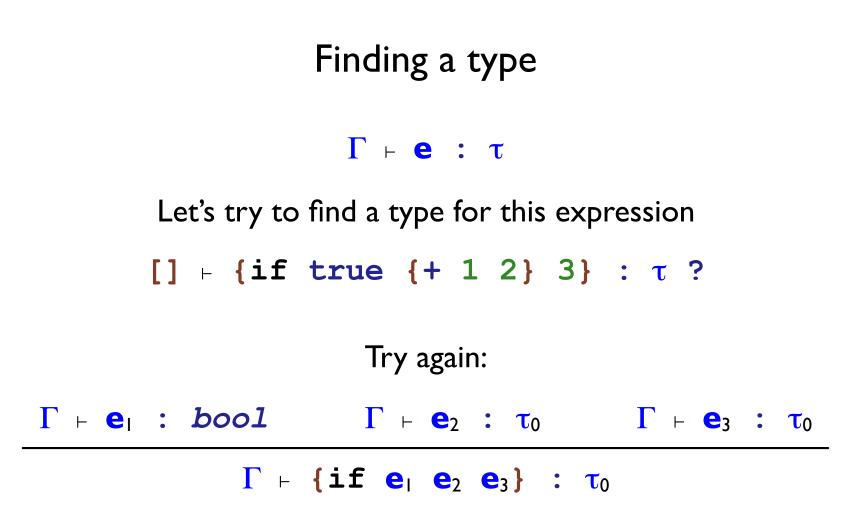
Let's try to find a type for this expression

[] \vdash {if true {+ 1 2} 3} : τ ?

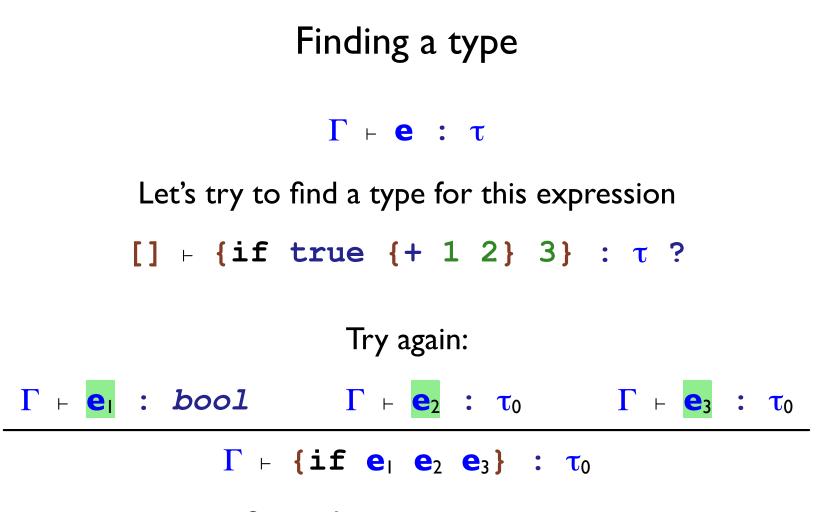


This one won't work (the expressions don't match)...





This works, but we still don't have a full derivation...



So we have to try again...

 $\Gamma \vdash \mathbf{e} : \tau$

Let's try to find a type for this expression

[] \vdash {if true {+ 1 2} 3} : τ ?

 $\Gamma \vdash \mathbf{e}_{1} : bool \qquad \Gamma \vdash \mathbf{e}_{2} : \tau_{0} \qquad \Gamma \vdash \mathbf{e}_{3} : \tau_{0}$ $\Gamma \vdash \{ \mathbf{if} \ \mathbf{e}_{1} \ \mathbf{e}_{2} \ \mathbf{e}_{3} \} : \tau_{0}$

 In general, we are stuck doing an expensive search where we try every rule for every expression (with backtracking).

 $\Gamma \vdash \mathbf{e} : \tau$

Let's try to find a type for this expression

[] \vdash {if true {+ 1 2} 3} : τ ?

 $\Gamma \vdash \mathbf{e}_{1} : bool \qquad \Gamma \vdash \mathbf{e}_{2} : \tau_{0} \qquad \Gamma \vdash \mathbf{e}_{3} : \tau_{0}$ $\Gamma \vdash \{ \texttt{if } \mathbf{e}_{1} \ \mathbf{e}_{2} \ \mathbf{e}_{3} \} : \tau_{0}$

- In general, we are stuck doing an expensive search where we try every rule for every expression (with backtracking).
- But actually the type rules have some nice properties, so things aren't really that difficult...

TFAE Rules

 $\Gamma \vdash \langle \text{num} \rangle$: num $[\dots \langle \text{id} \rangle \leftarrow \tau \dots] \vdash \langle \text{id} \rangle$: τ $\Gamma \vdash$ **true** : bool $\Gamma \vdash$ **false** : bool $\Gamma \vdash \mathbf{e}_1 : num \qquad \Gamma \vdash \mathbf{e}_2 : num$ $\Gamma \vdash \{+ \mathbf{e}_1 \mathbf{e}_2\} : num$ $\Gamma \vdash \mathbf{e}_1 : bool$ $\Gamma \vdash \mathbf{e}_2 : \tau_0$ $\Gamma \vdash \mathbf{e}_3 : \tau_0$ $\Gamma \vdash \{ \texttt{if } \mathbf{e}_1 \ \mathbf{e}_2 \ \mathbf{e}_3 \} : \tau_0$ Γ (<id> $\leftarrow \tau_1$) \vdash **e** : τ_0 $\Gamma \vdash \{ \texttt{fun } \{ \texttt{<id>} : \tau_1 \} \texttt{ e} \} : (\tau_1 \rightarrow \tau_0)$ $\Gamma \vdash \mathbf{e}_0 : (\tau_1 \rightarrow \tau_0) \qquad \Gamma \vdash \mathbf{e}_1 : \tau_1$ $\Gamma \vdash \{\mathbf{e}_0 \ \mathbf{e}_1\} : \tau_0$

$\Gamma \vdash \mathbf{e} : \tau$

• There is only one rule that applies to any TFAE expression

- There is only one rule that applies to any TFAE expression
- So there is only one (possible) type derivation for any expression

$\Gamma \vdash \mathbf{e} : \tau$

• For any rule, Γ and e always determine τ

- For any rule, Γ and e always determine τ
- Think of Γ and e as inputs they give us the necessary information for recursive calls (premises).
- Think of τ as an output the premises give us what we need to know to construct the result type

- For any rule, Γ and e always determine τ
- Think of Γ and e as inputs they give us the necessary information for recursive calls (premises).
- Think of τ as an output the premises give us what we need to know to construct the result type
- So we can easily turn the type judgment into a *function*:

```
; type-check \Gamma e \rightarrow \tau;
```