Part I

Quiz

Question #1: What is the value of the following expression?

```
{+ 1 2}
```

Wrong answer: **0**

Wrong answer: **42**

Answer: **3**

Quiz

Question #2: What is the value of the following expression?

```
{+ lambda 17 8}
```

Wrong answer: **error**

Answer: Trick question! `{+ lambda 17 8}` is not an expression

Language Grammar for Quiz

```
<Expr>  ::=  <Num>
           |  true
           |  false
           |  {+ <Expr> <Expr>}
           |  {* <Expr> <Expr>}
           |  {= <Expr> <Expr>}
           |  <Sym>
           |  {lambda {<Sym>*} <Expr>}
           |  {<Expr> <Expr>*}
           |  {if <Expr> <Expr> <Expr>}
```

Quiz

Question #3: Is the following an expression?

```
{{lambda {} 1} 7}
```

Wrong answer: **No**

Answer: **Yes** (according to our grammar)

Quiz

Question #4: What is the value of the following expression?

```
{{lambda {} 1} 7}
```

Answer: **1** (according to some interpreters)

But no real language would accept
`{{lambda {} 1} 7}`

Let's agree to call `{{lambda {} 1} 7}` an ***ill-formed expression***, because `{lambda {} 1}` should be used with only zero arguments

Let's agree to never evaluate ill-formed expressions

Quiz

Question #5: What is the value of the following expression?

```
{{lambda {} 1} 7}
```

Answer: **None** — the expression is ill-formed

Quiz

Question #6: Is the following a well-formed expression?

```
{+ {lambda {} 1} 8}
```

Answer: **Yes**

Quiz

Question #7: What is the value of the following expression?

```
{+ {lambda {} 1} 8}
```

Answer: **None** — it produces an error:

interp: not a number

Let's agree that a **`lambda`** expression cannot be inside a **+** form

Quiz

Question #8: Is the following a well-formed expression?

```
{+ {lambda {} 1} 8}
```

Answer: **No**

Quiz

Question #9: Is the following a well-formed expression?

```
{+ {{lambda {x} x} 7} 5}
```

Answer: Depends on what we meant by *inside* in our most recent agreement

- *Anywhere inside* — **No**
- *Immediately inside* — **Yes**

Since our intrepreter produces **12**, and since that result makes sense, let's agree on *immediately inside*

Quiz

Question #10: Is the following a well-formed expression?

```
{+ {{lambda {x} x} {lambda {y} y}} 5}
```

Answer: **Yes**, but we don't want it to be!

Quiz

Question #11: Is it possible to define ***well-formed*** (as a decidable property) so that we reject all expressions that produce errors?

Answer: **Yes**: reject *all* expressions!

Quiz

Question #12: Is it possible to define ***well-formed*** (as a decidable property) so that we reject *only* expressions that produce errors?

Answer: **No**

```
{+ 1 {if ... 1 {lambda {x} x}}}
```

If we always knew whether `...` produces true or false, we could solve the halting problem

Part 2

Types

We cannot reject *only* bad programs

In the process of rejecting expressions that are certainly bad, also reject some expressions that are good

```
{+ 1 {if {prime? 131101}
         1
         {lambda {x} x}}}
```

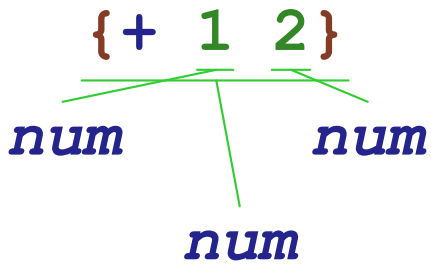
Overall strategy:

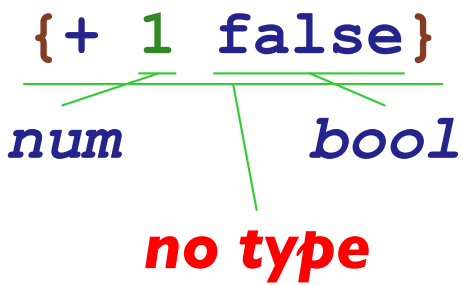
- Assign a ***type*** to each expression *without evaluating*
- Compute the type of a complex expression based on the types of its subexpressions

Types

```
1 : num
```

```
true : bool
```





Part 3

Type Rules

<Num> : *num*

true : *bool*

false : *bool*

$$\frac{\texttt{<Expr>}_1 : \textit{num} \qquad \texttt{<Expr>}_2 : \textit{num}}{\texttt{\{+ <Expr>}_1\ \texttt{<Expr>}_2\texttt{\}} : \textit{num}}$$

1 : *num*

true : *bool*

$$\frac{\texttt{1} : \textit{num} \qquad \texttt{2} : \textit{num}}{\texttt{\{+ 1 2\}} : \textit{num}}$$

$$\frac{\texttt{1} : \textit{num} \qquad \texttt{false} : \textit{bool}}{\texttt{\{+ 1 false\}} : \textit{no type}}$$

Type Rules

<Num> : *num*

true : *bool*

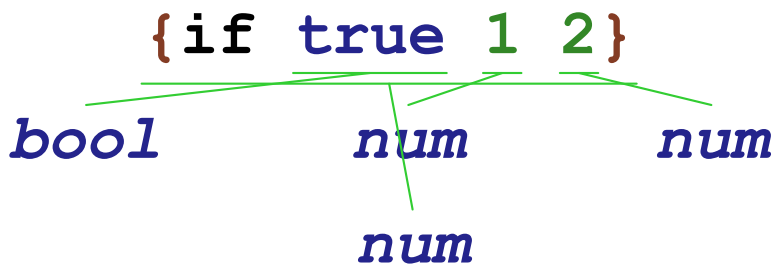
false : *bool*

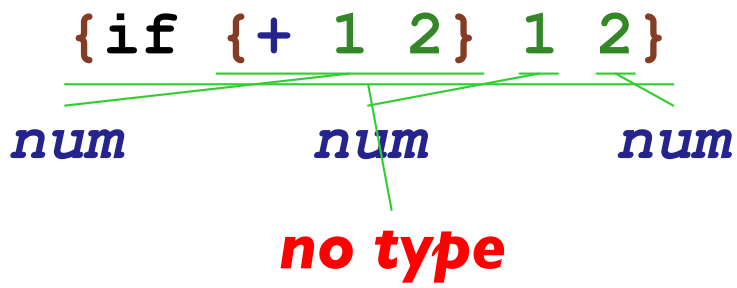
$$\frac{\texttt{<Expr>}_1 : \textit{num} \qquad \texttt{<Expr>}_2 : \textit{num}}{\texttt{\{+ <Expr>}_1\ \texttt{<Expr>}_2\texttt{\}} : \textit{num}}$$

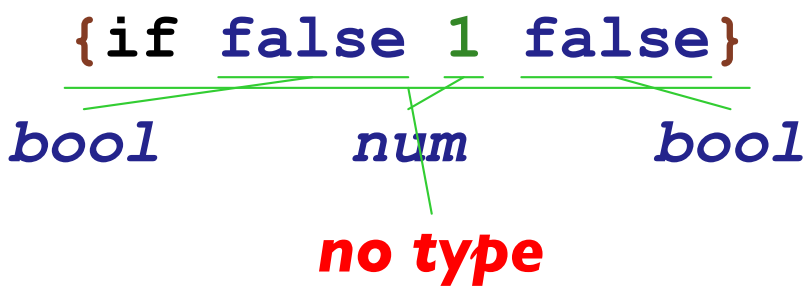
$$\frac{\dfrac{\texttt{1} : \textit{num} \qquad \texttt{2} : \textit{num}}{\texttt{\{+ 1 2\}} : \textit{num}} \qquad \texttt{3} : \textit{num}}{\texttt{\{+ \{+ 1 2\} 3\}} : \textit{num}}$$

Part 4

Types: Conditionals







Conditional Type Rules

$$\frac{\texttt{<Expr>}_1 \texttt{ : } \textit{bool} \qquad \texttt{<Expr>}_2 \texttt{ : <type>}_0 \qquad \texttt{<Expr>}_3 \texttt{ : <type>}_0}{\texttt{\{if <Expr>}_1 \texttt{ <Expr>}_2 \texttt{ <Expr>}_3\texttt{\}} \texttt{ : <type>}_0}$$

$$\frac{\texttt{true} \texttt{ : } \textit{bool} \qquad \texttt{1} \texttt{ : } \textit{num} \qquad \texttt{2} \texttt{ : } \textit{num}}{\texttt{\{if true 1 2\}} \texttt{ : } \textit{num}}$$

$$\frac{\texttt{\{+ 1 2\}} \texttt{ : } \textit{num} \qquad \texttt{1} \texttt{ : } \textit{num} \qquad \texttt{2} \texttt{ : } \textit{num}}{\texttt{\{if \{+ 1 2\} 1 2\}} \texttt{ : } \textit{no type}}$$

$$\frac{\texttt{false} \texttt{ : } \textit{bool} \qquad \texttt{1} \texttt{ : } \textit{num} \qquad \texttt{false} \texttt{ : } \textit{bool}}{\texttt{\{if false 1 false\}} \texttt{ : } \textit{no type}}$$

Part 5

Types: Variables and Functions

```
x : no type
```

```
{lambda {[x : bool]} x}
```

bool

(*bool* → *bool*)

```
{lambda {[x : bool]} {if x 1 2}}
```

bool *num* *num*

num

(*bool* → *num*)

Variable and Function Type Rules

$$[\ \ldots\ \texttt{<Sym>}\leftarrow\tau\ \ \ldots\ ]\ \vdash\ \texttt{<Sym>}\ :\ \tau$$

$$\frac{\Gamma[\ \texttt{<Sym>}\leftarrow\tau_1\ ]\ \vdash\ \texttt{e}\ :\ \tau_2}{\Gamma\ \vdash\ \{\texttt{lambda}\ \{[\texttt{<Sym>}\ :\ \tau_1]\}\ \texttt{e}\}\ :\ (\tau_1 \rightarrow \tau_2)}$$

Abbreviations: τ = **<Type>**
e = **<Expr>**
Γ = **<Env>**

Variable and Function Type Rules

$$[\ \ldots\ \texttt{<Sym>}\leftarrow\tau\ \ldots\ ] \vdash \texttt{<Sym>} : \tau$$

$$\frac{\Gamma[\texttt{<Sym>}\leftarrow\tau_1] \vdash \texttt{e} : \tau_2}{\Gamma \vdash \{\texttt{lambda}\ \{[\texttt{<Sym>} : \tau_1]\}\ \texttt{e}\} : (\tau_1 \rightarrow \tau_2)}$$

$$\emptyset \vdash \texttt{x} : \textbf{\textit{no type}}$$

$$\frac{[\texttt{x}\leftarrow\textit{bool}] \vdash \texttt{x} : \textit{bool}}{\emptyset \vdash \{\texttt{lambda}\ \{[\texttt{x} : \texttt{bool}]\}\ \texttt{x}\} : (\textit{bool} \rightarrow \textit{bool})}$$

$$\frac{\dfrac{[\texttt{x}\leftarrow\textit{bool}] \vdash \texttt{x} : \textit{bool} \quad [\texttt{x}\leftarrow\textit{bool}] \vdash \texttt{1} : \textit{num} \quad [\texttt{x}\leftarrow\textit{bool}] \vdash \texttt{2} : \textit{num}}{[\texttt{x}\leftarrow\textit{bool}] \vdash \{\texttt{if x 1 2}\} : \textit{num}}}{\emptyset \vdash \{\texttt{lambda}\ \{[\texttt{x} : \texttt{bool}]\}\ \{\texttt{if x 1 2}\}\} : (\textit{bool} \rightarrow \textit{num})}$$

Revised Rules

$$\Gamma \vdash \texttt{<Num>} : \textit{num}$$

$$\Gamma \vdash \texttt{true} : \textit{bool}$$

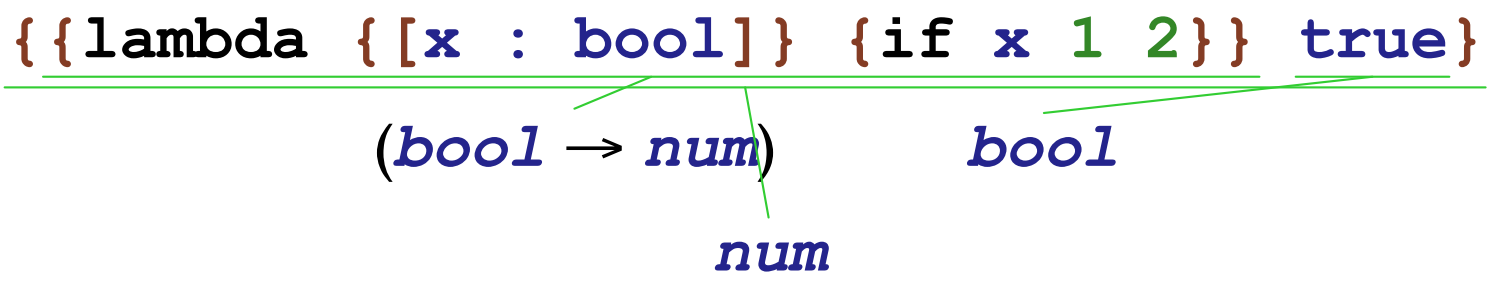
$$\Gamma \vdash \texttt{false} : \textit{bool}$$

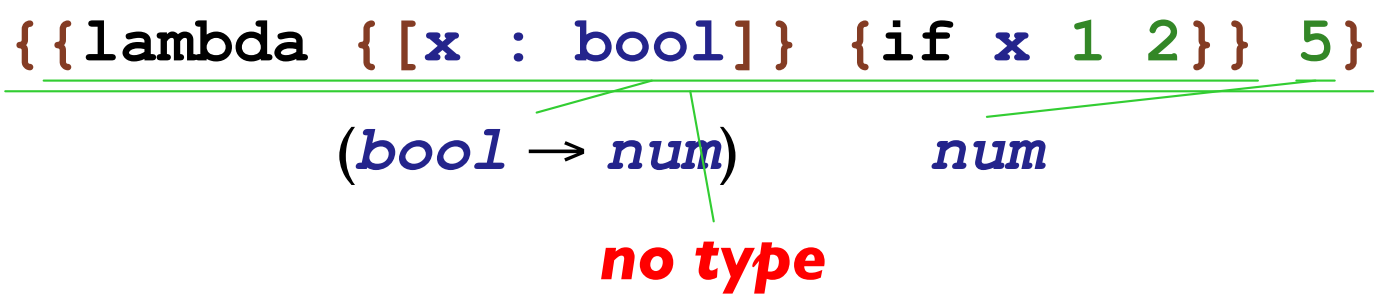
$$\frac{\Gamma \vdash \mathbf{e}_1 : \textit{num} \qquad \Gamma \vdash \mathbf{e}_2 : \textit{num}}{\Gamma \vdash \{\texttt{+}\ \mathbf{e}_1\ \mathbf{e}_2\} : \textit{num}}$$

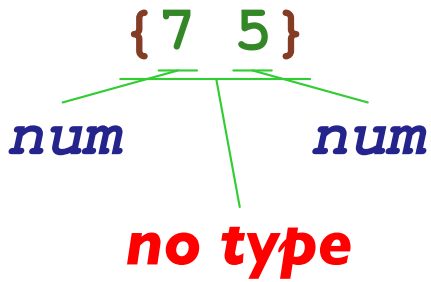
$$\frac{\Gamma \vdash \mathbf{e}_1 : \textit{bool} \qquad \Gamma \vdash \mathbf{e}_2 : \tau_0 \qquad \Gamma \vdash \mathbf{e}_3 : \tau_0}{\Gamma \vdash \{\texttt{if}\ \mathbf{e}_1\ \mathbf{e}_2\ \mathbf{e}_3\} : \tau_0}$$

Part 6

Types: Function Calls







Function Call Type Rule

$$\frac{\Gamma \vdash \mathbf{e}_1 : (\tau_2 \rightarrow \tau_3) \qquad \Gamma \vdash \mathbf{e}_2 : \tau_2}{\Gamma \vdash \{\mathbf{e}_1\ \mathbf{e}_2\} : \tau_3}$$

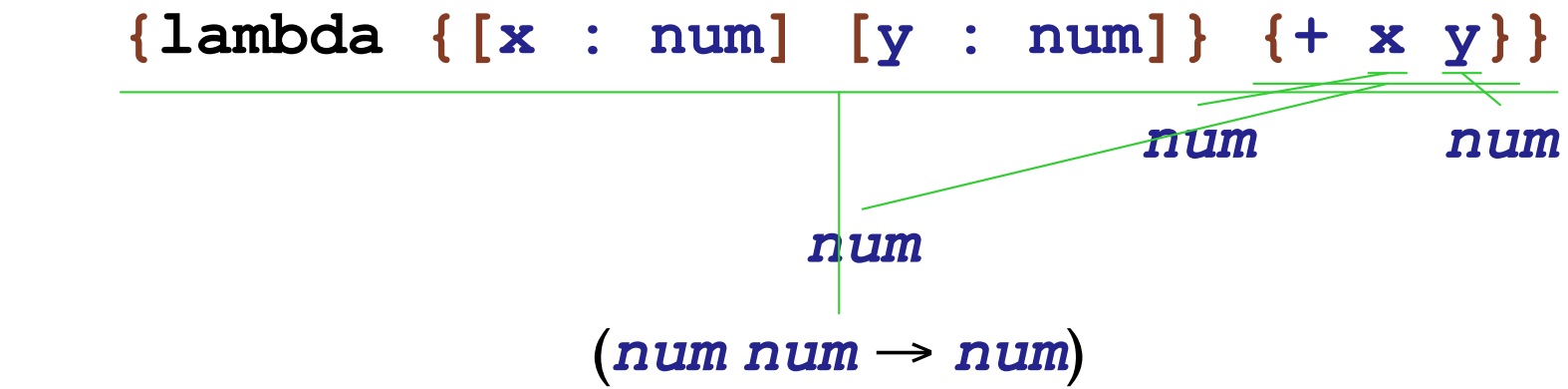
$$\frac{\varnothing \vdash \texttt{\{lambda \{[x : bool]\} \{if x 1 2\}\}} : (\textit{bool} \rightarrow \textit{num}) \qquad \varnothing \vdash \texttt{true} : \textit{bool}}{\varnothing \vdash \texttt{\{\{lambda \{[x : bool]\} \{if x 1 2\}\} true\}} : \textit{num}}$$

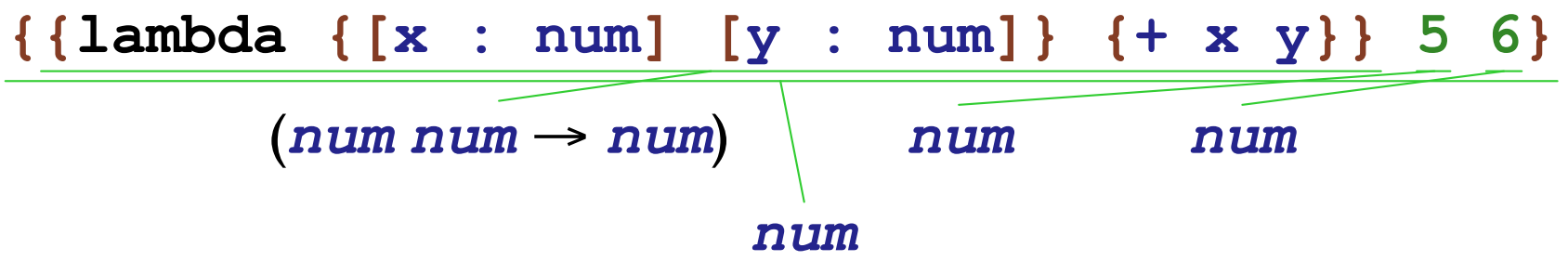
$$\frac{\varnothing \vdash \texttt{\{lambda \{[x : bool]\} \{if x 1 2\}\}} : (\textit{bool} \rightarrow \textit{num}) \qquad \varnothing \vdash \texttt{5} : \textit{num}}{\varnothing \vdash \texttt{\{\{lambda \{[x : bool]\} \{if x 1 2\}\} 5\}} : \textit{no type}}$$

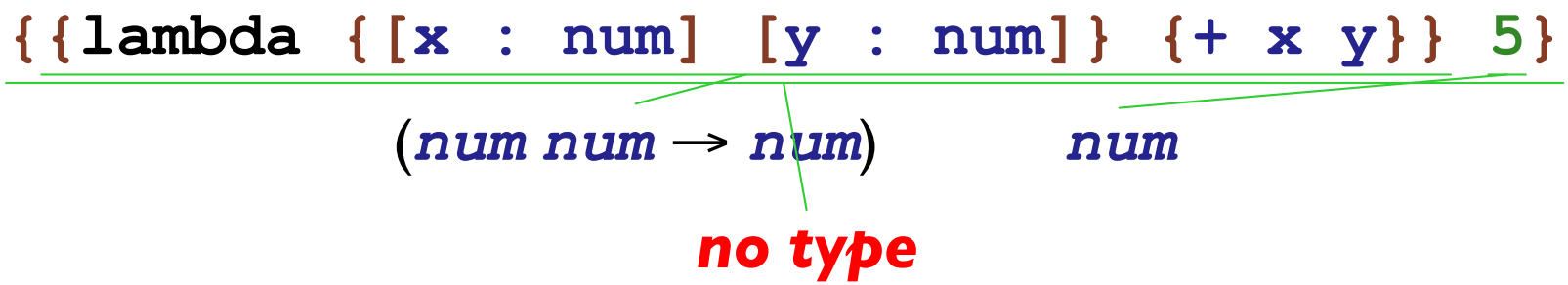
$$\frac{\varnothing \vdash \texttt{7} : \textit{num} \qquad \varnothing \vdash \texttt{5} : \textit{num}}{\varnothing \vdash \texttt{\{7 5\}} : \textit{no type}}$$

Part 7

Types: Multiple Arguments







Revised Function and Call Rules

$$\frac{\Gamma[\,\texttt{<Sym>}_1 \leftarrow \tau_1 \ldots \texttt{<Sym>}_n \leftarrow \tau_n\,] \vdash \mathbf{e} : \tau_0}{\Gamma \vdash \{\texttt{lambda}\ \{[\texttt{<Sym>}_1 : \tau_1] \ldots [\texttt{<Sym>}_n : \tau_n]\}\ \mathbf{e}\} : (\tau_1 \ldots \tau_n \rightarrow \tau_0)}$$

$$\frac{\Gamma \vdash \mathbf{e}_0 : (\tau_1 \ldots \tau_n \rightarrow \tau_0) \qquad \Gamma \vdash \mathbf{e}_1 : \tau_1 \qquad \ldots \qquad \Gamma \vdash \mathbf{e}_n : \tau_n}{\Gamma \vdash \{\mathbf{e}_0\ \mathbf{e}_1 \ \ldots\ \mathbf{e}_n\} : \tau_0}$$

Part 8

Typed Language

```
<Expr>  ::=  <Num>
          |  {+ <Expr> <Expr>}
          |  {* <Expr> <Expr>}
          |  <Sym>
          |  {lambda {[<Sym> : <Type>]} <Expr>}
          |  {<Expr> <Expr>}

<Type>  ::=  num
          |  bool
          |  (<Type> -> <Type>)
```

Expressions

```
(define-type ExprC
  [numC (n : number)]
  [idC (s : symbol)]
  [plusC (l : ExprC)
         (r : ExprC)]
  [multC (l : ExprC)
         (r : ExprC)]
  [lamC (n : symbol)
        (arg-type : Type)
        (body : ExprC)]
  [appC (fun : ExprC)
        (arg : ExprC)])
```

Types and Type Bindings

```
(define-type Type
  [numT]
  [boolT]
  [arrowT (arg : Type)
          (result : Type)])

(define-type TypeBinding
  [tbind (name : symbol)
         (type : Type)])

(define-type-alias TypeEnv (listof TypeBinding))
```

TFAE Type Checker

```
(define typecheck : (ExprC TypeEnv -> Type)
  (lambda (a tenv)
    (type-case ExprC a
      ...
      ...
      ...)))
```

TFAE Type Checker

```
(define typecheck : (ExprC TypeEnv -> Type)
  (lambda (a tenv)
    (type-case ExprC a
      ...
      [numC (n) ...]
      ...)))
```

$\Gamma \vdash$ **<Num>** : ***num***

TFAE Type Checker

```
(define typecheck : (ExprC TypeEnv -> Type)
  (lambda (a tenv)
    (type-case ExprC a
      ...
      [numC (n) (numT)]
      ...)))
```

$\Gamma \vdash$ **<Num>** : ***num***

TFAE Type Checker

```
(define typecheck : (ExprC TypeEnv -> Type)
  (lambda (a tenv)
    (type-case ExprC a
      ...
      [plusC (l r)
             ...]
      ...)))
```

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

TFAE Type Checker

```
(define typecheck : (ExprC TypeEnv -> Type)
  (lambda (a tenv)
    (type-case ExprC a
      ...
      [plusC (l r)
             ... (typecheck l tenv) ...
             ... (typecheck r tenv) ...]
      ...)))
```

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

TFAE Type Checker

```
(define typecheck : (ExprC TypeEnv -> Type)
  (lambda (a tenv)
    (type-case ExprC a
      ...
      [plusC (l r)
             (type-case Type (typecheck l tenv)
               [numT ()
                     ... (typecheck r tenv) ...]
               [else (type-error l "num")])]
      ...)))
```

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

TFAE Type Checker

```
(define typecheck : (ExprC TypeEnv -> Type)
  (lambda (a tenv)
    (type-case ExprC a
      ...
      [plusC (l r)
             (type-case Type (typecheck l tenv)
               [numT ()
                     (type-case Type (typecheck r tenv)
                       [numT () (numT)]
                       [else (type-error r "num")])]
               [else (type-error l "num")])]
      ...)))
```

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

TFAE Type Checker

```
(define typecheck : (ExprC TypeEnv -> Type)
  (lambda (a tenv)
    (type-case ExprC a
      ...
      [idC (name) ...]
      ...)))
```

$$[\ \ldots\ \texttt{<Sym>} \leftarrow \tau\ \ldots\ ] \vdash \texttt{<Sym>} : \tau$$

TFAE Type Checker

```
(define typecheck : (ExprC TypeEnv -> Type)
  (lambda (a tenv)
    (type-case ExprC a
      ...
      [idC (name) (type-lookup name tenv)]
      ...)))
```

$$[\ \ldots\ \texttt{<Sym>} \leftarrow \tau\ \ldots\ ] \vdash \texttt{<Sym>} : \tau$$

TFAE Type Checker

```
(define typecheck : (ExprC TypeEnv -> Type)
  (lambda (a tenv)
    (type-case ExprC a
      ...
      [lamC (n arg-type body)
            ...]
      ...)))
```

$$\frac{\Gamma[\,\texttt{<Sym>} \leftarrow \tau_1\,] \vdash \texttt{e} : \tau_2}{\Gamma \vdash \{\texttt{lambda}\ \{[\texttt{<Sym>} : \tau_1]\}\ \texttt{e}\} : (\tau_1 \rightarrow \tau_2)}$$

TFAE Type Checker

```
(define typecheck : (ExprC TypeEnv -> Type)
  (lambda (a tenv)
    (type-case ExprC a
      ...
      [lamC (n arg-type body)
            ... (typecheck body ...) ...]
      ...)))
```

$$\frac{\Gamma[\,\texttt{<Sym>} \leftarrow \tau_1\,] \vdash \texttt{e} : \tau_2}{\Gamma \vdash \{\texttt{lambda}\ \{[\texttt{<Sym>} : \tau_1]\}\ \texttt{e}\} : (\tau_1 \rightarrow \tau_2)}$$

TFAE Type Checker

```
(define typecheck : (ExprC TypeEnv -> Type)
  (lambda (a tenv)
    (type-case ExprC a
      ...
      [lamC (n arg-type body)
            ... (typecheck body (extend-env
                                 (tbind n arg-type)
                                 tenv)) ...]
      ...)))
```

$$\frac{\Gamma[\,\texttt{<Sym>} \leftarrow \tau_1\,] \vdash \texttt{e} : \tau_2}{\Gamma \vdash \texttt{\{lambda \{[<Sym> : } \tau_1 \texttt{]\} e\}} : (\tau_1 \rightarrow \tau_2)}$$

TFAE Type Checker

```
(define typecheck : (ExprC TypeEnv -> Type)
  (lambda (a tenv)
    (type-case ExprC a
      ...
      [lamC (n arg-type body)
            (arrowT arg-type
                    (typecheck body (extend-env
                                     (tbind n arg-type)
                                     tenv)))]
      ...)))
```

$$\frac{\Gamma[\texttt{<Sym>} \leftarrow \tau_1] \vdash \texttt{e} : \tau_2}{\Gamma \vdash \{\texttt{lambda } \{[\texttt{<Sym>} : \tau_1]\} \texttt{ e}\} : (\tau_1 \rightarrow \tau_2)}$$

TFAE Type Checker

```
(define typecheck : (ExprC TypeEnv -> Type)
  (lambda (a tenv)
    (type-case ExprC a
      ...
      [appC (fun arg)
            ...]
      ...)))
```

$$\frac{\Gamma \vdash \mathbf{e}_1 \ : \ (\tau_2 \rightarrow \tau_3) \qquad \Gamma \vdash \mathbf{e}_2 \ : \ \tau_2}{\Gamma \vdash \{\mathbf{e}_1 \ \mathbf{e}_2\} \ : \ \tau_3}$$

TFAE Type Checker

```
(define typecheck : (ExprC TypeEnv -> Type)
  (lambda (a tenv)
    (type-case ExprC a
      ...
      [appC (fun arg)
            ... (typecheck fun tenv) ...
            ... (typecheck arg tenv) ...]
      ...)))
```

$$\frac{\Gamma \vdash \mathbf{e}_1 : (\tau_2 \rightarrow \tau_3) \qquad \Gamma \vdash \mathbf{e}_2 : \tau_2}{\Gamma \vdash \{\mathbf{e}_1\ \mathbf{e}_2\} : \tau_3}$$

TFAE Type Checker

```
(define typecheck : (ExprC TypeEnv -> Type)
  (lambda (a tenv)
    (type-case ExprC a
      ...
      [appC (fun arg)
            (type-case Type (typecheck fun tenv)
              [arrowT (arg-type result-type)
                      ... (typecheck arg tenv) ...]
              [else (type-error fun "function")])]
      ...)))
```

$$\frac{\Gamma \vdash \mathbf{e}_1 : (\tau_2 \rightarrow \tau_3) \qquad \Gamma \vdash \mathbf{e}_2 : \tau_2}{\Gamma \vdash \{\mathbf{e}_1\ \mathbf{e}_2\} : \tau_3}$$

TFAE Type Checker

```
(define typecheck : (ExprC TypeEnv -> Type)
  (lambda (a tenv)
    (type-case ExprC a
      ...
      [appC (fun arg)
            (type-case Type (typecheck fun tenv)
              [arrowT (arg-type result-type)
                      (if (equal? arg-type
                                  (typecheck arg tenv))
                          result-type
                          (type-error arg
                                      (to-string arg-type)))]
              [else (type-error fun "function")])]
      ...)))
```

$$\frac{\Gamma \vdash \mathbf{e}_1 : (\tau_2 \rightarrow \tau_3) \qquad \Gamma \vdash \mathbf{e}_2 : \tau_2}{\Gamma \vdash \{\mathbf{e}_1\ \mathbf{e}_2\} : \tau_3}$$

Part 9

`typecheck` and `interp`

Only call `interp` on an expression for which `typecheck` produces a type

`typecheck` ***never*** calls `interp`

`interp` ***never*** calls `typecheck`

Part 10

Pairs

```
{let {[cons : (num -> (num -> (bool -> num)))
             {lambda {x : num}
               {lambda {y : num}
                 {lambda {s : bool}
                   {if s x y}}}}]}
  {let {[first : ((bool -> num) -> num)
                {lambda {p : (num -> num)}
                  {p true}}]}
    {let {[rest : ((bool -> num) -> num)
                {lambda {p : (num -> num)}
                  {p false}}]}
      {rest {{cons 1} 2}}}}}
```

Pairs

```
{let {[cons : (bool -> (bool -> (bool -> bool)))
             {lambda {x : bool}
               {lambda {y : bool}
                 {lambda {s : bool}
                   {if s x y}}}}]}
  {let {[first : ((bool -> bool) -> bool)
                {lambda {p : (num -> bool)}
                  {p true}}]}
    {let {[rest : ((bool -> bool) -> bool)
                 {lambda {p : (num -> bool)}
                   {p false}}]}
      {rest {{cons true} false}}}}}
```

Pairs

```
{let {[cons : (num -> (bool -> (bool -> …)))
             {lambda {x : num}
               {lambda {y : bool}
                 {lambda {s : bool}
                   {if s x y}}}}]}
  {let {[first : ((bool -> …) -> …)
                {lambda {p : (num -> …)}
                  {p true}}]}
    {let {[rest : ((bool -> …) -> …)
                 {lambda {p : (num -> …)}
                   {p false}}]}
      {rest {{cons 1} false}}}}}
```

No possible type for …

Language with Pairs

```
<Expr>  ::=  <Num>
          |  {+ <Expr> <Expr>}
          |  {* <Expr> <Expr>}
          |  <Sym>
          |  {lambda {[<Sym> : <TE>]} <Expr>}
          |  {<Expr> <Expr>}
          |  {cons <Expr> <Expr>}                    NEW
          |  {first <Expr>}                          NEW
          |  {rest <Expr>}                           NEW

<Type>  ::=  num
          |  bool
          |  (<Type> -> <Type>)
          |  (<Type> * <Type>)                       NEW
```

$$\frac{\Gamma \vdash \mathbf{e}_1 : \tau_1 \qquad \Gamma \vdash \mathbf{e}_2 : \tau_2}{\Gamma \vdash \{\mathbf{cons}\ \mathbf{e}_1\ \mathbf{e}_2\} : (\tau_1 \times \tau_2)}$$

Language with Pairs

```
<Expr>  ::=  <Num>
          |  {+ <Expr> <Expr>}
          |  {* <Expr> <Expr>}
          |  <Sym>
          |  {lambda {[<Sym> : <TE>]} <Expr>}
          |  {<Expr> <Expr>}
          |  {cons <Expr> <Expr>}                    NEW
          |  {first <Expr>}                          NEW
          |  {rest <Expr>}                           NEW

<Type>  ::=  num
          |  bool
          |  (<Type> -> <Type>)
          |  (<Type> * <Type>)                       NEW
```

$$\frac{\Gamma \vdash \mathbf{e} : (\tau_1 \times \tau_2)}{\Gamma \vdash \{\mathbf{first}\ \mathbf{e}\} : \tau_1}$$

Language with Pairs

```
<Expr>  ::=  <Num>
          |  {+ <Expr> <Expr>}
          |  {* <Expr> <Expr>}
          |  <Sym>
          |  {lambda {[<Sym> : <TE>]} <Expr>}
          |  {<Expr> <Expr>}
          |  {cons <Expr> <Expr>}                    NEW
          |  {first <Expr>}                          NEW
          |  {rest <Expr>}                           NEW

<Type>  ::=  num
          |  bool
          |  (<Type> -> <Type>)
          |  (<Type> * <Type>)                       NEW
```

$$\frac{\Gamma \vdash \mathbf{e} : (\tau_1 \times \tau_2)}{\Gamma \vdash \{\mathbf{rest}\ \mathbf{e}\} : \tau_2}$$