Part I

Expressions and Values

What is the value of the following expression?

```
{+ 1 2}
```

Wrong answer: **0**

Wrong answer: **42**

Answer: **3**

Expressions and Values

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

```
<Exp> ::= <Number>
       |  #t
       |  #f
       |  {+ <Exp> <Exp>}
       |  {* <Exp> <Exp>}
       |  {= <Exp> <Exp>}
       |  <Symbol>
       |  {lambda {<Symbol>*} <Exp>}
       |  {<Exp> <Exp>*}
       |  {if <Exp> <Exp> <Exp>}
```

Expressions and Values

Is the following an expression?

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

Wrong answer: **No**

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

```
<Exp> ::= <Number>
        | #t
        | #f
        | {+ <Exp> <Exp>}
        | {* <Exp> <Exp>}
        | {= <Exp> <Exp>}
        | <Symbol>
        | {lambda {<Symbol>*} <Exp>}
        | {<Exp> <Exp>*}
        | {if <Exp> <Exp> <Exp>}
```

Expressions and Values

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

Expressions and Values

What is the value of the following expression?

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

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

Expressions and Values

Is the following a well-formed expression?

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

Answer: **Yes**

Expressions and Values

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

Expressions and Values

Is the following a well-formed expression?

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

Answer: **No**

Expressions and Values

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*

Expressions and Values

Is the following a well-formed expression?

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

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

Expressions and Values

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!

Expressions and Values

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}}}
```

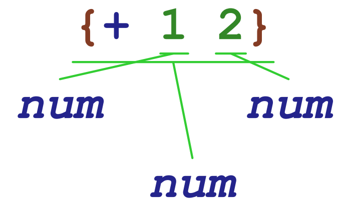
Overall strategy:

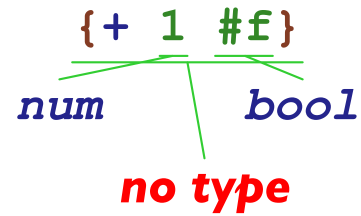
- Assign a ***type*** to each expression *without evaluating*
- Compute the type of a complex expression based on the types of its subexpressions

Types

```
1 : num

#t : bool
```





Part 3

Type Rules

<Number> : *num*

#t : *bool*

#f : *bool*

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

1 : *num*

#t : *bool*

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

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

Type Rules

<Number> : *num*

#t : *bool*

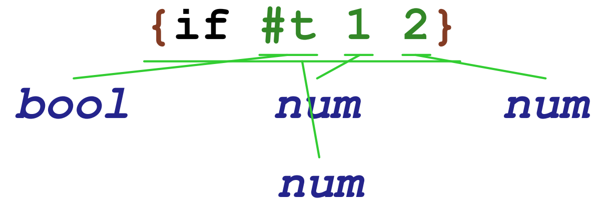
#f : *bool*

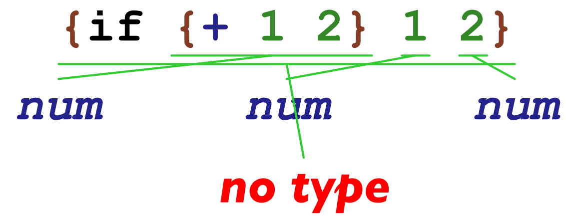
$$\frac{\texttt{<Exp>}_1 : num \qquad \texttt{<Exp>}_2 : num}{\texttt{\{+ <Exp>}_1\ \texttt{<Exp>}_2\texttt{\}} : num}$$

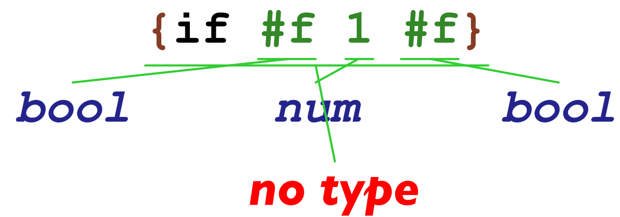
$$\frac{\dfrac{\texttt{1} : num \qquad \texttt{2} : num}{\texttt{\{+ 1 2\}} : num} \qquad \texttt{3} : num}{\texttt{\{+ \{+ 1 2\} 3\}} : num}$$

Part 4

Types: Conditionals







Conditional Type Rules

<Exp>$_1$: *bool*   <Exp>$_2$: <Type>$_0$ <Exp>$_3$: <Type>$_0$

{if <Exp>$_1$ <Exp>$_2$ <Exp>$_3$} : <Type>$_0$

#t : *bool*   1 : *num*   2 : *num*

{if #t 1 2} : *num*

{+ 1 2} : *num*   1 : *num*   2 : *num*

{if {+ 1 2} 1 2} : ***no type***

#f : *bool*   1 : *num*   #f : *bool*

{if #f 1 #f} : ***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

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

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

Abbreviations:
$\tau = \texttt{<Type>}$
$\mathbf{e} = \texttt{<Exp>}$
$\Gamma = \texttt{<Env>}$

Variable and Function Type Rules

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

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

$$\varnothing \vdash \texttt{x} : \textit{no type}$$

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

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

Revised Rules

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

$$\Gamma \vdash \texttt{\#t} : \mathit{bool}$$

$$\Gamma \vdash \texttt{\#f} : \mathit{bool}$$

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

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

Part 6

Types: Function Calls

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

(*bool* → *num*) *bool*

num

```
{{lambda {[x : bool]} {if x 1 2}} 5}
```

(*bool* → *num*) *num*

no type

```
{7 5}
```

num *num*

no type

Function Call Type Rule

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

$$\frac{\emptyset \vdash \texttt{\{lambda \{[x : bool]\} \{if x 1 2\}\}} : (bool \rightarrow num) \qquad \emptyset \vdash \texttt{\#t} : bool}{\emptyset \vdash \texttt{\{\{lambda \{[x : bool]\} \{if x 1 2\}\} \#t\}} : num}$$

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

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

Part 7

Types: Multiple Arguments

```
{lambda {[x : num] [y : num]} {+ x y}}
```

num num

num

(*num num* → *num*)

```
{{lambda {[x : num] [y : num]} {+ x y}} 5 6}
```

(*num num* → *num*) num num

num

```
{{lambda {[x : num] [y : num]} {+ x y}} 5}
```

(*num num* → *num*) num

no type

Revised Function and Call Rules

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

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

Part 8

Typed Language

```
<Exp>   ::= <Number>
          | {+ <Exp> <Exp>}
          | {* <Exp> <Exp>}
          | <Symbol>
          | {lambda {[<Symbol> : <Type>]} <Exp>}
          | {<Exp> <Exp>}

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

Expressions

```
(define-type Exp
  (numE [n : Number])
  (idE [s : Symbol])
  (plusE [l : Exp]
         [r : Exp])
  (multE [l : Exp]
         [r : Exp])
  (lamE [n : Symbol]
        [arg-type : Type]
        [body : Exp])
  (appE [fun : Exp]
        [arg : Exp]))
```

Types and Type Bindings

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

(define-type Type-Binding
  (tbind [name : Symbol]
         [type : Type]))

(define-type-alias Type-Env (Listof Type-Binding))
```

Type Checker

```
(define typecheck : (Exp Type-Env -> Type)
  (lambda (a tenv)
    (type-case Exp a
      ...
      ...
      ...)))
```

Type Checker

```
(define typecheck : (Exp Type-Env -> Type)
  (lambda (a tenv)
    (type-case Exp a
      ...
      [(numE n) ...]
      ...)))
```

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

Type Checker

```
(define typecheck : (Exp Type-Env -> Type)
  (lambda (a tenv)
    (type-case Exp a
      ...
      [(numE n) (numT)]
      ...)))
```

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

Type Checker

```
(define typecheck : (Exp Type-Env -> Type)
  (lambda (a tenv)
    (type-case Exp a
      ...
      [(plusE l r)
       ...]
      ...)))
```

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

Type Checker

```
(define typecheck : (Exp Type-Env -> Type)
  (lambda (a tenv)
    (type-case Exp a
      ...
      [(plusE l r)
       ... (typecheck l tenv) ...
       ... (typecheck r tenv) ...]
      ...)))
```

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

Type Checker

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

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

Type Checker

```
(define typecheck : (Exp Type-Env -> Type)
  (lambda (a tenv)
    (type-case Exp a
      ...
      [(plusE 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")])]
      ...)))
```

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

Type Checker

```
(define typecheck : (Exp Type-Env -> Type)
  (lambda (a tenv)
    (type-case Exp a
      ...
      [(idE name) ...]
      ...)))
```

[… `<Symbol>`$\leftarrow\tau$ …] $\vdash$ `<Symbol> :` τ

Type Checker

```
(define typecheck : (Exp Type-Env -> Type)
  (lambda (a tenv)
    (type-case Exp a
      ...
      [(idE name) (type-lookup name tenv)]
      ...)))
```

[… <Symbol>←τ …] ⊢ <Symbol> : τ

Type Checker

```
(define typecheck : (Exp Type-Env -> Type)
  (lambda (a tenv)
    (type-case Exp a
      ...
      [(lamE n arg-type body)
            ...]
      ...)))
```

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

Type Checker

```
(define typecheck : (Exp Type-Env -> Type)
  (lambda (a tenv)
    (type-case Exp a
      ...
      [(lamE n arg-type body)
       ... (typecheck body ...) ...]
      ...)))
```

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

Type Checker

```
(define typecheck : (Exp Type-Env -> Type)
  (lambda (a tenv)
    (type-case Exp a
      ...
      [(lamE n arg-type body)
       ... (typecheck body (extend-env
                             (tbind n arg-type)
                            tenv)) ...]
      ...)))
```

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

Type Checker

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

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

Type Checker

```
(define typecheck : (Exp Type-Env -> Type)
  (lambda (a tenv)
    (type-case Exp a
      ...
      [(appE fun arg)
       ...]
      ...)))
```

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

Type Checker

```
(define typecheck : (Exp Type-Env -> Type)
  (lambda (a tenv)
    (type-case Exp a
      ...
      [(appE fun arg)
       ... (typecheck fun tenv) ...
       ... (typecheck arg tenv) ...]
      ...)))
```

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

Type Checker

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

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

Type Checker

```
(define typecheck : (Exp Type-Env -> Type)
  (lambda (a tenv)
    (type-case Exp a
      ...
      [(appE 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 {[pair : (num -> (num -> (bool -> num)))
            {lambda {x : num}
              {lambda {y : num}
                {lambda {s : bool}
                  {if s x y}}}}]}
  {let {[fst : ((bool -> num) -> num)
               {lambda {p : (bool -> num)}
                 {p #t}}]}
    {let {[snd : ((bool -> num) -> num)
                 {lambda {p : (bool -> num)}
                   {p #f}}]}
      {snd {{pair 1} 2}}}}}
```

Pairs

```
{let {[pair : (bool -> (bool -> (bool -> bool)))
            {lambda {x : bool}
              {lambda {y : bool}
                {lambda {s : bool}
                  {if s x y}}}}]}
  {let {[fst : ((bool -> bool) -> bool)
               {lambda {p : (bool -> bool)}
                 {p #t}}]}
    {let {[snd : ((bool -> bool) -> bool)
                 {lambda {p : (bool -> bool)}
                   {p #f}}]}
      {snd {{pair #t} #f}}}}}
```

Pairs

```
{let {[pair : (num -> (bool -> (bool -> …)))
            {lambda {x : num}
              {lambda {y : bool}
                {lambda {s : bool}
                  {if s x y}}}}]}
  {let {[fst : ((bool -> …) -> …)
              {lambda {p : (bool -> …)}
                {p #t}}]}
    {let {[snd : ((bool -> …) -> …)
                {lambda {p : (bool -> …)}
                  {p #f}}]}
      {snd {{pair 1} #f}}}}}
```

No possible type for …

Language with Pairs

```
<Exp>   ::= <Number>
          |  {+ <Exp> <Exp>}
          |  {* <Exp> <Exp>}
          |  <Symbol>
          |  {lambda {[<Symbol> : <TE>]} <Exp>}
          |  {<Exp> <Exp>}
          |  {pair <Exp> <Exp>}                    NEW
          |  {fst <Exp>}                           NEW
          |  {snd <Exp>}                           NEW

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

$$\frac{\Gamma \vdash e_1 : \tau_1 \qquad \Gamma \vdash e_2 : \tau_2}{\Gamma \vdash \texttt{\{pair } e_1\ e_2\texttt{\}} : (\tau_1 \times \tau_2)}$$

Language with Pairs

```
<Exp>   ::= <Number>
          | {+ <Exp> <Exp>}
          | {* <Exp> <Exp>}
          | <Symbol>
          | {lambda {[<Symbol> : <TE>]} <Exp>}
          | {<Exp> <Exp>}
          | {pair <Exp> <Exp>}                     NEW
          | {fst <Exp>}                            NEW
          | {snd <Exp>}                            NEW

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

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

Language with Pairs

```
<Exp>   ::= <Number>
          | {+ <Exp> <Exp>}
          | {* <Exp> <Exp>}
          | <Symbol>
          | {lambda {[<Symbol> : <TE>]} <Exp>}
          | {<Exp> <Exp>}
          | {pair <Exp> <Exp>}          NEW
          | {fst <Exp>}                 NEW
          | {snd <Exp>}                 NEW

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

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