Part 1
Quiz

What is the type of the following expression?

\{ \text{lambda} \ {x} \ \{ + \ x \ 1 \} \}

**Answer:** Yet another trick question; it’s not an expression in our typed language, because the argument type is missing

But it seems like the answer should be $(\text{num} \rightarrow \text{num})$
Type Inference

*Type inference* is the process of inserting type annotations where the programmer omits them.

We’ll use explicit question marks, to make it clear where types are omitted.

\[
\{ \text{lambda} \, \{ [x : ?] \} \, \{ + \, x \, 1 \} \}
\]

\[
<Type> \ ::= \ \text{num} \\
| \ \text{bool} \\
| \ (\langle Type \rangle \rightarrow \langle Type \rangle) \\
| \ ?
\]
Part 2
Type Inference

\[
\{ \text{lambda } \{ [x : ?] \} \{ + x 1 \} \}
\]

\[
\text{T}_1 \quad \text{num}
\]

\[
\text{num} \quad \text{T}_1 = \text{num}
\]

\[
(\text{num} \rightarrow \text{num})
\]

- Create a new type variable for each `?`
- Change type comparison to install type equivalences
Type Inference

\[
\{\text{lambda } \{[x : ?]\} \{ + x 1\}\} \\
\downarrow \\
num \quad T_1 = num \\
\downarrow \\
(num \rightarrow num)
\]

\[
\{\text{lambda } \{[x : ?]\} \{\text{if true 1 x}\}\} \\
\downarrow \\
bool \quad num \quad T_1 \\
\downarrow \\
num \quad T_1 = num \\
\downarrow \\
(num \rightarrow num)
\]
Type Inference: Impossible Cases

\{\texttt{lambda} \{ [x : ?] \} \{ \texttt{if} \ x \ 1 \ x \}\}
Type Inference: Many Cases

\[
\{ \lambda \ \{ [y : ?] \} \ y \} \\
\text{T}_1 \\
(\text{T}_1 \rightarrow \text{T}_1)
\]

- Sometimes, more than one type works
  - \((\text{num} \rightarrow \text{num})\)
  - \((\text{bool} \rightarrow \text{bool})\)
  - \(((\text{num} \rightarrow \text{bool}) \rightarrow (\text{num} \rightarrow \text{bool}))\)

so the type checker leaves variables in the reported type
Part 3
Type Inference: Function Calls

\[
\begin{align*}
\{ \{ \text{lambda} \{ [y : ?] \} \ y \} \ & \ \{ \text{lambda} \{ [x : ?] \} \ \{ + \ x \ 1 \} \} \} \\
& \ (T_1 \rightarrow T_1) \quad (\text{num} \rightarrow \text{num}) \quad (\text{num} \rightarrow \text{num}) \\
\text{T}_1 &= (\text{num} \rightarrow \text{num})
\end{align*}
\]
Type Inference: Function Calls

{\texttt{lambda } \{ [y : ?] \} \ {\{ y \ 7 \} \}}

\( T_2 \) \( T_1 = (num \rightarrow T_2) \)

\( ((num \rightarrow T_2) \rightarrow T_2) \)

• In general, create a new type variable record for the result of a function call
Part 4
Type Inference: Cyclic Equations

\{ \text{lambda} \ \{ [x : \ ?] \} \ \{ x \ x \} \}

\[ T_1 \quad T_1 \]

\textit{no type: } T_1 \text{ can't be } (T_1 \rightarrow \ldots)

- \( T_1 \) can’t be \textit{num}
- \( T_1 \) can’t be \textit{bool}
- Suppose \( T_1 \) is \((T_2 \rightarrow T_3)\)
  - \( T_2 \) must be \( T_1 \)
  - So we won’t get anywhere!
Type Inference: Cyclic Equations

\{ \text{lambda} \ \{ [x : ?] \} \ \{ x \ x \} \}

\[ \text{T}_1 \quad \text{T}_1 \]

\textit{no type:} \text{T}_1 \text{ can't be} \ (\text{T}_1 \rightarrow \ldots)

The \textbf{occurs check}:  

• When installing a type equivalence, make sure that the new type for \text{T} doesn’t already contain \text{T}
Part 5
Type Unification

Unify a type variable $T$ with a type $\tau_2$:

• If $T$ is set to $\tau_1$, unify $\tau_1$ and $\tau_2$
• If $\tau_2$ is already equivalent to $T$, succeed
• If $\tau_2$ contains $T$, then fail
• Otherwise, set $T$ to $\tau_2$ and succeed

Unify a type $\tau_1$ to type $\tau_2$:

• If $\tau_2$ is a type variable $T$, then unify $T$ and $\tau_1$
• If $\tau_1$ and $\tau_2$ are both $\textit{num}$ or $\textit{bool}$, succeed
• If $\tau_1$ is $(\tau_3 \rightarrow \tau_4)$ and $\tau_2$ is $(\tau_5 \rightarrow \tau_6)$, then
  ○ unify $\tau_3$ with $\tau_5$
  ○ unify $\tau_4$ with $\tau_6$
• Otherwise, fail
Type Errors

Checking — report that an expression doesn’t have an expected type (expressed as a string):

\[
\text{type-error} : (\text{ExprC} \ \text{string} \rightarrow \ldots)
\]

Inference — report that, near some expression, two types are incompatible:

\[
\text{type-error} : (\text{ExprC} \ \text{Type} \ \text{Type} \rightarrow \ldots)
\]
Part 6
Type Grammar, Again

\[ <\text{Type}> ::= \text{num} \]
\[ | \quad \text{bool} \]
\[ | \quad (<\text{Type}> \to <\text{Type}>) \]
\[ | \quad ? \]
Representing Type Variables

(define-type Type
    [numT]
    [boolT]
    [arrowT (arg : Type)
            (result : Type)]
    [varT (is : (boxof (optionof Type)))]
)

(varT (box (none)))
Representing Type Variables

(define-type Type
  [numT]
  [boolT]
  [arrowT (arg : Type)
    (result : Type)]
  [varT (is : (boxof (optionof Type)))]
)

(varT (box (numT)))
Representing Type Variables

(\texttt{define-type Type}
\begin{itemize}
\item \texttt{numT}
\item \texttt{boolT}
\item \texttt{arrowT (arg : Type)}
\begin{itemize}
\item \texttt{(result : Type)}
\end{itemize}
\item \texttt{varT (is : (boxof (optionof Type)))}
\end{itemize})

(\texttt{type-case Type t}
\begin{itemize}
\item \texttt{...}
\item \texttt{[varT (b)}}
\begin{itemize}
\item [\texttt{...}]
\begin{itemize}
\item [\texttt{(set-box! b (numT))}]
\item [\texttt{...}]
\end{itemize}
\item [\texttt{...}]
\end{itemize}
\end{itemize})
Part 7
Unification Examples

(test (unify! (numT)
          (numT))
       (values))
Unification Examples

(test (unify! (boolT) (boolT)) (values))
Unification Examples

(test/exn (unify! (numT)
       (boolT))
    "no type")
(test (unify! (varT (box (none)))
   (numT))
   (values))
Unification Examples

(test (unify! (varT (box (numT))))
   (numT))
(values))
Unification Examples

```
(test/exn (unify! (varT (box (boolT))))
   (numT))
 "no type")
```
Unification Examples

(test/exn (let ([t (varT (box (none)))]))
  (begin
    (unify! t
      (numT))
    (unify! t
      (boolT))))
  "no type")
Unification Examples

(test (let ([t (varT (box (none)))]))
  (begin
    (unify! t
      (numT))
    (unify! t
      (numT)))
  (values))
Unification Examples

(test (let ([t (varT (box (none)))]))
  (begin
    (unify! (arrowT t (boolT))
      (arrowT (numT) (boolT)))
    (unify! t
      (numT)))
  (values))
Unification Examples

(test/exn (let ([t (varT (box (none)))]
   (unify! (arrowT t (boolT))
     t))
   "no type")
Unification Examples

(test (let ([t1 (varT (box (none)))]
  [t2 (varT (box (none)))]
  (unify! t1
t2))
(values))
(test/exn (let ([t1 (varT (box (none)))]
               [t2 (varT (box (none)))]
               (begin
                 (unify! t1 t2)
                 (unify! t1 (numT))
                 (unify! t2 (boolT)))
               "no type")
(test/exn (let ([t1 (varT (box (none)))]
             [t2 (varT (box (none)))]
    (begin
      (unify! t1
        t2)
      (unify! t2
        (boolT))
      (unify! t1
        (numT)))
    "no type")
Unification Examples

(test/exn (let ([t1 (varT (box (none)))]
                 [t2 (varT (box (none)))]
           (begin
                (unify! t1
                        (arrowT t2 (boolT)))
                (unify! t1
                        (arrowT (numT) t2)))
            "no type")
Part 8
Type Unification

(define (unify! [t1 : Type] [t2 : Type] [expr : ExprC])
  (type-case Type t1
    [varT (is1)
      ...
    ]
    [else
      (type-case Type t2
        [varT (is2) (unify! t2 t1 expr)]
        [numT () (type-case Type t1
          [numT () (values)]
          [else (type-error expr t1 t2)])]
        [boolT () (type-case Type t1
          [boolT () (values)]
          [else (type-error expr t1 t2)])]
        [arrowT (a2 b2) (type-case Type t1
          [arrowT (a1 b1)
            (begin
              (unify! a1 a2 expr)
              (unify! b1 b2 expr))]
          [else (type-error expr t1 t2)])])])}
Type Unification

(define (unify! [t1 : Type] [t2 : Type] [expr : ExprC])
  (type-case Type t1
    [varT (is1) (type-case (optionof Type) (unbox is1)
      [some (t3) (unify! t3 t2 expr)]
      [none () (local [(define t3 (resolve t2))]
        (if (eq? t1 t3)
          (values)
          (if (occurs? t1 t3)
            (type-error expr t1 t3)
            (begin
              (set-box! is1 (some t3))
              (values))))))))

[else ...]])
Type Unification Helpers

(define (resolve [t : Type]) : Type
  (type-case Type t
      [varT (is)
        (type-case (optionof Type) (unbox is)
            [none () t]
            [some (t2) (resolve t2)]])
      [else t]))

(define (occurs? [r : Type] [t : Type]) : boolean
  (type-case Type t
      [numT () false]
      [boolT () false]
      [arrowT (a b)
        (or (occurs? r a)
            (occurs? r b))]
      [varT (is) (or (eq? r t)
        (type-case (optionof Type) (unbox is)
            [none () false]
            [some (t2) (occurs? r t2)]))])))
Type Checker with Inference

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

(define typecheck : (ExprC TypeEnv -> Type)
  (lambda (a tenv)
    (type-case ExprC a
      ...
      [plusC (l r) (begin
        (unify! (typecheck l env) (numT) l)
        (unify! (typecheck r env) (numT) r)
        (numT))]
      ...))))
Type Checker with Inference

(define typecheck : (ExprC TypeEnv -> Type)
    (lambda (a tenv)
        (type-case ExprC a
            ...
            [idC (name) (get-type name env)]
            [lamC (n arg-type body)
                (arrowT arg-type
                    (typecheck body (aBind name arg-type env)))]
        ...
    )))
Type Checker with Inference

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

      [appC (fn arg)
        (local [(define result-type (varT (box (none))))]
          (begin
            (unify! (arrowT (typecheck arg env) result-type)
                (typecheck fn env)
                fn)
            result-type))]

      ...
    )))