From Typing Rules to a Type Checker

Typing rules vs Typechecker

Last time, we saw typing rules

- Formal, precise *specification* of the programs we want to allow
- Defined as inference rules (from logic)
- *Relation!* Rules related programs to types. We can see if a program and a type match, but no actual inputs and outputs!

Today, we'll write a typechecker

- Meta-program, *implementation* that allows or rejects programs
- *Function!* Takes programs as inputs, produces types as outputs

Will need to bridge the gap

TFAE Grammar

```
<TFAE>  ::=  <num>
          |  {+ <TFAE> <TFAE>}
          |  {- <TFAE> <TFAE>}
          |  <id>
          |  {fun {<id> : <TE>} <TFAE>}
          |  {<TFAE> <TFAE>}

<TE>    ::=  number
          |  (<TE> -> <TE>)
```

TFAE Types

```
(define-type Type
  [numberT]
  [arrowT (param : Type)
          (result : Type)])

(define-type TypeEnv
  [mtEnv]
  [aBind (name : symbol)
         (type : Type)
         (rest : TypeEnv)])
```

TFAE Expressions

```
(define-type TFAE
  [num (n : number)]
  [add (lhs : TFAE)
       (rhs : TFAE)]
  [sub (lhs : TFAE)
       (rhs : TFAE)]
  [id (name : symbol)]
  [fun (param-name : symbol)
       (param-type : Type)
       (body : TFAE)]
  [app (fun-expr : TFAE)
       (arg-expr : TFAE)])
```

TFAE Type Checker

```
(define typecheck : (TFAE TypeEnv -> Type)
  (lambda (fae env)
    (type-case TFAE fae
      ...)))
```

TFAE Type Checker

```
(define typecheck : (TFAE TypeEnv -> Type)
  (lambda (fae env)
    (type-case TFAE fae
      [num (n) ...])))
```

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

TFAE Type Checker

```
(define typecheck : (TFAE TypeEnv -> Type)
  (lambda (fae env)
    (type-case TFAE fae
      [num (n) (numberT)])))
```

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

TFAE Type Checker

```
(define typecheck : (TFAE TypeEnv -> Type)
  (lambda (fae env)
    (type-case TFAE fae
      [add (l r)
           ... (typecheck l env) ...
           ... (typecheck r env) ...])))
```

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

TFAE Type Checker

```
(define typecheck : (TFAE TypeEnv -> Type)
  (lambda (fae env)
    (type-case TFAE fae
      [add (l r)
           (type-case Type (typecheck l env)
             [numberT ()
                      ... (typecheck r env) ...]
             [else (error 'typecheck
                          "expected number")])])))
```

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

TFAE Type Checker

```
(define typecheck : (TFAE TypeEnv -> Type)
  (lambda (fae env)
    (type-case TFAE fae
      [add (l r)
           (type-case Type (typecheck l env)
             [numberT ()
                      (type-case Type (typecheck r env)
                        [numberT () (numberT)]
                        [else (error 'typecheck
                                     "expected number")])]
             [else (error 'typecheck
                          "expected number")])])))
```

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

TFAE Type Checker

```
(define typecheck : (TFAE TypeEnv -> Type)
  (lambda (fae env)
    (type-case TFAE fae
      [id (name) ...])))
```

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

TFAE Type Checker

```
(define typecheck : (TFAE TypeEnv -> Type)
  (lambda (fae env)
    (type-case TFAE fae
      [id (name) (get-type name env)])))
```

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

TFAE Type Checker

```
(define typecheck : (TFAE TypeEnv -> Type)
  (lambda (fae env)
    (type-case TFAE fae
      [fun (param-name param-type body)
           ...])))
```

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

TFAE Type Checker

```
(define typecheck : (TFAE TypeEnv -> Type)
  (lambda (fae env)
    (type-case TFAE fae
      [fun (param-name param-type body)
        ... (typecheck body (aBind param-name
                                   param-type
                                   env)) ...])))
```

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

TFAE Type Checker

```
(define typecheck : (TFAE TypeEnv -> Type)
  (lambda (fae env)
    (type-case TFAE fae
      [fun (param-name param-type body)
        (arrowT param-type
                (typecheck body (aBind param-name
                                       param-type
                                       env)))])))
```

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

TFAE Type Checker

```
(define typecheck : (TFAE TypeEnv -> Type)
  (lambda (fae env)
    (type-case TFAE fae
      [app (fun-expr arg-expr)
           ...])))
```

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

TFAE Type Checker

```
(define typecheck : (TFAE TypeEnv -> Type)
  (lambda (fae env)
    (type-case TFAE fae
      [app (fun-expr arg-expr)
           ... (typecheck fun-expr env) ...
           ... (typecheck arg-expr env) ...])))
```

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

TFAE Type Checker

```
(define typecheck : (TFAE TypeEnv -> Type)
  (lambda (fae env)
    (type-case TFAE fae
      [app (fun-expr arg-expr)
           (type-case Type (typecheck fun-expr env)
             [arrowT (param-type result-type)
                     ... (typecheck arg-expr env) ...]
             [else (error 'typecheck
                          "expected 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 : (TFAE TypeEnv -> Type)
  (lambda (fae env)
    (type-case TFAE fae
      [app (fun-expr arg-expr)
           (type-case Type (typecheck fun-expr env)
             [arrowT (param-type result-type)
                     (if (equal? param-type
                                 (typecheck arg-expr env))
                         result-type
                         (error
                          'typecheck
                          "type mismatch"))]
             [else (error 'typecheck
                          "expected 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 get-type : (symbol TypeEnv -> Type)
  (lambda (name env)
    (type-case TypeEnv env
      [mtEnv () (error 'typecheck
                       "free identifier")]
      [aBind (name2 type rest)
             (if (equal? name name2)
                 type
                 (get-type name rest))])))
```