TFAE Grammar

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

<TE> ::= num
| (<TE> -> <TE>)
TFAE Types

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

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

(define-type TFAE
  [num (n : number)]
  [add (l : TFAE)
     (r : TFAE)]
  [sub (l : TFAE)
     (r : TFAE)]
  [id (name : symbol)]
  [fun (name : symbol)
     (t : Type)
     (body : TFAE)]
  [app (rator : TFAE)
     (rand : TFAE)]
)
TFAE Type Checker

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

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

\[
\Gamma \vdash <\text{num}> : \text{num}
\]
TFAE Type Checker

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

Γ ⊢ <num> : num
TFAE Type Checker

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

Γ ⊢ e₁ : num       Γ ⊢ e₂ : num
-----------------------------
Γ ⊢ {+ e₁ e₂} : num
TFAE Type Checker

(define typecheck : (TFAE TypeEnv -> Type)
  (lambda (fae env)
    (type-case TFAE fae
      [add (l r)
        (type-case Type (typecheck l env)
          [numT ()
            ... (typecheck r env) ...]
          [else (error 'typecheck
                        "add expects a num")]))))))

Γ ⊢ e₁ : num   Γ ⊢ e₂ : num

Γ ⊢ {+ e₁ e₂} : num
TFAE Type Checker

(define typecheck : (TFAE TypeEnv -> Type)
  (lambda (fae env)
    (type-case TFAE fae
      [add (l r)
        (type-case Type (typecheck l env)
          [numT ()
            (type-case Type (typecheck r env)
              [numT () (numT)]
              [else (error 'typecheck
                            "add expects a num")])]))
        [else (error 'typecheck
                    "add expects a num")])]))
)

Γ ⊢ e₁ : num  Γ ⊢ e₂ : num

Γ ⊢ {+ e₁ e₂} : num
TFAE Type Checker

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

[ ... <id>←τ ... ] ⊢ <id> : τ
TFAE Type Checker

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

[... <id> ← τ ...] ⊢ <id> : τ
TFAE Type Checker

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

Γ[<id>←τ₁] ⊨ e : τ₂

Γ ⊨ {fun {<id> : τ₁} e} : (τ₁ → τ₂)
TFAE Type Checker

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

\[ \Gamma[<id>\leftarrow\tau_1] \vdash e : \tau_2 \]

\[ \Gamma \vdash \{\text{fun}\ \{<id> : \tau_1\} \ e\} : (\tau_1 \rightarrow \tau_2) \]
TFAE Type Checker

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

\[
\Gamma[\langle \text{id}\rangle \leftarrow \tau_1] \vdash e : \tau_2 \\
\Gamma \vdash \{\text{fun} \ \{\langle \text{id}\rangle : \tau_1\} \ e\} : (\tau_1 \rightarrow \tau_2)
\]
TFAE Type Checker

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

\[
\begin{align*}
\Gamma & \vdash e_1 : (\tau_2 \rightarrow \tau_3) \quad \Gamma \vdash e_2 : \tau_2 \\
\hline
\Gamma & \vdash \{ e_1 \ e_2 \} : \tau_3
\end{align*}
\]
TFAE Type Checker

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

Γ ⊢ e₁ : (τ₂ → τ₃)  Γ ⊢ e₂ : τ₂

Γ ⊢ {e₁ e₂} : τ₃
TFAE Type Checker

(define typecheck : (TFAE TypeEnv -> Type)
  (lambda (fae env)
    (type-case TFAE fae
      [app (fn arg)
        (type-case Type (typecheck fn env)
          [arrowT (arg-type result-type)
            ... (typecheck arg env) ...]
          [else (error 'typecheck
                        "app expects a function")]]))))

Γ ⊢ e₁ : (τ₂ → τ₃)    Γ ⊢ e₂ : τ₂
______________________
Γ ⊢ {e₁ e₂} : τ₃
TFAE Type Checker

(define typecheck : (TFAE TypeEnv -> Type)
  (lambda (fae env)
    (type-case TFAE fae
      [app (fn arg)
         (type-case Type (typecheck fn env)
          [arrowT (arg-type result-type)
             [if (equal? arg-type
                  (typecheck arg env))
              result-type
              (error 'typecheck
                   "arg mismatch")]
              [else (error 'typecheck
                         "app expects a function")]]]]))

\[ \Gamma \vdash e_1 : (\tau_2 \rightarrow \tau_3) \quad \Gamma \vdash e_2 : \tau_2 \]
\[ \Gamma \vdash \{ e_1 \ e_2 \} : \tau_3 \]
TFAE Type Checker

```
(define get-type : (symbol TypeEnv -> Type)
  (lambda (name env)
    (type-case TypeEnv env
      [mtEnv () (error 'typecheck
                      "unbound identifier")]
      [aBind (name2 type rest)
        (if (equal? name name2)
          type
          (get-type name rest))]]))
```