# TFAE Grammar

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

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

# TFAE Expressions and Types

```
(define-type TE
  [NumTE]
  [BoolTE]
  [ArrowTE (arg : TE) (result : TE)])

(define-type Type
  [NumT]
  [BoolT]
  [ArrowT (arg : Type) (result : Type)])

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

# TFAE Type Checker

```
(define (typecheck [fae : FAE] [env : TypeEnv]) : Type
  (type-case FAE fae
    ...))
```

# TFAE Type Checker

```
(define (typecheck [fae : FAE] [env : TypeEnv]) : Type
  (type-case FAE fae
    [(Num n) ...]))
```

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

# TFAE Type Checker

```
(define (typecheck [fae : FAE] [env : TypeEnv]) : Type
  (type-case FAE fae
    [(Num n) (NumT)]))
```

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

# TFAE Type Checker

```
(define (typecheck [fae : FAE] [env : TypeEnv]) : Type
  (type-case FAE fae
    [(Add l r)
     ... (typecheck l env) ...
     ... (typecheck r env) ...]))
```

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

# TFAE Type Checker

```
(define (typecheck [fae : FAE] [env : TypeEnv]) : Type
  (type-case FAE fae
    [(Add l r)
     (type-case Type (typecheck l env)
       [(NumT)
        ... (typecheck r (env) ...)]
       [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 [fae : FAE] [env : TypeEnv]) : Type
  (type-case FAE fae
    [(Add l r)
         (type-case Type (typecheck l env)
           [(NumT)
                 (type-case Type (typecheck r env)
                   [(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 [fae : FAE] [env : TypeEnv]) : Type
  (type-case FAE fae
    [(Add l r) (type-assert (list l r) (NumT) env (NumT))]))
```

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

# TFAE Type Checker

```
(define (typecheck [fae : FAE] [env : TypeEnv]) : Type
  (type-case FAE fae
    [(Id name) ...]))
```

[ … <id>←τ … ] ⊢ <id> : τ

# TFAE Type Checker

```
(define (typecheck [fae : FAE] [env : TypeEnv]) : Type
  (type-case FAE fae
    [(Id name) (type-lookup name env)]))
```

[ … <id>←τ … ] ⊢ <id> : τ

# TFAE Type Checker

```
(define (typecheck [fae : FAE] [env : TypeEnv]) : Type
  (type-case FAE fae
    [(Fun name te body)
         ...]))
```

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

# TFAE Type Checker

```
(define (typecheck [fae : FAE] [env : TypeEnv]) : Type
  (type-case FAE fae
    [(Fun name te body)
     ... (parse-type te) ...
     ... (typecheck body ...) ...]))
```

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

# TFAE Type Checker

```
(define (typecheck [fae : FAE] [env : TypeEnv]) : Type
  (type-case FAE fae
    [(Fun name te body)
     (let* ([arg-type (parse-type te)]
            [body-type (typecheck body (aBind name arg-type env))])
       ...)]))
```

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

# TFAE Type Checker

```
(define (typecheck [fae : FAE] [env : TypeEnv]) : Type
  (type-case FAE fae
    [(Fun name te body)
     (let* ([arg-type (parse-type te)]
            [body-type (typecheck body (aBind name arg-type env))])
       (ArrowT arg-type body-type))]))
```

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

# TFAE Type Checker

```
(define (typecheck [fae : FAE] [env : TypeEnv]) : Type
  (type-case FAE fae
    [(Call fn 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 [fae : FAE] [env : TypeEnv]) : Type
  (type-case FAE fae
    [(Call fn arg)
     ... (typecheck fn env) ...
     ... (typecheck arg 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 [fae : FAE] [env : TypeEnv]) : Type
  (type-case FAE fae
    [(Call fn arg)
     (type-case Type (typecheck fn env)
       [(ArrowT arg-type result-type)
               ... (typecheck arg env) ...]
       [else (type-error fn "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 [fae : FAE] [env : TypeEnv]) : Type
  (type-case FAE fae
    [(Call fn arg)
     (type-case Type (typecheck fn env)
       [(ArrowT arg-type result-type)
               (if (equal? arg-type
                           (typecheck arg env))
                   result-type
                   (type-error arg
                               (to-string arg-type)))]
       [else (type-error fn "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 [fae : FAE] [env : TypeEnv]) : Type
  (type-case FAE fae
    [(Call fn arg)
     (type-case Type (typecheck fn env)
       [(ArrowT arg-type result-type)
        (type-assert (list arg) arg-type env result-type)]
       [else (type-error fn "function")])]))
```

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

```
(define (type-assert exprs type env result) : Type
  (cond
    [(empty? exprs) result]
    [(not (equal? (typecheck (first exprs) env) type))
     (type-error (first exprs) (type-to-string type))]
    [else (type-assert (rest exprs) type env result)]))
```

# Pairs

```
{with {pair : (num -> (num -> (num -> num)))
            {fun {x : num}
                 {fun {y : num}
                      {fun {s : num}
                           {if0 s x y}}}}}
      {with {fst : ((num -> num) -> num)
                 {fun {p : (num -> num)}
                      {p 0}}}
            {with {snd : ((num -> num) -> num)
                       {fun {p : (num -> num)}
                            {p 1}}}
                  {snd {{pair 1} 2}}}}}
```

# Pairs

```
{with {pair : (bool -> (bool -> (num -> bool)))
            {fun {x : bool}
                 {fun {y : bool}
                      {fun {s : num}
                           {if0 s x y}}}}}
      {with {fst : ((num -> bool) -> bool)
                 {fun {p : (num -> bool)}
                      {p 0}}}
            {with {snd : ((num -> bool) -> bool)
                       {fun {p : (num -> bool)}
                            {p 1}}}
                  {snd {{pair true} false}}}}}
```

# Pairs

```
{with {pair : (num -> (bool -> (num -> …)))
            {fun {x : num}
                 {fun {y : bool}
                      {fun {s : num}
                           {if0 s x y}}}}}
      {with {fst : ((num -> …) -> …)
                 {fun {p : (num -> …)}
                      {p 0}}}
            {with {snd : ((num -> …) -> …)
                       {fun {p : (num -> …)}
                            {p 1}}}
                  {snd {{pair 1} false}}}}}
```

# Pairs

```
{with {pair : (num -> (bool -> (num -> …)))
            {fun {x : num}
                 {fun {y : bool}
                      {fun {s : num}
                           {if0 s x y}}}}}
      {with {fst : ((num -> …) -> …)
                 {fun {p : (num -> …)}
                      {p 0}}}
            {with {snd : ((num -> …) -> …)
                       {fun {p : (num -> …)}
                            {p 1}}}
                  {snd {{pair 1} false}}}}}
```

**No possible type for …**

# TPFAE Grammar

```
<TPFAE>  ::=  <num>
          |   ...
          |   {fun {<id> : <TE>} <TPFAE>}
          |   {<TPFAE> <TPFAE>}
          |   {pair <TPFAE> <TPFAE>}        NEW
          |   {fst <TPFAE>}                 NEW
          |   {snd <TPFAE>}                 NEW

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

# TPFAE Grammar

```
<TPFAE>  ::=  <num>
          |   ...
          |   {fun {<id> : <TE>} <TPFAE>}
          |   {<TPFAE> <TPFAE>}
          |   {pair <TPFAE> <TPFAE>}        NEW
          |   {fst <TPFAE>}                 NEW
          |   {snd <TPFAE>}                 NEW

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

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

# TPFAE Grammar

```
<TPFAE>  ::=  <num>
          |   ...
          |   {fun {<id> : <TE>} <TPFAE>}
          |   {<TPFAE> <TPFAE>}
          |   {pair <TPFAE> <TPFAE>}        NEW
          |   {fst <TPFAE>}                 NEW
          |   {snd <TPFAE>}                 NEW

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

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

# TPFAE Grammar

```
<TPFAE>  ::=  <num>
          |   ...
          |   {fun {<id> : <TE>} <TPFAE>}
          |   {<TPFAE> <TPFAE>}
          |   {pair <TPFAE> <TPFAE>}        NEW
          |   {fst <TPFAE>}                 NEW
          |   {snd <TPFAE>}                 NEW

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

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

# New Type Variants

```
(define-type TE
  …
  [PairTE (left : TE)
          (right : TE)])
```

# New Type Variants

```
(define-type TE
  …
  [PairTE (left : TE)
          (right : TE)])

(define-type Type
  …
  [PairT (left : Type)
         (right : Type)])
```

# TPFAE Type Checker

```
(define (typecheck [fae : FAE] [env : TypeEnv]) : Type
  (type-case FAE fae
    ...))
```

# TPFAE Type Checker

```
(define (typecheck [fae : FAE] [env : TypeEnv]) : Type
  (type-case FAE fae
    [(Pair l r) (PairT ...)]))
```

# TPFAE Type Checker

```
(define (typecheck [fae : FAE] [env : TypeEnv]) : Type
  (type-case FAE fae
    [(Pair l r) (PairT (typecheck l env) ...)]))
```

# TPFAE Type Checker

```
(define (typecheck [fae : FAE] [env : TypeEnv]) : Type
  (type-case FAE fae
    [(Pair l r) (PairT (typecheck l env)
                       (typecheck r env))]))
```

# TPFAE Type Checker

```
(define (typecheck [fae : FAE] [env : TypeEnv]) : Type
  (type-case FAE fae
    [(Pair l r) (PairT (typecheck l env)
                       (typecheck r env))]
    [(Fst ex) ...]))
```

# TPFAE Type Checker

```
(define (typecheck [fae : FAE] [env : TypeEnv]) : Type
  (type-case FAE fae
    [(Pair l r) (PairT (typecheck l env)
                       (typecheck r env))]
    [(Fst ex)
         (type-case Type (typecheck ex env)
           [...])]))
```

# TPFAE Type Checker

```
(define (typecheck [fae : FAE] [env : TypeEnv]) : Type
  (type-case FAE fae
    [(Pair l r) (PairT (typecheck l env)
                       (typecheck r env))]
    [(Fst ex)
         (type-case Type (typecheck ex env)
           [(PairT l r) l]
           [else (...)])]))
```

# TPFAE Type Checker

```
(define (typecheck [fae : FAE] [env : TypeEnv]) : Type
  (type-case FAE fae
    [(Pair l r) (PairT (typecheck l env)
                       (typecheck r env))]
    [(Fst ex)
         (type-case Type (typecheck ex env)
           [(PairT l r) l]
           [else (type-error ex "not a pair")])]))
```

# TPFAE Type Checker

```
(define (typecheck [fae : FAE] [env : TypeEnv]) : Type
  (type-case FAE fae
    [(Fst ex)
         (type-case Type (typecheck ex env)
           [(PairT l r) l]
           [else (type-error ex "not a pair")])]
    [(Snd ex)
         (type-case Type (typecheck ex env) ...)]))
```

# TPFAE Type Checker

```
(define (typecheck [fae : FAE] [env : TypeEnv]) : Type
  (type-case FAE fae
    [(Fst ex)
         (type-case Type (typecheck ex env)
           [(PairT l r) l]
           [else (type-error 'interp "not a pair")])]
    [(Snd ex)
         (type-case Type (typecheck ex env)
           [(PairT l r) r]
           [else (type-error 'interp "not a pair")])]))
```

# Recursion

```
{with {mk-rec {fun {body}
                   {{fun {fX} {fX fX}}
                    {fun {fX}
                         {{fun {f} {body f}}
                          {fun {x} {{fX fX} x}}}}}}}
      {with {fib {mk-rec
                  {fun {fib}
                       {fun {n}
                            {if0 n
                                 1
                                 {if0 {- n 1}
                                      1
                                      {+ {fib {- n 1}}
                                         {fib {- n 2}}}}}}}}}
            {fib 4}}}
```

# Typechecking mk-rec

```
((lambda (x) (x x))
 (lambda (x) (x x)))
```

# Typechecking mk-rec

```
((lambda (x) (x x))
 (lambda (x) (x x)))
```

```
(ω ω)
```

```
([ω : A→B] ω)
```

# Typechecking mk-rec

```
((lambda (x) (x x))
 (lambda (x) (x x)))
```

```
(ω ω)
```

```
([ω : (A→B) →B] ω)
```

# Typechecking mk-rec

```
((lambda (x) (x x))
 (lambda (x) (x x)))
```

```
(ω ω)
```

```
([ω : ((A→B) →B) →B] ω)
```

# Typed Recursion

```
{with {mk-rec : (((num -> num) -> (num -> num)) -> (num -> num))
              {fun {body : ((num -> num) -> (num -> num))}
                   {{fun {fX : … -> (num -> num)} {fX fX}}
                    {fun {fX : … -> (num -> num)}
                         {{fun {f : (num -> num)} {body f}}
                          {fun {x : num} {{fX fX} x}}}}}}}
      {with {fib : (num -> num)
                 {mk-rec
                  {fun {fib : (num -> num)}
                       {fun {n : num}
                            {if0 n
                                 1
                                 {if0 {- n 1}
                                      1
                                      {+ {fib {- n 1}}
                                         {fib {- n 2}}}}}}}}}
            {fib 4}}}
```

# Typed Recursion

```
{with {mk-rec : (((num -> num) -> (num -> num)) -> (num -> num))
              {fun {body : ((num -> num) -> (num -> num))}
                   {{fun {fX : … -> (num -> num)} {fX fX}}
                    {fun {fX : … -> (num -> num)}
                         {{fun {f : (num -> num)} {body f}}
                          {fun {x : num} {{fX fX} x}}}}}}}
      {with {fib : (num -> num)
                 {mk-rec
                  {fun {fib : (num -> num)}
                       {fun {n : num}
                            {if0 n
                                 1
                                 {if0 {- n 1}
                                      1
                                      {+ {fib {- n 1}}
                                         {fib {- n 2}}}}}}}}}
            {fib 4}}}
```

**Nothing works in place of …**

# Extending the Type System

When the type system rejects your prefectly good program, it may be time to extend the type system

In this case, we can add **rec** as a core form, again

```
{rec {fib : (num -> num)
          {fun {n : num}
               {if0 n
                    1
                    {if0 {- n 1}
                         1
                         {+ {fib {- n 1}}
                            {fib {- n 2}}}}}}}
     {fib 4}}
```

We'll add **if0**, too, while we're at it

# TRCFAE Grammar

```
<TRCFAE>  ::=  <num>
           |   {+ <TRCFAE> <TRCFAE>}
           |   {- <TRCFAE> <TRCFAE>}
           |   <id>
           |   {fun {<id> : <TE>} <TRCFAE>}
           |   {<TRCFAE> <TRCFAE>}
           |   {if0 <TRCFAE> <TRCFAE> <TRCFAE>}        NEW
           |   {rec {<id> : <TE> <TRCFAE>} <TRCFAE>}   NEW
<TE>      ::=  num
           |   (<TE> -> <TE>)
```

# TRCFAE Datatypes

```
(define-type FAE
  ...
  [If0 (test-expr : FAE)
       (then-expr : FAE)
       (else-expr : FAE)]
  [Rec (name : symbol)
       (ty : TE)
       (rhs-expr : FAE)
       (body-expr : FAE)])
```

# TRCFAE Type Checker

```
(define typecheck : (FAE TypeEnv -> Type)
  (lambda (fae env)
    (type-case FAE fae
      ...
      [(If0 test-expr then-expr else-expr)
           (type-case Type (typecheck test-expr env)
             [(NumT) (local [(define test-ty
                                 (typecheck then-expr env))]
                       (if (equal? test-ty
                                   (typecheck else-expr env))
                           test-ty
                           (type-error else-expr
                                       (to-string test-ty))))]
             [else (type-error test-expr "num")])]))))
```

$$\frac{\Gamma \vdash \mathbf{e}_1 : \mathit{num} \qquad \Gamma \vdash \mathbf{e}_2 : \tau_0 \qquad \Gamma \vdash \mathbf{e}_3 : \tau_0}{\Gamma \vdash \{\mathbf{if0}\ \mathbf{e}_1\ \mathbf{e}_2\ \mathbf{e}_3\} : \tau_0}$$

# TRCFAE Type Checker

```
(define typecheck : (FAE TypeEnv -> Type)
  (lambda (fae env)
    (type-case FAE fae
      ...
      [(If0 test-expr then-expr else-expr)
           (let* ([test-type (type-assert (list test-expr) (NumT) env (NumT))]
                  [then-type (typecheck then-expr env)]
                  [else-type (type-assert (list else-expr) then-type env then-type)])
             else-type)])))
```

$$\frac{\Gamma \vdash \mathbf{e}_1 : \mathit{num} \qquad \Gamma \vdash \mathbf{e}_2 : \tau_0 \qquad \Gamma \vdash \mathbf{e}_3 : \tau_0}{\Gamma \vdash \{\mathbf{if0}\ \mathbf{e}_1\ \mathbf{e}_2\ \mathbf{e}_3\} : \tau_0}$$

# TRCFAE Type Checker

```
(define typecheck : (FAE TypeEnv -> Type)
  (lambda (fae env)
    (type-case FAE fae
      ...
      [(Rec name ty rhs-expr body-expr)
       (let* ([rhs-ty (parse-type ty)]
              [new-env (aBind name rhs-ty env)])
         (if (equal? rhs-ty (typecheck rhs-expr new-env))
             (typecheck body-expr new-env)
             (type-error rhs-expr (to-string rhs-ty))))])))
```

$$\frac{\Gamma[\texttt{<id>} \leftarrow \tau_0] \vdash \texttt{e}_0 : \tau_0 \qquad \Gamma[\texttt{<id>} \leftarrow \tau_0] \vdash \texttt{e}_1 : \tau_1}{\Gamma \vdash \{\texttt{rec}\ \{\texttt{<id>} : \tau_0\ \texttt{e}_0\}\ \texttt{e}_1\} : \tau_1}$$

# TRCFAE Type Checker

```
(define typecheck : (FAE TypeEnv -> Type)
  (lambda (fae env)
    (type-case FAE fae
      ...
      [(Rec name ty rhs-expr body-expr)
       (let* ([rhs-ty (parse-type ty)]
              [new-env (aBind name rhs-ty env)])
         (type-assert (list rhs-expr) rhs-ty new-env
                      (typecheck body-expr new-env)))])))
```

$$\frac{\Gamma[\texttt{<id>} \leftarrow \tau_0] \vdash \texttt{e}_0 : \tau_0 \qquad \Gamma[\texttt{<id>} \leftarrow \tau_0] \vdash \texttt{e}_1 : \tau_1}{\Gamma \vdash \{\texttt{rec}\ \{\texttt{<id>} : \tau_0\ \texttt{e}_0\}\ \texttt{e}_1\} : \tau_1}$$

# Typechecking our favourite function

```
(typecheck
  (Rec 'fact (ArrowTE (NumTE) (NumTE))
       (Fun 'n (NumTE)
            (If0 (Id 'n)
                 (Num 1)
                 (Mul (Id 'n) (Call (Id 'fact) (Sub (Id 'n) (Num 1)))
       (Call (Id 'fact) (Num 5)))
 (mtEnv))
```

# Typechecking our favourite function

```
(typecheck
  (Rec 'fact (ArrowTE (NumTE) (NumTE))
       (Fun 'n (NumTE)
            (If0 (Id 'n)
                 (Num 1)
                 (Mul (Id 'n) (Call (Id 'fact) (Sub (Id 'n) (Num 1)))
       (Call (Id 'fact) (Num 5)))
 (mtEnv))
```

$$\frac{\Gamma[\texttt{<id>} \leftarrow \tau_0] \vdash \texttt{e}_0 : \tau_0 \qquad \Gamma[\texttt{<id>} \leftarrow \tau_0] \vdash \texttt{e}_1 : \tau_1}{\Gamma \vdash \{\texttt{rec}\ \{\texttt{<id>} : \tau_0\ \texttt{e}_0\}\ \texttt{e}_1\} : \tau_1}$$