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}}}
```

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 the …

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}}
```

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 TFAE
  ...
  [if0 (test-expr : TFAE)
       (then-expr : TFAE)
       (else-expr : TFAE)]
  [rec (name : symbol)
       (ty : Type)
       (rhs-expr : TFAE)
       (body-expr : TFAE)])
```

TRCFAE Interpreter

```
(define (interp a-fae ds)
  (type-case TFAE a-fae
    ...
    [if0 (test-expr then-expr else-expr)
         (if (numzero? (interp test-expr ds))
             (interp then-expr ds)
             (interp else-expr ds))]
    [rec (bound-id type named-expr body-expr)
         (local [(define value-holder (box (numV 42)))
                 (define new-ds (aRecSub bound-id
                                         value-holder
                                         ds))]
           (begin
             (set-box! value-holder (interp named-expr new-ds))
             (interp body-expr new-ds)))]))
```

TRCFAE Interpreter Lookup

```
(define (lookup name ds)
  (type-case DefrdSub ds
    [mtSub () (error 'lookup "free variable")]
    [aSub (sub-name val rest-ds)
          (if (symbol=? sub-name name)
              val
              (lookup name rest-ds))]
    [aRecSub (sub-name val-box rest-ds)
             (if (symbol=? sub-name name)
                 (unbox val-box)
                 (lookup name rest-ds))]))
```

TRCFAE Type Checker

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

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

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>
           |  {+ <TPFAE> <TPFAE>}
           |  {- <TPFAE> <TPFAE>}
           |  <id>
           |  {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>
           |  {+ <TPFAE> <TPFAE>}
           |  {- <TPFAE> <TPFAE>}
           |  <id>
           |  {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>
           |  {+ <TPFAE> <TPFAE>}
           |  {- <TPFAE> <TPFAE>}
           |  <id>
           |  {fun {<id> : <TE>} <TPFAE>}
           |  {<TPFAE> <TPFAE>}
           |  {pair <TPFAE> <TPFAE>}        NEW
           |  {fst <TPFAE>}                 NEW
           |  {snd <TPFAE>}                 NEW

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

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

TPFAE Grammar

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

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

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