Functions and Parsing

AE

```
{- 20 {+ {+ 10 10}
         {+ 10 10}}}

{- 20 {+ {+ 17 17}
         {+ 17 17}}}

{- 20 {+ {+ 3 3}
         {+ 3 3}}}
```

WAE

```
{with {x 10}
      {- 20 {+ {+ x x}
               {+ x x}}}}

{with {x 17}
      {- 20 {+ {+ x x}
               {+ x x}}}}

{with {x 3}
      {- 20 {+ {+ x x}
               {+ x x}}}}
```

FIWAE

```
{deffun {f x}
  {- 20 {+ {+ x x}
           {+ x x}}}}
```

```
{f 10}

{f 17}

{f 3}
```

F1WAE

```
{deffun {f x}                    {f 10}
  {- 20 {twice
          {twice x}}}}           {f 17}

{deffun {twice y}                {f 3}
  {+ y y}}
```

```
; interp : F1WAE? (listof FunDef?) -> number?
```

F1WAE

```
{deffun {f x}                    {f 10}
  {- 20 {twice
          {twice x}}}}           {f 17}

{deffun {twice y}                {f 3}
  {+ y y}}
```

<FunDef> ::= {deffun {<id> <id>} <F1WAE>}

F1WAE

```
{deffun {f x}                    {f 10}
  {- 20 {twice
          {twice x}}}}           {f 17}

{deffun {twice y}                {f 3}
  {+ y y}}
```

```
<FunDef>  ::=  {deffun {<id> <id>} <F1WAE>}
<F1WAE>   ::=  …
            |  {<id> <F1WAE>}
```

F1WAE Grammar

```
<FunDef>  ::=  {deffun {<id> <id>} <F1WAE>}        NEW

<F1WAE>   ::=  <num>
            |  {+ <F1WAE> <F1WAE>}
            |  {- <F1WAE> <F1WAE>}
            |  {with {<id> <F1WAE>} <F1WAE>}
            |  <id>
            |  {<id> <F1WAE>}                      NEW
```

F1WAE Datatypes

```
(define-type FunDef
  [fundef (fun-name symbol?)
          (param-name symbol?)
          (body F1WAE?)])

(define-type F1WAE
  [num (n number?)]
  [add (lhs F1WAE?)
       (rhs F1WAE?)]
  ...
  [app (fun-name symbol?)
       (arg F1WAE?)])
```

Interlude: Parsing

Parsing

```
{deffun {f x}
  {- 20 {+ {+ x x}
           {+ x x}}}}
```

vs

```
(fundef 'f 'x
        (sub (num 20) (add (add (id 'x)
                                (id 'x))
                           (add (id 'x)
                                (id 'x)))))
```

Parsing

```
{deffun {f x}
  {- 20 {+ {+ x x}
           {+ x x}}}}
```

What we want to write

Parsing

```
`{deffun {f x}
   {- 20 {+ {+ x x}
            {+ x x}}}}
```

What we have to write

The rules of Quasiquote

```
`{<exp> ...}  =>qq  (list `<exp> ...)
      `<id>   =>qq  `<id>
     `<num>   =>qq  <num>
```

For example:

```
    `{+ 3 {- x y}}
=>qq (list `+ `3 `{- x y})
=>qq (list `+ 3 `{- x y})
=>qq (list `+ 3 (list `- `x `y))
```

Writing a parser

```
(test (parse 1) (num 1))
(test (parse 'z) (id 'z))
(test (parse `{+ 1 2}) (add (num 1) (num 2)))
...
```

Writing a parser

```
(define (parse exp)
  (cond
    [(number? exp) (num exp)]
    [(symbol? exp) (id exp)]
    [(list? exp)
     (case (car exp)
       [(+)
        (check-pieces exp 3 "add")
        (add (parse (list-ref exp 1))
             (parse (list-ref exp 2)))]

       ; middle bit shown on next slide
        )]
    [else
     (parse-error "an expression" exp)]))
```

Writing a parser

```
[(-)
 (check-pieces exp 3 "sub")
 (sub (parse (list-ref exp 1))
      (parse (list-ref exp 2)))]
[(with)
 (check-pieces exp 3 "with")
 (check-pieces (list-ref exp 1) 2 "with binder")
 (with (list-ref (list-ref exp 1) 0)
       (parse (list-ref (list-ref exp 1) 1))
       (parse (list-ref exp 2)))]
[else
 (unless (symbol? (car exp))
   (parse-error "an expression" exp))
 (check-pieces exp 2 "app")
 (app (list-ref exp 0)
      (parse (list-ref exp 1)))]
```

Writing a parser

```
(define (check-pieces expression size what)
  (unless (and (list? expression)
               (= (length expression) size))
    (parse-error what expression)))

(define (parse-error what expression)
  (error 'parser
         "expected: ~a, found: ~a"
         what
         expression))
```

The actual rules of Quasiquote

$$\texttt{`\{<exp> ...\}} \Rightarrow_{qq} \texttt{(list `<exp> ...)}$$

$$\texttt{`<id>} \Rightarrow_{qq} \texttt{`<id>}$$

$$\texttt{`<num>} \Rightarrow_{qq} \texttt{<num>}$$

$$\texttt{`,<exp>} \Rightarrow_{qq} \texttt{<exp>}$$

Abstracting over programs

```
(define (n-additions n)
  (cond
    [(zero? n)
     `3]
    [else
     `{+ 1 ,(n-additions (sub1 n))}]))

(n-additions 2)
=>    `{+ 1 ,(n-additions 1)}
=>qq  (list `+ 1 (n-additions 1))
=>    (list `+ 1 `{+ 1 ,(n-additions 0)})
=>qq  (list `+ 1 (list `+ 1 (n-additions 0)))
=>    (list `+ 1 (list `+ 1 3))
aka   `{+ 1 {+ 1 3}}
```

Quote vs. Quasiquote

```
'{<exp> ...} =>q (list '<exp> ...)
      '<id> =>q '<id>
     '<num> =>q <num>
~~',<exp> =>q <exp>~~
```

Abbreviations

```
`<exp> = {quasiquote <exp>}
,<exp> = {unquote <exp>}
'<exp> = {quote <exp>}
```

Quoting mistakes

```
    '{a 'b}
=>q (list 'a ''b)
  = (list 'a '{quote b})
=>q (list 'a (list 'quote 'b))
```

Quoting mistakes

```
    '{a ,(n-additions 2)}
=>q (list 'a ',(n-additions 2))
  = (list 'a '{unquote (n-additions 2)})
=>q (list 'a
          (list 'unquote
                '(n-additions 2)))
=>q (list 'a
          (list 'unquote
                (list 'n-additions 2)))
```

Back to interpreting functions

F1WAE Interp

```
(define (interp an-f1wae fundefs)
  (type-case F1WAE an-f1wae
    [num (n) n]
    [add (l r) (+ (interp l fundefs)
                  (interp r fundefs))]
    ...
    [app (fun-name arg)
         ...]))


(test (interp (parse '{+ 1 1})
              empty)
      2)
```

F1WAE Interp

```
(define (interp an-f1wae fundefs)
  (type-case F1WAE an-f1wae
    [num (n) n]
    [add (l r) (+ (interp l fundefs)
                  (interp r fundefs))]
    ...
    [app (fun-name arg)
         ...]))


(test (interp (parse '{+ 1 1})
              (list
               (fundef 'f 'x
                       (parse '{+ x 3}))))
      2)
```

F1WAE Interp

```
(define (interp an-f1wae fundefs)
  (type-case F1WAE an-f1wae
    [num (n) n]
    [add (l r) (+ (interp l fundefs)
                  (interp r fundefs))]
    ...
    [app (fun-name arg)
         ...]))


(test (interp (parse '{f 1})
              (list
               (fundef 'f 'x
                       (parse '{+ x 3}))))
      4)
```

F1WAE Interp

```
(define (interp an-f1wae fundefs)
  (type-case F1WAE an-f1wae
    [num (n) n]
    [add (l r) (+ (interp l fundefs)
                  (interp r fundefs))]
    ...
    [app (fun-name arg)
         ...]))

(test (interp (parse '{f 10})
              (list
               (fundef 'f 'x
                       (parse '{- 20 {twice x}}))
               (fundef 'twice 'y
                       (parse '{+ y y}))))
              0)
```

F1WAE Interp

```
(define (interp an-f1wae fundefs)
  (type-case F1WAE an-f1wae
    [num (n) n]
    [add (l r) (+ (interp l fundefs)
                  (interp r fundefs))]
    ...
    [app (fun-name arg)
         ... (interp arg fundefs) ...]))
```

F1WAE Interp

```
(define (interp an-f1wae fundefs)
  (type-case F1WAE an-f1wae
    [num (n) n]
    [add (l r) (+ (interp l fundefs)
                  (interp r fundefs))]
    ...
    [app (fun-name arg)
         ... (lookup-fundef fun-name fundefs)
         ... (interp arg fundefs) ...]))
```

```
; lookup-fundef : symbol? (listof FunDef?) -> FunDef?
```

F1WAE Interp

```
(define (interp an-f1wae fundefs)
  (type-case F1WAE an-f1wae
    [num (n) n]
    [add (l r) (+ (interp l fundefs)
                  (interp r fundefs))]
    ...
    [app (fun-name arg)
         (local [(define a-fundef
                   (lookup-fundef fun-name fundefs))]
           (interp (subst (fundef-body a-fundef)
                          (fundef-param-name a-fundef)
                          (interp arg fundefs))
                   fundefs))]))
```

Lookup

```
; lookup-fundef : symbol? (listof FunDef?) -> FunDef?
(define (lookup-fundef name fundefs)
  ...)
```

Lookup

```
; lookup-fundef : symbol? (listof FunDef?) -> FunDef?
(define (lookup-fundef name fundefs)
  (cond
   [(empty? fundefs)
    ...]
   [else
    ... (first fundefs)
    ... (lookup-fundef name (rest fundefs))
    ...]))
```

Lookup

```
; lookup-fundef : symbol? (listof FunDef?) -> FunDef?
(define (lookup-fundef name fundefs)
  (cond
   [(empty? fundefs)
    (error 'interp "unknown function")]
   [else
    (if (symbol=? name (fundef-fun-name
                        (first fundefs)))
        (first fundefs)
        (lookup-fundef name (rest fundefs)))]))
```

Subst

```
; subst : F1WAE? symbol? number? -> F1WAE?
(define (subst a-f1wae sub-id val)
  (type-case F1WAE a-f1wae
    ...
    [app (fun-name arg)
         (app fun-name (subst arg sub-id val))]))
```