Administrative stuff

Use piazza for all course-related online discussion
(email for sensitive matters only).

Use link on course webpage (was broken, fixed now)

The Arbitrariness of Identifiers

The "Are the following two programs equivalent?" game

The Arbitrariness of Identifiers

Are the following two programs equivalent?

```
(define (f x) (+ x 1))
(f 10)
```

```
(define (f y) (+ y 1))
(f 10)
```

The Arbitrariness of Identifiers

Are the following two programs equivalent?

```
(define (f x) (+ x 1))
(f 10)
```

```
(define (f y) (+ y 1))
(f 10)
```

yes

argument is consistently renamed

The Arbitrariness of Identifiers

Are the following two programs equivalent?

```
(define (f x) (+ x 1))
(f 10)
```

```
(define (f x) (+ y 1))
(f 10)
```

The Arbitrariness of Identifiers

Are the following two programs equivalent?

```
(define (f x) (+ x 1))
(f 10)
```

```
(define (f x) (+ y 1))
(f 10)
```

no

not a use of the argument anymore

The Arbitrariness of Identifiers

Are the following two programs equivalent?

```
(define (f x) (+ x 1))
(f 10)
```

```
(define (f y) (+ x 1))
(f 10)
```

The Arbitrariness of Identifiers

Are the following two programs equivalent?

```
(define (f x) (+ x 1))
(f 10)
```

```
(define (f y) (+ x 1))
(f 10)
```

no

not a use of the argument anymore

The Arbitrariness of Identifiers

Are the following two programs equivalent?

```
(define (f x) (+ y 1))
(f 10)
```

```
(define (f z) (+ y 1))
(f 10)
```

The Arbitrariness of Identifiers

Are the following two programs equivalent?

```
(define (f x) (+ y 1))
(f 10)
```

```
(define (f z) (+ y 1))
(f 10)
```

yes

argument never used, so almost any name is ok

The Arbitrariness of Identifiers

Are the following two programs equivalent?

```
(define (f x) (+ y 1))
(f 10)
```

```
(define (f y) (+ y 1))
(f 10)
```

The Arbitrariness of Identifiers

Are the following two programs equivalent?

```
(define (f x) (+ y 1))
(f 10)
```

```
(define (f y) (+ y 1))
(f 10)
```

no

now a use of the argument

The Arbitrariness of Identifiers

Are the following two programs equivalent?

```
(define (f x) (+ y 1))
(f 10)
```

```
(define (f x) (+ z 1))
(f 10)
```

The Arbitrariness of Identifiers

Are the following two programs equivalent?

```
(define (f x) (+ y 1))
(f 10)
```

```
(define (f x) (+ z 1))
(f 10)
```

no

still an undefined identifier, but a different one

The Arbitrariness of Identifiers

Are the following two programs equivalent?

```
(define (f x)
  (local [(define y 10)]
    (+ x y)))
(f 0)
```

```
(define (f z)
  (local [(define y 10)]
    (+ z y)))
(f 0)
```

The Arbitrariness of Identifiers

Are the following two programs equivalent?

```
(define (f x)
  (local [(define y 10)]
    (+ x y)))
(f 0)
```

```
(define (f z)
  (local [(define y 10)]
    (+ z y)))
(f 0)
```

yes

argument is consistently renamed

The Arbitrariness of Identifiers

Are the following two programs equivalent?

```
(define (f x)
  (local [(define y 10)]
    (+ x y)))
(f 0)
```

```
(define (f x)
  (local [(define z 10)]
    (+ x z)))
(f 0)
```

The Arbitrariness of Identifiers

Are the following two programs equivalent?

```
(define (f x)
  (local [(define y 10)]
    (+ x y)))
(f 0)
```

```
(define (f x)
  (local [(define z 10)]
    (+ x z)))
(f 0)
```

yes

local identifier is consistently renamed

The Arbitrariness of Identifiers

Are the following two programs equivalent?

```
(define (f x)
  (local [(define y 10)]
    (+ x y)))
(f 0)
```

```
(define (f x)
  (local [(define x 10)]
    (+ x x)))
(f 0)
```

The Arbitrariness of Identifiers

Are the following two programs equivalent?

```
(define (f x)
  (local [(define y 10)]
    (+ x y)))
(f 0)
```

```
(define (f x)
  (local [(define x 10)]
    (+ x x)))
(f 0)
```

no

local identifier now shadows (hides) the argument

The Arbitrariness of Identifiers

Are the following two programs equivalent?

```
(define (f x)
  (local [(define y 10)]
    (+ x y)))
(f 0)
```

```
(define (f y)
  (local [(define y 10)]
    (+ y y)))
(f 0)
```

The Arbitrariness of Identifiers

Are the following two programs equivalent?

```
(define (f x)
  (local [(define y 10)]
    (+ x y)))
(f 0)
```

```
(define (f y)
  (local [(define y 10)]
    (+ y y)))
(f 0)
```

no

local identifier now shadows the argument

Free and Bound Identifiers

An identifier for the argument of a function or the name of a local identifier is a ***binding occurrence***

```
(define (f x y) (+ x y z))

(local [(define a 3)
        (define c 4)]
  (+ a b c))
```

Free and Bound Identifiers

A use of a function argument or a local identifier is a ***bound occurrence***

```
(define (f x y) (+ x y z))

(local [(define a 3)
        (define c 4)]
  (+ a b c))
```

Free and Bound Identifiers

A use of an identifier that is not function argument or a local identifier is a ***free identifier***

```
(define (f x y) (+ x y z))

(local [(define a 3)
        (define c 4)]
  (+ a b c))
```

Arithmetic Language

```
<AE>  ::=  <num>
        |  {+ <AE> <AE>}
        |  {- <AE> <AE>}
```

Arithmetic Language

```
<AE>  ::=  <num>
        |  {+ <AE> <AE>}
        |  {- <AE> <AE>}
```

```
(define-type AE
  [num (n number?)]
  [add (lhs AE?)
       (rhs AE?)]
  [sub (lhs AE?)
       (rhs AE?)])
```

Arithmetic Language

```
<AE>  ::=  <num>
        |  {+ <AE> <AE>}
        |  {- <AE> <AE>}
```

```
(define/contract (interp an-ae)
  (-> AE? number?)
  (type-case AE an-ae
    [num (n) n]
    [add (l r) (+ (interp l) (interp r))]
    [sub (l r) (- (interp l) (interp r))]))
```

Arithmetic Language

```
<AE>  ::=  <num>
        |  {+ <AE> <AE>}
        |  {- <AE> <AE>}
```

```
(define/contract (interp an-ae)
  (-> AE? number?)
  (type-case AE an-ae
    [num (n) n]
    [add (l r) (+ (interp l) (interp r))]
    [sub (l r) (- (interp l) (interp r))]))
```

No identifiers to help us study binding...

With Arithmetic Language

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

With Arithmetic Language

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

NEW

NEW

```
{with {x {+ 1 2}}
  {+ x x}}          ⇒   6
```

With Arithmetic Language

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

`x` ⇒ *error: free identifier*

With Arithmetic Language

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

```
{+ {with {x {+ 1 2}}
     {+ x x}}
   {with {x {- 4 3}}
     {+ x x}}}          ⇒   8
```

With Arithmetic Language

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

```
{+ {with {x {+ 1 2}}
     {+ x x}}
   {with {y {- 4 3}}
     {+ y y}}}          ⇒   8
```

With Arithmetic Language

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

```
{with {x {+ 1 2}}
  {with {x {- 4 3}}
    {+ x x}}}          ⇒   2
```

With Arithmetic Language

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

```
{with {x {+ 1 2}}
  {with {y {- 4 3}}
    {+ x x}}}          ⇒   6
```

With Arithmetic Language

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

```
(define-type WAE
  [num (n number?)]
  [add (lhs WAE?)
       (rhs WAE?)]
  [sub (lhs WAE?)
       (rhs WAE?)]
  [with (name symbol?)
        (named-expr WAE?)
        (body WAE?)]
  [id (name symbol?)])
```

With Arithmetic Language

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

```
(define/contract (interp a-wae)
  (-> WAE? number?)
  (type-case WAE a-wae
    [num (n) n]
    [add (l r) (+ (interp l) (interp r))]
    [sub (l r) (- (interp l) (interp r))]
    [with (bound-id named-expr body-expr)
      ...]
    [id (name)
        ...]))
```

With Arithmetic Language

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

```
(define/contract (interp a-wae)
  (-> WAE? number?)
  (type-case WAE a-wae
    [num (n) n]
    [add (l r) (+ (interp l) (interp r))]
    [sub (l r) (- (interp l) (interp r))]
    [with (bound-id named-expr body-expr)
      ...]
    [id (name)
        (error 'interp "free variable")]))
```

With Arithmetic Language

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

```
(define/contract (interp a-wae)
  (-> WAE? number?)
  (type-case WAE a-wae
    [num (n) n]
    [add (l r) (+ (interp l) (interp r))]
    [sub (l r) (- (interp l) (interp r))]
    [with (bound-id named-expr body-expr)
      ... (interp named-expr) ...]
    [id (name)
        (error 'interp "free variable")]))
```

With Arithmetic Language

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

```
(define/contract (interp a-wae)
  (-> WAE? number?)
  (type-case WAE a-wae
    [num (n) n]
    [add (l r) (+ (interp l) (interp r))]
    [sub (l r) (- (interp l) (interp r))]
    [with (bound-id named-expr body-expr)
      ... (interp named-expr)
      ... (interp body-expr) ...]
    [id (name)
        (error 'interp "free variable")]))
```

With Arithmetic Language

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

```
(define/contract (interp a-wae)
  (-> WAE? number?)
  (type-case WAE a-wae
    [num (n) n]
    [add (l r) (+ (interp l) (interp r))]
    [sub (l r) (- (interp l) (interp r))]
    [with (bound-id named-expr body-expr)
      (interp (subst body-expr bound-id
                     (interp named-expr)))]
    [id (name)
        (error 'interp "free variable")]))
```

Substitution

```
(define/contract (subst a-wae sub-id val)
  (-> WAE? symbol? number? WAE?)
  (type-case WAE a-wae
    [num (n) ...]
    [add (l r) ...]
    [sub (l r) ...]
    [with (bound-id named-expr body-expr)
      ...]
    [id (name) ...]))
```

Substitution

```
(define/contract (subst a-wae sub-id val)
  (-> WAE? symbol? number? WAE?)
  (type-case WAE a-wae
    [num (n) ...]
    [add (l r) ...]
    [sub (l r) ...]
    [with (bound-id named-expr body-expr)
      ...]
    [id (name) ...]))
```

Let's make examples/tests first...

Example Sustitutions

```
; 10 for x in  {+ 1 x}
```

Example Sustitutions

```
; 10 for x in  {+ 1 x} ⇒ {+ 1 10}
```

Example Sustitutions

```
; 10 for x in  {+ 1 x} ⇒ {+ 1 10}
(test (subst (add (num 1) (id 'x)) 'x 10)
      (add (num 1) (num 10)))
```

Example Sustitutions

```
; 10 for x in  {+ 1 x} ⇒ {+ 1 10}
(test (subst (add (num 1) (id 'x)) 'x 10)
      (add (num 1) (num 10)))

; 10 for x in  x
```

Example Sustitutions

```
; 10 for x in  {+ 1 x} ⇒ {+ 1 10}
(test (subst (add (num 1) (id 'x)) 'x 10)
      (add (num 1) (num 10)))

; 10 for x in  x ⇒ 10
```

Example Sustitutions

```
; 10 for x in  {+ 1 x} ⇒ {+ 1 10}
(test (subst (add (num 1) (id 'x)) 'x 10)
      (add (num 1) (num 10)))

; 10 for x in  x ⇒ 10
(test (subst (id 'x) 'x 10)
      (num 10))
```

Example Sustitutions

```
; 10 for x in  {+ 1 x} ⇒ {+ 1 10}
(test (subst (add (num 1) (id 'x)) 'x 10)
      (add (num 1) (num 10)))

; 10 for x in  x ⇒ 10
(test (subst (id 'x) 'x 10)
      (num 10))

; 10 for x in  y
```

Example Sustitutions

```
; 10 for x in  {+ 1 x} ⇒ {+ 1 10}
(test (subst (add (num 1) (id 'x)) 'x 10)
      (add (num 1) (num 10)))

; 10 for x in  x ⇒ 10
(test (subst (id 'x) 'x 10)
      (num 10))

; 10 for x in  y ⇒ y
```

Example Sustitutions

```
; 10 for x in  {+ 1 x} ⇒ {+ 1 10}
(test (subst (add (num 1) (id 'x)) 'x 10)
      (add (num 1) (num 10)))

; 10 for x in  x ⇒ 10
(test (subst (id 'x) 'x 10)
      (num 10))

; 10 for x in  y ⇒ y
(test (subst (id 'y) 'x 10)
      (id 'y))
```

Example Sustitutions

```
; 10 for x in  {+ 1 x} ⇒ {+ 1 10}
(test (subst (add (num 1) (id 'x)) 'x 10)
      (add (num 1) (num 10)))

; 10 for x in  x ⇒ 10
(test (subst (id 'x) 'x 10)
      (num 10))

; 10 for x in  y ⇒ y
(test (subst (id 'y) 'x 10)
      (id 'y))

; 10 for y in  {- x 1}
```

Example Sustitutions

```
; 10 for x in  {+ 1 x} ⇒ {+ 1 10}
(test (subst (add (num 1) (id 'x)) 'x 10)
      (add (num 1) (num 10)))

; 10 for x in  x ⇒ 10
(test (subst (id 'x) 'x 10)
      (num 10))

; 10 for x in  y ⇒ y
(test (subst (id 'y) 'x 10)
      (id 'y))

; 10 for y in  {- x 1} ⇒ {- x 1}
```

Example Sustitutions

```
; 10 for x in  {+ 1 x} ⇒ {+ 1 10}
(test (subst (add (num 1) (id 'x)) 'x 10)
      (add (num 1) (num 10)))

; 10 for x in  x ⇒ 10
(test (subst (id 'x) 'x 10)
      (num 10))

; 10 for x in  y ⇒ y
(test (subst (id 'y) 'x 10)
      (id 'y))

; 10 for y in  {- x 1} ⇒ {- x 1}
(test (subst (sub (id 'x) (num 1)) 'y 10)
      (sub (id 'x) (num 1)))
```

Substitution

```
; subst : WAE symbol num -> WAE
(define/contract (subst a-wae sub-id val)
  (-> WAE? symbol? number? WAE?)
  (type-case WAE a-wae
    [num (n) a-wae]
    [add (l r) (add (subst l sub-id val)
                    (subst r sub-id val))]
    [sub (l r) (sub (subst l sub-id val)
                    (subst r sub-id val))]
    [with (bound-id named-expr body-expr)
      ...]
    [id (name) (if (symbol=? name sub-id)
                   (num val)
                   a-wae)]))
```

Example Sustitutions

```
; 10 for x in  {with {y 17} x}
```

Example Sustitutions

```
; 10 for x in  {with {y 17} x} ⇒ {with {y 17} 10}
```

Example Sustitutions

```
; 10 for x in  {with {y 17} x} ⇒ {with {y 17} 10}
(test (subst (with 'y (num 17) (id 'x)) 'x 10)
      (with 'y (num 17) (num 10)))
```

Example Sustitutions

```
; 10 for x in  {with {y 17} x} ⇒ {with {y 17} 10}
(test (subst (with 'y (num 17) (id 'x)) 'x 10)
      (with 'y (num 17) (num 10)))

 ; 10 for x in  {with {y x} y}
```

Example Sustitutions

```
; 10 for x in  {with {y 17} x} ⇒ {with {y 17} 10}
(test (subst (with 'y (num 17) (id 'x)) 'x 10)
      (with 'y (num 17) (num 10)))

 ; 10 for x in  {with {y x} y} ⇒ {with {y 10} y}
```

Example Sustitutions

```
; 10 for x in  {with {y 17} x} ⇒ {with {y 17} 10}
(test (subst (with 'y (num 17) (id 'x)) 'x 10)
      (with 'y (num 17) (num 10)))

 ; 10 for x in  {with {y x} y} ⇒ {with {y 10} y}
 (test (subst (with 'y (id 'x) (id 'y)) 'x 10)
       (with 'y (num 10) (id 'y)))
```

Example Sustitutions

```
; 10 for x in  {with {y 17} x} ⇒ {with {y 17} 10}
(test (subst (with 'y (num 17) (id 'x)) 'x 10)
      (with 'y (num 17) (num 10)))

 ; 10 for x in  {with {y x} y} ⇒ {with {y 10} y}
 (test (subst (with 'y (id 'x) (id 'y)) 'x 10)
       (with 'y (num 10) (id 'y)))

 ; 10 for x in  {with {x y} x}
```

Example Sustitutions

```
; 10 for x in  {with {y 17} x} ⇒ {with {y 17} 10}
(test (subst (with 'y (num 17) (id 'x)) 'x 10)
      (with 'y (num 17) (num 10)))

 ; 10 for x in  {with {y x} y} ⇒ {with {y 10} y}
 (test (subst (with 'y (id 'x) (id 'y)) 'x 10)
       (with 'y (num 10) (id 'y)))

 ; 10 for x in  {with {x y} x} ⇒ {with {x y} x}
```

Example Sustitutions

```
; 10 for x in  {with {y 17} x} ⇒ {with {y 17} 10}
(test (subst (with 'y (num 17) (id 'x)) 'x 10)
      (with 'y (num 17) (num 10)))

; 10 for x in  {with {y x} y} ⇒ {with {y 10} y}
(test (subst (with 'y (id 'x) (id 'y)) 'x 10)
      (with 'y (num 10) (id 'y)))

; 10 for x in  {with {x y} x} ⇒ {with {x y} x}
(test (subst (with 'x (id 'y) (id 'x)) 'x 10)
      (with 'x (id 'y) (id 'x)))
```

Example Sustitutions

```
; 10 for x in  {with {y 17} x} ⇒ {with {y 17} 10}
(test (subst (with 'y (num 17) (id 'x)) 'x 10)
      (with 'y (num 17) (num 10)))

; 10 for x in  {with {y x} y} ⇒ {with {y 10} y}
(test (subst (with 'y (id 'x) (id 'y)) 'x 10)
      (with 'y (num 10) (id 'y)))

; 10 for x in  {with {x y} x} ⇒ {with {x y} x}
(test (subst (with 'x (id 'y) (id 'x)) 'x 10)
      (with 'x (id 'y) (id 'x)))
```

Substitution

Substitution replaces

- free identifiers with the same name
- no binding identifiers
- no bound identifiers

Substitution

Substitution replaces

- free identifiers with the same name
- no binding identifiers
- no bound identifiers

An identifier is bound when it appears in the body of a `with` binding the same name

Conversely, a free variable of a name appears in a `with` only if the `with` doesn't bind the name

Substitution

```
; subst : WAE symbol num -> WAE
(define/contract (subst a-wae sub-id val)
  (-> WAE? symbol? number? WAE?)
  (type-case WAE a-wae
    ...
    [with (bound-id named-expr body-expr)
          (with bound-id
                (subst named-expr sub-id val)
                (if (symbol=? bound-id sub-id)
                    body-expr
                    (subst body-expr sub-id val)))]
    ...))
```