# Desugaring Syntax Trees

Matt Might

University of Utah

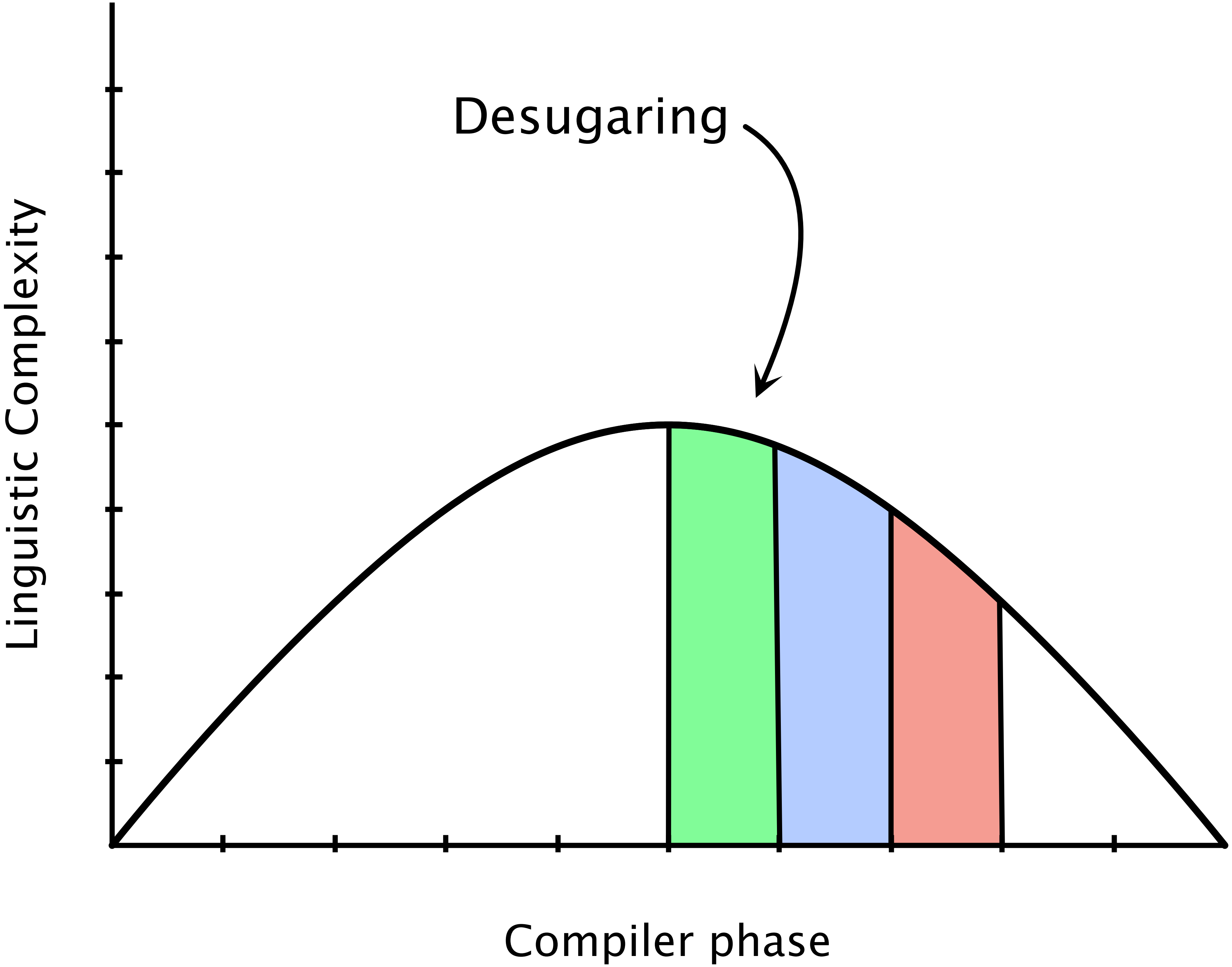
matt.might.net

# Desugaring Syntax Trees

Matt Might

University of Utah

matt.might.net


Desugaring
Linguistic Complexity
Compiler phase

# Pros & Cons

- Pro: Simplifies later transformations
- Con: May discard useful information

# Input language

```
<prog> ::= <top>*

<top> ::= <def> | <exp>

<def> ::= (define (<var> <var>*) <body>)
       |  (define <var> <exp>)
```

```
<exp> ::= <var>
       |  <literal>
       |  <prim>
       |  (lambda (<var>*) <body>)
       |  (let ((<var> <exp>)*) <body>)
       |  (letrec ((<var> <exp>)*) <body>)
       |  (cond (<exp> <exp>)* [(else <exp>)])
       |  (and <exp>*)
       |  (or <exp>*)
       |  (if <exp> <exp> [ <exp> ])
       |  (set! <var> <exp>)
       |  (begin <body>)
       |  (quote <s-exp>)
       |  (quasiquote <qq-exp 1>)
```

```
<qq-exp 0> ::= <exp>

<qq-exp n> ::= <symbol>
            |  <literal>
            |  <qq-exp n>*
            |  (quasiquote <qq-exp n+1>)
            |  (unquote <qq-exp n-1>)
            |  (unquote-splicing <qq-exp n-1>)

<body> ::= <top>* <exp>
```

# Output language

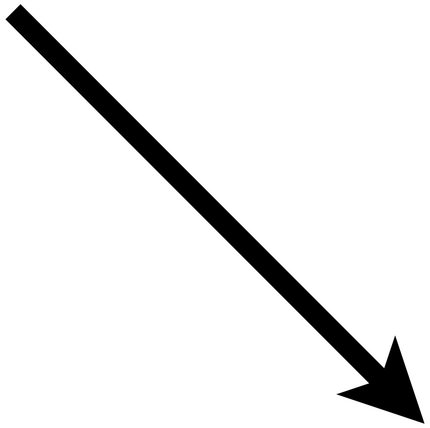
```
<prog> ::= <def>* <exp>*

<def> ::= (define <var> <exp>)

<exp> ::= <var>
       |  <literal>
       |  <prim>
       |  (quote <literal>)
       |  (lambda (<var>*) <exp>)
       |  (set! <var> <exp>)
       |  (if <exp> <exp> <exp>)
       |  (begin <exp>*)
       |  (<exp> <exp>*)
```

```
(define (desugar-program tops) ...)
(define (desugar-define def) ...)
(define (desugar-exp exp) ...)
(define (desugar-body body) ...)
(define (desugar-quote s-exp) ...)
(define (desugar-qq n qq-exp) ...)
```

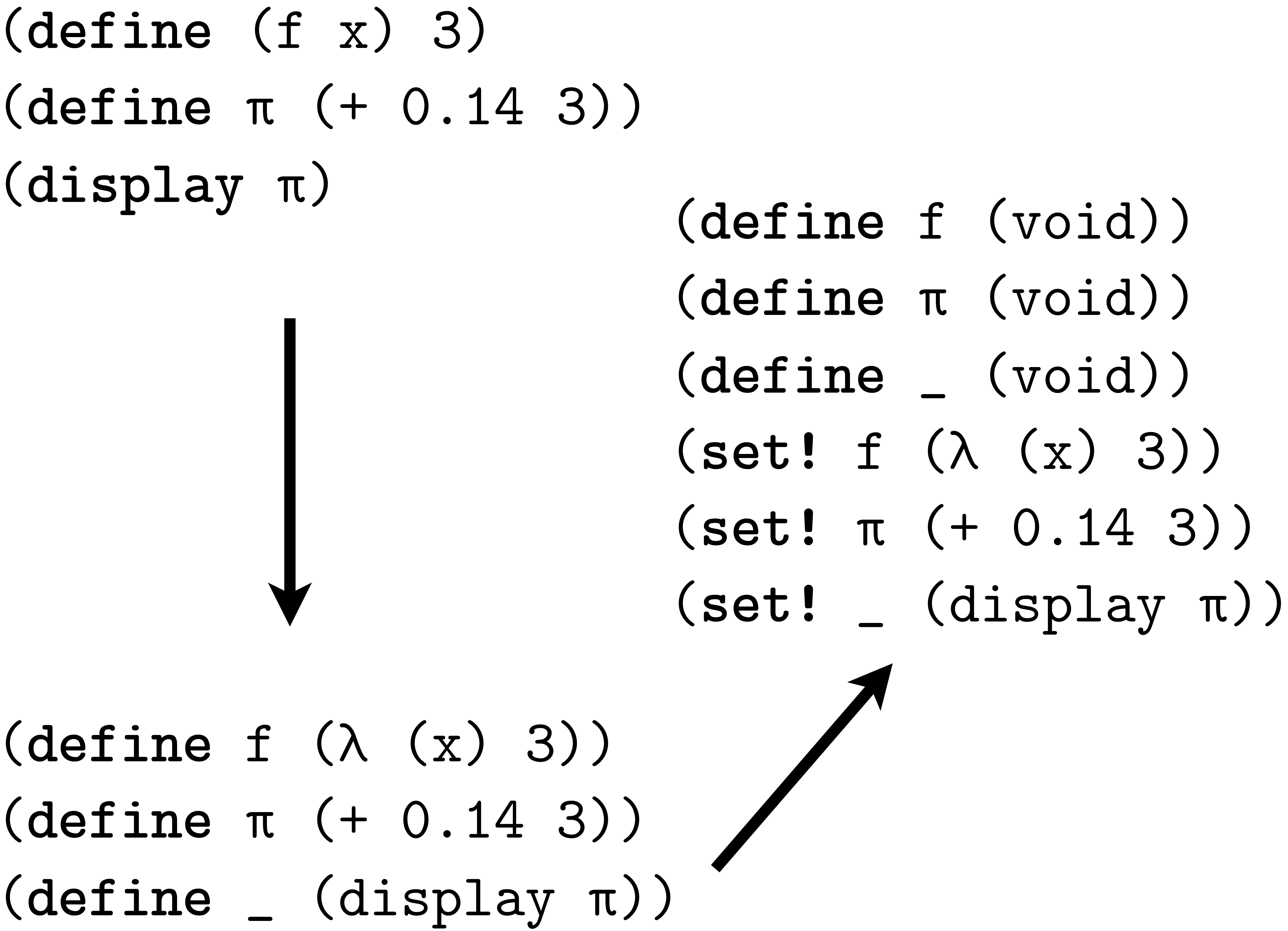
# Programs

```
(define (f x) 3)
(define π (+ 0.14 3))
(display π)
```

```
(define (f x) 3)
(define π (+ 0.14 3))
(display π)
```



```
(define f (λ (x) 3))
(define π (+ 0.14 3))
(define _ (display π))
```



```
(define (f x) 3)
(define π (+ 0.14 3))
(display π)
```

```
(define f (λ (x) 3))
(define π (+ 0.14 3))
(define _ (display π))
```

```
(define f (void))
(define π (void))
(define _ (void))
(set! f (λ (x) 3))
(set! π (+ 0.14 3))
(set! _ (display π))
```

```
; desugar : program -> program
(define (desugar-program prog)

  (set! prog (tops-to-defs prog))
  (set! prog (map desugar-define prog))
  (set! prog
    (partition-k
     atomic-define?
     prog
     (λ (atomic complex)
       (define bindings
         (for/list ([c complex])
           (match c
             [`(define ,v ,complex)
              `(,v (void))])))

       (define sets
         (for/list ([c complex])
           (match c
             [`(define ,v ,complex)
              `(set! ,v ,complex)])))

       (append atomic (list `(let ,bindings ,sets))))))

  prog)
```

```
; desugar : program -> program
(define (desugar-program prog)

  (set! prog (tops-to-defs prog))
  (set! prog (map desugar-define prog))
  (set! prog
    (partition-k
     atomic-define?
     prog
     (λ (atomic complex)
       (define bindings
         (for/list ([c complex])
           (match c
             [`(define ,v ,complex)
              `(,v (void))])))

       (define sets
         (for/list ([c complex])
           (match c
             [`(define ,v ,complex)
              `(set! ,v ,complex)])))

       (append atomic (list `(let ,bindings ,sets))))))

  prog)
```

```
; tops-to-defs : top list -> def list
(define (tops-to-defs tops)

  (define (top-to-def top)
    (match top
      [`(define (,f ,params ...) . ,body)
       `(define ,f (λ ,params . ,body))]

      [`(define ,v ,exp)
       `(define ,v ,exp)]

      [exp
       `(define ,(gensym '_) ,exp)]))

  (map top-to-def tops))
```

```
; desugar : program -> program
(define (desugar-program prog)

  (set! prog (tops-to-defs prog))
  (set! prog (map desugar-define prog))
  (set! prog
    (partition-k
     atomic-define?
     prog
     (λ (atomic complex)
       (define bindings
         (for/list ([c complex])
           (match c
             [`(define ,v ,complex)
              `(,v (void))])))

       (define sets
         (for/list ([c complex])
           (match c
             [`(define ,v ,complex)
              `(set! ,v ,complex)])))

       (append atomic (list `(let ,bindings ,sets))))))

  prog)
```

```
; desugar : program -> program
(define (desugar-program prog)

  (set! prog (tops-to-defs prog))
  (set! prog (map desugar-define prog))
  (set! prog
    (partition-k
     atomic-define?
     prog
     (λ (atomic complex)
       (define bindings
         (for/list ([c complex])
           (match c
             [`(define ,v ,complex)
              `(,v (void))])))

       (define sets
         (for/list ([c complex])
           (match c
             [`(define ,v ,complex)
              `(set! ,v ,complex)])))

       (append atomic (list `(let ,bindings ,sets))))))

  prog)
```

```
; desugar-define : define-term -> exp
(define (desugar-define def)
  (match def
    [`(define ,v ,exp)
     `(define ,v ,(desugar-exp exp))]

    [else
     (error (format "cannot desugar: ~s~n" def))]))
```

```
; desugar : program -> program
(define (desugar-program prog)

  (set! prog (tops-to-defs prog))
  (set! prog (map desugar-define prog))
  (set! prog
    (partition-k
     atomic-define?
     prog
     (λ (atomic complex)
       (define bindings
         (for/list ([c complex])
           (match c
             [`(define ,v ,complex)
              `(,v (void))])))

       (define sets
         (for/list ([c complex])
           (match c
             [`(define ,v ,complex)
              `(set! ,v ,complex)])))

       (append atomic (list `(let ,bindings ,sets))))))

  prog)
```

```
; desugar : program -> program
(define (desugar-program prog)

  (set! prog (tops-to-defs prog))
  (set! prog (map desugar-define prog))
  (set! prog
    (partition-k
     atomic-define?
     prog
     (λ (atomic complex)
       (define bindings
         (for/list ([c complex])
           (match c
             [`(define ,v ,complex)
              `(,v (void))])))

       (define sets
         (for/list ([c complex])
           (match c
             [`(define ,v ,complex)
              `(set! ,v ,complex)])))

       (append atomic (list `(let ,bindings ,sets))))))

  prog)
```

```
; atomic? : term -> boolean
(define (atomic? exp)
  (match exp
    [`(λ . ,_)      #t]
    [(? number?)    #t]
    [(? string?)    #t]
    [(? boolean?)   #t]
    [`(quote . ,_)  #t]
    ['(void)        #t]
    [else           #f]))

; atomic-define? : term -> boolean
(define (atomic-define? def)
  (match def
    [`(define ,v ,exp)  (atomic? exp)]
    [else               #f]))
```

# Expressions

```
; desugar-exp : exp -> exp
(define (desugar-exp exp)
  (match exp
    ...))
```

```
; desugar-exp : exp -> exp
(define (desugar-exp exp)
  (match exp
    [(? symbol?)                        exp]
    [`(quote ,s-exp)                    (desugar-quote s-exp)]

    ; binding forms:
    [`(let ((,vs ,es) ...) . ,body)     ...]
    [`(letrec ((,vs ,es) ...) . ,body)  ...]
    [`(λ ,params . ,body)               ...]

    ; conditionals:
    [`(cond)                            (void)]
    [`(cond (else ,exp))                ...]
    [`(cond (,test ,exp))               ...]
    [`(cond (,test ,exp) ,rest ...)     ...]

    [`(and)                             #t]
    [`(or)                              #f]
    [`(or ,exp)                        ...]
    [`(and ,exp)                       ...]
    [`(or ,exp . ,rest)                ...]
    [`(and ,exp . ,rest)               ...]

    [`(if ,test ,exp)                  ...]
    [`(if ,test ,exp1 ,exp2)           ...]

    ; mutation:
    [`(set! ,v ,exp)                   ...]

    ; quasiquotation:
    [`(quasiquote ,qq-exp)             (desugar-qq 1 qq-exp)]

    ; begins:
    [`(begin . ,body)                  (desugar-body body)]

    ; atoms:
    [(? atomic?)                       exp]

    ; function calls:
    [`(,f . ,args)                     ...]

    [else
     (printf "desugar fail: ~s~n" exp)
     exp]))
```

```
var => var
```

```
[(? symbol?)  exp]
```

```
(let ((v e) ...) body)
=>
((λ (v ...) body) e ...)
```

```
[`(let ((,vs ,es) ...) . ,body)
  ; =>
 `((λ ,vs ,(desugar-body body))
   ,@(map desugar-exp es))]
```

```
(let ((x 3)) (f x))
=>
((λ (x) (f x)) 3)
```

```
(letrec ((v e) ...) body)
=>
(let ((v (void)) ...)
  (set! v e) ...
  body))
```

```
[`(letrec ((,vs ,es) ...) . ,body)
  ; =>
 (desugar-exp
  `(let ,(for/list ([v vs])
           (list v '(void)))
     ,@(map (λ (v e)
              `(set! ,v ,e))
            vs es)
     ,@body))]
```

```
(letrec ((f (lambda (x) (g x)))
         (g (lambda (x) (+ x 1))))
 (f 20))
```

```
(let ((f (void))
      (g (void)))
  (set! f (lambda (x) (g x)))
  (set! g (lambda (x) (+ x 1)))
  (f 20))
```

```
(cond)
=>
(void)
```

```
(cond (else exp))
=>
exp
```

```
[`(cond (else ,exp))
  ; =>
  (desugar-exp exp)]
```

```
(cond (test exp) rest)
          =>
    (if test exp
      (cond rest))
```

```
[`(cond (,test ,exp) ,rest ...)
  ; =>
 `(if ,(desugar-exp test)
      ,(desugar-exp exp)
      ,(desugar-exp
         `(cond . ,rest)))]
```

```
(and)
=>
#t
```

```
[`(and)    #t]
```

```
(and e)
  =>
   e
```

```
[`(and ,exp)  (desugar-exp exp)]
```

(**and** $e_1$ $e_2$ ...)

=>

(**if** $e_1$ (**and** $e_2$ ...) #f)

```
[`(and ,exp . ,rest)
 `(if ,(desugar-exp exp)
      ,(desugar-exp
         `(and . ,rest))
      #f)]
```

```
(or)
=>
#f
```

```
[`(or)    #f]
```

(**or** *e*)

=>

*e*

```
[`(or ,e)    (desugar-exp e)]
```

```
(or e1 e2 ...)
=>
(let ([$t e1])
(if $t $t (or e2 ...))
```

```
[`(or ,exp . ,rest)
 (define $t (gensym 't))
 (desugar-exp
  `(let ((,$t ,exp))
     (if ,$t ,$t (or . ,rest))))]
```

# Questions?