

Original Grammar for Algebra Programs

```
<prog> ::= <defn>* <expr>
<defn> ::= <id>(<id>) = <expr>
<expr> ::= (<expr> + <expr>)
         ::= (<expr> - <expr>)
         ::= <id>(<expr>)
         ::= <id> | <num>
<id>   ::= a variable name: f, x, y, z, ...
<num>  ::= a number: 1, 42, 17, ...
```

- Change step-by-step to Scheme

Change #1: Move Prens and Add "define"

```
<prog> ::= <defn>* <expr>
<defn> ::= (define <id> <id>) <expr>)
<expr> ::= (+ <expr> <expr>)
         ::= (- <expr> <expr>)
         ::= <id> <expr>)
         ::= <id> | <num>
<id>   ::= a variable name: f, x, y, z, ...
<num>  ::= a number: 1, 42, 17, ...
```

- + and - moved to initial position

Change #1: Move Prens and Add "define"

```
<prog> ::= <defn>* <expr>
<defn> ::= (define <id> <id>) <expr>)
<expr> ::= (+ <expr> <expr>)
         ::= (- <expr> <expr>)
         ::= (<id> <expr>)
         ::= <id> | <num>
<id>   ::= a variable name: f, x, y, z, ...
<num>  ::= a number: 1, 42, 17, ...
```

- Open parenthesis for function call moved to before the function name

Change #1: Move Prens and Add "define"

```
<prog> ::= <defn>* <expr>
<defn> ::= (define (<id> <id>) <expr>)
<expr> ::= (+ <expr> <expr>)
         ::= (- <expr> <expr>)
         ::= (<id> <expr>)
         ::= <id> | <num>
<id>   ::= a variable name: f, x, y, z, ...
<num>  ::= a number: 1, 42, 17, ...
```

- Definition use the **define** keyword, and parenthesis moved

Change #1: Move Parens and Add "define"

```
<prog> ::= <defn>* <expr>
<defn> ::= (define (<id> <id>) <expr>)
<expr> ::= (+ <expr> <expr>)
        ::= (- <expr> <expr>)
        ::= (<id> <expr>)
        ::= <id> | <num>
<id>   ::= a variable name: f, x, y, z, ...
<num>  ::= a number: 1, 42, 17, ...
```

$f(x) = (x + 1)$
 $f((2 + 3))$  $(\text{define } (f\ x) (+\ x\ 1))$
 $(f\ (+\ 2\ 3))$

Change #2: Generalize + and -, Add Primitives

```
<expr> ::= ...
        ::= (+ <expr>* )
        ::= (- <expr>+ )
        ::= (* <expr>* )
        ::= (/ <expr>+ )
        ::= (modulo <expr> <expr>)
        ::= (expt <expr> <expr>)
        ::= ...
```

$(+ 1\ 2\ 3\ 7) \rightarrow 13$

Change #2: Generalize + and -, Add Primitives

```
<expr> ::= ...
        ::= (+ <expr>* )
        ::= (- <expr>+ )
        ::= (* <expr>* )
        ::= (/ <expr>+ )
        ::= (modulo <expr> <expr>)
        ::= (expt <expr> <expr>)
        ::= ...
```

$(+) \rightarrow 0$

Change #2: Generalize + and -, Add Primitives

```
<expr> ::= ...
        ::= (+ <expr>* )
        ::= (- <expr>+ )
        ::= (* <expr>* )
        ::= (/ <expr>+ )
        ::= (modulo <expr> <expr>)
        ::= (expt <expr> <expr>)
        ::= ...
```

$(- 1) \rightarrow -1$

Change #2: Generalize + and -, Add Primitives

```
<expr> ::= ...
        ::= (+ <expr>*)
        ::= (- <expr>+)
        ::= (* <expr>*)
        ::= (/ <expr>+)
        ::= (modulo <expr> <expr>)
        ::= (expt <expr> <expr>)
        ::= ...

        (*) → 1
```

Change #2: Generalize + and -, Add Primitives

```
<expr> ::= ...
        ::= (+ <expr>*)
        ::= (- <expr>+)
        ::= (* <expr>*)
        ::= (/ <expr>+)
        ::= (modulo <expr> <expr>)
        ::= (expt <expr> <expr>)
        ::= ...

        (/ 2) → 1/2
```

Change #2: Generalize + and -, Add Primitives

```
<expr> ::= ...
        ::= (+ <expr>*)
        ::= (- <expr>+)
        ::= (* <expr>*)
        ::= (/ <expr>+)
        ::= (modulo <expr> <expr>)
        ::= (expt <expr> <expr>)
        ::= ...
```

(expt 13 20) → 19004963774880799438801

Change #3: Generalize Defined Functions

```
<defn> ::= (define (<id> <id>*) <expr>)
<expr> ::= ...
        ::= (<id> <expr>*)
```

(define (f a b) (+ a b)) → (define (f a b) (+ a b))
(f 1 2) 3

Change #3: Generalize Defined Functions

```
<defn> ::= (define (<id> <id>*) <expr>)  
<expr> ::= ...  
        ::= (<id> <expr>*)
```

```
... (define (<id>0 <id>1...<id>k) <expr>a) ...
```

```
... (<id>0 <expr>1...<expr>k) ...
```

→

```
... (define (<id>0 <id>1...<id>k) <expr>a) ...
```

```
... <expr>b ...
```

where <expr>_b is <expr>_a with <id>_i replaced by <expr>_i

Change #4: Add Booleans

```
<expr> ::= ...  
        ::= <bool>  
        ::= (and <expr>*)  
        ::= (or <expr>*)  
        ::= (zero? <expr>)  
        ::= ...  
<bool> ::= #f  
        ::= #t
```

(and #t #f) → #f

Change #4: Add Booleans

```
<expr> ::= ...  
        ::= <bool>  
        ::= (and <expr>*)  
        ::= (or <expr>*)  
        ::= (zero? <expr>)  
        ::= ...  
<bool> ::= #f  
        ::= #t
```

(zero? 1) → #f

Change #4: Add Booleans

```
<expr> ::= ...  
        ::= <bool>  
        ::= (and <expr>*)  
        ::= (or <expr>*)  
        ::= (zero? <expr>)  
        ::= ...  
<bool> ::= #f  
        ::= #t
```

(zero? 0) → #t

Change #5: Add Symbols

```
<expr> ::= ...  
        ::= <symbol>  
        ::= (eq? <expr> <expr>)  
        ::= ...  
<symbol> ::= '<id>
```

(eq? 'a 'a) → #t

Change #5: Add Symbols

```
<expr> ::= ...  
        ::= <symbol>  
        ::= (eq? <expr> <expr>)  
        ::= ...  
<symbol> ::= '<id>
```

(eq? 'a 'b) → #f

Change #6: Add Conditionals

```
<expr> ::= ...  
        ::= (cond <cond-line>*)  
<cond-line> ::= [<expr> <expr>]  
              ::= [else <expr>]
```

(cond [#t 1]) → 1

Change #6: Add Conditionals

```
<expr> ::= ...  
        ::= (cond <cond-line>*)  
<cond-line> ::= [<expr> <expr>]  
              ::= [else <expr>]
```

(cond [#f 1] [#t 2]) → (cond [#t 2])

(cond [#t 2]) → 2

Change #6: Add Conditionals

```
<expr> ::= ...  
        ::= (cond <cond-line>*)  
<cond-line> ::= [<expr> <expr>]  
              ::= [else <expr>]
```

(cond [#t 1] [#t 2]) → 1

Change #6: Add Conditionals

```
<expr> ::= ...  
        ::= (cond <cond-line>*)  
<cond-line> ::= [<expr> <expr>]  
              ::= [else <expr>]
```

(cond [#f 1] [else 5]) → ... → 5

Change #6: Add Conditionals

```
<expr> ::= ...  
        ::= (cond <cond-line>*)  
<cond-line> ::= [<expr> <expr>]  
              ::= [else <expr>]
```

(cond [(zero? 5) 1] [else 2]) → (cond [#f 1] [else 2])

(cond [#f 1] [else 2]) → (cond [else 2]) → 2

Change #9: Values

While extending the set of expressions, we've also extended the set of **values**:

```
<val> ::= <num>  
       ::= <bool>  
       ::= <symbol>
```

- Evaluation stops when it reaches a member of <val>
- Function application requires a <val>

Change #9: Values

While extending the set of expressions, we've also extended the set of **values**:

```
<val> ::= <num>
       ::= <bool>
       ::= <symbol>
```

```
(define (f x) (+ x 1)) → (define (f x) (+ x 1))
(f (+ 1 2))           (f 3)
```

```
not (define (f x) (+ x 1))
    (+ (+ 1 2) 1)
```

Change #9: Values

While extending the set of expressions, we've also extended the set of **values**:

```
<val> ::= <num>
       ::= <bool>
       ::= <symbol>
```

```
... (define (<id>0 <id>1...<id>k) <expr>a) ...
... (<id>0 <val>1...<val>k) ...
```

→

```
... (define (<id>0 <id>1...<id>k) <expr>a) ...
... <expr>b ...
```

where <expr>_b is <expr>_a with <id>_i replaced by <val>_i

Change #8: Add Pairs

```
<expr> ::= ...
        ::= (cons <expr> <expr>)
        ::= (car <expr>)
        ::= (cdr <expr>)
<val>  ::= <num> | <bool> | <symbol>
        ::= (cons <val> <val>)
```

```
(cons 1 2) ∈ ? <val>
```

yes

Change #8: Add Pairs

```
<expr> ::= ...
        ::= (cons <expr> <expr>)
        ::= (car <expr>)
        ::= (cdr <expr>)
<val>  ::= <num> | <bool> | <symbol>
        ::= (cons <val> <val>)
```

```
(cons 1 (cons 2 (cons 3 4))) ∈ <val>
```

Change #8: Add Pairs

`<expr>` ::= ...
::= (**cons** `<expr>` `<expr>`)
::= (**car** `<expr>`)
::= (**cdr** `<expr>`)
`<val>` ::= `<num>` | `<bool>` | `<symbol>`
::= (**cons** `<val>` `<val>`)

`(cons 1 (+ 1 1))` ∈? `<val>`

no

`(cons 1 (+ 1 1))` → `(cons 1 2)`

Change #8: Add Pairs

`<expr>` ::= ...
::= (**cons** `<expr>` `<expr>`)
::= (**car** `<expr>`)
::= (**cdr** `<expr>`)
`<val>` ::= `<num>` | `<bool>` | `<symbol>`
::= (**cons** `<val>` `<val>`)

`(car (cons 1 2))` ∈? `<val>`

no

`(car (cons 1 2))` → 1

Change #8: Add Pairs

`<expr>` ::= ...
::= (**cons** `<expr>` `<expr>`)
::= (**car** `<expr>`)
::= (**cdr** `<expr>`)
`<val>` ::= `<num>` | `<bool>` | `<symbol>`
::= (**cons** `<val>` `<val>`)

More generally:

`(car (cons X Y))` → X

`(cdr (cons X Y))` → Y

Change #8: Add Pairs

`<expr>` ::= ...
::= (**cons** `<expr>` `<expr>`)
::= (**car** `<expr>`)
::= (**cdr** `<expr>`)
`<val>` ::= `<num>` | `<bool>` | `<symbol>`
::= (**cons** `<val>` `<val>`)

Also:

`(cadar (cons (cons X (cons Y Z)) W))` → Y

(or any combination of up to four "a"s and "d"s)

Change #9: Add the Empty List

$\langle \text{expr} \rangle ::= \dots$
 $\quad ::= '()$
 $\langle \text{val} \rangle ::= \langle \text{num} \rangle \mid \langle \text{bool} \rangle \mid \langle \text{symbol} \rangle$
 $\quad ::= (\text{cons } \langle \text{val} \rangle \langle \text{val} \rangle) \mid '()$
 $\langle \text{lst} \rangle ::= '()$
 $\quad ::= (\text{cons } \langle \text{val} \rangle \langle \text{lst} \rangle)$

$'() \in ? \langle \text{lst} \rangle$

yes

Change #9: Add the Empty List

$\langle \text{expr} \rangle ::= \dots$
 $\quad ::= '()$
 $\langle \text{val} \rangle ::= \langle \text{num} \rangle \mid \langle \text{bool} \rangle \mid \langle \text{symbol} \rangle$
 $\quad ::= (\text{cons } \langle \text{val} \rangle \langle \text{val} \rangle) \mid '()$
 $\langle \text{lst} \rangle ::= '()$
 $\quad ::= (\text{cons } \langle \text{val} \rangle \langle \text{lst} \rangle)$

$(\text{cons } 1 '()) \in ? \langle \text{lst} \rangle$

yes

Change #9: Add the Empty List

$\langle \text{expr} \rangle ::= \dots$
 $\quad ::= '()$
 $\langle \text{val} \rangle ::= \langle \text{num} \rangle \mid \langle \text{bool} \rangle \mid \langle \text{symbol} \rangle$
 $\quad ::= (\text{cons } \langle \text{val} \rangle \langle \text{val} \rangle) \mid '()$
 $\langle \text{lst} \rangle ::= '()$
 $\quad ::= (\text{cons } \langle \text{val} \rangle \langle \text{lst} \rangle)$

$(\text{cons } '() 1) \in ? \langle \text{lst} \rangle$

no

Change #9: Add the Empty List

$\langle \text{expr} \rangle ::= \dots$
 $\quad ::= '()$
 $\langle \text{val} \rangle ::= \langle \text{num} \rangle \mid \langle \text{bool} \rangle \mid \langle \text{symbol} \rangle$
 $\quad ::= (\text{cons } \langle \text{val} \rangle \langle \text{val} \rangle) \mid '()$
 $\langle \text{lst} \rangle ::= '()$
 $\quad ::= (\text{cons } \langle \text{val} \rangle \langle \text{lst} \rangle)$

$(\text{cons } 1 (\text{cons } 2 (\text{cons } 3 '())) \in ? \langle \text{lst} \rangle$

yes

Change #9: Add the Empty List

```
<expr> ::= ...
        ::= '()
<val>   ::= <num> | <bool> | <symbol>
        ::= (cons <val> <val>) | '()
<lst>   ::= '()
        ::= (cons <val> <lst>)
```

Is every <lst> a <val> ?

yes

List Shortcuts

- $(\text{list } 1\ 2\ 3) = (\text{cons } 1\ (\text{cons } 2\ (\text{cons } 3\ '())))$
- $'(1\ 2\ 3) = (\text{cons } 1\ (\text{cons } 2\ (\text{cons } 3\ '())))$
- $'(a\ b\ c) = (\text{cons } 'a\ (\text{cons } 'b\ (\text{cons } 'c\ '())))$
- $'(a\ (1\ 2)\ c)$
= $(\text{list } 'a\ '(1\ 2)\ 'c)$
= $(\text{cons } 'a\ (\text{cons } (\text{cons } 1\ (\text{cons } 2\ '()))\ (\text{cons } 'c\ '())))$

List Predicates

- $(\text{null? } '()) = \#t$
- $(\text{null? } (\text{cons } 1\ 2)) = \#f$
- $(\text{pair? } '()) = \#f$
- $(\text{pair? } (\text{cons } 1\ 2)) = \#t$
- $(\text{list? } '()) = \#t$
- $(\text{list? } (\text{cons } 1\ 2)) = \#f$
- $(\text{list? } (\text{cons } 1\ '())) = \#t$

Change #10: Functions Are Values

```
<val> ::= ...
       ::= <defined-id>
<defined-id> ::= an <id> that is defined
```

```
(define (twice f x) (f (f x))) → (define (twice f x) (f (f x)))
(define (g y) (+ y 1))          (define (g y) (+ y 1))
(twice g 0)                     (g (g 0))
```

Change #10: Functions Are Values

<val> ::= ...
 ::= <defined-id>
<defined-id> ::= an <id> that is **defined**

(define (twice f x) (f (f x))) → (define (twice f x) (f (f x)))
(define (g y) (+ y 1)) (define (g y) (+ y 1))
(g (g 0)) (g (+ 0 1))

Change #10: Functions Are Values

<val> ::= ...
 ::= <defined-id>
<defined-id> ::= an <id> that is **defined**

(define (twice f x) (f (f x))) → (define (twice f x) (f (f x)))
(define (g y) (+ y 1)) (define (g y) (+ y 1))
(g (+ 0 1)) (g 1)

Change #10: Functions Are Values

<val> ::= ...
 ::= <defined-id>
<defined-id> ::= an <id> that is **defined**

(define (twice f x) (f (f x))) → (define (twice f x) (f (f x)))
(define (g y) (+ y 1)) (define (g y) (+ y 1))
(g 1) (+ 1 1)

Change #10: Functions Are Values

<val> ::= ...
 ::= <defined-id>
<defined-id> ::= an <id> that is **defined**

(define (twice f x) (f (f x))) → (define (twice f x) (f (f x)))
(define (g y) (+ y 1)) (define (g y) (+ y 1))
(+ 1 1) 2

Alternate Notations (Used by the Book)

- **(define <id>₀ (lambda (<id>₁...<id>_k) <expr>))**
= **(define (<id>₀ <id>₁...<id>_k) <expr>)**
- **(if <expr>₁ <expr>₂ <expr>₃)**
= **(cond [<expr>₁ <expr>₂] [else <expr>₃])**

Stuff You Don't Need (Yet)

- local bindings
- **set!**
- **begin** (including implicit)
- I/O primitives
- vectors
- **do** (never need this one)