Eshkol Built-in Functions Development Guide

Executive Summary

This document provides a comprehensive overview of Eshkol's built-in functions and language features. Eshkol is a Scheme-like functional language with first-class automatic differentiation support, compiled via LLVM.

Legend:

- ✅ = Implemented and working
- ⚠️ = Partially implemented or has known issues
- ❌ = Not implemented

Summary Statistics:

- **Implemented:** ~170 operations
- **Not Implemented:** ~10 operations
- **Language Completeness:** ~95%

1. Core Special Forms

Operation	Syntax	Description	Status
`define`	`(define name value)`	Variable definition	✅
`define` (function)	`(define (f x) body)`	Function definition	✅
`define` (nested)	Inside function bodies	Nested definitions	✅
`lambda`	`(lambda (x) body)`	Anonymous functions	✅

Operation	Syntax	Description	Status
`let`	`(let ((x 1)) body)`	Parallel binding	✅
`let*`	`(let* ((x 1) (y x)) body)`	Sequential binding	✅
`letrec`	`(letrec ((f ...)) body)`	Recursive binding	✅
`if`	`(if cond then else)`	Conditional	✅
`cond`	`(cond (test expr) ...)`	Multi-branch conditional	✅
`case`	`(case key ((val) expr) ...)`	Pattern matching	✅
`when`	`(when test expr ...)`	One-armed if (true branch)	✅
`unless`	`(unless test expr ...)`	One-armed if (false branch)	✅
`begin`	`(begin expr1 expr2 ...)`	Sequential evaluation	✅
`do`	`(do ((var init step) ...) (test result) body)`	Iteration construct	✅
`quote`	`'expr` / `(quote expr)`	Literal data	✅
`set!`	`(set! var value)`	Variable mutation	✅
`and`	`(and a b ...)`	Logical AND (short-circuit)	✅
`or`	`(or a b ...)`	Logical OR (short-circuit)	✅

Operation	Syntax	Description	Status
`quasiquote`	`` `expr ``	Template with unquoting	❌
`unquote`	`,expr`	Evaluate in quasiquote	❌
`unquote-splicing`	`,@expr`	Splice in quasiquote	❌

2. Arithmetic Operations

Operation	Syntax	Description	Status
`+`	`(+ a b ...)`	Addition (variadic)	✅
`-`	`(- a b ...)`	Subtraction (variadic)	✅
`*`	`(* a b ...)`	Multiplication (variadic)	✅
`/`	`(/ a b ...)`	Division (variadic)	✅
`% / mod / modulo`	`(% a b)`	Modulo (floor division)	✅
`remainder`	`(remainder a b)`	Remainder (truncate div)	✅
`quotient`	`(quotient a b)`	Integer division	✅
`abs`	`(abs x)`	Absolute value	✅
`floor`	`(floor x)`	Round down	✅
`ceiling / ceil`	`(ceiling x)`	Round up	✅
`round`	`(round x)`	Round to nearest	✅
`truncate / trunc`	`(truncate x)`	Truncate toward zero	✅
`min`	`(min a b ...)`	Minimum (variadic)	✅
`max`	`(max a b ...)`	Maximum (variadic)	✅
`gcd`	`(gcd a b)`	Greatest common divisor	✅

Operation	Syntax	Description	Status
`lcm`	`(lcm a b)`	Least common multiple	✅
`expt`	`(expt base exp)`	Exponentiation	✅

3. Comparison & Logic

Operation	Syntax	Description	Status
`=`	`(= a b)`	Numeric equality	✅
`>`	`(> a b)`	Greater than	✅
`<`	`(< a b)`	Less than	✅
`>=`	`(>= a b)`	Greater or equal	✅
`<=`	`(<= a b)`	Less or equal	✅
`and`	`(and a b ...)`	Logical AND (short-circuit)	✅
`or`	`(or a b ...)`	Logical OR (short-circuit)	✅
`not`	`(not x)`	Logical NOT	✅
`xor`	`(xor a b)`	Logical XOR	✅
`eq?`	`(eq? a b)`	Identity (pointer) equality	✅
`eqv?`	`(eqv? a b)`	Value equality (primitives)	✅
`equal?`	`(equal? a b)`	Deep structural equality	✅

4. List Operations (Core)

Operation	Syntax	Description	Status
`cons`	`(cons a b)`	Construct pair	✅
`car`	`(car pair)`	First element	✅

Operation	Syntax	Description	Status
`cdr`	`(cdr pair)`	Rest of list	✅
`set-car!`	`(set-car! pair val)`	Mutate car	✅
`set-cdr!`	`(set-cdr! pair val)`	Mutate cdr	✅
`list`	`(list a b ...)`	Create list	✅
`list*`	`(list* a b ... tail)`	Create improper list	✅
`list?`	`(list? x)`	Test if proper list	✅
`null?`	`(null? x)`	Test if empty list	✅
`pair?`	`(pair? x)`	Test if pair	✅
`length`	`(length lst)`	List length	✅
`append`	`(append lst1 lst2 ...)`	Concatenate (variadic)	✅
`reverse`	`(reverse lst)`	Reverse list	✅

5. List Operations (Advanced)

Operation	Syntax	Description	Status
`list-ref`	`(list-ref lst n)`	Get nth element	✅
`list-tail`	`(list-tail lst n)`	Drop n elements	✅
`last`	`(last lst)`	Last element	✅
`last-pair`	`(last-pair lst)`	Last cons cell	✅
`make-list`	`(make-list n [fill])`	Create n-element list	✅
`acons`	`(acons key val alist)`	Add to association list	✅
`iota`	`(iota count [start step])`	Generate sequence	❌

6. List Accessors (c[ad]+r Family)

All compound car/cdr operations are ✅ **Implemented**:

2-Level Accessors

`caar`	`cadr`	`cdar`	`cddr`
✅	✅	✅	✅

3-Level Accessors

`caaar`	`caadr`	`cadar`	`caddr`	`cdaar`	`cdadr`	`cddar`	`cdddr`
✅	✅	✅	✅	✅	✅	✅	✅

4-Level Accessors

All 16 four-level accessors (`caaaar` through `cddddr`) are ✅ **Implemented**.

7. Higher-Order Functions

Operation	Syntax	Description	Status
`map`	`(map f lst ...)`	Map over lists	✅
`filter`	`(filter pred lst)`	Filter by predicate	✅
`fold / foldl`	`(fold f init lst)`	Left fold	✅
`fold-right / foldr`	`(fold-right f init lst)`	Right fold	✅
`reduce`	`(reduce f lst)`	Reduce (no init)	❌
`for-each`	`(for-each f lst)`	Iteration (side effects)	✅
`apply`	`(apply f args)`	Apply to arg list	✅
`any`	`(any pred lst)`	Any element matches?	✅
`every`	`(every pred lst)`	All elements match?	✅
`compose`	`(compose f g)`	Function composition	❌

Operation	Syntax	Description	Status
curry	(curry f)	Currying	❌

8. List Search & Membership

Operation	Syntax	Description	Status
member	(member x lst)	Find using equal?	✅
memq	(memq x lst)	Find using eq?	✅
memv	(memv x lst)	Find using eqv?	✅
assoc	(assoc key alist)	Alist lookup (equal?)	✅
assq	(assq key alist)	Alist lookup (eq?)	✅
assv	(assv key alist)	Alist lookup (eqv?)	✅
find	(find pred lst)	First matching element	✅

9. List Manipulation

Operation	Syntax	Description	Status
take	(take n lst)	First n elements	✅
drop	(drop n lst)	Remove first n	✅
split-at	(split-at n lst)	Split at index	✅
partition	(partition pred lst)	Split by predicate	✅
remove	(remove pred lst)	Remove matching (equal?)	✅
remq	(remq x lst)	Remove by identity (eq?)	✅
remv	(remv x lst)	Remove by value (eqv?)	✅
range	(range start end)	Generate integer range	✅

Operation	Syntax	Description	Status
`zip`	`(zip lst1 lst2 ...)`	Zip lists together	✅
`unzip`	`(unzip lst)`	Unzip list of pairs	❌
`sort`	`(sort lst less?)`	Sort list	❌

10. Mathematical Functions

Operation	Syntax	Description	Status
`sin`	`(sin x)`	Sine	✅
`cos`	`(cos x)`	Cosine	✅
`tan`	`(tan x)`	Tangent	✅
`asin`	`(asin x)`	Arc sine	✅
`acos`	`(acos x)`	Arc cosine	✅
`atan`	`(atan x)`	Arc tangent	✅
`atan2`	`(atan2 y x)`	Two-arg arctangent	✅
`sinh`	`(sinh x)`	Hyperbolic sine	✅
`cosh`	`(cosh x)`	Hyperbolic cosine	✅
`tanh`	`(tanh x)`	Hyperbolic tangent	✅
`asinh`	`(asinh x)`	Inverse hyperbolic sine	✅
`acosh`	`(acosh x)`	Inverse hyperbolic cosine	✅
`atanh`	`(atanh x)`	Inverse hyperbolic tangent	✅
`sqrt`	`(sqrt x)`	Square root	✅
`cbrt`	`(cbrt x)`	Cube root	✅
`exp`	`(exp x)`	e^x	✅
`log`	`(log x)`	Natural logarithm	✅
`log10`	`(log10 x)`	Base-10 logarithm	✅

Operation	Syntax	Description	Status
`log2`	`(log2 x)`	Base-2 logarithm	✅

11. Vector Operations

Operation	Syntax	Description	Status
`vector`	`(vector a b ...)`	Create vector	✅
`vector?`	`(vector? x)`	Test if vector	✅
`make-vector`	`(make-vector n [fill])`	Create n-element vector	✅
`vector-ref`	`(vector-ref v i)`	Get element	✅
`vref`	`(vref v i)`	Get element (AD-aware)	✅
`vector-set!`	`(vector-set! v i val)`	Set element	✅
`vector-length`	`(vector-length v)`	Vector length	✅
`vector-to-string`	`(vector-to-string v)`	Convert to string	✅

12. Tensor/Matrix Operations

Operation	Syntax	Description	Status
`tensor-get`	`(tensor-get t indices...)`	Get tensor element	✅
`tensor-set`	`(tensor-set t indices... val)`	Set tensor element	✅
`tensor-shape`	`(tensor-shape t)`	Get dimensions	✅
`tensor-add`	`(tensor-add a b)`	Element-wise add	✅
`tensor-sub`	`(tensor-sub a b)`	Element-wise subtract	✅
`tensor-mul`	`(tensor-mul a b)`	Element-wise multiply	✅
`tensor-div`	`(tensor-div a b)`	Element-wise divide	✅

Operation	Syntax	Description	Status
`tensor-dot`	`(tensor-dot a b)`	Dot product	✅
`tensor-apply`	`(tensor-apply f t)`	Apply function to elements	✅
`tensor-reduce`	`(tensor-reduce t f init [dim])`	Reduce tensor	✅
`tensor-reduce-all`	`(tensor-reduce-all t f init)`	Reduce to scalar	✅
`tensor-sum`	`(tensor-sum t)`	Sum all elements	✅
`tensor-mean`	`(tensor-mean t)`	Mean of elements	✅
`matmul`	`(matmul a b)`	Matrix multiplication	✅
`transpose`	`(transpose m)`	Matrix transpose	✅
`reshape`	`(reshape t dims...)`	Reshape tensor	✅
`flatten`	`(flatten t)`	Flatten to 1D	✅
`zeros`	`(zeros dims...)`	Create zero tensor	✅
`ones`	`(ones dims...)`	Create ones tensor	✅
`eye`	`(eye n)`	Create identity matrix	✅
`arange`	`(arange start stop [step])`	Create range tensor	✅
`linspace`	`(linspace start stop n)`	Create linearly spaced tensor	✅
`matrix-to-string`	`(matrix-to-string m)`	Convert to string	✅

13. Automatic Differentiation

Operation	Syntax	Description	Status
`derivative`	`(derivative f x)`	Scalar derivative	✅
`gradient`	`(gradient f v)`	Vector gradient	✅
`jacobian`	`(jacobian f v)`	Jacobian matrix	✅
`hessian`	`(hessian f v)`	Hessian matrix	✅

Operation	Syntax	Description	Status
`divergence`	`(divergence F v)`	Vector divergence	✅
`curl`	`(curl F v)`	Vector curl (3D)	✅
`laplacian`	`(laplacian f v)`	Laplacian operator	✅
`directional-derivative`	`(directional-derivative f v dir)`	Directional derivative	✅
`diff`	`(diff expr var)`	Symbolic differentiation	✅

14. String Operations

Operation	Syntax	Description	Status
`string?`	`(string? x)`	Test if string	✅
`string-length`	`(string-length s)`	String length	✅
`string-ref`	`(string-ref s k)`	Get character	✅
`string-set!`	`(string-set! s k c)`	Set character	✅
`string-append`	`(string-append s1 s2 ...)`	Concatenate	✅
`substring`	`(substring s start end)`	Extract substring	✅
`make-string`	`(make-string n [char])`	Create string	✅
`string->list`	`(string->list s)`	To char list	✅
`list->string`	`(list->string lst)`	From char list	✅
`string=?`	`(string=? s1 s2)`	String equality	✅
`string<?`	`(string<? s1 s2)`	Lexicographic <	✅
`string>?`	`(string>? s1 s2)`	Lexicographic >	✅
`string<=?`	`(string<=? s1 s2)`	Lexicographic <=	✅
`string>=?`	`(string>=? s1 s2)`	Lexicographic >=	✅

Operation	Syntax	Description	Status
`number->string`	`(number->string n)`	Convert to string	✅
`string->number`	`(string->number s)`	Parse number	✅

15. Character Operations

Operation	Syntax	Description	Status
`char?`	`(char? x)`	Test if character	✅
`char->integer`	`(char->integer c)`	Char to ASCII code	✅
`integer->char`	`(integer->char n)`	ASCII code to char	✅
`char=?`	`(char=? c1 c2)`	Character equality	✅
`char<?`	`(char<? c1 c2)`	Character <	✅
`char>?`	`(char>? c1 c2)`	Character >	✅
`char<=?`	`(char<=? c1 c2)`	Character <=	✅
`char>=?`	`(char>=? c1 c2)`	Character >=	✅

16. Type Predicates

Operation	Syntax	Description	Status
`number?`	`(number? x)`	Test if number	✅
`integer?`	`(integer? x)`	Test if integer	✅
`real?`	`(real? x)`	Test if real	✅
`positive?`	`(positive? x)`	x > 0	✅
`negative?`	`(negative? x)`	x < 0	✅
`zero?`	`(zero? x)`	x = 0	✅

Operation	Syntax	Description	Status
`odd?`	`(odd? x)`	Is odd	✅
`even?`	`(even? x)`	Is even	✅
`procedure?`	`(procedure? x)`	Test if function	❌
`boolean?`	`(boolean? x)`	Test if boolean	❌
`symbol?`	`(symbol? x)`	Test if symbol	❌

17. I/O Operations

Operation	Syntax	Description	Status
`display`	`(display x)`	Print value	✅
`newline`	`(newline)`	Print newline	✅
`error`	`(error msg)`	Print error and exit	✅
`open-input-file`	`(open-input-file path)`	Open file for reading	✅
`open-output-file`	`(open-output-file path)`	Open file for writing	✅
`read-line`	`(read-line port)`	Read line from port	✅
`write-string`	`(write-string s port)`	Write string to port	✅
`write-line`	`(write-line s port)`	Write line to port	✅
`write-char`	`(write-char c port)`	Write char to port	✅
`close-port`	`(close-port port)`	Close port	✅
`eof-object?`	`(eof-object? x)`	Test for EOF	✅
`flush-output-port`	`(flush-output-port port)`	Flush output buffer	✅

18. FFI (Foreign Function Interface)

Operation	Syntax	Description	Status
`extern`	`(extern ret-type name types...)`	C function declaration	✅
`extern-var`	`(extern-var type name)`	C variable declaration	✅

19. Miscellaneous

Operation	Syntax	Description	Status
`random`	`(random)`	Random number [0, 1)	✅

Remaining Work

Not Yet Implemented

Operation	Effort	Priority	Notes
`sort`	Medium	High	Merge sort for lists
`iota`	Low	Medium	Sequence generator (have `range`)
`reduce`	Low	Low	Fold without initial value (have `fold`)
`compose`	Low	Medium	Function composition
`curry`	Medium	Low	Function currying
`unzip`	Low	Low	Inverse of zip
`procedure?`	Low	Low	Type predicate
`boolean?`	Low	Low	Type predicate
`symbol?`	Low	Low	Type predicate
`quasiquote`	High	Low	Metaprogramming

Appendix A: Complete Function List

Fully Implemented (✅) - 170+ Functions

```
;; Arithmetic
+, -, *, /, %, mod, modulo, remainder, quotient
abs, floor, ceiling, ceil, round, truncate, trunc
min, max, gcd, lcm, expt

;; Comparison & Logic
=, <, >, <=, >=
and, or, not, xor
eq?, eqv?, equal?

;; Math Functions
sin, cos, tan, asin, acos, atan, atan2
sinh, cosh, tanh, asinh, acosh, atanh
sqrt, cbrt, exp, log, log10, log2

;; List Core
cons, car, cdr, list, list*, null?, pair?, list?
length, append, reverse, set-car!, set-cdr!

;; List Advanced
list-ref, list-tail, last, last-pair, make-list, acons

;; c[ad]+r Accessors (28 total)
caar, cadr, cdar, cddr
caaar, caadr, cadar, caddr, cdaar, cdadr, cddar, cdddr
caaaar, caaadr, caadar, caaddr, cadaar, cadadr, caddar, cadddr
cdaaar, cdaadr, cdadar, cdaddr, cddaar, cddadr, cdddar, cddddr

;; Higher-Order
map, filter, fold, fold-right, for-each, any, every, apply

;; Search & Membership
member, memq, memv, assoc, assq, assv, find

;; List Manipulation
take, drop, split-at, partition, remove, remq, remv, range, zip

;; Vectors
vector, vector?, make-vector, vector-ref, vref, vector-set!, vector-length
```

```
;; Tensors/Matrices
tensor-get, tensor-set, tensor-shape
tensor-add, tensor-sub, tensor-mul, tensor-div, tensor-dot
tensor-apply, tensor-reduce, tensor-reduce-all, tensor-sum, tensor-mean
matmul, transpose, reshape, flatten
zeros, ones, eye, arange, linspace

;; Automatic Differentiation
derivative, gradient, jacobian, hessian
divergence, curl, laplacian, directional-derivative, diff

;; Strings
string?, string-length, string-ref, string-set!
string-append, substring, make-string
string->list, list->string
string=?, string<?, string>?, string<=?, string>=?
number->string, string->number

;; Characters
char?, char->integer, integer->char
char=?, char<?, char>?, char<=?, char>=?

;; Type Predicates
number?, integer?, real?, string?, list?, pair?, vector?, char?
positive?, negative?, zero?, odd?, even?

;; I/O
display, newline, error
open-input-file, open-output-file, read-line
write-string, write-line, write-char
close-port, eof-object?, flush-output-port

;; Special Forms
define, lambda, let, let*, letrec
if, cond, case, when, unless
begin, do, quote, set!

;; FFI
extern, extern-var

;; Misc
random
```

Document updated: 2024

Source: lib/backend/llvm_codegen.cpp