Honors Project Proposal
CSC1061

Jordan Jacobson

January 21, 2026

Project Scope

Implement a typed Lisp compiler and VM for Sprout. The project focuses on a purely functional, lazy-by-default core, a System F type system, static compilation, and delimited continuations as the central control structure, alongside advanced effects/handlers.

The core implementation will include a reader/parser for the surface syntax, a typechecker for explicit System F style types, a bytecode compiler, and a stack-based VM that supports lazy evaluation with memoized thunks, all implemented in C++. The runtime will use a basic reference counting system with a mark-sweep collector for cyclic structures. A REPL or interpreted mode may run on top of the VM.

A listing of final language features, build and usage instructions, and a set of example programs will be provided. All source code and supplementary materials will be available in the sprout repository at https://github.com/unreasonablyEffective42/sprout

Minimum Functionality

- Core data types: Int, Float, Rational, Complex, Bool, String
  `42  3.14  5/7  2+3i  #t  "hi"`
- Symbols/identifiers and lists
  `foo  (list 1 2 3)  '(a b c)`
- Lambda, define, and application
  `(define inc (lambda ((x:Int)->Int) (+ x 1)))  (inc 5)`
- Typed let/lets/letr bindings with required annotations
  `(let ((x:Int 10)) (+ x 2))`
- Equality: eq? (identity) and equal? (structural)
  `(eq?  x y)  (equal?  '(1 2) '(1 2))`
- Reader and parser for the surface syntax (including reader macros for quote, quasiquote, unquote, unquote-splicing)
  `'x  `(a ,b ,@c)`

- Lazy evaluation with sharing (call-by-need)
  `(let ((x:Int (expensive))) (do x x))`
- Force and do forms with the specified evaluation behavior
  `(do (force x) (force y) z)`
- System F constructs: forall types, tlambda, tapply
  `(tlambda (A) (lambda ((x:A)->A) x))  (tapply id Int)`
- ADT declarations and constructors (data)
  `(data Maybe (A) (Nothing) (Just (A)))  (Just 3)`
- Pattern matching with list patterns and dotted tails
  `(match xs ((x y .  rest) x) (else 0))`
- Errors: error values, raise, try/catch sugar
  `(try (raise (error "IO" "fail")) (catch (e) e))`
- Delimited control: shift/reset with one-shot continuations
  `(reset (+ 1 (shift k (k 2))))`
- Effects/handlers: perform, handle, return clause
  `(handle (perform Op 1) (Op (x) k (k x)) (return (r) r))`
- Placeholder partial application using _
  `(define foo (lambda ((a:Int b:Int)->Int) (+ a b)))`
  `(let ((bar:(Int->Int) (foo 10 _))) (bar 5))`
- REPL and/or file execution
  `sprout repl  sprout run examples/demo.spr`

Stretch Goals

- Optimizer passes (inlining, strictness/forcing analysis)
- Expanded standard library and examples
- Better error reporting with source spans
- Testing harness for typechecking and runtime behaviors