LISP-FLAVORED LOGIC: A TOY S-EXPRESSION REPL EVALUATING PROPOSITIONAL LOGIC WFFS

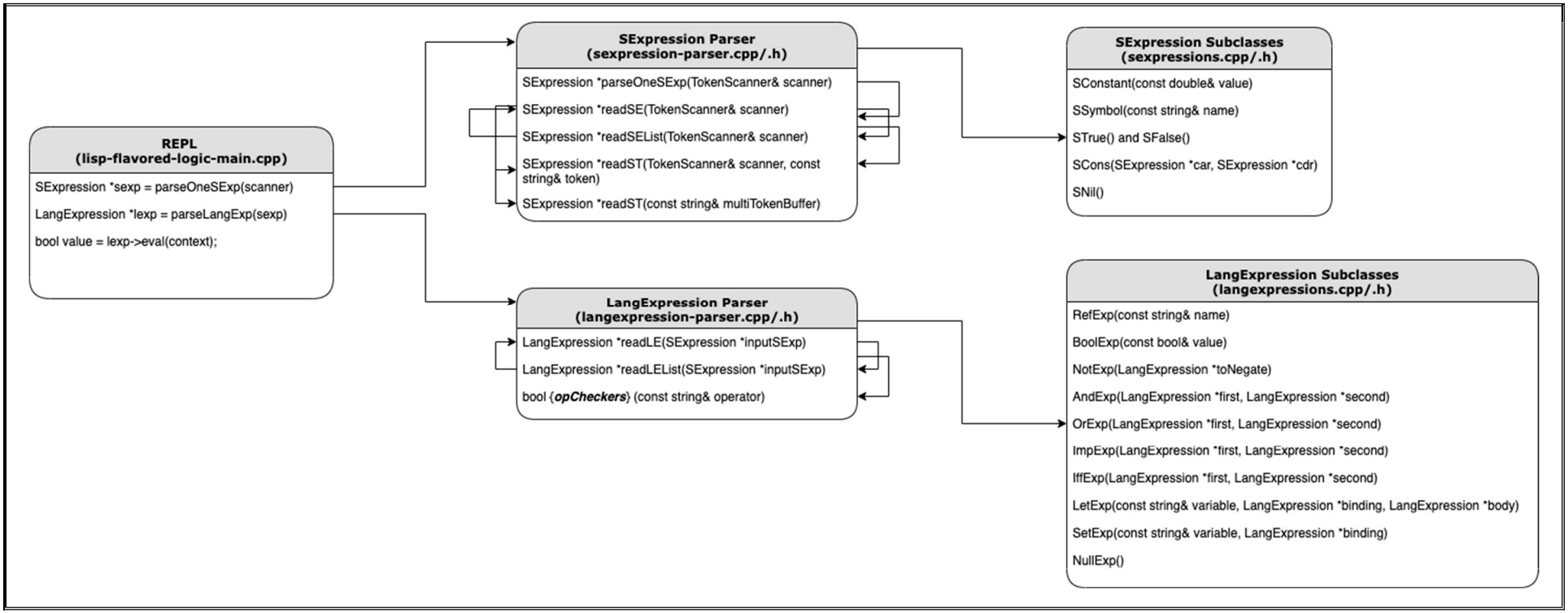
Josh Nkoy '21 (BS Symbolic Systems, Minor CS)

IN A NUTSHELL: Basically, my project cannibalizes the Expression class from lectures into two new expression classes for S-expression parsing and language syntax evaluation of the generated S-expressions in terms of propositional logic operators and both global and local bindings.

INSPIRATIONS:

- Previous experience learning Common Lisp and Clojure, as well as countless articles on Lisp-flavoring other languages (most notably Erlang but also Haskell, Go, and countless other smaller projects)
- An obsession with formal logic and Prolog (more in **FUTURE PLANS**)

PROJECT STRUCTURE:



S-EXPRESSION BNF GRAMMAR:

- <file> ::= <s-exp-list>
- <s-exp> ::= <atom>| '(' <s-exp-list> ')'
- <s-exp-list> ::= <sexp> <s-exp-list>
- <atom> ::= <symbol>|<integer> | #t | #f

(Courtesy of Matt Might's "Parsing S-Expressions in Scala" article)

LOGIC EXPRESSIONS BY PATTERN MATCHING[-ISH]:

SExpression Form	LangExpression Form
STrue(), SConstant(1.0)	BoolExp(true)
SFalse(), SConstant(0.0)	BoolExp(false)
SSymbol(name)	RefExp(name)
SCons(SCons(SSymbol(not/N/~/[-]/!)), <sexp>)	NotExp(parse(<sexp>))
SCons(SCons(SSymbol(and/K/&/[*])), <sexp1>, <sexp2>)	AndExp(parse(<sexp1>), parse(<sexp2>))
SCons(SCons(SSymbol(or/A/||/[+])), <sexp1>, <sexp2>)	OrExp(parse(<sexp1>), parse(<sexp2>))
SCons(SCons(SSymbol(implies/imp/C/=>)), <sexp1>, <sexp2>)	ImpExp(parse(<sexp1>), parse(<sexp2>))
SCons(SCons(SSymbol(iff/E/<=>)), <sexp1>, <sexp2>)	IffExp(parse(<sexp1>), parse(<sexp2>))
SCons(SCons(SSymbol(set)), SSymbol(name), <bindingSExp>)	SetExp(name, parse(<bindingSExp>))
SCons(SCons(SSymbol(let)), SSymbol(name), <bindingSExp>, <bodySExp>)	LetExp(name, parse(<bindingSExp>), parse(<bodySExp>))

CAPABILITIES: Can evaluate any well-formed formula (WFF) in the language of propositional logic (L^{bool}), utilizing the set of [Polish notation] operators {N, K, A, C, E}, with added capabilities for local bindings in the form of *let* statements and global bindings in the form of *set* statements.

FUTURE PLANS:

- Add abilities to convert L^{bool} WFFs into conjunctive and disjunctive normal forms, as well as analyzing their validity or satisfiability
- Add more Prolog-like first-order logic capabilities with SLD resolution of provided rules and facts, and Skolemization of WFFs in the language of first order logic (L^{FOL})
- Translate from C++ to a functional language with powerful pattern matching, like Haskell or F# or Scala (in order of my proficiency)

REFERENCES:

- Matt Might, "Parsing S-Expressions in Scala," matt.might.net/articles/parsing-s-expressions-scala
- Eric Roberts, *Programming Abstractions in C++*.