\[L^F\] is a bare-bones language with all the functionality of fully-fledged programming languages, but no paraphernalia. It has lists and functions only—hence the name: function-powered lists. It treats all values as good, as long as they can be sensibly interpreted in a given environment. Functions are first-class citizens, which means you can pass them as arguments, like other values. For the sake of simplicity, it does not return functions as values. (The last point is a nice bonus for the FP enthusiasts among you.)

The design philosophy of \[L^F\] is POLSWALCS (pronounced “Paul’s Walks”): The Principle Of Least Surprise, With A Little Concern for Students.

1 LEXICAL ASPECTS

**Keywords:** \[L^F\] has keywords and key constants, which cannot be redefined.\(^1\)

**Identifiers:** An identifier is a sequence of letters, digits, and underscores, starting with a non-digit. Uppercase letters are distinguished from lowercase. For the compiler, the significant length of an identifier is 16.

**Comments:** Anything after a # till the end of line is considered a comment.

**Integer constant:** A sequence of decimal digits is an integer constant that denotes the corresponding integer value.

**Float constant:** A sequence of decimal digits followed by a fractional part which consists of a period and a sequence of decimal digits is a float constant that denotes the corresponding float value.

**String constant:** A string constant is a sequence, between quotes (e.g. "this"), of zero or more printable characters, spaces, or escape sequences. Each escape sequence is introduced by the escape character \, and stands for a character sequence. The allowed escape sequences are as follows (other escape sequences are currently undefined):

- \n A single character interpreted by the system as end-of-line.
- \t Tab.
- \" quote as data

2 PROGRAM STRUCTURE

An \[L^F\] program is a newline-separated sequence of expressions, evaluated as they come.

\(^1\)Exercise: why not?
3 DECLARATIONS

There is no explicit variable declaration. They are considered declared after their first use.
Functions are declared.

4 DATA TYPES

\( \mathcal{L}^F \) has one primitive type, string, and one type constructor, \([ \ ]\), for a list of comma-separated items. That’s it.

Lists can be nested; strings cannot be nested.\(^2\)

A string can be either a quoted string constant or an unquoted token. If an unquoted string can be parsed as a number, it can be used as a number (see lexical aspects for numbers). A quoted string is a string even if it can be parsed as a number. An unquoted string which cannot be parsed as a number constant causes error.\(^3\)

For recursive manipulation of lists, the operator \(::\) does the following: \(x::xs\) is a list whose head is \(x\) and tail is \(xs\).

5 FUNCTIONS

Functions are declared with the keyword \texttt{def}, followed by a function name, which must be a legitimate identifier, followed by comma-separated sequence of formal arguments in parentheses, followed by the function body, followed by the keyword \texttt{end}.

If a formal argument is intended to be a function, only function name is written, not its formal arguments.\(^4\)

Function definitions can be nested.

6 SCOPE

Everything has local lexical scope.

A variable which is re-used as l-value in the same environment redefines the variable.
For example, an integer \(x\) can ‘become’ a list \(x\).

When functions are nested within other functions, the following rule applies.

\textbf{Shadowing:} A variable may be hidden by the re-use of the same name in a narrower scope.

\(^2\)Exercise: why not?

\(^3\)Exercise: Consider the psychological cost of allowing non-number unquoted strings such as \texttt{1045a9b3} as legitimate strings equivalent to \texttt{''1045a9b3’’}.

\(^4\)Exercise: What do we gain and/or lose by writing the formal arguments of the argument function?
7 VALUES

`nil` is a key constant, which cannot be redefined. It is only equal to itself.

Only variables have l-values.

Variables, constants, lists and expressions have r-values.

A function definition as an r-value returns `nil`. It will have the side effect of having defined a function.

A function call’s value is the result of the last expression evaluated by the function.

Comparative expressions evaluate to true or false. For compatibility with other expressions, everything that is not `nil` as r-value is true. True/false are not key constants of $\mathcal{L}^\mathbb{F}$.

8 EXPRESSIONS

Variables, constants, lists, definitions, assignment statement, if statement, while statement and the following unary/binary operations are expressions.

**List expression:** Any comma-separated expression in brackets is a list expression.

**Negation:** An integer- or float-valued expression may be prefixed by a minus sign.

**Operators:** Expressions of the form `exp op exp`, where `op` is `+`, `-`, `*`, `/`, can do arithmetic, string or list manipulation, where appropriate (see polymorphism).

**Comparison:** Expressions of the form `exp op exp`, where `op` is `==`, `<`, `>` or `<>`, `>=`, `<=` compare their operands for equality or inequality and produce true or false.

**Connectives:** Expressions of the form `exp op exp`, where `op` is `&` or `|`, are short-circuit boolean conjunction and disjunction: they do not evaluate the right-hand operand if the result is determined by the left-hand one.

**Precedence of operators:** Unary minus (negation) has the highest precedence. Then operators `/`, `*` have the next highest (tightest binding) precedence, followed by `+`, `-`, then by `==`, `<`, `>`, `>=`, `<=`, then by `&`, then by `|`.

**Associativity of operators:** The operators `*`, `/`, `+`, `-` are all left-associative. The comparison operators do not associate, so `a==b==c` is not a legal expression. Note that `a==b==c` is legal.

**Function call expression:** Calling a function requires its name and comma-separated actual arguments in parentheses (optional if the function has no argument), and returns its result as an r-value.

**Assignment:** The assignment expression `lvalue = exp` evaluates `exp`, then sets the contents of the `lvalue` to the result of the expression. The value of the assignment expression is the r-value of `exp`.

**If-then-else:** The if-expression `if exp1 then exp2 else exp3 end` evaluates the expression `exp1`. If the result is true it yields the r-value of `exp2`; otherwise it yields the
r-value of \( \text{exp}_3 \).

Else-part is optional, but keyword \texttt{end} is not.

Overall result of this expression is the r-value of the last expression evaluated.

**While-do:** The expression \texttt{while \texttt{exp}_1 \texttt{do} \texttt{exp}_2 \texttt{end}} evaluates \( \text{exp}_1 \); if the result is true, then \( \text{exp}_2 \) is evaluated, and the same process is repeated.

Overall result of this expression is the r-value of the last expression evaluated.

9 POLYMORPHISM and OVERLOADING

The operator \( + \) does addition, string append or list append, depending on type of arguments. If argument types differ, only numbers and variables whose values are unquoted numbers are legitimate.

The operator \( - \) does subtraction or list difference, depending on type of arguments. If argument types differ, only numbers and variables whose values are unquoted numbers are legitimate.

If the operands of arithmetic \( /, *, +, - \) are of different type, the result is the least common supertype. \( /, * \) are not available for quoted strings and lists (yet).

String is supertype of float, and float is supertype of integer.

The comparison operators may also be applied to strings and lists. Two strings are equal if their contents are equal. Inequality is according to lexicographic order.

Two lists are equal if they are of same size and memberwise equal. A list \( \text{L}_1 \) is less than another list \( \text{L}_2 \) iff a) they are of same size and \( \text{L}_1 \) is memberwise less than \( \text{L}_2 \), or b) \( \text{L}_1 \) is longer than \( \text{L}_2 \) and the first members of \( \text{L}_1 \) up to size of \( \text{L}_2 \) is memberwise less than \( \text{L}_2 \), c) \( \text{L}_2 \) is longer than \( \text{L}_1 \) and first members of \( \text{L}_1 \) are memberwise less than that of \( \text{L}_2 \) up to size of \( \text{L}_1 \).

10 STANDARD LIBRARY

Some functions are predefined:

\texttt{puts(s)}

\textit{Print s on standard output.}

\texttt{gets(s)}

\textit{Reads string s from standard input.}

\texttt{first(L)}

\textit{returns first element of list L.}

\texttt{rest(L)}

\textit{returns the remainder of list L.}