HLisp Specification

Taran Lynn

June 8, 2017

Abstract

HLisp is an attempt to make a minimal lisp with a very small core. All computations are done through list and symbol processing, and numeric support is not built-in. IO is also unsupported, as everything is expected to be done from the REPL. This makes HLisp practically just the lambda calculus extended with lists and symbols. As such HLisp is obviously designed for ease of implementation and academic research.

Contents

1 Data Types

HLisp supports three different primitive data types, which include

Lists Lists are what they sound like, lists of data.

Functions Functions should fulfill several requirements.

1. Functions should lazily evaluate their arguments.
2. All functions should be lexically scoped.

Symbols Symbols are named identifiers. They consist of any character that isn't whitespace or '(', ')', ';', or '.'.

All data is immutable.

2 Syntax and Semantics

Comments begin with a ; and extends to the end of their line.

2.1 EBNF Grammar

```
datum = list | quote | symbol ;

data = { datum } ;

list = '(', data, ')' ;

quote = "'", datum ;

symbol = { nonspecial } ;

nonspecial = ? any character except whitespace or '(', ')', ''', and '.'. ? ;
```

2.2 Evaluation

There are a few evaluation rules in HLisp, mainly

1. All quotes `'a` are expanded to `(quote a)`.

2. Symbols are looked up in the environment for a datum value.

3. For a list `(f ...)`, if `f` corresponds to the name of a special form, than an action specific to that special form is performed. If `f` is a function, then it is applied to the arguments. Otherwise, `f` is evaluated and the process is repeated.

3 Primitive Functions

The following are primitive functions defined in the run-time implementation. These functions force evaluation of their arguments. Note that true and false are defined as `(lambda (x y) x)` and `(lambda (x y) y)`, respectively.

(cons x xs) Appends an element `x` to the beginning of a list `xs`. This is an error if `xs` is not a list.

(car xs) Returns the datum at the beginning of the list `xs`. This is an error if `xs` is not a list.

(cdr xs) Returns all but the first datum at the beginning of the list `xs`. This is an error if `xs` is not a list.

(= x y) Returns true if the two objects are equal, otherwise it returns false. Functions are never equal to any other value.

(list? x) Returns true if `x` is a list, false otherwise.

4 Special Forms

The following are special forms, which are evaluated according to special rules.

(label name x) Binds the symbol `name` to `x` so that when it's evaluated it returns `x`. Binding is restricted to local scope within function definition. So `(lambda (x) (label y x))` will **not** create a global variable `y`.

lambda Forms for lambda include

- `(lambda name args body)`,

- `(lambda args body)`,
- `(lambda name (args...) body)`,
- and `(lambda (args...) body)`

It creates a new function. When evaluated this function evaluates `body` with the variable symbols rebound to the argument's values. Function arguments are either a list of symbols to be bound to positional argument, or a single symbol bound to a list of the arguments. This form should also handles closures. An optional name parameter enables recursion.

(quote x) Returns its argument `x` unevaluated.

(apply f xs) Applies a function to the elements of a list.