lispy Final Documentation

Contents

1 Design Justifications

LISP 1.5 was chosen as a base because it's syntactically very simple. 1.5 was chosen over a more modern dialect because it has a simpler feature set. Despite this, many features were still cut to even further simplify the implementation.

The language's three major features are higher-order functions, generic lists, and type inference. The former two were chosen because they are already part of LISP 1.5, and because they're comparatively simple to implement. The latter was chosen because it felt like a very natural addition to the language given the statically typed requirement. Lisp's syntax, despite being simple, can overwhelm a reader with parenthesis. Thus, having type inference helps with readability. However, the language is not fully type inferred for the sake of ease of implementation.

2 Examples

See the manual for more detailed explanations of features.

2.1 Example 1

```
(let
  ((a 1) (b nil))
  (set b
    (cond
      ((greaterp a 1) (list 1 2))
      (list 3 4)
    )
  )
  (car b)
)
```

`b` is reassigned based on whether `a` is greater than 1. The entire form evaluates to `(car b)`, which retrieves the first value of the list.

2.2 Example 2

```
(let
  (
    (add_10
      (lambda
        ((sum_floats (func (float float) float)) (x float))
        (progn
          (set x (sum_floats x 5.0))
          (sum_floats x 5.0)
        )
      )
    )
    (wrapped_sum
      (lambda
        ((a float) (b float))
        (sum a b)
      )
    )
  )
  (add_10 wrapped_sum 2.1)
)
```

A function `add_10` is defined by using a `lambda` form in the binding of a `let`. This function expects two arguments: a function which sums two floats, and a float to add 10 to. It adds 10 by adding 5 twice, the first time saving the value by reassigning the variable `x` in the scope of the function. `add_10` is called with `wrapped_sum` as the sum function and 2.1 as the value to add 10 to. `wrapped_sum` needs to wrap the special form `sum` because special forms are not higher-order functions.

3 Limitations

3.1 Deviations from Plans

- Code generation is not implemented.
- A majority of the planned special forms and built-ins are not implemented.
- `set` cannot create new bindings; it can only reassign.

- `car` and `cdr` should but don't disallow `nil` as the argument.
- `select` should but doesn't disallow comparing functions.

3.2 General

- No character objects / strings.
- No input support.
- No debugging, tracing, or error handling.
- No back-trace for runtime errors.
- No distinction between compiled and interpreted code; everything is compiled.
- Many built-in functions will be classified as special forms because the language's type system cannot describe these functions.
- Special forms are not first-class citizens.
- Special forms cannot be redefined.

3.3 Syntactic

- No comments.
- No octal numbers.
- No comma delimiter for list elements.
- No dot notation for S-expressions.

3.4 Typing

- All lists are homogenous.
- Types of lambda parameters must always be explicitly defined; they are never inferred.
- All branches of conditional expressions must evaluate to the same type.

3.5 Cut LISP 1.5 Features

- No macros.
- No arrays.
- No compiler/assembler functions.
- No `PROG`.
- No `QUOTE`.
- No `EVAL` or `EVALQUOTE`.
- No property lists (`GET`, `PUT`, `PROP`, `REMPROP`).
- No in-place list manipulation (`RPLACA`, `RPLACD`, `NCONC`).
- No user-defined functions using machine code.

4 Regrets and Learning Outcomes

I only have one regret and that is in how special forms were handled. I should have found a more generic approach to implementing them (perhaps with type variables), because currently each special form is hard-coded not only in the parser, but in the type checker too. This means that tests require more cases and take much longer to write. Given the amount of additional special forms I had planned, this approach would have not been sustainable; I would have probably lost my sanity writing so many tests.

5 Usage Instructions

The code generator is not implemented, so this section is not applicable.

6 Grammar

See the appendix in the manual.