

SoLISP Design

Yuheng Kuang (kuangyuheng@gmail.com)

April 19, 2011

1 Introduction

2 A taste of SoLISP

2.1 Hello World

Say hello world using `sys.stdout.write`. It represent the LISP natual of SoLISP and how it intergate with Python.

```
(import sys)
(def (hello :name "World")
  (@ sys stdout write ((% "Hello %s!\n" name))))
```

```
(hello) ; => "Hello World!\n"
(hello "Yuheng") ; => "Hello Yuheng!\n"
```

2.2 Everything is expression

As in LISP, Everything in SoLISP is expression.

```
(def (random_add :large False)
  (+ (if large 10000 else 1) 1))

(print (random_add)) ; => 1
(print (rnadom-add True)) ; => 10001
```

2.3 Powerful for

A simple list comprehension example.

```
(print (for x <- [1 4 2 3]
  (if (> x 2) (cont)) (* x x)))
; => [1 4]
```

A nested example.

```
(print
  (for x <- ["a" "b"]
    (emit* (for y <- [1 2] [x y])))) (cont)))
; => [["a" 1] ["a" 2] ["b" 1] ["b" 2]]
```

2.4 Restricted loop

Looping in SoLISP is more restricted. You specify init value of the loop, then use (cont next_value) to loop, the loop will break if you do nothing.

```
(print
  (loop total <- 1
    (if (< total 1000) (cont (* total 2)))))
; => 1024
```

2.5 Pattern matching

The test_proc is a one-argument function, that returns input when input is int and larger than 0, else raise an `MatchException`.

```
(= test_proc (# x :int ?(> x 0)))
(test_proc 1) ; => 1
(test_proc "1") ; => raise
(test_proc 0) ; => raise
```

Pattern matching can be used in assignment.

```
(= [0 x . remain] src)
```

SoLISP will check the value of src's first element (equals 0?), then assign second element to x and remaining elements to remain. It's called structural matching.

2.6 Proc: try-match-do model

Pattern matching is only a very limited case of the powerful Proc engine.

```
(= commander
  (# (str.split ["add" (int x) (int y)]) => (+ x y)
    # (str.split ["dec" (int x) (int y)]) => (- x y)
    # => -1))
```

```
(commander "dec 10 2") => 8
(commander "add 1 a") => -1
(commander "blahblahblah") => -1
```

It also shows how SoLISP can match virtually everything. The first two section of the proc split input string into list, then matching it into a list pattern. It's called "Extractor", we can use extractor to convert any object into basic data structure and match them.

2.7 Proc + Looping

Proc can be used in "for" and "loop".

Concise version of list comprehension.

```
(for (# x ?(<= x 2) => (* x x)) <- [4 1 3 2]) ; => [1 4]
```

```
(loop (# x ?(< x 1000)) <- 1 (cont (* x 2)))
```

2.8 Macros

We use "match" expression everywhere.

```
(match x
# ?(< _ 0) => "<"
# ?(== _ 0) => "=="
# => ">")
```

This expression is entirely implemented in SoLISP, by defining a macro.

```
(= macro_match (# ['match value . proc] => '('= proc value)))
```

It translates (match value . proc) into (= proc value), then assignment primitive do the job.

2.9 Full example

For full example, see Appendix A.

3 Data type

4 Basic syntax

5 Pattern matching

6 Looping and iterating

7 Customize SoLISP

A Full example: Generating DOM Tree