Design and implementation of psll – Lisp-like programming language which compiles to Pyramid Scheme

Marcin Konowalczyk[1, 2, a)]

[1)] *Department of Chemistry, University of Oxford, Chemistry Research Laboratory, Oxford OX1 3TA, U.K.*

[2)] *UCLA Samueli School of Engineering, University of California, Los Angeles, 7400 Boelter Hall, Los Angeles, CA 90095, United States*

(Dated: March 25, 2021)

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetuer id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.

Keywords: syntax tree; Pyramid Scheme; lisp; compilation

I. INTRODUCTION

In ancient Egypt, pyramids were constructed as the resting places of deceased pharaos, containing not only their mummified remains but also an assortment of keywords and type literals the pharaoh will need in their journey though afterlife. Pyramid Scheme (PS) is a variant of the Scheme dialect of Lisp, which honours these ancient traditions and accompanies *us* thorough our journey of computation.

PS was designed by Conor O'Brien, in the early 2017 (date of the earliest commit to the GitHub repository).[1] It is a turing-complete esoteric programming language (esolang)[2] which uses tree-like, as opposed to a serial code structure. Compilers make use of an intermediate representation of the language in the form of an abstract syntax tree (AST).[3] In contrast to most contemporary languages / frameworks, which build on top of the existing infrastructure to create "the stack" of software,[4,5] Pyramid Scheme aims to shed any unnecessary abstractions, including that of the AST. The computation in pPramid scheme is therefore represented as a literal syntax tree (LST) of ascii-art pyramidal constructs.

Pyramid Scheme is supported by the "Try It Online!" repository of online interpreters,[6] and, like many other esolangs, has been featured in many Code Golfing challenges.[7]

A. Pyramid Scheme

The original and, so far, the only implementation of PS is written in Ruby.[1] The LST of the program is first parsed and then mapped to a recursive evaluation chain. An example of one such program can be seen in Listing 1.

```
      ^ ^     ^           ^       ^       ^     ^
     ^-/ \   / \         / \     / \     / \   -^
    ^-/out\ /set\       /out\   /out\   /out\  / \
   / \-----^-----^     ^-----  ^-----  ^-----  /out\
  /set\   /x\   /+\   / \     / \     / \     ^-----
 ^-----^  ---  ^---^ /chr\   /chr\   /chr\   /x\
/x\   /#\     /x\ /1\-----^ -----^ -----^ ---
---  ^---     --- ---    / \     / \     / \
   /1\                  /43 \   /49 \   /61 \
  /ine\                 -----   -----   -----
  -----
```

Listing 1. A simple Pyramid Scheme program. It takes one input from `stdin` – `(set x (# stdin))`, increments it by one – `(set x (+ x 1))` and prints the result computation to the command line.

PS parser reads the body of each pyramid verbatim, concatenated line by line.[8] The parser begins at the tip (`^`), and walks down the left (`/`) and the right (`\`) side, collecting the characters in-between. When the two sides run out, it first checks for the presence of the pyramid base (`-`),[9] and then for the tips of the child pyramids, if present. The pyramids may connect *only* on these corners, such that, for example, the first pyramid with `chr` (which constructs a character + to be printed) in Listing 1 rightfully does not consider the pyramid `1` of the `set` branch as its child.

Note, however, that this allows for an existence of direct connection between neighbouring branches of the LST – in Listing 1, for example, the first print statement (`out` keyword), shares the node `x` with its neighbouring branch. This is an interesting parallel to the phenomenon of the lateral gene transfer observed in genetics, and suggests a more-proper description of the PS to be that of a Ewok village syntax tree (EWST).[10,11] Although this is undoubtedly one of the more

[a)] Electronic mail: marcin.konow@lczyk.xyz

interesting and powerful features of PS, it has not yet been implemented in the project described herein shortly, and therefore will not be considered further, but left for future work.

The specification of the pyramid structure does not preclude the existence of a pyramid with no content. Such a height-0 pyramid is falsey and evaluates to 0.[12][13] A pyramid with no content *does* however both evaluate its children, and pass them as an its output. This make the height-0 pyramid an important construct for code packing, as can be seen in the first branch in Listing 1

There are two types operators in PS: ones which implicitly evaluate both of their children, as well as those which do this only under certain circumstances. The first group maps very closely to its underlying Ruby implementation. There are basic binary arithmetic and comparison operators: `+`, `*`, `-`, `/`, `^`, `=`, `!` and `<=>`. Keyword `out` prints all of its inputs and `chr` converts number to a character. The keyword `arg` indexes arrays (or input arguments), and keywords `#` and `"` convert back and forth from and to a string. `#` character also allows one prompt user for input if given a (semi)keyword `line`.[14][1]

In that, the The memory makes us of a Ruby dictionary for storing variable Ruby Dictionary Memory

B. Motivation behind psll

[15]

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetuer id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.

Nam dui ligula, fringilla a, euismod sodales, sollicitudin vel, wisi. Morbi auctor lorem non justo. Nam lacus libero, pretium at, lobortis vitae, ultricies et, tellus. Donec aliquet, tortor sed accumsan bibendum, erat ligula aliquet magna, vitae ornare odio metus a mi. Morbi ac orci et nisl hendrerit mollis. Suspendisse ut massa. Cras nec ante. Pellentesque a nulla. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Aliquam tincidunt urna. Nulla ullamcorper vestibulum turpis. Pellentesque cursus luctus mauris.

Nulla malesuada porttitor diam. Donec felis erat, congue non, volutpat at, tincidunt tristique, libero. Vivamus viverra fermentum felis. Donec nonummy pellentesque ante. Phasellus adipiscing semper elit. Proin fermentum massa ac quam. Sed diam turpis, molestie vitae, placerat a, molestie nec, leo. Maecenas lacinia. Nam ipsum ligula, eleifend at, accumsan nec, suscipit a, ipsum. Morbi blandit ligula feugiat magna. Nunc eleifend consequat lorem. Sed lacinia nulla vitae enim. Pellentesque tincidunt purus vel magna. Integer non enim. Praesent euismod nunc eu purus. Donec bibendum quam in tellus. Nullam cursus pulvinar lectus. Donec et mi. Nam vulputate metus eu enim. Vestibulum pellentesque felis eu massa.

Quisque ullamcorper placerat ipsum. Cras nibh. Morbi vel justo vitae lacus tincidunt ultrices. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. In hac habitasse platea dictumst. Integer tempus convallis augue. Etiam facilisis. Nunc elementum fermentum wisi. Aenean placerat. Ut imperdiet, enim sed gravida sollicitudin, felis odio placerat quam, ac pulvinar elit purus eget enim. Nunc vitae tortor. Proin tempus nibh sit amet nisl. Vivamus quis tortor vitae risus porta vehicula.

Fusce mauris. Vestibulum luctus nibh at lectus. Sed bibendum, nulla a faucibus semper, leo velit ultricies tellus, ac venenatis arcu wisi vel nisl. Vestibulum diam. Aliquam pellentesque, augue quis sagittis posuere, turpis lacus congue quam, in hendrerit risus eros eget felis. Maecenas eget erat in sapien mattis porttitor. Vestibulum porttitor. Nulla facilisi. Sed a turpis eu lacus commodo facilisis. Morbi fringilla, wisi in dignissim interdum, justo lectus sagittis dui, et vehicula libero dui cursus dui. Mauris tempor ligula sed lacus. Duis cursus enim ut augue. Cras ac magna. Cras nulla. Nulla egestas. Curabitur a leo. Quisque egestas wisi eget nunc. Nam feugiat lacus vel est. Curabitur consectetuer.

II. LANGUAGE DESCRIPTION

A. Bracket structure

B. Syntactic sugar

The above specification is, in principle, enough to create fully fully functional PS programs. Certain tasks are, however, still rather cumbersome. This section outlines these cases, as well as syntactic sugar constructs introduced to `psll` to alleviate them. All of these are implemented as local (or semi-local) expansion macros, as described in Section III B. Despite authors best efforts, this introduces some sharp edges into the language (see Section II C).

Implicit bracket expansion Each bracket must have exactly three elements. For small expressions this is almost always the case, but becomes problematic for larger, flow-control and loop structures where each such expression can contain an arbitrarily large number of sub-expressions (see **[info]** for an example of such expression). Hence a bracket containing > 2 other brackets gets expanded as follows:

```
( (out 1) (out 2) (out 3) (out 4) (out 5) )
```

is interpreted as:

```
( (((out 1) (out 2)) ((out 3) (out 4))) (out 5) )
```

Each neighbouring pair or elements of the parent gets put together into a bracket, until the length of the parent is less than 2. This results in a (literal) balanced binary tree in the final PS code, and so for a parent bracket of N sub-expressions will result in a tree of height $\mathcal{O}\left(\log_2(N)\right)$.

String literals Single characters can be created in RDM with the `chr` keyword (Ruby `.to_i.chr`). It is also possible to construct longer strings in RDM since Ruby's "+" sign overloads string concatenation. The string *"hello"* is therefore:

```
(+ (+ (+ (+ (chr 72) (chr 101)) (chr 108))
   (chr 108)) (chr 111))
```

`psll` introduces string literals, such that `(set s "hello")` expands into the above code. Note that this is a very left-child heavy tree. To balance it, the above string could also be made by recursively concatenating its binary split:

```
(+ (+ (chr 72) (chr 101))
   (+ (chr 108) (+ (chr 108)) (chr 111)))
```

such that *"hello"* = *"he"* + *"llo"* = (*"h"* + *"e"*) + (*"l"* + *"lo"*) = ...

Array literals

Rolling sum and product

def keyword

Semi-local

C. Sharp edges

As mentioned at the beginning of Section II B, the introduction of syntactic sugar into `psll` introduces some edge cases which one ought to watch out for.

Underscore keyword _

Escape characters Because " is used for strings, and [ ] for arrays...

III. COMPILER

A. Abstract syntax tree

B. Local macro expansion

C. Optimisation

IV. EXAMPLE PROGRAMS

A. Pseudorandom number generation

B. Bubble sort

C. Chess engine

```
(set a 0) // Flip-flop
(set N 10) (set j 0) // N of iteration and loop counter
(loop (! (= j N)) (
    // Do some work...
    (out j (chr 32)) // Print j and space
    (out a (chr 10)) // Print a and newline
    (set a (! a)) // Flip a
(set j (+ j 1))
)) (set test "hi")
```

```
  (set newline "\n")
  (set nil (arg 999))

  // Array to be sorted
  (set a [3 1 4 1 5 9 2 6 5 3 5])

  // Get array length
  // (len a N)

  ( (set N 0) (loop (! (= (arg a N) nil)) (set N (+ N 1))) )

  (set N 0) // Pointer into the array
  (loop (! (= (arg a N) nil)) (set N (+ N 1))) // Increment pointe

  (def sup "hello")
  // Bubble sort the array
  (do again (
      (set again 0)
      (set n 0) // Position pointer
      (loop (! (! (<=> n (- N 1)))) ( // Go through all the pairs
          (set this (arg a n))
          (set next (arg a (+ n 1)))

          // This and next need swapping
          (set swap (! (<=> (<=> this next) -1)))
          (? swap (
              (set again 1) // Will need to go through the list ag
              (set b []) // Start b as an empty array

              // Add prefix of a
              (set l 0)
              (loop (= (<=> l n) -1) (
                  (set b (+ b (- ((arg a l) nil) (nil nil))))
```

```
            (set l (+ l 1))
            ))

            // Add two elements, swapped
            (set b (+ b (- ((arg a (+ n 1)) nil) (nil nil))))
            (set b (+ b (- ((arg a (+ n 0)) nil) (nil nil))))

            // Add suffix of a
            (set l (+ n 2))
            (loop (= (<=> l N) -1) (
                (set b (+ b (- ((arg a l) nil) (nil nil))))
            (set l (+ l 1))
            ))
            (set a b)
        ))
    (set n (+ n 1)) // Increment position pointer
    ))
    (out (* a ",") newline) // Print b + newline
))
```

V. CONCLUSIONS AND OUTLOOK

"Program in Pyramid Scheme! Teach your friends! Have them teach their *friends! Then have those friends teach* their *friends!"*

- Joined pyramids

Sed feugiat. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Ut pellentesque augue sed urna. Vestibulum diam eros, fringilla et, consectetuer eu, nonummy id, sapien. Nullam at lectus. In sagittis ultrices mauris. Curabitur malesuada erat sit amet massa. Fusce blandit. Aliquam erat volutpat. Aliquam euismod. Aenean vel lectus. Nunc imperdiet justo nec dolor.

Etiam euismod. Fusce facilisis lacinia dui. Suspendisse potenti. In mi erat, cursus id, nonummy sed, ullamcorper eget, sapien. Praesent pretium, magna in eleifend egestas, pede pede pretium lorem, quis consectetuer tortor sapien facilisis magna. Mauris quis magna varius nulla scelerisque imperdiet. Aliquam non quam. Aliquam porttitor quam a lacus. Praesent vel arcu ut tortor cursus volutpat. In vitae pede quis diam bibendum placerat. Fusce elementum convallis neque. Sed dolor orci, scelerisque ac, dapibus nec, ultricies ut, mi. Duis nec dui quis leo sagittis commodo.

Aliquam lectus. Vivamus leo. Quisque ornare tellus ullamcorper nulla. Mauris porttitor pharetra tortor. Sed fringilla justo sed mauris. Mauris tellus. Sed non leo. Nullam elementum, magna in cursus sodales, augue est scelerisque sapien, venenatis congue nulla arcu et pede. Ut suscipit enim vel sapien. Donec congue. Maecenas urna mi, suscipit in, placerat ut, vestibulum ut, massa. Fusce ultrices nulla et nisl.

Etiam ac leo a risus tristique nonummy. Donec dignissim tincidunt nulla. Vestibulum rhoncus molestie odio. Sed lobortis, justo et pretium lobortis, mauris turpis condimentum augue, nec ultricies nibh arcu pretium enim. Nunc purus neque, placerat id, imperdiet sed, pellentesque nec, nisl. Vestibulum imperdiet neque non sem accumsan laoreet. In hac habitasse platea dictumst. Etiam condimentum facilisis libero. Suspendisse in elit quis nisl aliquam dapibus. Pellentesque auctor sapien. Sed egestas sapien nec lectus. Pellentesque vel dui vel neque bibendum viverra. Aliquam porttitor nisl nec pede. Proin mattis libero vel turpis. Donec rutrum mauris et libero. Proin euismod porta felis. Nam lobortis, metus quis elementum commodo, nunc lectus elementum mauris, eget vulputate ligula tellus eu neque. Vivamus eu dolor.

Nulla in ipsum. Praesent eros nulla, congue vitae, euismod ut, commodo a, wisi. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Aenean nonummy magna non leo. Sed felis erat, ullamcorper in, dictum non, ultricies ut, lectus. Proin vel arcu a odio lobortis euismod. Vestibulum ante ipsum primis in faucibus orci luctus et ultrices posuere cubilia Curae; Proin ut est. Aliquam odio. Pellentesque massa turpis, cursus eu, euismod nec, tempor congue, nulla. Duis viverra gravida mauris. Cras tincidunt. Curabitur eros ligula, varius ut, pulvinar in, cursus faucibus, augue. `(chr 10)`

REPRODUCIBILITY

ACKNOWLEDGEMENTS

I would like to thank Blaine Rodgers and Samuel Hutton for discussions and helpful comments on the manuscript, as well as Jonathan Blow and David Beazley, for sparking a long-lasting interest in programming languages.

REFERENCES

[1] Conor O'Brien. Pyramid scheme. GitHub repository, https://github.com/ConorOBrien-Foxx/Pyramid-Scheme, 2017.

[2] Pyramid scheme. https://esolangs.org/wiki/Pyramid_Scheme.

[3] Linda Torczon and Keith Cooper. *Engineering A Compiler*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2nd edition, 2007.

[4] Bryan Cantrill. *Zebras All the Way Down*. Uptime 2017, https://youtu.be/fE2KDzZaxvE.

[5] Casey Muratori. The thirty million line problem. https://youtu.be/kZRE7HIO3vk, 2018.

[6] Try it online! https://tio.run.

[7] Code golf stackexchange. https://codegolf.stackexchange.com.

[8] Hence, for example, the bottom pyramid in the first stack in Listing 1 contains the (semi)keyword `line`, as opposed to two words: `l` and `ine`.

[9] Note that the base of the pyramid is a dash (`0x2d`), not an underscore.

[10] Patrick J. Keeling and Jeffrey D. Palmer. Horizontal gene transfer in eukaryotic evolution. *Nature Reviews Genetics*, 2008.

[11] Zachary Weinersmith. Ewok village of life. SMBC, https://www.smbc-comics.com/comic/2012-04-08.

[12] The term "height-0" can be ambiguous since the pyramid itself has height of 2 characters. In this work the pyramid's height, however, is the number of lines of the text in its body.

[13] Conor O'Brien. Pyramid scheme negation. https://codegolf.stackexchange.com/questions/147513/pyramid-scheme-negation.

[14] `line`, (as well as `stdin`, `readline`) are referenced to as *semi*keywords since they have a keyword meaning only when they're an input of the `#` command.

[15] David Beazley. *Reinventing the Parser Generator*. Pycon 2018, https://youtu.be/zJ9z6Ge-vXs.

[16] Blaine Rodgers. blen. GitHub repository, https://github.com/PaperclipBadger/high-octane-rumble-simulation-engine, 2017.

[17] Jonathan Blow. Making programming language parsers. https://youtu.be/MnctEW1oL-E, 2020.