Design and implementation of `ps11` – 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\-----^ -----^ -----^ ---
---  ^---     --- ---    / \    / \    / \
    /l\                 /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]

The second group of operators conditionally evaluates only one of their children. `set` sets the variable denoted by its left child to the evaluated right one. `loop` and `do` evaluate the right child subtree as long as the left one evaluates to true (with the difference being when is the check made – before and after right subtree evaluation respectively). Finally ? evaluates the right subtree only if the left one evaluates to true, else it evaluates to zero.

II. PSLL

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 subexpressions will result in a tree of height $\mathcal{O}(\log_2(N))$.

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
// This will be: (len a N)
(set N 0) // Pointer into the array
// Increment pointer until goes off the end
(loop (! (= (arg a N) nil)) (set N (+ N 1)))

// Bubble sort the array
(do again (
  (set again 0)
  (set n 0) // Position pointer
  (loop (! (! (<=> n (- N 1)))) ( // For all 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 again
      (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

Finally, I think every programmer shares a certain latent interest in the underlying structure and of the languages they use every day. I would encourage them to scratch that itch and learn more about. If you come from a formal computer science background this may not be new to you, but many of us, myself included, learned programming through other ways. There are plenty of resources to start, but I'm inclined to mirror the advice of Casey Muratori:[15] *"Look at all of the resources on these topics in in the following way: rather than reading what someone tells you about how to build a compiler (...) start programming one without knowing what you're doing (...) and see what you can learn. When you cannot make forward progress (...) [look for] solution to that particular problem you're having. (...) Now you have some context to evaluate what people what you (...) whereas if you read about stuff without ever actually having encountered a problem yet, then you're just gonna you have no idea [whether its valuable]."* If you really want a starting point though, I recommend David Beazley's ply and sly projects. [16–19]

REPRODUCIBILITY

At the time of writing, the commit SHA of the main Pyramid Scheme GitHub repo[1] is:

`fd183d296f08e0cba8bf55da907697eaf412f6a7`

and the `psll` repo:

`ea6c6d2e9c8278cc752894a9aff43f7d44bd93a9`

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] Words `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] Jonathan Blow and Casey Muratori. Q&a: Making programming language parsers. https://youtu.be/lcF-HzlFYKE, Starting at minute 8.00, 2020.

[16] David Beazley. Reinventing the parser generator. Pycon 2018, https://youtu.be/zJ9z6Ge-vXs.

[17] David Beazley. Sly (sly lex-yacc). GitHub repository, https://github.com/dabeaz/sly.

[18] David Beazley. Ply (python lex-yacc). GitHub repository, https://github.com/dabeaz/ply.

[19] John Levine, Doug Brown, and Tony Mason. *lex & yacc*. O'Reilly Media, Inc., 2nd edition.

[20] The lex & yacc page. http://dinosaur.compilertools.net.

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

[22] Jonathan Blow and Casey Muratori. Making programming language parsers. https://youtu.be/MnctEW1oL-E, 2020.

```
         ^            ^       ^               ^
        / \          ^-      /l\              -^
       /set\        ^-      /oop\             / \
      ^-----^      ^-      ^-----^           /   \
     /n\   / \    ^-      /!\   / \         / do  \
     ---  /arg\  ^-      ^---  /set\       ^-------^
         ^----- ^-      /=\   ^-----^     / \     / \
        / \    ^-      ^---^ /N\   /+\   /   \   /   \
       /999\  ^-      / \ /n\---  ^---^ /again\ /     \
       ----- / \     /arg\---    /N\ /1\-------/       \
            /   \   ^-----^      --- ---      /         \
           /     \ /a\   /N\                 /           \
          /       \---   ---                /             \
         ^---------^                       /               \
        / \       / \                     /                 \
       /set\     /set\                   ^-------------------^
      ^-----^   ^-----^                 / \                 / \
     /a\   / \ /N\   /0\               /   \               /   \
     ---  /   \---   ---              /     \             /     \
         /     \                     /       \           /       \
        /   +   \                   ^---------^         /         \
       ^---------^                 / \       / \       /           \
      / \       / \               /set\     /set\     /             \
     ^---^     /   \             ^-----^   ^-----^   ^---------------^
    /3\ /1\   /  +  \           / \   /0\ /p\   /0\ / \             / \
    --- ---  ^-------^         /   \  --- ---   ---/   \           /out\
            / \     / \       /again\             /     \         ^-----^
           ^---^   /   \      -------            /       \       /*\   / \
          /4\ /1\ /     \                       /         \     ^---^ /chr\
          --- ---/   +   \                     /   loop    \   /a\ / \-----^
                ^---------^                   ^-------------^  ---/chr\   / \
               / \       / \                 /!\           / \    -----^ /10 \
              ^---^     /   \               ^---          /   \       / \-----
             /5\ /9\   /  +  \             /!\           /     \     /44 \
             --- ---  ^-------^           ^---          /       \    -----
                     / \     / \         / \           ^---------^
                    ^---^   /   \       /<=>\         / \       / \
                   /2\ /6\ /     \     ^-----^       /   \     /set\
                   --- ---/   +   \   /p\   /-\     /     \   ^-----^
                         ^---------^  ---  ^---^   /       \ /p\   /+\
                        / \       / \     /N\ /1\ /         \---  ^---^
                       ^---^     /   \    --- ---/           \   /p\ /1\
                      /5\ /3\   /  -  \         /             \  --- ---
                      --- ---  ^-------^       /               \
                              / \     / \     /                 \
                             ^---^   ^---^   /                   \
                            /5\ /0\ /0\ /0\ /                     \
                            --- --- --- ---/                       \
                                          /                         \
                                         /                           \
                                        /                             \
                                       /                               \
```

```
53                                       ^---------------------------------^
54                                      / \                               / \
55                                     /   \                             /   \
56                                    /     \                           /     \
57                                   /       \                         /       \
58                                  /         \                       /         \
59                                 /           \                     /           \
60                                /             \                   ^-------------^
61                               ^---------------^                 / \           / \
62                              / \             / \               /set\         / ? \
63                             /   \           /   \             ^-----^       ^-----^
64                            / set \         / set \           /s\   /!\     /s\   / \
65                           ^-------^       ^-------^         /wap\ ^---    /wap\ /   \
66                          /t\     / \     /n\     / \        -----/ \      -----/     \
67                         /his\   /arg\   /ext\   /arg\           /   \         /       \
68                         -----  ^-----^  -----  ^-----^         / <=> \       ^---------^
69                               /a\   /p\       /a\   /+\       ^-------^     / \       / \
70                               ---   ---       ---  ^---^     / \     / \   /   \     /set\
71                                                   /p\ /1\   /<=>\   /-1 \ /     \   ^-----^
72                                                   --- ---  ^-----^  -----/       \ /a\   /b\
73                                                           /t\   /n\     /         \---   ---
74                                                          /his\ /ext\   /           \
75                                                          ----- -----  /             \
76                                                                      /               \
```

Listing 2. Bubble sort in Pyramid Scheme. Lines 1-155. This listing has been split into two sections. Lines 156-230 can be found in Listing 3.

```
156                         /                                                             \
157                        ^---------------------------------------------------------------^
158                       / \                                                             / \
159                      /   \                                                           /   \
160                     /     \                                                         /     \
161                    /       \                                                       /       \
162                   /         \                                                     /         \
163                  /           \                                                   /           \
164                 /             \                                                 /             \
165                /               \                                               /               \
166               /                 \                                             /                 \
167              /                   \                                           /                   \
168             /                     \                                         /                     \
169            ^-----------------------^                                       /                       \
170           / \                     / \                                     /                         \
171          /   \                   /   \                                   ^---------------------------^
172         /     \                 /     \                                 / \                         / \
173        /       \               /       \                               /   \                       /   \
174       ^---------^             ^---------^                             /     \                     /     \
175      / \       / \           / \       / \                           /       \                   /       \
176     /set\     /set\         /set\     /   \                         /         \                 /         \
177    ^-----^   ^-----^       ^-----^   /     \                       /           \               /           \
178   / \   /1\ /b\   / \     /1\   /0\ /       \                     /             \             /             \
179  /   \  --- ---  /   \    ---   ---/         \                   ^---------------^           ^---------------^
180 /again\         /  -  \           /   loop    \                 / \             / \         / \             / \
181 -------        ^-------^         ^-------------^               /set\           /set\       /set\           /   \
182               / \     / \       / \           / \             ^-----^         ^-----^     ^-----^         /     \
183              ^---^   ^---^     /   \         /   \           /b\   /+\       /b\   /+\   /1\   /+\       /       \
184             /0\ /0\ /0\ /0\   /  =  \       /     \          ---  ^---^      ---  ^---^  ---  ^---^     /         \
185             --- --- --- ---  ^-------^     /       \             /b\ / \         /b\ / \     /p\ /2\   /   loop    \
186                             / \     / \   /         \            ---/   \        ---/   \    --- ---  ^-------------^
187                            /<=>\   /-1 \ ^-----------^             /  -  \         /  -  \           / \           / \
188                           ^-----^  -----/ \         / \           ^-------^       ^-------^         /   \         /   \
189                          /1\   /p\     /set\       /set\         / \     / \     / \     / \       /  =  \       /     \
190                          ---   ---    ^-----^     ^-----^       ^---^   ^---^   ^---^   ^---^     ^-------^     /       \
191                                      /b\   /+\   /1\   /+\     / \ /n\ /n\ /n\ / \ /n\ /n\ /n\   / \     / \   /         \
192                                      ---  ^---^  ---  ^---^   /arg\--- --- ---/arg\--- --- ---  /<=>\   /-1 \ ^-----------^
193                                          /b\ / \     /1\ /1\ ^-----^         ^-----^           ^-----^  -----/ \         / \
194                                          ---/   \    --- ---/a\   /+\       /a\   /+\         /1\   /N\     /set\       /set\
195                                            /  -  \          ---  ^---^      ---  ^---^        ---   ---    ^-----^     ^-----^
196                                           ^-------^             /p\ /1\         /p\ /0\                   /b\   /+\   /1\   /+\
```

```
197                                          / \     / \            --- ---         --- ---                   ---  ^---^  ---  ^---^
198                                         ^---^   ^---^                                                         /b\ / \     /l\ /1\
199                                        / \ /n\ /n\ /n\                                                        ---/   \    --- ---
200                                       /arg\--- --- ---                                                          /  -  \
201                                      ^-----^                                                                   ^-------^
202                                     /a\   /l\                                                                 / \     / \
203                                     ---   ---                                                                ^---^   ^---^
204                                                                                                             / \ /n\ /n\ /n\
205                                                                                                            /arg\--- --- ---
206                                                                                                           ^-----^
207                                                                                                          /a\   /l\
208                                                                                                          ---   ---
```

Listing 3. Bubble sort in Pyramid Scheme. Lines 156-230. This listing has been split into two sections. Lines 1-155 can be found in Listing 2. The indentation of this listing has been reduced, and the middle section of an otherwise empty pyramid has been omitted for readability.