psilo

a parallel, streaming, iterative list operation language for writing interesting programs. View it on GitHub.

What is psilo?

psilo, like any lisp, will be a **lis**t **p**rocessor. However psilo will take advantage of a strong type system, linear types, and aggressive optimization techniques such as shortcut fusion. The goal is to make writing fast, resource-aware, parallelized stream processing programs as simple as possible.

Its goals are not to make programming easier, but to widen what is safely expressible.

It is also nowhere close to being finished; at the moment this is purely exploratory.

Technical Features (planned):

- No run-time garbage collection necessary owing to uniqueness types
- Static typing for compile-time verification and optimization
- Malleable syntax with macros
- Dead-simple parallelism, featuring special list types (à la sequenceL).
- Monadic continuations and iteratee composition made dead simple
- Orthogonal core syntax and semantics for your performance and my sanity

Philosophy:

- All programming is manipulating languages.
- Types define grammars; functions define parsers.
- The earlier a question may be answered, the better.
- If the computer can do it, it should.

Status

Psilo is still being designed. I have written a really simple evaluator for prototyping and experimenting with the language which is actively being developed.

It is not a psilo implementation - just a simple lisp to explore writing interpreters. While not production quality, it might be of some educational value right now.

A type inference system is in the works, which is motivating an evaluator rewrite. When it works I can begin experimenting with psilo's type system and the language proper.

Here is some code the interpreter runs right now:

```
; Utilities
(= promise (x) (\ () x))

; Pairs
(= unpair (p f)
  (p f))

(= Pair (f) f)

(= pair (x y)
  (Pair (\ (f) (f x y))))

(= fst (p)
  (unpair p (\ (a b) a)))

(= snd (p)
  (unpair p (\ (a b) b)))

; Options
(= maybe (o yes no)
   (o yes no))

(= Option (x) x)

(= just (x)
  (Option (\ (j n) (j x))))

(= none ()
  (Option (\ (j n) (n))))

; Lists
(= foldr (xs c n)
  (xs c n))

(= List (l) l)

(= cons (x xs)
  (List (\ (c n) (c x (foldr xs c n)))))

(= nil ()
  (List (\ (c n) (n))))

(= split (xs)
  (let ((f (\ (y ys)
             (pair (just y)
                   (List (\ (c n)
                     (maybe (fst ys)
                            (\ (x) (c x (foldr ((snd ys)) c n)))
                            n)))))))
  (foldr xs f (pair none nil))))

(= car (xs)
  (maybe (fst (split xs)) (\ (x) x) (promise -1)))

(= cdr (xs)
  (snd (split xs)))

(= sum (xs)
  (foldr xs (\ (y ys) (+ y ys)) (promise 0)))

(= map-list (f xs)
  (foldr xs (\ (y ys) (cons (f y) ys)) (promise (nil))))

; Miscellaneous examples
(= add1 (x) (+ 1 x))
(= square (x) (* x x))
(= fact (x)
  (if (=? x 0)
      1
      (* x (fact (- x 1)))))

(= lst1 (cons 3 (cons 2 (cons 1 (nil)))))

(print (+ (sum (map-list add1 (map-list square lst1)))
          (fact (car (cdr (lst1))))))
```

How to build

You need the Glasgow Haskell Compiler and a number of libraries; I suggest starting off with the Haskell platform.

Clone the repository:

```
git clone https://github.com/gatlin/psilo
```

Set up a cabal sandbox:

```
cabal sandbox init
cabal configure
cabal install --only-dependencies
```

Then make with:

```
make
```

And return to the Edenic, pre-build post-checkout status of the code with

```
make clean
```

Questions / comments / hate mail

Use the Issues feature of GitHub or email me: gatlin@niltag.net.