Lisp

https://github.com/
JonChesterfield/lisp-meetup.git

February 23, 2016

Lisp is mathematics

- John McCarthy, MIT, April 1960
- Instantiation of the λ calculus
- A basis set for computation

Lisp is a language

- (car cdr cons if quote eq atom)
- Kernel
- Scheme
- Emacs lisp
- Common lisp
- Clojure

Lisp is whatever you want

- Dynamically & statically typed
- Compiled & interpreted
- Deterministic & unpredictable
- Uniform & disambiguated
- Flexible & performant
- Customizable & standardised

Lisp can be what I want

- Referentially transparent
- Implicitly multithreaded
- Implicitly distributed
- Faster than C++
- Clearer than Python
- Provably correct

Lisp is a compiler's IR

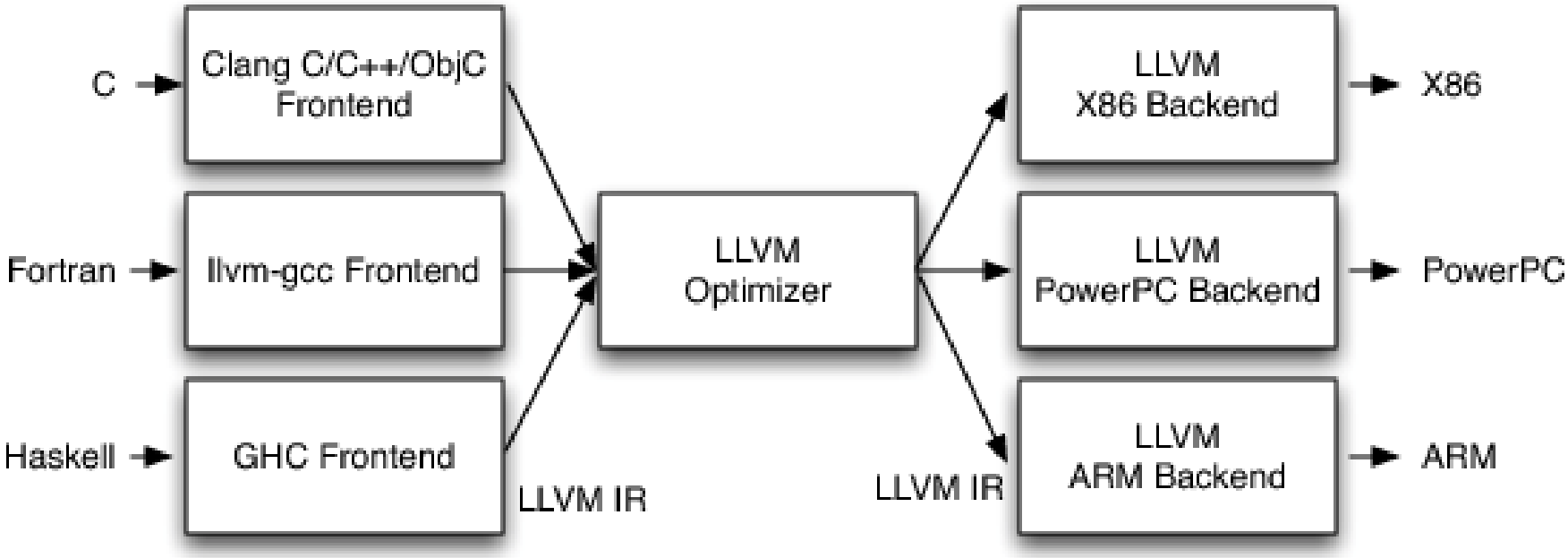

Figure: I can't work out who to cite!

So it tends to look like

```
(define (fact n)
  (if (<= n 1)
    1
    (* n (fact (- n 1)))))
(defun fact (n)
  (if (<= n 1)
    1
    (* n (fact (- n 1)))))
(defn fact [x]
  (loop [n x f 1]
    (if (<= n 1)
      f
      (recur (dec n) (* f n)))))
```

And that's not so bad

- sum (a, b, c) $\approx$ (sum a b c)
- (a + b + c + d) $\approx$ (+ a b c d)
- func two (x) { return 2 * x; } $\approx$ (define two (x) (* 2 x))
- (8 == two(4)) $\approx$ (= 8 (two 4))
- Maybe an hour to adjust.

Where to start?

- Scheme => guile, racket, chicken, gambit
- Common lisp => sbcl, ccl, gcl
- Clojure => Clojure
- Elisp => Emacs
- Kernel => Write your own

Most likely to be...

- Dynamically typed
- Lexically scoped
- Garbage collected
- Adequately fast
- Syntactically customisable
- Semantically customisable

Dynamically typed

```
(define (dyn val)
  (if (number? val)
    42
    ``life''))
(dyn 5)    ; => 42
(dyn dyn)  ; => ``life''

(define var 42)
var        ; => 42
(set! var (lambda () (``life'')))
var        ; => #<procedure var ()>
(var)      ; => ``life''
```

Lexically scoped

```
(define (print x)
  (begin (write x) (newline)))
(define (what x)
  (if (= a 1)
    (print ``Lexical'')
    (print ``Dynamic'')))
(let ((a 1))
  (let ((f (lambda () (what a))))
    (let ((a 2))
      (f)))) ; ``Lexical''
```

Garbage collected

- Objects appear to live forever
- Unwind-protect $\approx$ (with | using)
- Unwind-protect $\neq$ (with | using)

Syntactically customisable

- Clojure [1 2] & {: *a* 1, : *b* 2}
- Racket #(1 2) & (*hash* ′*a* 1 ′*b* 2)
- Common lisp has reader macros
- Scheme has srfi-10 e.g. #, (*foo*)
- I'd rather use a DSL & parser

Semantically customisable

- call-with-current-continuation
- defmacro
- syntax-rules
- fexpr
- hack up the compiler

Example 0

```
(define (fib n)
  (cond ((= n 0) 0)
        ((= n 1) 1)
        (#t (+ (fib (- n 1))
               (fib (- n 2))))))
(map fib '(0 1 2 3 4 5 6  7  8))
     ; => (0 1 1 2 3 5 8 13 21)
```

Example 1

```
(define eval-print
  (lambda (X)
    (begin
      (display X)
      (display ``=>'')
      (display (primitive-eval X))
      (newline))))
(eval-print '(letrec ((fact (lambda (n)
  (if (<= n 1) 1 (* n (fact (- n 1))))))))
    (fact 6))) ; (letrec...) => 720
```

Example 2

```
(define list-iter  (lambda (lst)
  (define iter (lambda () (call/cc cs)))
    (define cs
      (lambda (ret)
        (for-each (lambda (element)
          (set! ret (call/cc (lambda
            (resume-here)
              (set! cs resume-here)
              (ret element)))))
          lst)
        (ret 'EOL)))
    iter))
(define it (list-iter '(1 ``foo'' 3)))
(it) ; => 1
(it) ; => ``foo''
```

Cheat sheet

- (quote 1 2 3) == '(1 2 3)
- (list 1 2 3) == '(1 2 3)
- (define f (lambda (x) (* 2 x)))
- (if (equal? 1 2) 19 42)
- (car (cons 1 2 3)) => 1
- (cdr (cons 1 2 3)) => (2 3)
- (display "foobar")