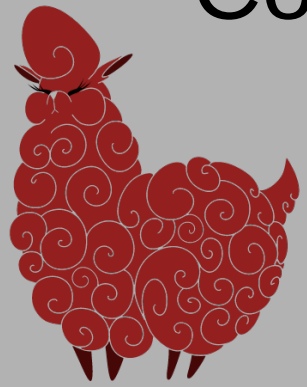


Concurrency

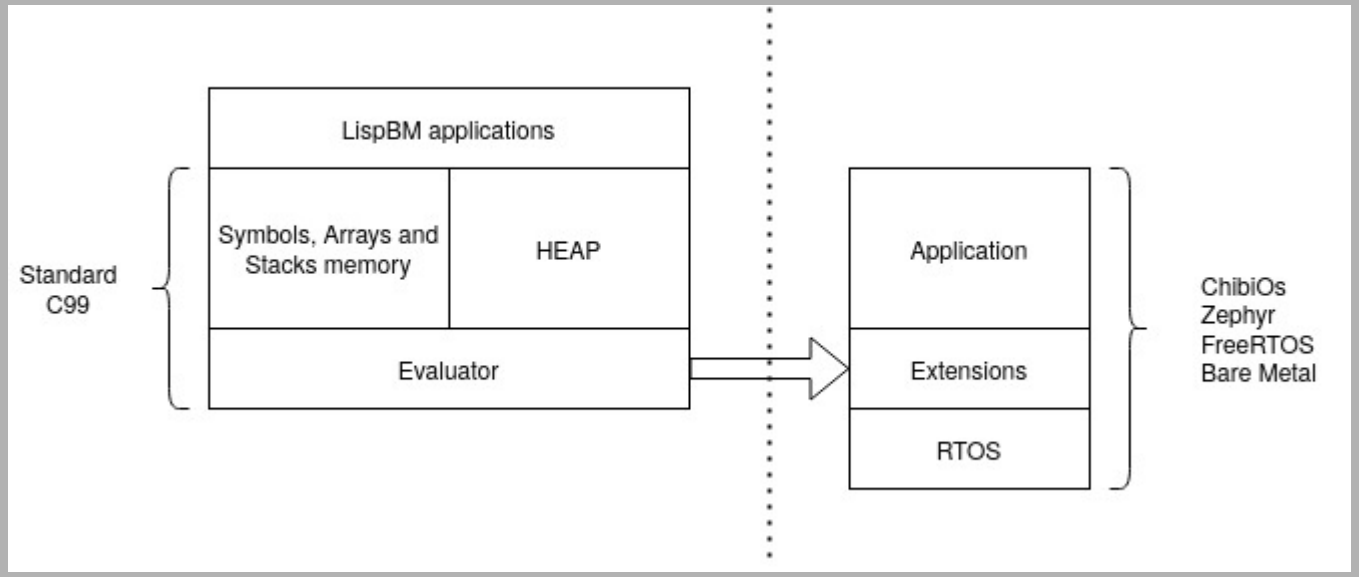
Message-passing

Pattern-matching

Work in progress

LispBM (LBM)

- Hobby coding.
  - A lisp-like language by a non-lisper.
  - Runs on 32bit platforms – microcontrollers.


LispBM applications
Symbols, Arrays and Stacks memory
HEAP
Evaluator
Standard C99
Application
Extensions
RTOS
ChibiOs
Zephyr
FreeRTOS
Bare Metal

Tiny Lisp primer

- There are lots of parenthesis.
  - This is because in lisp lists are enclosed in parenthesis and lists are used everywhere!
  - Your program is a list (nested).
  - Your data is in lists.
- No infix operators.
  - Application is a list and the first element of that list is applied to the rest of the elements of the list as arguments.
  - `(+ 1 2)`
  - `(f a b c)`
  - In applications, the arguments are evaluated before being passed to the function.
- There are “special forms” that look quite like applications but are very different under the hood.
  - (define apa 1)

Tiny Lisp primer

- `'(1 2 3)` is data
- `'(+ 1 2)` is also data
- `(1 2 3)` is an error
- `(+ 1 2)` is 3

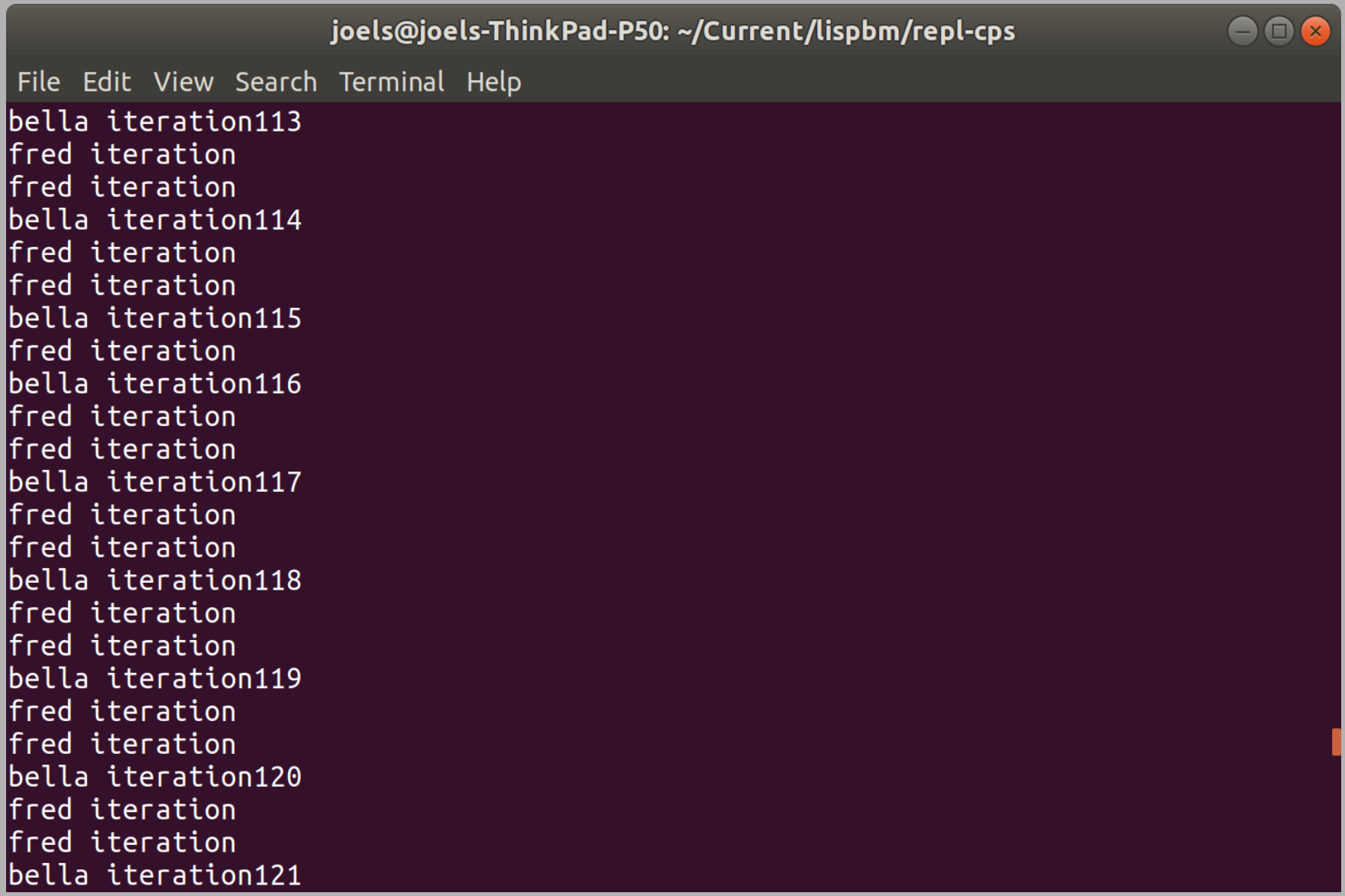
- `` `(1 2 3) `` is data
- `` `(+ 1 2) `` is also data
- `` `(+ 1 ,(+ 1 1)) `` is a mix of data and code

Concurrency

- Cooperative concurrency
  - Processes must be well behaved and go to sleep every now and then.

Concurrency

```
(define fred (lambda ()
           (progn (print "fred iteration" \#newline )
                  (yield 25000)
                  (fred))))

(define bella (lambda (x)
        (progn (print "bella iteration" x \#newline)
               (yield 50000)
               (bella (+ x 1)))))


(spawn '(fred) '(bella 0))
```

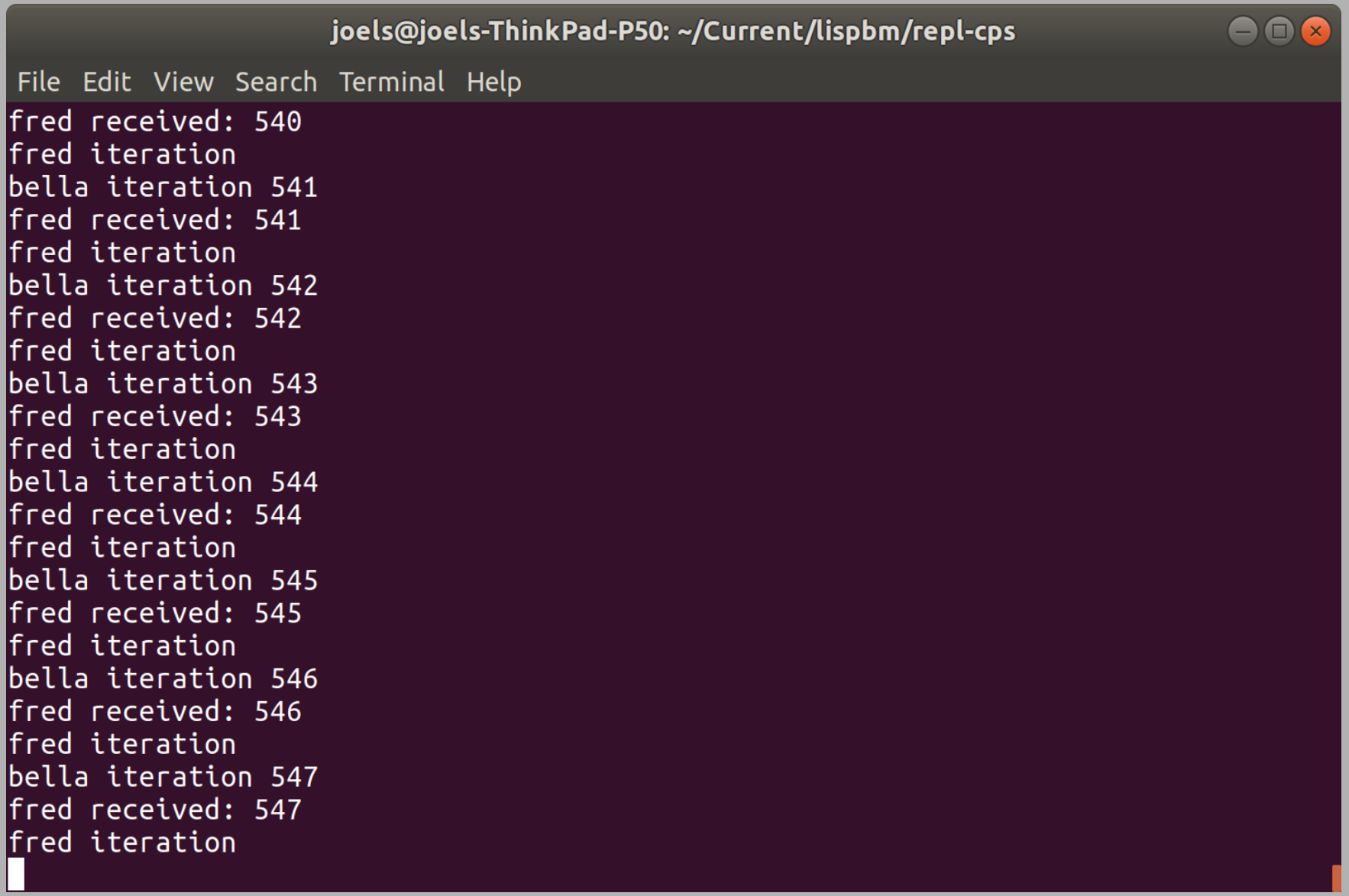

joels@joels-ThinkPad-P50: ~/Current/lispbm/repl-cps
File Edit View Search Terminal Help
bella iteration113
fred iteration
fred iteration
bella iteration114
fred iteration
fred iteration
bella iteration115
fred iteration
bella iteration116
fred iteration
fred iteration
bella iteration117
fred iteration
fred iteration
bella iteration118
fred iteration
fred iteration
bella iteration119
fred iteration
fred iteration
bella iteration120
fred iteration
fred iteration
bella iteration121

Message Passing

- Each Process has a mailbox and messages can be sent to a process if you know the process id.
  - LBM process ↔ LBM process
  - C → LBM process
  - Processes are blocked while waiting for a message.

Message Passing

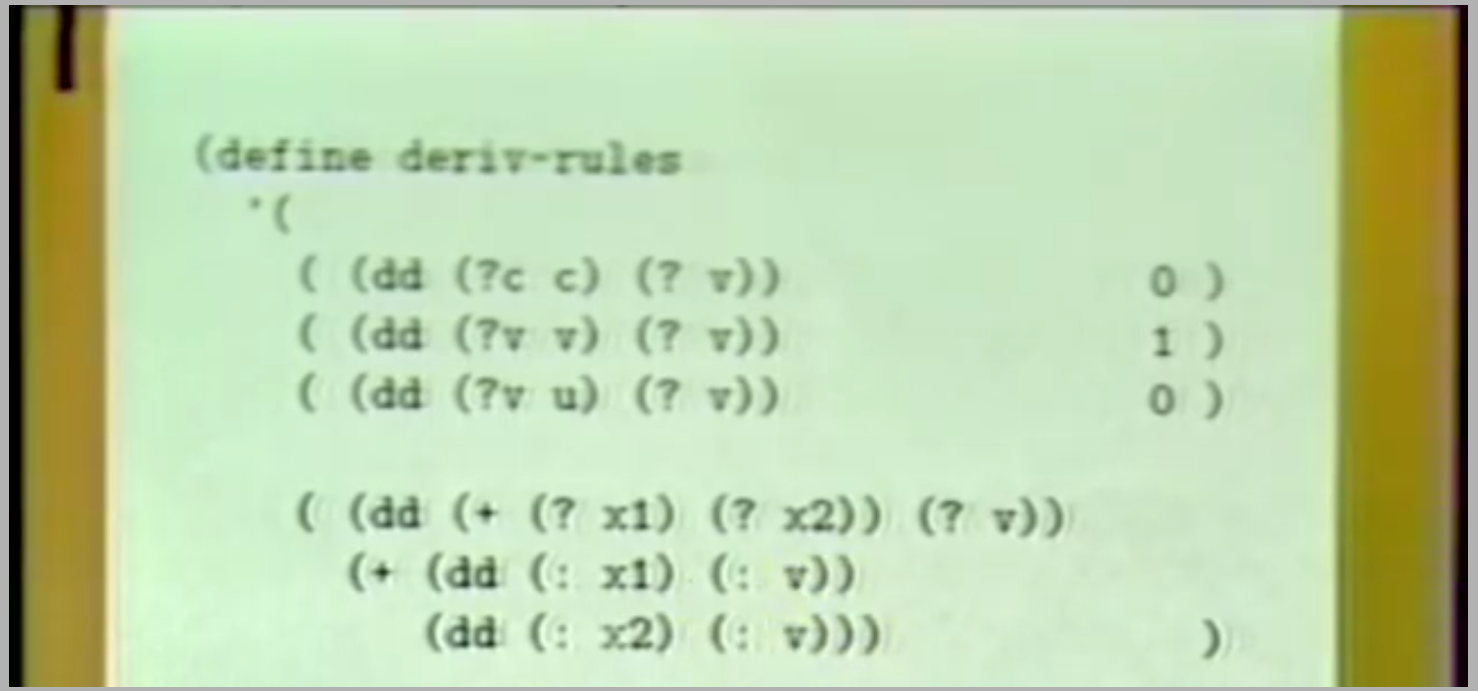
```
(define fred (lambda ()
               (progn (print "fred iteration" \#newline )
                      (recv ( (? x)  (print "fred received: " x \#newline)))
                      (fred))))

(define bella (lambda (pid x)
                (progn (print "bella iteration " x \#newline)
                       (send pid x)
                       (yield 50000)
                       (bella pid (+ x 1)))))

(define fredpid (spawn '(fred)))

(spawn '(bella (car fredpid) 0))
```


joels@joels-ThinkPad-P50: ~/Current/lispbm/repl-cps
File Edit View Search Terminal Help
fred received: 540
fred iteration
bella iteration 541
fred received: 541
fred iteration
bella iteration 542
fred received: 542
fred iteration
bella iteration 543
fred received: 543
fred iteration
bella iteration 544
fred received: 544
fred iteration
bella iteration 545
fred received: 545
fred iteration
bella iteration 546
fred received: 546
fred iteration
bella iteration 547
fred received: 547
fred iteration

Pattern Matching

- Inspired by SICP lecture 4A: https://youtu.be/_fXQ1SwKjDg

But simplified...



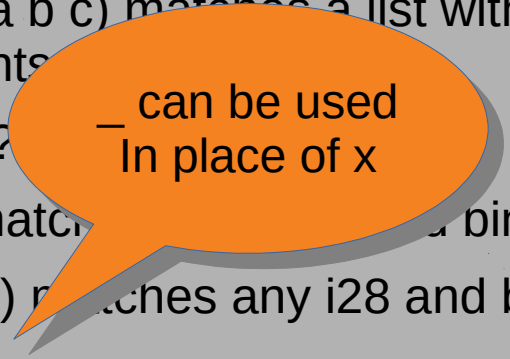
```
(define deriv-rules
  '(
    ( (dd (?c c) (? v))                    0 )
    ( (dd (?v v) (? v))                    1 )
    ( (dd (?v u) (? v))                    0 )

    ( (dd (+ (? x1) (? x2)) (? v))
      (+ (dd (: x1) (: v))
         (dd (: x2) (: v)))                  )
```

Pattern Matching

- Symbol foo matches only symbol foo.
- Values match values that are exactly the same (including type) so: 1i28 matches 1i28 but not 1i32 and so on.
- A list (a b c) matches a list with 3 element where a b c recursively matches with the elements of that list.
- _ and ? matches anything.
- (? x) matches anything and binds that anything to x
- (?i28 x) matches any i28 and binds that i28 value to x (and so on for i32, u28, u32, float).
- (?cons x) matches anything that is built out of a cons cell.

Pattern Matching

- Symbol foo matches only symbol foo.
- Values match values that are exactly the same (including type) so: 1i28 matches 1i28 but not 1i32 and so on.
- A list (a b c) matches a list with 3 element where a b c recursively matches with the elements
- _ and ?
- (? x) matc binds that anything to x
- (?i28 x) matches any i28 and binds that i28 value to x (and so on for i32, u28, u32, float).
- (?cons x) matches anything that is built out of a cons cell.



Pattern Matching

- There are two pattern matching forms.
  - recv
  - match

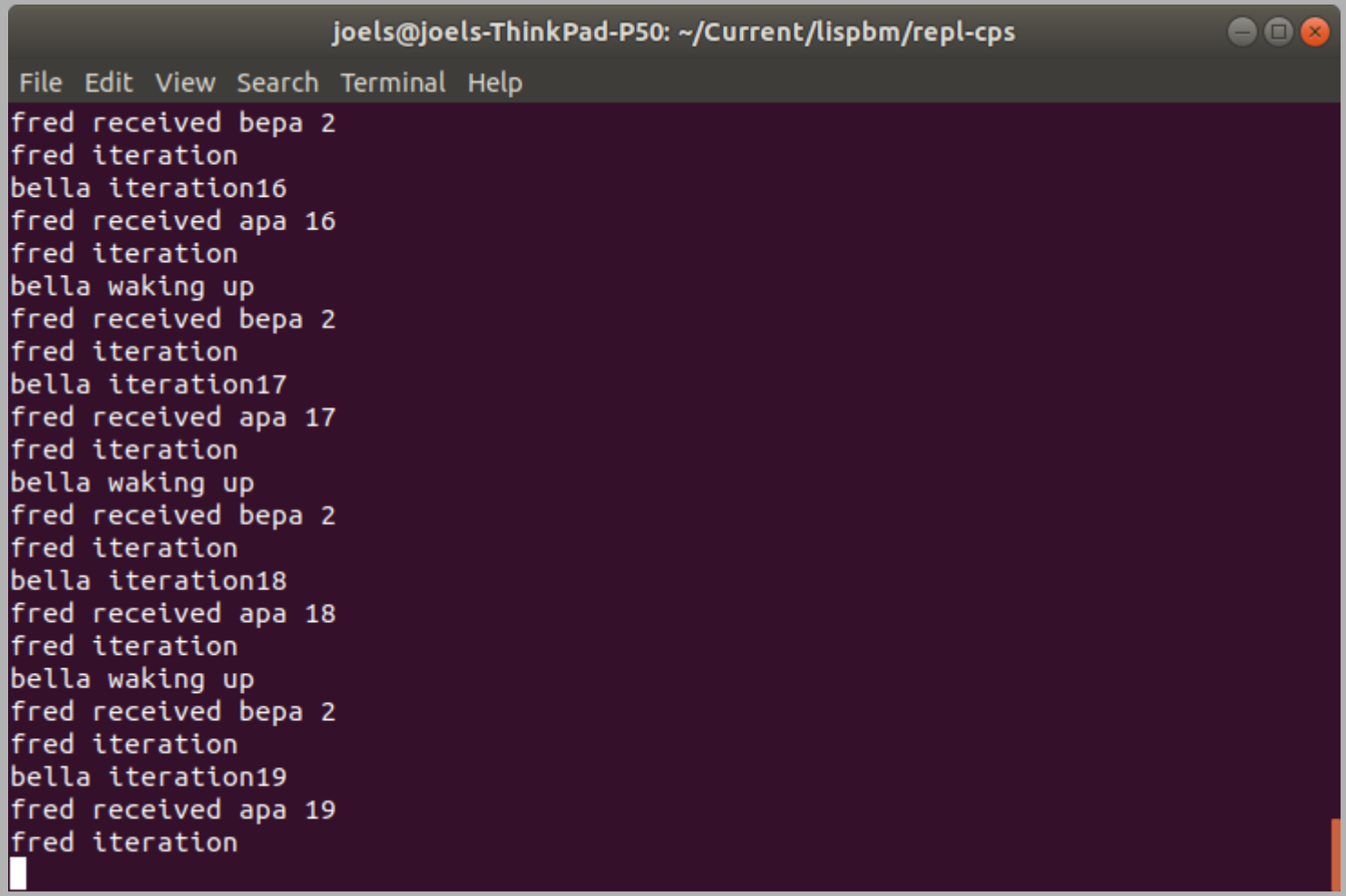
Pattern Matching

- There are two pattern matching forms.
  - recv
  - match

```
(define f (lambda (ls)
            (match ls
                   ( nil 0 )
                   ( (?cons c) (+ (car c) (f (cdr c))))
                   ( _ 'error-not-a-list))))
```

Pattern Matching

```
(define fred (lambda ()
               (progn (print "fred iteration" \#newline )
                      (recv ( (apa (? x) 107)  (print "fred received apa " x \#newline))
                            ( (bepa (?i28 x))  (print "fred received bepa " x \#newline)))
                      (fred))))

(define bella (lambda (pid x)
                (progn (print "bella iteration" x \#newline)
                       (send pid `(apa ,x 107))
                       (yield 500000)
                       (print "bella waking up" \#newline)
                       (send pid '(bepa 2))
                       (yield 500000)
                       (bella pid (+ x 1)))))
```


joels@joels-ThinkPad-P50: ~/Current/lispbm/repl-cps
File Edit View Search Terminal Help
fred received bepa 2
fred iteration
bella iteration16
fred received apa 16
fred iteration
bella waking up
fred received bepa 2
fred iteration
bella iteration17
fred received apa 17
fred iteration
bella waking up
fred received bepa 2
fred iteration
bella iteration18
fred received apa 18
fred iteration
bella waking up
fred received bepa 2
fred iteration
bella iteration19
fred received apa 19
fred iteration

Special forms

```
UINT sym_id = dec_sym(head);

      switch(sym_id) {
      case SYM_QUOTE:   eval_quote(ctx); return;
      case SYM_DEFINE:  eval_define(ctx); return;
      case SYM_PROGN:   eval_progn(ctx); return;
      case SYM_SPAWN:   eval_spawn(ctx); return;
      case SYM_LAMBDA:  eval_lambda(ctx); return;
      case SYM_IF:      eval_if(ctx); return;
      case SYM_LET:     eval_let(ctx); return;
      case SYM_AND:     eval_and(ctx); return;
      case SYM_OR:      eval_or(ctx); return;
      case SYM_MATCH:   eval_match(ctx); return;
      case SYM_RECEIVE: eval_receive(ctx); return;

      default: break; /* May be general application form. Checked below*/
      }
```

```
(match ls
       '( nil 0 )
       '( (?cons c) (+ (car c) (f (cdr c))))
       '( _ 'error-not-a-list))))

(match ls
       (f)
       (g))

(match ls
       '(( nil 0 )
         ( (?cons c) (+ (car c) (f (cdr c))))
         ( _ 'error-not-a-list)))))

(match ls (f))
```

```
(define apa 1)

(define 'apa 1)

(f apa 1)

(f 'apa 1)
```

Future work

- (DONE) The reader (parser) is stack hungry. Can it be improved?
  - Stack can be traded for heap if made tail-recursive.
    - The list reversal can be dropped with some imperative hacks. (set-cdr)
- Pattern matching is a recursive tree comparison. Also stack hungry. But patterns may generally be quite small trees.
  - Rewrite in the same CPS style as the evaluator and reader.
- There is 1k buffer in the GC for the recursion over “trees”.
  - Replace with Pointer reversal GC algorithm. (steal from sensevm)
- Interrupts, input, output
  - Not a priority

Future work

- More testing:
  - Currently run “infer” and clang’s “scan-build”.
    - Very useful!
  - 116 tiny test programs that are run on 14 different configurations of the evaluator. (32768 – 512) cons cells.

```
(= (+ 4i32 7i32) 11i32)
```

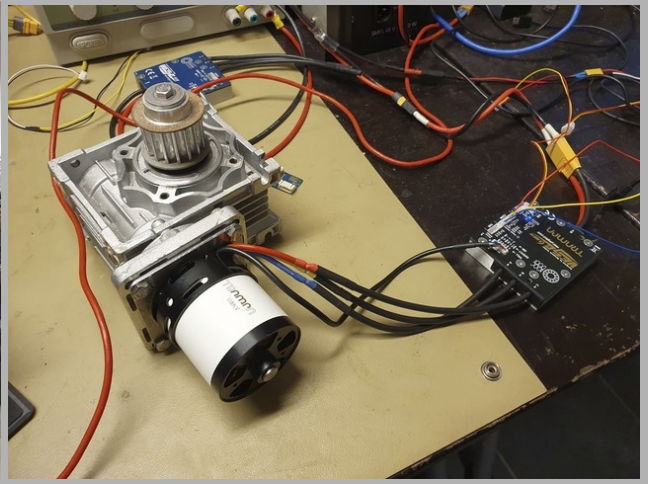
```
(define fold (lambda (f i xs)
                (if (= xs nil)
                    i
                    (fold f (f i (car xs)) (cdr xs)))))

(= (fold '+ 0 (list 1 2 3 4 5 6 7 8 9 10)) 55)
```

Looking for friends

- If anyone here like lisps and wants to chat I would love it.
- The implementation is interesting. If anyone likes strange C code and wants a walk-through, let me know!
- Collaboration? Maybe there is a nugget in this somewhere that we can develop and write about together.

More WIP

- Working with Benjamin Vedder on putting LBM inside of the VESC motor controller as a scripting language.
  - Future Octopi talk by Benjamin is possible
    - In Feb. or later.



https://vesc-project.com/

https://github.com/vedderb/bldc

Benchmarks

File	Load time (s)	Eval time (s)
q2.lisp (q2 6 7)	0.001799999	1.861400008
dec_cnt2.lisp	0.001399999	3.529599905
dec_cnt1.lisp	0.001300000	4.334300041
fibonacci.lisp (fib 23)	0.001300000	4.466599941
dec_cnt3.lisp	0.001300000	1.519999980
tak.lisp (tak 18 12 6)	0.001700000	4.285699844
fibonacci_tail.lisp (fib 23)	0.001900000	0.005200000
Insertionsort.lisp	0.002700000	0.006099999

STM32F405: 168 MHz ARM Cortex M4.