



Lisp Flavoured Erlang LFE

Adding a new flavour to Erlang

Robert Virding



What LFE isn't



- It isn't an implementation of Scheme
- It isn't an implementation of Common Lisp

In fact neither are possible on the Erlang VM

(Global data, destructive operations, ...)



What LFE is



- LFE is a (proper) Lisp based on the features and limitations of the Erlang VM
- LFE is attuned to vanilla Erlang and OTP
- LFE coexists seamlessly with vanilla Erlang and OTP

2008-10-27

3



LFE Features



- The usual good lisp stuff – macros, sexprs, code \leftrightarrow data, `
- Extensive use of pattern matching
- Uses Erlang data types
- Uses Erlang BIFs
- Functions of same name but different arity
- Built on small core extended with macros
- Compiler, interpreter, (simple) shell

2008-10-27

4



Syntax



- Pure lisp sepxrs
- [...] alternative to (...) (Scheme)
- Symbol is any atom which isn't a number
 - |quoted symbol|
- () [] { } . ' ` , , @ # (#b(separators
- #(...) tuple constant
- #b(...) binary constant
- "abc" ⇔ (97 98 99), needs quoting ☹
- #\a or #\xab; characters

2008-10-27

5



Core forms



- (case expr clause ...) ;An erlang case
- (if test true false) ;A lisp if
- (receive clause ... (after timeout body))
- (catch body)
- (try expr (case ...) (catch ...) (after ...))
- (lambda (arg ...) body)
- (match-lambda clause ...)
- (let ...), (let-function ...), (letrec-function ...)
- (cons ...), (list ...), (tuple ...), (binary ...)
- (func arg ...), (funcall var arg ...)
- (call mod func arg ...) ;Eval all args
- (define-function name ...)

2008-10-27

6



Core macros



- (: mod name arg ...) ;Literal mod name
- (flet ...), (fletrec ...)
- (let* ...), (flet* ...)
- (cond ...) ;(?= pat expr)
- (andalso ...), (orelse ...)
- (do ...) ;Scheme
- (lc (qualifier ...) expr ...) ;[expr || qualifier ...]
- (bc (qualifier ...) expr ...) ;<< expr || qualifier ... >>
- (fun name arity), (fun mod name arity)
- (++)
- Bunch of CL inspired macros – defun, defmacro, ...

2008-10-27

7



Lisp-1 vs. Lisp-2



- Tried Lisp-1 in LFE 0.1 but it didn't really work, resulted in funny behaviour
- Core Erlang tries to help but still see difference
- Lisp-2 "fits" Erlang VM better
- So Lisp-2 from LFE 0.2 onwards
- Result more consistent and better (I think)

2008-10-27

8

- In Lisp-1:

```
(define (foo x y) ...)  
(define (bar x y)  
  (let ((foo (lambda (a) ...)))  
    (foo x y)  
    ...))
```
- Which foo should be used?
 - Local foo variable and bad_arity error
 - Global foo/2 and succeed

```
(defun member (x es)  
  (cond ((== es ()) 'false)  
        ((== x (car es)) 'true)  
        (else (member x (cdr es))))))  
  
(defun member  
  ((x (e . es)) (when (== x e) 'true)  
  ((x (e . es)) (member x es))  
  ((x ()) 'false))
```



Function scoping



- Within a module
 - Default predefined Erlang BIFs
 - Explicit imports
 - Top functions in module
 - Local functions defined by flet and fletrec
- So no problem redefining Erlang BIFs or imports. Macros!
- Core forms can **never** be shadowed!

2008-10-27

11



Macros



- Macros are UNHYGIENIC!
 - Does hygiene really work when distributing compiled code?
- No (gensym)
 - Unsafe in long-lived systems
 - But probably must have
- Really only compile time at the moment
 - Except in shell

2008-10-27

12



Macros



- CL based macros, with pattern matching
(defmacro foo (a b) ...)
(defmacro foo
 (pat [guard] ...)
 (pat ...))
- Pattern matches whole argument list
- Scheme based syntax-rule macros with R5RS ellipsis

2008-10-27

13



Binaries



- (binary bitseg ...)
- bitseg = integer | (value bitspec ...)
 - (1.5 float big-endian (size 32))
 - (bin binary)
 - (bits bitstring)
 - ((foo a 35) integer little-endian (size 36))
 - But must do ((foo a 35)) ☹

2008-10-27

14



Patterns



- Like in vanilla Erlang patterns look like constructors
 - (binary (f float (size 32)) (rest binary))
- Use quote ' to match literals
 - (tuple 'ok val)
- But not for lists ☹
 - (a b c)
 - (h . t)

2008-10-27

15



Patterns



- Have aliases
 - (= (tuple 'ok a b) tup)
 - Checked in lint
- Can be used in
 - let, case, receive, match-lambda, macros
 - cond, lc , bc
- Anonymous variable _

2008-10-27

16



and Guards



(when (and (> x 5) (< x 10)))

- Guards are a (when <test>) expression directly after the pattern in clauses
- LFE guards are Erlang guards
- No implicit equality tests for patterns
{X,X} → (tuple x x1) (when (== x x1))
- Can be used after any pattern

2008-10-27

17



Records



(defrecord name field-def-1 field-def-2 ...)
field-def = field-name | (field-name default-value)
→ (make-name field-name val field-name val ...)
(is-name rec)
(match-name field-name pat field-name pat ...)
(name-field-1 rec)
(set-name-field-1 rec val)
...

2008-10-27

18



LFE module



- A module consists of
 - Macro definitions
 - Macro calls
 - Function definitions
 - Compile time function definitions
- Macros can be defined anywhere but must be defined before being used
- Macros can define functions and other macros

2008-10-27

19



LFE module



```
(defmodule foo
  (export (a 2) (b 1) (c 0))
  (export all)
  (import (from bar (x 2) (y 3))
          (rename baz ((m 4) bm)))
  (other-attribute (value)))
```

- Module definition must be the first non-macro form

2008-10-27

20



LFE compiler



- 3 passes
 - Macro expansion
 - Linting
 - Code generation
- Lint and codegen only see LFE core forms
- Generates Core erlang
- LFE core forms \leftrightarrow Core erlang
 - So compiler relatively simple

2008-10-27

21



LFE compiler



- Uses back-end of Erlang compiler
- Output should be closer to Erlang compiler core output \rightarrow better optimisation

2008-10-27

22



LFE shell



- Simple REPL
- Can evaluate all LFE expressions
- Builtin variables + ++ +++ - * ** ***
- Some builtin commands
- (slurp file) to load file and interpret all functions and macros
- Cannot define functions and macros yet
- No (spit file) yet either

2008-10-27

23



The BIG question



Apart from the Answer to Life, the Universe, and Everything

**Will LFE end the complaints
and moaning about Erlang
syntax?**

2008-10-27

24



The answer



42

NO!

2008-10-27

25



Implementing languages on the Erlang VM

A brief description of the Erlang
compiler

Robert Virding



Implement a language



Implement language by:

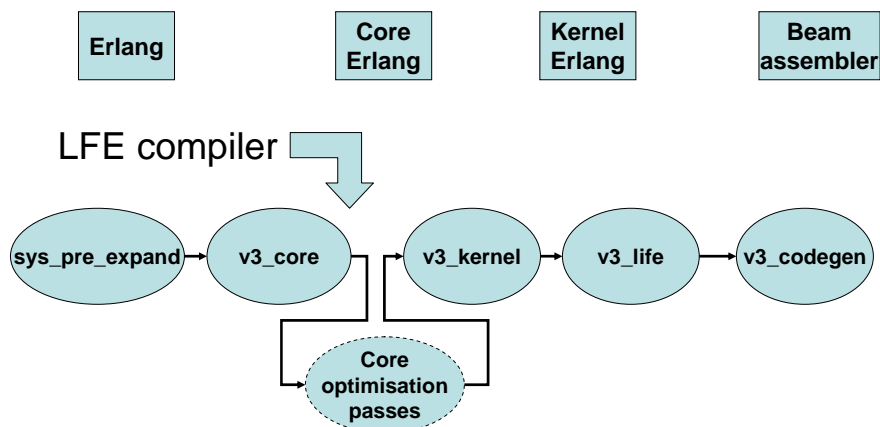
- Writing an interpreter
 - Easier but slower, more versatile
- Compiling to erlang
 - Code format complex, to file?
- Compile to "internal" language
 - Core erlang, kernel erlang

2008-10-27

27



Compiler overview



2008-10-27

28



Erlang compiler



- Core Erlang
 - simple functional language
 - lexically scoped
 - local recursive functions
 - pattern matching
 - basic Erlang constructions (case, try etc.)
 - but misses some useful constructions ☹
 - Erlang features make it slightly strange

2008-10-27

29



Core Erlang forms



- (case expr clause ...) ;An erlang case
- ~~(if test true false)~~ ;A lisp if
- (receive clause ... (after timeout body))
- (catch body)
- (try expr (case ...) (catch ...) (after ...))
- (lambda (arg ...) body)
- ~~(match-lambda clause ...)~~
- (let ...), ~~(let-function ...)~~, (letrec-function ...)
- (cons ...), (list ...), (tuple ...), (binary ...)
- (func arg ...), (funcall var arg ...)
- (call mod func arg ...) ;Eval all args
- (define-function name ...)

2008-10-27

30



Erlang compiler



- Kernel Erlang
 - flat code
 - lambda lifted
 - pattern matching compiled ☺
 - no nested code
 - receive expanded

2008-10-27

31



Erlang compiler



- sys_pre_expand
 - Expand records, packages, annotate funs
- v3_core
 - List comprehensions, add lexical scoping, return exported variables, sequentialise code, expand =, add explicit fail clauses
- v3_kernel
 - Compile pattern matching, lambda lift local functions and funs, flatten nested calls

2008-10-27

32