Implementing Languages in Erlang

Robert Virding
Principle Language Expert
at Erlang Solutions Ltd.
Overview

- A tiny bit of history
- Properties of the BEAM
- What languages?
- Basic Tools
- “Native” languages
- “Non-native” languages
- Interface + demo
Problem domain

- Lightweight, massive concurrency
- Fault-tolerance must be provided
- Timing constraints
- Continuous operation for a long time
- Continuous maintenance/evolution of the system
- Distributed systems
Some reflections

We were NOT trying to implement a functional language
We were NOT trying to implement the actor model

WE WERE TRYING TO SOLVE THE PROBLEM!
Some reflections

• This made the development of the language/system very focused
• We had a clear set of criteria for what should go into the language/system
  – Was it useful?
  – Did it or did it not help build systems?

The language/system evolved to solve the problem
What IS the BEAM?

A virtual machine to run Erlang
Properties of the Erlang system

- Lightweight, massive concurrency
- Asynchronous communication
- Process isolation
- Error handling
- Continuous evolution of the system
- Soft real-time
- Support for introspection and monitoring

These we seldom have to directly worry about in a language, except for receiving messages
Properties of the Erlang system

- Immutable data
- Pattern matching
- Functional language
- Predefined set of data types
- Modules
- No global data

These are what we mainly “see” directly in our languages
What languages?

- Anything written in another language
  - Config files
  - DSLs
  - Other “languages”
  - ...
Basic tools

- leex – lexical scanner generator
- yecc – parser generator
- syntax tools – for building erlang code
- XML parsers (xmerl)
- Erlang compiler (of course)
leex

- Lexical scanner generator
- Based on lex/flex (but simpler)
- Uses regular expressions to define tokens
- Generates scanning functions
  - direct use
    - string/2
  - for the i/o system
    - token/2, tokens/2
leex - example

Definitions.
U = [A–Z]
L = [a–z]
D = [0–9]

Rules.
({U}|{L})({U}|{L}|{D}|_)* :
   {token,{name,TokenLine,list_to_atom(TokenChars)}}.
[+-]?{D}+ :
   {token,{number,TokenLine,list_to_integer(TokenChars)}}.
[;=()*/+-] :
   Class = list_to_atom(TokenChars),
   {token,{Class,TokenLine}}.
[\000–\s] : skip_token.
(#^[\^\n]*) : skip_token.
yecc

- LALR(1) parser generator
- Based on yacc
- Generating parsing functions
  - parse/1
yecc - example

Nonterminals stmt expr term func.
Terminals '+' '-' '*' '/' '(' ')' number var.
Rootsymbol stmt.

stmt → expr : '$1'.
stmt → name '=' expr : {assign,'$1','$3'}. 
expr → expr '+' term : {'+','$1','$3'}. 
expr → expr '-' term : {'-','$1','$3'}. 
expr → term : '$1'.
term → term '*' func : {'*','$1','$3'}. 
term → term '/' func : {'/','$1','$3'}. 
term → func : '$1'. 
func → '(' expr ')': '$2'. 
func → number : '$1'. 
func → name : '$1'.

spell1

- LL(1) parser generator
- Generating parsing functions
- LL(1) grammars more limited than LALR(1) but spell1 has more versatile handling
  - Re-entrant
  - Can handle too many tokens
Other tools

- syntax tools
  - library for creating and working with erlang code
- other parsing systems
  - PEG parsers
    - neotoma
New Skin for the Old Ceremony

- Languages which keep the basic Erlang execution model and data types
  - New syntax
  - Different “packaging”

- Elixir

- LFE (Lisp Flavoured Erlang)
New Skin for the Old Ceremony

- The basic properties of these languages are based on the properties of the Erlang and the Erlang VM
- Make full use of the Erlang/OTP libraries
Erlang compiler

- Can work on files and Erlang abstract code
- Can generate .beam files or binaries
- Has Core, a nice intermediate language
  - Can be input to the compiler
  - simple and regular
  - easier to compile to
New Skin for the Old Ceremony

Elixir compiler

- sys_pre_expand
- v3_core

LFE compiler

- v3_kernel
- v3_life
- v3_codegen

Erlang

Erlang Core

Core optimisation passes

Kernel Erlang

Beam Assembler
Non-native languages

- Languages which are not just basic Erlang
  - different semantics
  - non-Erlang datatypes
  - non-Erlang handling of data
- Can give access to Erlang properties
- Erlog (prolog)
- Luerl (Lua)
Erlog

- Standard prolog, at least a strict subset
- Completely different semantics to Erlang
  - backtracking
  - logical variables
  - unification
- Good mapping between Erlog <-> Erlang data structures
  - except for logical variables
Erlog - syntax

- Can use leex for tokenising
- Grammar needs backtracking
  - Cannot use yecc or any other parser generator
  - Have a hand written parser
Luerl

- Implements standard Lua 5.2

- Lua is
  - Simple, rather neat little imperative language
  - Common scripting language in games
  - Dynamic language
  - Lexically scoped
  - Mutable variables/environments/global data
Luerl - Lua syntax

- Syntax simple and straightforward
- Can use leex for tokenising
- Can use yecc for parsing
  - One reduce-reduce conflict which is solved by having a general case then checking
Interfacing: native languages

• Simple conceptually (just Erlang)
• Usually needs large environment to be usable
  – Libraries
  – REPL
  – Tools
  – Emacs mode
  – …
Interfacing: non-native languages

- Can be compiled or interpreted
- Need an environment to function for all the non-Erlang features
  - Global state
  - Mutable data
  - ...
- Language specific functions
  - PHP
  - “EasyLanguage is an easy-to-learn, but powerful, computer programming language for creating technical indicators and trading strategies for the TradeStation trading platform.”
• Show leex definition
• Show yecc definition
• Show parsing
• Show interpreter
• Function calls
  – reduce–reduce conflicts
Lue VL - Lua datatypes

- nil
- booleans
- numbers (floating point)
- strings
- mutable key-value tables
  - which it uses as tables/arrays/lists/kitchen sink
  - updates are visible everywhere
**Luerl - global data**

- need to manage global data
  - global table store
    - orddict/array/ETS tables
  - environment store
    - array
  - environments
    - tuple
  - tables
    - array + ttdict
Luerl - global data

-record(luerl, {ttab, tfree, tnext,  %Table table, free, next
    ftab, ffree, fnext,  %Frame table, free, next
    g,  %Global table
    stk=[],  %Current stack
    meta=[],  %Data type metatables
    tag  %Unique tag
  }).

-record(tref, {i}).  %Table reference, index
-record(table, {a, t=[], m=nil}).  %Table type, array, tab, meta
Thank you

Robert Virding: robert.virding@erlang-solutions.com
@rvirding