CPS506 - Comparative Programming Languages

Elixir

Dr. Dave Mason
Department of Computer Science
Ryerson University

History

- Joe Armstrong worked at Ericson
- Erlang originally for lab development 1986
- 1995 became production on a phone switch - 9-9's
- 2006 became multi-processor

Overview

- Paradigms
  - Functional
  - Mostly immutable
  - Rich concurrency support
- Syntax
  - Erlang was Prolog-like; Elixir is more conventional
  - Infix multi-precedent operators
  - control structures are pattern matching
  - Functions are defined in modules
  - Only control structures are matching, recursion, list comprehensions
  - fixed arity functions, but can share a name with different arity
  - spawn, receive, send for communication
- Semantics
  - tail recursion is recognized
  - everything returns a value, control are parts of expressions
  - parameters are call-by-value
  - dynamicly typed
  - functions can be spawned as processes - can receive messages
- Pragmatics
  - Elixir runs on Erlang VM
  - byte-code interpreter (BEAM)

Elixir Example

```
defmodule Sequential do
  def square(collection) do
    collection
    |> Enum.map(fn x -> x * x end))
  end
end

result =Sequential.square 1..1000
```

Elixir Example

```
defmodule Sequential do
  def map(collection, func) do
    collection
    |> Enum.map(fn x -> func.(x) end))
  end
end

result =Sequential.map 1..1000, &(&1 * &1)
```

Elixir Example

OO is about manipulating state Elixir is about transforming data

```
defmodule Parallel do
  def pmap(collection, func) do
    collection
    |> Enum.map(fn x -> Task.async(fn -> func.(x) end) end))
    |> Enum.map(&Task.await/1)
  end
end

result = Parallel.pmap 1..1000, &(&1 * &1)
```

Elixir Example

```
defmodule Parallel do
  def pmap(collection, func) do
    collection
    |> Enum.map(&(Task.async(fn -> func.(&1) end)))
    |> Enum.map(&Task.await/1)
  end
end

result = Parallel.pmap 1..1000, &(&1 * &1)
```

Functional Programming

- Object orientation is not the only way to design code.
- Functional programming need not be complex or mathematical.
- The foundations of programming are not assignments, if statements, and loops.
- Concurrency does not need locks, semaphores, monitors, and the like.
- Processes are not necessarily expensive resources.
- Metaprogramming is not just something tacked onto a language.
- Even if it is work, programming should be fun.

from Elixir book

Binding and Immutability

- = is the binding operator
- *not* assignment
- assignment relates to GOTO

Expressions

- `1 + 2`
- `1 < 4.0`
- `:atom` like `#symbol`
- `variable`
- = is pattern-match/binding - once
- `[[3,4],5,6] [3,4|[5,6]]` can have improper lists
- `{2,3,4}`
- `{:valid,[h|t],x} = {:valid,[1,2,3],:blat}`
- generally use function patterns instead
- `if e0 do e1 else e2 end`
- ```
  cond do c1 -> e1
  c2->e2 end
  ```
- ```
  case e0 do p1 -> e1
  p2->e2 end
  ```

Definitions

- `c "matching_function”`
- `Matching_function.number(one)` - can apply as functions to index
- `negate = fn x -> -x end`
- guards

Higher Order Functions & Lists

- `Enum.reduce(numbers, 0, fn x, sum -> x + sum end)`
- `Enum.map(numbers, fn x -> x + 1 end)`
- `Enum.filter(numbers,small)`
- `Enum.all?([0, 1, 2],small)`
- `Enum.any?(numbers,small)`
- `Enum.take_while(numbers,3)`
- `Enum.take_while(numbers,small)`
- `Enum.drop_while(numbers,small).`
- list comprehensions
  - `for x <-[1,2,3,4], x<3, y<-[5,6], y<x, do: {x,y}`
  - generators and filters
  - `import Enum`
  - `deck = for rank <- ’23456789TJQKA’, suit <- ’CDHS’, do: [suit,rank]`
  - `deck |> shuffle |> take(13)`

Loops and Recursion

- tail-calls properly recognized
- loops are simply tail-recursive function calls
- full power of function pattern-matching for loop control

Processes, Concurrency & Failure

- `receive do pattern -> ...`
  `pattern -> end`
- `pid = spawn(fn xxxx)`
- `pid = spawn(Module,:fun,[args])`
- `send pid,:message`
- `link(pid)` leads to signal on exit
- `pid = spawn_link(fn xxxx)`
- `register(:atom,pid)`
- `self`
- `receive ... after`
- `exit(status)`

Map

- `%{key: value}`
- `map[:key]`
- keys can be anything

Keyword lists

- `[key: value]`
- options parameter
- keys must be atoms

Stream

- lazy evaluation
- equivalent to `Enum`
- `concat`, `cycle`, `take`, `drop`

Workflow & `mix`

- `mix` - project builder + package manager
- `mix new` *`projectName`*
- `mix test`
- `iex -S mix`

Macros

- `defmacro`
- hygenic
- `use` keyword usually defines macros
- used in test definition

Servers

- module dynamically loaded into running server
- current/old versions
- existing old code continues
- fully qualified function calls access current code
- another load kills old, moves current to old, new to current
- `-on_load(name/)` allows checking if load should proceed
- there are higher-level task managers in OTP

Evaluation

- Simplicity
  - Size of the grammar
  - complexity of navigating modules/classes
- Orthogonality
  - number of special syntax forms
  - number of special datatypes
- Extensibility
  - functional
  - syntactically
  - defining literals
  - overloading