Control Systems Optimization

Igor Wojnicki

AGH – Univeristy of Science and Technology

2010

Outline

1. Erlang: Useful Stuff

if Example

```
test_if(A, B) ->
    if
        A == 5 ->
            io:format("A == 5~n", []),
            a_equals_5;
        B == 6 ->
            io:format("B == 6~n", []),
            b_equals_6;
        A == 2, B == 3 ->                    %i.e. A equals 2
            io:format("A == 2, B == 3~n", []),
            a_equals_2_b_equals_3;
        A == 1 ; B == 7 ->                   %i.e. A equals 1
            io:format("A == 1 ; B == 7~n", []),
            a_equals_1_or_b_equals_7
    end.
```

case Example~

```
convert_length(Length) ->
    case Length of
        {centimeter, X} ->
            {inch, X / 2.54};
        {inch, Y} ->
            {centimeter, Y * 2.54}
    end.
```

More case Examples I

```
month_length(Year, Month) ->
    %% All years divisible by 400 are leap
    %% Years divisible by 100 are not leap (except the 400 rule
    %% Years divisible by 4 are leap (except the 100 rule above
    Leap = if
        trunc(Year / 400) * 400 == Year ->
            leap;
        trunc(Year / 100) * 100 == Year ->
            not_leap;
        trunc(Year / 4) * 4 == Year ->
            leap;
        true ->
            not_leap
    end,
```

More case Examples II

```
case Month of
    sep -> 30;
    apr -> 30;
    jun -> 30;
    nov -> 30;
    feb when Leap == leap -> 29;
    feb -> 28;
    jan -> 31;
    mar -> 31;
    may -> 31;
    jul -> 31;
    aug -> 31;
    oct -> 31;
    dec -> 31
end.
```

Writing a Module, Aids

- Export selected functions:

```
-export([...]).
```

- Export all functions (suitable upon writing a module):

```
-compile(export_all).
```

- Importing functions:

```
-import(module,functions).
```

Every module implements:

```
module_info()
```

Modules I

- erlang
  - `abs(Number)` (guard)
  - `apply(Module,Function,Args)`
  - `atom_to_list(Atom)`
  - `date()`
  - `display(Term)` nice for debugging
  - `element(N,Tuple)`
  - `float_to_list(Float)`
  - `halt()` `halt(Status)`
  - `hd(List)` (guard) head of the List
  - `integer_to_list(Integer)`
  - `is_*` (guards)
  - `length(List)` (guard)
  - `list_to_atom(List)`
  - `list_to_float(List)`
  - `list_to_integer(List)`

Modules II

- `erlang:loaded()` a list of loaded modules
- `erlang:localtime()`
- `erlang:max(Term1,Term2)`
- `erlang:min(Term1,Term2)`
- `now() -> {MegaSecs, Secs, MicroSecs}`
- `round(Number)` (guard)
- `size(Tuple)` (guard)
- `time()`
- `tl(List)` (guard) tail of the List
- `trunc(Number)`
- `tuple_to_list(Tuple)`

Modules III

- lists
  - `append(L1,L2)` `append(LL)` where `LL` is a list of lists
  - `concat(Things)` returns a string (a list of codes)
  - `delete(Elem,List)`
  - `flatten(List)`
  - `last(List)`
  - `max(List)`
  - `member(Elem,List)`
  - `min(List)`
  - `nth(N,List)`
  - `reverse(List)`
  - `seq(From,To)` `seq(From,To,Step)`
  - `sort(List)`
  - `split(N,List)`
  - `sum(List)`

Modules IV

- io
  - fwrite/format: `format(Format)` `format(Format,Data)` `format(DeviceId,Format,Data)` DeviceID returned by `file:open/2`
    ```
    io:fwrite("|~10.5c|~-10.5c|~5c|~n", [$a, $b, $c]).
    io:fwrite("Hello world!~n", []).
    ```
  - control sequences:
    - `c` number, a character,
    - `f/e/g` floating point number, different notations see manual,
    - `s` string (a list of ascii codes),
    - `w/p/W/P` terms - nice for debugging,
    - `B/b` integer
    - `n` new line

Modules V

```
> io:fwrite("~s ~w ~i ~w ~c ~n",
      ['abc def', 'abc def', {foo, 1},{foo, 1}, 65]).
abc def 'abc def'  {foo,1} A
ok
```

Modules VI

- `fread(Prompt, Format)` `fread(Device, Prompt, Format)`
  - format similar as for fwrite/format (see manual for details)
- `nl()` new line

Modules VII

- file
  - `get_cwd()` `set_cwd(Dir)` current working directory
  - `open(File,Modes)`
    - modes: `read`, `write`, `append`, `binary` (binary mode!), `readahead`, `compressed`
  - `close(IoDevice)`
  - `position(IoDevice,Location)`
  - `pread`, `pwrite`

Modules VIII

- Reading File Example 1

```
get_data(Name) ->
    opening(file:open(Name,[read,read_ahead])).

opening({ok,FD}) ->
            Val=process(FD),
            file:close(FD),
            Val;
opening(Error) -> Error.


process(FD) ->
    case io:fread(FD,'',"~f") of
        {ok,Value} -> io:format("~f~n",Value),
                      process(FD);
        Err -> Err
    end.
```

Modules IX

- Reading File Example 2

```
get_data(Name) ->
     case file:open(Name,[read]) of
        {ok,FD} ->
            Val=process(FD),
            file:close(FD),
            Val;
        Error -> Error
     end.

process(FD) ->
    case io:fread(FD,'',"~f") of
        {ok,Value} -> io:format("~f~n",Value),
                      process(FD);
        Err -> Err
    end.
```

Modules X

- timer
  - `sleep(Miliseconds)`
  - `tc(Module,Function,Arguments)` `tc(Module,Function)`
  - `seconds(S)` -> miliseconds
  - `minutes(M)` -> miliseconds
  - `hours(H)` -> miliseconds
  - `apply_after(Time, Module, Function, Arguments)`

List Comprehensions

```
> [2*X || X <- L ].
[2,4,6,8,10]
```

- `[ F(X) || X <- L]` means: the list of `F(X)` where `X` is taken from the list `L`.
- `[2*X || X <- L ]` means: the list of `2*X` where `X` is taken from the list L.

List Comprehension, Example

```
pythag(N) ->
    [ {A,B,C} ||
        A <- lists:seq(1,N),
        B <- lists:seq(1,N),
        C <- lists:seq(1,N),
        A+B+C =< N,       A*A+B*B =:= C*C
    ].
```

List Comprehension, Example cont. I

```
1> Buy=[{oranges,4},{newspaper,1},{apples,10},{pears,6},
        {milk,3}].
[{oranges,4},{newspaper,1},{apples,10},{pears,6},{milk,3}].
```

Now let's double the number of every item in the original list:

```
2> [{Name, 2*Number} || {Name, Number} <- Buy].
[{oranges,8},{newspaper,2},{apples,20},{pears,12},{milk,6}]
```

List Comprehension, Example cont. II

Assuming that there is a function `shop:cost/1` which calculates actual cost:

```
3> [{shop:cost(A), B} || {A, B} <- Buy].
[{5,4},{8,1},{2,10},{9,6},{7,3}]
```

Now multiply the numbers together:

```
4> [shop:cost(A) * B || {A, B} <- Buy].
[20,8,20,54,21]
```

And sum them up:

```
5> lists:sum([shop:cost(A) * B || {A, B} <- Buy])
123
```

Higher Order Functions: FUNs

```
> X = fun(X) -> X * 2 end.
> X(3)
6
```

This way a reference to a function can be passed:

FUNs, Example

```
8> TempConvert = fun({c,C}) -> {f, 32 + C*9/5};
8>
({f,F}) -> {c, (F-32)*5/9}
8>
end.
#Fun<erl_eval.6.56006484>
9> TempConvert({c,100}).
{f,212.000}
10> TempConvert({f,212}).
{c,100.000}
11> TempConvert({c,0}).
{f,32.0000}
```

FUNs, Examples cont. I

```
foreach(Fun, [First|Rest]) ->
    Fun(First),
    foreach(Fun, Rest);
foreach(Fun, []) ->
    ok.
map(Fun, [First|Rest]) ->
    [Fun(First)|map(Fun,Rest)];
map(Fun, []) ->
    [].

88> Add_3 = fun(X) -> X + 3 end.
#Fun<erl_eval.5.123085357>
89> lists:map(Add_3, [1,2,3]).
[4,5,6]
```

FUNs, Examples cont. II

```
90> Print_City = fun({City, {X, Temp}}) ->
                     io:format("~-15w ~w ~w~n",
                               [City, X, Temp])
                 end.
#Fun<erl_eval.5.123085357>
91> lists:foreach(Print_City, [{moscow, {c, -10}},
{cape_town, {f, 70}}, {stockholm, {c, -4}},
{paris, {f, 28}}, {london, {f, 36}}]).
moscow          c -10
cape_town       f 70
stockholm       c -4
paris           f 28
london          f 36
ok
```

FUNs, Examples cont. III

```
14> Even = fun(X) -> (X rem 2) =:= 0 end.
15> Even(8).
true
16> Even(7).
false
17> lists:map(Even, [1,2,3,4,5,6,8]).
[false,true,false,true,false,true,true]
18> lists:filter(Even, [1,2,3,4,5,6,8]).
[2,4,6,8]
```

Returning FUNs I

```
1> Fruit = [apple,pear,orange].
[apple,pear,orange]
2> MakeTest = fun(L) ->
                (fun(X) -> lists:member(X, L) end)
              end.
#Fun<erl_eval.6.56006484>
3> IsFruit = MakeTest(Fruit).
#Fun<erl_eval.6.56006484>
4> IsFruit(pear).
true
5> IsFruit(apple).
true
6> IsFruit(dog).
false
```

Returning FUNs II

```
7> lists:filter(IsFruit, [dog,orange,cat,apple,bear]).
[orange,apple]
```