Industrial Functional Programming [1]

Melinda Tóth, István Bozó

Dept. Programming Languages and Compilers
Eötvös Loránd University, Budapest, Hungary

[1] Supported by TÁMOP-4.1.2.A/1-11/1-2011-0052

Contents

Built In Functions (BIF)

- Implemented in C
- Interface in module “erlang”
- Low level operations, performance
- Functions with side-effect
- etc.
- `hd/1, tl/1, length/1`
- `size/1, element/2, setelement/2`
- `list_to_tuple/1, ineteger_to_list/1, ...`
- `is_atom/1, is_list/1, ...`
- `spawn/1, send/2, node/0, self/0, ...`

Documentation And STDLIB

- array, lists, dict, string, queue
- erlang
- io
- file, filename
- random, math
- `http://www.erlang.org/doc/man/`

Recursion

- Primitive recursion: sum
- Tail-recursion: sum_acc

Primitive Recursion

```
-module(a).
-export([sum/1]).

sum(0) -> 0;
sum(N) ->
    N + sum(N-1).
```

Tail-Recursion

```
-module(a).
-export([sum/1]).

sum(N) -> sum(N, 0).

sum(0, Acc) -> Acc;
sum(N, Acc) ->
    sum(N-1, N+Acc).
```

Tail-Recurssive Fibonacci

```
fib(N) ->
    fib(N, 1, 0).

fib(0, Acc1, Acc2) ->
    Acc2;

fib(1, Acc1, Acc2) ->
    Acc1;

fib(N, Acc1, Acc2) ->
    fib(N-1, Acc2+Acc1, Acc1).
```

Lists

- `http://www.erlang.org/doc/man/`
- `--` and `++`
- `length/1, tl/1, hd/1`
- Module `lists`: `{max/1, sum/1, nth/2, last/1, reverse/1, member/2, delete/2, sort/1, usort/1, zip/2, split/2}`
- Module `proplists`: `[{apple, 1}, {cat, 2}, {school, 3}]`

List Comprehension

```
[what to produce?
      || from which element?, what kind of?]
```

- `['elements' || generator1, ..., filter1, ...]`
- generator: `Pattern <- List`
- filter: `Boolean expressions`

Square of even numbers

```
[math:pow(X,2)||
        X <- lists:seq(1, 1000), (X rem 2) == 0]

[math:pow(X,Y)||
        X <- lists:seq(1, 1000),
        Y <- [1,2,3,4],
        (X rem 2) == 0,
        X > 500]
```

QuickSort

```
quicksort([])    ->
    [];
quicksort(List) ->
    quicksort([X || X <- List, X =< Y])
      ++ [Y]
        ++ quicksort([X || X <- List, X > Y])
```

Primitive Recursion

```
-module(a).
-export([sum/1]).

sum([]) -> 0;
sum([Head | Tail]) ->
    Head + sum(Tail).
```

Tail-Recursion

```
-module(a).
-export([sum/1]).

sum(List) -> sum(List, 0).

sum([], Acc) -> Acc;
sum([Head | Tail], Acc) ->
    sum(Tail, Head+Acc).
```

On the Next Lecture ...

- Guards
- Conditional evaluation
- Fun Expressions
- Dynamic function calls