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

Types of Run-time Errors

- `function_clause`
- `case_clause`
- `if_clause`
- `badmatch`
- `badarg`
- `undef`
- `badarith`
- `badfun`
- `badarity`

Types of Run-time Errors

-
  ```
  1> lists:max([]).
  ** exception error: no function clause matching
     lists:max([]) (lists.erl, line 313)
  ```

-
  ```
  2> X = 1.
  1
  3> X = 2.
  ** exception error: no match of right hand side
     value 2
  ```

Types of Run-time Errors

-
  ```
  5> length({}).
  ** exception error: bad argument
       in function  length/1
          called as length({})
  ```

-
  ```
  7> lists:maxxx([]).
  ** exception error: undefined function
     lists:maxxx/1
  ```

-
  ```
  8> 1 + apple.
  ** exception error: an error occurred when
     evaluating an arithmetic expression
       in operator  +/2
          called as 1 + apple
  ```

Trapping Run-time Errors

```
catch Expr
```

-
  ```
  1> catch lists:max([]).
  {'EXIT',{function_clause,[{lists,max,
                                [[]],
                                [{file,"lists.erl"},
                                 {line,313}]},
                               ...]}}
  ```

-
  ```
  2> X = 1.
  1
  3> catch X = 2.
  {'EXIT',{{badmatch,2},[{erl_eval,expr,3,[]}]}}
  ```

Trapping Run-time Errors

-
  ```
  5> catch length({}).
  {'EXIT',{badarg,[{erlang,length,[{}],[]},
                   ...]}}
  ```

-
  ```
  7> catch lists:maxxx([]).
  {'EXIT',{undef,[{lists,maxxx,[[]],[]},
                  ...]}}
  ```

-
  ```
  8> catch 1 + apple.
  {'EXIT',{badarith,[{erlang,'+',[1,apple],[]},
                     ...]}}
  ```

Pattern Matching on the value

```
case catch M:F(1,2) of
     {'EXIT', {undef, _} ->
         "Funtion is not defined";
     Value ->
         {return_value, Value}
end
```

Trapping Run-time Errors

```
try ExprList of
    Pattern1 [when Guard1] -> ExprList1;
    ...
    PatternN [when GuardN] -> ExprListN
catch
    Class1:ExcPattern1 [when ExcGuard1] ->
        ExcExprList1;
    ...
    ClassK:ExcPatternK [when ExcGuardK] ->
        ExcExprListK
after
    LastExprList
end
```

Classes: `error, throw, exit`

Trapping Run-time Errors

```
try M:F(1,2) of
    Value -> {return_value, Value}
catch
    error:undef ->
        "Funtion is not defined"
end
```

Raising Exeptions

- `erlang:error(new__error)`
- `throw(new__exeption)`
- `exit(program__exited)`

Records

- Creating new data structures
- Have to be defined before the first use
- Represented as tagged tuples

-
  ```
  -record(name, {field1 [= default1],
                   ...,
                 fieldn [= defaultn]}).
  ```
- Creating records:
  `RecExpr = #name{..., fieldi = Value, ...}`
- Record field access:
  `RecExpr#name.fieldi`
- Record update:
  `RecExpr#name{..., fieldi = NewValue, ...}`

Records

-
  ```
  -record(name, {field1 [= default1],
                           ...,
                         fieldn [= defaultn]}).
  ```
- Record filed index:
  `#name.fieldi`
- Record patterns:
  `RecExpr = #name{..., fieldi = Value, ...}`
- Shell functions: `rr/1, rd/2, rl/1, rf/1`
- BIFs: `record_info/2 (size, fields), is_record/2`

Records

```
-record(date, {month, day, year = 1900}).

create() ->
    #date{year=2012, day = 20, month = 12}.

select(Year, [Rec = #date{year = Year} | Tail]) ->
    {month, Rec#date.month};
select(_Year, [_ | Tail] ->
    select(Tail);
select(_Year, []) ->
    not_found.
```

“Lazy” list

```
next(Seq)->
  fun()-> [Seq | next(Seq + 1)] end.

SeqFun0 = next(0).
[Seq1 | SeqFun1] = SeqFun0().
```

...

On the Next Lecture ...

- Macros
- Binaries
- Input/Output