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

1. Guards
2. Conditional Evaluation
3. Fun Expressions
4. Dynamic Function Calls

Guard Expressions

- ',' and ';'
- Bound variables
- Literal
- Comparison
- Arithmetic expression
- Boolean expression
- Type checking
- guard built in functions (subset of BIFs)

Guard Expressions

- `A == {2, 3}, B + 2 > 2; C > 3`
- `(A == {2, 3}) and (B + 2 > 2) or (C > 3)`
- `X andalso Y, X not Y`
- `is_atom(A), is_list(B)`
- `list_to_atom(A) == 'an atom'`

Case Expression

- ```
  case Expr of
      Pattern1 [when Guard1] -> ExprList1;
      ...
      PatternN [when GuardN] -> ExprListN
  end
  ```
- Scope of a variables
  - variables bound outside of the case expression can be used
  - if the variable bound in every clause it can be used outside of the case expression
- Joker pattern: _
  ```

Case Expression

```
case lists:reverse(List) of
    [] -> Res = [];
    [H] -> Res = [];
    [_ | Tail] -> Res = Tail
end,
lists:reverse(Res)
-----------------------------------
Res = case lists:reverse(List) of
          [] -> [];
          [H] -> [];
          [_ | Tail] -> Tail
      end,
lists:reverse(Res)
```

If Expression

- ```
  if
      Guard1 -> ExprList1;
      ...
      GuardN -> ExprListN
  end
  ```
- Scope of a variables
  - variables bound outside of the if expression can be used
  - if the variable bound in every clause it can be used outside of the if expression
- Joker guard: `true`
  ```

If Expression

```
if
    X > Y -> {X-Y, Y};
    X < Y -> {X, Y-X};
    true  -> {X, X}
end
```

Explicit Fun Expressions

- ```
  fun(Pattern11,...,Pattern1M) [when Guard1] ->
      ExprList1;
      ...
     (PatternN1,...,PatternNM) [when GuardN] ->
      ExprListN
  end
  ```
- Scope of a variables
  - variables bound outside of the fun expression can be used
  - variables bound in the patterns of the fun expression hide the variables bound outside of the fun expression
- Joker pattern: _
  ```

Explicit Fun Expressions

```
Divisors =
    fun(N) ->
        [ X || X <- lists:seq(1,N), (N rem X)==0]
    end,
Prime =
    fun(N) when N == 0; N == 1 ->
        false;
       (2) ->
        true;
       (N) ->
        Divisors(N) == [1,N]
    end,
Prime(5),
[X || X <- lists:seq(1,N), Prime(X)]
```

Implicit Fun Expressions

```
fun ModName:FunName/Arity
fun         FunName/Arity
```

Implicit Fun Expressions

```
divisors(N) ->
    [ X || X <- lists:seq(1,N), (N rem X)==0].

prime(N) when N == 0; N == 1 ->
    false;
prime(2) ->
    true;
prime(N) ->
    divisors(N) == [1,N].

lists:filter(fun prime/1, lists:seq(1,N))
```

Dynamic constructs

- `apply(FunExpr, [Param1, ..., ParamN])`
- `apply(Mod, Fun, [Param1, ..., ParamN])`
- `Fun(Param1, ..., ParamN)`
- `Mod:Fun(Param1, ..., ParamN)`

Dynamic constructs

```
case read_par() of
    {M, F} when is_atom(F), is_atom(M) ->
        M:F();
    {M, F, A} when is_list(A) ->
        apply(M, F, A)
end
```

On the Next Lecture ...

- Run-time Errors
- Records
- Lazyness