ERLANG BITS AND PIECES

Curt Clifton
Rose-Hulman Institute of Technology

Update ErlangInClass, open bap.erl

GUARDS

- Guards are boolean-valued Erlang expressions, used in
  - Function definitions:
    ```
    max(X,Y) when X > Y -> X;
    max(_,Y) -> Y.
    ```
  - Case expressions
  - If expressions

RESTRICTIONS ON GUARDS

- Allowed:
  - *true*, *false*
  - Constants, including variable references
  - Guard predicates and built-in functions(*is_list()*, *length(L)*, …)
  - Comparisons (>,=,…)
  - Arithmetic (+, -, …)
  - Boolean expressions (*and*, *andalso*, *not*, …)
- Imposed to prevent side-effects during pattern matching
  - What! I thought Erlang was purely functional!

CASE EXPRESSIONS

optional

- Syntax: case *expr* of
  Pattern1 [when *Guard1*] -> *Expr_seq1*;
  Pattern2 [when *Guard2*] -> *Expr_seq2*;
  …
  end
- Example: case solveValid(substFirst(Puzzle, A)) of
  {ok, Answer} -> {ok, Answer};
  {fail, _} -> solve(Puzzle, Remaining)
  end

IF EXPRESSIONS

- Syntax:

```
if
    Guard1 -> Expr_seq1;
    Guard2 -> Expr_seq2;
    ...
end
```

- Example:

```
if
    (Mismatch == "") -> io:format(".");
    true -> io:format("Error!", [])
end
```

Remember, guards are restricted!

RAISING EXCEPTIONS IN ERLANG

- exit(*Why*)
  - Kills the process and broadcasts a "death certificate" to all associated processes
- throw(*Why*)
  - Used for exceptions that caller should catch
- erlang:error(*Why*)
  - We're all going to die!!!

CATCHING EXCEPTIONS

```
try ExprSeq of
    Pattern [when Guard] -> ExprSeq;
    ...
catch
    ExKind: ExPattern [when ExGuard] -> ExprSeq;
    ...
after
    ExprSeq
end
```

One of throw,
exit, or error

EXCEPTION IDIOM: WHEN ERRORS EXPECTED

```
case f(X) of
    {ok, Val} -> do_something_with(Val);
    {error, Why} -> handle_error(Why)
end
```

or

```
{ok, Val} = f(X),
do_something_with(Val)
```

Q2

OTHER EXCEPTION IDIOMS

- When errors are possible but rare, use *throw* and *try-catch*
- Catching all **thrown** exceptions

```
try Expr
catch
    _ -> …
end
```

- Catching **all** exceptions

```
try Expr
catch
    _:_ -> …
end
```

BUILT-IN FUNCTIONS–BIFS

- Used like regular functions, but natively implemented
- Many do things that can't be implemented as regular functions, like
  - Interact with OS (e.g. date and time, file I/O)
  - Convert between tuples and lists
  - Efficiently manipulate "binaries"
- See the *erlang* module

SOME COMMON BIFS

- apply(FunName, Args)
- *F*_to_*G*(X), is_*F*()
  - *F*, *G* ∈ {atom, list, tuple, term, binary, integer, float}
- date(), time(), now()
- element(N, Tuple)
- erlang:get_stacktrace()
- hd(), tl()

Q3

BINARIES

- Compactly store and efficiently reference large quantities of data
- Written like <<*240,128,42*>>
- Useful BIFs:
  - list_to_binary(IoList), flattens binaries and lists of ints, to any level of nesting
  - split_binary(Bin,Pos)
  - term_to_binary(Term), binary_to_term(Binary)

PATTERN MATCHING WITH BINARIES

- Called the "bit syntax", lets us easily manipulate packed binary data
- Syntax: <<*E*, …>>
  - Where each *E* is *Value* or *Value*:*Size*,
  - *Value* is an expression that evaluates to an integer, or a variable for pattern matching,
  - and *Size* is a number of bits
- Sum of *Sizes* must be divisible by 8

Created for network protocol programming.

Q4

MODULE ATTRIBUTES

- We've seen a couple:
  - *-module(modname).*
  - *-export([Name1/Arity1, Name2/Arity2, …]).*
- Others:
  - *-import(Mod, [Name1/Arity1, Name2/Arity2, …]).*
  - *-compile(Options)*
    - See compile module manual page for details
    - sudoku uses *-compile(export_all)*

Q5

MAKING FUNCTIONS FIRST CLASS

- Use *lists:map* to map the *days_until* function across a list of dates
- Need a way to make days_until first class
- Syntax:
  - *fun* LocalFunc/Arity
  - *fun* Mod:RemoteFunc/Arity

LIST OPERATIONS

- ++ appends two lists
- -- does (multi-)set subtraction
- How might set subtraction be useful for Sudoku?

MATCH OPERATOR IN PATTERNS

- Can bind whole subpattern matches to variables
- separation({circle, P1, Rad}, {point, X,Y}}) ->
  separation(P1, {point, X,Y}) - Rad.
- Better:
  separation({circle, P1, Rad}, {point, X,Y} **= P2**}) ->
  separation(P1, P2) - Rad.

PROCESS DICTIONARY

- A private, **mutable** data store for each process
- An associative array (a.k.a., map, hashmap, hashtable, dictionary)
- Process dictionary BIFs:
  - put(Key, Value)
  - get(Key), get(), get_keys(Value)
  - erase(Key), erase()

Generally avoid process dictionaries. But good for write-once, process-global data

COMPARISON

- >, <, =<, >=
  - Also work on unlike terms:
    - *100 < one_hundred*
    - *1000 < one_hundred* too!
- ==, /=
  - Only use for comparing floats and integers
- =:=, =/=
  - Almost always want to use these instead

WARNING: Pattern matching is **exact**.
f(0) -> "boo". doesn't match f(0.0)

Q6