Assignment 1

Advanced Functional Programming, 2017
(Avancerad funktionell programmering, 2017)

due 20 November 2017, 23:59

1 Road Construction (`road.erl` & `road.pdf`, 4 + 2 points)

In a place not so far away, the local authority wants to construct a road of length L ($1 \leq L \leq 1\,000\,000\,000$) units of measure. Unfortunately, the construction work is not so well organized. Each day, the construction team fixes some continuous segment of the road, with known start and end points, but the segment that gets constructed during the next day may start at some totally unrelated point or even partly overlap with segments of the road that were previously constructed. Luckily, we know the exact number of days N ($1 \leq N \leq 1\,000\,000$) the construction is to take place as well as the start and end (S_k and E_k) points of the road segments that will be constucted during these N days.

Given this information and a number X ($0 \leq X \leq L$), we want to find the number of days after which the largest consecutive segment of the road that is still unconstructed will not be larger than X. If a segment larger than X remains unconstructed even after construction is finished (i.e., after all N days have passed), then your program should return `-1`.

Task

Write a program in Erlang (`road.erl`), which defines a function `days/3` that returns the result described above. (**4 points**) To get the maximum number of points for this part, your program needs to be *efficient*, besides correct: it should return an answer in 10 seconds or less. Your solution should include tests, including property-based tests, for appropriately chosen parts of the implementation. Your submission must also include a report (`road.pdf`) explaining the algorithm you used and the properties you tested. (**2 points**)

Examples

Below we show two sample calls for the function:

```
1> road:days(30, [{1,5},{11,27},{2,14},{18,28}], 6).
2
2> road:days(30, [{1,5},{11,27},{2,14},{18,28}], 1).
-1
```

In both examples, the length L of the road to be constructed is 30, the construction lasts four days and involves the same list of segments. After the first day, the largest unconstructed segment (`{5,30}`) has length 25. After the second day, the largest unconstructed segment (`{5,11}`) has length 6. After the third day, the the largest unconstructed segment is `{27,30}` which has length 3. This segment gets reduced by one in the last day, so the largest unconstructed segment after all four days have passed has length 2. Thus, it's after the second day that the largest unconstructed segment is not larger than 6, which explains the answer `2` in the first example. The largest unconstructed segment will never become less or equal to 1, which explains the `-1` answer in the second example.

2 Approximating Pi (`calc_pi.erl`, 4 points)

In this exercise, you will be applying parallel programming in **Monte Carlo method**[1] to calculate Pi. You will generate N random 2D points in range $[0, 1]$, and count the number of points falling within the circle inscribed in the unit square, say C. Then, Pi could be approximated as $Pi = 4 * C/N$.

Task

Your solution must export a function `calc/2`. Its first argument, `N`, is the number of simulated points, and its second argument, `Schedulers`, is the number of schedulers to use. The function should return the approximated Pi value after `N` simulations.

Grading

Your implementation will be graded on a machine with at least eight cores against a reference implementation, which utilizes all `Schedulers` fully. A list of machines to use in order to benchmark your implementation can be found here[2]. A sample grader, which merely compares the execution time without validating the calculated Pi, is provided, **calc_pi_grader.beam**[3].

Example

```
$ erl -noshell -s calc_pi_grader calc_sample_grader -s init stop
For 1 scheduler(s) score is 0.97
For 2 scheduler(s) score is 1.00
For 4 scheduler(s) score is 1.00
For 8 scheduler(s) score is 1.00
Total Score: 4
```

[1] https://en.wikipedia.org/wiki/Monte_Carlo_method
[2] http://www.it.uu.se/datordrift/maskinpark/linux
[3] http://www.it.uu.se/edu/course/homepage/avfunpro/ht17/calc_pi_grader.beam

3 Vector Calculator Server (`vector_server.erl`, 6 points)

Task

Following the tutorial of Chapter 3 of the book "Erlang and OTP in Action", which is available at https://manning-content.s3.amazonaws.com/download/0/8c4c508-6e21-4e8b-ab22-ba9ff22f27e2/sample_Ch03_Erlang.pdf, implement in Erlang a simple RPC server that evaluates vector expressions given in the language described below. After a connection has terminated, the server should wait for a new connection.

Language

$$
\begin{aligned}
\langle \textit{top} \rangle &::= \langle \textit{expr} \rangle \\
\langle \textit{expr} \rangle &::= \langle \textit{vector} \rangle \\
&\mid \{\langle \textit{vector-op} \rangle, \langle \textit{expr} \rangle, \langle \textit{expr} \rangle\} \\
&\mid \{\langle \textit{scalar-op} \rangle, \langle \textit{int-expr} \rangle, \langle \textit{expr} \rangle\} \\
\langle \textit{vector} \rangle &::= [\langle \textit{integer} \rangle, \ldots] \\
\langle \textit{int-expr} \rangle &::= \langle \textit{integer} \rangle \\
&\mid \{\langle \textit{norm} \rangle, \langle \textit{expr} \rangle\}
\end{aligned}
$$

$$
\begin{aligned}
\langle \textit{vector-op} \rangle &::= \text{'add'} \\
&\mid \text{'sub'} \\
&\mid \text{'dot'} \\
\langle \textit{scalar-op} \rangle &::= \text{'mul'} \\
&\mid \text{'div'} \\
\langle \textit{norm} \rangle &::= \text{'norm\_one'} \\
&\mid \text{'norm\_inf'}
\end{aligned}
$$

Vector language semantics

- The binary vector operations are: addition, subtraction, and a **non-aggregated** "dot product"-like operator which is just a pairwise multiplication of the vectors' elements.
- **mul** is multiplication and **div** is integer division of all vector elements with an integer.
- **norm_one** or "Taxicab norm" for vectors is defined as $\|\mathbf{x}\|_1 := \sum_{i=1}^{n} |x_i|$.
- **norm_inf** or "Maximum norm" for vectors is defined as $\|\mathbf{x}\|_\infty := \max(|x_1|, \ldots, |x_n|)$.
- Integers are not bounded.

Evaluation rules

- The evaluation results in a vector, unless it fails.
- The evaluation **must fail** if:
  - An integer division with 0 is attempted.
  - Any vector in the input has a number of elements that is not between 1 and 100.
  - An expression is nesting deeper than 100 levels. (For example, `{'add', [1], [2]}` has level 0.)
  - The sizes of vectors in a binary vector operation (i.e., 'add', 'sub', 'dot') are not equal.

Input/Output

- The server's input is a string from the language above. Whitespace is not important. Consider parsing the string as an Erlang term before evaluating it.
- The response should be the result of the evaluation: a vector if the evaluation is successful or the message "error" if the evaluation has failed. Printing response to the socket **must** be done using `io_lib:fwrite("Res: ~w~n", [Result])`, assuming `Result` is what you want to print.

Sample

Similar to the tutorial's example, the server should be started from the Erlang shell with:

```
1> vector_server:start_link().
{ok,<0.35.0>}
```

After the server has started, you can use `telnet` to communicate:

```
$ telnet localhost 1055
Trying 127.0.0.1...
Connected to localhost.
Escape character is '^]'.
[1,2,3,4,5]
Res: [1,2,3,4,5]
{'dot', [6,6,6], [7,7,7]}
Res: [42,42,42]
{'mul', {'norm_one', [1,-1,2,-2]}, [7,-7,7]}
Res: [42,-42,42]
{'div', 0, [1,2,3,4,5]}
Res: error
```

Connection termination

The server described in the tutorial cannot handle connection termination correctly. Your implementation should take care of the messages received when the client closes the socket and wait for a new connection to be established.

```
$ telnet localhost 1055
Trying 127.0.0.1...
Connected to localhost.
Escape character is '^]'.
[1,2,3,4,5]
Res: [1,2,3,4,5]
{'dot', [6,6,6], [7,7,7]}
Res: [42,42,42]
^]

telnet> quit
Connection closed.
$ telnet localhost 1055
{'mul', {'norm_one', [1,-1,2,-2]}, [7,-7,7]}
Res: [42,-42,42]
{'div', 0, [1,2,3,4,5]}
Res: error
^]

telnet> quit
Connection closed.
$
```

Notice that the character `^]` is produced by `Ctrl+]`.

4 Property-Based Bug Hunting (`bughunt.erl`, 4 points)

The module **vectors.beam**[4], contains 50 implementations of an evaluator for the language used in the previous task. Unfortunately 46 of them have bugs...!

Task

Write properties that can be used to test the evaluators. Identify those that do not conform to the specification, by giving an input, the expected output and the buggy evaluator's output.

Interface of `vectors.beam`

The contents of the corresponding **vectors.erl**[5] were the following:

```
-module(vectors).

-export([vector/1,
         vector_1/1,
         ...
         vector_50/1]).

-type vector()    :: [integer(),...].
-type expr()      :: vector()
                   | {vector_op(),     expr(), expr()}
                   | {scalar_op(), int_expr(), expr()}.
-type int_expr()  :: integer()
                   | {norm_op(), expr()}.
-type vector_op() :: 'add' | 'sub' | 'dot'.
-type scalar_op() :: 'mul' | 'div'.
-type norm_op()   :: 'norm_one' | 'norm_inf'.

-spec vector(integer()) -> fun((expr()) -> vector() | 'error').
vector(Id) when Id > 0, Id < 51 ->
  Name = list_to_atom(lists:flatten(io_lib:format("vector_~p", [Id]))),
  fun ?MODULE:Name/1.

-spec vector_1(expr()) -> vector() | 'error'.
vector_1(Expr) ->
  %% ???

...

-spec vector_50(expr()) -> vector() | 'error'.
vector_50(Expr) ->
  %% ???
```

As you can see, the function `vector/1` can be used to get the evaluator corresponding to the provided `Id`. The evaluators can also be called directly using the `vector_N/1` functions.

[4] http://www.it.uu.se/edu/course/homepage/avfunpro/ht17/vectors.beam
[5] https://gist.github.com/aronisstav/626f0f8edd943c8ca998

Expected interface of `bughunt.erl`

Among other functions, the module `bughunt` should export a function `test/1` that takes as input an integer between 1 and 50 and returns one of the following within 5 seconds:

- if the input is the id of a correct evaluator, the atom `'correct'` is returned.
- if the input is the id of a buggy evaluator, the tuple {`Input, ExpectedOutput, ActualOutput, Comment`} is returned, where:
  - `Input` is an Erlang term of type `expr()`
  - `ExpectedOutput` is the expected output when evaluating the input
  - `ActualOutput` is the output returned by the buggy evaluator or the atom `'crash'` (if the evaluator crashes) and
  - `Comment` is a string that shortly describes a probable cause for the bug (you can leave it empty if you are not sure about the bug)

Sample

Assume that a hypothetical evaluator #51 is correct and evaluator #52 does not support addition.

```
1> vectors:vector_51({'div', {'norm_inf', [-1, 5, 10]}, [1, 10, 100, 9999]}).
[0, 1, 10, 999]
2> bughunt:test(51).
correct
3> vectors:vector_52({'add', [1], [1]}).
error
4> vectors:vector_52({'sub', [1], [1]}).
[0]
5> bughunt:test(52).
{{'add', [1], [1]}, [2], error, "The operation 'add' is not supported."}
```

Submission instructions

- Each student must send their own individual submission. You cannot work in groups.
- For this assignment, you must submit a single `afp_assignment1.zip` file at the relevant section in Studentportalen.
- `afp_assignment1.zip` should contain *six* (6) files, wihout any directory structure:
  - the *four* programs requested (`road.erl`, `calc_pi.erl` `vector_server.erl`, `bughunt.erl`) that should conform to the specified interfaces regarding exported functions, handling of input and format of output. Moreover, your programs should *not* produce any compiler warnings and, preferably, should also not produce any warnings from dialyzer.
  - the report `road.pdf` explaining your algorithm and how you tested `road.erl`.
  - a text file named `README.txt` whose first line should be your name. You can include any other comments about your solutions in this file.
- Remember that you have a total of *ten* (10) free "late" days for all assignments and the final project. Do not spend them all on this assignment!

Have fun!