Programming II (ID1019) 2017-06-07 1400-18:00

Instructions

- You are not allowed to have any material besides pen and paper. Mobiles etc, should be left to the guards.
- All answers should be written in these pages, use the space allocated after each question to write down your answer.
- Answers should be written in English.
- You should hand in the whole exam.
- No additional pages should be handed in.

Grade

The exam is divided into a number of questions where some are a bit harder than others. The harder questions are marked with a star *points**, and will give you points for the higher grades. The exam is thus divided into basic points and points for higher grades. First of all make sure that you pass the basic points before engaging with the higher points.

- E: 24 basic points
- D: 30 basic points
- C: 34 basic points
- B: 34 basic points and 14 higher points
- A: 34 basic points and 20 higher points

The limits could be adjusted to lower values but not raised.

Gained points

Namn:____________________________________

1 Data structures and pattern matching

1.1 what is Y [2 point]

What is the resulting binding for `Y` i the following pattern matching expressions (each one by its own), in the case that the matching succeeds:

- `[X, Y|_] = [1,2,3]` **Answer:** `Y = 2`
- `[X,_ |Y] = [1,2]` **Answer:** `Y = []`
- `[X,Z,Y] = [1|[2|[3]]]` **Answer:** `Y = 3`
- `Z = 2, X = {foo, Z}, {_, Y} = X` **Answer:** `Y = 2`
- `X = 1, Z = [], Y = [X,Z]` **Answer:** `Y = [1,[]]`

1.2 list operations [2 poäng]

Evaluate the expressions below, what is X? Write the answer in as simple form as possible i.e. not in the form that it is written now.

- `X = [[1,2] | [3,4]]` **Answer:** `X = [[1,2], 3, 4]`
- `X = [1] ++ [2,3,4]` **Answer:** `X = [1, 2, 3, 4]`
- `X = [1,2] ++ [[3,4]]` **Answer:** `X = [1, 2, [3 4]]`
- `X = [1 | [2,[3,4]]]` **Answer:** `X = [1, 2, [3, 4]]`

1.3 a deck of cards [2 points]

If we should represent a deck of cards we could either represent it as a tuple, where each element is a card, or a list of cards. There are benefits of both representations, which?

Answer: Om vi representerar den som en tuple kan vi på konsant tid hitta det n:te kortet i högen. Om vi representerar det som en lista så kan vi på konsant tid plocka bort det första kortet och representera resten som en hög

Namn:____________________________________

2 Recursion

2.1 simple recursion [2 poäng]

Write a function, `transf(X,Y,L)`, that takes two values, X and Y, and a list of integers and returns a list where all values that are equal to X have been removed and all other values multiplied Y. If we evaluate `transf(5, 2, [2,5,4,3,5])` the result should be `[4,8,6]`.

Answer:

```
transf(_, _, []) -> [];
transf(X, Y, [X|R]) -> transf(X, Y, R);
transf(X, Y, [H|R]) -> [H*Y|transf(X, Y, R)].
```

2.2 tail recursive [2 poäng]

Below is the definition of `sum/1` that sums the element in a list. Rewrite the function so that it is tail recursive.

```
sum([]) -> 0;
sum([H|T]) -> H + sum(T).
```

Answer:

```
sum(L) -> sum(L, 0);

sum([], S) -> S.
sum([H|T], S) -> sum(T, H+S).
```

Namn:____________________________________

2.3 a tree [2 poäng]

Assume that a tree is represented by the atom `nil` for an empty branch and the strukture `{node, Value, Left, Right}` for a node in the tree where the value is an integer. Write a function `min/1` that finds the smallest value in the tree or returns `inf` if the tree is empty. You can use the fact that all integers are *less than* atoms i.e. `3 < inf` is true.

Answer:

```
min(L) -> min(L, inf).

min(nil, M) -> M;
min({node, V, L, R}, M) ->
  if V < M -> min(L, min(R, V));
     true ->  min(L, min(R, M))
  end.
```

3 Higher order

3.1 filter and map [2 poäng]

Given the definitions below, what is returned by the call `foo([1,2,3,4,5])`?

- `filter(F, L)`: retuns a list of all elements, E, in L for which $F(E)$ evaluates to *true*.
- `map(F, L)`: returns a list of $F(E)$ for each element E in L.

```
foo(L) ->
  map(fun(X) -> X*3 end, filter(fun(X) -> X =/= 3 end, L)).
```

Answer: `[3,6,12,15]`

3.2 left or right [2 poäng*]

Given the definitions below, what is returned by a call to `bar([1,2,3,4])`?

- `foldl(F, A, L)`: returns the value $F(E_n, F(...F(E_1, A)))$ where E_k is the elements in L.

Namn:____________________________________

```
bar(L) ->
   foldl(fun(X, A) -> [X|A] end, [], L).
```

Answer: `[4,3,2,1]`

4 Evaluating expressions

We have during the course worked with describing how a language can be defined by formally describing which terms, expressions and data structures we have and how we by rules can describe what should happen when we evaluate expressions. The following questions assume that we have defined a small language given the guidelines we have presented.

4.1 free variables [2 points]

Which is/are the free variable/variables in the following sequence?

```
Z = {42, Y}, F = fun(X) -> foo(X, Y) end, F(Z)
```

Answer: `Y`

What is the result of evaluating the following sequence?

```
{X,Y} = {a, b}, F = fun(X) -> {X, Y} end, F(c)
```

Answer: `{c,b}`

Namn:____

4.2 pattern matching [2 points*]

The four first rules for pattern matching are:

- $P\sigma(a, s) \rightarrow \sigma$ if $a \equiv s$
- $P\sigma(\_, s) \rightarrow \sigma$
- $P\sigma(v, s) \rightarrow \sigma$ if $v/s \in \sigma$
- $P\sigma(v, s) \rightarrow \{v/s\} \cup \sigma$ if $v/t \notin \sigma$

There is a very important case that we do not cover by these rules, which case, what does the rule look like and when do we make use of it.

Answer: Fallet är där vi försöker göra en mönstermatching med en datastruktur (eller variabel bunden till en datastruktur) som inte är identisk med den datstruktur vi matcher mot. I detta fall är resultatet *fail* och det är viktigt att veta då vi implementerar t.ex. *case-uttryck*.

Namn:____________________________________

4.3 strange expression [4 points*]

Assume that we have extended our language to also handle foo expressions. We have a syntax for them and evaluate them using the following rules where *param* is a sequence of variables and *seq* a sequence of expressions.

- $E\sigma$((foo(param) -> sequence end) $\rightarrow foo(param, seq)$

We can then take this contruct and apply it on a sequence of expressions by using the following rule:

- $E\sigma(foo(param, sequence)(\text{args})) -> E\theta(sequence)$

This holds if *param* is a sequence of i variables $p_1, ..p_i$, *args* a sequence of expressions $a_1, ..a_i$ and $E\sigma(a_i) - > s_i$. The environment θ is the union of bindings p_1/s_1... and σ. We can however not take the union of these sets straight off without having a problem. Which is the problem and how would we handle it?

Answer: Om p_k finns i σ så skulle vi få dubbla bindningar för en variabel. Vi måste antingen ta bort alla element p_k/t ur σ innan vi tar unionen eller på något sätt döpa om parametrarna så att de är unika.

If we had a *foo contruct* in Erlang, what would then the evaluation of `test(2)` return as result? Motivate.

```
test(Y) ->
  F = bar(6),
  F(3).

bar(Y) ->
  foo(X) -> X + Y end.
```

Answer: Evalueringen skulle returnera 5 eftersom vi applicerar F på 3 i en omgivning där Y är 2. F är i detta skede en foo-struktur som har kroppen X + Y, variablen X får sin bindning till 3 men foo har inte med sig en omgivning (det är inte en closure) där Y har ett värde. Y får sitt värde i den omgivning där vi gör appliceringen.

Namn:______________________________________

5 Complexity

In the answers to the questions below make sure to define n and motivate why your answer is correct.

5.1 find the value [2 points]

The function below finds the value of a key in a tree. What is the assymptotic time complexity for the function, assume that the tree is balanced.

```
lookup(_, nil) -> false;
lookup(K, {node, K, V, _, _}) -> {ok, V};
lookup(K, {node, _, _, L, R}) ->
      case lookup(K, L) of
         false -> lookup(K, R);
         Found -> Found
      end.
```

Answer: Funktionen har tidskomplexitet $O(n)$ där n är antalet element i trädet. Trädet är inte ordnat och vi söker igenom trädet tills vi hittar värdet.

5.2 tail recursive [2 points]

The function `sentence/1` below takes a list of words and contructs a sentence. The list `[“hej”, “på”, “dig”]` gives the sentence `“Hej på dig.”`. The function `sentence/2` is tail recursive but we do this at a price. What is the assymptotic time complexity of the function.

```
sentence([]) -> []
sentence([[W|Ord] | Words ]) ->
      sentence(Words, [string:to_capital(W)|Ord]).

sentence([], Sofar) -> append(Sofar, [$\.]);
sentence([Word|Words], Sofar) ->
       sentence(Words, append(Sofar, [$\ | Word])).
```

Answer: Tidskomplexiteten är n^2, där n är antalet ord i listan, eftersom vi kommer utföra allt längre append-operationer; variablen `Sofar` kommer ha en längd som är proportionell till n.

5.3 traverse this [2 points*]

Below you have a function that traverser a ordered tree and returns all values as ordered list. What is the asymptotic time complexity of the function, assume the tree is balanced? Motivate your answer.

```
trav(nil) -> [];
trav({node, V, L, R}) -> trav(L) ++ [V|trav(R)].
```

Answer: Funktionen har tidskomplexitet $O(n * lg(n))$ där n är antalet element i trädet. Djupet på trädet är $lg(n)$ och på varja djup så görs att antal append-operationer som har komplexitet n

Namn:____________________________________

6 Concurrency

6.1 a sum [2 points]

Implement a procedure that starts a process that has a value as state. The process should be able to handle the follwing messages:

- `{add, N}` : add N to the value
- `{sub, N}` : subtract N from the value
- `{req, Pid}` : send `{total, V}`, where V is the value, to Pid

Answer:

```
start(N) -> spawn(fun() -> sum(N) end).

sum(S) ->
  receive
    {add, N} -> sum(S + N);
    {sub, N} -> sum(S - N);
    {req, Pid} -> Pid ! {total, S}, sum(S)
  end.
```

Namn:____

6.2 cardamom [2 points]

Assume that we have the definitions below of `cardamom/1` and `momcarda/1`. Also assume that we have started a process with the call `spawn(fun() -> cardamom(10) end)`. Then assume that send the following sequence of messages to the process: `{sub, 4}`, `{add, 10}`, `{mul, 2}`, `{sub, 10}`. What is now the state of the process?

```
cardamom(S) ->
    receive
        {add, X} -> momcarda(S + X);
        {sub, X} -> momcarda(S - X)
    end.

momcarda(S) ->
    receive
        {add, X} -> cardamom(S + X);
        {sub, X} -> cardamom(S - X);
        {mul, X} -> cardamom(S * X)
    end.
```

Answer: Processen kommer hantera meddelanden i den ordning de kommer men endast kunna hantera `mul` i tillståndet `momcard`. Efter sekvensen kommer processen att vara i tillståndet `cardamom(12)`. Det är viktigt att ange att vi är i `cardamom` eftersom vi då är i ett tillstånd där vi inte kan ta emot meddelanden på formen `{mul, N}`.

Namn:____________________________________

6.3 ping-ping [4 points*]

Below is a procedure that implements a process. Write a procedure that starts two such processes. Each process will have a unique name and the other process's identifer as its internal state. You can of course write procedures that will initalize these state.

```
ping(Name, Pid) ->
  receive
    {frw, Msg} ->
           Pid ! {msg, Msg},
           ping(Name, Pid);
    {msg, Msg} ->
           io:format("~w received ~w~n", [Name, Msg]),
           ping(Name, Pid);
    terminate ->
           ok
  end.
```

Answer:

```
start() ->
   Foo = spawn(fun() -> init(foo) end),
   Bar = spawn(fun() -> ping(bar, Foo) end),
   Foo ! {connect, Bar}.

init(Name) ->
   receive
     {connect, Pid} -> ping(Name, Pid)
   end.
```

Namn:____________________________________

7 Programming

7.1 arithmetic expressions

7.1.1 represent arithmetic expressions [2 points]

How could we represent arithmetic expressions, for example 2+3 and 2+3∗6, if we then should work with them and do for eaxmple simplifications etc.

Describe the representation of expressions where thes can be:

- integers: 1, 2, 3 ...
- sum of expressions: $2 + 3 \ldots$
- product of expressions: $2 * 4 \ldots$

Note that we should be able to represent expressions as for example $2 + 3 * (4 + 6)$ etc. Preferably use the Erlang typ notation when describing the representation.

Answer:

```
-type const() :: {const, integer()}.
-type sum() :: {add, expr(), expr()}.
-type prod() :: {mul, expr(), expr()}.
-type expr() :: sum()|prod()|const().
```

7.1.2 evaluation of expressions [2 points]

Using the representation in the previous question, implement a function `eval/1` that takes an expression and returns the *result*. If we call the function with the representation of $2 + (3 * 4)$ it should return 14.

Answer:

```
eval({const, C}) -> C;
eval({add, E1, E2}) -> eval(E1) + eval(E2);
eval({mul, E1, E2}) -> eval(E1) * eval(E2);
```

7.1.3 varying variables [2 points]

Extend the representation so that we also can handle expressions with variables. We should for example be able to represent expressions like: $2 + x$ and $2 + (3 * y)$. Describe how variables are represented and how this changes our

Namn:____________________________________

representation of expressions. Preferably use the Erlang typ notation when describing the representation.

Answer:

```
-type var() :: {var, atom()}.

-type expr() :: sum()|prod()|const()| var().
```

7.1.4 value of variables [2 points]

Describe a representation of a set of *variable bindings*, a mapping from variables to values, and a function `lookup/2` that takes variable and a set of bin bindings as a arguments and returns a valuefor the requested variable. We assume that the variable is in the mapping.

Answer: Eftersom vi vet att vi skall mappa variabler till värden kan vi välja en enklare representation. Vi kan skapa en mappning mellan atomerna som identifierar variablerna till godtyckliga värden.

```
-type env() :: [{atom(), any()}].

lookup(Var,[{Var, Value}|_]) -> Value;
lookup(Var,[_|Rest]) -> lookup(Var,Rest).
```

7.1.5 evaluate using variables [2 points*]

Implement a function `eval/2` that takes an expression and a set of vaiable bindings and returns the valueof the evaluated expression. Use the representation of variables and variable bindings and the function `lookup/2` in the previous questions.

Answer:

```
eval({const, C}, _) -> C;
eval({var, V}, Env) ->  lookup(V, Env);
eval({add, E1, E2}, Env) -> eval(E1, Env) + eval(E2, Env);
eval({mul, E1, E2}, Env) -> eval(E1, Env) * eval(E2, Env);
```

Namn:______________________________________

7.1.6 derivative of expression [2 points*]

Implement a function `deriv/2` that takes an expression, represented as in the questions above, and a variable, and returns the derivative of the expression with respect to the variable. The rules for derivative are as follows:

- $c' = 0$ givet att c är en konstant eller en variabel skild från den variabel vi deriverar över
- $x' = 1$ givet att x är den variabel vi deriverar över
- $(f(x) + g(x))' = f'(x) + g'(x)$
- $(f(x) * g(x))' = f'(x) * g(x) + f(x) * g'(x)$

Answer: Detta kan tyckas komplicerat men det är naturligtvis mycket enkelt.

```
deriv({const, _}, _) -> 0
deriv({var, V}, V) -> 1;
deriv({var, _}, _) -> 0;
deriv({add, F, G}, V) -> {add, deriv(F, V), deriv(G, V)};
deriv({mul, F, G}, V) ->
       {add, {mul, deriv(F, V), G}
             {mul, F, deriv(G, V)}}.
```

Namn:____________________________________

7.2 an addressable memmory

7.2.1 a cell [2 points]

In the module `cell`, implememt a process that holds a value as its internal state. The process should be able to handle a sequence of messages. The following messages should be handled:

- `{set, V}`: set the value to V
- `{get, Pid}`: send `{ok, V}`, where V is the value, to Pid
- `{free, Pid}`: send `{ok, V}`, where V is the value, to Pid, terminate the process

The module should exprort a function `start/1` that takes an initial value and returns a process identifier of a cell process.

Answer:

```
-module(cell).
-export([start/1]).

start(N) ->
    spawn(fun() ->  cell(N) end).

cell(N) ->
    receive
        {set, V} ->
            cell(V);
        {get, Pid} ->
            Pid ! {ok, N},
            cell(N)
        {free, Pid} ->
            Pid ! {ok, N},
            ok
    end.
```

Namn:______________________________________

7.2.2 a functional interface [2 points]

In the same module implement and export three functions:

- `set(Pid, V)`: given a pid of a cell the value is set to V, return ok
- `get(Pid)`: given the pid of a cell return `{ok, V}` where V is the value of the cell
- `free(Pid)`: given the pid of a cell return `{ok, V}` where V is the value of the cell, the cell terminates

Answer:

```
-export([set/2, get/1]).

set(Pid, V) ->
    Pid ! {set, V},
    ok.

get(Pid) ->
    Pid ! {get, self()},
    receive
        {ok, V} ->
            {ok, V}
    end.

free(Pid) ->
    Pid ! {free, self()},
    receive
        {ok, V} ->
            {ok, V}
    end.
```

Namn:____________________________________

7.2.3 update [2 points]

Given the definition below, what is the result when calling `test(1000)`. Motivate.

```
test(N) ->
     Cell = cell:start(0),
     Self = self(),
     spawn(fun() -> add(N, Cell, Self) end),
     spawn(fun() -> add(N, Cell, Self) end),
     receive
        done ->
            receive
               done ->
                    cell:free(Cell)
            end
     end.

add(0, _, Master) ->
    Master ! done;
add(N, Cell, Master) ->
    {ok, V}= cell:get(Cell),
    cell:set(Cell, V+1),
    add(N-1, Cell, Master).
```

Answer:

Namn:____________________________________

7.2.4 an addressable memory [2 points]

In a module `mem`, implement a function `new/1` that takes a list of values and returns a tuple of processidentifiers for a set of cells that have been initialized by the values in the list. You can use the builtin function `list_to_tuple/1` that takes a list of elements and returns the corresponding tuple.

Answer:

```
-module(mem).

-export([new/1]).

new(Values) ->
    Pids = lists:map(fun(V) -> cell:start(V) end, Values),
    list_to_tuple(Pids).
```

7.2.5 functional interface [2 points*]

Implement the functions `get/2` and `set/3` in the module `mem`. USe the builtin function `element/2` that takes a position and a tuple and returns the element at that position. The functions should use the exported functions from `cell` and not know the messages used internally by the cell process.

Answer:

```
set(Array, N, V) ->
    cell:set(element(N, Array), V).

get(Array, N) ->
    cell:get(element(N, Array)).
```

Namn:____________________________________

7.2.6 asynchronous call [2 points*]

The interface we now have to get a value is synchrounous but we would like to have a asynchrounous. Add the functions needed in `cell` to provide a function `get_asyn/2` that initiates a reading of a cell but returns a unique reference (use `make_ref/0`). The reference can then later be used to retreive the answer by calling the function `get_answ/1` that will return `{ok, V}`.

Answer:

```
get_asyn(Pid) ->
    Ref = make_ref(),
    Pid ! {get, Ref, self()},
    Ref.

get_answ(Ref) ->
    receive
        {ok, Ref, V} ->
            {ok, V}
    end.

cell(N) ->
    receive
        {set, V} ->
            cell(V);
        {get, Pid} ->
            Pid ! {ok, N},
            cell(N);
        {get, Ref, Pid} ->
            Pid ! {ok, Ref, N},
            cell(N)
    end.
```

7.2.7 sum the values [2 points*]

Use the asynchronous interface and implement a function `sum/1` that sums all the values in a memory. You should use higher order functions `lists:foldl(Fun, Acc, List)` and `lists:map(Fun, List)`. You can use the builtin functions `size/1` that returns teh size of a tuple and `lists:seq(From, To)` that returns a list with integers from From to To.

Answer:

```
sum(Array) ->
```

Namn:_______________________________________

```
    Ids = lists:seq(1, size(Array)),
    Refs = lists:map(fun(I) -> get_asyn(Array, I) end, Ids),
    lists:foldl(fun(R, A) -> {ok, V} = get_answ(R), A+V end, 0, Refs).
```