CpSc 418 **Homework 1** Solution Set

Please see http://www.ugrad.cs.ubc.ca/~cs418/2013-1/hw/1/soln/hw1.erl for the Erlang code for the solutions.

1. **Tail Recursion Optimization.**

   (a) `add_hr(N)` and `add_tr(N)`
   My solutions for `add_hr(N)`, `add_tr(N)`, `time_add_hr(N)`, and `time_add_tr(N)` are in http://www.ugrad.cs.ubc.ca/~cs418/2013-1/hw/1/soln/hw1.erl. Here are the times that I observed for running each for various values of `N`:

N	T_{hr}	T_{tr}	Speed-up	N	T_{hr}	T_{tr}	Speed-up	N	T_{hr}	T_{tr}	Speed-up
1	4.257e-8	4.168e-8	1.02	100	1.854e-6	1.317e-6	1.41	10000	1.802e-4	1.273e-4	1.42
2	6.136e-8	5.840e-8	1.05	200	3.747e-6	2.644e-6	1.42	20000	3.606e-4	2.548e-4	1.42
3	9.329e-8	6.672e-8	1.40	300	5.538e-6	3.893e-6	1.42	30000	5.395e-4	3.810e-4	1.42
5	1.295e-7	9.942e-8	1.30	500	9.268e-6	6.589e-6	1.41	50000	8.965e-4	6.351e-4	1.41
7	1.604e-7	1.180e-7	1.36	700	1.269e-5	8.935e-6	1.42	70000	1.256e-3	8.906e-4	1.41
10	2.092e-7	1.680e-7	1.25	1000	1.790e-5	1.272e-5	1.41	100000	1.801e-3	1.273e-3	1.42
20	4.101e-7	2.986e-7	1.37	2000	3.662e-5	2.642e-5	1.39	200000	3.603e-3	2.539e-3	1.42
30	5.919e-7	4.211e-7	1.41	3000	5.454e-5	3.908e-5	1.40	300000	5.396e-3	3.800e-3	1.42
50	9.319e-7	6.948e-7	1.34	5000	9.155e-5	6.356e-5	1.44	500000	9.190e-3	6.356e-3	1.45
70	1.292e-6	9.852e-7	1.31	7000	1.257e-4	8.914e-5	1.41	700000	1.279e-2	8.880e-3	1.44
								1000000	1.832e-2	1.265e-2	1.45

   where T_{hr} is the time for `add_hr(N)` and T_{tr} is the time for `add_tr(N)`. For small values of `N`, I'll guess that the timings are mainly reflecting the overhead of the function calls for `time_it:t` (and, in my solution, `time_it:perseverate`). I could do some more experiments to test this guess, but not now. For $\texttt{N} \geq 30$, the average speed-up is about 1.41. The tail-recursive version is definitely faster.

   (b) `sum_hr_smem` and `sum_trsmem`.
   My solutions for `sum_hr_smem(N)`, `sum_tr_smem(N)` are in http://www.ugrad.cs.ubc.ca/~cs418/2013-1/hw/1/soln/hw1.erl. My tester code calls `hw1:stack_size` before calling `sum_hr_smem` or `sum_trsmem` to get a base value `S0`. It then calls `sum_hr_smem` and `sum_trsmem` functions to get the maximum stack sizes as they execute. My tester code then subtracts `S0` from these values so I'm just measuring the stack space used by `sum_hr_smem` and `sum_trsmem` and not the space used by the Erlang shell, my testing function, etc. Here's the data:

N	S_{hr}	S_{tr}	N	S_{hr}	S_{tr}	N	S_{hr}	S_{tr}
1	3	5	100	201	5	10000	20001	5
2	5	5	200	401	5	20000	40001	5
3	7	5	300	601	5	30000	60001	5
5	11	5	500	1001	5	50000	100001	5
7	15	5	700	1401	5	70000	140001	5
10	21	5	1000	2001	5	100000	200001	5
20	41	5	2000	4001	5	200000	400001	5
30	61	5	3000	6001	5	300000	600001	5
50	101	5	5000	10001	5	500000	1000001	5
70	141	5	7000	14001	5	700000	1400001	5
						1000000	2000001	5

   where S_{hr} is the maximum stack size for `sum_hr_smem(N)` and S_{tr} is the time for `sum_tr_smem(N)`. From this, I see that the stack size for `sum_hr_smem(N)` is $2\texttt{N} + 1$ Erlang words, and that the stack size for `sum_tr_smem(N)` is five words independent of the value of `N`. For $\texttt{N} = 1$, the tail-recursive version uses *more* memory than the head recursive version because it takes more parameters. For larger values of `N`, the tail-recursive version is smaller because of tail-call elimination.

   (c) Reactive processes.
   It is important to write reactive processes as tail-recursive functions. Otherwise, a new stack-frame will be

created each time a request is processed. Because the function never returns, these stack frames are never freed. A reactive process that is not tail-recursive will eventually run out of memory.

Note that one might modify the function to make it head-recursive while debugging. This way, all of the previous stack frames remain, and it is possible to examine the history of the process that led it to an error (as long as it doesn't run out of memory first!).

2. List Flattening: cancelled.

3. **Counting Corners (30 points).** My solution is in
   http://www.ugrad.cs.ubc.ca/~cs418/2013-1/hw/1/soln/hw1.erl.

4. Parallel Collatz
   My solution is in
   http://www.ugrad.cs.ubc.ca/~cs418/2013-1/hw/1/soln/hw1.erl.

5. Feedback.
   We'll calculate summaries of the responses and add them to this solution set.