Documentatie laborator 2 – Recursivitate pe stiva/coada/arborescenta

Preliminarii: Downloadati DrRacket de la **http://racket-lang.org/**. Inainte de a incepe sa programati in Scheme alegeti din meniul Language limbajul Pretty Big (care cunoaste majoritatea functiilor de Scheme). Impreuna cu aceasta documentatie gasiti si un fisier Scheme cu exercitii rezolvate – pe care trebuie sa il parcurgeti. Programare placuta!

Recursivitate

Cititi textul de la adresa http://mitpress.mit.edu/sicp/full-text/book/book-Z-H-11.html#%_sec_1.2.1 de la inceput pana la exemplul cu „counting change". Apoi parcurgeti rezumatul urmator si exercitiile din el.

In timpul rularii, un program mentine o stiva cu informatiile de care (mai) are nevoie. La fiecare apel de functie, stiva creste. La fiecare terminare a unui apel cu o anumita valoare, stiva scade. Cand o recursivitate este foarte adanca (exista multe apeluri de functie lansate si neterminate inca), stiva devine foarte mare. Acest lucru afecteaza:

- Memoria (Ea este principala resursa afectata de recursivitatea pe stiva. La un moment dat trebuie retinute foarte multe informatii, de exemplu la functia factorial trebuie retinute toate inmultirile care se vor efectua pe masura ce revenim din adancimea recursivitatii. Observati exercitiile 1 si 2 din lab2-doc.rkt. In anumite implementari ale limbajelor de programare stiva are o dimensiune maxima, iar atunci cand aceasta este epuizata programul se termina cu eroare de tip „out of stack".)
- Timpul (Timpul folosit pentru a redimensiona stiva ar putea fi folosit de alte calcule, programul final fiind astfel mai rapid.)

Solutia consta in transformarea recursivitatii pe stiva in recursivitate pe coada (numita si transformare a recursivitatii in iteratie). Metoda de transformare studiata de noi este folosirea unei variabile de acumulare in care se acumuleaza rezultatul (in functia factorial: variabila product este variabila de acumulare). Aceasta se da ca parametru al functiei. Rezultatul temporar se paseaza mai departe in noi apeluri, evitand astfel efectuarea de operatii pe revenire.

Cand o functie este recursiva pe coada i se poate aplica tail-call-optimization. Implementarea de Scheme pe care o folositi are aceasta optimizare ca parte din specificatie. Ce se intampla efectiv este transformarea apelului recursiv pe coada intr-un goto cu argumente catre inceputul functiei. Astfel stiva nu se mai redimensioneaza, spatiul utilizat este O(1).

Observati exercitiile 3-5 din lab2-doc.rkt. Ati putea incerca sa le rezolvati singuri inainte de a citi rezolvarile.