Zadania: zestaw 3.

Podczas dzisiejszych zajęć poznamy książkowy przykład języka funkcyjnego, a mianowicie Ocaml. Głównym celem dzisiejszego projektu będzie – jak na języki funkcyjne przystało – definiowanie funkcji, ale, jako że jesteśmy na zajęciach z logiki, będą to funkcje w jakiś sposób związane z klasycznym rachunkiem zdań. Zaczniemy od zapoznania się z Ocaml; można to zrobić na wiele sposobów korzystając z jednego z wielu dostępnych tutoriali, ja polecam Wam następujący:

`http://www.cs.jhu.edu/~scott/pl/lectures/caml-intro.html`

Na potrzeby dzisiejszego zadania w zupełności wystarczy przeczytać fragment do definiowania funkcji włącznie, typami zajmiemy się później.

Po "przeparsowaniu" tutoriala i zapoznaniu się z zawartymi tam przykładami, zacznijcie definiować własne funkcje. Następujące sześć przykładów pomyślane jest jako zadania do rozwiązania na zajęciach, gdy już się z nimi uporacie, możecie przystąpić do rozwiązywania zadania domowego opisanego poniżej.

(1) Napisać funkcję *prod l : int list -> int* która zwraca iloczyn elementów listy l. Iloczyn powinien być równy 1, gdy lista jest pusta.
(2) Napisać funkcję *max l : int list -> int* która zwraca największą liczbę z listy l. Możecie założyć, że lista jest niepusta.
(3) Napisać funkcję *addTail l e : 'a list -> 'a -> 'a list* która bierze listę l i pojedyńczy element a i zwraca nową listę, w której a jest dopisany do końca.
(4) Napisać funkcję *index l e : 'a list -> 'a -> int* która bierze listę i pojedyńczy element i zwraca numer pozycji, pod którą po raz pierwszy element pojawia się na liście. Funkcja powinna zwracać -1, jeżeli element w niej nie występuje.
(5) Napisać funkcję *rindex l e : 'a list -> 'a -> int* jak poprzednia, ale zwracającą numer pozycji, pod którą element pojawia się po raz ostatni na liście.
(6) Napisać funkcję *fill n x : int -> 'a -> 'a list* która zwraca listę składającą się z n wystąpień x

1. Zadanie domowe

Celem niniejszego zadania domowego będzie napisanie funkcji, która będzie konwertowała formuły klasycznego rachunku zdań do koniunktywnej postaci normalnej (CNF). Przypomnijcie sobie stosowny fragment wykładu, w którym pojawiły się formuły w takiej postaci, przypomnijcie sobie też wzmiankowane wówczas trudności algorytmiczne w konwertowaniu formuł do CNF (złożoność obliczeniowa!).

Będziecie potrzebowali następującego typu danych:

```
type formula =
False
| True
| Var of char
| And of formula * formula
| Or of formula * formula
| Not of formula
```

Tutaj *False* i *True* reprezentują oczywiste wartości. *Var c* reprezentuje zmienną zdaniową c, a konstruktory *And, Or, Not* odpowiednio koniunkcję, alternatywę i negację. Przykładowo, formuła

$$(a \vee b) \wedge c$$

będzie reprezentowana jako następująca wartość Ocaml:

And(Or(Var 'a', Var 'b'), Var 'c')

Waszym zadaniem jest zatem napisanie funkcji *to_cnf : formula -> formula* biorącej dowolną formułę i zwracającą równoważną jej formułę w koniunktywnej postaci normalnej. Rozwiązane zadanie zapiszcie w pliku *lab3.ml* i prześlijcie do mnie do **16 maja** (lub do **30 maja**, w zależności od grupy ćwiczeniowej). Proszę pamiętać o umieszczeniu tagu [agh2rlog] w polu subject!