**Paradygmaty programowania IT, inż. II rok**

|  |  |  |
| --- | --- | --- |
| Imię i nazwisko | Data | Godzina |
| Jakub Radzik | 3.11.2021 | 7:30 |

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
| nr listy:  3  zrobione | V | OCaml | V |  |  |  |  |  |  |  |
| nr listy:  4  zrobione | V | V | V | V | V |  |  |  |  |  |

Lista 3:  
1.

a) Typ to (int -> int -> ‘a) -> ‘a ponieważ definiujemy funkcję f1 przyjmującą funkcję x która operuje na INT-ach co widać po prawej stronie równania. Stąd wiemy że x przyjmuje INT-a i zwraca funkcję przyjmującą INT-a która zwraca typ generyczny i to wszystko pakujemy do f1 który zwraca wynik funkcji x czyli ‘a.

b) Typ to (string -> ‘a) -> string -> string -> ‘a ponieważ f2 przyjmuje funkcję x która przyjmuje jako parametr y ^ z, z użycia operatora ^ (konkatenacja stringów) wiemy że y, z to stringi i x przyjmuje stringa jako parametr lecz nie wiemy co zwraca, stąd zapis (string -> ‘a). Dalsza część typu bierze się z tego że y, z to parametry przyjmowane po kolei przez f2 i zwracające inne funkcje a w wyniku daje to co funkcja x.

2.

a) (let curry3 f a b c = f(a,b,c);; ) przyjmuje funkcję f które operuje na parametrach a,b,c, potem przyjmuje parametr a i zwraca funkcję przyjmującą b która zwraca funkcję przyjmującą parametr c zwracającą wynik f od parametrów a,b,c. Stąd typ:   
(‘a \* ‘b \* ‘c -> ‘d) -> ‘a -> ‘b –> ‘c -> ‘d

b) (let uncurry f (a,b,c) = f a b c;; ) funkcja jest przeciwieństwem funkcji curry stąd jej typ:

(‘a -> ‘b –> ‘c -> ‘d) -> ‘a \* ‘b \* ‘c -> ‘d

Lista 4:

1.

a) Z zapisu funkcji wnioskuję że nie jesteśmy w stanie określić typów dla x, y, z a tym typu zwracanego przez x. Dlatego f2 przyjmuje i zwraca (a’ -> b’ -> ‘c). Więc jej typ to: (a’ -> b’ -> ‘c) -> a’ -> b’ -> ‘c .

b) Z zapisu x::y wiemy że dochodzi do przyłączenia elementu x do listy y, ale nie znamy typów. Więc x ma typ ‘a, a y typ ‘a list. Funkcja f2 bierze x zwraca funkcję przyjmującą y która zwraca funkcję przyjmującą z która zwraca x::y. Z tego wynika że typ to: ‘a -> ‘a list -> b -> ‘a list. Nie znamy typu parametru z ani nie jest powiedziane że to ten sam typ co x.