Programowanie funkcyjne na przykładzie języka Objective CAML

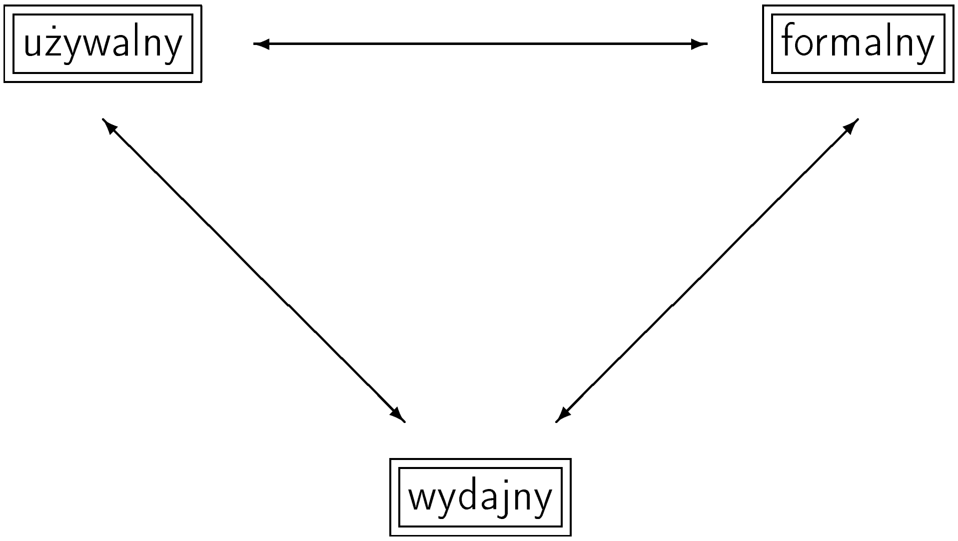
Grzegorz Grudziński

Instytut Informatyki UW
pok. 4180
e-mail: `gsg@mimuw.edu.pl`

LITERATURA
(to tylko wybrane pozycje, na temat programowania funkcyjnego ogólnie istnieje znacznie więcej książek i artykułów...)

- L.C. Paulson. *ML for the Working Programmer.* Cambridge University Press (1991).
- S. Sokołowski. *Applicative High Order Programming: The Standard ML Perspective.* Chapman & Hall (1991).
- C. Reade. *Elements of Functional Programming.* Addison-Wesley (1989).
- R. Stansfier. *An ML Primer.* Prentice-Hall (1992).
- C. Myers, C. Clack and E. Poon. *Programming with Standard ML.* Prentice-Hall (1993).
- R. Harrison. *Abstract Data Types in Standard ML.* Wiley (1993).
- A. Wikström. *Functional Programming using Standard ML.* Prentice-Hall (1987).
- R. Bird and P. Wadler. *Introduction to Functional Programming.* Prentice-Hall (1987).
- R. Harper. *Introduction to Standard ML.* Technical Report ECS-LFCS-86-14, LFCS, University of Edinburgh, revised edition (1989).
- M. Tofte. *Four Lectures on Standard ML.* Technical Report ECS-LFCS-89-73, LFCS, University of Edinburgh (1989).

Sprzeczności stojące przed językami programowania:



Programy powinny być *poprawne*, a także:

- czytelne;
- wydajne;
- bezpieczne;
- niezawodny;
- wygodne w użyciu;
- udokumentowane i czytelne;
- ...

Programowanie funkcyjne (aplikatywne):

- wyliczanie wartości wyrażeń, a nie modyfikowanie zmiennych za pomocą instrukcji;
- wiązanie wartości z identyfikatorami, a nie przypisania;
- rekursja zamiast iteracji;
- brak globalnego stanu: przekazywanie wartości przez jawne parametry i wyniki, nazywanie wartości tymczasowych.

ŚWIETNIE: większa czytelność, mniej okazji do popełnienia błędów,. . .

ALE: wydajność? wejście//wyjście? . . .

Zatem:

- Nowe techniki programowania, np. funkcje (wyższego rzędu) jako wartości;
- Potrzeba opracowania wydajnych kompilatorów (techniki optymalizacji, gospodarka pamięcią itd.);
- Kompromisy w rzeczywistych językach programowania: obecność cech imperatywnych . . .

$fib : Nat \to Nat$

$$\begin{aligned} fib(0) &= 1 \\ fib(1) &= 1 \\ fib(n+2) &= fib(n+1) + fib(n) \end{aligned}$$

Program w Caml-u:

```
# let rec f1 n =
  if n<=1 then 1
  else f1(n-1) + f1(n-2) ;;
```

Program w Pascalu:

```
function f4(n:integer):integer;
   var m,u,v,t : integer;
   begin
      if n<=1 then f4:=1
      else begin
              m:=1; u:=1; v:=1;
              while m<n do
                 begin m:=m+1; t:=u; u:=u+v; v:=t end;
              f4:=u
           end
   end;
```

Programy w Caml-u:

```
let rec f1 n =
    if n<=1 then 1
    else f1(n-1) + f1(n-2) ;;

let rec f2 n =
    if n<=1 then 1
    else let rec g n =
             if n=1 then (1,1)
             else let (u,v) = g(n-1) in (u+v,u)
         in  let (u,v) = g(n-1) in u+v
         ;;
```

Program w Caml-u:

```
let f3 n =
    if n<=1 then 1
    else let muv = ref (1,1,1)
         in ( (while let (m,u,v) = !muv in (m < n)
             do
               muv := (let (m,u,v) = !muv
                      in (m+1,u+v,u) );
               let (m,u,v) = !muv in u
             done
            ) ; let (m,u,v) = !muv in u )
    ;;
```

Program w Pascalu:

```
function f5(n:integer):integer;
    begin
        if n<=1 then f5:=1
        else f5 := f5(n-1) + f5(n-2)
    end;
```

Istotne cechy Caml-a:

- Caml jest językiem *funkcyjnym*.
- Caml jest językiem *interakcyjnym*.
- Caml jest językiem *silnie typowanym*.
- Caml ma *polimorficzny* system typów.
- Caml pozwala na definiowanie *abstrakcyjnych typów danych*.
- Caml ma *statyczne wiązanie identyfikatorów*.
- Caml ma bezpieczny mechanizm *wyjątków*.
- Caml ma *system modułów* pozwalający na *rozłączną kompilację*.
- Caml zapewnia automatyczną gospodarkę pamięcią i odśmiecanie.
- Caml ma bardzo dobry kompilator dla wielu różnych systemów operacyjnych.
- Caml ma (dość eleganckie) mechanizmy imperatywne.

Praca z Caml-em: interakcyja pętla "czytaj – wylicz – wypisz" .

```
# 27 * 37;;
- : int = 999
```

Błędy:

- błędy składniowe,
- błędy typowe,
- błędy wykonania obsługiwane jako wyjątki,
- pętle nieskończone.

Szkicowy plan wykładu:

- Krótki wstęp: o programowaniu funkcyjnym i o Caml-u.
- Monomorficzny fragment Caml-a: proste typy danych, wyrażenia, deklaracje.
- Monomorficzny fragment Caml-a: funkcje.
- Algebraiczne typy danych.
- Programowanie funkcyjne na standardowych typach danych (listy, drzewa): wstęp do polimorfizmu; wybrane techniki programowania funkcyjnego; wyjątki.
- Polimorfizm.
- Techniki programowania w językach z „leniwą ewaluacją".
- Wstęp do programowania modularnego.
- Sygnatury, struktury i funktory.
- Mechanizmy imperatywne.
- Biblioteki standardowe.