Seminarium: Programowanie w teorii typów

Przykładowe zastosowania typów zależnych

Wojciech Jedynak, Paweł Wieczorek

Instytut Informatyki Uniwersytetu Wrocławskiego

25 września 2011

1. Na początku był Haskell...
   - ...i wszystko było dobrze...
   - ...prawie wszystko.

Programowanie funkcyjne - mocne strony

- Programując w statycznie typowanym języku funkcyjnym mamy gwarancję, że jeśli kompilator zaakceptuje nasz program, to wykluczone zostały pewne klasy błędów programistycznych
- Co z błędami logicznymi?
- Używając sprytu i doświadczenia możemy sobie pomóc...

Dobra technika 1: Regularne drzewa binarne

```
data RTree a = Tip a | Node (RTree a) (RTree a)
```

Trafnie definiując typ indukcyjny wykluczamy (przez samą *konstrukcję*) możliwość utworzenia niepoprawnych wartości (np. wykluczamy drzewa puste).

Dobra technika 2: Listy o parzystej długości

```
mutual
    data EvenList a = ENil | ECons a (OddList a)
    data OddList  a = OCons a (EvenList a)
```

Możemy przez indukcję wzajemną udowodnić, że EvenList a zawsze będzie miało parzystą liczbę elementów.

Zła technika

Karciany przykład na tablicy.

Typ bool nie ma znaczenia!

W klasycznym programowaniu funkcyjnym typecheckerowi jest obojętne jaki będzie rezultat obliczenia predykatu w wyrażeniu warunkowym.

```
if null xs then
  head xs
else
  xs
```

“Well-typed programs can’t go wrong”?

Informacyjność typów - podejście "tradycyjne"

W Haskellu piszemy:

```
sort :: Ord a => [a] -> [a]
```

- Co możemy wywnioskować z takiej sygnatury?
- A czego **nie** możemy wywnioskować?

2 Typy zależne - motywacja

- Czy możemy lepiej?

Na czym polega programowanie z typami zależnymi?

- Programowanie w typami zależnymi opiera się na programowaniu funkcyjnym
- Dostajemy do dyspozycji znacznie bardziej rozbudowany system typów
- Główne hasło: *typ* wyrażenia może **zależeć** od jego *wartości*

Na czym polega programowanie z typami zależnymi? c.d.

- Zaciera się różnica między typami a wyrażeniami
- Żeby móc wydawać mocniejsze sądy wymuszamy *totalność*
- Dostajemy system wystarczająco silny, by mówić o **certyfikacji**

Wyrażanie własności danych

Systemy, którymi będziemy się tutaj zajmować pozwalają wyrażać właśności takie jak:

- liczba n jest parzysta
- lista l jest posortowana
- drzewo t jest zbalansowane
- plik f jest otwarty

Informacyjność typów - typy zależne

Używając typów zależnych możemy napisać (pseudokod)

```
sort :: Ord a => (xs : [a]) ->
  (ys : [a] && sorted ys && permutation_of xs ys)
```

Co możemy wywnioskować z takiej sygnatury?

Zastowanie praktyczne: projektowanie API

- Gdy projektujemy interfejs dla biblioteki rzadko udaje się wyrazić wszystkie założenia w sygnaturach.
- Stosując tradycyjne podejście pewne założenia musimy (w najlepszym wypadku) wyrazić jako komentarz...

Przykład. Funkcje dla list

Jaki jest problem z funkcjami null i head z preludium standardowego Haskella?

Projektowanie API c.d.

Dobra wiadomość

Typy zależne pozwalają nam traktować dowody na równi z wartościami. Możemy programować i dowodzić w ramach tego samego systemu!

Większa moc wyrazu = lepsza komunikacja

Wymagając, aby przy wywołaniu funkcji podano pewien dówód (certyfikat) przekazujemy pełen obraz sytuacji i nie mamy problemów z totalnością!

Uwagi

Żeby cała ta zabawa miała sens, logika wbudowana w język pozwalający na programowanie z typami zależnymi musi być niesprzeczna.