Zadania do prezentacji o monadach

Łukasz Dąbek

14 października 2013

Rozwiązania poniższych zadań należy wysłać na adres sznurek@gmail.com do dnia 17.11.2013 r. Na podany adres mailowy można przesyłać pytania dotyczące zadań bądź prezentacji.

1 Zadanie 1 - monadyczny interpreter

1.1 Treść zadania

Zadanie składa się z dwóch części: napisaniu interpretera języka While w stylu monadycznym[1] i rozszerzeniu go o instrukcję `abort` oraz niedeterminizm.

Załóżmy, że mamy daną kategorię gramatyczną dla liczb naturalnych `<nat>` oraz zmiennych `<var>`. Gramatyka języka While bez wspomnianych wyżej rozszerzeń jest następująca:

```
<a-op>   ::= + | - | *
<b-op>   ::= and | or
<ba-op>  ::= < | ==
<a-expr> ::= <nat> | <var> | <a-expr> <a-op> <a-expr>
<b-expr> ::= true | false | not <b-expr> | <b-expr> <b-op> <b-expr> |
             <a-expr> <ba-op> <a-expr>

<stmt>   ::= skip | <var> := <a-expr> | <stmt>; <stmt> |
             if <b-expr> then <stmt> else <stmt> |
             while <b-expr> do <stmt>
```

Przykładowy program w języku While liczący wartość `x!` wygląda następująco:

```
y := 0;
while 0 < x do (
    y := y * x;
    x := x - 1
)
```

Pamięć można reprezentować jako funkcję ze zmiennej w liczbę:

```
type Env = Var -> Integer
```

Dla wyrażeń arytmetycznych i logicznych należy napisać dwie funkcje interpretujące:

```
evalA :: Env -> Aexp -> Integer
evalB :: Env -> Bexp -> Integer
```

Dla instrukcji należy napisać interpreter w stylu monadycznym i przetestować go dla monady identycznościowej:

```
eval :: Monad m => Env -> Stmt -> m (Var -> Integer)
eval env Skip = return env
...
```

Kolejnym krokiem jest rozszerzenie języka While o dwie instrukcje:

```
<stmt> ::= (...) | abort <a-expr> | amb <stmt> <stmt>
```

Intuicyjnie instrukcja `abort n` powinna przerwać działanie programu z kodem błędu równym wartości wyrażenia `n`. Instrukcja `amb` niedeterministycznie wybiera jedną z gałęzi obliczeń.

Aby zamodelować wspomniane efekty należy:

1. Zdefiniować polimorficzny typ danych `M` (podpowiedź: powinien być kombinacją `Either` i listy).
2. Dla typu `M` zdefiniować instancję `Functor` i `Monad`.
3. Rozszerzyć wcześniej napisany interpreter o obsługę nowych instrukcji.

Podsumowując, ostateczny interpreter powinien mieć typ:

```
eval :: Env -> Stmt -> M Env
```

1.2 Ocenianie

Rozwiązanie powinno składać się z dwóch osobnych plików źródłowych (po jednym dla każdej części zadania). Na ocenę zadania składają się następujące kryteria, w kolejności malejącego znaczenia:

1. Wykonanie polecenia (tzn. interpretery napisane w innym stylu niż monadyczny nie będą oceniane).
2. Poprawność interpretera względem semantyki poznanej na przedmiocie Programowanie M.
3. Załączenie przykładowych programów w języku While prezentujących działanie instrukcji `abort` i `amb`.
4. Styl i czytelność kodu.

2 Zadanie 2 - monadyczny odkrywca

2.1 Treść zadania

W poprzednim zadaniu szukaliśmy monady dla konkretnego zastosowania. W tym zadaniu dla zadanego typu danych odkryjemy instancję monady oraz jej zastosowanie.

Zdefiniujmy nowy typ danych:

```
data Sel r a = Sel ((a -> r) -> a)
```

Aby wykonać zadanie należy:

1. Zdefiniować instancję klasy typów `Functor` dla typu `Sel`. Upewnij się, że Twoja definicja spełnia odpowiednie prawa.
2. Zdefiniować instancję klasy typów `Monad` dla typu `Sel`. Ponownie, upewnij się, że Twoja implementacja spełnia prawa monadyczne. [1]
3. (Najważniejsze i najtrudniejsze) Znaleźć zastosowanie dla właśnie zdefiniowanej monady.

Ponieważ ostatni krok jest trudny i niejednoznaczny, poniżej znajduje się kilka wskazówek:

1. Nazwa typu danych została wybrana nieprzypadkowo.
2. Rozważ przypadki, gdy `r = Bool` i `r = Int`. Następnie postaraj się zinterpretować typ `a -> Bool` jako predykat i `a -> Int` jako funkcję oceny.
3. Monada `Sel` ma związek z monadą kontynuacyjną (widzisz podobieństwo typów?).
4. Monada `Sel` może pomóc napisać Ci algorytm brutalny albo wygrać w Nima!
5. Wspomóż się haskellową notacją `do` aby wyrobić sobie intuicje.

2.2 Ocenianie

Na ocenę zadania składa się (z równą wagą):

1. Napisanie poprawnych instancji klas `Functor` i `Monad` dla typu `Sel`.
2. Znalezienie zastosowania dla odkrytej monady i zilustrowanie tego krótkim przykładem.

Przewidziane są bonusowe punkty za styl.

[1] W przeciwnym przypadku do Twoich drzwi może zapukać Międzygwiezdna Policja Monadyczna.

Literatura

[1] Philip Wadler, *The essence of functional programming*. Prentice Hall, 1992.