Multi-staged programming

Zaawansowane Programowanie Funkcyjne - Lista zadań 8

Radosław Warzocha

Wrocław, 27 stycznia 2014

Zadania należy przysłać do **12.02** na adres radoslaw.warzocha@gmail.com
Wszystkie pytania są mile widziane, choć czasem mogę odpowiedzieć z opóźnieniem.

1. Zadania

1.1. Materiały

- Sporo informacji dot. teorii programowania wieloetapowego znaleźć można w pracy doktorskiej Walida Tahy - twórcy MetaML'a - i innych pracach tego autora.
- Warto przeczytać oryginalną pracę, która położyła podwaliny pod Template Haskell'a (zwróć uwagę, że część zawartych tam informacji jest nie w pełni zgodna z obecną implementacją TH, np. typ `Expr` został zamieniony na `ExpQ`).
- W tym poście można przeczytać jak pracować z TH w `ghci`.
- Wreszcie ten wątek na stackoverflow pokazuje dlaczego Template Haskell nie jest jednak taki wspaniały

1.2. Zadanie 1

Zdefiniuj i przetestuj funkcję `tup2list :: Int -> Q Exp`, która zamienia n-elementową krotkę na listę długości n. Następnie zdefiniuj funkcję `genTup2List :: Int -> Q [Dec]`, która dla argumentu n generuje globalnie dostępne funkcje `tup2list`m dla wszystkich $1 \leq m \leq n$.

1.3. Zadanie 2

Zdefiniuj i przetestuj funkcję `tupFold :: Int -> Q Exp`, która generuje funkcję analogiczną do `fold` (w dowolnej wersji, lewej bądź prawej), ale działającą na krotkach n-elementowych. Nie używaj do tego funkcji z zad. 1. Sprawdź, czy

```
fold f acc $ tup2listn x = tupFoldn f acc x
```

1.4. Zadanie dodatkowe (dla ludzi, którzy nie mają dość interpreterów)

Rozważmy prosty język, wyrażony nastepującymi typami (nie będziemy zajmować się parsowaniem):

```
data Expr = Int Int | Var String | App String Expr
          | Add Expr Expr | Sub Expr Expr
          | Mul Expr Expr | Div Expr Expr | Ifz Expr Expr
data Defn = Declaration String String Expr
data Prog = Program [Defn] Expr
```

Gdzie

- `App s e` oznacza aplikację funkcji o nazwie `s` do wyrażenia `e`,
- `Ifz e e1 e2` oznacza wykonanie `e1` gdy wartość `e` jest równa 0 i `e2` w przeciwnym przypadku
- `Declaration n v e` oznacza deklarację funkcji o nazwie `n`, przy czym `v` jest nazwą argumentu tej funkcji, a `e` obliczanym wyrażeniem.

Napisz interpreter tego języka w stylu wieloetapowym. Następnie rozważ roszerzenie języka o błędy arytmetyczne (dzielenie przez 0). Postaraj się aby twoja implementacja była możliwe wydajna.

Wskazówka: Gdy dodamy błąd dzielenia przez 0 warto napisać interpreter w stylu CPS (Continuation Passing Style) – możemy wygenerować wydajniejszy kod.

2. Kryteria oceny

Ocenie podlegają

- Poprawność rozwiązania i wydajność generowanego kodu
- Czytelność kodu
- Przygotowane testy