Programowanie Funkcyjne 2022

Lista zadań nr 9

Na zajęcia 17, 19 i 23 stycznia 2023

Zadanie 1 (3p). W SKOSie znajduje się interpreter (`RefLang.hs`) i parser (`RefLangParser.hs`) prostego języka funkcyjnego z mutowalnym stanem. Interpreter jest napisany w stylu monadycznym (funkcja `eval`) przy założeniu, że posiadamy monadę która implementuje szereg potrzebnych operacji. Dodatkowo do reprezentacji lokacji na stercie używany jest typ `Loc m`, wyliczany z używanej monady `m` za pomocą rodziny typów `Loc`. Niestety interpretera nie można uruchomić, bo brakuje instancji monady, która posiada wszystkie potrzebne operacje. Zainstaluj biblioteczną monadę `ST` w klasach `MonadFresh` i `MonadHeap` tak by można było uruchomić interpreter. Jako implementację lokacji użyj typu `STRef`.

```
import Control.Monad.ST
import Data.STRef

type instance Loc (ST s) = STRef s (Value (ST s))

instance MonadFresh (ST s) where
  ...
instance MonadHeap (ST s) where
  ...
```

Uwaga: Jeśli chcesz do interpretera doimplementować operacje wejścia-wyjścia to zamiast typów `STRef` i `ST` możesz użyć typów `IORef` i `IO`.

Zadanie 2 (2p). Zaimplementuj transformator monad wzbogacający dowolną monadę o nowe możliwości opisane klasą `MonadFresh`. Użyj typu `Integer` do reprezentacji lokacji, zaś sam transformator monad powinien być transformatorem monady stanu, gdzie ukrytym stanem jest wartość typu `Integer` — licznik używany do generowania świeżych wartości. Postaraj się nie duplikować kodu i użyć bibliotecznego transformatora monad `StateT`, oczywiście owiniętego w konstruktor nowo utworzonego typu za pomocą konstrukcji `newtype`. Możesz użyć rozszerzenia `GeneralizedNewtypeDeriving` by ułatwić sobie zadanie.

Zadanie 3 (4p). Zaimplementuj transformator monad wzbogacający dowolną monadę `m` o nowe możliwości opisane klasą `MonadHeap`. Podobnie jak w poprzednim zadaniu użyj transformatora monady stanu, ale tym razem ukrytym stanem będzie słownik (z modułu `Data.Map`) który lokacjom przypisuje wartości. Dodaj też brakujące instancje klasy `MonadFail` tak by można było uruchomić interpreter.

Zadanie 4 (1p). W naszym interpreterze transformator monad z poprzedniego zadania musi być na wierzchu stosu transformatorów monad (dlaczego?). Zaproponuj rozwiązanie bazujące na rodzinach typów, które obchodzi ten problem.

Zadanie 5 (2p). Rozszerz interpreter o implementację operacji wejścia-wyjścia. Konstrukcja `input` powinna wyliczać się do liczby wczytanej ze standardowego wejścia, natomiast konstrukcja `output e` powinna ewaluować wyrażenie `e` do liczby (jeśli jest czymś innym, to należy zgłosić błąd) i ją wyświetlić. Twój interpreter powinien być dalej polimorficzny ze względu na używaną monadę. Dodaj brakujące instancje klas tak by można było uruchomić interpreter z zadania 3.

Wskazówka: Możesz zdefiniować własną klasę typów rozszerzającą klasę `Monad` o potrzebne metody, lub użyć bibliotecznej klasy `MonadIO`.

Zadanie 6 (3p). Zauważ, że w zamieszczonym w SKOSie interpreterze środowisko jest przekazywane jawnie, jako parametr funkcji `eval`. Do przekazywania środowiska można również użyć monad, a dokładniej bibliotecznego transformatora monad `ReaderT`. Zdefiniuj odpowiednią klasę typów, rozszerzającą klasę `Monad`

o nowe metody potrzebne do obsługi środowiska w ewaluatorze i zmodyfikuj ewaluator tak, by nie przekazywał środowiska jawnie. Następnie zdefiniuj odpowiedni transformator monad bazujący na `ReaderT` i zainstaluj go w zdefiniowanej przez siebie klasie. Dodaj też inne brakujące instancje tak, by dało się uruchomić ewaluator. Naiwne rozwiązanie tego zadania prowadzi do ewaluatora języka z dynamicznym wiązaniem zmiennych. Jeśli tak się stało w przypadku Twojego rozwiązania, to się nie przejmuj — naprawisz ten problem w następnym zadaniu.

Wskazówka: Jedna z metod Twojej klasy może mieć sygnaturę: `extendEnv :: Var -> EnvValue m -> m a -> m a`.

Zadanie 7 (1p). Rozwiąż poprzednie zadanie jeszcze raz upewniając się, że teraz Twój interpreter implementuje leksykalne wiązanie zmiennych.

Zadanie 8 (4p). Pokaż, że zadanie pierwsze z listy niepunktowanej (X1) można ładnie rozwiązać używając rodzin typów zamiast wieloparametrowych klas typów. Możesz dodatkowo rozwiązać zadania 2–4 z tej samej listy, by lepiej zilustrować elegancję takiego podejścia.