Lista zagadnień nr 13

Przed zajęciami

Tematem bieżącego tygodnia są **systemy typów** na przykładzie typowanego Racketa. Należy znać pojęcia **typu**, **typu parametrycznego**, **sumy typów**, oraz **zawężania typu** przez predykaty. Przed zajęciami należy zapoznać się z kodem źródłowym z wykładu oraz przejrzeć przewodnik po typowanym Rackecie **The Typed Racket Guide** (`https://docs.racket-lang.org/ts-guide/`).

Na zajęciach

Ćwiczenie 1.

Napisz funkcję `prefixes`, zwracającą wszystkie prefiksy listy podanej jako argument. Nadaj tej funkcji właściwy typ polimorficzny (tzn. wykorzystujący `All`).

Ćwiczenie 2.

Zdefiniuj typy wektorów dwuwymiarowych i trójwymiarowych używając struktur:

```
(struct vector2 ([x : Real] [y : Real]) #:transparent)
(struct vector3 ([x : Real] [y : Real] [z : Real]) #:transparent)
```

Zaimplementuj procedurę `vector-length` obliczającą długość wektora (dwuwymiarowego lub trójwymiarowego).

Można napisać tę procedurę na dwa sposoby – albo używając instrukcji warunkowej, albo dopasowania wzorca. Napisz obie wersje.

Ćwiczenie 3.

Jak widzieliśmy na wykładzie, procedurze `map` możemy nadać następujący kontrakt parametryczny:

```
(parametric->/c [a b] (-> (-> a b) (listof a) (listof b)))
```

W Rackecie z typami możemy nadać jej następujący, analogiczny do powyższego kontraktu typ parametryczny:

```
(All [a b] (-> (-> a b) (Listof a) (Listof b)))
```

Możemy rozważyć zmienione wersje kontraktu i typu powyżej, gdzie zamiast dwóch parametrów `a` i `b` użyjemy tylko jednego, `a`, który zastąpi wszystkie wystąpienia `a` i `b`. Odpowiedz na pytania:

- Jaka błędna implementacja procedury `map` będzie spełniać zmienioną wersję kontraktu i mieć zmienioną wersję typu, a zostanie odrzucona przez wersje oryginalne?
- Czy zmieniona wersja kontraktu ogranicza sposób użytkowania procedury? A zmieniona wersja typu?

Ćwiczenie 4.

Zdefiniuj w typowanym Rackecie typ drzew *rose trees* – to znaczy takich, których liście nie zawierają elementów, natomiast węzły posiadają jedną wartość oraz listę poddrzew. Podobnie jak typ drzew BST z wykładu, zdefiniowany typ powinien być sparametryzowany typem elementu. Zaimplementuj procedurę zwracającą listę elementów takiego drzewa w kolejności preorder.

Ćwiczenie 5.

Zmodyfikuj interpreter prostych wyrażeń arytmetycznych ze zmiennymi i let-wyrażeniami z wykładu szóstego, aby był dobrze otypowany w Rackecie z typami. W tym celu zdefiniuj typ wyrażeń arytmetycznych `Expr`.

Ćwiczenie 6.

Zmodyfikuj interpreter wyrażeń arytmetycznych z wyrażeniami warunkowymi i let-wyrażeniami z wykładu ósmego, aby był dobrze otypowany w Rackecie z typami. Oprócz typu wyrażeń `Expr` będziesz musiał wprowadzić dodatkowy typ `Value` wartości obliczanych przez interpreter. Zwróć uwagę, że mogą nimi być albo liczby rzeczywiste, albo wartości boolowskie. Warto również zdefiniować typ środowisk `Env`.

Niezbędna będzie modyfikacja procedury `op-to-proc`. Zauważ, że procedury zwracane przez `op-to-proc` oczekują wyłącznie liczb jako parametrów,

lecz wartości obliczane przez interpreter uwzględniają też wartości boolowskie. Procedura `op-to-proc` powinna zwracać procedury typu `(-> Value Value Value)`.

Ćwiczenie 7.

Pomimo tego, że interpreter z poprzedniego zadania jest napisany w języku z typami, interpretowany język wciąż jest językiem beztypowym. Napisz (nie używające let-wyrażeń ani zmiennych) wyrażenie w składni abstrakcyjnej tego języka, którego obliczenie generuje błąd.

Wprowadź typy do języka poprzedniego zadania. W tym celu zdefiniuj następujący typ typów wyrażeń:

```
(define-type EType (U 'integer 'boolean))
```

Następnie napisz procedurę `typecheck` o typie `(-> Expr EType Boolean)`. Procedura ta powinna zwracać `#t` wtedy i tylko wtedy, gdy wyrażenie z pierwszego argumentu oblicza się do wartości o typie podanym jako drugi argument.

Do rozwiązania tego zadania mogą być pomocne *środowiska typów*. Środowisko typów różni się od wcześniej poznanych środowisk tym, że jego elementami są typy, a nie wartości.