Lista zagadnień nr 13

Typy w języku Racket

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.

Procedurze `fold-right` możemy nadać następujący kontrakt parametryczny:

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

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

```
(All (a b) (-> (-> a b b) b (Listof a) 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 `fold-right` 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 podstawieniowy parser i interpreter prostych wyrażeń arytmetycznych ze zmiennymi i let-wyrażeniami z wykładu szóstego, aby były dobrze otypowane w Rackecie z typami. W tym celu zdefiniuj typ wyrażeń arytmetycznych `Expr`.

Ćwiczenie 6.

Zmodyfikuj parser i interpreter wyrażeń arytmetycznych z wyrażeniami warunkowymi i let-wyrażeniami z wykładu ósmego, aby były dobrze otypowane 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 tego języka, którego obliczenie generuje błąd.

Zadania domowe

Zadanie 21

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

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

Następnie napisz procedurę `typecheck` o typie `(-> Expr (U EType #f))`. Procedura ta powinna zwracać typ wyrażenia, albo `#f`, jeśli występuje błąd typów (np. w wyrażeniu występuje operator arytmetyczny zaaplikowany do wartości boolowskiej).

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. Środowisko typów odpowiada na pytanie, jakiego typu jest dana zmienna.

Reguły typowania są następujące:

- Stała liczbowa jest typu `'real`.
- Stała boolowska jest typu `'boolean`.
- Zmienna jest takiego typu, jak mówi środowisko.
- Obydwa argumenty operatora arytmetycznego muszą być typu `'real`. Wynik operacji arytmetycznej jest wtedy typu `'real`.
- Obydwa argumenty operatora porównania muszą być typu `'real`. Wynik porównania jest wtedy typu `'boolean`.
- Obydwa argumenty operatora logicznego muszą być typu `'boolean`. Wynik operacji logicznej jest wtedy typu `'boolean`.
- Jeśli pierwsze podwyrażenie w let-wyrażeniu jest typu `t` (dowolnego), to całe wyrażenie jest tego samego typu, co drugie podwyrażenie, otypowane w środowisku rozszerzonym o typ zmiennej `t`.A
- Pierwsze podwyrażenie wyrażenia warunkowego musi być typu `'boolean`. Wtedy typem całego wyrażenia jest typ obu pozostałych podwyrażeń –

drugiego i trzeciego. Jeśli te podwyrażenia mają różne typy, wyrażenie jest źle otypowane.

Rozwiązanie powinno być napisane w typowanym Rackecie, powinno zawierać parser i typechecker, natomiast nie powinno zawierać ewaluatora. Ustalenie typu wyrażenia nie wymaga jego ewaluacji. W pliku rozwiązania wyeksportowane przy użyciu **`provide`** powinny być tylko funkcje `parse` i `typecheck`.