Lista zagadnień nr 12

Przed zajęciami

Tematem bieżącego tygodnia są **systemy typów** na przykładzie języka Plait. Należy znać pojęcia **typu**, **typu parametrycznego**, **typu danych**. Przed zajęciami należy zapoznać się z dokumentacją języka Plait (`https://docs.racket-lang.org/plait/index.html`) oraz zainstalować potrzebny pakiet Racketa, np. poleceniem:

```
raco pkg install plait
```

Plait – język z typami

Ćwiczenie 1.

Napisz funkcję `prefixes`, zwracającą wszystkie prefiksy listy podanej jako argument. Przeczytaj wyinferowany typ tej funkcji i wyjaśnij jego znaczenie.

Ćwiczenie 2.

Zaimplementuj procedurę `sqrt` obliczającą pierwiastek kwadratowy, wzorując się na kodzie z pierwszego wykładu (`w1-kod.rkt`).

Zdefiniuj typ wektorów dwuwymiarowych lub trójwymiarowych:

```
(define-type Vector
    (vector2 [x : Number] [y : Number])
    (vector3 [x : Number] [y : Number] [z : Number]))
```

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

Można napisać tę procedurę na dwa sposoby – albo używając wyrażeń warunkowych, albo analizy przypadków. Napisz obie wersje.

Ćwiczenie 3.

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

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

W języku Plait ta funkcja otrzymuje następujący, analogiczny do powyższego kontraktu typ parametryczny:

```
(('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?

  Uwaga: z powodu nietypowego zachowania interpretera Plaita, aby sprawdzić, czy procedura `fold-right` ma powyższy typ, należy napisać w REPLu:

  ```
  (has-type foldr-right : (('a 'b -> 'b) 'b (Listof 'a) -> 'b))
  ```

  a następnie zwrócić uwagę, czy typ wypisany przez REPLa jest tym, którego żądaliśmy.

- Czy zmieniona wersja kontraktu ogranicza sposób użytkowania procedury? A zmieniona wersja typu?

Ćwiczenie 4.

Zdefiniuj w języku Plait 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.

Zaimplementuj w języku Plait podstawieniowy interpreter wyrażeń arytmetycznych z let-wyrażeniami. Można wzorować się na analogicznym interpreterze z wykładu 7.

Ćwiczenie 6.

Rozszerz interpreter wyrażeń arytmetycznych z let-wyrażeniami i wyrażeniami warunkowymi o operatory unarne (np. operator logiczny `not`). Rozszerz go następnie o pary, dodając trzy nowe operatory (`fst`, `snd`, `pair`) oraz nowy konstruktor wartości.

Zadania domowe

Zadanie 12

Język wyrażeń z let-wyrażeniami i wyrażeniami warunkowymi zaprezentowany na wykładzie jest językiem beztypowym, pomimo tego, że jego interpreter jest zaimplementowany w typowanym języku Plait. Przykładowo, obliczenie wyrażenia `(if 1 2 3)` kończy się błędem wykonania.

Celem zadania będzie wprowadzenie typów do języka z wykładu. W tym celu zdefiniuj następujący typ typów wyrażeń:

```
(define-type Type
    (number-type)
    (boolean-type))
```

Następnie napisz procedurę `typecheck` o typie `(ArithExpr -> (Optionof Type))`. Procedura ta powinna zwracać typ wyrażenia (np. `(some (number-type))`), albo `(none)`, 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 `(number-type)`.
- Stała boolowska jest typu `(boolean-type)`.
- Zmienna jest takiego typu, jak mówi środowisko typów.
- Obydwa argumenty operatora arytmetycznego muszą być typu `(number-type)`. Wynik operacji arytmetycznej jest wtedy typu `(number-type)`.
- Obydwa argumenty operatora porównania muszą być typu `(number-type)`. Wynik porównania jest wtedy typu `(boolean-type)`.

- Obydwa argumenty operatora logicznego muszą być typu `(boolean-type)`. Wynik operacji logicznej jest wtedy typu `(boolean-type)`.
- 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`.
- Pierwsze podwyrażenie wyrażenia warunkowego musi być typu `(boolean-type)`. 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 języku Plait, przez rozszerzenie pliku `let-env-if-plait.rkt` z wykładu. Ustalenie typu wyrażenia nie wymaga jego ewaluacji. Nie należy używać żadnej formy **`provide`** (język Plait nie posiada tej formy).