Implementacja pracy *Linear types can change the world!*

Jarosław Kwiecień

1 Wprowadzenie

W ramach projektu wykonałem interpreter języka zaproponowanego w pracy *Linear types can change the world!*, rozszerzonego podstawowy o mechanizm polimorfizmu. Głównym elementem pracy są liniowe typy, które służą do reprezentacji mutowalnych obiektów oraz wejścia/wyjścia. Ich najważniejszą cechą jest to, że muszą zostać użyte dokładnie raz, czyli nie można ich zduplikować (stworzyć dodatkowych referencji), ani zapomnieć, co w innych językach często skutkuje trudnymi do wykrycia błędami oraz wyciekami pamięci. Dodatkowo wprowadzona została specjalna konstrukcja let pozwalająca przy odpowiednich założeniach użyć zmiennych liniowych wielokrotnie w trybie *tylko do odczytu*.

2 Stan prac nad projektem

Zaimplementowany został niemal w pełni system typów, czyli najważniejszy element projektu. Niewątpliwie wymaga on jednak jeszcze przetestowania. Do napisania pozostał jeszcze ewaluator wyrażeń oraz więcej testów. Dokończenia wymaga też implementacja typów zdefiniowanych w kodzie źródłowym. Ten raport także zostanie jeszcze rozbudowany (m. in. o omówienie polimorfizmu).

3 Składnia (nieformalny opis)

1. Typy
   (a) typ prosty (zwykły lub liniowy): `x | !x`
   (b) krotka: `(type_1, typ_2, ..., typ_n)`
   (c) lista: `[type]`
   (d) tablica (pierwszy wariant tylko do odczytu, drugi liniowy): `[|type|] | ![|typ|]`
   (e) funkcji (funkcja liniowa): `type_1 -> type_2 | type_1 -o type2`
   (f) typ polimorficzny: `forall a, b, ?c, ?d . type`
2. Definicja typu

(a) nieliniowy: `type x = C1 of type_1 | C2 of type_2 | C3 | C4 | ... | Cn of type_n`

(b) liniowy: `type !x = ...`

3. Wzorzec

   - `_ | (pattern_1, ..., pattern_n) | pattern_1 :: pattern_2 | Constructor pattern | !Constructor pattern`

4. Wyrażenie

   (a) liczby naturalne: `12345`, stringi: `"text"`

   (b) zmienna: `x`

   (c) lista: `[e1, e2, ..., en]`

   (d) tablica: `[|e1, e2, ..., en|]`

   (e) krotka: `(e1, e2, ..., en)`

   (f) aplikacja: `e1 e2`

   (g) wyrażenia z operatorami: `e1` *op* `e2`

      - dostępne operatory: `+, -, *, /, >, <, >=, <=, ==, !=, , ||, ::, ;`

   (h) `if cond then e1 else e2`

   (i) `case e of pattern_1 -> e1 | pattern_2 -> e2 | ... | pattern_n -> en`

   (j) `let pattern = e in e1`

   (k) Let z trybem tylko do odczytu: `let {v1, v2, ..., vk} pattern = e in e1`

   (l) Funkcja (nielioniowa lub liniowa) `fun (pattern : type) -> e | fun (pattern : type) -o e`

5. Program - składa się z listy definicji typów, a następnie jednego wyrażenia

4 Wbudowane typy i zmienne

1. Typy proste: `string, int, void`

2. `type bool = True | False`

3. `fix : forall ?a . (a -> a) -> a`

4. `len : forall ?a . [|a|] -> int`

5. `arr_from_elem : forall a . int -> a -> ![|a|]`

6. `arr_from_list : forall ?a . [a] -> ![|a|]`
7. `lookup : forall ?a . int -> [|a|] -> a`
8. `update : forall ?a . int -> a -> ![|a|] -> ![|a|]`
9. `drop : forall ?a . a -> ()`

5 Uruchomienie programu

Aby skompilować program potrzebny jest OCaml w wersji przynajmniej `4.07` oraz dodatkowe zależności `ocamlbuild, ocamlfind, menhir`. Program można zbudować poleceniem `make`. Aby uruchomić program należy wpisać `./main.native ścieżka_do_pliku_źródłowego`, np. `./main.native tests/array.ll`. W obecnej wersji program wczytuje program oraz wypisuje na wyjście sparsowane wyrażenie oraz typ tego wyrażenia.