Lista zadań nr 13

Zadanie 1. (2 pkt)

W interpreterze z pliku `state-store-macros.rkt` mamy do czynienia z niejawnymi referencjami, drugiego rodzaju (rozszerzenie środowiska o wiązanie automatycznie powoduje utworzenie modyfikowalnej komórki pamięci związanej z wiązaną zmienną, przy czym sama referencja nie jest dostępna programiście). Rozszerz interpretowany język o referencje pierwszego rodzaju (referencje jako wartości), a zatem o konstrukcje `box` (utworzenie referencji), `unbox` (odczytanie wartości ze sterty, na którą wskazuje dana referencja) oraz `set-box!` (modyfikacja sterty dla danej referencji).

Zadanie 2. (2 pkt)

Zdefiniuj w języku Racket strumień wszystkich liczb pierwszych, opierając się na następującej obserwacji: dana liczba naturalna jest pierwsza (a więc powinna sie znaleźć w strumieniu), jeżeli nie dzieli sie przez żadną odpowiednio mniejszą liczbę pierwszą. Jest to podejście inne od tego opartego na sicie Eratostenesa, zaprezentowanego na wykładzie.

Zadanie 3. (1 pkt)

Zdefiniuj strumień S, taki, że $S_n = n!$, wykorzystując procedurę `map2` z wykładu do pomnożenia przez siebie dwóch strumieni.

Zadanie 4. (1 pkt)

Zdefiniuj procedurę `partial-sums`, która dla danego strumienia S liczb całkowitych zwróci strumień jego sum częściowych S_0, $S_0 + S_1$, $S_0 + S_1 + S_2$, ... W tym zadaniu z kolei można dodać do siebie dwa strumienie.

Zadanie 5. (2 pkt)

Rozwiąż problem Hamminga: zbuduj strumień różnych rosnąco uporządkowanych liczb naturalnych, które w rozkładzie na czynniki pierwsze zawierają tylko 2, 3 i 5. Napisz najpierw funkcję `merge`, która scala dwa rosnąco uporządkowane strumienie w jeden, niezawierający powtórzeń, a także funkcję `scale`, która mnoży strumień przez liczbę.

Zadanie 6. (2 pkt)

Wzorując się na implementacji strumieni z wykładu (plik `letrec-streams.rkt`) popraw swoje rozwiązanie zadania 1 z listy 12, tak by uczynić swój interpreter prawdziwie leniwym. Twój interpreter powinien liczyć wartość argumentu funkcji oraz wyrażenia definiującego w wyrażeniu `let` co najwyżej raz, tylko przy pierwszym użyciu. Obliczona wartość powinna zostać zapamiętana, a kolejne odwołania do zmiennej z nią związanej powinny polegać na zwróceniu raz obliczonej wartości.

Zadanie 7. (1 pkt)

Dodaj do języka z pliku `state-mc.rkt` wyrażenie {`while` b e}, które reprezentuje pętlę *while* znaną z języków imperatywnych: dopóki zachodzi b, wykonuj e.

Zadanie 8. (1 pkt)

Dodaj do języka z pliku `state-mc.rkt` wyrażenia {`read`} oraz {`write` e}, które pozwalają, odpowiednio, na przeczytanie wartości liczbowej z wejścia oraz na wypisanie wartości liczbowej na wyjście. Interpreter powinien używać odpowiednich operacji z Plaita do zinterpretowania tych rozszerzeń.