Zadania na pracownię nr 10 i 11

Zadanie na pracownię nr 10

W „kodzie z wykładu" zamieszczonym na SKOS-ie znaleźć można prosty test wydajności interpretera języka WHILE (w pliku `w8-while-interp.rkt`). Porównuje on czas potrzebny na interpretację programu napisanego w języku WHILE z czasem wykonania równoważnego programu napisanego w Rackecie. Nie jest to zbyt dobry test (m.in. dlatego, że jest uruchamiany tylko na jednym programie i to tylko raz), ale wynika z niego, że interpretacja programu jest wolniejsza w porównaniu z „natywną" implementacją o jakieś trzy rzędy wielkości. Nie jest to nic niezwykłego – z reguły właśnie tego należy się spodziewać po prostym, niezoptymalizowanym interpreterze języka. Spróbujmy więc zrobić nasz interpreter trochę bardziej wydajnym!

W programach napisanych w językach imperatywnych takich jak WHILE bardzo dużo instrukcji to instrukcje przypisania. Spójrzmy więc, jak nasz interpreter interpretuje tę instrukcję. Oblicza on wartość i odpowiednio modyfikuje aktualne środowisko (poprzez procedurę `env-update`). Ta modyfikacja to w rzeczywistości żadna modyfikacja – ponieważ działamy na trwałych strukturach danych, tak naprawdę wyrzucamy na śmietnik część środowiska i odbudowujemy je tak, by dana zmienna miała inną wartość. Może zamiast tego warto rzeczywiście zmienić wartość w środowisku używając zmiennego stanu?

Zadanie polega na tym, by w pliku `w8-while-interp.rkt` zmienić **reprezentację środowiska** tak, by była to mutowalna lista złożona z `mcons`-ów (a nie, tak jak teraz, trwała lista złożona z `cons`-ów), a procedura `env-update` naprawdę modyfikowała odpowiednią komórkę na liście. Ważne jest, by zachować konwencjonalną abstrakcję, czyli nie zmieniać interfejsu środowisk ani innych części kodu (np. interpretera).

Uwaga! Bardzo ważne jest, by zachować konwencjonalną abstrakcję, czyli nie zmieniać interfejsu środowisk ani innych części kodu (np. interpretera).

Uwaga! Rozwiązanie powinno znajdować się w pliku o nazwie `imie-nazwisko.rkt` (bez polskich znaków). Proszę załączyć testy (zgodnie z metodologią przedstawioną na wykładzie), które przekonają sprawdzającego, że interpreter z nową reprezentacją środowisk wciąż działa poprawnie.

Zadanie na pracownię nr 11

Spróbuj odpowiedzieć na pytanie, czy opłacało się robić optymalizację z poprzedniego zadania. Przetestuj czas działania oryginalnego i zmodyfikowanego interpretera (najlepiej w porównaniu z czasem wykonania równoważnych programów w Rackecie) i wyciągnij odpowiednie wnioski. Jako rozwiązanie prześlij plik zawierający różne testy i – w komentarzu – wnioski z nich płynące, które mówią, czy zaproponowana optymalizacja przyspiesza interpreter na tyle, żeby warto było ją wprowadzić na stałe.

Proszę porządnie przetestować rozwiązania – na więcej niż jednym przykładzie i na różnych danych. Odpowiedzią na główne pytanie nie musi być stanowcze TAK lub NIE, można zamieścić bardziej obszerne opracowanie wyników testów. Proszę szukać odpowiedzi na pytania w rodzaju „Dla jakich programów optymalizacja naprawdę działa?" albo „Czy istnieje chociaż jeden program, dla którego optymalizacja stanowczo skraca czas wykonania?".

Uwaga! Całość rozwiązania musi znajdować się w jednym pliku o nazwie `imie-nazwisko.rkt` (bez polskich znaków) i nie powinno używać zewnętrznych plików – oznacza to, że definicje obu interpreterów i testów powinny być w tym pliku. Ponieważ częścią zadania jest porównanie interpretera z wykładu z interpreterem z optymalizacją, warto sięgnąć po reprezentacje wielorakie (rozdział 2.4 podręcznika) albo zamieścić oba interpretery w osobnych modułach. Używanie modułów w Rackecie jest bardzo proste – wystarczy zerknąć na plik `example-modules.rkt` zamieszczony na SKOS-ie i wszystko stanie się jasne.