Lista zadań nr 10

Poniższe zadania rozwiąż w języku Plait.

Zadanie 1. (2 pkt)

Zmodyfikuj język wyrażeń arytmetycznych i jego interpreter z pliku `arith.rkt`, tak by operatory arytmetyczne mogły przyjmować dowolną, w tym zerową, liczbę argumentów.

Zadanie 2. (2 pkt)

W pliku `arith-vm.rkt` znajduje się definicja maszyny wirtualnej i kompilatora dla wyrażeń arytmetycznych. Zaimplementuj funkcję `decompile`, która dla danego ciągu instrukcji (typu `Code`) zwróci wyrażenie (typu `Exp`) takie, że `(decompile(compile` e`))` $\equiv e$ dla dowolnego wyrażenia e. Podejrzenie zachodzenia takiego twierdzenia wzmocnij odpowiednim zestawem testów.

Zadanie 3. (2 pkt)

W pliku `arith-am.rkt` znajduje się definicja maszyny abstrakcyjnej dla wyrażeń arytmetycznych. Zaimplementuj podobną maszynę abstrakcyjną dla języka wyrażeń z pliku `bool.rkt`, z którego usuwamy wyrażenia `cond`, ale dodajemy stałe prawdy `#t` i fałszu `#f`. Otrzymana maszyna powinna być ekstensjonalnie zgodna (na wspólnym podzbiorze wyrażeń) z interpreterem wyrażeń logicznych z pliku `bool.rkt`.

Zadanie 4. (1 pkt)

Interpreter wyrażeń z pliku `bool.rkt` przeplata ewaluację z odcukrzaniem wyrażeń cond. Zaimplementuj odcukrzanie jako oddzielną fazę w łańcuchu wykonania programu, która następuje po parsowaniu a przed ewaluacją. Zastanów się czy nie warto wprowadzić dodatkowego języka, który nie będzie zawierał wyrażeń `cond`.

Zadanie 5. (2 pkt)

Rozszerz język i interpreter wyrażeń z pliku `bool.rkt` o listy. W szczególności, język powinien zawierać operacje `cons`, `car`, `cdr`, `null?`, `null` i `list`.

Zadanie 6. (1 pkt)

Zmodyfikuj ewaluator z pliku `error-ans-monad-macros.rkt` tak, by błędy `''type error''` były zgłaszane przez zdefiniowaną przez nas funkcję `err`, a nie przez funkcję `error`, pochodzącą z Plaita. Jak zmienią się typy funkcji pomocniczych związanych z operatorami arytmetycznymi i logicznymi?

Dodatkowo, zadbaj o to by ewaluator zgłaszał błąd (przy pomocy funkcji `err`) w razie wystąpienia próby dzielenia przez 0.

Zadanie 7. (2 pkt)

Używając mechanizmu makr, a w szczególności kostrukcji `define-syntax`, zdefiniuj w języku Plait własne *wieloargumentowe* operacje koniunkcji `my-and` i alternatywy `my-or` w postaci form specjalnych, z leniwą semantyką, tzn. argumenty ewaluujemy od lewej do prawej i tylko tyle ile trzeba by wyznaczyć wartość danej operacji logicznej. Nie używaj form specjalnych `and` i `or`.

Zdefiniuj następnie formy specjalne `my-let` i `my-let*` odpowiadające konstrukcjom `let` i `let*` z Racketa. Nie używaj konstrukcji `let` i `let*`.