Lista zadań nr 10

Zadanie 1. (2 pkt)

Notacja polska to beznawiasowy zapis wyrażeń, w którym operator znajduje się przed argumentami. Np wyrażenie $(3+8)+4*(2-1)$ zapisujemy w notacji polskiej jako $++38*4-21$. Napisz kompilator ze składni abstrakcyjnej (`Ast.expr`) do notacji polskiej.

Ciekawostka: Notacja polska jest podobna do zwykłej notacji aplikacji funkcji do argumentu $f(x,y)$. Np. gdyby używać symboli operacji arytmetycznych jako nazw funkcji, wyrażenie $(3+8)+4*(2-1)$ zapisalibyśmy jako:

$$+(+(3,8),*(4,-(2,1)))$$

A wymazując nawiasy:

$$+\ +\ 3\ 8\ \ *\ 4\ -\ 2\ 1$$

Zadanie 2. (2 pkt)

Zaimplementuj ewaluator wyrażeń w notacji polskiej, który czyta wyrażenie od lewej do prawej (czyli tak samo jak w przypadku odwrotnej notacji poslkiej) i używa stosu do obliczenia wartości wyrażenia.

Wskazówka: W tym wypadku stos będzie musiał przechowywać więcej rodzajów elementów. Obliczając wartość wyrażenia $+35$ nie możemy zrobić dodawania, dopóki nie dotrzemy do liczby 5. Ale, gdy dotrzemy do 5, symbol $+$ i liczba 3 muszą gdzieś być zapamiętane (na stosie).

Zadanie 3. (2 pkt)

Dla danego wyrażenia arytmetycznego wyrażonego w odwrotnej notacji polskiej, można wyliczyć, jak dużego stosu potrzeba, żeby obliczyć jego wartość (można to wykorzystać w „dorosłym" kompilatorze, żeby od razu zaalokować potrzebną pamięć, a nie liczyć na jeden globalny stos). Zdefiniuj funkcję

`stack_size : prog -> int`, która oblicza maksymalną wysokość, którą osiąga stos w trakcie obliczania wartości wyrażenia.

Zadanie 4. (2 pkt)

Dodaj do definicji typu `cmd` nowy konstruktor `PushVar`, reprezentujący zmienną (w podobnym stylu, jak miało to miejsce to w składni wyrażeń `Ast.expr`). Rozbuduj interpreter wyrażeń w odwrotnej notacji polskiej tak, by przyjmował on jeszcze środowisko, mówiące jakie są wartości tych zmiennych.

Na przykład wartością wyrażenia `[PushInt 10, PushVar "x", Binop Add]` w środowisku, które przypisuje zmiennej `x` wartość 3, jest wartość 13.

Zadanie 5. (2 pkt)

Rozbuduj zadanie 3. do składni z zadania 4. Oczywiście, funkcja `stack_size` nadal ma typ `prog -> int` (czyli nie przyjmuje środowiska). W końcu kompilator w ogólności nie może wiedzieć, w jakim środowisku zostanie wykonany dany kawałek kodu.

Zadanie 6. (2 pkt)

Aktualna implementacja porównania (=) w kompilatorze porównuje po prostu wartości po dereferencji wskaźnika ze stosu. To podejście w oczywisty sposób nie działa, jeśli pozwolimy na pary w języku. Rozbuduj kompilator tak, by wykonywane było strukturalne porównanie wartości (z użyciem tagów, o których prowadzący mówił na wykładzie, ale które pojawiły się w kompilatorze post-factum na skosie).

Uwaga: Zadanie to może wymagać edytowania runtime systemu, żeby zaimplementować funkcję porównującą wartości w C. Będzie ona oczywiście miała strukturę analogiczną do znajdującej się tam funkcji wypisującej wartości na ekran.

Ciekawostka: Tak właśnie dzieje się w OCamlu, w którym operacja = zaimplementowana jest jako funkcja `https://github.com/ocaml/ocaml/blob/trunk/runtime/compare.c#L337` w runtimie.