Języki programowania

II UWr 2019/20

Lista zadań nr 8 (bonusowa)

Na ćwiczenia 3 grudnia 2019

Zadanie 1. W implementacji ewaluatora na wykładzie przyjęliśmy (bez głębszego zastanowienia) interpretację kontekstów ewaluacyjnych *od zewnątrz* (outside-in). Przyjmij interpretację *od wewnątrz* (inside-out) i zdefiniuj odpowiednie funkcje dekompozycji i wkładania wyrażenia w kontekst (typy powinny pozostać niezmienione). *Wskazówka*: aby zaimplementować dekompozycję, potrzebne będą dwie wzajemnie rekurencyjne funkcje pomocnicze:

- `decE : expr -> econt -> decomp`, zdefiniowana rekurencyjnie względem wyrażenia,
- `decC : econt -> value -> decomp`, zdefiniowana rekurencyjnie względem kontekstu ewaluacyjnego.

Język imperatywny W

Poniższa gramatyka definiuje składnię prostego języka imperatywnego, który nazywamy **W**[1]. W gramatyce x oznacza kategorię syntaktyczną „zmiennych"[2] naszego języka imperatywnego, zaś n — liczby całkowite.

$$
\begin{aligned}
e &::= x \mid \underline{n} \mid e + e \mid e * e \mid -e \\
c &::= \mathsf{skip} \mid c; c \mid \mathsf{if}\ e\ \mathsf{then}\ c\ \mathsf{else}\ c \mid \mathsf{while}\ e\ \mathsf{do}\ c \mid x := e \mid \mathsf{var}\ x := e\ \mathsf{in}\ c \\
p &::= (x\ \underline{n})^*\ \mathsf{in}\ c
\end{aligned}
$$

Wyrażenia e naszego języka obejmują operacje arytmetyczne (sumę, iloczyn, negację), literały całkowitoliczbowe (tworzone z liczb przy pomocy podkreślenia) i zmienne. Instrukcje c to instrukcja identycznościowa skip, złożenie sekwencyjne instrukcji, instrukcja warunkowa, pętla while, instrukcja przypisania i instrukcja var, która definiuje nową zmienną x (o początkowej wartości danej wyrażeniem e), której możemy używać w instrukcji c. Przyjmujemy że var wiąże nazwę zmiennej w instrukcji, zgodnie ze zwykłymi konwencjami leksykalnymi (w szczególności dopuszczamy przemianowania zmiennych związanych). Program p to instrukcja c wraz z pewnymi zmiennymi i ich wartościami początkowymi (sekwencja $(x\ \underline{n})^*$).

Zastanówmy się nad nieformalną semantyką języka **W**. Jest ona w dużej mierze zgodna z semantyką znaną z języków takich jak C, choć znacznie uproszczoną. Jedynymi wartościami jakie rozważamy są liczby całkowite: wyrażenia arytmetyczne e powinny obliczać się do liczb naturalnych przy pewnej pamięci opisującej wartości zmiennych które mogą występować w tym wyrażeniu. Instrukcje w języku imperatywnym nie mają wartości; są za to *transformatorami pamięci*: wykonanie instrukcji w pewnej pamięci początkowej daje nam pewną pamięć końcową, w której wartości niektórych zmiennych mogą być zmienione. Zmiany wprowadza do pamięci instrukcja przypisania, obliczając najpierw wartość wyrażenia arytmetycznego. Mamy dostępne instrukcję warunkową if, wybierającą lewą gałąź obliczeń jeśli wartość wyrażenia arytmetycznego jest niezerowa, a prawą —

[1] Język ten jest zmodyfikowaną wersją języka prostego języka imperatywnego pojawiającego się w wielu książkach pod nazwą IMP lub WHILE.

[2] Zmienna jest bardzo niefortunną nazwą dla modyfikowalnych komórek pamięci w językach imperatywnych, ale nie będziemy walczyli z powszechnie używanym nazewnictwem.

gdy jest ona zerem, a także instrukcję pętli while, która kontynuuje obliczanie swego ciała do momentu gdy w uzyskanej pamięci wartością wyrażenia e będzie zero. Instrukcja var modeluje zakres zmiennych: zmienna x wprowadzona przez instrukcję var jest dostępna wyłącznie na czas wykonywania tej instrukcji (z wartością początkową daną przez e i, oczywiście, z możliwością modyfikacji). Wartością całego programu jest pamięć w której zakończą się obliczenia instrukcji (zawierająca nowe wartości dla zmiennych z listy inicjalizacyjnej).

Zadanie 2. Zdefiniuj osądy mówiące że dane wyrażenie, instrukcja i program są *dobrze sformowane*, tj. używają tylko zmiennych które są dla nich widoczne.

Zadanie 3. Zdefiniuj semantykę ewaluacyjną (naturalną/dużych kroków) języka **W**. Jak w poprzednim zadaniu będziesz potrzebować osądów dla wyrażeń, instrukcji i programów.

Zadanie 4. Zdefiniuj strukturalną semantykę operacyjną (SOS/małych kroków) języka **W**. *Wskazówka*: Zastanów się co może stać się z instrukcją w jednym kroku obliczeń.

Zadanie 5. Udowodnij równoważność semantyk z dwóch poprzednich zadań.

Zadanie 6. Używając semantyki małych kroków udowodnij że poprawnie sformowane programy nie prowadzą do błędu (tj. stanu w którym nie możemy wykonać kroku, ale który nie jest stanem końcowym obliczeń).

Zadanie 7. Używając wybranej semantyki udowodnij że poprawnie sformowane programy w języku **W** mogą się zapętlać. *Wskazówka*: Zastanów się co (w wybranej semantyce) oznacza zapętlenie się.