Semantyka i weryfikacja programów 2023/24.
Egzamin 2/02/2024, zadanie 2 (semantyka denotacyjna)

Poniżej zaproponowana jest składnia pewnego języka programowania.

$$
\begin{aligned}
\textbf{Var} \ni x &::= x \mid y \mid z \mid \ldots \\
\textbf{WVar} \ni w &::= u \mid v \mid w \mid \ldots \\
\textbf{JVar} \ni j &::= i \mid j \mid k \mid \ldots \\
\textbf{Num} \ni n &::= \ldots \mid -1 \mid 0 \mid 1 \mid \ldots \\
\textbf{Expr} \ni e &::= n \mid x \mid e_1 + e_2 \mid e_1 - e_2 \mid e_1 \cdot e_2 \\
\textbf{Inst} \ni I &::= \textbf{skip} \mid x := e \mid I_1; I_2 \mid \textbf{if } e >= 0 \textbf{ then } \{I_1\} \textbf{ else } \{I_2\} \mid \\
&\quad\ \textbf{while}(w)\ e >= 0 \textbf{ do } \{I\} \mid \\
&\quad\ \textbf{break } w \mid \textbf{continue } w \mid \\
&\quad\ \textbf{save } j \mid \textbf{jump } j
\end{aligned}
$$

Wyrażenia i instrukcje oprócz wymienionych poniżej mają standardowe znaczenie. Można założyć, że wszystkie zmienne indywiduowe x, y,... są widoczne globalnie i mają początkowo określone wartości (są zainicjowane). Wyrażenia arytmetyczne wyliczają się do wartości liczbowych **Num** bez efektów ubocznych. Można założyć, że dana jest semantyka dla wyrażeń w stylu denotacyjnym, należy jedynie **jawnie zadeklarować** (sensowny) typ funkcji semantycznej.

Pętle **while** w tym języku posiadają dodatkowy identyfikator w ze zbioru **WVar**. Ich działanie jest analogiczne do standardowych pętli **while**. Dodatkowe instrukcje **break** w i **continue** w działają tak jak zwykłe instrukcje **break** i **continue**, ale to działanie dotyczy najbardziej wewnętrznej pętli **while** o podanym identyfikatorze: **break** w natychmiast przerywa działanie tej pętli, natomiast **continue** w powoduje ponowne rozpoczęcie obliczania danej pętli, począwszy od sprawdzenia jej warunku. Działanie tych instrukcji poza ciałem pętli o tym identyfikatorze w może nie być określone.

Dodatkowo, w języku wprowadzone są instrukcje **save** j oraz **jump** j, gdzie identyfikatory j pochodzą ze zbioru **JVar**. Instrukcja **save** j zapisuje pod identyfikatorem j aktualne miejsce wykonywania programu **wraz z aktualnym kontekstem zagnieżdżeń pętli while**. Instrukcja **jump** j powoduje natychmiastowy skok tuż za miejsce programu, w którym po raz ostatni została wykonana instrukcja **save** j (z tym samym identyfikatorem j). Dodatkowo, w trakcie tego skoku **przywrócony** zostaje kontekst pętli **while** działających w momencie ostatniego wykonania tej instrukcji **save** j. Jeżeli żadna instrukcja **save** j nie była jeszcze wykonana, to działanie instrukcji **jump** j (z tym identyfikatorem j) może nie być określone.

Zadanie

Zadanie polega na napisaniu semantyki denotacyjnej w stylu kontynuacyjnym dla kategorii syntaktycznej **Inst** instrukcji powyższego języka. W tym celu należy jednoznacznie zdefiniować typy funkcji semantycznych dla **Inst** oraz **Expr** wraz ze **wszystkimi** używanymi typami pomocniczymi. Można założyć, że dany jest typ **Ans** oznaczający finalne wyniki działania programu. Należy też podać równania semantyczne dla **wszystkich** postaci instrukcji **Inst** w tym języku.

verte!

Przykład

Rozważmy następującą instrukcję tego języka:

```
 0: x := 0;
 1: y := 1;
 2: while(u) y >= 0 do {
 3:   while(v) x >= 0 do {
 4:     save i;
 5:     y := y - 1;
 6:     if y >= 0 then {
 7:       break u
        } else {
 8:       x := x - 1;
 9:       y := y + 3;
10:       continue v
        }
      };
11:   jump j
    };
12: save j;
13: if y - 1 >= 0 then { skip } else { jump i }
```

Wyliczenie tej instrukcji w kolejnych linijkach spowoduje:

- `0:` Ustawienie wartości x na 0.
- `1:` Ustawienie wartości y na 1.
- `2:` Sprawdzenie warunku `y >= 0` i wejście do pętli `while` o identyfikatorze u.
- `3:` Sprawdzenie warunku `x >= 0` i wejście do pętli `while` o identyfikatorze v.
- `4:` Ustawienie punktu skoku i.
- `5:` Zmniejszenie wartości y do 0.
- `6:` Sprawdzenie warunku `y >= 0` i wykonanie pozytywnej gałęzi.
- `7:` Wykonanie instrukcji `break` dla pętli o identyfikatorze u.
- `12:` Ustawienie punktu skoku j.
- `13:` Sprawdzenie warunku `y - 1 >= 0` i wykonanie negatywnej gałęzi, czyli instrukcji `jump` do punktu o identyfikatorze i.
- `5:` Zmniejszenie wartości y do -1.
- `6:` Sprawdzenie warunku `y >= 0` i wykonanie negatywnej gałęzi.
- `8:` Zmniejszenie wartości x do -1.
- `9:` Zwiększenie wartości y do 2.
- `10:` Wykonanie instrukcji `continue` dla pętli o identyfikatorze v.
- `3:` Sprawdzenie warunku `x >= 0` i zakończenie pętli `while` o identyfikatorze v.
- `11:` Wykonanie instrukcji `jump` do punktu o identyfikatorze j.
- `13:` Sprawdzenie warunku `y - 1 >= 0` i wykonanie pozytywnej gałęzi, czyli instrukcji `skip`.

Na koniec działania programu zmienna x ma wartość -1 natomiast zmienna y ma wartość 2.

Przykładowe rozwiązanie

Rozważmy następujące typy pomocnicze:

$$\begin{aligned}
\mathbf{Vtore} &= \mathbf{Var} \rightharpoonup \mathbf{Num} \\
\mathbf{Jtore} &= \mathbf{JVar} \rightharpoonup \mathbf{Cont} \\
\mathbf{Store} &= \mathbf{Vtore} \times \mathbf{Jtore} \\
\mathbf{WEnv} &= \mathbf{WVar} \rightharpoonup (\mathbf{Cont} \times \mathbf{Cont}) \\
\mathbf{Cont} &= \mathbf{Store} \rightharpoonup \mathbf{Ans}
\end{aligned}$$

Używać będziemy funkcji semantycznych:

$$\begin{aligned}
\mathcal{E} &\colon \mathbf{Expr} \to \mathbf{Vtore} \rightharpoonup \mathbf{Num} \\
\mathcal{I} &\colon \mathbf{Inst} \to \mathbf{WEnv} \to \mathbf{Cont} \rightharpoonup \mathbf{Cont}
\end{aligned}$$

Funkcja semantyczna $\mathcal{E}$ jest dana, natomiast funkcję $\mathcal{I}$ zdefiniujemy następująco:

$$\mathcal{I}[\![\mathbf{skip}]\!]\ \rho_W\ \kappa = \kappa$$

$$\mathcal{I}[\![x := e]\!]\ \rho_W\ \kappa = \lambda(s_V, s_J).\ \kappa\ \big(s_V[x \mapsto \mathcal{E}[\![e]\!]\ s_V], s_J\big)$$

$$\mathcal{I}[\![I_1; I_2]\!]\ \rho_W\ \kappa = \mathcal{I}[\![I_1]\!]\ \rho_W\ \big(\mathcal{I}[\![I_2]\!]\ \rho_W\ \kappa\big)$$

$$\begin{aligned}
\mathcal{I}[\![\mathbf{if}\ e >= 0\ \mathbf{then}\ \{I_1\}\ \mathbf{else}\ \{I_2\}]\!]\ \rho_W\ \kappa = \lambda(s_V, s_J).\ & \\
\mathrm{ifte}\big(&\mathcal{E}[\![e]\!]\ s_V \geq 0, \\
&\mathcal{I}[\![I_1]\!]\ \rho_W\ \kappa\ (s_V, s_J), \\
&\mathcal{I}[\![I_2]\!]\ \rho_W\ \kappa\ (s_V, s_J)\big)
\end{aligned}$$

$$\begin{aligned}
\mathcal{I}[\![\mathbf{while}(w)\ e >= 0\ \mathbf{do}\ \{I\}]\!]\ \rho_W\ \kappa = \mathrm{let}\ \Phi(\kappa_w) = \lambda(s_V, s_J). & \\
\mathrm{let}\ \rho'_W = \rho_W[w \mapsto (\kappa, \kappa_w)]\ \mathrm{in} & \\
\mathrm{ifte}\big(&\mathcal{E}[\![e]\!]\ s_V \geq 0, \\
&\mathcal{I}[\![I]\!]\ \rho'_W\ \kappa_w\ (s_V, s_J), \\
&\kappa\ (s_V, s_J)\big) \\
\mathrm{in}\ \mathrm{Fix}(\Phi) &
\end{aligned}$$

$$\mathcal{I}[\![\mathbf{break}\ w]\!]\ \rho_W\ \kappa = \mathrm{let}\ (\kappa_B, \kappa_C) = \rho_W\ w\ \mathrm{in}\ \kappa_B$$

$$\mathcal{I}[\![\mathbf{continue}\ w]\!]\ \rho_W\ \kappa = \mathrm{let}\ (\kappa_B, \kappa_C) = \rho_W\ w\ \mathrm{in}\ \kappa_C$$

$$\mathcal{I}[\![\mathbf{save}\ j]\!]\ \rho_W\ \kappa = \lambda(s_V, s_J).\ \kappa\ \big(s_V, s_J[j \mapsto \kappa]\big)$$

$$\mathcal{I}[\![\mathbf{jump}\ j]\!]\ \rho_W\ \kappa = \lambda(s_V, s_J).\ s_J\ j\ (s_V, s_J)$$