Semantyka i weryfikacja programów 2025/26.
Egzamin I termin, 26/01/2026, zadanie 1 (semantyka)

Poniżej zaproponowana jest składnia pewnego języka programowania.

$$
\begin{aligned}
\textbf{Var} \ni x &::= x \mid y \mid z \mid \ldots \\
\textbf{PName} \ni P &::= P \mid Q \mid R \mid \ldots \\
\textbf{Num} \ni n &::= \ldots \mid \text{-1} \mid 0 \mid 1 \mid \ldots \\
\textbf{Expr} \ni e &::= n \mid x \mid e_1 + e_2 \mid e_1 - e_2 \mid e_1 * e_2 \\
\textbf{Decl} \ni d &::= \varepsilon \mid \textbf{var}\ x = e \mid \textbf{proc}\ P\{I\} \mid d_1; d_2 \\
\textbf{Inst} \ni I &::= \textbf{begin}\ d\ \textbf{in}\ I\ \textbf{end} \mid x := e \mid I_1; I_2 \mid \textbf{if}\ e > 0\ \textbf{then}\ \{I_1\}\ \textbf{else}\ \{I_2\} \mid \\
&\qquad \textbf{call}\ P \mid \textbf{again} \mid \textbf{back} \mid \textbf{exit}
\end{aligned}
$$

Wyrażenia, deklaracje zmiennych (z nadaną wartością początkową) i bezparametrowych, rekurencyjnych procedur oraz instrukcje, oprócz wymienionych poniżej, mają standardowe znaczenie. Zmienne i procedury lokalne są deklarowane w obrębie bloków **begin**...**in**...**end**. Widoczność identyfikatorów zmiennych i nazw procedur jest statyczna ze standardowymi regułami przesłaniania. Procedury sa bezparametrowe (i nie zwracają wartości), zmienne i nazwy procedur użyte wewnątrz ciał procedur są wiązane statycznie, procedury dopuszczają wywołania rekurencyjne. Instrukcja **call** P wywołuje procedurę o nazwie P widoczną w danym miejscu kodu. Wyrażenia arytmetyczne wyliczają się do wartości liczbowych **Num** bez efektów ubocznych.

Szczególną cechą języka są wymienione niżej instrukcje, których wykonanie wewnątrz ciała procedury powoduje:

- **again** — przerwanie zwykłego wykonywania aktualnego wywołania procedury i ponowne wykonanie jej ciała od początku (skok do początku ciała procedury);
- **back** — skok na początek instrukcji w najmniejszym wykonywanym w tym momencie bloku **begin**...**in**...**end** (czyli bezpośrednio za słowo **in** w tym bloku); to wymaga przerwania wykonywania przynajmniej jednego aktualnego wywołania procedury;
- **exit** — zakończenie działania bloku, w którym aktualnie wywołana procedura została zadeklarowana, i skok bezpośrednio za ten blok; to też wymaga przerwania wykonywania przynajmniej jednego aktualnego wywołania procedury.

Znaczenie instrukcji **again**, **back** i **exit** poza ciałem procedury może być dowolne. Podobnie, można nie określać znaczenia wywołań **call** P w miejscu, gdzie nie jest widoczna żadna procedura o nazwie P, ani przypisań $x := e$, gdzie zmienna x nie jest dostępna.

Zadanie

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

Można przyjąć, że dany jest nieskończony zbiór lokacji **Loc** oraz funkcja matematyczna newloc: $(\textbf{Loc} \rightharpoonup_{\text{fin}} \textbf{Num}) \to \textbf{Loc}$ spełniająca $\text{newloc}(s) \notin \text{dom}(s)$ dla s: $\textbf{Loc} \rightharpoonup_{\text{fin}} \textbf{Num}$ (gdzie $\textbf{Loc} \rightharpoonup_{\text{fin}} \textbf{Num}$ to zbiór funkcji częściowych z **Loc** do **Num**, których dziedzina jest skończona).

verte!

Przykład Wykonanie poniższej instrukcji spowoduje w kolejnych liniach kodu:

```
 0: begin var x = 2;
 1:       proc P { if x > 0 then { back }
 2:                else { if x + 1 > 0 then { x := x - 1; again }
 3:                       else { x := x - 1 } } }
 4: in begin in x := x - 1;
 5:             call P end;
 6:    begin var y = 2;
 7:          proc Q { if x > 0 then { x := x - 1; call Q }
 8:                   else { if y > 0 then { back }
 9:                          else { if y + 1 > 0
10:                                 then { begin in y := y - 1;
11:                                                if y + 2 > 0 then { call Q }
12:                                                else { x := x - 2; exit } }
13:                                 else { back } } } }
14:    in x := x + 3;
15:       begin var x = 7 in y := y - 1;
16:                          call Q end;
17:    end;
18:    call P
19: end
```

0) Zadeklarowanie zmiennej x_0 o wartości 2.
1) Zadeklarowanie procedury P, wiążącej statycznie zmienną x_0.
4) Zmniejszenie wartości zmiennej x_0 do wartości 1.
5) Wywołanie procedury P.
1) Warunek zachodzi; **back** przerywa wywołanie P i powoduje skok do instrukcji bloku z linii 4.
4) Zmniejszenie wartości zmiennej x_0 do wartości 0.
5) Drugie wywołanie procedury P.
1,2) Zachodzi warunek z linii 2; wartość x_0 zmniejsza się do -1; **again** powoduje powrót do początku ciała procedury P.
1,2,3) Warunki z linii 1 i 2 nie zachodzą; wartość x_0 zmniejsza się do -2, co kończy wywołanie P.
6) Zadeklarowanie zmiennej y_0 o wartości 2.
7) Zadeklarowanie procedury Q, wiążącej statycznie zmienne x_0 i y_0.
14) Zwiększenie wartości x_0 do 1.
15,16) Zadeklarowanie zmiennej x_1 o wartości 7; zmniejszenie y_0 do 1; wywołanie procedury Q.
7) Warunek zachodzi; zmniejszenie x_0 do 0; drugie, rekurencyjne wywołanie Q.
7,8) Zachodzi warunek z linii 8; **back** powoduje przerwanie wywołań Q (obu aktualnych wywołań: rekurencyjnego z linii 7 i zewnętrznego z linii 16) i skok do instrukcji bloku z linii 15.
15,16) Zmniejszenie y_0 do 0, kolejne wywołanie procedury Q.
7,8,9) Zachodzi warunek z linii 9.
10) Zmniejszenie y_0 do -1.
11) Warunek zachodzi; rekurencyjne wywołanie Q.
7,8,9) Żaden z tych warunków nie zachodzi.
13) **back** powoduje przerwanie rekurencyjnego wywołania Q i skok do instrukcji bloku z linii 10 (w zewnętrznym wywołaniu Q).
10) Zmniejszenie y_0 do -2.
11) Warunek nie zachodzi.
12) Zmniejszenie x_0 do -2; **exit** przerywa wywołanie procedury Q i powoduje skok bezpośrednio za blok z linii 6-17.
18) Ponowne wywołanie procedury P.
1,2,3) Warunki zlinii 1 i 2 nie zachodzą; zmniejszenie x_0 do -3 kończy to wywołanie P, całej rozważanej instrukcji (i przykład).

Rozwiązanie

Rozważmy następujące typy pomocnicze:

$$\begin{aligned}
\mathbf{Env} &= \mathbf{Var} \rightharpoonup \mathbf{Loc} \\
\mathbf{Store} &= \mathbf{Var} \rightharpoonup \mathbf{Num} \\
\mathbf{Cont} &= \mathbf{Store} \rightharpoonup \mathbf{Ans} \\
\mathbf{PEnv} &= \mathbf{PName} \rightharpoonup \mathbf{Proc} \\
\mathbf{Proc} &= \mathbf{Cont} \to \mathbf{Cont} \to \mathbf{Cont} \\
&\qquad (\mathbf{back} \mapsto \quad \text{za} \quad \mapsto \ \text{przed})
\end{aligned}$$

Używać będziemy funkcji semantycznych:

$$\begin{aligned}
\mathcal{E} &\colon \mathbf{Expr} \to \mathbf{Env} \to \mathbf{Store} \rightharpoonup \mathbf{Num} \\
\mathcal{D} &\colon \mathbf{Decl} \to \mathbf{Cont} \to (\mathbf{PEnv}, \mathbf{Env}, \mathbf{Store}) \rightharpoonup (\mathbf{PEnv}, \mathbf{Env}, \mathbf{Store}) \\
&\qquad (\mathbf{exit} \ \mapsto \ldots) \\
\mathcal{I} &\colon \mathbf{Inst} \to \mathbf{PEnv} \to \mathbf{Env} \to \mathbf{Cont}^3 \to \mathbf{Cont} \to \mathbf{Cont} \rightharpoonup \mathbf{Cont} \\
&\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad (\mathbf{back} \ \mapsto \quad \text{za} \quad \mapsto \text{przed}) \\
&\qquad\qquad\qquad\qquad \overbrace{(\mathbf{exit}\ (\kappa_E), \mathbf{again}\ (\kappa_A), \mathbf{back}\ (\kappa_B))}
\end{aligned}$$

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

$$\begin{aligned}
\mathcal{D}[\![\varepsilon]\!]\ \kappa_E\ (\rho_P, \rho_V, s) &= (\rho_P, \rho_V, s) \\
\mathcal{D}[\![\mathbf{var}\ x = e]\!]\ \kappa_E\ (\rho_P, \rho_V, s) &= \text{let } q = \mathcal{E}[\![e]\!]\ \rho_V\ s \text{ in} \\
&\qquad \text{let } \ell = \text{newloc}(s) \text{ in} \\
&\qquad \text{let } \rho_V' = \rho_V[x \mapsto \ell] \text{ in} \\
&\qquad \text{let } s' = s[\ell \mapsto q] \text{ in} \\
&\qquad\quad \big(\rho_P, \rho_V', s'\big) \\
\mathcal{D}[\![\mathbf{proc}\ P\{I\}]\!]\ \kappa_E\ (\rho_P, \rho_V, s) &= \text{let rec } X = \lambda\kappa_B.\ \lambda\kappa. \\
&\qquad\qquad \mathcal{I}[\![I]\!]\ \rho_P[P \mapsto X]\ \rho_V\ (\kappa_E, (X\ \kappa_B\ \kappa), \kappa_B)\ \kappa_B\ \kappa \text{ in} \\
&\qquad \big(\rho_P[P \mapsto X], \rho_V, s\big) \\
\mathcal{D}[\![d_1; d_2]\!]\ \kappa_E &= \big(\mathcal{D}[\![d_1]\!]\ \kappa_E\big); \big(\mathcal{D}[\![d_2]\!]\ \kappa_E\big)
\end{aligned}$$

$$\begin{aligned}
\mathcal{I}[\![\textbf{begin } d \textbf{ in } I \textbf{ end}]\!]\ \rho_P\ \rho_V\ \vec{\kappa} &= \lambda\kappa_B^{any}.\ \lambda\kappa.\ \lambda s. \\
&\quad \text{let } (\rho_P', \rho_V', s') = \mathcal{D}[\![d]\!]\ \kappa\ (\rho_P, \rho_V, s) \text{ in} \\
&\quad \text{let rec } \kappa_B' = \mathcal{I}[\![I]\!]\ \rho_P'\ \rho_V'\ \vec{\kappa}\ \kappa_B'\ \kappa \text{ in} \\
&\quad \kappa_B'\ s' \\
\mathcal{I}[\![\textbf{skip}]\!]\ \rho_P\ \rho_V\ \vec{\kappa}\ \kappa_B'\ \kappa &= \kappa \\
\mathcal{I}[\![x := e]\!]\ \rho_P\ \rho_V\ \vec{\kappa}\ \kappa_B'\ \kappa &= \lambda s.\ \kappa\ s[\rho_V(x) \mapsto \mathcal{E}[\![e]\!]\ \rho_V\ s] \\
\mathcal{I}[\![I_1; I_2]\!]\ \rho_P\ \rho_V\ \vec{\kappa}\ \kappa_B'\ \kappa &= \mathcal{I}[\![I_1]\!]\ \rho_P\ \rho_V\ \vec{\kappa}\ \kappa_B'\ \big(\mathcal{I}[\![I_2]\!]\ \rho_P\ \rho_V\ \vec{\kappa}\ \kappa_B'\ \kappa\big) \\
\mathcal{I}[\![\textbf{if } e > 0 \textbf{ then } I_1 \textbf{ else } I_2]\!]\ \rho_P\ \rho_V\ \vec{\kappa}\ \kappa_B'\ \kappa &= \lambda s.\text{ifte}\big(\mathcal{E}[\![e]\!]\ \rho_V\ s > 0, \\
&\qquad \mathcal{I}[\![I_1]\!]\ \rho_P\ \rho_V\ \vec{\kappa}\ \kappa_B'\ \kappa\ s, \\
&\qquad \mathcal{I}[\![I_2]\!]\ \rho_P\ \rho_V\ \vec{\kappa}\ \kappa_B'\ \kappa\ s\big) \\
\mathcal{I}[\![\textbf{call } P]\!]\ \rho_P\ \rho_V\ (\kappa_E, \kappa_A, \kappa_B)\ \kappa_B'\ \kappa &= \rho_P\ P\ \kappa_B'\ \kappa \\
\mathcal{I}[\![\textbf{again}]\!]\ \rho_P\ \rho_V\ (\kappa_E, \kappa_A, \kappa_B)\ \kappa_B'\ \kappa &= \kappa_A \\
\mathcal{I}[\![\textbf{back}]\!]\ \rho_P\ \rho_V\ (\kappa_E, \kappa_A, \kappa_B)\ \kappa_B'\ \kappa &= \kappa_B \\
\mathcal{I}[\![\textbf{exit}]\!]\ \rho_P\ \rho_V\ (\kappa_E, \kappa_B, \kappa_A)\ \kappa_B'\ \kappa &= \kappa_E
\end{aligned}$$