Semantyka i weryfikacja programów 2024/25.
Egzamin 28/01/2025, zadanie 1 (semantyka)

Poniżej zaproponowana jest składnia pewnego języka programowania.

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

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:

- **exit** — zakończenie działania aktualnego wywołania procedury i skok bezpośrednio za wywołującą ją instrukcję **call**;
- **back** — zakończenie działania aktualnego wywołania procedury i skok na początek instrukcji bloku, w którym aktualnie wywołana procedura została zadeklarowana (czyli bezpośrednio za słowo **in** w tym bloku),
- **abort** — zakończenie działania bloku, w którym aktualnie wywołana procedura została wywołana instrukcją **call**, i skok bezpośrednio za ten blok.

Znaczenie instrukcji **exit**, **abort** i **back** 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 = 3;
 1:       proc P { if x >= 0 then { begin  in back end } else { exit }; x := x + 5 }
 2: in x := x - 2;
 3:    call P;
 4:    x := x + 1;
 5:    begin var y = 0;
 6:          proc Q { if x + y >= 0
 7:                   then { y := y - 1;
 8:                          begin  in call Q end;
 9:                          x := x - 1 }
10:                   else { begin  in abort end } }
11:    in begin var y = 7 in call Q;
12:                          x := x + 1 end;
13:       call P end;
14:    x := x + 7
15: end
```

0) Zadeklarowanie zmiennej x_0 o wartości 3.
1) Zadeklarowanie procedury P, wiążącej statycznie zmienną x_0.
2) Zmniejszenie wartości zmiennej x_0 do wartości 1.
3) Wywołanie procedury P.
1) Warunek jest spełniony; instrukcja **back** wykonuje skok do linii 2.
2) Zmniejszenie wartości zmiennej x_0 do wartości -1.
3) Wywołanie procedury P.
1) Warunek nie zachodzi; instrukcja **exit** wykonuje skok do linii 4.
4) Zwiększenie wartości zmiennej x_0 do wartości 0.
5) Zadeklarowanie zmiennej y_0 o wartości 0.
6) Zadeklarowanie procedury Q, wiążącej statycznie zmienne x_0 i y_0.
11) Zadeklarowanie zmiennej y_1 o wartości 7 i (pierwsze) wywołanie procedury Q.
6,7) Warunek zachodzi; wartość zmiennej y_0 zmniejsza się do -1.
8) Drugie, rekurencyjne wywołanie Q (w zagnieżdżonym bloku).
6,10) Warunek nie zachodzi; **abort** powoduje skok do linii 9 w pierwszym wywołaniu Q.
9) Zmniejszenie wartości x_0 do -1 i zakończenie pierwszego wywołania Q.
12) Zwiększenie wartości zmiennej x_0 do 0.
13) Kolejne wywołanie procedury P.
1) Warunek jest spełniony; instrukcja **back** wykonuje skok do linii 2.
2) Zmniejszenie wartości zmiennej x_0 do wartości -2.
3) Jeszcze jedno wywołanie procedury P.
1) Warunek nie zachodzi; instrukcja **exit** wykonuje skok do linii 4.
4) Zwiększenie wartości zmiennej x_0 do wartości -1.
5) Zadeklarowanie zmiennej y_2 o wartości 0.
6) Zadeklarowanie procedury Q, wiążącej statycznie zmienne x_0 i y_2.
11) Zadeklarowanie zmiennej y_3 o wartości 7 i kolejne wywołanie procedury Q.
6,10) Warunek nie zachodzi; instrukcja **abort** wykonuje skok do linii 13.
13) Kolejne wywołanie procedury P.
1) Warunek nie zachodzi; instrukcja **exit** kończy wywołanie procedury P.
14) Zwiększenie wartości zmiennej x_0 do wartości 6.

Rozwiązanie

Rozważmy następujące typy pomocnicze:

$$\begin{aligned}
\mathbf{Env} &= \mathbf{Var} \rightharpoonup \mathbf{Loc} \\
\mathbf{Store} &= \mathbf{Var} \rightharpoonup \mathbf{Num} \\
\mathbf{Cont} &= \mathbf{Store} \rightharpoonup \mathbf{Ans} \\
\mathbf{PEnv} &= \mathbf{PName} \rightharpoonup \mathbf{Proc} \\
\mathbf{Proc} &= \mathbf{Cont} \to \mathbf{Cont} \to \mathbf{Cont} \\
&\qquad \mathbf{exit} \mapsto \mathbf{abort} \mapsto \mathbf{start}
\end{aligned}$$

Używać będziemy funkcji semantycznych:

$$\begin{aligned}
\mathcal{E} &\colon \mathbf{Expr} \to \mathbf{Env} \to \mathbf{Store} \rightharpoonup \mathbf{Num} \\
\mathcal{D} &\colon \mathbf{Decl} \to \mathbf{Cont} \to (\mathbf{PEnv}, \mathbf{Env}, \mathbf{Store}) \rightharpoonup (\mathbf{PEnv}, \mathbf{Env}, \mathbf{Store}) \\
\mathcal{I} &\colon \mathbf{Inst} \to \mathbf{PEnv} \to \mathbf{Env} \to \mathbf{Cont}^3 \to \mathbf{Cont} \to \mathbf{Cont} \rightharpoonup \mathbf{Cont}
\end{aligned}$$

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

$$\mathcal{D}[\![\varepsilon]\!]\ \kappa_B\ (\rho_P, \rho_V, s) = (\rho_P, \rho_V, s)$$

$$\begin{aligned}
\mathcal{D}[\![\mathbf{var}\ x = e]\!]\ \kappa_B\ (\rho_P, \rho_V, s) = {} & \\
& \text{let } q = \mathcal{E}[\![e]\!]\ \rho_V, s \text{ in} \\
& \text{let } \ell = \text{newloc}(s) \text{ in} \\
& \text{let } \rho'_V = \rho_V[x \mapsto \ell] \text{ in} \\
& \text{let } s' = s[\ell \mapsto q] \text{ in} \\
& \quad \big(\rho_P, \rho'_V, s'\big)
\end{aligned}$$

$$\begin{aligned}
\mathcal{D}[\![\mathbf{proc}\ P\{I\}]\!]\ \kappa_B\ (\rho_P, \rho_V, s) = {} & \\
& \text{let rec } X = \lambda\ \kappa_E.\ \lambda\ \kappa_A. \\
& \quad \mathcal{I}[\![I]\!]\ \rho_P[P \mapsto X]\ \rho_V\ (\kappa_E, \kappa_B, \kappa_A)\ \kappa_A\ \kappa_E \text{ in} \\
& \big(\rho_P[P \mapsto X], \rho_V, s\big)
\end{aligned}$$

$$\mathcal{D}[\![d_1; d_2]\!]\ \kappa_B = \big(\mathcal{D}[\![d_1]\!]\ \kappa_B\big); \big(\mathcal{D}[\![d_2]\!]\ \kappa_B\big)$$

$$\begin{aligned}\mathcal{I}[\![\textbf{begin } d \textbf{ in } I \textbf{ end}]\!]\ \rho_P\ \rho_V\ &= \lambda(\kappa_E,\kappa_B,\kappa_A).\ \lambda\kappa'_A.\ \lambda\kappa.\ \lambda s.\\ &\quad \text{let rec } \big((\rho'_P,\rho'_V,s'),\kappa'_B\big) =\\ &\quad\quad \big(\mathcal{D}[\![d]\!]\ \kappa'_B\ (\rho_P,\rho_V,s),\\ &\quad\quad\quad \mathcal{I}[\![I]\!]\ \rho'_P\ \rho'_V\ (\kappa_E,\kappa_B,\kappa_A)\ \kappa\ \kappa\big) \text{ in}\\ &\quad \kappa'_B\ s'\end{aligned}$$

$$\mathcal{I}[\![\textbf{skip}]\!]\ \rho_P\ \rho_V\ \vec{\kappa}\ \kappa'_A\ \kappa = \kappa$$

$$\mathcal{I}[\![x := e]\!]\ \rho_P\ \rho_V\ \vec{\kappa}\ \kappa'_A\ \kappa = \lambda s.\ \kappa\ s[\rho_V(x) \mapsto \mathcal{E}[\![e]\!]\ \rho_V\ s]$$

$$\mathcal{I}[\![I_1; I_2]\!]\ \rho_P\ \rho_V\ \vec{\kappa}\ \kappa'_A\ \kappa = \mathcal{I}[\![I_1]\!]\ \rho_P\ \rho_V\ \vec{\kappa}\ \kappa'_A\ \big(\mathcal{I}[\![I_2]\!]\ \rho_P\ \rho_V\ \vec{\kappa}\ \kappa'_A\ \kappa\big)$$

$$\begin{aligned}\mathcal{I}[\![\textbf{if } e \texttt{ >= } 0 \textbf{ then } I_1 \textbf{ else } I_2]\!]\ \rho_P\ \rho_V\ \vec{\kappa}\ \kappa'_A\ \kappa &= \lambda s.\\ &\quad \text{ifte}\big(\mathcal{E}[\![e]\!]\ \rho_V\ s \geq 0,\\ &\quad\quad \mathcal{I}[\![I_1]\!]\ \rho_P\ \rho_V\ \vec{\kappa}\ \kappa'_A\ \kappa\ s,\\ &\quad\quad \mathcal{I}[\![I_2]\!]\ \rho_P\ \rho_V\ \vec{\kappa}\ \kappa'_A\ \kappa\ s\big)\end{aligned}$$

$$\mathcal{I}[\![\textbf{call } P]\!]\ \rho_P\ \rho_V\ (\kappa_E,\kappa_B,\kappa_A)\ \kappa'_A\ \kappa = \rho_P\ P\ \kappa\ \kappa'_A$$

$$\mathcal{I}[\![\textbf{exit}]\!]\ \rho_P\ \rho_V\ (\kappa_E,\kappa_B,\kappa_A)\ \kappa'_A\ \kappa = \kappa_E$$

$$\mathcal{I}[\![\textbf{back}]\!]\ \rho_P\ \rho_V\ (\kappa_E,\kappa_B,\kappa_A)\ \kappa'_A\ \kappa = \kappa_B$$

$$\mathcal{I}[\![\textbf{abort}]\!]\ \rho_P\ \rho_V\ (\kappa_E,\kappa_B,\kappa_A)\ \kappa'_A\ \kappa = \kappa_A$$