Zadanie 2 Poniżej zaproponowana jest składnia pewnego języka programowania.

$$\begin{aligned}
\mathbf{Var} \ni x &::= x \mid y \mid \ldots \\
\mathbf{PName} \ni P &::= P_1 \mid P_2 \mid \ldots \\
\mathbf{Num} \ni n &::= \ldots \mid -1 \mid 0 \mid 1 \mid \ldots \\
\mathbf{Expr} \ni e &::= n \mid x \mid e_1 + e_2 \mid e_1 - e_2 \mid e_1 \cdot e_2 \\
\mathbf{Inst} \ni I &::= \mathbf{skip} \mid x := e \mid I_1; I_2 \mid \mathbf{if}\ e \geq 0\ \mathbf{then}\ I_1\ \mathbf{else}\ I_2 \mid \\
&\qquad P := \mathbf{proc}(x)\{I\} \mid P := P_1\ \mathbf{then}\ P_2 \mid \mathbf{call}\ P(e)
\end{aligned}$$

Wyrażenia i instrukcje oprócz wymienionych poniżej mają standardowe znaczenie. Można założyć, że wszystkie zmienne indywiduowe x, y, ... mają początkowo określone wartości (są zainicjowane). Wiązanie zmiennych indywiduowych jest statyczne, dopuszczalne wartości tych zmiennych są typu **Num**. 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. Podobnie można przyjąć, że dostępny jest nieskończony zbiór lokacji **Loc** oraz funkcja matematyczna newloc: $(\mathbf{Loc} \rightharpoonup_{\text{fin}} \mathbf{Num}) \to \mathbf{Loc}$ spełniająca $\text{newloc}(s) \notin \text{dom}(s)$ dla każdego s: $\mathbf{Loc} \rightharpoonup_{\text{fin}} \mathbf{Num}$ (gdzie $\mathbf{Loc} \rightharpoonup_{\text{fin}} \mathbf{Num}$ to zbiór funkcji częściowych z **Loc** do **Num**, których dziedzina jest skończona).

Język jest wzbogacony o jednoparametrowe procedury. Definicja procedury następuje z użyciem instrukcji $P := \mathbf{proc}(x)\{I\}$, która przypisuje nazwie procedury P procedurę o ciele I z parametrem formalnym x. Zmienne indywiduowe występujące w ciele I tak zdefiniowanej procedury powinny zostać w tym momencie związane statycznie, niezależnie od kontekstu, w jakim procedura będzie później wywoływana. Zmienna x będąca parametrem formalnym procedury staje się zmienną lokalną widoczną w ciele I (i w procedurach w nim wprowadzonych). Zmienna ta przesłania wcześniej zdefiniowaną zmienną o tej samej nazwie.

Wywołanie tak zdefiniowanej procedury P instrukcją $\mathbf{call}\ P(e)$ powoduje obliczenie aktualnej wartości wyrażenia e, a następnie przypisanie tej wartości zmiennej x wprowadzonej dla tego wywołania procedury i wykonanie ciała I. Oznacza to, że przekazywanie parametrów odbywa się przez wartość.

Instrukcja przypisania procedur $P := P_1\ \mathbf{then}\ P_2$ zmienia globalną wartość procedury P w ten sposób, że staje się ona złożeniem procedur P_1 i P_2 (identyfikatory procedur P, P_1 i P_2 nie muszą być wzajemnie różne). Wywołanie tak powstałej procedury instrukcją $\mathbf{call}\ P(e)$ powoduje wywołanie po kolei procedur $P_1(e)$ i $P_2(e)$, zgodnie z ich wartościami sprzed przypisania. W szczególności wyrażenie e będzie obliczane (przynajmniej) dwukrotnie, raz na potrzeby wykonania procedury P_1, a następnie na potrzeby wykonania P_2.

Wiązanie identyfikatorów procedur jest dynamiczne; w szczególności przy wykonaniu ciała procedury może dojść do rekurencyjnego wywołania tejże procedury lub innej procedury, która ją wywoła. Początkowo wszystkie nazwy procedur są zainicjowane na procedury, które nic nie robią, czyli $P := \mathbf{proc}(x)\{\mathbf{skip}\}$.

Zadanie

Zadanie polega na napisaniu semantyki denotacyjnej 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. Należy też podać równania semantyczne dla **wszystkich** postaci instrukcji **Inst** w tym języku.

verte!

Uwaga

Przypisywanie identyfikatorów procedur w tym języku odbywa się w sposób dynamiczny. Powoduje to, że faktyczna wartość procedury zmienia się w dynamicznie, w trakcie wykonywania programu. W szczególności, każde wywołanie procedury o nazwie P powoduje jej wywołanie zgodnie z aktualną wartością tego identyfikatora. W odróżnieniu od zmiennych indywiduowych, które mają zakresy widoczności i globalny, i lokalne, zmienne procedurowe mają zawsze zakres widoczności globalny, co oznacza, że zmiana wartości identyfikatora procedury w ciele jakiejś procedury ma efekt również po wyjściu z tej ostatniej. Jest to niestandardowa cecha tego języka — w przypadku większości języków rozważanych na tym przedmiocie wiązanie procedur odbywa się w sposób statyczny, a ich wartości są niezmienne w czasie.

Przykład

Rozważmy następującą instrukcję tego języka:

```
x := 14;
y := 8;
P := proc (x) {
       R := proc (z) {
         y := y + x;
       };
       call R(0);
       if y >= 0 then
         call P(x)
       else
         skip
     };
P := P then P;
call P(y - x);
call R(0)
```

Wyliczenie tych instrukcji ustawi najpierw wartości zmiennych x na 14 i y na 8. Następnie zdefiniowana jest procedura P o parametrze formalnym (zmiennej lokalnej) x. Kolejna instrukcja `P := P then P` powoduje, że od teraz procedura P będzie dwukrotnym złożeniem oryginalnie zdefiniowanej procedury. Wywołanie `call P(y - x)` spowoduje:

1) Wyliczenie aktualnej wartości `y - x`, równej -6, na potrzeby pierwszej kopii procedury P.

1) Wewnątrz ciała procedury P zmienna lokalna x przyjmie wartość -6, procedura R zostanie zdefiniowana i nastąpi wywołanie procedury R z argumentem 0. Jej wewnętrzna instrukcja `y := y + x` zmieni wartość zmiennej globalnej y na 2 (zmienna x w ciele procedury R jest wiązana statycznie do parametru formalnego x ciała procedury P).

1) Sprawdzony zostanie warunek `y >= 0`, który jest w tym momencie prawdziwy, co doprowadzi do wywołania rekurencyjnego aktualnej wartości procedury P z argumentem `x`.

1.1) W tym celu zostanie obliczona wartość x równa -6 na potrzeby pierwszej kopii procedury P.

1.1) W jej wnętrzu znów wywołana zostanie procedura R i dojdzie do zmniejszenia wartości zmiennej globalnej y do -4.

1.1) Sprawdzony zostanie warunek `y >= 0`, który jest w tym momencie fałszywy, więc wyliczanie ciała procedury się zakończy.

1.2) Ponownie obliczymy wartość `x`, wciąż równą -6, na potrzeby drugiej kopii procedury P.

1.2) W jej wnętrzu znów wywołana zostanie procedura R i dojdzie do zmniejszenia wartości zmiennej globalnej y do -10.

1.2) Sprawdzony zostanie warunek `y >= 0`, który jest w tym momencie fałszywy, więc wyliczanie ciała procedury się zakończy.

2) Wyliczenie aktualnej wartości `y - x`, równej -24, na potrzeby drugiej kopii procedury P.

2) Wewnątrz ciała procedury P zmienna lokalna x przyjmie wartość -24 i nastąpi wywołanie procedury R, co doprowadzi do wykonania instrukcji `y := y + x`, co zmieni wartość zmiennej globalnej y na -34.

2) Sprawdzony zostanie warunek `y >= 0`, który jest w tym momencie fałszywy, więc wyliczanie ciała procedury się zakończy.

Po tym wszystkim wywołana zostanie procedura R utworzona w ostatnim wykonaniu ciała P. W ciele procedury R zmienna y wskazuje na zmienną globalną o aktualnej wartości -34, zaś zmienna x wskazuje na zmienną lokalną z ostatniego wykonania ciała procedury P (jej wartość to -24). W związku z tym, po wykonaniu instrukcji `y := y + x` wartość y spadnie do -58.

Na koniec zmienna globalna x będzie miała wartość 14 zaś zmienna globalna y będzie miała wartość -58.

Rozwiązanie

Rozważmy następujące typy pomocnicze:

$$\begin{aligned}
\mathbf{Store} &= \mathbf{Vtore} \times \mathbf{Ptore} \\
\mathbf{Vtore} &= \mathbf{Loc} \rightharpoonup_{\text{fin}} \mathbf{Num} \\
\mathbf{Ptore} &= \mathbf{PName} \rightharpoonup \mathbf{Proc} \\
\mathbf{Proc} &= (\mathbf{Vtore} \rightharpoonup \mathbf{Num}) \to \mathbf{Store} \rightharpoonup \mathbf{Store} \\
\mathbf{VEnv} &= \mathbf{Var} \rightharpoonup \mathbf{Loc}
\end{aligned}$$

Używać będziemy funkcji semantycznych:

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

oraz funkcji newloc: $\mathbf{Vtore} \to \mathbf{Loc}$. Funkcja semantyczna $\mathcal{E}$ jest dana, natomiast funkcję $\mathcal{I}$ zdefiniujemy następująco:

$$\mathcal{I}[\![\mathbf{skip}]\!]\ \rho_V\ (s_V, s_P) = (s_V, s_P)$$

$$\mathcal{I}[\![x := e]\!]\ \rho_V\ (s_V, s_P) = \text{let } q = \mathcal{E}[\![e]\!]\ \rho_V\ s_V \text{ in } \big(s_V[(\rho_V\ x) \mapsto q], s_P\big)$$

$$\mathcal{I}[\![I_1; I_2]\!]\ \rho_V\ (s_V, s_P) = \big(\mathcal{I}[\![I_1]\!]\ \rho_V\big); \big(\mathcal{I}[\![I_2]\!]\ \rho_V\big)\ (s_V, s_P)$$

$$\mathcal{I}[\![\mathbf{if}\ e \geq 0\ \mathbf{then}\ I_1\ \mathbf{else}\ I_2]\!]\ \rho_V\ (s_V, s_P) = \text{ifte}\big(\mathcal{E}[\![e]\!]\ \rho_V\ s_V \geq 0,\ \mathcal{I}[\![I_1]\!]\ \rho_V\ (s_V, s_P),\ \mathcal{I}[\![I_2]\!]\ \rho_V\ (s_V, s_P)\big)$$

$$\begin{aligned}
\mathcal{I}[\![P := \mathbf{proc}\{I\}(x)]\!]\ \rho_V\ (s_V, s_P) = \text{let } R\ E\ (s'_V, s'_P) = \\
\text{let } q = E\ s'_V \text{ in} \\
\text{let } \ell = \text{newloc}(s'_V) \text{ in} \\
\text{let } \rho'_V = \rho_V[x \mapsto \ell] \text{ in} \\
\text{let } s''_V = s'_V[\ell \mapsto q] \text{ in} \\
\mathcal{I}[\![I]\!]\ \rho'_V\ (s''_V, s'_P) \\
\text{in } \big(s_V, s_P[P \mapsto R]\big)
\end{aligned}$$

$$\begin{aligned}
\mathcal{I}[\![P := P_1\ \mathbf{then}\ P_2]\!]\ \rho_V\ (s_V, s_P) = \text{let } R_1 = s_P\ P_1 \text{ in} \\
\text{let } R_2 = s_P\ P_2 \text{ in} \\
\text{let } R\ E = (R_1\ E); (R_2\ E) \text{ in} \\
\big(s_V, s_P[P \mapsto R]\big)
\end{aligned}$$

$$\mathcal{I}[\![\mathbf{call}\ P(e)]\!]\ \rho_V\ (s_V, s_P) = \text{let } R = s_P\ P \text{ in } R\ (\mathcal{E}[\![e]\!]\ \rho_V)\ (s_V, s_P)$$