Egzamin SWP 2/02/2024, zadanie 1 (semantyka operacyjna)

Napisz semantykę operacyjną dla języka, zniechęcającego do pisania rozbudowanych warunków za pomocą instrukcji `if`, o następującej gramatyce:

$$
\begin{array}{lcl}
\mathsf{Num} \ni n & ::= & \ldots -\texttt{1} \mid \texttt{0} \mid \texttt{1} \mid \ldots \\
\mathsf{Var} \ni x & ::= & x_1 \mid x_2 \mid \ldots \\
\mathsf{VarP} \ni p & ::= & p_1 \mid p_2 \mid \ldots \\
\mathsf{Expr} \ni e & ::= & n \mid x \mid e_1 + e_2 \mid e_1 - e_2 \\
\mathsf{Dec} \ni d & ::= & \texttt{var}\ x = e \mid \texttt{proc}\ p(x)\ I \mid d_1; d_2 \\
\mathsf{Instr} \ni I & ::= & x := e \mid I_1;\ I_2 \mid \texttt{skip} \mid \texttt{if}\ e\ \texttt{<>}\ 0\ \texttt{then}\ I_1\ \texttt{else}\ I_2 \mid \\
 & & \texttt{begin}\ d; I\ \texttt{end} \mid \texttt{while}\ e\ \texttt{<>}\ 0\ \texttt{do}\ I \mid \texttt{call}\ p(e) \\
\mathsf{Prog} \ni P & ::= & \texttt{prog}\ I
\end{array}
$$

W języku tym procedury są rekurencyjne, widoczność identyfikatorów wolnych w ciele procedury jest statyczna, a parametry przekazywane są przez wartość.

Niestandardowo działa instrukcja `if`, pozostałe konstrukcje języka działają standardowo. Globalne sterowanie programu wzbogacone jest o dodatkową wartość α, która jest inicjalizowana na 0 na początku programu i zwiększana o 1 po każdym wykonaniu sprawdzenia warunku w każdym wystąpieniu instrukcji `if` (ale nie jest zwiększana przy sprawdzaniu innych warunków). Po każdym zwiększeniu α do wartości większej od 3 wszystkie zmienne widoczne w obecnym zakresie widoczności zmieniają swoją wartość aktualną. Przyjmują wtedy swoją wartość początkową powiększoną o pewną liczbę k. To k jest równe liczbie dotychczas wykonanych w programie sprawdzeń warunków w instrukcjach `if` oraz w pętlach `while` (sic!). Po sprawdzeniu warunku i ewentualnej zmianie wartości zmiennych sterowanie w instrukcji `if` przebiega standardowo, zgodnie ze sprawdzonym warunkiem. *Wartość początkowa* w przypadku zmiennych deklarowanych w deklaracjach `var` jest ich wartością przypisaną w tej deklaracji, zaś w przypadku parametrów formalnych jest wartością parametru aktualnego w chwili bieżącego wywołania procedury. Każdorazowe wykonanie ciała procedury rozpoczyna się z wartością α równą 0. Jednak przy wyjściu z procedury przywracana jest wartość α taka, jak bezpośrednio przed tym jej wywołaniem.

Należy założyć, że identyfikatory zmiennych i procedur pochodzą z rozłącznych zbiorów (tzn. $\mathsf{Var} \cap \mathsf{VarP} = \emptyset$).

W rozwiązaniu należy zdefiniować zbiór konfiguracji oraz podać reguły przejścia dla programów, instrukcji oraz deklaracji.

Można przyjąć, że dana jest pomocnicza funkcja semantyczna dla wyrażeń:

$$\mathcal{E} : \mathsf{Expr} \to \mathsf{Env} \to \mathsf{Store} \to \mathbb{Z}$$

a także funkcja alloc : $\mathsf{Store} \to \mathsf{Store} \times \mathsf{Loc}$ alokująca nieużywaną lokację.

Przykład 1

```
00: prog
01: begin
02:  var x = 3;
03:  var y = 3;
04:  if x <> 0
05:  then x := x - 1
06:  else x := x + 1;
07:  while x <> 0 do begin
08:    var z = 1;
09:    if y <> 0
10:    then y := y - z
11:    else y := y + x;
12:    x := x - 1
13:  end
14: end
```

W wyniku działania tego programu, tuż po końcu pętli, za wierszem `13` mamy `x=0` i `y=1`.

Przykład 2

```
00: prog
01: begin
02:  var x = 4;
03:  var y = 4;
04:  if x <> 0
05:  then x := x - 1
06:  else x := x + 1;
07:  while x <> 0 do begin
08:    var z = 1;
09:    if y <> 0
10:    then y := y - z
11:    else y := y + x;
12:    x := x - 1
13:  end
14: end
```

W przypadku tego programu po trzecim sprawdzeniu warunku `y <> 0` w wierszu `09` zmienne `x`, `y`, `z` przybiorą wartości, odpowiednio $4+k$, $4+k$, $1+k$, gdzie $4, 4, 1$ to początkowe wartości odpowiednich zmiennych, a $k = 7$. Spowoduje to, że do sprawdzenia warunku wejścia do pętli w wierszu `07` przystąpimy z `x=10`. Zatem po sprawdzeniu warunku `if` w wierszu `09` nastąpi kolejna zmiana wartości `x`, `y`, `z` (wartość α uległa dalszemu zwiększeniu). Analogiczna zmiana wartości zmiennych w tym miejscu będzie się powtarzać przy każdym następnym osiągnięciu wiersza `09` z coraz większymi wartościami liczby k i pętla z wiersza `07` będzie się powtarzać w nieskończoność.

Przykład 3

```
00: prog
01: begin
02:  var x = 4;
03:  var y = 4;
04:  proc p(u) begin
05:     var z = 1;
06:     if y <> 0
07:      then y := y - z
08:      else y := y + x
09:  end;
10:  if x <> 0
11:  then x := x - 1
12:  else x := x + 1;
13:  while x <> 0 do begin
14:    call p(x);
15:    x := x - 1
16:  end
17: end
```

W przypadku tego programu każde sprawdzenie warunku `if` z wiersza `06` zakończy się z wartością α nie przekraczającą 3. Zatem w programie tym tuż po końcu pętli, za wierszem `16` mamy `x=0` i `y=1`.

Przykład 4

```
00: prog
01: begin
02:  var x = 5;
03:  var y = 5;
04:  proc p(u) begin
05:    while u <> 0 do begin
06:      var z = 1;
07:      if y <> 0
08:        then y := y - z
09:        else y := y + u
10:      u := u - 1
11:    end
12:  end;
13:  if x <> 0
14:  then x := x - 1
15:  else x := x + 1;
16:  call p(x)
17: end
```

W przypadku tego programu do procedury `p` wejdziemy z parametrem aktualnym równym 4, który przypisany zostanie na parametr formalny `u`, oraz zmiennymi `x=4` i `y=5`. Analogicznie do tego, co działo się w drugim przykładzie, po czwartym sprawdzeniu warunku `y <> 0` w wierszu `07` zmienne `x`, `y`, `z`, `u` przybiorą wartości, odpowiednio $5+k$, $5+k$, $1+k$, $4+k$, gdzie $5, 5, 1, 4$ to początkowe wartości odpowiednich zmiennych, a $k = 9$. Spowoduje to, że do sprawdzenia warunku wejścia do pętli w wierszu `05` przystąpimy z `u=12`. Zatem po sprawdzeniu warunku `if` nastąpi kolejna zmiana wartości `x`, `y`, `z`, `u` (wartość α wewnątrz procedury uległa dalszemu zwiększeniu). Analogiczna zmiana wartości zmiennych w tym miejscu będzie się powtarzać przy każdym następnym osiągnięciu wiersza `07` z coraz większymi wartościami liczby k i pętla z wiersza `05` będzie się powtarzać w nieskończoność.

Przykładowe rozwiązanie: semantyka naturalna (duże kroki)

Konfiguracje końcowe: $\mathsf{T} = \mathsf{Store} \times \mathbb{Z} \times \mathbb{Z}$.
Konfiguracje: $\Gamma = \mathsf{Instr} \times \mathsf{Env} \times \mathsf{Store} \times \mathbb{Z} \times \mathbb{Z} \cup \mathsf{T}$.

Dwie ostatnie współrzędne $\mathbb{Z}$ służą do utrzymywania bieżących wartości α oraz liczby sprawdzonych warunków w instrukcjach `if` oraz `while`.

- Środowiska są postaci $\rho \in \mathsf{Env}_n = \mathsf{EnvV} \times \mathsf{EnvP}_n \times \mathsf{EnvS}$, gdzie
  - $\mathsf{EnvV} = \mathsf{Var} \rightharpoonup_{\text{fin}} \mathsf{Loc}$
  - $\mathsf{EnvP}_n = \mathsf{VarP} \rightharpoonup_{\text{fin}} \mathsf{Proc}_{n-1}$
  - $\mathsf{EnvS} = \mathsf{Var} \rightharpoonup_{\text{fin}} \mathbb{Z}$
- Dla $\rho = \langle \rho_v, \rho_p, \rho_s \rangle \in \mathsf{Env}$ przyjmujemy notacje:
  - $\rho[x \mapsto l] = \langle \rho_v[x \mapsto l], \rho_p, \rho_s \rangle$ oraz $\rho(x) = \rho_v(x)$, gdy $x \in \mathsf{Var}, l \in \mathsf{Loc}$
  - $\rho[p \mapsto P] = \langle \rho_v, \rho_p[p \mapsto P], \rho_s \rangle$ oraz $\rho(p) = \rho_p(p)$, gdy $p \in \mathsf{VarP}, P \in \mathsf{Proc}$
  - $\rho[x \mapsto n] = \langle \rho_v, \rho_p, \rho_s[x \mapsto n] \rangle$ oraz $\rho(x)^s = \rho_s(x)$, gdy $x \in \mathsf{Var}, n \in \mathbb{Z}$

  Zakładamy tutaj, że $\mathsf{Loc} \cap \mathbb{Z} = \emptyset$.
- $\mathsf{Store} = \mathsf{Loc} \rightharpoonup_{\text{fin}} \mathbb{Z}$
- $\mathsf{Proc}_n = \mathsf{Var} \times \mathsf{Env}_{n-1} \times \mathsf{Instr}$
- $\mathsf{EnvP}_0 = \{\emptyset\}$
- $\mathsf{Env} = \bigcup_{i \in \mathbb{N}} \mathsf{Env}_i$
- $\mathsf{Proc} = \bigcup_{i \in \mathbb{N} - \{0\}} \mathsf{Proc}_i$

Niestandardowe środowisko EnvS przechowuje informacje o wartościach początkowych zmiennych.
Relacja "dojścia" ("dużych kroków") dla programów:

$$\rightsquigarrow_p \subseteq \mathsf{Prog} \times \mathsf{T}$$

Relacja "dojścia" ("dużych kroków") dla instrukcji:

$$\rightsquigarrow \subseteq (\mathsf{Instr} \times \mathsf{Env} \times \mathsf{Store} \times \mathbb{Z} \times \mathbb{Z}) \times \mathsf{T}$$

Relacja "dojścia" ("dużych kroków") dla deklaracji:

$$\rightsquigarrow_d \subseteq (\mathsf{Dec} \times \mathsf{Env} \times \mathsf{Store}) \times \mathsf{Env} \times \mathsf{Store}$$

Relacja uaktualniania składu:

$$\rightsquigarrow_u \subseteq (\mathsf{Store} \times \mathsf{EnvV} \times \mathsf{EnvS} \times \mathbb{Z}) \times \mathsf{Store}$$

Definicja relacji uaktualniania składu:

$$\frac{}{s, \emptyset, \rho_s, k \rightsquigarrow_u s} \qquad \frac{\langle x, l\rangle \in \rho_v \quad \rho'_v = \rho_v - \{\langle x, l\rangle\} \quad s[\rho_v(x) \mapsto \rho_s(x) + k], \rho'_v, \rho_s, k \rightsquigarrow_u s'}{s, \rho_v, \rho_s, k \rightsquigarrow_u s'}$$

Reguły semantyczne:

Reguły dla programu:

$$\frac{I, \emptyset, \emptyset, 0, 0 \rightsquigarrow s, n, k}{\texttt{prog}\ I \rightsquigarrow s}$$

Standardowe reguły dla zwykłych obliczeń:

$$\frac{}{x{:=}e, \rho, s, n, k \rightsquigarrow s[\rho(x) \mapsto \mathcal{E}[\![e]\!]\,\rho\,s], n, k} \qquad \frac{}{\texttt{skip}, \rho, s, n, k \rightsquigarrow s, n, k}$$

$$\frac{I_1, \rho, s, n, k \rightsquigarrow s', n', k' \quad I_2, \rho, s', n', k' \rightsquigarrow s'', n'', k''}{I_1; I_2, \rho, s, n, k \rightsquigarrow s'', n'', k''}$$

$$\frac{d, \rho, s \rightsquigarrow_d \rho', s' \quad I, \rho', s', n, k \rightsquigarrow s'', n'', k''}{\texttt{begin}\ d; I\ \texttt{end}, \rho, s, n, k \rightsquigarrow s'', n'', k''}$$

$$\frac{\begin{array}{c}\mathcal{E}[\![e]\!]\,\rho\,s = v \quad \rho(p) = \langle x, \rho', I\rangle \quad \mathrm{alloc}(s) = \langle s', l\rangle \\ I, \rho'[x \mapsto l][p \mapsto \rho(p)], s'[l \mapsto v], 0, k \rightsquigarrow s'', n'', k''\end{array}}{\texttt{call}\ p(e), \rho, s, n, k \rightsquigarrow s'', n, k''}$$

Reguły dla while:

$$\frac{\mathcal{E}[\![e]\!]\,\rho\,s = 0}{\texttt{while}\ e\ \texttt{<>}\ 0\ \texttt{do}\ I, \rho, s, n, k \rightsquigarrow s, n, k+1}$$

$$\frac{\begin{array}{c}\mathcal{E}[\![e]\!]\,\rho\,s \neq 0 \quad I, \rho, s, n, k+1 \rightsquigarrow s', n', k' \\ \texttt{while}\ e\ \texttt{<>}\ 0\ \texttt{do}\ I, \rho, s', n', k' \rightsquigarrow s'', n'', k''\end{array}}{\texttt{while}\ e\ \texttt{<>}\ 0\ \texttt{do}\ I, \rho, s, n, k \rightsquigarrow s'', n'', k''}$$

Reguły dla if:

$$\frac{I_1, \rho, s, n+1, k+1 \rightsquigarrow s', n', k' \quad n+1 \leq 3 \quad \mathcal{E}[\![e]\!]\,\rho\,s = 0}{\texttt{if}\ e\ \texttt{<>}\ 0\ \texttt{then}\ I_1\ \texttt{else}\ I_2, \rho, s, n, k \rightsquigarrow s', n', k'}$$

$$\frac{I_2, \rho, s, n+1, k+1 \rightsquigarrow s', n', k' \quad n+1 \leq 3 \quad \mathcal{E}[\![e]\!]\,\rho\,s \neq 0}{\texttt{if}\ e\ \texttt{<>}\ 0\ \texttt{then}\ I_1\ \texttt{else}\ I_2, \rho, s, n, k \rightsquigarrow s', n', k'}$$

$$\frac{\begin{array}{c}I_1, \rho, s'', n+1, k+1 \rightsquigarrow s', n', k' \quad n+1 > 3 \quad \rho = \langle \rho_v, \rho_p, \rho_s\rangle \\ s, \rho_v, \rho_s, k+1 \rightsquigarrow_u s'' \quad \mathcal{E}[\![e]\!]\,\rho\,s = 0\end{array}}{\texttt{if}\ e\ \texttt{<>}\ 0\ \texttt{then}\ I_1\ \texttt{else}\ I_2, \rho, s, n, k \rightsquigarrow s', n'}$$

$$\frac{\begin{array}{c}I_2, \rho, s'', n+1, k+1 \rightsquigarrow s', n', k' \quad n+1 > 3 \quad \rho = \langle \rho_v, \rho_p, \rho_s\rangle \\ s, \rho_v, \rho_s, k+1 \rightsquigarrow_u s'' \quad \mathcal{E}[\![e]\!]\,\rho\,s = 0\end{array}}{\texttt{if}\ e\ \texttt{<>}\ 0\ \texttt{then}\ I_1\ \texttt{else}\ I_2, \rho, s, n \rightsquigarrow s', n'}$$

Reguły dla deklaracji:

$$\frac{\mathrm{alloc}(s) = \langle s', l\rangle \quad \mathcal{E}[\![e]\!]\,\rho\, s = n}{\texttt{var}\ x = e, \rho, s \rightsquigarrow_d \rho[x \mapsto l], s'[l \mapsto n]}$$

$$\frac{}{\texttt{proc}\ p(x)\ I, \rho, s \rightsquigarrow_d \rho[p \mapsto \langle x, \rho, I\rangle], s}$$

$$\frac{d_1, \rho, s \rightsquigarrow_d \rho', s' \quad d_2, \rho', s' \rightsquigarrow_d \rho'', s''}{d_1\,;d_2, \rho, s \rightsquigarrow_d \rho'', s''}$$