Deklaracje lokalne – bloki

- Chcemy wydzielić deklaracje jako osobną kategorię syntaktyczną.

- Chcemy, aby deklaracja miała znaczenie sama w sobie, a nie tylko w powiązaniu z instrukcją.

- Rozszerzamy składnię języka o deklaracje.

Składnia

- $e ::= m \mid v \mid (e + e') \mid (e - e') \mid (e * e')$
- $b ::= t \mid b \vee b' \mid e = e' \mid \neg b$
- $d ::= \underline{\text{var}}\ v \mid d_1; d_2$
- $c ::= \underline{\text{skip}} \mid v := e \mid c; c' \mid \underline{\text{if}}\ \ b\ \underline{\text{then}}\ \ c\ \underline{\text{else}}\ \ c' \mid \underline{\text{while}}\ \ b\ \underline{\text{do}}\ \ c \mid$
  $\underline{\text{begin}}\ d; c\ \underline{\text{end}}$

Przykład 9

```
begin  var x; var y;
  x:=1;  y:=2;
  begin var x; var z;
    x:=4;
    z:=x+y;
    y:=y+1;
    print(z)
  end;
  x:=x+y;
  print(x)
end
```

Przykład 9

```
begin  var x; var y;
  x:=1;  y:=2;
  begin var x; var z;
    x:=4;
    z:=x+y;
    y:=y+1;
    print(z)
  end;
  x:=x+y;
  print(x)
end
```

Co się wydrukuje?

Przykład 9

```
begin  var x; var y;
  x:=1;  y:=2;
  begin var x; var z;
    x:=4;
    z:=x+y;
    y:=y+1;
    print(z)
  end;
  x:=x+y;
  print(x)
end
```

Co się wydrukuje?

6 4

■

Semantyka denotacyjna

Kategorie syntaktyczne

- $N = \{\ldots, -1, 0, 1, \ldots\}$
- Ide – zbiór identyfikatorów
- $B = \{tt, ff\}$
- $Aexp$ – wyrażenia arytmetyczne
- $Bexp$ – wyrażenia boolowskie
- $Decl$ – deklaracje
- Com – instrukcje

Dziedziny semantyczne

- $Loc = \{l_0, l_1, \ldots\}$ – zbiór lokacji (nieskończony).
- $Sv = N$ – zależna od języka dziedzina wartości, które mogą być pamiętane w lokacjach.
- $S = [Loc \to Sv + \{\ \underline{\text{wolna}}\ \}]$ – stany (pamięć)
- $Dv = Loc + \{\ \underline{\text{puste}}\ \}$ – zależna od języka dziedzina wartości oznaczanych identyfikatorem.
- $Env = [Ide \to Dv]$ – środowiska.

Będziemy korzystać z funkcji

$$new : S \to Loc \times S$$

dla $s \in S$

$$new(s) = (l_i, s[0/l_i]) \quad \text{jeśli } \exists j.s(l_j) = \underline{\text{wolna}} \wedge i = min\{j \mid s(l_j) = \underline{\text{wolna}}\}$$

Uwaga 3

Być może zainicjalizowanie zmiennej przez *new* jako $\perp_N$ byłoby bardziej intuicyjne. Sposób inicjalizacji nie odgrywa tu istotnej roli. ∎

Funkcje semantyczne

$$\mathcal{A} : Aexp \to [Env \to [S \to N_\perp]]$$

$$\mathcal{B} : Bexp \to [Env \to [S \to B_\perp]]$$

$$\mathcal{D} : Decl \to [Env \to [S \to Env \times S]]$$

$$\mathcal{C} : Com \to [Env \to [S \to S_\perp]]$$

Równania semantyczne

Aexp

$$\mathcal{A}[[n]] = \lambda\rho \in Env.\lambda s \in S.n$$

$$\mathcal{A}[[v]] = \lambda\rho \in Env.\lambda s \in S.\textbf{if } \rho v \notin Loc \textbf{ then } \perp_N \textbf{ else}$$
$$\quad \textbf{if } s(\rho v) = \perp_N \textbf{ then } \perp_N \textbf{ else } s(\rho v)$$

$$\mathcal{A}[[e_0 + e_1]] = \lambda\rho \in Env.\lambda s \in S.\textbf{if } \mathcal{A}[[e_0]]\rho s = \perp_N \textbf{ then } \perp_N$$
$$\quad \textbf{else if } \mathcal{A}[[e_1]]\rho s = \perp_N \textbf{ then } \perp_N \textbf{ else } (\mathcal{A}[[e_0]]\rho s + \mathcal{A}[[e_1]]\rho s)$$

$$\mathcal{A}[[e_0 - e_1]] = \lambda\rho \in Env.\lambda s \in S.\textbf{if } \mathcal{A}[[e_0]]\rho s = \perp_N \textbf{ then } \perp_N$$
$$\quad \textbf{else if } \mathcal{A}[[e_1]]\rho s = \perp_N \textbf{ then } \perp_N \textbf{ else } (\mathcal{A}[[e_0]]\rho s - \mathcal{A}[[e_1]]\rho s)$$

$$\mathcal{A}[[e_0 * e_1]] = \lambda\rho \in Env.\lambda s \in S.\textbf{if } \mathcal{A}[[e_0]]\rho s = \perp_N \textbf{ then } \perp_N$$
$$\quad \textbf{else if } \mathcal{A}[[e_1]]\rho s = \perp_N \textbf{ then } \perp_N \textbf{ else } (\mathcal{A}[[e_0]]\rho s * \mathcal{A}[[e_1]]\rho s)$$

Bexp

$$\mathcal{B}[[tt]] = \lambda\rho \in Env.\lambda s \in S.tt$$

$$\mathcal{B}[[ff]] = \lambda\rho \in Env.\lambda s \in S.ff$$

$$\mathcal{B}[[b_0 \vee b_1]] = \lambda\rho \in Env.\lambda s \in S.\textbf{if } \mathcal{B}[[b_0]]\rho s = \perp_B \textbf{ then } \perp_B$$
$$\textbf{else if } \mathcal{B}[[b_1]]\rho s = \perp_B \textbf{ then } \perp_B \textbf{ else if } \mathcal{B}[[b_0]]\rho s = tt$$
$$\textbf{then } tt \textbf{ else if } \mathcal{B}[[b_1]]\rho s = tt \textbf{ then } tt \textbf{ else } ff$$

$$\mathcal{B}[[e_0 = e_1]] = \lambda\rho \in Env.\lambda s \in S.\textbf{if } \mathcal{A}[[e_0]]\rho s = \perp_N \textbf{ then } \perp_B$$
$$\textbf{else if } \mathcal{A}[[e_1]]\rho s = \perp_N \textbf{ then } \perp_B \textbf{ else}$$
$$\textbf{if } \mathcal{A}[[e_0]]\rho s = \mathcal{A}[[e_1]]\rho s \textbf{ then } tt \textbf{ else } ff$$

$$\mathcal{B}[[\neg b]] = \lambda\rho \in Env.\lambda s \in S.\textbf{if } \mathcal{B}[[b_0]]\rho s = \perp_B \textbf{ then } \perp_B$$
$$\textbf{else if } \mathcal{B}[[b]]\rho s = tt \textbf{ then } ff \textbf{ else } tt.$$

Decl

$$\mathcal{D}[[\ \underline{\mathsf{var}}\ v]] = \lambda\rho \in Env.\lambda s \in S.(\rho[(\mathit{new}(s))_1/v], (\mathit{new}(s))_2)$$

$$\mathcal{D}[[d_1; d_2]] = \lambda\rho \in Env.\lambda s \in S.\mathcal{D}[[d_2]](\mathcal{D}[[d_1]]\rho s)_1(\mathcal{D}[[d_1]]\rho s)_2$$

Com

$$\mathcal{C}[[\underline{\text{skip}}]] = \lambda\rho \in Env.\lambda s \in S.s$$

$$\mathcal{C}[[v := e]] = \lambda\rho \in Env.\lambda s \in S.\textbf{if } \rho v \notin Loc \textbf{ then } \perp_S$$
$$\textbf{else if } \mathcal{A}[[e]]\rho s = \perp_N \textbf{ then } \perp_S \textbf{ else } s[\mathcal{A}[[e]]\rho s/\rho v]$$

$$\mathcal{C}[[c_1; c_2]] = \lambda\rho \in Env.\lambda s \in S.\textbf{if } \mathcal{C}[[c_1]]\rho s = \perp_S \textbf{ then } \perp_S$$
$$\textbf{else } \mathcal{C}[[c_2]]\rho(\mathcal{C}[[c_1]]\rho s)$$

$$\mathcal{C}[[\underline{\text{if}}\ b\ \underline{\text{then}}\ c_1\ \underline{\text{else}}\ c_2]] = \lambda\rho \in Env.\lambda s \in S.\textbf{if } \mathcal{B}[[b]]\rho s = \perp_B$$
$$\textbf{then } \perp_S \textbf{ else if } \mathcal{B}[[b]]\rho s = tt \textbf{ then } \mathcal{C}[[c_1]]\rho s \textbf{ else } \mathcal{C}[[c_2]]\rho s$$

$$\mathcal{C}[[\underline{\text{while}}\ b\ \underline{\text{do}}\ c]] = \lambda\rho \in Env.fix(f)$$

gdzie: $f = \mathcal{F}(\mathcal{B}[[b]], \mathcal{C}[[c]]) : \mathcal{D}'_{Com} \to \mathcal{D}'_{Com}$,

$\mathcal{D}'_{Com} = [S \to S_\perp]$

i $f = \lambda w \in \mathcal{D}'_{Com}.\lambda s \in S.$**if** $\mathcal{B}[[b]]\rho s = \perp_B$ **then** $\perp_S$ **else**
if $\mathcal{B}[[b]]\rho s = tt$ **then** $w\rho(\mathcal{C}[[c]]\rho s)$ **else** s

$$\mathcal{C}[[\ \underline{\text{begin}}\ d; c\ \underline{\text{end}}\ ]] = \lambda\rho \in Env.\lambda s \in S.\mathcal{C}[[c]](\mathcal{D}[[d]]\rho s)_1(\mathcal{D}[[d]]\rho s)_2$$

Semantyka aksjomatyczna

Cel:
Skonstruować regułę, która pozwoli wyprowadzić specyfikację

$$\{\alpha\}\ \underline{\text{begin}}\ d; c\ \underline{\text{end}}\ \{\beta\}$$

w zależności od własności samej instrukcji c.

Przykład 10

$$\{(x = -1) \wedge (z < 0)\}\ \underline{\text{begin}}\ \ \underline{\text{var}}\ y; y := 5\ \underline{\text{end}}\ \{x < 0\}$$

Prawdziwość tej specyfikacji wynika z prawdziwości specyfikacji

$$\{(x = -1) \wedge (z < 0)\} y := 5 \{x < 0\}$$

Ale wnioskowania tego nie możemy przenieść na analogiczną specyfikację

$$\{(x = -1) \wedge (z < 0)\} \underline{\text{begin}} \;\; \underline{\text{var}}\; x; x := 5 \;\underline{\text{end}}\; \{x < 0\}$$

bo specyfikacja

$$\{(x = -1) \wedge (z < 0)\} x := 5 \{x < 0\}$$

jest zawsze fałszywa. ∎

Gdzie leży istota problemu?

Ale wnioskowania tego nie możemy przenieść na analogiczną specyfikację

$$\{(x = -1) \wedge (z < 0)\} \ \underline{\text{begin}} \ \ \underline{\text{var}}\ x; x := 5 \ \underline{\text{end}} \ \{x < 0\}$$

bo specyfikacja

$$\{(x = -1) \wedge (z < 0)\} x := 5 \{x < 0\}$$

jest zawsze fałszywa. ∎

Gdzie leży istota problemu?

Odp: w kolizji nazw zmiennych.

Jak temu zaradzić?

Ale wnioskowania tego nie możemy przenieść na analogiczną specyfikację

$$\{(x = -1) \wedge (z < 0)\} \ \underline{\text{begin}} \ \ \underline{\text{var}}\ x; x := 5 \ \underline{\text{end}} \ \{x < 0\}$$

bo specyfikacja

$$\{(x = -1) \wedge (z < 0)\} x := 5 \{x < 0\}$$

jest zawsze fałszywa. ∎

Gdzie leży istota problemu?

Odp: w kolizji nazw zmiennych.

Jak temu zaradzić?

Odp: przemianować zmienne w bloku wewnętrznym.

Wprowadzimy następujące oznaczenia:
Niech $\overline{x} = (x_1, \ldots, x_n), \overline{y} = (y_1, \ldots, y_n)$ oznaczają wektory zmiennych. Przez $c[\overline{y}/\overline{x}]$ będziemy oznaczać instrukcję powstałą z c, w której wszystkie **wolne** w c wystąpienia x_i zastąpiono jednocześnie przez y_i, dla $i = 1, \ldots, n$.

7. **Reguła instrukcji bloku**

$$\frac{\{\alpha\}c[\overline{y}/\overline{x}]\{\beta\}}{\{\alpha\}\ \underline{\text{begin}}\ d; c\ \underline{\text{end}}\ \{\beta\}}$$

gdzie $\overline{x} = (x_1, \ldots, x_n)$ oznacza wektor zmiennych zadeklarowanych w d, a $\overline{y} = (y_1, \ldots, y_n)$ wektor dowolnych zmiennych takich, że y_i nie występuje w sposób wolny w α, β i c.

Przykład 11
Rozważmy następujące instrukcje:

$$c_1 \equiv \underline{\text{begin}}\ \ \underline{\text{var}}\ x; y := x\ \underline{\text{end}}$$

$$c_2 \equiv \underline{\text{begin}}\ \ \underline{\text{var}}\ u; z := u\ \underline{\text{end}}$$

W interpretacji, którą zdefiniowaliśmy za pomocą semantyki denotacyjnej, specyfikacja

$$\{tt\}c_1; c_2\{y = z\}$$

(rozumiana w sensie poprawności częściowej) jest oczywiście prawdziwa. Z drugiej strony, nie da się jej udowodnić w systemie H z regułą (7). Jest więc on niezupełny względem każdej klasy interpretacji zawierającej powyższą interpretację. Nie chcemy jednak "wzmacniać" systemu, ponieważ dowodzenie takich specyfikacji byłoby nieracjonalne. Niezupełność nie wynika bowiem z jego "słabości", lecz ze zbyt dokładnego określenia semantyki. ∎