SEMANTYKA I WERYFIKACJA - ĆW. 2

1. Semantyka operacyjna, wczesne wersje (lata 60te). Idea:

   - semantyka języka programowania to jego interpreter,
   - ale nie na prawdziwym komputerze, tylko na maszynie abstrakcyjnej.
   - maszyna ze stosem, stertą, kanałem wejście/wyjście itd. Przykład: wyrażenia arytmetyczne. Maszyna ze stosem i rejestrami $(x, y, z, \ldots)$ i operacjami:

     $$\begin{aligned} \texttt{Push} &: \mathit{Reg} \times \mathit{Stack} \to \mathit{Stack} \\ \texttt{Pop} &: \mathit{Stock} \to \mathit{Stack} \\ \texttt{Top} &: \mathit{Stock} \to \mathit{Reg} \end{aligned}$$

     do tego operacje arytmetyczne i przypisanie na rejestrach. Semantyką wyrażenia arytmetycznego jest ciąg operacji maszyny. Konkretnie:

     $$\begin{aligned} [\![n]\!] &= \texttt{push}(n) \\ [\![e_1 + e_2]\!] &= [\![e_1]\!];\ [\![e_2]\!];\ x := \texttt{Top};\ \texttt{Pop};\ y := \texttt{Top};\ \texttt{Pop}; z := x + y;\ \texttt{Push}(z) \end{aligned}$$

2. Semantyka operacyjna małych kroków (bardziej nowoczesna):

   - rozszerzamy składnię o termy "częściowo obliczone" (tzn. dopuszczamy zastępowanie podtermów przez wartości semantyczne). Nazywamy je **konfiguracjami**.
   - definiujemy graf skierowany na konfiguracjach (przejścia, ewaluacja)
   - niektóre konfiguracje określamy jako **końcowe**, z takich nie ma już żadnych przejść
   - Graf definiujemy przez indukcję po strukturze źródeł, na przykład:

     $$\frac{x \to x' \quad y \to y'}{f(x, y) \to g(y', x, y)}$$

3. Semantyka liczb zapisanych w formacie pozycyjnym:

   $$\frac{n \to n'}{nd \to n'd} \qquad \frac{}{\underline{n}d \to \underline{10n + d}}$$

4. Semantyka wyrażeń arytmetycznych:

   $$\frac{e \to e'}{e + f \to e' + f} \qquad \frac{f \to f'}{\underline{n} + f \to \underline{n} + f'} \qquad \frac{}{\underline{n} + \underline{m} \to \underline{n + m}}$$

5. Uwaga: w odróżnieniu od semantyki denotacyjnej, tutaj specyfikujemy kolejność wyliczania podwyrażeń. Tutaj: strategia lewostronna. Można też wybrać prawostronną, albo połączyć oba zestawy reguł. Wtedy system staje się niedeterministyczny, ale wynik obliczenia nie zależy od wybranej kolejności ewaluacji. **Zadanie:** Jak to udowodnić? *Rozwiązanie:* przez indukcję po długości (nie po budowie!) wyrażenia. Ogólniejsze rozwiązanie: Church-Rosser property albo, w tym przykładzie, nawet diamond property.

6. **Zadanie:** Narysować obliczenie (wraz z drzewami wyprowadzeń) wyrażenia $12 - 3$.

7. **Zadanie:** Uzupełnić semantykę tak, by obsłużyła dzielenie przez zero.

8. **Zadanie:** Dodać `if` do języka wyrażeń, napisać semantykę małych kroków. `If` ma sprawdzać, czy wyrażenie ewaluuje się do czegoś niezerowego.

9. **Zadanie:** Uzupełnić semantykę o zmienne i wyrażenia `let`.

   Rozwiązanie: Dodajemy *środowiska* (tj. wartościowania zmiennych) do definiowanej relacji przejścia. Środowisko to funkcja **częściowa**, żeby obsłużyć poprawnie błędne wyrażenia w rodzaju $5 + x$ (obliczenie powinno się zakleszczyć). Wychodzi zmodyfikowane stare:

   $$\frac{\rho \vdash e \to e'}{\rho \vdash e + f \to e' + f} \qquad \frac{\rho \vdash f \to f'}{\rho \vdash \underline{n} + f \to \underline{n} + f'} \qquad \frac{}{\rho \vdash \underline{n} + \underline{m} \to \underline{n+m}}$$

   i nowe rzeczy:

   $$\frac{}{\rho \vdash x \to \rho(x)} \qquad \frac{\rho \vdash e \to e'}{\rho \vdash \texttt{let } x = e \texttt{ in } f \to \texttt{let } x = e' \texttt{ in } f}$$

   $$\frac{\rho[x \mapsto n] \vdash f \to f'}{\texttt{let } x = \underline{n} \texttt{ in } f \to \texttt{let } x = \underline{n} \texttt{ in } f'} \qquad \frac{}{\rho \vdash \texttt{let } x = \underline{n} \texttt{ in } \underline{m} \to \underline{m}}$$

   (**Uwaga:** na wykładzie środowiska dodajemy do konfiguracji, tak jak stany. To co powyżej można wtedy uznać za skrót notacyjny.)

10. **Zadanie:** Narysować obliczenie (wraz z drzewami wyprowadzeń) wyrażenia

    $$\texttt{let } x = 2 + 3 \texttt{ in } x - 1$$

    w pustym (lub dowolnym) środowisku.

11. **Zadanie:** Porównać powyższe z rozwiązaniem, w którym środowisko jest traktowane jak stan, z regułami:

    $$\frac{}{\texttt{let } x = \underline{n} \texttt{ in } f,\ \rho \to f,\ \rho[x \mapsto n]} \qquad \frac{e,\ \rho \to e', \rho'}{\texttt{let } x = e \texttt{ in } f,\ \rho \to \texttt{let } x = e' \texttt{ in } f,\ \rho}$$

    Czy to się da zrobić? *Hint:* Pomyślmy o wyrażeniu

    $$\texttt{let } z = (\texttt{let } x = 2 \texttt{ in } x) \texttt{ in } x$$

    Zadanie: Narysować obliczenie dla tego wyrażenia.

12. **Zadanie:** Jak to poprawić? Rozszerzmy składnię o wyrażenie

$$e \texttt{ then } x = n$$

, które "przywraca" starą wartość zmiennej. Jest to w pewnym sensie dualne do wyrażenia `let`. Semantyka:

$$\frac{}{\texttt{let } x = n \texttt{ in } e,\ \rho \to e \texttt{ then } x = \rho(x), \rho[x \mapsto n]}$$

$$\frac{}{n \texttt{ then } x = m,\ \rho \to n,\ \rho[x \mapsto m]}$$

plus jedna reguła obliczania po lewej stronie `then`. Po prawej stronie obliczać nie trzeba, bo tam zawsze stoi liczba, nie dowolne wyrażenie.

Takie rozszerzanie składni o "robocze", nieużywane w programach wejściowych operacje to standardowe rozwiązanie.

Problem: Co z przywracaniem wartości zmiennych, które nie były wcześniej określone? *Rozwiązanie:* Można rozszerzyć dziedzinę semantyczną o dodatkową wartość $\bot$. Wtedy trzeba rozszerzyć operacje na ten dziedzinie. No i zadbać o to, żeby wyrażenie `let` $x = e_1$ `in` e_2 obliczało się tylko wtedy, kiedy e_1 nie wyewaluuje się do $\bot$.

13. **Zadanie:** Dodaj wyrażenia boolowskie z obliczaniem lewostronnym leniwym.