SEMANTYKA I WERYFIKACJA - ĆW. 8

Zaczynamy semantykę denotacyjną.

1. W semantyce denotacyjnej należy podać:
   - dziedziny syntaktyczne (zwykle po jednej dla każdego nieterminala)
   - dziedziny semantyczne (po jednej dla każdej dziedziny syntaktycznej)
   - ewentualnie jakieś dziedziny pomocnicze typu **State**
   - funkcje semantyczne z dziedzin syntaktycznych w semantyczne, zdefiniowane przez indukcję po budowie programów.

   Czasem dziedzina semantyczna jest równa syntaktycznej a funkcja semantyczna jest identycznością, jak np. w przypadku **Var**.

2. Przykład: wyrażenia arytmetyczne i typu `let`.

   Składnia: $E ::= x \mid n \mid e_1 + e_2 \mid \cdots \mid \texttt{let}\ x = e_1\ \texttt{in}\ e_2$

   Dziedziny syntaktyczne: **Var**, **Exp**

   Dziedziny pomocnicze: $\mathbf{Env} = \mathbf{Var} \to \mathbb{Q}$ (funkcje częściowe, jak poprzednio)

   Dziedziny semantyczne: $\mathbf{Var}, (\mathbf{Env} \to \mathbb{Q})$ (też funkcje częściowe, bo programy mogą się pętlić i nic nie zwracać!)

   Funkcja semantyczna: $\mathcal{E}[\![]\!] : \mathbf{Exp} \to (\mathbf{Env} \to \mathbb{Q})$

   Definicja:

$$\begin{aligned}
\mathcal{E}[\![n]\!]\rho &= n \\
\mathcal{E}[\![x]\!]\rho &= \rho(x) \\
\mathcal{E}[\![e_1 + e_2]\!]\rho &= \mathcal{E}[\![e_1]\!]\rho + \mathcal{E}[\![e_2]\!]\rho \\
\mathcal{E}[\![\texttt{let}\ x = e_1\ \texttt{in}\ e_2]\!]\rho &= \mathcal{E}[\![e_1]\!]\big(\rho[x \mapsto \mathcal{E}[\![e_2]\!]\rho]\big)
\end{aligned}$$

   Można to też zapisać lambdami.

3. Notacja pomocnicza: będziemy używać funkcji

$$if_D : \mathbf{Bool} \times D \times D \to D$$

   zdefiniowanej tak:

$$if_D(\texttt{tt}, d, e) = e \qquad\qquad if_D(\texttt{ff}, d, e) = e$$

4. **Zadanie:** Napisać semantykę denotacyjną TINY. (Było na wykładzie).

   Rozwiązanie: Naiwnie napisany `while` wygląda tak:

$$\mathcal{E}[\![\texttt{while}\ b\ \texttt{do}\ I]\!] = \lambda s.if(\mathcal{B}[\![b]\!]s, \lambda s'.\mathcal{E}[\![\texttt{while}\ b\ \texttt{do}\ I]\!](\mathcal{E}[\![I]\!]s), s)$$

(na wykładzie ta lambda w środku była oznaczana jako złożenie funkcji:

$$(\mathcal{E}[\![I]\!]; \mathcal{E}[\![\texttt{while } b \texttt{ do } I]\!])(s)$$

Ale to nie jest kompozycjonalne rozwiązanie i nie jest wcale jasne czy to równanie jednoznacznie definiuje funkcję.

Tak więc potrzebujemy funkcjonału na dziedzinie semantycznej funkcji częściowych:

$$\Phi : (\mathbf{State} \to \mathbf{State}) \to (\mathbf{State} \to \mathbf{State})$$

zdefiniowanego tak:

$$\Phi(F) = \lambda s.if(\mathcal{B}[\![b]\!]s, \lambda s'.F(\mathcal{E}[\![I]\!]s'), s)$$

i zdefiniować $\mathcal{E}[\![\texttt{while } b \texttt{ do } I]\!]$ jako jego najmniejszy punkt stały:

$$\mathcal{E}[\![\texttt{while } b \texttt{ do } I]\!] = \bigcup_{n \in \mathbb{N}} \Phi^n(\emptyset)$$

Ćwiczenie: upewnić się, że Φ jest monotoniczny, czyli ma najmniejszy punkt stały. Porządek na $\mathbf{State} \to \mathbf{State}$ jest taki, że bardziej zdefiniowana funkcja częściowa jest większa od mniej zdefiniowanej. (Porządek na **State** nie ma tu nic do rzeczy!)

5. Oblicz semantykę denotacyjną programu `while true do skip`. Przypomnienie funkcji, której punkt stały liczymy:

$$\Phi(F) = \lambda s.if(\mathcal{B}[\![b]\!]s, \lambda s'.F(\mathcal{E}[\![I]\!]s'), s)$$

Rozwiązanie: Funkcja pusta jest punktem stałym.

6. **Zadanie:** Obliczyć denotację programu

```
x := 1;
while x < 3 do
  x := x+1
```

Rozwiązanie: Trzeba po kolei wyliczyć wszystkie semantyki podprogramów. W przypadku pętli, zacząć od $\emptyset$ i wyliczać kolejne Φ^n aż się ustabilizuje. (W sumie nigdy się nie ustabilizuje! Ale najmniejszy punkt stały istnieje.)

7. **Zadanie:** To teraz zrobić inną pętlę:

$$I ::= \cdots \mid \texttt{repeat } I \texttt{ until } b$$

Rozwiązanie: $\mathcal{E}[\![\texttt{repeat } I \texttt{ until } b]\!]$ to najmniejszy punkt stały operatora:

$$\Psi(F) = \lambda s.cond(\mathcal{B}[\![b]\!], F, id)(\mathcal{E}[\![I]\!]s)$$

gdzie

$$cond(B, F, G) = \lambda s.(B(s), F(s), G(s))$$

Podpowiedź: Można najpierw pisać semantykę tak jak byśmy chcieli niekompozycjonalnie, a potem ją przerobić tworząc funkcjonał.

8. **Zadanie:** To teraz pętla `for`:

$$I ::= \cdots \mid \texttt{for}\ x := e_1\ \texttt{to}\ e_2\ \texttt{do}\ I$$

przy czym wyrażenie e_2 ma być obliczane przy każdym obrocie pętli.

Rozwiązanie: Chciałoby się napisać:

$$\mathcal{E}[\![\texttt{for}\ x := e_1\ \texttt{to}\ e_2\ \texttt{do}\ I]\!] = \mathcal{E}[\![x := e_1]\!]; \mathcal{E}[\![\texttt{while}\ x \leq e_2\ \texttt{do}\ (I; x := x{+}1)]\!]$$

Nie jest jasne czy to można to nazwać kompozycjonalną definicją, ale tak naprawdę to jest OK, bo można *zinlinować* tę pętlę `while` w bardziej matematyczną definicję. Formalnie, bierzemy funkcjonał:

$$\Phi(F) = \lambda s.if\big(s(x) \leq \mathcal{E}[\![e_2]\!]s, F((\mathcal{E}[\![I]\!]s)[x \mapsto s(x) + 1]), s\big)$$

i piszemy:

$$\mathcal{E}[\![\texttt{for}\ x := e_1\ \texttt{to}\ e_2\ \texttt{do}\ I]\!] = \lambda s.(fixF)(s[x \mapsto \mathcal{E}[\![e_1]\!]s])$$

9. **Zadanie:** To teraz przeróbmy pętlę `for` tak, żeby wyrażenie e_2 obliczało się tylko na początku.

Rozwiązanie: Dla każdej liczby n można zrobić oddzielny funkcjonał Φ_n:

$$\Phi_n(F) = \lambda s.if\big(s(x) \leq n, F((\mathcal{E}[\![I]\!]s)[x \mapsto s(x) + 1]), s\big)$$

(por. poprzednie zadanie) i użyć go tak:

$$\mathcal{E}[\![\texttt{for}\ x := e_1\ \texttt{to}\ e_2\ \texttt{do}\ I]\!] = \lambda s.(fixF_{\mathcal{E}[\![e_2]\!]s})(s[x \mapsto \mathcal{E}[\![e_1]\!]s])$$

10. **Zadanie:** Przeróbmy teraz funkcję tak, żeby liczba iteracji ustalała się od razu na początku, a zmiany wartości x w ciele I nie wpływały na zakończenie pętli.

Rozwiązanie: Tutaj w ogóle jest prościej, bo nie musimy jawnie konstruować punktów stałych. Semantyka idzie tak:

$$\mathcal{E}[\![\texttt{for}\ x := e_1\ \texttt{to}\ e_2\ \texttt{do}\ I]\!] = \lambda s.if(\mathcal{E}[\![e_1]\!]s \leq \mathcal{E}[\![e_2]\!]s, \mathcal{E}[\![I]\!]^n, \downarrow)$$

gdzie $\downarrow$ oznacza że funkcja jest niezdefiniowana, a $n = \mathcal{E}[\![e_2]\!]s - \mathcal{E}[\![e_1]\!]s$.

Ogólnie możemy stosować dowolną metanotację matematyczną, ale musimy uważać, żeby na pewno nasze równania jednoznacznie wyznaczały funkcję o odpowiednim typie.