SEMANTYKA I WERYFIKACJA - ĆW. 7

Dalszy ciąg semantyki naturalnej.

1. **Tyko w pierwszej grupie**: wady rozwiązania lokalnych deklaracji bez rozbicia na stany i środowiska:
   - nieelegancka semantyka małych kroków
   - inne mechanizmy robią się nieeleganckie. Na przykład przy wyjątkach, trzeba pracowicie zmieniać propagowany stan przy podnoszeniu wyjątków przez bloki. W rozwiązaniu z środowiskiem, podnosimy po prostu store a właściwe środowisko czeka w miejscu przechwycenia wyjątku.
   - Zadania z dzisiejszych zajęć, czyli procedury i funkcje z przekazywaniem przez zmienną, w ogóle ciężko zapisać.

2. **Tylko w drugiej grupie** Wyjątki: rozszerzmy język TINY o instrukcje:

   $$I ::= \cdots \mid \texttt{throw}\ x \mid \texttt{try}\ I_1\ \texttt{catch}\ x\ I_2$$

   gdzie x to identyfikatory wyjątków z jakiejś osobnej puli identyfikatorów **Exc**.

   Konfiguracje początkowe jak zwykle (nie trzeba nawet dodawać środowiska łapanych wyjątków!), konfiguracje końcowe to

   $$\mathbf{State} \cup \mathbf{Exc} \times \mathbf{State}$$

   Reguła dla `throw`:

   $$\frac{}{\texttt{throw}\ x,\ s \to x, s}$$

   Reguły dla `catch`:

   $$\frac{I_1, s \to s'}{\texttt{try}\ I_1\ \texttt{catch}\ x\ I_2,\ s \to s'}$$

   $$\frac{I_1, s \to x, s' \qquad I_2, s' \to \gamma}{\texttt{try}\ I_1\ \texttt{catch}\ x\ I_2,\ s \to \gamma} \qquad \frac{I_1, s \to y, s' \qquad y \neq x}{\texttt{try}\ I_1\ \texttt{catch}\ x\ I_2,\ s \to y, s'}$$

   (γ powyżej przebiega po wszystkich konfiguracjach; zauważmy jak stan s' nie jest gubiony przy podniesieniu wyjątku, w odróżnieniu od `fail` z poprzedniego zadania).

   Do tego reguły dla propagacji wyjątków (jedna już jest powyżej po prawej).

3. Rozszerzmy język TINY o deklaracje i wywołania procedur z jednym parametrem:

   $$I ::= \cdots \mid \texttt{begin}\ d;\ I\ \texttt{end} \mid \texttt{call}\ x_1(x_2)$$
   $$d ::= \texttt{var}\ x = e \mid \texttt{proc}\ x_1(x_2)\ I \mid d_1; d_2$$

   Zróbmy semantykę naturalną tak, by uzyskać:

- przekazywanie parametrów przez zmienną,
- statyczną widoczność identyfikatorów i normalne własności przesłaniania,
- na razie bez rekurencji.

Rozwiązanie: Dziedziny semantyczne są jak zwykle, tzn. w rozbiciu na stan i środowisko, przy czym środowisko może mapować identyfikatory nie tylko na lokacje, ale także na procedury. Z kolei procedura to nazwa parametru formalnego, treść (instrukcja) i środowisko z punktu deklaracji:

$$\begin{aligned} s \in \mathbf{Store} &= \mathbf{Loc} \to \mathbf{Num} \\ \rho \in \mathbf{Env} &= \mathbf{Var} \to (\mathbf{Loc} \cup \mathbf{Proc}) \\ P \in \mathbf{Proc} &= \mathbf{Var} \times \mathbf{Instr} \times \mathbf{Env} \end{aligned}$$

(tutaj znowu **Env** jest zadane równaniem rekurencyjnym). Przejścia w semantyce naturalnej będą postaci:

$$\rho \vdash I, s \to s' \qquad\qquad \rho \vdash d, s \to (\rho', s')$$

Zauważmy, jak deklaracja zmienia środowisko, co niejako przeczy ideologii, którą sprzedawałem tydzień temu...

Reguły dla deklaracji zmiennych:

$$\frac{\rho \vdash e, s \to n \qquad l = \texttt{newloc}(s)}{\rho \vdash \texttt{var}\ x = e,\ s \to (\rho[x \mapsto l], s[l \mapsto n])}$$

i procedur:

$$\frac{}{\rho \vdash \texttt{proc}\ x_1(x_2)\ I,\ s \to (\rho[x_1 \mapsto (x_2, I, \rho)], s)}$$

(tutaj uzyskujemy statyczną widoczność, ale bez rekursji). Składanie deklaracji:

$$\frac{\rho \vdash d_1, s \to (\rho', s') \qquad \rho' \vdash d_2, s' \to (\rho'', s'')}{\rho \vdash d_1, d_2, s \to (\rho'', s'')}$$

Pytanko: Podać przykłady programów, w których kolejność deklaracji ma znaczenie.

Teraz reguła dla bloku:

$$\frac{\rho \vdash d, s \to (\rho', s') \qquad \rho' \vdash I, s' \to s''}{\rho \vdash \texttt{begin}\ d;\ I\ \texttt{end},\ s \to s''}$$

i dla wywołania procedury:

$$\frac{\rho(x_1) = (y, I, \rho') \qquad \rho(x_2) = l \qquad \rho'[y \mapsto l] \vdash I, s \to s'}{\rho \vdash \texttt{call}\ x_1(x_2),\ s \to s'}$$

Reszta reguł jest standardowa.

Zadanko: Sprawdzić, co się dzieje przy próbie napisania procedury rekurencyjnej.

4. Dodać do tego możliwość rekurencji.

   Rozwiązanie: Trzeba zmienić jedynie regułę dla wywołania procedur, żeby przekazać do procedury odpowiednio zmienione środowisko:

$$\frac{\rho(x_1) = (y, I, \rho') \qquad \rho(x_2) = l \qquad \rho'[y \mapsto l][x_1 \mapsto (y, l, \rho')] \vdash I, s \to s'}{\rho \vdash \texttt{call}\ x_1(x_2),\ s \to s'}$$

   Zadanko: Co się dzieje jeśli $x_1 = y$? Kolejność modyfikacji środowiska w regule powyżej ma wtedy znaczenie. Rozważmy programy:

```
begin
   var y = 7;
   proc x(x) {x := x+1};
   call x(y)
end
```

```
begin
  var y = 7;
  proc x(x) {
    if y>0 then y := y+1; call x(y)  else skip;
  };
  call x(y)
end
```

   Jak one się zachowają w dwóch wersjach tej semantyki?

5. Przeróbmy to na dynamiczne wiązanie identyfikatorów (zmiennych i procedur).

   Rozwiązanie: Wystarczy przerobić regułę dla wywoływania procedur, aby do ciała procedury przekazać inne środowisko:

$$\frac{\rho(x_1) = (y, I, \rho') \qquad \rho(x_2) = l \qquad \rho[y \mapsto l] \vdash I, s \to s'}{\rho \vdash \texttt{call}\ x_1(x_2),\ s \to s'}$$

   (ρ zamiast ρ' w drugiej przesłance). Można teraz uprościć dziedzinę **Proc**, bo nie trzeba w niej pamiętać środowiska.

   Pytanko: a jak tu dodać rekursję? *Odpowiedź:* Tu już jest rekursja, zjawiła się automatycznie wraz z dynamicznym wiązaniem!

6. To teraz wróćmy do statycznego wiązania i przeróbmy to na przekazywanie przez wartość, wraz z rozszerzeniem składni wywoływania procedur:

$$I ::= \cdots \mid \texttt{call}\ x(e)$$

   Rozwiązanie: Dziedziny semantyczne są jak poprzednio, reguły też takie same, poza regułą wołania procedur:

$$\frac{\rho(x) = (y, I, \rho') \qquad \rho \vdash e, s \to n \qquad l = \texttt{newloc}(s) \qquad \rho'[y \mapsto l] \vdash I, s[l \mapsto n] \to s'}{\rho \vdash \texttt{call}\ x_1(e),\ s \to s'}$$

7. Możliwe rozszerzenia, jeśli starczy czasu:
    - funkcje zamiast procedur (uwaga: wyrażenia z efektami ubocznymi!)
    - procedury jako parametry aktualne procedur