SEMANTYKA I WERYFIKACJA - ĆW. 6

Dalszy ciąg semantyki naturalnej.

1. Przykład rozszerzenia języka TINY o dziwną operację:

$$I ::= \cdots \mid \texttt{for}\ x = e_1\ \texttt{to}\ e_2\ \texttt{try}\ I_1\ \texttt{else}\ I_2 \mid \texttt{fail}$$

która ma działać tak:

(1) obliczamy wartość e_1 i e_2, przypisujemy e_1 na x,
(2) jeśli $x > e_2$ to wykonujemy I_2 i koniec, a w przeciwnym razie
(3) wykonujemy I_1,
(4) jeśli w I_1 nie wystąpiła instrukcja `fail`, to kończymy i przywracamy początkową wartość zmiennej x,
(5) jeśli wystąpiło `fail`, to przywracamy wartości wszystkich zmiennych, zwiększamy x o 1 i wracamy do punktu (2).

Ogólnie chodzi tu o wykonanie instrukcji I_1 dla najmniejszego parametru x (z zadanego przedziału), który nie powoduje `fail`. Ta operacja jest źle zaprojektowana. Na przykład, dlaczego w (4) przywracamy wartość zmiennej x a w (2) nie?

Nic to, zróbmy semantykę naturalną.

- Konfiguracje początkowe: **Instr** $\times$ **State**, jak zwykle
- Konfiguracje końcowe: **State** $\cup \{\texttt{fail}\}$. Zauważmy różnicę z podejściem do `break` z poprzednich zajęć: tutaj w specjalnej konfiguracji `fail` nie trzeba przechowywać stanu.

Reguły dla `for`:

$$\frac{[\![e_1]\!]s > [\![e_2]\!]s \qquad I_2, s \to s'}{\texttt{for}\ x = e_1\ \texttt{to}\ e_2\ \texttt{try}\ I_1\ \texttt{else}\ I_2,\ s \to s'}$$

(powyżej, s' powinno przebiegać także po konfiguracji `fail`),

$$\frac{[\![e_1]\!]s \leq [\![e_2]\!]s \qquad I_1, s[x \mapsto [\![e_1]\!]s] \to s'}{\texttt{for}\ x = e_1\ \texttt{to}\ e_2\ \texttt{try}\ I_1\ \texttt{else}\ I_2,\ s \to s'[x \mapsto s(x)]}$$

(zauważmy jak przywracamy stan zmiennej x), i wreszcie

$$\frac{[\![e_1]\!]s \leq [\![e_2]\!]s \qquad I_1, s[x \mapsto [\![e_1]\!]s] \to \texttt{fail} \qquad \texttt{for}\ x = e_1 + 1\ \texttt{to}\ e_2\ \texttt{try}\ I_1\ \texttt{else}\ I_2,\ s \to s'}{\texttt{for}\ x = e_1\ \texttt{to}\ e_2\ \texttt{try}\ I_1\ \texttt{else}\ I_2,\ s \to s'}$$

(zauważmy jak *nie* przywracamy stanu zmiennej x, bo może nie wolno tego robić, jeśli pętla zakończyła się wykonaniem I_2.)

Reguła dla `fail`:

$$\frac{}{\texttt{fail}, s \to\ \texttt{fail}}$$

plus reguły dla propagowania `fail`.

2. Rozszerzmy język TINY o deklaracje zmiennych lokalnych:

$$I ::= \cdots \mid \texttt{begin var } x = e;\ I \texttt{ end}$$

Trzeba tu uzyskać zwykłe reguły widoczności (nie ma różnicy między statyczną i dynamiczną) i przesłaniania.

Tu pozwolić studentom trochę poświrować.

Kluczowy pomysł: rozbicie wartościowań zmiennych na dwa aspekty:

- stan zmieniający się w czasie, jak do tej pory stany w TINY: **Store** = **Loc** $\to$ **Num**, i
- środowisko niezmienne w czasie, jak wartościowania w wyrażeniach `let`: **Env** = **Var** $\to$ **Loc**,

gdzie **Loc** to zbiór lokacji, czyli "komórek pamięci".

Konfiguracja początkowa to program wraz ze stanem i środowiskiem. Tranzycję będziemy zapisywać tak:

$$\rho \vdash I, s \to s'$$

(motywacja dla $\vdash$ jest wyraźniejsza w semantyce operacyjnej małych kroków, bo podkreśla niezmienność środowiska ρ).

Zakładamy pomocniczą funkcję `newloc` : **Store** $\to$ **Loc**, która zwraca dowolną nieużywaną lokację.

Reguła dla deklaracji (przerzućmy się na semantykę operacyjną wyrażeń):

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

Inne reguły są proste, pouczająca jest reguła dla sekwencji:

$$\frac{\rho \vdash I_1, s \to s' \qquad \rho \vdash I_2, s' \to s''}{\rho \vdash I_1; I_2, s \to s''}$$

Zauważmy jak stan zmienia się w czasie, a środowisko nie.

Zadanko: dorobić do tego dealokację niepotrzebnych lokacji.

3. Jeśli starczy czasu, 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).