SEMANTYKA I WERYFIKACJA 2021/22 – Zadanie domowe nr 1

Napisz semantykę operacyjną małych kroków dla programów i instrukcji następującego języka:

$$
\begin{array}{rcl}
\mathbb{Z} \ni n & ::= & \texttt{0} \mid \texttt{1} \mid \texttt{-1} \mid \texttt{2} \mid \texttt{-2} \mid \ldots \\
\mathit{Var} \ni x & ::= & \texttt{x} \mid \texttt{y} \mid \ldots \\
\mathit{Expr} \ni e & ::= & n \mid x \mid e_1 + e_2 \mid e_1 * e_2 \mid e_1 - e_2 \\
\mathit{BExpr} \ni b & ::= & \texttt{true} \mid \texttt{false} \mid e_1 < e_2 \mid e_1 == e_2 \mid b_1 \texttt{ and } b_2 \mid \texttt{not } b \\
\mathit{Instr} \ni I & ::= & \texttt{skip} \mid x := e \mid I_1; I_2 \mid \texttt{if } b \texttt{ then } I_1 \texttt{ else } I_2 \\
 & & \mid \texttt{while } b \texttt{ do } I \mid \texttt{atomic}\{I\} \mid \texttt{escape} \\
\mathit{Prog} \ni P & ::= & I_1 \parallel I_2 \parallel \ldots \parallel I_n \text{ dla } n \geq 1
\end{array}
$$

Wyrażenia arytmetyczne i logiczne mają standardowe znaczenie, dane przez funkcje

$$\mathcal{E}[\![\_]\!] : \mathit{Expr} \to \mathbf{State} \to \mathbb{Z} \qquad \text{oraz} \qquad \mathcal{B}[\![\_]\!] : \mathit{BExpr} \to \mathbf{State} \to \{\texttt{tt}, \texttt{ff}\}$$

gdzie $\mathbf{State} = \mathit{Var} \to \mathbb{Z}$. Semantykę instrukcji należy zdefiniować jako relację przejścia w postaci dogodnej dla wykorzystania w semantyce programów. W razie potrzeby można rozszerzyć składnię programów lub instrukcji.

Program reprezentowany jest jako lista wątków. Każdy wątek to jakaś instrukcja. Program kończy działanie, gdy wszystkie jego wątki zakończą swoje działanie. Wątki współdzielą stan (zmiana wartości zmiennych w jednym wątku jest widoczna w pozostałych) i są wykonywane 'współbieżnie' – 'przeplotowo', jak opisano niżej.

Znaczenie instrukcji innych niż `atomic` i `escape` jest standardowe. Wykonanie instrukcji składa się z *elementarnych kroków* obliczeń, którymi są w szczególności instrukcje przypisania i sprawdzanie warunków w `if` i `while`. Elementarne kroki różnych wątków mogą się ze sobą przeplatać.

Instrukcja przypisania $x := e$ jest wykonywana 'atomowo' (jako krok elementarny) wraz z obliczeniem wartości e — inne wątki nie mogą zmienić stanu, w którym rozpoczęto jej wykonanie i w którym obliczana jest wartość wyrażenia, aż do zakończenia realizacji tej instrukcji.

W instrukcjach `if` oraz `while` sprawdzenie warunku jest krokiem elementarnym (czyli odbywa się 'atomowo'). Po obliczeniu warunku wybierana jest odpowiednia gałąź (w przypadku `if`) lub to, czy pętla ma być kontynuowana (`while`). Instrukcje znajdujące się w ciele gałęzi lub pętli mogą być już dowolnie przeplatane z instrukcjami innych wątków.

Blok `atomic` wymusza wykonanie całej zawartej w nim instrukcji w sposób 'atomowy': gdy w jakimś wątku wykonywane są instrukcje wewnątrz bloku `atomic`, instrukcje z innych wątków nie mogą być z nimi przeplatane. Bloki `atomic` mogą być zagnieżdżone. W takiej sytuacji dopiero zakończenie 'ostatniego' (tj. najbardziej zewnętrznego) bloku `atomic` pozwala na wznowienie pozostałych wątków.

Instrukcja `escape` kończy działanie aktualnego bloku `atomic` — po wykonaniu tej instrukcji dany wątek powinien kontynuować wykonanie od następnej instrukcji za tym blokiem (jeśli taka istnieje, w przeciwnym razie dany wątek kończy działanie). Jeśli bloki `atomic` są zagnieżdżone, to `escape` kończy działanie tylko najbardziej wewnętrznego bloku, w którym występuje. Wywołanie `escape` poza jakimkolwiek blokiem `atomic` działa jak `skip`.

Wymagamy, aby semantyka programów prawidłowo realizowała wyżej opisany sposób przeplatania operacji — poprawne rozwiązanie musi pozwalać na wszystkie zgodne z tym opisem przeploty. W szczególności nie można po prostu tylko wykonać wątków jeden po drugim.

Kilka przykładów

Program $x := 1 \parallel x := 2 \parallel x := 3$ może wykonać trzy instrukcje przypisania w dowolnej kolejności, tak że w stanie końcowym x może mieć dowolną wartość ze zbioru $\{1, 2, 3\}$.

Program $(x := 0; x := x + 1) \parallel x := 5$ skończy działanie w stanie, w którym x ma dowolną wartość ze zbioru $\{5, 6, 1\}$, bo umożliwia nastepujace przeploty:

- $x := 0; x := x + 1; x := 5$
- $x := 0; x := 5; x := x + 1$
- $x := 5; x := 0; x := x + 1$

Program $(x := 0; \texttt{if } x == 0 \texttt{ then } x := x + 1 \texttt{ else } x := x + 2) \parallel x := 10$ może zakończyć działanie w stanie, w którym x ma dowolną wartość spośród 1, 10, 11 lub 12, gdyż możliwe są następujące przeploty:

- Najpierw $x := 10$, potem cały pierwszy wątek (w stanie końcowym wartość x to 1).
- Najpierw $x := 0$, potem $x := 10$, i dalej `if` (w stanie końcowym wartość x to 12).
- Najpierw $x := 0$, potem sprawdzenie warunku w instrukcji `if`, ale bezpośrednio po tym sprawdzeniu, a przed wykonaniem instrukcji z wybranej gałęzi wykonywane jest $x := 10$, a dopiero potem gałąź $x := x + 1$ (w stanie końcowym wartość x to 11).
- Najpierw cały pierwszy wątek, potem drugi wątek (w stanie końcowym wartość x to 10).

Gdy instrukcję warunkową obejmiemy blokiem `atomic`:

$$(x := 0; \texttt{ atomic}\{\texttt{if } x == 0 \texttt{ then } x := x + 1 \texttt{ else } x := x + 2\}) \parallel x := 10$$

to możliwe wartości x w stanie końcowym zawęzimy do zbioru $\{1, 10, 12\}$, bo przypisanie $x := 10$ z drugiego wątku nie może już 'wpleść' się pomiędzy sprawdzenie warunku a wykonanie odpowiedniej gałęzi `if`.

Program $(x := 0; y := 1; \texttt{while } y == 1 \texttt{ do } x := x + 1) \parallel y := 0$ może się zapętlić, gdy drugi wątek zostanie 'zagłodzony' przez nieskończone wykonanie pętli pierwszego wątku (instrukcja drugiego wątku nie zostanie 'wpleciona' w to nieskończone wykonanie), a może się zakończyć (w stanie, w którym wartość x jest dowolną liczbą naturalną), gdy drugi wątek wplecie się przed sprawdzenie warunku pętli (po dowolnej liczbie wykonań jej ciała).

Program $\texttt{atomic}\{x := 1; \texttt{escape}; x := 2\}$ skończy działanie w stanie, gdzie wartość x to 1, gdyż `escape` przerwie dalsze wykonanie bloku.

Program

$$\texttt{atomic}\{\ \texttt{atomic}\{x := 1; \texttt{escape}; x := 2\};\ x := x + 10\}$$

skończy skończy działanie w stanie, gdzie wartość x to 11, gdyż `escape` przerwie wykonanie wewnętrznego bloku, ale działanie zewnętrznego bloku będzie kontynuowane.

Przykładowe rozwiązanie

Semantykę wyrażeń mamy daną jak w treści zadania.

Rozszerzamy składnię programu: wątek może być instrukcją I_i (jak w treści zadania) lub specjalnym znacznikiem `finished`, oznaczającym, że dany wątek zakończył już działanie[1].

Nie-końcowe konfiguracje programu rozszerzamy o dodatkowy znacznik rezerwacji $rs \in RS = \mathbb{N} \cup \{\texttt{all}\}$. Znacznik `all` mówi, że można wykonywać dowolny wątek, zaś liczba i w tym miejscu oznacza, że można wykonywać tylko obliczenia i-tego wątku aż do zwolnienia przez niego blokady.

Przejścia programu mają więc postać: $\langle P, s, rs\rangle \Rightarrow \langle P', s', rs'\rangle$ lub $\langle P, s, rs\rangle \Rightarrow s'$, gdzie $s, s' \in \mathbf{State}$ i $P, P' \in \mathit{Prog}$. Ta druga forma oznacza zakończenie wykonywania całego programu – wszystkie wątki się zakończyły. Ewaluację programu P zaczynamy od konfiguracji ze znacznikiem `all`. Wykonanie programu jest reprezentowane przez kroki w relacji

$$\Rightarrow \subseteq (\mathit{Prog} \times \mathbf{State} \times RS) \times ((\mathit{Prog} \times \mathbf{State} \times RS) \cup \mathbf{State})$$

gdzie konfiguracje końcowe to **State**. Zatem konfiguracja początkowa dla programu P to $\langle P, \emptyset_{Var\to\mathbb{Z}}, \texttt{all}\rangle$.

Dodatkowo definiujemy pomocniczą relację $\longrightarrow$ opisującą wykonanie instrukcji. Relacja ta prowadzi od instrukcji i pewnego stanu do pozostałej do wykonania instrukcji (jeśli nie nastąpił koniec wykonania), nowego stanu oraz dodatkowego znacznika $es \in ES = \{\bot, \texttt{escaping}, \texttt{reserved}\}$. Znacznik `reserved` informuje, że jesteśmy w trakcie wykonywania jakiegoś bloku `atomic`, a znacznik `escaping`, że nastąpiło 'przerwanie' bloku za pomocą `escape`. Zatem relacja ma postać:

$$\longrightarrow \subseteq (\mathit{Instr} \times \mathbf{State}) \times ((\mathit{Instr} \times \mathbf{State} \times ES) \cup (\mathbf{State} \times ES))$$

Czyli 'przejścia' dla instrukcji mają postać $\langle I, s\rangle \longrightarrow \langle I', s', es\rangle$ lub $\langle I, s\rangle \longrightarrow \langle s', es\rangle$. Tylko z pozoru może wydawać się problematyczne, że nie opisujemy tutaj niezależnej semantyki operacyjnej instrukcji (przejść relacji $\longrightarrow$ nie można składać po sobie, bo 'kształt' wyjścia nie zgadza się z 'kształtem' wejścia – różnią się o dodatkowy znacznik rezerwacji). Opisujemy bowiem obliczenia programów, których kroki definiuje relacja $\Rightarrow$, zaś $\longrightarrow$ jest tylko pomocniczą konstrukcją. Relacja $\Rightarrow$ będzie zdefiniowana korzystając z $\longrightarrow$ w powyższej postaci.

Instrukcja `atomic` normalnie wykonuje swoje kroki, ale po każdym kroku, gdy jeszcze jest coś do zrobienia, ustawia status `reserved`. Reguły przejścia dla programów zapewnią, że do czasu zwolnienia blokady jedynie ten wątek będzie wykonywany. W przypadku, gdy instrukcja wewnątrz `atomic` ustawi status `escaping`, blok kończy wywołanie, nawet jeśli pozostały w nim jeszcze jakieś instrukcje. Zagnieżdżanie bloków działa trywialnie: blok po prostu nadaje status `reserved` (niezależnie od tego, że ten status mógł już być ustawiony przez wewnętrzny blok). Zatem status `reserved` jest ustawiany zawsze, gdy ewaluacja postępuje wewnątrz co najmniej jednego bloku `atomic`.

$$\frac{\langle I, s\rangle \longrightarrow \langle I', s', es\rangle,\ es \neq \texttt{escaping}}{\langle \texttt{atomic}\{I\}, s\rangle \longrightarrow \langle \texttt{atomic}\{I'\}, s', \texttt{reserved}\rangle}$$

[1] Znacznik ten trzymamy po to, aby lista wątków się nigdy nie kurczyła. Wtedy dany wątek możemy zawsze identyfikować po numerze jego pozycji na tej liście. Oczywiście równie dobrze można jakoś inaczej przypisać wątkom unikalne nazwy i wtedy pozwolić na kurczenie listy. Można też pozwolić sobie na to, by w trakcie programu numer wątku zmieniał się (gdy inny 'zniknie') – tylko wtedy trzeba uważać, żeby to nic nie psuło, a jak zapewniamy, że numery są stałe, to nie trzeba się tym w ogóle przejmować.

$$\frac{\langle I, s\rangle \longrightarrow \langle s', es\rangle}{\langle \texttt{atomic}\{I\}, s\rangle \longrightarrow \langle s', \bot\rangle}$$

Instrukcja `escape` nadaje status `escaping`, który jest wykrywany przez `atomic`. Po złapaniu 'przerwania' przez `atomic`, dany blok nie jest już dalej wykonywany, ale przerwanie nie jest propagowane dalej – bo `escape` kończy realizację tylko jednego bloku `atomic`.

$$\frac{}{\langle \texttt{escape}, s\rangle \longrightarrow \langle s, \texttt{escaping}\rangle}$$

$$\frac{\langle I, s\rangle \longrightarrow \langle I', s', \texttt{escaping}\rangle}{\langle \texttt{atomic}\{I\}, s\rangle \longrightarrow \langle s', \bot\rangle}$$

Status `escaping` nie wpływa na wykonanie innych instrukcji i programów, zatem nadanie go poza blokiem `atomic` nie ma znaczenia – `escape` poza blokiem `atomic` działa więc jak `skip`.

Program ma dwa tryby ewaluacji: tryb `all`, gdzie wszystkie wątki mogą się przeplatać, i tryb 'zarezerwowany', gdzie tylko jeden wątek ma prawo kontynuować.

W trybie `all` może być realizowany dowolny z wątków. Jeśli taki wątek zaczął wykonywać swój blok `atomic`, zgłasza on 'rezerwację' i jest ona zapisywana w konfiguracji. Aż do momentu, gdy po wykonaniu danej operacji wątek ten przestanie zgłaszać rezerwację (co będzie oznaczało opuszczenie bloku `atomic`), tylko ten wątek będzie mógł być wykonywany.

$$\frac{\text{dla dowolnego } i \in \{1, ..., n\} \qquad \langle I_i, s\rangle \longrightarrow \langle I_i', s', es\rangle}{\langle I_1 \parallel ... \parallel I_n, s, \texttt{all}\rangle \Rightarrow \langle I_1 \parallel ... \parallel I_{i-1} \parallel I_i' \parallel I_{i+1} \parallel ... \parallel I_n, s', rs\rangle} \text{ gdzie } rs = \begin{cases} i, & \text{jeśli } es = \texttt{reserved} \\ \texttt{all}, & \text{w p.p.} \end{cases}$$

W trybie rezerwacji tylko wskazany wątek może być wykonywany. Jeśli wykonanie bloku `atomic` trwa w nim nadal, to nadal zgłasza on rezerwację i jest ona utrzymywana.

$$\frac{\langle I_i, s\rangle \longrightarrow \langle I_i', s', es\rangle}{\langle I_1 \parallel ... \parallel I_n, s, i\rangle \Rightarrow \langle I_1 \parallel ... \parallel I_{i-1} \parallel I_i' \parallel I_{i+1} \parallel ... \parallel I_n, s', rs\rangle} \text{ gdzie } rs = \begin{cases} i, & \text{jeśli } es = \texttt{reserved} \\ \texttt{all}, & \text{w p.p.} \end{cases}$$

Zwróćmy uwagę, że wątek postaci $\texttt{atomic}\{I_1\}; \texttt{atomic}\{I_2\}$ pozwala na wplecenie instrukcji z innych wątków pomiędzy sąsiadujące bloki. Ostatni krok wykonania bloku `atomic` (ten, po którym osiąga stan końcowy bez instrukcji) nie zwraca już rezerwacji – zatem również krok programu 'zdejmie' rezerwację, pozwalając na wznowienie także pozostałych wątków zanim tu rozpocznie się wykonywanie drugiego bloku `atomic`.

Pozostaje opisać sytuację, gdy wykonanie danego wątku kończy się.

$$\frac{\text{dla dowolnego } i \in \{1, ..., n\} \qquad \langle I_i, s\rangle \longrightarrow \langle s', es\rangle}{\langle I_1 \parallel ... \parallel I_n, s, \texttt{all}\rangle \Rightarrow \langle I_1 \parallel ... \parallel I_{i-1} \parallel \texttt{finished} \parallel I_{i+1} \parallel ... \parallel I_n, s', \texttt{all}\rangle}$$

$$\frac{\langle I_i, s\rangle \longrightarrow \langle s', es\rangle}{\langle I_1 \parallel ... \parallel I_n, s, i\rangle \Rightarrow \langle I_1 \parallel ... \parallel I_{i-1} \parallel \texttt{finished} \parallel I_{i+1} \parallel ... \parallel I_n, s', \texttt{all}\rangle}$$

Dla kompletności reguł dla programów pozostaje dodać regułę obsługującą zakończenie wszystkich wątków.

$$\frac{}{\langle \texttt{finished} \parallel ... \parallel \texttt{finished}, s, rs\rangle \Rightarrow s}$$

Poza tym zwykłe instrukcje są standardowe, jedyna ważna zmiana to potrzeba 'propagowania' znaczników *es* przy średniku. Musimy zapewnić, że jeśli zagnieżdżona instrukcja rzuci `escaping` lub jeśli aktualnie wykonywana jest zagnieżdżona instrukcja `atomic` to jej `reserved` zostanie rozpropagowane 'w dół' aż do przejścia dla programów. Dlatego średnik propaguje *es*, który zwróciła aktualnie wykonywana zagnieżdżona instrukcja.

$$\frac{\langle I_1, s\rangle \longrightarrow \langle I_1', s', es\rangle}{\langle I_1; I_2, s\rangle \longrightarrow \langle I_1'; I_2, s', es\rangle} \qquad \frac{\langle I_1, s\rangle \longrightarrow \langle s', es\rangle}{\langle I_1; I_2, s\rangle \longrightarrow \langle I_2, s', es\rangle}$$

Pozostałe instrukcje są już całkowicie standardowe, zwracają zawsze znacznik *es* ustawiony na $\bot$. Są wypisane poniżej jedynie dla przejrzystości.

$$\frac{}{\langle \texttt{skip}, s\rangle \longrightarrow \langle s, \bot\rangle}$$

$$\frac{}{\langle x := e, s\rangle \longrightarrow \langle s[x \mapsto (\mathcal{E}[\![e]\!]s)], \bot\rangle}$$

$$\frac{\mathcal{B}[\![b]\!]s = \texttt{tt}}{\langle \texttt{if } b \texttt{ then } I_1 \texttt{ else } I_2, s\rangle \longrightarrow \langle I_1, s, \bot\rangle} \qquad \frac{\mathcal{B}[\![b]\!]s = \texttt{ff}}{\langle \texttt{if } b \texttt{ then } I_1 \texttt{ else } I_2, s\rangle \longrightarrow \langle I_2, s, \bot\rangle}$$

$$\frac{\mathcal{B}[\![b]\!]s = \texttt{tt}}{\langle \texttt{while } b \texttt{ do } I, s\rangle \longrightarrow \langle I; \texttt{while } b \texttt{ do } I, s, \bot\rangle} \qquad \frac{\mathcal{B}[\![b]\!]s = \texttt{ff}}{\langle \texttt{while } b \texttt{ do } I, s\rangle \longrightarrow \langle s, \bot\rangle}$$