Program semantics and verification 2021/22
Second exam 22/02/2022, assignment 1 (operational semantics)

Consider the extension of TINY with the following forms of statements (E ranges over arithmetic expressions, B over Boolean expressions, and x over variables):

$$S \quad ::= \quad \texttt{skip} \mid S_1;S_2 \mid \texttt{if } B \texttt{ then } S_1 \texttt{ else } S_2 \texttt{ endif} \mid \texttt{while } B \texttt{ do } S \texttt{ od} \mid \{S_1\}\{S_2\} \mid \texttt{reject} \mid \texttt{return } E$$

To determine the meaning of the new form of statement $\{S_1\}\{S_2\}$, S_1 and S_2 compete with one another. Each of them is executed separately in the initial state, and may return a value via the `return` E statement. If both of them do so, the one with a larger value "wins" and its final state is the final state of $\{S_1\}\{S_2\}$. If one of them "gives up" via `reject` statement, and the other one does not (ends with or without returning a value) then the other one "wins". If both "give up", the whole construct is equivalent to `skip`. If both of them end without returning a value and without giving up, the winner is S_2.

More precisely, $\{S_1\}\{S_2\}$ in a state s is executed as follows:

- S_1 is executed in the state s, which may end in one of the following ways:
  - i) if `return` E_1 is encountered within this execution of S_1 then we keep the current state s_1, evaluate E_1 in s_1 obtaining a result r_1, and the rest of S_1 is omitted;
  - ii) if `reject` is encountered, the rest of S_1 is omitted;
  - iii) neither `return` nor `reject` statements are encountered, and the execution of S_1 ends in a state s_1.
- S_2 is executed in the state s, which may end in one of the following ways:
  - iv) if `return` E_2 is encountered within this execution of S_2 then we keep the current state s_2, evaluate E_2 in s_2 obtaining a result r_2, and the rest of S_2 is omitted;
  - v) if `reject` is encountered, the rest of S_2 is omitted;
  - vi) neither `return` nor `reject` statements are encountered, and the execution of S_2 ends in a state s_2.

Clearly, since the executions of S_1 and S_2 proceed independently, the semantics may (but do not have to) allow them to interleave. Once the executions of S_1 and S_2 end, the execution of $\{S_1\}\{S_2\}$ ends as follows:

- if S_1 ends with i) and S_2 ends with iv) then the execution of $\{S_1\}\{S_2\}$ in s ends in the state s_1 if $r_1 \geq r_2$ and in the state s_2 if $r_1 < r_2$;
- if S_1 ends with i) and S_2 ends with either v) or vi) then the execution of $\{S_1\}\{S_2\}$ in s ends in the state s_1;
- if S_2 ends with iv) and S_1 ends with either ii) or iii) then the execution of $\{S_1\}\{S_2\}$ in s ends in the state s_2;

- if S_1 ends with ii) and S_2 ends with v) then the execution of $\{S_1\}\{S_2\}$ in s ends in the state s;
- if S_1 ends with ii) or with iii) and S_2 ends with vi) then the execution of $\{S_1\}\{S_2\}$ in s ends in the state s_2;
- if S_2 ends with v) and S_1 ends with iii) then the execution of $\{S_1\}\{S_2\}$ in s ends in the state s_1.

The "competing" statements may be nested, and then `return` and `reject` statements affect only the execution of the closest (most recent) instance of $\{S_1\}\{S_2\}$. `reject` and `return` statements outside any $\{S_1\}\{S_2\}$ block the execution.

The statements of other forms work as usual.

Examples:

1. The execution of the following statement:

   $x := 1;$
   $\{x := x + 1; x := x + 1; \texttt{return}\ x\}\{x := x + 55; \texttt{return}\ x - 55\};$
   $\{x := x + 2; \texttt{return}\ x\}\{x := x + 55; \{\texttt{return}\ x\}\{\texttt{return}\ x + 10\}\};$
   $\{x := x + 55\}\{x := x + 2\};$
   $\{x := x + 2\}\{x := x + 55; \texttt{reject}\};$
   $\{x := x + 44; \texttt{reject}\}\{x := x + 55; \texttt{reject}\}$

   ends in a state with $x = 9$.

2. The execution of the following statement:

   $x := 1;$
   $\{x := x + 2\}\{\ \texttt{while}\ x < 3\ \texttt{do}\ \{x := x + 1; \texttt{return}\ x\}\{\texttt{return}\ 3\}\ \texttt{od}\}$

   does not terminate.

Define small-step operational semantics for this new language: define the set of configurations, the set of terminal configurations and inference rules for the statements of the form $\{S_1\}\{S_2\}$, `return` E and `reject`. Indicate if and how the standard rules for the statements of other forms have to be modified (or write them out). The semantics of expressions (and Boolean expressions) may be assumed to be given.