Software Verification

(2019/2020, Operational and Denotational Semantics)

prof. Francesco Ranzato

Homework: Do all the exercises or choose one project to do and then expose.

Exercises on Operational Semantics

Exercise 1

Formally prove the statement of Exercise 2.21 (2007 book): if $\langle S_1, s\rangle \Rightarrow^k s'$ then $\langle S_1; S_2, s\rangle \Rightarrow^k \langle S_2, s'\rangle$.

Exercise 2

1. Define formally a function $lvar : \mathbf{While} \to \wp(\mathbf{Var})$ such that for any $S \in \mathbf{While}$, $lvar(S)$ is the set of variables in $\mathbf{Var}$ that appear in the left-hand side of some assignment that occurs in the statement S.
2. Prove that for any $S \in \mathbf{While}$, $s, s' \in \mathbf{State}$:

$$\text{if } \langle S, s\rangle \Rightarrow^* s' \text{ then } \forall x \notin lvar(S).\, s(x) = s'(x).$$

Exercise 3

Assume that the language $\mathbf{While}^+$ includes a new iterative command

$$\texttt{loop}(b, S)$$

where $b \in \mathbf{Bexp}$ and $S \in \mathbf{While}^+$, whose semantics should be equivalent to the program

$$S; (\texttt{while}\ b\ \texttt{do}\ S); S$$

(a) Define the small step operational semantics of $\texttt{loop}(b, S)$ without relying on the semantics of different statements.

(b) Prove that $\texttt{loop}(b, S) \cong_{\text{sos}} S; (\texttt{while}\ b\ \texttt{do}\ S); S$

Exercise 4

Prove or disprove the following semantic equivalence:

$$(\texttt{while}\ b\ \texttt{do}\ S);\ \texttt{if}\ b\ \texttt{then}\ S\ \texttt{else skip} \cong_{\text{sos}} \texttt{while}\ b\ \texttt{do}\ S$$

where $b \in \mathbf{Bexp}$, $S \in \mathbf{While}$.

Exercise 5

Consider the sublanguage $\mathbf{While}^-$ of $\mathbf{While}$ where assignments are not allowed, that is,

$$\mathbf{While}^- \ni S ::= skip \mid S_1; S_2 \mid \texttt{if}\ b\ \texttt{then}\ S_1\ \texttt{else}\ S_2 \mid \texttt{while}\ b\ \texttt{do}\ S.$$

Prove that for any $S \in \mathbf{While}^-$ and $s \in \mathbf{State}$, either $\langle S, s\rangle \Rightarrow^* s$ or the execution of $\langle S, s\rangle$ loops.

Exercise 6

Prove or disprove the following semantic equivalence:

$$\texttt{while}\ b\ \texttt{do}\ S \cong_{\text{sos}} \texttt{if}\ b\ \texttt{then}\ (\texttt{do}\ S\ \texttt{while}\ b)\ \texttt{else skip}$$

where $b \in \mathbf{Bexp}$, $S \in \mathbf{While}$.

Exercises on Denotational Semantics

Exercise 7

Prove that

$$\texttt{while}\ b\ \texttt{do}\ (S; \texttt{while}\ b\ \texttt{do}\ S) \cong_{\text{ds}} \texttt{while}\ b\ \texttt{do}\ S$$

Exercise 8

Consider

$$P_1 \equiv \texttt{while}\ b_1\ \texttt{do}\ (\texttt{if}\ b_2\ \texttt{then}\ S\ \texttt{else}\ \texttt{skip})$$

$$P_2 \equiv \texttt{while}\ (b_1 \land b_2)\ \texttt{do}\ S$$

where $b_1, b_2 \in \mathbf{Bexp}$, $S \in \mathbf{Stm}$. Prove or disprove the following statements

(a) $\mathcal{S}_{\text{ds}}[\![P_1]\!] \sqsubseteq \mathcal{S}_{\text{ds}}[\![P_2]\!]$

(b) $\mathcal{S}_{\text{ds}}[\![P_2]\!] \sqsubseteq \mathcal{S}_{\text{ds}}[\![P_1]\!]$

Exercise 9

Prove or disprove the following semantic equivalence:

$$\texttt{while}\ b\ \texttt{do}\ S \ \cong_{\text{ds}}\ \texttt{while}\ b\ \texttt{do}\ (S; \texttt{if}\ b\ \texttt{then}\ S\ \texttt{else}\ \texttt{skip})$$

where $b \in \mathbf{Bexp}$, $S \in \mathbf{Stm}$.

Exercise 10

Prove or disprove the following semantic equivalence:

$$\texttt{while}\ b\ \texttt{do}\ S \ \cong_{\text{ds}}\ \texttt{if}\ b\ \texttt{then}\ (\texttt{do}\ S\ \texttt{while}\ b)\ \texttt{else}\ \texttt{skip}$$

where $b \in \mathbf{Bexp}$, $S \in \mathbf{Stm}$.

Exercise 11

Let D be a CPO and let $f, g : D \to D$ be continuous maps. Prove that

$$\text{FIX}(f \circ g) = f(\text{FIX}(g \circ f)).$$

Exercise 12

Let $\langle D, \leq \rangle$ be a cpo. A subset $S \subseteq D$ is *convex* when

$$\forall x, y \in S, z \in D.\ (x \leq z \leq y \ \Rightarrow z \in S)$$

Let $\mathit{conv}(D) \triangleq \{S \subseteq D \mid S \text{ is convex}\}$.

1. Prove or disprove the following property:

$$S_1, S_2 \in \mathit{conv}(D) \ \Rightarrow\ S_1 \cup S_2 \in \mathit{conv}(D)$$

2. Prove or disprove the following property: $\langle \mathit{conv}(D), \subseteq \rangle$ is a cpo.

Projects

Project 13

Read, understand and then expose Chapter 4 “Provably Correct Implementation” of Nielson and Nielson’s book (2007 edition, exercises not included), where the correctness proof of Section 4.3 given w.r.t. the big step operational semantics must be replaced by a correctness proof given w.r.t. the small step operational semantics, as described by the hints in Section 4.4.

Project 14

Read, understand and then expose Section 6.1 "Environments and Stores" of Nielson and Nielson's book (2007 edition, exercises not included).

Project 15

Read, understand and then expose Section 6.2 "Continuations" of Nielson and Nielson's book (2007 edition, exercises not included).

Project 16

Let **Aexp** include integer division $a_1 \div a_2$ and expressions whose evaluations may modify the current state, such as ++x, x++, --x, x--. Modify the operational and denotational semantics of **Aexp**, **Bexp** and **Stm** accordingly.

Project 17

Design an interpreter I for the call-by-value (eager) and/or call-by-name (lazy) denotational semantics of the functional language **REC** (a small core of SML for call-by-value and Haskell for call-by-name) as defined in Chapter 9 of the book:

"The formal semantics of programming languages" by G. Winskel, The MIT Press, 1993.

This means to write a program I (in the programming language you prefer) such that (we consider the call-by-value case):

- I takes as input any declaration $d = \{f_i(x_1, ..., x_{a_i}) = t_i\}_{i=1}^n$ with terms $t_i \in$ **REC**, a term $t \in$ **REC** and (some representation of) an environment $\rho \in \mathbf{Env}_{va}$.
- It must always happen that $I(d, t, \rho) = [\![t]\!]_{va}\delta_d\rho$, where δ_d is defined by *fix*(F_d) and $F_d : \mathbf{FEnv}_{va} \to \mathbf{FEnv}_{va}$ is the functional induced by the declaration d. Thus, it is required that:
  1. if $[\![t]\!]_{va}\delta_d\rho = \bot \in \mathbf{N}_\bot$ then the interpreter $I(d, t, \rho)$ does not terminate. **Remark:** this does not mean that the execution $I(d, t, \rho)$ gives rise to a run-time error or to an exception.
  2. if $[\![t]\!]_{va}\delta_d\rho = \lfloor n \rfloor \in \mathbf{N}_\bot$ then the interpreter $I(d, t, \rho)$ terminates and outputs the integer n.
- I therefore relies on Kleene-Knaster-Tarski fixpoint iteration sequence for evaluating $I(d, t, \rho)$. This means that if $F_d : \mathbf{FEnv}_{va} \to \mathbf{FEnv}_{va}$ is the continuous functional induced by a declaration d then $I(d, t, \rho)$ must be implemented as $\sqcup_{n\geq 0} F_d^n(\bot_{\mathbf{FEnv}_{va}})$. Therefore a call $I(d, t, \rho)$ has to look for the least $k \geq 0$ such that $[\![t]\!]_{va}(F_d^k(\bot_{\mathbf{FEnv}_{va}}))\rho = \lfloor n \rfloor$ for some $\lfloor n \rfloor \in \mathbf{N}_\bot$ so that the call $I(d, t, \rho)$ outputs the integer n, whereas if this least k cannot be found, namely for all $n \geq 0$, $[\![t]\!]_{va}(F_d^n(\bot_{\mathbf{FEnv}_{va}}))\rho = \bot$, then the call $I(d, t, \rho)$ does not terminate (with no run-time error; **remark:** be careful to stack overflow and out-of-memory run-time errors).
- **REC** may contain some basic functions as syntactic sugar.

For parsing programs terms and declarations one could use parsing capabilities of programming languages (e.g., pattern matching in functional(-oriented) programming languages, see https://en.wikipedia.org/wiki/Pattern_matching) or an automatic parser generator (GNU Bison, Yacc and JavaCC are popular open source parser generators).