Semantics of Programming Languages
University of Minho

Renato Neves (nevrenato@di.uminho.pt)

Winter 2024/25

Abstract

This handout briefly summarises our previous lectures and presents related problems. It addresses the three styles of semantics that we have studied so far: namely operational, denotational, and axiomatic semantics.

1 Operational Semantics

We started our lectures with the following very simple propositional language.

$$\mathtt{b} ::= \mathtt{x} \mid \mathtt{b} \wedge \mathtt{b} \mid \neg \mathtt{b}$$

(recall that $\mathtt{x}$ denotes a variable from a given stock of variables). We established a corresponding (small-step) operational semantics, which is detailed in Figure 1.

$$\frac{}{\langle \mathtt{x}, \sigma \rangle \longrightarrow \sigma(\mathtt{x})}\ (\text{var}) \qquad \frac{\langle \mathtt{b}, \sigma \rangle \longrightarrow v}{\langle \neg \mathtt{b}, \sigma \rangle \longrightarrow \neg v}\ (\text{neg}_1)$$

$$\frac{\langle \mathtt{b}, \sigma \rangle \longrightarrow \langle \mathtt{b}', \sigma' \rangle}{\langle \neg \mathtt{b}, \sigma \rangle \longrightarrow \langle \neg \mathtt{b}', \sigma' \rangle}\ (\text{neg}_2) \qquad \frac{\langle \mathtt{b}_1, \sigma \rangle \longrightarrow \mathtt{ff}}{\langle \mathtt{b}_1 \wedge \mathtt{b}_2, \sigma \rangle \longrightarrow \mathtt{ff}}\ (\text{and}_1)$$

$$\frac{\langle \mathtt{b}_1, \sigma \rangle \longrightarrow \mathtt{tt}}{\langle \mathtt{b}_1 \wedge \mathtt{b}_2, \sigma \rangle \longrightarrow \langle \mathtt{b}_2, \sigma \rangle}\ (\text{and}_2) \qquad \frac{\langle \mathtt{b}_1, \sigma \rangle \longrightarrow \langle \mathtt{b}_1', \sigma' \rangle}{\langle \mathtt{b}_1 \wedge \mathtt{b}_2, \sigma \rangle \longrightarrow \langle \mathtt{b}_1' \wedge \mathtt{b}_2, \sigma' \rangle}\ (\text{and}_3)$$

Figure 1: Small-step operational semantics for the propositional language

Observe the 'short-circuit' evaluation mechanism present in Rule (and_1), an apparent feature in the small-step semantics that becomes somewhat invisible in all other semantics that we studied.

Problem 1. It makes sense to add an 'implication construct' $\mathtt{b} \Rightarrow \mathtt{b}$ to the simple language. Describe which semantic rules would you add to Figure 1 in order to cover this new operation.

One observation from the lectures is that our small-step operational semantics is able to describe all computational steps performed by a program until it reaches an output. The

semantics thus provides an interesting abstraction level for talking about 'program complexity'. In order to drive this point home, in the lectures we defined a 'complexity function'

$$\begin{aligned} \mathrm{compl}(\mathtt{x}) &= 1 \\ \mathrm{compl}(\neg \mathtt{b}) &= \mathrm{compl}(\mathtt{b}) \\ \mathrm{compl}(\mathtt{b}_1 \wedge \mathtt{b}_2) &= \mathrm{compl}(\mathtt{b}_1) + \mathrm{compl}(\mathtt{b}_2) \end{aligned}$$

which intends to present an upper bound to the number of required steps for a proposition to be fully evaluated. Of course we still need to formally relate it to the small-step semantics and such is the topic of the following problem.

Problem 2. Show by induction (on the syntactic structure of propositions) that $\mathrm{compl}(\mathtt{b}) \geq 1$ for every $\mathtt{b}$. Next, show by induction (over the depth of derivation trees) that the implication below holds for all propositions $\mathtt{b}$ and $\mathtt{b}'$ and any memory σ.

$$\text{If } \langle \mathtt{b}, \sigma \rangle \longrightarrow \langle \mathtt{b}', \sigma \rangle \text{ then } \mathrm{compl}(\mathtt{b}) > \mathrm{compl}(\mathtt{b}')$$

Next, recall the grammars of arithmetic expressions and while-programs that were studied throughout the lectures: they were defined as follows

$$\mathtt{e} ::= \mathtt{n} \mid \mathtt{e} \cdot \mathtt{e} \mid \mathtt{x} \mid \mathtt{e} + \mathtt{e}$$

$$\mathtt{p} ::= \mathtt{x} := \mathtt{e} \mid \mathtt{p} \,;\, \mathtt{p} \mid \mathtt{if}\ \mathtt{b}\ \mathtt{then}\ \mathtt{p}\ \mathtt{else}\ \mathtt{p} \mid \mathtt{while}\ \mathtt{b}\ \mathtt{do}\ \{\ \mathtt{p}\ \}$$

where $\mathtt{n}$ is any natural number.

Problem 3. Remarkably in the lectures we did not see a small-step operational semantics for arithmetic expressions. Can you define one such semantics now?

What we did see however was a small-step operational semantics for the while-language, which is now detailed in Figure 2.

$$\frac{\langle \mathtt{e}, \sigma \rangle \longrightarrow^\star \mathtt{v}}{\langle \mathtt{x} := \mathtt{e}, \sigma \rangle \longrightarrow \sigma[\mathtt{v}/\mathtt{x}]}\ (\mathrm{asg}) \qquad \frac{\langle \mathtt{p}, \sigma \rangle \longrightarrow \sigma'}{\langle \mathtt{p} \,;\, \mathtt{q}, \sigma \rangle \longrightarrow \langle \mathtt{q}, \sigma' \rangle}\ (\mathrm{seq}_1)$$

$$\frac{\langle \mathtt{p}, \sigma \rangle \longrightarrow \langle \mathtt{p}', \sigma' \rangle}{\langle \mathtt{p} \,;\, \mathtt{q}, \sigma \rangle \longrightarrow \langle \mathtt{p}' \,;\, \mathtt{q}, \sigma' \rangle}\ (\mathrm{seq}_2) \qquad \frac{\langle \mathtt{b}, \sigma \rangle \longrightarrow^\star \mathtt{tt}}{\langle \mathtt{if}\ \mathtt{b}\ \mathtt{then}\ \mathtt{p}\ \mathtt{else}\ \mathtt{q}, \sigma \rangle \longrightarrow \langle \mathtt{p}, \sigma \rangle}\ (\mathrm{if}_1)$$

$$\frac{\langle \mathtt{b}, \sigma \rangle \longrightarrow^\star \mathtt{ff}}{\langle \mathtt{if}\ \mathtt{b}\ \mathtt{then}\ \mathtt{p}\ \mathtt{else}\ \mathtt{q}, \sigma \rangle \longrightarrow \langle \mathtt{q}, \sigma \rangle}\ (\mathrm{if}_2) \qquad \frac{\langle \mathtt{b}, \sigma \rangle \longrightarrow^\star \mathtt{ff}}{\langle \mathtt{while}\ \mathtt{b}\ \mathtt{do}\ \{\ \mathtt{p}\ \}, \sigma \rangle \longrightarrow^\star \sigma}\ (\mathrm{wh}_2)$$

$$\frac{\langle \mathtt{b}, \sigma \rangle \longrightarrow^\star \mathtt{tt}}{\langle \mathtt{while}\ \mathtt{b}\ \mathtt{do}\ \{\ \mathtt{p}\ \}, \sigma \rangle \longrightarrow \langle \mathtt{p} \,;\, \mathtt{while}\ \mathtt{b}\ \mathtt{do}\ \{\ \mathtt{p}\ \}, \sigma \rangle}\ (\mathrm{wh}_1)$$

Figure 2: A small-step operational semantics for the while-language.

Problem 4. Thus explain the meaning of the notation $\longrightarrow^\star$.

Great, now that we have a semantics (and that we understand it) we can use it to derive the chain of transitions that is generated by a program until it reaches an output. This is particularly useful for *debugging* and for studying non-terminating behaviours, among other things.

Problem 5. Calculate via the semantics in Figure 2 the chains of transitions that arise from the following programs.

- $\mathtt{x} := 1 \,;\, \mathtt{x} := 2$
- $\mathtt{if}\ \mathtt{tt}\ \mathtt{then}\ (\mathtt{x} := 1 \,;\, \mathtt{x} := 2)\ \mathtt{else}\ \mathtt{x} := 3$

Give an example of a program that generates an *infinite* sequence of transitions. Justify your choice mathematically :-).

We now lean over our big-step operational semantics. Recall that it abstracts away from all intermediate computational steps performed in the context of the small-step semantics. It is therefore better suited than the latter for calculating outputs and for studying program equivalence (among other things). The big-step semantics of our while-language is defined in Figure 3.

$$\frac{\langle \mathtt{e}, \sigma\rangle \Downarrow \mathtt{v}}{\langle \mathtt{x} := \mathtt{e}, \sigma\rangle \Downarrow \sigma[\mathtt{v}/\mathtt{x}]}\ (\mathrm{asg}) \qquad \frac{\langle \mathtt{p}, \sigma\rangle \Downarrow \sigma' \qquad \langle \mathtt{q}, \sigma'\rangle \Downarrow \sigma''}{\langle \mathtt{p} \,;\, \mathtt{q}, \sigma\rangle \Downarrow \sigma''}\ (\mathrm{seq})$$

$$\frac{\langle \mathtt{b}, \sigma\rangle \Downarrow \mathtt{tt} \qquad \langle \mathtt{p}, \sigma\rangle \Downarrow \sigma'}{\langle \mathtt{if}\ \mathtt{b}\ \mathtt{then}\ \mathtt{p}\ \mathtt{else}\ \mathtt{q}, \sigma\rangle \Downarrow \sigma'}\ (\mathrm{if}_1) \qquad \frac{\langle \mathtt{b}, \sigma\rangle \Downarrow \mathtt{ff} \qquad \langle \mathtt{q}, \sigma\rangle \Downarrow \sigma'}{\langle \mathtt{if}\ \mathtt{b}\ \mathtt{then}\ \mathtt{p}\ \mathtt{else}\ \mathtt{q}, \sigma\rangle \Downarrow \sigma'}\ (\mathrm{if}_2)$$

$$\frac{\langle \mathtt{b}, \sigma\rangle \Downarrow \mathtt{tt} \qquad \langle \mathtt{p}, \sigma\rangle \Downarrow \sigma' \qquad \langle \mathtt{while}\ \mathtt{b}\ \mathtt{do}\ \{\ \mathtt{p}\ \}, \sigma'\rangle \Downarrow \sigma''}{\langle \mathtt{while}\ \mathtt{b}\ \mathtt{do}\ \{\ \mathtt{p}\ \}, \sigma\rangle \Downarrow \sigma''}\ (\mathrm{wh}_1)$$

$$\frac{\langle \mathtt{b}, \sigma\rangle \Downarrow \mathtt{ff}}{\langle \mathtt{while}\ \mathtt{b}\ \mathtt{do}\ \{\ \mathtt{p}\ \}, \sigma\rangle \Downarrow \sigma}\ (\mathrm{wh}_2)$$

Figure 3: A big-step operational semantics for the while-language.

Problem 6. In the lectures we did not see a big-step operational semantics for arithmetic expressions. Can you define one such semantics now?

As already mentioned, the big-step semantics differs from the small-step counterpart in that it abstracts away from all intermediate computational steps. This renders output calculation conceptually different than the one performed in Problem (5). Let us try to make such differences more evident and palpable with the following problem.

Problem 7. Calculate via the semantics in Figure 3 the outputs of the two programs below.

- $\mathtt{x} := 1 \,;\, \mathtt{x} := 2$

- `if tt then (x := 1 ; x := 2) else x := 3`

Reflect on the differences between calculating the outputs of these two programs with respect to the small-step semantics and big-step semantics. Finally check whether for every program $\mathtt{p}$ and memory σ we always have $\langle \mathtt{p}, \sigma \rangle \Downarrow \sigma'$ for some memory σ'.

Right, now that we have two different, complementary semantics for our while-language (small-step and big-step semantics) it makes sense to prove that they 'agree' with each other, more formally that the equivalence below holds

$$\langle \mathtt{p}, \sigma \rangle \longrightarrow^\star \sigma' \text{ iff } \langle \mathtt{p}, \sigma \rangle \Downarrow \sigma'$$

for every program $\mathtt{p}$ and every state σ and σ'. We will focus on proving just the left-to-right direction of this equivalence.

Problem 8. Consider programs $\mathtt{p}$ and $\mathtt{p}'$ and states $\sigma, \sigma', \sigma''$. Prove by induction (over the depth of derivation trees) the following implication.

$$\langle \mathtt{p}, \sigma \rangle \longrightarrow \langle \mathtt{p}', \sigma' \rangle \Downarrow \sigma'' \text{ implies } \langle \mathtt{p}, \sigma \rangle \Downarrow \sigma''$$

Then use this last result to prove the implication below.

$$\langle \mathtt{p}, \sigma \rangle \longrightarrow^\star \sigma' \text{ implies } \langle \mathtt{p}, \sigma \rangle \Downarrow \sigma'$$

So as mentioned before the big-step semantics provides an interesting abstraction level for reasoning about program equivalence. In this regard, we started by considering the following tentative notion: two programs $\mathtt{p}$ and $\mathtt{q}$ will be equivalent (in symbols $\mathtt{p} \equiv_o \mathtt{q}$) if the equivalence below holds.

$$\langle \mathtt{p}, \sigma \rangle \Downarrow \sigma' \text{ iff } \langle \mathtt{q}, \sigma \rangle \Downarrow \sigma' \qquad \text{(for all states } \sigma, \sigma')$$

Let us thus see if we can establish some interesting program equivalences.

Problem 9. The two equivalences propounded below are tacitly used by programmers virtually all the time (even if only a few can prove them). Use the big-step semantics to show that these equivalences are indeed correct.

- $(\mathtt{p} \,;\, \mathtt{q}) \,;\, \mathtt{r} \equiv_o \mathtt{p} \,;\, (\mathtt{q} \,;\, \mathtt{r})$
- $(\mathtt{if}\ \mathtt{b}\ \mathtt{then}\ \mathtt{p}\ \mathtt{else}\ \mathtt{q}) \,;\, \mathtt{r} \equiv_o \mathtt{if}\ \mathtt{b}\ \mathtt{then}\ \mathtt{p} \,;\, \mathtt{r}\ \mathtt{else}\ \mathtt{q} \,;\, \mathtt{r}$

Next let us recall our second (and definite) notion of program equivalence – the one that compilers typically adopt. As mentioned in lectures it is referred to as *contextual equivalence* and recurs to the following grammar of 'contexts'.

$$C ::= [-] \mid C \,;\, \mathtt{p} \mid \mathtt{p} \,;\, C \mid \mathtt{if}\ \mathtt{b}\ \mathtt{then}\ C\ \mathtt{else}\ \mathtt{p} \mid \mathtt{if}\ \mathtt{b}\ \mathtt{then}\ \mathtt{p}\ \mathtt{else}\ \mathtt{C} \mid \mathtt{while}\ \mathtt{b}\ \mathtt{do}\ \{\ C\ \}$$

Two programs $\mathtt{p}$ and $\mathtt{q}$ will be contextually equivalent (in symbols $\mathtt{p} \equiv \mathtt{q}$) if the equivalence below holds.

$$\langle C[\mathtt{p}], \sigma \rangle \Downarrow \sigma' \text{ iff } \langle C[\mathtt{q}], \sigma \rangle \Downarrow \sigma' \qquad \text{(for all states } \sigma, \sigma' \text{ and contexts } C)$$

Problem 10. Reflect on why compilers use the notion of contextual equivalence and not the previous one (*i.e.* $\equiv_o$).

Problem 11. The natural question then arises of whether the equivalence below holds

$$\mathtt{p} \equiv \mathtt{q} \text{ iff } \mathtt{p} \equiv_o \mathtt{q}$$

for all programs `p` and `q`. The left-to-right direction is clear, but what about the inverse direction? Can you prove it? Where do you get stuck?

2 Denotational Semantics

We now focus on the denotational semantics of the previous lectures. Not only it helped us with Problem (11), it also allowed us to 'import' knowledge from other mathematical theories (*e.g.* 'program calculus').

A main challenge of this semantics was the interpretation of while-loops which required us to touch (even if just barely) on the surface of Domain theory. We start from this end, in particular with the notion of a partially ordered set (poset).

Problem 12. Consider a set State of states. Show that it is a poset when equipped with equality as the partial order. Assuming that $\mathrm{P}(-)$ denotes the *powerset construct*, show that the pair $(\mathrm{P}(X), \subseteq)$ (for any given set X) is a poset.

Next recall that given any poset $(X, \leq_X)$ we were able to define a new poset $(X_\perp, \leq)$. Prove that the latter is indeed a poset. Finally given two sets X and Y prove that the set $[X, Y]$ of maps between X and Y is a poset whenever Y is a poset.

Let us now go beyond posets and shift our attention to ω-CPOs (*i.e.* ω-complete partially ordered sets). We start by revisiting the previous problem from the latter's perspective.

Problem 13. Consider a set State of states. Show that it is an ω-CPO when equipped with equality as the partial order. Show as well that the pair $(\mathrm{P}(X), \subseteq)$ (for any given set X) is an ω-CPO.

Next, recall that given any ω-CPO $(X, \leq_X)$ we were able to define a new ω-CPO $(X_\perp, \leq)$. Prove that the latter is indeed an ω-CPO. Finally given two sets X and Y prove that the set $[X, Y]$ of maps between X and Y is an ω-CPO whenever Y is an ω-CPO.

Another key notion for defining the interpretation of while-loops is that of a *continuous* map $f : X \to Y$ (between ω-CPOs X and Y). Recall that a map f is called continuous if it satisfies the conditions below.

$$\begin{aligned} x_1 \leq x_2 \text{ entails } f(x_1) \leq f(x_2) \qquad & \text{(for all } x_1, x_2 \in X \text{)} \\ f(\vee_{n\in\mathbb{N}}\, x_n) = \vee_{n\in\mathbb{N}}\, f(x_n) \qquad & \text{(for every monotone sequence } (x_n)_{n\in\mathbb{N}}) \end{aligned}$$

Problem 14. Prove that the composition $g \cdot f : X \to Z$ of continuous maps $f : X \to Y$ and $g : Y \to Z$ is continuous.

These last three problems contain basic ingredients to interpret while-loops as fixpoints. Another basic ingredient is *Kleene's fixpoint theorem* which tells how to build the least fixpoint of a continuous map $f : X \to X$ on an ω-CPO X with a bottom element ($\bot$). Recall that the fixpoint is denoted by $\mathrm{lfp}\, f$ and explicitly built as,

$$\mathrm{lfp}\, f = \vee_{n \in \mathbb{N}} f^n(\bot)$$

Problem 15. Show that $\mathrm{lfp}\, f$ is indeed the least fixpoint of f.

Finally after some investments on Domain theory we were able to neatly establish the denotational semantics of our while-language, detailed in Figure 4.

$$\begin{aligned}
[\![\mathtt{x} := \mathtt{e}]\!] &= \sigma \mapsto \sigma[[\![\mathtt{e}]\!]/\mathtt{x}] \\
[\![\mathtt{p}\ ;\ \mathtt{q}]\!] &= [\![\mathtt{q}]\!] \cdot [\![\mathtt{p}]\!] \\
[\![\mathtt{if}\ \mathtt{b}\ \mathtt{then}\ \mathtt{p}\ \mathtt{else}\ \mathtt{q}]\!] &= [[\![\mathtt{p}]\!], [\![\mathtt{q}]\!]] \cdot \mathrm{dist} \cdot \langle [\![\mathtt{b}]\!], \mathrm{id} \rangle \\
[\![\mathtt{while}\ \mathtt{b}\ \mathtt{do}\ \{\ \mathtt{p}\ \}]\!] &= \mathrm{lfp}\ \Big(k \mapsto [k \cdot [\![\mathtt{p}]\!], \mathrm{id}] \cdot \mathrm{dist} \cdot \langle [\![\mathtt{b}]\!], \mathrm{id} \rangle \Big)
\end{aligned}$$

Figure 4: A denotational semantics for the while-language.

Problem 16. Although used in the new semantics of our while-language (Figure 4) in the lectures we did not see a denotational semantics for arithmetic expressions. Can you define one such semantics now?

Great! We now have three different, complementary semantics for our while-language (*viz.* small-step, big-step, and denotational). It is therefore useful to prove that they all 'agree' with each other. But in fact we just need to relate the new semantics with the big-step version, for we have already established a connection between the latter and the small-step semantics. Technically we achieve this desired connection by proving the equivalence below

$$\langle \mathtt{p}, \sigma \rangle \Downarrow \sigma' \text{ iff } [\![\mathtt{p}]\!](\sigma) = \sigma' \tag{1}$$

for every program $\mathtt{p}$ and states σ, σ'. We will focus on proving just the left-to-right direction of this equivalence.

Problem 17. Consider programs $\mathtt{p}$ and $\mathtt{p}'$ and states σ, σ'. Prove by induction (over the depth of derivation trees) the following implication.

$$\langle \mathtt{p}, \sigma \rangle \Downarrow \sigma' \text{ implies } [\![\mathtt{p}]\!](\sigma) = \sigma'$$

(Hint: Use the fixpoint equation that you learned in lectures).

From Equivalence (1) between the denotational and big-step semantics we easily establish that for all programs $\mathtt{p}$ and $\mathtt{q}$ the following equivalence holds.

$$\mathtt{p} \equiv_o \mathtt{q} \text{ iff } [\![\mathtt{p}]\!] = [\![\mathtt{q}]\!]$$

But we also want to learn more about the notion of contextual equivalence ($\equiv$). Recall for example that in Problem (11) we were not able to show whether $\mathtt{p} \equiv_o \mathtt{q}$ entails $\mathtt{p} \equiv \mathtt{q}$. Can denotational semantics help us in this quest?

Problem 18. Consider two programs $\mathtt{p}$ and $\mathtt{q}$. Prove via induction on the syntactic structure of programs that the following implication holds.

$$[\![\mathtt{p}]\!] = [\![\mathtt{q}]\!] \text{ entails } \Big(\text{for all contexts } C.\, [\![C[\mathtt{p}]]\!] = [\![C[\mathtt{q}]]\!]\Big)$$

Perfect, we established that contextual equivalence (which quantifies over infinitely many contexts) reduces to simple *equality* of denotations! We can thus use our denotational semantics (and all mathematical theories it inherits, such as 'program calculus') to prove equivalences used by programmers and compilers all the time.

Problem 19. Prove that the following equivalences indeed hold.

- $(\mathtt{p}\,;\mathtt{q})\,;\mathtt{r} \equiv \mathtt{p}\,;(\mathtt{q}\,;\mathtt{r})$
- $(\mathtt{if}\ \mathtt{b}\ \mathtt{then}\ \mathtt{p}\ \mathtt{else}\ \mathtt{q})\,;\mathtt{r} \equiv \mathtt{if}\ \mathtt{b}\ \mathtt{then}\ \mathtt{p}\,;\mathtt{r}\ \mathtt{else}\ \mathtt{q}\,;\mathtt{r}$
- $\mathtt{while}\ \mathtt{b}\ \{\mathtt{p}\} \equiv \mathtt{if}\ \mathtt{b}\ \mathtt{then}\ \mathtt{p}\,;\mathtt{while}\ \mathtt{b}\ \{\mathtt{p}\}\ \mathtt{else}\ \mathtt{skip}$
- $\mathtt{while}\ \mathtt{b}\ \{\mathtt{p}\}\,;\mathtt{q} \equiv \mathtt{if}\ \mathtt{b}\ \mathtt{then}\ \mathtt{p}\,;\mathtt{while}\ \mathtt{b}\ \{\mathtt{p}\}\,;\mathtt{q}\ \mathtt{else}\ \mathtt{q}$
- $\mathtt{while}\ \mathtt{ff}\ \{\mathtt{p}\}\,;\mathtt{q} \equiv \mathtt{q}$
- $\mathtt{while}\ \mathtt{tt}\ \{\mathtt{p}\} \equiv \mathtt{while}\ \mathtt{tt}\ \{\mathtt{q}\}$

Reflect on the differences between performing this exercise with the denotational, small-step, and big-step semantics.

Problem 20. Recall our big-step semantics and let us consider the program $\mathtt{while}\ \mathtt{tt}\ \{\mathtt{p}\}$ together with a state σ. Although obvious, can you prove that there is no state σ' such that $\mathtt{while}\ \mathtt{tt}\ \{\mathtt{p}\} \Downarrow \sigma'$? This can be problematic, because it is often cumbersome to prove the *non-existence* of things. However perhaps the denotational semantics can help us in this quest. What do you think?

3 Axiomatic Semantics

Finally we lean over axiomatic semantics, which as you know has a more logical nature and it is therefore better suited to study *program correctness*, among other things. This style of semantics puts the notion of proposition (*i.e.* condition) at center stage, which of course raises the question of which logic to choose for writing and reasoning about such propositions. We observed that one does not actually need to choose a logic right from the outset, but rather one makes basic assumptions about whatever logic is chosen. One of these assumptions is that every proposition Φ will correspond to a set $[\![\Phi]\!] \subseteq \mathrm{State}_\perp$ (of elements that satisfy this proposition). From this we worked out what a Hoare triple ought to mean mathematically, *viz.*

$$\{\Phi\}\,\mathtt{p}\,\{\Psi\} \qquad \text{means} \qquad \Big(\forall x \in \mathrm{State}_\perp.\, x \in [\![\Phi]\!] \text{ entails } [\![\mathtt{p}]\!](x) \in [\![\Psi]\!]\Big)$$

Note that we are recurring to the previous denotational semantics to give a meaning to Hoare triples.

Problem 21. Argue *informally* whether the following Hoare triples hold.

- $\{\mathtt{tt}\}$ `while` $\mathtt{tt}$ `skip` $\{\mathtt{ff}\}$
- $\{\mathtt{tt}\}$ `if` $\mathtt{b}$ `then` $\mathtt{x} := 2$ `else` $\mathtt{x} := 3$ $\{\mathtt{x} \geq 2\}$
- $\{\mathtt{x} = \mathtt{a} \wedge \mathtt{y} = \mathtt{b}\}$ $\mathtt{x} := \mathtt{y}\,;\, \mathtt{y} := \mathtt{x}$ $\{\mathtt{x} = \mathtt{b} \wedge \mathtt{y} = \mathtt{a}\}$
- $\{\mathtt{x} = \mathtt{a} \wedge \mathtt{y} = \mathtt{b}\}$ $\mathtt{aux} := \mathtt{x}\,;\, \mathtt{x} := \mathtt{y}\,;\, \mathtt{y} := \mathtt{aux}$ $\{\mathtt{x} = \mathtt{b} \wedge \mathtt{y} = \mathtt{a}\}$

We continued exploring the mathematical meaning of Hoare triples, and among other things made the following observation.

$$\Big(\forall x \in \mathrm{State}_\perp.\, x \in [\![\Phi]\!] \text{ entails } [\![\mathtt{p}]\!](x) \in [\![\Psi]\!]\Big) \text{ iff } [\![\Phi]\!] \subseteq [\![\mathtt{p}]\!]^{-1}([\![\Psi]\!])$$

Although apparently vapid it is in fact quite profound. It suggests that we study the validity of Hoare triples from the point of view of *inverse images*, via the denotational semantics, but also that the set $[\![\mathtt{p}]\!]^{-1}([\![\Psi]\!])$ has a prominent rôle in this context. Indeed the latter turns out to correspond precisely to the *weakest pre-condition* w.r.t. a program $\mathtt{p}$ and a post-condition Ψ.

As E. Dijkstra asserted it is quite informative to determine the weakest pre-condition related to a program $\mathtt{p}$ and a post-condition Ψ. However it would be a hassle to calculate inverse images of some set directly. The weakest pre-condition semantics we saw in the lectures circumvents this problem (among other things). It is detailed in Figure 5 (note our assumption about the logic having *infinite* conjunctions, negation, and disjunctions).

$$\begin{aligned}
\mathrm{wp}\,(\mathtt{x} := \mathtt{e}, \Phi) &= \Phi[\mathtt{e}/\mathtt{x}] \\
\mathrm{wp}\,(\mathtt{p}\,;\,\mathtt{q}, \Phi) &= \mathrm{wp}\,(\mathtt{p}, \mathrm{wp}\,(\mathtt{q}, \Phi)) \\
\mathrm{wp}\,(\mathtt{if}\ \mathtt{b}\ \mathtt{then}\ \mathtt{p}\ \mathtt{else}\ \mathtt{q}, \Phi) &= \mathtt{b} \wedge \mathrm{wp}\,(\mathtt{p}, \Phi) \ \vee\ \neg\mathtt{b} \wedge \mathrm{wp}\,(\mathtt{q}, \Phi) \\
\mathrm{wp}\,(\mathtt{while}\ \mathtt{b}\ \mathtt{do}\ \{\,\mathtt{p}\,\}, \Phi) &= \bigwedge_{n \in \mathbb{N}} \Psi_n
\end{aligned}$$

$$\begin{aligned}
\Psi_0 &= \mathtt{tt} \\
\Psi_{n+1} &= \neg\mathtt{b} \wedge \Phi \ \vee\ \mathtt{b} \wedge \mathrm{wp}\,(\mathtt{p}, \Psi_n)
\end{aligned}$$

Figure 5: A weakest pre-condition semantics for the while-language.

Let us try this new semantics with a series of very simple examples.

Problem 22. Calculate the weakest pre-conditions w.r.t. the following pairs.

- $(\mathtt{x} := \mathtt{y},\ \mathtt{x} \geq 1)$
- $(\mathtt{if}\ \mathtt{b}\ \mathtt{then}\ \mathtt{x} := 2\ \mathtt{else}\ \mathtt{x} := 3,\ \mathtt{x} \geq 2)$
- $(\mathtt{x} := \mathtt{y}\,;\, \mathtt{y} := \mathtt{x},\ \mathtt{x} = \mathtt{b} \wedge \mathtt{y} = \mathtt{a})$
- $(\mathtt{aux} := \mathtt{x};\, \mathtt{x} := \mathtt{y}\,;\, \mathtt{y} := \mathtt{aux},\ \mathtt{x} = \mathtt{b} \wedge \mathtt{y} = \mathtt{a})$

Go to your old material about 'algorithms and complexity' and check whether you can prove the weakest pre-conditions of other pairs.

Problem 23. Prove that $\mathrm{wp}\,(\mathtt{p}, \mathtt{tt}) \equiv \mathtt{tt}$ for every program $\mathtt{p}$.

Right, following our traditions we should connect this new semantics to the previous ones (*i.e.* we should be able to prove that they all 'agree' with each other). And indeed we saw in the lectures that the following equation holds for every program $\mathtt{p}$ and condition Φ.

$$[\![\mathrm{wp}\,(\mathtt{p}, \Phi)]\!] = [\![\mathtt{p}]\!]^{-1}([\![\Phi]\!]) \tag{2}$$

As mentioned in the lectures, the respective proof can be neatly established via Domain theory. Unfortunately the techniques involved are a bit more advanced than the ones addressed in the lectures :-(.

Problem 24. Resort to Equation (2) to prove the following equivalences.

- $\mathrm{wp}\,(\mathtt{p}, \mathtt{tt}) \equiv \mathtt{tt}$
- $\mathrm{wp}\,(\mathtt{p}, \Phi \wedge \Psi) \equiv \mathrm{wp}\,(\mathtt{p}, \Phi) \wedge \mathrm{wp}\,(\mathtt{p}, \Psi)$

What are differences between showing the first equivalence in this problem and in Problem (23)? Can you prove the second equivalence without recurring to Equation (2)?

We also saw how to generate a calculus from this pre-condition semantics for showing the validity of Hoare triples. This calculus is prominently simple: in fact it only contains one rule, *viz.*

$$\frac{\vdash \Phi \to \mathrm{wp}\,(\mathtt{p}, \Psi)}{\vdash \{\Phi\}\,\mathtt{p}\,\{\Psi\}}$$

Problem 25. Prove that the calculus is sound (*i.e.* correct) and relatively complete (*i.e.* we can prove that every valid Hoare triple is valid if the underlying logic is complete).

Later on we observed that although interesting the calculus is a bit 'too rigid' to prove the validity of Hoare triples: indeed the fact that it is strictly based on weakest pre-conditions and infinite conjunctions makes it less amenable to practical uses. This brings us back to old friends, namely the Hoare calculus that you saw in 'algorithms and complexity' and which is now detailed in Figure 6.

Problem 26. Recall Problem (22) where we derived weakest pre-conditions for different pairs. Show now via Hoare calculus that these weakest pre-conditions together with the respective pairs form valid Hoare triples. Try then to establish the validity of the following Hoare triple.

$$\{\mathtt{x} = \mathtt{n} \geq 0 \wedge \mathtt{y} = 1\}\ \mathtt{fact}\ \{\mathtt{y} = \mathtt{n}!\}$$

$\mathtt{fact} = \mathtt{while}\ \mathtt{x} > 0\ \{\mathtt{y} := \mathtt{x} \times \mathtt{y}\ ;\ \mathtt{x} := \mathtt{x} - 1\}$. Reflect on the differences between trying to prove the valid of Hoare triples w.r.t. weakest pre-condition calculus and Hoare calculus.

Well now that we have our old friend back, it makes sense to prove that it is correct, a fact that we were not able to formulate much less prove in 'algorithms and complexity'.

$$\frac{}{\vdash_H \{\Phi\}\,\mathtt{skip}\,\{\Phi\}} \qquad \frac{}{\vdash_H \{\Phi[\mathtt{e}/\mathtt{x}]\}\,\mathtt{x} := \mathtt{e}\,\{\Phi\}}$$

$$\frac{\vdash_H \{\Phi\}\,\mathtt{p}\,\{\Psi\} \qquad \vdash_H \{\Psi\}\,\mathtt{q}\,\{\Xi\}}{\vdash_H \{\Phi\}\,\mathtt{p}\,;\,\mathtt{q}\,\{\Xi\}} \qquad \frac{\vdash_H \{\Phi \wedge \mathtt{b}\}\,\mathtt{p}\,\{\Phi\}}{\vdash_H \{\Phi\}\,\mathtt{while}\,\mathtt{b}\,\{\,\mathtt{p}\,\}\,\{\neg\mathtt{b} \wedge \Phi\}}$$

$$\frac{\vdash_H \{\mathtt{b} \wedge \Phi\}\,\mathtt{p}\,\{\Psi\} \qquad \vdash_H \{\neg\mathtt{b} \wedge \Phi\}\,\mathtt{q}\,\{\Psi\}}{\vdash_H \{\Phi\}\,\mathtt{if}\,\mathtt{b}\,\mathtt{then}\,\mathtt{p}\,\mathtt{else}\,\mathtt{q}\,\{\Psi\}}$$

$$\frac{\vdash \Phi \to \Psi \quad \vdash_H \{\Psi\}\,\mathtt{p}\,\{\Xi\} \quad \vdash \Xi \to \Omega}{\vdash_H \{\Phi\}\,\mathtt{p}\,\{\Psi\}}$$

Figure 6: Hoare calculus for our while-language.

Problem 27. Prove by induction (on the depth of derivation trees) that the Hoare calculus is sound, *i.e.* that the following implication holds for every program $\mathtt{p}$ and conditions Φ and Ψ.

$$\vdash_H \{\Phi\}\,\mathtt{p}\,\{\Psi\} \text{ entails } [\![\Phi]\!] \subseteq [\![\mathtt{p}]\!]^{-1}([\![\Psi]\!])$$

You can skip the case of while-loops, for it is much harder to prove than the other cases. Note however that it can be neatly proved via Domain theory.

Finally another thing we saw in the lectures is that for every program $\mathtt{p}$ and condition Φ we were able to prove $\vdash_H \{\mathrm{wp}\,(\mathtt{p}, \Phi)\}\,\mathtt{p}\,\{\Psi\}$.

Problem 28. Use this last result to prove the relative completeness of the Hoare calculus from the relative completeness of the weakest pre-condition calculus.