Formal Programming Language Semantics note 11

Adequacy and its consequences

In this note we will consider the relationship between operational and denotational semantics. It may seem at first that there is not very much essential difference between the approaches, so first let us try to articulate informally what the important differences are. Certainly, one difference is just psychological: in operational semantics we think of processes in time, whereas in denotational semantics we tend to view the situation from the perspective of "eternity". However, there is another, more substantial difference. In operational semantics, we are concerned with statements about the behaviour of programs in *particular states*, and the relationships between such statements. In denotational semantics, we are concerned with statements about the behaviour of programs in *all states*, and the relationships between them. This makes a real difference, as we hope to make clear by the end of this note.

The adequacy theorem Before proceeding further, let us see how to prove that the operational and denotational semantics of **IMP** "agree". One often expresses this agreement by saying the denotational semantics is *adequate* with respect to the operational semantics. (This rather suggests we are taking the latter as the definition of the language and the former has to match it, though of course it could be the other way round.) The proof may seem rather hard work, but it is a worthwhile investment since many other results flow rather easily from it. The proof also provides two good illustrations of the important technique of *structural induction*. As is typical for such proofs, there are a lot of induction cases to check, but almost all of them are trivial and boring.

Theorem 1 (Adequacy) *Let P be any phrase of* **IMP**. *Then for any state σ and any result value R we have*

$$\langle P, \sigma\rangle \Downarrow R \quad \Longleftrightarrow \quad [\![P]\!](\sigma) = R.$$

PROOF $(\Rightarrow)$: We argue by induction on the height of the derivation tree for $\langle P, \sigma\rangle \Downarrow R$. We will show, first, that if $\langle P, \sigma\rangle \Downarrow R$ has a derivation of height 1 then $[\![P]\!](\sigma) = R$; and secondly, that if $[\![P]\!](\sigma) = R$ whenever $\langle P, \sigma\rangle \Downarrow R$ has a derivation of height r, the same holds for height $r + 1$.

For the first part, we are supposing $\langle P, \sigma\rangle \Downarrow R$ has a trivial derivation consisting of just one evaluation rule with no premises. There are four such rules in the operational semantics — rules (1), (2), (5) and (13) of Note 4 — so we just need to consider each of these in turn. All four cases are trivial; we do rule (2)

as an example. If the rule in question is rule (2), then P is some identifier X, and $R = \sigma(X)$. But by clause (2) of the definition of $[\![-]\!]$ (see Note 9), we have immediately that $[\![X]\!](\sigma) = \sigma(X)$ as required. Likewise for the other three cases.

For the second part, let us suppose the result holds for all derivations of height r or less, and let $\langle P, \sigma\rangle \Downarrow R$ be an assertion whose derivation has height $r+1$. Consider the final rule in this derivation; it has one or more premises $\langle P_i, \sigma_i\rangle \Downarrow R_i$, each with a derivation of height r or less, so by the induction hypothesis we have $[\![P_i]\!](\sigma_i) = R_i$ for each i. To show that $[\![P]\!](\sigma) = R$, we need to consider each possibility for the final rule in turn. All the cases except for rules (18) and (19) are trivial; let us do rule (16) as an example. Here P is the command `if` b `then` c_0 `else` c_1, and $R = \sigma'$; and we know by the induction hypothesis that $[\![b]\!](\sigma) = \mathit{true}$ and $[\![c_0]\!](\sigma) = \sigma'$. But now by clause (16/17) of Note 9 we have

$$[\![P]\!](\sigma) \;=\; \begin{cases} [\![c_0]\!](\sigma) & \text{if } [\![b]\!](\sigma) = \mathit{true} \\ [\![c_1]\!](\sigma) & \text{if } [\![b]\!](\sigma) = \mathit{false} \end{cases} \;=\; [\![c_0]\!](\sigma) \;=\; \sigma'$$

Now the two interesting cases. For rule (18), we have $P \equiv$ `while` b `do` c and $R = \sigma$, and by the induction hypothesis $[\![b]\!](\sigma) = \mathit{false}$. Let h_k and h be defined as in Note 9; then clearly $h_1(\sigma) = \sigma$. But $h_1 \subseteq h$, so $[\![P]\!](\sigma) = h(\sigma) = \sigma$ as required.

For rule (19), we have P as above, $R = \sigma''$, and by the induction hypothesis $[\![b]\!](\sigma) = \mathit{true}$, $[\![c]\!](\sigma) = \sigma'$ and $[\![P]\!](\sigma') = \sigma''$. Again, let h_k and h be as in Note 9, so that $[\![P]\!] = h$. Then $h(\sigma') = \sigma''$, so by the construction of h, we have $h_k(\sigma') = \sigma''$ for some k. Now by the definition of h_{k+1} it is clear that $h_{k+1}(\sigma) = \sigma''$. But now since $h_{k+1} \subseteq h$, we have $[\![P]\!](\sigma) = h(\sigma) = \sigma''$ as required. This completes the induction.

[**Note:** It is quite instructive to think about this argument in terms of the inductive definition of the evaluation relation E given in Note 4. There we defined E to be the smallest set of triples (P, σ, R) closed under the evaluation rules. Suppose we now take F to be the set of triples (P, σ, R) such that $[\![P]\!](\sigma) = R$. The above argument by cases essentially shows that F is closed under the evaluation rules. Since E is the smallest such set, we may conclude that $E \subseteq F$, which is exactly the left-to-right implication of the Theorem.]

($\Leftarrow$): Suppose that $[\![P]\!](\sigma) = R$; we wish to show that $\langle P, \sigma\rangle \Downarrow R$. Recall that the definition of $[\![-]\!]$ is given by induction on the syntactic structure of P, so this time we argue by structural induction on P (or by induction on the size of P, if you prefer), with a separate case for each syntactic construct of **IMP**. All the cases are trivial except the one for `while`. As before, we illustrate the idea by doing one "base case" (for variables), and one "induction case" (for if-commands).

Suppose P is just a variable X. Then by clause (2) of Note 9, $[\![P]\!](\sigma) = \sigma(X)$ so $R = \sigma(X)$. But then $\langle P, \sigma\rangle \Downarrow R$ immediately by evaluation rule (2).

Now suppose $P \equiv$ `if` b `then` c_0 `else` c_1, where $[\![b]\!](\sigma) = t$ say, and $[\![c_0]\!](\sigma) = \sigma'$, $[\![c_1]\!](\sigma) = \sigma''$. Then by clause (16/17) of Note 9, we have

$$[\![P]\!](\sigma) \;=\; \begin{cases} \sigma' & \text{if } t = \mathit{true} \\ \sigma'' & \text{if } t = \mathit{false} \end{cases}$$

In addition, by the induction hypothesis we have $\langle b, \sigma\rangle = t$, $\langle c_0, \sigma\rangle = \sigma'$, and $\langle c_1, \sigma\rangle = \sigma''$. There are now two subcases. If $t =$ *true* then by an application of rule (16) we may deduce $\langle P, \sigma\rangle = \sigma'$; but also $[\![P]\!](\sigma) = \sigma'$ in this case so we are done. If $t =$ *false*, a similar argument applies using rule (17).

Finally the interesting case. Suppose $P \equiv$ `while` b `do` c, and that $[\![P]\!](\sigma) = R$ Let h, h_k be as in Note 9. There are two subcases:

- If $[\![b]\!](\sigma) =$ *false*, then $h_1(\sigma) = \sigma$ so $R = h(\sigma) = \sigma$ since $h_1 \subseteq h$. But also $\langle b, \sigma\rangle =$ *false* by induction hypothesis, and so by rule (18) we have $\langle P, \sigma\rangle \Downarrow \sigma$.
- Suppose $[\![b]\!](\sigma) =$ *true*. Since $h(\sigma) = R$, we have $h_{k'}(\sigma) = R$ for some k'. We may assume $k' > 1$, and take $k' = k + 1$. By the definition of h_{k+1}, we have that $R = h_k([\![c]\!](\sigma))$; in particular, the value of the right hand side here is defined. Take $\sigma' = [\![c]\!](\sigma)$ and $\sigma'' = h_k(\sigma')$. By the induction hypothesis we have that $\langle b, \sigma\rangle \Downarrow$ *true*, $\langle c, \sigma\rangle \Downarrow \sigma'$, and (since $h_k \subseteq h = [\![P]\!]$) $\langle P, \sigma'\rangle \Downarrow \sigma''$. But now by rule (19) we may conclude that $\langle P, \sigma\rangle \Downarrow \sigma'' = R$ as required. □

[Exercise: The two halves of this proof look very similar. Why could we not roll them together into one single proof by induction?]

All that was a bit of a slog, but the above theorem appears to play a kind of central role for **IMP**, in that once we have proved it, a lot of other things we want to know about **IMP** flow from it rather easily. (We could of course prove these other things directly, but by proving the above theorem we factor out most of the hard work which is done once for all.) We will now look at a few consequences of the adequacy theorem.

As a first example, we mentioned in Note 4 the following "determinacy" properties of **IMP**, but didn't really prove them properly:

- For all a, σ [resp. b, σ] there is a unique n [t] such that $\langle a, \sigma\rangle \Downarrow n$ [$\langle b, \sigma\rangle \Downarrow t$].
- For all c, σ there is at most one σ' such that $\langle c, \sigma\rangle \Downarrow \sigma'$.

It isn't too hard to prove these directly (via some tedious inductions), but once we have the adequacy theorem they follow immediately, since $[\![a]\!], [\![b]\!]$ are (single-valued) total functions and $[\![c]\!]$ is a (single-valued) partial function.

Compositionality and observational equivalence. For a more substantial application of adequacy, let us return to some questions relating to observational equivalence which we introduced in Note 6. Recall that two terms P_1, P_2 are *functionally equivalent* if for all σ and R we have $\langle P_1, \sigma\rangle \Downarrow R$ iff $\langle P_2, \sigma\rangle \Downarrow R$. Clearly, this is the same as saying that $[\![P_1]\!] = [\![P_2]\!]$. Likewise, we say P_1, P_2 are *observationally equivalent* if for all contexts C of appropriate type, $[\![C[P_1]]\!] = [\![C[P_2]]\!]$. In Note 6 we raised the question whether functional equivalence is the same as observational equivalence. By thinking about the run-time behaviour of programs, one can see intuitively that the answer is yes, since the only way in which a

subprogram P can affect the result of a larger program is through the result obtained from running P in various initial states. (You may well have thought about this kind of thing when doing Q.4 of Exercise Sheet 1.) With a bit of effort, one can turn this intuition into a direct proof in terms of the operational semantics, but a much simpler and perfectly rigorous argument can now be given using the denotational semantics.

The argument is very general, and something of the kind will apply whenever we have a denotational semantics which is both *adequate* and *compositional*. Compositionality is one of the fundamental ideas of denotational semantics, so it is worth taking some time to understand it clearly. First, note that in the definition of $[\![-]\!]$ (see Note 9), all the clauses have the following form: if P is a phrase whose immediate subphrases are $P_1, \ldots, P_k$, then $[\![P]\!]$ is simply a function of $[\![P_1]\!], \ldots, [\![P_k]\!]$. More formally, for each syntactic construct $K[-_1, \ldots, -_k]$ of **IMP**, there is a fixed function κ such that if $P = K[P_1, \ldots, P_k]$ then $[\![P]\!] = \kappa([\![P_1]\!], \ldots, [\![P_k]\!])$.

Next, note that any context $C[-]$ at all can be built up by composing or plugging together various syntactic constructs $K[\cdots]$. So we have:

Theorem 2 (Compositionality) *For any context $C[-]$ of* **IMP***, where $-$ is a placeholder of type u and $C[-]$ has type u', there is a function $\xi : \mathcal{D}^u \to \mathcal{D}^{u'}$ such that for all phrases P of type u we have*

$$[\![C[P]]\!] \;=\; \xi([\![P]\!])$$

PROOF An easy induction on the structure of $C[-]$. The base case is when $C[-]$ is just $-$: here we just take ξ to be the identity function, and the result is trivial. All the induction cases may be handled together thanks to the above notation. Suppose $C[-] \equiv K[C_1[-], \ldots, C_k[-]]$ where $K[\cdots]$ is one of the syntactic constructs of **IMP**. (For some of these, k will be 0, but that's OK.) By the induction hypothesis, there are functions $\xi_1, \ldots, \xi_k$ such that $[\![C_i[P_i]]\!] = \xi_i([\![P_i]\!])$ for any P_i of suitable type. Let κ be the semantic function for K as above, and define $\xi = \Lambda x.\, \kappa(\xi_1(x), \ldots, \xi_k(x))$. It is clear that ξ has the desired property for $C[-]$. $\Box$

It is natural to think of ξ here as the "meaning" of $C[-]$. To summarize the theorem, everything we need to know about a term P in order to determine how P contributes to the meaning of any larger program is encapsulated by $[\![P]\!]$. It follows immediately that if $[\![P_1]\!] = [\![P_2]\!]$ then $[\![C[P_1]]\!] = [\![C[P_2]]\!]$; in other words, functional equivalence implies observational equivalence.

This means that a denotational semantics can be used to give rigorous proofs of observational equivalence of particular program phrases: to show that P_1 and P_2 are observationally equivalent, it is enough to check that $[\![P_1]\!] = [\![P_2]\!]$. A similar statement will also hold for *patterns* as in Note 6. [Exercise: Formulate a precise statement to this effect.] One could use this to prove, for example, that the pairs of patterns in Q.4 of Sheet 1 are indeed observationally equivalent.

John Longley