Chapter 7

Case study: Square root

In this chapter we follow a small but complete development from beginning to end. The key to success — as is very often the case — will be the finding of an appropriate invariant for an iteration. Application of refinement laws, and the setting out of developments, will become routine with practice; but finding invariants is always a fresh challenge.

7.1 Abstract program: the starting point

We are given a natural number s, and we must set the natural number r to the greatest integer not exceeding $\sqrt{s}$, where $\sqrt{\ }$ takes the non-negative square root of its argument. Thus starting from $s = 29$, for example, we would expect to finish with $s = 29 \wedge r = 5$.

Here is our abstract program:

$$\begin{array}{ll} \mathbf{var}\ r, s : \mathbb{N} \bullet & \\ r := \lfloor \sqrt{s} \rfloor\ . & \qquad \text{(i)} \end{array}$$

Although an assignment, the command (i) is not code, because in this case study we assume that neither $\sqrt{\ }$ nor $\lfloor\ \rfloor$ is code. Our aim in the development to follow will be to remove them from the program.

7.2 Remove 'exotic' operators

These first refinement steps remove the square-root and floor functions ('exotic' only because they are not code) from the program by drawing on their mathematical definitions. The steps are routine, and leave us with a specification from which $\sqrt{\ }$ and $\lfloor\ \rfloor$ have disappeared:

$$
\begin{array}{ll}
= & \text{“}\textit{simple specification}\text{ 1.7”} \\
 & r\text{: } [r = \lfloor \sqrt{s} \rfloor] \\
= & \text{“definition } \lfloor\ \rfloor\text{”} \\
 & r\text{: } [r \le \sqrt{s} < r+1] \\
= & \text{“definition } \sqrt{\ }\text{”} \\
 & r\text{: } [r^2 \le s < (r+1)^2] \ .
\end{array}
\qquad \text{(ii)}
$$

Comparing (i) and (ii), we can see that the assignment is written for the client: it uses powerful operators, leading to clear and succinct expression. Above all, it is easy to understand. But we have now moved from assignment to specification, and for two reasons: we need the freedom of a formula (rather than just an expression) to exploit the definitions of $\sqrt{\ }$ and $\lfloor\ \rfloor$; and a specification is easier to develop *from* than an assignment.

7.3 Look for an invariant

The postcondition in *iteration* 5.5 is of the form $inv \wedge \neg GG$, and so we should investigate rewriting our postcondition in (ii) that way. There are two immediate possibilities:

$$
\begin{array}{ll}
 & r^2 \le s \wedge \neg(s \ge (r+1)^2) \\
\text{and} & s < (r+1)^2 \wedge \neg(r^2 > s) \ .
\end{array}
$$

The first would lead to an iteration

$$\mathbf{do}\ s \ge (r+1)^2 \rightarrow \cdots\ \mathbf{od}\ ,$$

with invariant $r^2 \le s$. (The assignment $r := 0$ could establish the invariant initially.) The second would lead to

$$\mathbf{do}\ r^2 > s \rightarrow \cdots\ \mathbf{od}\ ,$$

with invariant $s < (r+1)^2$ (whose initialisation is not so straightforward — but perhaps $r := s$ would do).

Either of those two approaches would succeed (and in the exercises you are invited to try them). But the resulting programs are not as efficient as the one we are about to develop. We rewrite the postcondition as

$$r^2 \le s < q^2 \wedge r+1 = q\ ,$$

taking advantage of a new variable q that will be introduced for the purpose. (We use ‘rewrite’ here a bit loosely, since the two postconditions are definitely *not* equivalent. The new one implies the original, as it should — remember *strengthen postcondition* 1.1.) That surprising step is nevertheless a fairly common one in practice: one replaces an expression by a variable, adding a conjunct that makes them equal.

The refinement is the following:

$$\text{(ii)} \sqsubseteq \textbf{var } q : \mathbb{N} \bullet \\ q, r \colon [r^2 \le s < q^2 \wedge r + 1 = q] \ .$$

Now having separate bounds on s gives us more scope: initially, r and q could be far apart. Finally, we should establish $r + 1 = q$, and that will be the source of our increased efficiency: we can move them in big steps.

The next few refinements are routine when introducing an iteration: declare an abbreviation (I for the invariant, just to avoid writing it out again and again), establish the invariant with an assignment (initialisation), and introduce an iteration whose body maintains it.

The abbreviation $I \ \widehat{=} \ \cdots$ is written as a decoration of the refinement. Like other decorations there (**var**, **con**), it is available in the development from that point on.

$$
\begin{array}{lll}
\sqsubseteq & I \ \widehat{=} \ r^2 \le s < q^2 \bullet & \\
 & q, r \colon [I \wedge r + 1 = q] & \\
\sqsubseteq & q, r \colon [I] \, ; & \text{(iii)} \\
 & q, r \colon [I \ , \ I \wedge r + 1 = q] & \lhd \\
\sqsubseteq & \text{“invariant } I \text{, variant } q - r\text{”} & \\
 & \textbf{do } r + 1 \ne q \rightarrow & \\
 & \quad q, r \colon [r + 1 \ne q \ , \ I \ , \ q - r < q_0 - r_0] & \lhd \\
 & \textbf{od} \ . &
\end{array}
$$

Note that the invariant bounds the variant below — that is, we have $I \Rrightarrow 0 \le q - r$ — and so we need not write the '$0 \le \cdots$' explicitly in the postcondition. We leave the refinement of (iii) to Exercise 7.1.

Our next step is motivated by the variant: to decrease it, we must move r and q closer together. If we move one at a time, whichever it is will take a value strictly between r and q. So we introduce a local variable for that new value, and make this step:

$$
\begin{array}{lll}
\sqsubseteq & \textbf{var } p : \mathbb{N} \bullet & \\
 & p \colon [r + 1 < q \ , \ r < p < q] \, ; & \text{(iv)} \\
 & q, r \colon [r < p < q \ , \ I \ , \ q - r < q_0 - r_0] \ . & \lhd
\end{array}
$$

Strictly speaking, there should be an I in the postcondition of (iv), since in our use of *sequential composition* 3.3 the formula *mid* is clearly $r < p < q \wedge I$. (It is necessarily the same as the precondition of the $\lhd$-marked command, which includes I: recall *specification invariant* 5.6.) But in fact I is established by (iv) whether we write it there or not, since it was in the precondition of the iteration body and does not contain p (the only variable that (iv) can change). Thus informally we can see that (iv) cannot falsify I — but in fact we have appealed (tacitly) to this law:

Law 7.1 <u>remove invariant</u> Provided w does not occur in inv,

$$w\colon [pre\ ,\ inv\ ,\ post] \quad \sqsubseteq \quad w\colon [pre\ ,\ post]\ .$$

□

We now intend to re-establish $r^2 \leq s < q^2$ in the postcondition with an assignment: either $q := p$ or $r := p$. By investigating the proviso of *assignment* 5.2, calculating $(r^2 \leq s < q^2)[q\backslash p]$, we can see that the first requires a precondition $s < p^2$ (or at least as strong as that); similarly, the second requires $s \geq p^2$. That case analysis supplies the guards for our alternation:

$$\begin{array}{lll} \sqsubseteq & \mathbf{if}\ s < p^2 \rightarrow q\colon [s < p^2 \wedge p < q\ ,\ I\ ,\ q < q_0] & \text{(v)} \\ & [\!]\ \ s \geq p^2 \rightarrow r\colon [s \geq p^2 \wedge r < p\ ,\ I\ ,\ r_0 < r] & \text{(vi)} \\ & \mathbf{fi} & \\ \text{(v)} & \sqsubseteq\ q := p & \\ \text{(vi)} & \sqsubseteq\ r := p\ . & \end{array}$$

Note that the refinement markers (v) and (vi) refer to the bodies of the alternation branches, and do not include the guards.

The simplifications of the variant inequalities are possible because we have used *contract frame* 5.4 in each case. In (v) for example, removing r from the frame allows us to rewrite $q - r < q_0 - r_0$ as $q - r < q_0 - r$, thence just $q < q_0$.

Now only (iv) is left, and it has many refinements: the assignment $p := r + 1$ and $p := q - 1$ are two. But a faster decrease in the variant — hence our more efficient program — will result if we choose p midway between q and r:

$$\text{(iv)} \quad \sqsubseteq \quad p := (q + r) \div 2\ .$$

There we have reached code.

7.4 Exercises

Ex. 7.1 Refine (iii) to code.

Ex. 7.2 ♡ Write out the code of the entire square-root program.

Ex. 7.3 Why can we assume $r + 1 < q$ in the precondition of (iv)? Would $r < q$ have been good enough? Why?

Ex. 7.4 ♡ Justify the branches (v) and (vi) of the alternation: where does $p < q$ come from in the precondition of (v)? Why does the postcondition of (vi) contain an *increasing* variant?

Ex. 7.5 Return to (ii) and make instead the refinement

$$\begin{array}{ll} \sqsubseteq & I \mathrel{\widehat{=}} r^2 \leq s \bullet \\ & r\!: [I \wedge s < (r+1)^2] \ . \end{array}$$

Refine that to code. Compare the efficiency of the result with the code of Exercise 7.2.

Ex. 7.6 Supply *all* the missing justifications and/or steps in this proof of *remove invariant* 7.1:

$$\begin{array}{ll} & w\!: [pre \ , \ inv \ , \ post] \\ = & w\!: [pre \wedge inv \ , \ inv \wedge post] \\ \sqsubseteq & w\!: [pre \wedge inv \ , \ post] \\ \sqsubseteq & w\!: [pre \ , \ post] \ . \end{array}$$

Where does the proof fail when *inv* contains w?