Invariants

Square root

Suppose x is a natural number. We wish to find the integer part of its square root.

$$\begin{aligned} f &= \left\langle y'^2 \le x < (y'+1)^2 \right\rangle \\ &= \left\langle y'^2 \le x \wedge x < (y'+1)^2 \right\rangle \end{aligned}$$

If we have already searched the first y natural numbers we know $y^2 \le x$.

Let's introduce that as an *loop invariant*

$$\mathcal{I} \text{ is } y^2 \le x$$

$$g = (\langle \mathcal{I} \rangle \Rightarrow f)$$

Apart from primes, the invariant is one conjunct of the original specification. We have

$$f \sqsubseteq y := 0; g$$

If we use the other conjunct (negated) as the loop guard we get

$$g \sqsubseteq \textbf{if } (y+1)^2 \le x \textbf{ then } g0 \textbf{ else } g1$$

where the else-clause is

$$g1 = \left\langle y^2 \le x < (y+1)^2 \Rightarrow y'^2 \le x < (y'+1)^2 \right\rangle$$

We have a specification of the form $\langle \mathcal{A} \Rightarrow \mathcal{B} \rangle$ such that $\mathcal{A}$ is the same as $\widetilde{\mathcal{B}}$.

By the erasure law, such a specification can always be implemented by **skip.**

For the "then" clause we have:

$$g0 = \left\langle \mathcal{I} \land (y+1)^2 \le x \Rightarrow y'^2 \le x < (y'+1)^2 \right\rangle$$

We need to find a specification h so that

$$g0 \sqsubseteq h; g$$

The following will do

$$h = \left\langle \mathcal{I} \land (y+1)^2 \le x \Rightarrow \mathcal{I}' \land x' = x \right\rangle$$

where $\mathcal{I}'$ is the same as $\mathcal{I}$, but with primes:

$$\mathcal{I}' \text{ is } y'^2 \le x'$$

The key point about h is that its job is to *preserve the invariant*.

$$h = \langle \mathcal{I} \land \cdots \Rightarrow \mathcal{I}' \land \cdots \rangle$$

That is, if it starts in a state where $\mathcal{I}$ holds, it must end in a state where $\mathcal{I}$ holds.

$$\begin{array}{ll} & h \\ = & \text{By definition} \\ & \left\langle \mathcal{I} \land (y+1)^2 \le x \Rightarrow \mathcal{I}' \land x' = x \right\rangle \\ \sqsubseteq & \text{Strengthening and assignment laws} \\ & y := y + 1 \end{array}$$

The method of loop invariants

Let's try to separate the *method* used to solve the last problem, from the details of the problem.

The general form of the solution is

$$\begin{aligned} f &\sqsubseteq m; g \\ g &\sqsubseteq \textbf{while}\ \mathcal{A}\ \textbf{do}\ h \end{aligned}$$

The key to the method is the notion of an invariant $\mathcal{I}$, which is a condition on the initial state.

- Suppose $f = \langle \mathcal{B} \rangle$ and $\mathcal{B}$ depends on a list of input variables $\bar{x}$.
- We generalize $f = \langle \mathcal{B} \rangle$ to $g = \langle \mathcal{I} \Rightarrow \mathcal{B} \rangle$
- The role of the initialization statement m, is to *establish the invariant*, that is to make $\mathcal{I}$ true at the start of the loop. This suggests
  $$\langle \mathcal{I}' \rangle$$
  However, that is not enough, m must not change any of the variables $\bar{x}$ whose initial value is used in $\mathcal{B}$.
  $$m = \langle \mathcal{I}' \wedge \bar{x}' = \bar{x} \rangle$$
- Exercise: Show that $f \sqsubseteq m; g$
- Now we turn our attention to implementing $g = \langle \mathcal{I} \Rightarrow \mathcal{B} \rangle$ with $\textbf{while}\ \mathcal{A}\ \textbf{do}\ h$.
- The negation of the guard $\mathcal{A}$ together with the invariant should ensure that $\mathcal{B}$ is implemented by **skip**. That is
  $$\mathcal{I} \wedge \neg\mathcal{A} \Rightarrow \widetilde{\mathcal{B}}\text{, is universally true}$$
  (We can often use this formula to decide on $\mathcal{I}$.)

- The role of the loop's body h is to *preserve the invariant*; that is
$$\langle \mathcal{I} \Rightarrow \mathcal{I}' \rangle$$
We can assume that $\mathcal{A}$ is true to start with, so h needs to implement
$$\langle \mathcal{A} \wedge \mathcal{I} \Rightarrow \mathcal{I}' \rangle$$
However, h should not change variables whose initial value is used in $\mathcal{B}$. We have
$$h = \langle \mathcal{A} \wedge \mathcal{I} \Rightarrow \mathcal{I}' \wedge \bar{x}' = \bar{x} \rangle$$
- To ensure termination, the loop's body should also decrease the bound.
- Exercise: show that $g \sqsubseteq \langle A \rangle \Rightarrow (h; g)$

More on Invariants

A slightly faster (and smaller) square root

Consider the square root problem

$$f = \left\langle y'^2 \leq x < (y'+1)^2 \right\rangle$$

and its solution

$$\begin{aligned} \mathcal{I} \quad & \text{is } y^2 \leq x \\ g \;=\; & (\langle \mathcal{I} \rangle \Rightarrow f) \\ f \;\sqsubseteq\; & y := 0; g \\ g \;\sqsubseteq\; & \mathbf{while}\ (y+1)^2 \leq x\ \mathbf{do}\ y := y+1 \end{aligned}$$

On many computers, multiplications are slow. In hardware, multipliers are large, slow, or both.

Do we really need to multiply?

Introduce a new variable z and redefine the invariant as

$$\mathcal{I} : y^2 \leq x \wedge z = (y+1)^2$$

As before define

$$g = (\langle \mathcal{I} \rangle \Rightarrow f)$$

Now we must change the two places where the invariant is established. Note that

$$(y+2)^2 = y^2 + 4y + 4 = (y+1)^2 + 2(y+1) + 1$$

We get

$$\begin{aligned} f \;\sqsubseteq\; & y, z := 0, 1; g \\ g \;\sqsubseteq\; & \mathbf{while}\ z \leq x \\ & \mathbf{do}\ y, z := y+1, z + 2(y+1) + 1 \end{aligned}$$

Although there is still a multiplication by 2, such a multiplication is fast in software and trivial in hardware. (Just shift the binary representation to the left.)

[Exercise: Do this derivation in detail.]

Square Root by Binary Search

The number of iterations required above is still $\lfloor\sqrt{x}\rfloor$.

[The notation $\lfloor a\rfloor$, for $a \in \mathbb{R}$, means the largest integer not larger than a. I.e. $i \le \lfloor a\rfloor$ iff $i \le a$, for all integers i. Equivalently we have $\lfloor a\rfloor < i$ iff $a < i$, for all integers i and real numbes a. (As an exercise, prove these these definitons are equivalent.)]

Can we do better?

Rewrite f as

$$\begin{aligned} f \; &= \; \left\langle y'^2 \le x < (y'+1)^2 \right\rangle \\ &= \; \text{Monotonicity of squaring nonnegative reals} \\ & \quad\; \left\langle y' \le \sqrt{x} < y'+1 \right\rangle \\ &= \; \text{Facts about floor} \\ & \quad\; \left\langle y' \le \left\lfloor\sqrt{x}\right\rfloor < y'+1 \right\rangle \end{aligned}$$

Invariant

We obtain (another) $\mathcal{I}$ by replacing the expession $y'+1$ with a new variable, z, and erasing primes.

$$\begin{aligned} \mathcal{I} \; &: \; y \le \left\lfloor\sqrt{x}\right\rfloor < z \\ g \; &= \; \left\langle \mathcal{I} \Rightarrow y' \le \left\lfloor\sqrt{x}\right\rfloor < y'+1 \right\rangle \end{aligned}$$

$$\boxed{\begin{array}{ccccccccc} 0 & \cdots & & \cdots & \lfloor\sqrt{x}\rfloor & \cdots & & \cdots & x+1 \\ & & \uparrow & & & & \uparrow & & \\ & & y & & & & z & & \end{array}}$$

In terms of sets $\mathcal{I} = (\lfloor\sqrt{x}\rfloor \in \{y, ..z\})$

Initialization

It is a fact that, for all $x \in \mathbb{N}$

$$0 \le \left\lfloor \sqrt{x} \right\rfloor < x + 1$$

So

$$f \sqsubseteq y, z := 0, x+1; g$$

$$\boxed{\begin{array}{ccccccc} & 0 & \cdots & \left\lfloor \sqrt{x} \right\rfloor & \cdots & & x+1 \\ \uparrow & & & & & \uparrow & \\ y & & & & & z & \end{array}}$$

Guard

When $z = y + 1$, then g is done

$$\left\langle z = y + 1 \wedge \mathcal{I} \Rightarrow y' \le \left\lfloor \sqrt{x} \right\rfloor < y' + 1 \right\rangle \sqsubseteq \mathbf{skip}$$

So use $z \ne y + 1$ as the loop guard. Given the invariant, this is the same as $z > y + 1$.

$$\boxed{\begin{array}{ccccccc} 0 & \cdots & & \left\lfloor \sqrt{x} \right\rfloor & & \cdots & x+1 \\ & & \uparrow & & \uparrow & & \\ & & y & & z & & \end{array}}$$

Preserving the invariant

We need a body that refines

$$h = \langle \mathcal{I} \wedge z > y + 1 \Rightarrow \mathcal{I}' \wedge x' = x \rangle$$

[Before looking at the next page, try to find such a body. Try to cut $z - y$ approximately in half in the body.]

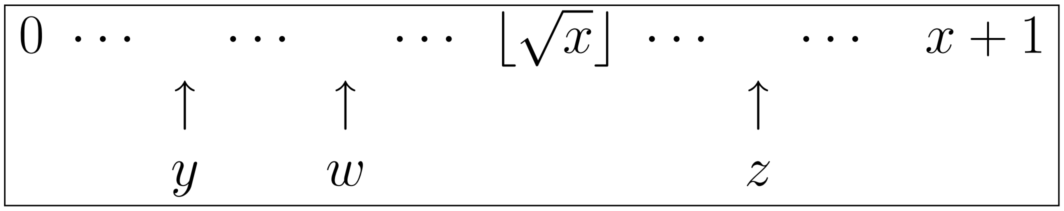
At the start of the body, $z - y$ is at least 2, so $\lfloor (z-y)/2 \rfloor$ is at least 1.

Let w stand for $y + \lfloor (z-y)/2 \rfloor$.

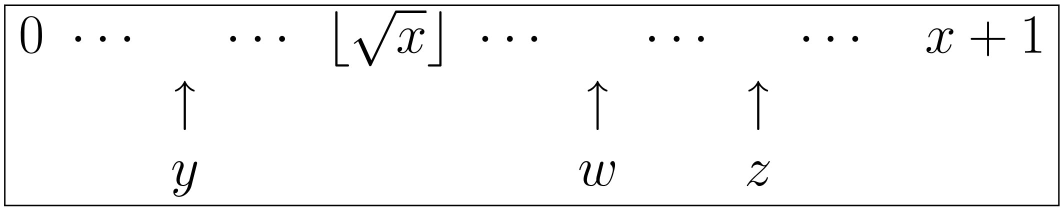
Note $y < w < z$. We have $\{y, ..z\} = \{y, ..w\} \cup \{w, ..z\}$

From $\mathcal{I}$ we have $\lfloor \sqrt{x} \rfloor \in \{y, ..w\} \vee \lfloor \sqrt{x} \rfloor \in \{w, ..z\}$

If $w \le \lfloor \sqrt{x} \rfloor$ then we can set y to w


$0 \cdots \quad \cdots \quad \cdots \quad \lfloor \sqrt{x} \rfloor \quad \cdots \quad \cdots \quad x+1$
y
w
z

If $w > \lfloor \sqrt{x} \rfloor$ then we can set z to w


$0 \cdots \quad \cdots \quad \lfloor \sqrt{x} \rfloor \quad \cdots \quad \cdots \quad \cdots \quad x+1$
y
w
z

In both cases the bound $z - y$ is decreased by at least one

We have

$$\begin{array}{ll} & \langle \mathcal{I} \wedge z > y+1 \Rightarrow \mathcal{I}' \wedge x' = x \rangle \\ \sqsubseteq & \mathbf{if}\ w \le \lfloor \sqrt{x} \rfloor \\ & \mathbf{then}\ y := w \\ & \mathbf{else}\ z := w \end{array}$$

[Exercise: check this in detail.]

But, we can't use the test $w \le \lfloor \sqrt{x} \rfloor$ as we can't calculate $\lfloor \sqrt{x} \rfloor$

(If we could easily calculate $\lfloor \sqrt{x} \rfloor$ there wouldn't be any point designing this agorithm.)

Can we find an equivalent expression?

$$\begin{aligned} & w \le \left\lfloor \sqrt{x} \right\rfloor \\ = \quad & \text{Since } w \text{ is an integer} \\ & w \le \sqrt{x} \\ = \quad & \text{Squaring both sides.} \\ & w^2 \le x \end{aligned}$$

The final algorithm is

```
    $f$
⊑   $y, z := 0, x + 1;$
    // inv. $y \le \left\lfloor \sqrt{x} \right\rfloor < z$
    **while** $z > y + 1$
    **do** **let** $w = y + \left\lfloor (z - y) / 2 \right\rfloor \cdot$
        **if** $w^2 \le x$
        **then** $y := w$
        **else** $z := w$
```

Binary Search

The invariant says the desired answer is in the set $\{y, y+1, ..., z-1\}$. We call this set the "search space"

In each iteration of the loop, we reduce the size of the search space by roughly 2. This technique is called *"binary search"*.

Since the size of the search space is roughly halved with each iteration, and the initial size of the search space is x, the number of iterations is roughly $\log_2 x$.

For large x, $\log_2 x$ is considerably smaller than $\sqrt{x}$. Consider $x = 10^{12} \cong 2^{40}$.$\sqrt{x} = 10^6$. $\log_2 x \cong 40$

Data Transformation

In both the above solutions we *augmented* the state space with a variable z and constrained the relationship of z to the other variables by strengthening the invariant.

This is an example of a **data transformation**. In a data transformation, we replace one set of variables with another while specifying an invariant relationship between the two state spaces.

In both examples we replaced $\{$“x”, “y”$\}$ with $\{$“x”, “y”,“z”$\}$ the added relationships being

$$z = (y+1)^2$$

and

$$y \le \left\lfloor \sqrt{x} \right\rfloor < z$$

respectively.

A challenge:

The Square Root by Binary Search algorithm above still has a multiplication operation in each iteration. For hardware implementation, this will use up considerable time and area, for software implementation, it uses time.

Can you eliminate the multiplication in the Square Root by Binary Search?

Hint: Use data transformation.

- Add one or more variables to track quantities that are expensive to calculate.
- Strengthen the invariant to indicate the relationship between these variables and the quantities they track.

A Nondeterministic ‘if’ Statement.

Before we go on, I’d like to introduce a new kind of ‘if’ statement.

$$\begin{array}{l}\textbf{if } \mathcal{A} \textbf{ then } f \\ \Box\ \mathcal{B} \textbf{ then } g \\ \textbf{else } h\end{array}$$

This means

- if $\mathcal{A}$ is true and $\mathcal{B}$ is false, execute f
- if $\mathcal{A}$ is false and $\mathcal{B}$ is true, execute g
- if both $\mathcal{A}$ and $\mathcal{B}$ are true, execute either f or g
- if neither are true, execute h.

The semantics is

$$(\langle\mathcal{A}\rangle \wedge f) \vee (\langle\mathcal{B}\rangle \wedge g) \vee (\langle\neg\mathcal{A} \wedge \neg\mathcal{B}\rangle \wedge h)$$

As long as $\mathcal{A} \vee \mathcal{B}$ is universally true, we can leave out the else part.

Furthermore, in that case, we have

$$\textbf{if } \mathcal{A} \textbf{ then } f \ \Box\ \mathcal{B} \textbf{ then } g \sqsubseteq \textbf{if } \mathcal{A} \textbf{ then } f \textbf{ else } g$$

An Abstract Binary Search algorithm

We can abstract away from the particulars of the square root problem to obtain a general seach problem. Suppose G is a constant, nonempty, “goal set”. (In our square root application G is $\{\lfloor\sqrt{x}\rfloor\}$.)

By *constant*, I mean that it does not depend on any variables that are changed.

We wish to find at least one member of the goal set.

$$f = \langle S' \subseteq G \wedge S' \neq \emptyset\rangle$$

We can abstract away from the particulars of the solution to the square root problem to obtain an *Abstract Binary Search* algorithm. The key is the invariant, which says that a set variable S always contains at least one member of the goal set.

```
$\mathcal{I}$ : $S \cap G \neq \emptyset$ $\wedge$ $S$ fin
$g = \langle\mathcal{I}\rangle \Rightarrow f$
$f \sqsubseteq S :=$ some finite set such that $S \cap G \neq \emptyset$ ; $g$
$g \sqsubseteq$ while $|S| > 1$
        do let $S_0, S_1 \mid S_0 \cup S_1 = S \wedge S_0 \subseteq S \wedge S_1 \subseteq S$.
            if $S_0 \cap G \neq \emptyset$ then $S := S_0$
            $\square$ $S_1 \cap G \neq \emptyset$ then $S := S_1$
```

Note that either $S_0 \cap G \neq \emptyset$ or $S_1 \cap G \neq \emptyset$ or both since

$$
\begin{aligned}
& S_0 \cap G \neq \emptyset \vee S_1 \cap G \neq \emptyset \\
= \;& \text{De Morgan} \\
& \neg\left((S_0 \cap G = \emptyset) \wedge (S_1 \cap G = \emptyset)\right) \\
= \;& \text{Since } A = \emptyset \wedge B = \emptyset \text{ iff } A \cup B = \emptyset \\
& \neg\left((S_0 \cap G) \cup (S_1 \cap G) = \emptyset\right) \\
= \;& \text{Distributivity} \\
& \neg\left((S_0 \cup S_1) \cap G = \emptyset\right) \\
= \;& \text{Since } S_0 \cup S_1 = S \\
= \;& \neg\left(S \cap G = \emptyset\right) \\
& \text{Definition of } \mathcal{I} \\
& \mathcal{I}
\end{aligned}
$$

The loop bound is the size of S, so we should ensure that both S_0 and S_1 are smaller than S. For efficiency it is best if S_0 and S_1 are disjoint ($S_0 \cap S_1 = \emptyset$) and approximately the same size. In that case the loop will iterate $\Theta(\log |S|)$ times.

Applying the Abstract Binary Search Algorithm

The Square Root by Binary Search algorithm can be obtained from the Abstract Binary Search Algorithm by the following data transform

$$
\begin{aligned}
G &= \left\{\left\lfloor \sqrt{x} \right\rfloor\right\} \\
S &= \{y, ..z\} \\
S_0 &= \{y, ..w\} \\
S_1 &= \{w, ..z\} \\
\text{where } w &= y + \lfloor (z - y)/2 \rfloor
\end{aligned}
$$