Choosing invariants and guards

(This slide set is based on work by David Gries.)

When using the method of invariants to develop a loop we have

$\langle\mathcal{B}\rangle$ the specification to be implemented

$\mathcal{I}$ the invariant condition

$\langle\mathcal{I} \Rightarrow \mathcal{B}\rangle$ the specification of the loop

$\langle\mathcal{B}\rangle \sqsubseteq m; \langle\mathcal{I} \Rightarrow \mathcal{B}\rangle$ where m establishes the invariant

$\mathcal{A}$ is the loop guard (another condition)

$\mathcal{I} \wedge \neg\mathcal{A} \Rightarrow \widetilde{\mathcal{B}}$ so that $\langle\mathcal{I} \wedge \neg\mathcal{A} \Rightarrow \mathcal{B}\rangle \sqsubseteq \mathbf{skip}$

$\langle\mathcal{I} \Rightarrow \mathcal{B}\rangle \sqsubseteq \mathbf{while}\ \mathcal{A}\ \mathbf{do}\ h$ where h the loop body

h must reestablish the invariant

starting in a state such that $\mathcal{I} \wedge \mathcal{A}$

h must decrease some bound expression $\mathcal{E}$

Deleting a conjunct — square root revisited

When $\mathcal{B}$ is a conjunction $\mathcal{B}_0 \wedge \mathcal{B}_1$ we can take one conjunct of $\widetilde{\mathcal{B}}$ as the invariant $\mathcal{I}$ and the negation of the remaining conjunct as the guard $\mathcal{A}$.

$\mathcal{I}$ should be easy to establish in the first place and $\mathcal{A}$ should be easy to check.

Example. In the square root example we had $y'^2 \le x < (y'+1)^2$ as $\mathcal{B}$.

This has two conjuncts $y'^2 \le x$ and $x < (y'+1)^2$.

Take $y^2 \le x$ as $\mathcal{I}$ and $x < (y+1)^2$ as $\neg\mathcal{A}$.

It is easy to establish $\mathcal{I}$ by setting y very low.

> $y := 0$;
> // Inv. $y^2 \le x$
> $\mathbf{while}\ x \ge (y+1)^2\ \mathbf{do}$
>   given $x \ge (y+1)^2$, reduce $x - (y+1)^2$
>   while maintaining $y^2 \le x$

Replacing a constant by a variable — summation redone

Let's revisit summing an array. Let n be constant and a and array $\{0, ..n\} \to \mathbb{R}$

$$\mathcal{B} \text{ is } \left\langle s' = \sum_{k \in \{0,..n\}} a(k) \right\rangle$$

We can replace the constant n with a variable i and erase primes (we also put a range on the variable) to get

$$\mathcal{I} \text{ is } \left\langle 0 \le i \le n \wedge s = \sum_{k \in \{0,..i\}} a(k) \right\rangle$$

Now $g =$

$$\left\langle 0 \le i \le n \wedge s = \left(\sum_{k \in \{0,..i\}} a(k) \right) \Rightarrow s' = \left(\sum_{k \in \{0,..n\}} a(k) \right) \right\rangle$$

The $\neg\mathcal{A}$ needs to say that we have got back to $\mathcal{B}$. That is $\mathcal{A}$ is $i \ne n$.

Now $\mathcal{I}$ is easy to establish with $i, s := 0, 0$. Our program so far is

$i, s := 0, 0$;
// Inv. $0 \le i \le n \wedge s = \sum_{k \in \{0,..i\}} a(k)$
$\textbf{while } i \ne n \textbf{ do}$
given $i \ne n$, reduce $n - i$ while maintaining

$$0 \le i \le n \wedge s = \sum_{k \in \{0,..i\}} a(k)$$

Exercise: Show that $s, i := s + a(i), i + 1$ reestablishes the invariant while reducing the bound.

The bound is $n - i$.

Note. We could equally well have replaced the 0 this would give

$$\mathcal{I} \text{ is } \left\langle 0 \le i \le n \wedge s = \sum_{k \in \{i,..n\}} a(k) \right\rangle$$

$\mathcal{A}$ is $i \neq 0$, $m = (i, s := n, 0)$, and $h = (s, i := s + a(i-1), i - 1)$

Replacing an expression by a variable — general binary search

The expression we replace with a variable, need not be a constant.

A rising edge is a point k where $a(k-1) = \mathfrak{false}$ but $a(k) = \mathfrak{true}$.

Consider searching a boolean sequence a for an rising edge.

To ensure there is a rising edge, we'll assume $a(-1) = \mathfrak{false}$ and $a(n) = \mathfrak{true}$.

We will assume that a has domain $\{-1, .., n\}$.

Aside: If it is also known that a has one edge, then the final value of k will be the minimum such that $a(k)$. A result of $k = 0$ means $a(j)$, for all $j \in \{0, ..n\}$, while a result of $k = n$ means $\neg a(j)$, for all $j \in \{0, ..n\}$. There is one edge when a is monotone

$$j \le i \Rightarrow (a(i) \Rightarrow a(j)) \text{, for all } i, j \in \{0, ..n\}$$

(end aside.)

Our specification is $\langle \neg a(k' - 1)) \wedge a(k') \wedge 0 \le k' \le n \rangle$.

To form the invariant, erase primes and replace the expression $k - 1$ with a variable i.

We have

$$\mathcal{I} \text{ is } \neg a(i) \wedge a(k) \wedge -1 \le i < k \le n$$

which means there is a rising edge somewhere in the interval $\{i + 1, .., k\}$.

This invariant is easy to establish with $i, k := -1, n$.

When $i = k - 1$, we have $\neg a(k-1) \wedge a(k)$ and we are done, so $\mathcal{A}$ is $i < k - 1$.

The bound is the size of the interval. Our algorithm so far is

$i, k := -1, n$;
// Inv. $\neg a(i) \wedge a(k) \wedge -1 \leq i < k \leq n$
while $i < k - 1$ **do**
  given $i < k - 1$, reduce $k - i$ while maintaining
    $\neg a(i) \wedge a(k) \wedge -1 \leq i < k \leq n$
// $\neg a(i) \wedge a(k) \wedge k = i + 1$

Aside: In the case where there is one edge, we also know that at the end

$$\begin{aligned} i &= \max \{ j \in \{-1, ..n\} \mid \neg a(j) \} \\ k &= \min \{ j \in \{0, .., n\} \mid a(j) \} \end{aligned}$$

Exercise: Find a body that gives the most efficient algorithm possible.

Enlarging the range of a variable — general linear search

Consider searching a boolean sequence again.

This time we are seeking the first true value. I.e. we wish to compute

$$\langle k' = \min \{j \in \{0, .., n\} \mid a(j)\}\rangle$$

Again we assume $a(n)$, so that the problem is well defined.

Let $m = \min \{j \in \{0, .., n\} \mid a(j)\}$ .We have

$$\langle k' = m\rangle$$

or

$$\langle m \leq k' \leq m\rangle$$

To find the invariant, we erase primes and enlarge the range of k so that the invariant is easy to establish. Let $\mathcal{I}$ be

$$0 \leq k \leq m$$

Note that $0 \leq m$. Thus $k := 0$ establishes the invariant.

For any i, if $a(i)$ is true then $m \leq i$.

If the invariant is true and so is $a(k)$ then we are done as we have both

$k \ \leq \ m$ from the invariant and

$m \ \leq \ k$ from $a(k)$

And so let $\mathcal{A}$ be $\neg a(k)$.

The code so far is

$k := 0$;
// Inv. $\langle 0 \leq k \leq m \rangle$, where

$$m = \min \{j \in \{0, .., n\} \mid a(j)\}$$

$\mathbf{while}\ \neg a(k)\ \mathbf{do}$
given $\neg a(k)$, reduce $m - k$ while maintaining

$$0 \leq k \leq m$$

If $k \leq m = \min \{j \in \{0, .., n\} \mid a(j)\}$, then either $a(k)$ or $k + 1 \leq m$. Since we've ruled out $a(k)$, we know $k + 1 \leq m$ and that means $k := k + 1$ reestablishes the invariant.

Applying algorithmic schemes

Linear search

The last algorithm we developed is called *the linear search scheme*.

We'll assume $a(n) = \mathfrak{true}$ and m is $\min \{j \in \{0, .., n\} \mid a(j)\}$

$$\begin{array}{ll} & \langle k' = m \rangle \\ \sqsubseteq & \\ & k := 0 \, ; \\ & \text{// Inv. } \langle 0 \leq k \leq m \rangle, \\ & \mathbf{while}\ \neg a(k)\ \mathbf{do}\ k := k + 1 \end{array}$$

We can apply data transformation to use this scheme to solve many different problems.

For example if we are searching an array $b \in \{0, ..n\} \to S$ for some item $x \in S$.

Our problem is $\langle k' = m \rangle$ where

$$m = \min \{j \in \{0, .., n\} \mid j = n \vee b(j) = x\}$$

Define $a(j) = (j = n \vee b(j) = x)$, for all $j \in \{0, .., n\}$ so that

$$m = \min \{j \in \{0, .., n\} \mid a(j)\}$$

Now

$\langle k' = m \rangle$

$\sqsubseteq$ "Linear search scheme"

$k := 0\ ;$

// Inv. $\langle 0 \leq k \leq m \rangle$

$\mathbf{while}\ \neg a(k)\ \mathbf{do}\ k := k + 1$

$\sqsubseteq$ "Definition of a"

$k := 0\ ;$

// Inv. $\langle 0 \leq k \leq m \rangle$,

$\mathbf{while}\ k \neq n \wedge b(k) \neq x\ \mathbf{do}\ k := k + 1$

Binary Search

The algorithm we developed to illustrate replacing an expression with a variable is the *binary search scheme*

Let $a(-1) = \mathfrak{false}$ and $a(n) = \mathfrak{true}$.

$$\langle \neg a(k'-1)) \wedge a(k') \wedge 0 \le k' \le n \rangle$$

$$\sqsubseteq$$

$i, k := -1, n$;
// Inv. $\neg a(i) \wedge a(k) \wedge -1 \le i < k \le n$
while $i < k - 1$ **do**
    let $j \in \{0, ..n\} \mid j = \left\lfloor \frac{i+k}{2} \right\rfloor \cdot$
    // $i < j < k$
    if $a(j)$ **then** $k := j$ **else** $i := j$

We will apply it to solve a classic search problem.

Suppose b is a monotone function on $\{0, ..n\}$, i.e.

$i \le j \Rightarrow b(i) \le b(j)$, for all $i, j \in \{0, ..n\}$

We are seeking the smallest value of k such that $b(k) \ge x$, if there isn't one, the answer is n.

Define

$a(j) = (j = n \vee j \ge 0 \wedge b(j) \ge x)$, for all $j \in \{-1, .., n\}$

Note that $a(-1) = \mathfrak{false}$ and $a(n) = \mathfrak{true}$ and there is exactly one edge.

Thus the binary search scheme finds the smallest k such that $a(k)$ is true.

$\langle k' = \min \{j \in \{0, .., n\} \mid j = n \vee b(j) \geq x\}\rangle$

$\sqsubseteq$ "Definition of a"

$\langle k' = \min \{j \in \{0, .., n\} \mid a(j)\}\rangle$

$\sqsubseteq$ "There is one edge"

$\langle \neg a(k' - 1) \wedge a(k') \wedge 0 \leq k' \leq n\rangle$

$\sqsubseteq$ "Binary search scheme"

$i, k := 0, n$;

// Inv. $\neg a(i) \wedge a(k) \wedge -1 \leq i < k \leq n$

while $i < k - 1$ **do**

  let $j \in \{0, ..n\} \mid j = \left\lfloor \frac{i+k}{2} \right\rfloor \cdot$

  // $i < j < k$

  if $a(j)$ **then** $k := j$ **else** $i := j$

$\sqsubseteq$ "Definition of a"

$i, k := 0, n$;

// inv. $\begin{pmatrix} & (i = -1 \vee b(i) < x) \\ \wedge & (k = n \vee b(k) \geq x) \\ \wedge & -1 \leq i < k \leq n \end{pmatrix}$

while $i < k - 1$ **do**

  let $j \in \{0, ..n\} \mid j = \left\lfloor \frac{i+k}{2} \right\rfloor \cdot$

  // $i < j < k$

  if $b(j) \geq x$ **then** $k := j$ **else** $i := j$