Loop Invariant Proofs

CS 234

April 9, 2025

0 Introduction

This document contains examples of good proofs for the loop invariant problems in class. These are not the only ways to write good proofs.

These proofs may contain footnotes explaining different thought processes that occurred in their construction, to help show you how to think about writing proofs. Commentary may also be provided at the end about alternative approaches.

1 Proofs

Insertion

```
1     def ins(i, lst):
2         val = lst[i]
3         j = i - 1
4         while j >= 0 and lst[j] > val:
5             lst[j+1] = lst[j]
6             j = j - 1
7         lst[j+1] = val
```

Theorem 1. *For all nonempty lists of integers* `lst` *and all integers* `i` *such that* $0 \leq i < \texttt{len}(\texttt{lst})$*, if* $\texttt{lst}[0:\texttt{i}]$ *is ascendingly sorted, then calling* $\texttt{ins}(\texttt{i}, \texttt{lst})$ *mutates* `lst` *to have all the same elements and also have* $\texttt{lst}[0:\texttt{i}+1]$ *ascendingly sorted.*

Proof. Let `lst` be some nonempty list of integers and `i` be some integer between 0 and $\texttt{len}(\texttt{lst})$. Further, let $\texttt{lst}[0:\texttt{i}]$ be ascendingly sorted.

The desired property is proven using the following loop invariant $P(n)$ which is defined as the conjunction of the following propositions where the values `i`, `j`, `val`, and `lst` are from just before iteration n (0-indexed), and where double-primed variables are from just before the first iteration.[1]

1. $\texttt{lst}[0:\texttt{j}+1] = \texttt{lst}''[0:\texttt{j}+1]$
2. $\texttt{lst}[\texttt{j}+2:\texttt{i}''+1] = \texttt{lst}''[\texttt{j}+1:\texttt{i}'']$
3. $\texttt{lst}[\texttt{i}''+1:] = \texttt{lst}''[\texttt{i}''+1:]$
4. every element of $\texttt{lst}[\texttt{j}+2:\texttt{i}''+1]$ is strictly greater than `val`
5. $\texttt{val} = \texttt{val}''$
6. $\texttt{j} < \texttt{i}''$

[1] I've chosen to be a bit verbose with this invariant writeup because I want to make it clear to you exactly which relations hold between the variables before and after the loop iterates once. Conditions 1, 3, and 5 all just state that these parts of the program do not change, and 6 just states that $\texttt{j}+1$ is a valid index, so 2 and 4 are the only interesting propositions. Had you elided 5, I would not care, as it is obviously not altered by the loop body. Proposition 6 is slightly more important, but still fine to leave out because $\texttt{j}+1$ pretty clearly starts in range and then shrinks. Propositions 1 and 3 are slightly more important still because the code does alter the list, and in fact proposition 1 plays a mildly important role in the proof. Proposition 3 could therefore be begrudgingly elided, proposition 1 should be kept. So don't be afraid—I would only expect you to find 1, 2, and 4, and honestly only 2 and 4 are really important. Most (if not all) loop invariants in this class will be easier too.

The only part of the program that is not constrained by this predicate is $\texttt{lst}[\texttt{j}+1]$ and `i`. For the former, this is because it does not matter what is at this index of the list; it will be overwritten immediately in line 5. For the latter, this is because no part of the code depends on `i` after line 3 (though the original value is still useful for phrasing our invariants).

Initialization From the start of the function until the guard is first checked in line 4, all the code does is set `val` to $\texttt{lst}[\texttt{i}'']$ and `j` to $\texttt{i}''-1$ in lines 2 and 3, respectively.

At this point, propositions 1, 3, and 5 of $P(0)$ are satisfied simply because the primed and un-primed variables both refer to the same point in the code and so are the same. Proposition 6 also holds because $j+1=i''$. All that remains to be shown are propositions 2 and 4.

Proposition 2 is confirmed via the following identity:

$$\begin{aligned}
\texttt{lst}[\texttt{j}+2:\texttt{i}''+1] =& \texttt{lst}[\texttt{i}''+1:\texttt{i}''+1] && [\texttt{j}=\texttt{i}''-1]\\
=&[] && [\textit{slicing}]\\
=&\texttt{lst}''[\texttt{i}'':\texttt{i}''] && [\textit{slicing}]\\
=&\texttt{lst}''[\texttt{j}''+1:\texttt{i}''] && [\texttt{j}=\texttt{i}''-1]
\end{aligned}$$

The above identity also confirms that $\texttt{lst}[\texttt{j}+2:\texttt{i}''+1]$ is empty, so proposition 4 holds vacuously.

Maintenance Suppose that $P(n)$ holds for when checking the loop guard for the n^{th} time and the loop guard continues to be satisfied such that an n^{th} iteration will occur. Let the original values of variables at the start of this iteration be indicated with primes.

The new iteration then proceeds to set $\texttt{lst}[\texttt{j}'+1]$ to $\texttt{lst}'[\texttt{j}']$ in line 5. (These indices are in bounds because $\texttt{j}'\geq 0$ from the guard, $\texttt{j}'<i$ by proposition 6, and $i<\texttt{len}(\texttt{lst})$ by assumption.)[2] Then the iteration sets `j` to $\texttt{j}'-1$ in line 6.

Propositions 1, 3, and 5 of $P(n+1)$ hold simply because these values are not touched by the code. Proposition 6 holds because $\texttt{j}<\texttt{j}+1=\texttt{j}'<\texttt{i}''$. All that remains to be shown are propositions 2 and 4.[3]

By proposition 2 of $P(n)$, we know that $\texttt{lst}[\texttt{j}'+2:\texttt{i}''+1]=\texttt{lst}''[\texttt{j}'+1:\texttt{i}'']$. Substituting $\texttt{j}=\texttt{j}'-1$ in, we find $\texttt{lst}[\texttt{j}+3:\texttt{i}''+1]=\texttt{lst}''[\texttt{j}+2:\texttt{i}'']$. This identity is almost all that needs to be shown to satisfy proposition 2 of $P(n+1)$, which is that $\texttt{lst}[\texttt{j}+2:\texttt{i}''+1]=\texttt{lst}''[\texttt{j}+1:\texttt{i}'']$. The only additional fact that must be shown is that $\texttt{lst}[\texttt{j}+2]=\texttt{lst}''[\texttt{j}+1]$. This fact can be verified as follows:[4]

$$\begin{aligned}
\texttt{lst}[\texttt{j}+2] =& \texttt{lst}[\texttt{j}'+1] && [\texttt{j}=\texttt{j}'-1]\\
=&\texttt{lst}'[\texttt{j}'] && [\textit{line } 5]\\
=&\texttt{lst}''[\texttt{j}'] && [P(n)\ \textit{prop.}\ 1]\\
=&\texttt{lst}''[\texttt{j}+1] && [\texttt{j}=\texttt{j}'-1]
\end{aligned}$$

[2]This is a bit more detailed than needed, but it should explain why we might care to track all these less-important properties.

[3]Look how easy it was to dispatch these parts of the invariant. Even though the invariant has a lot of parts, most of them are super easy and we can get down to the few interesting parts quickly.

[4]This is the key point where proposition 1 plays a role.

Termination Suppose that the loop guard is no longer satisfied after checking the loop guard for n^{th} time so that no more iterations will occur. As the initialization and maintenance steps inductive show that $P(k)$ at line 4 for all iteration counts k, it must be that $P(n)$ holds.[5] Then all parts of $P(n)$'s definition hold and either $\texttt{j} < 0$ or $\texttt{lst}[\texttt{j}] \leq \texttt{val}$.[6]. In fact, in the latter case, $\texttt{j} = -1$ since it was only ever decremented 1 at a time. Paired with proposition 6, this leaves $\texttt{j} + \texttt{1}$ a valid index of `lst`.

We now show that `lst` has all the same elements as $\texttt{lst}''$ after the program finishes. By proposition 5 of $P(n)$, we find that `val` is $\texttt{val}''$, which by line 2 is just $\texttt{lst}''[\texttt{i}'']$. Thus, $\texttt{lst}[\texttt{j}+\texttt{1}] = \texttt{lst}''[\texttt{i}'']$ when the final line 7 executes. Moreover, since $\texttt{lst}[\texttt{0} : \texttt{j}+\texttt{1}] = \texttt{lst}''[\texttt{0} : \texttt{j}+\texttt{1}]$ by proposition 1, $\texttt{lst}[\texttt{j}+\texttt{2} : \texttt{i}''+\texttt{1}] = \texttt{lst}''[\texttt{j}+\texttt{1} : \texttt{i}'']$ by proposition 2, and $\texttt{lst}[\texttt{i}''+\texttt{1} :] = \texttt{lst}''[\texttt{i}''+\texttt{1} :]$ by proposition 3, it follows that all elements of the original list $\texttt{lst}''$ are present in `lst`. All that remains to be shown is that the range $\texttt{lst}[\texttt{0} : \texttt{i}''+\texttt{1}]$ is ascendingly sorted.

Finally, we proceed by cases on how the loop guard became unsatisfied:

Case $\texttt{j} = -1$ Suppose that $\texttt{j} = -1$. Proposition 4 then tells us that every element in $\texttt{lst}[\texttt{j}+\texttt{2} : \texttt{i}''+\texttt{1}] = \texttt{lst}[\texttt{1} : \texttt{i}''+\texttt{1}]$ is greater than `val`. Thus, setting $\texttt{lst}[\texttt{j}+\texttt{1}] = \texttt{lst}[\texttt{0}]$ to `val` leaves the whole range $\texttt{lst}[\texttt{0} : \texttt{i}''+\texttt{1}]$ ascendingly sorted.

Case $\texttt{lst}[\texttt{j}] \leq \texttt{val}$ Suppose that $\texttt{lst}[\texttt{j}] \leq \texttt{val}$. Then, because $\texttt{lst}''[\texttt{0} : \texttt{i}'']$ was assumed sorted, $j < i''$ by proposition 6, and $\texttt{lst}[\texttt{0} : \texttt{j}+\texttt{1}] = \texttt{lst}''[\texttt{0} : \texttt{j}+\texttt{1}]$ by proposition 1, it must be that all elements of $lst[0 : j+1]$ are no greater than `val`. Proposition 4 then tells us that every element in $\texttt{lst}[\texttt{j}+\texttt{2} : \texttt{i}''+\texttt{1}]$ is greater than `val`. Thus, setting $\texttt{lst}[\texttt{j}+\texttt{1}]$ to `val` puts it in a sorted position, leaving the whole range $\texttt{lst}[\texttt{0} : \texttt{i}''+\texttt{1}]$ ascendingly sorted.

□

[5] I have varied my wording here to better emphasize how the first parts of the proof really are just induction.

[6] Hey, we negate the proposition here!

Sorting

```
1       def isort(lst):
2           for i in range(len(lst)):
3               ins(i, lst)
```

Theorem 2. *For any list of integers* `lst`*, calling* `isort(lst)` *sorts the list in place.*

Proof. Let `lst` be some list of integers.

The desired property is proven using the following loop invariant $Q(n)$ which is defined as the conjunction of the following propositions where the variables `lst` and `i` come from just prior to the n^{th} iteration of the loop, and the variable $\texttt{lst}''$ refers to the original input list.

1. $\texttt{i} \geq 0$
2. $\texttt{lst}[\texttt{0} : \texttt{i}]$ is ascendingly sorted
3. `lst` contains all the same elements as $\texttt{lst}''$

Initialization From the start of the function until the loop guard is first checked in line 2, all the code does is initialize a variable `i` to 0 via line 2's for-loop.

At this point, proposition 1 of $Q(0)$ is satisfied because $\texttt{i} = 0$, proposition 2 is satisfied because $\texttt{lst}[\texttt{0} : \texttt{i}] = \texttt{lst}[\texttt{0} : \texttt{0}] = []$ which is sorted, and proposition 2 of $Q(0)$ is satisfied because $\texttt{lst} = \texttt{lst}''$.

Maintenance Suppose that $Q(n)$ holds for when checking the loop guard for the n^{th} time and the loop guard continues to be satisfied such that an n^{th} iteration will occur. Let the original values of variables at the start of this iteration be indicated with primes. Then all parts of $Q(n)$'s definition hold and $\texttt{i}' < \texttt{len}(\texttt{lst}')$.[7]

Proposition 1 of $Q(n)$ lets us further conclude that $0 \leq \texttt{i}' < \texttt{len}(\texttt{lst}')$, and $\texttt{len}(\texttt{lst}') = \texttt{len}(\texttt{lst})$ since proposition 3 tells us they have the same elements. In addition, proposition 2 of $Q(n)$ tells us that $\texttt{lst}'[\texttt{0} : \texttt{i}']$ is ascendingly sorted. Together, these facts meet the preconditions of Theorem 1. Thus, when calling $\texttt{ins}(\texttt{i}', \texttt{lst}')$ in line 3, we know $\texttt{lst}'$ is mutated so that $\texttt{lst}[\texttt{0} : \texttt{i}' + \texttt{1}]$ is ascendingly sorted. This satisfies proposition 2 of $Q(n + 1)$.

We also know that `lst` is left with the same elements as $\texttt{lst}'$ from Theorem 1. In turn $\texttt{lst}'$ has the same elements as $\texttt{lst}''$ by proposition 3 of $Q(n)$. Putting these facts together, proposition 3 of $Q(n + 1)$ is satisfied.

Finally, before the loop guard is checked for the $(n + 1)^{th}$ time, `i` is incremented. Thus $i > i - 1 = i' \geq 0$ by proposition 1 of $Q(n)$, so proposition 1 of $Q(n + 1)$ is also satisfied.

[7] Remember, $\texttt{foriinrange(n)}$ is just shorthand for $\texttt{i} = \texttt{0}$ followed by $\texttt{whilei} < \texttt{n}$ with $\texttt{i} = \texttt{i} + \texttt{1}$ included at the end of the loop body.

Termination The for-loop terminates when $\texttt{i} \geq \texttt{len}(\texttt{lst})$. In particular, it must be that $\texttt{i} = \texttt{len}(\texttt{lst})$ because `i` is only incremented by 1 each time.

Because $Q(k)$ holds before every check of the loop guard, all parts of its definition must hold. Substituting $\texttt{i} = \texttt{len}(\texttt{lst})$ into proposition 2 tells us that $\texttt{lst}[\texttt{0} : \texttt{len}(\texttt{lst})] = \texttt{lst}$ is ascendingly sorted. In addition, proposition 3 tells us that `lst` has all the same elements as $\texttt{lst}''$. Thus, the list is sorted in place.

□