CS 281
Prof. Alex Pothen
Webbook:

www.cs.odu.edu/~webbook
www.cs.odu.edu/~pothen/Courses/CS281
CLASS NOW IN CONSTANT 120
Mon Oct 19, 1998
Lecture 14
(Lecture 12—Review
Lecture 13—Midterm 1)
Sec. 3.2 Induction, Sec 3.5
Program Correctness
Next Lecture: Recursive Algorithms
Induction

- To show that a statement $S(n)$ is true for all nonnegative integers $n \geq n_0$.

- Prove the basis case directly: Usually $S(0)$ or $S(1)$. The basis case could also be $S(k)$ for some integer $k$, when $S(n)$ is true for $n \geq k$.

- The inductive step: Here we prove that for all $n \geq k$, $S(n)$ implies $S(n+1)$. We assume that $S(n)$ is true: this is the inductive assumption. Then we show that $S(n+1)$ is true by making use of the assumption that $S(n)$ is true.
Strong Induction

• To show that a statement $S(n)$ is true for all nonnegative integers $n \geq a + k$.

• Prove the $(k + 1)$ basis cases $S(a)$, $S(a + 1)$, ..., $S(a + k)$ for some integers $a$ and $k$ directly.

• The inductive step: Here we prove that for all $n \geq (a + k)$, the statement $S(n - k)$ implies $S(n + 1)$. We assume $S(a)$, ..., $S(n)$ is true: this is the inductive hypothesis. Then we show that $S(n + 1)$ is true by making use of the assumption that these previous statements are true.

• More generally, we make use of some or all of the statements $S(a)$, $S(a + 1)$, ..., $S(a + k)$ to prove that $S(n + 1)$ is true.
Strong Induction

Problem: Suppose the only coins in a country are 3 and 5 cent coins. What amounts of money can we make change for?

- Small cases: can make 3, 5, 6, 8, 9, 10, 11 ... cents.

- Guess: can make change for \( n \) cents, where \( n \geq 8 \).

- Base cases: 8, 9, and 10 cents.

- Inductive hypothesis: true for all \( n \geq 10 \) cents.

- Inductive step: Make change for \( (n + 1) \) cents.
• Solution: First make change for \((n - 2)\) cents, which we can do by ind. hyp. Then add a 3 cent coin.
\((n - 2) + 3 = (n + 1)\).
Correctness of Programs

- To prove that a program segment $S$ does what it is intended to do, we need:
  - an initial assertion $p$, true of the input;
  - a final assertion $q$, true of the output.

- Hoare triple: $p \{S\} q$
  If $p$ is true for the input, and the program $S$ terminates, then $q$ is true for the output.

Example:

\[ \{p : a = 1\} \]
\[ b := 2;\]
\[ c := a + b;\]
\[ \{q : c = 3\} \]
Selection

if \( x < 0 \) then

abs := \(-x\);

else

abs := \(x\);

endif

initial assertion \( p: T \) (true)
final assertion \( q: abs = |x| \)
Selection II

Final Assertion:
After the if-then-else statement is executed, abs contains the absolute value of x.

When $x < 0$, then the $\langle$condition$\rangle$ is true, the $\langle$if-part$\rangle$ is executed, and $abs$ is assigned $-x$. Since $-x$ is positive, we have $abs = |x|$.

When $x \geq 0$, then the $\langle$condition$\rangle$ is false, and the $\langle$else-part$\rangle$ is executed. Thus $abs$ is assigned $x$, which is nonnegative. Again, we have $abs = |x|$.
Selection III

\[
\begin{align*}
\textbf{if} & \quad \langle \text{(condition)} \rangle \\
& \quad S_1 \langle \text{if-part} \rangle \\
\textbf{else} & \\
& \quad S_2 \langle \text{else-part} \rangle \\
\textbf{endif}
\end{align*}
\]

To prove that the final assertion \( q \) is correct, need to show

(1) \( p \textbf{ and } \langle \text{(condition)} \rangle \{ S_1 \} q \), and
(2) \( p \textbf{ and } \langle \neg \text{ condition} \rangle \{ S_2 \} q \).
Loop Invariants

- A technique for showing that a loop does what it is claimed to do.

- A loop invariant is a statement $S$ that is true each time we begin to execute the loop. We prove that $S$ is true by induction on the loop index (or some variable related to the loop index).

- Find the loop invariant by considering what happens as the loop is executed a few times.

- For loops terminate when the loop index becomes greater than the final value. We need to show that while loops terminate.
Factorial Algorithm

\[ n! = 1 \times 2 \times \cdots \times (n - 1) \times n, \quad \text{for } n \geq 1. \]

1. \( \text{fact} := 1; \ \text{number} := 2; \)
2. \( \text{while} \quad (\text{number} \leq n) \ \text{do} \)
3. \( \quad \text{fact} := \text{fact} \times \text{number}; \)
4. \( \quad \text{number} := \text{number} + 1; \)
   \( \text{end while} \)

Termination:
If \( n = 1 \), \textbf{while} loop is not executed. If \( n > 1 \), then \textit{number} increases by one each time the loop is executed, and hence the \textbf{while} loop terminates when \textit{number} becomes \( n + 1 \).

Loop Invariant: If we reach statement (2) with \textit{number} set to \( k \), then \( \text{fact} = (k - 1)! \).
Factorial Algorithm II

Basis: When \textit{number} is set to 2 in statement (1), then the loop has not been executed, and \( \text{fact} = 1 = (2 - 1)! \), also from statement (1).

Ind. Hyp.: When we reach (2) with \textit{number} set to \( k - 1 \), then \( \text{fact} = (k - 2)! \).

Ind. Step: Need to show that if we reach (2) with \textit{number} set to \( k \), then \( \text{fact} = (k - 1)! \).
If \( (k - 1) > n \), then the loop is not executed, and \textit{number} is never set to \( k \). Hence there is nothing to prove in this case.
Else, \( \text{fact} = (k - 2)! \) is multiplied by \textit{number} = \( (k - 1) \), and hence \textit{fact} now contains \( (k - 1)! \).
When \textit{number} becomes \( (n + 1) \), then the algorithm terminates with \textit{fact} set to \( n! \).
**Selection Sort Algorithm**

**procedure** Selection sort

**input:** $n$, $A[1 : n]$.

**output:** $A[1 : n]$ in non-decreasing order.

**for** $step := 1$ to $(n - 1)$ **do**

\{ $A[1 : step - 1]$ is sorted at begin. of this loop; \}

\{ $A[1 : step]$ is sorted at end of this loop. \}

\{ 1. select smallest element in $A[step : n]$ \}

$small := step$;

**for** $index := step + 1$ to $n$ **do**

\{ 2. move this element to posn. of $A[step]$ \}

\if $A[index] < A[small]$ \then

$small := index$;

\fi

**rof**


**rof**

**end**
Inner loop of Selection Sort

\{
Select smallest element in \( A[\text{step} : n] \) \}

(1) \( \text{small} := \text{step} \);
(2) \textbf{for} \quad \text{index} := \text{step} + 1 \ \textbf{to} \ n \ \textbf{do}

(3) \quad \textbf{if} \quad A[\text{index}] < A[\text{small}] \ \textbf{then}

(4) \quad \text{small} := \text{index} ;

(5) \quad \textbf{fi}

\textbf{rof}

Loop Invariant:

When \( \text{index} \) is set to \( k \) (in statement (2)), \( \text{small} \) is the index of a smallest element in \( A[\text{step} : k - 1] \).