CSE 20, Discrete Mathematics,Fall 1995
How to use induction

The principle of induction says that if $p(a) \wedge \forall n[p(n) \rightarrow p(n+1)]$, then $\forall n \in \mathbf{Z}, n \geq a \rightarrow p(n)$. In other words, if a process evolves in discrete steps, it starts out having a property, and in no one step does the property change from true to false, then it will always have the property after any number of steps. Thus, to prove some property by induction, it suffices to prove $p(a)$ and then prove the general rule $\forall n[p(n) \rightarrow p(n+1)]$.

Thus the format of an induction proof:

- Part 1: We prove a *base case*, $p(a)$. This is usually easy, but it is essential for a correct argument.
- Part 2: We prove the *induction step*. In the induction step, we prove $\forall n[p(n) \rightarrow p(n+1)]$. This calls for a proof by universal generalization, since the outside symbol is $\forall$. Thus, we let k be an arbitrary non-negative integer, and our sub-goal becomes: $p(k) \rightarrow p(k+1)$. So we need a proof by syllogism: We assume $p(k)$, and our sub-goal is now $p(k+1)$. Usually, since these steps are standard, we merge them: "Let k be any integer so that $p(k)$....blah, blah,..Therefore, $p(k+1)$. Thus we have proved the induction step." The assumption $p(k)$ is sometimes called the *induction hypothesis*.
- State what induction then allows us to conclude: "Since we have shown that the property (equation , inequality, relationship, predicate as appropriate) is true for $n = a$ in the base case, and since we have shown in the induction step that if the property is true for $n = k$ then it is also true for $n = k + 1$, by the principle of induction we have shown that the property is true for all integers $n \geq a$."

Induction is the method of choice for analyzing properties of programs with loops. A typical such program initializes some variables based on the inputs. It then changes variables according to some loop instructions until a guard condition fails. Then it produces some output based on the values of the variables. The end result that we want is perhaps how the output depends on the input. But to understand the program, we usually need to prove something stronger first, which tells us something about the values of the variables after t steps. For example, consider the following program:

1. Input:x; Integer
2. i:=0
3. y:=1
4. If$x \leq 0$then output error and halt.

5. While $y < x$ do:
6. begin;while
7. $i = i + 1$
8. $y = 2 * y$
9. end;while
10. Output i

We want to show that the program outputs $log_2 x$ round up to the nearest integer. But to do this we need to prove a stronger claim about what values y, i have throughout the loop.

We'll use the variable t to represent the number of times we looped. y, i change values, so let y_t be the value of y after t loops, and i_t be the value of i after t loops.

- First we determine the *loop invariant* , the property that the variables always have after any number of loops. Here, we see that after t loops, $i_t = t$ and $y_t = 2^t$. So the property we prove by induction is $p(t) \iff$ if the program makes t loops, $i_t = t$ and $y_t = 2^t$. We will prove this by induction on t.

- The base case is $t = 0$. To prove the base case, we just use the initialization part of the program. Before any loops occur, $i_0 = 0 = 0, y_0 = 1 = 2^0$. So the loop invariant holds for $t = 0$.

- To show the induction step, we use the loop instructions. Assume $i_k = k, y_k = 2^k$ are the values of i, y after k loops. After the next loop, $i_{k+1} = i_k + 1 = k + 1$ since i is incremented by 1 in the loop, and $y_{k+1} = 2y_k = 22^k = 2^{k+1}$ since y is doubled during the loop. Thus we have shown that if the loop invariant is true after k loops, and the program makes another loop, then it will be still true after $k + 1$ loops.

- Conclude, by induction, that the loop invariant holds for any number of loops.

- Use the loop invariant to discover the number of loops when the guard condition fails. Here we argue: "Since the value of y after t loops is 2^t and the loop guard fails if $y_t = 2^t \geq x$, it will fail when $t \geq log_2 x$,i.e., at the first integer greater than or equal to $log_2 x$, i.e., $log_2 x$ rounded up to the nearest integer."

- Use the loop invariant and this calculation to analyze the output: "Since the first time t when the guard fails is the $log_2 x$ round up, and since at this time the program outputs $i_t = t$, the program outputs $log_2 x$ round up."

Note that sometimes we are also interested in the time taken by the algorithm as well as the output. This is handled in a similar way. We just stop at the step before the last, where we figure out the number of time steps when the program halts. Sometimes we won't be able to exactly compute this number, but we can show that it is at most some function of the input.