This lecture completes our general introduction to proof methods. We begin with examples of induction principles for domains other than the natural numbers, including strings, trees, and pairs of numbers. We then introduce the general principle of **well-founded induction**, which yields as special cases all sorts of induction principles for all sorts of domains.

Induction over things besides numbers

Persons other than pure mathematicians often write programs that manipulate objects other than natural numbers—for example, strings, lists, trees, arrays, hash tables, programs, airline schedules, and so on. So far, the examples of induction we have seen deal with induction over the natural numbers. How does this help with these other domains?

STRINGS

One answer is that we can do inductive proofs over natural numbers that correspond to the *size* of the objects under consideration. Suppose we want to prove that $\forall s\ P(s)$ for the domain of **strings**. Then define a proposition on natural numbers as follows:

> $Q(n)$ is the property that every string s of length n satisfies $P(s)$.

Then a proof that $\forall n\ Q(n)$ by induction on n establishes that $\forall s\ P(s)$.

Similarly, we can prove things about trees by induction on the depth of the tree, or about programs by induction on the number of symbols in the program. These inductions can become quite cumbersome and unnatural. Let's suppose we had never heard of the natural numbers; could we still do anything with strings and trees and programs? It turns out that we can define very natural induction principles for these sorts of objects without mentioning numbers at all.

An induction principle for strings

Let's write a recursive algorithm for *reversing* a string and show that it works correctly.

SYMBOLS
ALPHABET

First, we will need to say what strings are. The elements of a string are **symbols** drawn from a set of symbols called an **alphabet**, which is usually denoted Σ. For example, if $\Sigma = \{a,b\}$, then strings can consist of sequences of as and bs. Σ^* denotes the set of all possible strings on the alphabet Σ, and always includes the empty string, which is denoted λ. Every symbol of Σ is also a string of length 1. (Note: this property in particular distinguishes strings from lists; but in general reasoning about strings is quite similar to reasoning about lists.)

CONCATENATION

The basic way to construct strings is by **concatenation**. If s_1 and s_2 are strings, then their concatenation is also a string and is written $s_1 s_2$ or $s_1 \cdot s_2$ if punctuation is needed for clarity. Concatenation is defined as follows:

Axiom 4.1 (Concatenation):

$\forall s \in \Sigma^* \; \lambda \cdot s = s \cdot \lambda = s$
$\forall a \in \Sigma \; \forall s_1, s_2 \in \Sigma^* \; (a \cdot s_1) \cdot s_2 = a \cdot (s_1 \cdot s_2)$

Just as Peano did for the natural numbers, we now provide axioms concerning what strings are, then we state an induction principle that allows proofs for all strings. Strings satisfy the following axioms:

Axiom 4.2 (Strings):

The empty string is a string: $\lambda \in \Sigma^*$
Joining any symbol to a string gives a string: $\forall a \in \Sigma \; \forall s \in \Sigma^* \;\; a \cdot s \in \Sigma^*$

Because these axioms do not strictly *define* strings, we need an induction principle to construct proofs over all strings:

Axiom 4.3 (String Induction):

For any property P,
if $P(\lambda)$ and $\forall a \in \Sigma \; \forall s \in \Sigma^* \; (P(s) \implies P(a \cdot s))$,
then $\forall s \in \Sigma^* \; P(s)$.

STRUCTURAL INDUCTION

CONSTRUCTOR

This is a simple instance of **structural induction**, where a set of axioms defines the way in which objects in a set are constructed and an induction principle uses the construction step repeatedly to cover the entire domain. Here, "·" is the **constructor** for the domain of strings, just as "+1" is the constructor for the natural numbers.

Notice that numbers appear nowhere in these axioms. We can do proofs thinking only about the objects in question. Let's define a function that reverses a string and prove that it works.

Axiom 4.4 (Reverse):

$r(\lambda) = \lambda$
$\forall a \in \Sigma \; \forall s \in \Sigma^* r(a \cdot s) = r(s) \cdot a$

We would like to say something like "for every string s, $r(s)$ reverses it." To make this a precise theorem, we'll need some independent, non-recursive way to say what we mean by reversing! There are several ways to do this, of which the easiest is to take advantage of "dot dot dot" notation:

Theorem 4.1: $\forall s \in \Sigma^*$*, let* $s = a_1 a_2 \ldots a_n$*; then* $r(s) = a_n \ldots a_2 a_1$

Proof: The proof is by induction over the strings on the alphabet Σ. Let $P(s)$ be the proposition that if $s = a_1 a_2 \ldots a_n$, then $r(s) = a_n \ldots a_2 a_1$.

- Base case: prove $P(\lambda)$.
  $P(\lambda)$ is the proposition that $r(\lambda) = \lambda$, which is true by definition.
- Inductive step: prove $P(s) \implies P(a \cdot s)$ for all $a \in \Sigma$, $s \in \Sigma^*$.
  1. The inductive hypothesis states that, for some arbitrary string s, if $s = a_1 a_2 \ldots a_n$, then $r(s) = a_n \ldots a_2 a_1$.
  2. To prove: for every symbol a, $r(a \cdot s) = a_n \ldots a_2 a_1 a$.
  3. By the axiom for reverse,

$$\begin{aligned} r(a \cdot s) &= r(s) \cdot a \text{ by the reverse axiom} \\ &= a_n \ldots a_2 a_1 a \text{ by the inductive hypothesis} \end{aligned}$$

Hence, by the string induction principle, for every string s, $r(s)$ reverses it. □

We could alternatively have proven this theorem by induction over the length of the input string. It is an excellent exercise to work out the details of how to do this, and compare to the above method.

Induction over binary trees

Trees are a fundamental data structure in computer science, underlying efficient implementations in many areas including databases, graphics, compilers, editors, optimization, game-playing, and so on. Trees are also used to represent expressions in formal languages. Here we study their most basic form: the **binary**

BINARY TREE

tree. Binary trees include lists (as in Lisp and Scheme), which have `nil` as the rightmost leaf.

ATOMS

In the theory of binary trees, we begin with **atoms**, which are trees with no branches. **A** is the set of atoms, which may or may not be finite. We construct trees (**T**) using the • (cons) operator. (In practice, any object can be an atom as long as it's distinguishable as one.) We will treat only the case of full binary trees, where every node has zero or two children.

Axiom 4.5 (Full Binary Trees):

> Every atom is a tree: $\forall a \in \mathbf{A}\ [a \in \mathbf{T}]$
> Consing any two trees gives a tree: $\forall t_1, t_2 \in \mathbf{T}\ [t_1 \bullet t_2 \in \mathbf{T}]$

The induction principle for trees says that if P holds for all atoms, and if the truth of P for any two trees implies the truth of P for their composition, then P holds for all trees:

Axiom 4.6 (Full Binary Tree Induction):

> For any property P,
> if $\forall a \in \mathbf{A}\ P(a)$
> and $\forall t_1, t_2 \in \mathbf{T}\ [P(t_1) \wedge P(t_2) \implies P(t_1 \bullet t_2)]$
> then $\forall t \in \mathbf{T}\ P(t)$.

Many useful predicates and functions can be defined on trees, including

- $\mathit{leaf}(a,t)$ is true iff atom a is a leaf of tree t.
- $t_1 \prec t_2$ is true iff tree t_1 is a proper subtree of tree t_2.
- $\mathit{count}(t)$ denotes the number of leaves of the tree t.

DEPTH

- $\mathit{depth}(t)$ denotes the **depth** of the tree, where any atom has depth 0.
- $\mathit{balanced}(t)$ is true iff t is a balanced binary tree.

Here we define *leaf*, leaving the others as exercises:

Axiom 4.7 (Leaf):

> $\forall a \in \mathbf{A}\ \forall t \in \mathbf{T}\ \mathit{leaf}(t,a) \Leftrightarrow t = a$
> $\forall a \in \mathbf{A}\ \forall t_1, t_2 \in \mathbf{T}\ \mathit{leaf}(a, t_1 \bullet t_2) \Leftrightarrow \mathit{leaf}(a,t_1) \vee \mathit{leaf}(a,t_2)$

It's not easy to *prove* that definitions of such basic functions are correct, since the "specification" of the function is hard to write in any form that is simpler than the definition itself. Let's look at a slightly less simple function: the function $maxleaf(t)$ returns the largest leaf of the tree t, where the atoms are constrained to be numbers.

Axiom 4.8 (Maxleaf):

$$\forall a \in \mathbf{A}\ maxleaf(a) = a$$
$$\forall t_1, t_2 \in \mathbf{T}\ maxleaf(t_1 \bullet t_2) = max(maxleaf(t_1), maxleaf(t_2))$$

The function *maxleaf* is "correct" if it satisfies two properties: first, $maxleaf(t)$ has to be greater than or equal to every leaf of t; second (and *often forgotten*), $maxleaf(t)$ has to be a leaf of t!

Let's prove the second property first:

Theorem 4.2: *For every tree, t, $maxleaf(t)$ is a leaf of t.*

Proof: The proof is by induction over the binary trees on the atoms $\mathbf{A}$. Let $P(t)$ be the proposition $leaf(maxleaf(t), t)$.

- Base case: prove $\forall a \in \mathbf{A}\ P(a)$.
  $P(a)$ is the proposition that $leaf(maxleaf(a), a)$, which is equivalent by substitution to the proposition $leaf(a, a)$, which is true by definition.
- Inductive step: prove $P(t_1) \wedge P(t_2) \implies P(t_1 \bullet t_2)$ for all $t_1, t_2 \in \mathbf{T}$.
  1. The inductive hypothesis states that $leaf(maxleaf(t_1), t_1) \wedge leaf(maxleaf(t_2), t_2)$.
  2. To prove: $leaf(maxleaf(t_1 \bullet t_2), t_1 \bullet t_2)$.
  3. By the definition above, $maxleaf(t_1 \bullet t_2) = max(maxleaf(t_1), maxleaf(t_2))$.
  4. Since $\forall x, y\ [(max(x,y) = x) \vee (max(x,y) = y)]$, we have
     $(maxleaf(t_1 \bullet t_2) = maxleaf(t_1)) \vee (maxleaf(t_1 \bullet t_2) = maxleaf(t_2))$.
  5. Substituting in the induction hypothesis, we obtain
     $leaf(maxleaf(t_1 \bullet t_2), t_1) \vee leaf(maxleaf(t_1 \bullet t_2), t_2)$.
  6. Hence, by the definition of *leaf*,
     $leaf(maxleaf(t_1 \bullet t_2), t_1 \bullet t_2)$.

Hence, by the binary induction principle, for every tree t, $maxleaf(t)$ is a leaf of t. $\square$

The other part of the verification is the following (the proof is left as an exercise):

Theorem 4.3: *For every tree, t, $maxleaf(t)$ is greater than or equal to every leaf of t.*

Tree induction seems very natural. Could we do a similar proof using natural number induction? Certainly we can prove facts about trees by induction over the *depth* of the tree. $P(n)$ would state that all trees of depth n satisfy some property Q. Unfortunately, the inductive step for a *simple* induction would look like this:

> Given: all trees t of depth n satisfy $Q(t)$
> Prove: all trees t of depth $n+1$ satisfy $Q(t)$

This is usually impossible: for a tree of depth $n+1$, one subtree has depth n, but not necessarily the other. *Strong* induction over the depth of the tree *does* work; in fact it can always be used instead of tree induction.

Note that the formalization of trees above described only full trees. However, it can be easily generalized to describe binary trees that are not necessarily full, i.e., where every node can have 0, 1, or 2 children. The details are easy to fill in, so we won't go through them here.

Induction over pairs of natural numbers

CARTESIAN PRODUCT
PAIRS

Often we need to prove properties over the **Cartesian product** of some given sets. The Cartesian product of sets $\mathbf{A}$ and $\mathbf{B}$ is written $\mathbf{A} \times \mathbf{B}$. It is the set of all **pairs** (a,b) where $a \in \mathbf{A}$ and $b \in \mathbf{B}$. For example, the set $\mathbf{N} \times \mathbf{N}$ is the set of all pairs of natural numbers. Such sets arise when we prove properties of functions with two arguments, when we prove facts about all points on a grid, etc.

Let's look at an example: the knight's tour. We will prove that a knight starting at (0,0) can visit every square on the unbounded nonnegative quadrant. Figure 1 shows (part of) the infinite board and illustrates the moves a knight can make.

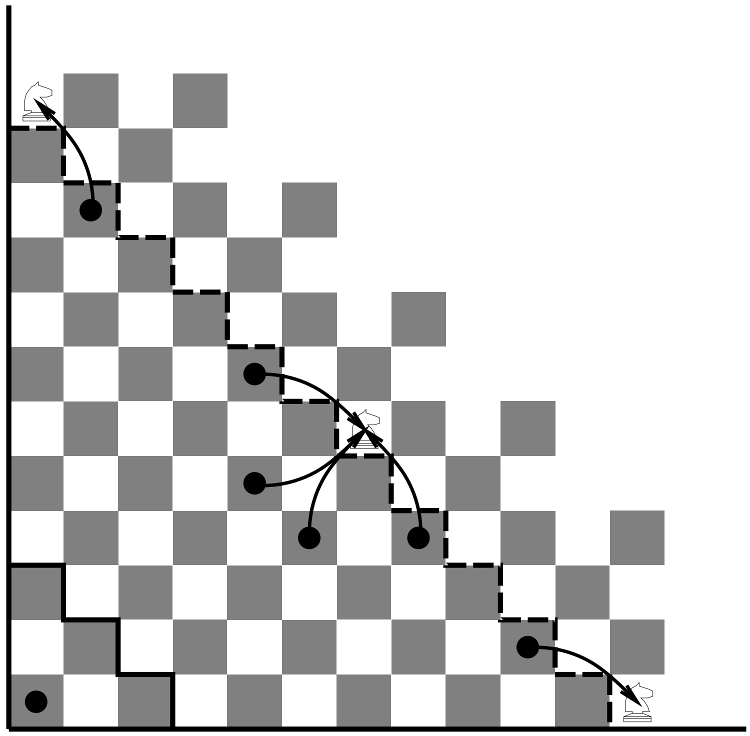

Figure 1: The knight's tour, showing the "base case" squares, the possible legal moves for a knight, and the "inductive step."

To prove this result, we'll need some facts about knight's moves. In particular, we'll need the following:

Axiom 4.9 (Knight's Move):

> If square $(x \pm 1, y \pm 2)$ or $(x \pm 2, y \pm 1)$ is reachable by a knight, then square (x,y) is reachable by a knight.

We'll also need an induction principle for pairs of natural numbers. The idea for the knight's move proof is to establish a region that is reachable and then to show that any square adjacent to that region is reachable; hence the region grows to fill the unbounded quadrant. There are many ways to define the shape of this region; we'll use the triangular region shown in Figure 1.

Our induction principle is, informally, that if the truth of P for every pair (x', y') in the region "just below" (x,y) implies the truth of P for (x,y), then P is true for all (x,y). Notice that this is a strong induction principle.

Axiom 4.10 (Strong Induction (Pairs)):

For any property P,
if $\forall x, y \in \mathbf{N}$
$[\forall x', y' \in \mathbf{N}\ (x' + y') < (x + y) \implies P(x', y')] \implies P(x, y)$
then $\forall x, y \in \mathbf{N}\ P(x, y)$.

But where is the base case? Actually, it's there but hidden. When $(x, y) = (0, 0)$, the condition $[\forall x', y' \in \mathbf{N}\ (x' + y') < (x + y) \implies P(x', y')]$ is vacuously true because there are no such pairs. Hence $P(0, 0)$ is part of the premise to be proved. More generally, the "base case" is the set of (x, y) pairs for which the inductive hypothesis does not suffice to provide a proof.

Now we are ready to prove our theorem:

Theorem 4.4: $\forall x, y \in \boldsymbol{N}$*, the square* (x, y) *is reachable by a knight starting at* $(0, 0)$.

Proof: The proof is by strong induction over the pairs of natural numbers. Let $P(x, y)$ be the proposition that square (x, y) is reachable by a knight starting at $(0, 0)$.

- Base case: the propositions $P(0,0), P(0,1), P(0,2), P(1,0), P(1,1), P(2,0)$, for which $x + y \leq 2$, must be established separately. Each of these can be established by appropriate application of the knight's move axiom.

- Inductive step: prove that, for all (x, y) such that $x + y > 2$,
  $[\forall x', y' \in \mathbf{N}\ (x' + y') < (x + y) \implies P(x', y')] \implies P(x, y)$.

  1. The inductive hypothesis states that, for all $x', y' \in \mathbf{N}$ such that $(x' + y') < (x + y)$, the square (x', y') is reachable from $(0, 0)$.
  2. All the squares $(x', y') = (x - 2, y \pm 1)$ and $(x', y') = (x \pm 1, y - 2)$ satisfy the condition $(x' + y') < (x + y)$.
  3. For any $x, y \in \mathbf{N}$ such that $x + y > 2$, at least one of these squares is on the board, i.e., satisfies $x', y' \in \mathbf{N}$ (proof by cases).
  4. Hence, by the knight's move axiom, (x, y) is reachable from $(0, 0)$.

Hence, by the strong induction principle for pairs, every square in the unbounded positive quadrant is reachable by a knight from $(0, 0)$. $\square$

The proof could also be done by strong induction on the natural numbers using $n = x + y$ as the induction variable. Which is more elegant is perhaps a matter of taste; but the important *insight* is the use of a suitable notion of "smaller" on pairs of natural numbers. For some proofs, "smaller" can be defined as "at least one of the pair is smaller and the other is no bigger", which gives rectangular regions that, stepwise, fill up the quadrant. In the knight's tour problem, however, some of the required moves violate this ordering.

Well-founded induction

Looking at all the induction principles we have seen so far, one recurring theme stands out: from properties of "smaller" elements, we prove properties of a "larger" element. n is smaller than $n + 1$; s is smaller than $a \cdot s$; t_1 and t_2 are smaller than $t_1 \bullet t_2$; and so on.

The strong induction principle for pairs, stated in the preceding section, gives a clue as to how to formalize this idea into a general induction principle. We simply supply a generalized notion of "smaller than" instead of using $<$. We denote this relation $\prec$, which is assumed to be defined on whatever set $\mathbf{X}$ we are interested in (natural numbers, sets, trees, pairs, strings, lists, airline schedules, etc.). For induction to work, we require that $\prec$ have the property of well-foundedness:

WELL-FOUNDED

Definition 4.1 (Well-founded): A relation $\prec$ on $\mathbf{X}$ is **well-founded** if there can be no infinite decreasing sequences of elements of $\mathbf{X}$ related by $\prec$.

WELL-FOUNDED INDUCTION

Given this, we can state the principle of **well-founded induction**, of which all our other principles are special cases:

Axiom 4.11 (Well-Founded Induction):

For any property P, and any wellfounded relation $\prec$ on $\mathbf{X}$,
if $\forall x \in \mathbf{X}\ [[\forall y \in \mathbf{X}\ \ y \prec x \implies P(y)] \implies P(x)]$
then $\forall x \in \mathbf{X}\ P(x)$.

As with induction over pairs, the well-founded induction principle includes the requirement for establishing the "base case"—that is, proving $P(x)$ independently for all those x where the inductive hypothesis does not suffice.

The property of well-foundedness is easy to see for all the cases we have covered. There is also a generalized equivalent of well-ordering:

WELL-ORDERED

Definition 4.2 (Well-ordering): A set $\mathbf{X}$ is **well-ordered** by the relation $\prec$ iff every nonempty subset of $\mathbf{X}$ has at least one minimal element with respect to $\prec$.

The following very general theorem can be proved:

Theorem 4.5: *A relation $\prec$ on $\mathbf{X}$ is well-founded iff $\mathbf{X}$ is well-ordered by $\prec$.*

Although this seems very abstract and useless, it is in fact used all the time by programmers who write recursive functions that do complex things to their arguments. Consider the following recursive skeleton:

$$f(x) = \text{ if } B(x) \text{ then } k \text{ else } f(g(x))$$

This will terminate iff $g(x) \prec x$ for some well-ordering of $\mathbf{X}$ with minimal element(s) satisfying $B(x)$. Thus, the programmer must be sure that repeated application of g cannot generate an infinite sequence of values that do not satisfy B.

Sometimes, "smaller" can be surprisingly nonobvious. Consider the following function on the natural numbers:

$$f(0) = 1; \quad f(1) = 1$$
$$\text{if } n > 1 \text{ is even then } f(n) = f(n/2), \text{ else } f(n) = f(3n+1).$$

COLLATZ CONJECTURE

The **Collatz conjecture** states that $\forall n \in \mathbf{N}\ f(n) = 1$. You may wish to check this out for various values of n. No proof is known.