Induction and Recursion 2

Today:

☐ More induction

☐ Recursion

1 Recursion on Definitions

When we've defined something recursively, you can induct on the number of steps used in that definition. Let's do an example.

Recall the definition of a boolean formula over a set of predicate symbols $\mathcal{P}$: First we said (1) that each variable $P \in \mathcal{P}$ was a well-formed formula. Then we said (2) that if α and β were well formed formulas over $\mathcal{P}$, then so were (2a) $(\alpha \vee \beta)$, (2b) $(\alpha \wedge \beta)$, and (2c) $(\neg\alpha)$.

Now supposed I'd like to make the simple claim that any boolean formula has the same number of left-parentheses as right-parentheses. Easy. What I do is to induct on the number of applications of steps (2a), (2b), or (2c) used to derive the formula ϕ. For the base cases—0 steps—our formula ϕ must have been derived by rule 1, meaning that is a predicate symbol, meaning that it has *no* parentheses. Well, actually this isn't vacuous—I better explicitly demand that predicate symbols not include either left or right parentheses, else we'll be hosed! Alternatively, our formula ϕ is derived by rule (2a), (2b), or (2c). In the first case, $\phi = (\alpha \vee \beta)$ and, by inductive hypothesis, α has an equal number of left and right parentheses, and so ϕ does too, having added one of each. Same if $\phi = (\alpha \wedge \beta)$ or $\phi = (\neg\alpha)$. And we are done. $\diamondsuit$

Inductions on definitions are common in theoretical computer science. It's really the natural way to prove things when one has defined things inductively.

2 Fundamental Theorem of Arithmetic

Let's prove the following:

Theorem 1 *Ever number $n \geq$ can be written uniquely as the product of increasing prime powers:*

$$n = p_1^{\alpha_1} p_2^{\alpha_2} \cdots p_k^{\alpha_k}$$

where $p_1 < p_2 < \cdots < p_k$ are primes, $\alpha_1, \ldots, \alpha_k \geq 1$.

Proof. By strong induction on n. When $n = 2$ the result is immediately true; that is our basis. Otherwise, suppose the theorem is true for all numbers less than n. We wish to show that the theorem holds for n.

Suppose the number n is prime. Then it can immediately be written as the product of primes.

Otherwise n is not prime, meaning that it is composite and n can written as the product of numbers a and b where $2 \le a < n$ and $2 \le b < n$. By inductive assumption, $a = p_1^{\alpha_1} \cdots p_a^{\alpha_a}$ can be written as the product of primes and $b = q_1^{\beta_1} \cdots q_b^{\beta_b}$ can be written as the product of primes, and therefore $ab = p_1^{\alpha_1} \cdots p_a^{\alpha_a}\, q_1^{\beta_1} \cdots q_b^{\beta_b}$ can be written as the product of primes. This establishes the result without the adverb *uniquely.*

To show show uniqueness I am going to assume the following lemma, which I will not prove. It says that if $p \mid ab$ where p is a prime, then $p \mid a$ or $p \mid b$. By repeatedly applying this lemma we can say that if $p \mid q_1^{\beta_1} \cdots q_b^{\beta_b}$ then $p = q_j$ for some j.

Now to business. To show uniqueness suppose for contradiction that there is some number that can be written as a product of increasing primes in two different way. Let n be the *smallest* such number. (Note: we are using well-ordering for this to be well-defined!) So $n = p_1^{\alpha_1} \cdots p_a^{\alpha_a}$ and $n = q_1^{\beta_1} \cdots q_b^{\beta_b}$. Is it possible that $p_i = q_j$ for some i, j? It is not, for if we had that condition then we could divide n by p_i and get a *smaller* number that has two decompositions as the product of increasing primes. We conclude, then, that $n = p_1^{\alpha_1} \cdots p_a^{\alpha_a} = q_1^{\beta_1} \cdots q_b^{\beta_b}$ where all of the $p_1, \ldots, p_a, q_1, \ldots, q_j$ are distinct. Now $p_1 \mid n$ and so, in particular, $p_1 \mid q_1^{\beta_1} \cdots q_b^{\beta_b}$. By the lemma from a paragraph back, this means that $p_1 = q_j$ for some j. But this contradicts our finding that the p_i values are distinct from the q_j values, finishing the proof. $\diamondsuit$

3 Well-Ordering Principle

Not covered to save time, but left in the notes.

The following *well-ordering principle* often provides an alternative to induction. In some settings, it gives more direct and elegant proofs. An example is the *division theorem.*

> **Well-ordering principle.** Any nonempty set of natural numbers has a least element: $S \subseteq \mathbb{N}$ implies that $s = \min S$ is well-defined.

Note: if you change $\mathbb{N}$ to $\mathbb{Z}$, it isn't true.

Envelope problem with well-ordering. For example, let's reprove the envelope problem using well-ordering. Last time: You can dispense any number $n \ge 44$ of envelopes in packages of 5 and 12. That is, $(\forall n \ge 44)(\exists a, b \in \mathbb{N})(n = 5a + 12b)$.

Proof. Assume for contradiction that the claim is false. Let C be the set of counter-examples to the claim: $c \in C$ means that $c \geq 44$ and $c \neq 5a + 12b$ for any $a, b \geq 0$. By well-ordering, C has a least element. Call it c. How big is c? We know $c \geq 44$. By the chart we did last time, we know that $c \geq 60$. Now $c - 5$ is not a counterexample (because c was the smallest counter-example), so $c - 5 = 5a + 12b$ for some $a, b \geq 0$. But then $c = 5(a + 1) + 12b$. $\diamondsuit$

Many proofs done by induction can be done by well-ordering. Some authors go so far as to say that well-ordering and induction are equivalent—in the sense that PA with one as the final axiom implies all other. But this isn't true. Induction (in the context of PA) is enough to prove well-ordering, but not the other way around.

Unnoticed well-ordering in a previous proof. We actually used well-ordering without comment in the proof of $\sqrt{2}$ being irrational. Remember how that went? We said: assume for contradiction that $\sqrt{2}$ is rational. That is, there exists integers p, q, $q \neq 0$, such that $\sqrt{2} = p/q$ and p and q have no common divisors bigger than 1. But wait! *Why* can we assume that p and q have no common divisors greater 1? We are effectively considering the set of all numbers P such that for each $p \in \P$ there is an integer q such that $\sqrt{2} = p/q$. Of course P is an infinite set of natural numbers. To find *the* p/q that is reduced, I am selecting the *least* element $p \in P$ and then the corresponding $q \in \mathbb{N}$ such that $\sqrt{2} = p/q$. Why does such a p exist? It would not if we were allowing real-valued p and q. But with $P \subseteq \mathbb{N}$, well ordering gives us the least p, from which we continue with our proof.

4 Recursion

Recursion entails solving a problem by solving the same problem, but on smaller instances. Those smaller problems are solved how? By solving the same problem, but on smaller instances. And so on, until problems are *so* small that you just *know* the answer.

Expressed mechanistically, in the language of programming, recursive algorithms call themselves. Or they call algorithms in call graphs that are otherwise cyclic.

Recursion is closely tied to induction (at least it feels close to me). There one proves some result by virtue of having proven related, smaller results.

The key to thinking recursively is this: don't let your mind "descend" into the recursion; instead, think of the smaller solutions as being solved as if by *magic*. Thinking inductively and thinking recursively are highly similar.

5 Example 1: Counting Tic-tac-toe Games

Let's count N_0 =the total number of possible tic-tac-toe games, where X moves first. I'm going to assume I have a computer to help me out with the work; my job is to write the equation that we can plug in.

It's good to first get a ballpark figure and use this to ascertain if our recursive decomposition is going to give a number, in a reasonable amount of time, when programmed up. For this we would note that $N < 9! = 362880 \approx 2^{18.5}$. If you think back to the comments I made on what is practical, this is well on the side of practical Way less than 2^{30}, which is where I said you needed to start to take some care.

How to represent a board? We can regard tic-tac-toes position as encoded by a string w from $\{\texttt{X}, \texttt{O}, \texttt{-}\}^9$. That is, we regard a board as a 9-character string. Not all of the 3^9 possible strings can arise. But it's still a good way to represent a board.

Let $S(x)$ be the set of all possible (immediate) successor positions from x. E.g., Ex:

$$\begin{aligned} S(\texttt{--- -X- ---}) \;=\; \{&\texttt{O-- -X- ---},\ \texttt{-O- -X- ---},\ \texttt{--O -X- ---},\\ &\texttt{--- -O- ---},\ \texttt{--- -XO ---},\ \texttt{--- -X- O--},\\ &\texttt{--- -X- -O-},\ \texttt{--- -X- --O}\}. \end{aligned}$$

On the other hand, $S(\texttt{OO-XXX---}) = \emptyset$, as `X` has already won this game, whence there are no successor positions. Note that either $|S(w)|$ is either 0 (when a player has won) or the number of dashes in the string w.

Let $N(w)$ =number of games that can continue from board w, including w. So we want to know $N_0 = N(\texttt{--- --- ---})$.

It is easy to compute N recursively. It's

$$N(w) = \begin{cases} 1 & \text{if } S(w) = \emptyset\text{, and} \\ \sum_{y \in S(w)} N(y) & \text{otherwise} \end{cases}$$

For example, $\mathbb{N}(\texttt{--- --- ---}) = N(\texttt{X-- ---}) + \cdots + N(\texttt{--- --- --X})$.

We can code it up!! The answer is $N_0 = 255\,168$ possible games. Took me 22 lines of code. See Figure 1.

6 Towers of Hanoi

For this problem n rings of increasing diameter are placed on peg A. The rings must be moved from peg A to peg C in a way that respects the following rules. First, only the topmost ring on a peg can be moved to another peg, where it again becomes the topmost ring. Second, a bigger ring cannot be placed atop a smaller one. See Figure 2

```
# Counts the number of possible tic-tac-toe games
# Uses recursively defined N(w) = number of continuations of board w.
# Board represented as a 10-element list of 'X', 'O', '-' chars, element 0 unused.

X,O,EMPTY,UNUSED= 'X','O','-','.'
Wins = [[1,2,3],[4,5,6],[7,8,9], [1,4,7],[2,5,8],[3,6,9], [1,5,9],[3,5,7]]

def win(w,P):              # return true if player P has won. P==X or P==O
    for [a,b,c] in Wins:
        if w[a] == w[b] == w[c] == P: return True
    return False

def whose_move(w):         # Return X or O depending on whose move it is
    if w.count(EMPTY)%2: return X
    return O

def game_over(w):          # True after a win or nowhere left to go
    return win(w,X) or win(w,O) or w.count(EMPTY)==0

def N(w):                  # Calculate number of games that elaborate board
    if game_over(w): return 1
    sum = 0
    for i in range(1,10):
        if w[i]!=EMPTY: continue
        y = w[:]
        y[i] = whose_move(w)
        sum += N(y)
    return sum

w = [UNUSED] + [EMPTY]*9   ###  MAIN PROGRAM  ###
print('Number of possible games is', N(w))
```

Figure 1: Recursive program to compute the number of possible tic-tac-toe games.

```
           |               |               |
          -|-              |               |
         --|--             |               |
        ---|---            |               |
       ----|----           |               |
===========|===============|===============|==========
           A               B               C
```

Figure 2: Towers of Hanoi problem.

The recursive way to solve the problem requires us to generalize it: move any number of increasing-diameter rings from a first peg to a second using a third as an intermediate peg. Here's an algorithm:

```
// Transfer n pegs from A to C using B as intermediate
algorithm TH (n, A,B, C) // Alternative convention A,B,C ?
if n = 1 then Move(A,C)
TH(n-1, A,B, C)
Move(A,C)
TH(n-1, B,C, A)
```

Thinking recursively, we are assuming some "black box" algorithm to move the first $n-1$ rings from A to C; then we move the big ring to C; and then we use our black-box method to pile of $n-1$ rings from B to C.

How many moves $T(n)$ does our algorithm take to move n rings to a new peg? Clearly $T(1) = 1$ and $T(n) = 2T(n-1)+1$ for $n \geq 1$. It is easy to guess the solution if one makes a table and looks at it:

1	2	3	4	5	6	7
1	3	7	16	31	63	127

from which one can guess that $T(n) = 2^n - 1$ for all $n \geq 1$. This can be proven by induction. Do it!

We can also show that the result is optimal. Let $S(n)$ be the *minimal* number of moves to move n rings from one peg to another in a way that abides the the given rules. In order to move n pegs from A to C there must be a first time when the largest ring moves from A to an unoccupied peg. How many moves does that take? At least $S(n-1)$. The largest ring must eventually moves for the last time to peg C. How many moves does that take? Exactly one. Now the $n-1$ rings must migrate from their current location to peg C. How long will that take? At least $S(n-1)$. All together, then, *any* solution must take at least $2S(n-1)$ moves:

$$S(n) \geq 2S(n) - 1.$$

By induction as before $S(n) \geq 2^n - 1$. Thus our former solution is optimal.

According to Wikipedia:

> The puzzle was introduced to the West by the French mathematician Édouard Lucas in 1883. Numerous myths regarding the ancient and mystical nature of the puzzle popped up almost immediately, including one about an Indian temple in Kashi Vishwanath containing a large room with three time-worn posts in it, surrounded by 64 golden disks. Acting out the command of an ancient prophecy, Brahmin priests have been moving these disks in accordance with the immutable rules of Brahma since that time. The puzzle is therefore also known as the Tower of Brahma. According to the legend, when the last move of the puzzle is completed, the world will end.

Fortunately, even if the priests move a ring each second, $2^n - 1$ seconds is about 600 billion years, so we should be safe for a while.