# Proof by Induction (cont'd) & Structural Induction

Today we continue demonstrating proofs by (ordinary and strong) induction.

## 1 Tournaments

An Internet gaming company is conducting a chess tournament on the Internet. It is run round robin, not by elimination. That is, everyone plays everyone exactly once. And we assume no ties. We can represent the results nicely by arrows, e.g.:

A beats B, C, D
B beats D
C beats D
D beats C

(This structure is called a "digraph". We will study them carefully soon.)
Now, the gaming company wants to use this information to make some decisions, e.g., award a first prize and rank everyone. It wants to do this in such a way that its customers don't complain about the rankings being unfair. There are some problems with doing this.

### 1.1 Cycles

The first problem is that the tournament graph can contain cycles. For example: above, B beats D beats C beats B. If there's a cycle, then any ranking method is a least a little problematic: someone who beat someone else would be ranked lower.

Of course, the company has no power to avoid cycles in play results. But the gaming company might hope that "most of the time", when there is a cycle, it's very big (so the people involved aren't likely to discover it). However, tournaments have an unfortunate property:

**Theorem 1.1** Any tournament that contains any cycle, contains a cycle of length 3 (3-cycle).

That is, three players, (e.g., B, C, D, above) have an "inconsistent" set of results.

This can be proved by ordinary induction, where $P(n)$ is the predicate "Any tournament that contains an $n$-cycle contains a 3-cycle."

Prove: $\forall n \geq 3(P(n))$
1. (Base) $P(3)$ Trivial by assumption.
2. (Inductive step) $\forall n \geq 3(P(n) \Rightarrow P(n+1))$

1. Fix $n \geq 3$
2. Assume $P(n)$
3. $P(n+1)$
    1. Fix $T$, a tournament, and $p_1 \to p_2 \to \ldots \to p_{n+1} \to p_1$, an $n+1$-cycle in $T$.
    2. $T$ contains a 3-cycle. ???
    3. QED Implication, UG
4. QED Implication, UG

3. QED

How can we find a 3-cycle in $T$? Consider the example of a 5-cycle $p_1 \to p_2 \to p_3 \to p_4 \to p_5$. To get a 3 cycle, notice that $p_1$ and $p_3$ have an arrow, one way or the other. If it's from $p_3$ to $p_1$, then $p_1, p_2, p_3$ form a cycle. If it's the other way, then we can omit $p_2$ and still have a cycle $p_1 \to p_3 \to p_4 \to p_5$. Then we can use induction on the smaller cycle. This generalizes:

2. $T$ contains a 3-cycle.
    1. If $p_3 \to p_1$ then $T$ contains a 3-cycle. Use $p_1 \to p_2 \to p_3 \to p_1$.
    2. If $p_1 \to p_3$ then $T$ contains a 3-cycle.
        1. Assume $p_1 \to p_3$.
        2. $T$ contains an $n$-cycle. Use $p_1 \to p_3 \to p_4 \to p_{n+1} \to p_1$.
        3. $T$ contains a 3 cycle. By inductive hypothesis (2.2)
    3. QED Cases

A detail to be careful about here: This argument depends on the fact that the arrow we add between $p_1$ and $p_3$ is not already part of the original cycle. This is true because we know that the original cycle had length $n+1$, which is at least 4. So there is a $p_4$ between $p_3$ and $p_1$ in the cycle.

### 1.2 Locally fair rankings

The gaming company decides that, although they cannot avoid inconsistent orders, they would at least like to make the ranking "locally fair", meaning that if one player $p$ is ranked immediately ahead of another player $q$, then $p$ must have beaten $q$. It's OK if the 5th ranked player loses to the 9th ranked player, but we want to be sure the 5th ranked player beats the 6th ranked player. (The company might want to tell each player who the players were that are ranked immediately above and below him, and don't want someone to complain that the guy ranked above him is someone he beat.)

They are happy to discover:

**Theorem 1.2** Theorem: Any tournament has a locally fair ranking.

We will prove the theorem by strong induction, where $P(n)$ says "any $n$-player tournament has a locally fair ranking". The strategy is to "split" the players into those that beat and those that lose to a particular player $x$. We then rank those two subsets separately, by using the inductive hypothesis. This works because we can consider each of the two sets as a "subtournaments." Then we put the rankings together.

Prove: $\forall n \geq 1(P(n))$

1. (Base) $P(1)$ — Obvious—just one player to be ranked.
2. (Inductive step) $\forall n \geq 1[(\forall i, 1 \leq i \leq n(P(n))) \Rightarrow P(n+1)]$
    1. Fix $n \geq 1$
    2. Assume $\forall i, 1 \leq i \leq n(P(i))$.
    3. $P(n+1)$
        1. Fix any $n+1$-player tournament $T$.
        2. $T$ has a locally fair ranking.
            1. Choose $x$ to be any player, $B$ the set of players that beat $x$, and $L$ the set of players that lose to $x$.
            2. $B$ has a locally fair ranking. — Inductive assumption (2.2).
            3. $L$ has a locally fair ranking. — Inductive assumption (2.2).
            4. Define a ranking $R$ of all the players by: $B$ ranking, then $x$, then $L$ ranking.
            5. $R$ is a locally fair ranking for $T$.
            6. QED — EG
        3. QED — UG
    4. QED — Implication, UG
3. QED — Strong induction

### 1.3 Locally fair rankings and number of wins

For a while, the customers are happy with just locally fair rankings. But then a game comes up where local fairness doesn't work out so well:

Example:
A beats B
B beats C
C beats A,D
D beats A,B,E
E beats A,B,C
(draw)

Here, A,B,C,D,E is a locally fair ranking. However, it's in the reverse order according to number of wins. This is clearly a problem. Now it occurs to the company that it would be nice if the rankings could also be consistent with numbers of wins. Unfortunately, we can't get this at the same time as local fairness.
Example: 6 players
A beats C,D,E,F (4 wins)
B beats A,E,F (3 wins)
C beats B,F (2 wins)
D beats C,B (2 wins)
E beats C,D (2 wins)
F beats D,E (2 wins)

Ordering consistently with number of wins would mean $A$ would have to be ranked immediately ahead of $B$. But $B$ beats $A$, violating local fairness.

So then they decide they will be happy to settle for a fair ranking in which the highest ranked player has the most wins (he may be tied for most wins). This would rule out the bad example

above.
It turns out this is possible: In fact, in the event of a tie for most wins, the company even has the option of choosing which best player is ranked first:

**Theorem 1.3** For any $n$-player tournament $T$ and any $z$ that has the largest number of wins in $T$, there exists a locally fair ranking with $z$ ranked first.

For the proof, take $P(n)$ to be the statement of the theorem.

Prove: $\forall n \geq 1(P(n))$

This can be proved by induction, but the inductive step is more complicated than in the previous proof.

Here's a first proof attempt. We'll try a proof like the previous one, this time splitting on some $x$ that loses to $z$. We split all-but-$x$ into $B$ and $L$ as before, based on their records with respect to (w.r.t.) $x$. We know that $z$ is in $B$, because it beats $x$. Now we try to use the inductive hypothesis to rank $B$ while keeping $z$ first, and to rank $L$. Then we combine as before.

This all sounds plausible—but it's wrong. Applying the inductive hypothesis requires that $z$ have the best won-lost record in the "subtournament" involving just the $B$ elements. But it might not.
Example:
Players A, B, x, and a lot of others
A and B beat x
x beats all of the others
B beats A
A beats all the others
all the others beat B
It doesn't matter what happens among the others.

A has the best record overall, but in the two-player subtournament, it is worse than B.

This is an example of one of the most common problems in an inductive proof. When we break down the problem to a smaller one, we forget to check that all the hypotheses in the theorem remain true. Here, in the smaller problem we lost the hypothesis that $z$ beat the most players.

So let's try different approach. It turns out that a correct proof can be given using ordinary (not strong) induction. Recall $P(n)$: For any $n$-player tournament $T$ and any $z$ that has the largest number of wins in $T$, there exists a locally fair ranking with $z$ ranked first.

Prove: $\forall n \geq 1(P(n))$
1. (Base) $P(1)$ — Only one possibility.
2. (Inductive step) $\forall n \geq 1[P(n) \Rightarrow P(n+1)]$
    1. Fix $n \geq 1$.
    2. Assume $P(n)$.
    3. $P(n+1)$.
        1. Fix $T$, an $n+1$-player tournament, and $z$, a player with the most wins.

| | |
|---|---|
| 2. $\exists$ a locally fair ranking of $T$ with $z$ first. | ??? |
| 3. QED | UG |
| 4. QED | Implication, UG |
| 3. QED | Induction |

The key step will break down into cases, based on whether or not $z$ lost any matches. The case where $z$ won everything is easy; the other case is tricky. Here's the easy case:

| | |
|---|---|
| 2. $\exists$ a locally fair ranking of $T$ with $z$ first. | |
| 1. If $z$ is undefeated in $T$ then $\exists$ a locally fair ranking of $T$ with $z$ first. | |
| 1. Assume $z$ is undefeated in $T$. | |
| 2. Choose $R$ to be a locally fair ranking of all-but-$z$. | Previous theorem says this exists, EI |
| 3. $z$ then $R$ is a locally fair ranking of $T$. | |
| 4. $\exists$ a locally fair ranking of $T$ with $z$ first. | EG |
| 5. QED | Implication |
| 2. If $z$ is defeated by someone then $\exists$ a locally fair ranking of $T$ with $z$ first. | ??? |
| 3. QED | Cases |

And the tricky case:

| | |
|---|---|
| 2. If $z$ is defeated by someone then $\exists$ a locally fair ranking of $T$ with $z$ first. | |
| 1. Assume $z$ is defeated by someone. | |
| 2. Choose $x$ to be someone who defeated $z$. | |
| 3. $z$ has the best record in the subtournament involving all-but-$x$. | $z$ had the best record in $T$, no wins are removed. |
| 4. $\exists$ ranking of subtournament having $z$ first. | Inductive hypothesis, $z$ has best record. |
| 5. Choose $R$ to be a ranking of all-but-$x$ having $z$ first. | EI |
| 6. In $T$, $x$ is defeated by someone. | $z$ was defeated, and $x$'s record in $T$ can't be better than $z$'s. |
| 7. Define ranking $S$ to be $R$, with $x$ inserted right after the lowest-ranked player that beats $x$. | |
| 8. $S$ is a locally fair ranking of $T$. | $R$ is locally fair, $x$ is beaten by the player ranked just above it and beats all ranked below it. |
| 9. $z$ is first in $S$. | $z$ is first in $R$, and $x$ is not inserted at the top. |
| 10. $\exists$ a locally fair ranking of $T$ with $z$ first. | $S$ works, EG |
| QED | Implication |

## 2 Induction, Strong Induction, and Well-Ordering

### 2.1 Induction vs. strong induction

Strong induction looks like a more powerful proof method than ordinary induction, because it allows more information to be assumed in the inductive step. But it's not really stronger—anything that can be proved with strong induction can also be proved with ordinary induction. Here's an example.

Recall the proof that every natural number $\geq 2$ can be written as product of primes (factored into primes). We used strong induction, where $P(n)$ was the predicate "$n$ can be factored into primes". The inductive step involved writing $n+1 = ab$, and using $P(a)$ and $P(b)$ to get factorizations for $a$ and $b$, then multiplying them.

Now we'll change this to an ordinary inductive proof. Define a new predicate $Q(n)$ to be $\forall i, 2 \leq i \leq n$ ($i$ can be factored). We'll prove $Q(n)$ by ordinary induction:

$Q(2)$: Same as $P(2)$, use same proof.

$Q(n) \Rightarrow Q(n+1)$:
Our goal is now to show: $\forall i, 2 \leq i \leq n+1$ ($i$ can be factored).
What we can assume is: $\forall i, 2 \leq i \leq n$ ($i$ can be factored).

Notice that all the needs to be shown is that $n+1$ can be factored (the others already assumed). And we can assume factorizations of all smaller numbers. So the old strong induction step can be used to show this.
This idea is quite general. We can always convert a strong inductive proof of $\forall n \geq k(P(n))$ to an ordinary inductive proof. Just define $Q(n)$ to be $\forall i, k \leq i \leq n(P(i))$, that is, $P(k) \wedge P(k+1) \wedge \cdots \wedge P(n)$.

**Base case:** Show $Q(k)$, same as $P(k)$, which has already been shown as part of the strong induction proof.

**Inductive step:** Show $Q(n) \Rightarrow Q(n+1)$, in other words, $P(k) \wedge P(k+1) \wedge \cdots \wedge P(n) \Rightarrow P(k) \wedge P(k+1) \wedge \cdots \wedge P(n) \wedge P(n+1)$.

This really amounts to showing $P(k) \wedge P(k+1) \wedge \cdots \wedge P(n) \Rightarrow P(n+1)$. But this has already been shown in the strong induction proof.

Of course, any ordinary inductive proof **is** a strong induction proof has well. We are allowed to assume $P(1), \ldots, P(n)$ to prove $P(n+1)$, but all we actually use is $P(n)$.

Induction and strong induction are said to be "equivalent" proof rules, in that each can prove the same things (in the context of some standard number axioms and logical deduction rules).

### 2.2 Well-ordering

There is another variation of induction that is sometimes useful: well-ordering. The well-ordering axiom looks nothing like the induction axiom:

**Axiom** Every nonempty subset $S \subseteq \mathbb{N}$ has a smallest element.

The well-ordering axiom is often called the well-ordering principle. It may seem obvious and useless!

As for obvious, note that this axiom would be false if the set of natural number, $\mathcal{N}$, were replaced by, say, the set of rational numbers. The subset $S$ consisting of positive rational numbers has no smallest element.

As for useless, it turns out the well-ordering is just as powerful as our very best weapon, strong induction!

For example: we can reprove the theorem about 3-cycles using well-ordering instead of induction.

**Theorem 2.1** Any tournament that contains a cycle contains a 3-cycle.

*Proof.* Assume tournament $T$ contains a cycle. Let $n$ be the smallest cycle length (there is some smallest, by well-ordering). $n$ must be at least 3. If it's 3, then we're done. So suppose $n \geq 4$. Consider a cycle with this length. Now consider $p_1$ and $p_3$ as before. Whichever way the arrow goes, we can use it in constructing a shorter cycle in $T$ (as we did before). This is a contradiction, since we assumed that $n$ is the smallest cycle length. Since we got a contradiction to our assumption that $n \geq 4$, we must have $n = 3$ as claimed. ∎

This proof is really very similar to the other one. In general, well-ordering is equivalent to induction (and strong induction), in that they can be used to prove the same theorems. Again the transformation is standard. Given an induction proof for proposition $P(n)$, we can build a well ordering proof as follows. Suppose for the purpose of contradiction that $P(n)$ is *not* true for every $n$. That means it is false for some $n$. So consider the set of elements $n$ for which $P(n)$ is false. We've just assumed this set is nonempty. So by well ordering it has a smallest element, say $k$. This means $P(k-1)$ is true (else $k-1$ would be in our set). But in our original inductive proof for $P(n)$, we showed that $P(k-1) \implies P(k)$ (this is the inductive step). So $P(k)$ must be true. This is a contradiction to our choice of $k$, which shows that our original assumption (that $P(n)$ is false for some $n$) is false.

# 3 Structural induction

Structural induction is a variant of induction for proving things about other data types besides $N$, in particular, about data types that are defined recursively. The induction is based explicitly on the recursive structure of the definition.

## 3.1 Some recursive definitions for data types

Fully parenthesized Boolean algebra expressions (with ands, ors, nots only).

**Definition 3.1** The set $F$ of fully parenthesized Boolean algebra expressions satisfies:

1. The symbols 0 and 1 are in $F$.
2. If $e$ and $e'$ are in $F$ then the expression $(e \wedge e')$ is in $F$.
3. If $e$ and $e'$ are in $F$ then the expression $(e \vee e')$ is in $F$.
4. If $e$ is in $F$ then the expression $(\neg e)$ is in $F$.

5. $F$ contains nothing else.

And here is one for binary trees.

**Definition 3.2** The set $T$ of binary trees satisfies:

1. A single node (just a dot) is in $T$.
2. If $t$ is in $T$, then the structure formed from $t$, a single new node, and an arrow labeled "left" from the new node to $t$, is also in $T$.
3. If $t$ is in $T$, then the structure formed from $t$, a single new node, and an arrow labeled "right" from the new node to $t$, is also in $T$.
4. If $t$ and $t'$ are in $T$, then the structure formed from $t$, $t'$, and a single new node, with an arrow labeled "left" from the new node to $t$ and another arrow labeled "right" from the new node to $t'$, is also in $T$.

Since the descriptions above are wordy, it's nice to introduce some notation, like makeleft(t) to denote the thing built up from $t$ in the first recursive step. Likewise, makeright(t), makeboth(t,t').

And here is one for strings over an alphabet $A$.

**Definition 3.3** The set $S$ of (finite length) strings over alphabet $A$ satisfies:

1. $\lambda$, the empty string, is in $S$.
2. If $s$ is in the set $S$ and $a \in A$, then $sa$ is in $S$.

3. $S$ contains nothing else.

This is a recursive description (definition) of the set of strings. Notice that it includes every (finite length) string. Moreover, every string has a unique derivation using these rules.

## 3.2 Inductive proofs based on recursive data type definitions

Structural induction is used to prove a property of all the elements of some recursively-defined data type. The proof consists of two steps:

- Prove for the "base cases" of the definition.
- Prove for the result of any combination rule, assuming that it is true for all the parts.

For Boolean expressions:

Prove: $\forall e \in F(P(e))$
1. (Base) $P(0)$
2. (Base) $P(1)$
3. (Inductive step) $\forall e, e' \in F(P(e) \wedge P(e') \Rightarrow P((e \wedge e')))$
4. (Inductive step) $\forall e, e' \in F(P(e) \wedge P(e') \Rightarrow P((e \vee e')))$
5. (Inductive step) $\forall e \in F(P(e) \Rightarrow P((\neg e)))$
6. QED Structural induction on Boolean expressions

For binary trees:

Prove: $\forall t \in T(P(t))$
1. (Base) $P(n)$, where $n$ is the single-node tree.
2. (Inductive step) $\forall t \in T(P(t) \Rightarrow P(makeleft(t)))$
3. (Inductive step) $\forall t \in T(P(t) \Rightarrow P(makeright(t)))$
4. (Inductive step) $\forall t, t' \in T(P(t) \wedge P(t') \Rightarrow P(makeboth(t, t')))$
5. QED — Structural induction on binary trees

For strings over alphabet $A = \{0, 1\}$:

Prove: $\forall s \in S(P(s))$
1. (Base) $P(\lambda)$
2. (Inductive step) $\forall s \in S(P(s) \Rightarrow P(s0))$
3. (Inductive step) $\forall s \in S(P(s) \Rightarrow P(s1))$
4. QED — Structural induction on strings.

You can see how this would extend to different alphabets.

An example for each of the data types:

**Theorem 3.4** Every Boolean expression has the same number of left and right parentheses.

*Proof.* Define $P(e)$: Expression $e$ has the same number of left and right parentheses.

Prove: $\forall e \in F(P(e))$
1. (Base) $P(0)$ — The expression 0 has no parens.
2. (Base) $P(1)$ — No parens
3. (Inductive step) $\forall e, e' \in F(P(e) \wedge P(e') \Rightarrow P((e \wedge e'))$
   1. Fix $e, e' \in F$.
   2. Assume $P(e) \wedge P(e')$, that is, each of $e$ and $e'$ has the same number of left and right parens.
   3. $P((e \wedge e'))$, that is, the expression $e \wedge e'$ has the same number of left and right parens. — By inductive hypothesis (3.2), and the fact that we're adding one left and one right paren.
   4. QED — Implication, UG
4. (Inductive step) $\forall e, e' \in F(P(e) \wedge P(e') \Rightarrow P((e \vee e'))$ — Similar to the previous case.
5. (Inductive step) $\forall e \in F(P(e) \Rightarrow P((\neg e))$ — Similar to the previous case.
6. QED — Structural induction on Boolean expressions

Be careful of confusion of parens above. Some are used for $P$, some are parts of the actual Boolean expressions...this really needs better notation. ∎

**Theorem 3.5** The number of edges in any binary tree is exactly one fewer than the number of nodes.

*Proof.* Define $P(t)$: Binary tree $t$ has exactly 1 fewer edge than nodes. We are using the notation $nodes(t)$, $edges(t)$. The key is that the inductive steps all preserve the "deficit" of 1. For instance, combining two trees involves adding two edges and one node; since each subtree had a "deficit" of 1, this maintains the deficit of 1:

Prove: $\forall t \in T(P(t))$
1. (Base) $P(n)$, where $n$ is the single-node tree. — 0 edges and 1 node
2. (Inductive step) $\forall t \in T(P(t) \Rightarrow P(makeleft(t))$
    1. Fix $t$.
    2. Assume $P(t)$, that is, $|edges(t)| + 1 = |nodes(t)|$.
    3. $P(makeleft(t))$, that is, $|edges(makeleft(t))| + 1 = |nodes(makeleft(t))|$.
        1. $|edges(makeleft(t))| = |edges(t)| + 1$
        2. $|nodes(makeleft(t))| = |nodes(t)| + 1$.
        3. QED — Algebra, using 2.2, 2.3.1, 2.3.2
    4. QED — Implication, UG
3. (Inductive step) $\forall t \in T(P(t) \Rightarrow P(makeright(t))$ — Similar to the previous case.
4. (Inductive step) $\forall t, t' \in T(P(t) \wedge P(t') \Rightarrow P(makeboth(t, t'))$
    1. Fix $t$.
    2. Assume $P(t) \wedge P(t')$, that is, $|edges(t)| + 1 = |nodes(t)|$ and $|edges(t')| + 1 = |nodes(t')|$.
    3. $P(makeboth(t, t'))$, that is, $|edges(makeboth(t, t'))| + 1 = |nodes(makeboth(t))|$.
        1. $|edges(makeboth(t, t'))| = |edges(t)| + |edges(t')| + 2$
        2. $|nodes(makeboth(t, t'))| = |nodes(t)| + |nodes(t')| + 1$.
        3. QED — Algebra, using 4.2, 4.3.1, and 4.3.2
    4. QED — Implication, UG
5. QED — Structural induction on binary trees

■

**Theorem 3.6** In a string of 0s and 1s, the number of occurrences of the pattern 01 is less than or equal to the number of occurrences of 10, plus one.

Let's try to prove this by structural induction. First we must define $P(s)$. Let's write $num(pat, s)$ as the number of occurrences of the pattern string pat in s. Now our inductive hypothesis is

$$P(s) : num(01, s) \leq num(10, s) + 1.$$

If you try to prove this by structural induction, you will get stuck. Why? Consider what happens when you add 1 at the end. This could increase the number of 01s without increasing the number of 10s.

So, to prove by structural induction on strings, let's strengthen the hypothesis by adding another clause. If a string ends in 0 then the number of 01s is less than or equal to the number of 10s. That solves the problem by weakening what we have to show when the string ends in 1. But maybe it causes another problem somewhere else. Let's give it a try:

Redefine $P(s)$:
$num(01, s) \leq num(10, s) + 1$,
and
If $s$ ends in 0 then $num(01, s) \leq num(10, s)$.

This means that, for each inductive step, we have two things to show:

Prove: $\forall s \in S(P(s))$
1. (Base) $P(\lambda)$ No patterns of either kind.
2. (Inductive step) $\forall s \in S(P(s) \Rightarrow P(s0))$
    1. Fix $s$.
    2. Assume $P(s)$.
    3. $P(s0)$
        1. $num(01, s0) \leq num(10, s0) + 1$. ???
        2. If $s0$ ends in 0 then $num(01, s0) \leq num(10, s0)$. ???
        3. QED Conjunction
    4. QED Implication, UG
3. (Inductive step) $\forall s \in S(P(s) \Rightarrow P(s1))$
    1. Fix $s$.
    2. Assume $P(s)$.
    3. $P(s1)$
        1. $num(01, s1) \leq num(10, s1) + 1$. ???
        2. If $s1$ ends in 0 then $num(01, s1) \leq num(10, s1)$. ???
        3. QED Conjunction
    4. QED Implication, UG
4. QED Structural induction on strings.

First let's consider $s1$. This is the case that looks dangerous, because it might increase the number of 01s. We have to prove two statements. The second is easy, because the new string doesn't end in 0. We say it's "vacuously true".

The first statement now takes some work. We might be adding to the number of 01s. However, if we do, the previous string must have ended with 0. Then the inductive hypothesis says that the previous string had to satisfy the stronger inequality in the second statement. Adding one to the LHS of the stronger inequality yields the weaker inequality we want.

The following proof fragment considers cases based on whether $s$ ends in 0 or not. If not, it might end in 1, or might be empty (don't forget this possibility).

3. $P(s1)$
    1. $num(01, s1) \leq num(10, s1) + 1$.
        1. If $s$ ends in 0 then $num(01, s1) \leq num(10, s1) + 1$.
            1. Assume $s$ ends in 0.
            2. $num(01, s) \leq num(10, s)$ Inductive hypothesis (3.2), part 2.
            3. $num(01, s1) = num(01, s) + 1$ Adding one more 01.
            4. $num(10, s1) = num(10, s)$
            5. $num(01, s1) \leq num(10, s1) + 1$. Algebra (combining 3.3.1.1.2, ...3, and ...4)

            QED Implication
        2. If $s$ ends in 1 then $num(01, s1) \leq num(10, s1) + 1$. Inductive hypothesis, part 1; no new 01s.
        3. If $s = \lambda$ then $num(01, s1) \leq num(10, s1) + 1$. $s1$ is just 1, which has no 01s.
        4. QED Cases
    2. If $s1$ ends in 0 then $num(01, s1) \leq num(10, s1)$.

Vacuously true, because $s1$ doesn't end in 0.

3. QED — Conjunction

Of course, you could also expand the step for $s$ ending in 1 into a careful series of inequalities.

Now consider $s0$. We hope that what we did to make the $s1$ case work doesn't mess up the $s0$ case. But We have to check.

The first statement is easy. It follows from the first statement of the inductive hypothesis for $s$, because we are not increasing the number of 01s.
But now the second statement takes more work. The difficulty is that the new string ends in 0, which means that we have to show the stronger inequality in the second statement. But to do this, we might only have the weaker inequality for the previous string. The argument again depends on what the previous string $s$ ended with. So again, we consider cases, based on whether $s$ ends in 0 or 1, or is empty.
If $s$ ends in 0 we rely on the second statement of the inductive hypothesis for $s$ (with the stronger inequality), whereas if $s$ ends in 1 we rely on the first statement (with the weaker inequality). In this case, we have to "turn the weaker inequality into the stronger inequality".

3. $P(s0)$
    1. $num(01, s0) \leq num(10, s0) + 1$. — Inductive hypothesis, part 1; no new 01s.
    2. If $s0$ ends in 0 then $num(01, s0) \leq num(10, s0)$.
        1. $num(01, s0) \leq num(10, s0)$.
            1. If $s$ ends in 0 then $num(01, s0) \leq num(10, s0)$.
               Inductive hypothesis, part 2; no new 01s
            2. If $s$ ends in 1 then $num(01, s0) \leq num(10, s0)$.
                1. Assume $s$ ends in 1.
                2. $num(01, s) \leq num(10, s) + 1$. — Inductive hypothesis, part 1
                3. $num(01, s0) = num(01, s)$
                4. $num(10, s0) = num(10, s) + 1$ — Exactly one new occurrence of 10.
                5. $num(01, s0) \leq num(10, s0)$ — Algebra
                6. QED — Implication
            3. If $s = \lambda$ then $num(01, s0) \leq num(10, s0)$.
               $s0$ is just 1, which has no 01s or 10s.
            4. QED — Cases
        2. QED — Propositional reasoning (truth table)
    3. QED — Conjunction

If you actually write out all these cases in the proof, you will notice that some facts are stated repeatedly, e.g., that when you add a 0 to the end of a string you are not increasing the number of 01s.
To avoid having to state these facts several times, you can move them earlier in the proof.

Note that these proofs aren't very different from ordinary inductive (or strong inductive) proofs. In fact, you can prove all the same theorems by strong induction on the number of "formation steps"—that is, the number of times you have to apply the recursive construction rules to get the object you are proving about. The base cases correspond to 0 applications of combination operations.

### 3.3 Recursively-defined functions on recursively-defined data types

Recursive definitions provide a natural way to define functions whose domains are recursively-defined data types.

**Example 3.7** (Boolean Expressions) Define $eval$, a function from Boolean expressions to $\{0,1\}$, by:
$eval(0) = 0$
$eval(1) = 1$
$eval((e \wedge e')) = 1$ if $eval(e) = eval(e') = 1$, 0 otherwise.
$eval((e \vee e')) = 0$ if $eval(e) = eval(e') = 0$, 1 otherwise.
$eval((\neg e)) = 0$ if $eval(e) = 1$, 1 otherwise.

**Example 3.8** (Binary Trees) Define the function $numnodes(t)$ (a formal definition of what we understand intuitively to be the number of nodes in the tree $t$), by:

$numnodes(\text{single node}) = 1$
$numnodes(makeleft(t)) = numnodes(t) + 1$
$numnodes(makeright(t)) = numnodes(t) + 1$
$numnodes(makeboth(t, t')) = numnodes(t) + numnodes(t') + 1$

Define $numedges(t)$:
$numedges(\text{single node}) = 0$
$numedges(makeleft(t)) = numedges(t) + 1$
$numedges(makeright(t)) = numedges(t) + 1$
$numedges(makeboth(t)) = numedges(t) + numedges(t') + 2$

We can use structural induction to prove properties of functions defined in this way. For example, we can reprove the relationship between numbers of nodes and edges in terms of the formal functions $numnodes$ and $numedges$. (The argument is pretty much the same as before, just using the formal function definitions instead of relying on our understanding of the sizes of sets.)

Define $P(t)$: $numnodes(t) = numedges(t) + 1$.

Prove: $\forall t \in T(P(t))$
1. (Base) $P(t_0)$, where $t_0$ is the single-node tree. By definition, $numnodes(t_0) = 1$ and $numedges(t_0) = 0$
2. (Inductive step) $\forall t \in T(P(t) \Rightarrow P(makeleft(t))$
   1. Fix $t$.
   2. Assume $P(t)$, that is, $numedges(t) + 1 = numnodes(t)$.
   3. $P(makeleft(t))$, that is, $numedges(makeleft(t)) + 1 = numnodes(makeleft(t))$.
      1. $numedges(makeleft(t)) = numedges(t) + 1$ Definition of $numedges$
      2. $numnodes(makeleft(t)) = numnodes(t) + 1$ Definition of $numnodes$

        3. QED — Algebra
    4. QED
3. (Inductive step) $\forall t \in T(P(t) \Rightarrow P(\mathit{makeright}(t))$ — Similar to the previous case.
4. (Inductive step) $\forall t, t' \in T(P(t) \wedge P(t') \Rightarrow P(\mathit{makeboth}(t,t'))$
    1. Fix $t$.
    2. Assume $P(t) \wedge P(t')$, that is,
       $\mathit{numedges}(t) + 1 = \mathit{numnodes}(t)$ and $\mathit{numedges}(t') + 1 = \mathit{numnodes}(t')$.
    3. $P(\mathit{makeboth}(t,t'))$, that is, $\mathit{numedges}(\mathit{makeboth}(t,t')) + 1 = \mathit{numnodes}(\mathit{makeboth}(t))$.
        1. $\mathit{numedges}(\mathit{makeboth}(t,t')) = \mathit{numedges}(t) + \mathit{numedges}(t') + 2$
           Definition of *numedges*
        2. $\mathit{numnodes}(\mathit{makeboth}(t,t')) = \mathit{numnodes}(t) + \mathit{numnodes}(t') + 1$
           Definition of *numnodes*
        3. QED — Algebra
    4. QED
5. QED — Structural induction on binary trees