COMP30026 Models of Computation

Induction Principles

Bach Le / Anna Kalenkova

Lecture Week 5 Part 2 (Zoom)

Semester 2, 2021

This Lecture is Being Recorded

Mathematical Induction

"Mathematical" induction is always a proof about the natural numbers, $\mathbb{N}$.

We're usually given a statement "for all n, $S(n)$."

We proceed in two steps.

1. In the basis step, we show $S(0)$.
2. In the inductive step, we take $S(n)$ as the induction hypothesis and use it to establish $S(n+1)$.

Proof by Induction

Theorem: For all $n \geq 0$,

$$\sum_{i=1}^{n} i^2 = \frac{n(n+1)(2n+1)}{6}$$

Proof: For the basis step, note that the statement is true for $n = 0$.

For the inductive step, assume the statement is true for some fixed n, and we shall show that it also holds true with $n+1$ substituted for n.

So the statement to prove is

$$\sum_{i=1}^{n+1} i^2 = \frac{(n+1)(n+2)(2n+3)}{6}$$

Proof by Induction

But the claim

$$\sum_{i=1}^{n+1} i^2 = \frac{(n+1)(n+2)(2n+3)}{6}$$

is the same as

$$\left(\sum_{i=1}^{n} i^2\right) + (n+1)^2 = \frac{(n+1)(n+2)(2n+3)}{6}$$

By the induction hypothesis, it suffices to show that

$$\frac{n(n+1)(2n+1)}{6} + (n+1)^2 = \frac{(n+1)(n+2)(2n+3)}{6}$$

This is done by simple polynomial algebra.

More General Induction

Sometimes more base cases may be needed.

Sometimes we need to use several statements $S(j), \dots, S(n)$ to establish $S(n+1)$.

Theorem: For all $n \geq 8$, n can be written as a sum of 3s and 5s.

Proof: For the basis step, observe that $S(8)$, $S(9)$, and $S(10)$ are true.

For the inductive step, assume that $n \geq 10$ and $S(8), \dots, S(n)$ are true. Since $S(n-2)$ is true, also $n+1$ can be written as a sum of 3s and 5s – just add 3 to the sum we had for $n-2$. Hence we have established $S(n+1)$.

We conclude that $S(n)$ holds for all $n \geq 8$.

Course-of-Values Induction

We can take the generality of "general induction" all the way:

To prove some claim $P(n)$, we are allowed to take the entire conjunction

$$P(0) \wedge P(1) \wedge \cdots \wedge P(n-1)$$

as our induction hypothesis.

This variant is called course-of-values induction.

At first it looks like performing induction without a base case!

But the base case is implicitly included in the inductive step, because we have to prove $P(0)$ from nothing, that is, from *true*, the empty conjunction.

Recursive Structure and Induction

We often deal with recursively defined objects. Lists and trees are examples.

The set of well-formed propositional logic formulas is another example.

We will later meet context-free grammars; the language defined by such a grammar is a third example.

Induction is the natural way of proving assertions about such objects.

In many cases we then rely on structural induction.

Structural Induction: An Example

Consider the Haskell type Exp defined like so:

```
data Exp = AND Exp Exp | NOT Exp | VAL
```



```
              NOT
               |
              AND
            /     \
        NOT         AND
         |         /    \
        AND      VAL     AND
       /   \            /   \
    VAL     VAL      NOT     VAL
                      |
                     VAL
```

On the left is an example tree.

For any such tree, let

- a be the number of AND nodes,
- n be the number of NOT nodes,
- v be the number of VAL nodes.

I claim that $v = a + 1$, always.

Structural Induction: An Example

The claim $v = a + 1$ applies to all trees that are inhabitants of `Exp`.

The definition of `Exp` told us that there are only three possible forms we need to deal with:

1. the tree `VAL`,
2. a tree of form (`AND` t_1 t_2), where t_1 and t_2 are trees,
3. a tree of form (`NOT` t), where t is a tree.

The first is a base case for induction.

It is straight-forward to prove $v = a + 1$ for the base case, since for `VAL`, a is 0 and v is 1.

Structural Induction: An Inductive Case

For the inductive case AND t_1 t_2, we proceed by assuming that the inductive hypothesis holds for t_1 and t_2.

That is, if the number of AND nodes in t_1 and t_2 is a_1 and a_2, respectively, and the number of VAL nodes is v_1 and v_2, respectively, then we make the assumptions $v_1 = a_1 + 1$ and $v_2 = a_2 + 1$.

To get the number a of AND nodes in AND t_1 t_2, we need to add the number of AND nodes in t_1 and t_2, and then add 1: $a = a_1 + a_2 + 1$.

To get the number of VAL nodes, we just have to add the number of VAL nodes in t_1 and t_2: $v = v_1 + v_2$.

So $v = v_1 + v_2 = a_1 + 1 + a_2 + 1 = a + 1$. Just as we claimed!

Structural Induction: A Second Inductive Case

The case of `NOT` t is even simpler.

Clearly the number of `AND` nodes in `NOT` t is the same as the number in t, and similarly for the `VAL` nodes.

So again we have that $v = a + 1$.

Since we have established that $v = a + 1$ in each of the three cases, we conclude that it really is an invariant; it must hold for all possible `Exp` trees.

Structural and Mathematical Induction

Structural induction is a natural generalisation of course-of-values mathematical induction.

In Haskell we could mimic the natural numbers with this definition:


```
data Natural = SUCCESSOR Natural | ZERO
```

Then structural induction over this type corresponds exactly to course-of-values induction.

Conversely,if you prefer mathematical induction, we could have shown $v = a + 1$ for the `Exp` trees, by doing induction on the height of the trees.

Next Week

We shall take another helping of mathematical vegetables (a large dish of sets, functions and relations).

This will be our sustenance for the remaining parts of the course, namely automata, formal language theory, and computability.