Notes on Inductive Sets and Induction

Finite Automata Theory and Formal Languages – TMV027/DIT321

Ana Bove, March 15th 2018

Contents

In the course we work with mathematical (or simple) induction and with course-of-values (or strong) induction. These are two *equivalent* inductive principles over the (inductive) set of Natural numbers.

We also see that mathematical induction is just a special case of structural induction. Indeed, any inductively defined set, as for example the Natural numbers, has an associated inductive principle that allows us to prove properties about the elements of the set.

In what follows we look into all these concepts in more detail.

1 Induction over the Natural Numbers

We know that the set $\mathbb{N}$ of Natural numbers is the infinite set containing 0, 1, 2, 3, ... This infinite set can be defined in a very elegant and concrete way with the following two rules:

$$\frac{}{0 \in \mathbb{N}} \qquad\qquad \frac{n \in \mathbb{N}}{\mathsf{suc}\ n \in \mathbb{N}}$$

The first rule says that there is an element, which we call 0, that is a Natural number. The second rule tells that we can construct a new Natural number $\mathsf{suc}\ n$ from a given Natural number n. Here, both 0 and suc are called *constructors*.

Which are the elements in $\mathbb{N}$ constructed by these rules? Clearly 0 is in $\mathbb{N}$ by the first rule. Then $\mathsf{suc}\ 0$ (usually denoted as 1) is in $\mathbb{N}$ by the second rule; now $\mathsf{suc}\ (\mathsf{suc}\ 0)$ (usually denoted as 2) is also in $\mathbb{N}$; further $\mathsf{suc}\ (\mathsf{suc}\ (\mathsf{suc}\ 0))$ (usually denoted as 3) is in $\mathbb{N}$; and we infinitely can go on like this constructing a new element in $\mathbb{N}$ from the one we just constructed. In practice, the Natural number that we denote n is constructed by applying the suc constructor n times to the element 0, that is $\mathsf{suc}^n\ 0$. Hence, each element in the set $\mathbb{N}$ is constructed after applying the rules defining the set a *finite* number of times. So here we have an infinite set where each element is finite.

As we will see later, the set of Natural numbers is an inductively defined set. Any set which is inductively defined contains only finite elements. The set itself can be finite or infinite. We will discuss more about this kind of sets in section 2.1.

In what follows, we simply write $n + 1$ to denote $\mathsf{suc}\ n$.

1.1 Mathematical (Simple) Induction

Now that we know what the Natural numbers are, we can see how to prove a certain property P over the set $\mathbb{N}$, in other words, how to prove $\forall\, n \in \mathbb{N}.\, P(n)$.

The mathematical (or simple) induction principle gives us the mean to do this:

$$\frac{\overbrace{P(0)}^{\text{base case}} \qquad \overbrace{\forall\, n \in \mathbb{N}.\, \overbrace{P(n)}^{\text{IH}} \Rightarrow P(n+1)}^{\text{inductive step}}}{\underbrace{\forall\, n \in \mathbb{N}.\, P(n)}_{\text{statement to prove}}}$$

where IH stands for "inductive hypothesis".

Let us analyse what the principle says: if we have a proof that the property P holds for 0, that is, $P(0)$ is true, and we have a method to prove $P(n+1)$ from a proof of $P(n)$ for a given Natural number n, then we have a way to prove the property P for ANY element in $\mathbb{N}$.

Let us try to understand why this principle indeed proves $\forall\, n \in \mathbb{N}.\, P(n)$. To prove that the property P is valid for any Natural number n we start by proving $P(0)$. After we have a proof of $P(0)$ we can prove $P(1)$ because we have a method that proves $P(n+1)$ from a proof of $P(n)$ for any n, in particular for 0. Now we use $P(1)$ to prove $P(2)$ using this same method. Once we have a proof of $P(2)$ we can prove $P(3)$, then $P(4)$, and so on. In this way we can prove $P(n)$ for any Natural number n!

There are cases where a certain property P is not really valid for ALL Natural numbers, but only for those numbers greater than a certain $i \in \mathbb{N}$. It could also be that we first need to prove P for a few cases before we can give the method that proves $P(n+1)$ from a proof of $P(n)$ for a given n. Hence, a more general presentation of the principle of mathematical induction is the following:

$$\frac{\overbrace{P(i), P(i+1), \ldots, P(j)}^{\text{base cases}} \qquad \overbrace{\forall\, n \in \mathbb{N}.\, j \leqslant n \Rightarrow (\overbrace{P(n)}^{\text{IH}} \Rightarrow P(n+1))}^{\text{inductive step}}}{\forall\, n \in \mathbb{N}.\, i \leqslant n \Rightarrow P(n)}$$

Let us now look at the use of the principle of mathematical induction in more detail with the help of some examples. In what follows, we simply write $\forall\, n \geqslant i.\, P(n)$ instead of $\forall\, n \in \mathbb{N}.\ i \leqslant n \Rightarrow P(n)$. In the examples, we use both $n \times m$ and nm to denote the multiplication of n and m indistinctly, depending on which notation might be more clear at each point.

1.1.1 $\sum_{i=0}^{i=n} i(i+1) = \frac{n(n+1)(n+2)}{3}$

We define $P(n)$ to be $\sum_{i=0}^{i=n} i(i+1) = \frac{n(n+1)(n+2)}{3}$ and we prove $\forall\, n \in \mathbb{N}.\, P(n)$ by mathematical induction (on the number n).

Our base case is $n = 0$. Then, $P(0)$ amounts to proving that $\sum_{i=0}^{i=0} i(i+1) = \frac{0(0+1)(0+2)}{3}$, hence $0(0+1) = \frac{0(0+1)(0+2)}{3}$, that is, $0 = 0$ which is indeed true.

In our inductive step we need to prove that if the statement is true for a given n, then it will also be true for $n+1$. In other words, for a given n, we need to show that if our inductive hypothesis IH $P(n)$ holds then $P(n+1)$ also holds.

Our IH is that $\sum_{i=0}^{i=n} i(i+1) = \frac{n(n+1)(n+2)}{3}$ is true.

We now need to prove $P(n+1)$, that is, $\sum_{i=0}^{i=n+1} i(i+1) = \frac{(n+1)(n+2)(n+3)}{3}$.

We know that $\sum_{i=0}^{i=n+1} i(i+1) = \sum_{i=0}^{i=n} i(i+1) + (n+1)(n+2)$. Using our IH we obtain

$$\sum_{i=0}^{i=n+1} i(i+1) = \frac{n(n+1)(n+2)}{3} + (n+1)(n+2).$$

Using mathematical properties we get that

$$\sum_{i=0}^{i=n+1} i(i+1) = \frac{n(n+1)(n+2)}{3} + \frac{3(n+1)(n+2)}{3} = \frac{(n+1)(n+2)(n+3)}{3},$$

which shows that $P(n+1)$ is true.

This concludes our proof by mathematical induction.

1.1.2 $\forall n \geqslant 7.\, n! > 3^n$

Recall that the factorial function $n!$ is (recursively) defined by the following equations

$$0! = 1 \qquad (n+1)! = (n+1) \times n!$$

(more on recursively defined functions in section 2.3).

We define $P(n)$ to be $n! > 3^n$ and we prove $\forall n \geqslant 7.\, P(n)$ by mathematical induction (on the number n).

Our base case is $n = 7$. Then, $P(7)$ amounts to proving that $7! > 3^7$, that is, $7 \times 6 \times 5 \times 4 \times 3 \times 2 \times 1 \times 0 = 5040 > 2187 = 3^7$, which is indeed true.

In our inductive step we need to prove that if the statement is true for a given $n \geqslant 7$, then it will also be true for $n+1$. In other words, for a given $n \geqslant 7$, we need to show that if our IH $P(n)$ holds then $P(n+1)$ also holds.

Our IH is that $n! > 3^n$ is true, for $n \geqslant 7$.

We now need to prove $P(n+1)$ for $n \geqslant 7$, that is, $(n+1)! > 3^{n+1}$.

Using the definition of factorial, our IH and mathematical properties we get that

$$(n+1)! = (n+1)n! > (n+1)3^n \geqslant (7+1)3^n = 8 \times 3^n > 3 \times 3^n = 3^{n+1},$$

which shows that $P(n+1)$ is true.

This concludes our proof by mathematical induction.

1.1.3 $\forall n \geqslant 3.\, n^2 < 4^{n-1}$

We define $P(n)$ to be $n^2 < 4^{n-1}$ and we prove $\forall n \geqslant 3.\, P(n)$ by mathematical induction (on the number n).

Our base case is $n = 3$. Then, $P(3)$ amounts to proving that $3^2 < 4^2$, that is, $9 < 16$, which is indeed true.

In our inductive step we need to prove that if the statement is true for a given $n \geqslant 3$, then it will also be true for $n+1$. In other words, for a given $n \geqslant 3$, we need to show that if our IH $P(n)$ holds then $P(n+1)$ also holds.

Our IH is that $n^2 < 4^{n-1}$ is true for $n \geqslant 3$.

We now need to prove $P(n+1)$ for $n \geqslant 3$, that is, $(n+1)^2 < 4^{n+1-1}$, in other words, $n^2 + 2n + 1 < 4^n$.

Observe that for $n \geqslant 3$ we have that $2n < 2n^2$ and $1 < n^2$ (these facts can easily be proved by mathematical induction as well). Using these inequalities, our IH and mathematical properties we get that $n^2 + 2n + 1 < n^2 + 2n^2 + n^2 = 4n^2 < 4 \times 4^{n-1} = 4^n$, which shows that $P(n+1)$ is true.

This concludes our proof by mathematical induction.

1.2 Course-of-Values (Strong) Induction

We could also use the inductive principle known as course-of-values (or strong) induction in order to prove that $\forall\, n \in \mathbb{N}.\, P(n)$. This principle states the following:

$$\frac{\overbrace{P(0)}^{\text{base case}} \qquad \overbrace{\forall\, n \in \mathbb{N}.\ (\overbrace{\forall m \in \mathbb{N}.\ 0 \leqslant m \leqslant n \Rightarrow P(m)}^{\text{IH}}) \Rightarrow P(n+1)}^{\text{inductive step}}}{\underbrace{\forall\, n \in \mathbb{N}.\, P(n)}_{\text{statement to prove}}}$$

This principle is similar to the previous one but the method to prove $P(n+1)$ for a certain Natural number n can not only use the proof of $P(n)$ but all proofs $P(0), P(1), \ldots, P(n)$! That is, when proving $P(n+1)$ we can use the fact that P holds for ALL Natural numbers m which are smaller or equal to n, that is, $0 \leqslant m \leqslant n$. Therefore, instead of just having an inductive hypothesis $P(n)$, we have *several* inductive hypotheses $P(0), P(1), \ldots, P(n)$.

Let us try to understand why this principle indeed proves $\forall\, n \in \mathbb{N}.\, P(n)$. To prove that the property P is valid for any Natural number n we start by proving $P(0)$. After we have a proof of $P(0)$ we can prove $P(1)$ because we have a method that proves $P(n+1)$ from the proofs that P holds for all Natural numbers m which are smaller or equal to n, in particular for 0. Now we use $P(0)$ and $P(1)$ to prove $P(2)$ using this same method. Once we have the proofs of $P(0), P(1)$ and $P(2)$ we can prove $P(3)$, then $P(4)$, and so on. In this way we can prove $P(n)$ for any Natural number n!

At first sight one could get the impression that the course-of-values principle is more powerful than that of mathematical induction. However, one can actually prove that these principles are *equivalent*.

There are again cases where a certain property P is not really valid for ALL Natural numbers but only for those numbers greater than a certain $i \in \mathbb{N}$. It could also be that we first need to prove P for a few cases before we can give the method that proves $P(n+1)$ from the proofs of $P(0), P(1), \ldots, P(n)$, for a given n. Hence, a more general presentation of the principle of course-of-values induction is the following:

$$\frac{\overbrace{P(i), P(i+1), \ldots, P(j)}^{\text{base cases}} \qquad \overbrace{\forall\, n \in \mathbb{N}.\ j \leqslant n \Rightarrow (\overbrace{\forall m.\ i \leqslant m \leqslant n \Rightarrow P(m)}^{\text{IH}}) \Rightarrow P(n+1)}^{\text{inductive step}}}{\forall\, n \in \mathbb{N}.\ i \leqslant n \Rightarrow P(n)}$$

Let us now look at the use of the principle of course-of-values induction in more detail with the help of some examples.

1.2.1 $\forall\, n \geqslant 12.\, \exists\, i, j \in \mathbb{N}.\, n = 4 \times i + 5 \times j$

We define $P(n)$ to be $\exists\, i, j \in \mathbb{N}.\, n = 4 \times i + 5 \times j$ and we prove $\forall\, n \geqslant 12.\, P(n)$ by course-of-values induction (on the number n).

Our base cases are $n = 12, 13, 14, 15$. We proceed to prove $P(12), P(13), P(14)$ and $P(15)$.

For $P(12)$ we have that $\exists\, 3, 0 \in \mathbb{N}.\, 12 = 4 \times 3 + 5 \times 0$.

For $P(13)$ we have that $\exists\, 2, 1 \in \mathbb{N}.\, 13 = 4 \times 2 + 5 \times 1$.

For $P(14)$ we have that $\exists\, 1, 2 \in \mathbb{N}.\, 14 = 4 \times 1 + 5 \times 2$.

For $P(15)$ we have that $\exists\, 0, 3 \in \mathbb{N}.\, 15 = 4 \times 0 + 5 \times 3$.

In our inductive step we need to prove that if the statement is true for $12, 13, \ldots, n$ for a given $n \geqslant 15$, then it will also be true for $n+1$. In other words, we need to show that if our inductive hypotheses (IH) $P(12), P(13), \ldots, P(n)$ hold then $P(n+1)$ also holds, for a given $n \geqslant 15$.

For each m such that $12 \leqslant m \leqslant n$, our IH is that $\exists\, i, j \in \mathbb{N}.\, m = 4 \times i + 5 \times j$.

We now need to prove $P(n+1)$ for $n \geqslant 15$, that is, $\exists\, i, j \in \mathbb{N}.\, n+1 = 4 \times i + 5 \times j$.

Since $n \geqslant 15$ then $n+1 \geqslant 16$ and $n+1 > (n+1) - 4 \geqslant 16 - 4 = 12$. So our IH applies to $(n+1) - 4$ and then we know that $\exists\, i, j \in \mathbb{N}.\, (n+1) - 4 = 4 \times i + 5 \times j$.

Hence $\exists\, (i+1), j \in \mathbb{N}.\, n+1 = 4 \times (i+1) + 5 \times j$, which shows that $P(n+1)$ is true.

This concludes our proof by course-of-values induction.

1.2.2 Fundamental Theorem of Arithmetics

The fundamental theorem of arithmetic, also called the unique prime factorisation theorem, states that every Natural number $n > 1$ is either a prime number (that is, its only factors are 1 and n) or it can be represented as the product of prime numbers. We prove this theorem here with the help of course-of-values induction. (Actually, the fundamental theorem of arithmetics also states that this factorisation is unique but we will not look into this here.)

Observe that any factor of a Natural number cannot be larger than the number itself, which means that any non-prime number n always has factors that are larger than 1 but strictly smaller than n.

We define $P(n)$ to be that either n is prime or n can be represented as the product of prime numbers, and we prove $\forall\, n \geqslant 2.\, P(n)$ by course-of-values induction (on the number n).

Our base case is $n = 2$. Since 2 is a primer number then $P(2)$ is true.

In our inductive step we need to prove that if the statement is true for $2, 3, \ldots, n$ for a given $n \geqslant 2$, then it will also be true for $n+1$. In other words, we need to show that if our IH $P(2), P(3), \ldots, P(n)$ hold then $P(n+1)$ also holds, for a given $n \geqslant 2$.

For each m such that $2 \leqslant m \leqslant n$, our IH is that m is either a primer number or it can be represented as the product of prime numbers.

We now need to prove $P(n+1)$, that is, we need to show that either $n+1$ is a prime number or $n+1$ can be represented as the product of primer numbers.

We have two cases here. If $n+1$ is prime, then $P(n+1)$ is true and we are done.

If $n+1$ is not prime then it should be the case that $n+1$ can be written as the product of two Natural number $2 \leqslant i, j < n$, that is, $n+1 = i \times j$. So our IH applies to both i and j, and we know that both i and j are either prime numbers or they can be represented as the product of prime numbers.

So $i = p_{i_1} \times p_{i_2} \times \ldots \times p_{i_k}$ and $j = p_{j_1} \times p_{j_2} \times \ldots \times p_{j_{k'}}$ where each p_{i_h} and each $p_{j_{h'}}$ is a primer number. Observe that if i and/or j are themselves prime numbers then they can be represented as the product of one prime number (themselves).

Now we have that $n+1 = i \times j = p_{i_1} \times p_{i_2} \times \ldots \times p_{i_k} \times p_{j_1} \times p_{j_2} \times \ldots \times p_{j_{k'}}$ where each p_{i_h} and each $p_{j_{h'}}$ is a primer number, which shows that $P(n+1)$ is true.

This concludes our proof by course-of-values induction.

1.2.3 Proving Properties of Grammars

Course-of-values induction comes very handy when we need to prove properties about the language generated by a grammar, where a useful technique is to reason over the length of the derivation of a certain word w in the language of the grammar.

Let us consider the language generated by the grammar

$$S \to SabaS \mid aSbSa \mid aba$$

We want to prove that any occurrence of a b in a word generated by the grammar is immediately preceded and followed by an a.

We define $P(n)$ to be if $S \Rightarrow^n w$ then w starts and ends with an a and any b in the word is immediately preceded and followed by an a, and we prove $\forall\, n \geqslant 1.\, P(n)$ by course-of-value induction (on the length n of the derivation of a word, $S \Rightarrow^n w$).

Observe that this is not exactly the property we need to prove. However, the property we wish to prove can easily be deduced from the property P once we prove that $P(n)$ is valid for all n.

Our base case is $n = 1$ since no word can be derived from S in 0 steps. In other words, the shortest derivation from the starting variable S to a word takes at least one step. So if w has been derived in one step, $S \Rightarrow w$, then the only production that could have been used for such a derivation should have been $S \to aba$ and then $w = aba$. Here, aba starts and ends with a and the only b in the words is clearly placed between two a's. So $P(1)$ is true.

In our inductive step we need to prove that if the statement is true for $1, 2, \ldots, n$ for a given $n \geqslant 1$, then it will also be true for $n + 1$. In other words, we need to show that if our IH $P(1), P(2), \ldots, P(n)$ hold then $P(n + 1)$ also holds, for a given $n \geqslant 1$.

Our IH is that if a word is derived in at most n steps, that is $S \Rightarrow^m w$ for $1 \leqslant m \leqslant n$, then w starts and ends with an a and any b in the word is immediately preceded and followed by an a.

We now need to prove $P(n + 1)$, that is, if $S \Rightarrow^{n+1} w$ with $n \geqslant 1$, then w starts and ends with an a and any b in the word is immediately preceded and followed by an a.

Since $n \geqslant 1$ then $n + 1 > 1$, so the first step in the derivation should have used the production $S \to SabaS$ or the production $S \to aSbSa$.

If the first step in the derivation used the production $S \to SabaS$ then we have that $S \Rightarrow SabaS \Rightarrow^n w$. Here we know that $w = w_1abaw_2$ with $S \Rightarrow^i w_1$, $S \Rightarrow^j w_2$, and $1 \leqslant i, j \leqslant n$ (actually we also know that $i + j = n$). Then our IH applies to both i and j, so we know that both w_1 and w_2 start and end with an a and any b in the these words is immediately preceded and followed by an a.

Since by the IH $P(i)$ w_1 starts with a then so does w, and since by the IH $P(j)$ w_2 ends with a, then so does w. Any b in w is either a b in w_1 or in w_2 and hence by our IH the b is immediately preceded and followed by an a, or is the b just added in the first step of the derivation, which is clearly immediately preceded and followed by an a.

If the first step in the derivation used the production $S \to aSbSa$ then we have that $S \Rightarrow aSbSa \Rightarrow^n w$. Here we know that $w = aw_1bw_2a$ with $S \Rightarrow^i w_1$, $S \Rightarrow^j w_2$, and $1 \leqslant i, j \leqslant n$ (again, we actually know that $i + j = n$). Then our IH applies to both i and

j, so we know that both w_1 and w_2 start and end with an a and any b in the these words is immediately preceded and followed by an a.

In this case w clearly starts and ends with an a. Any b in w is either a b in w_1 or in w_2 and hence by our IH the b is immediately preceded and followed by an a, or is the b just added in the first step of the derivation which is located between w_1 and w_2. Since by the IH $P(i)$ w_1 ends with an a then this b is immediately preceded by an a; since by the IH $P(j)$ w_2 starts with an a, then this b is immediately followed by an a.

In both cases, we have showed that $P(n+1)$ is true.

This concludes our proof by course-of-values induction.

Discussion note: If we try to performe this proof by mathematical (instead of course-of-values) induction we would get stuck in the inductive step.

Consider the case where the first step in the derivation used the production $S \to SabaS$ and we have that $S \Rightarrow SabaS \Rightarrow^n w$. Here we know that $w = w_1abaw_2$ with $S \Rightarrow^i w_1$, $S \Rightarrow^j w_2$ such that $i + j = n$. Since $1 \leqslant i, j$ then $i, j < n$! This means that our IH $P(n)$ is of no help here, since neither w_1 nor w_2 are derived in n steps!

In general, when several variables take place in the right-hand side of a production, we do not really know how long the derivations of the sub-words are, only than they are strictly shorter than the length of the derivation of whole word. Therefore we usually must use course-of-values induction and not mathematical induction.

2 Inductive Sets and Recursive Functions

2.1 Inductively Defined Sets

If we analyse the rules defining the Natural numbers given in section 1 we can distinguish two kinds of rules: the rule that directly constructs a Natural number, and the rule that takes an existing Natural number and constructs a new one from it.

In general, the rules defining an inductive set can be divided into such two groups: those rules directly constructing elements in the set without the need of previous elements in the set, and those rules constructing new elements in the set from already existing elements in the set. Either of these rules could however make use of elements in other (already defined) sets A's.

More concretely, the general structure of the rules defining an inductive set S are as follow:

$$\frac{x_1 \in A_1 \ \cdots \ x_{j_1} \in A_{j_1}}{\mathsf{e}_1[x_1, \ldots, x_{j_1}] \in S} \qquad \cdots \qquad \frac{x_1 \in A_1 \ \cdots \ x_{j_m} \in A_{j_m}}{\mathsf{e}_\mathsf{m}[x_1, \ldots, x_{j_m}] \in S}$$

$$\frac{x_1 \in A_1 \ \cdots \ x_{j'_1} \in A_{j'_1} \ s_1, \ldots, s_{k_1} \in S}{\mathsf{c}_1[x_1, \ldots, x_{j'_1}, s_1, \ldots, s_{k_1}] \in S} \qquad \cdots \qquad \frac{x_1 \in A_1 \ \cdots \ x_{j_n} \in A_{j_n} \ s_1, \ldots, s_{k_n} \in S}{\mathsf{c}_\mathsf{n}[x_1, \ldots, x_{j_n}, s_1, \ldots, s_{k_n}] \in S}$$

Every element in an inductive set is defined by applying the rules above a finite number of times. Hence all elements in the set are finite. Moreover, any element in an inductive set shall only be defined via these rules, that is, there are no elements in the set constructed by any other means. This is sometimes refers to as *clousure* of the set.

When we define a set S with the rules above, the elements $s_1, \ldots, s_{k_i} \in S$ are said to be *structurally smaller* than the element $\mathsf{c_i}[x_1, \ldots, x_{j_i}, s_1, \ldots, s_{k_i}] \in S$. If we look at the structure of $\mathsf{c_i}[\ldots]$ we then can see that all $s_1, \ldots, s_{k_i}$ are strict sub-terms of $\mathsf{c_i}[\ldots]$. Each element $s_1, \ldots, s_{k_i}$ in itself is a constructor $\mathsf{c_{i'}}$ applied to other structurally smaller elements, or simply en element of the form $\mathsf{e_{i'}}[\ldots]$.

Discussion note: The set S we define could actually be a *parametric* set (a set that depends on another set), which would allow us for example to defined the finite lists of elements of a certain set A. We will not try to formalise this particular feature in the rules connected to the definition of an inductive set. We hope however that the rules presented above and your experience with parametric types from the work in programming courses will allow you to define inductive sets that are parametric. This particular feature is however orthogonal to the concepts we want to treat in here, so no major problems if you keep yourselves to non-parametric inductive sets.

One could also *mutually* define two or more inductive sets, but we will not look into this here.

Discussion note: One can of course also define sets containing infinite elements. Such definitions are more complex and we will not study them here.

2.2 Example of Inductively Defined Sets

One of the most well-known inductive sets in computer science, besides that of the Natural numbers, is the set Bool of Boolean values:

$$\frac{}{\mathsf{true} \in \mathsf{Bool}} \qquad \frac{}{\mathsf{false} \in \mathsf{Bool}}$$

This set only has elements that can be directly constructed, that is, there is no element which is constructed from a previous one. So this set is not *recursively* defined since no element in the set is constructed from another element in the set. Often however one associates inductive sets just to those sets which have at least a constructor that constructs new elements in the set from previous ones, like in the sets of Natural numbers, lists or trees.

Another well-known inductive set is set of finite lists of Natural numbers:

$$\frac{}{\mathsf{nil} \in \mathsf{List}\mathbb{N}} \qquad \frac{n \in \mathbb{N} \quad xs \in \mathsf{List}\mathbb{N}}{n : xs \in \mathsf{List}\mathbb{N}}$$

The definition of the parametric set of finite lists of elements in the set A is very similar to that of list of Natural numbers:

$$\frac{}{\mathsf{nil} \in \mathsf{List}\ A} \qquad \frac{x \in A \quad xs \in \mathsf{List}\ A}{n : xs \in \mathsf{List}\ A}$$

Here we overload the list constructors so they define both lists of Natural numbers and parametric lists over a set A.

Binary trees with information in the nodes and no information in their leaves can be defined as follows:

$$\frac{}{() \in \mathsf{Tree}\ A} \qquad \frac{x \in A \quad t_1, t_2 \in \mathsf{Tree}\ A}{\mathsf{node}\,(t_1, x, t_2) \in \mathsf{Tree}\ A}$$

Arithmetic expressions over the Natural numbers and also containing variables could be defined as follows:

$$\frac{n \in \mathbb{N}}{\mathsf{num}\,(n) \in \mathsf{Exp}} \qquad \frac{x \in \mathsf{String}}{\mathsf{var}\,(x) \in \mathsf{Exp}} \qquad \frac{e_1, e_2 \in \mathsf{Exp}}{e_1 \oplus e_2 \in \mathsf{Exp}} \qquad \frac{e_1, e_2 \in \mathsf{Exp}}{e_1 \otimes e_2 \in \mathsf{Exp}}$$

Here we took the liberty to use some of the constructors in an infix way, so we write $e_1 \oplus e_2 \in \mathsf{Exp}$ instead of $\oplus(e_1, e_2) \in \mathsf{Exp}$.

The set of regular expressions RE over an alphabet Σ can also be defined inductively:

$$\frac{}{\emptyset \in \mathsf{RE}} \qquad \frac{}{\epsilon \in \mathsf{RE}} \qquad \frac{a \in \Sigma}{\mathsf{v}\,(a) \in \mathsf{RE}} \qquad \frac{r, s \in \mathsf{RE}}{r + s \in \mathsf{RE}} \qquad \frac{r, s \in \mathsf{RE}}{r \cdot s \in \mathsf{RE}} \qquad \frac{r \in \mathsf{RE}}{r^* \in \mathsf{RE}}$$

Discussion note: Those of you who are familiar with Haskell might want to compare the definitions presented here with the data types defining the corresponding types in Haskell. You could see that except for syntactical and presentation issues (rules vs data types), both definitions are very similar.

An important difference however, is that all elements in an inductively defined set are finite, while elements in a Haskell data type could be infinite!

2.3 Recursively Defined Functions over Inductively Defined Sets

(In what follows, we will denote function applications indistinctly by $f(x)$ or by $f\ x$, depending on which notation is more convenient in each case.)

A recursive function is a function that calls itself as part of its definition.

Consider for example the definition of the factorial function given in section 1.1.2:

$$0! = 1 \qquad (n+1)! = (n+1) \times n!$$

Here the first equation directly gives the value of the function for the element 0. The second equation however, tells that in order to compute the value of the function for the Natural number $n+1$, we need to compute the value of the function for the number n and then multiply that value with $n+1$. So in order to compute the value of $(n+1)!$ we need to compute the value of $n!$.

Let us look at the definition of the factorial function in more detail. We can see that there is one equation for each of the ways of constructing a Natural number. Moreover, the equation that defined the function on 0 directly gives the result of the function without the need of a recursive cals. On the other hand, the equation that defines the function on $n+1$ contains a recursive call on a structurally smaller element. This is very important since it *guarantees* that the function terminates for any input! Indeed, if we want to compute the factorial of a number of the form $n+1$, then the value of $(n+1)!$ depends on the value of $n!$. Now, to compute the value of $n!$ we need to look at whether n is 0, or whether $n = m+1$ for some Natural number m. Now, since all Natural number are constructed by a finite number of application of suc to the element 0, then when looking at the argument of the recursive call (which is necessarily structurally smaller than the original argument to the function!) we will eventually get to 0, for which we directly know the result of the function.

Let us try to compute 4!. By definition of the factorial function we have that $4! = 4 \times 3!$. In order to compute 3! we look again at the definition of the function and see that $3! = 3 \times 2!$. We also have that $2! = 2 \times 1!$ and that $1! = 1 \times 0!$. Now we know that $0! = 1$ so if we put all the pieces together we get that

$$4! = 4 \times 3! = 4 \times 3 \times 2! = 4 \times 3 \times 2 \times 1! = 4 \times 3 \times 2 \times 1 \times 0! = 4 \times 3 \times 2 \times 1 \times 1 = 24$$

The general structure of the definition of a recursive function $f : S \to A$ over the inductively defined set S, where the recursive calls are on *structurally smaller* elements is as follows:

$$\begin{array}{lcl}
f\,(\mathsf{e_1}[x_1, \ldots, x_{j_1}]) & = & g_1[x_1, \ldots, x_{j_1}] \\
\vdots & & \vdots \\
f\,(\mathsf{e_m}[x_1, \ldots, x_{j_m}]) & = & g_m[x_1, \ldots, x_{j_m}] \\
f\,(\mathsf{c_1}[x_1, \ldots, x_{j'_1}, s_1, \ldots, s_{n_1}]) & = & h_1[x_1, \ldots, x_{j'_1}, f\,(s_1), \ldots, f\,(s_{n_1})] \\
\vdots & & \vdots \\
f\,(\mathsf{c_k}[x_1, \ldots, x_{j_n}, s_1, \ldots, s_{n_k}]) & = & h_k[x_1, \ldots, x_{j_n}, f\,(s_1), \ldots, f\,(s_{n_k})]
\end{array}$$

where each x_i is of the correct type according to the definition of the set and each $s_j \in S$.

There might be cases where we need to look "deeper" into the elements of the set in order to define a function. Consider for example the Fibonacci function over the Natural numbers. For inputs $n > 1$, the function has a recursive call on $n-1$ and another on $n-2$. How can we write this using the structure of the elements of $\mathbb{N}$? A first step is to write

$$\begin{array}{lcl}
\mathsf{fibonacci}\ 0 & = & 0 \\
\mathsf{fibonacci}\ (n+1) & = & ??
\end{array}$$

Now, the n in the second equation is also a Natural number, hence this number is either 0 or the application of suc to a number. So we can use this information and transform the second equation into two new equations that easily allow us to define the function:

$$\begin{array}{lcl}
\mathsf{fibonacci}\ 0 & = & 0 \\
\mathsf{fibonacci}\ 1 & = & 1 \\
\mathsf{fibonacci}\ ((n+1)+1) & = & \mathsf{fibonacci}\ (n+1) + \mathsf{fibonacci}\ n
\end{array}$$

Observe that even here, both recursive calls are on structurally smaller elements: the first recursive call is on an element one step smaller in the structure of $(n+1)+1$, while the second call is on an element two steps smaller in the structure.

We could use this same schema to define functions over non-recursive inductive sets. These functions however will not be recursive in the strict sense of the concept.

$$\begin{array}{lcl}
\mathsf{if\ true\ then}\ x\ \mathsf{else}\ y & = & x \\
\mathsf{if\ false\ then}\ x\ \mathsf{else}\ y & = & y
\end{array}$$

2.4 Example of Recursively Defined Functions

Let us define a function which adds all the numbers in a list of Natural numbers:

$$\begin{array}{lcl}
\mathsf{sum\ nil} & = & 0 \\
\mathsf{sum}\ (x : xs) & = & x + (\mathsf{sum}\ xs)
\end{array}$$

Observe that here qqqqqis important that we work with Natural numbers since it might not be possible to add elements in the arbitrary type A.

Let us now define the height of a tree (the longest path from the root of the tree to its leaves):

$$\begin{array}{lcl} \mathsf{height}\ () & = & 0 \\ \mathsf{height}\ (\mathsf{node}\,(t_1, x, t_2)) & = & 1 + \mathsf{max}\,(\mathsf{height}\ t_1, \mathsf{height}\ t_2) \end{array}$$

To count the nodes in a tree we define the following function:

$$\begin{array}{lcl} \mathsf{nrnds}\ () & = & 0 \\ \mathsf{nrnds}\ (\mathsf{node}\,(t_1, x, t_2)) & = & 1 + \mathsf{nrnds}\ t_1 + \mathsf{nrnds}\ t_2 \end{array}$$

Observe that the last two functions could be defined for any tree, independently of the type of the information storaged in the nodes of the tree.

One could also define functions taking more than one argument, but where the recursion is made on only one of those arguments. The definition of the addition and multiplication of Natural numbers are well-known examples of such functions:

$$\begin{array}{lcl@{\qquad}lcl} 0 + m & = & m & 0 \times m & = & 0 \\ (n+1) + m & = & (n+m) + 1 & (n+1) \times m & = & (n \times m) + n \end{array}$$

Recall that $n + 1$ is just syntactic sugar for $\mathsf{suc}\ n$ and it must not be confused with the addition function we just defined, even though the value of the number $\mathsf{suc}\ n$ is the same as the result of adding 1 to n!

Let us now define the append and filter functions over lists with elements in the set A, where $p : A \to \mathsf{Bool}$:

$$\begin{array}{lcl@{\qquad}lcl} \mathsf{nil} +\!\!+\, ys & = & ys & \mathsf{filter}\ p\ \mathsf{nil} & = & \mathsf{nil} \\ (x : xs) +\!\!+\, ys & = & x : (xs +\!\!+\, ys) & \mathsf{filter}\ p\ (x : xs) & = & \mathsf{if}\ p(x)\ \mathsf{then}\ x : (\mathsf{filter}\ p\ xs) \\ & & & & & \qquad\quad \mathsf{else}\ \mathsf{filter}\ p\ xs \end{array}$$

The function that reverses a list can now be defined as follows:

$$\begin{array}{lcl} \mathsf{rec}\ \mathsf{nil} & = & \mathsf{nil} \\ \mathsf{rev}\ (x : xs) & = & (\mathsf{rev}\ xs) +\!\!+\, [x] \end{array}$$

where $[x]$ denotes $(x : \mathsf{nil})$.

We could count the number of single (just a number or a variable) or compound expressions (consisting of the addition or multiplication of expressions) as follows:

$$\begin{array}{lcl@{\qquad}lcl} \mathsf{nrsg}\ (\mathsf{num}\,(n)) & = & 1 & \mathsf{nrcp}\ (\mathsf{num}\,(n)) & = & 0 \\ \mathsf{nrsg}\ (\mathsf{var}\,(x)) & = & 1 & \mathsf{nrcp}\ (\mathsf{var}\,(x)) & = & 0 \\ \mathsf{nrsg}\ (e_1 \oplus e_2) & = & \mathsf{nrsg}\ e_1 + \mathsf{nrsg}\ e_2 & \mathsf{nrcp}\ (e_1 \oplus e_2) & = & 1 + \mathsf{nrcp}\ e_1 + \mathsf{nrcp}\ e_2 \\ \mathsf{nrsg}\ (e_1 \otimes e_2) & = & \mathsf{nrsg}\ e_1 + \mathsf{nrsg}\ e_2 & \mathsf{nrcp}\ (e_1 \otimes e_2) & = & 1 + \mathsf{nrcp}\ e_1 + \mathsf{nrcp}\ e_2 \end{array}$$

If we have an environment $\rho : \mathsf{String} \to \mathbb{N}$ that gives values to variables, we can define a function that evaluates an expression:

$$\begin{array}{lcl} \mathsf{eval}\ (\mathsf{num}\,(n))\ \rho & = & n \\ \mathsf{eval}\ (\mathsf{var}\,(x))\ \rho & = & \rho(x) \\ \mathsf{eval}\ (e_1 \oplus e_2)\ \rho & = & \mathsf{eval}\ e_1\ \rho + \mathsf{eval}\ e_2\ \rho \\ \mathsf{eval}\ (e_1 \otimes e_2)\ \rho & = & \mathsf{eval}\ e_1\ \rho \times \mathsf{eval}\ e_2\ \rho \end{array}$$

Discussion note: Those of you who are familiar with Haskell might want to compare these definitions with the equivalent ones performed by pattern-matching on the elements of corresponding data types.

Discussion note: It is also possible to do recursion on both arguments simultaneously. Consider for example the merge function over (ordered) lists of Natural numbers (here we need to be able to use the relation $<$ on $\mathbb{N}$):

$$\begin{array}{lcl}
\mathsf{merge}\ xs\ \mathsf{nil} & = & xs \\
\mathsf{merge}\ \mathsf{nil}\ ys & = & ys \\
\mathsf{merge}\ (x : xs)\ (y : ys) & = & \mathsf{if}\ (x < y)\ \mathsf{then}\ x : (\mathsf{merge}\ xs\ (y : ys))\ \mathsf{else}\ y : (\mathsf{merge}\ (x : xs)\ ys)
\end{array}$$

We can see that in each recursive call at least one of the lists is structurally smaller than the original lists while the other list remains the same, so the function is bound to terminate for any pair of lists. However, this function is strictly speaking not defined by recursion on lists but on pairs of lists, with the notion of structurally smaller coming from that of pairs of elements. We will however not go into details about this here.

Discussion note: There are of course recursive functions where the recursive call is not performed on structurally smaller elements. A well-known example is the quicksort algorithm on lists of Natural numbers:

$$\begin{array}{lcl}
\mathsf{quicksort}\ \mathsf{nil} & = & \mathsf{nil} \\
\mathsf{quicksort}\ (x : xs) & = & \mathsf{quicksort}\ (\mathsf{filter}\ (\leqslant x)\ xs) +\!\!+ [x] +\!\!+ \mathsf{quicksort}\ (\mathsf{filter}\ (> x)\ xs)
\end{array}$$

The definition of this function does not automatically guarantee its termination since the arguments on which we perform the recursive calls, $\mathsf{filter}\ (\leqslant x)\ xs$ and $\mathsf{filter}\ (> x)\ xs$, are syntactically not structurally smaller than the original argument $(x : xs)$ of the function.

Here however, we can *reason* about the termination of the function since the lengths of the lists on which we perform the recursive calls are strictly smaller than the length of the original list. So there is a *measure* which strictly decrease in each recursive call and allows us to show that the function terminates.

There are also more complex function definitions. Consider for example McCarthy's f91 function:

$$\mathsf{f91}\ n = \begin{cases} n - 10 & \text{if } n > 100 \\ \mathsf{f91}\,(\mathsf{f91}\,(n + 11)) & \text{if } n \leqslant 100 \end{cases}$$

Here the function is not really defined following the structure of the Natural numbers. Moreover, we have *nested* recursive calls, and also a recursive call on an argument which is clearly greater than the original one. It so happens that we can prove that this function terminates with the value 91 for all $n \leqslant 100$, and with the value $n - 10$ otherwise. But this is not a trivial task.

Reasoning about the termination of non-structurally smaller recursive definition is not trivial. For example, the function below is believed to always terminate with the value 1, but no one has really been able to prove it!

$$\begin{array}{lcl}
\mathsf{collatz}\ 0 & = & 1 \\
\mathsf{collatz}\ 1 & = & 1 \\
\mathsf{collatz}\ n & = & \begin{cases} \mathsf{collatz}\,(n/2) & \text{if } n \text{ is even and } n > 1 \\ \mathsf{collatz}\,(3n + 1) & \text{if } n \text{ is odd and } n > 1 \end{cases}
\end{array}$$

As a programmer, it is not always easy to define the solutions to our problems so that the functions only have recursive calls on structurally smaller arguments. We should anyhow make sure our programs terminate, no matter how complex our algorithms look like. Here however we will concentrate just on recursive functions where the recursive calls are on structurally smaller arguments.

3 Structural Induction

Associated to each inductively defined set we have an inductive principle which allows us to prove properties about the elements in the set and about the functions defined over such elements.

Recall the general structure of the rules defining an inductive set S presented in section 2.1. In order to prove $\forall\, s \in S.\, P(s)$, for a given property P over the elements of S, we proceed as follows:

Base cases: We prove $P(\mathsf{e_i}[\ldots])$ for each element $\mathsf{e_i}[\ldots] \in S$ constructed without the need of previous elements in the set;

Inductive steps: For each constructor $\mathsf{c_i}$ defining a new element in S in terms of previously defined elements $s_1, \ldots, s_{k_i} \in S$, we give a method to prove $P(\mathsf{c_i}[\ldots, s_1, \ldots, s_{k_i}])$ from the proofs of $P(s_1), \ldots, P(s_{k_i})$. In other words, for each constructor $\mathsf{c_i}$, given $s_1, \ldots, s_{k_i} \in S$, we need to prove that if our inductive hypotheses (IH) $P(s_1), \ldots, P(s_{k_i})$ hold then $P(\mathsf{c_i}[\ldots, s_1, \ldots, s_{k_i}])$ also holds.

After we have completed these steps, we then have a way to prove the property P for ANY element in S.

The general structure of the inductive principle associated to a set S defined as in section 2.1 is as follows:

$$\dfrac{\begin{array}{rl}
\text{base cases} & \left\{\begin{array}{c}
\forall\, x_1 \in A_1 \cdots x_{j_1} \in A_{j_1}.\, P(\mathsf{e}_1[x_1, \ldots, x_{j_1}]) \\
\vdots \\
\forall\, x_1 \in A_1 \cdots x_{j_m} \in A_{j_m}.\, P(\mathsf{e}_\mathsf{m}[x_1, \ldots, x_{j_m}])
\end{array}\right. \\
\text{inductive steps} & \left\{\begin{array}{c}
\forall\, x_1 \cdots x_{j'_1}\ s_1, \ldots, s_{k_1} \in S.\ \overbrace{P(s_1), \cdots, P(s_{k_1})}^{\text{IH}} \Rightarrow P(\mathsf{c}_1[x_1, \ldots, x_{j'_1}, s_1, \ldots, s_{k_1}]) \\
\vdots \\
\forall\, x_1 \cdots x_{j_n}\ s_1, \ldots, s_{k_n} \in S.\ \overbrace{P(s_1), \cdots, P(s_{k_n})}^{\text{IH}} \Rightarrow P(\mathsf{c}_\mathsf{n}[x_1, \ldots, x_{j_n}, s_{k_1}, \ldots, s_{k_n}])
\end{array}\right.
\end{array}}{\forall\, s \in S.\ P(s)}$$

Compare this general principle with that of mathematical induction. It is easy to see that mathematical induction is indeed just a special case of structural induction over the inductive set of Natural numbers.

Let us now look at the use of structural induction in more detail with the help of some examples.

3.1 $\forall\, xs, ys \in \mathsf{List}\ A.\, \mathsf{rev}\,(xs +\!\!+\ ys) = \mathsf{rev}\ ys +\!\!+\ \mathsf{rev}\ xs$

Here we assume that we already know that append is associative and that $\forall\, xs \in \mathsf{List}\ A.\, xs +\!\!+ \mathsf{nil} = xs$. These properties can easily be proved by structural induction on the appropriate argument.

We define $P(xs)$ to be $\forall\, ys \in \mathsf{List}\ A.\, \mathsf{rev}\,(xs +\!\!+ ys) = \mathsf{rev}\ ys +\!\!+ \mathsf{rev}\ xs$ and we prove $\forall\, xs \in \mathsf{List}\ A.\, P(xs)$ by structural induction (on the list xs).

Our base case is $xs = \mathsf{nil}$. Then, $P(\mathsf{nil})$ amounts to proving that $\forall\, ys \in \mathsf{List}\ A.\, \mathsf{rev}\,(\mathsf{nil} +\!\!+ ys) = \mathsf{rev}\ ys +\!\!+ \mathsf{rev}\ \mathsf{nil}$. By definition of reverse and append we then need to show that $\forall\, ys \in \mathsf{List}\ A.\, \mathsf{rev}\, ys = \mathsf{rev}\ ys +\!\!+ \mathsf{nil}$, which we know is true by one of our assumptions.

Given $xs \in \mathsf{List}\ A$, our IH $P(xs)$ is that $\forall\, ys \in \mathsf{List}\ A.\, \mathsf{rev}\,(xs +\!\!+ ys) = \mathsf{rev}\ ys +\!\!+ \mathsf{rev}\ xs$.

We now need to prove $P(x : xs)$ for a given $x \in A$, that is,
$\forall\, ys \in \mathsf{List}\ A.\, \mathsf{rev}\,((x : xs) +\!\!+ ys) = \mathsf{rev}\ ys +\!\!+ \mathsf{rev}\ (x : xs)$. By definitions of reverse and append we need to show that $\forall\, ys \in \mathsf{List}\ A.\, \mathsf{rev}\,(x : (xs +\!\!+ ys)) = \mathsf{rev}\ ys +\!\!+ (\mathsf{rev}\ xs +\!\!+ [x])$. Working further with the definitions of the functions and using the assumption that append is associative, we now need to show that
$\forall\, ys \in \mathsf{List}\ A.\, \mathsf{rev}\,(xs +\!\!+ ys) +\!\!+ [x] = (\mathsf{rev}\ ys +\!\!+ \mathsf{rev}\ xs) +\!\!+ [x]$, which is true by IH.

This shows that $P(x : xs)$ is true, which concludes our proof by structural induction.

3.2 $\forall\, xs \in \mathsf{List}\ A.\, \mathsf{rev}\,(\mathsf{rev}\,(xs)) = xs$

We define $P(xs)$ to be $\mathsf{rev}\,(\mathsf{rev}\,(xs)) = xs$ and we prove $\forall\, xs \in \mathsf{List}\ A.\, P(xs)$ by structural induction (on the list xs).

Our base case is $xs = \mathsf{nil}$. Then, $P(\mathsf{nil})$ amounts to proving that $\mathsf{rev}\,(\mathsf{rev}\ \mathsf{nil}) = \mathsf{nil}$, which is indeed true by definition of reverse.

Given $xs \in \mathsf{List}\ A$, our IH $P(xs)$ is that $\mathsf{rev}\,(\mathsf{rev}\,(xs)) = xs$.

We now need to prove $P(x : xs)$ for a given $x \in A$, that is, $\mathsf{rev}\,(\mathsf{rev}\,(x : xs)) = x : xs$. By definition of reverse this amounts to proving $\mathsf{rev}\,(\mathsf{rev}\ xs +\!\!+ [x]) = x : xs$. Using the property in section 3.1, we then need to show that $\mathsf{rev}\,[x] +\!\!+ \mathsf{rev}\,(\mathsf{rev}\,(xs)) = x : xs$.

Observe that by definitions of the reverse and append functions we have that $\forall\, x \in A.\, \mathsf{rev}\,[x] = [x]$ and $\forall\, x \in A.\, \forall\, xs \in \mathsf{List}\ A.\, [x] +\!\!+ xs = x : xs$. Using these observations on the appropriate arguments, we then need to show that $x : \mathsf{rev}\,(\mathsf{rev}\,(xs)) = x : xs$, which is indeed try by IH.

This shows that $P(x : xs)$ is true, which concludes our proof by structural induction.

3.3 $\forall\, t \in \mathsf{Tree}\ A.\, \mathsf{nrnds}\ t \leqslant 2^{(\mathsf{height}\ t)} - 1$

We define $P(t)$ to be $\mathsf{nrnds}\ t \leqslant 2^{(\mathsf{height}\ t)} - 1$ and we prove $\forall\, t \in \mathsf{Tree}\ A.\, P(t)$ by structural induction (on the tree t).

Our base case is $t = ()$. Then $P(())$ amounts to proving that $\mathsf{nrnds}\ () \leqslant 2^{(\mathsf{height}\ ())} - 1$. By definitions of the functions we need to show that $0 \leqslant 2^0 - 1$, that is $0 \leqslant 1 - 1 = 0$, which is indeed true.

Given $t_1, t_2 \in \mathsf{Tree}\ A$, our IH $P(t_1)$ and $P(t_2)$ say that $\mathsf{nrnds}\ t_1 \leqslant 2^{(\mathsf{height}\ t_1)} - 1$ and $\mathsf{nrnds}\ t_2 \leqslant 2^{(\mathsf{height}\ t_2)} - 1$, respectively.

We now need to prove that $P(\mathsf{node}\,(t_1, x, t_2))$ for a given $x \in A$, that is, $\mathsf{nrnds}\ (\mathsf{node}\,(t_1, x, t_2)) \leqslant 2^{(\mathsf{height}\ (\mathsf{node}\,(t_1,x,t_2)))} - 1$.

By definitions of the functions, we need to show that $1 + \mathsf{nrnds}\ t_1 + \mathsf{nrnds}\ t_2 \leqslant 2^{1+\mathsf{max}\,(\mathsf{height}\ t_1, \mathsf{height}\ t_2)} - 1 = 2 \times 2^{\mathsf{max}\,(\mathsf{height}\ t_1, \mathsf{height}\ t_2)} - 1$.

By IH and mathematical properties we know that $\mathsf{nrnds}\ t_1 + \mathsf{nrnds}\ t_2 \leqslant 2^{(\mathsf{height}\ t_1)} - 1 + 2^{(\mathsf{height}\ t_2)} - 1 \leqslant 2 \times 2^{\mathsf{max}\,(\mathsf{height}\ t_1, \mathsf{height}\ t_2)} - 2$.

By adding 1 to each side of this inequality we show that $P(\mathsf{node}\,(t_1, x, t_2))$ is true, which concludes our proof by structural induction.

3.4 $\forall\, e \in \mathsf{Exp}.\, \mathsf{nrsg}\ e = \mathsf{nrcp}\ e + 1$

We define $P(e)$ to be $\mathsf{nrsg}\ e = \mathsf{nrcp}\ e + 1$ and we prove $\forall\, e \in \mathsf{Exp}.\, P(e)$ by structural induction (on the expression e).

Our base cases are $e = \mathsf{num}\,(n)$ for a certain $n \in \mathbb{N}$, and $e = \mathsf{var}\,(x)$ for a certain $x \in \mathsf{String}$. To prove $P(\mathsf{num}\,(n))$ we need to show that $\mathsf{nrsg}\,(\mathsf{num}\,(n)) = \mathsf{nrcp}\,(\mathsf{num}\,(n)) + 1$. By definitions of the functions we then need to show that $1 = 0 + 1$, which is indeed true. Similarly, to prove $P(\mathsf{var}\,(x))$ we need to show that $\mathsf{nrsg}\,(\mathsf{var}\,(x)) = \mathsf{nrcp}\,(\mathsf{var}\,(x)) + 1$. By definitions of the functions we then need to show that $1 = 0 + 1$, which is indeed true.

Given $e_1, e_2 \in \mathsf{Exp}$, our IH $P(e_1)$ and $P(e_2)$ say that $\mathsf{nrsg}\ e_1 = \mathsf{nrcp}\ e_1 + 1$ and $\mathsf{nrsg}\ e_2 = \mathsf{nrcp}\ e_2 + 1$.

We now need to prove $P(e_1 \oplus e_2)$ and $P(e_1 \otimes e_2)$.

To prove $P(e_1 \oplus e_2)$ we need to show $\mathsf{nrsg}\,(e_1 \oplus e_2) = \mathsf{nrcp}\,(e_1 \oplus e_2) + 1$. By definitions of the functions we then need to show that $\mathsf{nrsg}\ e_1 + \mathsf{nrsg}\ e_2 = (1 + \mathsf{nrcp}\ e_1 + \mathsf{nrcp}\ e_2) + 1$. By IH and mathematical properties, we know that $\mathsf{nrsg}\ e_1 + \mathsf{nrsg}\ e_2 = (\mathsf{nrcp}\ e_1 + 1) + (\mathsf{nrcp}\ e_2 + 1)$, which shows that $P(e_1 \oplus e_2)$ is true.

Since the definitions of the functions nrsg and nrcp are identical for both $(e_1 \oplus e_2)$ and $(e_1 \otimes e_2)$, the proof of $P(e_1 \otimes e_2)$ is basically the same as the proof of $P(e_1 \oplus e_2)$.

So this concludes the proofs of our inductive steps and hence, our proof by structural induction.

4 Mutual Induction

Sometimes we cannot prove a particular property P by induction but we instead need to prove two or more properties $P_1, P_2, \ldots, P_n$ *simultaneously* by induction. So our property P is the conjunction of all $P_1, P_2, \ldots, P_n$.

It is generally the case that these properties are *mutually dependent*, that is, in order to prove P_i we need to use P_j for one or more j's. Observe however that mutual induction is in itself not a particular inductive principle; the actual inductive principle to be used will depend on the problem at hand.

Mutual induction is very useful when working with automata or grammars. In general, proving that a certain automaton satisfies a property P amounts to proving that each particular state q_i in the automaton satisfies a property P_i. Similarly, to prove that the language generated by a grammar satisfies a property P, we usually need to prove that the language generated by each particular variable V_j in the grammar satisfies a property P_j.

4.1 Mutually Defined Functions

Consider the following functions over the Natural numbers:

$$\begin{array}{lll} f(0) = 1 & g(0) = 0 & h(0) = 0 \\ f(n+1) = f(n) + 1 & g(n+1) = h(n) + n + 2 & h(n+1) = f(n) \end{array}$$

These functions do not really follow the structure in section 2.3: $g(n+1)$ does not call $g(n)$ but instead calls $h(n)$, similarly $h(n+1)$ calls $f(n)$. So to define g we make use of h and to define h we make use of f; hence there is a certain mutual dependency in the definitions of these functions. Observe however, that in each call the argument indeed decrease, which guarantees the termination of these functions for any input.

We want to prove that $\forall\, n \in \mathbb{N}.\, g(n) = 2 \times (f(n) - 1)$.

If we define $P(n)$ to be exactly $g(n) = 2 \times (f(n) - 1)$ and we attempt to prove $\forall\, n \in \mathbb{N}.\, P(n)$ by mathematical induction (on the number n) we will soon get stuck. Let us try and see.

Our base case would be $n = 0$. Here $P(0)$ amounts to proving that $g(0) = 2\times(f(0)-1)$, that is, $0 = 2 \times (1 - 1) = 0$, which is true.

Now, for a given $n \in \mathbb{N}$ we assume our IH $P(n)$ and try to prove $P(n+1)$.

Our IH says that $g(n) = 2 \times (f(n) - 1)$, and hence that $g(n) + 2 = 2 \times f(n)$.

We now need to prove that $g(n+1) = 2 \times (f(n+1) - 1)$. By definitions of g and f, we see that we need to prove that $h(n) + n + 2 = 2 \times (f(n) + 1 - 1) = 2 \times f(n)$. By our IH we see that to prove $P(n+1)$ we need to show that $h(n) + n + 2 = g(n) + 2$ and hence that $h(n) + n = g(n)$. But here we get stuck since we know nothing about the relation between h and f, or h and g.

So instead we actually prove another property $P'(n)$ consisting of three statements $P_g(n)$, $P_f(n)$ and $P_h(n)$. $P_g(n)$ is defined as $g(n) = 2 \times h(n)$, $P_h(n)$ is defined as $h(n) = n$ and $P_f(n)$ is defined as $f(n) = h(n) + 1$. We now prove $\forall\, n \in \mathbb{N}.\, P'(n)$ by mathematical induction (on the number n).

Our base case is $n = 0$ and we need to prove $P_g(0)$, $P_f(0)$ and $P_h(0)$. Here we need to prove that $g(0) = 2 \times h(0)$, $h(0) = 0$ and $f(0) = h(0) + 1$ which amounts to proving that $0 = 2 \times 0, 0 = 0$ and $1 = 0 + 1$, which are all true.

Given $n \in \mathbb{N}$, our IH is $P'(n)$, or in other words $P_g(n)$, $P_f(n)$ and $P_h(n)$. Here we have that $g(n) = 2 \times h(n), h(n) = n$ and $f(n) = h(n) + 1$, which also gives us that $g(n) = 2 \times n$ and $f(n) = n + 1$.

We now need to prove $P'(n+1)$, that is, $P_g(n+1)$, $P_f(n+1)$ and $P_h(n+1)$. Hence we need to prove that $g(n+1) = 2 \times h(n+1), h(n+1) = n+1$ and $f(n+1) = h(n+1)+1$. By definitions of the functions, we need to show that $h(n)+n+2 = 2 \times f(n), f(n) = n+1$ and $f(n)+1 = f(n)+1$. By the IH $P_h(n)$ and $P_f(n)$, $P_g(n+1)$ becomes $2 \times n+2 = 2 \times (n+1)$ which is indeed true. $P_h(n+1)$ is simply true by the IH $P_h(n)$ and $P_f(n)$, and $P_f(n+1)$ is trivially true.

This shows that $P'(n+1)$ is true, which concludes our proof by mutual induction.

Finally, we deduce $\forall\, n \in \mathbb{N}.\, g(n) = 2 \times (f(n) - 1)$ from $\forall\, n \in \mathbb{N}.\, P'(n)$: for a given n, we have that $g(n) = 2 \times h(n) = 2 \times n = 2 \times (f(n) - 1)$.

4.2 Proving Properties of Grammars

Consider the following grammar with start symbol S:

$$S \to 0A \mid 1B \qquad A \to 1 \mid 1S \qquad B \to 0 \mid 0S$$

We want to prove that $\mathcal{L}(S) = (01 + 10)^+$, but it will be difficult to just prove this statement since the $\mathcal{L}(S)$ depends on $\mathcal{L}(A)$ and $\mathcal{L}(B)$, which in turn both depend on $\mathcal{L}(S)$. So we will also need to prove statements about $\mathcal{L}(A)$ and $\mathcal{L}(B)$.

So we prove that $\mathcal{L}(S) = (01 + 10)^+$, $\mathcal{L}(A) = 1(01 + 10)^*$, and $\mathcal{L}(B) = 0(01 + 10)^*$.

We define $P(n)$ as the conjunction of three statements $P_S(n), P_A(n)$ and $P_B(n)$, where each $P_V(n)$ is defined as follows: if $V \Rightarrow^n w$ then $w \in \mathcal{L}(V)$, for $V \in \{S, A, B\}$ and $\mathcal{L}(V)$ as defined above. We prove these statements $\forall\, n \geqslant 1.\, P(n)$ by mathematical induction (on the length of the derivation n, $V \Rightarrow^n w$).

Our base case is $n = 1$ (recall there is no derivation to a word taking less than one step) and we need to prove $P_S(1), P_A(1)$ and $P_B(1)$.

$P_S(1)$ says that if $S \Rightarrow w$ then $w \in \mathcal{L}(S)$. Observe that it is not possible to derive a word from S in only one step, so $P_S(1)$ is true simply because the condition in the conditional-statement is false.

$P_A(1)$ says that if $A \Rightarrow w$ then $w \in \mathcal{L}(A)$. Here w must be 1 (by looking at the productions in the grammar), which clearly is in $\mathcal{L}(A)$.

Similarly, $P_B(1)$ says that if $B \Rightarrow w$ then $w \in \mathcal{L}(B)$. Here w must be 0 (by looking at the productions in the grammar), which clearly is in $\mathcal{L}(B)$.

Given $n \in \mathbb{N}$, our IH is $P(n)$, or in other words $P_S(n), P_A(n)$ and $P_B(n)$. So we have that if $V \Rightarrow^n w$ then $w \in \mathcal{L}(V)$, for $V \in \{S, A, B\}$.

We now need to prove $P(n+1)$, that is, $P_S(n+1), P_A(n+1)$ and $P_B(n+1)$.

Consider $S \Rightarrow^{n+1} w$. Then we have two cases: either $S \Rightarrow 0A \Rightarrow^n w = 0w'$ with $A \Rightarrow^n w'$, or $S \Rightarrow 1B \Rightarrow^n w = 1w'$ with $B \Rightarrow^n w'$.

In the first case, by the IH $P_A(n)$ we know that $w' \in \mathcal{L}(A) = 1(01 + 10)^*$. Hence $w = 0w' \in 01(01 + 10)^* \subseteq (01 + 10)^+ = \mathcal{L}(S)$.

In the second case, by the IH $P_B(n)$ we know that $w' \in \mathcal{L}(B) = 0(01 + 10)^*$. Hence $w = 1w' \in 10(01 + 10)^* \subseteq (01 + 10)^+ = \mathcal{L}(S)$.

This proves $P_S(n+1)$.

Let us now consider $A \Rightarrow^{n+1} w$. Then we must have that $A \Rightarrow 1S \Rightarrow^n w = 1w'$ with $S \Rightarrow^n w'$. (Observe that our base case is $n = 1$ so $n + 1 > 1$ and hence the production $A \to 1$ could not have been used as first step in the derivation $A \Rightarrow^{n+1} w$, which is longer than one step.) By the IH $P_S(n)$ we have that $w' \in \mathcal{L}(S) = (01 + 10)^+$. Hence $w = 1w' \in 1(01 + 10)^+ \subseteq 1(01 + 10)^* = \mathcal{L}(A)$, which proves $P_A(n+1)$.

Finally, let us consider $B \Rightarrow^{n+1} w$. Then we must have that $B \Rightarrow 0S \Rightarrow^n w = 0w'$ with $S \Rightarrow^n w'$. (Again, since our base case is $n = 1$ then $n + 1 > 1$ so the production $B \to 0$ could not have been used as first step in the derivation $B \Rightarrow^{n+1} w$.) By the IH $P_S(n)$ we have that $w' \in \mathcal{L}(S) = (01 + 10)^+$. Hence $w = 0w' \in 0(01 + 10)^+ \subseteq 0(01 + 10)^* = \mathcal{L}(B)$, which proves $P_B(n+1)$.

This concludes our proof by mutual induction.

Discussion note: Observe that we use mathematical induction here while we used course-of-values induction in section 1.2.3 when we also wanted to prove a property about the language generated by a grammar. The reason we can use mathematical induction here (and not there) is that there is at most one variable on the right-hand-side of each of the productions in the grammar. Hence we know that if $V \Rightarrow^{n+1} w$, and if the first step uses a production of the form $V \rightarrow aW$ then we must have that $V \Rightarrow aW \Rightarrow^{n} w$ with $w = aw'$ and $W \Rightarrow^{n} w'$.

5 Proofs by Induction: Overview of the Steps to Follow

To end, we summarise the steps that need to be followed when making a proof by induction. Observe that if any of these steps is missing then very likely the proof is not correct!

1. *State the property P to be proved by induction.*

   Recall that sometimes the property that we want to prove is not exactly the one we actually prove by induction!

2. *Determine and state the method to use in the proof!*

   For example: Mathematical induction on the length of the list, course-of-values induction on the length of a derivation, structural induction over a certain element in an inductive set, ...

3. *Identify and state base case(s).*

   Recall that we could have more than one base case!

4. *Prove base case(s).*

5. *Identify and state IH!*

   Our IH will depend on the method to be used (see step 2). It is very useful to actually state the IH in detail instead of simply saying "we assume our IH P(n)". By spelling out the IH in detail it becomes more clear what information we actually have at hand to work with.

6. *Prove inductive step(s).*

7. *Deduce the desired property to be proved from P.*

   Recall step 1 and the fact that the actual property to be proved might not exactly be P.