1 Structural Induction and Structural Recursion on Propositions

Remark 2.1.1 We defined the set of propositions **PROP** inductively. What distinguishes this set is that it is the smallest set (1) containing all propositional atoms, (2) containing $(\neg\,\alpha)$ when α is in the set, and (3) containing $(\alpha \diamond \beta)$ for each $\diamond \in \{\wedge\,,\,\vee\,,\,\rightarrow\,,\,\leftrightarrow\}$ when α and β are in the set.

What is most important to us is that the set of propositions supports a method of proof (the Principle of Structural Induction) and a method of defining operations (the Principle of Structural Recursion).

The method of proof is

Theorem 2.1.2 (Principle of Structural Induction) Suppose that X is a set of propositions which satisfies the following clauses:

Basis step. All propositional atoms are in X.

Inductive step.
(i) If α is in X, then so is $(\neg\,\alpha)$.
(ii) If α and β are in X, then so $(\alpha \diamond \beta)$ where $\diamond \in \{\wedge\,,\,\vee\,,\,\rightarrow\,,\,\leftrightarrow\}$.

Then $X = \mathbf{PROP}$.

Proof. The Basis and Inductive clauses express that X meets conditions (1) through (3) in Remark 2.1.1. But since X is a subset of all propositions **PROP**, and **PROP** is the smallest such set, $X = \mathbf{PROP}$.

An alternative way of proving this is by *mathematical induction* by the *depth* n of a proposition (as measured in the Bottom-Up II approach). The depth of a proposition means the depth of its formation tree (the longest path from root to leaf). There are two key points to note: (a) all propositions have *finite depth* and (b) if a proposition is constructed by rules (2) or (3), the propositions from which it was constructed (α and β above) have *smaller* depth.

Basis step. The only depth 0 propositions are propositional atoms, and they are in X by the Basis clause.

Inductive step. Suppose all propositions whose depth is less than $n+1$ are in X (inductive hypothesis). For a proposition of the form $(\neg\,\beta)$ whose depth is $n+1$, the depth of β is less than $n+1$. So, by the inductive hypothesis β is in X, and by the Inductive clause (i), $(\neg\,\beta)$ is also in X. For a proposition of the form $(\beta \diamond \gamma)$ of depth $n+1$ for some $\diamond \in \{\wedge\,,\,\vee\,,\,\rightarrow\,,\,\leftrightarrow\}$, then both β and γ have depth less than $n+1$. So, by the inductive hypothesis both β and γ are in X, and by by the Inductive clause (ii), $(\beta \diamond \gamma)$ is in X. It now follows that all propositions whose depth is no more than $n+1$ are in X.

It follows by mathematical induction that every proposition α whose depth is finite is in X, which by (a) above is all propositions. So, $X = \mathbf{PROP}$. □

Lets show that $(((P \wedge Q)$ is not a proposition. Intuitively, this is because there are more left parentheses then right, and all propositions have the same number of left as right parenthesis. The proof of this fact is by structural induction.

Lemma 2.1.3 Every formula has the same number of left as right parentheses.

Proof. Let S be the set of all propositional expressions having an equal number of left and right parentheses. The proof that $S = \mathbf{PROP}$ is by structural induction.

Basis Step. The propositional atoms have no parentheses (left parentheses = right parentheses = 0), so every propositional atom is in S.

Inductive Step. Assume that α and β are propositions in S, so each has the same number of left as right parentheses (inductive hypothesis). Then (i) $(\neg\,\alpha)$ has the same number of left as right parentheses and (ii) $(\alpha \diamond \beta)$ also has the same number of left as right parentheses, since each adds one additional left and right parenthesis. Each are in S.

It follows that S is the set of all propositional expressions. □

2 Principle of Structural Recursion

There is also a method for defining operations (functions) on propositions:

Theorem 2.2.1 (Principle of Structural Recursion) There is one and only one function F defined on the set **PROP** of propositions such that:

Basis clause. The value of F is specified explicitly on propositional atoms Z.

Recursive clause.
(i) The value of F on $(\neg\,\alpha)$ is specified in terms of the value of F on α.
(ii) The value of of F on $(\alpha\diamond\beta)$ is specified in terms of the value of F on α and β for each $\diamond \in \{\wedge\,, \vee\,, \rightarrow\,, \leftrightarrow\}$.

Remark 2.2.2 Notice that the Recursive clause of the definition defines the function F in terms of F's values on "simpler" propositions. The value of F on propositional atoms is stipulated in the Basis clause without requiring knowledge of F on any propositions. Informally, a procedure for executing the function F will have to step backwards from complex propositions to its simpler constituent parts, and will eventually reach propositional atoms. At this point the procedure works its way back upward from simpler propositions (whose value it now knows) to more complex propositions by applying the Recursive clause . This is vividly illustrated in the first example below.

This principle requires that there is only one way to decompose a proposition. This will be explained in more detail in Section 3 below. Before we do this, we will look at some examples of using structural recursion.

A proof is sketched below in Remark 2.3.4. It is useful to see how to define operations by structural recursion.

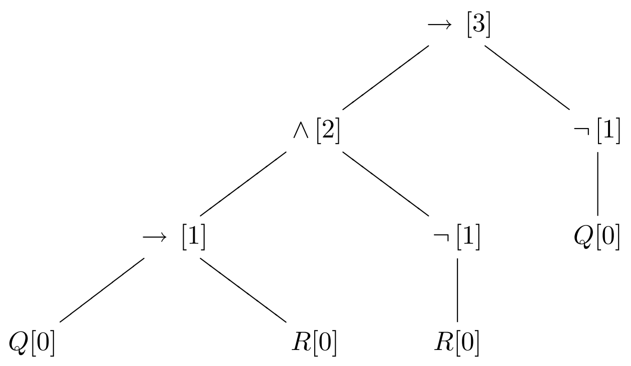
Example 2.2.3 (depth of proposition) The *depth* of a proposition was defined in Lecture I in (Bottom-up II) in Remark 1.3.1. It is simply the depth of the formation tree, which is the longest path from root to leaf. We define this function *depth* : **PROP** $\rightarrow \mathbb{N}$ by structural recursion.

$$\begin{aligned}
\mathit{depth}(Z) &= 0 \qquad \text{for each propositional atom } Z \\
\mathit{depth}(\neg\,\alpha) &= \mathit{depth}(\alpha) + 1 \\
\mathit{depth}(\alpha \diamond \beta) &= \max\{\mathit{depth}(\alpha), \mathit{depth}(\beta)\} + 1 \qquad \text{for each } \diamond \in \{\wedge\,, \vee\,, \rightarrow\,, \leftrightarrow\}.
\end{aligned}$$

It is instructive to see why this definition succeeds:

$$\mathit{depth}\big(((Q \rightarrow R) \wedge (\neg\,R)) \rightarrow (\neg\,Q)\big) = 3$$

which can be verified using the formation tree for this proposition:



I have place the value of *depth* next to each node in the tree. Recursive procedures typically "dive bomb" a proposition *depth first* following the left-most branch until it hits a leaf. The values at the leaves are stipulated by BASIS CLAUSE. The procedure then backtracks one level up, and attacks the right branch, if there is one. Once values for the left and right branch are computed, the value at the parent node can be computed by RECURSIVE CLAUSE.

Example 2.2.4 (support of proposition) The *support* of a proposition are the propositional atoms occurring in it. Intuitively, the support is the set of symbols occurring at leaves in the formation tree. We define a function *support* on the proposition whose values are *sets of propositional atoms*: by recursion:

$$\begin{aligned} \mathit{support}\big(Z\big) &= \{Z\} \qquad \text{for each propositional atom } Z, \\ \mathit{support}\big(\neg\,\alpha)\big) &= \mathit{support}(\alpha), \\ \mathit{support}\big(\alpha \diamond \beta\big) &= \mathit{support}(\alpha) \cup \mathit{support}(\beta) \qquad \text{for each } \diamond \in \{\,\wedge\,,\,\vee\,,\,\rightarrow\,,\,\leftrightarrow\}. \end{aligned}$$

$$\mathit{support}\big(((P_0\,\vee\,P_2)\,\wedge\,(\neg\,P_1))\big) = \{P_0, P_1, P_2\}.$$

Example 2.2.5 (substitution operation) Let γ and δ be propositions of **PROP**. We define a function $\mathit{Subst}^{\delta}_{\gamma}$: **PROP** → **PROP** which substitutes γ for all occurrences of δ in a given proposition. In the formation tree this mean replacing each subtree δ with the tree γ. If δ is a leaf, this means replacing each leaf with the tree γ. The recursive definition is

$$\begin{aligned} \mathit{Subst}^{\delta}_{\gamma}(A) &= \begin{cases} \gamma & \text{if } \delta = A \\ A & \text{otherwise} \end{cases} \qquad \text{for any atom } A, \\ \mathit{Subst}^{\delta}_{\gamma}(\neg\,\alpha) &= \begin{cases} \gamma & \text{if } \delta = (\neg\,\alpha) \\ \neg\,\mathit{Subst}^{\delta}_{\gamma}(\alpha) & \text{otherwise,} \end{cases} \\ \mathit{Subst}^{\delta}_{\gamma}(\alpha \diamond \beta) &= \begin{cases} \gamma & \text{if } \delta = (\alpha \diamond \beta) \\ (\mathit{Subst}^{\delta}_{\gamma}(\alpha) \diamond \mathit{Subst}^{\delta}_{\gamma}(\alpha)) & \text{otherwise} \end{cases} \\ &\qquad \text{for each } \diamond \in \{\,\wedge\,,\,\vee\,,\,\rightarrow\,,\,\leftrightarrow\} \end{aligned}$$

For example,

$$\mathit{Subst}^{(\neg\,Q)}_{(P\,\wedge\,R)}\big((((\neg\,Q)\,\vee\,P)\,\rightarrow\,(\neg\,Q))\big) = (((P\,\wedge\,R)\,\vee\,P)\,\rightarrow\,(P\,\wedge\,R)).$$

It still must be shown that the range of $\mathit{Subst}^{\delta}_{\gamma}$ is really **PROP**, which is proven by structural induction and is left as an exercise.

We will utilize the following commonly used abbreviation

$$\mathit{Subst}^{\delta}_{\gamma}(\alpha) = \alpha^{\delta}_{\gamma}.$$

It is also possible to extend the definition of substitution to the *simultaneous substitution* of several propositions $\delta_1, \ldots, \delta_n$ for $\gamma_1, \ldots, \gamma_n$ into a proposition.

3 Unique Parsing of Propositions

The principle of structural recursion depends on the fact that propositions can be uniquely parsed. Formally, this means

Theorem 2.3.1 (Unique Decomposition) Every proposition α in **PROP** meets *exactly one* of the following conditions.

(a) α is a uniquely determined propositional atom.

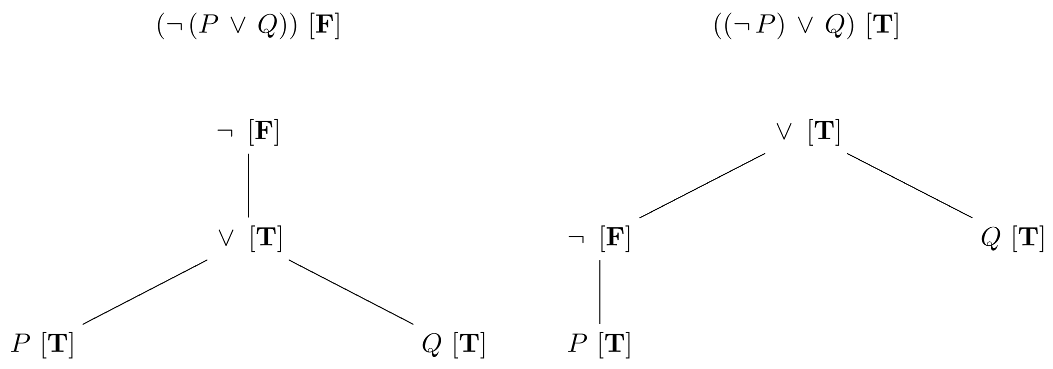
(b) There is a uniquely determined proposition β and α is $(\neg\beta)$.

(c) There are uniquely determined propositions β and γ and a uniquely determined binary connective $\diamond \in \{\wedge, \vee, \rightarrow, \leftrightarrow\}$ such that α is $(\beta \diamond \gamma)$.

Example 2.3.2 (dropping parentheses) If we drop parentheses, then we would lose unique decomposition. For example, the expression $\neg P \vee Q$ could be interpreted as either proposition:

$$(\neg(P \vee Q)) \quad \text{or} \quad ((\neg P) \vee Q).$$

We want to determine the truth value of propositions based on the interpretation of $\neg$ as *not* and $\vee$ as *or*. There are two truth values: **T**for *true* and **F**for *false*. Suppose that P and Q are both true. Since the expression $\neg P \vee Q$ is *ambiguous* between the two propositions, it has two formation trees. Suppose we try to determine the truth value of $\neg P \vee Q$ by recursion using the formation trees:



We must introduce some way of deciding between which of these two formation trees correctly represents the way the proposition $\neg P \vee Q$ is constructed in order to assign a unique truth value using structural recursion. Our precedence ordering, Convention 1.2.5, would take $\neg P \vee Q$ to be the proposition $((\neg P) \vee Q)$ since $\neg$ has a higher precedence than $\vee$.

Remark 2.3.3 Proving that our propositional expressions in Definition 1.2.3 can be uniquely parsed is not entirely trivial, and it is even that much harder to show it holds with all our conventions. I will provide a handout of how it is done with the full use of parentheses from the original definition.

There is an alternative approach treating propositions as trees, given in the next section. Before this we complete the justification of the principle of structural recursion.

Remark 2.3.4 (Justification of the principle of structural recursion) The proof is by structural induction 2.4.4. We will show that the conditions BASIS CLAUSE and RECURSIVE CLAUSE determine a unique value on each proposition. Then the function F simply returns this value. The details of the construction of F require a bit of care, and this is done in Appendix 2, but the idea is easy to understand.

Let X be the subset of propositions of **PROP** which have a uniquely determined value given by the two clauses Basis clause and Recursive clause.

Basis step. All propositional atoms have a uniquely determined value by the Basis clause. We need to know that **PROP** is uniquely decomposable: the only way an propositional atom could get a value is through Basis clause. So, all propositional atoms are in X.

Inductive step. Suppose α and β are in X (inductive hypothesis), so that each has a uniquely determined value. A proposition of the form $(\neg\alpha)$ receives a value through condition (i) of the Recursive clause. Since all propositions are uniquely decomposable, the only way this proposition could receive a value is through condition (i) from the uniquely determined value of α. So, $(\neg\alpha)$ is in X. Similarly, a proposition of the form $(\alpha \diamond \beta)$ receives a value through condition (ii) of the Recursive clause, and since this decomposition of the proposition is unique, the value is uniquely determined by the values for α and β as in condition (ii). So, $(\alpha \diamond \beta)$ is in X.

It follows that $P =$ **PROP**, that is, all propositions receive a uniquely determined value through the clauses Basis clause and Recursive clause.

4 Propositions as trees

The material in this section is not strictly necessary to continue. It makes more precise the notion of a formation tree from earlier.

Remark 2.4.1 There is an alternative approach, which is the one we will officially take. A proposition is binary tree inductively defined in 2.4.3. These trees are just the formation trees from earlier. The advantage of this is that these trees *transparently* satisfy the conditions of unique decomposition and we no longer need parentheses. You can think of Definition 1.2.3 as providing a user friendly way of describing propositions so that we can manipulate them in proofs and definitions (like a high-level program), and the tree definition 2.4.3 is the low-level mathematical implementation.

A propositional expressions is now *unambiguous* if each expression has a unique proposition tree.

Definition 2.4.2 (Tree) Trees are formally defined in Appendix A.3. A *tree* consists a set of *nodes* ordered by the a relation $<$ called the *successor relation*. If $x < y$ then y is a *successor* of x and x is a *predecessor* of y. Furthermore, if $x < y$ but there is no z between x and y, then y is the *immediate successor* of x and x is the *immediate predecessor*. This relation satisfies natural ordering properties:

1. (Transitivity): If $x < y$ and $y < z$, then $x < z$.
2. (Antisymmetry): There are no x and y with $x < y < x$. (Intuitively, there are no cycles – you cannot follow a branch and come back to a node you have already passed.)
3. (Finiteness): Every node has finitely many predecessors and each of these are comparable to each other. This means that every node has a unique predecessor.

There is a unique node called the *root*, denoted by λ, which is the predecessor of every node. The immediate successors of a node are called its children. If every node has finitely many children, then it is said to be a *finitely branching*. A *binary tree* is a tree in which each node has no more than two children. A node with no children is called a *leaf*. Finally, a *path* π in a tree is a set of nodes (possibly infinite) $\{\lambda < x_1 < x_2 < x_3 < \ldots\}$ where each x_{i+1} is an immediate successor of x_i and which cannot be extended to a longer path in the tree. So, if a path π is finite, then it ends in a leaf.

There are two *constructors* we will use to build trees:

(a) $mkleaf(\ell)$: creates a one-node tree whose root is labeled ℓ.

(b) $mknode(\ell, \tau_1 \ldots, \tau_k)$: creates a tree whose root is labeled with ℓ and which has k branches whose subtrees are $\tau_1, \ldots, \tau_k$.

A *labeled tree* is a tree whose nodes are assigned labels from some alphabet of symbols.

Definition 2.4.3 Our proposition trees will be labeled from the alphabet of propositional atoms and logical connectives from Definition 1.2.1 (except for the parentheses).

A *proposition tree* is the smallest class of labeled binary trees such that

1. $mkleaf(\alpha)$ is a proposition tree for each propositional atom α.

2. If α is a proposition tree, then so is $mknode(\neg, \alpha)$.

3. If α and β are proposition trees, then so is $mknode(\diamond, \alpha, \beta)$ for each $\diamond \in \{\wedge, \vee, \rightarrow, \leftrightarrow\}$.

Since propositional trees are inductively defined, there is a version of structural induction (and recursion) appropriate for trees. Structural induction is

Theorem 2.4.4 (Principle of Structural Induction for trees) Suppose that X is a set of proposition trees which satisfies the following clauses:

Basis step. All one-node trees labeled with propositional atoms are in X.

Inductive step.
(i) If α is a tree in X, then so is $mknode(\neg, \alpha)$.
(ii) If α and β are trees in X, then so is $mknode(\diamond, \alpha, \beta)$ where $\diamond \in \{\wedge, \vee, \rightarrow, \leftrightarrow\}$.

Then X is the set of all proposition trees.

Unique decomposability now takes the form:

Theorem 2.4.5 (Unique Decomposition of proposition trees) Every proposition tree α meets *exactly one* of the following conditions.

(a) α is $mkleaf(\beta)$ for some uniquely determined propositional atom β.

(b) There is a uniquely determined proposition tree β and α is $mknode(\neg, \beta)$.

(c) There are uniquely determined proposition trees β and γ and a uniquely determined binary connective $\diamond \in \{\wedge, \vee, \rightarrow, \leftrightarrow\}$ such that α is $mknode(\diamond, \alpha, \beta)$.

Proof. The proof is by structural induction on the tree α.

Basis step. A one-node proposition tree is uniquely determined by its label, a propositional atom. Since these are the only one-node trees (conditions (2) and (3) always generate trees with more than one node), all one-node trees are uniquely determined by (a), so are in X.

Inductive step. Suppose α and β are trees in X (inductive hypothesis). (i) The tree $mknode(\neg, \alpha)$ is uniquely determined by the label on its root and the subtree α. Since α is uniquely determined, $mknode(\neg, \alpha)$ uniquely satisfies condition (b), so is in X. (ii) The tree $mknode(\diamond, \alpha, \beta)$ is uniquely determined by the label on its root and the subtrees α and β. Since these subtrees are uniquely determined, $mknode(\diamond, \alpha, \beta)$ is uniquely determined by (c), so is in X.

Thus, the set X is the set of all proposition trees. □