Chapter 4

Functions Defined Recursively on $\mathbb{PL}$

Reminder. *Recall that when X is a set, by 2^X we denote the* powerset of X, *that is, the set of all subsets of X. For example, if $X = \{1, 2, 3\}$, then $2^X = \{\emptyset, \{1\}, \{2\}, \{3\}, \{1, 2\}, \{1, 3\}, \{2, 3\}, \{1, 2, 3\}\}$. When X is finite, note that $|2^X| = 2^{|X|}$, which partly explains this notation. You might have used the notation $\mathcal{P}(X)$ instead of 2^X in highschool.*

Whenever a set is inductively defined, we can define recursive functions that operate on the elements of the set, function which make use of the *structure* of the elements.

Here is an example of a function computing the set of *subformulae* of a formula:

$subf : \mathbb{PL} \to 2^{\mathbb{PL}}$, defined by:

$$subf\Big(\varphi\Big) = \begin{cases} \{\varphi\}, & \text{if } \varphi \in A; \\ \{\varphi\} \cup subf\Big(\varphi'\Big), & \text{if } \varphi = \neg\varphi' \text{ and } \varphi' \in \mathbb{PL}; \\ \{\varphi\} \cup subf\Big(\varphi_1\Big) \cup subf\Big(\varphi_2\Big), & \\ \qquad \text{if } \varphi = (\varphi_1 \wedge \varphi_2) \text{ and } \varphi_1, \varphi_2 \in \mathbb{PL}; & \\ \{\varphi\} \cup subf\Big(\varphi_1\Big) \cup subf\Big(\varphi_2\Big), & \\ \qquad \text{if } \varphi = (\varphi_1 \vee \varphi_2) \text{ and } \varphi_1, \varphi_2 \in \mathbb{PL}. & \end{cases}$$

Here is how we can compute $subf\Big(\varphi\Big)$ when $\varphi = \neg(\mathtt{p} \vee \mathtt{q})$:

$$\begin{aligned} subf\Big(\neg(\mathtt{p} \vee \mathtt{q})\Big) &= \{\neg(\mathtt{p} \vee \mathtt{q})\} \cup subf\Big((\mathtt{p} \vee \mathtt{q})\Big) \\ &= \{\neg(\mathtt{p} \vee \mathtt{q})\} \cup \{(\mathtt{p} \vee \mathtt{q})\} \cup subf\Big(\mathtt{p}\Big) \cup subf\Big(\mathtt{q}\Big) \\ &= \{\neg(\mathtt{p} \vee \mathtt{q})\} \cup \{(\mathtt{p} \vee \mathtt{q})\} \cup \{\mathtt{p}\} \cup \{\mathtt{q}\} \\ &= \{\neg(\mathtt{p} \vee \mathtt{q}), (\mathtt{p} \vee \mathtt{q}), \mathtt{p}, \mathtt{q}\}. \end{aligned}$$

Notice that we use two different types of brackets in the computation above. On one hand, we have the brackets that surround the argument to the function *subf*, and on the other hand we have the usual brackets that are part of the alphabet L. The former brackets are the usual brakets in mathematics, while the later brackets are simply symbols in our alphabet. In order to differentiate between them, we adopt the convention to use bigger brakets for the function call and the usual brackets in blue for the auxiliary symbols.

Note that both are important. For example, it would be an error to write $subf\Big(\mathtt{p} \vee \mathtt{q}\Big)$ instead of $subf\Big((\mathtt{p} \vee \mathtt{q})\Big)$, as the word $\mathtt{p} \vee \mathtt{q} \notin \mathbb{PL}$ is not a formula.

The function *subf* is the first in a long line of functions defined recursively on the structure of a formula $\varphi \in \mathbb{PL}$. These functions are called *structurally recursive*.

It is very important to understand how such functions operate in order to understand the remainder of this course. Here are several more examples of such functions.

4.1 The Abstract Syntax Tree of a Formula

Here is an example of a function computing the *abstract syntax tree* of a formula:

$ast : \mathbb{PL} \to Trees$, defined by:

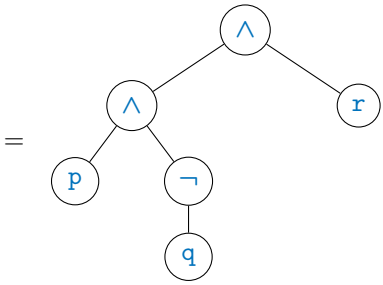
$$ast\Big(\varphi\Big) = \begin{cases} \boxed{F}, & \text{if } \varphi \in A; \\ \boxed{\neg}\ ast\Big(\varphi'\Big), & \text{if } \varphi = \neg\varphi' \text{ and } \varphi' \in \mathbb{PL}; \\ ast\Big(\varphi_1\Big)\ \boxed{\vee}\ ast\Big(\varphi_2\Big), & \text{if } \varphi = (\varphi_1 \wedge \varphi_2) \text{ and } \varphi_1, \varphi_2 \in \mathbb{PL}; \\ ast\Big(\varphi_1\Big)\ \boxed{\wedge}\ ast\Big(\varphi_2\Big), & \text{if } \varphi = (\varphi_1 \vee \varphi_2) \text{ and } \varphi_1, \varphi_2 \in \mathbb{PL}. \end{cases}$$

Here is the computation of the abstract syntax tree of the formula $((\mathtt{p} \wedge \neg\mathtt{q}) \wedge \mathtt{r})$.

$$ast\Big(((\mathtt{p} \wedge \neg\mathtt{q}) \wedge \mathtt{r})\Big) = ast\Big((\mathtt{p} \wedge \neg\mathtt{q})\Big)\ \boxed{\wedge}\ ast\Big(\mathtt{r}\Big)$$

$$= \Big(ast\Big(\mathtt{p}\Big)\ \boxed{\wedge}\ ast\Big(\neg\mathtt{q}\Big)\Big)\ \boxed{\wedge}\ \mathtt{r}$$

$$= \Big(\mathtt{p}\ \boxed{\wedge}\ \big(\boxed{\neg}\ ast\Big(\mathtt{q}\Big)\big)\Big)\ \boxed{\wedge}\ \mathtt{r}$$



Note that *Trees* is the set of labeled, rooted trees, considered here intuitively, without a formal definition.

Note that abstract syntax trees are very important. In fact, conceptually, a propositional formula *is* its abstract syntax tree. However, because writing trees is cumbersome, we have resort to the more conventional left-to-right notation.

4.2 Other Examples of Recursively Defined Functions

Here are three more examples of functions defined by structural recursion on formulae.

The first function, $height : \mathbb{PL} \to \mathbb{N}$, computes the *height* of the ast o formula:

$$height\Big(\varphi\Big) = \begin{cases} 1, & \text{if } \varphi \in A; \\ 1 + height\Big(\varphi'\Big), & \text{if } \varphi = \neg\varphi' \text{ and } \varphi' \in \mathbb{PL}; \\ 1 + max\Big(height\Big(\varphi_1\Big), height\Big(\varphi_2\Big)\Big), & \text{if } \varphi = (\varphi_1 \wedge \varphi_2) \text{ and } \varphi_1, \varphi_2 \in \mathbb{PL}; \\ 1 + max\Big(height\Big(\varphi_1\Big), height\Big(\varphi_2\Big)\Big), & \text{if } \varphi = (\varphi_1 \vee \varphi_2) \text{ and } \varphi_1, \varphi_2 \in \mathbb{PL}. \end{cases}$$

The function $max : \mathbb{N} \times \mathbb{N} \to \mathbb{N}$, which occurs in the definition of *height*, is the well-known function that computes the maximum between two natural numbers.

The next function, $size : \mathbb{PL} \to \mathbb{N}$, computes the *size* of the abstract syntax tree of a formula (i.e., the number of nodes):

$$size\Big(\varphi\Big) = \begin{cases} 1, & \text{if } \varphi \in A; \\ 1 + size\Big(\varphi'\Big), & \text{if } \varphi = \neg\varphi' \text{ and } \varphi' \in \mathbb{PL}; \\ 1 + size\Big(\varphi_1\Big) + size\Big(\varphi_2\Big), & \text{if } \varphi = (\varphi_1 \wedge \varphi_2) \text{ and } \varphi_1, \varphi_2 \in \mathbb{PL}; \\ 1 + size\Big(\varphi_1\Big) + size\Big(\varphi_2\Big), & \text{if } \varphi = (\varphi_1 \vee \varphi_2) \text{ and } \varphi_1, \varphi_2 \in \mathbb{PL}. \end{cases}$$

The final example function, $prop : \mathbb{PL} \to 2^A$, compute the set of propositional variables occurring in a formula:

$$prop = \begin{cases} \{\varphi\}, & \text{if } \varphi \in A; \\ prop, & \text{if } \varphi = \neg\varphi' \text{ and } \varphi' \in \mathbb{PL}; \\ prop \cup prop, & \text{if } \varphi = (\varphi_1 \wedge \varphi_2) \text{ and } \varphi_1, \varphi_2 \in \mathbb{PL}; \\ prop \cup prop, & \text{if } \varphi = (\varphi_1 \vee \varphi_2) \text{ and } \varphi_1, \varphi_2 \in \mathbb{PL}. \end{cases}$$

Make sure that you are proficient at understanding, defining and computing with functions such as those given above.

4.3 Proofs by Structural Induction

We will sometimes do proofs by *structural induction*. You are already familiar with proofs by induction on the naturals.

In order to prove a theorem of the form

$$\text{For all } n \in \mathbb{N}, \text{ it is true that } P(n),$$

the mathematical induction principle postulates that it is sufficient to show that:

1. (Base Case) $P(0)$ holds;
2. (Induction Case) $P(k)$ implies $P(k+1)$, for all $k \in \mathbb{N}$.

Proofs by structural induction generalize the principle above to any inductively defined set such as $\mathbb{PL}$.

For the case of $\mathbb{PL}$, the structural induction principle is that, in order to show a theorem of the form

For any propositional formula $\varphi \in \mathbb{PL}$, it is true that $P\Big(\varphi\Big)$,

it is sufficient to prove that:

1. (Base Case) $P\Big(\varphi'\Big)$ holds for any atomic formula $\varphi' \in A$ (otherwise put, the property P holds of every atomic formula);
2. (Inductive Case i) $P\Big(\neg\varphi'\Big)$ holds whenever $P\Big(\varphi'\Big)$ holds;
3. (Inductive Case ii) $P\Big((\varphi_1 \wedge \varphi_2)\Big)$ holds whenever $P\Big(\varphi_1\Big)$ and $P\Big(\varphi_2\Big)$ hold;
4. (Inductive Case iii) $P\Big((\varphi_1 \vee \varphi_2)\Big)$ holds whenever $P\Big(\varphi_1\Big)$ and $P\Big(\varphi_2\Big)$ hold.

Here is an example of a theorem and its proof by structural induction:

Theorem 18 (Example of Theorem Provable by Structural Induction)**.** *For any propositional formula φ, we have that $height\Big(\varphi\Big) \leq size\Big(\varphi\Big)$.*

Proof: We proceed by structural induction (the property $P(\varphi)$ is simply that $height\Big(\varphi\Big) \leq size\Big(\varphi\Big)$). It is sufficient to show that:

1. (Base Case) $height\Big(\varphi'\Big) \leq size\Big(\varphi'\Big)$ for any propositional variable $\varphi' \in A$.

   But by definition, $height\Big(\varphi'\Big) = 1$ and $size\Big(\varphi'\Big) = 1$ when $\varphi' \in A$. And $1 \leq 1$, which is what we had to show.

2. (Inductive Case i) Assuming that $height\Big(\varphi'\Big) \leq size\Big(\varphi'\Big)$, we show that $height\Big(\neg\varphi'\Big) \leq size\Big(\neg\varphi'\Big)$.

   But by definition, $height\Big(\neg\varphi'\Big) = 1 + height\Big(\varphi'\Big)$ and $size\Big(\neg\varphi'\Big) = 1 + size\Big(\varphi'\Big)$. And $1 + height\Big(\varphi'\Big) \leq 1 + size\Big(\varphi'\Big)$ (what we had to show), as we already know $height\Big(\varphi'\Big) \leq size\Big(\varphi'\Big)$.

3. (Inductive Case ii) Assuming that $height\Big(\varphi_1\Big) \leq size\Big(\varphi_1\Big)$ and $height\Big(\varphi_2\Big) \leq size\Big(\varphi_2\Big)$, we show $height\Big((\varphi_1 \vee \varphi_2)\Big) \leq size\Big((\varphi_1 \vee \varphi_2)\Big)$.

But, by definition, $height\Big((\varphi_1 \vee \varphi_2)\Big) = 1 + max\Big(height\Big(\varphi_1\Big), height\Big(\varphi_2\Big)\Big)$ and, as $max(a,b) \leq a+b$ (for any naturals $a, b \in \mathbb{N}$), we have that $height\Big((\varphi_1 \vee \varphi_2)\Big) \leq 1 + height\Big(\varphi_1\Big) + height\Big(\varphi_2\Big)$. But $height\Big(\varphi_i\Big) \leq size\Big(\varphi_i\Big)$ $(1 \leq i \leq 2)$ by our hypothesis, and therefore $height\Big((\varphi_1 \vee \varphi_2)\Big) \leq 1 + size\Big(\varphi_1\Big) + size\Big(\varphi_2\Big)$. But, by definition, $size\Big((\varphi_1 \vee \varphi_2)\Big) = 1 + size\Big(\varphi_1\Big) + size\Big(\varphi_2\Big)$, and therefore $height\Big((\varphi_1 \vee \varphi_2)\Big) \leq size\Big((\varphi_1 \vee \varphi_2)\Big)$, what we had to prove.

4. (Inductive Case iii) Similar to Inductive Case ii.

q.e.d.

4.4 Exercise Sheet

Exercise 19. *Compute, using the function subf, the set of subformulae of the following formulae:*

1. $((\mathtt{p} \wedge \neg\mathtt{q}) \wedge \mathtt{r})$; *2.* $((\mathtt{p} \vee \neg\mathtt{q}) \wedge \mathtt{r})$; *3.* $\neg((\mathtt{p} \vee \neg\mathtt{q}) \wedge \mathtt{r})$.

Exercise 20. *Compute the abstract syntax trees of the following formulae:*

1. $((\mathtt{p} \wedge \neg\mathtt{q}) \wedge \mathtt{r})$;
2. $((\mathtt{p} \vee \neg\mathtt{q}) \wedge \mathtt{r})$;
3. $\neg((\mathtt{p} \vee \neg\mathtt{q}) \wedge \mathtt{r})$;
4. $(\neg(\mathtt{p} \vee \neg\mathtt{q}) \wedge \mathtt{r})$.

Exercise 21. *Recall the recursive definition of the function height* $: \mathbb{PL} \to \mathbb{N}$*, which computes, given a formula, the height of its abstract syntax tree. Compute the height of the formulae shown in Exercise 20.*

Exercise 22. *Recall the recursive definition of the function size* $: \mathbb{PL} \to \mathbb{N}$*, which computes the number of nodes in abstract syntax tree of a formula. Compute the size of the formulae shown in Exercise 20.*

Exercise 23. *Recall the recursive definition of the function prop* $: \mathbb{PL} \to 2^A$*, which computes, given a formula, the set of propositional variables occuring in the formula. Compute the set of propositional variables occuring in the formulae shown in Exercise 20.*

Exercise 24. *Show by structural induction that* $height(\varphi) < size(\varphi) + 1$ *for any formula* $\varphi \in \mathbb{PL}$.