Logic for Computer Science - Week 2
The Syntax of Propositional Logic

Ștefan Ciobâcă

November 30, 2017

1 An Introduction to Logical Formulae

In the previous lecture, we have seen what makes an argument valid. We have seen that several valid arguments could follow the same pattern. For example, take the following two arguments:

A1:

1. "All men are mortal."
2. "Socrates is a man."
3. So, "Socrates is mortal."

A2:

1. "All students are smart."
2. "John is a man."
3. So, "The Earth is round."

Both of the arguments are obviously instances of the same pattern:

P1

1. "All F are G."
2. "x is F."
3. So, "x is G."

In fact, any instance of this pattern (i.e. argument obtained by replacing "F" and "G" by properties and "x" by a particular individual) is valid.

Here is another example of argument pattern:

P2

1. "Either P or Q."
2. "Not P."

3. So, "Q."

Any instance of this pattern is valid as well. Here are two such instances that we have seen in the previous lecture:

Oranges are either fruits or musical instruments. Oranges are not fruits. Therefore, oranges are musical instruments.

Either John is in the library or in the bar. John is not in the library. Therefore, John is in the bar.

The schematic propositions that appear in such argument patterns such as "Either P or Q" are called *logical forms*, or, more modernly and simpler, *formulae*.

Propositional logic is concerned with the systematic study with formulae such as those appearing in the second argument patterns, namely the formulae that can be constructed from atomic propositions such as "P" and "Q" and logical connectives such as "and", "or", "not".

Formulae such as those appearing in the first argument pattern (e.g., "All F are G.") are called first-order formulae and we will study them in the second part of this course.

1.1 On the Use of Natural Language

As we have seen in the previous lecture, natural language (English, French, Romanian, etc.) features inherent ambiguities. We have already seen an example of such an ambiguity: "John and Mary are married" could mean either that they are married to each other or that they are married, not necessarily to each other.

We are no anticipating first-order logic (discussed during the second half of the course).

> When we will get to first-order logic, we will see that we can disambiguate between the two possible meanings of the sentence by:
>
> - either using a two-argument predicate "MarriedToEachOther(X, Y)" that means that "X" and "Y" are married to each other and writing the sentence as "MarriedToEachOther(John, Mary)";
> - or using a single-argument predicate "Married(X)" that means that "X" is married (without saying who "X" is married to) and writing the sentence as "Married(John) and Married(Mary)".

End of divagation into first-order logic.

Such ambiguities also occur in (informal) propositional logic. Take the sentence "It is not true that John is tall and Jane is short." It could either mean that the conjunction "John is tall and Jane is short" is not true, or simply that "John is not tall and Jane is short".

Such ambiguities impede the study of propositional logic. Therefore we will design a *formal language*, the language of propositional logic, where no ambiguity can occur.

Normally, when people say "formal", they mean it in a bad way, such as having to dress or act formally to go to dinner.

However, in computer science (and in mathematics), formal is a good thing: it means making things so precise that there is no possibility of misunderstanding.

2 The Formal Syntax of Propositional Logic

Propositional formulae will be strings (sequences of characters) over the alphabet of propositional logic.

The alphabet of propositional logic is simply a set of symbols. The symbols are categorised as follows:

1. $A = \{\mathtt{p}, \mathtt{q}, \mathtt{r}, \mathtt{p}', \mathtt{q}_1, \ldots\}$ is an infinite set of *propositional variables* that we fix from the very beginning;
2. $\{\neg, \wedge, \vee\}$ is the set of *logical connectives*;
3. $\{(,)\}$ is the set of auxiliary symbols; in our case, it consists of two symbols called *brakets*.

The set $L = A \cup \{\neg, \wedge, \vee, (,)\}$ is called the alphabet of propositional logic. We call a set an alphabet in computers science when we will use the elements of the set to make words.

Here are a few examples of words over L: $))\mathtt{p} \vee \wedge$, $\vee \vee \neg(\mathtt{p})$, $\neg(\mathtt{p} \vee \mathtt{q})$. Words, or strings, are simply sequences of symbols of the alphabet L. Some of these words will be formulae or, equivalently, well-formed formulae (wff). Some authors prefer to use the terminology "wff", but we will simply use "formula" by default in these lecture notes.

As an example, the last word above, $\neg(\mathtt{p} \vee \mathtt{q})$ is a formula of propositional logic, but $\vee \vee \neg(\mathtt{p})$ is not. The following definition captures exactly the set of propositional formulae.

Definition 2.1 (The Set of Propositional Formulae (PL))**.** *The set of formulae of propositional logic, denoted PL from hereon, is the only set of word over L satisfying the following conditions:*

1. *(Base Case) Any propositional variable, seen as a* 1*-symbol word, is in PL (equivalently, $A \subseteq PL$);*
2. *(Inductive Step i) If $\varphi \in PL$, then $\neg\varphi \in PL$ (equivalently, if the word φ is a propositional formula, then so is the word starting with the symbol $\neg$ and continuing with the symbols in φ);*
3. *(Inductive Step ii) If $\varphi_1, \varphi_2 \in PL$, then $(\varphi_1 \vee \varphi_2) \in PL$ (equivalently, exercise);*
4. *(Inductive Step iii) If $\varphi_1, \varphi_2 \in PL$, then $(\varphi_1 \wedge \varphi_2) \in PL$ (equivalently, exercise);*

5. *(Minimality Constraint) No other word (other than those constructible using the rules above) is in PL.*

Here are examples of elements of PL:

$$\mathtt{p} \quad \mathtt{q} \quad \neg\mathtt{p} \quad \neg\mathtt{q} \quad \neg\mathtt{p}' \quad \neg\neg\mathtt{p}_1 \quad (\mathtt{p} \vee \mathtt{q}) \quad (\mathtt{p} \wedge \mathtt{q}) \quad \neg(\mathtt{p} \vee \mathtt{q})$$

$$(\neg\mathtt{p} \wedge \neg\mathtt{q}) \qquad \neg(\neg\neg\mathtt{p} \vee \mathtt{p}) \qquad ((\mathtt{p} \vee \mathtt{q}) \wedge \mathtt{r}) \qquad (\mathtt{p} \vee (\mathtt{q} \wedge \mathtt{r}))$$

Here are examples of words not in PL:

$$\mathtt{pp} \qquad \mathtt{q}\neg\mathtt{q} \qquad \mathtt{q} \wedge \neg\mathtt{p} \qquad \mathtt{p} + \mathtt{q}$$

The definition of the set PL is an example of an *inductive definition*. Such definitions are really important in computer science and it is a must to understand them very well. In inductive definitions of set, there are usually some base cases, which say what "base" elements are part of the set and some inductive cases, which explain how to obtain new elements of the set from old elements. Another important part of an inductive definition is the minimality constraint, which says that nothing other than what is provable by the base case(s) and the inductive case(s) belongs to the set. It can be shown that the above set PL exists and is unique, but the proof is beyond the scope of this course.

2.1 Showing That a Word Is In *PL*

We can show that a word belongs to PL by explaining how the rules of the inductive definition were applied. Here is an example of a proof that $\neg(\mathtt{p} \vee \mathtt{q}) \in PL$:

1. $\mathtt{p} \in PL$ (by the Base Case, because $\mathtt{p} \in A$);
2. $\mathtt{q} \in PL$ (by the Base Case, as $\mathtt{q} \in A$);
3. $(\mathtt{p} \vee \mathtt{q}) \in PL$ (by the Inductive Case ii, with $\varphi_1 = \mathtt{p}$ and $\varphi_2 = \mathtt{q}$);
4. $\neg(\mathtt{p} \vee \mathtt{q}) \in PL$ (by the Inductive Case i, with $\varphi = (\mathtt{p} \vee \mathtt{q})$).

We can rearrange the above into an equivalent *annotated construction tree* for the formula $\neg(\mathtt{p} \vee \mathtt{q})$:

$$\dfrac{\dfrac{\dfrac{}{\mathtt{p} \in PL}\ \textit{Base Case} \qquad \dfrac{}{\mathtt{q} \in PL}\ \textit{Base Case}}{(\mathtt{p} \vee \mathtt{q}) \in PL}\ \textit{Inductive Case ii}}{\neg(\mathtt{p} \vee \mathtt{q}) \in PL}\ \textit{Inductive Case i}$$

You will see this notation several times in Computer Science and it is worth getting to know it. Each line is called an inference; below each line is the

conclusion of the inference and above the lines are the hypotheses (0, 1 or more). Besides each lines is the name of the rule that was applied.

If we drop all annotation, we obtain the following bare *construction tree* for the formula:

$$\dfrac{\dfrac{\dfrac{}{\mathtt{p}} \quad \dfrac{}{\mathtt{q}}}{(\mathtt{p} \vee \mathtt{q})}}{\neg(\mathtt{p} \vee \mathtt{q})}$$

It is easy to see that a word belongs to PL iff there is a construction tree for it.

2.2 The Main Connective of a Formula

A formula that consists of a single propositional variable, such as $\mathtt{p}$ or $\mathtt{q}$, is called an *atomic formula*. This explains why the set A of propositional variables is called A (A stands for *atomic*).

More complex formulae, such as $\neg\mathtt{p}$ or $\mathtt{p} \vee \mathtt{q}$, are called *molecular* (to distinguish them from atomic formulae).

Each molecular formula has a *main connective*, which is given by the last inference in its construction tree. For example, the main connective of the formula $\neg(\mathtt{p} \vee \mathtt{q})$ is $\vee$ (the disjunction), while the main connective of the fomrula $(\neg\mathtt{p} \vee \mathtt{q})$, is $\neg$ (the negation). We call formulae whose main connective is $\wedge$ *conjunctions*. Similarly, if the main connective of a formula is a $\vee$, it is a *disjunction* and if the main connective is $\neg$, then it is a *negation*.

2.3 Showing That a Word Is Not in *PL*

It is somewhat more difficult (or rather more verbose) to show that a word is not in PL.

If the word is not over the right alphabet, such as the three-symbol word $\mathtt{p} + \mathtt{q}$, which uses a foreign symbol $+$, then we can easily dismiss it as PL only contains words over L.

However, if the word is over the right alphabet and we want to show that it is not part of PL, we must make use of the minimality condition. Here is an example showing that $(\mathtt{p}\neg\mathtt{q}) \notin PL$:

Let's assume (by contradiction) that $(\mathtt{p}\neg\mathtt{q}) \in PL$. Then, by the minimality condition, it follows that $(\mathtt{p}\neg\mathtt{q}) \in PL$ must be explained by one of the rules Base Case, Inductive Case i – iii.

But we cannot apply the Base Case to show that $(\mathtt{p}\neg\mathtt{q}) \notin PL$, since $(\mathtt{p}\neg\mathtt{q}) \notin A$.

But we cannot apply the Inductive Case i either, because there is no formula φ such that $(\mathtt{p}\neg\mathtt{q}) = \neg\varphi$ (no matter how we would choose $\varphi \in PL$, the first symbol of the word on the lhs would be (, while the first symbol of the word on the rhs would be $\neg$).

But we cannot apply the Inductive Case ii either, because there are no formulae $\varphi_1, \varphi_2 \in PL$ such that $(\mathtt{p}\neg\mathtt{q}) = (\varphi_1 \vee \varphi_2)$ (no matter how we would choose $\varphi_1, \varphi_2 \in PL$, the word on the lhs would not contain the symbol $\neg$, while while the the word on the rhs would contain it).

By similar reasons, we cannot apply Inductive Case iii either.

But this contradicts our assumption and therefore it must be the case that $(\mathtt{p}\neg\mathtt{q}) \notin PL$.

2.4 Unique Readability

The definition of PL has an important property called *unique readability*:

Theorem 2.1 (Unique Readability of Propositional Formulae)**.** *For any word $\varphi \in PL$, there is a unique construction tree.*

The property above is also sometimes called unambiguity of the grammar. Proving the Unique Readability Theorem is beyond the scope of the present lecture notes. Instead we will try to understand the importance of the property above by showing how an alternative, fictive, definition of the set PL would be ambiguous:

Beginning of Wrong Definition of *PL*.

Imagine that the Inductive Case ii would read:

⋮

3. (Inductive Step ii) If $\varphi_1, \varphi_2 \in PL$, then $\varphi_1 \vee \varphi_2 \in PL$ (equivalently, exercise);

⋮

The only difference would be that we do not place parantheses around disjunctions. With this alternative, fictive, definition of PL, we have that the word $\neg p \vee q \in PL$. However, this word has two different construction trees:

$$\dfrac{\dfrac{\overline{\mathtt{p}} \qquad \overline{\mathtt{q}}}{\mathtt{p} \vee \mathtt{q}}}{\neg\mathtt{p} \vee \mathtt{q}} \qquad \dfrac{\dfrac{\overline{\mathtt{p}}}{\neg\mathtt{p}} \qquad \overline{\mathtt{q}}}{\neg\mathtt{p} \vee \mathtt{q}}$$

Such an ambiguity would be very troubling, because we would not know if $\neg p \vee q$ is a disjunction (if we would use the construction tree on the right) or a negation (the construction tree on the left). In fact, avoiding such syntactic ambiguities was the main reason why we left natural language and began studying formal logic.

End of Wrong Definition of *PL*.

I hope that you can now see that the Unique Readability Theorem is non-trivial (as an exercise to the more mathematically inclined students, I suggest

that you try to prove it) and that it is a very important property of our definition of formulae, since it essential says that any propositional formula can be read unambiguously.

3 Functions Defined Inductively on *PL*

Reminder 3.1. *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:

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

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

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 for the auxiliary symbols in L.

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 $\mathtt{p} \vee \mathtt{q} \notin PL$ is not a formula.

The function *subf* is the first in a long line of functions defined recursively on the structure of a formula $\varphi \in PL$. 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.

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

$ast : PL \to Trees$, defined by:

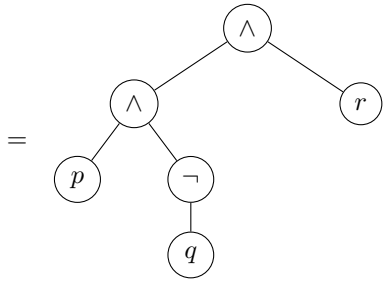
$$ast\Big(\varphi\Big) = \begin{cases} \boxed{F}, & \text{if } \varphi \in A; \\ \boxed{\neg}\ ast\Big(\varphi'\Big), & \text{if } \varphi = \neg\varphi' \text{ and } \varphi' \in PL; \\ ast\Big(\varphi_1\Big)\ \boxed{\vee}\ ast\Big(\varphi_2\Big), & \text{if } \varphi = (\varphi_1 \wedge \varphi_2)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

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

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

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

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



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.

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

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

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

The next function, $size : PL \to \mathbb{N}$, computes the size of the ast o formula (i.e. the number of nodes):

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

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

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

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

3.1 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 PL.

For the case of PL, the structural induction principle is that, in order to show a theorem of the form

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

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) $P\Big((\varphi_1 \vee \varphi_2)\Big)$ holds whenever $P\Big(\varphi_1\Big)$ and $P\Big(\varphi_2\Big)$ hold;
4. (Inductive Case iii) $P\Big((\varphi_1 \wedge \varphi_2)\Big)$ 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 3.1 (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.

□