CS 2742 (Logic in Computer Science)
Lecture 8

Antonina Kolokolova

February 3, 2014

3 Resolution.

Recall that a formula is in the CNF (conjunctive normal form) if it is a $\wedge$ of $\vee$s of literals (variables or their negation.)

In this lecture we will talk about proving (or, actually, finding contradictions) statements that are in this special form.

Definition 1 (Resolution rule)**.** *: Given two clauses of the form $(C \vee x)$ and $(D \vee \neg x)$, where C and D are (possibly empty) disjunction of variables, can derive a (possibly empty) clause $(C \vee D)$.*

That is,

$$(C \vee x) \wedge (D \vee \neg x) \rightarrow (C \vee D)$$

where $C = (l_1 \vee \cdots \vee l_k)$ and $D = (l'_1 \ldots l'_{k'})$ for some literals. If there is a repeated literal, only write it once.

Note that you may end up deriving an "empty" clause, that is, a $\vee$ of zero literals. But the only way this could happen is when two clauses being resolved are (x) and $(\neg x)$. But $x \wedge \neg x$ is always false, a contradiction. So deriving an empty clause proves that the original formula was a contradiction (which is what we usually want to show).

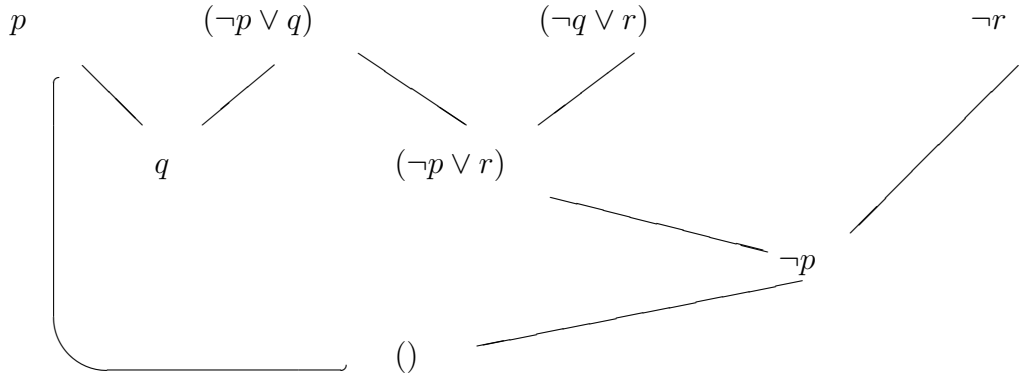
Example 1. Consider the following statements:
p: it is sunny
q: the weather is good
r: I spend the day outside

Now consider the following argument:

p
$p \to q$
$q \to r$
$\therefore r$

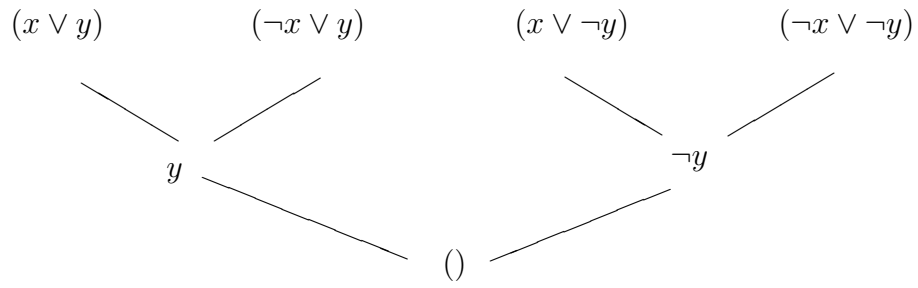
We want to prove that this is a valid argument, that is, p, $p \to q$ and $q \to r$ together imply r (this in general is called a *transitivity* law). Resolution proof method allows us to find contradictions: so we represent this problem as a contradiction $p \wedge (p \to q) \wedge (q \to r) \wedge \neg r$. Note that here again we are using the fact that the negation of an implication (in this case, premises imply the conclusion) is a conjunction of premises and negation of the conclusion.

Resolution works with CNF formula, so the first step is to convert the formula above into CNF. In this case, it is very easy: just apply the definition of implication to the second and third clause to obtain $p \wedge (\neg p \vee q) \wedge (\neg q \vee r) \wedge \neg r$.



For comparison, you can also prove this using rules of inference such as Modus Ponens. That allows you to derive q from the first two lines (p and $p \to q$ give q), and then derive r from q and $q \to r$.

Example 2. Let's look at another example. Now there are only two variables, and the four clauses contain all possible combinations (so the formula is a contradiction) $(x \vee y) \wedge (\neg x \vee y) \wedge (x \vee \neg y) \wedge (\neg x \vee \neg y)$. You can easily see that any truth assignment to x and y falsifies one clause.



Resolution proof system is very powerful in that given any formula in CNF form it can check

whether it is a contradiction. However, resolution is not very efficient. Although it is much more efficient in practice than using truth tables (for example, the $(x_1 \wedge (x_1 \to x_2) \wedge (x_2 \to x_3) \wedge \cdots \wedge (x_{n_1} \to x_n) \wedge \neg x_n$ can be done by a resolution proof system in n steps, whereas the truth table would have 2^n lines), there are examples of problems on which resolution proof system has to explore all possibilities, and so is similar to the truth table method in efficiency. The most prominent such example is the PigeonHole Principle, which says that it is impossible to put $n+1$ element (pigeon) into n places (holes) without two elements getting into the same place. You needed to use this principle, this kind of counting reasoning to solve the last lecture puzzle.

The resolution is still the most popular method of programming automated proof systems because if its clarity and simplicity. However, when working with or programming such a system keep in mind that resolution "can't count".

At this point a natural question is: how can we use resolution to deal with general propositional formulas, that is, ones not in CNF form. Of course, we could create a truth table and write a corresponding CNF formula, but if we have a truth table then we can just test whether a formula is a contradiction by checking if all entries in the last column are Fs. Another idea is to manipulate the formula using logic identities until it becomes a CNF. This is possible, but unfortunately can result in a very large (size comparable to the truth table) formula, especially when converting a DNF into a CNF (think of converting a formula $(x_1 \wedge y_1) \vee (x_2 \wedge y_2) \ldots (x_n \wedge y_n)$ into a CNF: you will end up with 2^n clauses on n variables each, one for every combination of Xs and Ys).

A different idea is to add new variables in such a way that the new formula is a contradiction if and only if the original one was a contradiction. Here is one way of doing it.

1) Assign a new variable to every binary connector $(\wedge, \vee, \to, \leftrightarrow)$ in the formula (starting from the outermost connector, operation to be done last). For example, if the original formula is $((x_1 \to x_2) \vee \neg((\neg x_1 \leftrightarrow x_3) \vee x_4)) \wedge \neg x_2$, then assign variable y_1 to outermost $\wedge$ connector,then the variable y_2 to the $\vee$ before x_4 and so on (think of writing a parse tree of the formula and assigning y_1 to the top, y_2 and (possibly) y_3 to its children, etc).

2) Next, write the conditions describing the new variables: for example, if our y_6 was assigned to $\leftrightarrow$ in $\neg x_1 \leftrightarrow x_3$, then we need the condition $(y_6 \leftrightarrow (\neg x_1 \leftrightarrow x_3))$ which says that y_6 is true if and only if the expression $(\neg x_1 \leftrightarrow x_3)$ is true. The continue writing similar expressions for other y_is, using other y_i variables to denote subformulas (for example, the top $\wedge$ denoted by y_1 gets $(y_1 \leftrightarrow (y_2 \wedge \neg x_2))$. Finally, add clause (y_1) to the formula which will mean that the whole original formula is true.

3) We are almost done: our formula is a conjunction of small parts; most importantly, each small part $(y_i \leftrightarrow (\ldots))$ only involves three variables (for example, y_1, y_2 and x_2 in definition of y_1, or $y2 \leftrightarrow (y_3 \vee \neg y_4)$ for y_3). Three variables give us a small enough truth

table (8 lines), that we can write a CNF for this small table with just 7 or less clauses, each with at most 3 variables.. So the size of the new formula becomes approximately the number of logical connectives (times a constant).

4) The resulting formula is a CNF, and although it has many more variables, it is a contradiction if and only if the original formula was.

3.1 Complete set of connectives

From what we have done it is easy to see that any possible truth table (we will start saying "boolean function" soon) can be represented by a formula written with just $\wedge, \vee$ and $\neg$. Such set of connectives is called *complete.*

A stronger result tells us that even $\vee$ is not necessary in this set (homework exercise). But $\neg$ is necessary, because every formula without $\neg$ is true when all of its variables are set to true. So contradiction cannot be described by such a formula, and any truth table with a 0 in the last column when all of its inputs are true cannot be represented by a formula using only $\vee$ and $\wedge$.

We have seen that there are two connectives that allow us to construct formulas representing all possible truth tables. Is there a single connective that can be used to construct formulas with all possible truth tables? The answer is yes, and such connective is called NAND (stands for "not-and") in digital circuit design, or Sheffer's stroke (written as $|$). It is defined so $p|q$ is false when both p and q are true, and true otherwise.

p	q	$p\|q$
1	1	0
1	0	1
0	1	1
0	0	1

You can see here that it has values exactly opposite of the values of $p \wedge q$, hence the name NAND. To show that it is complete, let us look at how to simulate $\neg$ and $\wedge$ with it. Once we have both $\neg$ and $\wedge$, we can rewrite any formula to an equivalent one which only uses $|$ as a connective using ideas from the problem in your homework assignment.

To show how to represent $\neg p$, note that the first row of the truth table has 0 for the value of the Sheffer's stoke, and the last row has a 1. So $\neg p \iff p|p$. Similarly, $p \wedge q \iff \neg(p|q) \iff (p|q)|(p|q)$, since Sheffer's stroke is the negation of $\wedge$ and we just showed how to do $\neg$.