6c Lecture 5 & 6: April 15 & 17, 2014

4 Proofs by resolution and a Hilbert-style formal proof system

Our next goal is to discuss some formal proof systems. These systems will be for providing a formal proof that S implies ϕ, for some set of formulas S and ϕ such that this is true. Here we will write $S \vdash \phi$ to indicate that there is a formal proof of ϕ from S (according to whatever proof system we are discussing).

We'll be especially interested in verifying two properties of the systems we will discuss. First, *soundness*, which is the property that if $S \vdash \phi$, then S implies ϕ (simply stated, our formal proof system doesn't prove false things). Second, *completeness*, which is the property that if S implies ϕ, then $S \vdash \phi$ (simply stated, our formal proof system proves every true thing). Together one can think of soundness and completeness as saying that our proof system is perfect: it proves every true thing and no false things.

Recall that by compactness, since whenever S implies ϕ we have that some finite subset S' of S implies ϕ, it's enough for us to describe proof systems which only prove logical implications from finite sets of assumptions.

Now of course, we already have one way of proving logical implications: truth tables. However, truth table are very inefficient in practice, and time consuming both to compute initially and to verify as correct afterwards. Our goal now is to describe some systems which have the potential for being much more efficient, and closer to real mathematical practice.

4.1 Efficiently converting logical implications into checking satisfiability of formulas in CNF

We'll begin by showing that having a system for proving logical implications is equivalent (in a way that can be computed efficiently) to having a system for proving when CNF formulas are contradictions. We start with the following lemma which gets us halfway there.

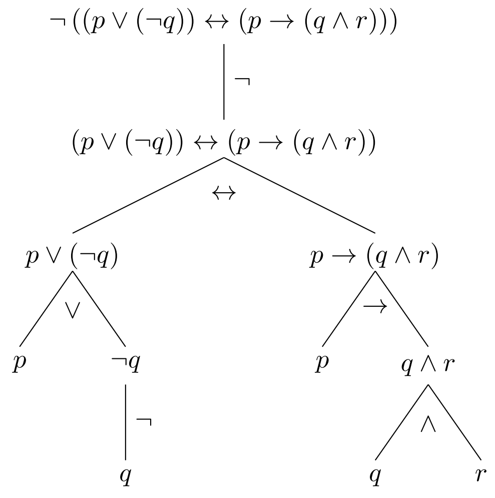
Lemma 4.1. *If $S = \{\phi_1, \phi_2, \dots, \phi_n\}$ is a finite set of formulas, then S implies ψ iff $(\phi_1 \wedge \phi_2 \wedge \dots \wedge \phi_n) \to \psi$ is a tautology iff $\neg\phi_1 \vee \neg\phi_2 \vee \dots \neg\phi_n \vee \psi$ is contradictory.*

Proof. Given in class. □

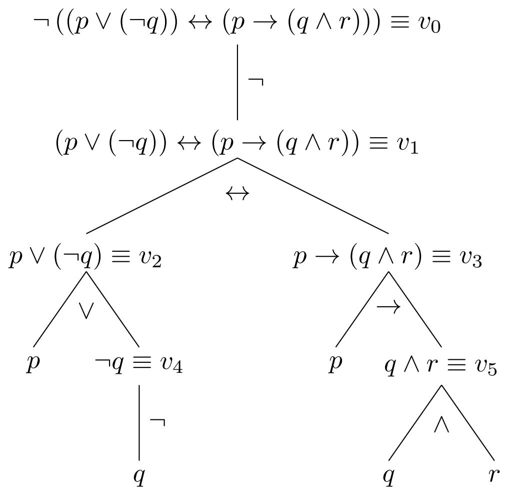
Next, we show that we can assume our formulas which we want to prove contradictory are in CNF, since there is an efficient algorithm for transforming any satisfiability problem into an equivalent one for a CNF formula.

Theorem 4.2. *There is an efficient (linear-time) algorithm for converting any formula ϕ into a formula ψ in CNF which is satisfiable iff ϕ is.*

Proof. As we describe the algorithm in abstract, we will also work an example to illustrate the algorithm. Suppose we have a formula such as $\phi = \neg((p \vee (\neg q)) \leftrightarrow (p \rightarrow (q \wedge r)))$. Then consider the parse tree for this formula:


$\neg((p \vee (\neg q)) \leftrightarrow (p \rightarrow (q \wedge r)))$
$\neg$
$(p \vee (\neg q)) \leftrightarrow (p \rightarrow (q \wedge r))$
$\leftrightarrow$
$p \vee (\neg q)$
$p \rightarrow (q \wedge r)$
$\vee$
$\rightarrow$
p
$\neg q$
p
$q \wedge r$
$\neg$
$\wedge$
q
q
r

For each node having children in this tree starting from the root, we first associate a new propositional variable $v_0, v_1, \ldots, v_n$. So now we have:


$\neg((p \vee (\neg q)) \leftrightarrow (p \rightarrow (q \wedge r))) \equiv v_0$
$\neg$
$(p \vee (\neg q)) \leftrightarrow (p \rightarrow (q \wedge r)) \equiv v_1$
$\leftrightarrow$
$p \vee (\neg q) \equiv v_2$
$p \rightarrow (q \wedge r) \equiv v_3$
$\vee$
$\rightarrow$
p
$\neg q \equiv v_4$
p
$q \wedge r \equiv v_5$
$\neg$
$\wedge$
q
q
r

Having done this, now each node in the tree is associated a variable (either it has no children and is associated one of the original variables, or it is has children and has been assigned one of our new variables). Now for each node v_i having children in the tree, make the formula stating v_i is equivalent to the connective associated to this node applied to its children. In our example, this gives the formulas $v_0 \leftrightarrow (\neg v_1)$, $v_1 \leftrightarrow (v_2 \leftrightarrow v_3)$, $v_2 \leftrightarrow (p \vee v_4)$, $v_3 \leftrightarrow (p \rightarrow v_5)$, $v_4 \leftrightarrow (\neg q)$, and $v_5 \leftrightarrow (q \wedge r)$. Now take the conjunction of all these formulas with the variable assigned to our root v_0:

$$v_0 \wedge [v_0 \leftrightarrow (\neg v_1)] \wedge [v_1 \leftrightarrow (v_2 \leftrightarrow v_3)] \wedge [v_2 \leftrightarrow (p \vee v_4)] \wedge \\ [v_3 \leftrightarrow (p \rightarrow v_5)] \wedge [v_4 \leftrightarrow (\neg q)] \wedge [v_5 \leftrightarrow (q \wedge r)]$$

One can prove by induction on formulas that this process yields a formula which is satisfiable iff the original formula ϕ is satisfiable.

To efficiently convert this new formula to conjunctive normal form, it suffices to note that for each of the possible forms of the conjuncts we have in such a formula (corresponding to each logical connective), there is a short and equivalent way of expressing the formula in conjunctive normal form:

Original formula	Equivalent form
$x \leftrightarrow (\neg y)$	$(\neg x \vee \neg y) \wedge (x \vee y)$
$x \leftrightarrow (y \wedge z)$	$(\neg x \vee y) \wedge (\neg z \vee z) \wedge (z \vee \neg y \vee \neg z)$
$x \leftrightarrow (y \vee z)$	$(\neg x \vee y \vee z) \wedge (x \vee \neg y) \wedge (x \vee \neg z)$
$x \leftrightarrow (y \rightarrow z)$	$(\neg x \vee \neg y \vee z) \wedge (x \vee y) \wedge (x \vee \neg z)$
$x \leftrightarrow (y \leftrightarrow z)$	$(\neg x \vee \neg y \vee z) \wedge (\neg x \vee y \vee \neg z) \wedge (x \vee \neg y \vee \neg z) \vee (\neg x \wedge \neg y \wedge z)$

For example, our example ϕ now yields the following formula ψ:

$$v_0 \wedge [(\neg v_0 \vee \neg v_1) \wedge (v_0 \vee v_1)] \wedge \\ [(\neg v_1 \vee \neg v_2 \vee v_3) \wedge (\neg v_1 \vee v_2 \vee \neg v_3) \wedge (v_1 \vee \neg v_2 \vee \neg v_3) \vee (\neg v_1 \wedge \neg v_2 \wedge v_3)] \wedge \\ [(\neg v_2 \vee p \vee v_4) \wedge (v_2 \vee \neg p) \wedge (v_2 \vee \neg v_4)] \wedge [(\neg v_3 \vee \neg p \vee v_5) \wedge (v_3 \vee p) \wedge (v_3 \vee \neg v_5)] \wedge \\ [(\neg v_4 \vee \neg q) \wedge (v_4 \vee q)] \wedge [(\neg v_5 \vee q) \wedge (\neg r \vee r) \wedge (r \vee \neg q \vee \neg r)]$$

It will be an exercise to prove that for every formula, the formula ψ obtained by this algorithm is satisfiable iff ϕ is satisfiable. □

4.2 The method of resolution

Now we will describe a method for proving that a formula in CNF is unsatisfiable. Recall that a formula in CNF has the form $(\ell_{1,1} \vee \ldots, \ell_{1,n_1}) \wedge (\ell_{1,2} \vee \ldots, \ell_{1,n_2}) \wedge \ldots \wedge (\ell_{1,k} \vee \ldots, \ell_{1,n_k})$, where the $\ell_{i,j}$ are literals. Since the order of the literals $\ell_{i,1} \ldots \ell_{i,n_i}$ does not matter inside each conjunct, we can think of them as simply being a set of literals, which we call a clause. It is this observation which motivates the following definition, which essentially is just new terminology for parts of a formula in CNF.

Definition 4.3. A *clause* is a set of literals. A clause is *satisfied* by a valuation v if at least one of the literals in the clause is true according to the valuation. A set of clauses is *satisfiable* iff there is some valuation making each clause in the set satisfied.

For example, the CNF formula $\{(p \vee \neg q \vee r) \wedge (\neg p \vee q \vee r) \wedge (\neg p \vee \neg q \vee \neg r)\}$ is associated to the set of clauses $\{\{p, \neg q, r\}, \{\neg p, q, r\}, \{\neg p, \neg q, \neg r\}\}$, and this set of clauses is satisfiable iff the original formula is.

Now suppose we have two disjunctions of literals where some propositional variable p appears in the first, and $\neg p$ appears in the second. For example, $p \vee q \vee r$ and $\neg p \vee q \vee \neg t$. If both these disjunctions are true, then we can conclude that at least one of the remaining literals other than p and $\neg p$ is true. This is because if p is false, then q or r is true (to make the first disjunction true), and if p is true, then q or $\neg t$ is true. Hence, from $p \vee q \vee r$ and $\neg p \vee q \vee \neg t$ we can conclude $q \vee r \vee \neg t$. This idea is what leads us to make the definition of a resolvent of two clauses.

Definition 4.4. Suppose c_1 and c_2 are clauses. We say that a third clause c is a *resolvent* of c_1 and c_2 if there is a propositional variable p such that c_1 contains p, c_2 contains $\neg p$, and $c = (c_1 \setminus \{p\}) \cup (c_2 \setminus \{\neg p\})$.

We denote this by the diagram

$$\begin{tikzcd} & c \arrow[dl, no head] \arrow[dr, no head] & \\ c_1 & & c_2 \end{tikzcd}$$

Hence, for example:

- $\{r, \neg s, t\}$ is a resolvent of $\{\neg p, r, t\}$ and $\{p, \neg s, r, t\}$.
- The empty clause $\{\}$ is a resolvent of $\{q\}$ and $\{\neg q\}$.
- $\{p, \neg p\}$ is a resolvent of $\{p, \neg q\}$ and $\{\neg p, q\}$.
- $\{\neg q, q\}$ is a resolvent of $\{p, \neg q\}$ and $\{\neg p, q\}$.

Note that the empty clause is unsatisfiable; there is no literal in this clause which can be made true by any valuation.

By our reasoning above, we have the following fact, proved in class:

Lemma 4.5. *If $\{c_1, c_2\}$ is a set of clauses and c is a resolvent of c_1 and c_2, then if $\{c_1, c_2\}$ is satisfiable, then so is c.*

Definition 4.6. A *proof by resolution* of a clause c from a set of clauses C is a finite tree whose vertices are each assigned a clause such that

1. The root is assigned c
2. If a vertex assigned c has children it has exactly two children and c is a resolvent of the clauses assigned to them.
3. If a vertex has no children it is assigned a clause from C.

We give an example of a proof by resolution of $\{q\}$ from the set of clauses $\{\{p, s\}, \{\neg p, q\}, \{\neg q, s\}, \{\neg p, q, \neg s\}, \}$:

$$\begin{tikzcd}
 & & & \{q\} \arrow[dl, dash] \arrow[drr, dash] & & & \\
 & & \{s\} \arrow[dl, dash] \arrow[dr, dash] & & & \{\neg s, q\} \arrow[dl, dash] \arrow[dr, dash] & \\
 & \{q, s\} \arrow[dl, dash] \arrow[dr, dash] & & \{\neg q, s\} & \{\neg p, q, \neg s\} & & \{p, s\} \\
\{p, s\} & & \{\neg p, q\} & & & &
\end{tikzcd}$$

The following theorem is essentially soundness for proofs by resolution.

Theorem 4.7 (Soundness for proofs by resolution)**.** *If there is a proof by resolution of a clause c from a set of clauses C, then if C is satisfiable, then so is c.*

Proof. In class, proved by induction on formulas. The key fact one needs to use in this proof is Lemma 4.5 above. □

In class we also proved the completeness theorem for proofs by resolution. We began with the following definition:

Definition 4.8. Suppose C is a set of clauses. Then we define

$$C(p = \text{True}) = \{c \setminus \{\neg p\} : c \in C \wedge p \notin c\}.$$

and

$$C(p = \text{True}) = \{c \setminus \{p\} : c \in C \wedge \neg p \notin c\}.$$

For example, if $C = \{\{p, s\}, \{\neg p, q\}, \{\neg q, s\}, \{\neg p, q, \neg s\}, \}$, then $C(p = \text{True}) = \{\{q\}, \{\neg q, s\}, \{q, \neg s\}\}$, and $C(p = \text{False}) = \{\{s\}, \{\neg q, s\}\}$.

Lemma 4.9. *Suppose C is a set of clauses. Then $C(p = \text{True})$ is satisfiable iff there is a valuation satisfying C such that p is true. Further, $C(p = \text{False})$ is satisfiable iff there is a valuation satisfying C such that p is false.*

Theorem 4.10 (Completeness for proofs by resolution)**.** *If C is an unsatisfiable set of clauses, then there is a proof by resolution of the empty clause from C.*

Proof. Given in class. The rough idea is to prove the theorem by induction on the number of propositional variables $\{p_1, \ldots, p_n\}$ used in the set of clauses. Assuming C is an unsatisfiable set of clauses in the variables $\{p_1, \ldots, p_n, p_{n+1}\}$, both $C(p_n = \text{True})$ and $C(p_n = \text{False})$ are unsatisfiable by Lemma 4.9. We then explained a way of combining proofs of the empty clause from $C(p_n = \text{True})$ and $C(p_n = \text{False})$ to obtain a proof of the empty clause from C. □

4.3 A Hilbert-style formal proof system

One disadvantage of proofs of resolution is that they bear little resemblance to the mathematical proofs we generally use in practice. Next, we describe a formal proof system which is closer to the proof we usually use in mathematics. The Hilbert-style system we will describe has 3 logical axioms and 1 rule. This system deals only with proofs of formulas involving the connectives $\neg$ and $\to$.

Definition 4.11. For any formulas ϕ, ψ, θ, the following are logical axioms:

1. $\phi \to (\psi \to \phi)$.
2. $(\phi \to (\psi \to \theta)) \to ((\phi \to \psi) \to (\phi \to \theta))$
3. $(\neg\phi \to \neg\psi) \to ((\neg\phi \to \psi) \to \phi)$.

In addition, we have the rule (modus ponens) that if ϕ and $\phi \to \psi$ are true, then ψ is true, which we represent with the diagram:

$$\begin{tikzcd} & \psi \arrow[dl, no head] \arrow[dr, no head] & \\ \phi & & \phi \to \psi \end{tikzcd}$$

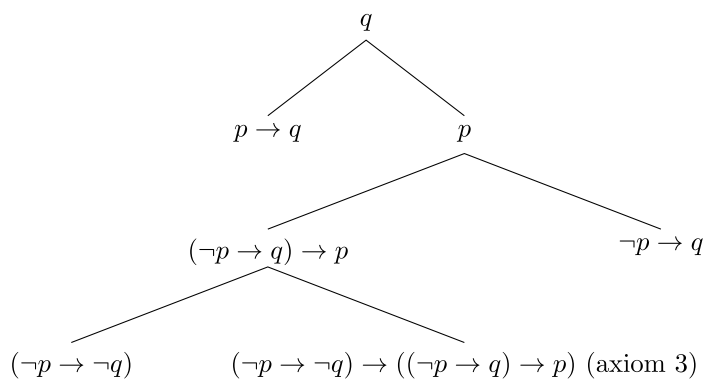
Now we can define a formal proof in this system as follows:

Definition 4.12. A *Hilbert-style proof* of a formula θ from a set of formulas S is a finite tree whose vertices are each assigned a formula such that

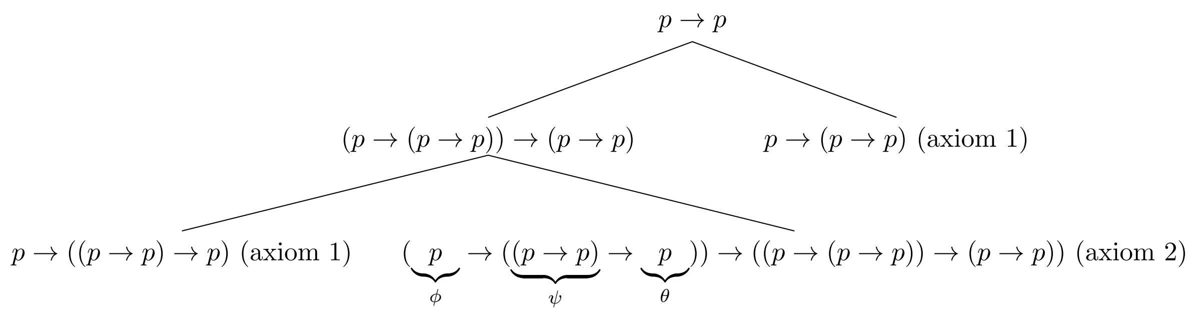
1. The root is assigned θ
2. If a vertex assigned ψ has children it has exactly two children, one is assigned some formula ϕ and the other is assigned $\phi \to \psi$ (so this corresponds to an instance of modus ponens).
3. If a vertex has no children it is assigned a formula from S or a logical axiom.

If there is such a proof θ from S, we write $S \vdash \theta$.

We give an example showing that $\{\neg p \to q, \neg p \to \neg q, p \to q\} \vdash q$



We give another example showing the from the empty set of formulas we can prove $p \to p$ (and similarly we can prove $\phi \to \phi$ for every formula ϕ).


$p \to p$
$(p \to (p \to p)) \to (p \to p)$
$p \to (p \to p)$ (axiom 1)
$p \to ((p \to p) \to p)$ (axiom 1)
$(\underbrace{p}_{\phi} \to (\underbrace{(p \to p)}_{\psi} \to \underbrace{p}_{\theta})) \to ((p \to (p \to p)) \to (p \to p))$ (axiom 2)