Lecture 3 (Sep 11)

Lecture outline:

- logic programming systems
- proof by contradiction
- conjuctive normal form
- logic programming representation of problems

As you may have already noticed logic programming (LP) systems in general, and Prolog in particular, operate in the following manner:

- **Input:** a problem description given as a set P of logical sentences, and a goal g which is another logical sentence.
- **Output:** "yes" if g logically follows from P ($P \models g$), "no" otherwise or if some problem occured.

Thus, an LP system is in essence an implementation of a *logical inference procedure*. A *logical inference procedure* is a procedure (an algorithm) which given a set of logical sentences P and another sentence g determines if $P \models g$.
Resolution is one of such inference procedures, based on the *proof by contradiction*:

$$P \models g \text{ if and only if the set } P \cup \{\neg g\} \text{ is inconsistent .}$$

"Inconsistent" means "contains a contradiction", but you should already know that.
An example of proof by contradiction:

Example 1. Given $P = \{\forall N f(N) = 2 \cdot N\}$
Lets prove goal $g_1 = (f(2) = 4)$ by contradiction. P and the negation of the goal provide the following assumptions:

$$\forall N f(N) = 2 \cdot N, \tag{1}$$

$$\neg(f(2) = 4). \tag{2}$$

From (1) using the rules of predicate logic and arithmetic we obtain $f(2) = 4$ (for $N = 2$), which contradicts (2).
Lets prove the goal $g_2 = \exists M(f(3) = M)$. P and the negation of the goal provide the following assumptions:

$$\forall N f(N) = 2 \cdot N, \tag{3}$$

$$\neg\exists M(f(3) = M) \equiv \forall M \neg(f(3) = M) \quad (4)$$

From (3) we have $f(3) = 6$ (for $N = 3$), from (4) we have $\neg f(3) = 6$ (for $M = 6$), contradiction. Note that during the proof we had to assign the value 6 to M. Thus, g_2 could be seen as a query $f(3) =?$ – during the proof we obtain the answer as a "side-effect" of the proof. You saw some of this in the Prolog examples I showed in the class. □

Now, we begin to describe how resolution procedure actually works. We start with propositional case.
The standard propositional resolution works on formulas in *conjunctive normal form (CNF)*. There is also resolution procedure for arbitrary formulas, but we won't talk about it. What is CNF ? First a few definitions:

Definition 2. *A* literal *is a propositional variable or its negation*

For example, p, q, $\neg r$, $\neg q$ are literals. The first two are examples of *positive* literals, the other two are examples of *negative* literals.

Definition 3. *A* clause *is a disjunction ($\vee$) of literals*

For example, $p \vee \neg q \vee s$ is a clause. Note that so is just p, i.e. one literal is a clause. Also, an *empty clause*, i.e. a clause that does not have literals, is a clause – the truth value of such clause is always 0 (false).

Definition 4. *A propositional formula is said to be in* conjunctive normal form (CNF) *if it is a conjunction of clauses.*

For example, the formula

$$(p \vee \neg q) \wedge (r \vee \neg s \vee \neg q) \wedge t$$

is in CNF (remember, one-literal clauses are allowed).
Why CNF ? Easy representation in computer systems (array of arrays, for example), many algorithms are more straightforward when defined on CNF, including the resolution inference procedure.

Fact 5. *For every propositional formula, there exists an equivalent formula in CNF.*

From logical deduction point of view, a formula in CNF $clause_1 \wedge clause_2 \wedge \cdots \wedge clause_n$ is equivalent to a *set* of clauses $\{clause_1, clause_2, \ldots, clause_n\}$. In this context , means conjunction ($\wedge$). By convention, a clause $l_1 \vee l_2 \vee ... \vee l_k$ is also seen as a set – this time a set of literals $\{l_1, l_2, ..., l_k\}$. In this context , means disjunction ($\vee$). To represent clauses in logic programming we group literals depending on polarity, and write a clause in the following way:

positive literals :– negative literals (but without negation)

For example,

Example 6. Represent the clause $p \vee \neg q \vee \neg s \vee t \vee r$ in logic programming notation.
First, write the clause as set of literals:

$$\{p, \neg q, \neg s, t, r\},$$

then, group positive literals to the left, negative to the right:

$$\{p, t, r \neg q, \neg s\},$$

and now, the logic programming notation for the above clause is

$$p, t, r :- q, s. \tag{5}$$

Another way to look at this: implication, i.e. (5) is equivalent to

$$(p \vee t \vee r) \leftarrow (q \wedge s)$$

So, to represent a problem described as a set P of propositional formulas $f_1, \ldots, f_n$, we

1. Convert each f_i to CNF – let's call it $CNF(f_i)$.
2. Represent each clause of $CNF(f_i)$ using the logic programming notation.
3. Put all the clauses together.

Example 7. Let

$$f_1 = (p \leftrightarrow q) \rightarrow (s \vee \neg q),$$
$$f_2 = s \rightarrow (q \wedge \neg p).$$

Represent $P = \{f_1, f_2\}$ using logic programming notation.

Ok, so we need to translate f_1 and f_2 to CNF. The technique is as follows: first, get rid of implications ($\rightarrow$) and equalities ($\leftrightarrow$) so the resulting formula contains only $\wedge, \vee, \neg$; second, push all the negations to the variables, using the distributive laws.
So, for f_1 (I did it in a longer way in the class):

$$\begin{aligned}
&(p \leftrightarrow q) \rightarrow (s \vee \neg q) \equiv \\
&\neg(p \leftrightarrow q) \vee (s \vee \neg q) \equiv \\
&\neg((p \wedge q) \vee (\neg p \wedge \neg q)) \vee (s \vee \neg q) \equiv \\
&[\neg(p \wedge q) \wedge \neg(\neg p \wedge \neg q))] \vee (s \vee \neg q) \equiv \\
&[(\neg p \vee \neg q) \wedge (p \vee q)] \vee (s \vee \neg q) \equiv \\
&(\neg p \vee \neg q \vee s \vee \neg q) \wedge (p \vee q \vee s \vee \neg q) \equiv \\
&\neg p \vee \neg q \vee s
\end{aligned}$$

The LP representation of this clause is $s :- p, q.$
For f_2:

$$
\begin{aligned}
& s \rightarrow (q \wedge \neg p) \equiv \\
& \neg s \vee (q \wedge \neg p) \equiv \\
& (\neg s \vee q) \wedge (\neg s \vee \neg p)
\end{aligned}
$$

The LP represenation of the first clause is $q :- s$, and the second $:- s, p.$
Thus, the LP representation of the original problem P is

$$
\begin{aligned}
s &:- p, q \\
q &:- s \\
&:- s, p
\end{aligned}
$$