Lecture 4 (Sep 16)

Lecture outline:

- classification of clauses
- propositional resolution rule
- inconsistency
- resolution refutation

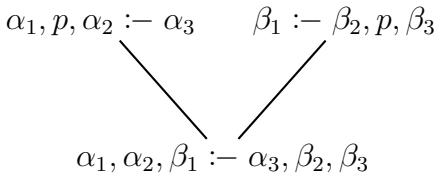
Classification of clauses:

- $s :- p, q, r$ - a *rule*, because its equivalent to $p \wedge q \wedge r \rightarrow s$.
- $s :-$ - a *fact*, because its equivalent to p.
- $:- p, q, r$ - a *goal*, because its equivalent to $\neg(p \wedge q \wedge r)$, and in logic programming we prove goals by contradiction.
- $:-$ - an *empty clause*, or *contradiction*, because its equivalent to F.

The four types of clauses listed above are called *definite* clauses, probably because the implicant is uniquely defined in these clauses. Prolog, and most of other logic programming systems, allows only definite clauses.

Definition 1. *A* logic program *is a finite set of rules and/or facts.*

Definition 2. *The* resolution rule *(propositional case) is an logical inference rule formulated as follows: from two clauses $\alpha_1, p, \alpha_2 :- \alpha_3$ and $\beta_1 :- \beta_2, p, \beta_3$, where α_i and β_i are arbitrary, possibly empty, sequences of literals, derive a clause $\alpha_1, \alpha_2, \beta_1 :- \alpha_3, \beta_2, \beta_3$ - this clause is called a* resolvent *of the first two, the variable p is said to be* resolved upon. *Graphically, we picture an application of the resolution rule as follows:*



Note: resolution rule cannot be applied to two variables at the same time.

Example 3. This shows how we derive contradictions using resolution:

Note that $p :-$ means p, while $:- p$ means $\neg p$, and so indeed p and $\neg p$ give a contradiction.

A few more defintions are required to discuss properties of the resolution rule:

Definition 4. *An* interpretation *(or,* valuation*) is an assignment of truth-values true (1) and false (0) to propositional variables.*

Definition 5. *A set S of clauses is* consistent *if there exists an interpretation that makes all clauses in S true at the same time.*

Definition 6. *A set S is* inconsistent *if it is not consistent (i.e. there is no valuation that makes all clauses in S true at the same time).*

Example 7. The set $S = \{p :- q, s\ ,\ s :-\ ,\ t :- p\}$ is consistent, because the valuation $p = q = s = t = 1$ makes all clauses in S true. Note that even though if $p = s = t = 1$ the value of s is irrelevant, we still need to specify some value for s.

Example 8. The set $S = \{p, q :-\ ,\ p :- q\ ,\ q :- p\ ,\ :- p, q\}$ is inconsistent. One way to check – to go through every possible valuation for p and q, and verify that at least one clause is false for each valuation.

This last method of verifying inconsistency has a problem: the time of such verification procedure would be exponential in the number of variables. A better way: using repeated application of resolution rule.

Definition 9. *Let S be a set of clauses. A* resolution refutation *of S is a sequence of clauses $C_1, C_2, \ldots, C_n$, such that*

1. *C_n is the empty clause :-.*
2. *Every clause C_i is either in S or is obtained from two earlier clauses C_j, C_k $(j, k < i)$ via an application of the resolution rule.*

Example 10. Let, as in the previous example, $S = \{p, q :-\ ,\ p :- q\ ,\ q :- p\ ,\ :- p, q\}$. The resolution refutation of S can be constructed as follows (note that there are many possible ways to do it):

$C_1 = :- p, q$	(from S)
$C_2 = q :- p$	(from S)
$C_3 = :- p$	(resolved C_1 and C_2 on q)
$C_4 = p, q :-$	(from S)
$C_5 = p :- q$	(from S)
$C_6 = p :-$	(resolved C_4 and C_5 on q)
$C_7 = :-$	(resolved C_3 and C_6 on p)

Theorem 11 (Alan Robinson, 1965)**.** *A set S of clauses is inconsistent if and only if there exists a resolution refutation of S (i.e. :- can be derived from S using resolution).*

Note that Theorem 11 works in both directions:

- If $:-$ can be derived from S, then S is inconsistent. Thus, the set of clauses in Example 10 is inconsistent.
- Also, if S is inconsistent, then Theorem 11 guarantees that there exists a resolution refutation of S - unfortunately, it does not say how to find one. In some cases, it will actually be impossible to find it under realistic resource constraints.

So, now let's elaborate on how the logic programming system based on resolution rule (Prolog is such a system) works (see the beginning of Sep 11 lecture notes for the initial picture):

- **Input:** A logic program P (i.e. a set of fact and rule clauses) and a goal clause g.
- **Output:** "yes" if a resolution refutation of the set $P \cup \{\neg g\}$ could be found. 'no" otherwise.

So, given a logic program P and a goal g, a logic programming system *tries to find* a resolution refutation of the set $P \cup \{\neg g\}$. If it succeeds, then by Theorem 11 this set is inconsistent, and therefore (proof by contradiction) $P \rightarrow g$, and the system answers "yes" (meaning, P implies g). The system can fail to find a resolution refutation for the following two reasons:

1. All of the search space has been explored (i.e. all possible resolution sequences have been tried), and no refutation is found. In this case we know that resolution refutation of $P \cup \{\neg g\}$ does not exist, and so, by Theorem 11, the set $P \cup \{\neg g\}$ is consistent, and so $P \nrightarrow g$ (P does not imply g). The system answers "no" in this case.
2. Search space contains infinite resolution sequences, and the system goes into an "infinite loop". This may happen both when the set is consistent, and when its inconsistent.

The situation 2 could be avoided in propositional case via a clever implementation that would record all generated clauses. But, as we will see later, it cannot be avoided in principle in predicate logic case, and so, Prolog doesn't implement such "clever" resolution and could be thrown into an infinite loop even in propositional case.