Lecture 5 (Sep 18)

Lecture outline:

- propositional resolution example
- refutation tree
- linear refutation
- resolution for predicate logic: conversion of predicate logic formulas to CNF

We start with the example of using resolution refutation.

Example 1. Consider a battery-operated car with engine, battery and on-off switch. Define the following propositional variables:

$$\begin{aligned} b_{ok} &\text{ -- true if the battery is charged} \\ e_{ok} &\text{ -- true if the engine is in working condition} \\ s_{on} &\text{ -- true if the switch is on} \\ e_{works} &\text{ -- true if the engine is working (i.e. the car is moving)} \end{aligned}$$

Let's say we have the following problem description

$$\begin{aligned} P = \{&b_{ok} \wedge s_{on} \wedge e_{ok} \rightarrow e_{works}, \\ &b_{ok} \wedge s_{on} \wedge \neg e_{works}\} \end{aligned}$$

and we want to show that $P \rightarrow g$, where $g = \neg e_{ok}$ (that is, the engine is not in working condition). We proceed by contradiction, in other words, we show that the set $P \cup \{\neg g\}$ is inconsistent, using resolution.

First, convert the set $P \cup \{\neg g\}$ to logic programming clauses. Let

$$\begin{aligned} \alpha &= b_{ok} \wedge s_{on} \wedge e_{ok} \rightarrow e_{works}, \\ \beta &= b_{ok} \wedge s_{on} \wedge \neg e_{works}, \\ \gamma &= e_{ok}. \end{aligned}$$

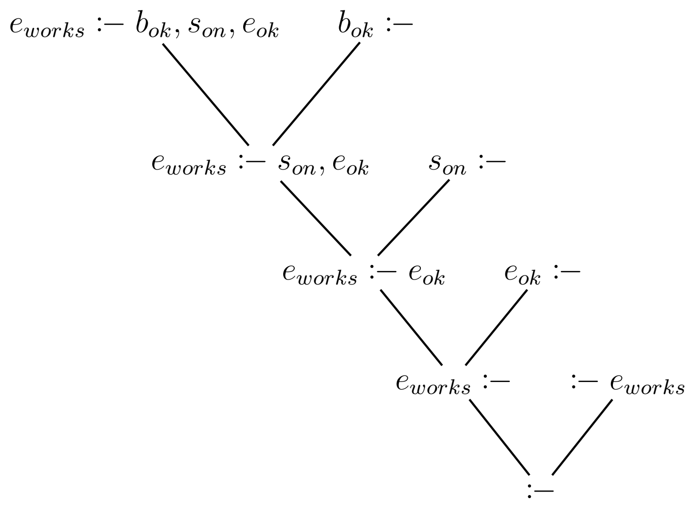
Then, we have the following set of logic programming clauses S:

$$\begin{aligned} e_{works} &:- \; b_{ok}, s_{on}, e_{ok} \\ b_{ok} &:- \\ s_{on} &:- \\ &:- \; e_{works} \\ e_{ok} &:- \end{aligned}$$

If we can construct resolution refutation of S, then, by Theorem 11 from last lecture (Robinson's theorem) S is inconsistent, and so $P \rightarrow g$. One of the possible refutations is as follows:

$$
\begin{aligned}
&C_1 = e_{works} :- b_{ok}, s_{on}, e_{ok} \qquad \text{(from } S\text{)}\\
&C_2 = b_{ok} :- \text{(from } S\text{)}\\
&C_3 = e_{works} :- s_{on}, e_{ok} \qquad \text{(resolved } C_1 \text{ and } C_2 \text{ on } b_{ok}\text{)}\\
&C_4 = s_{on} :- \text{(from } S\text{)}\\
&C_5 = e_{works} :- e_{ok} \qquad \text{(resolved } C_3 \text{ and } C_4 \text{ on } s_{on}\text{)}\\
&C_6 = e_{ok} :- \text{(from } S\text{)}\\
&C_7 = e_{works} :- \qquad \text{(resolved } C_5 \text{ and } C_6 \text{ on } e_{ok}\text{)}\\
&C_8 =:- e_{works} \qquad \text{(from } S\text{)}\\
&C_9 =:- \qquad \text{(resolved } C_7 \text{ and } C_8 \text{ on } e_{works}\text{)}
\end{aligned}
$$

A more convenient way to represent resolution refutations is by means of *refutation tree.* For the previous example, the refutation tree looks as follows:



The formal definition of the refutation tree is a follows:

Definition 2. *A* refutation tree *of a set S of clauses is a binary tree T such that:*

- *the root of T is labeled with $:-$,*
- *the leafs of T are labeled with clauses from S,*
- *if N is a node in T and N_1, N_2 are its children, then N is the resolvent of N_1 and N_2.*

Note that the resolvent of a goal clause with a fact or a rule clause is *always* a goal clause:

$$\begin{tikzcd}
:- l_1, \dots, l_k, \dots l_n \arrow[dr, dash] & & l_k :- \alpha \arrow[dl, dash] \\
& :- l_1, \dots, l_{k-1}, l_{k+1}, \dots l_n, \alpha &
\end{tikzcd}$$

This observation is important: remember, a logic program is a set of fact and rule clauses, and in logic programming there's exactly one goal clause. Thus, we can always start the refutation with the goal clause, pick one of the fact or rule clauses from the logic program, and as a result obtain another goal clause. This next goal clause can again be resolved with another clause from the program. This way at least on of the clauses for the resolution is determined in advance, and so the search space is significantly reduced (although can still be exponential). This special type of resolution refutation is called *linear refutation.*

Definition 3. *A* linear refutation *of a set of program (i.e. facts and rules) clauses S with a goal clause g us a resolution refutation which has the following form:*

$$\begin{tikzcd}
g \arrow[dr, dash] & & cl_1 \in S \arrow[dl, dash] & & \\
& g_1 \arrow[dr, dash] & & cl_2 \in S \arrow[dl, dash] & \\
& & g_2 \arrow[dr, dash, dotted] & & cl_i \in S \arrow[dl, dash] \\
& & & :- &
\end{tikzcd}$$

The following theorem basically states that in logic programming setting linear resolution refutations can be used instead of the "regular" resolution refutations:

Theorem 4. *Let P be a logic program (i.e. a finite set of fact and rule clauses), and let g be a goal clause. Then the set $P \cup \{g\}$ is inconsistent if and only if there exists a* linear *resolution refutation of this set.*

Resolution for Predicate Logic

Just like in propositional case, resolution in predicate logic operates on formulas in conjunctive normal form (CNF). The transformation of predicate logic formulas into CNF is done in the following three steps:

Step 1. Move all quantifiers "outside" – the resulting formula is a formula in *prenex normal form*, that is

$$Q_1 X_1 \dots Q_n X_n \ \alpha,$$

where $Q_i \in \{\forall, \exists\}$, and α is quantifier-free (i.e. no quantifiers inside it). To perform this transformation, first get rid of all $\leftrightarrow$ and $\rightarrow$ (just like during conversion to CNF in propositional case), and then use the following equivalences until all quantifiers are pushed outside:

$$\begin{aligned} \neg\forall X\alpha &\equiv \exists X\neg\alpha \\ \neg\exists X\alpha &\equiv \forall X\neg\alpha \\ (\forall X\alpha) \vee \beta &\equiv \forall X(\alpha \vee \beta) \\ (\exists X\alpha) \vee \beta &\equiv \exists X(\alpha \vee \beta) \\ (\forall X\alpha) \wedge \beta &\equiv \forall X(\alpha \wedge \beta) \\ (\exists X\alpha) \wedge \beta &\equiv \exists X(\alpha \wedge \beta) \end{aligned}$$

Important: the last four equivalences hold only if X does not occur in β. If it does – replace X in α and the quantifier with a *fresh* variable. For example,

$$\begin{aligned} &(\forall X p(X)) \vee q(X) \equiv \\ &\forall Y(p(Y) \vee q(X)) \end{aligned}$$

Step 2. Eliminate existential quantifiers via *skolemization*. Skolemization is a repeated application of the following rule:

Definition 5 (skolemization rule)**.** *Replace the formula*

$$\forall X_1 \ldots \forall X_n \exists Y \alpha,$$

where all X_i are quantified universally, and α is in prenex normal form, with the formula

$$\forall X_1 \ldots \forall X_n \alpha_S,$$

where α_S is obtained from α by replacing every occurrence of Y with a fresh function symbol $f(X_1, \ldots, X_n)$.

Note that the rule works for $n = 0$ – in this case the variable is replaced with a fresh constant symbol (function of 0 arguments is constant). For example, the formula $\exists X\forall Y p(X,Y)$ would be replaced with $\forall Y p(c, Y)$. Also, note that there are no restrictions on α, except that its in prenex normal form, and so to eliminate multiple existential quantifiers we eliminate them one-by-one *starting from the outside.* For example,

Example 6.

$$\forall X\forall Y\exists Z\forall T\exists M(p(X,Y,Z) \wedge q(Y,M)).$$

We start with Z. Since the universally quantified variables in front of Z are X and Y, we will replace Z with a function symbol $f(X,Y)$. This gives us

$$\forall X\forall Y\forall T\exists M(p(X,Y,f(X,Y)) \wedge q(Y,M)).$$

So, now we take care of M. Again, the universally quantified variables in front of M are X, Y, T, and so we replace M with a function symbol $h(X, Y, T)$. Thus, we have

$$\forall X \forall Y \forall T (p(X, Y, f(X, Y)) \wedge q(Y, h(X, Y, T))).$$

As a result of skolemization we obtain a formula in prenex normal form in which all quantifiers are universal. Note that the resulting formula is *not* equivalent to the original one (we talked in the class as to why this is so). However the resulting formula is consistent if and only if the original one is consistent, and since in logic programming we only care about consistency (or, inconsistency), for logic programming skolemization is appropriate.