Lecture 3
Knowledge representation using logic

Reading

Winston (3rd edition) Chapter 13

Background

We stressed in Lecture 1 that knowledge requires representation, and that handling knowledge requires a reasoning process. In this lecture, we introduce predicate calculus as a representing facts, and develop resolution as a method of reasoning with those facts. We shall see that logical reasoning, far from being effortless, requires search.

There are a variety of logics or calculi. Common ones are

- Propositional logic, where every fact has a separate utterance, as in Bill-plays-Saxophone
- First-order predicate calculus, where predicates (the verb-like things) can have arguments, as in plays(Bill,Saxophone),
- Second-order predicate calculus, where predicates can be variables.

Here we consider only first-order predicate calculus. This is a monotonic logic, which indicates that as you add extra axioms to the database, conclusions you drew from the previous database should not change. (In everyday life, we often use non-monotonic reasoning — this is an area of study in its own right.)

3.1 Logical utterances

A set of sentences like
animal(elephant)
has-trunk(elephant)
has-tusks(elephant)
ishuge(elephant)
has-trunk(Nellie)
∀x[has-trunk(x) ⇒ elephant(x)]

form a knowledge database. These facts asserted a priori comprise our axioms.

We can relate logical utterances by connectives:

\[
\begin{array}{c}
\wedge \quad \text{AND} \\
\vee \quad \text{OR} \\
\neg \quad \text{NOT} \\
\Rightarrow \quad \text{implies} \\
\iff \quad \text{mutual implication}
\end{array}
\]

The following are the truth tables for the connectives:

<table>
<thead>
<tr>
<th>Expression</th>
<th>Truth Table</th>
</tr>
</thead>
</table>
| \( E \land F \) | \[
| E \downarrow F | \quad \text{True} \quad \text{False} \\
| \text{True} | \text{True} \quad \text{False} \\
| \text{False} | \text{False} \quad \text{False} \\
| \] |
| \( \neg E \) | \[
| E \downarrow | \quad \text{False} \\
| \text{True} | \quad \text{True} \\
| \text{False} | \quad \text{True} \\
| \] |
| \( E \lor F \) | \[
| E \downarrow F | \quad \text{True} \quad \text{False} \\
| \text{True} | \quad \text{True} \quad \text{True} \\
| \text{False} | \quad \text{True} \quad \text{False} \\
| \] |
| \( E \Rightarrow F \) | \[
| E \downarrow F | \quad \text{True} \quad \text{False} \\
| \text{True} | \quad \text{True} \quad \text{False} \\
| \text{False} | \quad \text{True} \quad \text{True} \\
| \] |

Note that looking at the last table

\[ E \Rightarrow F \iff \neg E \lor F . \]
3.1. LOGICAL UTTERANCES

We also see that ‘False ⇒ True’ is True assertion. Another way of looking at this is to recall that abduction is not a sound rule of inference. The statement

\[ \text{has-drank-too-much}(\text{Eric}) \Rightarrow \text{has-headache}(\text{Eric}) \]

does not indicate that if Eric has a headache, that Eric has drunk too much — perhaps we shouldn’t be too surprised if it was because he had spent the evening banging his head on his desk in response to a Finals question. So the rhs being true doesn’t say anything about the lhs, so F ⇒ T must be true.

The following commutative, distributive, associative rules apply, as do De Morgan’s theorems.

| Commutative | \( E \land F \Leftrightarrow F \land E \)  \\ |  | \( E \lor F \Leftrightarrow F \lor E \) |
|-------------|----------------------------------|
| Distributive | \( E \land (F \lor G) \Leftrightarrow (E \land F) \lor (E \land G) \)  \\ |  | \( E \lor (F \land G) \Leftrightarrow (E \lor F) \land (E \lor G) \) |
| Associative | \( E \land (F \land G) \Leftrightarrow (E \land F) \land G \)  \\ |  | \( E \lor (F \land G) \Leftrightarrow (E \lor F) \lor G \) |
| De Morgan’s | \( \neg(E \land F) \Leftrightarrow \neg E \lor \neg F \)  \\ |  | \( \neg(E \lor F) \Leftrightarrow \neg E \land \neg F \) |
| Negation | \( \neg(\neg E) \Leftrightarrow E \) |

### 3.1.1 Variables and quantification

First-order predicate calculus allows variable arguments. These are quantified in two ways, existentially and universally. The universal one

\[ \forall x \ [\text{Expression}(x)] \]

says that for all x, the Expression is true. For example,

\[ \forall x \ [\text{elephant}(x) \Rightarrow \text{has-tusks}(x)]. \]

The expression in [ ] is in the scope of the x.

Existential quantification appears like
∃x [Expression(x)]

and indicates that there is at least one x that make Expression(x) true.

The negations of the universal and existential quantifications warrant a careful look:

¬∀x P(x) ⇔ ∃x ¬P(x); and
¬∃x P(x) ⇔ ∀x ¬P(x).

1. **Predicates** are verb-like: adored(), boy() etc.
2. **Terms** are the only things that can appear as the arguments to predicates: predicate(term,term,...). Terms come in three flavours:
   a. Objects: these are specific – Marilyn-Monroe, BigBen
   b. Variables: x
   c. Functions: function() must return a term
3. **Atomic formulae** are the predicates and arguments, eg adored(Marilyn-Monroe,x)
4. **Literals** are atomic formulae and their negations, eg, ¬huge(BigBen).
5. **Well-formed formulae** are defined recursively. Literals are wffs, wffs connected by connectives are wffs, wffs surrounded by quantifiers are wffs.
6. **Sentences** are wffs in which all the variables (if any) are inside the scope of the corresponding quantifiers: eg,

   ∀x[has-trunk(x) ⇒ elephant(x)]
   has-trunk(Nellie) ⇒ elephant(Nellie)

are both sentences.
7. **Bound variables** are those that appear within the scope of a quantifier, otherwise they are **free variables**.

### 3.2 Logical proofs

The sentences a priori, and which form our initial knowledge base, eg

has-trunk(Nellie)
∀x[has-trunk(x) ⇒ elephant(x)]

are our **axioms**. Suppose that we are able to show that all interpretations that make the axioms true also make

elephant(Nellie)

true, then we have proved elephant(Nellie) is a theorem w.r.t. the axioms. Note that if we are given the further axiom that
\[ \forall x [\text{student}(x) \Rightarrow \text{has-trunk}(x)] \]
	hen we can prove that

\[ \forall x [\text{student}(x) \Rightarrow \text{elephant}(x)] \]

That is, all students are elephants. There is nothing wrong with our proof — it follows logically from the axioms. Nor logically are the axioms wrong — axioms are just axiomatic, and can't help if they are nonsense in the worldly sense.

### 3.2.1 Sound rules of inference

Earlier we saw that that has-drunk-to-much(Eric) \(\Rightarrow\) has-headache(Eric) does not mean that if Eric has a headache then he has drunk to much. Abduction, as this is called is not a sound rule of inference.

Looking at the truth tables you will easily see that the sound rules are:

**Modus ponens.** If there is an axiom \( E \Rightarrow F \) and an axiom \( E \), then \( F \) logically follows.

**Modus tolens.** If there is an axiom \( E \Rightarrow F \) and an axiom \( \neg F \), then \( \neg E \) follows logically.
Resolution. If there is an axiom $E \lor F$ and an axiom $\neg F \lor G$ then $E \lor G$ follows logically.

As we shall see, resolution can subsume both modus ponens and modus tollens. It can also be generalized so that there can be any number of disjuncts in either of the two resolving expressions, including just one. (Note, disjunct are expressions connected by $\lor$, conjuncts are those connected by $\land$.) The only requirement is that one expression contains the negation of one disjunct from the other. For example:

$$\text{has-trunk(Nellie)} \quad \text{(NB 1 disjunct)}$$
$$\text{elephant(Nellie)} \lor \neg \text{has-trunk(Nellie)} \lor \text{student(Nellie)}$$
leads to
$$\text{elephant(Nellie)} \lor \text{student(Nellie)}$$

We will return to look at resolution in more detail but first try to answer the question “Is Marcus alive?” using the following knowledge base.

$$\begin{array}{l}
1 \quad \text{man(Marcus)} \\
2 \quad \text{pompeian(Marcus)} \\
3 \quad \text{born(Marcus,40)} \\
4 \quad \forall x [\text{man}(x) \Rightarrow \text{mortal}(x)] \\
5 \quad \forall x [\text{pompeian}(x) \Rightarrow \text{died}(x,79)] \\
6 \quad \text{erupted(Vesuvius,79)} \\
7 \quad \forall x \forall t_1 \forall t_2 [\text{mortal}(x) \land \text{born}(x,t_1) \land \text{gt}(t_2-t_1,130) \Rightarrow \text{dead}(x,t_2)] \\
8 \quad \forall x \forall t [\neg \text{alive}(x,t) \Rightarrow \text{dead}(x,t)] \\
9 \quad \forall x \forall t_1 \forall t_2 [\text{died}(x,t_1) \land \text{gt}(x,t_2 \land \text{gt}(t_2-t_1,130) \Rightarrow \text{dead}(x,t_2)] \\
10 \quad \text{Now=2000}
\end{array}$$

One Solution. Let us try to prove/disprove “alive(Marcus,Now)”.

<table>
<thead>
<tr>
<th>Step</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>Modus Ponens $\neg \text{dead(Marcus,Now)}$</td>
</tr>
<tr>
<td>9</td>
<td>Modus Tolens $\neg \text{mortal(Marcus)} \lor \neg \text{born(Marcus,t_1)} \lor \neg \text{gt(Now-t_1,130)}$</td>
</tr>
<tr>
<td>3</td>
<td>Resolution $\neg \text{mortal(Marcus)} \lor \neg \text{gt(Now-40,130)}$</td>
</tr>
<tr>
<td>4</td>
<td>Modus Tolens $\neg \text{mortal(Marcus)} \lor \neg \text{gt(Now-40,130)}$</td>
</tr>
<tr>
<td>1</td>
<td>Resolution $\neg \text{gt(2000,79)}$</td>
</tr>
<tr>
<td></td>
<td>FALSE</td>
</tr>
</tbody>
</table>

Two points arise:

- A variety of processes contributed to the proof: matching, substitution, and a haphazard choice of the sound rules of inference.
- Even simple questions require a considerable number of steps.

What we seem to require is a single, well specified reasoning process. One way of achieving this is using resolution.
3.3 Resolution Proofs

Resolution produces an assertion by

- converting the assertion and database axioms into a canonical form, the *Conjunctive Normal Form*; then
- using refutation — ie, by attempting to show that the negated assertion produces a contradiction with known axioms in the database.

### 3.3.1 Conversion to Conjunctive Normal Form

We now examine the recipe to reach the conjunctive normal form needed in resolution. It is rather long, but not difficult to follow.

Suppose we assert that

“All musiclovers who enjoy Bach either dislike Wagner or think that anyone who dislikes any composer is a philistine”.

We shall use enjoy() for enjoying a composer, and similarly for dislike, giving

\[ \forall x [ \text{musiclover}(x) \land \text{enjoy}(x,\text{Bach}) \Rightarrow \text{dislike}(x,\text{Wagner}) \lor (\forall y [\exists z [\text{dislike}(y,z)] \Rightarrow \text{think-philistine}(x,y)])] \]

1. Recall that \( E \Rightarrow F \iff \neg E \lor F \), and thereby filter the expression to remove \( \Rightarrow \) symbols.

\[ \forall x [ \neg (\text{musiclover}(x) \land \text{enjoy}(x,\text{Bach})) \lor \text{dislike}(x,\text{Wagner}) \lor (\forall y [\neg \exists z [\text{dislike}(y,z)] \lor \text{think-philistine}(x,y)])] \]

2. Filter to obtain negations just in front of the predicates, using the following relationships:

\[-( \neg P) \iff P; \]
\[-( a \land b ) \iff \neg a \lor \neg b; \]
\[-( a \lor b ) \iff \neg a \land \neg b; \]
\[-\forall x P(x) \iff \exists x \neg P(x); \text{ and } \]
\[-\exists x P(x) \iff \forall x \neg P(x). \]

Our expression becomes

\[ \forall x [ \neg \text{musiclover}(x) \lor \neg \text{enjoy}(x,\text{Bach}) \lor \text{dislike}(x,\text{Wagner}) \lor \forall y [\neg [\forall z [\neg \text{dislike}(y,z)] \lor \text{think-philistine}(x,y)])]] \]

3. Standardize variables so that each quantifier binds a unique variable. This is already the case in our expression, but the following is an example.
∀x Pred1(x) ∨ ∀x Pred2(x)
becomes
∀x Pred1(x) ∨ ∀y Pred2(y).

4. Step 3 allows us to move all the quantifiers to the left in Step 4. Our expression becomes

∀x∀y∀z [¬musiclover(x) ∨ ¬enjoy(x,Bach) ∨
dislike(x,Wagner) ∨ ¬dislike(y,z) ∨ think-philistine(x,y)].

This is called the prenex normal form.

5. This step will seem a bit of a fiddle. We eliminate existential quantifiers, by arguing that we could find the instance as an object. For example, if

∃y dislike(y,Wagner)

we could actually find that person-Object S1 to replace the Variable y. So this gets replaced simply by

dislike(S1,Wagner).

Now, if existential quantifiers exist within the scope of universal quantifiers, we can’t use merely an object, but rather a function that returns an object. The function will depend on the universal quantifier. For example, every piece of music has a composer

∀x∃y composer-of(y,x)

So for the piece of music x, we find the composer S2(x). Thus the expression is replaced by

∀x composer-of(S2(x),x).

This process is called Skolemization. S1 is a Skolem object and the S2 is a Skolem function.

6. This step is merely to save writing. Any variable left must be universally quantified out on the left, so don’t bother writing the quantifier. Our expression becomes:

¬musiclover(x) ∨ ¬enjoy(x,Bach) ∨
dislike(x,Wagner) ∨ ¬dislike(y,z) ∨ think-philistine(x,y).

7. Convert everything into a conjunction of disjunctions using the associative, commutative, distributive laws. The form you want is like:

(A(x) ∨ B(x,y) ∨ C(y) ∨ ...) ∧ (P(x,y) ∨ Q(x,y,z) ∨ ...) ∧ ...

Call each conjunct a separate clause. In order for the entire wff to be true, each clause must be true separately.

C1: (A(x) ∨ B(x,y) ∨ C(y) ∨ ...)
C2: (P(x,y) ∨ Q(x,y,z) ∨ ...)
C3: ...
8. Standardize apart the variables in the set of clauses generated in 7. This requires renaming the variables so that no two clauses make reference to the same variable. Remember that all variables are implicitly universally quantified to the left, so that
\[
\forall x P(x) \land Q(x) \to \forall x P(x) \land \forall x Q(x) \iff \forall x P(x) \land \forall y Q(y) \to P(x) \land Q(y)
\]
One way of ensuring this is to attach the clause number as a subscript:

Clause1: \( A(x_1) \lor B(x_1,y_1) \lor C(y_1) \lor \ldots \)
Clause2: \( P(x_2,y_2) \lor Q(x_2,y_2,z_2) \lor \ldots \)
Clause3: \( R(z_3,x_3) \lor \neg Q(S_1(z_3),y_3,z_4) \)

(NB it does not matter at all that there is no \( z_1 \).)

This completes the recipe. After application to a set of wffs, we end up with a set of clauses each of which is a disjunction of literals.

The Resolution proof procedure is as follows:

1. Negate the theorem to be proved
2. Turn the theorem and the axioms into clause form.
3. Until the empty clause is produced or there are no resolvable clauses, find pairs of resolvable clauses, resolve them (including Unification), and add them to the list of clauses.
4. If the empty clause was produced, the negated theorem contradicted the axioms, and the (unnegated) theorem is TRUE, w.r.t. the axioms. If there were no more resolvable clauses, the theorem is FALSE w.r.t. the axioms.

### 3.3.2 Pairwise Resolution and Unification

To perform resolution we need to spot pairs of clauses involving unnegated/negated pairs of predicates like \( P() \) and \( \neg P() \). However, once spotted, before we can resolve the clauses we need look carefully at what the arguments of \( P() \) and \( \neg P() \) are to ensure that the statements are really contradictory. For example, consider

\[
\text{elephant(Nelly)} \text{ and } \neg \text{elephant(Nelly)}.
\]

These are obviously contradictory, and can be resolved. The following are obviously not contradictory

\[
\text{genius(J.S.Bach)} \text{ and } \neg \text{genius(C.P.E.Bach)}
\]

and so cannot be resolved. Finally
cool(x) and ¬cool(Flared-trousers)

are contradictory, and allow resolution, provided we let x be Flared-trousers henceforth.

What is required is a method for checking terms and substituting for the variables. This match and substitute process is called *unification*.

**A Unification Algorithm**

The resolver finds the two clauses with unnegated and negated predicate P and ¬P, but these may have different arguments sets:

\[ P(\text{argset}1) \text{ and } ¬P(\text{argset}2) \]

The unifier determines whether and how \( \text{argset}1 \) and \( \text{argset}2 \) are unifiable.

We start by initializing an empty unification set \( \Phi = \{ \} \). Then work left to right through corresponding terms of the argsets, until there is a difference in the terms. If neither of the terms is a variable, give up. Otherwise convert variables into terms by trying to substitute \( v_p \) for \( t_p \) (often written \( t_p/v_p \)). At any stage in the unification, to add \( (t_p/v_p) \) to the unification set, we must check that the variable does not already appear in the variable position in the set. That is, if \( \Phi = \{(t_1/v_1, \ldots, t_k/v_k)\} \) we check that \( v_p \neq v_i, i = 1, \ldots, k \). Finally, before adding \( (t_p/v_p) \) to \( \Phi \) we apply the substitution to \( \Phi \) itself.

Remember that terms can be Objects, variables or functions, so that a variable can be replaced by

- An Object: \( \text{Object}/x \)
- A Variable: \( y/x \)
- A function: \( g(\ldots)/x \) provided the arguments of \( g \) do not involve \( x \)

So trying to unify \( C1 \) and \( C2 \) from

\[
\begin{align*}
C1: & \ P(f(A),x1) \lor Q(x1) \\
C2: & \ ¬P(y2,g(B)) \lor R(g(B),y2) \\
C3: & \ ¬P(g(A),x) \lor R(x)
\end{align*}
\]

we would grow from \( \Phi = \{ \} \) to \( \Phi = \{((f(A)/y2), (g(B)/x1))\} \) so that the resolved clause would be

\[ C5: Q(g(B)) \lor R(g(B),f(A)) \]

whereas \( C1 \) and \( C3 \) are not unifiable, because neither of \( f(A) \) and \( g(A) \) is a variable.
### 3.3. Example proof by resolution

Suppose our database contains the following statements.

<table>
<thead>
<tr>
<th>No</th>
<th>Sentence</th>
</tr>
</thead>
<tbody>
<tr>
<td>S1</td>
<td>man(Marcus)</td>
</tr>
<tr>
<td>S2</td>
<td>pompeian(Marcus)</td>
</tr>
<tr>
<td>S3</td>
<td>∀x[pompeian(x) ⇒ roman(x)]</td>
</tr>
<tr>
<td>S4</td>
<td>ruler(Caesar)</td>
</tr>
<tr>
<td>S5</td>
<td>∀x[roman(x) ⇒ loyalto(x2,Caesar) ∨ hate(x,Caesar)]</td>
</tr>
<tr>
<td>S6</td>
<td>∀x[∃loyalto(x,y)]</td>
</tr>
<tr>
<td>S7</td>
<td>∀∀y [man(x)∧ ruler(y) ∧ trykill(x,y) ⇒ ¬loyalto(x,y)]</td>
</tr>
<tr>
<td>S8</td>
<td>trykill(Marcus,Caesar)</td>
</tr>
<tr>
<td>S9</td>
<td>∀x [man(x) ∨ woman(x) ⇒ mortal(x)]</td>
</tr>
</tbody>
</table>

The are converted to conjunctive normal form as

<table>
<thead>
<tr>
<th>No</th>
<th>Clause</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>C1</td>
<td>man(Marcus)</td>
<td></td>
</tr>
<tr>
<td>C2</td>
<td>pompeian(Marcus)</td>
<td></td>
</tr>
<tr>
<td>C3</td>
<td>¬pompeian(x3) ∨ roman(x3)</td>
<td></td>
</tr>
<tr>
<td>C4</td>
<td>ruler(Caesar)</td>
<td></td>
</tr>
<tr>
<td>C5</td>
<td>¬roman(x5) ∨ loyalto(x5,Caesar) ∨ hate(x5,Caesar)</td>
<td></td>
</tr>
<tr>
<td>C6</td>
<td>loyalto(x6,S1(x6))</td>
<td></td>
</tr>
<tr>
<td>C7</td>
<td>¬man(x7) ∨ ¬ruler(y7) ∨ ¬trykill(x7,y7) ∨ ¬loyalto(x7,y7)</td>
<td></td>
</tr>
<tr>
<td>C8</td>
<td>trykill(Marcus,Caesar)</td>
<td></td>
</tr>
<tr>
<td>C9</td>
<td>¬man(x9) ∨ mortal(x9)</td>
<td></td>
</tr>
<tr>
<td>C10</td>
<td>¬woman(x10) ∨ mortal(x10)</td>
<td></td>
</tr>
</tbody>
</table>

Suppose we want to prove that Marcus hates Caesar. We start with the negation and proceeds like:
3.4 * Horn clauses and Prolog (an aside)

Proof by resolution involved finding the conjunctive normal form, which comprised conjoined clauses, each of which was a disjunction of literals — ie,

\[ L_1 \lor L_2 \lor L_3 \lor ... \]

A slightly different form is the “Horn Clause”, where the restriction is introduced that only one of the literals is un-negated. For example, the following are valid Horn clauses:

1. \( \neg L_1 \lor \neg L_2 \lor \neg L_3 \lor ... \)
2. \( \neg L_1 \lor \neg L_2 \lor \neg L_3 \lor ... \lor M \)
3. M
4. \( \neg L_1 \)

It is straightforward to show that, for example, (2) is equivalent to

\[ L_1 \land L_2 \land L_3 \lor ... \Rightarrow M \]

or in “goal-directed” form:

\[ M \Leftarrow L_1 \land L_2 \land L_3 \lor ... \]

This is the typical form of a statement in the language for logic programming, Prolog.

There are two modes in Prolog. In the consult mode, one supplies the system with axioms. (Note that the “\( \Leftarrow \)” is replaced by “\( :- \)”, that capitalized symbols are variables and that the “.” is important.)
grandson(X,Y) :- son(X,Z),parent(Y,Z).
son(charles,elizabeth).
parent(george,elizabeth).

In query mode we might ask

?- grandson(elizabeth,charles).
no

with Prolog telling us that there are no database facts to verify the statement. Or we might ask

?- grandson(W,george).

Starting with this query, Prolog tries to justify each literal on the RHS by finding a matching literal in the LHS of another clause in the database. In our example, it would find dbase statement 1 and perform the unification

\{ W/X, george/Y \}.

So the RHS becomes

son(X,Z),parent(george,Z).

It then uses dbase statement 2 to perform the unification

(X/charles, Z/elizabeth),

leaving

parent(george,elizabeth)

to be verified. Obviously, dbase statement 3 does this with no unification. The system returns the overall unification for W which is (W/X/charles), ie (W/charles), as in

?- grandson(W,george).
W = charles ?
yes
3.5 Summary

In this lecture we have introduced first-order predicate logic, and explained proof by resolution as a method of sound logical inference. We outlined unification.

Logic might seem the answer to everything, but there are some difficulties.

- How in the proof were we clever enough to pick up the right clauses. Certainly by matching \( P, \neg P \) pairs, but also by using our intuition to find a solution. If we have to use intuition, the machine has to search.

- There are limits to pure logic, but if we introduce extras, our simple-minded rules won’t always work.

- Machines are going to need search strategies for looking into the database. In turn these searches might suffer from:
  - Combinatorial explosion
  - The halting problem — search is not guaranteed to terminate.