Module 6

Knowledge Representation and Logic – (First Order Logic)

Lesson 15

Inference in FOL - I

6.2.8 Resolution

We have introduced the inference rule Modus Ponens. Now we introduce another inference rule that is particularly significant, Resolution.

Since it is not trivial to understand, we proceed in two steps. First we introduce Resolution in the Propositional Calculus, that is, in a language with only truth valued variables. Then we generalize to First Order Logic.

6.2.8.1 Resolution in the Propositional Calculus

In its simplest form Resolution is the inference rule:

```
                   {A OR C, B OR (NOT C)}
                   ----------------------
                           A OR B
```

More in general the **Resolution Inference Rule** is:

- Given as premises the clauses C1 and C2, where C1 contains the literal L and C2 contains the literal (NOT L), infer the clause C, called the **Resolvent** of C1 and C2, where C is the union of (C1 - {L}) and (C2 -{(NOT L)})

In symbols:

```
                  {C1, C2}
        ---------------------------------
        (C1 - {L}) UNION (C2 - {(NOT L)})
```

Example:

The following set of clauses is inconsistent:

```
1. (P OR (NOT Q))
2. ((NOT P) OR (NOT S))
3. (S OR (NOT Q))
4. Q
```

In fact:

```
5. ((NOT Q) OR (NOT S))              from 1. and 2.
6. (NOT Q)                            from 3. and 5.
7. FALSE                              from 4. and 6.
```

Notice that 7. is really the empty clause [why?].

Theorem: The Propositional Calculus with the Resolution Inference Rule is sound and Refutation Complete.

NOTE: This theorem requires that clauses be represented as sets, that is, that each element of the clause appear exactly once in the clause. This requires some form of membership test when elements are added to a clause.

C1 = {P P}

C2 = {(NOT P) (NOT P)}
C = {P (NOT P)}

From now on by resolution we just get again C1, or C2, or C.

6.2.8.2 Resolution in First Order Logic

Given clauses C1 and C2, a clause C is a **RESOLVENT of C1 and C2**, if

1. There is a subset C1' = {A1, .., Am} of C1 of literals of the same sign, say positive, and a subset C2' = {B1, .., Bn} of C2 of literals of the opposite sign, say negative,
2. There are substitutions s1 and s2 that replace variables in C1' and C2' so as to have new variables,
3. C2" is obtained from C2 removing the negative signs from B1 .. Bn
4. There is an Most General Unifier s for the union of C1'.s1 and C2".s2

and C is
((C1 - C1').s1 UNION (C2 - C2').s2).s

In symbols this **Resolution** inference rule becomes:

```
{C1, C2}
--------
   C
```

If C1' and C2' are singletons (i.e. contain just one literal), the rule is called **Binary Resolution**.

Example:
C1 = {(P z (F z)) (P z A)}
C2 = {(NOT (P z A)) (NOT (P z x)) (NOT (P x z))
C1' = {(P z A)}
C2' = {(NOT (P z A)) (NOT (P z x))}
C2" = {(P z A) (P z x)}
s1 = [z1/z]
s2 = [z2/z]
C1'.s1 UNION C2'.s2 = {(P z1 A) (P z2 A) (P z2 x)}
s = [z1/z2 A/x]
C = {(NOT (P A z1)) (P z1 (F z1))}

Notice that this application of Resolution has eliminated more than one literal from C2, i.e. it is not a binary resolution.

Theorem: First Order Logic, with the Resolution Inference Rule, is sound and refutation complete.

We will not develop the proof of this theorem. We will instead look at some of its steps, which will give us a wonderful opportunity to revisit Herbrand. But before that let's observe that in a sense, if we replace in this theorem "Resolution" by "Binary Resolution", then the theorem does not hold and Binary Resolution is not Refutation Complete. This is the case when in the implementation we do not use sets but instead use bags.This can be shown using the same example as in the case of propositional logic.

Given a clause C, a subset D of C, and a substitution s that unifies D, we define C.s to be a **Factor** of C.
The **Factoring Inference Rule** is the rule with premise C and as consequence C.s.

Theorem: For any set of clauses S and clause C, if C is derivable from S using Resolution, then C is derivable from S using Binary Resolution and Factoring.

When doing proofs it is efficient to have as few clauses as possible. The following definition and rule are helpful in eliminating redundant clauses:

A clause C1 **Subsumes** a clause C2 iff there is a substitution s such that C1.s is a subset of C2.

Subsumption Elimination Rule: If C1 subsumes C2 then eliminate C2.

Herbrand Revisited

We have presented the concept of Herbrand Universe H_S for a set of clauses S. Here we meet the concept of **Herbrand Base**, H(S), for a set of clauses S.
H(S) is obtained from S by considering the ground instances of the clauses of S under all the substitutions that map all the variables of S into elements of the Herbrand universe of S. Clearly, if in S occurs some variable and the Herbrand universe of S is infinite then H(S) is infinite.
[NOTE: Viceversa, if S has no variables, or S has variables and possibly individual constants, but no other function symbol, then H(S) is finite. If H(S) is finite then we can, as we will see, decide if S is or not satisfiable.]
[NOTE: it is easy to determine if a finite subset of H(S) is satisfiable: since it consists of ground clauses, the truth table method works now as in propositional cases.]

The importance of the concepts of Herbrand Universe and of Herbrand Base is due to the following theorems:

Herbrand Theorem: If a set S of clauses is unsatisfiable then there is a finite subset of H(S) that is also unsatisfiable.

Because of the theorem, when H(S) is finite we will be able to decide is S is or not satisfiable. Herbrand theorem immediately suggests a general refutation complete proof procedure:

given a set of clauses S, enumerate subsets of H(S) until you find one that is unsatisfiable.

But, as we shall soon see, we can come up with a better refutation complete proof procedure.

Refutation Completeness of the Resolution Proof Procedure

Given a set of clauses S, the **Resolution Closure** of S, R(S), is the smallest set of clauses that contains S and is closed under Resolution. In other words, it is the set of clauses obtained from S by applying repeatedly resolution.

Ground Resolution Theorem: If S is an unsatisfiable set of ground clauses, then R(S) contains the clause FALSE.

In other words, there is a resolution deduction of FALSE from S.

Lifting Lemma: Given clauses C1 and C2 that have no variables in common, and ground instances C1' and C2', respectively, of C1 and C2, if C' is a resolvent of C1' and C2', then there is a clause C which is a resolvent of C1 and C2 which has C' as a ground instance

With this we have our result, that the Resolution Proof procedure is Refutation Complete. Note the crucial role played by the Herbrand Universe and Basis. Unsatisfiability of S is reduced to unsatisfiability for a finite subset $H_S(S)$ of H(S), which in turn is reduced to the problem of finding a resolution derivation for FALSE in $H_S(S)$, derivation which can be "lifted" to a resolution proof of FALSE from S.

Dealing with Equality

Up to now we have not dealt with equality, that is, the ability to recognize terms as being equivalent (i.e. always denoting the same individual) on the basis of some equational information. For example, given the information that

S(x) = x+1

then we can unify:

F(S(x) y) and F(x+1, 3).

There are two basic approaches to dealing with this problem.

- The first is to add inference rules to help us replace terms by equal terms. One such rule is the **Demodulation Rule**: Given terms t1, t2, and t3 where t1 and t2 are unifiable with MGU s, and t2 occurs in a formula A, then

```
{t1 = t3, A(... t2 ...)}
------------------------
      A(... t3.s ...)
```

  Another more complex, and useful, rule is **Paramodulation**.

- The second approach is not to add inference rules and instead to add non-logical axioms that characterize equality and its effect for each non logical symbol. We first establish the reflexive, symmetric and transitive properties of "=":

  ```
          x=x
          x=y IMPLIES y=x
          x=y AND y=z IMPLIES x=z
  ```

  Then for each unary function symbol F we add the equality axiom

  ```
    x=y IMPLIES F(x)=F(y)
  ```

  Then for each binary function symbol F we add the equality axiom

  ```
    x=z AND y=w IMPLIES F(x y)=F(z w)
  ```

  And similarly for all other function symbols.

  The treatment of predicate symbols is similar. For example, for the binary predicate symbol P we add

  ```
    x=z AND y=w IMPLIES ( P(x y) IFF P(z w))
  ```

Whatever approach is chosen, equality substantially complicates proofs.

Answering True/False Questions

If we want to show that a clause C is derivable from a set of clauses S={C1 C2 .. Cn}, we add to S the clauses obtained by negating C, and apply resolution to the resulting set S' of clauses until we get the clause FALSE.

Example:

We are back in the Block World with the following state

```
        +--+
        |C |
        +--+    +--+
        |A |    |B |
    ----+--+----+--+-------
```

which gives us the following State Clauses:

- ON(C A)
- ONTABLE(A)
- ONTABLE(B)
- CLEAR(C)
- CLEAR(B)

In addition we consider the non-logical axiom:

- (ALL x (CLEAR(x) IMPLIES (NOT (EXISTS y ON(y x)))))

which in clause form becomes

- NOT CLEAR(x) OR NOT ON(y x)

If we now ask whether (NOT (EXISTS y (ON(y C)))), we add to the clauses considered above the clause ON(F C) and apply resolution:

```
  {NOT CLEAR(x) OR NOT ON(y x)} {ON(F C)} {CLEAR(C)}
                       \        /          /
                        \      /          /
                         \    /          /
                       {NOT CLEAR(C)}   /
                                  \    /
                                   \  /
                                    \/
                                    {}
```

Example:

We are given the following predicates:

- S(x) : x is Satisfied
- H(x) : x is Healthy
- R(x) : x is Rich
- P(x) : x is Philosophical

The premises are the non-logical axioms:

- S(x) IMPLIES (H(x) AND R(x))
- EXISTS x (S(x) and P(x))

The conclusion is

- EXISTS x (P(x) AND R(x))

The corresponding clauses are:

1. NOT S(x) OR H(x)
2. NOT S(x) OR R(x)
3. S(B)
4. P(B)
5. NOT P(x) OR NOT R(x)

where B is a Skolem constant.

The proof is then:

```
NOT P(x) OR NOT R(x)    P(B)    NOT S(x) OR R(x)    S(B)
               \        /                  \        /
                \      /                    \      /
                 NOT R(B)                     R(B)
                         \                   /
                          \                 /
                           \               /
                            \             /
                             \           /
                              \         /
                                   {}
```

Answering Fill-in-Blanks Questions

We now determine how we can identify individual(s) that satisfy specific formulae.

EXAMPLE:

NON-LOGICAL SYMBOLS:

- SW(x y): x is staying with y
- A(x y): x is at place y
- R(x y): x can be reached at phone number y
- PH(x): the phone number for place x
- Sally, Morton, UnionBldg: Individuals

NON-LOGICAL AXIOMS:

1. SW(Sally Morton)
2. A(Morton UnionBlidg)
3. SW(x y) AND A(y z) IMPLIES A(x z), which is equivalent to the clause
    1. NOT SW(x y) OR NOT A(y z) OR A(x z)
4. A(x y) IMPLIES R(x PH(y)), which is equivalent to the clause
    1. NOT A(u v) OR R(u PH(v))

GOAL: Determine where to call Sally

- NOT EXISTS x R(Sally x), which is equivalent to the clause
    1. NOT R(Sally w).

To this clause we add as a disjunct the literal, **Answer Literal**, Ans(w) to obtain the clause :

5. Ans(w) OR NOT R(Sally w).

PROOF

6. Ans(v) OR NOT A(Sally v). from 4. and 5.
7. Ans(v) OR NOT SW(Sally y) OR NOT A(y v), from 6. and 3.
8. Ans(v) OR NOT A(Morton v), from 7. and 1.
9. Ans(UnionBldg), from 8. and 2.

The proof procedure terminates when we get a clause that is an instance of the Answer Literal. 9. and gives us the place where we can call Sally.

General Method

If A is the Fill-In-Blanks question that we need to answer and x1 .. xn are the free variables occurring in A, then we add to the Non-Logical axioms and Facts GAMMA the clause

NOT A OR ANS(x1 .. xn)

We terminate the proof when we get a clause of the form

ANS(t1 .. tn)

t1 .. tn are terms that denote individuals that simultaneously satisfy the query for, respectively x1 .. xn.

We can obtain all the individuals that satisfy the original query by continuing the proof looking for alternative instantiations for the variables x1 .. xn.

If we build the proof tree for ANS(t1 .. tn) and consider the MGUs used in it, the composition of these substitutions, restricted to x1 .. xn, gives us the individuals that answer the original Fill-In-Blanks question.