Module 6

Knowledge Representation and Logic – (First Order Logic)

Lesson 16

Inference in FOL - II

Version 1 CSE IIT, Kharagpur

6.2.9 Proof as Search

Up to now we have exhibited proofs as if they were found miraculously. We gave formulae and showed proofs of the intended results. We did not exhibit how the proofs were derived.

We now examine how proofs can actually be found. In so doing we stress the close ties between theorem proving and search.

A General Proof Procedure

We use binary resolution [we represent clauses as sets] and we represent the proof as a tree, the **Proof Tree**. In this tree nodes represent clauses. We start with two disjoint sets of clauses INITIAL and OTHERS.

1. We create a node START and introduce a hyperarc from START to new nodes, each representing a distinct element of INITIAL. We put in a set OPEN all these new nodes. These nodes are called **AND Nodes**.
2. If OPEN is empty, we terminate. Otherwise we remove an element N from OPEN using a selection function SELECT.
3. If N is an AND node, we connect N with a single hyperarc to new nodes N1 ... Nm, one for each literal in the clause C represented at N. These nodes are also labeled with C and are called **OR Nodes**. All of these nodes are placed into OPEN.
   [NOTE 1: In the case that C consists of a single literal, we can just say that N is now an OR node.]

   [NOTE 2: One may decide not to create all the nodes N1 .. Nm in a single action and instead to choose one of the literals of C and to create a node Ni to represent C with this choice of literal. Ni is inserted into OPEN. N also goes back into OPEN if not all of its literals have been considered. The rule used to choose the literal in C is called a **Selection Rule**]

   Repeat from 2.

4. If N is an OR node, say, corresponding to the ith literal of a clause C, we consider all the possible ways of applying binary resolution between C and clauses from the set OTHERS, resolving on the ith literal of C.

   Let D1 .. Dp be the resulting clauses. We represent each of these clauses Dj by an AND node N(Dj) as in 1. above. We put an arc from N to N(Dj). We set OPEN to NEXT-OPEN(OPEN, C, {D1, .., Dp}). We set OTHERS to NEXT-OTHERS(OTHERS, C, {D1, .., Dp}). If in the proof tree we have, starting at START, a hyperpath (i.e. a path that may include hyperarcs) whose leaves have all label {}, we terminate with success.

[NOTE 3: One may decide, instead of resolving C on its ith literal with all possible element of OTHERS, to select one compatible element of OTHERS and to resolve C with it, putting this resolvent and C back into OPEN. We would call a rule for selecting an element of OTHERS a **Search Rule**.]

Repeat from 2.

In this proof procedure we have left indetermined:

- The sets INITIAL and OTHERS of clauses
- The function SELECT by which we choose which node to expand
- The function NEXT-OPEN for computing the next value of OPEN
- The function NEXT-OTHERS for computing the next value of OTHERS

There is no guaranty that for any choice of INITIAL, OTHERS, SELECT, NEXT-OPEN and NEXT-OTHERS the resulting theorem prover will be "complete", i.e. that everything that is provable will be provable with this theorem prover.

Example

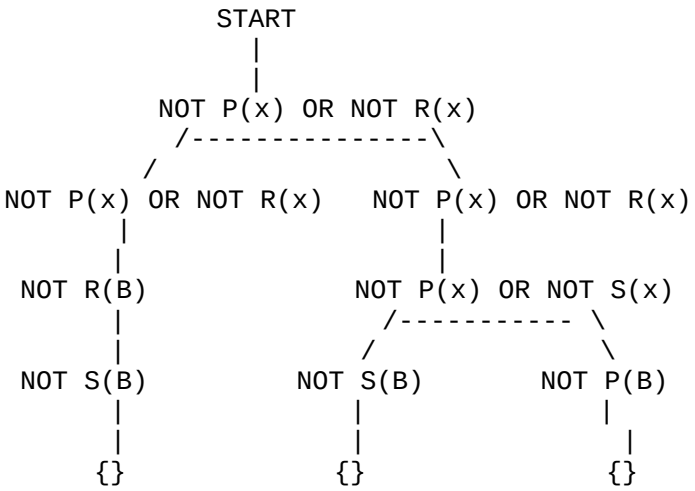
Suppose that we want to prove that

1. NOT P(x) OR NOT R(x)
   is inconsistent with the set of clauses:
2. NOT S(x) OR H(x)
3. NOT S(x) OR R(x)
4. S(b)
5. P(b)

The following are possible selections for the indeterminates:

INITIAL: {1.}, that is, it consists of the clauses representing the negation of the goal.
OTHERS: {2. 3. 4. 5.}, that is, it consists of the non-logical axioms.
SELECT: We use OPEN as a FIFO queue, i.e. we do breadth-first search.
NEXT-OPEN: It sets NEXT-OPEN(OPEN, C, {D1, .., Dp}) to the union of OPEN, {C}, and {D1, .., Dp}.
NEXT-OTHERS: It leaves OTHERS unchanged

The Proof Tree is then (we underline AND nodes and all their outgoing arcs are assumed to form a single hyperlink)



At an OR node we apply resolution between the current clause at its selected literal and all the compatible elements of OTHERS.

Example

The following example uses Horn clauses in propositional logic. I use the notation that is common in Prolog: we represent the implication:

A1 AND A2 AND .. AND An IMPLIES A

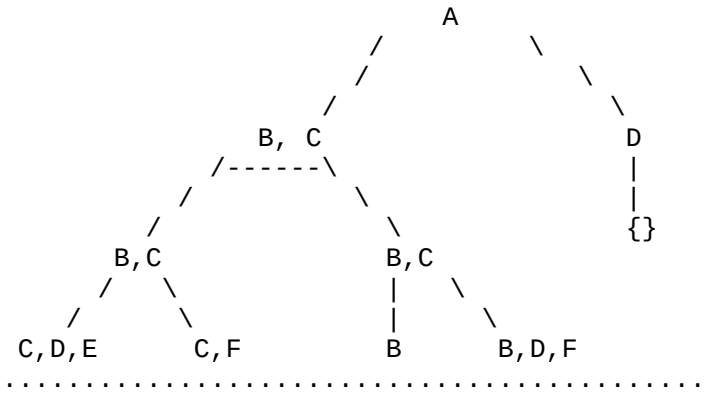
as

A < = A1, A2, .. , An

Our problem is:

1. A, This is what we want to prove
2. A < = B, C
3. A < = D
4. B < = D, E
5. B < = F
6. C < =
7. C < = D, F
8. D < =
9. F < =

We now partially represent the proof tree. We do not apply breadth first because we want to see how great is the impact of the choice of SELECT.



You can keep on expanding the tree and see how depth first would generate a large tree while breadth first rapidly derives D from A, and {} from D.
In other circumstances other strategies would be appropriate as we see below.

6.2.10 Some Proof Strategies

From the early sixties people have looked for strategies that would simplify the search problem in theorem proving. Here are some of the strategies suggested.

6.2.10.1 Unit Preference

When we apply resolution if one of the premises is a unit clause (it has a single literal), the resolvent will have one less literal than the largest of the premises, thus getting closer to the desired goal {}. Thus it appears desirable to use resolution between clauses one of which is the unit clause. This unit preference is applied both when selecting from the OPEN set (i.e. at the leaves of the tree) and when we at an OR node we select an element of OTHERS to resolve with it.

6.2.10.2 Set of Support Strategy

When we use the Set of Support Strategy we have:
NEXT-OPEN(OPEN, C, {D1, .., Dp}) is the union of OPEN, {C}, and {D1,..,Dp}
NEXT-OTHERS(OTHERS, C, {D1,..,Dp}) is OTHERS.

In simple words, the set of support strategy uses the OPEN set as its set of support. Each application of resolution has as a premise an element of the set of support and adds that premise and the resolvent to the set of support.

Usually the INITIAL set (i.e. the initial set of support) consists of all the clauses obtained by negating the intended "theorem".
The set of support strategy is complete if

- The OTHERS set of clauses is satisfiable. That is the case when we are given a satisfiable set of non-logical axioms and we use it as OTHERS.

6.2.10.3 Input Resolution

In Input Resolution:

- INITIAL consists of the union of the negation of the "theorem" and of the set of non-logical axioms.
- OTHERS usually is the set of non-logical axioms.
- NEXT-OPEN(OPEN, C, {D1,..,Dp}) is the union of OPEN and {C}, that is, OPEN does not change in successive iterations
- NEXT-OTHERS(OTHERS, C, {D1,..,Dp}) is the union of OTHERS, {C}, and {D1,..,Dp}.

In other words, in each resolution one of the premises is one of the original clauses.

In general Input Resolution is incomplete, as it can be seen with the following unsatisfiable set of clauses (from Nilsson) from which we are unable to derive FALSE using Input Resolution:

- Q(u) OR P(A)
- NOT Q(w) OR P(w)
- NOT Q(x) OR NOT P(x)
- Q(y) OR NOT P(y)

[You can use this set of clauses as both OPEN and OTHERS.]

Input resolution is complete in the case that all the clauses are Horn clauses.

6.2.10.4 Linear Resolution

Linear Resolution, also known as Ancestry-Filtered Form Strategy, is a generalization of Input Resolution. The generalization is that now in each resolution one of the clauses is one of the original clauses or it is an ancestor in the proof tree of the second premise.

Linear Resolution is complete.

6.2.10.5 SLD-Resolution

Selected Linear Resolution for Definite Clauses, or simply SLD-Resolution, is the basis for a proof procedure for theorems that are in the form of conjunctions of positive literals (this conjunction is called a **Goal**) given non-logical axioms that are definite clauses.

More precisely:

- A Goal is the conjunction of positive literals
- A **Selection Rule** is a method for selecting one of the literals of a goal (the first literal, or the last literal, etc.)

- In SLD-Resolution, given a goal G = A1 .. An, a definite clause C: A < = B1 .. Bm, and a subgoal Ai selected from G using a Selection Rule S, where Ai and A are unifiable with MGU s, the **Resolvent of G and C under S** is

  (A1 .. Ai-1 B1 ..Bm Ai+1..An).s

- An **SLD-Derivation** of a goal G given a selection rule S and a set P of definite clauses, is a sequence of triples (Gi, Ci, si), for i from 0 to k, where
  - G0 is G
  - for each i > 0, Ci is obtained from a clause in P by replacement of all of its variables with new variables
  - Gi+1 is the SLD-Resolvent of Gi and Ci by use of S and with MGU si.
- A **Refutation** of a goal G given definite clauses P and selection rule S, is a finite SLD-derivation of G given S and P whose last goal is the null clause.
  If s1 .. sk are the MGUs used in the refutation, then s = si.s2....sk is a substitution that, restricted to the variables of G, makes G true whenever the clauses of P are true.
- The goal G **succeeds** for given selection rule S and set of definite clauses P if it has a refutation for P and S; otherwise it **Fails**.

Theorem: SLD-Resolution is Sound and Complete for conjunctive positive goals and definite clauses.

An important consequence of this theorem is that it remains true no matter the selection rule we use to select literals in goals. Thus we can select literals as we please, for instance left-to right. An other important aspect is that the substitution s = s1.s2...sn gives us a method for finding the individuals that satisfy the goal in the structures that satisfy the clauses in P.

Nothing has been said in SLD-Resolution about what rule should be used to select the clause of P to resolve with the current literal of the current goal (such a rule is called a **Search rule**).

Example

Suppose that we are given the following clauses:

- WOMAN(MOTHER(v)) ; Every mother is a woman
- GOOD(HUSBAND(ANN)) ; The husband of Ann is good
- GOOD(z) < = LIKES(MOTHER(z), z); if z likes his mother then z is good

and we want to find out a woman that likes's someone's husband.

The goal can be stated as:

- WOMAN(x),LIKES(x,HUSBAND(y))
  [NOTE: the variables x and y are implicitly existentially quantified.]

The SLD-Derivation is:

```
((WOMAN(x),LIKES(x,HUSBAND(y))), WOMAN(MOTHER(v)), [MOTHER(v)/x])
((LIKES(MOTHER(v),HUSBAND(y))), GOOD(z) < = LIKES(MOTHER(z), z),
                                [HUSBAND(y)/v, HUSBAND(y)/z])
((GOOD(HUSBAND(y))), GOOD(HUSBAND(ANN)), [ANN/y])
{}
```

and the substitution is:

[MOTHER(HUSBAND(ANN))/x, ANN/y]

6.2.11 Non-Monotonic Reasoning

First order logic and the inferences we perform on it is an example of monotonic reasoning.

In monotonic reasoning if we enlarge at set of axioms we cannot retract any existing assertions or axioms.

Humans do not adhere to this monotonic structure when reasoning:

- we need to jump to conclusions in order to plan and, more basically, survive.
  - we cannot anticipate all possible outcomes of our plan.
  - we must make assumptions about things we do not specifically know about.

6.2.11.1 Default Reasoning

This is a very common from of non-monotonic reasoning. Here *We want to draw conclusions based on what is most likely to be true.*

We have already seen examples of this and possible ways to represent this knowledge.

We will discuss two approaches to do this:

- Non-Monotonic logic.
- Default logic.

DO NOT get confused about the label *Non-Monotonic* and *Default* being applied to reasoning and a particular logic. Non-Monotonic reasoning is generic descriptions of a class of reasoning. Non-Monotonic logic is a specific theory. The same goes for Default reasoning and Default logic.

Non-Monotonic Logic

This is basically an extension of first-order predicate logic to include a *modal* operator, *M*. The purpose of this is to allow for consistency.

For example: $\forall x$: plays_instrument(x) $\wedge M$ improvises(x) $\rightarrow$ jazz_musician(x)

states that for all x is x plays an instrument and if the fact that x can improvise is consistent with all other knowledge then we can conclude that x is a jazz musician.

How do we define *consistency*?

One common solution (consistent with PROLOG notation) is

to show that fact P is true attempt to prove $\neg P$. If we fail we may say that P is consistent (since $\neg P$ is false).

However consider the famous set of assertions relating to President Nixon.

$\forall x$: Republican(x) $\wedge M \neg$ Pacifist(x) $\rightarrow \neg$Pacifist(x)

$\forall x$: Quaker(x) $\wedge M$ Pacifist(x) $\rightarrow$ Pacifist(x)

Now this states that Quakers tend to be pacifists and Republicans tend not to be.

BUT Nixon was both a Quaker and a Republican so we could assert:

Quaker(Nixon)

Republican(Nixon)

This now leads to our total knowledge becoming inconsistent.

Default Logic

Default logic introduces a new inference rule:

$$\frac{A : B}{C}$$

which states if A is deducible and it is consistent to assume B then conclude C.

Now this is similar to Non-monotonic logic but there are some distinctions:

- New inference rules are used for computing the set of plausible extensions. So in the Nixon example above Default logic can support both assertions since is does not say anything about how choose between them -- it will depend on the inference being made.
- In Default logic any nonmonotonic expressions are rules of inference rather than expressions

6.2.11.2 Circumscription

Circumscription is a rule of conjecture that allows you to jump to the conclusion that the objects you can show that posses a certain property, *p*, are in fact all the objects that posses that property.

Circumscription can also cope with default reasoning.

Suppose we know: bird(tweety)

$\forall x$: penguin(x) $\rightarrow$ bird(x)

$\forall x$: penguin(x) $\rightarrow$ $\neg$flies(x)

and we wish to add the fact that *typically, birds fly*.

In circumscription this phrase would be stated as:

A bird will fly if it is not abnormal

and can thus be represented by:

$\forall x$: bird(x) $\wedge \neg$ abnormal(x) $\rightarrow$ flies(x).

However, this is not sufficient

We cannot conclude

flies(tweety)

since we cannot prove

$\neg$ abnormal(tweety).

This is where we apply circumscription and, in this case,

we will assume that those things that are shown to be abnormal are the only things to be abnormal

Thus we can rewrite our *default rule* as:

$\forall x$: bird(x) $\wedge \neg$ flies(x) $\rightarrow$ abnormal(x)

and add the following

$\forall x$: $\neg$ abnormal(x)

since there is nothing that cannot be shown to be abnormal.

If we now add the fact:

penguin(tweety)

Clearly we can prove

abnormal(tweety).

If we circumscribe abnormal now we would add the sentence,

a penguin (tweety) is the abnormal thing:

$\forall x$: abnormal(x) $\rightarrow$ penguin(x).

Note the distinction between Default logic and circumscription:

Defaults are sentences in language itself not additional inference rules.

6.2.11.3 Truth Maintenance Systems

A variety of *Truth Maintenance Systems* (TMS) have been developed as a means of implementing Non-Monotonic Reasoning Systems.

Basically TMSs:

- all do some form of dependency directed backtracking
- assertions are connected via a network of dependencies.

Justification-Based Truth Maintenance Systems (JTMS)

- This is a simple TMS in that it does not know anything about the structure of the assertions themselves.
- Each supported belief (assertion) in has a justification.
- Each justification has two parts:

    - An *IN-List* -- which supports beliefs held.
    - An *OUT-List* -- which supports beliefs *not* held.

- An assertion is connected to its justification by an arrow.
- One assertion can *feed* another justification thus creating the network.
- Assertions may be labelled with a *belief status*.
- An assertion is *valid* if every assertion in the IN-List is believed and none in the OUT-List are believed.
- An assertion is non-monotonic is the OUT-List is not empty or if any assertion in the IN-List is non-monotonic.

Logic-Based Truth Maintenance Systems (LTMS)

Similar to JTMS except:

- Nodes (assertions) assume no relationships among them except ones explicitly stated in justifications.
- JTMS can represent P and ¬P simultaneously. An LTMS would throw a contradiction here.
- If this happens network has to be reconstructed.

Assumption-Based Truth Maintenance Systems (ATMS)

- JTMS and LTMS pursue a single line of reasoning at a time and backtrack (dependency-directed) when needed -- *depth first search*.
- ATMS maintain alternative paths in parallel -- *breadth-first search*
- Backtracking is avoided at the expense of maintaining multiple contexts.
- However as reasoning proceeds contradictions arise and the ATMS can be *pruned*
    - Simply find assertion with no valid justification.

Questions

1.a. Consider a first-order logical system which uses two 1-place predicates namely, *Big* and *Small*. The set of object constants is given by *{A, B}*. Enumerate all possible models in this case.

1.b. For each of the following sentences, identify the models in which the given sentence is true.

a. $Big(A) \wedge Big(B)$
b. $Big(A) \vee Big(B)$
c. $\forall x\ Big(x)$
d. $\forall x \neg Big(x)$
e. $\exists x\ Big(x)$
f. $\exists x \neg Big(x)$
g. $\forall x\ Big(x) \wedge Small(x)$
h. $\forall x\ Big(x) \vee Small(x)$
i. $\forall x\ Big(x) \Rightarrow \neg Small(x)$

1.c Determine whether the expressions *p* and *q* unify with each other in each of the following cases. If so, give the most general unifier; If not, give a brief explanation (Assume that the upper case letters are (object, predicate, or function) constants and that the lower case letters are variables).

a. $p = F(G(v), H(u, v)); q = F(w, J(x, y))$
b. $p = F(x, F(u, x)); q = F(F(y, A), F(z, F(B, z)))$
c. $p = F(x_1, G(x_2, x_3), x_2, B); q = F(G(H(A, x_5), x_2), x_1, H(A, x_4), x_4)$

1.d. Put the following FOL formulas into conjunctive form (CNF):

a. $\forall x\ \forall y\ ((P(x) \wedge Q(y)) \Rightarrow \exists z\ R(x, y, z))$
b. $\exists x\ \forall y\ \exists z\ (P(x) \Rightarrow (Q(y) \Rightarrow R(z)))$

2. Convert the following English statements to statements in first order logic

a. every boy or girl is a child
b. every child gets a doll or a train or a lump of coal
c. no boy gets any doll
d. no child who is good gets any lump of coal

e. Jack is a boy

3. Using the above five axioms construct a proof by refutation by resolution of the statement

"if Jack doesn't get a train, then Jack is not a good boy"

4. Consider the following set of axioms

i. Everyone who loves someone who loves them back is happy

ii. Mary loves everyone.

iii. There is someone who loves Mary.

From the above statements conclude that:

Mary is happy.

Solution

1.a. Possible models are:

$M_0 = \{\}$
$M_1 = \{\text{Big(A)}\}$
$M_2 = \{\text{Big(B)}\},$
$M_3 = \{\text{Small(A)}\}$
$M_4 = \{\text{Small(B)}\}$
$M_5 = \{\text{Big(A), Big(B)}\}$
$M_6 = \{\text{Big(A),Small(A)}\}$
$M_7 = \{\text{Big(A), Small(B)}\}$
$M_8 = \{\text{Small(A), Big(B)}\}$
$M_9 = \{\text{Big(B),Small(B)}\}$
$M_{10} = \{\text{Small(A), Small(B)}\}$
$M_{11} = \{\text{Big(A), Big(B), Small(A)}\}$
$M_{12} = \{\text{Big(A), Big(B), Small(B)}\}$
$M_{13} = \{\text{Small(A), Small(B), Big(A)}\}$
$M_{14} = \{\text{Small(A), Small(B), Big(B)}\}$
$M_{15} = \{\text{Big(A), Big(B), Small(A), Small(B)}\}$

1.b Answers:

(a) $Big(A) \wedge Big(B) : M_5 M_{11} M_{12} M_{15}$

(b) $Big(A) \vee Big(B) : M_1 M_2 M_5 M_6 M_7 M_8 M_9 M_{11} M_{12} M_{13} M_{14} M_{15}$

(c) $\forall x Big(x)$(equivalent to $Big(A) \wedge Big(B)) : M_5 M_{11} M_{12} M_{15}$

(d) $\forall x \neg Big(x)$(equivalent to $\neg Big(A) \wedge \neg Big(B)) : M_0 M_3 M_4 M_{10}$

(e) $\exists x Big(x)$(equivalent to$Big(A) \vee Big(B)) : M_1 M_2 M_5 M_6 M_7 M_8 M_9 M_{11} M_{12} M_{13} M_{14} M_{15}$

(f) $\exists x \neg Big(x)$(equivalent to $\neg Big(A) \vee \neg Big(B)$ which is equivalent to $\neg(Big(A) \wedge Big(B))) :$
$M_0 M_1 M_2 M_3 M_4 M_6 M_7 M_8 M_9 M_{10} M_{13} M_{14}$

(g) $\forall x Big(x) \wedge Small(x)$(equivalent to $(Big(A) \wedge Small(A)) \wedge (Big(B) \wedge Small(B))) : M_{15}$

(h) $\forall x Big(x) \vee Small(x)$(equivalent to$(Big(A) \vee Small(A)) \wedge (Big(B) \vee Small(B))) : M_5 M_7 M_8 M_{10} M_{11} M_{12} M_{13} M_{14} M_{15}$

(i) $\forall x Big(x) \Rightarrow Small(x)(equivalentto(\neg Big(A) \vee \neg Small(A)) \wedge (\neg Big(B) \vee \neg Small(B))) :$
$M_0 M_1 M_2 M_3 M_4 M_5 M_7 M_8 M_{10}$

1.c. Answers:

(a) $p = F(G(v), H(u, v)); q = F(w, J(x, y))$.
Substitute $w/G(v)$: $p = F(G(v), H(u, v)); q = F(G(v), J(x, y))$ but cannot find a unifier for $H(u, v)$ and $J(x, y)$ because they are different objects (predicates, functions, constants).

(b) $p = F(x, F(u, x)); q = F(F(y, A), F(z, F(B, z))$.
Substitute $x/F(y, A)$: $p = F(F(y, A), F(u, F(y, A)); q = F(F(y, A), F(z, F(B, z))$.
Substitute u/z: $p = F(F(B, A), F(z, F(y, A)); q = F(F(y, A), F(z, F(B, z))$.
Substitute y/B and z/A: $p = F(F(B, A), F(A, F(B, A)); q = F(F(B, A), F(A, F(B, A))$.
So Unify(p, q)=$\{x/F(y, A), u/z, y/B, z/A\}$.

(c) $p = F(x_1, G(x_2, x_3), x_2, B)$; $q = F(G(H(A, x_5), x_2), x_1, H(A, x_4), x_4)$.
Substitute x_4/B: $p = F(x_1, G(x_2, x_3), x_2, B)$; $q = F(G(H(A, x_5), x_2), x_1, H(A, B), B)$.
Substitute $x_1/G(x_2, x_3)$: $p = F(G(x_2, x_3), G(x_2, x_3), x_2, B)$; $q = F(G(H(A, x_5), x_2), G(x_2, x_3), H(A, B), B)$.
Substitute $x_2/H(A, x_5)$: $p = F(G(H(A, x_5), x_3), G(H(A, x_5), x_3), H(A, x_5), B)$;
$q = F(G(H(A, x_5), H(A, x_5)), G(H(A, x_5), x_3), H(A, B), B)$.
Substitute $x_3/H(A, x_5)$: $p = F(G(H(A, x_5), H(A, x_5)), G(H(A, x_5), H(A, x_5)), H(A, x_5), B)$;
$q = F(G(H(A, x_5), H(A, x_5)), G(H(A, x_5), H(A, x_5)), H(A, B), B)$.
Substitute x_5/B so Unify(p, q)=$\{x_4/B, x_1/G(x_2, x_3), x_2/H(A, x_5), x_3/H(A, x_5), x_5/B\}$.

1.d Answers

(a) $\forall x \forall y((P(x) \wedge Q(y) \Rightarrow \exists z R(x, y, z)) =$
$\forall x \forall y(\neg(P(x) \wedge Q(y)) \vee \exists z R(x, y, z)) =$
$\forall x \forall y(\neg P(x) \vee \neg Q(y) \vee R(x, y, G(x, y))) =$
$\forall x \forall y((P(x) \wedge Q(y) \Rightarrow R(x, y, G(x, y)))$

(b) $\exists x \forall y \exists z(P(x) \Rightarrow$
$(Q(y) \Rightarrow R(z))) =$
$\exists x \forall y \exists z(\neg P(x) \vee (Q(y) \Rightarrow R(z))) =$
$\exists x \forall y \exists z(\neg P(x) \vee (\neg Q(y) \vee R(z))) =$
$\exists x \forall y \exists z(\neg P(x) \vee \neg Q(y) \vee R(z))$ (where B is any constant not used elsewhere) $=$
$\forall y(\neg P(B) \vee \neg Q(y) \vee R(G(y))) =$
$\forall y(P(B) \wedge Q(y)) \Rightarrow R(G(y))$

2. The FOL statements are

a. forall x ((boy(x) or girl(x)) -> child(x))
b. forall y (child(y) -> (gets(y,doll) or gets(y,train) or gets(y,coal)))
c. forall w (boy(w) -> !gets(w,doll))
d. forall z ((child(z) and good(z)) -> !gets(z,coal))
e. boy(Jack)

3. The statement to be proved can be written in FOL as:

!gets(Jack,train) -> !good(Jack)

The proof consists of three steps:

- Negate the conclusion.
- Translate all statements to clausal form.
- Apply resolution to the clauses until you obtain the empty clause

The steps are shown below:

- Transform axioms into clause form:
  1. forall x ((boy(x) or girl(x)) -> child(x))
     !(boy(x) or girl(x)) or child(x)
     (!boy(x) and !girl(x)) or child(x)
     (!boy(x) or child(x)) and (!girl(x) or child(x))
  2. forall y (child(y) -> (gets(y,doll) or gets(y,train) or gets(y,coal)))
     !child(y) or gets(y,doll) or gets(y,train) or gets(y,coal)
  3. forall w (boy(w) -> !gets(w,doll))
     !boy(w) or !gets(w,doll)
  4. forall z ((child(z) and good(z)) -> !gets(z,coal))
     !(child(z) and good(z)) or !gets(z,coal)
     !child(z) or !good(z) or !gets(z,coal)
  5. boy(Jack)
  6. !(!gets(Jack,train) -> !good(Jack)) negated conclusion
     !(gets(Jack,train) or !good(Jack))
     !gets(Jack,train) and good(Jack)
- The set of CNF clauses:
  1. (a) !boy(x) or child(x)
     (b) !girl(x) or child(x)
  2. !child(y) or gets(y,doll) or gets(y,train) or gets(y,coal)
  3. !boy(w) or !gets(w,doll)
  4. !child(z) or !good(z) or !gets(z,coal)
  5. boy(Jack)

6. (a) !gets(Jack,train)
   (b) good(Jack)

- Resolution:
  - 4. !child(z) or !good(z) or !gets(z,coal)
    6.(b). good(Jack)
    ---
    7. !child(Jack) or !gets(Jack,coal) (substituting z by Jack)
  - 1.(a). !boy(x) or child(x)
    5. boy(Jack)
    ---
    8. child(Jack) (substituting x by Jack)
  - 7. !child(Jack) or !gets(Jack,coal)
    8. child(Jack)
    ---
    9. !gets(Jack,coal)
  - 2. !child(y) or gets(y,doll) or gets(y,train) or gets(y,coal)
    8. child(Jack)
    ---
    10. gets(Jack,doll) or gets(Jack,train) or gets(Jack,coal) (substituting y by Jack)
  - 9. !gets(Jack,coal)
    10. gets(Jack,doll) or gets(Jack,train) or gets(Jack,coal)
    ---
    11. gets(Jack,doll) or gets(Jack,train)
  - 3. !boy(w) or !gets(w,doll)
    5. boy(Jack)
    ---
    12. !gets(Jack,doll) (substituting w by Jack)
  - 11. gets(Jack,doll) or gets(Jack,train)
    12. !gets(Jack,doll)
    ---
    13. gets(Jack,train)
  - 6.(a). !gets(Jack,train)
    13. gets(Jack,train)
    ---
    14. empty clause

4. The axioms in FOL are:

A1: for-all x, for-all y, [loves(x,y) & loves(y,x) = > happy(x)]

A2: for-all z, [loves(mary,z)]

A3: there-is w, [loves(w,mary)]

The conclusion can be written as

C: happy(mary)

The proof is as follows:

- Translate the axioms and the negation of the conclusion into clausal form. Show each step of the translation (remember that the order in which you apply the steps is crucial).
  1. **A1:** !loves(x,y) || !loves(y,x) || happy(x)
     note that p = > q is equivalent to !p || q and that !(r & s) is equivalent to !r || !s
  2. **A2:** loves(mary,z)
  3. **A3:** loves(**a**,mary), where **a** is a new constant symbol
  4. **!C:** ! happy(mary)

- Apply resolution (state explicitly what clauses are resolved and which substitutions are made) until the empty clause is found.

```
!C: ~~! happy(mary)~~
A1:  !loves(x,y) || !loves(y,x) || ~~happy(x)~~
_____________________________________________  with x=mary
 A4:  ~~!loves(mary,y)~~ || !loves(y,mary)
 A2: ~~loves(mary,z)~~
_____________________________________________  with z=y
 A5:  ~~!loves(y,mary)~~
 A3: ~~loves(a,mary)~~
 _____________________________________________  with y=a
    empty clause
```

Since assuming that "mary is not happy" yields a contradiction in the presence of A1, A2, A3, then the statement "mary is happy" is a logical consequence of A1, A2, and A3.