CHAPTER 7

Resolution

We shall extend the Horn clause model of problem-solving to non-Horn clauses. With non-Horn clauses

(1) goals and assertions can be negative as well as positive,

(2) the application of procedures to goals can generate assertions as well as subgoals,

(3) the solution of subgoals can require the analysis of several alternative cases and

(4) solutions can be disjunctions: $x = t_1$ or t_2 or ... or t_m.

Top-down and bottom-up inference can be extended to non-Horn clauses. The new rules, as well as the old ones, are all special cases of the general resolution rule introduced by Robinson [1965a].

Negative goals and assertions

In many cases a set of non-Horn clauses can be reexpressed as Horn clauses by renaming predicate symbols [Meltzer 1966]. The non-Horn clause

Pleasant(x), Nightmare(x) <– Dream(x)

for example, can be rewritten as the Horn clause

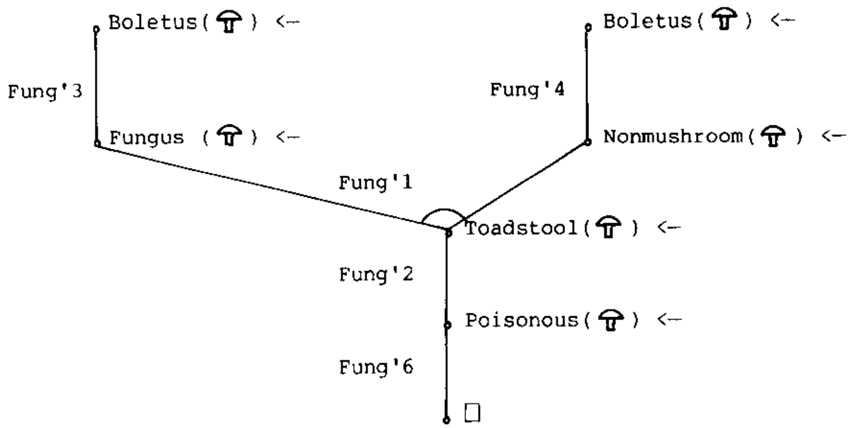
Nightmare(x) <– Dream(x), Unpleasant(x)

by reexpressing the negative atom not-Pleasant(x) as the positive atom Unpleasant(x).

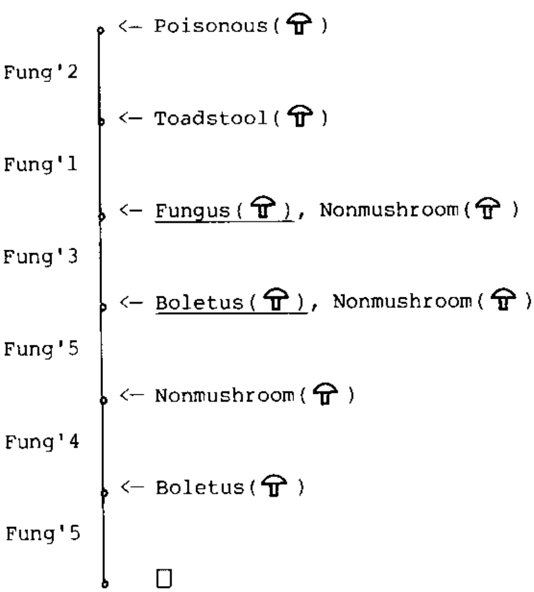
Similarly the non-Horn clause problem of showing that every boletus is poisonous can be transformed into a Horn clause problem by eliminating the predicate symbol "Mushroom" and using the new predicate symbol "Nonmushroom" instead. The unnegated atom, Nonmushroom(x), means the same as the negated atom, not-Mushroom(x). The new Horn clause problem Fung'1-6 can be solved top-down or bottom-up.

```
Fung'1    Toadstool(x) <- Fungus(x), Nonmushroom(x)
Fung'2    Poisonous(x) <- Toadstool(x)
Fung'3    Fungus(x) <- Boletus(x)
Fung'4    Nonmushroom(x) <- Boletus(x)
Fung'5    Boletus(🍄) <-
Fung'6    <- Poisonous(🍄)
```

<u>A</u> <u>bottom-up</u> <u>solution</u>:



<u>A</u> <u>top-down</u> <u>solution</u>:



The bottom-up derivation of the assertion

Nonmushroom(🍄) <-

from the Horn clauses Fung'4 and Fung'5 is equivalent to the derivation of the negative "assertion"

<- Mushroom(🍄)

directly from the original clauses Fung 4-5,

<- Boletus(x), Mushroom(x)

Boletus(🍄) <- .

Similarly the top-down derivation of the positive subgoals

<- Fungus(🍄), Nonmushroom(🍄)

from the goal statement

<- Toadstool(🍄)

by means of the Horn clause Fung'1 is equivalent to the direct derivation of the clause

Mushroom(x) <- Fungus(x)

from the same goal statement

<- Toadstool(🍄)

by means of the non-Horn clause

Fung1 Toadstool(x), Mushroom <- Fungus(x).

Resolution

In general, top-down and bottom-up inference for both Horn clauses and non-Horn clauses are special cases of the resolution rule: To create a resolvent of two clauses it is necessary first to rename variables so that different clauses contain different variables.

> Given a condition in one clause and a conclusion in the other, the resolvent exists if the condition and the conclusion match. The two clauses are said to be the parents of the resolvent clause. An atom is a condition of the resolvent if it is obtained by applying the matching substitution to a condition, different from the matched condition, of one of the parents. Similarly, an atom is a conclusion of the resolvent if it is obtained by applying the matching substitution to a conclusion, different from the matched conclusion, of one of the parent clauses.

The definition can be expressed by means of Horn clauses. Let

res(x,u,y,v)	name the resolvent which exists when, after appropriate renaming of variables, the condition u in x matches the conclusion v in y,
cond(x)	the collection of conditions of clause x,
concl(x)	the collection of conclusions of clause x,
union(x,y)	the union of x and y,
Apply(x,w,x')	express that the result of applying to x the substitution w is x',
Rename(x,y,w)	the substitution w applied to clauses x and y results in clauses which contain no variables in common,
Match(u,v,w)	substitution w matches the atoms u and v,
Member(u,x)	u is a member of x,
Combine(w_1,w_2,w)	the substitution w has the combined effect of first applying substitution w_1 and then applying substitution w_2,
Resolves(x,u,y,v,w)	the resolvent of x and y on atoms u and v exists and w is the combined substitution which both renames variables and matches atoms.

Resolves(x,u,y,v,w) <- Rename(x,y,w_1),Member(u,cond(x)),Apply(u,w_1,u'),
Member(v,concl(y)),Apply(v,w_1,v'),Match(u',v',w_2),
Combine(w_1,w_2,w)

Member(z, cond(res(x,u,y,v))) <- Resolves(x,u,y,v,w),
Member(z', union(cond(x),cond(y))),
Diff(z',u), Apply(z',w,z)

Member(z, concl(res(x,u,y,v))) <- Resolves(x,u,y,v,w),
Member(z', union(concl(x),concl(y))),
Diff(z',v), Apply(z',w,z)

Member(z, union(x,y)) <- Member(z,x)

Member(z, union(x,y)) <- Member(z,y)

Notice that the definition can be used either top-down or bottom-up. The Boyer-Moore structure-sharing implementation of resolution [1972] can be regarded as using the definition top-down but saving solved subgoals of the form Resolves(x,u,y,v,w) as lemmas.

The definition given here is less general than Robinson's which also incorporates the factoring rule described later in the chapter.

Middle out reasoning with Horn clauses

In addition to top-down and bottom-out inference, resolution includes middle-out reasoning with Horn clauses. The resolvent of the two clauses

<u>Fallible(x)</u> <- Human(x)
Mortal(x) <- <u>Fallible(x)</u>

for example, is the clause Mortal(x) <- Human(x).

Middle-out reasoning can also be applied to different copies of the same clause. From two copies of the definition of ancestor, for example

```
Ancestor(x,y) <- Ancestor(x,z), Ancestor(z,y)
Ancestor(u,v) <- Ancestor(u,w), Ancestor(w,v)
```

we can derive the resolvent

```
Ancestor(x,y) <- Ancestor(x,w),Ancestor(w,z),Ancestor(z,y).
```

Propositional logic example

The clauses which define the semantics of propositional logic provide instructive examples of the resolution rule. Here if x and y name propositions x* and y* respectively then

x & y	names the proposition	x* and y*
x V y		x* or y*
x ⊃ y		if x* then y*
x<=>y		x* if and only if y*
¬ x		it is not the case that x*.

where &, V , ⊃, <=> and ¬ are infix function symbols. Read True(x) as stating that x is true. The following set of clauses cannot be reexpressed as Horn clauses by renaming predicate symbols.

T1	True(x&y) <- True(x), True(y)
T2	True(x) <- True(x&y)
T3	True(y) <- True(x&y)
T4	True(xVy) <- True(x)
T5	True(xVy) <- True(y)
T6	True(x), True(y) <- True(xVy)
T7	True(x⊃y),True(x) <-
T8	True(x⊃y) <- True(y)
T9	True(y) <- True(x), True(x⊃y)
T10	True(x<=>y) <- True(x y), True(y⊃x)
T11	True(x⊃y) <- True(x<=>y)
T12	True(y⊃x) <- True(x<=>y)
T13	True(¬x), True(x) <-
T14	<- True(¬x), True(x)

Clauses T1-3 state that

x & y is true if and only if
x is true and y is true.

Clause T1 is the if-half of the statement and clauses T2-3 are the only-if-half. Similarly the remaining clauses state that

T4-6 x V y is true if and only if
x is true or y is true;

T7-9 x ⊃ y is true if and only if
if x is true then y is true;
T10-12 x ⇔ y is true if and only if
x ⊃ y is true and y ⊃ x is true;

T13-14 ¬ x is true if and only if
x is not true.

This set of clauses is based upon a more general definition of "truth" for sentences in the standard form of logic formulated by Colmerauer [unpublished].

The if-halves of the statements are useful top-down to reduce problems concerning the truth of a complex proposition to subproblems concerning the truth of simpler propositions. The only-if halves, on the other hand, are useful bottom-up to derive conclusions concerning the truth of simple propositions from assumptions concerning the truth of more complicated ones.

For example, to show that

p & ¬q is true if p is true and q is not true

it is natural to reason top-down from the goal

<- True(p & ¬q)

using the assumptions

A1 True(p) <-
A2 <- True(q)

and regarding the second assumption A2 as a negative assertion.

```
     <- True(p & ¬q)
T1
     <- True(p), True(¬q)
A1
     <- True(¬q)
T13
        True(q) <-
A2
        □
```

Here the clause T13 can be regarded as reducing the problem of showing that ¬q is true to the problem of showing that q is not true, which is solved directly by assumption A2.*

On the other hand, to show that

p is true and q is not true if p & ¬q is true

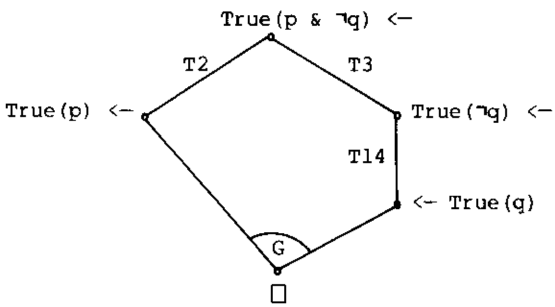
it is more natural to reason bottom-up from the assumption

True(p & ¬q) <– .

The clause

G True(q) <– True(p)

can be interpreted as expressing the goal of showing that p is true and q not true.



Clause T14 can be regarded as deriving the negative assertion that q is not true, which solves the negative goal in G. Notice that the bundle of arcs labelled G represents two successive resolution steps. The order in which the steps are performed is not significant.

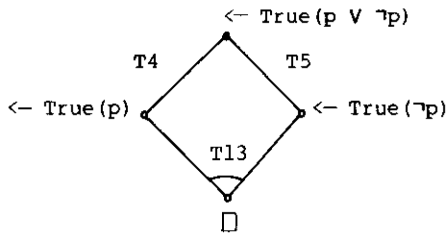
The problem of showing that

p V ¬p is true

illustrates another characteristic feature of top-down problem-solving with non-Horn clauses: <u>No one method adequately solves the problem, but several alternative methods exhaust all the cases</u>.

---

*Throughout this chapter only resolution refutations are exhibited. Search spaces will be investigated in the next chapter.



Methods T4 and T5 reduce the original problem to subproblems which exhaust the two cases asserted by the non-Horn clause T13.

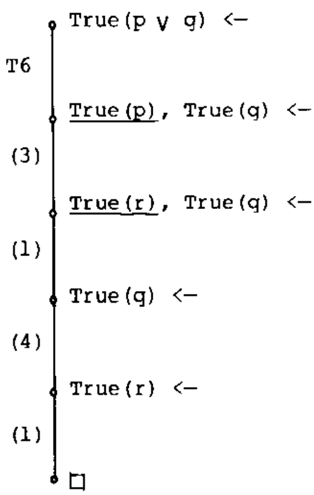
A bottom-up solution of the same problem would involve reasoning by cases. Case analysis by bottom-up reasoning can be seen more clearly, however, for the problem of showing that

r is true if p V q is true,

assuming that

r is true if p is true, and r is true if q is true.

(1) <- True(r)
(2) True(p V q) <-
(3) True(r) <- True(p)
(4) True(r) <- True(q)



Clause T6 derives a non-Horn clause which expresses that there are two cases. The solution reasons bottom-up, first solving the goal in the case that p is true and then solving it in the case that q is true. It "remembers" the second case while it is working on the first one.

Given a goal and a Horn clause which reduces the goal to subgoals, non-Horn clauses can be used to derive assumptions to assist the solution of the subgoals. Such non-Horn clauses typically arise from non-clausal sentences of the form

A <- [B <- C], D

in which a condition is an implication. In the problem-solving interpretation, the clausal form of such a sentence

A, C <- D
A <- B, D

can be regarded as stating that

in order to solve A, solve D, and solve B assuming C.

The clauses T7-8 arise from such a non-clausal sentence:

True(x ⊃ y) <- [True(y) <- True(x)]

To show that x ⊃ y is true,
show that y is true assuming that x is true.

In some cases only one of the clauses T7-8 is needed to solve the problem. If x is not true as in the case

<- True((p & ¬p) ⊃ q)

then only the non-Horn clause T7 which derives the assertion

True(p & ¬p) <-

is needed. But if y is true as in the case

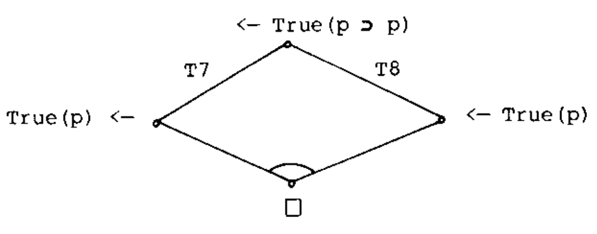
< True(q ⊃ (p V ¬p))

then only the Horn clause T8 which derives the subgoal

<- True(p V ¬p)

is needed.

In most cases, however, both clauses need to be used. The simplest problem which requires the cooperation of clauses T7-8 is that of showing that p p is true.



The derived subgoal of showing that p is true is solved by the derived assertion that p is true. The bundle of arcs associated with the resolution step is unlabelled, because only derived clauses are involved in the inference.

The problem of showing that

$$p \supset q \text{ is true if } p \supset r \text{ is true and } r \supset q \text{ is true}$$

is more interesting. Here it is natural to reason bi-directionally, both forward from the two assumptions and backward from the conclusion. Moreover, when reasoning backward from the conclusion

<- True(p $\supset$ q)

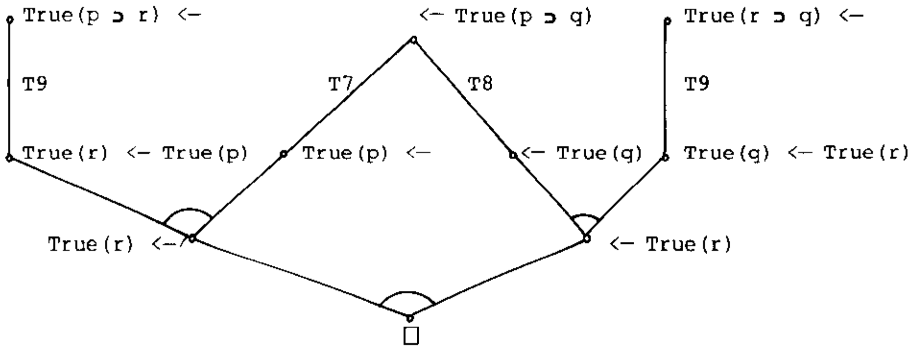
it is natural to reason forward from the derived assertion

True(p) <-

and backward from the derived subgoal

<- True(q)

The following resolution proof formalises the argument.



<u>Arrow notation for non-Horn clauses</u>

The arrow notation used earlier for Horn clauses, to indicate the combination of top-down and bottom-up inference, can also be used for non-Horn clauses. The problem-solving interpretation, in particular, of sentences of the form

A <- [B <- C]

can be indicated by arrows associated with the corresponding clauses

```
1   2               1
⇓   ⇑               ⇓
A,  C  <-           A  <-  B
                           ⇓
                           2
```

The notation associated with the first clause indicates that it should wait for a subgoal of the form A and then derive the assertion C <- . The notation associated with the second clause indicates that it should wait for a subgoal of the form A and then derive the subgoal B.

The use of arrow notation to control the behaviour of a problem-solver will be investigated in the next chapter.

Disjunctive solutions to non-Horn clause problems

Plan-formation tasks, described by means of non-Horn clauses, may require the construction of conditional plans from disjunctive solutions.

Consider, for example, the problem of putting the maximum of two numbers A and B in a location L:

```
M1          <- Holds(val(L,x), w), Max(A,B,x)
M2          Numb(A) <-
M3          Numb(B) <-
M4          Location(L) <-
M5          u ≤ v, v ≤ u <- Numb(u), Numb(v)
M6          Max(u,v,u) <- v ≤ u
M7          Max(u,v,v) <- u ≤ v
```

Suppose that the only action available is the assignment operation. Given a state w, it generates the new state

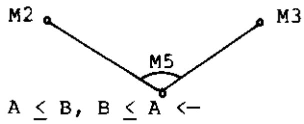
assign(u,v,w)

which results from w by putting v in location u. The "semantics" of the action are described by specifying its preconditions and the statements which are added and deleted when the action is performed. To simplify matters, the single precondition, that u be a location, can be incorporated into the clauses which specify the added (M8) and deleted (M9) statements:

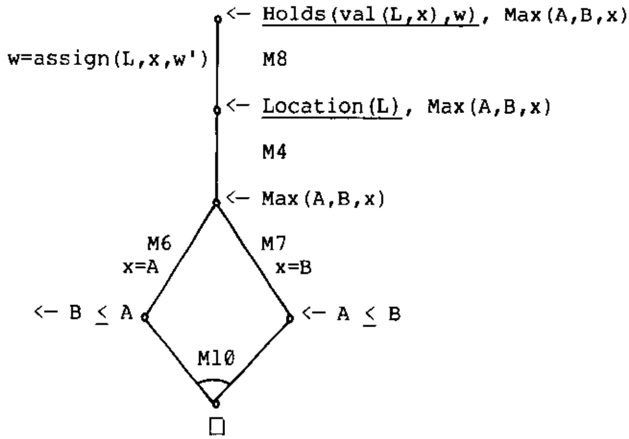
```
M8          Holds(val(u,v), assign(u,v,w)) <- Location(u)
M9          Holds(x, assign(u,v,w)) <- Holds(x,w), Diff(x, val(u,y)),
                                       Location(u)
```

Before solving the problem top-down it is convenient to reason one step bottom-up:

M1Ø



The top-down solution using the derived lemma M1Ø requires that the two procedures M6 and M7 cooperate to solve the single subgoal Max(A,B,x).



The solution is a disjunction of two possibilities

w = assign(L,A,w') or assign(L,B,w'), for any w'.

A solution exists, but it is not determined_3 which of the two possibilities it is.

$\underline{\text{Non-determinism}_3}$ contrasts with non-determinism_1. A problem is $\text{non-deterministic}_3$ if its solution

$$x = t_1 \text{ or } t_2 \text{ or } \ldots \text{ or } t_m$$

is underspecified. It is $\text{non-deterministic}_1$ if its solution is overspecified

$$x = t_1 \text{ and } t_2 \text{ and } \ldots \text{ and } t_m.$$

The treatment of program construction as an application of plan-formation was first proposed by Green [1969b] and Lee and Waldinger [1969]. Lee and Waldinger, in particular, present an algorithm for extracting conditional programs, such as

```
If A ≤ B then w = assign(C,B,w')
          else w = assign(C,A,w')
```

from disjunctive solutions. The relationship between plan-formation and axiomatic semantics of programming languages has been investigated by Moss [1977].

Factoring

The resolution rule alone is complete for demonstrating the inconsistency of Horn clauses. Moreover, it is also adequate for many, but not all, non-Horn clause problems. The combination of factoring and resolution, first described in Robinson's original, unpublished paper is equivalent to the published version of the resolution rule [Robinson 1965a]. Consequently, the completeness proof in the published paper establishes completeness of resolution and factoring combined.

The barber paradox is a simple example which requires the use of factoring.

> Suppose that all barbers shave all people who do not shave themselves and no barber shaves anyone who shaves himself. Then there are no barbers.

To establish the conclusion we assert that there is a barber and attempt to derive a contradiction.

B1	Shave(x,y), Shave(y,y) <- Barber(x)
B2	<- Shave(x,y), Shave(y,y), Barber(x)
B3	Barber(☺) <-

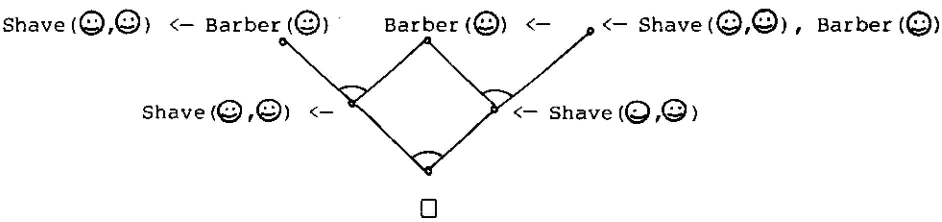
That the three clauses are inconsistent can be demonstrated by instantiating the first two clauses

Shave(☺,☺), Shave(☺,☺) <- Barber(☺)
<- Shave(☺,☺), Shave(☺,☺), Barber(☺)

deleting duplicate atoms

Shave(☺,☺) <- Barber(☺)
<- Shave(☺,☺), Barber(☺)

and applying resolution.



That resolution alone is inadequate for demonstrating inconsistency can be seen more clearly by considering a simpler example:

S1	S(x), S(y) <-
S2	<- S(u), S(v)

The two clauses are inconsistent because they have instances

S(x), S(x) <-
<- S(u), S(u)

which, after removal of duplicate atoms, are directly contradictory:

S(x) <-
<- S(u)

However, no matter how many times resolution is applied to clauses S1-2 and their descendants, every resolvent contains exactly two atoms, and consequently no resolvent is the empty clause (which contains no atoms).

The factoring rule, which needs to supplement resolution in these examples, generates instances of clauses in order to delete duplicate atoms. The instantiating substitution can be restricted so that it matches the two atoms which become duplicates. Applied to the two clauses B1 and B2, factoring generates instances which are more general than the two instances considered before.

B1 Shave(x,y), Shave(y,y) <- Barber(x)
(match underlined atoms)

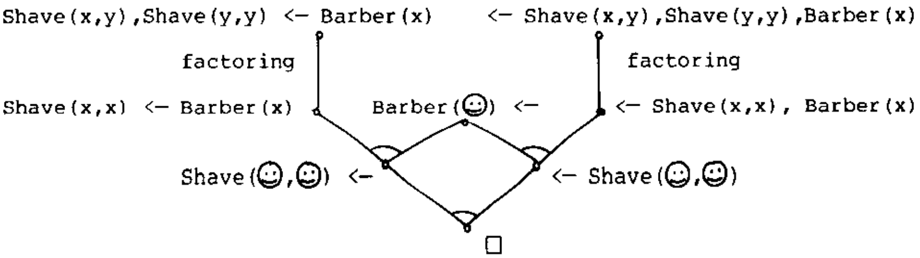
Shave(x,x), ~~Shave(x,x)~~ <- Barber(x)
(delete duplicates)

B'1 Shave(x,x) <- Barber(x)

B'1 is the only factor of B1. Similarly B'2 is the only factor of B2:

B'2 <- Shave(x,x), Barber(x)

Application of factoring and the combined resolution and factoring refutation can be exhibited in a graph.



Factoring is only necessary infrequently and it creates redundancy if it is applied too often. Perhaps the most restrictive constraint on the use of factoring, without affecting completeness, is the one incorporated in the model elimination proof procedure [Loveland 1968, 1969, 1978].

Exercises

1) Use resolution and factoring to show that the assumptions

John likes anyone who doesn't like himself.
John likes no one who likes himself.

are inconsistent.

2) Suppose I believe:

(a) There exists a dragon.
(b) The dragon either sleeps in its cave or hunts in the forest.
(c) If the dragon is hungry then it cannot sleep.
(d) If the dragon is tired then it cannot hunt.

Use resolution to answer the following questions:

What does the dragon do when it is hungry?
What does the dragon do when it is tired?
What does the dragon do when it is hungry and tired?

To answer the questions it is necessary to make explicit the assumption:

If x cannot do y then x does not do y.

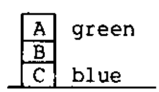
3) Express the following assumptions in clausal form:

Everyone admires a hero.
A failure admires everyone.
Anyone who is not a hero is a failure.

Use resolution and factoring to find a pair of individuals (not necessarily distinct) who admire one another.

4) This problem is discussed by Moore [1975]. Suppose there are three blocks A, B and C.

A is on B which is on C.
A is green, C is blue and
the colour of B is unknown.



Use resolution (and factoring if necessary) to find a green block on a block which is not green. You must assume that blue is not green. What block does the proof find?

5) Using resolution and factoring, show that the following conclusions follow from assumptions T1-14.

(a) If $p \supset (r \& q)$ is true
then $(p \supset r) \& (p \supset q)$ is true.

(b) If $p \supset q$ is true
then there is an r such that $(p \supset r) \& (r \supset q)$.
What r does the proof find?

6) The relation Plus(x,y,z) which holds when x+y = z can be defined using non-Horn clauses

Plus(x,y,z), Add(∅,y) <-
Plus(x,y,z) <- Add(x,z)
Add(s(x),s(z)) <- Add(x,z)

where s(x) names the successor of x. Use resolution and factoring to solve the problem

<- Plus(x,y,s(y)), Plus(x,x,y).