Augmenting Resolution:
Deductive Problem Solving and Question Answering; Handling Equality through Paramodulation

You can solve problems (or answer questions) by proving that a solution (or answer) exists

Formal statement of resolution rule

The previous examples should have made the resolution rule reasonably clear; but let's summarize by stating it formally:

$$\frac{\pi(\sigma_1, ..., \sigma_k) \vee \alpha_1 \vee ... \vee \alpha_m, \quad \neg\pi(\tau_1, ..., \tau_k) \vee \beta_1 \vee ... \vee \beta_n}{\alpha'_1 \vee ... \vee \alpha_m \vee \beta'_1 \vee ... \vee \beta'_n}$$

where $(\sigma_1, ..., \sigma_k)$ and $(\tau_1, ..., \tau_k)$ are unifiable, with unifier γ; and primed literals denote the original unprimed literals after application of the substitution γ. (This is often written as $\alpha'_1 = \alpha_1\gamma$, $\alpha'_2 = \alpha_2\gamma$, etc. Of course, the literals $\pi(\sigma_1, ..., \sigma_k)$ and $\neg\pi(\tau_1, ..., \tau_k)$ need not be first in their respective clauses.

Deductive Problem Solving and Question Answering

In the early days of AI, the emphasis in research on problem solving was on *search*, i.e., on effective ways of navigating through a large space of possibilities, until a solution to the given problem was found. Thus, problems were conceptualized as consisting of (a) a description of the initial problem state, (b) a description of the desired goal state (where (a) and (b) can use any representations suitable for the problem), and (c) a set of "operators" or actions that could be applied to move from one problem state in the search space to another, until a state was reached wherein the goal description is satisfied.

For instance, in a calculus integration problem the initial description might specify an expression to be integrated, and the goal description might call for an algebraic expression free of integral signs whose derivative is the expression to be integrated, and the operators might be ways of transforming the given integral (e.g., integration by parts; e.g., rewriting the integral of x^k as $x^{k+1}/(k+1) + c$)). In a robot planning system, the initial description might be the current situation (including the robot's position, etc.) in the robot's environment, and the goal description might specify some relationships among objects that don't currently hold (such as that a particular box will be in a new location). The operators might be descriptions of actions available to the robot (with specification of "preconditions" and "effects" of those operators).

In all of this, there was little commonality among the 'descriptions' used for different problem domains. This generated a lot of talk, and some research, on systematic ways of changing problem representations so as to make problem solving easier in particular cases. The only commonality that was apparent was the need for search, and so various general search strategies, using various heuristics to guide them, were developed (e.g., Newell, Shaw and Simon's GPS, Slagle's MULTIPLE, and Nilsson's A* strategy).

The idea of Green's method

In a classic paper in 1969, Cordell Green suggested the use of logic and theorem proving as a general domain-independent framework for problem solving ("Application of theorem proving to problem solving", Proc. of IJCAI'69; reprinted in Webber & Nilsson, *Readings in AI*). The essential idea is this:

> **To solve a problem, prove that a solution exists; then extract the solution from the variable bindings in the proof.**

When we study deductive planning later in the course, we will see various examples of how this idea can be implemented to derive plans deductively in the *situation calculus*. This uses FOL to describe states of the world, and changes in these states through actions, modelled as functions that map states to states.

Right now we focus on *question answering*, which is really problem solving in a slightly different guise. For instance, instead of posing the problem "Find the integral of $\sqrt{1+x^2}$", we can equally well ask "What is the integral of $\sqrt{1+x^2}$?"; and instead of saying, "Find a plan for getting from the current state to a state satisfying the goal description", we can ask, "Is there a state satisfying the goal description?" (where we assume that the robot's world has been specified in such a way that such a state exists just in case there is a series of actions that gets to that state). Thus, we can reformulate the above idea more or less equivalently like this:

> **To answer a question, prove that an answer exists; then extract the answer from the variable bindings in the proof.**

In a second 1969 paper, Green showed how to use resolution in this way ("Theorem proving by resolution as a basis for question-answering systems", in Meltzer & Michie, *Machine Intelligence 4*). Let's first note that there are two types of questions, *yes-no* questions and *wh*-questions. Answering a yes-no question requires determining the truth or falsity of a proposition; answering a *wh*-question (a *who-, what-, which-, where-, when-, why-,* or *how*-question) requires coming up with some object(s) that correctly "fills in the blank(s)" in a question. For instance, in "Who loves Juliet?", the *who* can be viewed as a blank to be filled by some object description, such as "Romeo".

Green's method for yes-no questions

We can try to determine the truth or falsity of a proposition by making parallel attempts to prove the proposition and to prove its denial.

For example, in the earlier "paranoia" example, instead of posing the problem "Prove that there is someone whom all nonparanoids dislike", we could have asked the yes-no question, "Is there someone whom all nonparanoids dislike?". We could then have attempted a proof and a disproof. For the disproof, we would have directly transformed the assertion that there is such an individual to clause form, and attempted to use this in conjunction with the given facts (in clause form) to derive the empty clause. In other words, if the proposition in question leads to a contradiction, the answer to the question must be "no". Of course, in this example we know we would have failed to derive the empty clause in such a refutation attempt, since the answer to the question is "yes"! Similarly, the group theory example could have been formulated as a yes-no question, "Is there a right identity element?", in which case we would also have considered *dis*proving there is such an element, again by attempting to derive a contradiction from the assumption that there is one. And again, the disproof attempt would have failed, since the correct answer is "yes, there is such an element".

Here we should make an important observation about the computability properties of this method. In general, given that resolution theorem proving is complete (at least if augmented with factoring and paramodulation, depending on whether we are using narrow or wide resolution and whether the premises involve equality), we know that if a set of clauses is unsatisfiable, we can derive the empty clause in a finite amount of time. However, what if the clauses happen to be satisfiable? Well, then our refutation attempt might go on forever, without our ever finding out whether the empty clause is derivable or not. (Even if we've gone on for a million steps, it may be that in the next few steps we could derive $\square$!) This is an inevitable manifestation of the fact that FOL is not *decidable*, but only *semidecidable*: we can effectively confirm that a set of premises is unsatisfiable, but not, in general, that it is satisfiable.

This is why it's important to make proof and disproof attempts in parallel (i.e., in some time-shared fashion) in automated question answering. We want to be sure we don't go on forever with a proof attempt, when in fact the *denial* of the proposition we are trying to prove follows from the premises. In other words, we don't want to go on forever trying to prove that the answer to the question is "yes", when in fact we could have proved that the answer is "no" in a finite amount of time. Conversely, we don't want to go on forever trying to prove the answer is "no" when in fact a "yes" answer can be proved in a finite amount of time.

Now, shouldn't the answer to a yes-no question always be either "yes" or "no", i.e., shouldn't either the proof attempt or the disproof attempt succeed eventually? Unfortunately we can't assume that – the premises may not be strong enough to answer the question either way. No finite system can know the answer to every question! So

there is no escape from the undecidability of FOL; any QA-procedure that is guaranteed to always find a "yes" answer when the proposition in question follows from the premises, and a "no" answer when its denial follows from the premises, has a serious flaw: it will run on forever for some problems. Conversely, any QA algorithm that is guaranteed to halt for all problems has the opposite flaw: it will sometimes fail to find an answer when in fact the premises entail a "yes" answer or a "no" answer.

So in practice, we interleave proof and disproof attempts, and limit the overall attempt by some reasonable resource bounds. But note that there are sometimes better methods than waiting to run out of time or space for determining that a question is unanswerable. For example, suppose we have a KB that has absolutely no occurences of the predicate "Loves" in it, and we ask the question "Loves(Romeo, Juliet)?". Then we should immediately answer "I don't know", since no amount of theorem proving will be able to derive a contradiction from either Loves(Romeo, Juliet) or from ¬Loves(Romeo, Juliet) – there's just nothing to resolve against either of these literals. In our later discussion of proof strategies, we'll see a more general method for discovering such cases, as part of the *connection graph* method.

Green's method for *wh-* questions

To see how variable-binding comes into play in deductively answering *wh*-questions, let's look at a very simple example. Suppose we are given the following premises and question:

Given: 1. ∀x. At(Bob,x) ⇒ At(Carol,x) (Carol is wherever Bob is)
2. At(Bob,Dance) (Bob is at the dance)
Question: Where is Carol?

Note that we can view this question equivalently as the 'problem' of determining where Carol is. And since we solve problems deductively by proving that a solution exists, our goal is to prove the following

Theorem. There exists a place where Carol is.

Denial: ¬∃x. At(Carol,x), i.e., ∀x. ¬At(Carol,x)

What we'll do is to prove this theorem in the usual way, using a resolution refutation. But then we'll add something to the proof (shown in small script below): we add a literal Ans(x) to the denial of the conclusion (where x is the 'questioned' variable); then we carry this extra literal through the proof, making whatever substitutions for x that are made in the proof. Then Ans(...) will appear at the end of the proof attached to the empty clause, with the dots containing a term that answers the question:

1.	¬At(Bob,x) ∨ At(Carol,x)	Given
2.	At(Bob,Dance)	Given
3.	¬At(Carol,x) ∨ Ans(x)	Denial of conclusion
4.	At(Carol,Dance)	r[1a,2]
5.	□∨ Ans(Dance)	r[3,4]

Thus we have obtained the answer to the question "Where is Carol?", namely at "Dance".

A subtle point about answer extraction is that different proofs can give different answers, and some answers are more informative than others. To see this, consider the following premises and question:

Given: 1. $\forall x\, \exists y$. Works-for(x,y) Everyone works for someone
2. Works-for(Bob,Carol) Bob works for Carol
Question: $\exists$x. Works-for(Bob,x) Who does Bob work for?
Denial: $\neg\exists$x. Works-for(Bob,x), i.e., $\forall$x. $\neg$Works-for(Bob,x)

First proof (with answer extraction):

1.	Works-for(x,f(x))	Given
2.	Works-for(Bob,Carol)	Given
3.	$\neg$Works-for(Bob,x) $\vee$ Ans(x)	Denial of conclusion
4.	$\Box\vee$ Ans(f(Bob))	r[1,3]

So the answer with this proof is that Bob works for f(Bob), which we can see is just a Skolem function term expressing "the individual Bob works for"! This provides no useful information at all. On the other hand, there is clearly a second proof (also a 1-step proof), differing from the first in that instead of 4 we have

4'. $\Box\vee$ Ans(Carol) r[2,3]

This is much better – we have obtained the answer that is in the knowledge base. From this we may reasonably draw the conclusion that it's best to avoid getting Skolem constants or functions in the answer extraction process. How can we do this? Well, we can simply avoid doing any resolutions that substitute a Skolem constant or function for a question variable (i.e., an Ans-variable).

This will not prevent us from finding a proof, if a proof that yields a Skolem-free answer exists at all. The reason for this is that once any constant or function has been substituted into an Ans-variable, there is no way that further substitutions can get rid of the constant or function. So no proof that ends with a Skolem-free answer can possibly involve any substitutions of Skolem constants or functions into variables within the Ans-argument(s).

However, it may be that the only proof there is is one that involves substitution of Skolem constants or functions into Ans-variables. So if we disallow such proof steps, we may prevent discovery of a successful refutation, even though one exists. Besides, a Skolem answer is not necessarily as uninformative as the one above. For instance, consider the 'paranoia' example again. (As an exercise, you should go back and make the additions to the proof needed to answer the question "Whom does every nonparanoid dislike?"). It turns out that the answer you get is the Skolem constant for the paranoid individual who is known to exist. We would feel better about this answer if it were an actual (non-Skolem) name for the paranoid individual (e.g., if we had been given that

Bob is a paranoid, rather than just that some paranoid exists). But still, it may be quite useful to know that the individual in question is a paranoid. In general, the usefulness of an extracted answer seems to be greatest if it contains no Skolem terms, but it can also be quite useful if it contains Skolem terms, and we have significant amounts of additional knowledge about those Skolem terms.

We postpone looking at more interesting examples of question answering than the ones above until we have added *paramodulation* to our inference rules, allowing us to deal with premises and conclusions involving equality.

Using paramodulation to reason with equalities

To obtain first-order logic, rather than just the first-order predicate calculus (FOPC), we need to allow for equality. This is important not just for expressing mathematical facts, but for ordinary, commonsense knowledge. For instance, consider the statement

Beethoven was the only composer who was deaf.

This expresses several facts: that Beethoven was a composer, that he was deaf, and that he was the *only* individual with those properties. The third fact is the one of particular interest here. We can paraphrase it as saying that every deaf composer is identical with Beethoven. So the logical content of the above sentence is as follows:

$\mathrm{C(B)} \wedge \mathrm{D(B)} \wedge \forall \mathrm{x}.\ \mathrm{C(x)} \wedge \mathrm{D(x)} \Rightarrow (\mathrm{x} = \mathrm{B})$.

Another example of a statement that implicitly involves equality is the following:

John respects everyone but himself.

In other words, for every individual not identical with himself, John respects that individual, but he doesn't respect himself:

$(\forall \mathrm{x}.\ \neg(\mathrm{x} = \mathrm{John}) \Rightarrow \mathrm{Respects(John,x)}) \ \wedge\ \neg\mathrm{Respects(John,John)}$

It is possible to handle equality without any new inference rules, but then we need various equality axiom schemas, most importantly one of form

$$\neg(\sigma = \tau) \vee \neg\lambda_i \vee ((\lambda_1)_{\sigma/\tau} \vee ... \vee (\lambda_n)_{\sigma/\tau}),$$

where $\lambda_1 \vee ... \vee \lambda_n$ is any clause. This is easily derived from the equality schema $(\sigma = \tau) \wedge \phi \Rightarrow \phi_{\sigma/\tau}$ we saw previously in an axiomatic system for FOL.

But use of such a schema would greatly alter the character of resolution proofs. As it turns out, it is much more natural to use another inference rule, called *paramodulation* instead of the above schema. Basically, this a rule allowing substitution of equals, but the rule fits in very naturally with resolution in the way it uses unification and two parent clauses.[1]

[1]Here's a small challenge. It's possible to derive another schema from the one above, using resolution with $\lambda_1 \vee ... \vee \lambda_n$ and factoring. This schema can then be seen to give the same result as paramodulation when used to resolve against a clause containing a positive equality atom unifiable with $\sigma = \tau$.

In words, paramodulation works like this: Given a clause containing a positive equality literal $\sigma = \tau$,

- we unify σ with some term in another clause;
- we apply the unifier γ throughout both clauses;
- we substitute $\tau\gamma$ (i.e., τ after application of the unifier) for the term we unified with σ; and
- we infer the clause that consists of all the literals in the two clauses (after application of the unifier and substitution of $\tau\gamma$ for $\sigma\gamma$ in the second clause), *except* for the equality literal.

We say that the first clause, containing the equality, has been "paramodulated into" the second clause. The conclusion is called the *paramodulant*. Another, equally valid way to paramodulate is with the roles of σ and τ interchanged (i.e., we unify τ with a term in another clause, etc.) Here's a simple example of paramodulation:

Paramodulate 13. f(A,B)=P $\vee$ C(A) $\vee$ C(B) into 6. D(P):

14. D(f(A,B)) $\vee$ C(A) $\vee$ C(B) p[13a$_R$,6]

If clause 6 had additional literals, these would also appear in the paramodulant, 14. The clause numbers used here just anticipate an upcoming example of a complete resolution/paramodulation proof, containing the above step. Note the way the paramodulation step is indicated on the right: it says "paramodulate the RHS (R) of the first literal (a) of clause 13 into clause 6". If clause 6 had additional literals we would also use a, b, c, ... to indicate which literal contains the term we substituted for.

At this point let's state the paramodulation rule formally:

Paramodulation:

$$\frac{\sigma = \tau \vee \alpha_1 \vee ... \vee \alpha_m, \quad \beta_1(\sigma') \vee ... \vee \beta_n}{\alpha_1'' \vee ... \vee \alpha_m'' \vee \beta_1''(\tau'') \vee ... \vee \beta_n''}$$

where $\beta_1(\sigma')$ denotes a literal β_1 with an occurrence of a term σ' somewhere within it; σ and σ' are unifiable, with unifier γ; $\tau'' = \tau\gamma$ (i.e., τ after application of the unifier γ); and other double-primed symbols denote corresponding unprimed literals after application of the substitution γ. Of course neither the equality literal $\sigma = \tau$ nor the literal β_1 need to be the first ones in their respective clauses; and the equality literal could equally well be $\tau = \sigma$.

The paramodulation rule is not quite enough, by itself, for completeness (when combined with wide resolution, or narrow resolution plus factoring). In general, we also need to add the following simple self-identity axiom to our premises:

x = x

Some books also say (and prior versions of these notes said) that you also need, for each n-place function constant f occurring in the clauses to be refuted,

$f(x_1,...,x_n) = f(x_1,...,x_n)$.

But while this issue was unresolved for some years, it was first shown in 1975 (by D. Brand *et al.*) that the functional identities are superfluous. Here's one more example of paramodulation before we look at a complete resolution/paramodulation proof.

Paramodulate clause 1 into 2, p[1a$_L$,2a]:

1. f(x,B)=C $\vee$ P(x,y)
2. Q(f(A,z)) $\vee$ R(w,z) $\vee$ S(f(A,B))

The unifier of f(x,B) and f(A,z) is (x/A)(z/B). Applying this substitution to 1 and 2 gives

1'. f(A,B)=C $\vee$ P(A,y)

2'. Q(f(A,B)) $\vee$ R(w,B) $\vee$ S(f(A,B))

Replacing the first occurrence of f(A,B) in 2' by C gives

2". Q(C) $\vee$ R(w,B) $\vee$ S(f(A,B)) (*N.B:* Not a valid inference by itself!!)

The paramodulant consists of all the literals in 1' and 2" except the equality literal in 1':

3. P(A,y) $\vee$ Q(C) $\vee$ R(w,B) $\vee$ S(f(A,B)) p[1a$_L$,2a]

In a proof we would do this in one step, perhaps supplying the unifier for clarity.

Example: The princess and the pea. This example is from Earl Hunt's book, *Artificial Intelligence*.

1. M(A)	A is a man
2. $\neg$M(B)	B is a woman (ungallant but convenient formulation!)
3. $\neg$M(x) $\vee$ C(x) $\vee$ x=K	If x is a man & not a commoner, he is the king
4. M(y) $\vee$ C(y) $\vee$ y=Q	If y is a woman & not a commoner, she is the queen
5. f(K,Q) = P	The daughter of the king & queen is the princess
6. D(P)	The princess can detect a pea under 20 featherbeds

To be proved: If neither A nor B are commoners, then the daughter of A and B can detect a pea under 20 featherbeds.

The denial of this, in words, is: Neither A nor B are commoners, but the daughter of A and B cannot detect a pea under 20 featherbeds. This clearly leads to the following 3 unit clauses:

7. $\neg$C(A)
8. $\neg$C(B)
9. $\neg$D(f(A,B))

Refutation:

10.	$C(A) \vee A{=}K$	r[1,3a]
11.	$C(B) \vee B{=}Q$	r[2,4a]
12.	$f(A,Q) = P \vee C(A)$	p[10b$_R$,5]
13.	$f(A,B) = P \vee C(A) \vee C(B)$	p[11b$_R$,12]
14.	$D(f(A,B)) \vee C(A) \vee C(B)$	p[13a$_R$,6]

Clearly at this point 3 resolution steps using 7, 8, 9 *versus* 14 will yield $\square$.

Example: Unrequited love The premises are as follows (already numbered to indicate how many clauses result from each statement):

1. Everyone loves someone
2. Anyone whose love is not returned is not happy (i.e., anyone who loves someone who does not love him/her is not happy).

3, 4. Al loves Bonnie but not Diane.

5. Bonnie loves at most one person (i.e., any x and y that she loves must be identical).

6, 7. Clyde loves Bonnie and Diane.

8. Clyde is happy.

Question:

9. Who is unhappily in love with whom?
   (i.e., Who loves whom, and is unhappy?)

Logical translations:

1.	$\forall x\, \exists y.\ L(x,y)$
2.	$\forall x\, \forall y.\ (L(x,y) \wedge \neg L(y,x)) \Rightarrow \neg H(x)$
3, 4.	$L(A,B),\ \neg L(A,D)$
5.	$\forall x\, \forall y.\ (L(B,x) \wedge L(B,y)) \Rightarrow x = y$
6, 7.	$L(C,B),\ L(C,D)$
8.	$H(C)$

Existence statement: Someone is unhappy and loves someone, i.e.,

9. $\exists x\, \exists y.\ \neg H(x) \wedge L(x,y)$

Clause form:

1. $L(x,f(x))$
2. $\neg L(x,y) \vee L(y,x) \vee \neg H(x)$
3. $L(A,B)$
4. $\neg L(A,D)$
5. $\neg L(B,x) \vee \neg L(B,y) \vee x = y$
6. $L(C,B)$
7. $L(C,D)$
8. $H(C)$
9. $H(x) \vee \neg L(x,y) \vee \mathrm{Ans}(x,y)$

First proof:

10.	$\neg$L(B,z) $\vee$ f(B) = z	r[1,5a]
11.	$\neg$L(C,y) $\vee$ L(y,C)	r[8,2c]
12.	L(B,C)	r[6,11a]
13.	f(B) = C	r[10a,12]
14.	L(B,A) $\vee \neg$H(A)	r[3,2a]
15.	f(B) = A $\vee \neg$H(A)	r[10a,14a]
16.	C = A $\vee \neg$H(A)	p[13_L,15a]
17.	$\neg$L(C,D) $\vee \neg$H(A)	p[16_R,4]
18.	$\neg$H(A)	r[7,17]
19.	$\neg$L(A,y) $\vee$ Ans(A,y)	r[18,9a]
20.	$\square \vee$ Ans(A,B)	r[19,3]

Second proof (without using 1!):