
Automated Theorem Proving: Course Content

- Introduction
 - What is ATP?
 - Overview of ATP (Local copy)
 - What is ATP?
 - Logical Consequence
 - Propositional logic
 - The Language
 - Logical Consequence by Truth Tables
 - Satisfiability Preserving Transformations
 - 1st order logic
 - The Language
 - Clause Normal Form
 - Herbrand Interpretation
 - Resolution
 - The Resolution Procedure
 - The ANL Loop
 - Using ATP Systems
 - General Purpose Improvements
 - Heuristics
 - Restrictions
 - Prolog
 - Linear Input Resolution
 - Introduction
 - Data
 - Control
 - Meta-programming
 - Refinements of the Resolution Procedure – Saturation Based
 - Syntactic Refinements
 - Semantic Refinements
 - Equality Reasoning
 - Paramodulation
 - Superposition
 - Knuth–Bendix completion, etc
 - Refinements of the Resolution Procedure – Goal Oriented
 - Linear Refinements
 - Chain Format Linear Refinements
-

What is Automated Theorem Proving?

The basic problem that is dealt with by ATP is as follows:

Given a set of *axiom formulae* in some language, determine whether another formula is a *logical consequence* of the set.

Roughly, a *formula* is a statement about a domain of application. Statements are either *TRUE* or *FALSE*.

Example
A = { All men are mortal, Socrates is a man } C = Socrates is mortal

Here the set of axioms formulae A leads to the conclusion C . The question to ask is "Is that conclusion reasonable, or logical?". The answer normally given is "Yes", for one or both of two reasons. The first reason is that it is known that Socrates is a Greek philosopher who lived about 469–399BCE, and it is known that everyone from that era is now dead. The second reason is that the argument sounds reasonable. The first reason uses the meaning (formally, the *semantics*) of the formulae to motivate the reasonableness of the conclusion. The second reason is *syntactic*, i.e., it does not rely on the meaning of the formulae. It is this second reason that is relevant to ATP. In ATP words, it is said that the conclusion `Socrates is mortal` is a logical consequence of the axioms `All men are mortal` and `Socrates is a man`.

The important phrase in the above is "logical consequence", indicating a logical argument form. Logical consequence will be formally defined later, but roughly, logical consequence means that the conclusion is TRUE regardless of the meanings of the words used – the conclusion follows inevitably from the axioms. For example, given that A implies B , and you know that A is true, you can logically conclude that B is true, even though A and B have unknown meaning. B is a logical consequence of A implies B , and A . This is acceptable to everyone who knows the meaning of "implies", regardless of what A and B mean. A could mean "Being a good student" and B "Will pass exams", or alternatively A could mean "Being extremely tired" and B "Likely to sleep". For either meaning (or any other meaning) B is still TRUE. In fact, the statements don't have to mean anything, and B is still TRUE.

ATP is about establishing logical consequence, using a computer. From the point of view of a computer scientist, ATP is the study and development of computer programs that build proofs of theorems. These programs are called ATP systems. The inputs to an ATP system are axioms (including hypotheses) and the theorem statement, and the output is a proof that shows that the theorem statement is a logical consequence of the axioms. Computers have no idea about the meaning of statements in English or any other language we use. As logical consequence is independent of meaning, a computer is quite adequate as a tool for establishing logical consequence.

For an ATP system to be at all useful, it must not be possible for the system to 'prove' non-logical consequences. Every proof found from a set of axioms must be a proof of a logical consequence of those axioms. This property, called *soundness* is discussed in the context of specific ATP techniques later. As well as being sound, it is desirable that ATP systems be *complete*. This means that the system is able to find a proof of all logical consequences of a set of axioms. Again, this property is discussed for specific cases later.

Automated Theorem Proving

What is Automated Theorem Proving?

Automated Theorem Proving (ATP) deals with the development of computer programs that show that some statement (the *conjecture*) is a *logical consequence* of a set of statements (the *axioms* and *hypotheses*). ATP systems are used in a wide variety of domains. For examples, a mathematician might prove the conjecture that groups of order two are commutative, from the axioms of group theory; a management consultant might formulate axioms that describe how organizations grow and interact, and from those axioms prove that organizational death rates decrease with age; a hardware developer might validate the design of a circuit by proving a conjecture that describes a circuit's performance, given axioms that describe the circuit itself; or a frustrated teenager might formulate the jumbled faces of a Rubik's cube as a conjecture and prove, from axioms that describe legal changes to the cube's configuration, that the cube can be rearranged to the solution state. All of these are tasks that can be performed by an ATP system, given an appropriate formulation of the problem as axioms, hypotheses, and a conjecture.

The **language** in which the conjecture, hypotheses, and axioms (generically known as *formulae*) are written is a logic, often classical 1st order logic, but possibly a non-classical logic and possibly a higher order logic. These languages allow a precise formal statement of the necessary information, which can then be manipulated by an ATP system. This formality is the underlying strength of ATP: there is no ambiguity in the statement of the problem, as is often the case when using a natural language such as English. Users have to describe the problem at hand precisely and accurately, and this process in itself can lead to a clearer understanding of the problem domain. This in turn allows the user to formulate their problem appropriately for submission to an ATP system.

The **proofs** produced by ATP systems describe how and why the conjecture follows from the axioms and hypotheses, in a manner that can be understood and agreed upon by everyone, even other computer programs. The proof output may not only be a convincing argument that the conjecture is a logical consequence of the axioms and hypotheses, it often also describes a process that may be implemented to solve some problem. For example, in the Rubik's cube example mentioned above, the proof would describe the sequence of moves that need to be made in order to solve the puzzle.

ATP systems are enormously powerful computer programs, capable of solving immensely difficult problems. Because of this extreme capability, their application and operation sometimes needs to be guided by an expert in the domain of application, in order to solve problems in a reasonable amount of time. Thus ATP systems, despite the name, are often used by domain experts in an interactive way. The interaction may be at a very detailed level, where the user guides the inferences made by the system, or at a much higher level where the user determines intermediate lemmas to be proved on the way to the proof of a conjecture. There is often a synergetic relationship between ATP system users and the systems themselves:

- The system needs a precise description of the problem written in some logical form,
- the user is forced to think carefully about the problem in order to produce an appropriate formulation and hence acquires a deeper understanding of the problem,
- the system attempts to solve the problem,
- if successful the proof is a useful output,
- if unsuccessful the user can provide guidance, or try to prove some intermediate result, or examine the formulae to ensure that the problem is correctly described,
- and so the process iterates.

ATP is thus a **technology** very suited to situations where a clear thinking domain expert can interact with a powerful tool, to solve interesting and deep problems. Potential ATP users need not be concerned that they need to write an ATP system themselves; there are many ATP systems readily available for use. At the 1st order level some well known and successful systems are [Otter](#), [E](#), [SPASS](#), [Vampire](#), and [Waldmeister](#). Higher order systems include [ACL2](#), [Coq](#), [HOL](#), and [Nqthm](#). For examples of problems written in classical 1st order logic, the [TPTP problem library](#) is a useful resource, which also has a simple [interface](#) for users to try out many 1st order ATP systems.

Fields where ATP has been successfully used include logic, mathematics, computer science, engineering, and social science; some outstanding successes are described below. There are potentially many more fields where ATP could be used, including biological sciences, medicine, commerce, etc. The technology is waiting for users from such fields.

What has Automated Theorem Proving been Really Useful for?

Many significant problems have been, and continue to be, solved using ATP. The fields where the most notable successes have been achieved are mathematics, and software generation and verification, protocol verification, and hardware verification.

The most exciting recent success in **mathematics** has been the settling of the Robbins problem by the ATP system [EQP](#). In 1933 Herbert Robbins conjectured that a particular group of axioms form a basis for Boolean algebra, but neither he nor anyone else (until the solution by EQP) could prove this. The proof that confirms that Robbins' axioms are a basis for Boolean algebra was found October 10, 1996, after about 8 days of search by EQP, on an RS/6000 processor. This result was reported in the [New York Times](#).

In the mathematical domain of quasi-groups there have been several successes using ATP systems. Otter has been used by the mathematician Ken Kunen to prove several results in quasi-groups. Fujita, Slaney, and Bennett (the first two being ATP researchers, the last a mathematician) decided many quasi-group problems using a system built at ICOT in Japan. William McCune and assorted colleagues have [used Otter](#) to find minimal axiom sets, find new single axioms, and solve other interesting problems, in a range of algebraic structures. In the higher order setting [NUPRL](#) helped to confirm Higman's lemma and Gerard's paradox, both of which were under active investigation by humans at the time. The specialist geometry prover [Geometry Expert](#), of Chou, Gao, and Zhang, has been used to obtain new results in Euclidean geometry.

Software generation is an economically important real world application of ATP. Although the use of ATP in software generation is in its infancy, there have already been some interesting results. The [KIDS](#) system developed at Kestrel Institute has been used to derive scheduling algorithms that have outperformed currently used algorithms. KIDS provides intuitive, high level operations for transformational development of programs from specifications. The [AMPHION](#) project, sponsored by NASA, is used to determine appropriate subroutines to be combined to produce programs for satellite guidance. By encapsulating usable functionality in software components (e.g. subroutines, object classes), and then reusing those components, AMPHION can develop software of greater functionality in less time than human programmers, with some assurance that the overall system is correct because it is built up from trusted components

Software verification is an obvious and attractive goal, which is slowly being realized using ATP technology. Three examples are given here, while many more are indexed at the [Formal Methods page](#). The [Karlsruhe Interactive Verifier](#) (KIV) was designed as an experimental platform for interactive program verification at the University of Karlsruhe, and has since been used to successfully verify a range of software applications. These include some case studies of academic software, e.g., implementation of set functions, tree and graph representation and manipulation, and verification of a Prolog to WAM compiler. KIV is at the threshold of industrial application, with pilot studies undertaken in various domains, including a software controlled railway switch, safe command transfer in a space vehicle, and supervision of neutron flow in a nuclear reactor. [PVS](#) is a verification system that has been used in various applications, including diagnosis and scheduling algorithms for fault tolerant architectures, and requirements specification for portions of the space shuttle flight control system. [NASA uses ATP](#) to certify safety properties of aerospace software that has been automatically generated from high-level specifications. Their code generator produces safety obligations that are provable only if the code is safe. An ATP system discharges these obligations, and the output proofs, which can be verified by an independent proof checker, serve as certificates.

Security protocol verification techniques analyse protocols that are used to transmit data over networks in a secure fashion, in an attempt to verify or find flaws in the protocol. The SPASS system has been used to [analyse the Neuman-Stubblebine protocol](#), various cryptographic protocols have been [analysed by transformation to Horn clauses](#), the [TAPS](#) system does verification based on invariants, and the [Coral](#) system has been used to find attacks on faulty security protocols.

Hardware verification is the largest industrial application of ATP. IBM, Intel, and Motorola are among the companies that employ ATP technology for verification. A few good examples of the use of ATP systems for hardware verification are listed here, while many more are indexed at the [Formal Methods page](#). The [ACL2](#) system has been used to obtain a proof of the correctness of the floating point divide code for AMD's PENTIUM-like AMD5K86 microprocessor, while ANALYTICA has been used to verify a division circuit that implements the floating point standard of IEEE. [PVS](#) (mentioned above in the context of software verification) has been used to verify a microprocessor for aircraft flight control. The RRL system has verified commercial size adder and multiplier circuits. The [HOL](#) system has been used at Bell Laboratories for hardware verification. [Nqthm](#) has been used to produce a mechanical proof of correctness of the FM9001 microprocessor. [Safelogic](#) is a Swedish company that provides ATP based tools for verification of a system's logical functionality.

Interpretation

In the discussion above, the word "meaning" keeps coming up. It is necessary to formalize what is meant by "meaning", as used in the above informal definition of logical consequence. An *interpretation* defines the meaning of formulae.




An interpretation of a language takes the words used (the syntax) and associates them with the objects that they represent in the wider world. (That such an association is necessary for intelligent action is motivated in Newell and Simon's Turing Award paper [NS76], and that association motivates the use of semantics for search guidance, as mentioned earlier.) An interpretation determines whether statements written in the language are TRUE or FALSE (often abbreviated to T and F).

Interpretation of connectives

Connectives are the words that join together nouns and verbs, e.g., *and*, *or*, *all*. There are 16 binary and 4 unary connectives. In English we have an intuition of their meaning (interpretation). In logic their interpretation is formally defined.

Interpretation of statements

The component parts of statements are given some interpretation, by associating each different part with an object in a set called the *domain* of the interpretation. Statements and connected statements are then interpreted using the interpretations of the component parts and the connectives.

Example	
man	→ 
mortal	→ 
Socrates	→ 
All men are mortal	→ TRUE
Socrates is a man	→ TRUE
Socrates is mortal	→ TRUE
Socrates could fly	→ FALSE

Models, satisfiability, unsatisfiability

There are some definitions that make it easy to discuss the relationships between formulae and interpretations.

An interpretation is a *model* of a formula (set of formulae) iff the formula (each formula in the set) is TRUE in the interpretation. For example, the interpretation above is a model of the formula All men are mortal, and of the set of formulae {All men are mortal, Socrates is a man, Socrates is mortal}.

A formula (set of formulae) is *satisfiable* iff it has a model. For example, the formula All men are mortal, and the set of formulae {All men are mortal, Socrates is a man, Socrates is mortal} are both satisfiable.

A formula (set of formulae) is *unsatisfiable* iff it has no models. For example, the set of formulae {All men are mortal, Socrates is a man, Socrates is not mortal} is unsatisfiable (if you think about it, but it will be proved formally later). Unsatisfiable formulae are also called *contradictions*.

A formula (set of formulae) is *valid* iff every interpretation is a model. For example, the formula Socrates is either mortal or not mortal is a tautology (assuming the usual meaning of "either-or"). Valid formulae are also called *tautologies*, written \models . The example is written \models (Socrates is either mortal or not mortal).

Logical Consequence

Now that the notion of "meaning" has been formalised as interpretation, it is possible to formalize logical consequence – a conclusion that is TRUE regardless of the meaning of the statements used.

An alternative way of looking at logical consequence, as opposed to the conclusion being TRUE regardless of the meaning, is to think of logical consequence as a conclusion that is TRUE for **all** possible meanings. If it is TRUE for all possible meanings then you can ignore the meaning, because it doesn't matter which one you choose. This is the idea used here.

The formal definition of logical consequence is then:

A formula is a *logical consequence* of an axiom set if every model of the axiom set is also a model of the formula.

Notice the use of the word "every". That's what captures the ideas of "all possible meanings".

The form used for writing the axioms and logical consequence formulae is:

Axioms

Logical consequence




If all the axioms above the line are TRUE, then logical consequence below the line is necessarily TRUE, for all possible meanings of the symbols used.

The definition requires access to "every model of the axiom set". Is this possible? Consider again the very first example:



Example
A = { All men are mortal, Socrates is a man }
C = Socrates is mortal

The deduced formula is apparently a logical consequence of the axioms, i.e., the symbols (man, mortal, Socrates) and the statements given can have any interpretation, and the conclusion will still be TRUE in that context.

The intended interpretation (and model of the axiom set) is :

man	→	
mortal	→	
Socrates	→	
All men are mortal	→	TRUE
Socrates is a man	→	TRUE

and it is TRUE that Socrates is mortal. That makes sense, and that's what is intended. That "checks" one model of the axiom set. Another model of the axiom set is:

man	→	
mortal	→	\$\$\$
Socrates	→	
All men are mortal	→	TRUE
Socrates is a man	→	TRUE

Here the formula All men are mortal means every motor car is expensive, and Socrates is a man means a Corvette is a motor car. The conclusion Socrates is mortal means that a Corvette is expensive. Again the conclusion makes sense. That's another model checked. This process of checking models of the axiom set must continue for "every model of the axiom set". In general, there can be an infinite number of models, and so there appears to be a problem here, but this will be discussed and overcome in the context of propositional and 1st order logic, later. However, it is the case that the conclusion Socrates is mortal is a logical consequence of the axioms All men are mortal and Socrates is a man (unless you change what is meant by the connective all). In the argument form:




All men are mortal
Socrates is a man

Socrates is mortal




To contrast with the above, here's an example of non-logical consequence:

Example (of non-logical consequence)
A = { All men are mortal, Socrates is a man }
C = Socrates has a beard

Semantically, this conclusion is correct; Socrates did have a beard. There is indeed an interpretation that is a model of the axioms that is also a model of the conclusion.

man	→	
mortal	→	
Socrates	→	
All men are mortal	→	TRUE
Socrates is a man	→	TRUE
Socrates has a beard	→	TRUE

However, there is another model of the axiom set that is not a model of the conclusion:

man	→	
mortal	→	
Socrates	→	
All men are mortal	→	TRUE
Socrates is a man	→	TRUE

Socrates has a beard → FALSE

Thus it is not the case that "every model of the set is also a model of the formula", and the conclusion is not a logical consequence of the axioms.

Checking for logical consequence

Generically, the process of checking for logical consequence can be thought of as checking rows of a table. There is one row of the table for each possible interpretation of the formulae, and a column for each axiom formula and a column for the conclusion formula. Then, for each row which is a model of all the axioms columns, it is necessary to check that the row is a model of the conclusion column. If this check always succeeds then the conclusion is a logical consequence of the axioms.

Interpretation	Axiom 1	Axiom 2	...	Conclusion
1	Model	Model	all models	Model
2	Model	Model	all models	Model
3	Model	Not model	...	Doesn't matter
4	Model	Model	all models	Model
5	Not model	Doesn't matter
...
?	Model	Model	all models	Model

Note that it is necessary to check the conclusion column only for those rows that are models of all the axiom columns. Other rows are not relevant to this process.

If one or more rows are models of the axioms but are not models of the conclusion, then the conclusion is not a logical consequence of the axioms.

Interpretation	Axiom 1	Axiom 2	...	Conclusion
1	Model	Model	all models	Model
2	Model	Model	all models	Model
3	Model	Not model	...	Doesn't matter
4	Model	Model	all models	Not model
5	Doesn't matter			
...	Doesn't matter			
?	Doesn't matter			

If the axiom set has no models (i.e., it is unsatisfiable) then vacuously every formula is a logical consequence of the axiom set. (Relevance logics are designed to avoid this paradox.) In practical application of ATP it is sensible to check that axiom sets used are not unsatisfiable. This is done by building a model of the axiom set, possibly using a model generation ATP system (see later ACK HAVE TO WRITE THIS).

Interpretation	Axiom 1	Axiom 2	...	Conclusion
1	Model	Not model	...	Doesn't matter
2	Not model	Doesn't matter
3	Model	Not model	...	Doesn't matter
...
?	Model	Not model	...	Doesn't matter

Logical Consequence via Unsatisfiability

The method for checking for logical consequence given above requires checking that each model of the axioms is a model of the conclusion. An equivalent method is to check that every model of the axioms is not a model of the negation of the conclusion. If the conclusion is a logical consequence, then no interpretation will be a model of both the axioms and the negated conclusion. That is, the set consisting of the axioms and the negated conclusion has no models, i.e., is unsatisfiable. This then provides an alternative method for checking that a conclusion c is a logical consequence of a set \mathcal{A} :

A formula is a logical consequence of an axiom set iff every model of the axiom set is not a model of the negation of the formula, i.e., if $\mathcal{A} \cup \{\sim c\}$ has no models, i.e. if $\mathcal{A} \cup \{\sim c\}$ is unsatisfiable.

Interpretation	Axiom 1	Axiom 2	...	\sim Conclusion
1	Model	Not model	...	Not model
2	Not model	Model	...	Not model

3	Model	Model	Not model	Not model
4	Model	Not model	...	Not model
5	Not model	Not model
...	Not models	Not models
?	Not model	Model	all models	Not model

Exam Style Questions

1. Define the following terms used in classical logic:
 - o Interpretation
 - o Model
 - o Satisfiable
 - o Unsatisfiable
 - o Logical consequence
-

Logics

All the preceding discussion about logical consequence has been very generic. No particular language or syntax has been specified. In ATP, problems are expressed in some formal language. Most commonly the problems are expressed in a logic, ranging from classical propositional logic to more exotic logics, such as modal and temporal logics. Current research in ATP is dominated by the use of classical logic, at the propositional and 1st order levels. These logics, and proof within these logics, are well understood and documented.

Propositional (0th order) Logic

Propositional logic is a simple and well known language for representing knowledge. It is very simple to test for logical consequence in propositional logic, as is shown below.

The Language of Propositional logic

The syntax of propositional logic is most easily introduced through an example.

Example
A = { If I am clever then I will pass, If I will pass then I am clever, Either I am clever or I will pass }
C = I am clever and I will pass

The conclusion that I am clever and I will pass is a logical consequence of the axioms. The example is written in English, but it is easily translated into propositional logic. There are two propositions in the example: I am clever and I will pass. In propositional logic, these propositions can be represented by their text. Using the Prolog convention of starting propositions with lowercase alphabetic, the axiom set and conclusion become :

```
A = { If i_am_clever then i_will_pass,  
      If i_will_pass then i_am_clever,  
      Either i_am_clever or i_will_pass }  
C = i_am_clever and i_will_pass
```

To remove the if-then and other English words, connectives are used. The commonly used connectives of propositional logic are:

- \sim for negation. This is "not" in English. Negation takes a single formula as its argument, e.g., $\sim i_am_clever$, and is thus a unary connective.
- \wedge or $\&$ for conjunction This is "and" in English. Conjunction is an infix binary connective, taking two formulae as arguments, e.g., $i_am_clever \& i_will_pass$.
- \vee or $|$ for disjunction This is also known as "inclusive or", and corresponds to some uses of "or" in English. Disjunction is also infix binary, e.g., $i_am_clever | i_will_pass$.
- \Rightarrow for implication. This corresponds to the English if-then construction. Implication is binary infix. The left hand operand is called the antecedent, and the right hand operand is called the consequent. E.g., $i_am_clever \Rightarrow i_will_pass$.
- \Leftrightarrow for equivalence. This is also known as double implication, and has the meaning of "if and only if" in English. Equivalence is binary infix, e.g., $i_am_clever \Leftrightarrow i_will_pass$.

There are other less common connectives, giving a total of four unary connectives and 16 binary ones. Any good introductory text on logic, e.g., [Chu56] will provide details. For ATP, the above are adequate.

Denoting arbitrary propositional formulae by uppercase variable letters, e.g., P , propositional formulae are defined recursively by:

- Propositions are formulae.
- If P and Q are formulae then $\sim P$, $P \& Q$, $P | Q$, $P \Rightarrow Q$, $P \Leftrightarrow Q$, and (P) are formulae.

The precedence order of the above operators is $\sim | \& \Rightarrow \Leftrightarrow$, i.e., \sim binds most tightly, down to \Leftrightarrow . This precedence ordering allows some brackets to be omitted, e.g., $\sim a | b \Rightarrow c$ means $((\sim a) | b) \Rightarrow c$.

Now the above example can be written in propositional logic as follows:

Example
A = { $i_am_clever \Rightarrow i_will_pass$, $i_will_pass \Rightarrow i_am_clever$, $i_am_clever i_will_pass$ }
C = $i_am_clever \& i_will_pass$

Interpretation of Propositional Logic

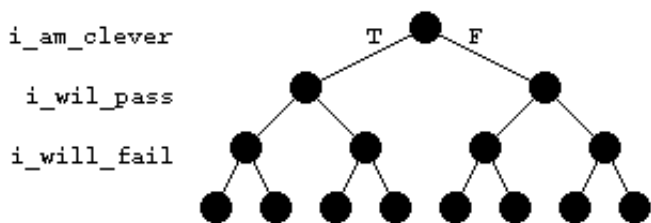
Recall that an interpretation assigns truth values (TRUE and FALSE) to statements, and possibly values to component parts of the statements. In

propositional logic there are no "component parts", only statements in the form of propositional formulae, e.g., i_am_clever and $i_am_clever \Rightarrow \sim i_will_pass$. For propositional logic, interpretation consists of assigning truth values to propositions and defining how truth values are combined by the connectives. The manner in which the connectives are interpreted is globally agreed on, as described later. The assignment of truth values to the propositions is determined by the user.

The number of possible assignments of truth values to N propositions, and thus the number of possible interpretations of N propositions, is 2^N . All the interpretations can be captured in a truth table with 2^N rows. For a language that contains three propositions, i_am_clever , i_will_pass , and i_will_fail , there are eight possible interpretations:

Example			
	i_am_clever	i_will_pass	i_will_fail
1	T	T	T
2	T	T	F
3	T	F	T
4	T	F	F
5	F	T	T
6	F	T	F
7	F	F	T
8	F	F	F

An alternative way of representing all the interpretations is as a semantic tree:



The following table defines how truth values are combined with connectives:

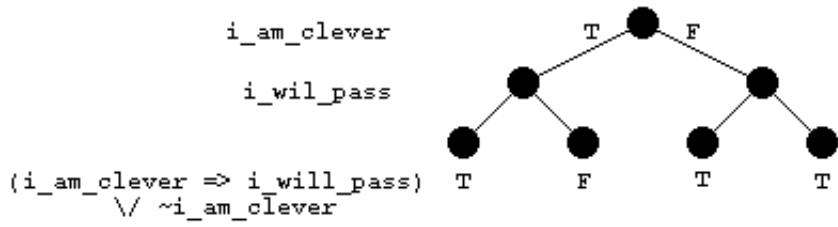
Truth table for connectives						
P	Q	$\sim P$	$P \& Q$	$P \mid Q$	$P \Rightarrow Q$	$P \Leftrightarrow Q$
T	T	F	T	T	T	T
T	F	F	F	T	F	F
F	T	T	F	T	T	F
F	F	T	F	F	T	T

The definitions for negation, conjunction, and disjunction are common, and correspond directly to their English meanings. Equivalence is also quite straight forward. The definition for implication is less intuitive (to the English speaker). There's a simple example that helps to remember the definition: A politician says "If I am elected, then I will reduce taxes" ($i_am_elected \Rightarrow i_will_reduce_taxes$). If the politician is elected, and he does reduce taxes, then he didn't lie ($TRUE \Rightarrow TRUE$ is $TRUE$). If he does not get elected, than it does not matter about his promise ($FALSE \Rightarrow TRUE$ and $FALSE \Rightarrow FALSE$ are both $TRUE$). Only if he is elected and he does not reduce taxes is the politician a liar ($TRUE \Rightarrow FALSE$ is $FALSE$).

Given an interpretation, a truth table or semantic tree can be used to calculate the truth value of a propositional formula for all possible interpretations.

Example				
$I = \{ i_am_clever \Rightarrow TRUE, i_will_pass \Rightarrow FALSE \}$ $F = (i_am_clever \Rightarrow i_will_pass) \mid \sim i_am_clever$				
i_am_clever	i_will_pass	$i_am_clever \Rightarrow i_will_pass$	$\sim i_am_clever$	$(i_am_clever \Rightarrow i_will_pass) \mid \sim i_am_clever$
T	T	T	F	T
T	F	F	F	F
F	T	T	T	T
F	F	T	T	T

Represented using a semantic tree:



Exam Style Questions

1. How many possible interpretations are there of statements written in propositional logic?
 2. What is a semantic tree?
-

Logical Consequence in Propositional Logic

Logical consequence in propositional logic can be established using truth tables to check that "every model of the axiom set is a model of the formula".

Example						
$A = \{ i_am_clever \Rightarrow i_will_pass, \\ i_will_pass \Rightarrow i_am_clever, \\ i_am_clever \mid i_will_pass \}$ $C = i_am_clever \ \& \ i_will_pass$						
i_am_clever	i_will_pass	$i_am_clever \Rightarrow i_will_pass$	$i_will_pass \Rightarrow i_am_clever$	$i_am_clever \mid i_will_pass$	A	$i_am_clever \wedge i_will_pass$
T	T	T	T	T	T	T
T	F	F	T	T	F	F
F	T	T	F	T	F	F
F	F	T	T	F	F	F

It is simply necessary to check that each row that is a model of the axiom set, i.e., in those rows where A is TRUE the conclusion is also TRUE. In this example only the first row needs to be checked. In general, the maximal number of rows to be checked is 2^N , where N is the number of propositions in the language. Thus this constitutes a simple decision procedure for logical consequence in propositional logic.

To avoid accidental use of the intended meaning of the propositions, the propositions can be reduced to meaningless symbols. This indicates that the conclusion follows from the axioms, regardless of the meaning of the symbols, i.e., it is a logical consequence. The truth table is the same as before:

Example						
$A = \{ p \Rightarrow q, \\ q \Rightarrow p, \\ p \mid q \}$ $C = p \ \& \ q$						
p	q	$p \Rightarrow q$	$q \Rightarrow p$	$p \mid q$	A	$p \ \& \ q$
T	T	T	T	T	T	T
T	F	F	T	T	F	F
F	T	T	F	T	F	F
F	F	T	T	F	F	F

Another example to illustrate logical consequence in propositional logic, using meaningless propositions:

Example							
$A = \{ q \mid r, \\ q \Rightarrow \sim p, \\ \sim(r \ \& \ p) \}$ $C = \sim p$							
p	q	r	$q \mid r$	$q \Rightarrow \sim p$	$\sim(r \ \& \ p)$	A	$\sim p$
T	T	T	T	F	F	F	F
T	T	F	T	F	T	F	F
T	F	T	T	T	F	F	F
T	F	F	F	T	T	F	F
F	T	T	T	T	T	T	T
F	T	F	T	T	T	T	T
F	F	T	T	T	T	T	T
F	F	F	F	T	T	F	T

Each of the three models of the axiom set is also a model of the conclusion, i.e., the conclusion is a logical consequence of the axiom set.

The same example, illustrating logical consequence in propositional logic by unsatisfiability:

Example

$A = \{ q \mid r, q \Rightarrow \sim p, \sim(r \ \& \ p) \}$ $C = \sim p$							
p	q	r	$q \mid r$	$q \Rightarrow \sim p$	$\sim(r \ \& \ p)$	A	$A \cup \{\sim C\}$
T	T	T	T	F	F	F	F
T	T	F	T	F	T	F	F
T	F	T	T	T	F	F	F
T	F	F	F	T	T	F	F
F	T	T	T	T	T	T	F
F	T	F	T	T	T	T	F
F	F	T	T	T	T	T	F
F	F	F	F	T	T	F	F

None of the interpretations are models of $A \cup \{\sim C\}$, i.e. $A \cup \{\sim C\}$ is unsatisfiable and the conclusion is a logical consequence of the axiom set.

Another example to illustrate logical consequence of an unsatisfiable axiom set:

Example								
$A = \{ p \Rightarrow q, q \Rightarrow p, p \mid q, p \Rightarrow \sim q \}$ $C = \sim r$								
p	q	r	$p \Rightarrow q$	$q \Rightarrow p$	$p \mid q$	$p \Rightarrow \sim q$	A	$\sim r$
T	T	T	T	T	T	F	F	F
T	T	F	T	T	T	F	F	T
T	F	T	F	T	T	T	F	F
T	F	F	F	T	T	T	F	T
F	T	T	T	F	T	T	F	F
F	T	F	T	F	T	T	F	T
F	F	T	T	T	F	T	F	F
F	F	F	T	T	F	T	F	T

There are no models of the axiom set, so every model of the axiom set is a model of the conclusion, i.e., the conclusion is a logical consequence of the axiom set. Intuitively, it seems unreasonable to conclude $\sim r$ from the axiom set, as r is not even mentioned in the axioms.

Here's an example where the formula is not a logical consequence of the axiom set:

Example (of non-logical consequence)				
$A = \{ p \Rightarrow q, q \}$ $C = p \ \& \ q$				
p	q	$p \Rightarrow q$	A	$p \ \& \ q$
T	T	T	T	T
T	F	F	F	F
F	T	T	T	F
F	F	T	F	F

The third interpretation is a model of the axiom set, but is not a model of the conclusion. The conclusion is thus not a logical consequence of the axiom set.

From the observation that the truth table for any propositional logic contains 2^N rows for N propositions, it is clear that the complexity of determining whether or not a conclusion is a logical consequence of an axiom set is $O(E)$. Note that it is always possible to decide whether or not a conclusion is a logical consequence of a set of axioms, i.e., the problem is *decidable*. In fact the problem can be shown to be NP-complete [REF].

Exercises

Determine if each F is a logical consequence of the Axioms. Do some directly and some using unsatisfiability.

Axioms	F	F
a a \Rightarrow b	b	
p \Rightarrow q \sim q p p q	p & q	
p \Rightarrow q p q	p & q	
p q r r \Rightarrow (p q) (q & r) \Rightarrow p \sim p \vee q \vee r	q & (r \Rightarrow p)	q r
p \Rightarrow q \sim q p p q \sim (p & q)	(\sim q \Rightarrow p) \Rightarrow (q \sim p)	
q p q \Rightarrow r (s & r) \Rightarrow t t \Rightarrow p	p	
q p p \Rightarrow q q \Rightarrow (\sim r p) r	p	
\sim n \sim t m q n m \Rightarrow l q \Rightarrow l \sim l \sim p r p n r \Rightarrow \sim l	\sim t	

Exam Style Questions

- Use a truth table to show that $(q \Rightarrow p) \mid \sim(q \Rightarrow (p \mid r))$ is a logical consequence of the set $\{ p \mid q \mid r, r \Rightarrow (p \mid q), (q \& r) \Rightarrow p, \sim p \mid q \mid r \}$
- Use a truth table to prove that $\sim p$ is a logical consequence of the set $\{ q \mid r, q \Rightarrow \sim p, \sim(r \& p) \}$.
- Use a truth table to show that $p \& q$ is not a logical consequence of the set $\{ p \Rightarrow q, q \}$.
- Encode the following scenario in propositional logic, and prove the conclusion:
If the races are fixed or the gambling houses are crooked, then the tourist trade will decline. If the tourist trade declines then the police force will be happy. The police force is never happy.
 Conclusion: *The races are not fixed*

Satisfiability Preserving Transformations

One of the techniques for establishing that c is a logical consequence of Δx is to show that $\Delta x \cup \{\neg c\}$ is unsatisfiable. Thus the ability to detect that a set of formulae is unsatisfiable, is important. It may be difficult or impossible to show directly that a set is unsatisfiable. A *satisfiability preserving transformation* (SPT) takes a set of formulae and transforms it to a new set, such that if the original set is satisfiable then so is the transformed set. Conversely, if the transformed set is unsatisfiable then so is the original set.

If, by repetitive use, the application of SPTs produces a set that is obviously unsatisfiable (e.g., containing FALSE) from a set of formulae, then, iteratively, this means that the very original set is also unsatisfiable. If the original set is of the form $\Delta x \cup \{\neg c\}$, this establishes that c is a logical consequence of Δx .

Logical Inference

An inference rule *infers* formulae from parent formulae.

An inference rule is *sound* if it infers only logical consequences of its parents.

The operation of applying a sound inference rule to parents from a set, and then adding the inferred formula back into the set, is a SPT (think why!). (There are other SPTs, but this is an important one.) Thus sound and refutation complete inference rules form the basis of a simple algorithm for establishing logical consequence.

An inference system is *refutation complete* if, by repetitive use, it can infer FALSE (a contradiction) from every unsatisfiable set.

An inference system is *complete* if, by repetitive use, it can infer every logical consequence of its input.

Note: A complete inference system can infer all logical consequences, while a refutation complete inference system can only check (via the unsatisfiability idea) for logical consequence.

Exam Style Questions

1. What does it mean for an inference system to be {sound, complete, refutation complete}? Which of these properties are desirable in an CNF based ATP system?
 2. Explain how unsatisfiability can be used to establish logical consequence in classical logic.
-

Predicate (1st order) Logic

It is relatively easy to check for logical consequence in proposition logic because propositional logic is not very expressive. It is desirable to have a more expressive logic in which to write the axioms and conclusions. A first weakness of propositional logic is that it does not provide variables which can be quantified over, e.g., it is not possible to represent the `all men in All men are mortal` by a variable which can range of the possible values of specific men. Similarly, it is not possible to claim the existence of something with a given property by representing that something by a variable, as in "There exists a prime number greater than 27". A second weakness of propositional logic is the lack of syntax for representing objects in the domain of interest; propositional logic only permits statements about the domain of interest, e.g., it is not possible to have a symbol for `Socrates`, rather it is possible only to make statements about `Socrates`. Associated with this is the absence of functional abstraction, e.g., it is not possible to refer to the brother of `Socrates`.

1st order logic overcomes these two weaknesses of propositional logic by providing a richer language. The cost of this increased expressivity is the loss of decidability for logical consequence. 1st order logic is semi-decidable, which means that if a formula is a logical consequence of a set of axioms, it is possible to show that it is. However, if a formula is not a logical consequence of a set of axioms, it may not be possible to show that it is not. In addition, 1st order logic requires more complex mechanisms for checking for logical consequence.

The Language of 1st order logic

A 1st order language consists of three sets of symbols:

- V is a set of *variables* (for our purposes, infinite)
- F is a set of *functors*, each of which has an *arity*. The arity specifies the number of arguments they take (see below). Functors of arity 0 are often called *constants*.
- P is a set of *predicate symbols*, each of which has an arity. Predicate symbols of arity 0 are propositions.

Prolog syntax is used here: variables start with upper case alphabetic, and functors and predicate symbols start with lower case alphabetic. Functors and predicate symbols are distinguished by the context, as described below. When referring directly to a functor or predicate symbol, its arity is given after a `,` e.g., in `blah/2`, `blah` is the functor or predicate symbol and the arity is 2.

Example
<pre>V = { v : v starts with uppercase } F = { geoff/0, jim/0, brother_of/1 } P = { wise/1, taller/2 }</pre>

From the components of the language, two types of expressions are built. This is in contrast to propositional logic where only propositions exist. The first type of expression is *terms*. Terms are used to denote (possibly arbitrary) objects in the domain of interest. Terms thus correspond roughly to data in conventional programming. Terms are defined recursively by:

- A functor of arity 0 (a constant) is a term
- A variable is a term
- A functor with the appropriate number of terms as arguments, is a term.

Example
<pre>geoff Person brother_of(jim) brother_of(brother_of(X))</pre>

Note that if there is a functor of arity greater than 0, then there is an infinite number of terms. As might be expected, ATP for 1st order logic is much easier if the number of terms is finite (in fact it reduces to ATP for propositional logic). For this reason, it is desirable to formulate ATP problems without any functors of arity greater than 0.

The second type of expression is *atoms*, which correspond to the propositions of propositional logic. Atoms describe relationships between terms (objects in the domain). Atoms are defined by:

- A predicate symbol of arity 0 (a proposition) is an atom
- A predicate symbol with the appropriate number of terms as arguments, is an atom.

Example
<pre>wise(geoff) taller(Person,brother_of(jim)) wise(brother_of(brother_of(X)))</pre>

If the number of terms is infinite (which occurs if there is a functor of arity greater than 0), and there is a predicate symbol of arity greater than 0, then there is an infinite number of atoms.

Connectives are used to combine atoms into the formulae of 1st order logic. The connectives include all those used in propositional logic, and two new *quantifiers*:

- \sim for negation.
- \wedge or $\&$ for conjunction.
- \vee or $|$ for disjunction.
- \Rightarrow for implication.
- \Leftrightarrow for equivalence.
- \forall or $!$ for universal quantification. This corresponds to saying "for all" in English, e.g., $\forall x (\text{man}(x) \Rightarrow \text{mortal}(x))$ says "for all objects X, X is a man implies X is mortal". It is easy to remember that \forall means "for all" by noticing that the \forall is an upside down A from for All.
- \exists or $?$ for existential quantification. This corresponds to saying "there exists" in English, e.g., $\exists x (\text{prime}(x) \& \text{greater}(x,27))$ says "There exists an object X such that X is a prime number and X is greater than 27". It is easy to remember that \exists means "there exists" by noticing that the \exists is a backwards E from there Exists.

Denoting arbitrary 1st order formulae by uppercase variable letters, e.g., P , 1st order formulae are defined recursively by:

- Atoms are formulae
- If P and Q are formulae then $\sim P$, $P \& Q$, $P | Q$, $P \Rightarrow Q$, $P \Leftrightarrow Q$, and (P) are formulae.
- If P is a formula and x is a variable, then $\forall x P$ and $\exists x P$ are formulae. The $\forall x$ and $\exists x$ are called *quantifications* of x , and they quantify all occurrences of x in P . The quantification has precedence over any earlier (to the left) quantification of x . Variables with the same name but which are quantified by different quantifiers, are different variables.

The precedence order of the above operators is $\forall \exists \sim | \& \Rightarrow \Leftrightarrow$, i.e., \forall and \exists bind most tightly, down to \Leftrightarrow . This precedence ordering allows some brackets to be omitted, e.g., $\forall x (\text{clever}(x) \Rightarrow \text{pass}(x) | \text{lazy}(x))$ means $\forall x (\text{clever}(x) \Rightarrow (\text{pass}(x) | \text{lazy}(x)))$. Note that the $()$ s round the $(\text{clever}(x) \Rightarrow \text{pass}(x) | \text{lazy}(x))$ are necessary, for otherwise the quantification would apply to only the $\text{clever}(x)$.

Example
<pre> wise(geoff) ~taller(Person,brother_of(jim)) wise(geoff) & wise(brother_of(brother_of(Person))) ~taller(Person,brother_of(jim)) wise(brother_of(brother_of(Person))) wise(brother_of(brother_of(Person)) => wise(geoff) wise(geoff) <=> ~taller(Person,brother_of(jim)) (wise(geoff) & wise(brother_of(brother_of(Person)))) wise(geoff) forall Person ~taller(Person,brother_of(jim)) exists Person wise(brother_of(brother_of(Person)) => wise(geoff) </pre>

Atoms and the negations of atoms are called *literals*.

Example
<pre> wise(brother_of(X)) ~taller(geoff,jim) </pre>

Ground formulae are formulae that contain no variables.

Example
<pre> wise(geoff) wise(geoff) <=> ~taller(geoff,brother_of(jim)) </pre>

The *scope* of a quantification is the formula which follows the quantification, e.g., the scope of $\exists \text{Person}$ in $\exists \text{Person wise}(\text{brother_of}(\text{brother_of}(\text{Person})) \Rightarrow \text{wise}(\text{geoff}))$ is $\text{wise}(\text{brother_of}(\text{brother_of}(\text{Person})))$. A variable is *bound* if it occurs in a quantification or it is within the scope of a quantification of that variable. Otherwise a variable is *free*. A formulae that has no free variables is a *closed* formula; otherwise a formula is *open*.

Example
<pre> Person is bound in in the closed formula: forall Person ~taller(Person,brother_of(jim)) Person is free in the open formula: ~taller(Person,brother_of(jim)) Person is bound and free in the open formula: forall Person ~taller(Person,jim) wise(Person) Person is bound in the closed formula: forall Person(~taller(Person,jim) wise(Person)) </pre>

A free variable in a expression can be *instantiated* with a value (typically a term), thus *binding* the variable to that value and forming an *instance* of the expression.

Example
<pre> </pre>

The free variable `Person` in:
`~taller(Person, brother_of(jim))`

can be instantiated with `brother_of(geoff)` to form the instance:
`~taller(brother_of(geoff), brother_of(jim))`

For ATP open formulae are not useful, because it is not possible to interpret (in the sense of giving meaning) open formulae. Open formulae can be closed by universal or existential quantification. This is done by placing either a universal or existential quantifier in front of the bracketed formula. The free standing quantifier notation is shorthand for a universal or existential quantification of every free variable in the formula.

Example

`~taller(Person, brother_of(jim)) | wise(brother_of(brother_of(Person)))`

can be closed either universally:

`∀(~taller(Person, brother_of(jim)) | wise(brother_of(brother_of(Person))))`

or existentially:

`∃(~taller(Person, brother_of(jim)) | wise(brother_of(brother_of(Person))))`

In English universal quantification is usually meant if no explicit quantification is given, e.g., "Sheep are stupid" usually means "All sheep are stupid", not "There exists a sheep that is stupid".

Substitutions

A *substitution* θ is a finite set of the form $\{x_1/t_1, \dots, x_n/t_n\}$ where each x_i is a distinct variable and each t_i is a term (or more generally, a member of some given set of objects) distinct from x_i . Each element x_i/t_i is called a binding for x_i . θ may be applied to an formula F to obtain $F\theta$, the expression obtained from F by simultaneously replacing each occurrence of the x_i s in F by t_i . Similarly $S\theta$ may be obtained from a set S of formulae.

A substitution can be applied to another substitution to form their composition. Let $\theta = \{x_1/t_1, \dots, x_n/t_n\}$ and $\sigma = \{y_1/s_1, \dots, y_m/s_m\}$. The variables x_i must be distinct from the variables y_j , and no x_i can appear in a s_j . The composition $\theta\sigma$ is formed by applying σ to the t_i , and combining the two sets. For example, let $\theta = \{P1/jim, P2/brother_of(P4)\}$ and $\sigma = \{P3/brother_of(P5), P4/geoff\}$. Then $\theta\sigma = \{P1/jim, P2/brother_of(geoff), P3/brother_of(P5), P4/geoff\}$.

Translation from English to 1st Order Logic

There is often a direct translation from a problem stated in a natural language to 1st order logic. In order to make such a translation, it is necessary to:

- Determine the predicates and functors of the 1st order language.
 - Objects often correspond to constants
 - Relationships that identify a constant often correspond to functors
 - Properties and activities often correspond to predicates
- Translate the axioms.
- Make appropriate definitions for the predicates and functors.
- Translate the conjecture.

Example

There is a barbers' club that obeys the following three conditions:

- Four of the members are named Guido, Lorenzo, Petrucio, and Cesare.
- If any member A has shaved any other member B - whether himself or another - then all members have shaved A.
- Guido has shaved Cesare.

Prove Petrucio has shaved Lorenzo

$V = \{ v : v \text{ starts with uppercase} \}$
 $F = \{ \text{guido}/0, \text{lorenzo}/0, \text{petrucio}/0, \text{cesare}/0 \}$
 $P = \{ \text{member}/1, \text{shaved}/2, \text{all_shaved}/1 \}$

- Four of the members are named Guido, Lorenzo, Petrucio, and Cesare.

`member(guido)`
`member(lorenzo)`
`member(petrucio)`
`member(cesare)`

- If any member A has shaved any other member B - whether himself or another - then all members have shaved A.

$\forall M1 \forall M2 (\text{member}(M1) \ \& \ \text{member}(M2) \ \& \ \text{shaved}(M1, M2)) \Rightarrow \text{all_shaved}(M1)$

- Guido has shaved Cesare.

```
shaved(guido,cesare)
```

- Definitions

```
 $\forall M1 ( \text{all\_shaved}(M1) \iff ( \text{member}(M1) \ \& \ \forall M2 ( \text{member}(M2) \implies \text{shaved}(M2,M1) ) ) )$ 
```

- Petrucio has shaved Lorenzo.

```
shaved(petrucio,lorenzo)
```

Exercises

Convert the following into 1st order logic:

Wolves, foxes, birds, caterpillars, and snails are animals, and there are some of each of them. Also there are some grains, and grains are plants. Every animal either likes to eat all plants or all animals much smaller than itself that like to eat some plants. Caterpillars and snails are much smaller than birds, which are much smaller than foxes, which in turn are much smaller than wolves. Wolves do not like to eat foxes or grains, while birds like to eat caterpillars but not snails. Caterpillars and snails like to eat some plants. Prove that there is an animal that likes to eat a grain eating animal.

The basic notion of set theory is membership of a set. Using a `member/2` predicate, define set equality, intersection, union, power set, empty set, and difference. For example, intersection is defined as:

```
! [X,A,B] :  
  ( member(X,intersection(A,B))  
<=> ( member(X,A)  
      & member(X,B) ) )
```

Now write conjectures for transitivity of subset, associativity of set intersection, and distribution of intersection over union.

Equality

Many domains require the use of the notion of equality. In 1st order logic equality statements use the `equal/2` predicate, or infix `=/2` and `!=/2` predicates.

Example

- Axioms

```
even(sum(two_squared,b))  
two_squared = four  
 $\forall X ( \text{zero}(X) \implies \text{difference}(four,X) = \text{sum}(four,X) )$   
zero(b)
```

- Conjecture

```
even(difference(two_squared,b))
```

Although the conjecture may seem like a logical consequence to humans, that's because humans "know" what *equal* means (even without knowing what the "maths" means). However, there are many models of the axioms that are not a models of the conjecture, e.g., simply by making `even(sum(four,b))` be `FALSE`. Such models are possible because the axioms are missing definitions for equality. These definitions are the axioms of equality, and must be included to force equality to have its usual meaning. They are:

- Reflexivity – everything equals itself
- Symmetry – If x equals y then y equals x
- Transitivity – If x equals y and y equals z , then x equals z

```
 $\forall X ( X = X )$ 
```

```
 $\forall X \forall Y ( X = Y \implies Y = X )$ 
```

```
 $\forall X \forall Y \forall Z ( ( X = Y \ \& \ Y = Z ) \implies X = Z )$ 
```

- Function substitution – If x equals y then $f(x)$ equals $f(y)$. For every argument position of every functor.
- Predicate substitution – If x equals y and $p(x)$ is `TRUE`, then $p(y)$ is `TRUE` For every argument position of every predicate.

```
 $\forall X \forall Y \forall Z ( X = Y \implies \text{sum}(X,Z) = \text{sum}(Y,Z) )$ 
```

```
 $\forall X \forall Y \forall Z ( X = Y \implies \text{sum}(Z,X) = \text{sum}(Z,Y) )$ 
```

```
 $\forall X \forall Y \forall Z ( X = Y \implies \text{difference}(X,Z) = \text{difference}(Y,Z) )$ 
```

```
 $\forall X \forall Y \forall Z ( X = Y \implies \text{difference}(Z,X) = \text{difference}(Z,Y) )$ 
```

```
 $\forall X \forall Y ( ( X = Y \ \& \ \text{even}(X) ) \implies \text{even}(Y) )$ 
```

```
 $\forall X \forall Y ( ( X = Y \ \& \ \text{zero}(X) ) \implies \text{zero}(Y) )$ 
```

Exercises

Convert the following into 1st order logic, and generate the required axioms of equality:

Osama Bin Laden is a terrorist. If someone is a terrorist and is dangerous, then his leader is a terrorist and is dangerous. There are some terrorists who are their own leader. Dangerous people care about only themselves, or about no-one (not even themselves!).



















Someone who lives in Dreadbury Mansion killed Aunt Agatha. Agatha, the butler, and Charles live in Dreadbury Mansion, and are the only people who live therein. A killer always hates his victim, and is never richer than his victim. Charles hates no one that Aunt Agatha hates. Agatha hates everyone except the butler. The butler hates everyone not richer than Aunt Agatha. The butler hates everyone Aunt Agatha hates. No one hates everyone. Agatha is not the butler. Therefore : Agatha killed herself.

Interpretation

Interpretation of 1st order formulae requires a more complex structure than for propositional logic, because there are "component parts" of 1st order formulae. If the Herbrand base is finite (due to a finite Herbrand universe) then it is possible to assign truth values to elements of the Herbrand base. In this case logical consequence can be decided as for propositional logic, using a truth table. Otherwise, as is the case when the full expressive power of 1st order logic is used, a more complex structure is required. Henceforth only the latter, more complex, 1st order languages are considered.

An interpretation of a 1st order logic consists of three parts:







- D is the domain of the interpretation.
- F is a mapping from functions of domain elements, to the domain.
- R is a mapping from predicates of domain elements, to the truth values TRUE and FALSE.

Example
<p>One possible interpretation of the language</p> <p>V = { v : v starts with uppercase }</p> <p>F = { geoff/0, jim/0, brother_of/1 }</p> <p>P = { wise/1, taller/2 }</p> <p>is</p> <p>D = { ,  }</p> <p>F = { geoff → ,</p> <p style="padding-left: 2em;">jim → ,</p> <p style="padding-left: 2em;">brother_of() → ,</p> <p style="padding-left: 2em;">brother_of() →  }</p> <p>R = { wise() → TRUE,</p> <p style="padding-left: 2em;">wise() → TRUE,</p> <p style="padding-left: 2em;">taller(, ) → FALSE,</p> <p style="padding-left: 2em;">taller(, ) → FALSE,</p> <p style="padding-left: 2em;">taller(, ) → TRUE,</p> <p style="padding-left: 2em;">taller(, ) → FALSE }</p>

As with propositional logic, all the possible RS can be presented as a truth table or a semantic tree.

Given an interpretation, the truth value of a closed formula can be determined, with respect to that interpretation. The procedure for ground formula is:

- Map all constants to the domain.
- Repeatedly map functions of domain elements to domain elements, until only predicates of domain elements remain.
- Map the predicates of domain elements to truth values.
- Combine the truth values according to the truth tables for the connectives. Note that as this procedure is for ground formulae, quantifiers do not need to be interpreted here.


Example
<p>Using the interpretation above, the ground formula</p> <p>wise(geoff) <=> ~taller(geoff, brother_of(jim))</p> <p>is interpreted as follows</p> <p>wise(geoff) <=> ~taller(geoff, brother_of(jim))</p> <p>wise() <=> ~taller(, brother_of())</p> <p>wise() <=> ~taller(, )</p> <p>TRUE <=> ~FALSE</p> <p>TRUE <=> TRUE</p> <p>TRUE</p>


Non-ground closed formulae are interpreted as follows:

- A universally quantified formula is **TRUE** iff the unquantified formula is **TRUE** with the quantified variable instantiated to each of the domain elements.
- An existentially quantified formula is **TRUE** iff the unquantified formula is **TRUE** with the quantified variable instantiated to at least one of the domain elements.

Example

To interpret the universally quantified formula $\forall \text{Person } (\sim \text{taller}(\text{Person}, \text{jim}) \mid \text{wise}(\text{Person}))$, the universally quantified variable **Person** is instantiated to all the domain elements.


- **Person** = 
 - $\sim \text{taller}(\text{person}, \text{jim}) \mid \text{wise}(\text{person})$
 - $\sim \text{taller}(\text{person}, \text{h}) \mid \text{wise}(\text{person})$
 - $\sim \text{FALSE} \mid \text{TRUE}$
 - $\text{TRUE} \mid \text{TRUE}$
 - TRUE**

- **Person** = 
 - $\sim \text{taller}(\text{h}, \text{jim}) \mid \text{wise}(\text{h})$
 - $\sim \text{taller}(\text{h}, \text{h}) \mid \text{wise}(\text{h})$
 - $\sim \text{FALSE} \mid \text{TRUE}$
 - $\text{TRUE} \mid \text{TRUE}$
 - TRUE**

The unquantified formula is **TRUE** with the quantified variable instantiated to each of the domain elements, thus the universally quantified formula is **TRUE**.

Example

To interpret the existentially quantified formula $\exists \text{Person } \text{wise}(\text{brother_of}(\text{brother_of}(\text{Person})) \Rightarrow \text{wise}(\text{geoff}))$ the existentially quantified variable **Person** is instantiated to find a **TRUE** instance of the formula.

- **Person** = 
 - $\text{wise}(\text{brother_of}(\text{brother_of}(\text{person}))) \Rightarrow \text{wise}(\text{geoff})$
 - $\text{wise}(\text{brother_of}(\text{h})) \Rightarrow \text{wise}(\text{person})$
 - $\text{wise}(\text{person}) \Rightarrow \text{wise}(\text{person})$
 - $\text{TRUE} \Rightarrow \text{TRUE}$
 - TRUE**

The unquantified formula is **TRUE** with the quantified variable instantiated to at least one of the domain elements, thus the existentially quantified formula is **TRUE**.

If the domain of the interpretation is finite, interpretation of closed 1st order formulae is a finite process. If the domain is infinite it is not possible to determine that a universally quantified formula is **TRUE**, nor is it possible to determine that an existentially quantified formula is **FALSE**.

Some Interpretations

In general, an interpretation must have the structure discussed above. However, interpreting a formula in such a general structure can be very expensive. There are some simple interpretations in which the interpretation of a literal can be determined syntactically.

- The positive interpretation maps all atoms to **TRUE**.
- The negative interpretation maps all atoms to **FALSE**.
- A predicate partition, which divides the predicate symbols of the 1st order language into two partitions P_1 and P_2 , map atoms whose predicate symbols are in P_1 to **TRUE** and atoms whose predicate symbols are in P_2 to **FALSE**.

The Quest for Logical Consequence

To show that **C** is a logical consequence of **A**, it is necessary to:

- Show that every model of **A** is a model of **C**
- OR
- Show that $S' = A \cup \{\sim C\}$ is unsatisfiable.

For every 1st order language there is an infinite number of possible interpretations. New interpretations can be constructed by changing the domain and by changing the F and R mappings. Thus it is not possible to use the "table" method from propositional logic, as there is an infinite number of rows to check.

- Show that S' is unsatisfiable.

Exam Style Questions

1. What are the two major limitations of propositional logic?
2. What are the three components of a 1st order logic language?
3. Which or the following formulae are {ground, closed}?
<insert some formulae here>
4. What is important about closed formulae?
5. Translate the following into 1st order logic:

Suming, Yi, and Yury are students, and are the only three students. If a student works hard then they get a good grade. At least one of the students works hard. Therefore at least one student will get a good grade.

6. Express the following scenario and conclusion in 1st order logic, in a form ready to be converted to clause normal form.

There are some students who fail. If a student fails, then either there is a lecturer who has taught the student badly, or the student is stupid. No lecturer teaches any student badly. Therefore there is some student who is stupid.

7. In what circumstances is it necessary to add the axioms of equality to the formulae of a problem?
8. Name the five (groups of) axioms of equality.
9. What axioms of equality are needed with this formula?
 $p(X) \mid \sim p(f(cat,dog)) \mid equal(X,cat)$
10. What are the three components of an interpretation of a first order language? Give an example for the language:

```
V = { V : V starts with uppercase }
F = { coke/0, pepsi/0, competitor/1 }
P = { fizzy/1, sells-more/2 }
```

11. Given the 1st order logic language:

```
V = { V : V starts with uppercase }
F = { holden/0, ford/0, honda/0 main_competitor/1 }
P = { fast/1, faster/2 }
```

and the interpretation:

```
D = { commodore, laser, prelude }
F = { holden → commodore,
      ford → laser,
      honda → prelude,
      main_competitor(commodore) → prelude,
      main_competitor(laser) → prelude,
      main_competitor( Prelude ) → commodore }
R = { fast(commodore) → TRUE,
      fast(laser) → FALSE,
      fast( Prelude ) → TRUE,
      faster(commodore,commodore) → FALSE,
      faster(commodore,laser) → TRUE,
      faster(commodore, Prelude ) → TRUE,
      faster(laser,commodore) → FALSE,
      faster(laser,laser) → FALSE,
      faster(laser, Prelude ) → FALSE,
      faster( Prelude ,commodore) → FALSE,
      faster( Prelude ,laser) → TRUE,
      faster( Prelude , Prelude ) → FALSE }
```

Show the steps of the interpretation of:

- $faster(main_competitor(laser),commodore)$
- $fast(main_competitor(main_competitor(Prelude))$
- $forall X (faster(main_competitor(X),laser) \& fast(X))$
- $exists X (fast(main_competitor(main_competitor(X))) \mid forall Y faster(X,Y)))$

12. What are the positive and negative interpretations?
 13. Explain why truth tables cannot be used to establish logical consequence in 1st order logic.
-

Clause Normal Form

Clause Normal Form (CNF) is a sub-language of 1st order logic. A clause is an expression of the form $L_1 \mid \dots \mid L_m$ where each L_i is a literal. Clauses are denoted by uppercase letters with a superscript \mid , e.g., c^i .

There are satisfiability preserving transformations from 1st order logic to CNF, i.e., if a set of (1st order) formulae are satisfiable, then their CNF is satisfiable. Conversely, if the CNF of a set of formulae is unsatisfiable, then the formulae are unsatisfiable. This is then useful for showing logical consequence. The benefit of converting to CNF is that more is possible using just the Herbrand interpretations. Computationally, CNF is easier to work with, and is the form used by the resolution inference rule.

Transforming to Clause Normal Form

There are several algorithms for this conversion, but they all have some parts in common, and all have similarities. The best algorithm is the one which produces the CNF which is most easily shown to be unsatisfiable. In general such a determination is impossible to make, but a common measure is to aim to produce a CNF with the fewest symbols (where a symbol is a variable, predicate, or constant). The key feature of all the algorithms is that they are satisfiability preserving.

The starting point is a set of closed formulae.

The first operation is to *simplify* the formulae so that they contain only the \forall , \exists , $\&$, \mid and \sim connectives. This is done by using known logical identities. The identities are easily verified using truth tables, to show that the original formula has the same value as the simplified formula for all possible interpretations of the component formulae, denoted by uppercase variable letters P and Q .

Simplify	
Formula	Rewrites to
$\sim(P \Leftrightarrow Q)$	$(P \mid Q) \& (\sim P \mid \sim Q)$
$\sim(P \Rightarrow Q)$	$P \& \sim Q$
$P \Leftrightarrow Q$	$(P \Rightarrow Q) \& (Q \Rightarrow P)$
$P \Rightarrow Q$	$\sim P \mid Q$

The second operation is to *move negations in* so that they apply to only atoms. This is done by using known logical identities (including De Morgan's laws), which are easily verified as above. After moving negations in the formulae are in *literal normal form*.

Move negations in	
Formula	Rewrites to
$\sim\forall x P$	$\exists x \sim P$
$\sim\exists x P$	$\forall x \sim P$
$\sim(P \& Q)$	$\sim P \mid \sim Q$
$\sim(P \mid Q)$	$\sim P \& \sim Q$
$\sim\sim P$	P

The third operation is to *move quantifiers out* so that each formula is in the scope of all the quantifiers in that formula. This is done by using known logical identities, which are easily verified as above. In these identities, if the variable x already exists in Q , it is renamed to a new variable name that does not exist in the formula. After moving quantifiers out the formulae are in *prenex normal form*.

Move quantifiers out	
Formula	Rewrites to
$\forall x P \& Q$	$\forall x (P \& Q)$
$\exists x P \& Q$	$\exists x (P \& Q)$
$Q \& \forall x P$	$\forall x (Q \& P)$
$Q \& \exists x P$	$\exists x (Q \& P)$
$\forall x P \mid Q$	$\forall x (P \mid Q)$
$\exists x P \mid Q$	$\exists x (P \mid Q)$
$Q \mid \forall x P$	$\forall x (Q \mid P)$
$Q \mid \exists x P$	$\exists x (Q \mid P)$

The fourth operation is to *Skolemize* to remove existential quantifiers. This step replaces existentially quantified variables by *Skolem functions* and removes all quantifiers. The variables remaining after Skolemization are all implicitly universally quantified. Whereas the previous three operations maintain equivalence between the original and transformed formulae, Skolemization does not. However, Skolemization does maintain the satisfiability of the formulae, which is what is required for the overall goal of establishing logical consequence. After Skolemization the formulae are in *Skolem normal form*.

Skolemize	
Formula	Rewrites to
Initially	Set $\gamma = \{\}$
$\forall x P$	P and set $\gamma = \gamma \cup \{x\}$
$\exists x P$	$P[x/x(\gamma)]$ where x is a new Skolem functor.

The fifth operation is to *distribute disjunctions* so that the formulae are conjunctions of disjunctions. This is done by using known logical identities, which are easily verified as above. After distributing disjunctions the formulae are in *conjunctive normal form*.

Distribute disjunctions	
Formula	Rewrites to
$P \mid (Q \ \& \ R)$	$(P \mid Q) \ \& \ (P \mid R)$
$(Q \ \& \ R) \mid P$	$(Q \mid P) \ \& \ (R \mid P)$

The last operation is to *convert to Clause Normal Form*. This is done by removing the conjunctions, to form a set of clauses.

Convert to CNF	
Formula	Rewrites to
$P_1 \ \& \ \dots \ \& \ P_n$	$\{P_1, \dots, P_n\}$

Example
1st order formula $\forall Y (\forall X (\text{taller}(Y,X) \mid \text{wise}(X)) \Rightarrow \text{wise}(Y))$
Simplify $\forall Y (\sim \forall X (\text{taller}(Y,X) \mid \text{wise}(X)) \mid \text{wise}(Y))$
Move negations in $\forall Y (\exists X (\sim \text{taller}(Y,X) \ \& \ \sim \text{wise}(X)) \mid \text{wise}(Y))$
Move quantifiers out $\forall Y (\exists X ((\sim \text{taller}(Y,X) \ \& \ \sim \text{wise}(X)) \mid \text{wise}(Y)))$
Skolemize $\exists X ((\sim \text{taller}(Y,X) \ \& \ \sim \text{wise}(X)) \mid \text{wise}(Y)) \ \gamma = \{Y\}$ $(\sim \text{taller}(Y,x(Y)) \ \& \ \sim \text{wise}(x(Y))) \mid \text{wise}(Y)$
Distribute disjunctions $(\sim \text{taller}(Y,x(Y)) \mid \text{wise}(Y)) \ \& \ (\sim \text{wise}(x(Y)) \mid \text{wise}(Y))$
Convert to CNF $\{ \sim \text{taller}(Y,x(Y)) \mid \text{wise}(Y),$ $\sim \text{wise}(x(Y)) \mid \text{wise}(Y) \}$

Special Forms

Clauses may be expressed in Kowalski form. Kowalski form forms the antecedent of an implication by conjoining the atoms of the negative literals in a clause, and forms the consequent from the disjunction of the positive literals.

Example
$\sim \text{taller}(Y,x(Y)) \mid \text{wise}(Y) \mid \sim \text{wise}(x(Y)) \mid \text{taller}(x(Y),Y)$
is written as $\text{taller}(Y,x(Y)) \ \& \ \text{wise}(x(Y)) \Rightarrow \text{wise}(Y) \mid \text{taller}(x(Y),Y)$
If the antecedent is empty it is set to TRUE , and if the consequent is empty it is set to FALSE .

Horn clauses are clauses that have one or zero positive literals. Sets of Horn clauses are also called Horn.

Example
$\sim \text{taller}(Y,x(Y)) \mid \text{wise}(Y) \mid \sim \text{wise}(x(Y))$ $\sim \text{wise}(\text{geoff}) \mid \sim \text{taller}(\text{Person}, \text{brother_of}(\text{jim}))$

are Horn clauses. Written in Kowalski form, these Horn clauses are :

```
taller(Y,x(Y)) & wise(x(Y)) => wise(Y)
wise(geoff) & taller(Person,brother_of(jim)) => FALSE
```

Non-Horn clauses are clauses that have two or more positive literals. Sets of clauses that contain one or more non-Horn clauses are also called non-Horn.

Example

```
~taller(Y,x(Y)) | ~wise(Y) | wise(x(Y)) | taller(x(Y),x(x(Y)))
wise(X) | wise(brother_of(X))
```

are non-Horn clauses. Written in Kowalski form, these non-Horn clauses are :

```
taller(Y,x(Y)) & wise(Y) => wise(x(Y)) | taller(x(Y),x(x(Y)))
TRUE => wise(X) | wise(brother_of(X))
```

In terms of ATP, the disjunction of positive literals in non-Horn clauses corresponds to a choice point in any goal directed search. This is a source of search. Thus, in general, Horn problems are easier than non-Horn problems.

Exercises

Convert the following to CNF:

- $\sim\exists X (s(X) \& q(X))$
Answer: $\{ \sim s(X) \mid \sim q(X) \}$
- $\forall X (p(X) \Rightarrow (q(X) \mid r(X)))$
Answer: $\{ \sim p(X) \mid q(X) \mid r(X) \}$
- $\sim\exists X (p(X) \Rightarrow \exists X q(X))$
Answer: $\{ p(X), \sim q(X) \}$
- $\forall X (q(X) \mid r(X) \Rightarrow s(X))$
Answer: $\{ \sim q(X) \mid s(X), \sim r(X) \mid s(X) \}$
- $\exists X (p \Rightarrow f(X))$
Answer: $\{ \sim p \mid f(skX) \}$
- $\exists X (p \Leftrightarrow f(X))$
Answer: $\{ \sim p \mid f(skX), p \mid \sim f(skX) \}$
- $\forall Z \exists Y \forall X (f(X,Y) \Leftrightarrow (f(X,Z) \& \sim f(X,X)))$
Answer: $\{ \sim f(A,skA(B)) \mid f(A,B), \sim f(A,skA(B)) \mid \sim f(A,A), \sim f(A,B) \mid f(A,A) \mid f(A,skA(B)) \}$
- $\forall X \forall Y (q(X,Y) \Leftrightarrow \forall Z (f(Z,X) \Leftrightarrow f(Z,Y)))$
Answer: $\{ \sim q(A,B) \mid \sim f(C,A) \mid f(C,B), \sim q(A,B) \mid \sim f(C,B) \mid f(C,A), f(sk1(A,B),B) \mid f(sk1(A,B),A) \mid q(B,A), f(sk1(A,B),B) \mid \sim f(sk1(A,B),B) \mid q(B,A), \sim f(sk1(A,B),A) \mid f(sk1(A,B),A) \mid q(B,A), \sim f(sk1(A,B),A) \mid \sim f(sk1(A,B),B) \mid q(B,A) \}$
- $\forall X \exists Y ((p(X,Y) \Leftrightarrow \forall X \exists T q(Y,X,T)) \Rightarrow r(Y))$
Answer: $\{ \sim p(A,sk1(A)) \mid r(sk1(A)), q(sk1(A),B,sk2(B,A)) \mid r(sk1(A)) \}$
- $\forall X (p(X) \Rightarrow \exists Y \sim (q(X,Y) \mid \sim r(Y)))$
Answer: TBA

Satisfiability Preservation

A set of formulae is satisfiable iff its clause normal form is satisfiable.

Proof

- **A formula is equivalent to its simplified form**
Directly from the truth tables for the connectives.
- **A simplified formula is equivalent to its literal normal form**
Consider, for example, the rewriting of $\sim\forall X P$ to $\exists X \sim P$:

```
~∀X P is TRUE in an interpretation I
iff
∀X P is FALSE in I
iff
For some element e in the domain of I, P{X/e} is FALSE in I
iff
~P{X/e} is TRUE in I
iff
∃X ~P is TRUE in I.
```

Thus $\sim\forall X P$ is equivalent to $\exists X \sim P$. Similar argument establishes the equivalence of formulae rewritten using the other literal normal form rewritings.

- **A simplified formula is equivalent to its prenex normal form**
Consider, for example, the rewriting of $\forall X P \& Q$ to $\forall X (P \& Q)$:

$\forall x P \ \& \ Q$ is TRUE in an interpretation I
 iff
 $\forall x P$ is TRUE in I and Q is TRUE in I
 iff
 for all elements e in the domain of I, $P\{x/e\}$ is TRUE in I, and for all elements e in the domain of I, $Q\{x/e\}$ is TRUE in I
 iff
 for all elements e in the domain of I $(P \ \& \ Q)\{x/e\}$ is TRUE in I
 iff
 $\forall x (P \ \& \ Q)$ is TRUE in I

Thus $\forall x P \ \& \ Q$ is equivalent to $\forall x (P \ \& \ Q)$. Similar argument establishes the equivalence of formulae rewritten using the other prenex normal form rewritings.

• **A formula is satisfiable iff its Skolem normal form is satisfiable**

The first rewriting, which eliminates universal quantifiers, has no effect as the universal quantifiers are still there implicitly.

Consider the rewriting of $\exists x P$ to $P[x/x(\gamma)]$ Let γ be of the form x_1, \dots, x_n , i.e., x_1, \dots, x_n are the (implicitly) universally quantified variables at this point.

- If $\exists x P$ is satisfiable with model I, then for all substitutions $\theta = \{x_1/e_1, \dots, x_n/e_n\}$, there exists a domain element e such that $P\theta\{x/e\}$ is TRUE in I. Extend I to include the function mappings $x(e_1, \dots, e_n) \Rightarrow e$, for each θ . Then $P\theta[x/x(\gamma)]$ is TRUE in the extended interpretation. As this is the case for all substitutions θ , I is a model for $P[x/x(\gamma)]$, which is thus satisfiable.
- If $P[x/x(\gamma)]$ is satisfiable with model I, then for all substitutions $\theta = \{x_1/e_1, \dots, x_n/e_n\}$, where each e_i is from the domain of I, $P[x/x(\gamma)]\theta$ is TRUE in I. Let e be the interpretation of $x(\gamma)\theta$ in I. Then $P\theta\{x/e\}$ is TRUE in I. As this is the case for all substitutions θ , I is a model for $\exists x P$ which is thus satisfiable.

• **A formula is equivalent to its conjunctive normal form**

This is evident from the definitions of the connectives being manipulated.

• **A formula is satisfiable iff its clause normal form is satisfiable**

This is evident from the definitions of $\&$ and satisfiability of sets.

The above points combined prove the theorem.

Example

Translating the formula $\forall Y (\forall X \sim \text{taller}(X, Y) \Rightarrow \text{wise}(Y))$

- $\forall Y (\forall X \sim \text{taller}(X, Y) \Rightarrow \text{wise}(Y))$ is equivalent to its simplified form $\forall Y (\sim \forall X \sim \text{taller}(X) \mid \text{wise}(Y))$. Directly from the truth tables for the connectives.
- $\forall Y (\sim \forall X \sim \text{taller}(X, Y) \mid \text{wise}(Y))$ is equivalent to its literal normal form $\forall Y (\exists X \text{ taller}(X, Y) \mid \text{wise}(Y))$

For $Y = \text{geoff}$...

$\sim \forall X \sim \text{taller}(X, \text{geoff})$ is TRUE in I =

$D = \{ \text{geoff}, \text{jim} \}$
 $F = \{ \text{geoff} \rightarrow \text{geoff},$
 $\text{jim} \rightarrow \text{jim},$
 $\text{brother_of}(\text{geoff}) \rightarrow \text{jim},$
 $\text{brother_of}(\text{jim}) \rightarrow \text{geoff} \}$
 $R = \{ \text{wise}(\text{geoff}) \rightarrow \text{FALSE},$
 $\text{wise}(\text{jim}) \rightarrow \text{TRUE},$
 $\text{taller}(\text{geoff}, \text{geoff}) \rightarrow \text{FALSE},$
 $\text{taller}(\text{geoff}, \text{jim}) \rightarrow \text{FALSE},$
 $\text{taller}(\text{jim}, \text{geoff}) \rightarrow \text{TRUE},$
 $\text{taller}(\text{jim}, \text{jim}) \rightarrow \text{FALSE} \}$

iff

$\forall X \sim \text{taller}(X, \text{geoff})$ is FALSE in I
 iff

$\sim \text{taller}(\text{jim}, \text{geoff})$ is FALSE in I
 iff

$\sim\text{taller}(\text{Jim}, \text{Geoff})$ is TRUE in I
iff

$\text{taller}(\text{Jim}, \text{Geoff})$ is TRUE in I
iff

$\exists X \sim\text{taller}(X, \text{Geoff})$ is TRUE in I.

- $\forall Y (\exists X \text{taller}(X, Y) \mid \text{wise}(Y))$ is equivalent to its prenex normal form $\forall Y \exists X (\text{taller}(X, Y) \mid \text{wise}(Y))$

$\forall Y (\exists X \text{taller}(X, Y) \mid \text{wise}(Y))$ is TRUE in I (as above)
iff

$\exists X \text{taller}(X, \text{Geoff}) \mid \text{wise}(\text{Geoff})$ is TRUE in I and $\exists X \text{taller}(X, \text{Jim}) \mid \text{wise}(\text{Jim})$ is TRUE in I
iff

$\text{taller}(\text{Jim}, \text{Geoff}) \mid \text{wise}(\text{Geoff})$ is TRUE in I and $\text{taller}(\text{Jim}, \text{Jim}) \mid \text{wise}(\text{Jim})$ is TRUE in I
iff

$\forall Y (\text{taller}(\text{Jim}, Y) \mid \text{wise}(Y))$ is TRUE in I
iff

$\forall Y \exists X (\text{taller}(X, Y) \mid \text{wise}(Y))$ is TRUE in I

- $\forall Y \exists X (\text{taller}(X, Y) \mid \text{wise}(Y))$ satisfiable iff its Skolem normal form $\text{taller}(x(Y), Y) \mid \text{wise}(Y)$ is satisfiable

After dropping the $\forall Y, \gamma = [Y]$

- If $\exists X (\text{taller}(X, Y) \mid \text{wise}(Y))$ is satisfiable, then for the substitution $\theta = \{Y/\text{Geoff}\}$, there exists the domain element Jim such that $\text{taller}(\text{Jim}, \text{Geoff}) \mid \text{wise}(\text{Geoff})$ is TRUE in I. Extend I to include the function mapping $x(\text{Geoff}) \Rightarrow \text{Jim}$. Then $\text{taller}(x(\text{Geoff}), \text{Geoff}) \mid \text{wise}(\text{Geoff})$ is TRUE in the extended interpretation. Similarly, for the substitution $\theta = \{Y/\text{Jim}\}$, extend I to include the function mapping $x(\text{Jim}) \Rightarrow \text{Jim}$. Therefore the extended I can be extended a model for $\text{taller}(x(Y), Y) \mid \text{wise}(Y)$, which is thus satisfiable.
- If $\text{taller}(x(Y), Y) \mid \text{wise}(Y)$ is satisfiable with model I =

$D = \{ \text{Geoff}, \text{Jim} \}$
 $F = \{ \text{geoff} \rightarrow \text{Geoff},$
 $\text{jim} \rightarrow \text{Jim},$
 $\text{brother_of}(\text{Geoff}) \rightarrow \text{Jim},$
 $\text{brother_of}(\text{Jim}) \rightarrow \text{Geoff},$
 $x(\text{Geoff}) \rightarrow \text{Jim},$
 $x(\text{Jim}) \rightarrow \text{Jim} \}$
 $R = \{ \text{wise}(\text{Geoff}) \rightarrow \text{FALSE},$
 $\text{wise}(\text{Jim}) \rightarrow \text{TRUE},$
 $\text{taller}(\text{Geoff}, \text{Geoff}) \rightarrow \text{FALSE},$
 $\text{taller}(\text{Geoff}, \text{Jim}) \rightarrow \text{FALSE},$
 $\text{taller}(\text{Jim}, \text{Geoff}) \rightarrow \text{TRUE},$
 $\text{taller}(\text{Jim}, \text{Jim}) \rightarrow \text{FALSE} \}$

then for the substitution $\theta = \{Y/\text{Jim}\}$, $\text{taller}(x(\text{Jim}), \text{Jim}) \mid \text{wise}(\text{Jim})$ is TRUE in I. Jim is the interpretation of $x(\text{Jim})$ in I. Then $\text{taller}(\text{Jim}, \text{Jim}) \mid \text{wise}(\text{Jim})$ is TRUE in I. As this is the case for all substitutions θ , I can be extended a model for $\exists X (\text{taller}(X, Y) \mid \text{wise}(Y))$, which is thus satisfiable.

The Quest for Logical Consequence

To show that C is a logical consequence of A, it is necessary to:

- Show that every model of A is a model of C
OR
- Show that $S' = A \cup \{\sim C\}$ is unsatisfiable.

S' is unsatisfiable iff S = CNF(S') is unsatisfiable

- Show that S is unsatisfiable.
-

Exam Style Questions

1. Convert the following to CNF. Notation: & for AND, | for OR, ! for universal quantifier, ? for existential quantifier.

```
( ? [Y] :  
  ! [X] :  
    ( element(X,Y)  
  <=> element(X,X) )  
=> ~ ( ! [X1] :  
  ? [Y1] :  
    ! [Z] :  
      ( element(Z,Y1)  
    <=> ~ element(Z,X1) ) ) )
```

2. Explain, with examples, the difference between Horn and non-Horn clauses.
 3. Explain, with examples, what is meant by Kowalski form for clauses. Why is Kowalski form important?
 4. Prove that a set of formulae is satisfiable iff its clause normal form is satisfiable.
-

The Herbrand Universe and Herbrand Base

The *Herbrand universe* of a 1st order language is the set of all ground terms. If the language has no constants, then the language is extended by adding an arbitrary new constant. It is enumerable, and infinite if a functor of arity greater than 0 exists.

The *Herbrand base* of a 1st order language is the set of all ground atoms formed using elements of the Herbrand universe as arguments. Like the Herbrand universe, it is enumerable, and infinite if the Herbrand universe is infinite and there is a predicate symbol of arity greater than 0.

Example

The Herbrand universe and Herbrand base of the language

```
V = { V : V starts with uppercase }
F = { geoff/0, jim/0, brother_of/1 }
P = { wise/1, taller/2 }
```

are

```
HU = { geoff,
      jim,
      brother_of(geoff),
      brother_of(jim),
      brother_of(brother_of(geoff)),
      ... }
HB = { wise(geoff),
      taller(geoff,geoff),
      wise(jim),
      taller(jim,jim),
      taller(geoff,jim),
      taller(jim,geoff),
      wise(brother_of(geoff)),
      taller(geoff,brother_of(geoff)),
      taller(brother_of(geoff),geoff),
      taller(brother_of(geoff),brother_of(geoff)),
      ... }
```

Herbrand Interpretations

A Herbrand interpretation has

- D = the Herbrand universe,
- F = the identity function,
- R = a subset of the Herbrand base that is TRUE.

R defines a mapping from the Herbrand base to {TRUE, FALSE}.

Example

One possible Herbrand interpretation of the language

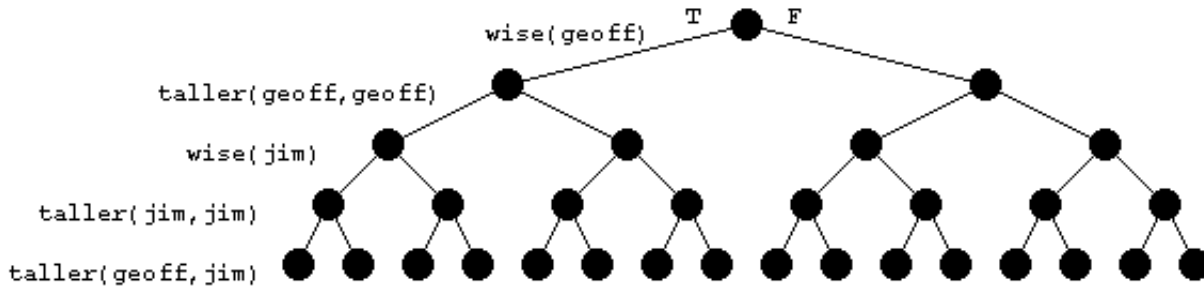
```
V = { V : V starts with uppercase }
F = { geoff/0, jim/0, brother_of/1 }
P = { wise/1, taller/2 }
```

is

```
D = { geoff,
      jim,
      brother_of(geoff),
      brother_of(jim),
      brother_of(brother_of(geoff)),
      ... }
F = { geoff => geoff,
      jim => jim,
      brother_of(geoff) => brother_of(geoff),
      brother_of(jim) => brother_of(jim),
      ... }
R = { wise(geoff) => TRUE,
      taller(geoff,geoff) => TRUE,
      wise(jim) => TRUE,
      taller(jim,jim) => FALSE,
      taller(geoff,jim) => FALSE,
      taller(jim,geoff) => FALSE,
      wise(brother_of(geoff)) => TRUE,
      taller(geoff,brother_of(geoff)) => FALSE,
```

```
taller(brother_of(geoff), geoff) => FALSE,
taller(brother_of(geoff), brother_of(geoff)) => FALSE,
... }
```

As R is the only variable part of a Herbrand interpretation, it is possible to identify a Herbrand interpretation with R. Each branch of the semantic tree for the Herbrand base identifies a Herbrand interpretation. A semantic tree for a Herbrand interpretation is *fair* if the Herbrand Base elements are ordered by non-decreasing symbol count; assume all semantic trees are fair.



A Herbrand interpretation is a Herbrand model of a formula (set of formulae) iff the formula (each formula in the set) is TRUE in the Herbrand interpretation

A Herbrand interpretation removes the infinite number of domains, and leaves one infinite domain and a possibly infinite number of Rs.

Model Preservation

A set of clauses has a model iff it has a Herbrand model.

Proof

- **If a set of clauses has a Herbrand model then it has a model.**

This is proved by constructing a model from the Herbrand model. In the model

- The domain is the Herbrand universe
- The function mapping is the identity function
- The predicate mapping is as in Herbrand model.

- **If a set of clauses has a model, then it has a Herbrand model.**

This is proved by constructing a Herbrand model H from the existing model M. Let the subset of the Herbrand base that is TRUE in H be the subset that is TRUE in M. Let $c^|$ be a clause that is TRUE in M. It is necessary to show that $c^|$ is TRUE in H.

- Let $\theta = \{x_1/t_1, \dots, x_n/t_n\}$ be any Herbrand universe substitution (each t_i is a Herbrand universe element) such that $c^|\theta$ is ground.
- Let e_i be the interpretation of t_i in M, and $\sigma = \{x_1/e_1, \dots, x_n/e_n\}$.
- $c^|\sigma$ is TRUE (accepting the technical scruple) in M.
- Let $L\sigma$ be a literal of $c^|\sigma$ that is TRUE in M ($L\sigma$ must exist), and let $L\theta$ be the corresponding literal in $c^|\theta$.
- By the definition of H, $L\theta$ is TRUE in H. Thus $c^|\theta$ is TRUE in H.
- But θ is arbitrary, i.e., $c^|\theta$ is TRUE in H for all θ .
- Thus $c^|$ is TRUE in H, i.e., H is a Herbrand model of $c^|$.

Example

Consider the clause $C^|$:

\sim taller(Person, jim) | wise(Person)

- \sim taller(Person, jim) | wise(Person) has the Herbrand model that interprets every Herbrand base element as TRUE. This Herbrand model is also a normal model.
- \sim taller(Person, jim) | wise(Person) has the normal model M:

$D = \{ \text{geoff}, \text{jim} \}$

$F = \{ \text{geoff} \rightarrow \text{geoff},$
 $\text{jim} \rightarrow \text{jim},$
 $\text{brother_of}(\text{geoff}) \rightarrow \text{jim},$
 $\text{brother_of}(\text{jim}) \rightarrow \text{geoff} \}$

$R = \{ \text{wise}(\text{geoff}) \rightarrow \text{FALSE},$
 $\text{wise}(\text{jim}) \rightarrow \text{TRUE},$
 $\text{taller}(\text{geoff}, \text{geoff}) \rightarrow \text{FALSE},$

$\text{taller}(\text{Jim}, \text{John}) \rightarrow \text{FALSE},$
 $\text{taller}(\text{John}, \text{Jim}) \rightarrow \text{TRUE},$
 $\text{taller}(\text{John}, \text{John}) \rightarrow \text{FALSE} \}$

Construct the Herbrand interpretation H which interprets Herbrand base elements according to their interpretation in M.

- Let $\theta = \{\text{Person}/\text{brother_of}(\text{brother_of}(\text{jim}))\}$ be a Herbrand universe substitution, and $\sim\text{taller}(\text{Person}, \text{jim}) \mid \text{wise}(\text{Person})\theta$ is the ground clause $\sim\text{taller}(\text{brother_of}(\text{brother_of}(\text{jim})), \text{jim}) \mid \text{wise}(\text{brother_of}(\text{brother_of}(\text{jim})))$
- John is the interpretation of $\text{brother_of}(\text{brother_of}(\text{jim}))$ in M, and $\sigma = \{\text{Person}/\text{John}\}$.
- $\sim\text{taller}(\text{Person}, \text{jim}) \mid \text{wise}(\text{Person})\{\text{Person}/\text{John}\} = \sim\text{taller}(\text{John}, \text{jim}) \mid \text{wise}(\text{John})$ is TRUE (accepting the technical scruple) in M.
- $\text{wise}(\text{John})$ is a literal of $\sim\text{taller}(\text{John}, \text{jim}) \mid \text{wise}(\text{John})$ that is TRUE in M, and $\text{wise}(\text{brother_of}(\text{brother_of}(\text{jim})))$ is the corresponding literal in $\sim\text{taller}(\text{brother_of}(\text{brother_of}(\text{jim})), \text{jim}) \mid \text{wise}(\text{brother_of}(\text{brother_of}(\text{jim})))$.
- By the definition of H, $\text{wise}(\text{brother_of}(\text{brother_of}(\text{jim})))$ is TRUE in H. Thus $\sim\text{taller}(\text{brother_of}(\text{brother_of}(\text{jim})), \text{jim}) \mid \text{wise}(\text{brother_of}(\text{brother_of}(\text{jim})))$ is TRUE in H.
- But $\{\text{Person}/\text{brother_of}(\text{brother_of}(\text{jim}))\}$ is arbitrary, i.e., $\sim\text{taller}(\text{Person}, \text{jim}) \mid \text{wise}(\text{Person})\theta$ is TRUE in H for all θ .
- Thus $\sim\text{taller}(\text{Person}, \text{jim}) \mid \text{wise}(\text{Person})$ is TRUE in H, i.e., H is a Herbrand model of $\sim\text{taller}(\text{Person}, \text{jim}) \mid \text{wise}(\text{Person})$.

The above theorem does not hold for non-clauses, e.g. $\{p(a), \exists x \sim p(x)\}$ has the model:

$D = \{0, 1\}$
 $F = \{a \rightarrow 0\}$
 $R = \{p(0) \rightarrow \text{TRUE},$
 $\quad p(1) \rightarrow \text{FALSE}\}$

but the only Herbrand interpretations are

$R = \{p(a) \rightarrow \text{FALSE}\}$

and

$R = \{p(a) \rightarrow \text{TRUE}\}$

neither of which are Herbrand models of the formulae. This is because a non-Herbrand model can have domain elements that do not represent any Herbrand universe element. Skolemization removes existentially quantified variables, and replaces them with terms. Ground instances of such Skolem terms are Herbrand universe elements which can be aligned with domain elements.

Unsatisfiability for sets of clauses can now be considered in terms of Herbrand interpretations.

The Quest for Logical Consequence

To show that C is a logical consequence of A, it is necessary to:

- Show that every model of A is a model of C
- OR
- Show that $S' = A \cup \{\sim C\}$ is unsatisfiable.

S' is unsatisfiable iff $S = \text{CNF}(S')$ is unsatisfiable

- Show that S is unsatisfiable.

A set of clauses has a model iff it has a Herbrand model, so

- Show that S is Herbrand unsatisfiable.

Exam Style Questions

- What are the Herbrand Universe and Herbrand Base of a 1st order language?
- Define the three components of a Herbrand interpretation of a 1st order language.
- Prove that a set of clauses has a model iff it has a Herbrand model.

Unification

A substitution θ *unifies* a set s of expressions if $s\theta$ is a singleton. A unifier θ is a most general unifier (mgu) for a set s if for each unifier σ of s , there exists a substitution γ such that $\sigma = \theta\gamma$.

We are interested in substitutions that unify expressions.

Example
$U = \{ \text{wise}(X), \\ \text{wise}(\text{brother_of}(Y)), \\ \text{wise}(\text{brother_of}(Z)) \}$
is unified by $\theta = \{X/\text{brother_of}(Z), Y/Z\}$.

The disagreement set of a set s of expressions is defined as follows : Locate the left most symbol position at which not all expressions in s have the same symbol and extract from each expression in s the subexpression beginning at that symbol position. The set of all such subexpressions is called the disagreement set.

Example
$U = \{ \text{wise}(X), \\ \text{wise}(\text{brother_of}(Y)), \\ \text{wise}(\text{brother_of}(Z)) \}$
has the disagreement set $D = \{X, \text{brother_of}(Y), \text{brother_of}(Z)\}$

Unification algorithm for a set u .

- Put $k=0$ and $\theta_0 = \{\}$
- If $u\theta_k$ is a singleton then θ_k is a mgu for u . Otherwise find the disagreement set D_k of $u\theta_k$.
- If there exist variable v and term t in D_k such that v does not occur in t , then put $\theta_{k+1} = \theta_k\{v/t\}$, increment k and loop. Otherwise stop with failure.

Example			
$U = \{ \text{wise}(X), \\ \text{wise}(\text{brother_of}(Y)), \\ \text{wise}(\text{brother_of}(Z)) \}$			
k	θ_k	$u\theta_k$	D_k
0	$\{\}$	$\{\text{wise}(X), \\ \text{wise}(\text{brother_of}(Y)), \\ \text{wise}(\text{brother_of}(Z))\}$	$\{X, \\ \text{brother_of}(Y), \\ \text{brother_of}(Z)\}$
1	$\{X/\text{brother_of}(Y)\}$	$\{\text{wise}(\text{brother_of}(Y)), \\ \text{wise}(\text{brother_of}(Z))\}$	$\{Y, \\ Z\}$
2	$\{X/\text{brother_of}(Z), \\ Y/Z\}$	$\{\text{wise}(\text{brother_of}(Z))\}$	

Examples	
u	Result
$\{p(X, f(\text{cat})), \\ p(f(Y), f(Y)), \\ p(f(Z), T)\}$	$\{X/f(\text{cat}), Y/\text{cat}, T/f(\text{cat})\}$
$\{p(X, f(\text{cat})), \\ p(f(Y), f(Y)), \\ p(f(\text{dog}), Z)\}$	Failure
$\{p(f(Y), f(Y)), \\ p(f(Z), Z)\}$	Occur check failure

The occur check is omitted in most Prolog implementations.

Resolution

Propositional binary resolution

$$\begin{aligned}
 C^1 \mid L + D^1 \mid \sim L \\
 = C^1 \mid D^1
 \end{aligned}$$

Example

i_am_clever i_will_pass + ~i_am_clever i_will_pass	
$C^1 = i_will_pass$	$D^1 = i_will_pass$
$L = i_am_clever$	$\sim L = \sim i_am_clever$
$C^1 \mid D^1$	$= i_will_pass \mid i_will_pass$

An intuitive view of binary resolution is as an implementation of modus ponens and modus tolens. Full resolution generalizes the notion.

Propositional full resolution

$$\begin{aligned}
 C^1 \mid L_1 \mid \dots \mid L_n + D^1 \mid \sim L_1 \mid \dots \mid \sim L_m \\
 = C^1 \mid D^1
 \end{aligned}$$

Example

i_am_clever i_will_pass i_am_clever + ~i_am_clever i_will_pass	
$C^1 = i_will_pass$	$D^1 = i_will_pass$
$L_i = i_am_clever$	$\sim L_i = \sim i_am_clever$
$C^1 \mid D^1$	$= i_will_pass \mid i_will_pass$

Binary resolution

When performing resolution (and other inference steps) with formulae that contain variables, it is necessary to rename variables such that no variables are common across the two clauses.

$$\begin{aligned}
 C^1 \mid L + D^1 \mid \sim M \\
 L\theta \equiv M\theta \\
 = (C^1 \mid D^1)\theta
 \end{aligned}$$

The two parent clauses resolved against one another, upon the literals that unify. The result is a *resolvent*. If no literals remain after resolution, the resolvent is FALSE (indicating that the parent clauses contradicted each other). There may be multiple possible resolvent of a pair of parents, by selecting different *pairs* of literals to resolve upon.

Example

$$\sim wise(Z) \mid wise(brother_of(Z)) + \sim wise(X) \mid \sim wise(brother_of(Y)) \mid taller(X,Y)$$

1st option

$C^1 = \sim wise(Z)$	$D^1 = \sim wise(brother_of(Y)) \mid taller(X,Y)$
$L = wise(brother_of(Z))$	$\sim M = \sim wise(X)$
$\theta = \{X/brother_of(Z)\}$	
$C^1\theta = \sim wise(Z)$	$D^1\theta = \sim wise(brother_of(Y)) \mid taller(brother_of(Z),Y)$
$L\theta = wise(brother_of(Z))$	$M\theta = wise(brother_of(Z))$
$(C^1 \mid D^1)\theta$	$= \sim wise(Z) \mid \sim wise(brother_of(Y)) \mid taller(brother_of(Z),Y)$

2nd option

$C^1 = \sim wise(Z)$	$D^1 = \sim wise(X) \mid taller(X,Y)$
$L = wise(brother_of(Z))$	$\sim M = \sim wise(brother_of(Y))$
$\theta = \{Y/Z\}$	
$D^1\theta = \sim wise(X) \mid taller(X,Z)$	$M\theta = wise(brother_of(Z))$
$L\theta = wise(brother_of(Z))$	$C^1\theta = \sim wise(Z)$
$(C^1 \mid D^1)\theta$	$= \sim wise(Z) \mid \sim wise(X) \mid taller(X,Z)$

Full resolution

$$\begin{aligned}
 C^1 \mid L_1 \mid \dots \mid L_n + D^1 \mid \sim M_1 \mid \dots \mid \sim M_m \\
 L_j\theta \equiv M_j\theta
 \end{aligned}$$

$$= (c^l \mid d^l)\theta$$

There may be multiple possible resolvent of a pair of parents, by selecting different *combinations* of literals to resolve upon.

Example	
$\sim\text{wise}(Z) \mid \text{wise}(\text{brother_of}(Z)) + \sim\text{wise}(X) \mid \sim\text{wise}(\text{brother_of}(Y)) \mid \text{taller}(X,Y)$	
1st option - See binary resolution 1st option	
2nd option - See binary resolution 2nd option	
3rd option	
$c^l = \sim\text{wise}(Z)$	$d^l = \text{taller}(X,Y)$
$L_1 = \text{wise}(\text{brother_of}(Z))$	$\sim M_1 = \sim\text{wise}(X)$
	$\sim M_2 = \sim\text{wise}(\text{brother_of}(Y))$
$\theta = \{X/\text{brother_of}(Z), Y/Z\}$	
$c^l\theta = \sim\text{wise}(Z)$	$d^l\theta = \text{taller}(\text{brother_of}(Z), Z)$
$L_1\theta = \text{wise}(\text{brother_of}(Z))$	$M_1\theta = \text{wise}(\text{brother_of}(Z))$
	$M_2\theta = \text{wise}(\text{brother_of}(Z))$
$(c^l \mid d^l)\theta$	$= \sim\text{wise}(Z) \mid \text{taller}(\text{brother_of}(Z), Z)$

Resolution is a sound inference rule

If a set S of clauses is Herbrand satisfiable, then S U {a resolvent from S} is also Herbrand satisfiable.

(Or, equivalently, if a set S U {a resolvent from S} of clauses is Herbrand unsatisfiable, then S is Herbrand unsatisfiable.)

Proof	
Let $c^l \mid L_1 \mid \dots \mid L_n$ and $d^l \mid \sim M_1 \mid \dots \mid \sim M_k$ be two clauses in a Herbrand satisfiable set of clauses S, such that the L_i s and M_j s are unifiable with most general unifier θ , i.e. the clauses resolve to produce $(c^l \mid d^l)\theta$. Let σ be any Herbrand universe substitution so that $(c^l \mid d^l)\theta\sigma$ is ground, and γ any Herbrand universe substitution so that $(c^l \mid L_1 \mid \dots \mid L_n)\theta\sigma\gamma$ and $(d^l \mid \sim M_1 \mid \dots \mid \sim M_k)\theta\sigma\gamma$ are ground.	
$(c^l \mid L_1 \mid \dots \mid L_n)\theta\sigma\gamma$ and $(d^l \mid \sim M_1 \mid \dots \mid \sim M_k)\theta\sigma\gamma$ are TRUE in a Herbrand model H of S.	
<ul style="list-style-type: none"> • If $c^l\theta\sigma\gamma$ is TRUE in H then $(c^l \mid d^l)\theta\sigma\gamma$ is TRUE in H. • If some $L_i\theta\sigma\gamma$ is TRUE in H, then all $L_i\theta\sigma\gamma$ are TRUE in H, and all $\sim M_i\theta\sigma\gamma$ are FALSE in H. Then $d^l\theta\sigma\gamma$ must be TRUE in H and hence $(c^l \mid d^l)\theta\sigma\gamma$ is TRUE in H. 	
As σ and γ are arbitrary, the resolvent $(c^l \mid d^l)\theta$, is TRUE in H, i.e., S U $(c^l \mid d^l)\theta$ is satisfiable.	
Example	
Consider the resolution between the following two clauses, selected from a Herbrand satisfiable set S. Let H be a Herbrand model of S.	
$\sim\text{wise}(Z) \mid \text{wise}(\text{brother_of}(Z)) + \sim\text{wise}(X) \mid \sim\text{wise}(\text{brother_of}(Y)) \mid \text{taller}(X,Y)$	
$c^l = \sim\text{wise}(Z)$	$d^l = \text{taller}(X,Y)$
$L_1 = \text{wise}(\text{brother_of}(Z))$	$\sim M_1 = \sim\text{wise}(X)$
	$\sim M_2 = \sim\text{wise}(\text{brother_of}(Y))$
$\theta = \{X/\text{brother_of}(Z), Y/Z\}$	
$(c^l \mid d^l)\theta$	$= \sim\text{wise}(Z) \mid \text{taller}(\text{brother_of}(Z), Z)$
One possible σ is $\{Z/\text{jim}\}$, and γ can be $\{\}$. Applying $\theta\sigma\gamma$ to the two clauses gives:	
$(c^l \mid L_1)\theta\sigma\gamma = \sim\text{wise}(\text{jim}) \mid \text{wise}(\text{brother_of}(\text{jim}))$	
$(d^l \mid \sim M_1 \mid \sim M_2)\theta\sigma\gamma = \sim\text{wise}(\text{brother_of}(\text{jim})) \mid \sim\text{wise}(\text{brother_of}(\text{jim})) \mid \text{taller}(\text{brother_of}(\text{jim}), \text{jim})$	
which are both TRUE in H.	
If $c^l\theta\sigma\gamma = \sim\text{wise}(\text{jim})$ is TRUE in H then $(c^l \mid d^l)\theta\sigma\gamma = \text{taller}(\text{brother_of}(\text{jim}), \text{jim}) \mid \sim\text{wise}(\text{jim})$ is TRUE in H.	
If $L_1\theta\sigma\gamma = \text{wise}(\text{brother_of}(\text{jim}))$ is TRUE in H then $\sim M_1\theta\sigma\gamma = \sim M_2\theta\sigma\gamma = \sim\text{wise}(\text{brother_of}(\text{jim}))$ is FALSE in H and $d^l\theta\sigma\gamma = \text{taller}(\text{brother_of}(\text{jim}), \text{jim})$ must be TRUE in H. Hence $(c^l \mid d^l)\theta\sigma\gamma = \text{taller}(\text{brother_of}(\text{jim}), \text{jim}) \mid \sim\text{wise}(\text{jim})$ is TRUE in H.	
As σ and γ are arbitrary $(c^l \mid d^l)\theta = \text{taller}(\text{brother_of}(Z), Z) \mid \sim\text{wise}(Z)$ is TRUE in H.	

This theorem shows that all models of S are models of the resolvent. In other words, resolvents are logical consequences of the set. If S U {a resolvent from S} is Herbrand unsatisfiable then the S is Herbrand unsatisfiable. An empty clause is derived from a contradictory pair of

clauses, and the existence of such a contradictory pair means that the set cannot have any model, i.e., the set is (Herbrand) unsatisfiable.

The Quest for Logical Consequence

To show that C is a logical consequence of A, it is necessary to:

- Show that every model of A is a model of C
- OR
- Show that $S' = A \cup \{\sim C\}$ is unsatisfiable.

S' is unsatisfiable iff $S = \text{CNF}(S')$ is unsatisfiable

- Show that S is unsatisfiable.

A set of clauses has a model iff it has a Herbrand model, so

- Show that S is Herbrand unsatisfiable.

If a set $S \cup \{a \text{ resolvent from } S\}$ of clauses is Herbrand unsatisfiable, then S is Herbrand unsatisfiable, so

- Do resolution and hope an obviously unsatisfiable set is produced, e.g., one containing FALSE

Factoring

$$\begin{aligned}
 C \mid L_1 \mid \dots \mid L_n + \text{Factoring} \\
 L_i \theta = L_j \theta \\
 = (C \mid L_1) \theta
 \end{aligned}$$

Example

$\sim \text{wise}(X) \mid \sim \text{wise}(\text{brother_of}(Y)) \mid \text{taller}(X,Y) \mid \sim \text{wise}(\text{brother_of}(\text{jim}))$

1st option

$$\begin{aligned}
 C \mid &= \sim \text{wise}(X) \mid \text{taller}(X,Y) & + \theta &= \{Y/\text{jim}\} \\
 L_1 &= \sim \text{wise}(\text{brother_of}(Y)) \\
 L_2 &= \sim \text{wise}(\text{brother_of}(\text{jim})) \\
 & & & = \sim \text{wise}(X) \mid \text{taller}(X,\text{jim}) \mid \sim \text{wise}(\text{brother_of}(\text{jim}))
 \end{aligned}$$

2nd option

$$\begin{aligned}
 C \mid &= \sim \text{wise}(\text{brother_of}(Y)) \mid \text{taller}(X,Y) & + \theta &= \{X/\text{brother_of}(\text{jim})\} \\
 L_1 &= \sim \text{wise}(X) \\
 L_2 &= \sim \text{wise}(\text{brother_of}(\text{jim})) \\
 & & & = \sim \text{wise}(\text{brother_of}(Y)) \mid \text{taller}(\text{brother_of}(\text{jim}),Y) \mid \\
 & & & \quad \sim \text{wise}(\text{brother_of}(\text{jim}))
 \end{aligned}$$

3rd option

$$\begin{aligned}
 C \mid &= \sim \text{wise}(\text{brother_of}(\text{jim})) \mid & + \theta &= \{X/\text{brother_of}(Y)\} \\
 \text{taller}(X,Y) \\
 L_1 &= \sim \text{wise}(X) \\
 L_2 &= \sim \text{wise}(\text{brother_of}(Y)) \\
 & & & = \sim \text{wise}(\text{brother_of}(\text{jim})) \mid \text{taller}(\text{brother_of}(Y),Y) \mid \sim \text{wise}(\text{brother_of}(Y))
 \end{aligned}$$

4th option

$$\begin{aligned}
 C \mid &= \text{taller}(X,Y) & + \theta &= \{X/\text{brother_of}(\text{jim}), Y/\text{jim}\} \\
 L_1 &= \sim \text{wise}(X) \\
 L_2 &= \sim \text{wise}(\text{brother_of}(Y)) \\
 L_3 &= \sim \text{wise}(\text{brother_of}(\text{jim})) \\
 & & & = \text{taller}(\text{brother_of}(\text{jim}),\text{jim}) \mid \sim \text{wise}(\text{brother_of}(\text{jim}))
 \end{aligned}$$

Factoring is used in conjunction with binary resolution to produce full resolution.

Example

Full resolution

$$\begin{aligned}
 \sim \text{wise}(X) \mid \sim \text{wise}(\text{brother_of}(Y)) \mid \text{taller}(X,Y) & + \sim \text{wise}(Z) \mid \text{wise}(\text{brother_of}(Z)) \\
 & = \text{taller}(\text{brother_of}(Y),Y) \mid \sim \text{wise}(Y)
 \end{aligned}$$

Factoring and Binary resolution

$$\sim \text{wise}(X) \mid \sim \text{wise}(\text{brother_of}(Y)) \mid \text{taller}(X,Y) + \text{Factoring}$$

$$\begin{aligned}
 &= \sim\text{wise}(\text{brother_of}(Y)) \mid \text{taller}(\text{brother_of}(Y), Y) \\
 \sim\text{wise}(\text{brother_of}(Y)) \mid \text{taller}(\text{brother_of}(Y), Y) + \sim\text{wise}(Z) \mid \text{wise}(\text{brother_of}(Z)) \\
 &= \text{taller}(\text{brother_of}(Y), Y) \mid \sim\text{wise}(Y)
 \end{aligned}$$

All full resolvents of two clauses are formed from all binary resolvents of the two clauses, and all binary resolvents of all factors of the two clauses. Some resolution strategies are complete using binary resolution and factoring, with the application of factoring restricted to certain clauses. This format produces a smaller search space than using full resolution (which in effect allows factoring on all clauses).

Exam Style Questions

1. What is the most general unifier of the following atoms:

$p(X, f(Y), Z)$
 $p(T, T, g(\text{cat}))$
 $p(f(\text{dog}), S, g(W))$

2. What is the disagreement set of the following atoms:

$p(X, f(Y), Z)$
 $p(T, T, g(\text{cat}))$
 $p(f(\text{dog}), S, g(W))$

3. List all the binary resolvents of the following two clauses:

$p(X, f(Y), Z) \mid p(T, T, g(\text{cat})) \mid r(X, T) \mid \sim s(Z, T)$
 $\sim p(f(\text{dog}), S, g(W)) \mid s(\text{big}, \text{rat}) \mid \sim s(\text{small}, \text{hamster})$

4. List all the resolvents of the following two clauses:

$p(X, f(Y), Z) \mid p(T, T, g(\text{cat})) \mid r(X, T) \mid \sim s(Z, T)$
 $\sim p(f(\text{dog}), S, g(W)) \mid s(\text{big}, \text{rat}) \mid \sim s(\text{small}, \text{hamster})$

5. List all the factors of the following clause:

$p(X, f(Y), Z) \mid \sim s(Z, T) \mid p(T, T, g(\text{cat})) \mid p(f(\text{dog}), S, g(W)) \mid \sim s(\text{small}, \text{hamster})$

6. One resolvent of the clauses:

$p(X, f(Y), Z) \mid p(T, T, g(\text{cat})) \mid r(X, T) \mid \sim s(Z, T)$
 $\sim p(f(\text{dog}), S, g(W)) \mid s(\text{big}, \text{rat}) \mid \sim s(\text{small}, \text{hamster})$

is

$r(f(\text{dog}), f(\text{dog})) \mid \sim s(g(\text{cat}), f(\text{dog})) \mid s(\text{big}, \text{rat}) \mid \sim s(\text{small}, \text{hamster})$

Show how that may be formed by factoring and binary resolution.

7. Prove that if a set S of clauses is Herbrand satisfiable, then $S \cup \{a \text{ resolvent from } S\}$ is also Herbrand satisfiable
8. Use resolution to derive the empty clause from the set

$S = \{ \sim p(X) \mid \sim p(f(X)),$
 $p(f(X)) \mid p(X)$
 $\sim p(X) \mid p(f(X)) \}$

The Resolution Procedure

The basic resolution procedure algorithm is :

```
While a refutation has not been found
  Copy two clauses from the set
  Resolve the clauses
  Put resolvents into the set
```

Note the "copy", indicating that a clause may be used multiple times with a new set of variables each time.

Example	
$S = \{ \text{wise}(\text{jim}), \sim\text{wise}(X) \mid \text{wise}(\text{brother_of}(X)), \sim\text{wise}(\text{brother_of}(\text{brother_of}(\text{jim}))) \}$	
$\sim\text{wise}(X) \mid \text{wise}(\text{brother_of}(X))$	$+ \sim\text{wise}(\text{brother_of}(\text{brother_of}(\text{jim})))$
	$= \sim\text{wise}(\text{brother_of}(\text{jim}))$
$\sim\text{wise}(\text{brother_of}(\text{jim}))$	$+ \sim\text{wise}(X) \mid \text{wise}(\text{brother_of}(X))$
	$= \sim\text{wise}(\text{jim})$
$\text{wise}(\text{jim})$	$+ \sim\text{wise}(\text{jim})$
	$= \text{FALSE}$

Example	
$S = \{ p(f(X)) \mid q(Y), \sim p(X) \mid q(\text{dog}), p(f(\text{dog})) \mid \sim q(X), \sim p(f(Z)) \}$	
$p(f(X)) \mid q(Y)$	$+ \sim p(X) \mid q(\text{dog})$
	$= q(\text{dog}) \mid q(Y)$
$p(f(\text{dog})) \mid \sim q(X)$	$+ \sim p(f(Z))$
	$= \sim q(X)$
$q(\text{dog}) \mid q(Y)$	$+ \sim q(X)$
	$= \text{FALSE}$

Example	
Axioms	
$\forall M1, M2 (\text{member}(M1) \ \& \ \text{member}(M2) \ \& \ \text{shaved}(M1, M2)) \Rightarrow \text{all_shaved}(M1)$ $\text{member}(\text{guido})$ $\text{member}(\text{lorenzo})$ $\text{member}(\text{petrucio})$ $\text{member}(\text{cesare})$ $\text{shaved}(\text{guido}, \text{cesare})$ $\forall M1 (\text{all_shaved}(M1) \Leftrightarrow (\text{member}(M1) \ \& \ \forall M2 (\text{member}(M2) \Rightarrow \text{shaved}(M2, M1))))$	
Conjecture	
$\text{shaved}(\text{petrucio}, \text{lorenzo})$	
Axioms U {~C} Converted to CNF	
$\text{all_shaved}(A) \mid \sim\text{member}(B) \mid \sim\text{member}(A) \mid \sim\text{shaved}(A, B)$ $\text{member}(\text{guido})$ $\text{member}(\text{lorenzo})$ $\text{member}(\text{petrucio})$ $\text{member}(\text{cesare})$ $\text{shaved}(\text{guido}, \text{cesare})$ $\text{member}(A) \mid \sim\text{all_shaved}(A)$ $\text{shaved}(B, A) \mid \sim\text{all_shaved}(A) \mid \sim\text{member}(B)$ $\text{all_shaved}(A) \mid \text{member}(\text{sk1}(A)) \mid \sim\text{member}(A)$ $\text{all_shaved}(A) \mid \sim\text{member}(A) \mid \sim\text{shaved}(\text{sk1}(A), A)$ $\sim\text{shaved}(\text{petrucio}, \text{lorenzo})$	
Refutation	

```

member(cesare)      + all_shaved(A) | ~member(B) | ~member(A) | ~shaved(A,B)
                   = all_shaved(A) | ~member(A) | ~shaved(A,cesare)
member(guido)      + all_shaved(A) | ~member(A) | ~shaved(A,cesare)
                   = all_shaved(guido) | ~shaved(guido,cesare)
shaved(guido,cesare) + all_shaved(guido) | ~shaved(guido,cesare)
                   = all_shaved(guido)
member(lorenzo)    + shaved(B,A) | ~all_shaved(A) | ~member(B)
                   = shaved(lorenzo,A) | ~all_shaved(A)
all_shaved(guido)  + shaved(lorenzo,A) | ~all_shaved(A)
                   = shaved(lorenzo,guido)
member(guido)      + all_shaved(A) | ~member(B) | ~member(A) | ~shaved(A,B)
                   = all_shaved(A) | ~member(A) | ~shaved(A,guido)
member(lorenzo)    + all_shaved(A) | ~member(A) | ~shaved(A,guido)
                   = all_shaved(lorenzo) | ~shaved(lorenzo,guido)
shaved(lorenzo,guido) + all_shaved(lorenzo) | ~shaved(lorenzo,guido)
                   = all_shaved(lorenzo)
member(petrucio)   + shaved(B,A) | ~all_shaved(A) | ~member(B)
                   = shaved(petrucio,A) | ~all_shaved(A)
all_shaved(lorenzo) + shaved(petrucio,A) | ~all_shaved(A)
                   = shaved(petrucio,lorenzo)
~shaved(petrucio,lorenzo) + shaved(petrucio,lorenzo)
                   = FALSE

```

Full resolution is refutation complete for general clauses [Robinson 1965], and binary resolution is refutation complete for Horn clauses [Henschen & Wos 1974]. This means that if the input set is unsatisfiable, then the empty clause can always be derived in this manner, given infinite time and memory (i.e., resolution is theoretically refutation complete, practically incomplete). Such a deduction is called a refutation. If the input set is satisfiable, the resolution procedure may continue for ever. This is expected as 1st order logic is semi-decidable.

Example (showing that binary resolution is not refutation complete)

```

S = { p(X) | q(Y) | q(X)
      ~q(a) | ~q(Z),
      ~p(a) }

```

Full resolution

```

p(X) | q(Y) | q(X) + ~q(a) | ~q(Z)
                   = p(a)
~p(a)              + p(a)
                   = FALSE

```

Binary resolution

Try it, and fail

Proof Trees

A refutation can be represented as a proof tree.

Example

```

S2 = { ~p(X) | q(X) | r(X),
        p(a) | p(b),
        ~q(Y),
        ~r(a),
        ~r(b) }

```

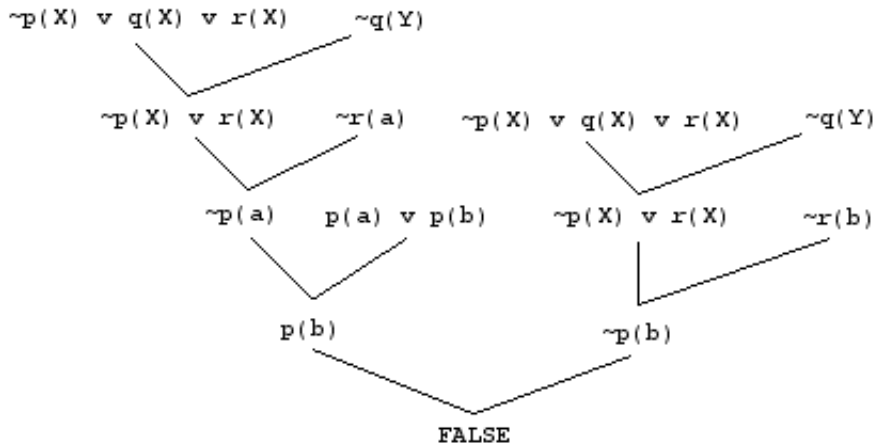
```

~p(X) | q(X) | r(X) + ~q(Y)
                   = ~p(X) | r(X)
~p(X) | r(X)      + ~r(a)
                   = ~p(a)
~p(X) | r(X)      + ~r(b)
                   = ~p(b)

```

$\sim p(a)$	$+ p(a) \mid p(b)$
	$= p(b)$
$p(b)$	$+ \sim p(b)$
	$= \text{FALSE}$

This refutation has the proof tree :



Notice that the clause $\sim p(X) \mid r(X)$ is used twice in the proof tree but is generated only once by resolution.

Axiomatic Proofs

An axiomatic proof is one in which the conjecture is derived step-by-step from the axioms and inferred formulae, using sound inferences. A refutation can often be rearranged to form an axiomatic proof.

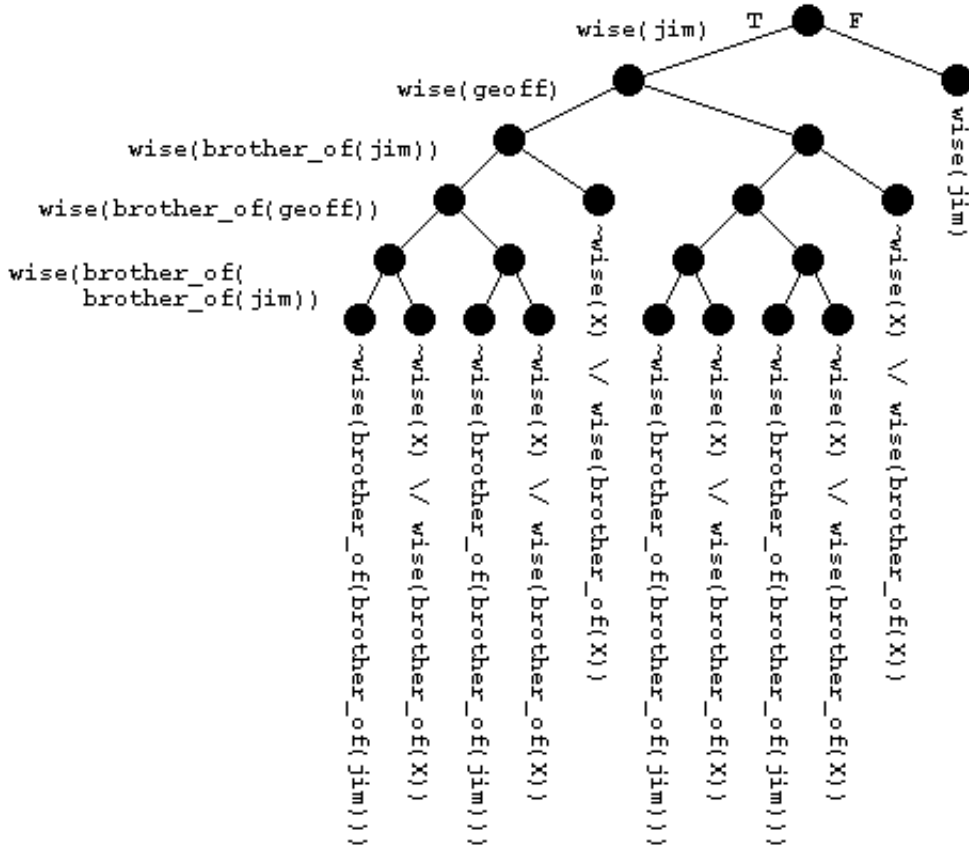
Example
$A = \{ \forall X (p(X) \Rightarrow q(X) \mid r(X)),$ $p(a) \mid p(b),$ $\forall Y \sim q(Y) \}$
$C = r(a) \mid r(b)$
The CNF of A is:
$S = \{ \sim p(X) \mid q(X) \mid r(X),$ $p(a) \mid p(b),$ $\sim q(Y) \}$
An axiomatic proof of $r(a) \mid r(b)$ from S is:
$\sim p(X) \mid q(X) \mid r(X) + \sim q(Y)$ $\quad \quad \quad = \sim p(X) \mid r(X)$
$\sim p(X) \mid r(X) \quad + p(a) \mid p(b)$ $\quad \quad \quad = r(a) \mid p(b)$
$\sim p(X) \mid r(X) \quad + r(a) \mid p(b)$ $\quad \quad \quad = r(a) \mid r(b)$

Failure Trees

A failure tree for an unsatisfiable set of clauses is a minimal finite subtree of the semantic tree of the Herbrand base, such that for each branch there is a clause that is FALSE in the sub-interpretation identified by that branch. FALSE has a single node as its failure tree.

Example
$S = \{ \text{wise}(\text{jim}),$ $\sim \text{wise}(X) \mid \text{wise}(\text{brother_of}(X)),$ $\sim \text{wise}(\text{brother_of}(\text{brother_of}(\text{jim}))) \}$

This unsatisfiable clause set has the failure tree :



Example
$S = \{ p(a) \mid q(X), \\ \sim p(Y) \mid q(f(Y)), \\ p(a) \mid \sim q(T), \\ \sim p(A) \mid \sim q(W) \}$

A (Herbrand) unsatisfiable set of clauses has a failure tree

Proof

If a set of clauses is (Herbrand) unsatisfiable then for each branch of the semantic tree of the Herbrand base, i.e., for each Herbrand interpretation, there exists a clause in the set which has a ground instance that is FALSE in the Herbrand interpretation identified by that branch.

As there are only a finite number of literals in each clause, each branch has a smallest sub-branch such that the truth value of the corresponding clause instance is determined by the sub-interpretation identified by the sub-branch. The leaf of such a sub-branch is called a failure node for the clause. The sub-branches define a finite sub-tree of the semantic tree. Prune any descendants of failure nodes to form a minimal sub-tree, i.e. a failure tree.

Failures trees are used in the proofs of Herbrand's theorem and the completeness of resolution.

Refutation Completeness

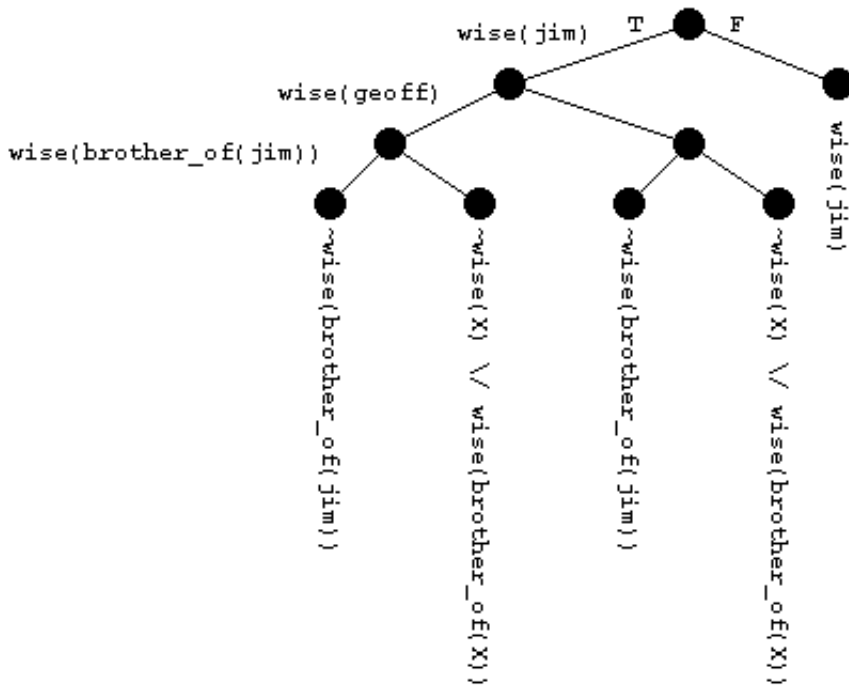
The resolution procedure is refutation complete

Proof

For a set of clauses S, let R(S) mean S U {all resolvents from S}, and Rⁿ(S) mean R applied n times. If S is unsatisfiable, then it needs to be shown that there exists n such that FALSE is an element of Rⁿ(S).

If S is unsatisfiable, then it has a failure tree. If the tree is trivial, then FALSE is an element of S. Otherwise:

- Pick a failure node Y of maximum depth in the failure tree. Let X be the parent of Y. X is not a failure node as a failure tree is minimal. Let Z be the other offspring, which must also be a failure node as Y is of maximum depth.
- Let κ be the Herbrand base element whose truth value is determined by the arcs X-Y (FALSE) and X-Z (TRUE).
- As Y and Z are failure nodes, there exist clauses that are FALSE at Y and Z respectively. Since both these clauses are TRUE at X (X is



The Quest for Logical Consequence

To show that C is a logical consequence of A , it is necessary to:

- Show that every model of A is a model of C
- OR
- Show that $S' = A \cup \{\sim C\}$ is unsatisfiable.

S' is unsatisfiable iff $S = \text{CNF}(S')$ is unsatisfiable

- Show that S is unsatisfiable.

A set of clauses has a model iff it has a Herbrand model, so

- Show that S is Herbrand unsatisfiable.

If a set $s \cup \{a \text{ resolvent from } s\}$ of clauses is Herbrand unsatisfiable, then s is Herbrand unsatisfiable, so

- Do resolution and hope an obviously unsatisfiable set is produced, e.g., one containing **FALSE**

The resolution procedure is refutation complete, so

- Confidently do the resolution procedure until **FALSE** is produced

Exam Style Questions

1. Write out the basic resolution procedure algorithm.
2. What things could happen if the basic resolution procedure algorithm is run on an unsatisfiable set of clauses?
3. What things could happen if the basic resolution procedure algorithm is run on a satisfiable set of clauses?
4. In what circumstances does the basic resolution procedure algorithm retain refutation completeness when using only binary resolution?
5. Draw the proof tree for the following refutation:

$$\begin{array}{l}
 \sim p(X) \mid q(X) \mid r(X) + \sim q(Y) \\
 \qquad \qquad \qquad = \sim p(X) \mid r(X) \\
 \sim p(X) \mid r(X) \qquad + \sim r(a) \\
 \qquad \qquad \qquad = \sim p(a) \\
 \sim p(X) \mid r(X) \qquad + \sim r(b) \\
 \qquad \qquad \qquad = \sim p(b) \\
 \sim p(a) \qquad \qquad + p(a) \mid p(b) \\
 \qquad \qquad \qquad = p(b) \\
 p(b) \qquad \qquad \qquad + \sim p(b)
 \end{array}$$

= FALSE

6. What is an axiomatic proof?

7. Draw the failure tree for the following set of clauses:

$$S = \{ p(f(x)) \mid q(y), \\ \sim p(x) \mid q(\text{dog}), \\ p(f(\text{dog})) \mid \sim q(x), \\ \sim p(f(z)) \}$$

8. Prove that a Herbrand unsatisfiable set of clauses has a failure tree.

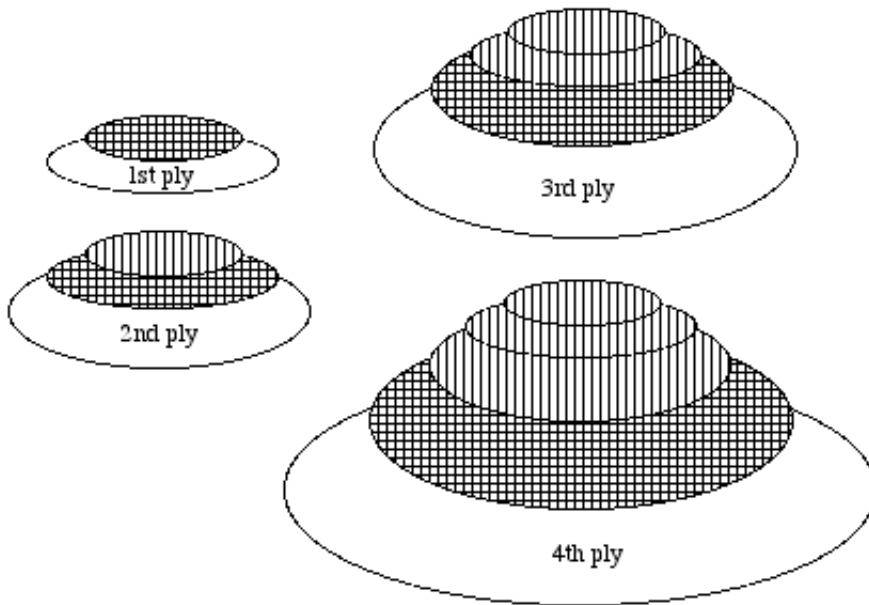
9. Prove that the resolution procedure is refutation complete.

The Search Problem

In the resolution procedure there are choices to be made

- Which chosen clause to use
- Which other clause to use
- Which literals to resolve upon

The combination of decisions made for these choices guide the search for a refutation. Given the decisions made, a single resolvent can be generated. Regardless of the search strategy used, some mechanism is required to prevent the same resolution step being repeated. The basis of the *ANL Loop* (so named, as it was first used in ATP systems developed at Argonne National Laboratory, the most famous of which is now Otter), is to divide the resolvents that can be generated into plies. Each ply contains resolvents that are the same number of resolution steps away from input clauses. Diagrammatically, this can be thought of as follows :



where at each iteration one parent is chosen from the cross-hatched area and the other is chosen from the cross-hatched or vertical striped area. The resolvents are put into the blank area, ready for the next iteration. Evidently a duplication free search is being performed.

Example
$S1 = \{ p(b) \mid r(Y) \mid p(Y), \\ \sim p(S) \mid p(b), \\ \sim p(b), \\ \sim r(a), \\ \sim r(c) \}$
<p>1st ply</p> $p(b) \mid r(Y) \mid p(Y) + \sim p(S) \mid p(b) \\ = r(Y) \mid p(Y) \mid p(b) \\ p(b) \mid r(Y) \mid p(b) \\ r(Y) \mid p(b)$ $p(b) \mid r(Y) \mid p(Y) + \sim p(b) \\ = r(Y) \mid p(Y) \\ p(b) \mid r(b) \\ r(b)$ $p(b) \mid r(Y) \mid p(Y) + \sim r(a) \\ = p(b) \mid p(a)$ $p(b) \mid r(Y) \mid p(Y) + \sim r(c) \\ = p(b) \mid p(c)$ $\sim p(S) \mid p(b) + \sim p(b) \\ = \sim p(S)$
<p>2nd ply</p> $r(Y) \mid p(Y) \mid p(b) + \sim p(S) \mid p(b)$

	= r(Y)	p(b)	p(b)
	r(Y)	p(Y)	p(b)
	r(b)	p(b)	
r(Y) p(Y) p(b) + ~p(b)			
	= r(b)	p(b)	
	r(Y)	p(Y)	
	r(b)		
r(Y) p(Y) p(b) + ~r(a)			
	= p(a)	p(b)	
r(Y) p(Y) p(b) + ~r(c)			
	= p(c)	p(b)	
r(Y) p(Y) p(b) + ~p(S)			
	= r(Y)	p(b)	
	r(Y)	p(Y)	
	r(b)		
p(b) r(Y) p(b) + ~p(S) p(b)			
	= r(Y)	p(b)	p(b)
	p(b)	r(Y)	p(b)
	r(b)	p(b)	
p(b) r(Y) p(b) + ~p(b)			
	= r(Y)	p(b)	
	p(b)	r(Y)	
	r(Y)		
p(b) r(Y) p(b) + ~r(a)			
	= p(b)	p(b)	
p(b) r(Y) p(b) + ~r(c)			
	= p(b)	p(b)	
p(b) r(Y) p(b) + ~p(S)			
	= r(Y)	p(b)	
	p(b)	r(Y)	
	r(b)		
r(Y) p(b)	+ ~p(S)	p(b)	
	= r(Y)	p(b)	
r(Y) p(b)	+ ~p(b)		
	= r(Y)		
r(Y) p(b)	+ ~r(a)		
	= p(b)		
r(Y) p(b)	+ ~r(c)		
	= p(b)		
r(Y) p(b)	+ ~p(S)		
	= r(Y)		
r(Y) p(Y)	+ ~p(S) p(b)		
	= r(Y) p(b)		
r(Y) p(Y)	+ ~p(b)		
	= r(b)		
r(Y) p(Y)	+ ~r(a)		
	= p(a)		
r(Y) p(Y)	+ ~r(c)		
	= p(c)		
r(Y) p(Y)	+ ~p(S)		
	= r(Y)		
p(b) r(b)	+ ~p(S) p(b)		
	= r(b) p(b)		
p(b) r(b)	+ ~p(b)		
	= r(b)		
p(b) r(b)	+ ~p(S)		
	= r(b)		
p(b) p(a)	+ ~p(S) p(b)		
	= p(a) p(b)		
	p(b) p(b)		

$p(b) \mid p(a)$	$+ \sim p(b)$
	$= p(a)$
$p(b) \mid p(a)$	$+ \sim p(S)$
	$= p(a)$
	$p(b)$
$p(b) \mid p(c)$	$+ \sim p(S) \mid p(b)$
	$= p(c) \mid p(b)$
	$p(b) \mid p(b)$
$p(b) \mid p(c)$	$+ \sim p(b)$
	$= p(c)$
$p(b) \mid p(c)$	$+ \sim p(S)$
	$= p(c)$
	$p(b)$
3rd ply	
Lots and lots	
$p(b)$	$+ \sim p(b)$
	$= \text{FALSE}$

Clearly things can get out-of-hand very quickly. The total number of clauses typically grows exponentially with the number of plys. It is necessary to:

- Be selective about which resolvents are generated when.
- Use restriction strategies to prevent certain resolvents from being generated. A common effect of restrictions is to force a longer refutation while restricting the choices.
- Use deletion strategies to delete unnecessary clauses.

The ANL Loop

The ANL loop is a flexible algorithm for controlling the generation of resolvents.

```

Let CanBeUsed = {}
Let ToBeUsed = Input clauses
While a refutation has not been found
  Select the ChosenClause from ToBeUsed
  Move the ChosenClause to CanBeUsed
  Infer all possible clauses using the ChosenClause and other clauses from CanBeUsed.
  Add the inferred clauses to ToBeUsed

```

Depending on how the ChosenClause is selected from the ToBeUsed set, different search strategies can be implemented.

- Depth first search
 - Select a most recently created resolvent as the ChosenClause.
 - Does not guarantee a complete search (could get into an infinite loop)
 - Does not guarantee finding the shortest refutation.
- Breadth first search
 - Select a least recently created resolvent as the ChosenClause.
 - Will find the shortest refutation.
 - Implements a ply-by-ply search.
- Best first search
 - Select the 'best' clause as the ChosenClause. the best possible literals.
 - The notion of "best" is determined by a heuristic function.

Example, using BFS

```

S1 = { p(b) | r(Y) | p(Y),
      ~p(S) | p(b),
      ~p(b),
      ~r(a),
      ~r(c) }

p(b) | r(Y) | p(Y) is the ChosenClause
~p(S) | p(b)      is the ChosenClause
~p(S) | p(b)      + p(b) | r(Y) | p(Y)
                  = r(Y) | p(Y) | p(b)
                  p(b) | r(Y) | p(b)
                  r(b) | p(b)

~p(b) is the ChosenClause

```

```

~p(b)          + p(b) | r(Y) | p(Y)
               = r(Y) | p(Y)
               p(b) | r(Y)
               r(b)

~p(b)          + ~p(S) | p(b)
               = ~p(S)

~r(a)          is the ChosenClause
~r(a)          + p(b) | r(Y) | p(Y)
               = p(b) | p(a)

~r(c)          is the ChosenClause
~r(c)          + p(b) | r(Y) | p(Y)
               = p(b) | p(c)

r(Y) | p(Y) | p(b) is the ChosenClause
r(Y) | p(Y) | p(b) + ~p(S) | p(b)
               = r(Y) | p(b) | p(b) p(b) | p(c)
               r(Y) | p(Y) | p(b)
               r(Y) | p(b)

r(Y) | p(Y) | p(b) + ~p(b)
               = r(Y) | p(b)          r(Y) | p(Y) r(b)

r(Y) | p(Y) | p(b) + ~r(a)
               = p(a) | p(b)

r(Y) | p(Y) | p(b) + ~r(c)
               = p(c) | p(b)

etc, etc, etc

```

Example

```

S = { ~n | ~t,
      m | q | n,
      l | ~m,
      l | ~q,
      ~l | ~p,
      r | p | n,
      ~r | ~l,
      t }

```

Example

```

S = { ~p(b),
      p(X) | l(X) | q,
      ~q | l(Y),
      ~l(Z) | r(Z),
      ~r(b) | ~l(T) }

```

Exam Style Questions

1. Write out the ANL loop algorithm.
2. Show the execution of the ANL loop to derive the empty clause from:

```

S = { ~r(Y) | ~p(Y),
      p(b),
      r(a),
      p(S) | ~p(b) | ~r(S),
      r(c) }

```

Use any selection strategy you want, and number the chosen clause at each iteration.

General Purpose Heuristics

Unit Preference Strategy

This strategy causes unit clauses to be selected from the ToBeUsed set in preference to non-unit clauses.

Example, using unit preference	
$S1 = \{ p(b) \mid r(Y) \mid p(Y),$ $\quad \sim p(S) \mid p(b),$ $\quad \sim p(b),$ $\quad \sim r(a),$ $\quad \sim r(c) \}$	
$\sim p(b)$	is the ChosenClause
$\sim r(a)$	is the ChosenClause
$\sim r(c)$	is the ChosenClause
$\sim p(S) \mid p(b)$	is the ChosenClause
$\sim p(S) \mid p(b)$	+ $\sim p(b)$ = $\sim p(S)$
$\sim p(S)$	is the ChosenClause
$\sim p(S)$	+ $\sim p(S) \mid p(b)$ = $\sim p(S)$
$\sim p(S)$	is a loop (more about this later)
$p(b) \mid r(Y) \mid p(Y)$	is the ChosenClause
$p(b) \mid r(Y) \mid p(Y) + \sim p(b)$	= $r(Y) \mid p(Y)$ + $\sim r(a)$ = $p(b) \mid p(a)$ + $\sim r(c)$ = $p(b) \mid p(c)$ + $\sim p(S) \mid p(b)$ = $p(b) \mid p(b) \mid r(Y) \mid p(Y)$ + $\sim p(S)$ = $r(Y) \mid p(Y)$
$r(Y) \mid p(Y)$	is the ChosenClause
$r(Y) \mid p(Y)$	+ $\sim p(b)$ = $r(b)$ + $\sim r(a)$ = $p(a)$ + $\sim r(c)$ = $p(c)$ + $\sim p(S) \mid p(b)$ = $r(Y) \mid p(b)$ + $\sim p(S)$ = $r(Y)$
$r(b)$	is the ChosenClause
$p(a)$	is the ChosenClause
$p(a)$	+ $\sim p(S) \mid p(b)$ = $p(b)$ + $\sim p(S)$ = FALSE

Least Symbol Count

This strategy selects the clause with the least number of symbols from the ToBeUsed set.

Example, using least symbol count

$$S5 = \{ \sim p1(X, f(Y)) \mid q1(f(X), Y) \mid q2(X, f(Y)), \\ p1(a, Y), \\ \sim q2(X, f(Y)), \\ \sim p2(X, f(Y)) \mid q3(f(X), Y) \mid q4(X, f(Y)), \\ \sim q3(X, g(Y)), \\ \sim q4(f(f(X)), f(g(f(Y))))), \\ \sim q1(X, f(Y)) \mid p2(f(X), Y) \}$$

General Purpose Restrictions

Pure Literal Deletion

If an input clause has a literal for which there is no complimentary unifiable literal in the input set, then that literal can never be resolved upon. Such literals are said to be *pure*. Clauses in the input set containing pure literals can be discarded. By discarding such clauses, more pure literals may be created. The process continues until no pure literals remain.

Example
$S = \{ \begin{array}{l} r(X) \mid p(X,a), \\ \neg p(Y,Z) \mid q(Z,Y), \\ \neg q(a,b) \mid s, \\ \neg s, \\ q(a,T) \end{array} \}$
$r(X) \mid p(X,a)$ is pure by $r(X)$
$\neg p(Y,Z) \mid q(Z,Y)$ is pure by $\neg p(Y,Z)$

Tautology Deletion

In the basic resolution procedure, and most refinements of the resolution procedure, it can be shown that tautologies cannot be part of a refutation. Thus tautologous clauses can be discarded. Tautology deletion is applied to the input set and to each resolvent inferred.

Example with contrived control
$S = \{ \begin{array}{l} r(X) \mid p(X,a), \\ \neg p(a,Y) \mid \neg r(Y), \\ \neg p(b,Y) \mid \neg r(Y), \\ \neg r(b), \\ r(a) \end{array} \}$
$r(X) \mid p(X,a)$ is the ChosenClause
$\begin{aligned} r(X) \mid p(X,a) + \neg p(a,Y) \mid \neg r(Y) \\ = r(a) \mid \neg r(a) \\ \text{is a tautology} \\ + \neg p(b,Y) \mid \neg r(Y) \\ = r(b) \mid \neg r(a) \\ + \neg r(b) \\ = p(b,a) \end{aligned}$
$r(b) \mid \neg r(a)$ is the ChosenClause
etc, etc

Subsumption

Subsumption prevents the use of clauses that contain information that is the same as, or more specific than, information in other clauses. For example, if it is known that "everyone tells the truth", there is no point in also knowing (storing the information) that "Geoff tells the truth". The earlier information subsumes the latter. In terms of clauses, subsumption checks if a (subsuming) clause can do everything that a (subsumed) clause could do. If that is the case, the subsumed clause can be discarded because the subsuming clause can be used instead.

Clause c subsumes clause d iff there is an instance $c|\theta$ of c such that each literal of $c|\theta$ is a literal of d .

Example	
$\text{wise}(\text{geoff})$	$\text{wise}(X)$
Subsumed by	$\theta = \{X/\text{geoff}\}$
$\text{taller}(\text{jim}, \text{geoff}) \mid \text{wise}(\text{geoff}) \mid \text{taller}(X, \text{brother_of}(X))$	
Subsumed by	$\text{wise}(Y) \mid \text{taller}(\text{jim}, Z)$
	$\theta = \{Y/\text{geoff}, Z/\text{geoff}\}$

```
taller(jim,geoff)
```

```
Subsumed by
```

```
taller(jim,geoff) | taller(jim,Y) | taller(Z,geoff)
```

```
 $\theta = \{Y/geoff, Z/jim\}$ 
```

θ subsumption insists that, in addition to the above restriction, the subsuming clause must have no more literals than the subsumed clause. When using binary resolution and factoring, θ subsumption must be used to maintain refutation completeness.

Introspective subsumption occurs when a clause in a set of clauses is subsumed by another clause in the set, in which case the subsumed clause is discarded. Introspective subsumption may be done on the input set. *Forward subsumption* occurs when a when inferred clause is subsumed by an existing clause, in which case the inferred clause is discarded. *Backward subsumption* occurs when an existing clause is subsumed by an inferred clause, in which case the inferred clause is discarded. When a new clause is inferred, forward subsumption is applied, and if the clause is not forward subsumed, backwards subsumption is applied.

The ANL algorithm is extended with subsumption as follows:

```
Apply introspective subsumption to Input clauses
```

```
Let CanBeUsed = {}
```

```
Let ToBeUsed = Input clauses
```

```
While a refutation has not been found
```

```
  Select the ChosenClause from ToBeUsed
```

```
  Move the ChosenClause to CanBeUsed
```

```
  Infer all possible Resolvants using the ChosenClause and other clauses from CanBeUsed.
```

```
  As each Resolvant is inferred do
```

```
    If the Resolvant is forward subsumed by any clause in ToBeUsed or CanBeUsed then
```

```
      Discard the resolvant
```

```
    Else Discard any clauses in ToBeUsed or CanBeUsed that are backward subsumed by the resolvant
```

```
      If the resolvant subsumed the ChosenClause then
```

```
        Discard all other resolvants from this iteration of the loop
```

```
      Add the resolvant to ToBeUsed
```

Subsumption is a critical component of any ATP system. Beware: some refinements of the resolution procedure are not compatible with subsumption checking.

The ANL Algorithm with Restrictions

Repetitively pure literal deletion, tautology deletion, and introspective subsumption to the Input clauses, until no simplification is possible

```
Let CanBeUsed = {}
```

```
Let ToBeUsed = Input clauses
```

```
While a refutation has not been found
```

```
  Select the ChosenClause from ToBeUsed
```

```
  Move the ChosenClause to CanBeUsed
```

```
  Infer all possible clauses using the ChosenClause and other clauses from CanBeUsed.
```

```
  As each clause is inferred do
```

```
    If the inferred clause is a tautology or pure then
```

```
      Discard the clause
```

```
    ElseIf the inferred clause is forward subsumed by any clause in ToBeUsed or CanBeUsed then
```

```
      Discard the inferred clause
```

```
    Else Discard any clauses in ToBeUsed or CanBeUsed that are backward subsumed by the inferred clause
```

```
      If the inferred clause subsumed the ChosenClause then
```

```
        Discard all other inferred clauses from this iteration of the loop
```

```
      Add the inferred clause to ToBeUsed
```

Example

```
S1 = { p(b) | r(Y) | p(Y),  
      t(Y,c) | r(a),  
      ~p(S) | p(b),  
      ~r(a) | ~q(a),  
      ~r(b) | r(b),  
      q(a) | ~t(X,c),  
      ~p(b),  
      ~r(a),  
      ~r(c) }  
}
```

```
~r(b) | r(b)      is a tautology
```

```
~r(a) | ~q(a)     is subsumed by ~r(a)
```

```
q(a) | ~t(X,c)   is pure
```

```
t(Y,c) | r(a)    is pure
```

```
~p(b)            is the ChosenClause
```

```

~r(a)           is the ChosenClause
~r(c)           is the ChosenClause
~p(S) | p(b)   is the ChosenClause
~p(S) | p(b)   + ~p(b)
                = ~p(S)
Backward subsumes ~p(S) | p(b) (the ChosenClause)
                  ~p(b)
~p(S)           is the ChosenClause
p(b) | r(Y) | p(Y) is the ChosenClause
p(b) | r(Y) | p(Y) + ~r(a)
                  = p(b) | p(a)
                  + ~r(c)
                  = p(b) | p(c)
                  + ~p(S)
                  = r(Y) | p(Y)
Backward subsumes p(b) | r(Y) | p(Y) (the ChosenClause)
p(b) | p(a)     is the ChosenClause
p(b) | p(a)     + ~p(S)
                = p(a)
Backward subsumes p(b) | p(a) (the ChosenClause)
p(a)            is the ChosenClause
p(a)            + ~p(S)
                = FALSE

```

Linear Input (binary) Resolution

Given an input set of clauses and a clause C_1 chosen from the input set, a linear input deduction of C_n from the input set, with top clause C_1 , is a sequence of centre clauses C_1, \dots, C_n . Each deduced clause C_{i+1} , $i = 1..n-1$, is deduced from the centre clause C_i and a side clause chosen from the input set, by binary resolution. For any C_i , the centre clauses C_1 to C_{i-1} are the ancestor clauses of C_i . Linear input deduction is a refinement of linear deduction which does not permit any form of ancestor resolution.

Example
$S5 = \{ \sim p(z),$ $q(a,b),$ $r(d), \quad (\text{Top clause})$ $p(y) \mid \sim r(x) \mid \sim q(y,x),$ $p(y) \mid \sim r(x) \mid \sim s(y,x),$ $s(c,d) \}$

Linear input deduction is complete for input sets of Horn clauses [Henschen & Wos, 1974], but is not complete for input sets that contain non-Horn clauses. Ringwood [1988] provides an interesting synopsis and references for the history of linear input deduction systems. Practical notes:

- If a refutation exists, one exists with one of the negative clauses as top clause. Thus it's safe to try only each of the negative clauses as top clauses.
- The selection of which centre clause literal to resolve upon is an AND decision, and can be committed to once made. A common selection is the right most literal.
- During a linear deduction, it is necessary to allow for alternative side clauses at each step. One way to implement this is via *backtracking*.
- If the top clause is negative, then all center clauses are necessarily negative, and only positive literal in each mixed or positive clause can be resolved against at each step.

Example																
$S5 = \{ \sim p(z),$ $q(a,b),$ $r(d),$ $p(y) \mid \sim r(x) \mid \sim q(y,x),$ $p(y) \mid \sim r(x) \mid \sim s(y,x),$ $s(c,d) \}$																
<table> <thead> <tr> <th>Center clauses</th> <th>Side clauses</th> </tr> </thead> <tbody> <tr> <td>$\sim p(z)$</td> <td>$+ p(y) \mid \sim r(x) \mid \sim q(y,x)$</td> </tr> <tr> <td>$= \sim r(x) \mid \sim q(z,x) + q(a,b)$</td> <td></td> </tr> <tr> <td>$= \sim r(b)$</td> <td>$+ \text{Backtrack}$</td> </tr> <tr> <td>$= \sim p(z)$</td> <td>$+ p(y) \mid \sim r(x) \mid \sim s(y,x)$</td> </tr> <tr> <td>$= \sim r(x) \mid \sim s(z,x) + s(c,d)$</td> <td></td> </tr> <tr> <td>$= \sim r(d)$</td> <td>$+ r(d)$</td> </tr> <tr> <td>$= \text{FALSE}$</td> <td></td> </tr> </tbody> </table>	Center clauses	Side clauses	$\sim p(z)$	$+ p(y) \mid \sim r(x) \mid \sim q(y,x)$	$= \sim r(x) \mid \sim q(z,x) + q(a,b)$		$= \sim r(b)$	$+ \text{Backtrack}$	$= \sim p(z)$	$+ p(y) \mid \sim r(x) \mid \sim s(y,x)$	$= \sim r(x) \mid \sim s(z,x) + s(c,d)$		$= \sim r(d)$	$+ r(d)$	$= \text{FALSE}$	
Center clauses	Side clauses															
$\sim p(z)$	$+ p(y) \mid \sim r(x) \mid \sim q(y,x)$															
$= \sim r(x) \mid \sim q(z,x) + q(a,b)$																
$= \sim r(b)$	$+ \text{Backtrack}$															
$= \sim p(z)$	$+ p(y) \mid \sim r(x) \mid \sim s(y,x)$															
$= \sim r(x) \mid \sim s(z,x) + s(c,d)$																
$= \sim r(d)$	$+ r(d)$															
$= \text{FALSE}$																

Example
$S7 = \{ \sim t(x) \mid \sim r(x),$ $r(a) \mid \sim p(s) \mid \sim q(s),$ $p(a),$ $p(b),$ $q(b),$ $t(x) \mid \sim p(a) \}$

Prolog is linear input resolution using a negative top clause and a depth first search choosing the left most literal of the centre clause at each step.

Example
The set S7 would be written as a Prolog program: $r(a):-$ $ p(s),$ $ q(s).$ $p(a).$

```
p(b).  
q(b).  
t(X):-  
    p(a).
```

and a query at the Prolog prompt:

```
?- t(X),r(X).
```

Example

```
SN = { ~p(b,X) | ~q(X),  
        r(a),  
        ~q(X) | r(X),  
        r(b) | ~q(c),  
        p(b,a) | ~r(a),  
        p(Y,c) | ~r(Y),  
        q(c) }
```

Example

The set SN would be written as a Prolog program:

```
r(a).  
r(X):-  
    q(X).  
r(b):-  
    q(c).  
p(b,a):-  
    r(a).  
p(Y,c):-  
    r(Y).
```

```
q(c).
```

and a query at the Prolog prompt:

```
?- p(b,X),q(X).
```

Exam Style Questions

1. Use linear-input resolution to derive the empty clause from the set

```
S = { ~r(Y) | ~p(Y),  
        p(b),  
        r(a),  
        p(S) | ~p(b) | ~r(S),  
        r(c) }
```

Syntactic Refinements

The Set of Support Strategy

The Set of Support (SoS) strategy places some of the input clauses into the Set of Support, and all resolvents are put in the SoS. Resolution of two clauses not in the SoS is prohibited, i.e., at least one parent clause must be in the SoS. The ANL algorithm implements the SoS strategy directly, with the `ToBeUsed` set holding the SoS and the remaining input clauses placed into the `CanBeUsed` set. This prevents the generation of resolvents from two clauses not in the SoS.

The choice of SoS (while retaining refutation completeness) is determined by semantic considerations, which will be explained later. Two possible choices of SoS are all the negative clauses or all the positive clauses. The smaller the SoS the more effective the restriction.

Example
$S4 = \{ \sim p(a),$ $p(X) \mid l(X) \mid q,$ $\sim q \mid l(a),$ $\sim l(a) \mid r,$ $\sim r \mid \sim l(Z) \}$ $SoS = \{ p(X) \mid l(X) \mid q \}$

Example
$S1 = \{ p(b) \mid r(Y) \mid p(Y),$ $\sim p(S) \mid p(b),$ $\sim p(b),$ $\sim r(a),$ $\sim r(c) \}$ $SoS = \{ p(b) \mid r(Y) \mid p(Y) \}$

Unit (Binary) Resolution

In each binary resolution step, one parent must be a unit clause. For the ANL loop, unit input clauses are placed in the `ToBeUsed` set, non-units in `CanBeUsed`.

Example
$S2 = \{ \sim p(X) \mid q(X) \mid \sim r(X),$ $p(a) \mid p(b),$ $\sim q(Y),$ $r(a),$ $r(b) \}$

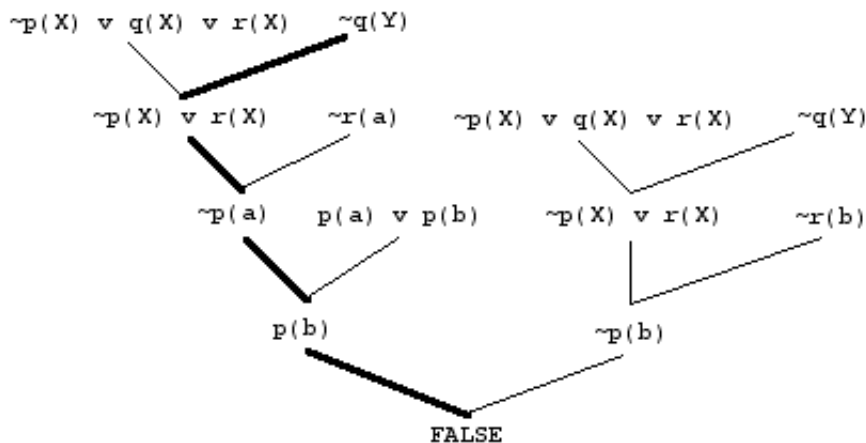
Example
$S = \{ bf(a),$ $\sim bg(a),$ $\sim bf(X) \mid bh(X),$ $\sim bj(X) \mid \sim bi(X) \mid bf(X),$ $\sim bh(X) \mid bg(X) \mid \sim bi(Y) \mid \sim bh(Y),$ $bj(b),$ $bi(b) \}$

This refinement is refutation complete for Horn clauses, but refutation incomplete for non-Horn clauses, although it does sometimes work for non-Horn clauses. A special case of refutation completeness for non-Horn sets is when the non-Horn set is renamable (by consistently swapping signs for some predicates) to a Horn set.

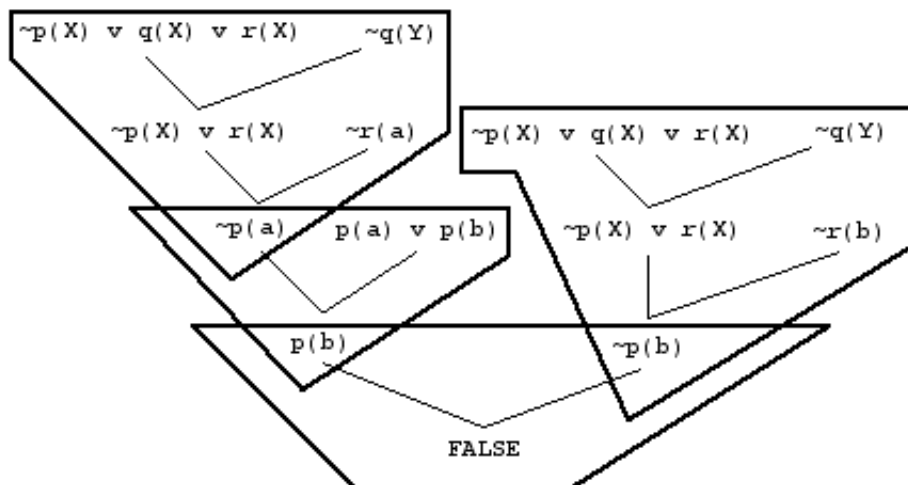
Example, S2 renames to
$S' = \{ p(X) \mid \sim q(X) \mid \sim r(X),$ $\sim p(a) \mid \sim p(b),$ $q(Y),$ $r(a),$ $r(b) \}$

UR resolution

Consider the proof tree from the unit resolution refutation of S2, and any path from a leaf unit clause towards the root of the tree :



On such a path, it is necessary that at some stage a unit clause or the empty clause must be reached. In the path indicated in the above tree, the first unit clause reached is $\sim p(a)$. Starting again towards the root, again a unit or empty clause must be reached, in this case immediately at $p(b)$. So, a unit resolution proof can be broken down into chunks that end with the production of a unit resolvent or the empty clause. In the above proof tree the chunks are :



Unit resulting (UR) resolution builds refutations by creating such chunks. The unit clauses used are called electrons, and the single non-unit clause the nucleus. The intermediate clauses of each chunk are not kept, but are instead regenerated, e.g., $\sim p(X) \mid r(X)$ has to be generated twice. UR resolution can be implemented directly in the ANL algorithm by initially placing all unit input clauses into `ToBeUsed` and the non-units into `CanBeUsed`. The unit resolvents are placed in `ToBeUsed`. The first electron of each chunk is the `ChosenClause`. The nucleus and other electrons are taken from `CanBeUsed`. There is some loss of effectiveness of subsumption, due to intermediate clauses not being used in subsumption (smart solution - if an intermediate clause subsumes another, replace the other clause with the intermediate clause).

Example
$S_2 = \{ \sim p(X) \mid q(X) \mid r(X),$ $p(a) \mid p(b),$ $\sim q(Y),$ $\sim r(a),$ $\sim r(b) \}$

Example
$S = \{ bf(a),$ $\sim bg(a),$ $\sim bf(X) \mid bh(X),$ $\sim bj(X) \mid \sim bi(X) \mid bf(X),$ $\sim bh(X) \mid bg(X) \mid \sim bi(Y) \mid \sim bh(Y),$ $bj(b),$ $bi(b) \}$

Ordering

An ordering is imposed on the Herbrand base elements, and full resolutions between only the literals with the largest atoms in each clause are permitted. This restriction is imposed on the completed refutation.

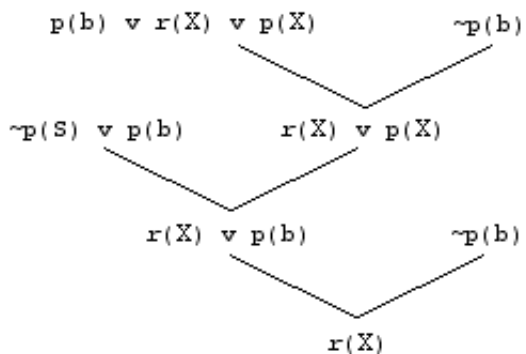
Example
$S_2 = \{ \neg p(X) \mid q(X) \mid r(X),$ $p(a) \mid p(b),$ $\neg q(Y),$ $\neg r(a),$ $\neg r(b) \}$
$HB = \{ p(a), p(b), q(a), q(b), r(a), r(b) \}$
$O = p(a) > p(b) > q(a) > q(b) > r(a) > r(b)$

Notice all the possible resolutions that are prevented by the ordering strategy.

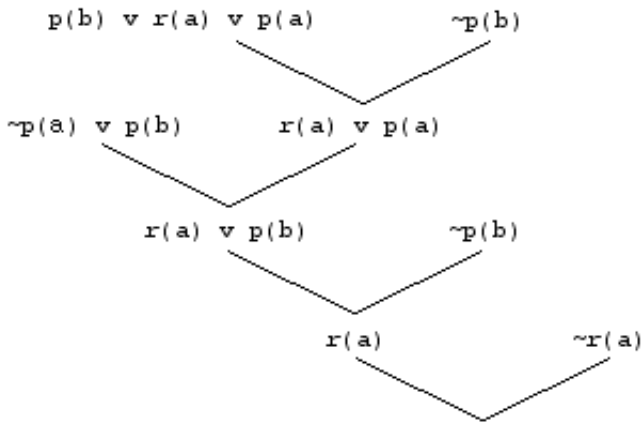
In the above example, the resolved upon literals all were or became ground in unification. Thus it was always possible to check that the largest Herbrand based literal was used in each clause. However, this is not always the case. If variables are unified then it is not necessarily known that the literal is the largest Herbrand based in the clause.

Example
$S_1 = \{ p(b) \mid r(Y) \mid p(Y),$ $\neg p(S) \mid p(b),$ $\neg p(b),$ $\neg r(a),$ $\neg r(c) \}$
$HB = \{ p(a), p(b), p(c), r(a), r(b), r(c) \}$
$O = p(a) > p(b) > p(c) > r(a) > r(b) > r(c)$

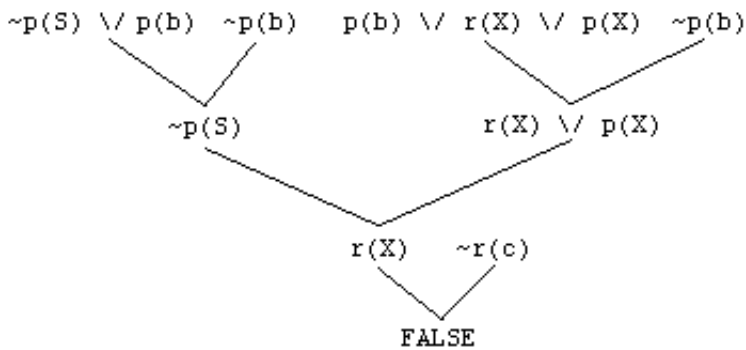
Here is the proof tree for the first part of a derivation.



If $\neg r(a)$ is used to complete the refutation, the proof does not conform to the ordering restriction at the first resolution. Here's the proof tree with variables instantiated up the tree. Note that $p(a)$ is the largest atom in $p(b) \mid r(a) \mid p(a)$



However, until the time of that last inference, it is not known at the time what value the variable x will instantiated to. Similarly, if $\sim r(c)$ is used in the last step instead of $\sim r(a)$ then it will still not conform. A proof tree that conforms is:



The solution is to either do retrospective checks, which is very expensive, or to have a less restrictive ordering, e.g., Knuth-Bendix ordering, or ordering the only predicate symbols. In the latter case, any of the literals that could contain the largest atom in the clause can be used.

Example
$S2 = \{ \sim p(X) \mid q(X) \mid r(X),$ $p(a) \mid p(b),$ $\sim q(Y),$ $\sim r(a),$ $\sim r(b) \}$
$O = p/1 > q/1 > r/1$

Example
$S = \{ bf(a),$ $\sim bg(a),$ $\sim bf(X) \mid bh(X),$ $\sim bj(X) \mid \sim bi(X) \mid bf(X),$ $\sim bh(X) \mid bg(X) \mid \sim bi(Y) \mid \sim bh(Y),$ $bj(b),$ $bi(b) \}$
$O = bf/1 > bg/1 > bh/1 > bi/1 > bj/1$

Ordering is a refutation complete strategy

Proof
To be filled in. Based on working up the failure tree.

Knuth-Bendix Ordering

KBO provides a way to order terms and atoms in a way that is consistent with an ordering on the Herbrand Universe or Herbrand Base. First some preliminaries ...

A reduction ordering is ...

- Stable under substitution, i.e., if $T_1 > T_2$ then $T_1\theta > T_2\theta$
- Stable under context, i.e., if $T_1 > T_2$ and $T_3[P]$ then $T_3[T_1] > T_3[T_2]$ for an arbitrary subterm P
- Terminating, i.e., there is no infinite sequence of terms $T_1 > T_2 > T_3 > \dots$. Note that it is possible to have a sequence $T_1 < T_2 < T_3 < \dots$.

A simplification ordering is a reduction ordering with the property ...

- For a term $s[P]$, $s > P$, i.e., a term is larger than all of its subterms.

An ordering R is *ground complete* if for all pairs of ground terms (elements of the Herbrand Universe) HUT_1 and HUT_2 , one of the following holds

...

- $HUT_1 <_R HUT_2$
- HUT_1 is HUT_2
- $HUT_1 >_R HUT_2$

An (simplification) ordering O is *ground completable* if there exists a ground complete reduction ordering R such that O is a subset of R , i.e., the pairs of terms ordered by O are ordered the same by R , and R may order some more pairs of terms that are not ordered by O .

Example

```
F = { joe/1, kevin/1, turkey/0, chicken/0 }
if joe(X) > kevin(X) then joe(turkey) > kevin(turkey)
if turkey > chicken then joe(turkey) > joe(chicken)
there is no infinite sequence joe(turkey) > joe(chicken) > kevin(turkey) > kevin(chicken) > turkey > chicken > ????
```

Knuth-Bendix Ordering is an example of a simplification ordering. It is not ground complete, but is ground completable. KBO uses a precedence ordering ρ and a weighting function ω .

- ρ totally orders the predicate and function symbols, and does not order variables wrt anything.
- ω maps each predicate and function symbol to a non-zero positive integer, and maps all variables to the same positive number less than the mapping of any function symbol. ω then maps a term T to the sum of the mappings of the symbols in T .

ρ and ω do not need to agree. KBO orders $T_1 > T_2$ if

- $\omega(T_1) > \omega(T_2)$ and every variable in T_2 occurs at least the same number of times in T_1 ,
or
- $\omega(T_1) = \omega(T_2)$ and $\rho(\text{symbol_of}(T_1)) > \rho(\text{symbol_of}(T_2))$
or
- $\omega(T_1) = \omega(T_2)$ and $\text{symbol_of}(T_1) \equiv \text{symbol_of}(T_2)$ and $\text{args}(1 \dots i-1, T_1) \equiv \text{args}(1 \dots i-1, T_2)$ and $\text{args}(i, T_1) > \text{args}(i, T_2)$

Example

```
S = { even(sum(two_squared,b)),
      two_squared = four,
      ~zero(X) | difference(four,X) = sum(four,X),
      zero(b),
      ~even(difference(two_squared,b)) }

\rho := > even > zero > = > sum > difference > four > two_squared > b

\rho(even)      \to 5
\rho(zero)      \to 5
\rho(=)         \to 3
\rho(sum)       \to 4
\rho(difference) \to 3
\rho(four)      \to 4
\rho(two_squared) \to 4
\rho(b)         \to 3
\rho(X)         \to 1

\rho(zero(X))   \to 6
\rho(difference(four,X)) \to 8
\rho(sum(four,X)) \to 9
\rho(difference(four,difference(four,four))) \to 18
\rho(difference(four,X) = sum(four,X)) \to 20
```

sum(four,X)	>	difference(four,X)
sum(four,X) = difference(four,X)	>	zero(X)
even(X)	>	zero(X)
even(sum(X,four))	>	even(sum(X,two_squared))
even(sum(X,four))	!>	even(sum(four,two_squared))
even(sum(X,four))	!>	even(sum(Y,two_squared))
difference(four,difference(four,four))	!>	sum(four,X)

KBO can be used to determine the largest literals in a clause, according to their atoms.

Example

```
S1 = { p(b) | r(Y) | p(Y),
      ~p(S) | p(b),
      ~p(b),
      ~r(a),
      ~r(c) }
```

- p is alphabetic
- ω maps all predicates and function symbols to 2, and all variables to 1.

```
~p(b)           is the ChosenClause
~r(a)           is the ChosenClause
~r(c)           is the ChosenClause
~p(S) | p(b)    is the ChosenClause
~p(S) | p(b)    + ~p(b)
                = ~p(S)
Backward subsumes ~p(S) | p(b) (the ChosenClause)
                ~p(b)
~p(S)           is the ChosenClause
p(b) | r(Y) | p(Y) is the ChosenClause with p(b) largest
p(b) | r(Y) | p(Y) + ~p(S)
                = r(Y) | p(Y)
Backward subsumes p(b) | r(Y) | p(Y) (the ChosenClause)
r(Y) | p(Y)    is the ChosenClause
r(Y) | p(Y)    + ~p(S)
                = r(Y)
Backward subsumes r(Y) | p(Y) (the ChosenClause)
r(Y)           is the ChosenClause
r(Y)           + ~r(a)
                = FALSE
```

Other Orderings

LPO (lexicographic path ordering) compares terms in a complicated lexicographic way.

Lock Resolution

[Boyer 1971] (a very restrictive ordering)

Every occurrence of a literal in the input clauses is uniquely numbered, and binary resolution is only permitted on literals of lowest index in their clause. Resolvent literals inherit their indices from parent clauses. A lock factor is a factor of a clause where the lowest index of the factored literals is retained (hmmm, must one of the factored literals have the lowest index in the clause?). Lock resolution is not compatible with many things (including set of support), but is used in conjunction with model resolution in HLR.

Example

```
S1 = { p(b)1 | r(Y)2 | p(Y)3,
      ~p(S)4 | p(b)5,
      ~p(b)6,
```

$\sim r(a)_7,$
 $\sim r(c)_8 \}$

Notice the marked reduction of the search space.

Example

$S_2 = \{ \sim p(x)_1 \mid q(x)_2 \mid r(x)_3,$
 $p(a)_4 \mid p(b)_5,$
 $\sim q(y)_6,$
 $\sim r(a)_7,$
 $\sim r(b)_8 \}$

Example

$S = \{ bf(a),$
 $\sim bg(a),$
 $\sim bf(x) \mid bh(x),$
 $\sim bj(x) \mid \sim bi(x) \mid bf(x),$
 $\sim bh(x) \mid bg(x) \mid \sim bi(y) \mid \sim bh(y),$
 $bj(b),$
 $bi(b) \}$

Lock resolution is a refutation complete strategy

Proof

To be filled in.

Semantic Refinements

Model Resolution

Model resolution imposes the restriction that one parent of each full resolution must be **FALSE** in a given interpretation. For the ANL loop, **TRUE** input clauses are placed in **CanBeUsed**, and **FALSE** input clauses are placed in **ToBeUsed**. The interpretation should be chosen so as to minimize the number of **FALSE** clauses.

Example	
$I = \{$	
$p(X) \rightarrow$	FALSE ,
$q(X, Y) \rightarrow$	FALSE ,
$r(X, Y) \rightarrow$	FALSE ,
$s \rightarrow$	FALSE ,
$t(X) \rightarrow$	FALSE }
$S5 = \{$	
$p(X) \mid p(Y) \mid \neg q(X, Y) \mid \neg s,$	(TRUE)
$q(X, Y) \mid r(X, Y),$	(FALSE)
$\neg r(a, b) \mid t(Y) \mid t(X),$	(TRUE)
$s,$	(FALSE)
$\neg p(X),$	(TRUE)
$\neg t(X) \mid \neg t(Y) \}$	(TRUE)

Example (not so illustrative)	
$I = \{$	
$l(X) \rightarrow$	FALSE ,
$p(X) \rightarrow$	FALSE ,
$q \rightarrow$	TRUE ,
$r \rightarrow$	TRUE }
$S4 = \{$	
$\neg p(a),$	(TRUE)
$p(X) \mid l(X) \mid q,$	(TRUE)
$\neg q \mid l(a),$	(FALSE)
$\neg l(a) \mid r,$	(TRUE)
$\neg r \mid \neg l(Z) \}$	(TRUE)

Model resolution is a refutation complete strategy

Proof
To be filled in. Based on forcing the two clauses to 'be from' different branches of the failure tree.

- P_1 resolution is model resolution using the negative interpretation.
- N_1 resolution is model resolution using the positive interpretation.
- P_p resolution is model resolution using a predicate partition.
- A **FALSE preference** strategy selected a **FALSE** ChosenClause in preference to a **TRUE** one.

The Set of Support Strategy

Recall, the Set of Support (SoS) strategy places some of the input clauses into the Set of Support, and all resolvents are put in the SoS. Resolution of two clauses not in the SoS is prohibited, i.e., at least one parent clause must be in the SoS. The SoS strategy is actually a weakened form of model resolution. For completeness, the initial SoS must be chosen such that the non-SoS clauses have a model, i.e., are satisfiable. This specification ensures that one parent is not necessarily **TRUE** in the model of the non-SoS clauses. Model resolution can be viewed as repeated application of the SoS strategy after each inference step.

As indicated earlier, two possible choices of SoS are all the negative clauses (the non-SoS clauses have the positive interpretation as a model) or all the positive clauses (the non-SoS clauses have the negative interpretation as a model).

Model Resolution with Ordering

Model resolution is compatible with ordering on the **FALSE** parent. Full resolution is still used, and the atoms of the resolved upon literals must be the largest in their clauses, in the completed refutation. As before, retrospective checking is prohibitively expensive, so ordering on the predicate symbols is often used.

Example	
$I = \{$	
$p(X) \rightarrow$	FALSE ,
$q(X, Y) \rightarrow$	FALSE ,
$r(X, Y) \rightarrow$	FALSE ,

s	\rightarrow	FALSE,	
$t(X)$	\rightarrow	FALSE}	
$S5 = \{$	$p(X) \mid p(Y) \mid \sim q(X,Y) \mid \sim s,$	(TRUE)	
	$q(X,Y) \mid r(X,Y),$	(FALSE)	
	$\sim r(a,b) \mid t(Y) \mid t(X),$	(TRUE)	
	$s,$	(FALSE)	
	$\sim p(X),$	(TRUE)	
	$\sim t(X) \mid \sim t(Y) \}$	(TRUE)	
$O = p(X) > q(X,Y) > r(X,Y) > s > t(X)$			

Example	
$I = \{$	$l(X) \rightarrow$ FALSE,
	$p(X) \rightarrow$ FALSE,
	$q \rightarrow$ TRUE,
	$r \rightarrow$ TRUE }
$S4 = \{$	$\sim p(a),$ (TRUE)
	$p(X) \mid l(X) \mid q,$ (TRUE)
	$\sim q \mid l(a),$ (FALSE)
	$\sim l(a) \mid r,$ (TRUE)
	$\sim r \mid \sim l(z) \}$ (TRUE)
$O = l/1 > p/1 > q/0 > r/0$	

Notice that a slightly longer proof is obtained.

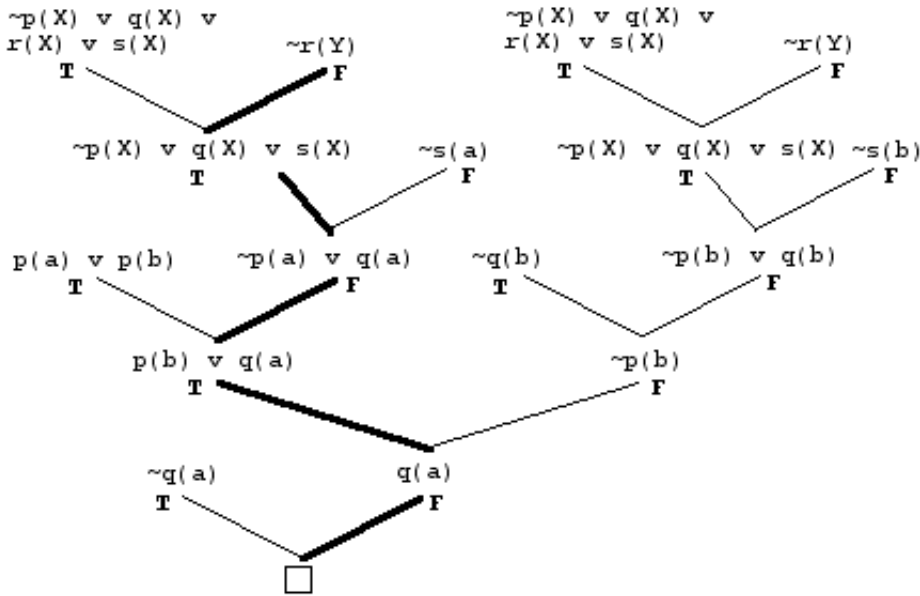
Model resolution, with Ordering on the FALSE parent, is refutation complete.

Proof
To be filled in. Based on forcing the two clauses to 'be from' different branches of the failure tree. The one from the branch which is FALSE in I has to use its deepest literal. Then the clause from the true branch has a sibling literal, and the failure tree gets smaller after the resolution.

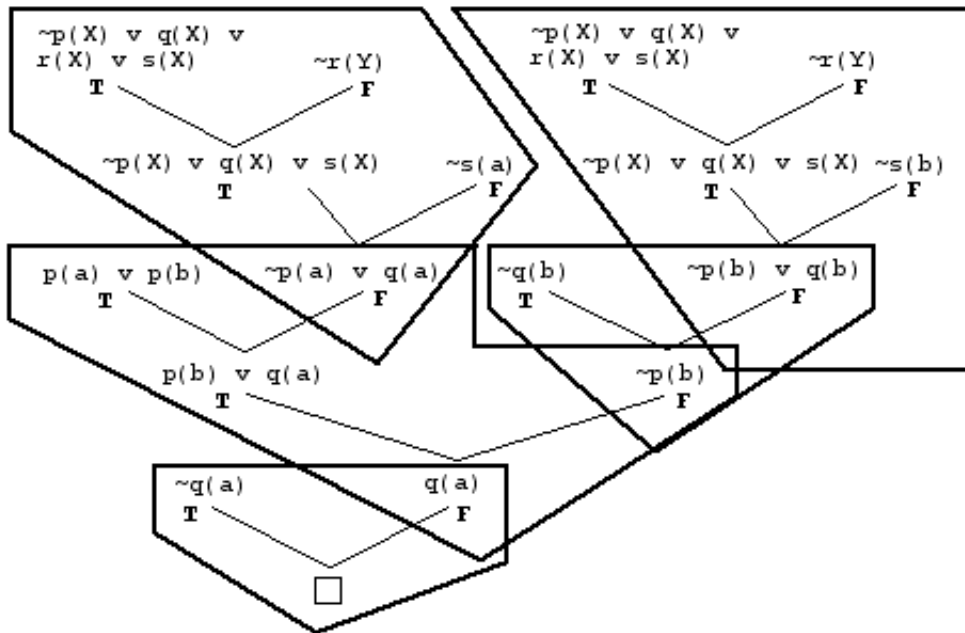
Semantic Resolution

Consider the proof tree from the model resolution refutation of $S3$, and any path from a FALSE leaf clause towards the root of the tree:

Example	
$I = \{$	$p(X) \rightarrow$ TRUE,
	$r(X) \rightarrow$ TRUE,
	$s(X) \rightarrow$ TRUE,
	$q(X) \rightarrow$ FALSE}
$S3 = \{$	$\sim p(X) \mid q(X) \mid r(X) \mid s(X),$ (TRUE)
	$\sim r(Y),$ (FALSE)
	$\sim s(a),$ (FALSE)
	$\sim s(b),$ (FALSE)
	$\sim q(a),$ (TRUE)
	$\sim q(b),$ (TRUE)
	$p(a) \mid p(b) \}$ (TRUE)



On such a path, it is necessary that at some stage a FALSE clause must be reached (at latest, the root of the tree). In the path indicated in the above tree, the first FALSE clause reached is $\sim p(a) \mid q(a)$. Starting again towards the root, again a FALSE clause must be reached, in this case at $q(a)$. So, a model resolution proof can be broken down into chunks that end with the production of a FALSE resolvent. In the above proof tree the chunks are:



Unordered semantic resolution builds refutations by creating such chunks. The FALSE clauses used are called electrons, and the single TRUE clause the nucleus. The intermediate clauses are not kept, but are instead regenerated, e.g., $\sim p(X) \mid q(X) \mid s(X)$ has to be generated twice. If an intermediate clause is forward subsumed, then that option in the chunk is abandoned. If an intermediate clause backward subsumes an existing clause, then the intermediate clause can replace the existing clause. These intermediate subsumption checks are often omitted in implementations (but the play off is not well known). For the ANL loop, TRUE input clauses are placed in *CanBeUsed*, and FALSE input clauses are placed in *ToBeUsed*. The first electron of each chunk is the *ChosenClause*. The nucleus and other electrons are taken from *CanBeUsed*.

Example	
$I = \{$	$p(X) \rightarrow \text{FALSE},$
	$q(X, Y) \rightarrow \text{FALSE},$
	$r(X, Y) \rightarrow \text{FALSE},$
	$s \rightarrow \text{FALSE},$
	$t(X) \rightarrow \text{FALSE}\}$
$S5 = \{$	$p(X) \mid p(Y) \mid \sim q(X, Y) \mid \sim s, \quad (\text{TRUE})$
	$q(X, Y) \mid r(X, Y), \quad (\text{FALSE})$

$\sim r(a,b) \mid t(Y) \mid t(X),$	(TRUE)
$s,$	(FALSE)
$\sim p(X),$	(TRUE)
$\sim t(X) \mid \sim t(Y) \}$	(TRUE)

Example	
$I = \{ l(X) \rightarrow \text{FALSE},$ $p(X) \rightarrow \text{FALSE},$ $q \rightarrow \text{TRUE},$ $r \rightarrow \text{TRUE} \}$	
$S4 = \{ \sim p(a), \quad (\text{TRUE})$ $p(X) \mid l(X) \mid q, \quad (\text{TRUE})$ $\sim q \mid l(a), \quad (\text{FALSE})$ $\sim l(a) \mid r, \quad (\text{TRUE})$ $\sim r \mid \sim l(z) \} \quad (\text{TRUE})$	

Ordered Semantic Resolution

Because model resolution is compatible with ordering on the FALSE parent, semantic resolution can also be restricted in the same way. Given an interpretation I of the clauses and an ordering O of the Herbrand base of the clauses, in a *semantic clash* (or PI-clash) of a set of clauses $\{E_1, \dots, E_q, N\}$:

- The E_i are the *electrons* (satellites)
- N is the *nucleus*
- The E_i are FALSE in I
- Let $R_1 = N$. For each $i=1..q$, there is a (full) resolvable R_{i+1} of R_i and E_i
- The resolved upon literals in E_i are the largest in E_i .
- R_{q+1} is FALSE in I.

R_{q+1} is called a PI-resolvable. A semantic resolution refutation is built from semantic clashes. A full resolvable is necessary (consider $S = \{p(X) \mid p(Y), \sim p(X) \vee \sim p(Y)\}$, $I = \{p(X) \rightarrow \text{FALSE}\}$).

Example	
$I = \{ p(X) \rightarrow \text{FALSE},$ $q(X,Y) \rightarrow \text{FALSE},$ $r(X,Y) \rightarrow \text{FALSE},$ $s \rightarrow \text{FALSE},$ $t(X) \rightarrow \text{FALSE} \}$	
$S5 = \{ p(X) \mid p(Y) \mid \sim q(X,Y) \mid \sim s, \quad (\text{TRUE})$ $q(X,Y) \mid r(X,Y), \quad (\text{FALSE})$ $\sim r(a,b) \mid t(Y) \mid t(X), \quad (\text{TRUE})$ $s, \quad (\text{FALSE})$ $\sim p(X), \quad (\text{TRUE})$ $\sim t(X) \mid \sim t(Y) \} \quad (\text{TRUE})$	
$O = p(X) > q(X,Y) > r(X,Y) > s > t(X)$	

Example	
$I = \{ l(X) \rightarrow \text{FALSE},$ $p(X) \rightarrow \text{FALSE},$ $q \rightarrow \text{TRUE},$ $r \rightarrow \text{TRUE} \}$	
$S4 = \{ \sim p(a), \quad (\text{TRUE})$ $p(X) \mid l(X) \mid q, \quad (\text{TRUE})$ $\sim q \mid l(a), \quad (\text{FALSE})$ $\sim l(a) \mid r, \quad (\text{TRUE})$ $\sim r \mid \sim l(z) \} \quad (\text{TRUE})$	
$O = l/1 > p/1 > q/0 > r/0$	

Ordered semantic resolution is refutation complete.

Proof
To be filled in. [Slagle 1967]

If binary resolution is used in semantic resolution, it is sufficient to factor only the electrons. This is possible because electrons can be reused to remove factorable literals in nuclei, and the duplicate literals in the resolvents factor away themselves. The initial electrons must be factored, and the resolvents are factored as they are created. This is the normal way of implementing semantic resolution and its specialisations. For the ANL loop, place all TRUE clauses in CanBeUsed, and FALSE clauses and FALSE factors in ToBeUsed.

Example	
$I = \{$	$p(X) \rightarrow \text{FALSE},$
	$q(X,Y) \rightarrow \text{FALSE},$
	$r(X,Y) \rightarrow \text{FALSE},$
	$s \rightarrow \text{FALSE},$
	$t(X) \rightarrow \text{FALSE}\}$
$S5 = \{$	$p(X) \mid p(Y) \mid \sim q(X,Y) \mid \sim s, \quad (\text{TRUE})$
	$q(X,Y) \mid r(X,Y), \quad (\text{FALSE})$
	$\sim r(a,b) \mid t(Y) \mid t(X), \quad (\text{TRUE})$
	$s, \quad (\text{FALSE})$
	$\sim p(X), \quad (\text{TRUE})$
	$\sim t(X) \mid \sim t(Y) \}$
	(TRUE)
$O =$	$p(X) > q(X,Y) > r(X,Y) > s > t(X)$

Note that for positive, negative, and predicate-partition interpretations, factoring preserves FALSENESS, so factored electrons remain electrons. For general interpretations this is not the case.

Example	
$I = \{$	$p(a) \rightarrow \text{FALSE},$
	$p(b) \rightarrow \text{TRUE},$
	$q(X) \rightarrow \text{FALSE} \}$
$S = \{$	$p(X) \mid q(b) \mid q(X) \}$
	(FALSE)
$p(X) \mid q(b) \mid q(X) + \text{Factoring}$	
	$= p(b) \mid q(b) \quad (\text{TRUE})$

Hyper-resolution

Positive hyper-resolution is semantic resolution using the negative interpretation. Negative hyper-resolution is semantic resolution using the positive interpretation. AM-clash resolution is semantic resolution using a predicate partition.

Equality

Many domains require the use of the notion of equality. In 1st order logic equality statements use the prefix `equal/2` predicate, or infix `=/2` and `!=/2` predicates.

Example
<ul style="list-style-type: none">• Axioms <pre>even(sum(two_squared,b)) two_squared = four ∀X (zero(X) => difference(four,X) = sum(four,X)) zero(b)</pre> <ul style="list-style-type: none">• Conjecture <pre>even(difference(two_squared,b))</pre>

Although the conjecture may seem like a logical consequence to humans, that's because humans "know" what *equal* means (even without knowing what the "maths" means). One can use ones intuitive understanding of equality in a resolution refutation.

Example
<pre>S = { even(sum(two_squared,b)), two_squared = four, ~zero(X) difference(four,X) = sum(four,X), zero(b), ~even(difference(two_squared,b)) }</pre> <pre>even(sum(two_squared,b)) + two_squared = four » even(sum(four,b)) ~zero(X) difference(four,X) = sum(four,X) + zero(b) » difference(four,b) = sum(four,b) even(sum(four,b)) + difference(four,b) = sum(four,b) » even(difference(four,b)) even(difference(four,b)) + two_squared = four » even(difference(two_squared,b)) ~even(difference(two_squared,b)) + even(difference(two_squared,b)) » FALSE</pre>

However, there are models of the axioms that are not a models of the conjecture:

Example
<pre>D = { cat, dog } F = { b → cat, two_squared → cat, four → cat, difference(cat,cat) → dog, difference(cat,dog) → cat, difference(dog,cat) → cat, difference(dog,dog) → cat, sum(cat,cat) → cat, sum(cat,dog) → cat, sum(dog,cat) → cat, sum(dog,dog) → cat } R = { even(cat) → TRUE, even(dog) → FALSE, cat = cat → TRUE, cat = dog → FALSE, dog = cat → TRUE, dog = dog → FALSE, zero(cat) → TRUE, zero(dog) → FALSE }</pre>

Such models are possible because the axioms are missing definitions for equality. These definitions are the axioms of equality, and must be included to force equality to have its usual meaning. They are:

- Reflexivity – everything equals itself

- Symmetry – If x equals y then y equals x
- Transitivity – If x equals y and y equals z, then x equals z

$\forall X (X = X)$
 $\forall X \forall Y (X = Y \Rightarrow Y = X)$
 $\forall X \forall Y \forall Z ((X = Y \ \& \ Y = Z) \Rightarrow X = Z)$

- Function substitution – If x equals y then f(x) equals f(y). For every argument position of every functor.
- Predicate substitution – If x equals y and p(x) is TRUE, then p(y) is TRUE For every argument position of every predicate.

$\forall X \forall Y \forall Z (X = Y \Rightarrow \text{sum}(X,Z) = \text{sum}(Y,Z))$
 $\forall X \forall Y \forall Z (X = Y \Rightarrow \text{sum}(Z,X) = \text{sum}(Z,Y))$
 $\forall X \forall Y \forall Z (X = Y \Rightarrow \text{difference}(X,Z) = \text{difference}(Y,Z))$
 $\forall X \forall Y \forall Z (X = Y \Rightarrow \text{difference}(Z,X) = \text{difference}(Z,Y))$
 $\forall X \forall Y ((X = Y \ \& \ \text{even}(X)) \Rightarrow \text{even}(Y))$
 $\forall X \forall Y ((X = Y \ \& \ \text{zero}(X)) \Rightarrow \text{zero}(Y))$

With these axioms, it is possible to expand the example to a complete resolution refutation.

Example

```

S = { even(sum(two_squared,b)),
      two_squared = four,
      ~zero(X) | difference(four,X) = sum(four,X),
      zero(b),
      ~even(difference(two_squared,b)),
      X = X,
      X != Y | Y = X,
      X != Y | Y != Z | X = Z,
      X != Y | sum(X,Z) = sum(Y,Z),
      X != Y | sum(Z,X) = sum(Z,Y),
      X != Y | difference(X,Z) = difference(Y,Z),
      X != Y | difference(Z,X) = difference(Z,Y),
      X != Y | ~even(X) | even(Y),
      X != Y | ~zero(X) | zero(Y) }

even(sum(two_squared,b))                + X != Y | ~even(X) | even(Y)
                                         » sum(two_squared,b) != Y | even(Y)
sum(two_squared,b) != Y | even(Y)       + X != Y | sum(X,Z) = sum(Y,Z)
                                         » two_squared != Y | even(sum(Y,b))
two_squared != Y | even(sum(Y,b))        + two_squared = four
                                         » even(sum(four,b))
~zero(X) | difference(four,X) = sum(four,X) + zero(b)
                                         » difference(four,b) != sum(four,b)
even(sum(four,b))                        + X != Y | ~even(X) | even(Y)
                                         » sum(four,b) != Y | even(Y)
sum(four,b) != Y | even(Y)              + X != Y | Y = X
                                         » Y != sum(four,b) | even(Y)
Y != sum(four,b) | even(Y)              + difference(four,b) = sum(four,b)
                                         » even(difference(four,b))
even(difference(four,b))                 + X != Y | ~even(X) | even(Y)
                                         » difference(four,b) != Y | even(Y)
difference(four,b) != Y | even(Y)        + X != Y | difference(X,Z) = difference(Y,Z)
                                         is four != Y | even(difference(Y,b))
four != Y | even(difference(Y,b))         + X != Y | Y = X
                                         » Y != four | even(difference(Y,b))
Y != four | even(difference(Y,b))         + two_squared = four
                                         » even(difference(two_squared,b))
~even(difference(two_squared,b))         + even(difference(two_squared,b))
                                         » FALSE

```

Example (you fill in the equality axioms)

```

S = { ~mother(C,M) | husband_of(M) = father_of(C),
      mother(geoff,maggie),
      bob = husband_of(maggie),
      father_of(geoff) != bob }

```

The general problem is to show that two atoms can be made unifiable, by virtue of a sequence of resolution steps with equality axioms and other equality

Paramodulation

Paramodulation is an inference rule generates all "equal" versions of clauses, modulo conditions on the equality information. Paramodulation does the job of all the equality axioms except reflexivity.

The paramodulation operation takes two parent clauses, the *from* clause and the *into* clause. The from clause must contain a positive equality literal. One of the equality literal's arguments must unify with a subterm in the into clause. After unification of the argument and the subterm, the subterm is replaced by the other argument of the equality literal. The literals of the from clause, except the equality literal, and the literals of the into clause are disjoined to form the paramodulant.

$$T1 = T2 \mid C \mid + D[T3]$$

$$T1\theta = T3\theta$$

$$\gg (C \mid \mid D[T2])\theta$$

or

$$T2\theta = T3\theta$$

$$\gg (C \mid \mid D[T1])\theta$$

Example

```
live_in_uk => ruler = king P+ idiot(king)
      » live_in_uk => idiot(ruler)
```

Paramodulation replaces equal subterms in clauses. One subterm is replaced at a time; multiple paramodulation steps can be used to do multiple replacements.

It is (almost) never necessary to paramodulate into variables, i.e., it is (almost) never necessary to replace a variable subterm by a non-variable equality literal argument, using paramodulation. As the general aim of paramodulation is to make atoms unifiable, replacing a variable with a term will not help. If the variable later becomes instantiated to a 'unsuitable' subterm, paramodulation can then replace that subterm. In the definition above, this constrains T3 to be a non-variable. Similarly, depending on how paramodulation is used, it is often unnecessary to paramodulate from variables, i.e., it is often unnecessary to a paramodulation where one of the equality literal's arguments is a variable, and that variable is unified with the subterm which is then replaced. In the definition above, this constrains T1 and T2 to be non-variables. I MUST CHECK OUT THE DETAILS OF THIS.

Example

```
S = { even(sum(two_squared,b)),
      two_squared = four,
      ~zero(X) | difference(four,X) = sum(four,X),
      zero(b),
      ~even(difference(two_squared,b)),
      X = X }

even(sum(two_squared,b))          P+ two_squared = four
      » even(sum(four,b))

~zero(X) | difference(four,X) = sum(four,X) + zero(b)
      » difference(four,b) = sum(four,b)
even(sum(four,b))                P+ difference(four,b) = sum(four,b)
      » even(difference(four,b))

even(difference(four,b))          P+ two_squared = four
      » even(difference(two_squared,b))
~even(difference(two_squared,b)) + even(difference(two_squared,b))
      » FALSE
```

Example

```
S = { ~mother(C,M) | husband_of(M) = father_of(C) }.
      mother(geoff,maggie),
      bob = husband_of(maggie),
      father_of(geoff) != bob }
```

One of the problems with paramodulation is that it is pre-emptive. It generates all equal variants of clauses without knowing which of the variants will be useful.

Example
$$S = \{ \begin{array}{l} X = X, \\ A \neq B \mid \sim p(A) \mid p(B), \\ X = c \mid X = d, \\ p(a), \\ p(b), \\ a \neq b, \\ \sim p(e) \end{array} \}$$

Superposition

Superposition is a paramodulation-like inference rule, with constraints based on orderings on the terms and equality literals. Equality factoring and equality resolution and combined with superposition to form a complete inference system for equality. Some problems are naturally expressed using only equality – these are pure equality problems. Superposition+EF+ER is a complete inference system for pure equality problems.

Superposition+EF+ER is used in conjunction with other inference rules. For example, superposition+EF+ER combined with resolution forms a refutation complete system. The resolution can be ordered using the superposition ordering for literal selection, so that in each parent of every inference only the largest literals are used.

The orderings on terms and literals must have certain properties; these are discussed below. The orderings need not be total, i.e., some pairs or terms or literals may not be comparable. All uses of the ordering are expressed using "negation by failure", so that incomparable pairs can be considered.

Let $T_1 = T_2$ represent also $T_2 = T_1$. Then ...

$$\begin{aligned}
 T_1 = T_2 \mid C^l + L[TT] \mid D^l \\
 T_1\theta \equiv TT\theta \\
 (T_1 = T_2)\theta \text{ is not smaller than any literal in } (T_1 = T_2 \mid C^l)\theta \\
 T_1\theta \text{ is not smaller than } T_2\theta \text{ in the term ordering} \\
 L\theta \text{ is not smaller than any literal in } (L \mid D^l)\theta \\
 \gg (L[T_2] \mid C^l \mid D^l)\theta
 \end{aligned}$$

Example

$a = f(x,b) \mid p(y,x) + q(g(f(b,z)),c) \mid q(z,z)$	
T_1 is $f(x,b)$	L is $q(g(f(b,z)),c)$
T_2 is a	TT is $f(b,z)$
C^l is $p(y,x)$	D^l is $q(z,z)$
θ is $\{x/b, z/b\}$	
It must be the case that:	$a = f(b,b)$ is not smaller than $p(y,b)$ $f(b,b)$ is not smaller than a $q(g(f(b,b)),c)$ is not smaller than $q(b,b)$ $\gg q(g(a),c) \mid p(y,b) \mid q(b,b)$

If L is an (in)equality literal, then a tighter constraint may be imposed. Let $\cdot =$ mean $=$ or \neq , i.e., $T_1 \cdot = T_2$ means any of $T_1 = T_2, T_2 = T_1, T_1 \neq T_2, T_2 \neq T_1$. Then ...

$$\begin{aligned}
 T_1 = T_2 \mid C^l + T_3[TT] \cdot = T_4 \mid D^l \\
 T_1\theta \equiv TT\theta \\
 (T_1 = T_2)\theta \text{ is not smaller than any literal in } (T_1 = T_2 \mid C^l)\theta \\
 T_1\theta \text{ is not smaller than } T_2\theta \text{ in the term ordering} \\
 (T_3 \cdot = T_4)\theta \text{ is not smaller than any literal in } (T_3 \cdot = T_4 \mid D^l)\theta \\
 T_3\theta \text{ is not smaller than } T_4\theta \text{ in the term ordering} \\
 \gg (T_3[T_2] \cdot = T_4 \mid C^l \mid D^l)\theta
 \end{aligned}$$

Example

$a = f(x,b) \mid p(y,x) + g(f(b,z)) \neq c \mid q(z,z)$	
T_1 is $f(x,b)$	T_3 is $g(f(b,z))$
T_2 is a	T_4 is c
C^l is $p(y,x)$	TT is $f(b,z)$
θ is $\{x/b, z/b\}$	D^l is $q(z,z)$
It must be the case that:	$a = f(b,b)$ is not smaller than $p(y,b)$ $f(b,b)$ is not smaller than a $g(f(b,b)) \neq c$ is not smaller than $q(b,b)$ $g(f(b,b))$ is not smaller than c $\gg g(a) \neq c \mid p(y,b) \mid q(b,b)$

Equality Factoring and Equality Resolution

$$\begin{aligned}
 T_1 = T_2 \mid T_3 = T_4 \mid C^l + \text{Equality factoring} \\
 T_1\theta \equiv T_3\theta
 \end{aligned}$$

$(T1 = T2)\theta$ is not smaller than any literal in $(T1 = T2 \mid T3 = T4 \mid c^l)\theta$
 $\gg (T2 \neq T4 \mid T1 = T2 \mid c^l)\theta$

Example

$b = f(a) \mid f(X) = c \mid p(X,d) \mid \sim q(d)$ + Equality factoring

T1 is $f(a)$

T2 is b

T3 is $f(X)$

T4 is c

c^l is $p(X,d) \mid \sim q(d)$

θ is $\{X/a\}$

It must be the case that:

$b = f(a)$ is not smaller than $f(X) = c$

$b = f(a)$ is not smaller than $p(X,d)$

$b = f(a)$ is not smaller than $\sim q(d)$

$\gg b \neq c \mid f(a) = b \mid p(a,d) \mid \sim q(d)$

$T1 \neq T2 \mid c^l$ + Equality resolution

$T1\theta \equiv T2\theta$

$(T1 \neq T2)\theta$ is not smaller than any literal in $(T1 \neq T2 \mid c^l)\theta$

$\gg c^l\theta$

Example

$f(a) \neq Y \mid f(X) = c \mid p(f(Y),g(X))$ + Equality resolution

T1 is $f(a)$

T2 is Y

c^l is $f(X) = c \mid p(f(Y),g(X))$

θ is $\{Y/f(a)\}$

It must be the case that:

$f(a) \neq Y$ is not smaller than $f(X) = c$

$f(a) \neq Y$ is not smaller than $p(f(f(a)),g(X))$

$\gg f(X) = c \mid p(f(f(a)),g(X))$

Term and Literal Ordering

Superposition requires a ground completable reduction ordering on the terms. Superposition can also use a ground completable simplification ordering, which is even stronger. [Knuth-Bendix Ordering](#) provides a suitable term ordering. Additionally, superposition requires an ordering on the literals of clauses, with some special cases:

- An equality literal is represented by its largest argument.
- A negative literal is greater than a positive literal with the equivalent atom.

A literal ordering extends a term ordering to literals, and must also be stable under substitution and renaming. An example of a literal ordering that can be used with superposition is *bag ordering*: Given two bags, $B1$ and $B2$, discard pairs of identical elements taken one from each bag, to produce $B1'$ and $B2'$. Then $B1 > B2$ if for each $x2$ in $B2'$ there exists a $x1$ in $B1'$ such that $x1 > x2$ (the $x1$ can be used for multiple $x2$ s). Literals are compared by bag-comparing representative bags:

- A positive non-equality literal L is represented by a bag $\{L\}$ (a bag of a bag of the atom)
- A negative non-equality literal $\sim L$ is represented by a bag $\{L,L\}$ (a bag of a bag of two copies of the atom – the effect is to make negative literals bigger than positive ones)
- A positive equality literal $T1 = T2$ is represented by a bag $\{T1\}, \{T2\}$ (a bag of two bags, each containing one term)
- A negative equality literal $T1 \neq T2$ is represented by a bag $\{T1, T2\}$ (a bag containing one bag, containing both terms).

An optional preordering can be imposed, that all negative literals are larger than positive literals, and then do bag ordering after that.

Example

```
S = { even(sum(two_squared,b)),
      two_squared = four,
      ~zero(X) | difference(four,X) = sum(four,X),
      zero(b),
      ~even(difference(two_squared,b)) }
```

$\rho : = > \text{even} > \text{zero} > = > \text{sum} > \text{difference} > \text{four} > \text{two_squared} > b$

$\omega(\text{even}) \rightarrow 5$

$\omega(\text{zero}) \rightarrow 5$

$\omega(=) \rightarrow 3$

$\omega(\text{sum}) \rightarrow 4$


```

ω(difference) → 3
ω(four) → 4
ω(two_squared) → 4
ω(b) → 3
ω(X) → 1

~zero(X) → {{zero(X), zero(X)}}
difference(four, X) = sum(four, X) → {{difference(four, X)}, {sum(four, X)}}

difference(four, X) > zero(X)
... therefore ...
{difference(four, X)} > {zero(X), zero(X)}
... therefore ...
{{difference(four, X)}, {sum(four, X)}} > {{zero(X), zero(X)}}
... therefore ...
difference(four, X) = sum(four, X) > ~zero(X)
Therefore that literal is used, and sum(four, X) must be the chosen term for superposition.

```

Example

```

S = { even(sum(two_squared, b)),
      two_squared = four,
      ~zero(X) | difference(four, X) = sum(four, X),
      zero(b),
      ~even(difference(two_squared, b)) }

p := > even > zero > = > sum > difference > four > two_squared > b

ω(even) → 5
ω(zero) → 5
ω(=) → 3
ω(sum) → 4
ω(difference) → 3
ω(four) → 4
ω(two_squared) → 4
ω(b) → 3
ω(X) → 1

two_squared = four
~zero(X) | difference(four, X) = sum(two_squared, X)
~zero(b) | even(difference(four, b))
~zero(b) | even(difference(two_squared, b))
~zero(b)

S+ ~zero(X) | difference(four, X) = sum(four, X)
  » ~zero(X) | difference(four, X) = sum(two_squared, X)
S+ even(sum(two_squared, b))
  » ~zero(b) | even(difference(four, b))
S+ two_squared = four
  » ~zero(b) | even(difference(two_squared, b))
+ ~even(difference(two_squared, b))
  » ~zero(b)
+ zero(b)
  » FALSE

```

Pure Equality Problems

Example (ignoring the term and literal ordering issues)

```

S = { a = f(X) | Y = f(b),
      f(b) = c,
      a != c }

a = f(X) | Y = f(b) + Equality factoring
  » a = f(X) | f(X) != f(b)

```

$a = f(x) \mid f(x) \neq f(b)$	+ Equality resolution
	» $a = f(b)$
$f(b) = c$	+ $a \neq c$
	» $a \neq f(b)$
$a = f(b)$	+ $a \neq f(b)$
	» $a \neq a$
$a \neq a$	+ Equality resolution
	» FALSE

Practical Notes:

- Rewriting is really needed to make this effective.
- Literal selection strategies, which determine which literal to select, are important.
- Without the ordering, superposition is paramodulation, equality factoring is conditional factoring of equality literals and equality resolution is resolution with reflexivity.

Transforming Not Pure Equality Problems

Problems that use predicates other than $=/2$ and $\neq/2$ can be converted to pure equality, so that they can be solved using superposition plus equality resolution and factoring. A positive literal L is converted to $L = \text{true}$, and a negative literal M is converted to $M \neq \text{true}$. In this manner all predicate symbols become function symbols. In the use of the inference rules it is necessary to ensure that no variable becomes instantiated to (what was previously) an atom, or to the new special constant `true`. Clauses containing $\text{true} = \text{true}$ are tautologies and can be discarded.

For the KBO, `true` is mapped to something less than all other symbols and greater than the mapping of the variables.

Example

```
S = { even(sum(two_squared,b)),
      two_squared = four,
      ~zero(X) | difference(four,X) = sum(four,X),
      zero(b),
      ~even(difference(two_squared,b)) }
```

... gets converted to ...

```
S = { even(sum(two_squared,b)) = true,
      two_squared = four,
      zero(X) != true | difference(four,X) = sum(four,X),
      zero(b) = true,
      even(difference(two_squared,b)) != true }
```

$\text{two_squared} = \text{four}$	+ $\text{even}(\text{sum}(\text{two_squared},b)) = \text{true}$
	» $\text{even}(\text{sum}(\text{four},b)) = \text{true}$
$\text{zero}(X) \neq \text{true} \mid \text{difference}(\text{four},X) = \text{sum}(\text{four},X)$	+ $\text{even}(\text{sum}(\text{four},b)) = \text{true}$
	» $\text{even}(\text{difference}(\text{four},b)) = \text{true} \mid \text{zero}(b) \neq \text{true}$
$\text{even}(\text{difference}(\text{four},b)) = \text{true} \mid \text{zero}(b) \neq \text{true}$	+ $\text{zero}(b) = \text{true}$
	» $\text{even}(\text{difference}(\text{four},b)) = \text{true} \mid \text{true} \neq \text{true}$
$\text{even}(\text{difference}(\text{four},b)) = \text{true} \mid \text{true} \neq \text{true}$	+ Equality resolution
	» $\text{even}(\text{difference}(\text{four},b)) = \text{true}$
$\text{even}(\text{difference}(\text{four},b)) = \text{true}$	+ $\text{two_squared} = \text{four}$
	» $\text{even}(\text{difference}(\text{two_squared},b)) = \text{true}$
$\text{even}(\text{difference}(\text{two_squared},b)) = \text{true}$	+ $\text{even}(\text{difference}(\text{two_squared},b)) \neq \text{true}$
	» $\text{true} \neq \text{true}$
$\text{true} \neq \text{true}$	+ Equality resolution
	» FALSE

Example

```
S = { X = X,
      A != B | ~p(A) | p(B),
      X = c | X = d,
      p(a),
      p(b),
      a != b,
```



Linear Refinements

Linear Resolution

Given an input set of clauses and a clause C_1 chosen from the input set, a linear deduction of C_n from the input set, with top clause C_1 , is a sequence of centre clauses C_1, \dots, C_n . Each deduced clause C_{i+1} , $i = 1..n-1$, is deduced from the centre clause C_i and a side clause. The side clauses are the input set and C_1, \dots, C_{i-1} . For any C_i , the centre clauses C_1 to C_{i-1} are the ancestor clauses of C_i . Refutations are written in a linear style.

Example	
$S8 = \{ p(X) \mid l(X) \mid q, \quad (\text{top clause})$ $\quad \sim q \mid l(Y),$ $\quad \sim l(Z) \mid \sim m(S) \mid r(Z),$ $\quad \sim r(b) \mid \sim l(T) \mid \sim s(b),$ $\quad s(b),$ $\quad m(a) \mid r(b),$ $\quad \sim p(b) \}$	
Center clauses	Side clauses
$p(X) \mid l(X) \mid q$	$+ \sim q \mid l(Y)$
$= p(X) \mid l(X) \mid l(Y)$	$+ \sim l(Z) \mid \sim m(S) \mid r(Z)$
$= p(X) \mid \sim m(S) \mid r(X)$	$+ \sim r(b) \mid \sim l(T) \mid \sim s(b)$
$= p(b) \mid \sim m(S) \mid \sim l(T) \mid \sim s(b) + s(b)$	
$= p(b) \mid \sim m(S) \mid \sim l(T)$	$+ p(X) \mid l(X) \mid l(Y)$
$= p(X) \mid p(b) \mid \sim m(S)$	$+ m(a) \mid r(b)$
$= p(X) \mid p(b) \mid r(b)$	$+ \sim r(b) \mid \sim l(T) \mid \sim s(b)$
$= p(X) \mid p(b) \mid \sim l(T) \mid \sim s(b) + s(b)$	
$= p(X) \mid p(b) \mid \sim l(T)$	$+ p(X) \mid l(X) \mid l(Y)$
$= p(X') \mid p(b) \mid p(X)$	$+ \sim p(b)$
$= \text{FALSE}$	

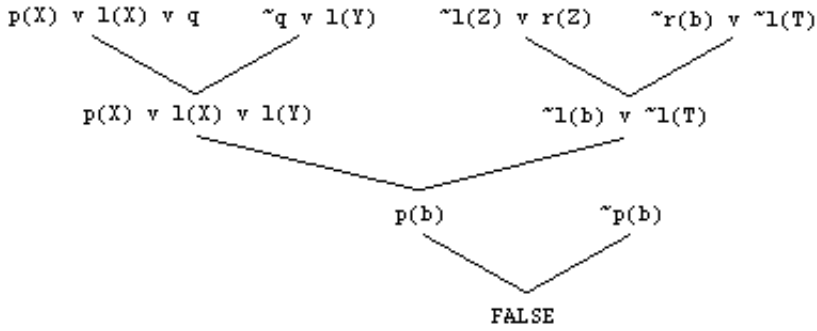
Note the duplicated steps (which will be dealt with by refinements).

Example
$S9 = \{ \sim p(X) \mid \sim q(X,b), \quad (\text{Top clause})$ $\quad \sim s(W) \mid q(W,W),$ $\quad p(b),$ $\quad q(Y,T) \mid r(T) \mid s(Y),$ $\quad \sim r(Z) \mid s(X) \}$

Completeness by Construction

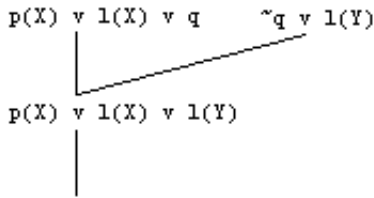
Any normal refutation can be converted to a linear refutation. Consider the proof tree from this example.

Example	
$S4a = \{ \sim p(b),$ $\quad p(X) \mid l(X) \mid q, \quad (\text{Top clause})$ $\quad \sim q \mid l(Y),$ $\quad \sim l(Z) \mid r(Z),$ $\quad \sim r(b) \mid \sim l(T) \}$	
$p(X) \mid l(X) \mid q$	$+ \sim q \mid l(Y)$
	$= p(X) \mid l(X) \mid l(Y)$
$\sim l(Z) \mid r(Z)$	$+ \sim r(b) \mid \sim l(T)$
	$= \sim l(b) \mid \sim l(T)$
$p(X) \mid l(X) \mid l(Y) + \sim l(b) \mid \sim l(T)$	
	$= p(b)$
$p(b)$	$+ \sim p(b)$
	$= \text{FALSE}$

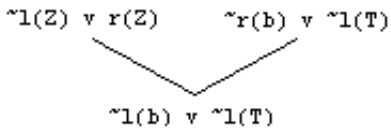


The linearisation proceeds as follows. Choose any leaf of the proof tree. This clause becomes the top clause of the linear refutation. Follow the refutation down until a resolution with a non-input clause is found. Call the centre clause at that point c^l and the other clause s^l . Call the resolved upon literals from the centre clause cl^l , and the resolved upon literals from the non-input clause sl^l . Find one of the originals from sl^l in a leaf in the subtree; call it L^s . Recursively linearise the sub-refutation from that leaf down to the non-input clause. That linearised sub-refutation is then inserted under c^l . This will leave the literals $sl^l - L^s$ in the resultant centre clause. An ancestor resolution against the literals cl^l in c^l will get to the resolvent (possibly with some literals inherited from $c^l - cl^l$ - they are collapsed in the next resolution step or can be factored away) below the point where the non-input clause is used in the original refutation.

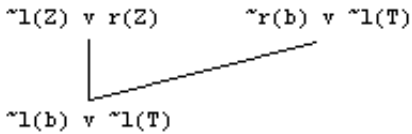
In the above example, choose $p(X) \mid l(X) \mid q$ as the initial leaf. The refutation is followed down from $p(X) \mid l(X) \mid q$ to the resolution between $c^l = p(X) \mid l(X) \mid l(Y)$ and $s^l = \sim l(b) \mid \sim l(T)$. $cl^l = l(X) \mid l(Y)$ and $sl^l = \sim l(b) \mid \sim l(T)$.



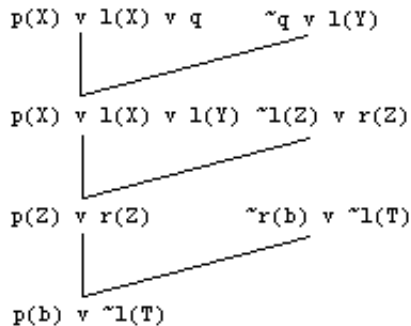
The subtree to be linearised is



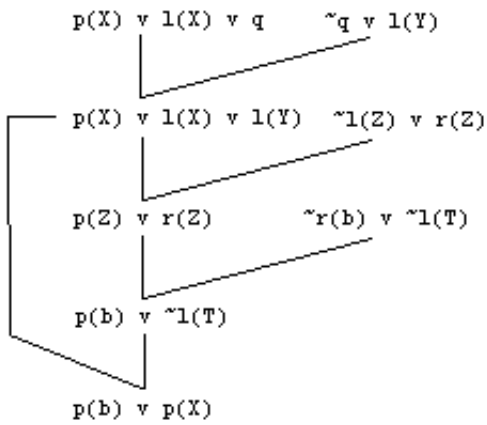
Choosing $L^s = \sim l(Z)$ in $\sim l(Z) \mid r(Z)$, this linearises trivially to



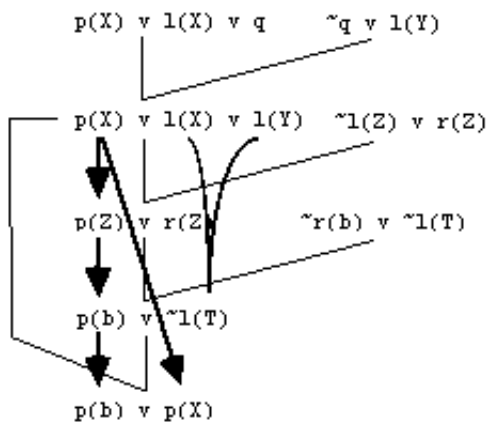
which is then inserted after the initial linear segment, to get



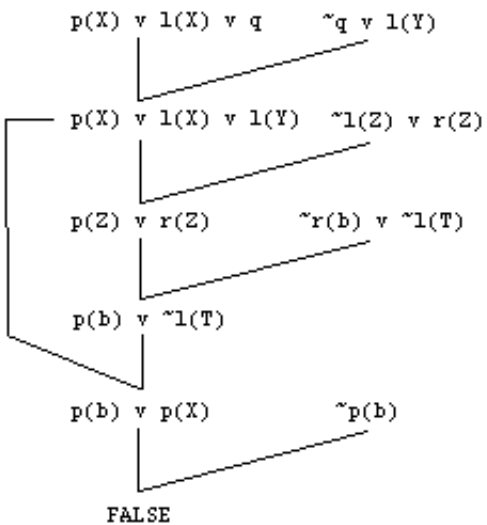
An ancestor resolution with $p(X) \mid l(X) \mid l(Y)$ is then needed to remove $sl^l - L^s = \sim l(T)$ from $p(b) \mid \sim l(T)$.



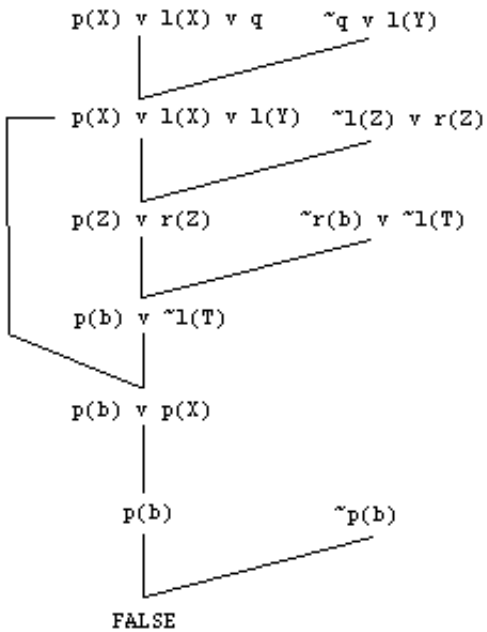
Note the extra $c|_{-CL} = p(X)$ that is introduced. Any extra literals in $c|_{-CL}$ subsume literals inherited from the ancestor clause. The extra literals in $c|_{-CL}$ "subsume" the inherited ones because the inherited ones may have become instantiated in the subdeduction between the ancestor and the ancestor resolution.



So $p(X)$ subsumes the $p(b)$ inherited from $p(X) \vee l(X) \vee l(Y)$, and can thus be resolved upon at the same time as the inherited one.



Alternatively the extra ancestor literals can be factored out.



Example
$S_9 = \{ \sim p(X) \mid \sim q(X,b),$ $\sim s(W) \mid q(W,W),$ $p(b),$ $q(Y,T) \mid r(T) \mid s(Y),$ $\sim r(Z) \mid s(X) \}$

Linear resolution is refutation complete.

Proof
By the construction above [Luckham 1970, discovered 1968].

Practical Notes:

- When building a linear refutation, it is not known in advance which of the input clauses will be used. Thus it is not known which of the input clauses will be leaves in the proof tree, and would be suitable as a top clause for the linear deduction. It is known, however, that if a satisfiable subset of the input set can be extracted, then no refutation can be found using that subset. Thus one of the clauses in the difference must be used in a refutation, and hence one of the clauses in the difference is necessarily a suitable top clause.
- Forward subsumption is used to prevent loops. If an ancestor clause subsumes a derived clause then you're in a loop and that alternative can be rejected. Backtracking takes place.
- During a linear deduction, it is necessary to allow for alternative side clauses at each step. One way to implement this is via *backtracking* in a consecutively bounded depth first search.
- There is no straight forward ANL loop implementation for linear deduction, as the ANL loop does not directly provide a means to record which clauses are input clauses and the ancestors of the chosen clause, and hence eligible for resolution. It is possible with extra information stored with each clause. In addition, if an input clause is subsumed by a resolvent, that resolvent needs to "become" an input clause.

Example										
$S_6 = \{ p(Z) \mid q(Z),$ $\sim p(X) \mid t(X),$ $\sim p(X) \mid s(X),$ $\sim t(b) \mid p(b),$ $\sim t(b) \mid s(b),$ $\sim s(b) \mid m(W),$ $\sim s(b) \mid \sim m(a),$ $\sim q(a),$ $\sim q(T) \mid s(T) \}$										
<table border="0"> <tr> <td>Center clauses</td> <td>Side clauses</td> </tr> <tr> <td>$p(Z) \mid q(Z)$</td> <td>$+ \sim q(a)$</td> </tr> <tr> <td>$= p(a)$</td> <td>$+ \sim p(X) \mid t(X)$</td> </tr> <tr> <td>$= t(a)$</td> <td>$+ \text{Backtrack}$</td> </tr> <tr> <td>$= p(a)$</td> <td>$+ \sim p(X) \mid s(X)$</td> </tr> </table>	Center clauses	Side clauses	$p(Z) \mid q(Z)$	$+ \sim q(a)$	$= p(a)$	$+ \sim p(X) \mid t(X)$	$= t(a)$	$+ \text{Backtrack}$	$= p(a)$	$+ \sim p(X) \mid s(X)$
Center clauses	Side clauses									
$p(Z) \mid q(Z)$	$+ \sim q(a)$									
$= p(a)$	$+ \sim p(X) \mid t(X)$									
$= t(a)$	$+ \text{Backtrack}$									
$= p(a)$	$+ \sim p(X) \mid s(X)$									

```

= s(a)          + Backtrack
= p(Z) | q(Z)  + ~q(T) | s(T)
= p(Z) | s(Z)  + ~s(b) | m(W)
= p(b) | m(W)  + ~s(b) | ~m(a)
= p(b) | ~s(b) + p(Z') | s(Z')
= p(b) | p(b)  + ~p(X) | t(X)
= t(b)          + ~t(b) | p(b)
= p(b)          + Backtrack
= t(b)          + ~t(b) | s(b)
= s(b)          + ~s(b) | ~m(a)
= ~m(a)         + ~s(b) | m(W)
= ~s(b)         + s(b)
= FALSE

```

Example

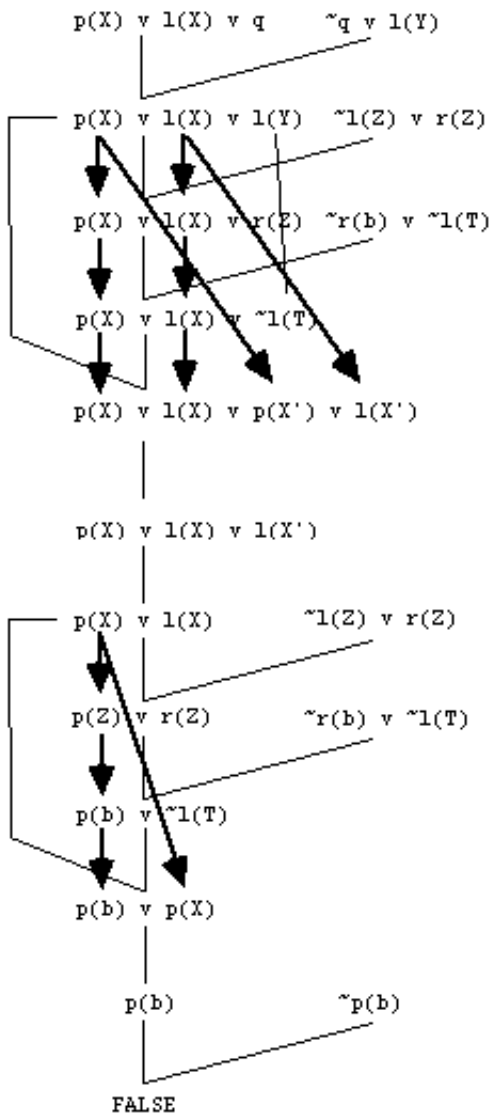
```

S9 = { ~p(X) | ~q(X,b),    (top clause)
      ~s(W) | q(W,W),
      p(b),
      q(Y,T) | r(T) | s(Y),
      ~r(Z) | s(X) }

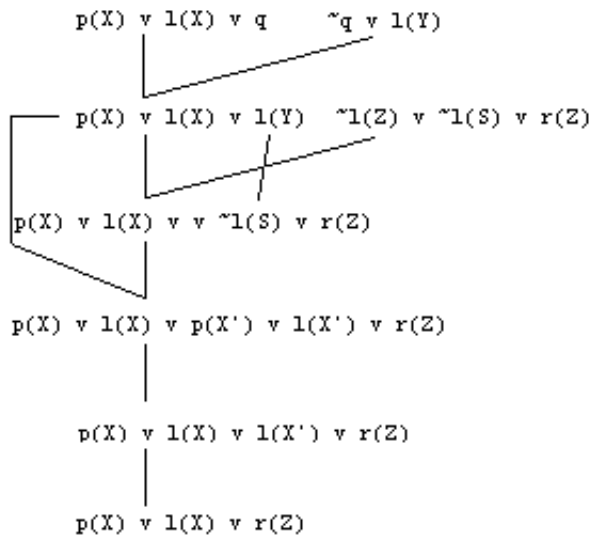
```

Linear Binary Resolution

In a linear refutation, any centre clause literals that are involved in a genuinely full resolution (i.e., c_L^l contains more than one literal) with an input side clause, can be resolved upon individually using binary resolution and duplicating the steps required. Let the resolved upon literal of c_L^l be L^c . If binary resolution is used in the ancestor resolution steps, then all the literals $c^l_{-L^c}$ from the ancestor clause will be in the resolvent. As before, these literals subsume the inherited copies, and can be factored out. In the above example:



Any extra literals from side clauses that are involved in full resolution steps can be dealt with by ancestor resolving with the parent centre clause (either immediately or later), and factoring the duplicated literals from the centre clause. The immediate case happens naturally as part of linearization, as shown in this extract from the linearization of example s4.



Thus linear resolution is refutation complete using binary resolution and 'compulsory' factoring of extra literals after ancestor resolution steps. The ancestor resolution and the following factoring will always shorten the centre clause.

Practical notes:

- Note that when a centre clause literal is resolved against, the other centre clause literals remain untouched until all the steps required to remove that centre clause literal are completed. That this is always the case is evident from the linearisation process; the steps that 'remove' the resolved upon literal from CL are all within a separate subtree of the normal resolution proof. Thus the selection of which centre clause literal to resolve upon is an AND decision, and can be committed to once made. A common selection is the right most literal.
- The rightmost selection policy makes it necessary to deal with extra literals from side clauses that are involved in full resolution steps later (rather than immediately as demonstrated above).

Example	
$S8 = \{ p(X) \mid l(X) \mid q, \quad (\text{top clause})$ $\quad \sim q \mid l(Y),$ $\quad \sim l(Z) \mid \sim m(S) \mid r(Z),$ $\quad \sim r(b) \mid \sim l(T) \mid \sim s(b),$ $\quad s(b),$ $\quad m(a) \mid r(b),$ $\quad \sim p(b) \}$	
Center clauses	Side clauses
$p(X) \mid l(X) \mid q$	$+ \sim q \mid l(Y)$
$= p(X) \mid l(X) \mid l(Y)$	$+ \sim l(Z) \mid \sim m(S) \mid r(Z)$
$= p(X) \mid l(X) \mid \sim m(S) \mid r(Z)$	$+ \sim r(b) \mid \sim l(T) \mid \sim s(b)$
$= p(X) \mid l(X) \mid \sim m(S) \mid \sim l(T) \mid \sim s(b) + s(b)$	
$= p(X) \mid l(X) \mid \sim m(S) \mid \sim l(T)$	$+ p(X') \mid l(X') \mid l(Y)$
$= p(X') \mid l(X') \mid p(X) \mid l(X) \mid \sim m(S) + \text{Factoring}$	
$= p(X) \mid l(X) \mid \sim m(S)$	$+ m(a) \mid r(b)$
$= p(X) \mid l(X) \mid r(b)$	$+ \sim r(b) \mid \sim l(T) \mid \sim s(b)$
$= p(X) \mid l(X) \mid \sim l(T) \mid \sim s(b)$	$+ s(b)$
$= p(X) \mid l(X) \mid \sim l(T)$	$+ p(X) \mid l(X) \mid l(Y)$
$= p(X') \mid l(X') \mid p(X) \mid l(X)$	$+ \text{Factoring}$
$= p(X) \mid l(X)$	$+ \sim l(Z) \mid \sim m(S) \mid r(Z)$
$= p(X) \mid \sim m(S) \mid r(X)$	$+ \sim r(b) \mid \sim l(T) \mid \sim s(b)$
$= p(b) \mid \sim m(S) \mid \sim l(T) \mid \sim s(b)$	$+ s(b)$
$= p(b) \mid \sim m(S) \mid \sim l(T)$	$+ p(X) \mid l(X)$
$= p(X) \mid p(b) \mid \sim m(S)$	$+ \text{Factoring}$
$= p(b) \mid \sim m(S)$	$+ m(a) \mid r(b)$
$= p(b) \mid r(b)$	$+ \sim r(b) \mid \sim l(T) \mid \sim s(b)$
$= p(b) \mid \sim l(T) \mid \sim s(b)$	$+ s(b)$
$= p(b) \mid \sim l(T)$	$+ p(X) \mid l(X)$
$= p(X) \mid p(b)$	$+ \text{Factoring}$
$= p(b)$	$+ \sim p(b)$
$= \text{FALSE}$	

Note the duplicated duplicated (not a typo) steps (which will be dealt with by refinements).

Example
$S9 = \{ \sim p(X) \mid \sim q(X,b), \quad (\text{top clause})$ $\quad \sim s(W) \mid q(W,W),$ $\quad p(b),$ $\quad q(Y,T) \mid r(T) \mid s(Y),$ $\quad \sim r(Z) \mid s(X) \}$

[Link to continuation if no server-side includes](#)

Linear Input (binary) Resolution

Given an input set of clauses and a clause C_1 chosen from the input set, a linear input deduction of C_n from the input set, with top clause C_1 , is a sequence of centre clauses C_1, \dots, C_n . Each deduced clause C_{i+1} , $i = 1..n-1$, is deduced from the centre clause C_i and a side clause chosen from the input set, by binary resolution. For any C_i , the centre clauses C_1 to C_{i-1} are the ancestor clauses of C_i . Linear input deduction is a refinement of linear deduction which does not permit any form of ancestor resolution.

Example
$S5 = \{ \sim p(Z),$ $q(a,b),$ $r(d), \quad (\text{Top clause})$ $p(Y) \mid \sim r(X) \mid \sim q(Y,X),$ $p(Y) \mid \sim r(X) \mid \sim s(Y,X),$ $s(c,d) \}$

Linear input deduction is complete for input sets of Horn clauses [Henschen & Wos, 1974], but is not complete for input sets that contain non-Horn clauses. Ringwood [1988] provides an interesting synopsis and references for the history of linear input deduction systems. Practical notes:

- If a refutation exists, one exists with one of the negative clauses as top clause. Thus it's safe to try only each of the negative clauses as top clauses.
- The selection of which centre clause literal to resolve upon is an AND decision, and can be committed to once made. A common selection is the right most literal.
- During a linear deduction, it is necessary to allow for alternative side clauses at each step. One way to implement this is via *backtracking*.
- If the top clause is negative, then all center clauses are necessarily negative, and only positive literal in each mixed or positive clause can be resolved against at each step.

Example																
$S5 = \{ \sim p(Z),$ $q(a,b),$ $r(d),$ $p(Y) \mid \sim r(X) \mid \sim q(Y,X),$ $p(Y) \mid \sim r(X) \mid \sim s(Y,X),$ $s(c,d) \}$																
<table> <thead> <tr> <th>Center clauses</th> <th>Side clauses</th> </tr> </thead> <tbody> <tr> <td>$\sim p(Z)$</td> <td>$+ p(Y) \mid \sim r(X) \mid \sim q(Y,X)$</td> </tr> <tr> <td>$= \sim r(X) \mid \sim q(Z,X) + q(a,b)$</td> <td></td> </tr> <tr> <td>$= \sim r(b)$</td> <td>$+ \text{Backtrack}$</td> </tr> <tr> <td>$= \sim p(Z)$</td> <td>$+ p(Y) \mid \sim r(X) \mid \sim s(Y,X)$</td> </tr> <tr> <td>$= \sim r(X) \mid \sim s(Z,X) + s(c,d)$</td> <td></td> </tr> <tr> <td>$= \sim r(d)$</td> <td>$+ r(d)$</td> </tr> <tr> <td>$= \text{FALSE}$</td> <td></td> </tr> </tbody> </table>	Center clauses	Side clauses	$\sim p(Z)$	$+ p(Y) \mid \sim r(X) \mid \sim q(Y,X)$	$= \sim r(X) \mid \sim q(Z,X) + q(a,b)$		$= \sim r(b)$	$+ \text{Backtrack}$	$= \sim p(Z)$	$+ p(Y) \mid \sim r(X) \mid \sim s(Y,X)$	$= \sim r(X) \mid \sim s(Z,X) + s(c,d)$		$= \sim r(d)$	$+ r(d)$	$= \text{FALSE}$	
Center clauses	Side clauses															
$\sim p(Z)$	$+ p(Y) \mid \sim r(X) \mid \sim q(Y,X)$															
$= \sim r(X) \mid \sim q(Z,X) + q(a,b)$																
$= \sim r(b)$	$+ \text{Backtrack}$															
$= \sim p(Z)$	$+ p(Y) \mid \sim r(X) \mid \sim s(Y,X)$															
$= \sim r(X) \mid \sim s(Z,X) + s(c,d)$																
$= \sim r(d)$	$+ r(d)$															
$= \text{FALSE}$																

Example
$S7 = \{ \sim t(X) \mid \sim r(X),$ $r(a) \mid \sim p(S) \mid \sim q(S),$ $p(a),$ $p(b),$ $q(b),$ $t(X) \mid \sim p(a) \}$

Prolog is linear input resolution using a negative top clause and a depth first search choosing the left most literal of the centre clause at each step.

Example
<p>The set S7 would be written as a Prolog program:</p> <pre> r(a):- p(S), q(S). p(a). p(b). q(b). t(X):- p(a). </pre> <p>and a query at the Prolog prompt:</p>

?- t(X),r(X).

Example

SN = { $\sim p(b,X) \mid \sim q(X),$
r(a),
 $\sim q(X) \mid r(X),$
r(b) $\mid \sim q(c),$
p(b,a) $\mid \sim r(a),$
p(Y,c) $\mid \sim r(Y),$
q(c) }

Example

The set SN would be written as a Prolog program:

r(a).

r(X):-
q(X).

r(b):-
q(c).

p(b,a):-
r(a).

p(Y,c):-
r(Y).

q(c).

and a query at the Prolog prompt:

?- p(b,X),q(X).

Exam Style Questions

1. Use linear-input resolution to derive the empty clause from the set

S = { $\sim r(Y) \mid \sim p(Y),$
p(b),
r(a),
p(S) $\mid \sim p(b) \mid \sim r(S),$
r(c) }

Chain Format Linear Refinements

It was noted in the section on linear resolution that in an ancestor binary resolution the ancestor literals that are not resolved upon subsume the corresponding ones that are inherited through the linear steps, and can therefore be factored out. The chain format linear resolution systems take advantage of this knowledge, and simply record the existence of \mathcal{L}^c and leave $c^{\perp} - \mathcal{L}^c$ implicit, secure in the knowledge that the literals $c^{\perp} - \mathcal{L}^c$ can be factored away when an ancestor resolution is performed. These recorded literals, called *A-literals*, are discarded when the deduction that removes \mathcal{L}^c (corresponding to a completed linearisation of a subtree in a normal deduction) is completed, after which one of $c^{\perp} - \mathcal{L}^c$ is necessarily resolved upon and is no longer available for factoring. The chain format systems also use the ability to ignore other centre chain literals once one has been selected for attention (the AND decision).

Chains

A *chain* is an ordered sequence of literals. Each literal in a chain is classified as either an *A*-, *B*-, or *C*-literal. The disjunction of the B-literals in a chain makes up the clause that is represented by the chain. An uninterrupted sequence of B-literals in a chain is called a *cell*. Input clauses are used to form *input chains* which consist entirely of B-literals. Chains that contain a single B-literal are called unit chains. Centre clauses of linear deductions are represented by *centre chains*. Centre chains may contain A-, B- and C-literals. Various items of information may be associated with the literals in a chain.

Notation :

- A-literals are placed in rectangles.
- B-literals stand free.
- C-literals are placed in ellipses.
- Literals in a chain are simply separated by spaces (the | is omitted).

Core operations

All chain format linear deduction systems have a common core of deduction operations. There are two inference operations based on binary resolution and one bookkeeping operation. The inference operations are *extension* and *reduction*.

Extension resolves a B-literal in the rightmost cell of a centre chain against a B-literal in an input chain. The deduced chain is formed by

- Placing the resolved upon centre chain B-literal at the right-hand end of the centre chain and reclassifying it as an A-literal
- Adding the non-resolved upon input chain B-literals to the right of the new A-literal.

Reduction unifies a B-literal in a centre chain with a complementary A-literal to its left. The deduced chain is formed by removing the B-literal from the centre chain. Reduction implements ancestor resolution, followed by a sequence of factoring operations. The reduction of a B-literal against the A-literal immediately to its left may also be viewed as factoring of the corresponding input chain.

The bookkeeping operation is *truncation* (also called contraction). Truncation removes an A- or C-literal from the right-hand end of a centre chain. In some deduction systems, the truncation of an A-literal may cause the insertion of another A- or C-literal.

The Model Elimination (ME) Procedure

Example			
$S_4 = \{ \sim p(b),$ $p(X) \mid l(X) \mid q, \quad (\text{top clause})$ $\sim q \mid l(Y),$ $\sim l(Z) \mid \sim l(S) \mid r(Z),$ $\sim r(b) \mid \sim l(T) \}$			
Center clauses	Side clauses	Center chains	Side chains
$p(X) \mid l(X) \mid q$	$+ \sim q \mid l(Y)$	$\mid p(X) \mid l(X) \mid q$	$+ \sim q \mid l(Y)$
$= p(X) \mid l(X) \mid l(Y)$	$+ \sim l(Z) \mid \sim l(S) \mid r(Z)$	$\mid = p(X) \mid l(X) \mid [q] \mid l(Y)$	$+ \sim l(Z) \mid \sim l(S) \mid r(Z)$
$= p(X) \mid l(X) \mid \sim l(S) \mid r(Y)$	$+ \sim r(b) \mid \sim l(T)$	$\mid = p(X) \mid l(X) \mid [q] \mid [l(Y)] \mid \sim l(S) \mid r(Y)$	$+ \sim r(b) \mid \sim l(T)$
$= p(X) \mid l(X) \mid \sim l(S) \mid \sim l(T)$	$+ p(X') \mid l(X') \mid l(Y)$	$\mid = p(X) \mid l(X) \mid [q] \mid [l(b)] \mid \sim l(S) \mid [r(b)] \mid \sim l(T)$	$+ \text{Reduction}$
$= p(X) \mid l(X) \mid \sim l(S) \mid p(X') \mid l(X')$	$+ \text{Factoring}$	$\mid = p(X) \mid l(X) \mid [q] \mid [l(b)] \mid \sim l(S) \mid [r(b)]$	$+ \text{Truncation}$
$= p(X) \mid l(X) \mid \sim l(S)$	$+ p(X') \mid l(X') \mid l(Y)$	$\mid = p(X) \mid l(X) \mid [q] \mid [l(b)] \mid \sim l(S)$	$+ \text{Reduction}$
$= p(X) \mid l(X) \mid p(X') \mid l(X')$	$+ \text{Factoring}$	$\mid = p(X) \mid l(X) \mid [q] \mid [l(b)]$	$+ \text{Truncation}$
$= p(X) \mid l(X)$	$+ \sim l(Z) \mid \sim l(S) \mid r(Z)$	$\mid = p(X) \mid l(X)$	$+ \sim l(Z) \mid \sim l(S) \mid r(Z)$
$= p(X) \mid \sim l(S) \mid r(X)$	$+ \sim r(b) \mid \sim l(T)$	$\mid = p(X) \mid [l(X)] \mid \sim l(S) \mid r(X)$	$+ \sim r(b) \mid \sim l(T)$
$= p(b) \mid \sim l(S) \mid \sim l(T)$	$+ p(X) \mid l(X)$	$\mid = p(b) \mid [l(b)] \mid \sim l(S) \mid [r(b)] \mid \sim l(T)$	$+ \text{Reduction}$

= p(b) ~l(S) p(X)	+ Factoring	= p(b) [l(b)] ~l(S) [r(b)]	+ Truncation
= p(b) ~l(S)	+ p(X) l(X)	= p(b) [l(b)] ~l(S)	+ Reduction
= p(b) p(X)	+ Factoring	= p(b) [l(b)]	+ Truncation
= p(b)	+ ~p(b)	= p(b)	+ ~p(b)
		= [p(b)]	+ Truncation
= FALSE		= FALSE	

Practical Notes:

- Each of the chain format linear deduction systems can use *admissibility* restrictions to reject some centre chains. Common admissibility restrictions are :
 - No two non-B-literals have identical atoms (in a loop, or should have reduced).
 - No A- or C-literal is to the left of an identical B-literal (in a loop).
 - No B-literal is complementarily identical to another B-literal in the same cell, or to the A-literal immediately to the right of the cell (a tautology has been used).
 - No A-literal is complementarily subsumed by a unit input chain, unless the A-literal is rightmost in the centre chain (should have extended).
- An ANL loop implementation places the input clauses in the CanBeUsed list, and a copy of the top chain in the ToBeUsed list. The ChosenClause is simply removed from the ToBeUsed list (not transferred to the CanBeUsed list). A special form of subsumption is required (I'VE GOT THE DETAILS SOMEWHERE (IMPLEMENTED AT LEAST)).

Example

```
S8 = { p(X) | l(X) | q, (top clause)
      ~q | l(Y),
      ~l(Z) | ~m(S) | r(Z),
      ~r(b) | ~l(T) | ~s(b),
      s(b),
      m(a) | r(b),
      ~p(b) }
```

Center chains

```
p(X) l(X) q
= p(X) l(X) [q] l(Y)
= p(X) l(X) [q] [l(Z)] ~m(S) r(Z)
= p(X) l(X) [q] [l(b)] ~m(S) [r(b)] ~l(T) ~s(b)
= p(X) l(X) [q] [l(b)] ~m(S) [r(b)] ~l(T) [s(b)]
= p(X) l(X) [q] [l(b)] ~m(S) [r(b)] ~l(T)
= p(X) l(X) [q] [l(b)] ~m(S) [r(b)]
= p(X) l(X) [q] [l(b)] ~m(S)
= p(X) l(X) [q] [l(b)] [~m(a)] r(b)
= p(X) l(X) [q] [l(b)] [~m(a)] [r(b)] ~l(T) ~s(b)
= p(X) l(X) [q] [l(b)] [~m(a)] [r(b)] ~l(T) [~s(b)]
= p(X) l(X) [q] [l(b)] [~m(a)] [r(b)] ~l(T)
= p(X) l(X) [q] [l(b)] [~m(a)] [r(b)]
= p(X) l(X)
= p(X) [l(X)] ~m(S) r(X)
= p(b) [l(b)] ~m(S) [r(b)] ~l(T) ~s(b)
= p(b) [l(b)] ~m(S) [r(b)] ~l(T) [~s(b)]
= p(b) [l(b)] ~m(S) [r(b)] ~l(T)
= p(b) [l(b)] ~m(S) [r(b)]
= p(b) [l(b)] ~m(S)
= p(b) [l(b)] [~m(a)] r(b)
= p(b) [l(b)] [~m(a)] [r(b)] ~l(T) ~s(b)
= p(b) [l(b)] [~m(a)] [r(b)] ~l(T) [~s(b)]
= p(b) [l(b)] [~m(a)] [r(b)] ~l(T)
= p(b) [l(b)] [~m(a)] [r(b)]
= p(b)
= [p(b)]
= FALSE
```

Side clauses

```
+ ~q l(Y)
+ ~l(Z) ~m(S) r(Z)
+ ~r(b) ~l(T) ~s(b)
+ s(b)
+ Truncate
+ Reduce
+ Truncate
+ m(a) r(b)
+ ~r(b) ~l(T) ~s(b)
+ s(b)
+ Truncate
+ Reduce
+ Truncate
+ ~l(Z) ~m(S) r(Z)
+ ~r(b) ~l(T) ~s(b)
+ s(b)
+ Truncate
+ Reduce
+ Truncate
+ m(a) r(b)
+ ~r(b) ~l(T) ~s(b)
+ s(b)
+ Truncate
+ Reduce
+ Truncate
+ ~p(b)
+ Truncate
```

Example

```

S9 = { ~p(X) | ~q(X,b),    (top clause)
      ~s(W) | q(W,W),
      p(b),
      q(Y,T) | r(T) | s(Y),
      ~r(Z) | s(X) }

```

Problem – repeated deductions.

Linear resolution with Selection function (SL-resolution)

Factoring provides a mechanism by which two literals, that could possibly both be removed from the centre chain (proved to not be in any model of the input set) by the same sequence of deduction operations, are combined so that they may be removed simultaneously by a single subdeduction. SL-resolution avoids repeated deductions by using (m-)factoring.

Example

```

S4 = { ~p(b),
      p(X) | l(X) | q,    (top clause)
      ~q | l(Y),
      ~l(Z) | ~l(S) | r(Z),
      ~r(b) | ~l(T) }

p(X) l(X) q                + ~q l(Y)
= p(X) l(X) [q] l(Y)      + Factoring
= p(X) l(X) [q]           + Truncation
= p(X) l(X)                + ~l(Z) ~l(S) r(Z)
= p(X) [l(X)] ~l(S) r(X)   + ~r(b) ~l(T)
= p(b) [l(b)] ~l(S) [r(b)] ~l(T) + Reduction
= p(b) [l(b)] ~l(S) [r(b)]   + Truncation
= p(b) [l(b)] ~l(S)         + Reduction
= p(b) [l(b)]               + Truncation
= p(b)                       + ~p(b)
= [p(b)]                     + runcation
= FALSE

```

The steps that are repeated in the simple ME proof are only performed once here, because the unifiable literals are factored. Factoring thus preempts the necessity for repeating the subdeduction, and results in a shorter refutation. The preemptive nature of factoring can be a disadvantage – it may be inappropriate or even impossible to remove both the literals using a single subdeduction. In the following example, this would be the case if $\sim p(a)$ were to be replaced by $\sim l(b)$ and $\sim p(b)$ in the input set. Unless the two factored literals are identical, it is necessary to permit the alternative of not factoring, i.e., a choice point must be established. This increases the size of the search space.

Example

```

S8 = { p(X) | l(X) | q,    (top clause)
      ~q | l(Y),
      ~l(Z) | ~m(S) | r(Z),
      ~r(b) | ~l(T) | ~s(b),
      s(b),
      m(a) | r(b),
      ~p(b) }

```

Example

```

S9 = { ~p(X) | ~q(X,b),    (top clause)
      ~s(W) | q(W,W),
      p(b),
      q(Y,T) | r(T) | s(Y),
      ~r(Z) | s(X) }

```

The ME Procedure with Lemmas

The truncation operation of the ME procedure includes a mechanism by which lemma chains can be produced and added to the input set. The literals that constitute a lemma chain are determined from integer *scope values*, that are associated with the A-literals in centre chains. Scope values are initially set to 0, and may be modified when a reduction takes place. If the number of A-literals to the right of the A-literal involved

is greater than its scope value, its scope value is set to that number of A-literals. On truncation of an A-literal, a lemma consisting of the negated A-literal and the negations of all A-literals whose scope values are equal to the numbers of A-literals to their right, is created. The scope values of the A-literals used is decremented. Lemma chains added to the input set may be used in subsequent extension operations. Using a lemma is equivalent to repeating (possibly an instance of) the subdeduction that lead to the creation of the lemma, i.e., the repetition is avoided.

Example	
$S8 = \{ p(X) \mid l(X) \mid q, \quad (\text{top clause})$ $\quad \sim q \mid l(Y),$ $\quad \sim l(Z) \mid \sim m(S) \mid r(Z),$ $\quad \sim r(b) \mid \sim l(T) \mid \sim s(b),$ $\quad s(b),$ $\quad m(a) \mid r(b),$ $\quad \sim p(b) \}$	
Center chains	Side clauses
$p(X) \mid l(X) \mid q$	$+ \sim q \mid l(Y)$
$= p(X) \mid l(X) \mid [q]^0 \mid l(Y)$	$+ \sim l(Z) \mid \sim m(S) \mid r(Z)$
$= p(X) \mid l(X) \mid [q]^0 \mid [l(Z)]^0 \mid \sim m(S) \mid r(Z)$	$+ \sim r(b) \mid \sim l(T) \mid \sim s(b)$
$= p(X) \mid l(X) \mid [q]^0 \mid [l(b)]^0 \mid \sim m(S) \mid [r(b)]^0 \mid \sim l(T) \mid \sim s(b)$	$+ s(b)$
$= p(X) \mid l(X) \mid [q]^0 \mid [l(b)]^0 \mid \sim m(S) \mid [r(b)]^0 \mid \sim l(T) \mid [s(b)]^0$	$+ \text{Truncate}$
Lemma produced	$\sim s(b) \text{ (Subsumed)}$
$= p(X) \mid l(X) \mid [q]^0 \mid [l(b)]^0 \mid \sim m(S) \mid [r(b)]^0 \mid \sim l(T)$	$+ \text{Reduce}$
$= p(X) \mid l(X) \mid [q]^0 \mid [l(b)]^1 \mid \sim m(S) \mid [r(b)]^0$	$+ \text{Truncate}$
Lemma produced	$\sim l(b) \mid \sim r(b)$
$= p(X) \mid l(X) \mid [q]^0 \mid [l(b)]^0 \mid \sim m(S)$	$+ m(a) \mid r(b)$
$= p(X) \mid l(X) \mid [q]^0 \mid [l(b)]^0 \mid [\sim m(a)]^0 \mid r(b)$	$+ \sim l(b) \mid \sim r(b)$
$= p(X) \mid l(X) \mid [q]^0 \mid [l(b)]^0 \mid [\sim m(a)]^0 \mid [r(b)]^0 \mid \sim l(b)$	$+ \text{Reduce}$
$= p(X) \mid l(X) \mid [q]^0 \mid [l(b)]^2 \mid [\sim m(a)]^0 \mid [r(b)]^0$	$+ \text{Truncate}$
Lemma produced	$\sim l(b) \mid \sim r(b) \text{ (Subsumed)}$
$= p(X) \mid l(X) \mid [q]^0 \mid [l(b)]^1 \mid [\sim m(a)]^0$	$+ \text{Truncate}$
Lemma produced	$\sim l(b) \mid m(a) \text{ (Subsumed)}$
$= p(X) \mid l(X) \mid [q]^0 \mid [l(b)]^0$	$+ \text{Truncate}$
Lemma produced	$\sim l(b) \text{ (Subsumes } \sim l(b) \mid \sim r(b), \sim l(b) \mid m(a))$
$= p(X) \mid l(X) \mid [q]^0$	$+ \text{Truncate}$
Lemma produced	$\sim q \text{ (Subsumes } \sim q \mid l(Y))$
$= p(X) \mid l(X)$	$+ \sim l(b)$
$= p(b) \mid [l(b)]^0$	$+ \text{Truncate}$
Lemma produced	$\sim l(b) \text{ (Subsumed)}$
$= p(b)$	$+ \sim p(b)$
$= [p(b)]^0$	$+ \text{Truncate}$
Lemma produced	$\sim p(b) \text{ (Subsumed)}$
$= \text{FALSE}$	

The steps that are repeated in the simple ME proof are only performed once here. The use of the lemma $\sim l(b)$ avoids the repeated subdeduction. The lemma 'contains' that subdeduction. The most striking feature of lemmas is their persistent nature – they remain in the input set even if the branch of the search which creates the lemma chain leads to failure. This has both beneficial and detrimental effects. On the up side, as in the example, a lemma chain may be used in a branch of the search other than that in which it was created. In the environment of a consecutively bounded search, a lemma created in one iteration of the search may be used in subsequent iterations. On the down side, the persistent nature of lemma chains often increases the size of the search space unacceptably, due to the increased number of side chains available for use in extension operations. This detrimental effect has been noted in various places, the principal problem being cited that "lemmas tend to be highly redundant – they are often subsumed by other lemmas and input chains" [Shostak, 1976, p. 63].

Practical Notes:

- As the lemma mechanism is not necessary for the completeness of the host deduction system, restrictions can be used to reduce this problem. Examples include keepig a lemma only if subsumes another clause, or additionally keeping unit lemmas. This approach has been taken by Fleisig, et al., [1974], whose implementation of the lemma mechanism includes a simplified subsumption test, and by Astrachan and Stickel [1992], whose METEOR system includes demodulation of lemmas and a subsumption test.

Example


```
S9 = { ~p(X) | ~q(X,b),    (top clause)
      ~s(W) | q(W,W),
      p(b),
      q(Y,T) | r(T) | s(Y),
      ~r(Z) | s(X) }
```

The Graph Construction (GC) Procedure

C-literals, introduced by the GC procedure, are inserted into the centre chain when an A-literal is truncated. A C-literal is formed from the negation of the truncated A-literal. The point in the centre chain at which the C-literal is inserted is called the C-point of the truncated A-literal, and is immediately to the right of the rightmost A-literal that would be negated to form part of a lemma produced by the truncation, or at the leftmost end of the centre chain if there are no such A-literals. The scope values of the A-literals are decremented as if a lemma had been created. B-literals may reduce against C-literals in a similar manner as against A-literals. The scope values of the A-literals whose scope values determined the insertion point of the C-literal are increased (to the number of A-literals to their right). C-reduction is equivalent to repeating (possibly an instance of) the subdeduction that lead to the insertion of the C-literal, i.e., the repetition is avoided.

Example

```
S8 = { p(X) | l(X) | q,    (top clause)
      ~q | l(Y),
      ~l(Z) | ~m(S) | r(Z),
      ~r(b) | ~l(T) | ~s(b),
      s(b),
      m(a) | r(b),
      ~p(b) }
```

Center chains

```
p(X) l(X) q
= p(X) l(X) [q]0 l(Y)
= p(X) l(X) [q]0 [l(Z)]0 ~m(S) r(Z)
= p(X) l(X) [q]0 [l(b)]0 ~m(S) [r(b)]0 ~l(T) ~s(b)
= p(X) l(X) [q]0 [l(b)]0 ~m(S) [r(b)]0 ~l(T) [s(b)]0 + Truncate
  C-literal produced for left-most position
= p(X) l(X) [q]0 [l(b)]0 ~m(S) [r(b)]0 ~l(T)
= p(X) l(X) [q]0 [l(b)]1 ~m(S) [r(b)]0
  C-literal produced
= p(X) l(X) [q]0 [l(b)]0 {~r(b)} ~m(S)
= p(X) l(X) [q]0 [l(b)]0 {~r(b)} [~m(a)]0 r(b)
= p(X) l(X) [q]0 [l(b)]1 {~r(b)} [~m(a)]0
  C-literal produced
= p(X) l(X) [q]0 [l(b)]0 {m(a)} {~r(b)}
  C-literal produced
= {~l(b)} p(X) l(X) [q]0
  C-literal produced
= {q} {~l(b)} p(X) l(X)
= {q} {~l(b)} p(b)
= {q} {~l(b)} [p(b)]0
  C-literal produced
= {q} {~l(b)}
```

Side clauses

```
+ ~q l(Y)
+ ~l(Z) ~m(S) r(Z)
+ ~r(b) ~l(T) ~s(b)
+ s(b)
+ Truncate
  ~s(b) (Subsumed and not inserted)
+ Reduce
+ Truncate
  {~r(b)}
+ m(a) r(b)
+ C-reduce
+ Truncate
  {m(a)} (Subsumed)
+ Truncate
  {~l(b)} (Subsumes ~l(b) ~r(b), ~l(b) m(a))
+ Truncate
  {~q} (Subsumes ~q l(Y))
+ C-reduce
+ ~p(b)
+ Truncate
  {~p(b)} (Subsumed)
+ Truncate
```

The steps that are repeated in the simple ME proof are only performed once here. The use of the C-literal ($\sim l(b)$) avoids the repeated subdeduction. The C-literal 'contains' that subdeduction.

Example

```
S9 = { ~p(X) | ~q(X,b),    (top clause)
      ~s(W) | q(W,W),
      p(b),
      q(Y,T) | r(T) | s(Y),
      ~r(Z) | s(X) }
```

