Resolution (14A)

Young W. Lim
8/15/14

based on the following document:
http://www.learnprolognow.org/ Learn Prolog Now!

Proposition

In Aristotelian logic a proposition is
a particular kind of sentence,
one which affirms or denies a **predicate** of a subject.

In formal logic a proposition is considered as
objects of a formal language.
A formal language begins with different types of symbols.

Propositional logic includes only
operators and propositional constants as symbols in its language.

The propositions in this language are
propositional constants (considered atomic propositions),
and composite propositions, (recursive application of operators to propositions).

Predicate logic include variables, operators, predicate and function symbols, and quantifiers
as symbols in their languages. The propositions in these logics are more complex.

From Old French, from Latin prōpositiō (“a proposing, design, theme, case”).

The content of an assertion that may be taken as being true or false and is considered abstractly without reference to the linguistic sentence that constitutes the assertion.

Predicate

(grammar) The part of the sentence (or clause) which states something about the subject or the object of the sentence.

"The dog barked very loudly"
the subject is "the dog"
the predicate is "barked very loudly".

(logic) A term of a statement,
where the statement may be true or false depending on whether the thing referred to by the values of the statement's variables has the property signified by that (predicative) term.

From Middle French predicate (French prédicat), from post-classical Late Latin praedicatum ("thing said of a subject"), a noun use of the neuter past participle of praedicare ("proclaim"), as Etymology 2, below.
From Latin predicātus, perfect passive participle of praedicō, from prae + dicō ("declare, proclaim"), from dicō ("say, tell").

Premise

A **premise** : an assumption that something is true.

an **argument** requires

a set of (at least) **two** declarative sentences ("propositions") known as the **premises**

along with **another** declarative sentence ("proposition") known as the **conclusion**.

two premises and **one conclusion** :
the basic **argument** structure

Because all men are mortal and Socrates is a man, Socrates is mortal.

From Middle English, from Old French premisse, from Medieval Latin premissa (“set before”) (premissa propositio (“the proposition set before”)), feminine past participle of Latin praemittere (“to send or put before”), from prae- (“before”) + mittere (“to send”).

2 premises
1 conclusion

3 propositions

Valid Argument Forms (Propositional)

Modus ponens (MP)

If A, then B
A
Therefore, B

Modus tollens (MT)

If A, then B
Not B
Therefore, not A

Hypothetical syllogism (HS)

If A, then B
If B, then C
Therefore, if A, then C

Disjunctive syllogism (DS)

A or B
Not A
Therefore, B

Modus ponens
(Latin) “the way that affirms by affirming"

Modus tollens
(Latin) "the way that denies by denying"

Syllogism
(Greek: συλλογισμός syllogismos) – "conclusion," "inference"

Modus Ponens

The Prolog resolution algorithm
based on the modus ponens form of inference

a general **rule** – the major premise and
a specific **fact** – the minor premise

All men are mortal	rule
Socrates is a man	fact
Socrates is mortal	

modus ponendo ponens
(Latin) “the way that affirms by affirming";
often abbreviated to **MP** or **modus ponens**

P implies Q;
P is asserted to be true,
so therefore Q must be true

one of the accepted mechanisms for the construction of deductive proofs
that includes the "rule of definition" and the "rule of substitution"

Facts	a	a
Rules	$a \rightarrow b$	b :- a
Conclusion	b	b

Facts	man(’Socrates’).
Rules	mortal(X) :- man(X).
Conclusion	mortal(’Socrates’).

Syllogism (1)

A syllogism (Greek: συλλογισμός – syllogismos – "conclusion," "inference") is

a kind of logical argument that applies deductive reasoning to arrive at a conclusion based on two or more **propositions** that are asserted or assumed to be true.

In its earliest form, defined by Aristotle,
from the combination of
a general statement (the major premise) and $\Leftrightarrow$ rule
a specific statement (the minor premise), $\Leftrightarrow$ fact
a conclusion is deduced.

For example, knowing
that all men are mortal (major premise) and $\Leftrightarrow$ rule
that Socrates is a man (minor premise), $\Leftrightarrow$ fact
we may validly conclude that Socrates is mortal.

Syllogism (2)

A categorical syllogism consists of three parts:

Major premise:	All humans are mortal.	⇔	major term	(the predicate of the conclusion)
Minor premise:	All Greeks are humans.	⇔	minor term	(the subject of the conclusion)
Conclusion:	All Greeks are mortal.			

Each part - a categorical proposition - two categorical terms

In Aristotle, each of the premises is in the form

"All A are B"	universal proposition
"Some A are B"	particular proposition
"No A are B"	universal proposition
"Some A are not B"	particular proposition

Each of the premises has one term in common with the conclusion:
this common term is called
a major term in a major premise (the predicate of the conclusion)
a minor term in a minor premise (the subject of the conclusion)

Mortal is the major term,
Greeks is the minor term.
Humans is the middle term

Modus Ponens (revisited)

Facts	a	a	minor term
Rules	$a \rightarrow b$	b :- a	major term
Conclusion	b	b	

Derivation

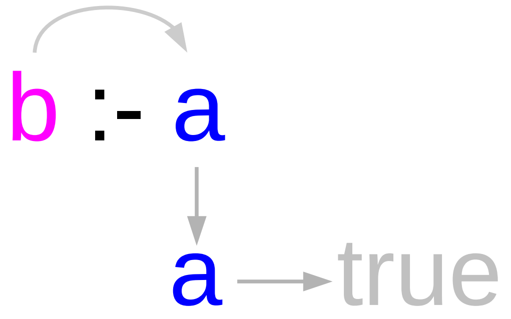
A **reversed modus ponens** is used in Prolog

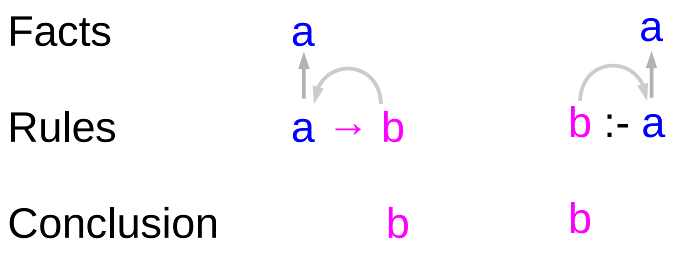
Prolog tries to prove that
a query (b) is a consequence of
the database content (a, a ⇒ b).

Using the major premise, it goes from b to a,
and using the minor premise, from a to true.

Such a sequence of goals is called a **derivation**.

A derivation can be **finite** or **infinite**.





Facts	a
Rules	b :- a
Conclusion	b

Horn Clause

the **resolvent** of **two Horn clauses** is itself **a Horn clause**
the **resolvent** of **a goal clause** and **a definite clause** is **a goal clause**

These properties of Horn clauses can lead to greater efficiencies in proving a theorem
(represented as the negation of a goal clause).

Propositional **Horn clauses** are also of interest in computational complexity,
where the problem of finding truth value assignments
to make a conjunction of **propositional** **Horn clauses** true
is a **P-complete** problem (in fact solvable in linear time), sometimes called **HORNSAT**.
(The **unrestricted Boolean** **satisfiability** problem is an **NP-complete** problem however.)
Satisfiability of **first-order** **Horn clauses** is undecidable.

By iteratively applying the resolution rule, it is possible

- to tell whether a propositional formula is **satisfiable**
- to prove that a first-order formula is unsatisfiable;

- this method may prove the **satisfiability** of a first-order formula,
- but not always, as it is the case for all methods for first-order logic

Definite Clause

clause : a disjunction of literals

Horn clause : a clause with at most **one positive (unnegated) literal (0, 1)**

definite clause	: a Horn clause with **exactly one positive literal (1)**
fact	: a definite clause with **no negative literals** but with **one positive literal only (1)**
goal clause	: a Horn clause **without a positive literal (0)**

Dual-Horn clause : a clause with at most **one negated literal**

	Disjunction form	Implication form
Definite clause	$\neg p \lor \neg q \lor \ldots \lor \neg t \lor u$	$u \leftarrow p \land q \land \ldots \land t$
Fact	u	u
Goal clause	$\neg p \lor \neg q \lor \ldots \lor \neg t$	$\text{false} \leftarrow p \land q \land \ldots \land t$

- assume that **u** holds if p and q and ... and t all hold
- assume that **u** holds
- show that p and q and ... and t all hold

Resolution

Resolution is **a rule of inference**
leading to a refutation theorem-proving technique
for sentences in propositional logic and first-order logic.

By iteratively applying the resolution rule, it is possible

- to tell whether a propositional formula is **satisfiable**
- to prove that a first-order formula is unsatisfiable;

- this method may prove the **satisfiability** of a first-order formula,
- but not always, as it is the case for all methods for first-order logic

Resolvent : the clause produced by a resolution rule

A simple example

$a \lor b$, $\neg a \lor c$

$b \lor c$

Suppose a is false. In order for the premise **$a \lor b$** to be true, **b** must be true.
Suppose a is true. In order for the premise **$\neg a \lor c$** to be true, **c** must be true.
Therefore regardless of falsehood or veracity of **a**, if both premises hold,
then the conclusion **$b \lor c$** is true.

Refutation

Refute
To prove (something) to be false or incorrect.
To deny the truth or correctness of (something).

Reductio ad absurdum
Latin: "**reduction to absurdity**"; pl.: reductiones ad absurdum

also known as
argumentum ad absurdum
Latin: **argument to absurdity**

is a common form of argument which seeks to demonstrate that a statement is true
by showing that a false, untenable, or absurd result follows from its denial,
or in turn to demonstrate that a statement is false
by showing that a false, untenable, or absurd result follows from its acceptance.

SLD Resolution

The SLD (Selective Linear Definite clause) resolution
SLD stands for “Linear resolution” with a “Selection function” for “Definite clauses”

“definite clauses” are just another name for Prolog clauses.

"L" stands for the fact that a resolution proof can be restricted to a linear sequence of clauses:

$$C_1, C_2, \ldots \rightarrow C_i, C_{i+1}, \ldots, C_l$$

where the "top clause" C_1, is **an input clause**,
and every other clause C_{i+1}, is a **resolvent**
one of whose parents is the previous clause C_i
The proof is a **refutation**
if the last clause C_l , is the **empty clause**.

In **SLD**, **all of the clauses** in the sequence are **goal clauses**, and **the other parent** is **an input clause in the given set of definite clauses S**.

In **SL** resolution, **the other parent** is either **an input clause** or **an ancestor clause** earlier in the sequence.

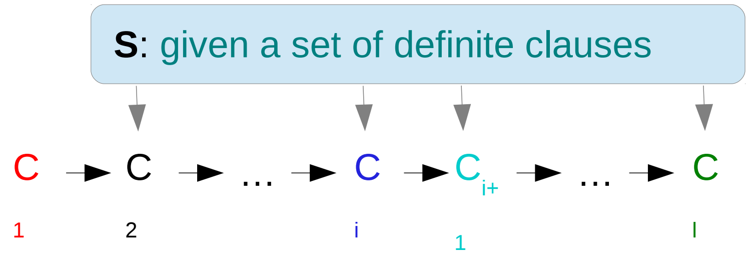
In both **SL** and **SLD**, "S" stands for the fact that the only literal resolved upon in any clause C_i, is one that is uniquely selected by a **selection rule** or **selection function**.

In **SL** resolution, the selected literal is restricted to one which has been most recently introduced into the clause. In the simplest case, such a last-in-first-out selection function can be specified by the order in which literals are written, as in Prolog.

However, the selection function in **SLD** resolution is more general than in *SL resolution* and in *Prolog*. There is *no restriction* on the literal that can be selected.

SLD vs SL Resolution (1)

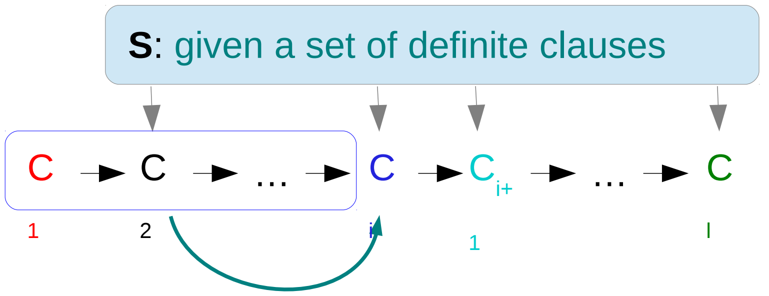
	Disjunction form	Implication form
Definite clause	¬p ∨ ¬q ∨ ... ∨ ¬t ∨ u	u ← p ∧ q ∧ ... ∧ t
Fact	u	u
Goal clause	¬p ∨ ¬q ∨ ... ∨ ¬t	false ← p ∧ q ∧ ... ∧ t



$$\begin{tikzcd} C_1 \arrow[r] & C_2 \arrow[r] & \dots \arrow[r] & C_i \arrow[r] & C_{i+1} \arrow[r] & \dots \arrow[r] & C_l \end{tikzcd}$$

In **SLD**, **all of the clauses** in the sequence are **goal clauses**, and **the other parent** is **an input clause in the given set of definite clauses S**.

In **SL** resolution, **the other parent** is either **an input clause** or **an ancestor clause** earlier in the sequence.

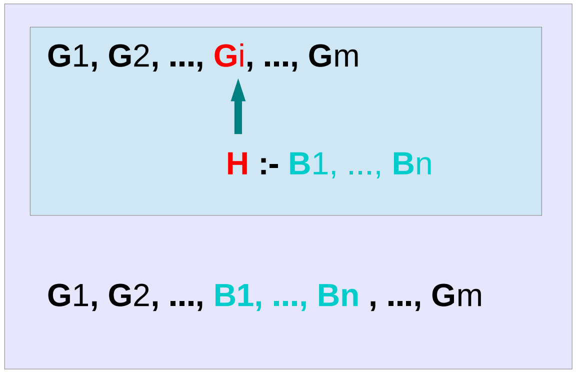


$$\begin{tikzcd} \boxed{C_1 \arrow[r] & C_2 \arrow[r] & \dots} \arrow[r] & C_i \arrow[r] & C_{i+1} \arrow[r] & \dots \arrow[r] & C_l \end{tikzcd}$$

http://www.doc.ic.ac.uk/~rak/papers/History.pdf

SLD vs SL Resolution (2)

	Disjunction form	Implication form
Definite clause	¬p ∨ ¬q ∨ ... ∨ ¬t ∨ u	u ← p ∧ q ∧ ... ∧ t
Fact	u	u
Goal clause	¬p ∨ ¬q ∨ ... ∨ ¬t	false ← p ∧ q ∧ ... ∧ t



selection

In both **SL** and **SLD**, "S" stands for the fact that the only literal resolved upon in any clause C_i, is one that is uniquely selected by a **selection rule** or **selection function**.

In **SL** resolution, **the selected literal** is restricted to one which has been most recently introduced into the clause. In the simplest case, such a last-in-first-out selection function can be specified by the order in which literals are written, as in Prolog.

However, **the selection function** in **SLD** resolution is more general than in *SL resolution* and in *Prolog*. There is *no restriction* on the literal that can be selected.

http://www.doc.ic.ac.uk/~rak/papers/History.pd
f

Satisfiability

A formula is **satisfiable**
if it is possible to find **an** interpretation (model)
that makes the formula **true**.

A formula is **valid**
if **all** interpretations make the formula **true**.

A formula is **unsatisfiable**
if **none** of the interpretations make the formula
true

A formula is **invalid**
if **some** such interpretation makes the formula
false.

φ is **valid if and only if ¬φ** is **unsatisfiable**
it is not true that ¬φ is satisfiable.

φ is **satisfiable if and only if ¬φ** is **invalid**.

satisfiability is decidable
for **propositional formulae**.
satisfiability is an **NP-complete** problem

Satisfiability is **undecidable** and indeed it isn't
even a semidecidable property of formulae in
first-order logic (FOL).

This fact has to do with the **undecidability** of the
validity problem for **FOL**.

Logic Programming

$(p \land q \land \ldots \land t) \rightarrow u$

to show u, show p and show q and ... and show t.

$u \leftarrow (p \land q \land \ldots \land t)$ **u :- p, q, ..., t.**

$\exists X\ (p \land q \land \ldots \land t)$

$\forall X\ (\text{false} \leftarrow p \land q \land \ldots \land t)$ **:- p, q, ..., t.**

the resolution of a goal clause with a definite clause to produce a new goal clause
is the basis of the SLD resolution inference rule
a definite clause behaves as a goal-reduction procedure

the negation of a problem to be solved as a goal clause.
the problem of solving the existentially quantified conjunction of positive literals
is represented by negating the problem (denying that it has a solution)

Solving the problem amounts to deriving a contradiction,
which is represented by the empty clause (or "false").
The solution of the problem is a substitution of terms for the variables in the goal clause,
which can be extracted from the proof of contradiction.

The Prolog notation is actually ambiguous, and the term “goal clause” is sometimes also used ambiguously.
The variables in a goal clause can be read as universally or existentially quantified,
and deriving “false” can be interpreted either as deriving a contradiction
or as deriving a successful solution of the problem to be solved.

Resolution Algorithm

Resolvent : a conjunction of current goals to prove (initially Q)

The resolution algorithm

- **selects** a goal from the **resolvent**
- **searches** a clause in the database
- **replaces** the goal with the body of the clause.
  whose head **unifies** with the goal.

The resolution **loop** replaces successively goals of the **resolvent**
until they all reduce to true and the **resolvent** becomes empty.

a success with a possible instantiation of the query goal Q',
the final substitution is the composition of all the MGUs
involved in the resolution restricted to the variables of Q.

a failure if no rule unifies with the goal.

Refutation: This type of derivation, which terminates when the resolvent is empty

A Resolution Algorithm

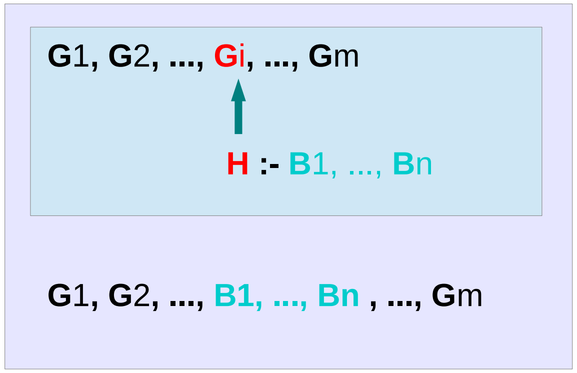
Most General Unifier

- **Initialization**

```
Initialize Resolvent to Q, the initial goal of the resolution algorithm.
Initialize the final substitution σ to {}
Initialize failure to false
```

- **Loop with Resolvent = G1, G2, ..., Gi, ..., Gm**

```
while (Resolvent ≠ ∅) {
    1. Select the goal Gi ∈ Resolvent;
    2. If Gi == true, delete it and continue;
    3. Select the rule H :- B1, ..., Bn in the database
        such that Gi and H unify with the MGU θ.
        If there is no such a rule then set failure to true; break;
    4. Replace Gi with B1, ..., Bn in Resolvent
        % Resolvent = G1,...,Gi−1, B1,...,Bn, Gi+1,..., Gm
    5. Apply θ to Resolvent and to Q;
    6. Compose σ with θ to obtain the new current σ;   %the final substitution
}
```



Lists

Each goal in the resolvent (in the body of a rule)
must be different from a variable.

Otherwise, this goal must be instantiated
to a nonvariable term before it is called.

The call/1 built-in predicate then executes it as in the rule:

```
daughter(X, Y) :- mother(Y, X), G = female(X), call(G).
```

where call(G) solves the goal G just as if it were female(X).

In fact, Prolog automatically inserts call/1 predicates
when it finds that a goal is a variable.
G is thus exactly equivalent to call(G),
and the rule can be rewritten more concisely in:

```
daughter(X, Y) :- mother(Y, X), G = female(X), G.
```

References

[1] en.wikipedia.org
[2] en.wiktionary.org
[3] U. Endriss, “Lecture Notes : Introduction to Prolog Programming”
[4] http://www.learnprolognow.org/ Learn Prolog Now!
[5] http://www.csupomona.edu/~jrfisher/www/prolog_tutorial
[6] www.cse.unsw.edu.au/~billw/cs9414/notes/prolog/intro.html
[7] www.cse.unsw.edu.au/~billw/dictionaries/prolog/negation.html
[8] http://ilppp.cs.lth.se/, P. Nugues, An Intro to Lang Processing with Perl and Prolog