

Resolution  
Theorem Proving

Robert Keller  
February 2011

### What is this?

- Resolution is a special kind of theorem proving used in:
  - Automated theorem proving and reasoning, where the goal is complete automation
  - Databases and Answer Extraction (AI)
  - Prolog language (a subset of general resolution)
- Resolution forms a complete proof system for refutation.

### History

- Resolution was introduced by Prof. J. Alan Robinson in 1965 at Syracuse U.
- There were previous hints at it by D. Prawitz (1960, for the function-free case) and Herbrand (1930's).

### How it works

- A logic formula is first negated, then converted into "clausal form". (Some heavy logic is wired into this step.)
- In clausal form, quantifiers have been eliminated.
- The clausal form is proved by **refutation**, i.e. showing that its negation is unsatisfiable.

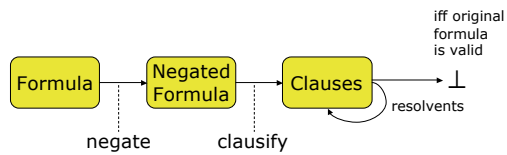
### Two Types of Resolution

- Predicate calculus resolution:
  - Our main objective
- Propositional resolution:
  - Needed to understand predicate resolution
  - Big role in algorithms and complexity theory (NP completeness, for example)

### Propositional Form of Resolution

- A **literal** is a proposition symbol or its negation.
- A **clause** is a disjunction of literals.
- The **negation** of the formula to be proved is first converted to a **clause set**, effectively a **conjunction** of those clauses.
- The original formula is a theorem iff the set of clauses is **not satisfiable**.

## Resolution Schematic



## Example

- Clause set:
  - $p \vee \neg q$
  - $\neg q \vee \neg r$
  - $q$
- This clause set is satisfiable:
  - Valuation  $p = T, q = T, r = F$  will satisfy it.

## Example

- Clause set:
  - $p \vee \neg q$
  - $q \vee r$
  - $\neg p$
  - $\neg r$
- This clause set is unsatisfiable:
  - There is **no valuation** that makes all formulas T at the same time.
  - Why not?  
We'd need  $p = r = F$  in order to satisfy all clauses, but then there is no way to set  $q$  so that all clauses are satisfied.

## Conjunctive Normal Form

- A clause set is another way of representing a propositional formula.
- A formula is in conjunctive normal form (CNF) iff it is a conjunction of disjunctions of literals (an "and" of "ors" of propositions and their negations"

## Sets

- Throughout this discussion, we want each clause to be a mathematical **set**, meaning that duplicates literals are eliminated.
- Likewise, a set of clauses (i.e. CNF) has no clause duplicated.

## Example

- Clause set:
  - $p \vee \neg q$
  - $q \vee r$
  - $\neg p$
  - $\neg r$
- is representable as
 
$$\{\{p, \neg q\}, \{q, r\}, \{\neg p\}, \{\neg r\}\}$$
 where the  $\vee$  connective is implicit within clauses, and the  $\wedge$  connective is implicit among them.

## Equivalence of Propositional Formulas

- Two propositional formulas are **equivalent** ( $\equiv$ ) iff they are satisfied by the same valuations.
- Examples:
  - $p \wedge \neg q \equiv \neg(q \vee \neg p)$
  - $p \rightarrow (q \rightarrow r) \equiv (p \wedge q) \rightarrow r$

## Equivalence of Clause Sets

- Two clause sets are called **equivalent** if they are satisfied by the same set of valuations.
- In particular, if two clause sets are equivalent, they are either:
  - both satisfiable, or
  - both unsatisfiable

## How General is the Clausal Form?

- Claim: Every propositional formula can be represented in clausal form.
- Examples:
  - $p \vee q$  in clausal form is  $\{p \vee q\}$  (one clause)
  - $p \wedge q$  in clausal form is  $\{p, q\}$  (two clauses)
  - $p \rightarrow q$  in clausal form is  $\{\neg p \vee q\}$  (one clause)
  - $p \leftrightarrow q$  in clausal form is  $\{\neg p \vee q, p \vee \neg q\}$  (two clauses)

## Extreme Cases

- $\emptyset$  or  $\{\}$ , the empty set of clauses, is equivalent to True  
(as T is the identity for the  $\wedge$  operation, and a set of clauses is a conjunction)
- Note: Every valuation satisfies every clause in  $\{\}$ , because there are no clauses to satisfy.
- $\{\perp\}$  or  $\{\emptyset\}$  or  $\{\{\}\}$ , the set containing the empty clause, is equivalent to False.

## The Empty Clause

- The **empty clause** is  $\perp$ , sometimes denoted by an empty box  $\square$ .
- Do not confuse the empty clause with an empty set of clauses.
- Observation: Any clause set containing the empty clause is unsatisfiable, because no valuation can make  $\perp$  true.

Example:  $\{\neg p \vee q, p, \perp\}$

## Conversion to Clausal Form

- We rely on rules that can be proved in propositional logic:
  - Commutative
    - $A \wedge B = B \wedge A$
    - $A \vee B = B \vee A$
  - Distributive
    - $A \wedge (B \vee C) = (A \wedge B) \vee (A \wedge C)$
    - $A \vee (B \wedge C) = (A \vee B) \wedge (A \vee C)$
  - DeMorgan
    - $\neg(A \wedge B) = \neg A \vee \neg B$
    - $\neg(A \vee B) = \neg A \wedge \neg B$

## Conversion to Clausal Form

1. Replace each  $\varphi \rightarrow \psi$  with  $(\neg\varphi \vee \psi)$ .
2. Replace each  $\varphi \leftrightarrow \psi$  with  $(\neg\varphi \vee \psi) \wedge (\varphi \vee \neg\psi)$ .
3. Push  $\neg$  inward, toward proposition symbols:
  - Replace  $\neg(\varphi \wedge \psi)$  with  $(\neg\varphi \vee \neg\psi)$ .
  - Replace  $\neg(\varphi \vee \psi)$  with  $(\neg\varphi \wedge \neg\psi)$ .
  - Replace  $\neg\neg\varphi$  with  $\varphi$ .
4. Push  $\vee$  inward toward literals:
  - Replace  $\chi \vee (\varphi \wedge \psi)$  with  $(\chi \vee \varphi) \wedge (\chi \vee \psi)$ .
  - Replace  $(\varphi \wedge \psi) \vee \chi$  with  $(\varphi \vee \chi) \wedge (\psi \vee \chi)$ .

## Example of Conversion to Clauses

- $\neg(p \rightarrow (\neg q \wedge (r \wedge \neg s)))$  replace  $\rightarrow$
- $\neg(\neg p \vee (\neg q \wedge (r \wedge \neg s)))$  push  $\neg$  inward
- $\neg\neg p \wedge \neg(\neg q \wedge (r \wedge \neg s))$  delete  $\neg\neg$
- $p \wedge (\neg\neg q \wedge (r \wedge \neg s))$  push  $\neg$  inward
- $p \wedge (\neg\neg q \vee \neg(r \wedge \neg s))$  delete  $\neg\neg$
- $p \wedge (q \vee \neg(r \wedge \neg s))$  push  $\neg$  inward
- $p \wedge (q \vee \neg r \vee \neg\neg s)$  delete  $\neg\neg$
- $p \wedge (q \vee \neg r \vee s)$  conjuncts are clauses
- $\{p, q \vee \neg r \vee s\}$

## Try this one

- $(\neg p \wedge (\neg q \rightarrow (r \leftrightarrow s)))$

## Clausal Form from a Truth Table

- A truth table is typically thought of as displaying one of possibly many **disjunctive normal forms** (DNF).
- Each row is a **min-term**:
  - $p_1^* \wedge p_2^* \wedge \dots \wedge p_n^*$  where  $p_i^*$  indicates a **literal**.
- The overall truth-function is a **disjunction** of min-terms:
  - $\mathbf{V}(p_1^* \wedge p_2^* \wedge \dots \wedge p_n^*)$  over the rows for which the result is **T**.

## Example: Truth Table

	p	q	r	value
	F	F	F	F
	F	F	T	F
$(\neg p \wedge q \wedge \neg r)$	F	T	F	T
$(\neg p \wedge q \wedge r)$	F	T	T	T
$(p \wedge \neg q \wedge \neg r)$	T	F	F	T
	T	F	T	F
$(p \wedge q \wedge \neg r)$	T	T	F	T
$(p \wedge q \wedge r)$	T	T	T	T

A DNF:  $(\neg p \wedge q \wedge \neg r) \vee (\neg p \wedge q \wedge r) \vee (p \wedge \neg q \wedge \neg r) \vee (p \wedge q \wedge \neg r) \vee (p \wedge q \wedge r)$

## Clausal Form from Truth Table

- A DNF has the value **True** exactly when at least one of the disjuncts does, e.g.
  - $(\neg p \wedge q \wedge \neg r) \vee (\neg p \wedge q \wedge r) \vee \dots$
- Similarly a CNF has the value **False** exactly when at least one of the conjuncts does, e.g.
  - $(p \vee q \vee r) \wedge (p \vee q \vee \neg r) \wedge (\neg p \vee q \vee \neg r)$
- Hence we can "read off" a CNF from the False rows of the table.

### Example: Clausal Form from Truth Table

p	q	r	value
F	F	F	F
F	F	T	F
F	T	F	T
F	T	T	T
T	F	F	T
T	F	T	F
T	T	F	T
T	T	T	T

A CNF:  $(p \vee q \vee r) \wedge (p \vee q \vee \neg r) \wedge (\neg p \vee q \vee \neg r)$

### Housekeeping: Reduced Clause Sets

- A clause set is **reduced** provided:
  - No literal occurs multiple times in any clause.
    - $p \vee \neg q \vee p$  is disallowed in a reduced set.
  - No clause contains a literal and its negation.
    - $p \vee q \vee \neg p$  is disallowed in a reduced set.
- Any clause set S is equivalent to a reduced set  $\text{reduce}(S)$ :
  - Replace multiple occurrences of a literal with a **single occurrence** of the literal.
  - Drop** any clauses containing a **literal and its negation**. (Such clauses are equivalent to T, and do thus do not affect satisfiability of the set of clauses.)
  - Replace multiple occurrences of a clause (as a **set**) with a single occurrence.

### reduce example

$$\text{reduce}(\{p \vee \neg q \vee p, p \vee q \vee \neg p \vee q\}) = \{p \vee \neg q\}$$

### Resolution Method

- Input: A reduced set of clauses.
- Output: A set of clauses equivalent to the input set, such that the original set is unsatisfiable iff the final set contains the **empty clause**  $\perp$ .
- There is no valuation that satisfies  $\perp$ , much less  $\perp$  together with other clauses.

### How Resolution Works

- Do Repeatedly, until no further steps can be taken:
  - From the set of clauses, pick a pair from which a **new** clause, called the "resolvent", can be created. (Must resolve the pair to find this out.)
  - Add the resolvent to the set.
  - If  $\perp$  is ever added to the set, stop. The original set of clauses is unsatisfiable.
- Conversely, if the original set of clauses is unsatisfiable, it is possible to eventually derive  $\perp$ .

### What is the Resolvent?

- Suppose p is a proposition symbol.
- If the set contains clauses of both forms
  - $p \vee \phi$
  - $\neg p \vee \psi$
- where  $\phi$  and  $\psi$  are formulas (either could be empty), then the resolvent is the reduced version of
 
$$\phi \vee \psi.$$
- p and  $\neg p$  are said to be "clashing" literals.

## Resolution as a Deduction Rule

$$\frac{p \vee \varphi \quad \neg p \vee \psi}{\varphi \vee \psi} \quad (\text{in reduced form})$$

where  $p$  is any proposition symbol and  $\varphi$  and  $\psi$  are clauses (possibly empty).

## Example of Resolvents

- Consider the clauses
  - $p \vee \neg q \vee \neg s$
  - $q \vee r \vee \neg s$
- A resolvent (based on literal  $q$ ) is:
  - $p \vee r \vee \neg s$

## Example of Resolvents

- Consider the clauses
  - $p \vee r$
  - $\neg r$
- The resolvent is:
  - $p$

## Example of Resolvents

- Consider the clauses
  - $p$
  - $\neg p$
- Since  $p$  and  $\neg p$  occur in different clauses, the resolvent is:
  - $\perp$

## Example of Resolvents

- Consider the clauses
  - $p \vee \neg q \vee r$
  - $q \vee \neg r \vee \neg s$
- One resolvent (based on literal  $q$ ) is:
  - $p \vee r \vee \neg r \vee \neg s$
- Another (based on literal  $r$ ) is:
  - $p \vee q \vee \neg q \vee \neg s$
- Both of these would be **dropped** in reducing, however, since each contains a literal and its negation.

## Resolution Algorithm

- Start with a set  $S$  of reduced clauses.
- while  $S$  does not contain  $\perp$  and the following step adds something new to  $S$ :
  - Add to  $S$  the resolvent  $R$  of any two clauses such that  $R$  is not already in  $S$ .
- The original  $S$  is unsatisfiable iff  $\perp$  is in  $S$ .

## Unit Clauses

- A clause with exactly one literal is called a unit clause.
- The penultimate step prior to resolving to  $\perp$  will be to resolve two unit clauses.
- Resolving a unit clause with a clause having  $n > 0$  literals results in a clause with fewer than  $n$  literals.

## Note on choice of clauses

- Preferring unit clauses is a good heuristic.

## Example 1 (Highlighting unit clauses)

- $S = \{p \vee \neg q, q \vee r, \neg p, \neg r\}$   
resolve  $p \vee \neg q$  with  $\neg p$ , adding  $\neg q$  to  $S$ .
- $S = \{p \vee \neg q, q \vee r, \neg p, \neg r, \neg q\}$   
resolve  $q \vee r$  with  $\neg q$ , adding  $r$  to  $S$ .
- $S = \{p \vee \neg q, q \vee r, \neg p, \neg r, \neg q, r\}$   
resolve  $\neg r \vee r$  adding  $\perp$  to  $S$ .
- Stop  $\perp \in S$ .
- The original  $S$  is unsatisfiable, as  $\perp \in S$ .

## Example 2

- $S = \{p \vee \neg q, q \vee r, \neg p\}$   
Resolve  $p \vee \neg q$  with  $\neg p$ , adding  $\neg q$  to  $S$ .
- $S = \{p \vee \neg q, q \vee r, \neg p, \neg q\}$   
Resolve  $q \vee r$  with  $\neg q$ , adding  $r$  to  $S$ .
- $S = \{p \vee \neg q, q \vee r, \neg p, \neg q, r\}$   
Resolve  $p \vee \neg q$  with  $q \vee r$ , adding  $p \vee r$  to  $S$ .
- $S = \{p \vee \neg q, q \vee r, \neg p, \neg q, r, p \vee r\}$   
Stop. No new resolvents are possible. The original set is satisfiable, as  $\perp \notin S$ .

**Soundness:** A valuation satisfying both  $p \vee \varphi$  and  $\neg p \vee \psi$  satisfies  $\varphi \vee \psi$ .

- Suppose  $v$  satisfies **both**  $p \vee \varphi$  and  $\neg p \vee \psi$ .
- Either  $v(p) = T$  or  $v(p) = F$ .
- If  $v(p) = T$ , then  $v(\neg p) = F$ . In order to satisfy  $\neg p \vee \psi$  then,  $v(\psi) = T$ . Thus  $v(\varphi \vee \psi) = T$ .
- If  $v(p) = F$ , in order to satisfy  $p \vee \varphi$ ,  $v(\varphi) = T$ . Thus  $v(\varphi \vee \psi) = T$ .
- Thus adding the resolvent preserves the valuations that satisfy the set of clauses.

## Completeness

- Completeness is more complicated and we will not prove it here.
- We'd have to show that if a set is unsatisfiable, there is a set of resolution steps that result in the empty clause.
- For a constructive proof, see: J. Gallier, 2006: <http://www.cis.upenn.edu/~cis510/tcl/resol.pdf>

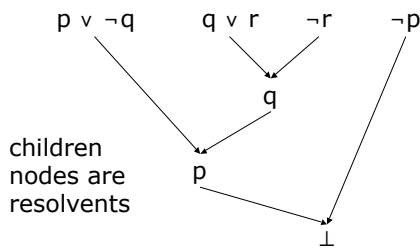
## Resolution Algorithm Termination

- Closure is always achievable.
- The set of distinct reduced clause sets for a given set of proposition symbols is **finite**.
- At worst, every possible clause (regarding reordering of symbols as equivalent) will be generated.
- How many distinct clauses can there be for  $n$  proposition symbols?

## Resolution in tabular form

1.	$p \vee \neg q$	Premise
2.	$q \vee r$	Premise
3.	$\neg r$	Premise
4.	$\neg p$	Premise
5.	$q$	Resolution 2, 3
6.	$p$	Resolution 1, 5
7.	$\perp$	Resolution 6, 4

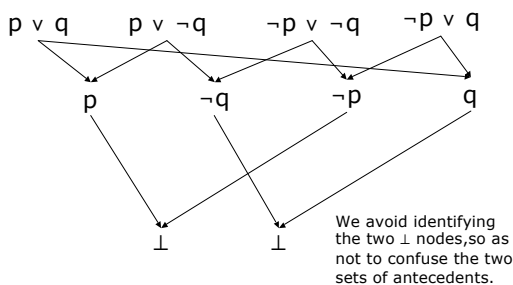
## Resolution as a Tree



## Try resolving these clause sets:

- $\neg p \vee \neg q \vee \neg r,$   
 $\neg q \vee r,$   
 $q \vee s,$   
 $\neg s,$   
 $p$
- $p \vee \neg q \vee r,$   
 $q \vee r,$   
 $\neg p$

## Sometimes a DAG is more appropriate than a tree for showing all options



## Useful Editing Short-cuts

- Uncomplemented Literal Lemma

If a **literal** appears in one or more clauses, but its complement appears in no clause, then every clause containing that literal can be deleted from the set without changing satisfiability.

- **Rationale:** The literal in question can be assigned  $T$  without loss of generality, thus clauses containing it cannot affect satisfiability.

### Example of Uncomplemented Literal Lemma

- $\neg p \vee q \vee r,$   
 $\neg q \vee r,$   
 $q \vee s,$   
 $\neg s,$   
 $p$
- $r$  occurs only uncomplemented.
- The clause set is unsatisfiable iff the following set is:
- $q \vee s,$   
 $\neg s,$   
 $p$
- and this set is unsatisfiable iff  $\neg s$  is unsatisfiable (which it isn't).

### Further Editing Short-Cuts

#### Unit Clause Lemma:

If a **unit** clause (clause with only one literal  $L$ ) exists within the set, the following operation may be performed without affecting satisfiability:

- Remove **all** clauses containing  $L$ .
- Remove the complement of  $L$  from all remaining clauses.
- **Rationale:** The literal in question **must** be assigned  $T$  in a satisfying interpretation. Hence all clauses containing it will be  $T$  and contribute nothing to the set. Likewise, its complement must be assigned  $F$ , and contribute nothing to the individual clauses.

### Example of Unit Clause Lemma

- $\neg p \vee q \vee r,$   
 $\neg q \vee r,$   
 $q \vee s,$   
 $\neg s,$   
 $p$
- $\neg s$  is a unit clause. The complement of  $\neg s$  is  $s$ .
- The clause set is unsatisfiable iff the following set is:
- $\neg p \vee q \vee r,$   
 $\neg q \vee r,$   
 $q,$   
 $p$   
 etc.

### Further Useful Optimizations

#### Subsumption Lemma:

- One clause **subsumes** another if the former's literals are a **subset** of the latter's.
- If one clause of a set subsumes another, the subsumed clause (the larger one) can be dropped from the set.
- **Rationale:** If  $C$  subsumes  $D$ , then any interpretation satisfying  $C$  must also satisfy  $D$  (because the literals are disjoined). Thus the satisfiability of the set of clauses is unaffected if  $D$  is removed.

### Example of Subsumption Lemma

- $\neg p \vee q \vee r,$   
 $\neg p \vee r,$   
 $p \vee r \vee q$
- The second clause subsumes the first.
- The clause set is unsatisfiable iff the following set is:
- $\neg p \vee r,$   
 $p \vee r \vee q$

### Common Special Case of Clause Set

- Often we want to prove a **sequent** such as:
  - $\varphi_{11} \wedge \varphi_{12} \wedge \dots \wedge \varphi_{1m1} \rightarrow \psi_1,$
  - $\varphi_{21} \wedge \varphi_{22} \wedge \dots \wedge \varphi_{2m2} \rightarrow \psi_2,$
  - $\dots$
  - $\varphi_{n1} \wedge \varphi_{n2} \wedge \dots \wedge \varphi_{nmn} \rightarrow \psi_n$       $\vdash \neg \chi_1 \wedge \chi_2 \wedge \dots \wedge \chi_p$
 where each symbol represents a literal.     "axioms"     "theorem"
- This can be done by showing that the following clause set is **unsatisfiable**:

$$\{ \neg \varphi_{11} \vee \neg \varphi_{12} \vee \dots \vee \neg \varphi_{1m1} \vee \psi_1, \\ \neg \varphi_{21} \vee \neg \varphi_{22} \vee \dots \vee \neg \varphi_{2m2} \vee \psi_2, \\ \dots, \\ \neg \varphi_{n1} \vee \neg \varphi_{n2} \vee \dots \vee \neg \varphi_{nmn} \vee \psi_n, \\ \neg \chi_1 \vee \neg \chi_2 \vee \dots \vee \neg \chi_p \}$$

## Strategic Optimizations

- **Unit-Preference:** Prefer resolving with unit clauses. These reduce the size of resulting clauses.
- **Set-of-Support:** Divide the clauses into two sets:
  - A known-satisfiable subset.
  - Other
- Always resolve with an "other" or a clause derived from one.
- The latter are called the "set of support" (SOS).

## Set-of-Support

- Showing that the following clause set is **unsatisfiable**:

$$\begin{aligned} & \{ \neg\varphi_{11} \vee \neg\varphi_{12} \vee \dots \vee \neg\varphi_{1m_1} \vee \psi_1, \\ & \neg\varphi_{21} \vee \neg\varphi_{22} \vee \dots \vee \neg\varphi_{2m_2} \vee \psi_2, \\ & \dots \\ & \neg\varphi_{n1} \vee \neg\varphi_{n2} \vee \dots \vee \neg\varphi_{nm_n} \vee \psi_n \} \end{aligned} \quad \text{Satisfiable "axioms"}$$

$$\neg\chi_1 \vee \neg\chi_2 \vee \dots \vee \neg\chi_p \} \quad \text{Initial set of support}$$

## Horn Clauses

- A Horn clause is one in which there is **at most one non-negated literal**:
  - $\neg\varphi_1 \vee \neg\varphi_2 \vee \dots \vee \neg\varphi_m \vee \psi$  (**one** non-negated)
  - or
  - $\neg\varphi_1 \vee \neg\varphi_2 \vee \dots \vee \neg\varphi_m$  (**no** non-negated)
- Horn clause are the basis of the **Prolog** language, where:
  - $\neg\varphi_1 \vee \neg\varphi_2 \vee \dots \vee \neg\varphi_m \vee \psi$  is written
  - $\psi :- \varphi_1, \varphi_2, \dots, \varphi_m$ .
  - interpreted as
  - $\varphi_1 \wedge \varphi_2 \wedge \dots \wedge \varphi_m \rightarrow \psi$
  - If  $m = 0$ , then we just write
  - $\psi$ .

Prolog uses a special form of resolution to do its work ("SLD" = Selective Linear Definite resolution)

- $\{ p \vee \neg r \vee \neg s, \\ r \vee \neg q, \\ s \vee \neg q, \\ q, \\ \neg p, \\ \} \}$  **Non-negated literals in red.**
- becomes in Prolog syntax:
  - $p :- r, s.$
  - $r :- q.$
  - $s :- q.$
  - $q.$
  - $?- p.$

Dialog with Prolog:

```
consult(user).
p :- r, s.
r :- q.
s :- q.
q.
^D
| ?- p.
yes
```

## Resolution Theorem Provers

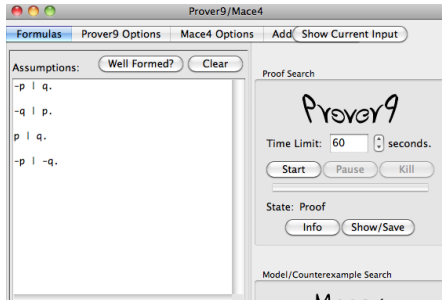
- Prolog cannot handle general negation
- Resolution theorem provers can
- Examples: Prover9, Vampire, ...

Prover9



- Extends the former program "Otter"
- Developed at Argonne National Laboratory
- Free download for all platforms
- <http://www.cs.unm.edu/~mccune/prover9/>
- Also includes Mace for constructing counterexamples

## Prover9 GUI (- is "not", | is "or")



## Prover9 Proof (\$F is empty clause)

```

----- PROOF -----
% ----- Comments from original proof -----
% Proof 1 at 0.00 (+ 0.00) seconds.
% Length of proof is 7.
% Level of proof is 3.
% Maximum clause weight is 2.
% Given clauses 3.
1 -p | q. [assumption].
2 -q | p. [assumption].
3 p | q. [assumption].
4 -p | -q. [assumption].
5 p. [resolve(3,b,2,a),merge(b)].
6 -q. [back_unit_del(4),unit_del(a,5)].
7 $F. [back_unit_del(1),unit_del(a,5),unit_del(b,6)].
----- end of proof -----

```

## Resolution for Predicate Logic

- **Predicate Clausal Form:**
  - A **literal** is an atomic formula or its negation (instead of a proposition symbol or its negation).
  - **The variables of each clause are implicitly  $\forall$ -quantified.**
  - The **variables** of each clause are thus **independent** from the other clauses; even if they are the same, they should be thought of as being **different** (e.g. implicitly rename by indexing with a clause number).

## Example: Predicate Clausal Form

- Clause set  $\{p(X), q(X, Y), \neg q(X, X) \vee p(X)\}$  stands for the conjunction
  - $\forall X p(X)$
  - $\wedge \forall X \forall Y q(X, Y)$
  - $\wedge \forall X \forall Y (\neg q(X, X) \vee p(X))$
 which is the same as
  - $\forall X_1 p(X_1)$
  - $\wedge \forall X_2 \forall Y_2 q(X_2, Y_2)$
  - $\wedge \forall X_3 \forall Y_3 (\neg q(X_3, X_3) \vee p(X_3))$
 i.e. the clause set
  - $\{p(X_1), q(X_2, Y_2), \neg q(X_3, X_3) \vee p(X_3)\}$

## How General is This?

- We will see later that it is completely general, as far as showing unsatisfiability is concerned.

## Examples of Predicate Clausal Form

- $\neg \text{man}(X) \vee \text{mortal}(X)$
- $\text{man}(\text{socrates})$
- $\neg \text{mortal}(\text{socrates})$
- These clauses can be used to prove the syllogism:
  - All men are mortal.
  - Socrates is a man.
  - Therefore Socrates is mortal.

## Resolution for Predicate Clauses

- To resolve predicate clauses, it is no longer sufficient to look for just a literal and its negation in two distinct clauses, e.g.  $X$  in

$$\begin{aligned} & \neg q(X, X) \vee p(X) \\ & \neg p(Z) \vee r(Z, Y) \end{aligned}$$

- For one thing, the identity of the **variables** is **independent** in each.
- For another, the arguments are generally **terms**, not just simple variables:  
 $\neg q(X, X) \vee p(f(X))$   
 $\neg p(X) \vee r(g(X), c)$

## Example of What Resolution Must Do

- Suppose we have derived three formulas (where  $c$  is a constant symbol):
  - $p(c)$
  - $\forall X (p(X) \rightarrow q(f(X)))$
  - $\forall X (q(X) \rightarrow r(X, g(X)))$
- We would expect to be able to infer
  - $q(f(c))$
  - $r(f(c), g(f(c)))$
- Resolution must be able to handle such things.

## Equivalent Clausal Form

- The clausal form of
  - $p(c)$
  - $\forall X (p(X) \rightarrow q(f(X)))$
  - $\forall X (q(X) \rightarrow r(X, g(X)))$is
  - $\{p(c), \neg p(X) \vee q(f(X)), \neg q(X) \vee r(X, g(X))\}$
- Resolution has to "make a connection" between  $p(c)$  and  $p(X)$ , and between  $q(f(X))$  and  $q(X)$ .

## Unification

- The "connection" alluded to on the previous slide is known as **unification**.
- Two **atoms** are **unifiable** if there is a uniform **set of substitutions** of terms for their variables that makes them **identical**.
- If such a substitution set exists, it is **applied to all** literals in the formulas prior to resolution.

## Unification Examples

- Consider atoms  $p(c)$ ,  $p(X)$  ( $c$  is a constant,  $X$  a variable).
- These terms are **unifiable**, since the substitution  $[c/X]$  (substitute  $c$  for  $X$ ) makes them identical.

## Unification Examples

- Consider  $q(c, d)$ ,  $q(X, X)$  ( $c$  and  $d$  are constants,  $X$  a variable).
- These terms are **not unifiable**.
- Distinct **constant symbols do not unify**. There is no substitution that will make them identical.
- (Note: This is not the same as saying constant symbols cannot be equated. They can, with a separate equation such as  $c = d$ . Equality is handled separately.)

## Renaming Apart

- Consider  $p(X, f(a))$  vs.  $p(g(Y), f(X))$
- These might appear not to unify, since we would have a conflict  $[g(Y)/X]$  vs.  $[a/X]$ .
- However, if we **rename** the variables in the second clause we get:  
 $p(X, f(a))$  vs.  $p(g(Z), f(W))$ .
- These unify, using  $[g(Z)/X, a/W]$ .
- Note:** Renaming apart is done only at the **start** of considering unification of two clauses, and all variables in the clause are renamed **uniformly**.

## Notation for Variable Substitutions

- In general, a **substitution** consists of a set of **bindings of variables** to terms, e.g.  
 $\beta = [Z/X, f(Z, c)/Y, c/W]$
- If  $\tau$  is a **term**, then  $\tau\beta$  denotes the result of making the substitutions  $\beta$  in for variables in  $\tau$ , e.g.  
 if  $\tau = p(X, g(Y, W))$   
 then  $\tau\beta = p(Z, g(f(Z, c), c))$

## Composing Variable Substitutions

- If  $\beta$  and  $\gamma$  are substitutions and  $\tau$  is a term, then  $(\tau\beta)\gamma$  denotes the result of first applying  $\beta$  to  $\tau$ , then  $\gamma$  to the result, e.g.  
 $\tau = p(X, g(Y, W))$                       literal  
 $\beta = \{Z/X, f(Z, c)/Y, c/W\}$             sub  
 $\gamma = \{V/Z\}$                                       sub  
 $(\tau\beta)\gamma = p(V, g(f(V, c), c))$
- The **composition**  $\beta\gamma$  of substitutions  $\beta$  and  $\gamma$  is the substitution such that for **all** terms  $\tau$   
 $\tau(\beta\gamma) = (\tau\beta)\gamma$   
 e.g.  $\{V/X, f(V, c)/Y, c/W\}$             above

## Unifiers

- A set of substitutions that unifies two literals is called a **unifier**.

## More Unification Examples

Term 1	Term 2	Unifier, if any?
$p(X, X)$	$p(f(Y), f(Z))$	
$p(X, X)$	$p(f(Y), g(Y))$	
$p(X, Y)$	$p(Z, f(Z))$	
$p(X, f(X))$	$p(g(Y), W)$	
$p(X, f(X))$	$p(f(Y), Y)$	

## Even More Unification Examples

Term 1	Term 2	Unifier, if any?
$p(X, Y)$	$p(f(Z), g(Z))$	
$p(X, f(X))$	$p(f(Z), U)$	
$p(f(X), g(X))$	$p(f(U), U)$	
$p(f(X), f(X))$	$p(c, c)$	
$p(f(X), g(X))$	$p(Y, g(Y))$	

## Most General Unifiers (mgu)

- If two literals unify at all, they have a "most general unifier", one which adds no unneeded constraints.
- Example:  $p(X)$  vs.  $p(f(Y))$  could be unified with the substitution  $[f(c)/X, c/Y]$ .
- However, this would **not** be the most general, since we could leave  $Y$  as a variable:  $[f(Z)/X]$  and each of the original literals would unify with this.

## Generality of Substitutions

- Substitution  $\beta$  is **as general as** substitution  $\nu$  if there is a  $\gamma$  such that  $\nu = \beta \gamma$ .
- To say that  $\beta$  is a "most general unifier" means that it is as general as *any* unifier.

## Find the MGU or indicate non-unifiable

Term 1	Term 2	MGU?
$p(X, Y)$	$p(Z, Z)$	
$p(X, c)$	$p(Y, Y)$	
$p(f(X), Y)$	$p(W, f(Z))$	
$p(f(X), Y)$	$p(Z, Y)$	
$p(f(Z), g(X))$	$p(Y, g(Y))$	

## MGU Algorithm (Martelli & Montanari)

- Input:** Two terms, or two atoms,  $\tau_1, \tau_2$ , **already renamed apart**.
  - Output:** Either the most general unifier for  $\tau_1, \tau_2$ , or "not unifiable".
- ```

S := {[t1, t2]}; // functions as a sort-of stack
μ := the empty substitution;
while( S ≠ ∅ )
  remove a pair [L, R] from S; // pop case (1)
  if( L = R )
    do nothing;
  else if( L = f(s1, s2, ..., sn) and R = f(t1, t2, ..., tn) ) // same f, n (2)
    S := S ∪ {[s1, t1], [s2, t2], ..., [sn, tn]}; // pushes
  else if( L = x where x is a variable not occurring in R ) // (3)
    μ := μ[R/x];
    S := applytoallpairs([R/x], S); // composing
  else if( R = x where x is a variable not occurring in L ) // (4)
    μ := μ[L/x];
    S := applytoallpairs([L/x], S);
  else return "not unifiable"; // (5)
return μ as the MGU;

```

## Intuitive Unification

- Remember when two things **don't** unify:
  - Distinct constant symbols don't unify.
  - Terms with outermost function symbols that are distinct don't unify.
  - A term with an outermost function symbol doesn't unify with a constant.
  - Two terms with the same outermost function symbol don't unify if some of their arguments don't pairwise unify.
- Remember that substitutions are **cumulative** during unification.

## Example

- $p(X, f(X))$  vs.  $p(Y, f(Y))$  initial
- $S := \{[p(X, f(X)), p(Y, f(Y))]\}$
- $\mu := []$
- Remove  $[p(X, f(X)), p(Y, f(Y))]$  case 2
- $S := \{[X, Y], [f(X), f(Y)]\}$
- Remove  $[X, Y]$  case 3
- $\mu := [Y/X]; S := \{[f(Y), f(Y)]\}$
- Remove  $[f(Y), f(Y)]$  case 1
- $S := \{\}$
- Result: unifiable with mgu  $[Y/X]$

## Diagrammatically

- $$\begin{array}{c} p(X, f(X)) \\ | \uparrow \\ p(Y, f(Y)) \end{array} \quad \text{substitution } [Y/X]$$
- $$\begin{array}{c} p(Y, f(Y)) \\ | \quad | \quad | \\ p(Y, f(Y)) \end{array}$$

## Example

- $p(X, f(X))$  vs.  $p(f(Y), Y)$       initial  
 $S := \{[p(X, f(X)), p(f(Y), Y)]\}$   
 $\mu := \{\}$
- Remove  $[p(X, f(X)), p(f(Y), Y)]$       case 2  
 $S := \{[X, f(Y)], [f(X), Y]\}$
- Remove  $[X, f(Y)]$       case 3  
 $\mu := [f(Y)/X]; S := \{[f(f(Y)), Y]\}$
- Remove  $[f(f(Y)), Y]$       case 5  
**Result: not unifiable**

## Diagrammatically

- $$\begin{array}{c} p(X, f(X)) \\ | \updownarrow \\ p(f(Y), Y) \end{array} \quad \text{substitution } [f(Y)/X]$$
- $$\begin{array}{c} p(f(Y), f(f(Y))) \\ | \quad | \quad | \\ p(f(Y), Y) \end{array} \quad \text{occur check fails, not unifiable}$$

## Example

- $p(X, g(Z), X)$  vs.  $p(f(Y), Y, W)$       initial  
 $S := \{[p(X, g(Z), X), p(f(Y), Y, W)]\}$   
 $\mu := \{\}$
- Remove  $[p(X, g(Z), X), p(f(Y), Y, W)]$       case 2  
 $S := \{[X, f(Y)], [g(Z), Y], [X, W]\}$
- Remove  $[X, f(Y)]$       case 3  
 $\mu := [f(Y)/X]; S := \{[g(Z), Y], [f(Y), W]\}$
- Remove  $[g(Z), Y]$       case 4  
 $\mu := [f(g(Z))/X, g(Z)/Y]; S := \{[f(f(g(Z))), W]\}$
- Remove  $[f(f(g(Z))), W]$       case 4  
 $\mu := [f(g(Z))/X, g(Z)/Y, f(g(Z))/W]; S := \{\}$
- Result: unifiable with mgu  $[f(g(Z))/X, g(Z)/Y, f(g(Z))/W]$**

## Diagrammatically

- $$\begin{array}{c} p(X, g(Z), X) \text{ vs.} \\ | \uparrow \\ p(f(Y), Y, W) \end{array} \quad \text{substitution } [f(Y)/X]$$
- $$\begin{array}{c} p(f(Y), g(Z), f(Y)) \text{ vs.} \\ | \quad | \quad | \\ p(f(Y), Y, W) \end{array} \quad \text{substitution } [g(Z)/Y, f(g(Z))/X]$$
- $$\begin{array}{c} p(f(g(Z)), g(Z), f(g(Z))) \text{ vs.} \\ | \quad | \quad | \\ p(f(g(Z)), g(Z), W) \end{array} \quad \text{substitution } [f(g(Z))/W, g(Z)/Y, f(g(Z))/X]$$
- $$\begin{array}{c} p(f(g(Z)), g(Z), f(g(Z))) \text{ vs.} \\ | \quad | \quad | \\ p(f(g(Z)), g(Z), f(g(Z))) \end{array} \quad \text{substitution } [f(g(Z))/W, g(Z)/Y, f(g(Z))/X]$$

## Note on Unification in Prolog

- In Prolog, unification is used in goal matching and in the = (unify) operator.
- However, Prolog's unification is slightly **abridged**: it bypasses the "occur check":  
 $X = f(X)$   
*will* unify in Prolog, but not in ordinary unification. In effect, X gets bound to the infinite term:  
 $f(f(f(\dots)))$

## Checking Unifiability with Prolog

- As long as there are no occur-check violations, can use = to test.

```
$ /opt/local/bin/swipl
```

```
Welcome to SWI-Prolog
```

```
?- p(X, g(Z), X) = p(f(Y), Y, W).
```

```
X = f(g(Z)),
Y = g(Z),
W = f(g(Z))
```

## Try These

| $\tau_1$           | $\tau_2$              | mgu<br>(or not unifiable) |
|--------------------|-----------------------|---------------------------|
| $p(X, f(X), d)$    | $p(c, f(c), Y)$       |                           |
| $p(f(g(X)), g(Z))$ | $p(f(Y), Y)$          |                           |
| $p(f(g(X)), Z)$    | $p(g(Y), Y)$          |                           |
| $p(f(g(X)), X)$    | $p(f(g(h(Z))), h(Z))$ |                           |

## Resolving Predicate Calculus Clauses

- Resolvable clauses must contain literals with the **same predicate** symbol but of **opposite sign** (one negated, the other not).
- Pick two such literals, one from each clause.
- Rename the clauses apart.
- Determine whether the literals are unifiable, with mgu  $\mu$ . If they are, apply  $\mu$  to **all** literals in both clauses. If not, the clauses don't resolve on these particular literals.
- In the modified clauses, remove **all** instances of the modified literals used in unification, and form the disjunction of the remaining literals.

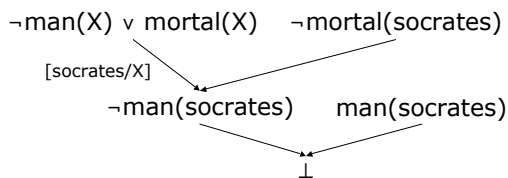
## Complete Predicate Resolution Process

- The process is similar the propositional case, except that we have to
  - rename variables, then
  - unify literals prior to resolution, and
  - apply the mgu to all literals in the two clauses, before obtaining the resolvent.
- There are also a special issue: "factoring", that needs to be considered.

## Example of Predicate Resolution

### • Clauses:

- $\neg\text{man}(X) \vee \text{mortal}(X)$
- $\text{man}(\text{socrates})$
- $\neg\text{mortal}(\text{socrates})$



## Example Resolving Predicate Clauses

- clause 1:  $p(X, g(Z), X) \vee q(X, h(Z))$
- clause 2:  $\neg p(f(Y), Y, W) \vee r(f(Y), g(W))$
- These are already renamed apart.

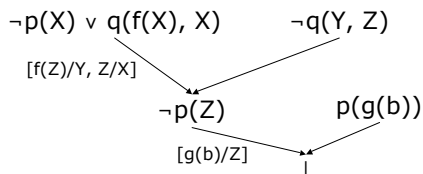
- The first literals of each unify with mgu  $[f(g(Z))/X, g(Z)/Y, f(g(Z))/W]$

- Apply the mgu to both clauses:
- clause 1':  $p(f(g(Z)), g(Z), f(g(Z))) \vee q(f(g(Z)), h(Z))$
- clause 2':  $\neg p(f(g(Z)), g(Z), f(g(Z))) \vee r(f(g(Z)), g(f(g(Z))))$
- Remove the instances of the unified atoms and form the disjunction.
- Resolvent:  $q(f(g(Z)), h(Z)) \vee r(f(g(Z)), g(f(g(Z))))$

## Example of Predicate Resolution

- Clauses:

- $\neg p(X) \vee q(f(X), X)$
- $p(g(b))$
- $\neg q(Y, Z)$



## Unit Preference

- As with propositional resolution, resolving with unit clauses first is a good heuristic.

## Try This Set

1.  $\neg e(X) \vee q(X) \vee s(X, f(X))$
2.  $\neg e(X) \vee q(X) \vee r(f(X))$
3.  $p(a)$
4.  $e(a)$
5.  $\neg s(a, Y) \vee p(Y)$
6.  $\neg p(X) \vee \neg q(X)$
7.  $\neg p(X) \vee \neg r(X)$

## Remember to treat clauses as sets.

- $q(b, X) \vee p(X) \vee q(b, a)$
- $\neg q(Y, a) \vee p(Y)$
- These are already renamed apart.
- unify  $q(b, X)$  with  $\neg q(Y, a)$
- mgu is  $[a/X, b/Y]$
- Modified clauses:
- $q(b, a) \vee p(a) \vee q(b, a)$
- $\neg q(b, a) \vee p(b)$
- There are **two** instances of  $q(b, a)$  in the first clause; both are removed in resolving.
- Resolvent:  $p(a) \vee p(b)$

## Binary Resolution and Factoring

- What we have seen so far is "binary" resolution — unifying two literals to achieve a resolvent.
- In general, binary resolution is not enough.
- We might need to "factor" two or more literals in the same clause to make progress.

## Factoring

- Two or more literals of the same sign in one clause can be unified (without renaming apart) so that the resulting literals can be collapsed into one.
- The resulting clause is called a **factor** of the original.
- The factor (with all variables quantified) is logically implied by the more-general original (with all variables quantified).

## Factoring Example

- Consider the clause:

$$P(x) \vee P(f(y)) \vee \neg Q(x)$$

- The first two literals can be unified using the substitution  $[f(y)/x]$ .

- The resulting factor is:

$$P(f(y)) \vee \neg Q(f(y))$$

- $(\forall x \forall y (P(x) \vee P(f(y)) \vee \neg Q(x)) \rightarrow \forall y (P(f(y)) \vee \neg Q(f(y))))$  is valid

## Use of Factoring

- Suppose our clause set includes:

$$P(x) \vee P(f(y)) \vee \neg Q(x) \\ \neg P(f(a))$$

- With binary resolution, we'd get the resolvent:  
 $P(x) \vee \neg Q(x)$ .

- If we **first factor**, to get  $P(f(y)) \vee \neg Q(f(y))$  as on the previous slide, we can get a resolvent  $\neg Q(f(a))$ , which is better.

## Full Resolution of Two Clauses

- Binary resolution of the clauses.
- Binary resolution of one clause with a factor of the other.
- Binary resolution of factors of both clauses.

## Case Where Factoring is Necessary

- $P(x) \vee P(y)$
- $\neg P(a) \vee \neg P(b)$
- Without factoring, generate:
- $P(y) \vee \neg P(b)$
- $P(x) \vee \neg P(a)$
- and more similar clauses, but never the empty clause.

## Case Where Factoring is Necessary

- $P(x) \vee P(y)$
- $\neg P(a) \vee \neg P(b)$
- With factoring, get factor  $p(x)$  from first clause, then generate:
- $\neg P(b)$
- $\perp$

## Clausal Form for Predicate Logic

- Often, we'll want to prove a sequent of the form
  - $\forall x \forall y (...)$
  - $\forall x \forall y (...)$
  - $\vdash \text{---}$
- For premises of the form  $\forall x \forall y (...)$  where ... has no quantifiers, we can just drop the quantifiers.
- We need to **negate** the conclusion  $\text{---}$ .

## Mushroom Example

1. Every fungus is a mushroom or a toadstool.
2. Every boletus is a fungus.
3. All toadstools are poisonous.
4. No boletus is a mushroom.
5. Specimen b is a boletus.
6. Therefore: Specimen b is poisonous.

## Mushroom Example

1.  $\forall X \text{ fungus}(X) \rightarrow (\text{mushroom}(X) \vee \text{toadstool}(X))$
2.  $\forall X \text{ boletus}(X) \rightarrow \text{fungus}(X)$
3.  $\forall X \text{ toadstool}(X) \rightarrow \text{poisonous}(X)$
4.  $\forall X \text{ boletus}(X) \rightarrow \neg \text{mushroom}(X)$
5.  $\text{boletus}(b)$
6. Therefore:  $\text{poisonous}(b)$

## Mushroom Clauses

1.  $\neg \text{fungus}(X) \vee \text{mushroom}(X) \vee \text{toadstool}(X)$
2.  $\neg \text{boletus}(X) \vee \text{fungus}(X)$
3.  $\neg \text{toadstool}(X) \vee \text{poisonous}(X)$
4.  $\neg \text{boletus}(X) \vee \neg \text{mushroom}(X)$
5.  $\text{boletus}(b)$
6.  $\neg \text{poisonous}(b)$  (negated conclusion)

## Mushroom Clauses in Prover9

```

-fungus(x) | mushroom(x) | toadstool(x) .
-boletus(x) | fungus(x) .
-toadstool(x) | poisonous(x) .
-boletus(x) | ¬mushroom(x) .
boletus(b) .

```

Goal:

```
poisonous(b) .
```

## Prover9 Output for Mushrooms

```

===== PROOF =====
% ----- Comments from original proof -----
% Proof 1 at 0.00 (-0.00) seconds.
% Length of proof is 13.
% Level of proof is 5.
% Maximum clause weight is 0.
% Given clauses 0.
1 poisonous(b) # Label(non_clause) # Label(goal). [goal].
2 ¬boletus(x) | fungus(x). [assumption].
3 ¬fungus(x) | mushroom(x) | toadstool(x). [assumption].
4 ¬boletus(x) | mushroom(x) | toadstool(x). [resolve(2,b,3,a)].
5 ¬toadstool(x) | poisonous(x). [assumption].
6 boletus(b). [assumption].
7 ¬boletus(x) | ¬mushroom(x). [assumption].
8 ¬boletus(x) | mushroom(x) | poisonous(x). [resolve(4,c,5,a)].
9 mushroom(b) | poisonous(b). [resolve(8,a,6,a)].
10 ¬poisonous(b). [deny(1)].
11 mushroom(b). [resolve(9,b,10,a)].
12 ¬mushroom(b). [resolve(6,a,7,a)].
13 $F. [resolve(11,a,12,a)].
===== end of proof =====

```

## Checking Unifiability with Prover9

- In contrast to Prolog, Prover9 *does* use an occur-check.

unification succeeds

```

-p(f(y), y).
p(x, g(z)).

```

unification fails due to occur-check

```

-p(f(y), y).
p(z, g(z)).

```

Proof:

```

1 p(x,g(y)). [assumption].
2 ¬p(f(x),x). [assumption].
3 $F. [resolve(1,a,2,a)].

```



## Clausal Form for Predicate Logic

- Often, we'll want to prove a sequent of the form
  - $\forall x \forall y (...)$ ,
  - $\forall x \forall y (...)$
  - $\vdash \forall x \forall y (...)$
- For premises of the form  $\forall x \forall y (...)$  where ... has no quantifiers, we can just drop the quantifiers.
- We need to **negate** the conclusion, so that will become
  - $\neg \forall x \forall y (...)$  which is equivalent to
  - $\exists x \exists y \neg (...)$ .

**We cannot simply drop the quantifiers in this case!!**

## Clausal Form for Predicate Logic

- Consider the sequent
 
$$\forall y p(y) \mid \neg \forall y p(x)$$
- The premise translates to a clause
 
$$p(y)$$
- The **conclusion** is negated to become  $\exists x \neg p(x)$ .
- How do we handle this?

## Skolem Constants/Functions to the Rescue!

- To get rid of the quantifier in
 
$$\exists x \neg p(x)$$
 we use a trick:
 

Create a **new constant**, say  $b$  (called a Skolem constant) and replace  $x$  with that:

$$\neg p(b)$$
- Some thought will show that:
 

There is an interpretation that satisfies  $\neg p(b)$  **iff** there is one that satisfies the original formula  $\exists x \neg p(x)$ .
- Colloquially, we get to pick the value for  $b$  in finding a satisfying interpretation, just as we get to pick the value for  $x$  in  $\exists x$ .

## Clausal Form for Predicate Logic

- Consider the sequent
 
$$\forall y p(y) \mid \neg \forall x p(x)$$
- The premise translates to a clause
 
$$p(y)$$
- The negated conclusion translates to a clause, where  $b$  is a Skolem function.
 
$$\neg p(b)$$
- We are good to go!
- Resolution produces  $\perp$  in 1 step.

## Another Example

- Consider the sequent
 
$$\exists x \forall y p(x, y) \mid \neg \forall y \exists x p(x, y)$$
- Premise clause:
 
$$p(b, y)$$
- Conclusion clause:
 
$$\neg p(x, c)$$

Resolution produces  $\perp$  in 1 step.

## Another Example

- Consider the sequent
 
$$\forall x a(x) \rightarrow \exists x b(x) \mid \neg \exists x (a(x) \rightarrow b(x))$$
- Premise clause:
 
$$\neg a(c) \vee b(d)$$
- Conclusion clauses:
 
$$a(x)$$

$$\neg b(x)$$

Resolution produces  $\perp$  in 2 steps.

## Skolem Functions for the General Case

- $\forall x \forall y \dots \exists v \dots$
- $v$  is replaced with  $f(x, y, \dots)$
- $f$  is a **new function symbol**, the arguments of which are the  $\forall$  quantified variables on the left.
- The rationale here is that "the  $v$ " that exists **depends on**  $x, y, \dots$
- Again, there is an interpretation satisfying the original formula iff there is an interpretation satisfying the revised formula.

## Example: Skolem with Arguments

- Prove: "The composition of two onto [surjective] functions is onto."
- **Represent the two functions as binary predicates.**  
 $F(x, y)$  means  $y$  is the image of  $x$ .
- "F is onto":  $\forall y \exists x F(x, y)$
- "G is onto":  $\forall z \exists y G(y, z)$
- "H is the composition of F and G":  

$$\forall x \forall y \forall z ((F(x, y) \wedge G(y, z)) \rightarrow H(x, z))$$

$$\wedge \forall x \forall z (H(x, z) \rightarrow \exists y (F(x, y) \wedge G(y, z)))$$
- "H is onto":  $\forall z \exists x H(x, z)$

## Translation to Clausal Form

- $\forall y \exists x F(x, y)$  becomes  $F(f(x), y)$  [ $f$  is a Skolem function]
- $\forall z \exists y G(y, z)$  becomes  $G(g(z), z)$  [ $g$  is a Skolem function]
- $\forall x \forall y \forall z ((F(x, y) \wedge G(y, z)) \rightarrow H(x, z))$  becomes  
 $\neg F(x, y) \vee \neg G(y, z) \vee H(x, z)$
- $\forall x \forall z (H(x, z) \rightarrow \exists y (F(x, y) \wedge G(y, z)))$  becomes  
 $\neg H(x, z) \vee F(x, h(x, z))$  [ $h$  is a Skolem function]  
 $\neg H(x, z) \vee G(h(x, z), z)$
- $\neg \forall z \exists x H(x, z)$  becomes  $\exists z \forall x \neg H(x, z)$ ,  
 which as a clause is:  
 $\neg H(x, a)$  [ $a$  is a Skolem constant]

## Resolution Proof

- |       |                                               |           |
|-------|-----------------------------------------------|-----------|
| 1.    | $F(f(x), y)$                                  | Premises  |
| 2.    | $G(g(z), z)$                                  |           |
| 3.    | $\neg F(x, y) \vee \neg G(y, z) \vee H(x, z)$ |           |
| 4.    | $\neg H(x, z) \vee F(x, h(x, z))$             |           |
| 5.    | $\neg H(x, z) \vee G(h(x, z), z)$             |           |
| 6.    | $\neg H(x, a)$                                |           |
| <hr/> |                                               |           |
| 7.    | $\neg F(x, y) \vee \neg G(y, a)$              | from 3, 6 |
| 8.    | $\neg G(y, a)$                                | from 1, 7 |
| 9.    | $\perp$                                       | from 2, 8 |
- (3 and 4 were not needed in the proof.)

## How to get a clause form in general?

- First convert the formula into "**prenex form**" (all quantifiers are outside on the left). [The parts of this form are the "prefix" and the "matrix".]
- Skolemize  $\exists$  quantified variables.
- Drop  $\forall$  quantifiers.
- Convert the resulting matrix to CNF.

## Conversion to Prenex Form

- Replace all connectives other than  $\wedge \vee \neg$  with their  $\wedge \vee \neg$  counterparts.
- Push negations inward
- Pull quantifiers to the outside using the rules on the next page.

## Basic Prenex Quantifier Rules

(for pulling quantifiers to the outside)

- We can show that the following replacements are equivalent.
- Here  $\Rightarrow$  means "replace with"
  - $(\forall x F) \wedge G \Rightarrow \forall x (F \wedge G)$ , provided  $x$  is not free in  $G$
  - $(\forall x F) \vee G \Rightarrow \forall x (F \vee G)$ , provided  $x$  is not free in  $G$
  - $(\exists x F) \wedge G \Rightarrow \exists x (F \wedge G)$ , provided  $x$  is not free in  $G$
  - $(\exists x F) \vee G \Rightarrow \exists x (F \vee G)$ , provided  $x$  is not free in  $G$
  - plus the symmetric counterparts of these rules with  $G$  part quantified instead of the  $F$  part.
  - Renaming some variables may be needed to enable the rule to be applied
- Example:
 

|                                            |               |                                            |                                   |
|--------------------------------------------|---------------|--------------------------------------------|-----------------------------------|
| $(\exists x F[x]) \wedge (\forall x G[x])$ | $\Rightarrow$ | $(\exists x F[x]) \wedge (\forall y G[y])$ | (by renaming second $x$ )         |
| $(\exists x F[x]) \wedge (\forall y G[y])$ | $\Rightarrow$ | $\exists x (F[x] \wedge (\forall y G[y]))$ | (by rule 3, as $x$ is not free)   |
| $\exists x (F[x] \wedge (\forall y G[y]))$ | $\Rightarrow$ | $\exists x \forall y (F[x] \wedge G[y])$   | (by rule 1 symmetric counterpart) |

## Example of Prenex Conversion

- $\forall x \forall y ((\exists z (p(x, z) \wedge p(y, z))) \Rightarrow \exists u q(x, y, u))$
  - $\forall x \forall y (\neg(\exists z (p(x, z) \wedge p(y, z))) \vee \exists u q(x, y, u))$
  - $\forall x \forall y ((\forall z \neg(p(x, z) \wedge p(y, z))) \vee \exists u q(x, y, u))$
  - $\forall x \forall y ((\forall z \neg(p(x, z) \vee \neg p(y, z))) \vee \exists u q(x, y, u))$
  - $\forall x \forall y \forall z (\neg p(x, z) \vee \neg p(y, z) \vee \exists u q(x, y, u))$
  - $\forall x \forall y \forall z \exists u (\neg p(x, z) \vee \neg p(y, z) \vee q(x, y, u))$
- prefix
matrix

## Completion of Conversion

- $\forall x \forall y \forall z \exists u (\neg p(x, z) \vee \neg p(y, z) \vee q(x, y, u))$
- Skolemize as  $f(x, y, z)$  and drop  $\forall x \forall y \forall z$
- $\neg p(x, z) \vee \neg p(y, z) \vee q(x, y, f(x, y, z))$

## Example: Group Theory Clauses

- $f$  is the group operation,  $e$  is the equality predicate
- $\forall x \forall y \forall x e(f(x, f(y, z)), f(f(x, y), z))$   
becomes  
 $e(f(x, f(y, z)), f(f(x, y), z))$
- $\forall x e(f(x, u), x)$   
becomes  
 $e(f(x, u), x)$
- $\forall x e(f(x, i(x)), u)$   
becomes  
 $e(f(x, i(x)), u)$

## Example: Equality Theory Clauses

- We need to axiomatize equality predicate  $e$ , e.g.
- $\forall x e(x, x)$   
becomes  
 $e(x, x)$
- $\forall x \forall y \forall u \forall v (e(x, y) \wedge e(v, w) \rightarrow e(f(x, v), f(y, w)))$   
becomes  
 $\neg e(x, y) \vee \neg e(v, w) \vee e(f(x, v), f(y, w))$
- $\forall x \forall y e(x, y) \rightarrow e(y, x)$   
becomes  
 $\neg e(x, y) \vee e(y, x)$
- etc.

## Example of Group Theory Clauses with Negated Conclusion

- $e(f(x, f(y, z)), f(f(x, y), z))$
- $e(f(x, u), x)$
- $e(f(x, i(x)), u)$
- $e(x, x)$
- $\neg e(x, y) \vee \neg e(v, w) \vee e(f(x, v), f(y, w))$
- $\neg e(x, y) \vee e(y, x)$
- $\neg e(x, y) \vee \neg e(y, z) \vee e(x, z)$
- $\neg e(i(i(b)), b)$

This is to show that  $\forall x e(i(i(x)), x)$ :

"In a group, the inverse of the inverse of an element is the element itself."

## Equality: Paramodulation

- Prover9 has a built-in equality, so the approach illustrated in the previous example is not generally necessary.
- Equality can be handled, for example, by the "paramodulation" rule, which essentially captures one of the two ND rules for equality:

$$\frac{\alpha \vee (s = t) \quad \beta \vee \chi[r]}{(\alpha \vee \beta \vee \chi[t])\theta}$$

$\chi[r]$  is a literal containing term  $r$   
 $\theta = \text{unify}(s, r)$

## Justification of Rules Using Natural Deduction: $\forall \wedge$ Rule a.

```

1:  $\forall x.F(x) \wedge G$  premise
2:  $G$   $\wedge$  elim 1
3:  $\forall x.F(x)$   $\wedge$  elim 1
4: actual i assumption
5:  $F(i)$   $\forall$  elim 3,4
6:  $F(i) \wedge G$   $\wedge$  intro 5,2
7:  $\forall x.(F(x) \wedge G)$   $\forall$  intro 4-6
    
```

Proviso is introduced by prefixing 'WHERE x NOTIN G IS' in Jape.

For equivalence, we need the converse of each rule:  $\forall \wedge$  Rule b.

```

1:  $\exists x.T, \forall x.(F(x) \wedge G)$  premises
2: actual i1 assumption
3:  $F(i) \wedge G$   $\forall$  elim 1,2,2
4:  $F(i)$   $\wedge$  elim 3
5:  $\forall x.F(x)$   $\forall$  intro 2-4
6: actual i, T assumptions
7:  $F(i) \wedge G$   $\forall$  elim 1,2,6,1
8:  $G$   $\wedge$  elim 7
9:  $G$   $\exists$  elim 1,1,6-8
10:  $\forall x.F(x) \wedge G$   $\wedge$  intro 5,9
    
```

Need the non-empty universe assumption in this direction.  
Otherwise, there is no way to get  $G$  by itself.

## Justification of Rules Using Natural Deduction: $\forall \vee$ Rule a.

```

1:  $\forall x.F(x) \vee G$  premise
2: actual i assumption
3:  $\forall x.F(x)$  assumption
4:  $F(i)$   $\forall$  elim 3,2
5:  $F(i) \vee G$   $\vee$  intro 4
6:  $G$  assumption
7:  $F(i) \vee G$   $\vee$  intro 6
8:  $F(i) \vee G$   $\vee$  elim 1,3-5,6-7
9:  $\forall x.(F(x) \vee G)$   $\forall$  intro 2-8
    
```

## Justification of Rules Using Natural Deduction: $\forall \vee$ Rule b.

```

1:  $\forall x.(F(x) \vee G)$  premise
2:  $G \vee \neg G$  Theorem E- $\vee$ -E
3:  $G$  assumption
4:  $\forall x.F(x) \vee G$   $\forall$  intro 3
5:  $\neg G$  assumption
6: actual i assumption
7:  $F(i) \vee G$   $\forall$  elim 1,6
8:  $F(i)$  assumption
9:  $G$  assumption
10:  $\perp$   $\neg$  elim 9,5
11:  $F(i)$  contra (constructive) 10
12:  $F(i)$   $\vee$  elim 7,8-9,11
13:  $\forall x.F(x)$   $\forall$  intro 6-12
14:  $\forall x.F(x) \vee G$   $\vee$  intro 13
15:  $\forall x.F(x) \vee G$   $\vee$  elim 2,3-4,5-14
    
```

## Justification of Rules Using Natural Deduction: $\exists \vee$ a.

```

1:  $\exists x.T, \exists x.F(x) \vee G$  premises
2:  $\exists x.F(x)$  assumption
3: actual i, F(i) assumptions
4:  $F(i) \vee G$   $\vee$  intro 3,2
5:  $\exists x.(F(x) \vee G)$   $\exists$  intro 4,3,1
6:  $\exists x.(F(x) \vee G)$   $\exists$  elim 2,3-5
7:  $G$  assumption
8: actual i1, T assumptions
9:  $F(i) \vee G$   $\vee$  intro 7
10:  $\exists x.(F(x) \vee G)$   $\exists$  intro 9,8,1
11:  $\exists x.(F(x) \vee G)$   $\exists$  elim 1,1,8-10
12:  $\exists x.(F(x) \vee G)$   $\vee$  elim 1,2,2-6,7-11
    
```

Non-empty universe assumption, needed in 7-11

## Justification of Rules Using Natural Deduction: $\exists \vee$ b.

| $\exists x.(F(x) \vee G) \vdash \exists x.F(x) \vee G$ |                         |
|--------------------------------------------------------|-------------------------|
| 1: $\exists x.(F(x) \vee G)$                           | premise                 |
| 2: <b>actual i</b> , $F(i) \vee G$                     | assumptions             |
| 3: $F(i)$                                              | assumption              |
| 4: $\exists x.F(x)$                                    | $\exists$ intro 3,2,1   |
| 5: $\exists x.F(x) \vee G$                             | $\vee$ intro 4          |
| 6: $G$                                                 | assumption              |
| 7: $\exists x.F(x) \vee G$                             | $\vee$ intro 6          |
| 8: $\exists x.F(x) \vee G$                             | $\vee$ elim 2,2,3-5,6-7 |
| 9: $\exists x.F(x) \vee G$                             | $\exists$ elim 1,2-8    |
| Provided:<br>x NOTIN G                                 |                         |

## Justification of Rules Using Natural Deduction: $\exists \wedge$ a.

| $\exists x.F(x) \wedge G \vdash \exists x.(F(x) \wedge G)$ |                       |
|------------------------------------------------------------|-----------------------|
| 1: $\exists x.F(x) \wedge G$                               | premise               |
| 2: $G$                                                     | $\wedge$ elim 1       |
| 3: $\exists x.F(x)$                                        | $\wedge$ elim 1       |
| 4: <b>actual i</b> , $F(i)$                                | assumptions           |
| 5: $F(i) \wedge G$                                         | $\wedge$ intro 4,2,2  |
| 6: $\exists x.(F(x) \wedge G)$                             | $\exists$ intro 5,4,1 |
| 7: $\exists x.(F(x) \wedge G)$                             | $\exists$ elim 3,4-6  |
| Provided:<br>x NOTIN G                                     |                       |

## Justification of Rules Using Natural Deduction: $\exists \wedge$ b.

| $\exists x.(F(x) \wedge G) \vdash \exists x.F(x) \wedge G$ |                       |
|------------------------------------------------------------|-----------------------|
| 1: $\exists x.(F(x) \wedge G)$                             | premise               |
| 2: <b>actual i1</b> , $F(i1) \wedge G$                     | assumptions           |
| 3: $F(i1)$                                                 | $\wedge$ elim 2,2     |
| 4: $\exists x.F(x)$                                        | $\exists$ intro 3,2,1 |
| 5: $\exists x.F(x)$                                        | $\exists$ elim 1,2-4  |
| 6: <b>actual i</b> , $F(i) \wedge G$                       | assumptions           |
| 7: $G$                                                     | $\wedge$ elim 6,2     |
| 8: $G$                                                     | $\exists$ elim 1,6-7  |
| 9: $\exists x.F(x) \wedge G$                               | $\wedge$ intro 5,8    |
| Provided:<br>x NOTIN G                                     |                       |