Introduction to Logic
Resolution

Michael Genesereth
Computer Science Department
Stanford University

Problem With Natural Deduction

Assumption

$$p(a)$$
$$p(f(a))$$
$$p(x) => q(x)$$
$$\sim p(x) => q(x)$$

Universal Elimination

$$\forall x.\forall y.q(x,y)$$

$q(a,a)$	$q(a,[c])$
$q(a,b)$	$q(a,[c])$
$q(a, f(a))$	$q(a, f([c]))$
…	…

Resolution Principle

The *Resolution Principle* is a rule of inference. Using the Resolution Principle *alone*, it is possible to build a proof system (called *Resolution*) that can prove everything that can be proved in Fitch (without domain closure or induction).

There is no need to make arbitrary assumptions or replacements.

The search space using the Resolution Principle is better controlled than that of natural deduction systems.

Relies on functional notation.

Programme

Clausal Form

Unification

Resolution Rule of Inference

Unsatisfiability

Logical Entailment

Answer Extraction

Clausal Form

Clausal Form

Resolution works only on expressions in *clausal form*.

Fortunately, it is possible to convert any set of sentences into an equally satisfiable set of expressions in clausal form.

Clausal Form

A *literal* is either an atomic sentence or a negation of an atomic sentence.

$$p(a),\ \neg p(y)$$

A *clausal sentence* is either a literal or a disjunction of literals.

$$p(a),\ \neg p(y),\ \ p(a) \vee \neg p(y)$$

A *clause* is a set of literals.

$$\{p(a)\}, \{\neg p(y)\}, \{p(a), \neg p(y)\}$$

Empty Sets

The empty clause {} is unsatisfiable.

Why? It is equivalent to an empty disjunction.

Inseado

Implications Out:

$$\varphi_1 \Rightarrow \varphi_2 \quad \rightarrow \quad \neg \varphi_1 \vee \varphi_2$$

$$\varphi_1 \Leftrightarrow \varphi_2 \quad \rightarrow \quad (\neg \varphi_1 \vee \varphi_2) \wedge (\varphi_1 \vee \neg \varphi_2)$$

Inseado

Implications Out:

$$\begin{aligned}
\varphi_1 \Rightarrow \varphi_2 &\quad\rightarrow\quad \neg\varphi_1 \vee \varphi_2 \\
\varphi_1 \Leftrightarrow \varphi_2 &\quad\rightarrow\quad (\neg\varphi_1 \vee \varphi_2) \wedge (\varphi_1 \vee \neg\varphi_2)
\end{aligned}$$

Negations In:

$$\begin{aligned}
\neg\neg\varphi &\quad\rightarrow\quad \varphi \\
\neg(\varphi_1 \wedge \varphi_2) &\quad\rightarrow\quad \neg\varphi_1 \vee \neg\varphi_2 \\
\neg(\varphi_1 \vee \varphi_2) &\quad\rightarrow\quad \neg\varphi_1 \wedge \neg\varphi_2 \\
\neg\forall\nu.\varphi &\quad\rightarrow\quad \exists\nu.\neg\varphi \\
\neg\exists\nu.\varphi &\quad\rightarrow\quad \forall\nu.\neg\varphi
\end{aligned}$$

Inseado

Standardize variables

$$\forall x.p(x) \vee \forall x.q(x) \quad \rightarrow \quad \forall x.p(x) \vee \forall y.q(y)$$

Inseado

Standardize variables

$$\forall x.p(x) \lor \forall x.q(x) \quad \rightarrow \quad \forall x.p(x) \lor \forall y.q(y)$$

Existentials Out (Outside in)

$$\exists x.p(x) \quad \rightarrow \quad p(a)$$

Inseado

Standardize variables

$$\forall x.p(x) \vee \forall x.q(x) \quad \rightarrow \quad \forall x.p(x) \vee \forall y.q(y)$$

Existentials Out (Outside in)

$$\exists x.p(x) \quad \rightarrow \quad p(a)$$

$$\forall x.(p(x) \wedge \exists z.q(x,y,z)) \quad \rightarrow \quad \forall x.(p(x) \wedge q(x,y,f(x,y)))$$

Inseado

Alls Out

$$\forall x.(p(x) \land q(x,y,f(x,y))) \quad \rightarrow \quad p(x) \land q(x,y,f(x,y))$$

Inseado

Alls Out

$$\forall x.(p(x) \wedge q(x,y,f(x,y))) \quad \rightarrow \quad p(x) \wedge q(x,y,f(x,y))$$

Distribution

$$\begin{aligned}
\varphi_1 \vee (\varphi_2 \wedge \varphi_3) &\quad\rightarrow\quad (\varphi_1 \vee \varphi_2) \wedge (\varphi_1 \vee \varphi_n) \\
(\varphi_1 \wedge \varphi_2) \vee \varphi_3 &\quad\rightarrow\quad (\varphi_1 \vee \varphi_3) \wedge (\varphi_2 \vee \varphi_3) \\
\varphi \vee (\varphi_1 \vee \ldots \vee \varphi_n) &\quad\rightarrow\quad (\varphi \vee \varphi_1 \vee \ldots \vee \varphi_n) \\
(\varphi_1 \vee \ldots \vee \varphi_n) \vee \varphi &\quad\rightarrow\quad (\varphi_1 \vee \ldots \vee \varphi_n \vee \varphi) \\
\varphi \wedge (\varphi_1 \wedge \ldots \wedge \varphi_n) &\quad\rightarrow\quad (\varphi \wedge \varphi_1 \wedge \ldots \wedge \varphi_n) \\
(\varphi_1 \wedge \ldots \wedge \varphi_n) \wedge \varphi &\quad\rightarrow\quad (\varphi_1 \wedge \ldots \wedge \varphi_n \wedge \varphi)
\end{aligned}$$

Inseado

Operators Out

$$\varphi_1 \wedge \dots \wedge \varphi_n \quad \rightarrow \quad \begin{array}{l} \varphi_1 \\ \dots \\ \varphi_n \end{array}$$

$$\varphi_1 \vee \dots \vee \varphi_n \quad \rightarrow \quad \{\varphi_1, \dots, \varphi_n\}$$

Example

	$\exists y.(g(y) \wedge \forall z.(r(z) \Rightarrow f(y,z)))$
I	$\exists y.(g(y) \wedge \forall z.(\neg r(z) \vee f(y,z)))$
N	$\exists y.(g(y) \wedge \forall z.(\neg r(z) \vee f(y,z)))$
S	$\exists y.(g(y) \wedge \forall z.(\neg r(z) \vee f(y,z)))$
E	$g(greg) \wedge \forall z.(\neg r(z) \vee f(greg,z)))$
A	$g(greg) \wedge (\neg r(z) \vee f(greg,z)))$
D	$g(greg) \wedge (\neg r(z) \vee f(greg,z)))$
O	$\{g(greg)\}$
	$\{\neg r(z), f(greg,z)\}$

Example

$$\neg \exists y.(g(y) \wedge \forall z.(r(z) \Rightarrow f(y,z)))$$

I $$\neg \exists y.(g(y) \wedge \forall z.(\neg r(z) \vee f(y,z)))$$

N $$\neg \exists y.(g(y) \wedge \forall z.(\neg r(z) \vee f(y,z)))$$

$$\forall y.\neg(g(y) \wedge \forall z.(\neg r(z) \vee f(y,z)))$$

$$\forall y.(\neg g(y) \vee \neg \forall z.(\neg r(z) \vee f(y,z)))$$

$$\forall y.(\neg g(y) \vee \exists z.\neg(\neg r(z) \vee f(y,z)))$$

$$\forall y.(\neg g(y) \vee \exists z.(\neg \neg r(z) \wedge \neg f(y,z)))$$

$$\forall y.(\neg g(y) \vee \exists z.(r(z) \wedge \neg f(y,z)))$$

S $$\forall y.(\neg g(y) \vee \exists z.(r(z) \wedge \neg f(y,z)))$$

Example

	$\forall y.(\neg g(y) \vee \exists z.(r(z) \wedge \neg f(y,z)))$
E	$\forall y.(\neg g(y) \vee (r(h(y)) \wedge \neg f(y,h(y))))$
A	$\neg g(y) \vee (r(h(y)) \wedge \neg f(y,h(y)))$
D	$(\neg g(y) \vee r(h(y))) \wedge (\neg g(y) \vee \neg f(y,h(y)))$
O	$\{\neg g(y), r(h(y))\}$
	$\{\neg g(y), \neg f(y,h(y))\}$

Clausal Form

Bad News: The result of converting a set of sentences is not necessarily logically equivalent to the original set of sentences. Why? Introduction of new constants and functions.

Good News: The result of converting a set of sentences is satisfiable in the expanded language if and only if the original set of sentences is satisfiable in the original language. Important because we use satisfiability to determine logical entailment.

Potential problem with domain closure and induction eliminated by keeping new constants separate form old.

Unification

Unification

Unification is the process of determining whether two expressions can be *unified*, i.e. made identical by appropriate substitutions for their variables.

Substitutions

A *substitution* is a finite set of pairs of variables and terms, called *replacements*.

$$\{x \leftarrow a, y \leftarrow f(b), v \leftarrow w\}$$

The result of applying a substitution σ to an expression φ is the expression φσ obtained from φ by replacing every occurrence of every variable in the substitution by its replacement.

$$p(x,x,y,z)\{x \leftarrow a, y \leftarrow f(b), v \leftarrow w\} = p(a,a,f(b),z)$$

Cascaded Substitutions

$$r\{x,y,z\}\{x \leftarrow a,\ y \leftarrow f(u),\ z \leftarrow v\} = r\{a, f(u), v\}$$

$$r\{a, f(u), v\}\{u \leftarrow d,\ v \leftarrow e,\ z \leftarrow g\} = r(a, f(d), e)$$

$$r\{x,y,z\}\{x \leftarrow a,\ y \leftarrow f(d),\ z \leftarrow e,\ u \leftarrow d,\ v \leftarrow e\} = r(a, f(d), e)$$

Composition of Substitutions

The *composition* of substitution σ and τ is the substitution (written *compose(*σ,τ*)* or, more simply, στ) obtained by
(1) applying τ to the replacements in σ
(2) adding to σ pairs from τ with different variables
(3) deleting any assignments of a variable to itself.

$$\{x \leftarrow a, y \leftarrow f(u), z \leftarrow v\}\{u \leftarrow d, v \leftarrow e, z \leftarrow g\}$$

$$= \{x \leftarrow a, y \leftarrow f(d), z \leftarrow e\}\{u \leftarrow d, v \leftarrow e, z \leftarrow g\}$$

$$= \{x \leftarrow a, y \leftarrow f(d), z \leftarrow e, u \leftarrow d, v \leftarrow e\}$$

Unification

A substitution σ is a *unifier* for an expression φ and an expression ψ if and only if $\varphi\sigma=\psi\sigma$.

$$p(x,y)\{x\leftarrow a, y\leftarrow b, v\leftarrow b\}=p(a,b)$$
$$p(a,v)\{x\leftarrow a, y\leftarrow b, v\leftarrow b\}=p(a,b)$$

If two expressions have a unifier, they are said to be *unifiable*. Otherwise, they are *nonunifiable*.

$$p(x,x)$$
$$p(a,b)$$

NonUniqueness of Unification

Unifier 1:

$$p(x,y)\{x \leftarrow a, y \leftarrow b, v \leftarrow b\} = p(a,b)$$
$$p(a,v)\{x \leftarrow a, y \leftarrow b, v \leftarrow b\} = p(a,b)$$

Unifier 2:

$$p(x,y)\{x \leftarrow a, y \leftarrow f(w), v \leftarrow f(w)\} = p(a,f(w))$$
$$p(a,v)\{x \leftarrow a, y \leftarrow f(w), v \leftarrow f(w)\} = p(a,f(w))$$

Unifier 3:

$$p(x,y)\{x \leftarrow a, y \leftarrow v\} = p(a,v)$$
$$p(a,v)\{x \leftarrow a, y \leftarrow v\} = p(a,v)$$

Most General Unifier

A substitution σ is a *most general unifier* (*mgu*) of two expressions if and only if it is as general as or more general than any other unifier.

Theorem: If two expressions are unifiable, then they have an mgu that is unique up to variable permutation.

$$p(x,y)\{x \leftarrow a, y \leftarrow v\} = p(a,v)$$
$$p(a,v)\{x \leftarrow a, y \leftarrow v\} = p(a,v)$$

$$p(x,y)\{x \leftarrow a, v \leftarrow y\} = p(a,y)$$
$$p(a,v)\{x \leftarrow a, v \leftarrow y\} = p(a,y)$$

Unification Procedure

One good thing about our language is that there is a simple and inexpensive procedure for computing a most general unifier of any two expressions if it exists.

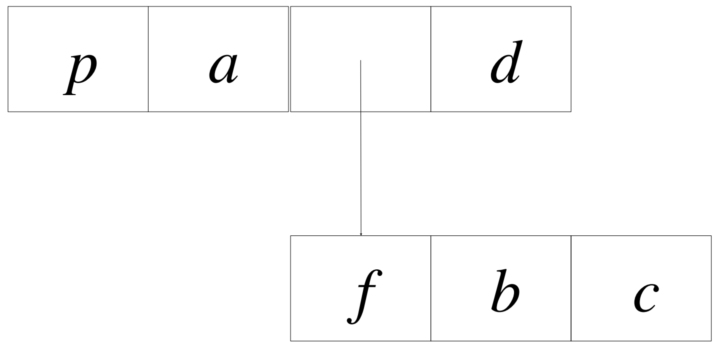
Expression Structure

Each expression is treated as a sequence of its immediate subexpressions.

Linear Version:

$$p(a, f(b, c), d)$$

Structured Version:



Unification Procedure

(1) If two expressions being compared are identical, succeed.

(2) If neither is a variable and at least one is a constant, fail.

(3) If at least one of the expressions is a variable, proceed as described shortly.

(4) If both expressions are sequences, iterate across the expressions, comparing as described above.

Dealing With Variables

If one of the expressions is a variable, check whether the variable has a binding in the current substitution.

(a) If so, try to unify the binding with the other expression.

(b) If no binding, check whether the other expression contains the variable. If the variable occurs within the expression, fail; otherwise, set the substitution to the composition of the old substitution and a new substitution in which variable is bound to the other expression.

Example

Call: $p(x,b)$, $p(a,y)$, $\{\}$

Call: p, p, $\{\}$
Exit: $\{\}$

Call: x, a, $\{\}$
Exit: $\{\}\{x \leftarrow a\} = \{x \leftarrow a\}$

Call: b, y, $\{x \leftarrow a\}$
Exit: $\{x \leftarrow a\}\{y \leftarrow b\} = \{x \leftarrow a, y \leftarrow b\}$

Exit: $\{x \leftarrow a, y \leftarrow b\}$

Example

Call: $p(x,x)$, $p(a,y)$, $\{\}$

  Call: p, p, $\{\}$
  Exit: $\{\}$

  Call: x, a, $\{\}$
  Exit: $\{\}\{x\leftarrow a\} = \{x\leftarrow a\}$

  Call: x, y, $\{x\leftarrow a\}$
    Call: a, y, $\{x\leftarrow a\}$
    Exit: $\{x\leftarrow a\}\{y\leftarrow a\} = \{x\leftarrow a, y\leftarrow a\}$
  Exit: $\{x\leftarrow a, y\leftarrow a\}$

Exit: $\{x\leftarrow a, y\leftarrow a\}$

Example

Call: $p(x,x)$, $p(a,b)$, $\{\}$

Call: p, p, $\{\}$
Exit: $\{\}$

Call: x, a, $\{\}$
Exit: $\{\}\{x\leftarrow a\} = \{x\leftarrow a\}$

Call: x, b, $\{x\leftarrow a\}$
Call: a, b, $\{x\leftarrow a\}$
Exit: *false*
Exit: *false*

Exit: *false*

Example

Call: $p(x,x)$, $p(y,f(y))$, $\{\}$

  Call: p, p, $\{\}$
  Exit: $\{\}$

  Call: x, y, $\{\}$
  Exit: $\{\}\{x \leftarrow y\} = \{x \leftarrow y\}$

  Call: x, $f(y)$, $\{x \leftarrow y\}$
    Call: y, $f(y)$, $\{x \leftarrow y\}$
    Exit: *false*
  Exit: *false*

Exit: *false*

Reason

Circularity Problem:

$$\{x \leftarrow f(y), y \leftarrow f(y)\}$$

Unification Problem:

$$p(x,x)\{x \leftarrow f(y), y \leftarrow f(y)\} \quad = p(f(y),f(y))$$

$$p(y,f(y))\{x \leftarrow f(y), y \leftarrow f(y)\} = p(f(y),f(f(y)))$$

Solution

Before assigning a variable to an expression, first check that the variable does not occur within that expression.

This is called, oddly enough, the *occur check* test.

Prolog does not do the occur check (and is proud of it).

Resolution Principle

Propositional Resolution

$$\frac{\{\varphi_1,\ldots,\varphi,\ldots,\varphi_m\} \qquad \{\psi_1,\ldots,\neg\varphi,\ldots,\psi_n\}}{\{\varphi_1,\ldots,\varphi_m,\psi_1,\ldots,\psi_n\}}$$

Resolution (simple version)

$$\frac{\begin{array}{l}\{\varphi_1,\ldots,\quad\varphi,\ldots,\varphi_m\}\\ \{\psi_1,\ldots,\neg\psi,\ldots,\psi_n\}\end{array}}{\{\varphi_1,\ldots,\varphi_m,\psi_1,\ldots,\psi_n\}\sigma}$$

where $\sigma = mgu(\varphi,\psi)$

Example

$$\frac{\begin{array}{l}\{\ \ p(a,y), r(y)\} \\ \{\neg p(x,b), s(x)\}\end{array}}{\begin{array}{l}\{r(y), s(x)\}\{x \leftarrow a, y \leftarrow b\} \\ \{r(b), s(a)\}\end{array}}$$

Problem

$$\frac{\begin{array}{l}\{\ \ p(a,x)\}\\ \{\neg p(x,b)\}\end{array}}{\text{Failure}}$$

Resolution (improved)

$$\frac{\begin{array}{l}\{\varphi_1,\ldots,\quad \varphi,\ldots,\varphi_m\}\\ \{\psi_1,\ldots,\neg\psi,\ldots,\psi_n\}\end{array}}{\{\varphi_1\tau,\ldots,\varphi_m\tau,\psi_1,\ldots,\psi_n\}\sigma}$$

where $\sigma = mgu(\varphi\tau,\psi)$

where τ is a variable renaming on φ

Example

$$\begin{array}{l} \{\ p(a,x)\} \\ \{\neg p(x,b)\} \\ \hline \text{Failure} \end{array} \qquad \text{———} \qquad \begin{array}{l} \{\ p(a,y)\} \\ \{\neg p(x,b)\} \\ \hline \{\}\{x \leftarrow a, y \leftarrow b\} \end{array}$$

Problem

$$\frac{\{p(x), p(y)\} \quad \{\neg p(u), \neg p(v)\}}{\{p(y), \neg p(v)\}}$$

$$\{p(x), \neg p(v)\}$$

$$\{p(y), \neg p(u)\}$$

$$\{p(x), \neg p(u)\}$$

Factors

If a subset of the literals in a clause Φ has a most general unifier γ, then the clause Φ' obtained by applying γ to Φ is called a *factor* of Φ.

Clause

$$\{p(x), p(f(y)), r(x,y)\}$$

Factors

$$\{p(f(y)), r(f(y),y)\}$$

$$\{p(x), p(f(y)), r(x,y)\}$$

Resolution (final version)

$$\frac{\begin{array}{l}\Phi \\ \Psi\end{array}}{((\Phi' - \{\phi\})\tau \cup (\Psi' - \{\neg\psi\}))\sigma}$$

where $\phi \in \Phi'$, a factor of Φ

where $\neg\psi \in \Psi'$, a factor of Ψ

where $\sigma = mgu(\varphi\tau, \psi)$

where τ is a variable renaming on φ

Example

$$\begin{array}{lcl}
\{p(x),p(y)\} & \rule[0.5ex]{3em}{0.4pt} & \{p(x)\} \\
\underline{\{\neg p(u),\neg p(v)\}} & \rule[0.5ex]{3em}{0.4pt} & \underline{\{\neg p(u)\}} \\
\{p(y),\neg p(v)\} & & \{\} \\
\{p(x),\neg p(v)\} & & \\
\{p(y),\neg p(u)\} & & \\
\{p(x),\neg p(u)\} & &
\end{array}$$

Need for Original Clauses

1. $\{p(a,y), p(x,b)\}$	Premise
2. $\{\neg p(a,d)\}$	Premise
3. $\{\neg p(c,b)\}$	Premise
4. $\{p(x,b)\}$	1, 2
5. $\{\}$	3, 4

1. $\{p(a,y), p(x,b)\}$	Premise
2. $\{\neg p(a,d)\}$	Premise
3. $\{\neg p(c,b)\}$	Premise
4. $\{p(a,b)\}$	Factor of 1

Resolution Reasoning

Resolution Derivation

A *resolution derivation* of a conclusion from a set of premises is a finite sequence of clauses terminating in the conclusion in which each clause is either a premise or the result of applying the resolution principle to earlier elements of the sequence.

Example

1.	$\{p(art,bob)\}$	Premise
2.	$\{p(art,bud)\}$	Premise
3.	$\{p(bob,cal)\}$	Premise
4.	$\{p(bud,coe)\}$	Premise
5.	$\{\neg p(x,y), \neg p(y,z), g(x,z)\}$	Premise

Example

1.	$\{p(art,bob)\}$	Premise
2.	$\{p(art,bud)\}$	Premise
3.	$\{p(bob,cal)\}$	Premise
4.	$\{p(bud,coe)\}$	Premise
5.	$\{\neg p(x,y), \neg p(y,z), g(x,z)\}$	Premise
6.	$\{\neg p(bob,z), g(art,z)\}$	1, 5

Example

1.	$\{p(art,bob)\}$	Premise
2.	$\{p(art,bud)\}$	Premise
3.	$\{p(bob,cal)\}$	Premise
4.	$\{p(bud,coe)\}$	Premise
5.	$\{\neg p(x,y), \neg p(y,z), g(x,z)\}$	Premise
6.	$\{\neg p(bob,z) , g(art,z)\}$	1, 5
7.	$\{g(art,cal)\}$	3, 6

Example

1.	$\{p(art,bob)\}$	Premise
2.	$\{p(art,bud)\}$	Premise
3.	$\{p(bob,cal)\}$	Premise
4.	$\{p(bud,coe)\}$	Premise
5.	$\{\neg p(x,y), \neg p(y,z), g(x,z)\}$	Premise
6.	$\{\neg p(bob,z), g(art,z)\}$	1, 5
7.	$\{g(art,cal)\}$	3, 6
8.	$\{\neg p(bud,z), g(art,z)\}$	2, 5

Example

1.	$\{p(art,bob)\}$	Premise
2.	$\{p(art,bud)\}$	Premise
3.	$\{p(bob,cal)\}$	Premise
4.	$\{p(bud,coe)\}$	Premise
5.	$\{\neg p(x,y), \neg p(y,z), g(x,z)\}$	Premise
6.	$\{\neg p(bob,z), g(art,z)\}$	1, 5
7.	$\{g(art,cal)\}$	3, 6
8.	$\{\neg p(bud,z), g(art,z)\}$	2, 5
9.	$\{g(art,coe)\}$	4, 8

Example

1.	$\{p(art,bob)\}$	Premise
2.	$\{p(art,bud)\}$	Premise
3.	$\{p(bob,cal)\}$	Premise
4.	$\{p(bud,coe)\}$	Premise
5.	$\{\neg p(x,y), \neg p(y,z), g(x,z)\}$	Premise
6.	$\{\neg p(bob,z), g(art,z)\}$	1, 5
7.	$\{g(art,cal)\}$	3, 6
8.	$\{\neg p(bud,z), g(art,z)\}$	2, 5
9.	$\{g(art,coe)\}$	4, 8

Example

1.	$\{p(art,bob)\}$	Premise
2.	$\{p(art,bud)\}$	Premise
3.	$\{p(bob,cal)\}$	Premise
4.	$\{p(bud,coe)\}$	Premise
5.	$\{\neg p(x,y), \neg p(y,z), g(x,z)\}$	Premise
6.	$\{\neg p(bob,z), g(art,z)\}$	1, 5
7.	$\{g(art,cal)\}$	3, 6
8.	$\{\neg p(bud,z), g(art,z)\}$	2, 5
9.	$\{g(art,coe)\}$	4, 8

Example

1.	$\{p(art,bob)\}$	Premise
2.	$\{p(art,bud)\}$	Premise
3.	$\{p(bob,cal)\}$	Premise
4.	$\{p(bud,coe)\}$	Premise
5.	$\{\neg p(x,y), \neg p(y,z), g(x,z)\}$	Premise
6.	$\{\neg p(bob,z), g(art,z)\}$	1, 5
7.	$\{g(art,cal)\}$	3, 6
8.	$\{\neg p(bud,z), g(art,z)\}$	2, 5
9.	$\{g(art,coe)\}$	4, 8

Resolution Not Generatively Complete

Using the Resolution Principle alone, it is not possible to generate every clause that is logically entailed by a set of premises.

Examples:

$$\{\} \models \{p(a), \neg p(a)\}$$

But resolution cannot generate these results.

Unsatisfiability

Demonstrating Unsatisfiability

Start with premises.

Apply resolution repeatedly.

If empty clause generated, the original set is unsatisfiable.

Example

1.	$\{p(a,b), q(a,c)\}$	Premise
2.	$\{\neg p(x,y), r(x)\}$	Premise
3.	$\{\neg q(x,y), r(x)\}$	Premise
4.	$\{\neg r(z)\}$	Premise

Example

1.	$\{p(a,b), q(a,c)\}$	Premise
2.	$\{\neg p(x,y), r(x)\}$	Premise
3.	$\{\neg q(x,y), r(x)\}$	Premise
4.	$\{\neg r(z)\}$	Premise
5.	$\{q(a,c), r(a)\}$	1, 2

Example

1.	$\{p(a,b), q(a,c)\}$	Premise
2.	$\{\neg p(x,y), r(x)\}$	Premise
3.	$\{\neg q(x,y), r(x)\}$	Premise
4.	$\{\neg r(z)\}$	Premise
5.	$\{q(a,c), r(a)\}$	1, 2
6.	$\{r(a)\}$	5, 3

Example

1.	$\{p(a,b), q(a,c)\}$	Premise
2.	$\{\neg p(x,y), r(x)\}$	Premise
3.	$\{\neg q(x,y), r(x)\}$	Premise
4.	$\{\neg r(z)\}$	Premise
5.	$\{q(a,c), r(a)\}$	1, 2
6.	$\{r(a)\}$	5, 3
7.	$\{\}$	6, 4

Logical Entailment

Provability

A *resolution derivation* of a clause φ from a set Δ of clauses is a sequence of clauses ending in φ in which each item is
(1) a member of Δ or
(2) the result of applying the resolution to earlier items.

A sentence φ is *provable* from a set of sentences Δ by resolution if and only if there is a derivation of the empty clause from the clausal form of $\Delta \cup \{\neg\varphi\}$.

A resolution *proof* is a derivation of the empty clause from the clausal form of the premises and the negation of the desired conclusion.

Example

Everybody loves somebody. Everybody loves a lover.
Show that everybody loves everybody.

$$\forall x . \exists y . loves(x,y)$$

$$\forall u . \forall v . \forall w . (loves(v,w) \Rightarrow loves(u,v))$$

$$\neg \forall x . \forall y . loves(x,y)$$

$$\{loves(x, f(x))\}$$

$$\{\neg loves(v,w), loves(u,v)\}$$

$$\{\neg loves(jack, jill)$$

Example

1.	$\{loves(x,f(x))\}$	Premise
2.	$\{\neg loves(v,w), loves(u,v)\}$	Premise
3.	$\{\neg loves(jack,jill)\}$	Negated Goal

Example (continued)

1.	$\{loves(x,f(x))\}$	Premise
2.	$\{\neg loves(v,w), loves(u,v)\}$	Premise
3.	$\{\neg loves(jack,jill)\}$	Negated Goal
4.	$\{loves(u,x)\}$	1, 2

Example (concluded)

1.	$\{loves(x,f(x))\}$	Premise
2.	$\{\neg loves(v,w), loves(u,v)\}$	Premise
3.	$\{\neg loves(jack,jill)\}$	Negated Goal
4.	$\{loves(u,x)\}$	1, 2
5.	$\{\}$	4, 3

Inferential Equivalence

Equivalence Theorem: It is possible to prove a conclusion from a set of premises using Resolution if and only if it is possible to prove that conclusion from those premises using Fitch (without domain closure or induction).