MSc Module CS612 Automated Reasoning
Prolog, Resolution and Logic Programming

Alan Williams    Room 2.107

email: alanw@cs.man.ac.uk

School of Computer Science
November 2005
## This Part of the Course

<table>
<thead>
<tr>
<th>Mon</th>
<th>Session 1 (9:00-10:30)</th>
<th>Session 2 (11:00-12:30)</th>
<th>Lunch</th>
<th>Session 3 (2:00-3:30)</th>
<th>Session 4 (4:00-5:00)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>(AW1) Intro: Pre-Course (lecture)</td>
<td>(AW2) Prop Res I (lecture)</td>
<td></td>
<td>(AW3) Prop Res II (lect &amp; ex)</td>
<td>(AW4) Pred Intro (lecture)</td>
</tr>
<tr>
<td>Tue</td>
<td>(AW5) Pred Res I (lecture)</td>
<td>(AW6) Prolog I (lab)</td>
<td></td>
<td>(AW7) Pred Res II (lect &amp; ex)</td>
<td>(AW8) Prolog II (lab)</td>
</tr>
<tr>
<td>Wed</td>
<td>(AW9) Log Prog (lecture)</td>
<td>(AW10) Prolog III (lab)</td>
<td></td>
<td>(RS1) Orderings (lect &amp; ex)</td>
<td>(RS2) H. models (lect &amp; ex)</td>
</tr>
<tr>
<td>Thu</td>
<td>(RS3) model construction, completeness (lect &amp; ex)</td>
<td>(RS4) F.o. resol, lifting, consequences (lect &amp; ex)</td>
<td></td>
<td>(RS5) ordered resol with selection; redundancy (lect &amp; ex)</td>
<td>(RS6) res prover in practice; hyper-resol; lpos (lect &amp; ex)</td>
</tr>
<tr>
<td>Fri</td>
<td>(RS7) key exchange protocol (lect &amp; lab)</td>
<td>(RS8) Using MSPASS, VAMPIRE (lab)</td>
<td></td>
<td>(RS9) Prop tableau (lect &amp; ex)</td>
<td>(RS10) F.o. tableau (lect &amp; ex)</td>
</tr>
</tbody>
</table>
Books


Contents of First Part

- Propositional Resolution
- First Order Predicate Logic (FOPL)
- Resolution for Predicate Logic
- Prolog
- Logic Programming
- Course Summary
Lecture 2: Propositional Resolution
Propositional Resolution: Plan

• Proof by refutation (as with semantic tableaux)
• Normal Forms
• CNF: Conjunctive Normal Form
• NNF: Negative Normal Form
• Clausal form
• Resolution Principle
• Resolution Algorithm
• Simplifications
• Soundness, Completeness, Termination
• Resolution Strategies (brief)
Aside: Prolog Tableaux Construction

/*
Propositional Logic Semantic Tableaux
From Ben-Ari: Mathematical Logic for Computer Science (Appendix B)
(in /home/ta5/staff/alanw/PROLOG/semtab.pl)
*/

/* Define logical operators: */
:- op(650, xfy, '#').
:- op(640, xfy, '=>').
:- op(630, yfx, '¬').
:- op(620, yfx, 'v').
:- op(610, fy, '¬').

/* Top-level algorithm */

semantic_tableau(F) :- T = t(_, _, [F]),
extend_tableau(T),
write_tableau(T, 0).

extend_tableau(t(closed, empty, L)) :-
check_closed(L).
extend_tableau(t(open, empty, L)) :-
    contains_only_literals(L).

extend_tableau(t(Left, empty, L)) :-
    alpha_rule(L,L1),
    Left = t(_,_,L1),
    extend_tableau(Left).

extend_tableau(t(Left, Right, L)) :-
    beta_rule(L,L1,L2),
    Left = t(_,_,L1),
    Right = t(_,_,L2),
    extend_tableau(Left),
    extend_tableau(Right).

/* tableau extension */

check_closed(L) :-
    mymember(F,L), mymember( ~ F, L).

contains_only_literals([]).
contains_only_literals([F | Tail]) :-
    literal(F),
    contains_only_literals(Tail).
literal(F) :- atom(F).
literal(˜ F) :- atom(F).

alpha_rule(L, [A1, A2 | Ltemp]) :-
    alpha_formula(A,A1,A2),
    mymember(A,L),
    delete(A,L, Ltemp).
alpha_rule(L, [A1 | Ltemp]) :-
    A = ˜ ˜ A1,
    mymember(A,L),
    delete(A,L, Ltemp).

beta_rule(L, [B1 | Ltemp], [B2 | Ltemp]) :-
    beta_formula(B,B1,B2),
    mymember(B,L),
    delete(B,L, Ltemp).

alpha_formula(A1 ˆ A2, A1, A2).
alpha_formula(˜ (A1 => A2), A1, ˜ A2).
alpha_formula(˜ (A1 v A2), ˜ A1, ˜ A2).
alpha_formula(˜ (A1 # A2), ˜ (A1 => A2), ˜( A2 => A1) ).

beta_formula(A1 v A2, A1, A2).

Aside: Prolog Tableaux Construction
beta_formula(A1 => A2, ¬ A1, A2).
beta_formula(¬ (A1 ^ A2), ¬ A1, ¬ A2).
beta_formula(A1 # A2, A1 => A2, A2 => A1).

/* printing the tableau */

write_formula_list([F]) :- write(F).
write_formula_list([F | Tail]) :-
    write(F),
    write(','),
    write_formula_list(Tail).

write_tableau(empty,_).
write_tableau(closed,_) :-
    write(' Closed').
write_tableau(open,_) :-
    write(' Open').
write_tableau(t(Left, Right, List), K) :-
    nl, tab(K), K1 is K+3,
    write_formula_list(List),
    write_tableau(Left,K1),
    write_tableau(Right,K1).

Aside: Prolog Tableaux Construction
/* standard list operations */

mymember(X, [X | _]).
mymember(X, [_ | Tail]) :- mymember(X, Tail).

delete(X, [X | Tail], Tail).
delete(X, [Head | Tail], [Head | Tail1]) :- delete(X, Tail, Tail1).

/* Example: from above

semantic_tableau( ((p ^ q) ^ ~ q) ).

*/
Normal Forms

**DNF** (disjunctive normal form):

\[ D_1 \lor D_2 \lor \cdots \lor D_m \]

where \( D_i = L_1 \land L_2 \land \cdots \land L_n \)

**CNF** (conjunctive normal form):

\[ C_1 \land C_2 \land \cdots \land C_m \]

where \( C_i = L_1 \lor L_2 \lor \cdots \lor L_n \)

\( L_j \) is a literal, either \( A \) or \( \neg(A) \) for atomic propositional formula \( A \).

For literal \( L \), then \( \overline{L} \) is the **complement** of \( L \).

i.e. if \( L = A \), then \( \overline{L} = \neg(A) \)
Converting to CNF

- eliminate implication (‘→’): \((F \rightarrow G) \equiv (\neg(F) \lor G)\)

- ‘push in’ negations using De Morgan’s Laws:
  \[\neg(F \land G) \equiv (\neg(F) \lor \neg(G))\]
  \[\neg(F \lor G) \equiv (\neg(F) \land \neg(G))\]

- remove double negation (‘\neg\neg’): \(\neg(\neg(F)) \equiv F\)

- now formula is in **Negative Normal Form (NNF)**

- finally, remove conjunction within disjunction, via distributive law:
  \[(F \lor (G \land G_2)) \equiv ((F \lor G) \land (F \lor G_2))\]
Theorem 11  Every propositional formula $F$ can be converted into a CNF formula $F'$ such that $F$ is logically equivalent to $F'$, i.e.:

$$F \equiv F'$$
Example

\[ \neg ((A \land B) \rightarrow B) \]
Example

\[ \neg ((A \land B) \rightarrow B) \]
\[ \neg (\neg (A \land B) \lor B) \]
Example

\[ \neg((A \land B) \rightarrow B) \]
\[ \neg(\neg(A \land B) \lor B) \]
\[ \neg(\neg(A \land B)) \land \neg(B) \]
Example

\[ \neg ((A \land B) \rightarrow B) \]
\[ \neg (\neg (A \land B) \lor B) \]
\[ \neg (\neg (A \land B)) \land \neg (B) \]
\[ (A \land B) \land \neg (B) \]
datatype SENT = Prop of string | Not of SENT |
    And of (SENT * SENT) | Or of (SENT * SENT) | True | False;

fun Imp (x,y) = Or(Not(x),y); fun BiImp(x,y) = And(Imp(x,y),Imp(y,x));

fun nnf (Prop a) = Prop a  | nnf (Not (Prop a)) = Not (Prop a)
    | nnf (Not (Not a)) = nnf a
    | nnf (Not (And(a,b))) = nnf(Or(Not a, Not b))
    | nnf (Not (Or(a,b))) = nnf(And(Not a, Not b))
    | nnf (And(a,b)) = And(nnf a, nnf b)
    | nnf (Or(a,b)) = Or(nnf a, nnf b);

fun distrib (p, And(q,r)) = And(distrib(p,q),distrib(p,r))
    | distrib (And(q,r),p) = And(distrib(q,p),distrib(r,p))
    | distrib (p,q) = Or(p,q);

fun cnf(And(p,q)) = And(cnf(p),cnf(q))
    | cnf(Or(p,q)) = distrib(cnf(p),cnf(q)) | cnf(p) = p;

fun docnf(p) = cnf(nnf(p));
- **sent1;**
  \[\left(\left(\neg A \lor \neg B \right) \land (\neg A \land B) \right) \lor \left(\neg A \lor B \right) \lor \left(\neg A \land B \right) \lor \left(\neg A \lor B \right)\]

- **docnf(Not(sent1));**
  \[\left(\neg \left(\neg A \lor \neg B \right) \land \left(\neg A \land B \right) \right) \lor \left(\neg A \lor B \right) \lor \left(\neg A \land B \right) \lor \left(\neg A \lor B \right)\]

- **sent2**
  \[\left(\neg \left(\neg A \lor \neg B \lor C_2 \right) \right) \lor \left(\neg \left(\neg A \land B \right) \land \left(\neg A \lor C_2 \right) \right)\]

  \[\left(\left(\neg A \rightarrow (B \rightarrow C_2) \right) \lor \left(\neg \left(\neg A \land B \right) \right) \lor \left(\neg \left(\neg A \lor C_2 \right) \right)\right) \rightarrow \left(\left(\neg A \rightarrow B \right) \rightarrow \left(\neg A \rightarrow C_2 \right) \right)\]

- **docnf(Not(sent2));**
  \[\left(\neg \left(\neg A \lor \neg B \lor C_2 \right) \lor \left(\neg \left(\neg A \land B \right) \lor \left(\neg A \lor C_2 \right) \right)\right) \lor \left(\neg \left(\neg A \land B \right) \lor \left(\neg A \lor \neg C_2 \right) \right)\]
Clausal Form

**Clausal Formula**: Consider a (CNF) clause $L_1 \lor L_2 \lor \cdots \lor L_n$

‘$\lor$’ is associative, commutative, idempotent, so…

Represent clausal formula as a set of literals, or a **Clause**, $C$:

Notation: $[L_1; L_2; \ldots; L_n]$

A **Unit Clause** has a single literal: $[L]$

Write CNF as a set of clauses, $N$, in **Clausal Form**:

$$\{C_1, C_2, \cdots, C_m\}$$

where $C_i$ are clauses.

Let $CF(X)$ be the clausal form of a set of formulae $X$

An interpretation $\nu$ satisfies clause $C$ *iff* $\nu(L) = \text{true}$ for some $L$ in $C$.

Define $\nu$ on empty clause: $\nu([[]]) = \text{false}$, i.e. $[]$ is a contradiction.

**Note**: $\{[]\}$ is **unsatisfiable**, **BUT** $\{\}$ is **valid**
Example

\[ F = (A \land B) \land \neg(B) \]
\[ CF(F) = \{[A], [B], [\neg(B)]\} \]

\[ F = (((\neg A \land \neg B) \lor (\neg A \land B)) \lor (A \land \neg B)) \lor (A \land B) \]
\[ CF(\neg(F)) = \{[A; B], [A; \neg B], [\neg A; B], [\neg A; \neg B]\} \]

\[ F = ((A \rightarrow B) \land (B \rightarrow A)) \]
\[ CF(F) = \{[\neg(A); B], [\neg(B); A]\} \]
Resolution Principle

Based on:

\[((F \lor G) \land (G_2 \lor \neg (G))) \models (F \lor G_2)\]

$C_1$ and $C_2$ are **Clashing Clauses** if $L \in C_1$ and $\overline{L} \in C_2$.

For **Parent Clauses** $C_1, C_2$, their **Resolvent** is

\[Res(C_1, C_2) = (C_1 \setminus \{L\}) \cup (C_2 \setminus \{\overline{L}\})\]

**Theorem 14 Resolution Rule:** $C_1, C_2 \models Res(C_1, C_2)$
The Resolution Algorithm

Start with set of clauses $N_0$

Given set of clauses, $N_i$ at stage $i$:

- Choose a pair of clashing clauses, $C_1, C_2 \in N_i$
- Let $C = Res(C_1, C_2)$
- if $C = []$ then terminate ($N_0$ is unsatisfiable)
  
  else $N_{i+1} = N_i \cup \{C\}$

- if $N_{i+1} = N_i$ for all ways of choosing $C_1, C_2$
  then terminate ($N_0$ is satisfiable)
Example

(1) Let $F = (A \land B) \rightarrow B$ and test for $\models (A \land B) \rightarrow B$

$\neg F = (A \land B) \land \neg (B)$

$CF(\neg (F)) = \{[A], [B], [\neg (B)]\}$

1. $A$
2. $B$
3. $\neg (B)$
Example

(1) Let $F = (A \land B) \rightarrow B$ and test for $\models (A \land B) \rightarrow B$

$\neg F = (A \land B) \land \neg (B)$

$CF(\neg (F)) = \{[A], [B], [\neg (B)]\}$

1  $A$

2  $B$

3  $\neg (B)$

4  $Res(B, \neg (B)) = [\ ]$  2, 3
Example

(1) Let \( F = (A \land B) \rightarrow B \) and test for \( \models (A \land B) \rightarrow B \)

\( \neg F = (A \land B) \land \neg B \)

\( CF(\neg F) = \{[A], [B], [\neg B] \} \)

1  \( A \)
2  \( B \)
3  \( \neg B \)
4  \( Res(B, \neg B) = \[ \] \)

(2) \( (A \rightarrow B) \land (B \rightarrow C) \models (A \rightarrow C) \)

(3)  

| 1 | [A] |
| 2 | [\neg(A); B] |
| 3 | [\neg(A)] |

4a  | [B] | 1, 2 |
4b  | [ ] | 1, 3; terminate |

(4a) leaves \( \{ [B], [\neg(A)] \} \) with no clashing clauses.

\( S_0 \) satisfiable?? No! Need to backtrack and consider all other choices of clashing clauses (i.e. (4b)).
(4) Find clausal form for: \((A \land (A \to (B \lor C))) \models \neg A \to (\neg A \land B \land \neg C)\)

(5) Find clausal form for: \(F = ((A \to B) \land (B \to A))\)

(6) Check satisfiability of: \(F = (((\neg(A \land \neg(B)) \lor (\neg(A \land B))) \lor (A \land \neg(B))) \lor (A \land B))\)

(7) Check satisfiability of: \(F = ((A \to B) \land (B \to A))\)

(8) Prove: \((A \to B) \models (B \to A)\)

(9) Prove: \((A, A \to B) \models A)\)
Simplifications

Let $N \approx N'$ mean $N$ is satisfiable iff $N'$ is satisfiable.

**Lemma 1 – Purity Deletion:** If $L$ appears in $N$ but $\overline{L}$ is not in $N$.
Then delete all $C_i$ containing $L$ leaving $N'$.
Then $N \approx N'$.

**Lemma 2 – Unit Propagation:** If unit clause $[L] \in N$
Then delete all $C_i$ containing $L$, and delete $\overline{L}$ from remaining clauses, to leave $N'$.
Then $N \approx N'$.

**Lemma 3 – Tautology Deletion:** If a clause $C$ contains $L$ and $\overline{L}$
Then $N' = N \setminus \{C\}$.
Then $N \approx N'$.

**Lemma 4 – Subsumption Deletion:** If $C_1 \subseteq C_2$ then $C_1$ subsumes $C_2$.
Then $N' = N \setminus \{C_2\}$
Then $N \approx N'$.

[2, p.20],[3, p.67]
Example

\{ [A; B], [A_2], [A_2; B_2], [A; B; B_2], [\neg B, \neg A_2], [\neg A], [\neg A_3, A_3] \}
Soundness, Completeness, Termination

Let $N_0 = CF(X \cup \{\neg F\})$

**Refutation** of $N_0$ *iff* the resolution procedure derives $[]$ from $N_0$.

Call this $X \vdash_R F$

**Soundness** of $\mathcal{R}$ w.r.t. the semantics of $PC$: If $X \vdash_R F$ then $X \models F$.

Use the resolution procedure to decide if $X \cup \{\neg F\}$ is satisfiable, i.e. if $X \models F$.

**Completeness** of $\mathcal{R}$ w.r.t. the semantics of $PC$: If $X \models F$ then $X \vdash_R F$.

**Termination**: $\mathcal{R}$ terminates
Resolution Strategies

Start from $N_0$.

(1) Linear Resolution: each resolvent $R_{i+1} = \text{Res}(R_i, B_i)$, the centre clause, is obtained from the previous centre clause $R_i$ and a side clause, $B_i$, which is either taken from $N_0$ or is a previous centre clause.

Complete.

(2) Input Resolution:

A sub-case of linear resolution: the same but all side clauses now taken from $N_0$ (each element in $N_0$ is an input clause)

Easier to implement, more efficient, but not complete.
(3) Unit Resolution:
At least one parent clause is a unit clause.
Equivalent to input resolution

(4) Set-of-Support Resolution:
Let \( N_2 \subset N \) and \( N \setminus N_2 \) is satisfiable.
At least one parent clause must come from \( N_2 \).
Complete.
Example

Input Resolution (also Linear):

\{[\neg A; \neg B], [\neg A_2; \neg B_2; A], [B_2; \neg A_2], [B], [A_2]\}

1. \([\neg A; \neg B]\)
2. \([B_2; \neg A_2]\)
3. \([\neg A_2; \neg B_2; A]\)
4. \([B]\)
5. \([A_2]\)

6. \([\neg B; \neg B_2; \neg A_2]\) 1, 3(A)

7. \([\neg B; \neg A_2]\) 2, 6(B_2)

8. \([\neg A_2]\) 4, 7(B)

9. \([]\) 5, 8(A_2)
Lecture 3: First Order Predicate Logic (FOPL)
Introduction

1. The successor of each number is always greater than that number

2. There is a number greater than 3

Restrict numbers to the set \( \{1, \ldots, 5\} \).

1. \( gt\_2\_1 \land gt\_3\_2 \land gt\_4\_3 \land gt\_5\_4 \)

2. \( gt\_1\_3 \lor gt\_2\_3 \lor gt\_3\_3 \lor gt\_4\_3 \lor gt\_5\_3 \)
Write proposition $gt \cdot i, j$ as predicate $gt(i, j)$, where $gt : \mathbb{N} \times \mathbb{N} \rightarrow \mathbb{B}$.

1. $gt(2, 1) \land gt(3, 2) \land gt(4, 3) \land gt(5, 4)$
2. $gt(1, 3) \lor gt(2, 3) \lor gt(3, 3) \lor gt(4, 3) \lor gt(5, 3)$

and allow functions, e.g.

1. $gt(succ(1), 1) \land gt(succ(2), 2) \land gt(succ(3), 3) \land gt(succ(4), 4)$
2. $gt(1, 3) \lor gt(2, 3) \lor gt(3, 3) \lor gt(4, 3) \lor gt(5, 3)$

But still cannot (easily) express for all or for some in propositional logic (and impossible for infinite sets of objects)

So introduce quantifiers.
The Language $L$ of FOPL

$n$-ary predicate symbols (of arity $n$)

$n$-ary function symbols (of arity $n$)

constant symbols

variables $x \in \mathcal{X}$

**terms in $L$:**

- a constant or variable from $L$
- if $t_1, \ldots, t_n$ are terms in $L$, and $f_n$ is an $n$-ary function symbol in $L$, then $f_n(t_1, \ldots, t_n)$ is a term in $L$

**atomic formulae in $L$:**

- true, false are atomic formulae in $L$
- if $t_1, \ldots, t_n$ are terms in $L$, and $p_n$ is an $n$-ary predicate symbol in $L$, then $p_n(t_1, \ldots, t_n)$ is an atomic formula in $L$
formulae in $L$:

- atomic formulae from $L$ are formulae in $L$
- if $F, G$ are formulae and $x$ is a variable in $L$, then the following are formulae in $L$:
  - $(F \land G)$, $(F \lor G)$, $\neg(F)$, $(F \rightarrow G)$,
  - $\forall x \, F$ (universal quantification)
  - $\exists x \, F$ (existential quantification)

Variable Binding:

- $x$ is bound in $\forall x \, F$ or $\exists x \, F$.
- $F$ is the scope of $x$
- A variable which isn’t bound is free
Example

\[(p_1(x) \land \forall x \ (p_2(a, x) \land q_1(y)) \rightarrow (\exists y \ (r_2(x, y) \land \forall x \ \neg (q_3(x, z, b)))))) \]

\[(p_1(x) \land \forall x' \ (p_2(a, x') \land q_1(y)) \rightarrow (\exists y' \ (r_2(x', y') \land \forall x'' \ \neg (q_3(x'', z, b)))))) \]
Semantics of FOPL

Structure $\mathcal{M} = (D, R, F, C)$:

- domain $D$ (non-empty)
- $R$: assign $k$-ary relation $p^\mathcal{M}$ on $D$ to each $k$-ary predicate symbol $p$ of $L$;
- $F$: assign $k$-ary function $f^\mathcal{M}$ on $D$ to each $k$-ary function symbol $f$ of $L$;
- $C$: assign element $a^\mathcal{M}$ from $D$ to each constant symbol $a$ of $L$.

Assignment $s$ over $\mathcal{M}$: $s(x) \in D$ for each $x \in X$. 
**Values for terms**: $t$ is given value $t^M,s \in D$:

<table>
<thead>
<tr>
<th>Term in $L$</th>
<th>Value in $D$</th>
</tr>
</thead>
<tbody>
<tr>
<td>constant $a$</td>
<td>$a^M,s = a^M$</td>
</tr>
<tr>
<td>variable $x$</td>
<td>$x^M,s = s(x)$</td>
</tr>
<tr>
<td>$n$-ary function $f$</td>
<td>$f(t_1,t_2,\ldots,t_n)^M,s = f^M(t_1^M,s, t_2^M,s, \ldots, t_n^M,s)$</td>
</tr>
<tr>
<td></td>
<td>($t_1, \ldots, t_n$ are terms)</td>
</tr>
</tbody>
</table>
Truth values for formulae: $v_{\mathcal{M},s}(A) \in \{\text{true}, \text{false}\}$

<table>
<thead>
<tr>
<th>Logic Symbol</th>
<th>Truth Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>constant $t$</td>
<td>$v_{\mathcal{M},s}(t) = \text{true}$</td>
</tr>
<tr>
<td>constant $f$</td>
<td>$v_{\mathcal{M},s}(f) = \text{false}$</td>
</tr>
<tr>
<td>predicate</td>
<td>$v_{\mathcal{M},s}(p(t_1, \ldots, t_n)) = \text{true if and only if} (t_1^{\mathcal{M},s}, \ldots, t_n^{\mathcal{M},s}) \in p^{\mathcal{M}}$</td>
</tr>
<tr>
<td>connective (e.g.)</td>
<td>$v_{\mathcal{M},s}(F \land G) = \text{true if and only if} v_{\mathcal{M},s}(F) = \text{true and } v_{\mathcal{M},s}(G) = \text{true}$</td>
</tr>
<tr>
<td>quantifier $\forall$</td>
<td>$v_{\mathcal{M},s}(\forall xF') = \text{true if for all } d \in D, v_{\mathcal{M},s[x \mapsto d]}(F) = \text{true}$</td>
</tr>
<tr>
<td>quantifier $\exists$</td>
<td>$v_{\mathcal{M},s}(\exists xF') = \text{true if for some } d \in D, v_{\mathcal{M},s[x \mapsto d]}(F) = \text{true}$</td>
</tr>
</tbody>
</table>
If $v_{\mathcal{M},s}(F) = \text{true}$, write $\models_{\mathcal{M},s} F$

If $F$ is closed, then $v_{\mathcal{M},s}(F)$ is independent of $s$, so write $\models_{\mathcal{M}} F$

$\mathcal{M}$ is also a model for $F$.

A closed formula is **satisfiable** if it is true in *some* structure

A closed formula is **valid** if it is true in *all* structures: write $\models F$
Example

\[ A = \forall x \forall y (q(x, y) \rightarrow (p(x, y) \lor \exists z (p(x, z) \land q(z, y)))) \]

Structure \( M = \langle \text{people}, \{ q \mapsto \text{ancestor}, p \mapsto \text{parent} \}, 0, 0 \rangle \)

Any assignment \( s \)

- \( v_{M, s}(\forall x \forall y (q(x, y) \rightarrow (p(x, y) \lor \exists z (p(x, z) \land q(z, y)))))) = \text{true} \) iff
- for all \( d \in D \), \( v_{M, s[x^\rightarrow d]}(\forall y (q(x, y) \rightarrow (p(x, y) \lor \exists z (p(x, z) \land q(z, y)))))) = \text{true} \) iff
- for all \( d \in D \), for all \( d' \in D \), \( v_{M, s[x^\rightarrow d][y^\rightarrow d']} (q(x, y) \rightarrow (p(x, y) \lor \exists z (p(x, z) \land q(z, y)))) = \text{true} \) iff
- for all \( d \in D \), for all \( d' \in D \), if \( v_{M, s[x^\rightarrow d][y^\rightarrow d']} (q(x, y)) = \text{true} \) then \( v_{M, s[x^\rightarrow d][y^\rightarrow d']} (p(x, y) \lor \exists z (p(x, z) \land q(z, y))) = \text{true} \) iff
- for all \( d \in D \), for all \( d' \in D \), if \( (d, d') \in \text{ancestor} \) then either \( v_{M, s[x^\rightarrow d][y^\rightarrow d']} (p(x, y)) \) or \( v_{M, s[x^\rightarrow d][y^\rightarrow d']} (\exists z (p(x, z) \land q(z, y))) \) iff
- for all \( d \in D \), for all \( d' \in D \), if \( (d, d') \in \text{ancestor} \) then either \( (d, d') \in \text{parent} \) or there exists a \( d'' \in D \), such that \( v_{M, s[x^\rightarrow d][y^\rightarrow d'][z^\rightarrow d'']} (p(x, z) \land q(z, y)) = \text{true} \) iff
- for all \( d \in D \), for all \( d' \in D \), if \( (d, d') \in \text{ancestor} \) then either \( (d, d') \in \text{parent} \) or there exists a \( d'' \in D \), such that \( v_{M, s[x^\rightarrow d][y^\rightarrow d'][z^\rightarrow d'']} (p(x, z)) = \text{true} \) and \( v_{M, s[x^\rightarrow d][y^\rightarrow d'][z^\rightarrow d'']} (q(z, y)) = \text{true} \) iff
- for all \( d \in D \), for all \( d' \in D \), if \( (d, d') \in \text{ancestor} \) then either \( (d, d') \in \text{parent} \) or there exists a \( d'' \in D \), such that \( (d, d'') \in \text{parent} \) and \( (d'', d') \in \text{ancestor} \) iff
- for all people \( d \) and \( d' \), if \( d \) is an ancestor of \( d' \), then either \( d' \) is a parent of \( d \), or there exists another person \( d'' \) such that \( d'' \) is a parent of \( d \) and \( d'' \) is an ancestor of \( d' \).
- which is ‘clearly’ true, since \text{ancestor} is the transitive closure of \text{parent}.
Lecture 4: Resolution for Predicate Logic
Predicate Resolution: Plan

- Proof by refutation again
- Normal Forms (again!)
- CNF, NNF, Clausal form (again!)
- Prenex CNF (new)
- Preclausal Form (new)
- Existential quantifier elimination: Skolemisation
- Substitution
- Unification
- Resolution Principle (again!)
- Resolution Algorithm (again!)
- Soundness, Completeness, Termination
Normal Forms

Prenex conjunctive normal form:

\[ F = Q_1x_1 \cdots Q_kx_kM \]

where

- \( F \) is closed
- \( Q_i \) is a quantifier,
- \( M \) is a formula in CNF (quantifier-free), the matrix of \( F \)
- free variables \( x_1, \ldots, x_k \) of \( M \)

Preclausal form: prenex conjunctive normal form and \( Q_i \) are all universal quantifiers.

Just use \( M \) to represent the universal closure of \( M \).

Clausal form: preclausal form + write \( M \) as a set of clauses (it's in CNF)
Conversion to Clausal Form

- rename bound variables apart
- rewrite all logical connectives (e.g. $\rightarrow$) using $\land$ and $\lor$ (propositional)
- move $\neg$ inward (propositional)
- move all quantifiers out to the front (see Kelly)
- put the matrix into CNF using distributive laws (propositional)
Existential Quantifier Elimination

... by Skolemisation

Consider prenex clausal form: $Q_1x_1 \cdots Q_kx_kM$

- Choose leftmost $\exists_{n+1}x_{n+1}$
- Create a new $n$-ary function symbol ‘$f_n$’
- Replace occurrences of $x_{n+1}$ in $M$ by ‘$f_n(x_1, \ldots, x_n)$’ (a Skolem function)
- Remove $\exists_{n+1}x$

If $\exists_1x_1$ is first, then use a new constant symbol $c$ (a Skolem constant)

Theorem 18 (Skolem) There is a purely syntactic procedure which, given a closed formula $F$, produces a formula $F'$ which is in preclausal form such that $F$ is satisfiable if and only if $F'$ is satisfiable.
Substitution

\[ \sigma = \{x_1/t_1, x_2/t_2, \ldots, x_n/t_n\} \]

where

variable \( x_i \in X \)

terms \( t_i (t_i \neq x_i) \)

\('t\sigma' means simultaneously replace each \( x_i \) in \( t \) with \( t_i \).

Use \( \sigma, \theta, \mu \) for substitutions ...

Aside: Notation Alert!

Note: \('x/t' means \( x \) is substited by \( t \) (not \( t \) substituted by \( x \))

Also, we have used \('x_1 \leftarrow t_1' (e.g. in G&P)\)
Composing Substitutions:

Let $\theta = \{x_1/s_1, \ldots, x_n/s_n\}$, $\sigma = \{y_1/t_1, \ldots, y_m/t_m\}$

Want: $(t\theta)\sigma = t(\theta \circ \sigma)$ [2, Theorem 16]

$\theta \circ \sigma =$

$\{x_1/s_1\sigma, \ldots, x_n/s_n\sigma\}$

$\setminus \{x_1/x_1, \ldots, x_n/x_n\}$

$\cup \sigma'$

$\sigma'$ is obtained from $\sigma$ by removing any substitutions $x_i/s_i'$, where $x_i$ appears in $\theta$

(i.e. Using ‘domain subtraction’ operation ‘$\triangleleft$’: $\sigma' = \{x_1, \ldots x_n\} \triangleleft \sigma$)
Example

Let $\theta = \{w/v, x/f(z), y/g(x), z/y\}$ and $\sigma = \{v/w, z/a, y/b, x/h(v)\}$
and let $t = f(v, w, x, y, z)$

\[
\begin{align*}
 t\theta &= f(v, v, f(z), g(x), y) \\
 (t\theta)\sigma &= f(w, w, f(a), g(h(v)), b) \\
 \theta' &= \{w/v\sigma, x/f(z)\sigma, y/g(x)\sigma, z/y\sigma\} \\
 &= \{w/w, x/f(z)\sigma, y/g(x)\sigma, z/y\sigma\} \\
 &= \{x/f(a), y/g(h(v)), z/b\} \\
 \sigma' &= \{w, x, y, z\} \triangleleft \sigma \\
 &= \{v/w\} \\
 \theta \circ \sigma &= \theta' \cup \sigma' \\
 &= \{v/w, x/f(a), y/g(h(v)), z/b\} \\
 t(\theta \circ \sigma) &= f(w, w, f(a), g(h(v)), b)
\end{align*}
\]
\[ A = \]
\[
( p_1(x) \land \\
( \forall x \\
( p_2(a, x) \land q_1(y)) \rightarrow \\
( \exists y \\
( r_2(x, y) \land \\
( \forall x \\
( \neg (q_3(x, z, b)))))))) \\
\]

\[ A\{x/f(z), y/g(x), z/y\} = \]
\[
( p_1(f(z)) \land \\
( \forall x' \\
( p_2(a, x') \land q_1(g(x))) \rightarrow \\
( \exists y' \\
( r_2(x', y') \land \\
( \forall x'' \\
( \neg (q_3(x'', y, b)))))))) \\
\]
Unification

Terms $s, t$ are **unifiable** if there is a $\theta$ so that $s\theta = t\theta$.

**Most General Unifier (MGU):**

An MGU makes the ‘least number of changes’

A unifier $\theta$ is a MGU for terms $s, t$ if, for all unifiers $\sigma$ of $s, t$ then $\sigma = \theta \circ \mu$ for some substitution $\mu$.

**Theorem 17**  *If two terms are unifiable then they have a most general unifier.*
Algorithm for Computing MGUs:

unify(s, t):
    if s = t then return 0
    let s₁, t₁ = disagreement pair of s, t
    if s₁, t₁ are variables then from = s₁, to = t₁
    if s₁ is a variable and s₁ ∉ Vars(t₁) then from = s₁, to = t₁
    if t₁ is a variable and t₁ ∉ Vars(s₁) then from = t₁, to = s₁
    else return fail
    ret = unify(s{from/to}, t{from/to})
    if ret = µ (a MGU) then return {from/to} ◦ µ
    if ret = fail then return fail

Vars(t₁) returns the set of variables in t₁
‘s₁ ∉ Vars(t₁)’ is the Occurs Check
Example

1. $\text{unify}(f(a, g(x)), f(x, g(y)))$
2. $\text{unify}(f(x, g(x)), f(y, g(h(y))))$
3. $\text{unify}(f(a, g(x)), f(x', g(y)))$
4. $\text{unify}(f(x, g(z)), f(y, g(a)))$
First Order Resolution

... at last ...

**Clashing Clauses** $C_1, C_2$:

$L_1 \in C_1, L_2 \in C_2$ and $L_1, L_2$ have a MGU $\theta$

Assume $C_1, C_2$ have no variables in common (otherwise, rename variables)

Suppose clash on $L_1, L_2$ with MGU $\theta$.

**A Binary Resolvent** of $C_1, C_2$:

$$(C_1 \theta - \{L_1 \theta\}) \cup (C_2 \theta - \{L_2 \theta\})$$

**Factoring** necessary for completeness of FOR.

$$\{L_1, L_2, \cdots, L_n\} \subseteq C$$ such that $\{L_1, L_2, \cdots, L_n\}$ has an MGU $\theta$:

$C \theta$ is a **factor** of $C$.

**A Resolvent** of $C_1, C_2$ is a binary resolvent of $C_1', C_2'$ (where $C_i'$ may be a factor of $C_i$)
Resolution Procedure

(similar to propositional case)

Given set of clauses $N_i$:

- **Choose** a pair of clashing clauses, $C_1, C_2 \in N_i$ (rename variables apart)
- Let $C = Res(C_1, C_2)$
- if $C = [\ ]$ then terminate ($N_0$ is unsatisfiable)
  - else $N_{i+1} = N_i \cup \{C\}$
- if $N_{i+1} = N_i$ for all ways of choosing $C_1, C_2$
  (and the clashing literal) then terminate ($N_0$ is satisfiable)
Example

\[ F = (\forall x (A(x) \rightarrow B(x))) \rightarrow ((\exists x A(x)) \rightarrow (\exists x B(x))) \]

\[ CF(\neg F) = \{ [\neg A(x); B(x)], [A(a)], [\neg B(x)] \} \]

1. \[ [\neg A(x); B(x)] \]
2. \[ [A(a)] \]
3. \[ [\neg B(x)] \] (standardise apart first)
4. \[ [\neg A(x)] \] 1, 3(B)
5. \[ [ ] \] 2, 4(A), \{x/a\}
Soundness, Completeness, Termination

**Soundness**: If $X \not\vdash_R A$ (i.e. there is a refutation (with factoring) of $X \cup \{\neg A\}$) then $X \models A$.

**Completeness**: If $X \models A$ then $X \vdash_R A$.

i.e. if $X \not\vdash_R A$ then $X \not\models A$.

**Termination**:

Resolution is a **semi-decision** procedure.

Terminates if $X \models A$, but may not terminate if $X \not\models A$. 
Herbrand Models

Set of clauses: \( S \), containing...

- Set of constant symbols: \( C \);
- Set of function symbols: \( F \)

The **Herbrand Universe** \( H_{S} \) of \( S \):

- for \( a \in C \): \( a \in H_{S} \)
- for \( f \in F \): \( f(t_{1}, \ldots, t_{n}) \in H_{S} \), with \( t_{i} \in H_{S} \)

If there are no constants, then include an arbitrary constant symbol \( a \).

**Herbrand Base** \( B(S) \): the set of all ground atoms formed from predicate symbols in \( S \) and \( H_{S} \).

**Herbrand Interpretation**: a subset of the Herbrand base, containing ground atoms assumed to be satisfied.

**Herbrand Model** for \( S \) is a Herbrand interpretation which satisfies \( S \).
Example

\[ S = \{ [p(a)], [q(b)], [r(c)], [\neg q(x), p(x)], [\neg p(y), r(y)] \} \]

\[ H(S) = \{ a, b, c \} \]

\[ B(S) = \{ p(a), q(a), r(a), p(b), q(b), r(b), p(c), q(c), r(c) \} \]

\[ \text{model} = \{ p(a), p(b), q(a), q(b), r(a), r(b), r(c) \} \]

\[ \text{interpretation} = \{ p(a), p(b), q(a), q(b), r(a), r(c) \} \]

(but not a model)

Q: What use is all this?

**Theorem** \( S \) has a model iff it has a Herbrand model

**Theorem** If \( S \) is unsatisfiable then some finite set of ground clauses of \( S \) is unsatisfiable
Example

\[ A = (\forall x(p(x) \rightarrow q(x))) \rightarrow ((\forall x p(x)) \rightarrow (\forall x q(x))) \]

\[ CF(\neg A) = \{[\neg p(x), q(x)], [p(y)], [\neg q(a)]\} \]

Ground Clauses with \{x/a, y/a\}:

\[ \{[\neg p(a), q(a)], [p(a)], [\neg q(a)]\} \]
Lecture 5: Prolog
The Basics

Terms:
- constants, variables
- compound: functors: name(arg₁, . . . , argₖ)

Ground terms: terms with no variables

Clauses:
- Rules: Head :- Goal₁, . . . , Goalₖ.
- Facts: Head.
  i.e. a rule without any goals or body
- Goals: Goal₁, . . . , Goalₖ.
  i.e. a rule without a head.
Some Examples

parent(john, juliet).
Some Examples

parent(john, juliet).

parent(john, sue, juliet).
Some Examples

parent(john, juliet).

parent(john, sue, juliet). (no. of arguments)
Some Examples

parent(john, juliet).

parent(john, sue, juliet).  (no. of arguments)

:- parent(john, X).
Some Examples

```prolog
parent(john, juliet).
parent(john, sue, juliet). (no. of arguments)

:- parent(john, X).

parent(X, juliet).
```
Some Examples

parent(john, juliet).

parent(john, sue, juliet). (no. of arguments)

:- parent(john, X).

parent(X, juliet).

greater_than(succ(X), zero).
Procedure: rules with same Head name

ancestor(X,Y) :- mother(X,Y).
ancestor(X,Y) :- father(X,Y).
ancestor(X,Y) :- aunt(X,Y).
...

Meaning of a Rule:
If \( \text{Goal}_1 \) and \( \text{Goal}_2 \) and \( \ldots \) and \( \text{Goal}_k \) all hold, then \( \text{Head} \) holds.

Program: a list of clauses

The meaning of a Prolog Program \( P \):
the set of ground goals deducible from \( P \)
Another Example

1: ancestor(X,Y) :- father(X,Y).
2: father(X,Y) :- parent(X,Y), male(X).
3: parent(john,juliet).
4: male(john).
Another Example

1: ancestor(X, Y) :- father(X, Y).
2: father(X, Y) :- parent(X, Y), male(X).
3: parent(john, juliet).
4: male(john).

:- ancestor(john, juliet).
Another Example

1: ancestor(X,Y) :- father(X,Y).
2: father(X,Y) :- parent(X,Y), male(X).
3: parent(john,juliet).
4: male(john).

:- ancestor(john,juliet).
(1) :- father(john,juliet).
Another Example

1: ancestor(X,Y) :- father(X,Y).
2: father(X,Y) :- parent(X,Y), male(X).
3: parent(john,juliet).
4: male(john).

:- ancestor(john,juliet).
(1) :- father(john,juliet).
(2) :- parent(john,juliet),male(john).
Another Example

1: ancestor(X, Y) :- father(X, Y).
2: father(X, Y) :- parent(X, Y), male(X).
3: parent(john, juliet).
4: male(john).

:- ancestor(john, juliet).

(1) :- father(john, juliet).
(2) :- parent(john, juliet), male(john).
(3,4) Yes
Now with recursion:

1: ancestor(X, Y) :- parent(X, Y).
Now with recursion:

1: \text{ancestor}(X, Y) :- \text{parent}(X, Y).

2: \text{ancestor}(X, Y) :- \text{parent}(X, Z), \text{ancestor}(Z, Y).
Now with recursion:

1: \text{ancestor}(X, Y) :- \text{parent}(X, Y).
2: \text{ancestor}(X, Y) :- \text{parent}(X, Z), \text{ancestor}(Z, Y).
3: \text{parent}(\text{chaz}, \text{john}).
4: \text{parent}(\text{john}, \text{juliet}).
Now with recursion:

1: ancestor(X, Y) :- parent(X, Y).
2: ancestor(X, Y) :- parent(X, Z), ancestor(Z, Y).
3: parent(chaz, john).
4: parent(john, juliet).

:- ancestor(chaz, juliet).
Now with recursion:

1: \( \text{ancestor}(X, Y) :\!- \text{parent}(X, Y) \).
2: \( \text{ancestor}(X, Y) :\!- \text{parent}(X, Z), \text{ancestor}(Z, Y) \).
3: \( \text{parent}(\text{chaz}, \text{john}) \).
4: \( \text{parent}(\text{john}, \text{juliet}) \).

\[
\vdash \text{ancestor}(\text{chaz}, \text{juliet}).
\]

(2) \( \vdash \text{parent}(\text{chaz}, Z), \text{ancestor}(Z, \text{juliet}) \).
Now with recursion:

1: ancestor(X, Y) :- parent(X, Y).
2: ancestor(X, Y) :- parent(X, Z), ancestor(Z, Y).
3: parent(chaz, john).
4: parent(john, juliet).

(2) :- ancestor(chaz, juliet).
(3) :- parent(chaz, Z), ancestor(Z, juliet).
(3) :- parent(chaz, john), ancestor(john, juliet).
Now with recursion:

1: ancestor(X, Y) :- parent(X, Y).
2: ancestor(X, Y) :- parent(X, Z), ancestor(Z, Y).
3: parent(chaz, john).
4: parent(john, juliet).

:- ancestor(chaz, juliet).
(2) :- parent(chaz, Z), ancestor(Z, juliet).
(3) :- parent(chaz, john), ancestor(john, juliet).
(2) :- ancestor(john, juliet).
Now with recursion:

1: ancestor(X, Y) :- parent(X, Y).
2: ancestor(X, Y) :- parent(X, Z), ancestor(Z, Y).
3: parent(chaz, john).
4: parent(john, juliet).

:- ancestor(chaz, juliet).

(2) :- parent(chaz, Z), ancestor(Z, juliet).
(3) :- parent(chaz, john), ancestor(john, juliet).
(2) :- ancestor(john, juliet).
(1) :- parent(john, juliet).
Now with recursion:

1: ancestor(X, Y) :- parent(X, Y).
2: ancestor(X, Y) :- parent(X, Z), ancestor(Z, Y).
3: parent(chaz, john).
4: parent(john, juliet).

:- ancestor(chaz, juliet).

(2) :- parent(chaz, Z), ancestor(Z, juliet).
(3) :- parent(chaz, john), ancestor(john, juliet).
(2) :- ancestor(john, juliet).
(1) :- parent(john, juliet).
(4) Yes
Multiple Solutions

:- ancestor(chaz, X).
Multiple Solutions

:- ancestor(chaz, X).

(3) :- parent(chaz, john)
Multiple Solutions

:- ancestor(chaz,X).

(3) :- parent(chaz,john)
    X = john
Multiple Solutions

\[\text{- ancestor(chaz,X).}\]

\[\text{(3) :- parent(chaz,john)}\]
\[X = \text{john ;}\]
Multiple Solutions

\[\text{:- ancestor(chaz, X).}\]

\[(3) \quad \text{:- parent(chaz, john)}\]
\[X = \text{john ;}\]

\[(2) \quad \text{:- parent(chaz, Y), ancestor(Y, X).}\]
Multiple Solutions

:- ancestor(chaz,X).

(3) :- parent(chaz,john)
    X = john ;

(2) :- parent(chaz,Y),ancestor(Y,X).
(3) :- parent(chaz,john),ancestor(john,juliet).
(2) :- ancestor(john,juliet).
(1) :- parent(john,juliet).
Multiple Solutions

:- ancestor(chaz,X).

(3) :- parent(chaz,john)
    X = john ;

(2) :- parent(chaz,Y),ancestor(Y,X).
(3) :- parent(chaz,john),ancestor(john,juliet).
(2) :- ancestor(john,juliet).
(1) :- parent(john,juliet).
    X = juliet
Multiple Solutions

:- ancestor(chaz,X).

(3) :- parent(chaz,john)
    X = john ;

(2) :- parent(chaz,Y),ancestor(Y,X).
(3) :- parent(chaz,john),ancestor(john,juliet).
(2) :- ancestor(john,juliet).
(1) :- parent(john,juliet).
    X = juliet

Search Strategy: left-to-right, top-to-bottom (but see later… )
Arithmetic, Equality

Built-in predicates which perform evaluation:

operators: +, *, -, /

comparison: <, >, <=, >=

equality: =, \=

\[ \begin{align*}
X &= 2 \times 3 \times 7 \\
42 &= 2 \times 3 \times 7
\end{align*} \]

invoke evaluation: 42 is 2 * 3 * 7

X is 2 * 3 * 7
Lists

Notation: \([ \text{val}_1, \ldots, \text{val}_k ]\)

Empty: \([ \ ]\)

Cons: \([1|2,3]\) (cf. Lisp, SML)

\[
\text{length}([ \ ], 0).
\]
\[
\text{length}([X|Y], N) \leftarrow \text{length}(Y, N_1), N \text{ is } N_1+1.
\]

(consider: \(\text{length}([X|Y], N) \leftarrow N \text{ is } N_1+1, \text{length}(Y, N_1).\))
Fail, Cut

- **fail**: a predicate that *always* fails (what use is that?)
- **cut**: denoted by ‘!’; a predicate that *always* succeeds. Its side effects alter back-tracking, and possibilities to re-try satisfying previous goals *(see later).*
Lecture 5: Logic Programming
Logic Programming: Plan

- Horn Clauses
- Resolution with Horn Clauses
- Prolog
- Search Strategy: SLD-Resolution and SLD-Trees
- Cut, Fail, Negation-as-Failure
Horn Clauses

As usual, some definitions…

**positive literal**: atomic formula $p(t_1, \ldots, t_n)$

**negative literal**: negated atomic formula $\neg p(t_1, \ldots, t_n)$.

**Horn clause**: a clause containing *at most one* positive literal e.g. $[H_1; \neg B_2; \ldots \neg B_n]$

**Definite clause**: a Horn clause with *exactly one* positive literal.

$[H_1; \neg B_2; \ldots \neg B_n]$

**Fact**: a definite clause with *no* negative literals. $[H_1]$

**Goal clause**: a Horn clause with *no positive* literals. $[\neg B_1; \ldots \neg B_n]$

Set of Horn clauses $X$

Goal clause $\mathcal{G}$.

Write a Horn clause as $H_1 \leftarrow B_2, \ldots, B_n$
Resolution with Horn Clauses

Apply linear input resolution: $\mathcal{G}$ as initial centre clause.

- Choose
  1. a negative literal $\neg G_i \in \mathcal{G}$
  2. a clause $C^1 \in X$ with $C^1 = [H^1; \neg B^1_1; \ldots; \neg B^1_{n_1}]$

... so that $\neg G_i$ and $H^1$ clash: compute $\theta_1 = unify(G_i, H^1)$.

(first renaming apart common variables) Success? Returns MGU $\theta_1$.

- New centre clause:

$$\mathcal{G}' = (\mathcal{G}\theta_1 - \neg G_i\theta_1) \cup [\neg B^1_1\theta_1; \ldots; \neg B^1_{n_1}\theta_1]$$
Now compute $G''$ from a negative literal in $G'$ and a (positive) head $H^2$ of some clause $C^2 \in X$.

i.e. compute: $G, G', G'', \ldots$ and MGUs $\theta_1, \theta_2, \ldots$ until empty clause is reached: $X \cup \{G\}$ is unsatisfiable.

**Computed Answer Substitution:** $\theta = \theta_1 \circ \theta_2 \circ \ldots \circ \theta_n$

(restricted to free variables in goal)

Two possible choices at each iteration:

- negative literal $\neg G_i$ from centre clause
- clause $C \in X$, with clashing head $H$ to resolve with $\neg G_i$

If no clashes with one set of choices then *backtrack* and try with other choices (if available).

If there are no choices, then fail.
Prolog

Notation

variables: begin with capital letter: X, Y, Answer

constants, functors, predicates: begin with lower-case letter:
parent/2, john, chaz

definite clauses, or rules: H :- B_1, ... , B_n

procedure: sequence of rules with same head

fact: single positive literals, with no body

goal: headless rule, just the body: :- B_1, ... , B_n

Body variables are existentially quantified; head variables are universally quantified over the rule.

Also, built-in operations,
e.g. list [a, b, c], empty list [], add element to list [X | Xs]

Arithmetic evaluation: X is X+1
Search Strategy: SLD-Resolution

SLD-Resolution: **Select literal, Linear resolution, Definite clauses**

(1) **breadth-first** or (2) **depth-first**.

(1) will guarantee to find a finite resolution refutation, but (2) more efficient…

Prolog performs a **depth-first** search, matching rules from **top-to-bottom**, and resolving goal clauses from **left-to-right**.
Example

(1) ancestor(X, Y) :- parent(X, Y).
(2) ancestor(X, Y) :- parent(X, Z), ancestor(Z, Y).

(3) parent(jim, roy).  (4) parent(john, juliet).
(5) parent(roy, sue).  (6) parent(roy, alan).  (7) parent(chaz, john).
(8) parent(sue, toby).  (9) parent(sue, juliet).

(G) :- ancestor(P, juliet).

1. Resolve ancestor(P, juliet) with rule (1): with \( \theta_1 = \{P/X, Y/juliet\} \),
   to yield \( G_1 = :- \) parent(X, juliet).

2. Resolve :- parent(X, juliet) with fact (4), with \( \theta_2 = \{X/john\} \), to yield empty clause

So... the answer substitution is \( \{P/X\} \circ \{X/john\} = \{P/john\} \)
i.e. john is an ancestor of juliet
Example

:- op(650,xfy, '##'). :- op(640,xfy, '=>'). :- op(630,yfx, '^').
:- op(620,yfx, 'v'). :- op(610, fy, '˜').

semantic_tableau(F) :- T = t(_, _, [F]), extend_tableau(T), write_tableau(T,0).

extend_tableau(t(closed, empty, L)) :- check_closed(L).
extend_tableau(t(open, empty, L)) :- contains_only_literals(L).
extend_tableau(t(Left, empty, L)) :-
    alpha_rule(L,L1), Left = t(_,_,L1), extend_tableau(Left).
extend_tableau(t(Left, Right, L)) :- beta_rule(L,L1,L2), Left = t(_,_,L1),
    Right = t(_,_,L2), extend_tableau(Left), extend_tableau(Right).

check_closed(L) :- member(F,L), member(˜ F, L).
contains_only_literals([]).
contains_only_literals([F | Tail]) :- literal(F), contains_only_literals(Tail).

literal(F) :- atom(F). literal(˜ F) :- atom(F).

alpha_rule(L, [A1, A2 | Ltemp]) :-

beta_rule(L, [B1 | Ltemp], [B2 | Ltemp]) :-
    beta_formula(B,B1,B2), member(B,L), delete(B,L, Ltemp).

alpha_formula(A1 ^ A2, A1, A2). alpha_formula(˜ (A1 => A2), A1, ˜ A2).
alpha_formula(˜ (A1 v A2), ˜ A1, ˜ A2).
alpha_formula(˜ (A1 # A2), ˜ (A1 => A2), ˜ { A2 => A1 }).

Search Strategy: SLD-Resolution (Continued)

member(X, [X | _]).
member(X, [_ | Tail]) :- member(X, Tail).

delete(X, [X | Tail]).
delete(X, [Head | Tail], [Head | Tail]) :- delete(X, Tail, Tail).

write_formula_list([F]) :- write(F).
write_formula_list([F | Tail]) :- write(F), write(','), write_formula_list(Tail).

write_tableau(empty, _). write_tableau(closed, _) :- write(' Closed').
write_tableau(open, _) :- write(' Open').
write_tableau(t(Left, Right, List), K) :- nl, tab(K), K1 is K+3,
    write_formula_list(List), write_tableau(Left, K1), write_tableau(Right, K1).
An **SLD-Tree** of goal $G_0$, with respect to a program $P$ and computation rule $R$ is a labelled tree:

- $G_0$ at root
- $G_{i+1}$ is a child of $G_i$ if it is a resolvent of $G_i$ and some clause $C_i$ from $P$, under $R$.
- a branch is (finitely) **failed** if there is no resolvent
- a branch is **closed** if the resolvent is empty
- a branch may be **infinite**
• **Cut**: prune SLD-Tree — don’t backtrack and search alternatives after failure.
   Insert ‘!’ goal into clause body.
   ‘!’ succeeds and *forces* all choices since containing clause was unified with parent goal.
   It’s ok to prune failed branches, but not (necessarily) ok to prune success branches: ‘green’ and ‘red’ cuts.

• **Fail**: a goal that *always* fails.
Negation as Failure: if search for resolution of $G$ finitely fails then conclude not $G$.
('safe' if $G$ is ground, but 'floundering' if $G$ is non-ground).

$$\text{disjoint}(Xs,Ys) :- \text{not} \ (\text{member}(X,Xs), \text{member}(X,Ys)).$$

$$\text{teacher}(\text{alan}). \quad \text{student}(\text{ashley}).$$
$$\text{takes\_courses}(X) :- \text{student}(X).$$

(1) :- not \ takes\_courses(\text{alan}).
(2) :- not \ takes\_courses(Y).

not $X$ :- $X$, !, fail.
not $X$. 

Search Strategy: SLD-Resolution (Continued)
The Handout of Handouts

• Course:
  – ‘Introduction and Pre-Course Work’
  – Post-Course Assignment Details

• Advanced Topics (Renate)
  – Lecture notes
  – Exercise and Laboratory notes

• Resolution Part:
  – ‘Notes on Automated Reasoning’ (Goré + Peim)
  – Outline solutions to selected exercises
  – Lecture slides
  – Laboratory exercises
  – ‘Resolution’ exercises (solutions later)
Assessed Work and Deadlines

- **Exam (40%)**: Open-book, 2 hours. Two Parts. Attempt three out of four questions in each Part.

- **Assessed Work in Teaching Week (30%)**:  
  - ‘Advanced Topics’ Laboratories and Exercises (0.5x30%) deadline: end of Teaching Week (labs); Friday, 18th November 2005 (exercises).
  - Prolog Laboratories (0.3x30%) deadline: end of Teaching Week. Please ensure you email your exercises to alanw@cs.man.ac.uk.
  - Resolution Exercises (0.2x30%) deadline: Friday, 18th November 2005

- **Post-Course Assignment (30%)**: deadline Friday 9th December, 2005. Choose topic preferences ASAP (latest by Wednesday 16th November, 2005 please!)

Work to be submitted to Post-Graduate Office, as usual
Exam: Part One

It will not contain questions on:

- Kripke structures
- Temporal Logic
- Predicate Logic semantic tableaux
- Binary Decision Diagrams

(i.e. which have appeared in some past papers)

The exam may contain questions on topics covered in both the Pre-Course week and the Teaching Week.

Even though the exam is ‘Open Book’, you need to have a good knowledge of the topics covered, and in particular be proficient and up-to-speed in applying them.
Highlights

- deductive reasoning system vs meaning of formula
- meta-language vs object-language
- $\vdash$ vs $\models$
- soundness and completeness
- termination: decision procedures
- $F$ is satisfiable if there is some interpretation for which evaluates to true.
- Model: a satisfying interpretation
- Validity: true in every interpretation
- refutation: $X \models F$ iff $X \cup \neg F$ is unsatisfiable
- If $\neg F$ is unsatisfiable then $F$ is valid
(Propositional Logic) Semantic tableaux:

- Prove $F$ by refuting $\neg F$: set root node label to be $\neg F$
- if parent is satisfiable then at least one child is.
- if all branches are closed then root formula is unsatisfiable
- counter-example generated from interpretation of atoms in open branch(es)

(Propositional Logic) Resolution:

- CNF, Clausal Form (CF): translation preserves equivalence
- Empty clause $[\ ]$ is invalid (but empty CF $\{\ }$ is valid)
- CF containing empty clause $\{C_1, \ldots, [\ ], \ldots, C_k\}$ is therefore invalid
- Proof by refutation: look for empty clause
- Resolution algorithm
- Choose Clashing Clauses, containing complementary pair of literals
• Always terminates (finite number of clashing clauses);
• Simplifications preserve satisfiability \((N \approx N')\), possibly using different interpretations.
• sound and complete (assuming *systematic* rule application), but possible non-termination
(Predicate Logic) Resolution:

- Consider closed formula only (in this course)
- Translate into Prenex CNF: rename bound variables (where possible); bring quantifiers to front (put existential quantifiers first if possible)
- Remove existential quantifiers using Skolem functions and constants
- Clausal Form: Matrix with free variables implicitly quantified
- Look for Clashing Clauses (cf propositional resolution); Unification between literals in pair of clauses (standardise variables apart first!)
- Factoring: look for unifiable subset of literals within same clause (do not standardise variables apart!)
- sound and complete (with factoring), but possible non-termination
Unification:

- Composition of substitution: defined to be associative \(((t \theta) \sigma = t(\theta \circ \sigma))\).
- ‘Occurs Check’ to avoid infinite iteration
- construct Most General Unifier
- Unification Algorithm

Logic Programming:

- Linear Input Resolution with Horn Clauses: only one positive literal in each clause, so cuts down choices.
- Answer substitution produced on successful refutation
- Correspondence between Logic Programming and Resolution
Prolog Programming:

- Syntax: constants, variables, terms, user-defined operators
- Rules, facts, goals
- Program execution = resolution steps
- Answer substitution
- Techniques: recursion, accumulator argument
- SLD Search strategy: search rules top-to-bottom; choose left-hand literal in goal;
- Depth-first search of SLD-tree;
- Cuts: prune SLD-Tree to search; negation-as-failure: return false when there's no negative result
# Glossary (Partial!)

## Propositional:

<table>
<thead>
<tr>
<th>Propositional</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Prop. Language</td>
<td>$\mathcal{P}$</td>
</tr>
<tr>
<td>Truth symbols</td>
<td>t, f</td>
</tr>
<tr>
<td>Connectives</td>
<td>$\neg, \land, \lor, \rightarrow$</td>
</tr>
<tr>
<td>Truth values</td>
<td>true, false</td>
</tr>
<tr>
<td>Some Literal</td>
<td>$L, \bar{L}$</td>
</tr>
<tr>
<td>Some atomic proposition</td>
<td>$A$ or $B$ or $B_2$ or $A'$ …</td>
</tr>
<tr>
<td>Some prop. formulae</td>
<td>$F$ or $G$ or $G_2$ or $F'$ …</td>
</tr>
<tr>
<td>Set of prop. formulae</td>
<td>$X$ or $X'$ …</td>
</tr>
<tr>
<td>Valuation (prop. formulae)</td>
<td>$v(F)$</td>
</tr>
<tr>
<td>Logical consequence</td>
<td>$\models$</td>
</tr>
<tr>
<td>Logical equivalence</td>
<td>$\models \equiv$</td>
</tr>
<tr>
<td>Terms</td>
<td>Definition</td>
</tr>
<tr>
<td>-------</td>
<td>------------</td>
</tr>
<tr>
<td>Clauses</td>
<td>$C$ or $D$ or $C_1$ or $D'$ …</td>
</tr>
<tr>
<td>Clause</td>
<td>$[L_1; L_2; \ldots; L_m]$</td>
</tr>
<tr>
<td>Set of clauses</td>
<td>${C_1, C_2, \ldots, C_n}$</td>
</tr>
<tr>
<td>Set of clauses</td>
<td>$N$ or $N_0$ or $N'$ …</td>
</tr>
<tr>
<td>valuation (clausal form)</td>
<td>$v(X)$</td>
</tr>
<tr>
<td>Substitution</td>
<td>${x/t}$</td>
</tr>
<tr>
<td>Set difference</td>
<td>$N \setminus N'$</td>
</tr>
</tbody>
</table>
**Predicate:**

<table>
<thead>
<tr>
<th>FOPL. Language</th>
<th>$L$</th>
</tr>
</thead>
<tbody>
<tr>
<td>variables</td>
<td>$x$ or $x_1$ or $y$ ...</td>
</tr>
<tr>
<td>constant symbols</td>
<td>$a$ or $a_1$ or $b$ ...</td>
</tr>
<tr>
<td>function symbols</td>
<td>$f$ or $f_1$ or $g$ ...</td>
</tr>
<tr>
<td>(data) term</td>
<td>$t$ or $t_1$ or $s'$ ...</td>
</tr>
<tr>
<td>predicate symbols</td>
<td>$p$ or $p_1$ or $q'$ ...</td>
</tr>
<tr>
<td>substitutions</td>
<td>$\sigma$ or $\theta$ ...</td>
</tr>
</tbody>
</table>