Propositional logic

- **Alphabet A:**
  - Propositional symbols (identifiers)
- **Connectives:**
  - $\land$ (conjunction)
  - $\lor$ (disjunction)
  - $\neg$ (negation)
  - $\leftrightarrow$ (logical equivalence)
  - $\rightarrow$ (implication)
Propositional logic

• **Well-formed formulas** (*wffs*, denoted by *F*) over alphabet *A* is the smallest set such that:
  • If *p* is a predicate symbol in *A* then *p* ∈ *F*.
  • If the *wffs* *F*, *G* ∈ *F* then so are (*¬*F), (F ∧ G), (F ∨ G), (F → G) and (F ↔ G).
Interpretation

• An *interpretation* I is a subset of propositions in an alphabet A

• Alternatively, you can view I as a mapping from the set of all propositions in A to a 2-values Boolean domain \{true, false\}

• This name, “*interpretation*”, is more commonly used for predicate logic
  • in the propositional case, this is sometimes called a “*substitution*” or “*truth assignment*”
Semantics of Well-Formed Formulae

- A formula’s meaning is given w.r.t. an interpretation \( I \):
  
  \[ I \models p \iff p \in I \]
  
  \[ I \models \neg F \iff I \not\models F \text{ (i.e., } I \text{ does not entail } F) \]
  
  \[ I \models F \land G \iff I \models F \text{ and } I \models G \]
  
  \[ I \models F \lor G \iff I \models F \text{ or } I \models G \text{ (or both)} \]
  
  \[ I \models F \rightarrow G \iff I \models G \text{ whenever } I \models F \]
  
  \[ I \models F \leftrightarrow G \iff I \models F \rightarrow G \text{ and } I \models G \rightarrow F \]

Notes: we read "\( \models \)" as entails, models, "is a semantic consequence of"

We read \( I \models p \) as "\( I \) entails \( p \)".
Models

• An interpretation I such that I ⊨ F is called “a model” of F

• “G is a logical consequence of F” (denoted by F ⊨ G) iff every model of F is also a model of G
  • in other words, G holds in every model of F; or G is true in every interpretation that makes F true
Models

• A formula that has at least one model is said to be “satisfiable”

• A formula for which every interpretation is a model is called a “tautology”

• A formula is “inconsistent” if it has no models
Models

• Checking whether or not a formula is satisfiable is NP-Complete (the SAT problem) because there are exponentially many interpretations.

• Many interesting combinatorial problems can be reduced to checking satisfiability: hence, there is a significant interest in efficient algorithms/heuristics/systems for solving the SAT problem.
Logical Consequence

- Let $P$ be a set of clauses $\{C_1, C_2, \ldots, C_n\}$, where
  - each clause $C_i$ is of the form $(L_1 \lor L_2 \lor \ldots \lor L_k)$, and where
  - each $L_j$ is a literal: a proposition or a negated proposition
- A model for $P$ makes every one of $C_i$s in $P$ true
- Let $G$ be a literal (called “Goal”)
  - Consider the question: does $P \models G$?
    - We can use a proof procedure, based on resolution to answer this question
Proof System for Resolution

\[
\{C\} \cup P \vdash C
\]

\[
P \vdash (A \lor C_1) \quad P \vdash (\neg A \lor C_2)
\]

\[
P \vdash (C_1 \lor C_2)
\]

Resolution

• The above notation is of “inference rules” where each rule is of the form:

  Antecedent(s) \[\quad\] Conclusion

• \(P \vdash C\) is called as a “sequent”
  • \(P \vdash C\) means \(C\) can be proved if \(P\) is assumed true
Proof System for Resolution

• The turnstile, $\vdash$, represents **syntactic consequence** (or "derivability")
  • $P \vdash C$ means that $C$ is **derivable** from $P$ using the proof procedure
• It is often read as "proves" or "yields"
Proof System for Resolution

- Modus ponens can be seen as a special case of resolution (of a one-literal clause and a two-literal clause) because

\[ \frac{p \rightarrow q, p}{q} \]  

is equivalent to

\[ \frac{\neg p \lor q, p}{q} \]
Proof System for Resolution

- Given a sequent, a **derivation** of a sequent (sometimes called its "proof") is a tree with:
  - that sequent as the root,
  - empty leaves, and
  - each internal node is an instance of an inference rule.

- A proof system based on Resolution is
  - **Sound**: i.e. if $F \vdash G$ then $F \models G$.
  - **not Complete**: i.e. there are $F, G$ s.t. $F \models G$ but $F \not\vdash G$.
    - E.g., $p \models (p \lor q)$ but there is no way to derive $p \vdash (p \lor q)$ using only resolution
Resolution Proof (in pictures)

\[ P = \{(p \lor q), (\neg p \lor r), (\neg q \lor r)\} \]

\[ \frac{(p \lor q) \quad (\neg p \lor r)}{q \lor r} \quad \frac{(q \lor r)}{r} \quad \frac{(\neg q \lor r)}{r} \]
Resolution Proof (An Alternative View)

• The clauses of P are all in a “pool”/table
• Resolution rule picks two clauses from the “pool”, of the form \( A \lor C_1 \) and \( \neg A \lor C_2 \)
• and adds \( C_1 \lor C_2 \) to the “pool”
• The newly added clause can now interact with other clauses and produce yet more clauses
• Ultimately, the “pool” consists of all clauses C such that \( P \vdash C \)
Resolution Proof (An Example)

- \( P = \{(p \lor q), (\neg p \lor r), (\neg q \lor r)\} \)

- Here is a proof for \( P \models r \):

<table>
<thead>
<tr>
<th>Clause Number</th>
<th>Clause</th>
<th>How Derived</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>( p \lor q )</td>
<td>( \in P )</td>
</tr>
<tr>
<td>2</td>
<td>( \neg p \lor r )</td>
<td>( \in P )</td>
</tr>
<tr>
<td>3</td>
<td>( \neg q \lor r )</td>
<td>( \in P )</td>
</tr>
<tr>
<td>4</td>
<td>( q \lor r )</td>
<td>Res. 1 &amp; 2</td>
</tr>
<tr>
<td>5</td>
<td>( r )</td>
<td>Res. 3 &amp; 4</td>
</tr>
</tbody>
</table>
Refutation Proofs

- While resolution alone is incomplete for determining logical consequences, resolution is sufficient to show inconsistency (i.e. show when $P$ has no model):

- **Refutation** proofs (*Reductio ad absurdum = reduction to absurdity*) for showing logical consequence:
  - Say we want to determine $P \models r$?, where $r$ is a proposition
  - This is equivalent to checking if $P \cup \{\neg r\}$ has an empty model
  - This we can check by constructing a resolution proof for $P \cup \{\neg r\} \vdash \Box$, where $\Box$ denotes the unsatisfiable empty clause
Refutation Proofs (An Example)

- Let $P = \{(p \lor q), (\neg p \lor r), (\neg q \lor r), (p \lor s)\}$, and
- $G = (r \lor s)$

<table>
<thead>
<tr>
<th>Clause Number</th>
<th>Clause</th>
<th>How Derived</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>$p \lor q$</td>
<td>$\in P \cup \neg G$</td>
</tr>
<tr>
<td>2</td>
<td>$\neg p \lor r$</td>
<td>$\in P \cup \neg G$</td>
</tr>
<tr>
<td>3</td>
<td>$\neg q \lor r$</td>
<td>$\in P \cup \neg G$</td>
</tr>
<tr>
<td>4</td>
<td>$\neg r$</td>
<td>$\in P \cup \neg G$</td>
</tr>
<tr>
<td>5</td>
<td>$\neg s$</td>
<td>$\in P \cup \neg G$</td>
</tr>
<tr>
<td>6</td>
<td>$q \lor r$</td>
<td>Res. 1 &amp; 2</td>
</tr>
<tr>
<td>7</td>
<td>$r$</td>
<td>Res. 3 &amp; 6</td>
</tr>
<tr>
<td>8</td>
<td>$\Box$</td>
<td>Res. 4 &amp; 7</td>
</tr>
</tbody>
</table>
Clausal form

• Propositional Resolution works only on expressions in **clausal form**

• There is a simple procedure for **converting** an arbitrary set of Propositional Logic sentences to an equivalent set of clauses

• **Implications (I):**

  - $\phi \rightarrow \psi \rightarrow \neg \phi \lor \psi$
  - $\phi \leftarrow \psi \rightarrow \phi \lor \neg \psi$
  - $\phi \leftrightarrow \psi \rightarrow (\neg \phi \lor \psi) \land (\phi \lor \neg \psi)$

• **Negations (N):**

  - $\neg \neg \phi \rightarrow \phi$
  - $\neg (\phi \land \psi) \rightarrow \neg \phi \lor \neg \psi$
  - $\neg (\phi \lor \psi) \rightarrow \neg \phi \land \neg \psi$
Clausal form

- **Distribution (D):**
  - $\phi \lor (\psi \land \chi) \rightarrow (\phi \lor \psi) \land (\phi \lor \chi)$
  - $(\phi \land \psi) \lor \chi \rightarrow (\phi \lor \chi) \land (\psi \lor \chi)$
  - $\phi \lor (\phi_1 \lor ... \lor \phi_n) \rightarrow \phi \lor \phi_1 \lor ... \lor \phi_n$
  - $(\phi_1 \lor ... \lor \phi_n) \lor \phi \rightarrow \phi_1 \lor ... \lor \phi_n \lor \phi$
  - $\phi \land (\phi_1 \land ... \land \phi_n) \rightarrow \phi \land \phi_1 \land ... \land \phi_n$
  - $(\phi_1 \land ... \land \phi_n) \land \phi \rightarrow \phi_1 \land ... \land \phi_n \land \phi$

- **Operators (O):**
  - $\phi_1 \lor ... \lor \phi \rightarrow \{\phi_1, ... , \phi_n\}$
  - $\phi_1 \land ... \land \phi_n \rightarrow \{\phi_1\}, ... , \{\phi_n\}$
Clausal form: Example

- Convert the sentence \((g \land (r \rightarrow f))\) to clausal form:

  \[
  g \land (r \rightarrow f)
  \]

  I  \[
  g \land (\neg r \lor f)
  \]

  N  \[
  g \land (\neg r \lor f)
  \]

  D  \[
  g \land (\neg r \lor f)
  \]

  O  \[
  \{g\}
  \]

  \[
  \{\neg r, f\}
  \]
Clausal form: Example

• Convert the sentence \( \neg(g \land (r \rightarrow f)) \) to clausal form:

\[
\neg(g \land (r \rightarrow f)) \\
I \quad \neg(g \land (\neg r \lor f)) \\
N \quad \neg g \lor \neg(\neg r \lor f) \\
\quad \neg g \lor (\neg \neg r \land \neg f) \\
\quad \neg g \lor (r \land \neg f) \\
D \quad (\neg g \lor r) \land (\neg g \lor \neg f) \\
O \quad \{\neg g, r\} \\
\quad \{\neg g, \neg f\}
\]
Soundness of Resolution

- If $F \vdash G$ then $F \models G$:
  - For $F \vdash G$, we will have a derivation (aka “proof”) of finite length
  - We can show that $F \models G$ by induction on the length of that derivation
Refutation-Completeness of Resolution

• If $F$ is inconsistent, then $F \vdash \Box$:
  • Note that $F$ is a set of clauses
    • A clause is called an \textit{unit clause} if it consists of a single literal.
  • If all clauses in $F$ are unit clauses, then for $F$ to be inconsistent, clearly a literal and its negation will be two of the clauses in $F$
    • Then resolving those two will generate the empty clause
  • A clause with $n + 1$ literals has “$n$ excess literals”
    • The proof of refutation-completeness is by induction on the number of excess literals in $F$ (each one of them has to be eliminated to bring to inconsistency)
Refutation-Completeness of Resolution

- Induction: Assume refutation completeness holds for all clauses with \( n \) excess literals; show that it holds for clauses with \( n + 1 \) excess literals:
  - From F, pick some clause C with excess literals
  - Pick some literal, say A from C
  - Consider \( C' = C - \{A\} \)
  - Both \( F_1 = (F - \{C\}) \cup \{C'\} \) and \( F_2 = (F - \{C\}) \cup \{A\} \) are inconsistent and have at most \( n \) excess literals
    - By induction hypothesis, both have refutations
  - If there is a refutation of \( F_1 \) not using \( C' \), then that is a refutation for \( F \) as well
  - If the refutation of \( F_1 \) uses \( C' \), then construct a resolution of \( F \) by adding A to the first occurrence of \( C' \) (and its descendants); that will end with \( \{A\} \)
  - From here on, follow the refutation of \( F_2 \). This constructs a refutation of \( F \)
A simple theorem prover in Prolog

- Operators for formulas:
  
  ```prolog
  :- op(100, fy, ~). %Negation
  :- op(110, xfy, &). %Conjunction
  :- op(120, xfy, v). %Disjunction
  :- op(130, xfy, =>). %Implication
  :- op(800, xfx, --->). %Implication
  ```

- Clausal form:

  ```prolog
  transform(~ (~X), X) :- %Double negation
                  !.
  transform(X => Y, ~X v Y) :- %Eliminate implication
                  !.
  transform(~(X & Y), ~X v ~Y) :- %De Morgan's law
                  !.
  transform(~(X v Y), ~X & ~Y) :- %De Morgan's law
                  !.
  transform(X & Y v Z, (X v Z) & (Y v Z)) :- !. %Distribution
  transform(X v Y & Z, (X v Y) & (X v Z)) :- !. %Distribution
  transform(X v Y, X1 v Y) :- %Transform subexpression
    transform(X, X1), !.
  transform(X v Y, X v Y1) :- %Transform subexpression
    transform(Y, Y1), !.
  transform(~X, ~X1) :- %Transform subexpression
    transform(X, X1).
  ```
Resolution:

:- dynamic(done/3).
% Contradicting clauses
[clause(X), clause(~X)] --->
  [write('Contradiction found'), stop].
% Remove a true clause
[clause(C), in(P, C), in(~P, C)] --->
  [retract(C)].
% Simplify a clause
[clause(C), delete(P, C, C1), in(P, C1)] --->
  [replace(clause(C), clause(C1) )].
% Resolution step, a special case
[clause(P), clause(C), delete(~P, C, C1), not done(P, C, P)] --->
  [assert(clause(C1) ), assert(done(P, C, P) )].
% Resolution step, a special case
[clause(~P), clause(C), delete(P, C, C1), not done(~P, C, P)] --->
  [assert(clause(C1) ), assert(done(~P, C, P) )].
% Resolution step, general case
[clause(C1), delete(P, C1, CA), clause(C2), delete(~P,C2,CB), not
done(C1,C2,P)] --->
  [assert(clause(CA v CB) ), assert(done(C1, C2, P) )].
% Last rule: resolution process stuck
[] ---> [write('Not contradiction'), stop].
% \text{delete}(P, E, E_1) \text{ means: delete a disjunctive subexpression } P \text{ from } E \\
% \text{ giving } E_1 \\
delete(X, X \lor Y, Y). \\
delete(X, Y \lor X, Y). \\
delete(X, Y \lor Z, Y \lor Z_1) :- \\
    
    delete(X, Z, Z_1). \\
delete(X, Y \lor Z, Y_1 \lor Z) :- \\
    delete(X, Y, Y_1). \\
% \text{in}(P, E) \text{ means: } P \text{ is a disjunctive subexpression in } E \\
in(X, X). \\
in(X, Y) :- \\
    delete(X, Y, _). \\
% \text{Translate conjunctive formula} \\
translate(F \& G) :- \\
    !,
    translate(F), \\
    translate(G). \\
% \text{Transformation step on Formula} \\
translate(Formula) :- \\
    transform(Formula, NewFormula), \\
    !,
    translate(NewFormula). \\
% \text{No more transformation possible} \\
translate(Formula) :- \\
    assert(clause(Formula) ).
run :-
  Condition --> Action, % A production rule
  test(Condition), % Precondition satisfied?
  execute(Action).
run(State) :-
  Condition --> Action,
  test(Condition, State),
  execute(Action, State).

% Empty condition
% test([]).
% Test conjunctive condition
call(First),
  test(Rest).

execute([stop]) :- !. % Stop execution
execute([]) :- % Empty action (execution cycle completed)
  run. % Continue with next execution cycle
execute([First | Rest]) :-
  call(First),
  execute(Rest).

replace(A, B) :- % Replace A with B in database
  retract(A), !, % Retract once only
  assert(B).

?- translate( ~(a => b) & (b => c) => (a => c) ), run.
  Contradiction found
  yes