Propositional Logic and Resolution

CSE 505 – Computing with Logic
Stony Brook University

http://www.cs.stonybrook.edu/~cse505
Propositional logic

• Alphabet $A$:
  • Propositional symbols (identifiers)
  • Connectives:
    • $\wedge$ (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 $p \in F$.
  - If the wffs $F, G \in F$ then so are $\neg F$, $(F \land G)$, $(F \lor G)$, $(F \rightarrow G)$ and $(F \leftrightarrow G)$. 
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 $I$:
  
  $I \vDash p$ iff $p \in I$

  $I \vDash \neg F$ iff $I \not\vDash F$ (i.e., $I$ does not entail $F$)

  $I \vDash F \land G$ iff $I \vDash F$ and $I \vDash G$

  $I \vDash F \lor G$ iff $I \vDash F$ or $I \vDash G$ (or both)

  $I \vDash F \rightarrow G$ iff $I \vDash G$ whenever $I \vDash F$

  $I \vDash F \iff G$ iff $I \vDash F \rightarrow G$ and $I \vDash G \rightarrow F$

Notes: we read "$\vDash$" as *entails, models, is a semantic consequence of*'

We read $I \vDash p$ 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 $(L_1 \lor L_2 \lor \ldots \lor L_k)$,
  - each $L_j$ is a literal: i.e. a possibly 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

\[
\frac{\{C\} \cup P \vdash C}{P \vdash (A \lor C_1)} \quad \frac{P \vdash (\neg A \lor C_2)}{P \vdash (C_1 \lor C_2)}
\]

Resolution

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

  \[
  \text{Antecedent(s)} \quad \frac{}{\text{Conclusion}}
  \]

- \( P \vdash C \) 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 $P$
• It is often read as "yields" or "proves"
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

\[
\frac{p \rightarrow q, p}{q} \quad \text{is equivalent to} \quad \frac{\neg p \lor q, p}{q}
\]
Proof System for Resolution

• Given a sequent, a \textit{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 $p \vdash (p \lor q)$.
Resolution Proof (in pictures)

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

\[
\begin{align*}
(p \lor q) & \quad (\neg p \lor r) \\
(q \lor r) & \quad (\neg q \lor r) \\
& \downarrow \\
r & \quad r
\end{align*}
\]
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>

(c) Paul Fodor (CS Stony Brook) and Elsevier
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):
  • \( \phi \lor (\psi \land \chi) \rightarrow (\phi \lor \psi) \land (\phi \lor \chi) \)
  • \((\phi \land \psi) \lor \chi \rightarrow (\phi \lor \chi) \land (\psi \lor \chi) \)
  • \(\phi \lor (\phi_1 \lor \ldots \lor \phi_n) \rightarrow \phi \lor \phi_1 \lor \ldots \lor \phi_n \)
  • \((\phi_1 \lor \ldots \lor \phi_n) \lor \phi \rightarrow \phi_1 \lor \ldots \lor \phi_n \lor \phi \)
  • \(\phi \land (\phi_1 \land \ldots \land \phi_n) \rightarrow \phi \land \phi_1 \land \ldots \land \phi_n \)
  • \((\phi_1 \land \ldots \land \phi_n) \land \phi \rightarrow \phi_1 \land \ldots \land \phi_n \land \phi \)

• Operators (O):
  • \(\phi_1 \lor \ldots \lor \phi \rightarrow \{\phi_1, \ldots, \phi_n\} \)
  • \(\phi_1 \land \ldots \land \phi_n \rightarrow \{\phi_1\}, \ldots, \{\phi_n\} \)
Clausal form: Example

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

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

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

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

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

\[
O \quad \{g\}
\]

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

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

$I$  \hspace{1cm} $\neg(g \land (\neg r \lor f))$

$N$  \hspace{1cm} $\neg g \lor \neg(\neg r \lor f) = \neg g \lor (\neg \neg r \land \neg f)$

$D$  \hspace{1cm} $(\neg g \lor r) \land (\neg g \lor \neg f)$

$O$  \hspace{1cm} $\{\neg g, r\}$

$\{\neg g, \neg f\}$
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 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 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.
Refutation-Completeness of Resolution

- **If F is inconsistent, then F ⊬ □:**
  - 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 F1 = (F - {C}) ∪ {C'} and F2 = (F - {C}) ∪ {A} are inconsistent and have at most n excess literals.
  - By induction hypothesis, both have refutations. If there is a refutation of F1 not using C’, then that is a refutation for F as well.
  - If refutation of F1 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 F2. 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, --->).
  ```

- 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:

```prolog
:- 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].
```
% delete(P, E, E1) means: delete a disjunctive subexpression P from E
% giving E1
delete(X, X v Y, Y).
delete(X, Y v X, Y).
delete(X, Y v Z, Y v Z1):-
    delete(X, Z, Z1).
delete(X, Y v Z, Y1 v Z) :-
    delete(X, Y, Y1).
% in(P, E) means: P is a disjunctive subexpression in E
in(X, X).
in(X, Y):-
    delete(X, Y, _).
% Translate conjunctive formula
translate(F & G) :-
    !,
    translate(F),
    translate(G).
% Transformation step on Formula
translate(Formula) :-
    transform(Formula, NewFormula),
    !,
    translate(NewFormula).
% 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).
test([]).  % Empty condition
test([First|Rest]):- % Test conjunctive condition
    call(First),
    test(Rest).
% execute([Action1, Action2, ...]): execute list of actions
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