Proof Methods for Propositional Logic

Outline

- Automated Propositional Proof Methods
  1. Resolution
  2. A Practical Method: Walksat

Resolution

Conjunctive Normal Form (CNF)

- Resolution inference rule (for CNF):

\[
\begin{align*}
& (l_1 \lor \cdots \lor l_i - 1 \lor l_{i+1} \lor \cdots \lor l_k) \land
\end{align*}
\]

\[
\begin{align*}
& (m_1 \lor \cdots \lor m_j - 1 \lor m_{j+1} \lor \cdots \lor m_n)
\end{align*}
\]

\[
\begin{align*}
& \implies \\
& (l_1 \lor \cdots \lor l_i - 1 \lor l_{i+1} \lor \cdots \lor l_k) \lor (m_1 \lor \cdots \lor m_j - 1 \lor m_{j+1} \lor \cdots \lor m_n)
\end{align*}
\]

where \( l_i \) and \( m_j \) are complementary literals, i.e. \( \xi = \neg m_i \)

e.g. \( p_{1,3} \lor p_{2,2} \lor \neg p_{1,2} \)

- Resolution is sound and complete for propositional logic

For convenience: Logical equivalence

- To manipulate logical sentences we need some rewrite rules.
- Two sentences are logically equivalent iff they are true in same models: \( \alpha \equiv \beta \) iff \( \alpha \vdash \beta \) and \( \beta \vdash \alpha \)

\[
\begin{align*}
& (\alpha \land \beta) \equiv (\beta \land \alpha) \quad \text{commutativity of } \land \\
& (\alpha \lor \beta) \equiv (\beta \lor \alpha) \quad \text{commutativity of } \lor \\
& ((\alpha \land \beta) \land \gamma) \equiv (\alpha \land (\beta \land \gamma)) \quad \text{associativity of } \land \\
& ((\alpha \lor \beta) \lor \gamma) \equiv (\alpha \lor (\beta \lor \gamma)) \quad \text{associativity of } \lor \\
& \neg(\alpha) \equiv \alpha \quad \text{double-negation elimination} \\
& (\alpha \Rightarrow \beta) \equiv (\neg \alpha \lor \beta) \quad \text{implication elimination} \\
& (\alpha \Leftarrow \beta) \equiv (\alpha \Rightarrow \beta \land (\beta \Rightarrow \alpha)) \quad \text{biconditional elimination} \\
& \neg(\alpha \land \beta) \equiv (\neg \alpha \lor \neg \beta) \quad \text{de Morgan} \\
& \neg(\alpha \lor \beta) \equiv (\neg \alpha \land \neg \beta) \quad \text{de Morgan} \\
& (\alpha \land (\beta \lor \gamma)) \equiv ((\alpha \land \beta) \lor (\alpha \land \gamma)) \quad \text{distributivity of } \land \text{ over } \lor \\
& (\alpha \lor (\beta \land \gamma)) \equiv ((\alpha \lor \beta) \land (\alpha \lor \gamma)) \quad \text{distributivity of } \lor \text{ over } \land
\end{align*}
\]
Conversion to CNF: General Procedure

Example: \( B_{1,1} \equiv (P_{1,2} \lor P_{2,1}) \)

1. **Eliminate** \( \equiv \), replacing \( \alpha \equiv \beta \) with \( (\alpha \implies \beta) \land (\beta \implies \alpha) \).

\[
(B_{1,1} \equiv (P_{1,2} \lor P_{2,1})) \land ((P_{1,2} \lor P_{2,1}) \implies B_{1,1})
\]

2. **Eliminate** \( \implies \), replacing \( \alpha \implies \beta \) with \( \neg \alpha \lor \beta \).

\[
(\neg B_{1,1} \lor P_{1,2} \lor P_{2,1}) \land ((\neg P_{1,2} \lor \neg P_{2,1}) \lor B_{1,1})
\]

3. **Move** \( \neg \) inwards using de Morgan’s rules and (often, but not here) double-negation:

\[
(\neg B_{1,1} \lor P_{1,2} \lor P_{2,1}) \land (\neg P_{1,2} \land \neg P_{2,1}) \lor B_{1,1}
\]

4. **Flatten** by applying distributivity law (\( \land \) over \( \lor \)):

\[
(\neg B_{1,1} \lor P_{1,2} \lor P_{2,1}) \land (\neg P_{1,2} \land \neg P_{2,1}) \lor B_{1,1}
\]

Review: Validity and satisfiability

A sentence is **valid** if it is true in all models, e.g. True, \( A \lor \neg A \), \( A \implies (A \lor B) \).

**Validity** is connected to inference via the **Deduction Theorem**: \( KB \vdash \alpha \iff (KB \implies \alpha) \) is valid

A sentence is **satisfiable** if it is true in some model e.g. \( A \lor B \), \( C \)

A sentence is **unsatisfiable** if it is false in all models e.g. \( A \land \neg A \)

Satisfiability is connected to inference via the following:

\( KB \vdash \alpha \iff \neg (KB \land \neg \alpha) \) is unsatisfiable

(The there is no model for which \( KB \land \neg \alpha \) is false)

Resolution algorithm

- Proof by contradiction, i.e., prove \( \alpha \) by showing \( KB \land \neg \alpha \) unsatisfiable

Resolution example

- \( KB = (B_{1,1} \equiv (P_{1,2} \lor P_{2,1})) \land \neg B_{1,1} \)
- \( \alpha = \neg P_{1,2} \)

The WalkSAT algorithm

- A practical, simple algorithm for propositional inference
- **Sound**
- **Incomplete**
- A hill-climbing search algorithm
- Balance between greediness and randomness
  - Evaluation function: **min-conflict heuristic** of minimizing the number of unsatisfied clauses
  - Uses random jumps to escape local minima

The WalkSAT algorithm

Function: \( \text{WalkSAT}(\text{clauses}, p, \max\, \text{flips}) \) returns a satisfying model or failure

Inputs: clauses, a set of clauses in propositional logic
\( p \), the probability of choosing to do a “random walk” move
\( \max\, \text{flips} \), number of flips allowed before giving up

model \( \rightarrow \) a random assignment of true/false to the symbols in clauses

for \( i = 1 \) to \( \max\, \text{flips} \) do

if model satisfies clauses then return model

\( \text{clauses} \rightarrow \) a randomly selected clause from clauses that is false in model

with probability \( p \), flip the value in model of a randomly selected symbol from clause

else flip whichever symbol in clause maximizes the number of satisfied clauses

return failure
Hard satisfiability problems

- Consider random 3-CNF sentences, e.g.,
  \[(\neg D \vee \neg B \vee C) \land (B \vee \neg A \vee \neg C) \land (C \vee \neg B \vee E) \land (E \vee D \vee B) \land (B \vee E \vee \neg C)\]

  \(m\) = number of clauses
  \(n\) = number of symbols

- Hard problems seem to cluster near \(m/n = 4.3\) (critical point)

- Here:
  \(m=4, n=|\{A,B,C,D,E\}| = 5\)
  \(m/n = 4/5 = 0.8\)

Hard satisfiability problems

- Median runtime for 100 satisfiable random 3-CNF sentences, \(n = 50\)

Encoding Wumpus in propositional logic

- 4x4 Wumpus World
  - The “physics” of the game
    - \(B_{k,8} \equiv (P_{k,4} \vee P_{k-1,4} \vee P_{k+1,4} \vee P_{k+3,4})\)
    - \(S_{k,8} \equiv (W_{k,2} \vee W_{k,3} \vee W_{k,4} \vee W_{k,8} \vee W_{k,9})\)
  - At least one wumpus on board
    - \(W_{1,1} \vee W_{1,2} \vee \cdots \vee W_{4,3} \vee W_{4,4}\)
  - A most one wumpus on board (for any two squares, one is free)
    - \(m^2\) rules like: \(W_{i,j} \Rightarrow \neg(W_{i+1,j} \lor W_{i,j+1} \lor \cdots \lor W_{i,j+3})\)
  - No instant death:
    - \(P_{i,j}\)
    - \(\neg W_{i,j}\)

Expressiveness limitation of propositional logic

- KB contains “physics” sentences for every single square
- Rapid proliferation of clauses
Forward and backward chaining

- **Horn Clause** (restricted)
  - Horn clause:
    - A proposition symbol \( \rightarrow \) \( \) symbol
    - E.g.: \( A \rightarrow B \rightarrow C \rightarrow D \rightarrow B \)
  - KB = conjunction of Horn clauses
    - E.g., \( C \land (B \Rightarrow A) \land (C \land D \Rightarrow B) \)
  - Modus Ponens (for Horn Form): complete for Horn KBs
    - \( \frac{\alpha_1, \ldots, \alpha_n}{\alpha_1 \land \ldots \land \alpha_n \Rightarrow \beta} \)

- Used with forward chaining or backward chaining.
- These algorithms are very natural and run in linear time.

Forward chaining

- Idea: Apply modus ponens to any Horn Clause whose premises are satisfied in the KB
  - Add its conclusion to the KB until query is found
  - Easy to visualize informally in graphical form:

\[
\begin{align*}
P & \Rightarrow Q \\
L \land M & \Rightarrow P \\
B \land L & \Rightarrow M \\
A \land P & \Rightarrow L \\
A \land B & \Rightarrow L \\
A & \\
B &
\end{align*}
\]

Forward chaining algorithm

- For forward chaining:
  - Sound and complete for Horn KBs

Forward chaining example
Forward chaining example

Proof of completeness

FC derives every atomic sentence that is entailed by KB

1. FC reaches a fixed point where no new atomic sentences are derived
2. Consider the final state as a model $m$, assigning true/false to symbols
3. Every clause in the original $KB$ is true in $m$
4. Hence $m$ is a model of $KB$
5. If $KB \models q$, $q$ is true in every model of $KB$, including $m$
**Backward chaining**

Idea: work backwards from the query $q$:
- to prove $q$ by BC,
  - check if $q$ is known already, or
  - prove by BC all premises of some rule concluding $q$

Avoid loops: check if new subgoal is already on the goal stack

Avoid repeated work: check if new subgoal
1. has already been proved true, or
2. has already failed

---

**Backward chaining example**

[Diagram showing backward chaining process with nodes and arrows]

---

**Backward chaining example**

[Diagram showing another backward chaining process]

---

**Backward chaining example**

[Diagram showing a third backward chaining process]

---

**Backward chaining example**

[Diagram showing a fourth backward chaining process]
Backward chaining example

Forward vs. backward chaining

- FC is data-driven, automatic, unconscious processing,
  - e.g., object recognition, routine decisions
- May do lots of work that is irrelevant to the goal
- BC is goal-driven, appropriate for problem-solving,
  - e.g., Where are my keys? How do I get into a PhD program?
- Complexity of BC can be much less than linear in size of KB