Proof Methods for Propositional Logic

Outline

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

Proof methods

I. Application of Inference Rules
   - Each application yields the legitimate (sound) generation of a new sentence from old
   - Proof = a sequence of sound inference rule applications
   - Proofs can be found using search
     - Inference Rules as operators for a standard search algorithm
   - Typically require transformation of sentences into a normal form
   - Example: Resolution

II. Model Checking Methods
   - Examples:
     - Truth Table Enumeration (tests satisfiability, validity)
     - WalkSat (tests satisfiability)

Resolution

Applies to a DB of Sentences in Conjunctive Normal Form (CNF)

\[ \alpha \lor \beta \lor \gamma \lor \delta \lor \varepsilon \lor \zeta \lor \eta \lor \theta \lor \iota \lor \kappa \lor \lambda \lor \mu \lor \nu \lor \xi \lor \omicron \lor \pi \lor \rho \lor \sigma \lor \tau \lor \upsilon \lor \psi \lor \phi \lor \chi \lor \psi \lor \omega \]

Resolution inference rule (for CNF):

\[
\begin{align*}
-\psi & \lor \ldots \lor \psi_i \lor \ldots \lor \psi_k \lor \psi_1 \lor \ldots \lor \psi_j \lor \ldots \lor \psi_n \\
-\psi & \lor \ldots \lor \psi_i \lor \ldots \lor \psi_k \lor \psi_1 \lor \ldots \lor \psi_j \lor \ldots \lor \psi_n
\end{align*}
\]

where \( \psi_i \) and \( \psi_j \) are complementary literals, i.e. \( \psi_i = \neg \psi_j \)

- Resolution is sound and complete for propositional logic

Soundness of resolution inference rule

If \( \xi = \neg \psi_i \)

\[
\begin{align*}
-\psi & \lor \ldots \lor \psi_i \lor \ldots \lor \psi_k \\
-\psi & \lor \ldots \lor \psi_i \lor \ldots \lor \psi_k
\end{align*}
\]

Given that \( \psi \lor \beta = (\neg \alpha \lor \beta) \)

Review: Validity and satisfiability

A sentence is valid if it is true in all models, e.g. \( \text{True, } A \lor \neg A, A \Rightarrow A, (A \land (A \Rightarrow B)) \Rightarrow B \)

Validity is connected to inference via the Deduction Theorem:

\( KB \vdash \alpha \) if and only if \( (KB \Rightarrow \alpha) \) 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:

\( KB \vdash \alpha \) if and only if \( (KB \Rightarrow \neg \alpha) \) is unsatisfiable

(there is no model for which KB=true and \( \alpha \) is false)
Resolution algorithm

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

Resolution example

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

Resolution example diagram:

1. \( \neg B_{1,1} \lor P_{1,2} \lor P_{2,1} \)
2. \( \neg P_{1,2} \lor \neg B_{1,1} \lor P_{2,1} \)
3. \( \neg P_{2,1} \lor \neg B_{1,1} \lor P_{1,2} \)
4. \( P_{1,2} \lor \neg B_{1,1} \lor \neg P_{2,1} \)

Conversion to CNF: General Procedure

Example:

- \( B_{1,1} \iff (P_{1,2} \lor P_{2,1}) \)

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

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

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

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

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

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

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

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

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 \( \alpha \models \beta \) and \( \beta \models \alpha \)

The WalkSAT algorithm

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

The WalkSAT algorithm diagram:

Function \( \text{WALKSAT}(\text{clauses}, m, \text{max-flips}) \) returns a satisfying model or failure.

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

Algorithm:
- A random assignment of true/false to the symbols in clauses
- For \( i = 1 \) to \( m \) max-flips do
  - If model satisfies clauses then return model
  - Else flip a randomly selected clause from clauses that is in conflict with \( m \) with probability \( p \)

For convenience: Logical equivalence

I told you you needed to know these!
Hard satisfiability problems

- Consider random 3-CNF sentences, e.g.,
  \((-D \lor \neg B \lor \neg C) \land (B \lor \neg A \lor \neg \neg C) \land (E \lor D \lor \neg B) \land (A \lor E \lor \neg C)\)
  
  \(m = \text{number of clauses}\)
  \(n = \text{number of symbols}\)

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

  - Here:
    \(m=4, n=|\{A,B,C,D,E\}| = 5\)
    \(m/n = 4/5 = .8\)

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
    - \(D_{x,y} \leftrightarrow (P_{x+1,y} \lor P_{x-1,y} \lor P_{x,y+1} \lor P_{x,y-1})\)
    - \(S_{x,y} \leftrightarrow (W_{x,y+1} \lor W_{x+1,y} \lor W_{x-1,y} \lor W_{x,y-1})\)
  - At least one wumpus on board
    - \(W_{1,1} \lor W_{1,2} \lor \ldots \lor W_{4,1} \lor W_{4,2}\)
  - A most one wumpus on board (for any two squares, one is free)
    - \(m^2\) rules like: \(W_{i,j} \iff \neg(W_{i+1,j} \lor W_{i,j+1} \lor \ldots \lor W_{i,j-1})\)
  - 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:
    - proposition symbol
    - (conjunction of symbols) \( \Rightarrow \) symbol
  - E.g.: \( A \land B \Rightarrow B \land C \lor D \Rightarrow B \)
  - **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
  \[
  \alpha_1, \ldots, \alpha_n, \alpha_1 \land \ldots \land \alpha_n \Rightarrow \beta
  \]

- 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:
    \[
    P \Rightarrow Q \\
    L \land M \Rightarrow P \\
    B \land L \Rightarrow M \\
    A \land P \Rightarrow L \\
    A \land B \Rightarrow L \\
    A \\
    B
    \]

---

Forward chaining algorithm

Function: `PL-PF-EXTEND(KB)` that returns true or false

- Local variables: `count`, a table, indexed by clause, initially the number of premises in formula, `infered`, a table indexed by symbol, each entry initially false
- `agenda`: a list of symbols, initially the symbols known to be true
  - `agenda` is not empty do
    - `p = PL-PF-AGENDA()`
    - `unileeferd = infered[p]`
    - `count[p] = count[p] + 1`
    - `for each Horn clause c in whose premise p appears do`
      - `count[c] = count[c] + 1`
      - `if count[c] = 0 then do`
      - `if infered[c] = true then return true`
      - `P = pl-PF-AGENDA()`
    - `return false`

---

Forward chaining example

Diagram: Graphical representation of the process of applying modus ponens to Horn clauses.

---

Forward chaining example

Diagram: Another example illustrating the application of forward chaining.
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 $m$
   \[ A_1 \land \ldots \land A_k \rightarrow 0 \]
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

Backward chaining example

Backward chaining example

Backward chaining example

Front 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