Outline

- Automated Propositional Proof Methods
  1. Resolution
  2. A Practical Method: Walksat
  3. Proof Methods for Horn Clauses
     — Forward and Backward Chaining

Proof Methods for Propositional Logic

Outline

- Automated Propositional Proof Methods
  1. Resolution
  2. A Practical Method: Walksat
  3. Proof Methods for Horn Clauses
     — Forward and Backward Chaining

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
     - 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)

\[ \text{conjunction} \text{ of clauses of disjunctions of literals and negated literals} \]

\[ (A \lor \neg B) \lor (B \lor \neg C) \lor (C \lor \neg D) \]

Resolution inference rule (for CNF):

\[ l_1 \lor \cdots \lor l_i \lor \cdots \lor m_1 \lor \cdots \lor m_j \]

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

\[ e.g. \quad P_{1,3} \lor P_{2,2} \lor \neg P_{2,2} \]

Resolution is sound and complete for propositional logic

Soundness of resolution inference rule

If \( \xi = \neg m_j \)

\[ (\xi \lor \cdots \lor \xi_i \lor \cdots \lor \xi_l) \Rightarrow \xi \]

\[ \neg \xi \Rightarrow (m_1 \lor \cdots \lor m_{i-1} \lor m_{i+1} \lor \cdots \lor m_j) \]

\[ (\neg l_i \lor \cdots \lor \neg \xi_i \lor \cdots \lor \neg \xi_l) \Rightarrow (m_1 \lor \cdots \lor m_{i-1} \lor m_{i+1} \lor \cdots \lor m_j) \]

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

Review: Validity and satisfiability

A sentence is valid if it is true in all models,

e.g. True, \( A \lor \neg A, A \Rightarrow A, (A \land (A \Rightarrow B)) \Rightarrow B \)

Validity is connected to inference via the Deduction Theorem:

\[ KB \vdash \alpha \text{ if and only if } (KB \Rightarrow \alpha) \text{ is valid} \]

A sentence is satisfiable if it is true in some model

e.g. \( A \lor B \lor C \)

A sentence is unsatisfiable if it is false in all models

e.g. \( A \land \neg A \land B \land \neg B \land C \land \neg C \land \neg B \lor B \lor A \land \neg A \lor A \lor \neg A \)

Satisfiability is connected to inference via the following:

\[ KB \vdash \alpha \text{ if and only if } (KB \land \neg \alpha) \text{ is unsatisfiable} \]

(there is no model for which KB=true and \( \alpha \) is false)
Proof by Resolution: Proof by contradiction

- I.E.: prove \( \alpha \) by showing \( KB \land \neg \alpha \) unsatisfiable
- Example: \( KB = (B_{1,1} \iff (P_{1,2} \lor P_{2,1})) \land \neg B_{1,1} \)
  - Prove \( \neg P_{1,2} \)
- KB in Conjunctive Normal Form:
  \[ (\neg B_{1,1} \lor P_{1,2} \lor P_{2,1}) \land (\neg P_{1,2} \lor B_{1,1}) \land (\neg P_{2,1} \lor B_{1,1}) \land \neg B_{1,1} \]
- Negate \( \alpha \): \( P_{1,2} \)

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 \((\alpha \implies \beta) \land (\beta \implies \alpha)\).
   
   \[ \begin{aligned}
   (B_{1,1} \implies (P_{1,2} \lor P_{2,1})) \land ((P_{1,2} \lor P_{2,1}) \implies B_{1,1})
   
   \end{aligned} \]

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

   \[ \begin{aligned}
   \neg B_{1,1} \lor P_{1,2} \lor P_{2,1}
   
   \end{aligned} \]

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

   \[ \begin{aligned}
   (\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})
   
   \end{aligned} \]

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

   \[ \begin{aligned}
   (\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})
   
   \end{aligned} \]

For convenience: Logical equivalence

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

Resolution algorithm

- **Iteratively apply resolution to all pairs of clauses**

Resolution example

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

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

Function: `WalkSAT(clauses, p, max-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-flips`, number of flips allowed before giving up

1. Let `t = 1`
2. If model satisfies clauses then return model
3. Else if model satisfies clauses then return new model
4. Else flip a randomly selected clause from a randomly selected symbol with probability `p`
5. If model is satisfied then return model
6. Else flip a randomly selected clause from a randomly selected symbol
7. Return failure.

Hard satisfiability problems

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

  \[
m = \text{number of clauses}
  \]

  \[
n = \text{number of symbols}
  \]

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

- Here:
  
  \[
m = 4, n = \{A, B, C, D, E\} = 5
  \]

  \[
m/n = 4/5 = .8
  \]

Encoding Wumpus in propositional logic

- 4x4 Wumpus World
  
  - At least one Wumpus on the board
    
    \[W_1 \lor W_2 \lor W_3 \lor W_4 \lor W_{1,1} \lor W_{1,2} \lor W_{1,3} \lor W_{1,4}\]
  
  - At most one Wumpus on the board (for any two squares, one is free)
    
    \[\neg W_1 \lor \neg W_2 \lor \neg W_3 \lor \neg W_4\]
  
  - No instant death:
    
    \[-P_{1,1}\]
    
    \[-W_{1,1}\]
**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) ⇒ symbol
  - E.g., \( A \land B \Rightarrow A \lor C \lor D \Rightarrow B \)
- **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:

---

**Forward chaining example**

---

**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

Backward chaining example

Backward chaining example

Backward chaining example

Backward chaining example

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