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)

A sentence is valid if it is true in all models,
  e.g., \( A \lor \lnot A \lor A \rightarrow A \land (A \lor (A \rightarrow B)) \rightarrow B \)

Validity is connected to inference via the Deduction Theorem:

\( KB \models \alpha \) if and only if \( (KB \rightarrow \alpha) \) is valid

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

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

Satisfiability is connected to inference via the following:

\( KB \not\models \alpha \) if and only if \( (KB \land \lnot \alpha) \) is unsatisfiable
  (there is no model for which KB=true and \( \alpha \) is false)

Soundness of resolution inference rule

If \( \xi = \lnot \eta \)

\[ \neg((\xi \lor \ldots \lor \xi_i \lor \ldots \lor \xi_k) \Rightarrow \xi) \]

\[ \neg\eta \Rightarrow (\xi_i \lor \ldots \lor \xi_{i+1} \lor \ldots \lor \xi_k) \]

Given that

\[ (\alpha \Rightarrow \beta) = (\neg \alpha \lor \beta) \]
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 $\alpha$: $\neg P_{1,2}$
- $KB$ in Conjunctive Normal Form:
  $$(-B_{1,1} \lor P_{1,2} \lor P_{2,1}) \land (-P_{1,2} \lor B_{1,1}) \land (-P_{2,1} \lor B_{1,1})$$
- 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 $(\alpha \Rightarrow \beta) \land (\beta \Rightarrow \alpha)$.
   $$(B_{1,1} \Rightarrow (P_{1,2} \lor P_{2,1})) \land ((P_{1,2} \lor P_{2,1}) \Rightarrow B_{1,1})$$
2. Eliminate $\Rightarrow$, replacing $\alpha \Rightarrow \beta$ with $\neg \alpha \lor \beta$.
   $$(-B_{1,1} \lor P_{1,2} \lor P_{2,1}) \land (-P_{1,2} \lor P_{2,1} \lor B_{1,1})$$
3. Move $\neg$ inwards using de Morgan’s rules and (often, but not here) double-negation:
   $$(-B_{1,1} \lor P_{1,2} \lor P_{2,1}) \land ((\neg P_{1,2} \lor P_{2,1}) \land B_{1,1})$$
4. Flatten by applying distributivity law ($\land$ over $\lor$):
   $$(-B_{1,1} \lor P_{1,2} \lor P_{2,1}) \land ((\neg P_{1,2} \lor P_{2,1}) \land B_{1,1})$$

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, n, max-flips) returns a satisfying model or failure:

Inputs:
- clauses: a set of clauses in propositional logic
- n: the probability of choosing to do a "random walk" move
- max-flips: number of flips allowed before giving up

1. model← a random assignment of true/false to the symbols in clauses
2. Let r← 1 to max-flips do
3. if model satisfies clauses then return model
4. else select a randomly selected clause from clauses that is false in model
5. with probability n flip the value of a randomly selected symbol from clause
6. else flip whichever symbol in clauses maximizes the number of satisfied clauses
7. return failure.

Hard satisfiability problems

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

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

- 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
    \[-B_{1,1} \Rightarrow (P_{2,2} \lor P_{2,3} \lor P_{2,4} \lor P_{3,4})\]
    \[-S_{1,1} \Rightarrow (W_{2,2} \lor W_{2,3} \lor W_{4,2} \lor W_{4,3} \lor W_{4,4})\]
  - At least one wumpus on board
    \[-W_{1,1} \lor W_{1,2} \lor \ldots \lor W_{4,3} \lor W_{4,4}\]
  - A most one wumpus on board (for any two squares, one is free)
    \[-n^2\text{ rules like:}\]
    \[W_{1,1} \Rightarrow \neg(W_{2,1} \lor W_{2,2} \lor \ldots \lor W_{4,4})\]
  - 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) \( \Rightarrow \) symbol
  - E.g., \( A \ B \ B \Rightarrow A \ C \ D \Rightarrow B \)
- **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-FC-ENTAILS(\( q \)) returns true or false:

local variables:
  const, a table, indexed by clause, initially the number of premises
  inferred, a table, indexed by symbol, each entry initially false
 agenda, a list of symbols, initially the symbols known to be true

while agenda is not empty do
  p = Pop(agenda)
  unless inferred[p] = true
    for each Horn clause \( c \) whose premise \( p \) appears do
      e = \( c \land \neg p \Rightarrow \beta \)
      if \( e \land \neg p \) then do
        if Head(e) \( \equiv q \) then return true
        P = P \land Head(e)
        agenda += P
      end
    end
  end
return false
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 $KB \vdash q$, $q$ 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
**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**