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)
Applies to a DB of Sentences in **Conjunctive Normal Form** (CNF) conjunction of clauses of disjunctions of literals and negated literals

E.g., \((A \lor \neg B) \land (B \lor \neg C \lor \neg D)\)

- **Resolution** inference rule (for CNF):

  \[
  \begin{align*}
  \ell_1 \lor \ldots \lor \ell_{i-1} \lor \ell_i \lor \ell_{i+1} \lor \ldots \lor \ell_k, & \quad m_1 \lor \ldots \lor m_{j-1} \lor m_j \lor m_{j+1} \lor \ldots \lor m_n \\
  \ell_1 \lor \ldots \lor \ell_{i-1} \lor \ell_{i+1} \lor \ldots \lor \ell_k & \quad m_1 \lor \ldots \lor m_{j-1} \lor m_{j+1} \lor \ldots \lor m_n
  \end{align*}
  \]

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

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

  \[
  \begin{align*}
  \frac{P_{1,3} \lor P_{2,2}, \quad \neg P_{2,2}}{P_{1,3}}
  \end{align*}
  \]

- Resolution is **sound** and **complete** for propositional logic

CIS 521-Intro to AI
Soundness of resolution inference rule

if \( \xi = \neg m_j \)

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

\[ \neg m_j \Rightarrow (m_1 \lor \ldots \lor m_{j-1} \lor m_{j+1} \lor \ldots \lor m_n) \]

\[ \neg(\xi \lor \ldots \lor \xi_{i-1} \lor \xi_{i+1} \lor \ldots \lor \xi_k) \Rightarrow (m_1 \lor \ldots \lor m_{j-1} \lor m_{j+1} \lor \ldots \lor m_n) \]

Given that

\[(\alpha \Rightarrow \beta) \equiv (\neg \alpha \lor \beta)\]
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**: 
\[
KB \models \alpha \text{ if and only if } (KB \Rightarrow \alpha) \text{ is valid}
\]

A sentence is **satisfiable** if it is true in some model e.g. A \( \lor B \), 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: 
\[
KB \models \alpha \text{ if and only if } (KB \land \neg \alpha) \text{ is unsatisfiable}
\]

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

- Proof by contradiction, i.e., prove $\alpha$ by showing $\text{KB} \land \neg \alpha$ unsatisfiable

```plaintext
function PL-RESOLUTION(KB, $\alpha$) returns true or false

  clauses $\leftarrow$ the set of clauses in the CNF representation of $\text{KB} \land \neg \alpha$
  new $\leftarrow \{\}$
  loop do
    for each $C_i, C_j$ in clauses do
      resolvents $\leftarrow$ PL-RESOLVE($C_i, C_j$)
      if resolvents contains the empty clause then return true
      new $\leftarrow$ new $\cup$ resolvents
    if new $\subseteq$ clauses then return false
    clauses $\leftarrow$ clauses $\cup$ new
```

$A \land \neg A$ will have just been resolved
Resolution example

- $KB = (B_{1,1} \iff (P_{1,2} \lor P_{2,1})) \land \neg B_{1,1}$  
  $\alpha = \neg P_{1,2}$
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 \( (\alpha \implies \beta) \land (\beta \implies \alpha) \).
   
   \[
   (B_{1,1} \implies (P_{1,2} \lor P_{2,1})) \land ((P_{1,2} \lor P_{2,1}) \implies B_{1,1})
   \]

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** \( \neg \text{ inwards} \) 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 \( \lor \)):
   
   \[
   (\neg B_{1,1} \lor P_{1,2} \lor P_{2,1}) \land (\neg P_{1,2} \lor B_{1,1}) \land (\neg P_{2,1} \lor B_{1,1})
   \]
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 \( \beta \models \alpha \)

\[
\begin{align*}
(\alpha \land \beta) & \equiv (\beta \land \alpha) \quad \text{commutativity of } \land \\
(\alpha \lor \beta) & \equiv (\beta \lor \alpha) \quad \text{commutativity of } \lor \\
((\alpha \land \beta) \land \gamma) & \equiv (\alpha \land (\beta \land \gamma)) \quad \text{associativity of } \land \\
((\alpha \lor \beta) \lor \gamma) & \equiv (\alpha \lor (\beta \lor \gamma)) \quad \text{associativity of } \lor \\
\neg(\neg\alpha) & \equiv \alpha \quad \text{double-negation elimination} \\
(\alpha \Rightarrow \beta) & \equiv (\neg\beta \Rightarrow \neg\alpha) \quad \text{contraposition} \\
(\alpha \Rightarrow \beta) & \equiv (\neg\alpha \lor \beta) \quad \text{implication elimination} \\
(\alpha \iff \beta) & \equiv ((\alpha \Rightarrow \beta) \land (\beta \Rightarrow \alpha)) \quad \text{biconditional elimination} \\
\neg(\alpha \land \beta) & \equiv (\neg\alpha \lor \neg\beta) \quad \text{de Morgan} \\
\neg(\alpha \lor \beta) & \equiv (\neg\alpha \land \neg\beta) \quad \text{de Morgan} \\
(\alpha \land (\beta \lor \gamma)) & \equiv ((\alpha \land \beta) \lor (\alpha \land \gamma)) \quad \text{distributivity of } \land \text{ over } \lor \\
(\alpha \lor (\beta \land \gamma)) & \equiv ((\alpha \lor \beta) \land (\alpha \lor \gamma)) \quad \text{distributivity of } \lor \text{ over } \land
\end{align*}
\]
The \textit{WalkSAT} algorithm

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

\begin{verbatim}
function \texttt{WalkSAT}(\texttt{clauses}, p, \texttt{max-flips}) returns a satisfying model or \texttt{failure}
inputs: \texttt{clauses}, a set of clauses in propositional logic
        \textit{p}, the probability of choosing to do a "random walk" move
        \textit{max-flips}, number of flips allowed before giving up

\texttt{model} \leftarrow a random assignment of \texttt{true}/\texttt{false} to the symbols in \texttt{clauses}
for \texttt{i} = 1 to \texttt{max-flips} do
    if \texttt{model} satisfies \texttt{clauses} then return \texttt{model}
    \texttt{clause} \leftarrow a randomly selected clause from \texttt{clauses} that is false in \texttt{model}
    with probability \textit{p} flip the value in \texttt{model} of a randomly selected symbol
    from \texttt{clause}
    else flip whichever symbol in \texttt{clause} maximizes the number of satisfied clauses

return \texttt{failure}
\end{verbatim}
Hard satisfiability problems

- Consider random 3-CNF sentences. e.g.,

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

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

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

- Here:
  \[m=4, \ n=|\{A,B,C,D,E\}| = 5\]
  \[m/n = 4/5 = 0.8\]
Hard satisfiability problems
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
    - \( B_{x,y} \iff (P_{x,y+1} \lor P_{x,y-1} \lor P_{x+1,y} \lor P_{x-1,y}) \)
    - \( S_{x,y} \iff (W_{x,y+1} \lor W_{x,y-1} \lor W_{x+1,y} \lor W_{x-1,y}) \)
  - At least one wumpus on board
    - \( W_{1,1} \lor W_{1,2} \lor \cdots \lor W_{4,3} \lor W_{4,4} \)
  - A most one wumpus on board (for any two squares, one is free)
    - \( n^2 \) rules like: \( W_{1,1} \implies \neg(W_{1,2} \lor W_{1,3} \lor \cdots \lor W_{4,4}) \)
  - No instant death:
    - \( \neg P_{1,1} \)
    - \( \neg W_{1,1} \)
function PL-Wumpus-Agent( percept ) returns an action

inputs: percept, a list, [stench, breeze, glitter]

static: KB, initially containing the "physics" of the wumpus world
    x, y, orientation, the agent's position (init. [1,1]) and orient. (init. right)
    visited, an array indicating which squares have been visited, initially false
    action, the agent's most recent action, initially null
    plan, an action sequence, initially empty

update x, y, orientation, visited based on action
if stench then TELL(KB, S_{x,y}) else TELL(KB, ¬ S_{x,y})
if breeze then TELL(KB, B_{x,y}) else TELL(KB, ¬ B_{x,y})
if glitter then action ← grab
else if plan is nonempty then action ← POP(plan)
else if for some fringe square [i, j], ASK(KB, (¬ P_{i,j} ∧ ¬ W_{i,j})) is true or
    for some fringe square [i, j], ASK(KB, (P_{i,j} ∨ W_{i,j})) is false then do
        plan ← A*-Graph-Search(Route-PB([x, y], orientation, [i, j], visited))
        action ← POP(plan)
else action ← a randomly chosen move
return action
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.: $A \land B \Rightarrow A \land C \land D \Rightarrow B$

- **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
  \[
  \alpha_1, \ldots, \alpha_n, \alpha_1 \land \ldots \land \alpha_n \Rightarrow \beta \\
  \beta
  \]

- **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 algorithm

```
function PL-FC-ENTAILS?(KB, q) returns true or false
    local variables: count, 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] do
            inferred[p] ← true
            for each Horn clause c in whose premise p appears do
                decrement count[c]
                if count[c] = 0 then do
                    if HEAD[c] = q then return true
                    PUSH(HEAD[c], agenda)
        
    return false
```
Forward chaining example
Forward chaining example
Forward chaining example
Forward chaining example
Forward chaining example
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 $m$
   \[ a_1 \land \ldots \land a_k \Rightarrow b \]
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
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