Inference Techniques for Logical Reasoning

Inference/Proof Techniques

- Two kinds (roughly):
  - Model checking
    - Truth table enumeration (always exponential in $n$)
    - Efficient backtracking algorithms, e.g., Davis-Putnam-Logemann-Loveland (DPLL)
    - Local search algorithms (sound but incomplete)
      e.g., randomized hill-climbing (WalkSAT)
  - Successive application of inference rules
    - Generate new sentences from old in a sound way
    - Proof = a sequence of inference rule applications
    - Use inference rules as successor function in a standard search algorithm
Inference Technique I: Resolution

Terminology:
Literal = proposition symbol or its negation
E.g., A, ¬A, B, ¬B, etc.
Clause = disjunction of literals
E.g., (B ∨ ¬C ∨ ¬D)

Resolution assumes sentences are in
Conjunctive Normal Form (CNF):
sentence = conjunction of clauses
E.g., (A ∨ ¬B) ∧ (B ∨ ¬C ∨ ¬D)

Conversion to CNF

E.g., B₁₁ ⇔ (P₁₂ ∨ P₂₁)

1. Eliminate ⇔, replacing α ⇔ β with (α ⇒ β) ∧ (β ⇒ α).
   (B₁₁ ⇒ (P₁₂ ∨ P₂₁)) ∧ ((P₁₂ ∨ P₂₁) ⇒ B₁₁)

2. Eliminate ⇒, replacing α ⇒ β with ¬α ∨ β.
   (¬B₁₁ ∨ P₁₂ ∨ P₂₁) ∧ (¬(P₁₂ ∨ P₂₁) ∨ B₁₁)

3. Move ¬ inwards using de Morgan’s rules and double-negation:
   (¬B₁₁ ∨ P₁₂ ∨ P₂₁) ∧ (¬(¬P₁₂ ∧ ¬P₂₁) ∨ B₁₁)

4. Apply distributivity law (∧ over ∨) and flatten:
   (¬B₁₁ ∨ P₁₂ ∨ P₂₁) ∧ (¬P₁₂ ∨ B₁₁) ∧ (¬P₂₁ ∨ B₁₁)

   This is in CNF – Done!
Resolution motivation

There is a pit in [1,3] or
There is a pit in [2,2] 
There is no pit in [2,2]

= there is a pit in [1,3]

More generally,
\[
\frac{\ell_1 \lor \ldots \lor \ell_k, \quad \neg \ell_i}{\ell_1 \lor \ldots \lor \ell_{i-1} \lor \ell_{i+1} \lor \ldots \lor \ell_k}
\]

Inference Technique: Resolution

• General Resolution inference rule (for CNF):

  \[
  \frac{\ell_1 \lor \ldots \lor \ell_k, \quad m_1 \lor \ldots \lor m_n}{\ell_1 \lor \ldots \lor \ell_{i-1} \lor \ell_{i+1} \lor \ldots \lor \ell_k \lor m_1 \lor \ldots \lor m_{j-1} \lor m_{j+1} \lor \ldots \lor m_n}
  \]

  where \(\ell_i\) and \(m_j\) are complementary literals.

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

  \(P_{1,3}\)

• Resolution is sound
  for propositional logic
Resolution

Soundness of resolution inference rule:

\[ \neg (l_1 \lor \ldots \lor l_{i-1} \lor l_i \lor \ldots \lor l_k) \Rightarrow l_i \]

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

\[ \neg (l_1 \lor \ldots \lor l_{i-1} \lor l_i \lor \ldots \lor l_k) \Rightarrow (m_1 \lor \ldots \lor m_{j-1} \lor m_{j+1} \lor \ldots \lor m_n) \]

(since \( l_i = \neg m_j \))

Resolution algorithm

- To show \( KB \models \alpha \), use proof by contradiction, i.e., show \( KB \land \neg \alpha \) unsatisfiable

```python
function PL-Resolution(KB, \alpha) returns true or false
    clauses ← the set of clauses in the CNF representation of KB \land \neg \alpha
    new ← { }
    loop do
        for each \( C_i, C_j \) in clauses do
            resolvents ← PL-Resolve(\( C_i, C_j \))
            if resolvents contains the empty clause then return true
            new ← new \cup resolvents
        if new ⊆ clauses then return false
        clauses ← clauses \cup new
```

Resolution example

Given no breeze in [1,1], prove there's no pit in [1,2]

\[ KB = (B_{1,1} \iff (P_{1,2} \lor P_{2,1})) \land \neg B_{1,1} \text{ and } \alpha = \neg P_{1,2} \]

Resolution: Convert to CNF and show KB \land \neg \alpha is unsatisfiable
Resolution example

Empty clause
(i.e., $KB \land \neg \alpha$ unsatisfiable)

Inference Technique II: Forward/Backward Chaining

- Require sentences to be in Horn Form:
  - $KB = \text{conjunction of Horn clauses}$
  - Horn clause =
    - proposition symbol or
    - “(conjunction of symbols) $\Rightarrow$ symbol”
      (i.e. clause with at most 1 positive literal)
  - e.g., $KB = C \land (B \Rightarrow A) \land (C \land D \Rightarrow B)$
- F/B chaining based on “Modus Ponens” rule:
  \[
a_1, \ldots, a_n, \quad a_1 \land \ldots \land a_n \Rightarrow \beta
  \]
  \[
  \beta
  \]
  - Complete for Horn clauses
- Very natural and linear time complexity in size of $KB$
Forward chaining

- Idea: fire any rule whose premises are satisfied in KB, add its conclusion to KB, until query q is found

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

Query = "Is Q true?"  
AND-OR Graph

Forward chaining algorithm

```python
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 is sound & complete for Horn KB
Forward chaining example

Query = Q
(i.e. “Is Q true?”)
Forward chaining example
Forward chaining example
Backward chaining

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

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

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

Backward chaining example

![Diagram of 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
- FC may do lots of work that is irrelevant to the goal
- BC is goal-driven, appropriate for problem-solving, e.g., How do I get an A in this class? e.g., What is my best exit strategy out of the classroom? e.g., How can I impress my date tonight?
- Complexity of BC can be much less than linear in size of KB

Efficient propositional inference

Two families of efficient algorithms for propositional inference based on model checking:

Complete backtracking search algorithms
  DPLL algorithm (Davis, Putnam, Logemann, Loveland)
  Similar to TT enumeration from last class but with heuristics

Incomplete local search algorithms
  WalkSAT algorithm
The DPLL algorithm

Determine if an input propositional logic sentence (in CNF) is satisfiable.

Improvements over truth table enumeration:

1. Early termination
   A clause is true if any literal is true.
   A sentence is false if any clause is false.

2. Pure symbol heuristic
   Pure symbol: always appears with the same "sign" in all clauses.
   e.g., In the three clauses \((A \lor \neg B), (\neg B \lor \neg C), (C \lor A)\), \(A\) and \(B\) are pure, \(C\) is impure.
   Make a pure symbol literal true.

3. Unit clause heuristic
   Unit clause: only one literal in the clause
   The only literal in a unit clause must be true.

The DPLL algorithm

function DPLL-SATISFIABLE?(s) returns true or false
inputs: s, a sentence in propositional logic
clauses ← the set of clauses in the CNF representation of s
symbols ← a list of the proposition symbols in s
return DPLL(clauses, symbols, [])

function DPLL(clauses, symbols, model) returns true or false
if every clause in clauses is true in model then return true
if some clause in clauses is false in model then return false
P, value ← Find-Pure-Symbol(symbols, clauses, model)
if P is non-null then return DPLL(clauses, symbols-P, |P = value|model)
P, value ← Find-Unit-Clause(clauses, model)
if P is non-null then return DPLL(clauses, symbols-P, |P = value|model)
P ← First(symbols), rest ← Rest(symbols)
return DPLL(clauses, rest, |P = true|model) or
       DPLL(clauses, rest, |P = false|model)
Next Time

- WalkSAT
- HW #1 due
- Read Chapter 8
  First-Order Logic