Inference in PL

\(R_1: \neg P_{1,1}\)
\(R_5: B_{1,1} \iff (P_{1,2} \lor P_{2,1})\)
\(R_3: B_{2,1} \iff (P_{1,1} \lor P_{2,2} \lor P_{3,1})\)
\(R_4: \neg B_{1,1}\)
\(R_9: B_{2,1} \land (P_{2,1} \lor P_{2,2})\)
\(R_8: (B_{1,1} \Rightarrow (P_{2,1} \lor P_{2,2}) \land (P_{1,2} \lor P_{2,2} \lor P_{3,1}))\)
\(R_7: ((P_{2,1} \lor P_{2,2}) \Rightarrow B_{1,1})\)
\(R_6: (\neg B_{1,1} \Rightarrow (P_{2,1} \lor P_{2,2}))\)
\(R_{10}: \neg P_{1,2} \land \neg P_{2,1}\)

(a): moving back to [1,2] from [2,1] stench and no breeze:
\(R_{11}: \neg B_{1,2}\)
\(R_{12}: B_{2,2} \iff (P_{1,1} \lor P_{2,2} \lor P_{3,1})\)

similar prior \(R_{10}\)

(b)

Resolution

**Conjunctive Normal Form (CNF):** conjunction of disjunctions of literals

\[\text{clauses}\]

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

- **Unit resolution** inference rule (for CNF):
  \[\xi \lor \ldots \lor \xi_k, m_1 \lor \ldots \lor \xi_k\]
  where \(\xi\) and \(m\) are complementary literals.

- **Full Resolution** inference rule (for CNF):
  \[\xi \lor \ldots \lor \xi_k, m_1 \lor \ldots \lor m_n\]
  where \(\xi\) and \(m\) are complementary literals.

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

Resolution is sound and complete for propositional logic.
Resolution

- Factoring: Resulting clause should only contain one copy of a literal:
  Elimination of multiple literal copies is called factoring (not necessary for
  set notation of clauses).
- Soundness of resolution inference rule:
  \[ \neg (\xi \lor \ldots \lor \xi_1 \lor \xi_1+1 \lor \ldots \lor \xi_k) \Rightarrow \xi \]
  \[ \neg \neg m \Rightarrow (m_1 \lor \ldots \lor m_{j-1} \lor m_{j+1} \lor \ldots \lor m_n) \]
- Resolution is basis of a whole family of complete inference procedures:
  - Every complete search algorithm which uses resolution can derive any
    conclusion entailed by any knowledge base in PL.
- Resolution is **refutation complete**:
  - Given that \( A \) is true, it can be used to answer if \( A \lor B \) is true but not to
    generate the consequence \( A \lor B \)

Conversion to CNF

- Every sentence in PL is equivalent to a conjunction of disjunction of literals.
- A sentence in k-CNF has exactly k literals per clause.

Example transformation for: \( B_{1,1} \leftrightarrow (P_{1,2} \lor P_{2,1}) \)

1. Eliminate \( \Leftrightarrow \), replacing \( \alpha \leftrightarrow \beta \) 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 \( \neg \alpha \lor \beta \).
   \[ (\neg B_{1,1} \lor P_{1,2} \lor P_{2,1}) \land (\neg (P_{1,2} \lor P_{2,1}) \lor B_{1,1}) \]
3. Move \( \neg \) inwards using de Morgan’s rules and 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. Apply distributivity law (\( \land \) over \( \lor \)) and flatten:
   \[ (\neg B_{1,1} \lor P_{1,2} \lor P_{2,1}) \land (\neg P_{1,2} \lor B_{1,1}) \land (\neg P_{2,1} \lor B_{1,1}) \]
Resolution algorithm

- 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 \subseteq clauses then return false
        clauses ← clauses \cup new
    end loop
    return false
```

Resolution example

- $KB = (B_{1,1} \iff (P_{1,2} \lor P_{2,1})) \land \neg B_{1,1} \land \neg \alpha = \neg P_{1,2}$
Forward and backward chaining

- **Horn Form** (restricted)
  - KB = conjunction of Horn clauses
  - Horn clause =
    - proposition symbol; or
    - (conjunction of symbols) ⇒ symbol
  - E.g., C ∧ (B ⇒ A) ∧ (C ∧ D ⇒ B)
- **Modus Ponens** (for Horn Form): complete for Horn KBs
  \[ \alpha_1, \ldots, \alpha_n, \alpha_1 \land \ldots \land \alpha_n \Rightarrow \beta \]
- Can be used with **forward chaining** or **backward chaining**.
- These algorithms are very natural and run in linear time.

Forward chaining

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

\[ P \Rightarrow Q \\
L \land M \Rightarrow P \\
B \land L \Rightarrow M \\
A \land P \Rightarrow L \\
A \land B \Rightarrow L \\
A \\
B \]
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
        end if
      end for
    end if
    Push(Head[c], agenda)
  end while
  return false

• Forward chaining is sound and complete for Horn KB

Forward chaining example

\[
P \Rightarrow Q \\
L \land M \Rightarrow P \\
B \land L \Rightarrow M \\
A \land P \Rightarrow L \\
A \land B \Rightarrow L \\
A \\
B
\]
Forward chaining example

\[ P \Rightarrow Q \]
\[ L \land M \Rightarrow P \]
\[ B \land L \Rightarrow M \]
\[ A \land P \Rightarrow L \]
\[ A \land B \Rightarrow L \]
\[ A \]
\[ B \]
Forward chaining example

- $P \Rightarrow Q$
- $L \land M \Rightarrow P$
- $B \land L \Rightarrow M$
- $A \land P \Rightarrow L$
- $A \land B \Rightarrow L$
- $A$
- $B$
Forward chaining example

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

Forward chaining example

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

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

$$P \Rightarrow Q$$
$$L \land M \Rightarrow P$$
$$B \land L \Rightarrow M$$
$$A \land P \Rightarrow L$$
$$A \land B \Rightarrow L$$
$$A$$
$$B$$
Backward chaining example

\[ P \Rightarrow Q \]
\[ L \land M \Rightarrow P \]
\[ B \land L \Rightarrow M \]
\[ A \land P \Rightarrow L \]
\[ A \land B \Rightarrow L \]
\[ A \]
\[ B \]
Backward chaining example

\[ P \Rightarrow Q \]
\[ L \land M \Rightarrow P \]
\[ B \land L \Rightarrow M \]
\[ A \land P \Rightarrow L \]
\[ A \land B \Rightarrow L \]
\[ A \]
\[ B \]
Backward chaining example

\[ P \Rightarrow Q \]
\[ L \land M \Rightarrow P \]
\[ B \land L \Rightarrow M \]
\[ A \land P \Rightarrow L \]
\[ A \land B \Rightarrow L \]
\[ A \]
\[ B \]
Backward chaining example

\[ P \Rightarrow Q \]
\[ L \land M \Rightarrow P \]
\[ B \land L \Rightarrow M \]
\[ A \land P \Rightarrow L \]
\[ A \land B \Rightarrow L \]
\[ A \]
\[ B \]
Backward chaining example

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

Forward 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
Efficient propositional inference

Two families of efficient algorithms for propositional inference:

Complete backtracking search algorithms
- DPLL algorithm (Davis, Putnam, Logemann, Loveland)
- 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 ∨ ¬B), (¬B ∨ ¬C), (C ∨ 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)
```

The WalkSAT algorithm

• Incomplete, local search algorithm
• Evaluation function: The min-conflict heuristic of minimizing the number of unsatisfied clauses
• Balance between greediness and randomness
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
model ← a random assignment of true/false to the symbols in clauses
for i = 1 to max-flips do
  if model satisfies clauses then return model
  clause ← a randomly selected clause from clauses that is false in model
  with probability p flip the value in model of a randomly selected symbol
  from clause
  else flip whichever symbol in clause maximizes the number of satisfied clauses
return failure

Hard satisfiability problems

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

  \(m = \text{number of clauses}\)
  \(n = \text{number of symbols}\)

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

- Median runtime for 100 **satisfiable** random 3-CNF sentences, $n = 50$