Resolution Algorithm

- To check whether KB $\models \alpha$, we will check whether (KB $\land \neg\alpha$) is NOT satisfiable
  - Convert (KB $\land \neg\alpha$) into CNF and let S = Set of all clauses in the CNF
  - Apply the resolution rule whenever possible and keep on adding them to S
    - If at any point of time, Empty Clause is added to S then return **KB entails α**
    - If there are no new clauses that can be added to S, then return **KB does not entail α**

- Empty clause - Disjunction of 0 literals
  - Will be obtained when we resolve P and ¬P
  - Empty Clause is not satisfiable (because, for a disjunction to be true, at least one of the disjuncts should be true)

Resolution Algorithm

- For any formula α if the algorithm derives an empty clause then α is not satisfiable.
  - By induction on the number of steps needed to derive the empty clause.
  - Base case : If it takes one step then S contains two clauses of the form { P } and { ¬P } Hence α (same as S) not satisfiable.
  - Induction Step : S → S' → ...... → S^0 where S' to S^0 takes n-1 steps and S^o contains empty set. So By induction, S' is not satisfiable. Suppose S is satisfiable, let M be the model such that M ⊨ S.

    We can argue that M ⊨ S' which is a contradiction. Pick an arbitrary clause C from S'
    - If C is already in S then M ⊨ C
    - Otherwise, C is obtained because of resolution rule applied to two clauses A and B from S.
      $A = \ell_1 \vee \ell_2 \vee \ldots \ell_{i-1} \vee \ell_i \vee \ell_{i+1} \vee \ldots \ell_k$
      $B = m_1 \vee m_2 \vee \ldots m_{j-1} \vee m_j \vee m_{j+1} \vee \ldots m_n$
      $C = \ell_1 \vee \ldots \ell_{i-1} \vee \ell_{i+1} \vee \ldots \ell_k \vee m_1 \vee \ldots m_{j-1} \vee m_{j+1} \vee \ldots m_n$
    - By assumption, M ⊨ A and M ⊨ B.
    - Note that M assigns either ℓ_i to True or m_j to True
    - In both cases M ⊨ C.

Resolution Algorithm

- To check whether KB $\models \alpha$, we will check whether (KB $\land \neg\alpha$) is NOT satisfiable
    - Convert (KB $\land \neg\alpha$) into CNF and let S = Set of all clauses in the CNF
    - Apply the resolution rule whenever possible and keep on adding them to S
        - If at any point of time, Empty Clause is added to S then return **KB entails α**
        - If there are no new clauses that can be added to S, then return **KB does not entail α**

- Soundness: If the Algorithm returns KB entails α then it is actually the case
    - Algorithm returns KB entails α only when Empty Clause is derived for (KB $\land \neg\alpha$)
    - This implies that (KB $\land \neg\alpha$) is unsatisfiable, hence KB entails α

Resolution Algorithm

- Completeness:

  For any KB and **α** if KB entails **α** then the Algorithm returns KB entails **α**

  - Same as proving: If (KB ∧ ¬**α**) is unsatisfiable then Empty Clause is added to S at some point

- Ground Resolution Theorem : If a set of Clauses is unsatisfiable then the Resolution of those clause will contain empty clause
  - Proof by Contraposition:
    - Let S be a set of clauses and let RC(S) be the set of all clauses in the Resolution Closure of S
    - Note that S is contained in RC(S)
    - We will prove that if RC(S) does not contain empty clause then S is satisfiable

Resolution Algorithm : Completeness

- Assume that RC(S) does not contain empty clause
  - Let $P_1 P_2$ ....P_k be the set of all atomic sentences that occur in S
  - For i = 1 to k
    - If there is a clause C in RC(S) such that $\neg P_i$ is a literal in C and all other literals of C are FALSE under the assignment chosen for $P_1 P_2$ ....P_{i-1} then assign P_i = FALSE
    - Otherwise, assign P_i = TRUE

- This model satisfies all the clauses in RC(S). Suppose not. Then:
  - Choose the smallest i such that assignment of P_i causes some clause C in RC(S) to becomes FALSE
  - Hence all other literals of C have already been assigned to FALSE. So, at step i the clause C looks like $(\ell_1 \vee \ell_2 \vee \ldots \ell_k \vee P_i)$ or $(m_1 \vee m_2 \vee \ldots m_n \vee \neg P_i)$ where each ℓ_i (or) m_j is is a literal over $P_1 P_2$ ....P_{i-1} and the literal is assigned to FALSE
  - Now if just one of the above two clauses were present in RC(S) then the algorithm would assign P_i appropriately so that the clause is satisfied
  - If both are present then using P_i to resolve the two clauses we get $(\ell_1 \vee \ell_2 \vee \ldots \ell_k \vee m_1 \vee m_2 \vee \ldots m_n)$
  - This new clause would be FALSE with the current assignment for P_1 P_2 ....P_{i-1} (Contradiction to minimality of i)

Resolution Algorithm

- To check whether KB $\models \alpha$, we will check whether (KB $\wedge \neg\alpha$) is NOT satisfiable
  - Convert (KB $\wedge \neg\alpha$) into CNF and let S = Set of all clauses in the CNF
  - Apply the resolution rule whenever possible and keep on adding them to S
    - If at any point of time, Empty Clause is added to S then return **KB entails α**
    - If there are no new clauses that can be added to S, then return **KB does not entail α**

- Sound and Complete Algorithm for making inferences
- Worst Case : Exponential Time
  - Can it be made better?
    - Most likely No (Open problem)

Horn Clauses

- In many real world applications, the propositional sentences have a particular structure
  - We can have more efficient algorithm to check for entailment for such formulas
- Definite Clause : Disjunction of literals where exactly one literal is positive
- Horn Clause : Disjunction of literals where at most one literal is positive
- Examples:
  - $(\neg P_{1,1} \vee \neg W_{1,1} \vee B_{1,1})$ is a definite clause $(\neg P_{1,1} \vee W_{1,1} \vee B_{1,1})$ is not a definite clause
  - $(\neg P_{1,1} \vee \neg W_{1,1} \vee \neg B_{1,1})$ is not a definite clause but it is a horn clause
- Every Definite clause is a Horn Clause

Horn Clauses

- Every Definite Clause can be written as implication whose premise is a conjunction of positive literals and conclusion is a positive literal
  - (¬WumpusAhead ∨ ¬Arrow ∨ Shoot) can be written as (WumpusAhead ∧ Arrow) ⇒ Shoot
  - Can you write a single literal in implication form?
    - $B_{1,1}$ can be written as $T \Rightarrow B_{1,1}$
- Can we write Horn Clauses in implicational form?
  - Yes
  - How to write a clause with only negative literals in implication form?
    - $(\neg P_{1,1} \vee \neg W_{1,1} \vee \neg B_{1,1})$ can be written as $(P_{1,1} \wedge W_{1,1} \wedge B_{1,1}) \Rightarrow \bot$

- In a Horn clause in its implicational form:
  - Premise is called the body of the clause
  - Conclusion is called the head of the clause
  - Clause with a single literal is called a fact
- Horn clauses are closed under Resolution
  - Resolving Horn clauses will give always result in Horn clauses
- Entailment for Horn formulas can be done in Linear Time in the size of the KB
  - Forward Chaining
  - Backward Chaining

Horn Clauses : Forward Chaining

- Given a KB (Set of Horn clauses) and a Proposition P, does KB $\models$ P ?
  - Start with S as the set of all facts in KB
  - Repeat until S saturates
    - If there is a horn clause C such that all the propositions in the body of C are in S then add the head of C to S
  - If P is in S then return YES, else return No

$$P \Rightarrow Q$$
$$L \wedge M \Rightarrow P$$
$$B \wedge L \Rightarrow M$$
$$A \wedge P \Rightarrow L$$
$$A \wedge B \Rightarrow L$$
$$A$$
$$B$$

- Example:
  - Initialize S to { A,B}
  - Add L to S
  - Add M to S
  - Add P to S
  - Add Q to S
  - Finally S = { A, B, L, M, P, Q }

Horn Clauses : Forward Chaining

- Given a KB (Set of Horn clauses) and a Proposition P, does KB $\models$ P ?
    - Start with S as the set of all facts in KB
    - Repeat until S saturates
        - If there is a horn clause C such that all the propositions in the body of C are in S then add the head of C to S
    - If P is in S then return YES, else return No

- Forward Chaining is Sound
    - It is repeated application of Modus Ponens

- Forward Chaining is Complete : If P is not in S finally then KB $\not\models$ P
    - Take the final S and assign every P in S to TRUE and the rest to FALSE
    - This model satisfies KB and falsifies all propositions not in S

Horn Clauses : Forward Chaining

- Given a KB (Set of Horn clauses) and a Proposition P, does $KB \models P$?
    - Start with S as the set of all facts in KB
    - Repeat until S saturates
        - If there is a horn clause C such that all the propositions in the body of C are in S then add the head of C to S
    - If P is in S then return YES, else return No

- This is a Data-driven approach
    - S can be computed only using KB, no need of P as input
    - S can be precomputed and for any given P
    - Might be doing unnecessary computation and waste space and time

Horn Clauses : Backward Chaining

- Given a KB (Set of Horn clauses) and a Proposition P, does KB $\models$ P ?
  - Start from the Goal
  - Identify the clauses where the Goal is the head
    - Can we conclude the body of this clause?
      - Recursively do this, carefully avoiding loops

$$P \Rightarrow Q$$
$$L \wedge M \Rightarrow P$$
$$B \wedge L \Rightarrow M$$
$$A \wedge P \Rightarrow L$$
$$A \wedge B \Rightarrow L$$
$$A$$
$$B$$

- This is a Goal-driven approach
- Running time is generally less than linear time in the size of KB
    - Since it only checks for relevant clauses

Propositional Logic : Inference tools

- The problem of checking whether KB $\models \alpha$ is a coNP-complete
    - Unlikely to have efficient algorithms
- However there are tools that work very well on almost all real world input instances
    - With millions or clauses and variables
- Approaches used in these tools:
    - Search Algorithm : Local search with clever cost and neighbourhoods.
    - DPLL Algorithm : Based on resolution

Search Based Algorithms

- Start state: An arbitrary assignment
- Neighbours: 1bit flip, 2 bit-flip, …
- Cost/Value : Number of clauses satisfied by the assignment

- Useful variant of Neighbourhood search (WalkSat):
  - Randomly choose on of the following actions:
    - Choose neighbour of the current assignment that maximizes the number of satisfied clauses
    - Among the currently unsatisfied clauses, pick one clause C at random, Flip the assignment of some variable of the clause
  - Return Failure if you run out of a threshold time limit

- Takes a lot of time when the formula is unsatisfiable
  - Useful when we know for sure that the formula is satisfiabile and we want to find an assignment

- Inference: Agent can say one of the following:
  - The formula is satisfiable, here is the assignment
  - I tried for 1 hour, but I could not come up with any satisfying assignment

Resolution Based DPLL Algorithm

- WALKSAT is not complete

- DPLL algorithm is also complete and works well as a tool
  - Named after its creators : Davis, Putnam, Logemann and Loveland
  - Uses Resolution

- Features:
  - Early Detection : Returns True/False even with partial assignments

  - Unit Clause Heuristics : Unit clauses are assigned True/False with priority
    - Includes clauses where all literals are set to False except for one literal

  - Pure Symbol Heuristics : Symbols that occur only positively (or) only negatively in all unsatisfied clauses
    - Easy to set True/False for Pure symbols

Resolution Based DPLL Algorithm

- DPLL-Algorithm (Formula S)
  - C ← Clauses of S in CNF form
  - V ← Variables of S
  - Return DPLL-Find (C, V, { })

- DPLL-Find (Clauses C, unassigned variables V, Partial Model M)
  - If every clause in C is true in M then Return True
  - If there is some clause in C that is false in M then Return False
  - P,value ← FindPureSymbols (C, M)
  - If P is not empty then Return DPLL-Find (C, V \ {P}, M U {P = value})
  - P,value ← FindUnitClauses (C, M)
  - If P is not empty then Return DPLL-Find (C, V \ {P}, M U {P = value})
  - P ← First(V)
  - Return DPLL-Find(C, V \ { P}, M U { P = True}) OR DPLL-Find(C,V \ {P}, M U {P = False})

Resolution Based DPLL Algorithm

- Lot of engineering goes into the implementation
    - Component Analysis : Partition clauses into subsets that do not contain common variables and solve them separately

    - Pick variable in the last case intelligently (like the most frequently occurring variable)
        - Also pick which branch to explore first : True branch or the False branch?

    - Backtrack carefully : Go back to the relevant point instead of one step back
        - Maintain checkpoints

    - Random restarts : If it seems that there is no progress, restart with different choice

    - Clever Indexing : Use data structures that can give quick answers to questions like:
        - Set of all unsatisfied clauses where the proposition P occurs positively
        - Data structure needs to be dynamic since unsatisfied clauses keep changing

WalkSAT v/s DPLL Algorithm

- There are tools based on both these approaches (and many more)

- WalkSAT is much faster than DPLL
  - Refer book for a graph on the running time comparison

- WalkSAT is not complete (negative instances are hard to detect)

- DPLL is complete

Equipping agents with the power of inferencing

- Till now we only looked and how Inference can be done

- We need to integrate this power to the agent.
  - A drone inside a building in fire should:
    - Do such inference, figure out where to go and find an optimal path to its destination

- Agent should keep track of the percept history and use it to make inferences

Equipping agents with the power of inferencing

- The knowledge base typically includes two things
  - General rules of the framework:
    - Like when do we detect stench, Breeze, There is exactly one Wumpus, arrow can be shot only once ….
  - Knowledge based on percept history in the current scenario:
    - There is no pit in (2,1), (3,2) is safe, Wumpus is dead, Arrow is still there …

- Typically at the start Knowledge Base contains the General Rules

- As the system evolves, Agent TELLs the KB about new information based on percepts

- Handling Fluents
- Representing the Fluent axioms in the KB
- Hybrid Agent for Wumpus World
- Using Propositional Inferencing to make Plan

Current State of the world

- There are propositions whose truth values keeps changing as the current state of the world changes
  - I am sensing Stench / I have an arrow / I am facing East / ….
- Fluents : Propositions whose True/False change over time
- Atemporal variables : Propositions whose True/False is fixed
  - Location (3,1) has stench / Wumpus is in (4,3) / ….
- We cannot have a single proposition for Fluents
  - It should consider the time step also

Propositions for Fluents

- KB should have the information on how fluents are updated
  - Example : If I have an arrow at time t and my action was move forward then I will also have arrow at time t+1
  - These are called as Effect Axioms
  - $(L^0_{1,1} \wedge FacingEast^0 \wedge Forward^0) \Rightarrow (L^1_{2,1} \wedge \neg L^1_{1,2} \wedge \neg L^1_{2,2} \wedge ....)$

- Handling Fluents
- Representing the Fluent axioms in the KB
- Hybrid Agent for Wumpus World
- Using Propositional Inferencing to make Plan

Frame Problem

- $(L^0_{1,1} \wedge FacingEast^0 \wedge Forward^0) \Rightarrow (L^1_{2,1} \wedge \neg L^1_{1,2} \wedge \neg L^1_{2,2} \wedge ....)$
  - We need for every point (x,y) and for every time step t (we do not even have an a priori limit on the time)
  - We need such formulas in the knowledge base for every action
    Grab / Shoot / Climb / TurnLeft / ....

- There is still something unspecified:
  - We also need to say what remains unchanged
    - Example: When Forward action is performed at time t, Wumpus Alive/Dead is unchanged
  - This is what is called the frame problem
    - In a frame (either in inertial frame of physics or movie frame) Every action changes a few things and most things remain unchanged.

Towards lesser number of axioms

- We also need to say what fluents remains unchanged depending on the action:
  - $(\text{Forward}^t) \Rightarrow (\text{haveArrow}^t \Leftrightarrow \text{haveArrow}^{t+1})$
  - $(\text{Forward}^t) \Rightarrow (\text{wumpusAlive}^t \Leftrightarrow \text{wumpusAlive}^{t+1})$

- If there are m Actions and n Fluents, how many such axioms do we need at every time step?
  - O(mn)
  - This explosion is called : Representational frame problem

- Can we have smaller number of formulas that encodes the same information ?

Representing Frame axioms

- Instead having axioms for each action, have one axiom for each fluent stating when it changes:
  - $haveArrow^{t+1} \Leftrightarrow (haveArrow^{t} \wedge \neg shoot^{t})$
  - $L^{t+1}_{1,1} \Leftrightarrow (L^{t+1}_{1,1} \wedge (\neg Forward^{t} \vee Bump^{t})) \vee$
    $(L^{t}_{1,2} \wedge FacingSouth^{t} \wedge Forward^{t}) \vee$
    $(L^{t}_{2,1} \wedge FacingWest^{t} \wedge Forward^{t})$

- This way of presenting the axioms have one formula per fluent at every time step.

- We have O(n) formulas at every time step where n is the number of fluents.

Qualification Problem

- Suppose we have encoded all information about the wumpus world efficiently in our knowledge
- Use state of the art SAT solver to make inferences

- Can we be confident that our job is done?
  - Maybe not. When the agent moves forward, there might be fire!
  - We cannot anticipate everything that a drone might encounter when it is dealing with a building on fire

- This is called the qualification problem
  - No solution using Logic
  - One possible solution : Use probability
    (Action succeeds with some probability)

- Handling Fluents
- Representing the Fluent axioms in the KB
- Hybrid Agent for Wumpus World
- Using Propositional Inferencing to make Plan

Equipping agents with the power of inferencing
Hybrid Agent

- Agent starts with a Knowledge Base containing Atemporal axioms
  - Axioms that do not depend on time steps

- At every step:
  - New percept sentence is added
  - All the axioms that depend on t are added
  - The agent uses logical inference, by ASKing questions of the knowledge base, to work out which squares are safe and which have yet to be visited.
  - Take the most appropriate action

- Which action to be taken?
  - Should be based on priority

Hybrid Agent for Wumpus World

- If there is glitter in the current location, perform GRAB and plan to move to the initial square and perform CLIMB

- Otherwise, choose one of the safe location that is not yet visited and plan to move there only using safe locations.
    - This can be done using A* or other search techniques

- If there are no safe squares to explore:
    - If the agent still has an arrow try to make a safe square by shooting at one of the possible wumpus locations.
    - Otherwise, look for a location that is not provably unsafe—that is, a square for which ASK(KB, ¬OK) is False.
    - If there is no such square, then the mission is impossible and the agent retreats to [1, 1] and climbs out of the cave.