Study Notes for AI (CS 440/520)
Lectures 7 & 8: Logic-based Inference and Satisfiability

Corresponding Book Chapters:
Note: These notes provide only a short summary and some highlights of the material covered in the corresponding lecture based on notes collected from students. Make sure you check the corresponding chapters. Please report any mistakes in or any other issues with the notes to the instructor.

1 A Running Example: Wumpus World

This section introduces an example that we will use in this and future lectures to demonstrate some important ideas related to constraint satisfaction problems and how they can be formulated formally through propositional logic. It is called the Wumpus World.

The Wumpus World is a cave consisting of rooms connected by passageways. Lurking somewhere in the cave is the wumpus, a beast that eats anyone who enters its room. The wumpus can be shot by an agent, but the agent has only one arrow. Some rooms contain bottomless pits that will trap anyone who wanders into these rooms (except for the wumpus, which is too big to fall in). The only mitigating feature of living in this environment is the possibility of finding a heap of gold.

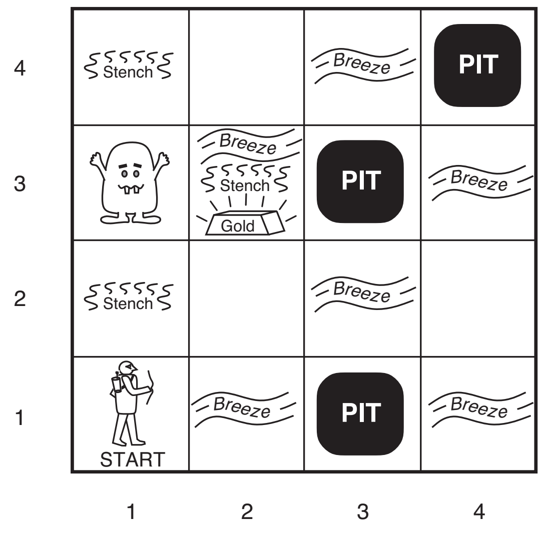


Figure 1: A typical Wumpus World. The agent is in the bottom left corner.

A simple wumpus world is shown in Figure 1. The precise definition of the task environment is given by the PEAS description:

- **Performance measure:** +1000 for picking up the gold, -1000 for falling into a pit or being eaten by the wumpus, -1 for each action taken and -10 for using up the arrow.
- **Environment:** A 4x4 grid of rooms. The agent always starts in the square labeled [1,1], facing to the right. The locations of the gold and the wumpus are chosen randomly, with a uniform distribution, from the squares other than the start square. In addition, each square other than the start can be a pit, with probability 0.2.
- **Actuators:** The agent can move forward, turn left by 90^o, or turn right by 90^o. The agent dies a miserable death if it enters a square containing a pit or a live wumpus. (It is safe, albeit smelly, to enter a square with a dead wumpus.) Moving forward has no effect if there is a wall in front of the agent. The action *Grab* can be used to pick up an object that is in the same square as the agent. The action *Shoot* can be used to fire an arrow in a straight line in the direction the agent is facing. The arrow continues until it either hits (and hence kills) the wumpus or hits a wall. The agent only has one arrow, so only the first *Shoot* action has any effect.
- **Sensors:** The agent has five sensors, each of which gives a single bit of information:
  - In the square containing the wumpus and in the directly (not diagonally) adjacent squares the agent will perceive a stench.
  - In the squares directly adjacent to a pit, the agent will perceive a breeze.
  - In the square where the gold is, the agent will perceive a glitter.
  - When an agent walks into a wall, it will perceive a bump.
  - When the wumpus is killed, it emits a woeful scream that can be perceived anywhere in the cave.

  The percepts will be given to the agent in the form of a list of five symbols; for example, if there is a stench and a breeze, but no glitter, bump, or scream, the agent will receive the percept [*Stench, Breeze, None, None, None*].

Let's consider an agent exploring the environment shown in Figure 1. The agent's initial knowledge contains the rules of the environment, as listed previously; in particular, it knows that it is in [1,1] and that [1,1] is a safe square. Figure 2 shows how the knowledge evolves as new percepts arrive and actions are taken.

The first percept is [*None, None, None, None, None*], from which the agent can conclude that its neighboring squares are safe. In the figure, letter *B* stands for breeze and OK marks a safe square (no pit, nor wumpus).

From the fact that there was no stench or breeze in [1,1], the agent can infer that [1,2] and [2,1] are free of dangers. They are marked with an OK to indicate this. A cautious agent will move only into a square that it knows is OK. Let us suppose the agent decides to move forward to [2,1]. The agent detects a breeze in [2,1], so there must be a pit in a neighboring square. The pit cannot be in [1,1], by the rules of the game, so there must be a pit in [2,2] or [3,1] or both. At this point, there is only one known square that is *OK* and has not been visited yet. So the prudent agent will turn around, go back to [1,1] and then proceed to [1,2].

The new percept in [1,2] is [*Stench, None, None, None, None*]. The stench in [1,2] means that there must be a wumpus nearby. But the wumpus cannot be in [1,1], by the rules of the game, and it cannot be in [2,2] (or the agent would have detected a stench when it was in [2,1]). Therefore, the agent can infer that the wumpus is in [1,3]. The notation *W*! in Figure 3 indicates this. Moreover, the lack of a *Breeze* in [1,2] implies that there is no pit in [2,2]. Yet we already inferred that there must be a pit in either [2,2] or [3,1], so this means it must be in [3,1].

The agent has now proved to itself that there is neither a pit nor a wumpus in [2,2], so it is *OK* to move there. In [2,3], the agent will detect a glitter, so it should grab the goal and thereby end the game.

In each case, where the agent draws a conclusion from the available information, that conclusion is guaranteed to be correct if the available information is correct. This is a fundamental property of logical reasoning. In the rest of this lecture, we will describe how to build agents that can represent the necessary information of a problem (build a knowledge base) and draw conclusions (answer queries). `CSP`s are problems that directly lend themselves to such representation.

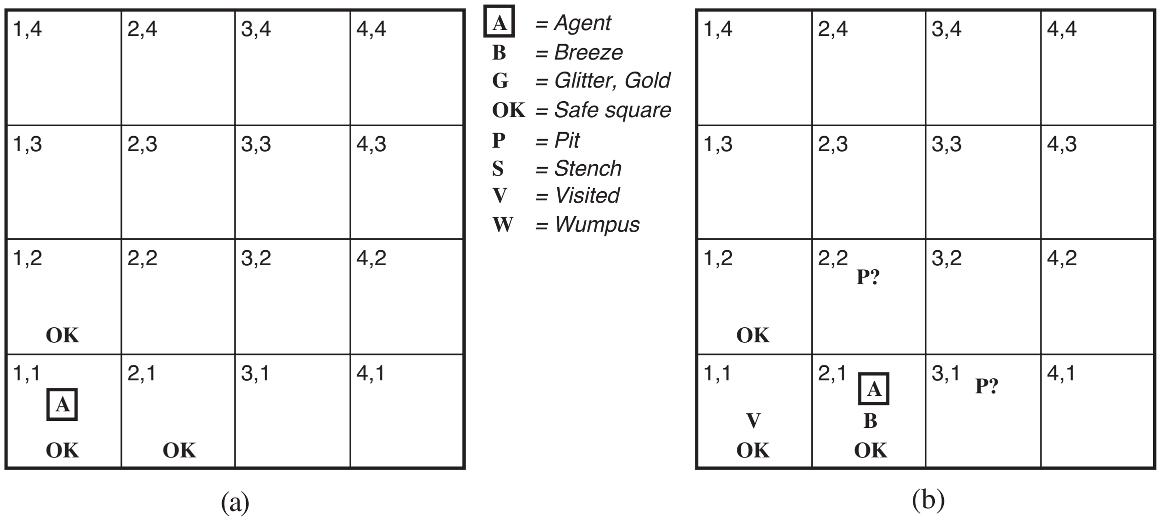


Figure 2: The first step taken by the agent in the wumpus world. (a) The initial situation, after percept [*None, None, None, None, None*]. (b) After one move, with percept [*None, Breeze, None, None, None*].

2 Propositional Logic

In propositional logic, also known as boolean logic, each variable has two possible assignments, either true or false. More complex sentences can be constructed with the following operands:

$\neg$ (not): negation
$\wedge$ (and): conjunction
$\vee$ (or): disjunction
$\Rightarrow$ (implies): implication
$\Leftrightarrow$ (if and only if): biconditional

Parentheses can be used to avoid ambiguity, e.g., $((A \wedge B) \Rightarrow C)$. Without parentheses the following order defines the precedence of operands: $\neg, \wedge, \vee, \Rightarrow, \Leftrightarrow$. Hence the proposition $\neg P \vee Q \vee R \Rightarrow S$ is equivalent to the sentence: $((\neg P) \vee (Q \wedge R)) \Rightarrow S$.

We can use the tools of propositional logic to represent knowledge and inference procedures in the Wumpus World. For example, let's focus on detecting pits in the 4x4 grid and use the following variables:

$P_{i,j}$: True if a pit in cell [i,j].
$B_{i,j}$: True if a breeze in cell [i,j].

Then we can define the following rules about the Wumpus World given the knowledge base about the problem:

$$R1 : \neg P_{1,1} \qquad \text{(No pit in [1,1])}$$

Moreover, since a square is breezy if and only there is a pit in a neighboring square (for cells [1,1] and [2,1]):

$$R2 : B_{1,1} \Leftrightarrow (P_{1,2} \wedge P_{2,1})$$

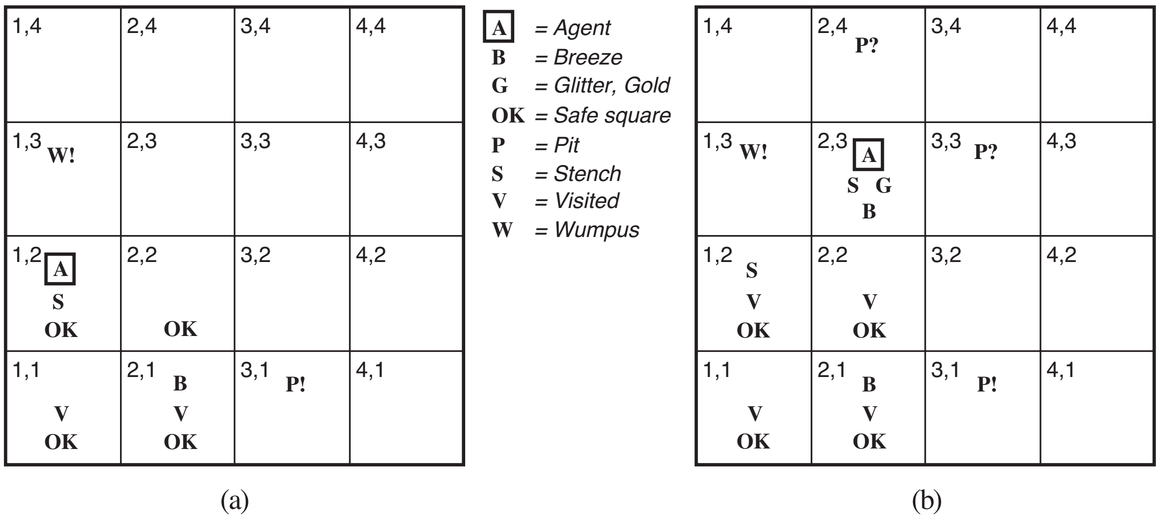


Figure 3: Two later stages in the progress of the agent. (a) After the third move, with percept [*Stench, None, None, None, None*] (b) After the fifth move, with percept [*Stench, Breeze, Glitter, None, None*].

$$R3 : B_{2,1} \Leftrightarrow (P_{1,1} \land P_{2,2} \land P_{3,1})$$

Furthermore, since there is no breeze at [1,1] but there is one at [2,1]:

$$R4 : \neg B_{1,1}$$

$$R5 : B_{2,1}$$

Given this knowledge base, we then want to answer queries like the following: is $P_{2,2}$ or $P_{1,2}$ true?

One way to answer such queries is to construct the entire truth table for the variables involved in the queries. Figure 4 illustrates how the truth table looks like for the knowledge based defined by rules R_1 to R_5. The knowledge base is true if all the rules are true. If a certain variable, such as $P_{1,2}$, has the same value for all the assignments that correspond to the knowledge base ($P_{1,2}$ is false), then we have found the value of this variable. On the other hand, we cannot answer a query about a variable that acquires multiple values for the assignments that correspond to the knowledge base (like $P_{2,2}$).

Although we can use the truth table approach to logically answer queries given a knowledge base it is by far not an efficient approach. If we have n birary variables in the problem definition, then the number of possible truth assignments are 2^n and consequently exponential to the number of variables.

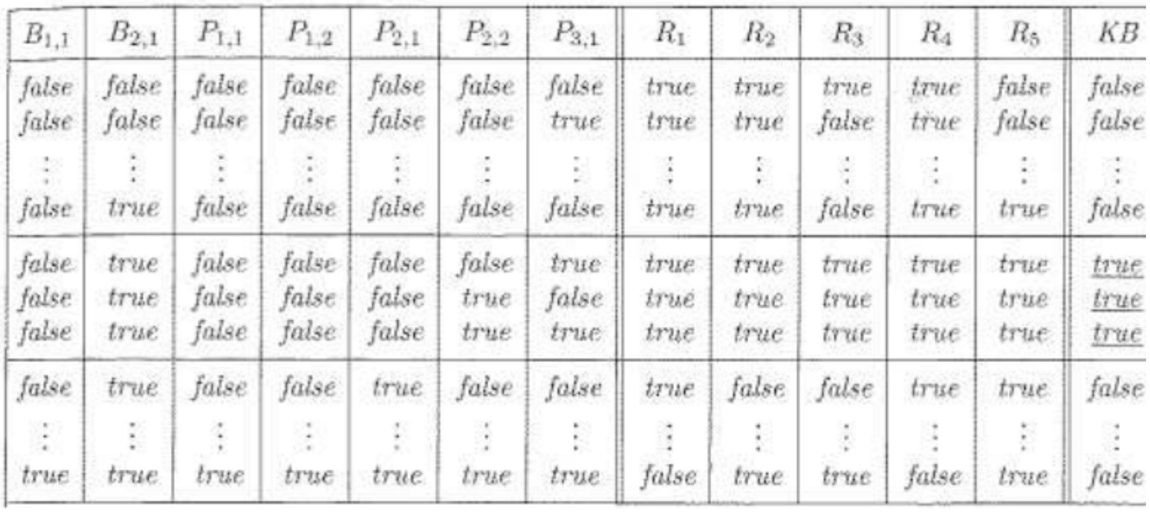
An alternative methodology is to make use of logical inference rules to try to extract new knowledge from the rules of our knowledge base. A fundamental inference rule in logic is known as Modus Ponens:

$$\frac{\alpha \Rightarrow \beta, \alpha}{\beta}$$

It specifies that if the following two propositions $\alpha \Rightarrow \beta$ and α are true then we can infer that also β is true. Another important inference rule is the and-elimination rule:

$$\frac{\alpha \land \beta}{\alpha}$$

The following logical equivalences also hold:



$B_{1,1}$	$B_{2,1}$	$P_{1,1}$	$P_{1,2}$	$P_{2,1}$	$P_{2,2}$	$P_{3,1}$	R_1	R_2	R_3	R_4	R_5	KB
false	*false*	*false*	*false*	*false*	*false*	*false*	*true*	*true*	*true*	*true*	*false*	*false*
false	*false*	*false*	*false*	*false*	*false*	*true*	*true*	*true*	*false*	*true*	*false*	*false*
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
false	*true*	*false*	*false*	*false*	*false*	*false*	*true*	*true*	*false*	*true*	*true*	*false*
false	*true*	*false*	*false*	*false*	*false*	*true*	*true*	*true*	*true*	*true*	*true*	*true*
false	*true*	*false*	*false*	*false*	*true*	*false*	*true*	*true*	*true*	*true*	*true*	*true*
false	*true*	*false*	*false*	*false*	*true*	*true*	*true*	*true*	*true*	*true*	*true*	*true*
false	*true*	*false*	*false*	*true*	*false*	*false*	*true*	*false*	*false*	*true*	*true*	*false*
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
true	*true*	*true*	*true*	*true*	*true*	*true*	*false*	*true*	*true*	*false*	*true*	*false*

Figure 4: A truth table constructed for the knowledge base given here. The knowledge base is true if R_1 through R_5 are true, which occurs in just 3 of the 128 rows. In all 3 rows, $P_{1,2}$ is false, so there is not pit in [1,2]. On the other hand, there might (or might not) be a pit in [2,2].

$$
\begin{array}{rl}
(\alpha \land \beta) \equiv (\beta \land \alpha) & \text{commutativity of } \land \\
(\alpha \lor \beta) \equiv (\beta \lor \alpha) & \text{commutativity of } \lor \\
((\alpha \land \beta) \land \gamma) \equiv (\alpha \land (\beta \land \gamma)) & \text{associativity of } \land \\
((\alpha \lor \beta) \lor \gamma) \equiv (\alpha \lor (\beta \lor \gamma)) & \text{associativity of } \lor \\
\neg(\neg\alpha) = \alpha & \text{double-negation elimination} \\
(\alpha \Rightarrow \beta) \equiv (\neg\beta \Rightarrow \neg\alpha) & \text{contraposition} \\
(\alpha \Rightarrow \beta) \equiv (\neg\alpha \lor \beta) & \text{implication elimination} \\
(\alpha \Leftrightarrow \beta) \equiv ((\alpha \Rightarrow \beta) \land (\beta \Rightarrow \alpha)) & \text{biconditional elimination} \\
\neg(\alpha \land \beta) \equiv (\neg\alpha \lor \neg\beta) & \text{De Morgan} \\
\neg(\alpha \lor \beta) \equiv (\neg\alpha \land \neg\beta) & \text{De Morgan} \\
(\alpha \land (\beta \lor \gamma)) \equiv ((\alpha \land \beta) \lor (\alpha \land \gamma)) & \text{distributivity of } \land \text{ over } \lor \\
(\alpha \lor (\beta \land \gamma)) \equiv ((\alpha \lor \beta) \land (\alpha \lor \gamma)) & \text{distributivity of } \lor \text{ over } \land
\end{array}
$$

We can now use these logical equivalences and inference rules to answer queries given a knowledge base. For the Wumpus World example, we can define the following new rules:

Biconditional elimination on R2:

$$R6 : (B_{1,1} \Rightarrow (P_{1,2} \lor P_{2,1})) \land ((P_{1,2} \lor P_{2,1}) \Rightarrow B_{1,1})$$

And elimination on R6:

$$R7 : ((P_{1,2} \lor P_{2,1}) \Rightarrow B_{1,1})$$

Contraposition on R7:

$$R8 : (\neg B_{1,1} \Rightarrow \neg(P_{1,2} \vee P_{2,1}))$$

Modus Ponens with R_8 and R_4:

$$R9 : \neg(P_{1,2} \vee P_{2,1})$$

De Morgan's rule on R9:

$$R10 : \neg P_{1,2} \wedge \neg P_{2,1}$$

The last rule proves that neither [1,2] nor [2,1] contains a pit. We have not shown any rule that informs us about the cell [2,2]. This is exactly the same result as the one we reached with the truth table approach. The hope is that through logical inference this process can be achieved more efficiently. The following lecture will focus on how to use a new inference rule that does allow for efficient automated logical inference.

3 Resolution

Inference rules are relationship between premises and conclusions, where the conclusion is derivable from the premises. We have a knowledge base expressed through logical expressions and we want to infer new knowledge. An example of inference rule is Modus Ponens, $\frac{\alpha \implies \beta, \alpha}{\beta}$, and Modus Tollen,$\frac{\alpha \implies \beta, \neg\beta}{\neg\alpha}$. Here we will describe a new inference rules that will allow us to check automatically whether a knowledge base implies a certain logical expression.

3.1 New Inference Rule

$$\frac{(l_1 \vee l_2 \vee \ldots \vee l_{k-1} \vee l_k), \neg l_i}{(l_1 \vee \ldots \vee l_{i-1} \vee l_{i+2} \vee \ldots \vee l_k)} \tag{1}$$

Equation 1 describes the Resolution rule. The Resolution rule specifies that if a variable appears in two disjunctive expressions with opposite signs then it can be "resolved", meaning that a new disjunctive expression is implied where all the variables from the original disjunctions appear except from the one that appears with opposite signs. More examples are provided in Equations 2 and 3:

$$\frac{(l_1 \vee l_2), (\neg l_2 \vee l_3)}{(l_1 \vee l_3)} \tag{2}$$

$$\begin{gathered}(\alpha \vee \beta \vee \gamma) \equiv True\\ \neg\beta \equiv True\\ \\ (\alpha \vee \gamma) \equiv True\end{gathered} \tag{3}$$

Equations 2 and 3 both show that the variable that appears with both the positive and negative sign will be removed.

3.2 Example

We can show that resolution is helpful for inference on the Wumpus world example. Assume that we have 1 additional sensing input on top of the rules $R_1 - R_{10}$ described in the previous lecture:
$R_{11} : \neg B_{1,2}$ (No breeze out (1,2))
$R_{12} : B_{1,2} \Leftrightarrow (P_{1,1} \vee P_{2,2} \vee P_{1,3})$ (by definition)
following a similar process as in the last lecture we can infer that
$R_{13} : \neg P_{2,2}$
$R_{14} : \neg P_{1,3}$

By applying bi-conditional elimination to R_3 and Modus Ponens with R_5
$R_{15} : P_{1,1} \vee P_{2,2} \vee P_{3,1}$
Then we can apply the resolution rule in R_{13} and R_{15} to get:
$R_{16} : P_{1,1} \vee P_{3,1}$
And apply again the resolution rule on R_{16} and R_{14} to get:
$R_{17} : P_{3,1}$

3.3 CNF

The importance of the resolution rule is that it defines a complete proving system. This means that any complete search algorithm, applying only the resolution rule, can derive any conclusion entailed by any knowledge base in propositional logic. This is due to the fact that we can turn every sentence in propositional logic in a form in which the resolution rule can be applied. This section describes this form.

We first have to define the following terms:

- Literals correspond to a variable or its negation, e.g., α or $\neg\alpha$.
- Clauses are disjunctions on literals, e.g., $(l_1 \vee l_2 \vee \ldots \vee l_n)$.

Every sentence in propositional logic can be expressed in CNF (Conjunctive Normal Form), which corresponds to a conjuction of clauses:

$$(l_{1,1} \vee \ldots \vee l_{1,k}) \wedge (l_{2,2} \vee \ldots \vee l_{2,k}) \wedge \ldots$$

An example of how a proposition can be turned into CNF is shown below.

$$\alpha \Leftrightarrow \beta$$
$$(\alpha \implies \beta) \wedge (\beta \implies \alpha) \text{ (applied bi-conditional elimination)}$$
$$(\neg\alpha \vee \beta) \wedge (\neg\beta \vee \alpha) \text{ (applied implication elimination)}$$

The above example shows how the biconditional ($\Leftrightarrow$) and conditional ($\implies$) operands can be replaced by conjunctions of clauses, thus turned into CNF.

3.4 Another Example

Anexample on how we can turn expressions in CNF from the Wumpus World:

$B_{1,1} \Leftrightarrow P_{1,2} \vee P_{2,1}$
(apply biconditional elimination)
$(B_{1,1} \implies (P_{12} \vee P_{21})) \wedge ((P_{1,2} \vee P_{2,1}) \implies B_{1,1})$
(implication elimination)
$(\neg B_{1,1} \vee P_{1,2} \vee P_{2,1}) \wedge (\neg(P_{1,2} \vee P_{2,1}) \vee B_{1,1})$
(De Morgan)
$(\neg B_{1,1} \vee P_{1,2} \vee P_{2,1}) \wedge ((\neg P_{1,2} \wedge \neg P_{2,1}) \vee B_{1,1})$
(distributivity)
$(\neg B_{1,1} \vee P_{1,2} \vee P_{2,1}) \wedge (\neg P_{1,2} \vee B_{1,1}) \wedge (\neg P_{2,1} \vee B_{1,1})$
which is in CNF.

4 Solving Problems with Logical Inference

In this section we are going to discuss how we can prove a rule using logical inference. We are going to use the approach: "proof by Contradiction".

4.1 Satisfiability Algorithm

Input: KB (a set of rules / constraints) (knowledge base).
Lets say you are asked to prove α is True. This means that we have to show KB $\models \alpha$ (KB entails α).
We can show that $(KB \land \neg\alpha)$ is unsatisfiable, which means every assignment of variables does not satisfy the sentence. By contradiction, this will limply KB $\models \alpha$.
Steps of the algorithm:

1. Represent KB in Propositional Logic
2. Take sentence $(KB \land \neg\alpha)$ and turn it into CNF.
3. Apply repetitively the resolution rule until 1 of the following 2 happens:
   a) There is no new clause you can create with resolution. (KB does not entail α)
   b) 2 clauses resolve to the empty clause. $(KB \land \neg\alpha)$ is unsatisfiable, so $(KB \models \alpha)$

The set of clauses derivable by repeated applications of the resolution rule on the set S is called the **Resolution Closure (RC)**. The inference process is complete because the above algorithm will terminate, since RC(s) is a finite set if we have a finite number of clauses in sentence S. Then the above algorithm, called the satisfiability algorithm, has the following property:

Ground Resolution Theorem: "If a set of clauses is unsatisfiable then the resolution closure of those clauses contains the empty clause."

4.2 Example Problem

The example is from the Wumpus world where $KB = R_2 \land R_5$ and $\alpha = \neg P_{1,2}$.

1. $KB = R_2 \land R_5 = (B_{1,1} \Leftrightarrow (P_{1,2} \lor P_{2,1})) \land \neg B_{1,1}$

2. Lets prove that $KB \models \neg P_{1,2}$ by considering the satisfiability of $(KB \land P_{1,2})$
   By our previous example at Section 2.4, we get:
   $(\neg B_{1,1} \lor P_{1,2} \lor P_{2,1}) \land (\neg P_{1,2} \lor B_{1,1}) \land (P_{1,1} \lor B_{1,1}) \land \neg B_{1,1} \land P_{1,2}$

3. We can apply the resolution rule and get a new clause: $\neg P_{1,2}$, from clauses $(\neg P_{1,2} \lor B_{1,1})$ and $B_{1,1}$.
   However since we have $P_{1,2}$ and $\neg P_{1,2}$, they will resolve to the empty clause. $(\bot)$
   Therefore, since $(KB \land P_{1,2})$ is not satisfiable, $KB \models \neg P_{1,2}$.

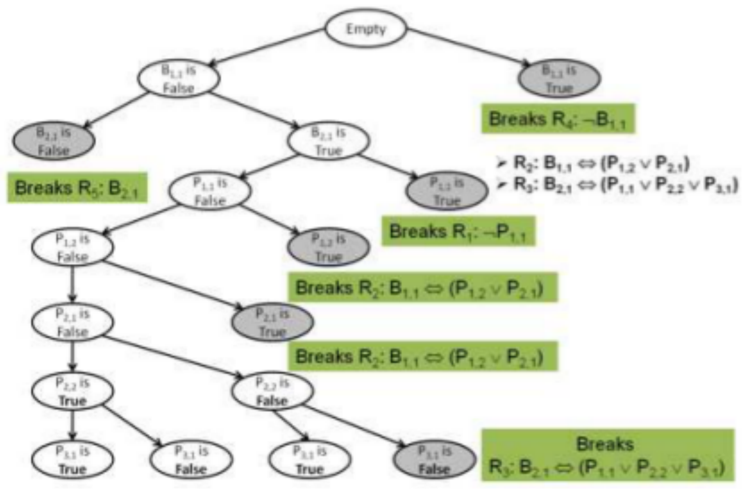
5 Backtracking Search

In order to resolve such logic-based inference problems, we can also apply backtracking on the variables in order to answer our queries. So for the above example we have:

Since the satisfiability algorithm can be used to solve `CSP`s, then is reasonable to expect that backtracking, a search-based technique for solving `CSP`s, can be applied to logic-based problems. A specific version of backtracking search, called the Davis-Putnam Algorithm, can be used to check the satisfiability of the logical expression.

5.1 Davis-Putnam Algorithm

The Davis-Putnam (DP) Algorithm was first proposed in a seminal paper by Martin Davis and Hilary Putnam in 1960. It models constraint satisfaction problems using propositional logic and solves them through a backtracking depth-first search. The DP algorithm has three generic heuristics that, when used properly, make this search complete.



Empty
$B_{1,1}$ is False
$B_{1,1}$ is True
Breaks R_4: $\neg B_{1,1}$
$B_{2,1}$ is False
$B_{2,1}$ is True
Breaks R_5: $B_{2,1}$
R_2: $B_{1,1} \Leftrightarrow (P_{1,2} \vee P_{2,1})$
R_3: $B_{2,1} \Leftrightarrow (P_{1,1} \vee P_{2,2} \vee P_{3,1})$
$P_{1,1}$ is False
$P_{1,1}$ is True
Breaks R_1: $\neg P_{1,1}$
$P_{1,2}$ is False
$P_{1,2}$ is True
Breaks R_2: $B_{1,1} \Leftrightarrow (P_{1,2} \vee P_{2,1})$
$P_{2,1}$ is False
$P_{2,1}$ is True
Breaks R_2: $B_{1,1} \Leftrightarrow (P_{1,2} \vee P_{2,1})$
$P_{2,2}$ is True
$P_{2,2}$ is False
$P_{3,1}$ is True
$P_{3,1}$ is False
$P_{3,1}$ is True
$P_{3,1}$ is False
Breaks R_3: $B_{2,1} \Leftrightarrow (P_{1,1} \vee P_{2,2} \vee P_{3,1})$

5.1.1 Early Termination Heuristic

The early termination heuristic detects whether a conjunction of two or more clauses must be true or false, even with only a partial model. A clause is true if *any* literal is true, even if the other literals do not yet have assignments. For example, given the sentence $(A \vee B) \wedge (A \vee C)$, it evaluates true if A is true, regardless of the values of B and C. Similarly, a sentence is false if any one of the clauses in it evaluates false. These kinds of evaluations can occur long before the entire logic tree is complete, and avoids the examination of entire subtrees during search and vise versa.

5.1.2 Pure Symbol Heuristic

The Davis-Putnam algorithm also defined a *pure symbol* as a symbol that always appears with the same "sign" in all clauses. For example, in the three clauses $(A \vee \neg B), (\neg B \vee \neg C)$, and $(C \vee A)$, the symbol A is pure because it only appears as a positive literal. B is pure because it only appears as a negative literal. C is impure because it appears with both positive and negative literals. It is easy to see that if there is a pure positive literal, that assigning a true value to that variable will make that particular clause true in all situations.

5.1.3 Unit Clause Heuristic

A unit clause, as defined by Davis and Putnam, is a clause with a single literal, or a clause in which all literals but one are assigned false. For example, given $B = false$, and a clause $(B \vee \neg C)$, the clause becomes a unit clause; it is equivalent to $(false \vee \neg C)$. In order for this clause to be true, C has to be false. The heuristic specifies that all literals in unit clauses must be assigned first. This can cause a cascading effect of assigning variables directly.

6 Local Search for Logic-based Inference

We can also apply local search algorithms on the satisfiability problem such as hill climbing. The evaluation function for the hill climbing algorithm can be the number of unsatisfied clauses. At each iteration, the technique flips the truth value of one symbol and checks whether the evaluation function improves or not.

6.1 WALKSAT Algorithm

The WALKSAT algorithm is a hill climbing technique specifically designed for the satisfiability problem. It selects an unsatisfiable clause and then a symbol to flip. There are two ways to choose the symbol:

- greedily (minimize the number of unsatisfied clauses)
- randomly

Is WALKSAT complete?

- If it returns a model, it has found a solution
- If it hasn't, we do not know.

So it is probabilistically complete, eventually it will find one solution. Consequently, WALKSAT is more useful when a solution exists. But in practice, it is much faster than Davis-Putnam.

7 When Propositional Logic Fails / Some Solution

Given the satisfiability algorithm, for any sentence that can be expressed in propositional logic we can use resolution to infer new knowledge and conclusions. Unfortunately, not everything can be succinctly represented in propositional logic.

7.1 Examples of Limitations

For example, in the Wumpus World the following cases are difficult to be expressed in propositional logic:

- When we need a rule for every cell, e.g., the fact that a breeze implies that a neighoring cell is a pit: $B_{i,j} \implies \forall_{k,l} P_{k,l}$ where (k, l) neighbor of (i, j)
- When we need to represent time. e.g. assuming that:
  $L_{i,j}$: agent's location, then
  $L_{1,1} \wedge$ Facing Right $\wedge$ Forward $\implies L_{1,2}$
  The abobe sentence imlies that both $L_{1,1}$ and $L_{1,2}$ are true simultaneously, which is impossible. What we actually want to express is the fact that when $L^t_{1,1}$ is true at time t then $L^{t+1}_{1,2}$ will be true at time $t + 1$.

Because of the above limitations, more powerful types of logic have been designed.

7.2 First-Order Logic

Additional Operands:

- $\forall$ (always), $\exists$ (exists)
- Functions
- Equality

7.2.1 Description

- More powerful than Propositional Logic
- New methods for inference

7.3 Second-Order Logic

Additional Operands:

- Always True
- Until, After then

7.3.1 Description

- Even stronger than First-Order Logic