UNIT - V: Reasoning in Uncertain Situations

Syllabus:

Introduction to non-monotonic reasoning, truth maintenance systems, logics for non-monotonic reasoning, classical planning problem: Goal stack, hierarchical planning.

Outcomes:

Student will be able to:

- Distinguish monotonic and nonmonotonic reasoning.
- Understand different non monotonic logics.
- Represent knowledge in Truth maintenance systems.
- Solve problems using goal stack planning.
- Perform hierarchical planing.

5.1 Introduction to non-monotonic reasoning

- **Nonmonotonic reasoning**, in which the axioms/or the rules of inference are extended to make it possible to reason with incomplete information. These systems preserve, however, the property that, at any given moment, a statement is either believed to be true, believed to be false, or not believed to be either.
- Consider an example: **ABC Murder story**.

  Let Abbott, Babbitt, and Cabot be suspects in a murder case. Abbott has an alibi, in the register of a respectable hotel in Albany. Babbitt also has an alibi, for his brother-in-law testified that Babbitt was visiting him in Brooklyn at the time. Cabot pleads alibi too, claiming to have been watching a ski meet in the Catskills, but we have only his word for that. So we believe:

  (1) That Abbott did not commit the crime,

  (2) That Babbitt did not.

  (3) That Abbott or Babbitt or Cabot did.

  But presently Cabot documents his alibi—he had the good luck to have been caught by television in the sidelines at the ski meet. A new belief is:

  (4) That Cabot did not.

- **Which has the weakest evidence?** The basis for (1) in the hotel register is good, since it is a fine old hotel. The basis for (2) is weaker, since Babbitt's brother-in-law might be lying. The basis for (3) is

perhaps twofold; that there is no sign of burglary and that only Abbott, Babbitt and Cabot seem to have stood to gain from the murder apart from burglary. This exclusion of burglary seems conclusive, but the other consideration does not: there could be some fourth beneficiary. For (4), finally, the basis is conclusive: the evidence from television. Thus (2) and (3) are the weak points. To resolve the inconsistency of (1) through (4) we should reject (2) or (3), thus either incriminating Babbitt or widening our net for some new suspect.

- **Conventional reasoning systems** such as first-order predicate logic, are designed to work with information that has three important properties:
  - It is complete with respect to the domain of interest. In other words, all the facts that are necessary to solve a problem are present in the system or can be derived from those that are by the conventional rules of first-order logic.
  - It is consistent.
  - The only way it can change is that new facts can be added as they become available. If these new facts are consistent with all the other facts that have already been asserted, then nothing will ever be retracted from the set of facts that are known to be true. This property is called **monotonicity.**
- If any of these properties is not satisfied, conventional logic-based reasoning systems become inadequate. **Nonmonotonic reasoning** systems, on the other hand, are designed to be able to solve problems in which all of these properties may be missing.
- In order to do this, we must address several key issues, including the following:
  - **How can the knowledge base be extended to allow inferences to be made on the basis of lack of knowledge as well as on the presence of it?** For example, we would like to be able to say things like. "If you have no reason to suspect that a particular person committed a crime, then assume he didn't." or "if you have no reason to believe that someone is not getting along with her relatives, then assume that the relatives will try to protect her." Specifically. we need to make clear the distinction between:

    It is known that ¬P

    It is not known whether P.

    We call any inference that depends on the lack of some piece of knowledge a **nonmonotonic inference**.

    Allowing such reasoning has a significant impact on a knowledge base. Nonmonotonic reasoning systems derive their

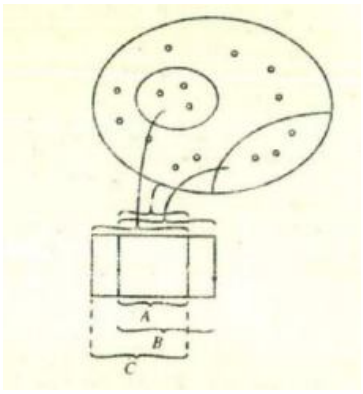
name from the fact that because of inferences that depend on lack of knowledge, knowledge bases may not grow monotonically as new assertions are made. Adding a new assertion may invalidate an inference that depended on the absence of that assertion. First-order predicate logic system, on the other hand, is monotonic in this respect. As new axioms are asserted, new wff's may become provable, but no old proofs ever become invalid. In other words, if some set of axioms T entails the truth of some statement w, then T combined with another set of axioms N also entails w. Because nonmonotonic reasoning does not share this property, it is also called **defeasible**; a nonmonotonic inference may be defeated (rendered invalid) by the addition of new information that violates assumptions that were made during the original reasoning process.

- **How can the knowledge base be updated properlv when a new fact is added to the system (or when an old one is removed)?** In particular, in nonmonotonic systems, since the addition of a fact can cause previously discovered proofs to be become invalid, how can those proofs, and all the conclusions that depend on them be found? The usual solution to this problem is to keep track of proofs, which are often called **Justifications**. This makes it possible to find all the justifications that depended on the absence of the new fact, and those proofs can be marked as invalid.
- **How can knowledge be used to help resolve conflicts, when there are several inconsistent nonmonotonic inferences that could be drawn?** It turns out that when inferences can be based on the lack of knowledge as well as on its presence. contradictions are much more likely to occur than they were in conventional logical systems in which the only possible contradictions were those that depended on facts that were explicitly asserted to be true. In particular, in nonmonotonic systems, there are often portions of the knowledge base that are locally consistent but mutually (globally) inconsistent.

5.2 Logics for non-monotonic reasoning

- Single formalism with all the desired properties that supports nonmonotonic reasoning has not yet emerged.). In particular, we would like to find formalism that does all of the following things:
  - Defines the set of possible worlds that could exist given the facts that we do have. We require a mechanism for defining the set of models of any set of wff's we are given.

- Provides a way to say that we prefer to believe in some models rather than others.
- Provides the basis for a practical implementation of this kind of reasoning.
- Corresponds to our intuitions about how this kind of reasoning works.



- The figure visualizes how nonmonotonic reasoning works. The box labeled A corresponds to an original set of wff's. The large circle contains all the models of A. When we add some nonmonotonic reasoning capabilities to A, we get a new set of wff's, which we've labelled B, which contains more information than A does. As a result, fewer models satisfy B than A. The set of models corresponding to B is shown at the lower right of the large circle. Now suppose we add some new wff's (representing new information) to A. We represent A with these additions as the box C. A difficulty may arise, with the set of models corresponding to C, since it is disjoint with the models for B. In order to find a new set of models that satisfy C, we need to accept models that had previously been rejected. To do that, we need to eliminate the wff's that were responsible for those models being thrown away. This is the essence of nonmonotonic reasoning.
- **Default reasoning:** We use nonmonotonic reasoning to perform what is commonly called **default reasoning**. We draw conclusions based on what is most likely to be true.
  - The two approaches for doing this:
    - Nonmonotonic Logic
    - Default Logic
  - Two common kinds of nonmonotonic reasoning that can be defined in those logics:
    - Abduction
    - Inheritance

5.2.1 Nonmonotonic Logic

- The basis for default reasoning is Nonmonotonic logic(NML), , in which the language of first-order predicate logic is augmented with a modal operator **M**, which can be read as "**is consistent**".
- For example, the formula:

  ∀x,y Related(x,y) Λ M GetAlong(x,y) → WillDefend(x,y)

  should be read as**, "For all x and y, if x and y are related and if the fact that x gets along with y is consistent with everything else that is believed, then conclude that x will defend y."**
- For example, consider the following set of assertions:

  ∀x: Republican(x) Λ M¬Pacifist(x) →¬Pacifist(x)

∀x: Quaker(x)Λ MPacifist(x) →Pacifist(x)
Republican(Dick)
Quaker(Dick)

- The definition of NML that we have given supports two distinct ways of augmenting this knowledgebase, In one, we first apply the first assertion, which allows us to conclude ¬Pacifist(Dick). Having done that, the second assertion cannot apply, since it is not consistent to assume Pacifist(Dick).
- The other thing we could do, however, is apply the second assertion first. This results in the conclusion Pacifist(Dick), which prevents the first one from applying. So, in this example, no conclusion about Dick's pacifism can be derived.
- For example, given:
  A Λ MB→ B
  ¬A Λ MB → B
  we can derive the expression M B → B

5.2.2 Default Logic

- An alternative logic for performing default-based reasoning is **Default Logic (DL)**, in which a new class of inference rules is introduced. In this approach, we allow inference rules of the form:

$$\frac{A : B}{C}$$

Such a rule should be read as, "**If A is provable and it is consistent to assume B then conclude C**"

- For example, given the two rules:

$$\frac{A : B}{B} \qquad \frac{\neg A : B}{B}$$

and no assertion about A, no conclusion about B will be drawn, since neither inference rule applies.

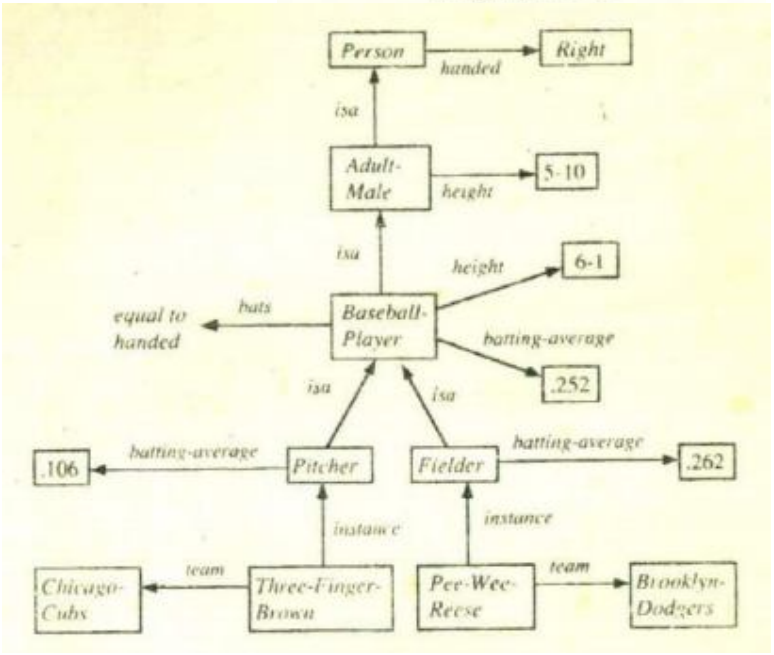
5.2.3 Abduction

- Standard logic performs deduction. Given two axioms:
  ∀x : A(x)→B(x)
  A(C)
  we can conclude B(C) using deduction.
- For example, suppose the axiom we have is:
  ∀x: Measles(x) → Spots(x)
  The axiom says that having measles implies having spots. But suppose we notice spots, we might like to conclude measles. Such a conclusion is not licensed by the rules of standard logic and it may be wrong. Deriving conclusions in this way is another form of default reasoning, called **"Abductive Reasoning".**
- The process of abductive reasoning can be described as:
  "Given two wff 's (A → B) and (B) for any expressions A and B, if it is consistent to assume A. do so."

5.2.4 Inheritance

- One very common use of nonmonotonic reasoning is as a basis for inheriting attribute values from a prototype description of a class to the individual entities that belong to the class.
- "An object inherits attribute values from all the classes of which it is a member unless doing so leads to a contradiction, in which case a value from a more restricted class has precedence over a value from a broader class."
- A rule for the inheritance of a default value for the height of a baseball player as:

$$\frac{Baseball\text{-}Player(x) : height(x, 6\text{-}1)}{height(x, 6\text{-}1)}$$



- Now suppose we assert Pitcher(Three-Finger-Brown). Since this enables us to conclude that Three-Finger-Brown is a baseball player, our rule allows us to conclude that his height is 6-1. If, on the other hand, we had asserted a conflicting value for Three Finger's height, and if we had an axiom like:

$$\forall x, y, z : height(x, y) \wedge height(x, z) \rightarrow y = z,$$

which prohibits someone from having more than one height, then we would not be able to apply the default rule. Thus an explicitly stated value will block the inheritance of a default value.

- Now suppose, consider the default rule for the height of adult males.

$$\frac{Adult\text{-}Male(x) : height(x, 5\text{-}10)}{height(x, 5\text{-}10)}$$

the resulting theory contains two extensions: one in which our first rule tells Brown's height is 6-1 and one in which this new rule applies and Brown's height is 5-10. Neither of these extensions is preferred. In order to state that we prefer to get a value from the more specific category, baseball player, we could rewrite the default rule for adult males in general as:

$$\frac{Adult\text{-}Male(x) : \neg Baseball\text{-}Player(x) \wedge height(x, 5\text{-}10)}{height(x, 5\text{-}10)}$$

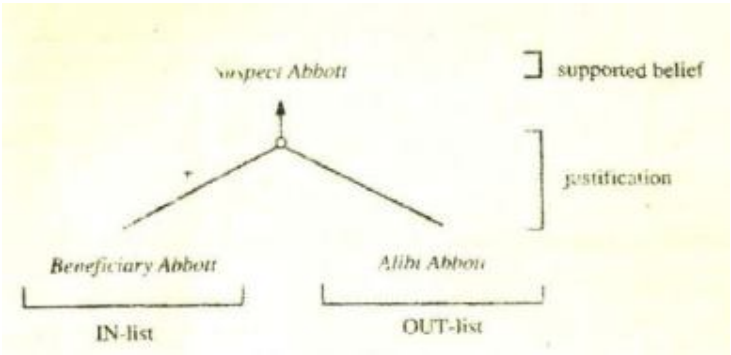
5.3 Truth Maintenance Systems (TMS)

5.3.1 Justification-Based Truth Maintenance Systems (JTMS)

- The idea of a truth maintenance system or TMS is providing the ability to do **dependency-directed backtracking** and so to support **non-monotonic reasoning.**
- **Dependency-Directed Backtracking:**
  - We need to know a fact, F. which cannot be derived monotonically from what we already know, but which can be derived by making some assumption A which seems plausible.
  - So we make assumption A, derive F, and then derive some additional facts G and H from F. We later derive some other facts M and N, but they are completely independent of A and F.
  - A little while later, a new fact comes in that invalidates A. We need to rescind our proof of F, and also our proofs of G and H since they depended on F. But what about M and N? They didn't depend on F, so there is no logical need to invalidate them.
  - But if we use a conventional backtracking scheme, we have to back up past conclusions in the order in which we derived them. So we have to backup past M and N, thus undoing them, in order to get back to F, G, H and A.
  - To get around this problem, we need a slightly different notion of backtracking, one that is based on logical dependencies rather than the chronological order in which decisions were made. This new method is called **"Dependency-directed backtracking".**
- A TMS allows assertions to be connected via a spreadsheet-like network of dependencies.
- Let us see how TMS works in ABC Murder story. Initially, we might believe that Abbott is the primary suspect because he was a beneficiary of the deceased and he had no alibi. There are three assertions here, a specific combination of which we now believe, although we may change our beliefs later. We can represent these assertions in shorthand as follows:

- **Suspect** Abbott (Abbott is the primary murder suspect.)
- **Beneficiary** Abbott (Abbott is a beneficiary of the victim.)
- **Alibi** Abbott (Abbott was at an Albany hotel at the time.)

➢ A TMS dependency network offers a purely syntactic domain-independent way to represent belief and change it consistently.



➢ The assertion **Suspect Abbott** has an associated TMS Justification. Each justification consists of two parts: **an IN-list and an OUT-List.** In the figure, the assertions on the IN-list are connected to the justification by "+" links, those on the OUT-list by "-"links. The justification is connected by an arrow to the assertion that it supports. In the justification shown, there is exactly one assertion in each list. Beneficiary Abbott is in the IN-list and Alibi Abbott is in the OUT-list. Such a justification says that Abbott should be a suspect just when it is believed that he is a beneficiary and it is not believed that he has an alibi.

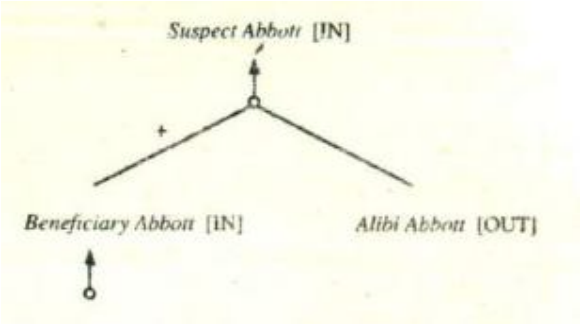
➢ Assertions (usually called nodes) in a TMS dependency network are believed when they have a **valid justification**. A justification is valid if every assertion in the IN-list is believed and none of those in the OUT list is. A justification is nonmonotonic if its OUT-list is not empty, or, recursively, if any assertion in its IN-list has a nonmonotonic justification. Otherwise, it is monotonic.

➢ In a TMS network, nodes are labelled with a belief status. If the assertion corresponding to the node should be believed, then in the TMS it is labelled **IN**. If there is no good reason to believe the assertion, then it is labelled **OUT**.

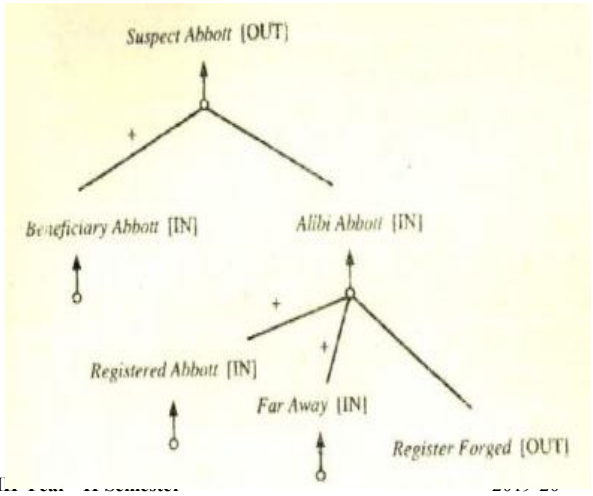
➢ The labelling task of a TMS is to label each node so that two criteria about the dependency network structure are met. The first criterion is consistency: every node labelled IN is supported by at least one valid justification and all other nodes are labelled OUT.

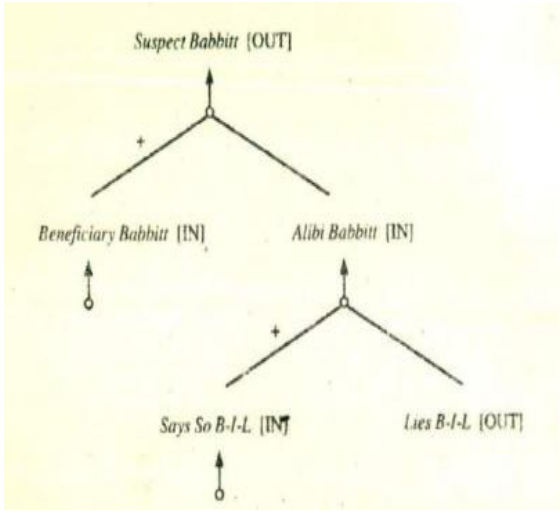
➢ **A justification is valid if every node in its IN-list is labelled IN and every node in it's OUT-list is labelled OUT.**

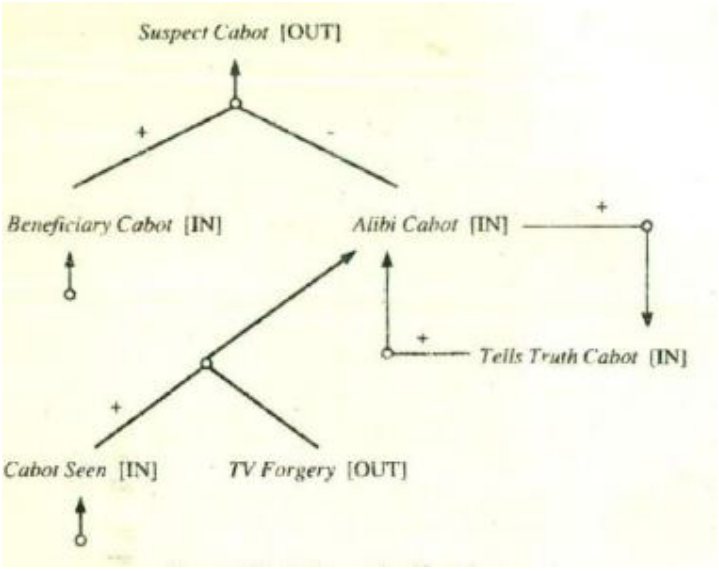
- We are told that Abbott is a beneficiary. We have no further justification for this fact; we must simply accept it. For such facts, we give a premise justification: a justification with empty IN and OUT-lists. Premise justifications are always valid. The figure shows such a justification added to the network and a consistent labelling for that network, which shows Suspect Abbott labelled IN.



- That Abbot is the primary suspect represents an initial slate of the murder investigation. Subsequently, the detective establishes that Abbott is listed on the register of a good Albany hotel on the day of the murder. This provides a valid reason to believe Abbott's alibi. The figure shows the effect of adding such justification to the network. That Abbott was registered at the hotel. Registered Abbott, was told to us and has a premise justification and so is labelled IN. That the hotel is far away is also asserted as a premise. The register might have been forged, but we have no good reason to believe it was. Thus Register Forged lacks any justification and is labelled OUT. That Abbott was on the register of a far away hotel and the lack of belief that the register was forged will cause the appropriate forward rule to fire and create a justification for Alibi Abbott, which is thus labelled IN. This means that Suspect Abbott no longer has a valid justification and must be labelled OUT. Abbott is no longer a suspect.







- The key reasoning operations that are performed by a JTMS:
  - consistent labelling
  - contradiction resolution
- A set of important reasoning operations that a JTMS does not perform, includes:
  - Applying rules to derive conclusions.
  - Creating justifications for the results of applying rules (although justifications are created as part of contradiction resolution).
  - Choosing among alternative ways of resolving a contradiction.
  - Detecting contradictions.

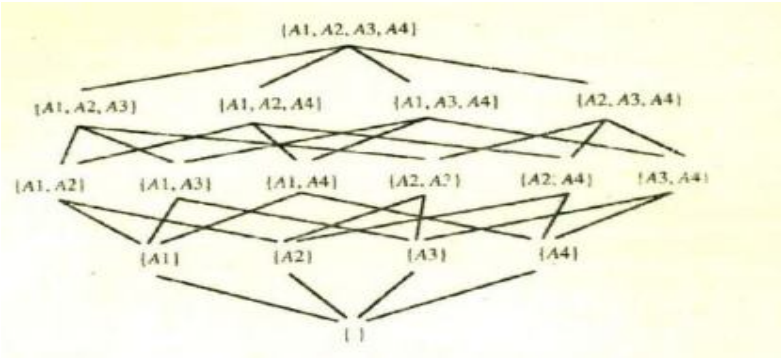
5.3.2 Logic-Based Truth Maintenance Systems (LTMS)

- In a JTMS, the nodes in the network are treated as atoms by the TMS, which assumes no relationships among them except the ones that are explicitly stated in the justifications.
- A JTMS has no problem simultaneously labelling both P and ⌐P, IN. For example, we could have represented explicitly both Lies B-I-L and Not Lies B-I-L and labelled both of them IN. No contradiction will be detected automatically. In an LTMS, on the other hand, a contradiction would be asserted automatically in such a case.

5.3.3 Assumption-Based Truth Maintenance Systems (ATMS)

- The ATMS is an alternative way of implementing nonmonotonic reasoning. In both JTMS and LTMS systems, a single line of reasoning is pursued at a time, and dependency-directed backtracking occurs whenever it is necessary to change the system's assumptions.
- In an ATMS, alternative paths are maintained in parallel. Backtracking is avoided the expense of maintaining multiple contexts, each of which corresponds to a set of consistent assumptions. As reasoning proceeds in an ATMS-based system, the universe of consistent contexts is pruned as contradictions are discovered.
- The remaining consistent contexts are used to label assertions, thus indicating the contexts in which each assertion has a valid justification.
- Assertions that do not have a valid justification in any consistent context can be pruned from consideration by the problem solver. As the set of consistent contexts gets smaller, so too does the set of assertions that can consistently be believed by the problem solver.
- An ATMS system works breadth-first, considering all possible contexts at once, while both JTMS and LTMS systems operate depth-first.
- The ATMS, uses a problem solver whose job is to:
  - Create nodes that correspond to assertions (both those that are given as axion1s and those that are derived by the problem solver).
  - Associate with each such node one or more justifications, each of which describes reasoning chain that led to the node.
  - Inform the ATMS of inconsistent contexts.
- The role of the ATMS system is to:
  - Propagate inconsistencies, thus ruling out contexts, that include sub contexts (sets of assertions) that are known to be inconsistent
  - Label each problem solver node with the contexts in which it has valid justification. This is done by combining contexts that correspond to the components of a justification.

Al Λ A2 Λ.... Λ An→C



- One problem with this approach is that given a set of n assumptions, the number of possible contexts that may have to be considered is 2^n.
- Consider how an ATMS-based problem solver works, in ABC Murder story. Again, our goal is to find a primary suspect. We need the assumptions:
  - Al. Hotel register was forged.
  - A2. Hotel register was not forged.
  - A3. Babbitt's brother-in-law tied.
  - A4. Babbitt's brother-in-law did not lie.
  - A5. Cabot lied.
  - A6. Cabot did not lie.
  - A7. Abbott, Babbitt, and Cabot are the only possible suspects.
  - A8. Abbott. Babbitt, and Cabot are not the only suspects

	Nodes	Justifications		Node Labels
[1]	Register was not forged	{A2}	{A2}	{A2}
[2]	Abbott at hotel	[1] → [2]	{A2}	{A2}
[3]	B-1-L didn't lie	{4}	{A4}	{A4}
[4]	Babbitt at B-1-L	[3] → [4]	{A4}	{A4}
[5]	Cabot didn't lie	{6}	{A6}	{A6}
[6]	Cabot at ski show	[5] → [6]	{A6}	{A6}
[7]	A, B, C only suspects	{A7}	{A7}	{A7}
[8]	Prime Suspect Abbott	[7] ∧ [13] ∧ [14] → [8]	{A7,A4,A6}	{A7,A4,A6}
[9]	Prime Suspect Babbitt	[7] ∧ [12] ∧ [14] → [9]	{A7,A2,A6}	{A7,A2,A6}
[10]	Prime Suspect Cabot	[7] ∧ [12] ∧ [13] → [10]	{A7,A2,A4}	{A7,A2,A4}
[11]	A, B, C not only suspects	{A8}	{A8}	{A8}
[12]	Not prime suspect Abbott	[2] → [12]	{A2}	{A2},{A8}
		[11] → [12]	{A8}	
		[9] → [12]	{A7,A2,A6}	
		[10] → [12]	{A7,A2,A4}	
[13]	Not prime suspect Babbitt	[4] → [13]	{A4}	{A4},{A8}
		[11] → [13]	{A8}	
		[8] → [13]	{A7,A4,A6}	
		[10] → [13]	{A7,A4,A2}	
[14]	Not prime suspect Cabot	[6] → [14]	{A6}	{A6},{A8}
		[11] → [14]	{A8}	
		[8] → [14]	{A7,A4,A6}	
		[9] → [14]	{A7,A2,A6}	

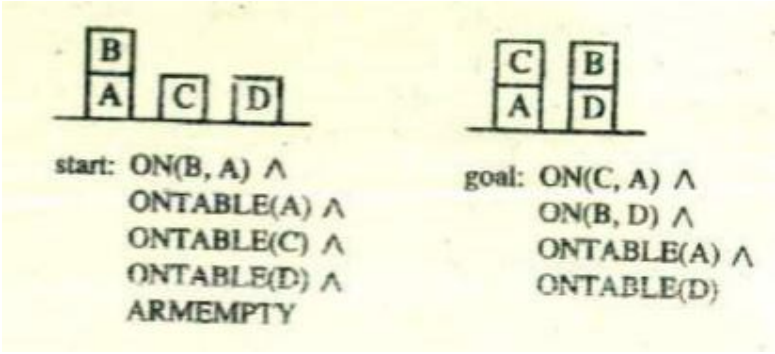
5.4 Planning

- Divide the problem that must be solved into smaller pieces and solve those pieces separately, in the extent that that is possible.
- It becomes important to be able to work on small pieces of a problem separately and then to combine the partial solutions at the end into a complete problem solution.
- First of all, we must avoid having to recompute the entire problem state when we move from one state in the next. Instead, we want to consider only that part of the state that may have changed.

- The second important way in which decomposition can make the solution of hard problems easier is the division of a single difficult problem into several, easier, sub problems.
- **Planning** focus on ways of decomposing the original problem into appropriate subparts and on ways of recording and handling interactions among the subparts as they are detected during the problem-solving process.
- **Planning** refers to the process of computing several steps of a problem-solving procedure before executing any of them.
- **Components of a Planning System:**
  - Choose the best rule to apply next, based on the best available heuristic information.
  - Apply the chosen rule to compute the new problem state that arises from its application.
  - Detect when a solution has been found.
  - Detect dead ends so that they can be abandoned and the system's effort directed in more fruitful directions.
  - Detect when an almost correct solution has been found and eniploy special techniques to make it totally correct.

5.4.1 Goal Stack Planning

- It is the technique developed for solving compound goals that may interact.
- The problem solver makes use of a single stack that contains both **goals and operators** that have been proposed to satisfy those goals.
- The problem solver relies on a database that describes the current situation and a set of operators described as: **PRECONDITION ADD, and DELETE lists.**

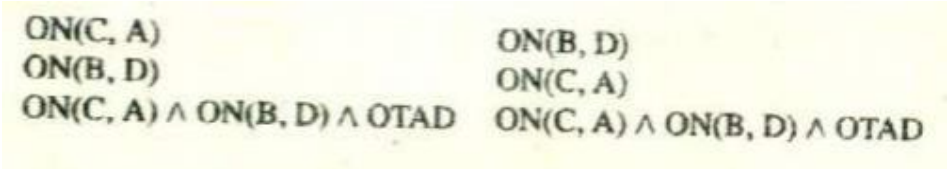


When we begin solving this problem, the goal stack is simply:

ON(C, A) Λ ON(B, D) Λ ONTABLE(A) Λ ONTABLE(D)

- we want to separate this problem into four subproblems, one for each component of the original goal. Two of the subproblems: ONTABLE(A) and ONTABLE(D) are already true in the initial state.

- So we will work on only the remaining two. Depending on the order in which we want to tackle the subproblems, there are two goal stacks that could be created as our first step, where each line represents one goal on the stack. Let OTAD is an abbreviation for ONTABLE(A) **Λ** ONTABLE(D)



- At each succeeding step of the problem-solving process, the **top goal on the stack** will be pursued.
- When a sequence of operators that satisfies it is found, that sequence is applied to the state description, yielding a new description. Next, the goal that is then at the top of the stack is explored and an attempt is made to satisfy it, starting from the situation that was produced as a result of satisfying the first goal.
- This process continues until the goal stack is empty. Then, as one last check, the original goal is compared to the final state derived from the application of the chosen operators. If any components of the goal are not satisfied in that state, then those unsolved parts of the goal are reinserted onto the stack and the process resumed.
- Let us assume that we choose first to explore alternative 1. We first check to see whether ON(C, A) is true in the current state. Since it is not, we check for operators that could cause it to be true. Of the four operators there is only one: **STACK**, and it is called with C and A. So we place STACK(C, A) or the stack in place of ON(C, A),yielding:

STACK(C, A)
ON(B, D)
ON(C, A) Λ ON(B, D) Λ OTAD

- STACK(C, A) replaced ON(C, A) because after performing the STACK we are guaranteed that ON(C, A) will hold. But in order to apply STACK(C, A), its preconditions must hold, so we must establish them as subgoals. Again we must separate a compound goal:

CLEAR(A) Λ HOLDING(C)

- HOLDING(x) is very easy to achieve. At most, it is necessary to put down something else and then to pick up the desired object. If HOLDING is one of several goals to be achieved at once, it should be tackled last. This produces the new goal stack:

CLEAR(A)
HOLDING(C)
CLEAR(A) Λ HOLDING(C)

STACK(C, A)
ON(B, D)
ON(C, A) Λ ON(B. D) Λ OTAD

- Next we check to see if CLEAR(A) is true. It is not. The only operator that could make it true is UNSTACK(B, A). So we will attempt to apply it. This produces the goal stack:

ON(B, A)
CLEAR(B)
ARMEMPTY
ON(B, A) Λ CLEAR(B) Λ ARMEMPTY
UNSTACK(B, A)
HOLDING(C)
CLEAR(A) Λ HOLDING(C)
STACK(C, A)
ON(B, D)
ON(C, A) Λ 0N(B, D) Λ OTAD

- when we compare the top element of the goal stack, ON(B, A, it is satisfied. So we pop it off and consider the next goal, CLEAR(B). Any block with no blocks on it is clear, a theorem prover, could derive CLEAR(B). So that goal, too, can be popped from the stack. The third precondition for UNSTACK(B, A) is ARMEMPTY, and it is also true, so it can be popped off the stack. The next element on the stack is the combined goal representing all of the prccondiiions for UNSTACK(B, A). We check to see, it is satisfied and combined goal can be popped from the stack.
- Now the top element of the stack is the operator UNSTACK(B, A). We are now guaranteed that its preconditions are satisfied, so it can be applied to produce a new world model from which the rest of the problem solving process can continue.
- At this point, the database corresponding to the world model is:

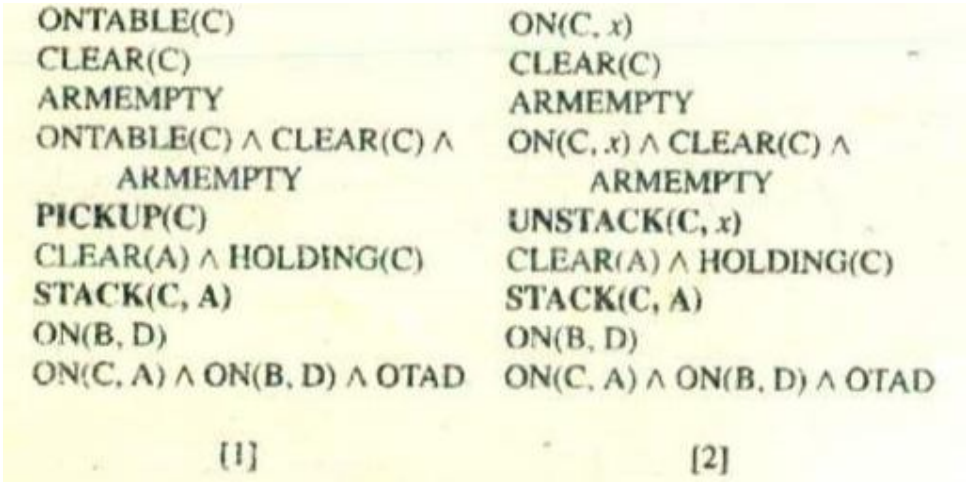
ONTABLE(A) Λ ONTABLE(C) Λ ONTABLE(D) Λ HOLDING(B) Λ CLEAR(A)

The goal stack now is:

HOLDING(C)
CLEAR(A) Λ HOLDING(C)
STACK(C, A)
ON(B, D)
ON(C, A) Λ ON(B, D) Λ OTAD

- We now attempt to satisfy the goal HOLDING(C). There are two operators that might make HOLDING(C) true: **PICKUP(C) and UNSTACK(C,x),** where x could be any block from which C could be

unstacked. So we create two branches of the search tree, corresponding to the following goal stacks:



- Alternative 1 is better than unstacking it because it is not currently on anything. The top element on the goal stack is ONTABLEC), which is already satisfied, so we pop it off. The next element is CLEAR(C), which is also satisfied, so we pop it off. The remaining precondition of PICKUP(C) is ARMEMPTY, which is not satisfied since HOLDING(B) is true.
- There are two operators that could be applied to make ARMEMPTY true **STACK(B ,x) and PUTDOWN(B).** In other words, we can either put B on the table or we can put it on another block. It would be more efficient to put B on D, than to place it on table. So we choose to apply **STACK(B , D).** This makes the goal stack:

CLEAR(D)
HOLDING(B)
CLEAR(D) Λ HOLDING(B)
STACK(B, D)
ONTABLE(C) Λ CLEAR(C) Λ ARMEMPTY
PICKUP(C)
CLEAR(A) Λ HOLDING(C)
STACK(C,A)
ON(B, D)
ON(C, A) Λ ON(B, D)) Λ OTAD

- CLEAR(D) and HOLDING(B) are both true. Now the operation STACK(B, D) can be perfonmd, producing the world model:

ONTABLE(A) Λ ONTABLE(C) Λ ONTABLE(D) Λ ON(B, D) Λ ARMEMPTY

- All of the preconditions for PICKUP(C) arc now satisfied so it can be executed. Then all of the preconditions of STACK(C, A) are true, so it can be executed. Now we can begin work on the second part of our original goal.

ON(B, D). But it has already been satisfied by the operations that were used to satisfy the first subgoal. So we now pop ON(B,D) off the goal stack.

- We then do check of the combined goal ON(C, A) **Λ** ON (B, D) **Λ** ONTABLE(A) **Λ** ONTABLE(D) to make sure that all four parts still hold. The problem solver can now halt and return as its answer the plan:

1.UNSTACK(B, A)
2. STACK(B,D)
3. PICKUP(C)
4. STACK(C,A)

5.4.2 Hierarchical Planning

- In order to solve hard problems, a problem solver may have to generate long plans. In order to do that efficiently, it is important to be able to eliminate some of the details of the problem until a solution that addresses the main issues is found. Then an attempt can be made to till in the appropriate details.
- Early attempts to do this involved the use of macro-operators, in which larger operators were built from smaller ones. But in this approach, no details were eliminated from the actual descriptions of the operators.
- A better approach is developed in the **ABSTRIPS** system, which actually planned in a **hierarchy of abstraction spaces,** in each of which preconditions at a lower level of abstraction were ignored.
- Suppose you want to visit a friend in Europe, but you have a limited amount of cash to spend. It makes sense to check air fares first, since finding an affordable flight will be the most difficult part of the task. You should not worry about getting out of your driveway, planning a route to the airport, or parking your car until you are sure you have a flight.
- **ABSTRIPS** approach to problem solving is as follows: First solve the problem completely, considering only preconditions whose criticality value is the highest possible. These values reflect the expected difficulty of satisfying the precondition.
- Once this is done, use the constructed plan as the outline of a complete plan and consider preconditions at the next-lowest criticality level. Augment the plan with operators that satisfy those preconditions. Again, in choosing operators, ignore all preconditions whose criticality is less than the level now being considered.
- Continue this process of considering less and less critical preconditions, until all of the preconditions of the original rules have been considered.
- Because this process explores entire plans at one level of detail before it looks at the lower-level details of any one of them. it is called **"length-first search".**

- The **assignment of appropriate criticality** values is crucial to the success of this hierarchical planning method. Those preconditions that no operators can satisfy are clearly the most critical.
- For example, if we are trying to solve a problem involving a robot moving around in a house and we are considering the operator PUSH-THROUGH-DOOR, the precondition that there exist a door big enough for the robot to get through is of high criticality since there is nothing we can do about it if it is not true.
- Given these values, the basic process can function in very much the same way that nonhierarchical planititig does But effort will not be wasted tilling in the details of plans that do not even come ciot fo solving the problem.

Unit- V

Artificial Intelligence: Reasoning in Uncertain Situations

(Open Elective –I)

Assignment-Cum-Tutorial Questions

Objective Questions

1. In _____reasoning, the axioms/or the rules of inference are extended to make it possible to reason with incomplete information.

(a) Monotonic (b) Non monotonic (c) Logical (d) Inferential

2. Define the term " Monotonicity".
3. Any inference that depends on the lack of some piece of knowledge is a _____ inference. [ ]

(a) Monotonic (b) Non monotonic (c) Logical (d) Inferential

4. What is a "Justification"?
5. In non monotonic logic, operator **M** is read as _____.
6. _____categorizes and organizes the information in a meaningful way.

(a) Knowledge Engineer (b) Human Expert (c) User (d) Tool

7. What is "default logic"?
8. Given two axioms:

∀x : A(x)→B(x)
A(C)
We can conclude _____) using deduction. [ ]

(a) B(C) (b) B(x) (c) A(x) (d) none

9. A justification is valid if every assertion in the IN-list is believed and none of those in the OUT list is. (TRUE/FALSE) **[ ]**
10. _____refers to the process of computing several steps of a problem-solving procedure before executing any of them.
11. The process which process explores entire plans at one level of detail before it looks at the lower-level details of any one of them is called _____ first search. [ ]

(a) Length (b) Breadth (c) Best (d) Depth

SECTION-B

Descriptive Questions

1. Discuss ABC murder story and how reasoning is performed using non-monotonic reasoning?
2. Illustrate the situations where conventional reasoning is inadequate and how non monotonic reasoning serves the purpose?
3. Explain the logics of non monotonic reasoning?
4. Explain the following:

(i) Non monotonic logic
(ii) Default Logic
(iii) Abduction

5. Illustrate how knowledge can reasoned using JTMS?
6. Discuss about dependency directed backtracking.
7. How can knowledge be reasoned using ATMS? Explain.
8. What is Planning? List the components of Planning?
9. Explain about Goal Stack Planning with a suitable example.
10. Explain about hierarchical planning