if DLS(i, target, limit?1)
return true

return false

An important thing to note is, we visit top level nodes multiple times. The last (or max depth) level is visited once, second last level is visited twice, and so on. It may seem expensive, but it turns out to be not so costly, since in a tree most of the nodes are in the bottom level. So it does not matter much if the upper levels are visited multiple times.

Planning

Blocks World Problem

In order to compare the variety of methods of planning, we should find it useful to look at all of them in a single domain that is complex enough that the need for each of the mechanisms is apparent yet simple enough that easy-to-follow examples can be found.

- There is a flat surface on which blocks can be placed.
- There are a number of square blocks, all the same size.
- They can be stacked one upon the other.
- There is robot arm that can manipulate the blocks.

Actions of the robot arm

1. UNSTACK(A, B): Pick up block A from its current position on block B.
2. STACK(A, B): Place block A on block B.
3. PICKUP(A): Pick up block A from the table and hold it.
4. PUTDOWN(A): Put block A down on the table.

Notice that the robot arm can hold only one block at a time.

Predicates

- In order to specify both the conditions under which an operation may be performed and the results of performing it, we need the following predicates:
    1. ON(A, B): Block A is on Block B.
    2. ONTABLES(A): Block A is on the table.
    3. CLEAR(A): There is nothing on the top of Block A.
    4. HOLDING(A): The arm is holding Block A.
    5. ARMEMPTY: The arm is holding nothing.

Robot problem-solving systems (STRIPS)

- List of new predicates that the operator causes to become true is ADD List
- Moreover, List of old predicates that the operator causes to become false is DELETE List
- PRECONDITIONS list contains those predicates that must be true for the operator to be applied.

STRIPS style operators for BLOCKs World

STACK(x, y)
P: CLEAR(y)^HOLDING(x)
D: CLEAR(y)^HOLDING(x)
A: ARMEMPTY^ON(x, y)
UNSTACK(x, y)
PICKUP(x)
P: CLEAR(x) ^ ONTABLE(x) ^ARMEMPTY
D: ONTABLE(x) ^ ARMEMPTY

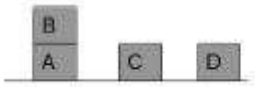
A: HOLDING(x)
PUTDOWN(x)

Goal Stack Planning

To start with goal stack is simply:

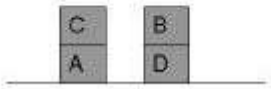
- ON(C,A)^ON(B,D)^ONTABLE(A)^ONTABLE(D)

This problem is separate into four sub-problems, one for each component of the goal.
Two of the sub-problems ONTABLE(A) and ONTABLE(D) are already true in the initial state.



Start:

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



Goal: ON(C,A)^ON(B,D)^ ONTABLE(A)^ONTABLE(D)

Alternative 1: Goal Stack:

- ON(C,A)
- ON(B,D)
- ON(C,A)^ON(B,D)^OTAD

Alternative 2: Goal stack:

- ON(B,D)
- ON(C,A)
- ON(C,A)^ON(B,D)^OTAD

Exploring Operators

- Pursuing alternative 1, we check for operators that could cause ON(C, A)
- Out of the 4 operators, there is only one STACK. So it yields:
    - STACK(C,A)
    - ON(B,D)
    - ON(C,A)^ON(B,D)^OTAD
- Preconditions for STACK(C, A) should be satisfied, we must establish them as sub-goals:
    - CLEAR(A)
    - HOLDING(C)
    - CLEAR(A)^HOLDING(C)
    - STACK(C,A) o ON(B,D)
    - ON(C,A)^ON(B,D)^OTAD
- Here we exploit the Heuristic that if HOLDING is one of the several goals to be achieved at once, it should be tackled last.

Goal stack Planning

- Next, we see if CLEAR(A) is true. It is not. The only operator that could make it true is UNSTACK(B, A). Also, This produces the goal stack:
    - ON(B, A)
    - CLEAR(B)
    - ON(B,A)^CLEAR(B)^ARMEMPTY
    - UNSTACK(B, A)
    - HOLDING(C)

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

- We see that we can pop predicates on the stack till we reach HOLDING(C) for which we need to find a suitable operator.
- Moreover, The operators that might make HOLDING(C) true: PICKUP(C) and UNSTACK(C, x). Without looking ahead, since we cannot tell which of these operators is appropriate. Also, we create two branches of the search tree corresponding to the following goal stacks:

ALT1:	ALT2:
ONTABLE(C)	ON(C, x)
CLEAR(C)	CLEAR(C)
ARMEMPTY	ARMEMPTY
ONTABLE(C) ^CLEAR(C)^ARMEMPTY	ON(C,x)^CLEAR(C)^ARMEMPTY
PICKUP(C)	UNSTACK(C,x)
CLEAR(A)^HOLDING(C)	CLEAR(A)^HOLDING(C)
STACK(C,A)	STACK(C,A)
ON(B,D)	ON(B,D)
ON(C,A)^ON(B,D)^OTAD	ON(C,A)^ON(B,D)^OTAD

Complete plan

1. UNSTACK(C, A)
2. PUTDOWN(C)
3. PICKUP(A)
4. STACK(A, B)
5. UNSTACK(A, B)
6. PUTDOWN(A)
7. PICKUP(B)
8. STACK(B, C)
9. PICKUP(A)
10. STACK(A,B)

Planning Components

- Methods which 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 are often called as planning.
- 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.

- The most widely used technique for selecting appropriate rules to apply is first to isolate a set of differences between desired goal state and then to identify those rules that are relevant to reduce those differences.
- If there are several rules, a variety of other heuristic information can be exploited to choose among them.

Apply the chosen rule to compute the new problem state that arises from its application.

- In simple systems, applying rules is easy. Each rule simply specifies the problem state that would result from its application.
- In complex systems, we must be able to deal with rules that specify only a small part of the complete problem state.
- One way is to describe, for each action, each of the changes it makes to the state description.

Detect when a solution has found.

- A planning system has succeeded in finding a solution to a problem when it has found a sequence of operators that transform the initial problem state into the goal state.
- How will it know when this has done?
- In simple problem-solving systems, this question is easily answered by a straightforward match of the state descriptions.
- One of the representative systems for planning systems is predicate logic. Suppose that as a part of our goal, we have the predicate P(x).
- To see whether P(x) satisfied in some state, we ask whether we can prove P(x) given the assertions that describe that state and the axioms that define the world model.

Detect dead ends so that they can abandon and the system's effort directed in more fruitful directions.

- As a planning system is searching for a sequence of operators to solve a particular problem, it must be able to detect when it is exploring a path that can never lead to a solution.
- The same reasoning mechanisms that can use to detect a solution can often use for detecting a dead end.
- If the search process is reasoning forward from the initial state. It can prune any path that leads to a state from which the goal state cannot reach.
- If search process reasoning backward from the goal state, it can also terminate a path either because it is sure that the initial state cannot reach or because little progress made.

Detect when an almost correct solution has found and employ special techniques to make it totally correct.

- The kinds of techniques discussed are often useful in solving nearly decomposable problems.
- One good way of solving such problems is to assume that they are completely decomposable, proceed to solve the sub-problems separately. And then check that when the sub-solutions combined. They do in fact give a solution to the original problem.

Goal Stack Planning

- Methods which 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 are often called as planning.
- Planning refers to the process of computing several steps of a problem-solving procedure before executing any of them.
-

Goal Stack Planning Method

- In this method, the problem solver makes use of a single stack that contains both goals and operators. That have proposed to satisfy those goals.

- The problem solver also relies on a database that describes the current situation and a set of operators described as PRECONDITION, ADD and DELETE lists.
- The goal stack planning method attacks problems involving conjoined goals by solving the goals one at a time, in order.
- A plan generated by this method contains a sequence of operators for attaining the first goal, followed by a complete sequence for the second goal etc.
- At each succeeding step of the problem-solving process, the top goal on the stack will pursue.
- When a sequence of operators that satisfies it, found, that sequence applied to the state description, yielding new description.
- Next, the goal that then at the top of the stack explored. And an attempt made to satisfy it, starting from the situation that 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 compared to the final state derived from the application of the chosen operators.
- If any components of the goal not satisfied in that state. Then those unsolved parts of the goal reinserted onto the stack and the process resumed.

Nonlinear Planning using Constraint Posting

- Difficult problems cause goal interactions.
- The operators used to solve one sub-problem may interfere with the solution to a previous sub-problem.
- Most problems require an intertwined plan in which multiple sub-problems worked on simultaneously.
- Such a plan is called nonlinear plan because it is not composed of a linear sequence of complete sub-plans.

Constraint Posting

- The idea of constraint posting is to build up a plan by incrementally hypothesizing operators, partial orderings between operators, and binding of variables within operators.
- At any given time in the problem-solving process, we may have a set of useful operators but perhaps no clear idea of how those operators should order with respect to each other.
- A solution is a partially ordered, partially instantiated set of operators to generate an actual plan. And we convert the partial order into any number of total orders.

Constraint Posting versus State Space search

State Space Search

- Moves in the space: Modify world state via operator
- Model of time: Depth of node in search space
- Plan stored in Series of state transitions

Constraint Posting Search

- Moves in the space: Add operators, Oder Operators, Bind variables Or Otherwise constrain plan
- Model of Time: Partially ordered set of operators
- Plan stored in Single node

Algorithm: Nonlinear Planning (TWEAK)

1. Initialize S to be the set of propositions in the goal state.
2. Remove some unachieved proposition P from S.

3. Moreover, Achieve P by using step addition, promotion, DE clobbering, simple establishment or separation.
4. Review all the steps in the plan, including any new steps introduced by step addition, to see if any of their preconditions unachieved. Add to S the new set of unachieved preconditions.
5. Also, If S is empty, complete the plan by converting the partial order of steps into a total order, instantiate any variables as necessary.
6. Otherwise, go to step 2.

Hierarchical Planning

- In order to solve hard problems, a problem solver may have to generate long plans.
- 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 make to fill in the appropriate details.
- Early attempts to do this involved the use of macro operators, in which larger operators were built from smaller ones.
- In this approach, no details eliminated from actual descriptions of the operators.

ABSTRIPS

A better approach developed in ABSTRIPS systems which actually planned in a hierarchy of abstraction spaces, in each of which preconditions at a lower level of abstraction ignored.
ABSTRIPS approach 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.
- To do this, do exactly what STRIPS did, but simply ignore the preconditions of lower than peak criticality.
- Once this 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.
- Because this approach explores entire plans at one level of detail before it looks at the lower-level details of any one of them, it has called length-first approach.

The assignment of appropriate criticality value is crucial to the success of this hierarchical planning method.
Those preconditions that no operator can satisfy are clearly the most critical.
Example, solving a problem of moving the robot, for applying an 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.

Other Planning Techniques

Reactive Systems

- The idea of reactive systems is to avoid planning altogether, and instead, use the observable situation as a clue to which one can simply react.
- A reactive system must have access to a knowledge base of some sort that describes what actions should be taken under what circumstances.
- A reactive system is very different from the other kinds of planning systems we have discussed. Because it chooses actions one at a time.
- It does not anticipate and select an entire action sequence before it does the first thing.
- The example is a Thermostat. The job of the thermostat is to keep the temperature constant inside a room.
- Reactive systems are capable of surprisingly complex behaviors.
- The main advantage reactive systems have over traditional planners is that they operate robustly in domains that are difficult to model completely and accurately.
- Reactive systems dispense with modeling altogether and base their actions directly on their perception of the world.
- Another advantage of reactive systems is that they are extremely responsive since they avoid the combinatorial explosion involved in deliberative planning.
- This makes them attractive for real-time tasks such as driving and walking.

Other Planning Techniques

Triangle tables

- Provides a way of recording the goals that each operator expected to satisfy as well as the goals that must be true for it to execute correctly.

Meta-planning

- A technique for reasoning not just about the problem solved but also about the planning process itself.

Macro-operators

- Allow a planner to build new operators that represent commonly used sequences of operators.

Case-based planning:

- Re-uses old plans to make new ones.

UNDERSTANDING

Understanding is the simplest procedure of all human beings. Understanding means ability to determine some new knowledge from a given knowledge. For each action of a problem, the mapping of some new actions is very necessary. Mapping the knowledge means transferring the knowledge from one representation to another representation. For example, if you will say "I need to go to New Delhi" for which you will book the tickets. The system will have "understood" if it finds the first available plane to New Delhi. But if you will say the same thing to you friends, who knows that your family lives in "New Delhi", he/she will have "understood" if he/she realizes that there may be a problem or occasion in your family. For people, understanding applies to inputs from all the senses. Computer understanding has so far been applied primarily to images, speech and typed languages. It is important to keep in mind that the success or failure of an "understanding" problem can rarely be measured in an absolute sense but must instead be measured with respect to a particular task to be performed. There are some factors that contribute to the difficulty of an understanding problem.

(a) If the target representation is very complex for which you cannot map from the original representation.

(b) There are different types of mapping factors may arise like one-to-one, one-to-many and many to many.
(c) Some noise or disturbing factors are also there.
(d) The level of interaction of the source components may be complex one.
(e) The problem solver might be unknown about some more complex problems.
(f) The intermediary actions may also be unavailable.

Consider an example of an English sentence which is being used for communication with a keyword based data retrieval system. Suppose I want to know all about the temples in India. So I would need to be translated into a representation such as The above sentence is a simple sentence for which the corresponding representation may be easy to implement. But what for the complex queries?

Consider the following query.
"Ram told Sita he would not eat apple with her. He has to go to the office".

This type of complex queries can be modeled with the conceptual dependency representation which is more complex than that of simple representation. Constructing these queries is very difficult since more informationare to be extracted. Extracting more information will require some more knowledge. Also the type of mapping process is not quite easy to the problem solver. Understanding is the process of mapping an input from its original form to a more useful one. The simplest kind of mapping is "one-toone".
In one-to-one mapping each different problems would lead to only one solution. But there are very few inputs which are one-to-one. Other mappings are quite difficult to implement. Many-to-one mappings are frequent is that free variation is often allowed, either because of the physical limitations of that produces the inputs or because such variation simply makes the task of generating the inputs.
Many to one mapping require that the understanding system know about all the ways that a target representation can be expressed in the source language. One-to-many mapping requires a great deal of domain knowledge in order to make the correct choice among the available target representation.
The mapping process is simplest if each component can be mapped without concern for the other components of the statement. If the number of interactions increases, then the complexity of the problem will increase. In many understanding situations the input to which meaning should be assigned is not always the input that is presented to the under stander.
Because of the complex environment in which understanding usually occurs, other things often interfere with the basic input before it reaches the under stander. Hence the understanding will be more complex if there will be some sort of noise on the inputs.

Natural Language Processing

Introduction to Natural Language Processing

- Language meant for communicating with the world.
- Also, By studying language, we can come to understand more about the world.
- If we can succeed at building computational mode of language, we will have a powerful tool for communicating with the world.
- Also, We look at how we can exploit knowledge about the world, in combination with linguistic facts, to build computational natural language systems.

Natural Language Processing (NLP) problem can divide into two tasks:

1. Processing written text, using lexical, syntactic and semantic knowledge of the language as well as the required real-world information.
2. Processing spoken language, using all the information needed above plus additional knowledge about phonology as well as enough added information to handle the further ambiguities that arise in speech.

Steps in Natural Language Processing

Morphological Analysis

- Individual words analyzed into their components and non-word tokens such as punctuation separated from the words.

Syntactic Analysis

- Linear sequences of words transformed into structures that show how the words relate to each other.
- Moreover, Some word sequences may reject if they violate the language's rule for how words may combine.

Semantic Analysis

- The structures created by the syntactic analyzer assigned meanings.
- Also, A mapping made between the syntactic structures and objects in the task domain.
- Moreover, Structures for which no such mapping possible may reject.

Discourse integration

- The meaning of an individual sentence may depend on the sentences that precede it. And also, may influence the meanings of the sentences that follow it.

Pragmatic Analysis

- Moreover, The structure representing what said reinterpreted to determine what was actually meant.

Summary

- Results of each of the main processes combine to form a natural language system.
- All of the processes are important in a complete natural language understanding system.
- Not all programs are written with exactly these components.
- Sometimes two or more of them collapsed.
- Doing that usually results in a system that is easier to build for restricted subsets of English but one that is harder to extend to wider coverage.

Steps Natural Language Processing

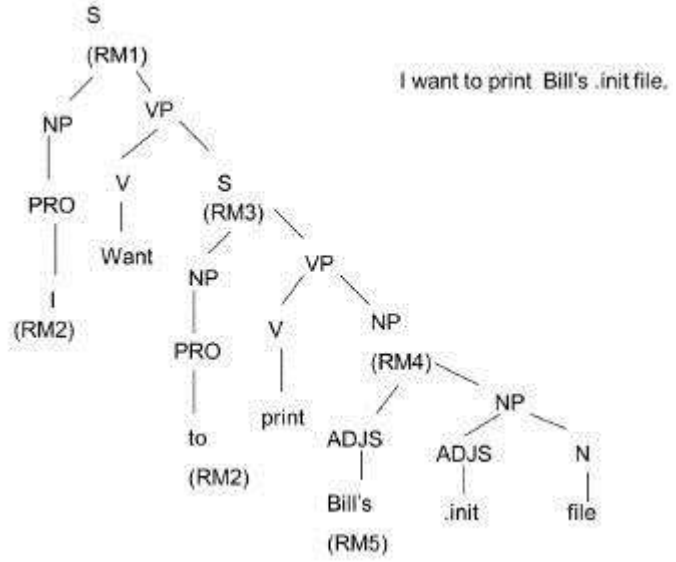
Morphological Analysis

- Suppose we have an English interface to an operating system and the following sentence typed: I want to print Bill's .init file.
- The morphological analysis must do the following things:

- Pull apart the word “Bill’s” into proper noun “Bill” and the possessive suffix “’s”
- Recognize the sequence “.init” as a file extension that is functioning as an adjective in the sentence.
- This process will usually assign syntactic categories to all the words in the sentence.

Syntactic Analysis

- A syntactic analysis must exploit the results of the morphological analysis to build a structural description of the sentence.
- The goal of this process, called parsing, is to convert the flat list of words that form the sentence into a structure that defines the units that represented by that flat list.
- The important thing here is that a flat sentence has been converted into a hierarchical structure. And that the structure corresponds to meaning units when a semantic analysis performed.
- Reference markers (set of entities) shown in the parenthesis in the parse tree.
- Each one corresponds to some entity that has mentioned in the sentence.
- These reference markers are useful later since they provide a place in which to accumulate information about the entities as we get it.



Semantic Analysis

- The semantic analysis must do two important things:
    1. It must map individual words into appropriate objects in the knowledge base or database.
    2. It must create the correct structures to correspond to the way the meanings of the individual words combine with each other.

Discourse Integration

- Specifically, we do not know whom the pronoun “I” or the proper noun “Bill” refers to.
- To pin down these references requires an appeal to a model of the current discourse context, from which we can learn that the current user is USER068 and that the only person named “Bill” about whom we could be talking is USER073.
- Once the correct referent for Bill known, we can also determine exactly which file referred to.

Pragmatic Analysis

- The final step toward effective understanding is to decide what to do as a result.

- One possible thing to do to record what was said as a fact and done with it.
- For some sentences, a whose intended effect is clearly declarative, that is the precisely correct thing to do.
- But for other sentences, including this one, the intended effect is different.
- We can discover this intended effect by applying a set of rules that characterize cooperative dialogues.
- The final step in pragmatic processing to translate, from the knowledge-based representation to a command to be executed by the system.

Syntactic Processing

- Syntactic Processing is the step in which a flat input sentence converted into a hierarchical structure that corresponds to the units of meaning in the sentence. This process called parsing.
- It plays an important role in natural language understanding systems for two reasons:
    1. Semantic processing must operate on sentence constituents. If there is no syntactic parsing step, then the semantics system must decide on its own constituents. If parsing is done, on the other hand, it constrains the number of constituents that semantics can consider.
    2. Syntactic parsing is computationally less expensive than is semantic processing. Thus it can play a significant role in reducing overall system complexity.
- Although it is often possible to extract the meaning of a sentence without using grammatical facts, it is not always possible to do so.
- Almost all the systems that are actually used have two main components:
    1. A declarative representation, called a grammar, of the syntactic facts about the language.
    2. A procedure, called parser that compares the grammar against input sentences to produce parsed structures.

Grammars and Parsers

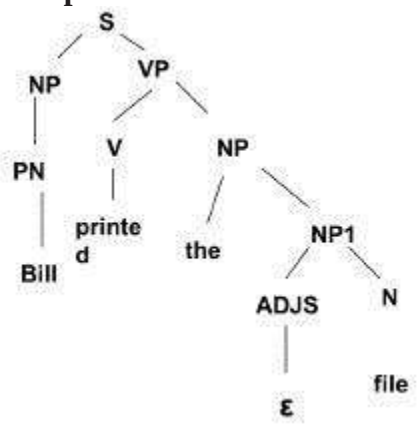
- The most common way to represent grammars is a set of production rules.
- The first rule can read as "A sentence composed of a noun phrase followed by Verb Phrase"; the Vertical bar is OR; ε represents the empty string.
- Symbols that further expanded by rules called non-terminal symbols.
- Symbols that correspond directly to strings that must found in an input sentence called terminal symbols.
- Grammar formalism such as this one underlies many linguistic theories, which in turn provide the basis for many natural language understanding systems.
- Pure context-free grammars are not effective for describing natural languages.
- NLPs have less in common with computer language processing systems such as compilers.
- Parsing process takes the rules of the grammar and compares them against the input sentence.
- The simplest structure to build is a Parse Tree, which simply records the rules and how they matched.
- Every node of the parse tree corresponds either to an input word or to a non-terminal in our grammar.
- Each level in the parse tree corresponds to the application of one grammar rule.

Example for Syntactic Processing – Augmented Transition Network

Syntactic Processing is the step in which a flat input sentence is converted into a hierarchical structure that corresponds to the units of meaning in the sentence. This process called parsing. It plays an important role in natural language understanding systems for two reasons:

1. Semantic processing must operate on sentence constituents. If there is no syntactic parsing step, then the semantics system must decide on its own constituents. If parsing is done, on the other hand, it constrains the number of constituents that semantics can consider.
2. Syntactic parsing is computationally less expensive than is semantic processing. Thus it can play a significant role in reducing overall system complexity.

Example: A Parse tree for a sentence: Bill Printed the file



The grammar specifies two things about a language:

1. Its weak generative capacity, by which we mean the set of sentences that contained within the language. This set made up of precisely those sentences that can completely match by a series of rules in the grammar.
2. Its strong generative capacity, by which we mean the structure to assign to each grammatical sentence of the language.

Augmented Transition Network (ATN)

- An augmented transition network is a top-down parsing procedure that allows various kinds of knowledge to incorporated into the parsing system so it can operate efficiently.
- ATNs build on the idea of using finite state machines (Markov model) to parse sentences.
- Instead of building an automaton for a particular sentence, a collection of transition graphs built.
- A grammatically correct sentence parsed by reaching a final state in any state graph.
- Transitions between these graphs simply subroutine calls from one state to any initial state on any graph in the network.
- A sentence determined to be grammatically correct if a final state reached by the last word in the sentence.
- The ATN is similar to a finite state machine in which the class of labels that can attach to the arcs that define the transition between states has augmented.

Arcs may label with:

- Specific words such as “in’.
- Word categories such as noun.

- Procedures that build structures that will form part of the final parse.
- Procedures that perform arbitrary tests on current input and sentence components that have identified.

Semantic Analysis

- The structures created by the syntactic analyzer assigned meanings.
- A mapping made between the syntactic structures and objects in the task domain.
- Structures for which no such mapping is possible may rejected.
- The semantic analysis must do two important things:
    - It must map individual words into appropriate objects in the knowledge base or database.
    - It must create the correct structures to correspond to the way the meanings of the individual words combine with each other. Semantic Analysis AI
- Producing a syntactic parse of a sentence is only the first step toward understanding it.
- We must produce a representation of the meaning of the sentence.
- Because understanding is a mapping process, we must first define the language into which we are trying to map.
- There is no single definitive language in which all sentence meaning can describe.
- The choice of a target language for any particular natural language understanding program must depend on what is to do with the meanings once they constructed.
- Choice of the target language in Semantic Analysis AI
    - There are two broad families of target languages that used in NL systems, depending on the role that the natural language system playing in a larger system:
    - When natural language considered as a phenomenon on its own, as for example when one builds a program whose goal is to read the text and then answer questions about it. A target language can design specifically to support language processing.
    - When natural language used as an interface language to another program (such as a db query system or an expert system), then the target language must legal input to that other program. Thus the design of the target language driven by the backend program.

Discourse and Pragmatic Processing

To understand a single sentence, it is necessary to consider the discourse and pragmatic context in which the sentence was uttered.
There are a number of important relationships that may hold between phrases and parts of their discourse contexts, including:

1. Identical entities. Consider the text:
    - Bill had a red balloon. o John wanted it.
    - The word “it” should identify as referring to the red balloon. These types of references called anaphora.
2. Parts of entities. Consider the text:
    - Sue opened the book she just bought.
    - The title page was torn.
    - The phrase “title page” should be recognized as part of the book that was just bought.

3. Parts of actions. Consider the text:
    - John went on a business trip to New York.
    - He left on an early morning flight.
    - Taking a flight should recognize as part of going on a trip.
4. Entities involved in actions. Consider the text:
    - My house was broken into last week.
    - Moreover, They took the TV and the stereo.
    - The pronoun “they” should recognize as referring to the burglars who broke into the house.
5. Elements of sets. Consider the text:
    - The decals we have in stock are stars, the moon, item and a flag.
    - I’ll take two moons.
    - Moons mean moon decals.
6. Names of individuals:
    - Dev went to the movies.
7. Causal chains
    - There was a big snow storm yesterday.
    - So, The schools closed today.
8. Planning sequences:
    - Sally wanted a new car
    - She decided to get a job.
9. Implicit presuppositions:
    - Did Joe fail CS101?

The major focus is on using following kinds of knowledge:

- The current focus of the dialogue.
- Also, A model of each participant’s current beliefs.
- Moreover, The goal-driven character of dialogue.
- The rules of conversation shared by all participants.

Statistical Natural Language Processing

Formerly, many language-processing tasks typically involved the direct hand coding of rules, which is not in general robust to natural-language variation. The machine-learning paradigm calls instead for using statistical inference to automatically learn such rules through the analysis of large ***corpora*** of typical real-world examples (a *corpus* (plural, "corpora") is a set of documents, possibly with human or computer annotations).

Many different classes of machine learning algorithms have been applied to natural-language processing tasks. These algorithms take as input a large set of "features" that are generated from the input data. Some of the earliest-used algorithms, such as decision trees, produced systems of hard if-then rules similar to the systems of hand-written rules that were then common. Increasingly, however, research has focused on statistical models, which make soft, probabilistic decisions based on attaching real-valued weights to each input feature. Such models have the advantage that they can express the relative certainty of many different possible answers rather than only one, producing more reliable results when such a model is included as a component of a larger system.

Systems based on machine-learning algorithms have many advantages over hand-produced rules:

- The learning procedures used during machine learning automatically focus on the most common cases, whereas when writing rules by hand it is often not at all obvious where the effort should be directed.
- Automatic learning procedures can make use of statistical inference algorithms to produce models that are robust to unfamiliar input (e.g. containing words or structures that have not been seen before) and to erroneous input (e.g. with misspelled words or words accidentally omitted). Generally, handling such input gracefully with hand-written rules—or more generally, creating systems of hand-written rules that make soft decisions—is extremely difficult, error-prone and time-consuming.
- Systems based on automatically learning the rules can be made more accurate simply by supplying more input data. However, systems based on hand-written rules can only be made more accurate by increasing the complexity of the rules, which is a much more difficult task. In particular, there is a limit to the complexity of systems based on hand-crafted rules, beyond which the systems become more and more unmanageable. However, creating more data to input to machine-learning systems simply requires a corresponding increase in the number of man-hours worked, generally without significant increases in the complexity of the annotation process.

Spell Checking

Spell checking is one of the applications of natural language processing that impacts billions of users daily. A good introduction to spell checking can be found on Peter Norvig's webpage. The article introduces a simple 21-line spell checker implementation in Python combining simple language and error models to predict the word a user intended to type. **The language model estimates how likely a given word `c` is in the language for which the spell checker is designed, this can be written as `P(C)`. The error model estimates the probability `P(w|c)` of typing the misspelled version `w` conditionally to the intention of typing the correctly spelled word `c`.**The spell checker then returns word `c` corresponding to the highest value of `P(w|c)P(c)` among all possible words in the language.

Module 3

LEARNING

Learning is the improvement of performance with experience over time.
Learning element is the portion of a learning AI system that decides how to modify the performance element and implements those modifications.
We all learn new knowledge through different methods, depending on the type of material to be learned, the amount of relevant knowledge we already possess, and the environment in which the learning takes place. There are five methods of learning . They are,

1. Memorization (rote learning)
2. Direct instruction (by being told)
3. Analogy
4. Induction