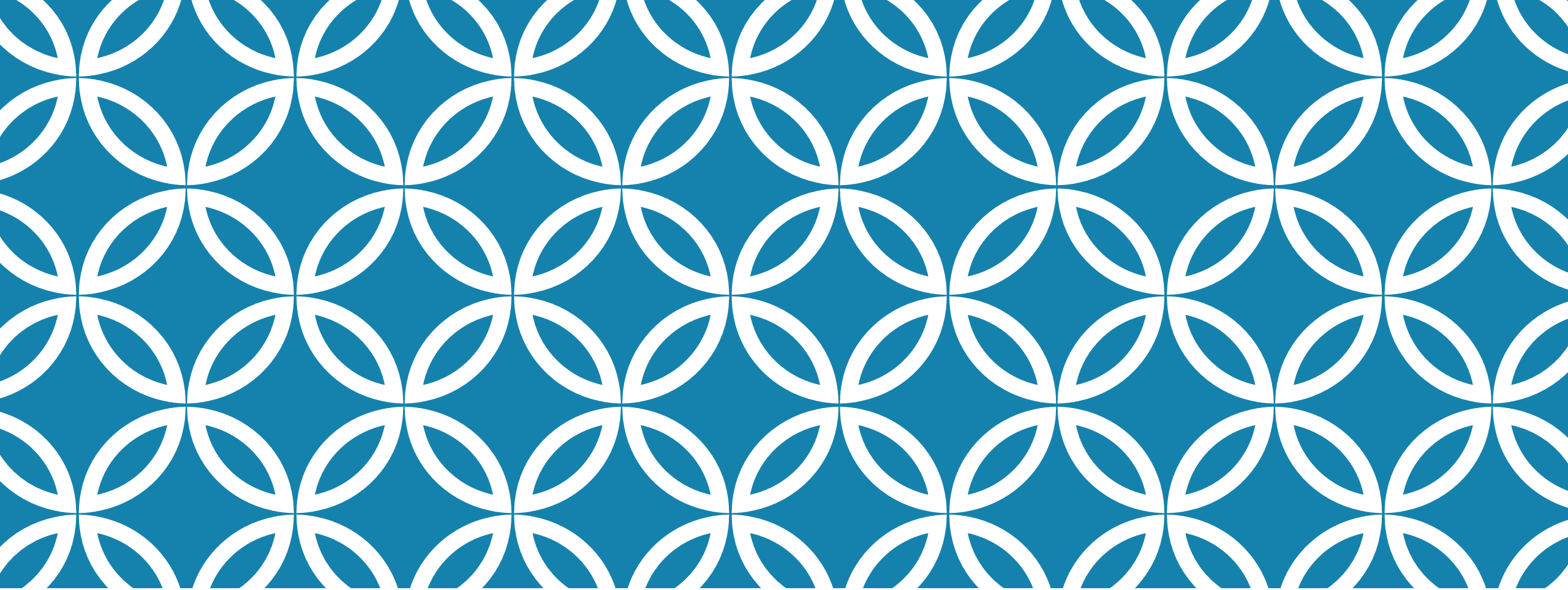
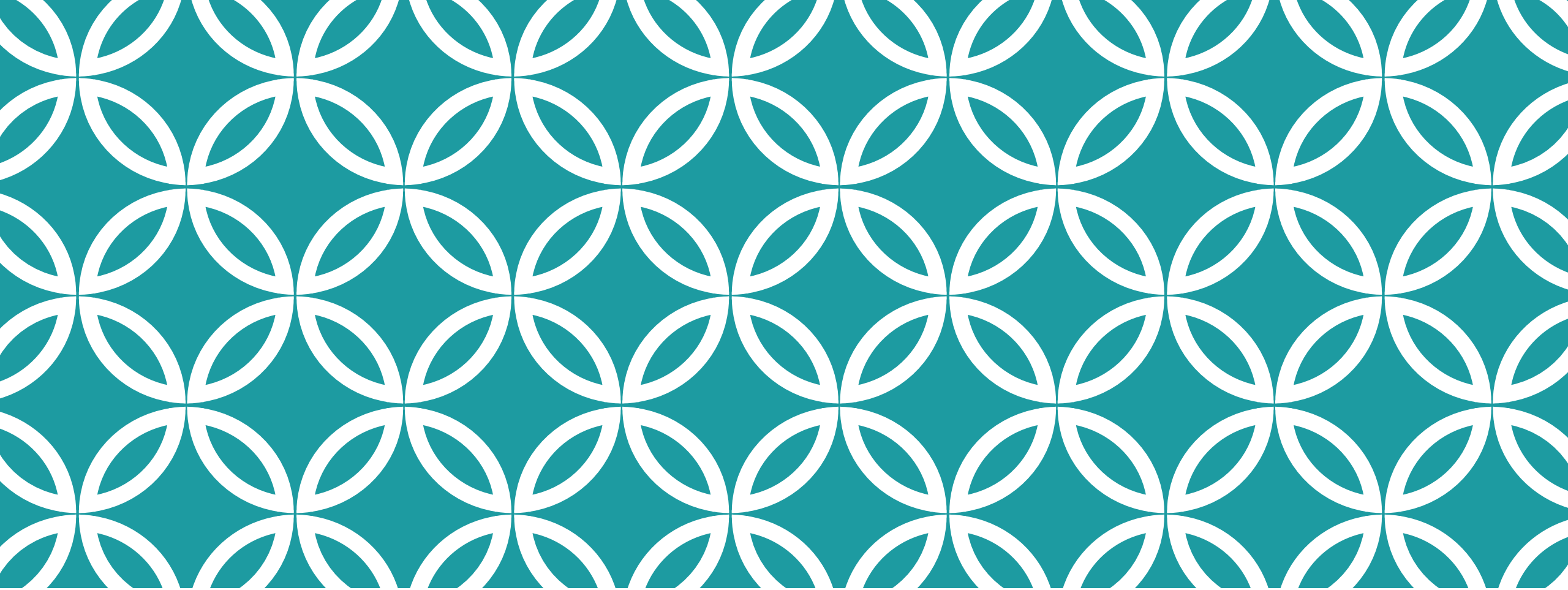


MOTION AND GAMING

Chapter 5



WEAK AND STRONG SLOT-AND-FILLER STRUCTURES

SLOT-AND-FILLER STRUCTURES

The knowledge in slot-and-filler systems is structured as a set of entities and their attributes.

A slot is an attribute value pair in its simplest form.

A filler is a value that a slot can take.

Two kinds of slot-and-filler structures:

- Weak slot-and-filler structures
- Strong slot-and-filler structures

SLOT-AND-FILLER STRUCTURES

Weak slot-and-filler structures:

- Semantics Nets
- and Frames

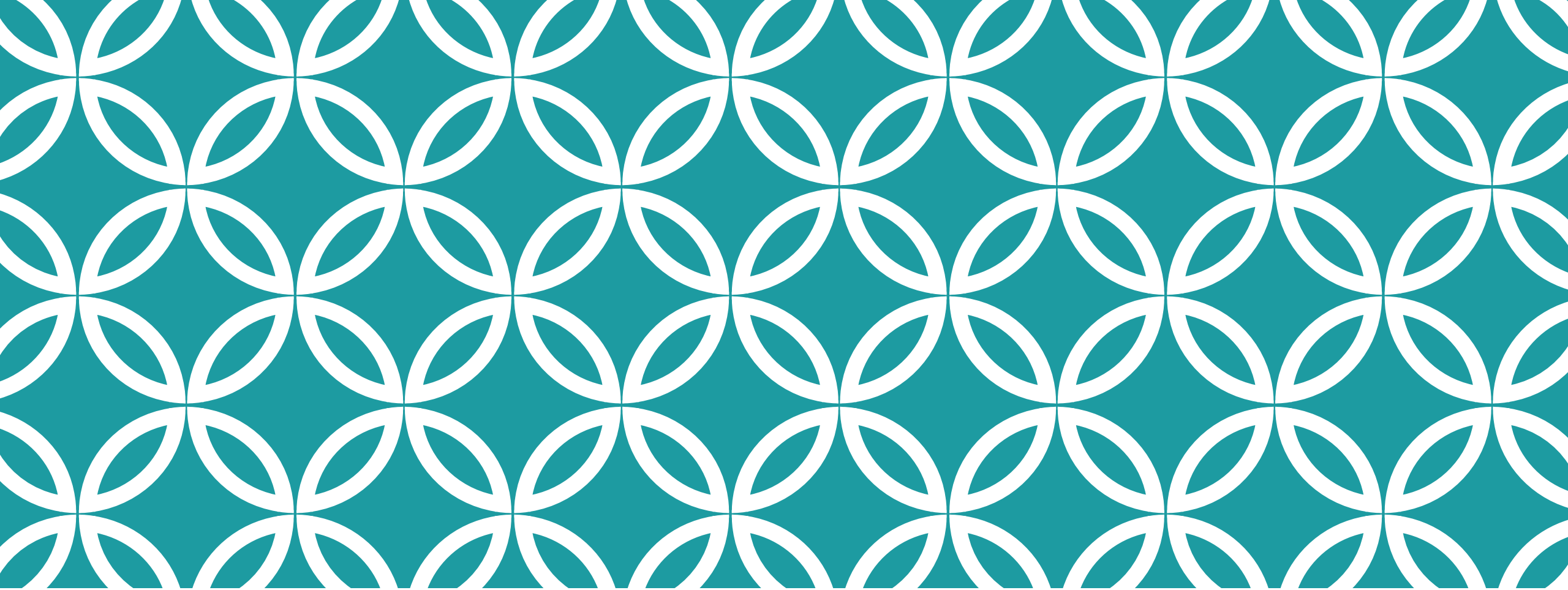
Strong slot-and-filler structures:

- Scripts
- Conceptual dependencies

WEAK VS STRONG SLOT-AND-FILLER STRUCTURES

The main problem with semantic networks and frames is that they lack formality, there is no specific guideline on how to use the representation.

Strong slot-and-filler structures typically represent links between objects according to more rigid rules, specific notions of what types of object and relations between them are provided.



GAME PLAYING

BOARD GAMES

Games are the abstractions of interactions between agents.

In game playing, we will focus on board games like chess, in which the two players play a sequence of alternating moves and the outcome is determined only when the game ends.

Board games: Checkers, Chess, Othello, Go

CLASSES OF GAMES

Two person: Have exactly 2 players

Zero sum: Total payoff is zero. One player's gain is other player's loss. One wins, the other loses.

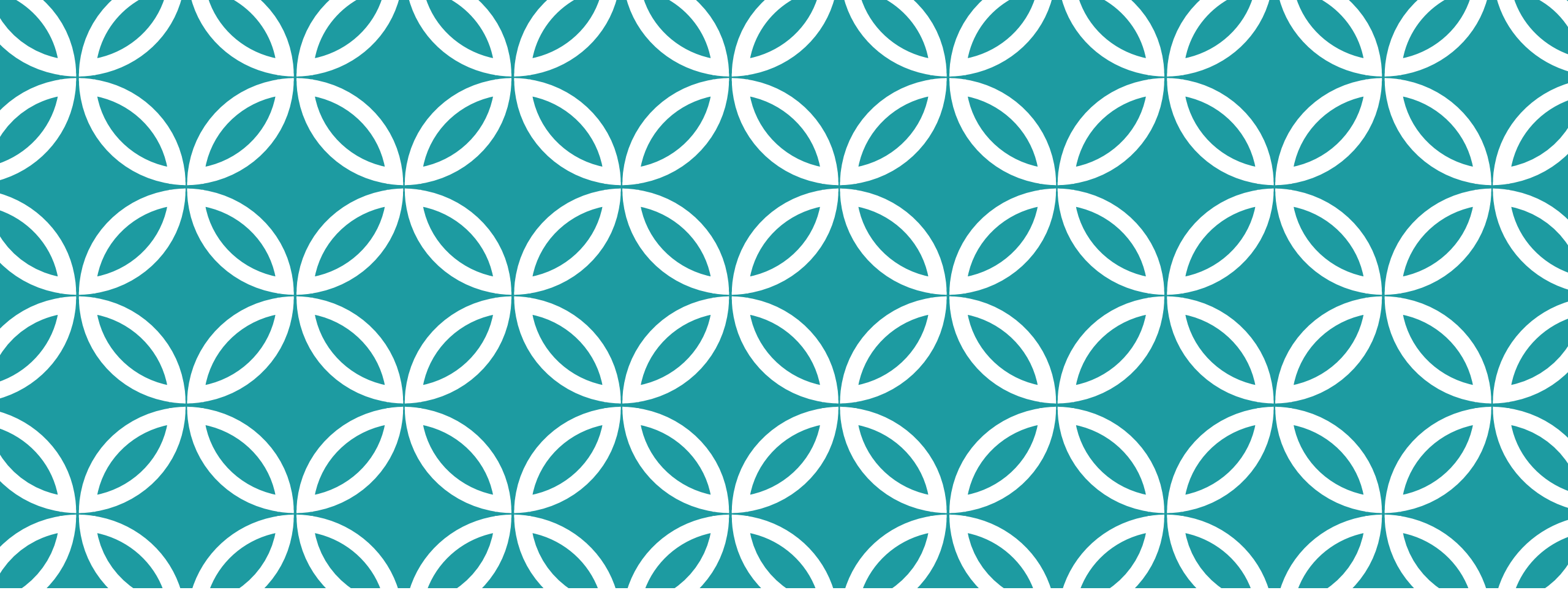
Complete information: Both players can see the board and thus know the options the other player has.

Alternate moves: Players take turns.

Deterministic: There is no element of chance in the moves that one can make. Adding dice to a board game introduces an element of chance.

GAME PLAYING ALGORITHMS

- Minimax search
- Minimax search with alpha-beta pruning
- Iterative deepening



PLANNING

PLANNING

The task of coming up with a sequence of actions that will achieve a goal is called planning.

PROBLEM-SOLVING AGENT VS PLANNING AGENT

Let us consider what can happen when an ordinary problem-solving agent using standard search algorithms: depth-first, A * , and so on, comes up against large, real-world problems.

The most obvious difficulty is that the problem-solving agent can be overwhelmed by irrelevant actions.

Consider the task of buying a book of AI from an online bookseller.

Suppose there is one buying action for each 10-digit ISBN number, for a total of 10 billion actions.

The most obvious difficulty is that the problem-solving agent can be overwhelmed by irrelevant actions.

Consider the task of buying a book of AI from an online bookseller.

Suppose there is one buying action for each 10-digit ISBN number, for a total of 10 billion actions.

- The search algorithm would have to examine the outcome states of all 10 billion actions to find one that satisfies the goal, which is to own a copy of ISBN 0137903952.

The most obvious difficulty is that the problem-solving agent can be overwhelmed by irrelevant actions.

Consider the task of buying a book of AI from an online bookseller.

Suppose there is one buying action for each 10-digit ISBN number, for a total of 10 billion actions.

- The search algorithm would have to examine the outcome states of all 10 billion actions to find one that satisfies the goal, which is to own a copy of ISBN 0137903952.
- A sensible planning agent, on the other hand, should be able to work back from an explicit goal description such as Have(ISBN 0137903952) and generate the action Buy(ISBN 0137903952) directly.
  - To do this, the agent simply needs the general knowledge that Buy(x) results in Have(x).
  - Given this knowledge and the goal, the planner can decide in a single unification step that Buy(ISBN 0137903952) is the right action.

The next difficulty is finding a good heuristic function.

Suppose the agent's goal is to buy four different books online.

Then there will be 10^{40} plans of just four steps, so searching without an accurate heuristic is out of the question.

The next difficulty is finding a good heuristic function.

Suppose the agent's goal is to buy four different books online.

Then there will be 10^{40} plans of just four steps, so searching without an accurate heuristic is out of the question.

A good heuristic estimate for the cost of a state is the number of books that remain to be bought; unfortunately, this insight is not obvious to a problem-solving agent, because it sees the goal test only as a black box that returns true or false for each state.

Therefore, the problem-solving agent lacks autonomy; it requires a human to supply a heuristic function for each new problem.

The next difficulty is finding a good heuristic function.

Suppose the agent's goal is to buy four different books online.

Then there will be 10^{40} plans of just four steps, so searching without an accurate heuristic is out of the question.

A good heuristic estimate for the cost of a state is the number of books that remain to be bought; unfortunately, this insight is not obvious to a problem-solving agent, because it sees the goal test only as a black box that returns true or false for each state.

Therefore, the problem-solving agent lacks autonomy; it requires a human to supply a heuristic function for each new problem.

On the other hand, if a planning agent has access to an explicit representation of the goal as a conjunction of subgoals, then it can use a single domain-independent heuristic: the number of unsatisfied conjuncts.

For the book-buying problem, the goal would be Have(A) ∧ Have(B) ∧ Have(C) ∧ Have(D), and a state containing Have(A) ∧ Have(C) would have cost 2.

Finally, the problem solver might be inefficient because it cannot take advantage of problem decomposition.

Consider the problem of delivering a set of overnight packages to their respective destinations, which are scattered across the country.

Finally, the problem solver might be inefficient because it cannot take advantage of problem decomposition.

Consider the problem of delivering a set of overnight packages to their respective destinations, which are scattered across the country.

It makes sense to find out the nearest airport for each destination and divide the overall problem into several subproblems, one for each airport.

Within the set of packages routed through a given airport, whether further decomposition is possible depends on the destination city.

Finally, the problem solver might be inefficient because it cannot take advantage of problem decomposition.

Consider the problem of delivering a set of overnight packages to their respective destinations, which are scattered across the country.

It makes sense to find out the nearest airport for each destination and divide the overall problem into several subproblems, one for each airport.

Within the set of packages routed through a given airport, whether further decomposition is possible depends on the destination city.

The ability to do this kind of decomposition contributes to the efficiency of planners: in the worst case, it can take O(n!) time to find the best plan to deliver n packages, but only $O((n/k)! \times k)$ time if the problem can be decomposed into k equal parts.

NEARLY DECOMPOSABLE PROBLEMS

The planner can work on subgoals independently, but might need to do some additional work to combine the resulting subplans.

For some problems, this assumption breaks down because working on one subgoal is likely to undo another subgoal. These interactions among subgoals are what makes puzzles (like the 8-puzzle) puzzling.

THE LANGUAGE OF PLANNING PROBLEMS

The STRIPS language

Representation of states:

Planners decompose the world into logical conditions and represent a state as a conjunction of positive literals.

For example, Poor ∧ Unknown might represent the state of a hapless agent.

At(Plane1, Melbourne) ∧ At(Plane2, Sydney) might represent a state in the package delivery problem.

Literals in first-order state descriptions must be ground and function-free.

Literals such as At(x, y) or At(Father(Fred), Sydney) are not allowed.

The closed-world assumption is used, meaning that any conditions that are not mentioned in a state are assumed false.

Representation of goals:

A goal is a partially specified state, represented as a conjunction of positive ground literals, such as Rich ∧ Famous or At(P2, Tahiti).

A propositional state s satisfies a goal g if s contains all the atoms in g (and possibly others).

For example, the state Rich ∧ Famous ∧ Miserable satisfies the goal Rich ∧ Famous.

Representation of actions:

An action is specified in terms of the preconditions that must hold before it can be executed and the effects that ensue when it is executed.

For example, an action for flying a plane from one location to another is:

- Action(Fly(p, from, to)
- PRECOND: At(p, from) ∧ Plane(p) ∧ Airport(from) ∧ Airport(to)
- EFFECT: ¬At(p, from) ∧ At(p, to))

This is more properly called an action schema, meaning that it represents a number of different actions that can be derived by instantiating the variables p, from, and to different constants.

In general, an action schema consists of three parts:

- The action name and parameter list: for example, Fly(p, from, to) serves to identify the action.
- The precondition is a conjunction of function-free positive literals stating what must be true in a state before the action can be executed. Any variables in the precondition must also appear in the action's parameter list.
- The effect is a conjunction of function-free literals describing how the state changes when the action is executed. A positive literal P in the effect is asserted to be true in the state resulting from the action, whereas a negative literal ¬P is asserted to be false. Variables in the effect must also appear in the action's parameter list.

First, we say that an action is applicable in any state that satisfies the precondition; otherwise, the action has no effect.

For a first-order action schema, establishing applicability will involve a substitution θ for the variables in the precondition.

For example, suppose the current state is described by:

At(P1, JFK) ∧ At(P2, SFO) ∧ Plane(P1) ∧ Plane(P2) ∧ Airport(JFK) ∧ Airport(SFO)

This state satisfies the precondition:

At(p, from) ∧ Plane(p) ∧ Airport(from) ∧ Airport(to) with substitution {p/P1, from/JFK, to/SFO} (among others).

Thus, the concrete action Fly(P1, JFK, SFO) is applicable.

Starting in state s, the result of executing an applicable action a is a state s' that is the same as s except that any positive literal P in the effect of a is added to s' and any negative literal ¬P is removed from s'.

Thus, after Fly(P1, JFK, SFO), the current state becomes At(P1, SFO) ∧ At(P2, SFO) ∧ Plane(P1) ∧ Plane(P2) ∧ Airport(JFK) ∧ Airport(SFO).

Note that if a positive effect is already in s it is not added twice, and if a negative effect is not in s, then that part of the effect is ignored.

This definition embodies the so-called STRIPS assumption: that every literal not mentioned in the effect remains unchanged.

Finally, we can define the solution for a planning problem.

In its simplest form, this is just an action sequence that, when executed in the initial state, results in a state that satisfies the goal.

ADL

In recent years, it has become clear that STRIPS is insufficiently expressive for some real domains.

As a result, many language variants have been developed.

Next figure briefly describes one important one, the Action Description Language or ADL, by comparing it with the basic STRIPS language.

In ADL, the Fly action could be written as:

- Action(Fly(p : Plane, from : Airport, to : Airport)
- PRECOND: At(p, from) ∧ (from != to)
- EFFECT: ¬At(p, from) ∧ At(p, to))

STRIPS **Language**	ADL **Language**
Only positive literals in states: $Poor \wedge Unknown$	Positive and negative literals in states: $\neg Rich \wedge \neg Famous$
Closed World Assumption: Unmentioned literals are false.	Open World Assumption: Unmentioned literals are unknown.
Effect $P \wedge \neg Q$ means add P and delete Q.	Effect $P \wedge \neg Q$ means add P and $\neg Q$ and delete $\neg P$ and Q.
Only ground literals in goals: $Rich \wedge Famous$	Quantified variables in goals: $\exists x At(P_1, x) \wedge At(P_2, x)$ is the goal of having P_1 and P_2 in the same place.
Goals are conjunctions: $Rich \wedge Famous$	Goals allow conjunction and disjunction: $\neg Poor \wedge (Famous \vee Smart)$
Effects are conjunctions.	Conditional effects allowed: **when** P: E means E is an effect only if P is satisfied.
No support for equality.	Equality predicate $(x = y)$ is built in.
No support for types.	Variables can have types, as in $(p : Plane)$.

EXAMPLE: AIR CARGO TRANSPORT

An air cargo transport problem involves loading and unloading cargo onto and off of planes and flying it from place to place.

The problem can be defined with three actions: Load, Unload, and Fly.

The actions affect two predicates:

- In(c, p) means that cargo c is inside plane p, and
- At(x, a) means that object x (either plane or cargo) is at airport a.

Note that cargo is not At anywhere when it is In a plane, so At really means "available for use at a given location."

The following plan is a solution to the problem:

[Load(C1, P1, SFO), Fly(P1, SFO, JFK), Load(C2, P2, JFK), Fly(P2, JFK, SFO)]

$$
\begin{aligned}
&Init(At(C_1, SFO) \land At(C_2, JFK) \land At(P_1, SFO) \land At(P_2, JFK) \\
&\quad \land Cargo(C_1) \land Cargo(C_2) \land Plane(P_1) \land Plane(P_2) \\
&\quad \land Airport(JFK) \land Airport(SFO)) \\
&Goal(At(C_1, JFK) \land At(C_2, SFO)) \\
&Action(Load(c, p, a), \\
&\quad \text{PRECOND: } At(c, a) \land At(p, a) \land Cargo(c) \land Plane(p) \land Airport(a) \\
&\quad \text{EFFECT: } \neg At(c, a) \land In(c, p)) \\
&Action(Unload(c, p, a), \\
&\quad \text{PRECOND: } In(c, p) \land At(p, a) \land Cargo(c) \land Plane(p) \land Airport(a) \\
&\quad \text{EFFECT: } At(c, a) \land \neg In(c, p)) \\
&Action(Fly(p, from, to), \\
&\quad \text{PRECOND: } At(p, from) \land Plane(p) \land Airport(from) \land Airport(to) \\
&\quad \text{EFFECT: } \neg At(p, from) \land At(p, to))
\end{aligned}
$$

Our representation is pure STRIPS. In particular, it allows a plane to fly to and from the same airport.

Inequality literals in ADL could prevent this.

EXAMPLE: THE SPARE TIRE PROBLEM

Consider the problem of changing a flat tire.

More precisely, the goal is to have a good spare tire properly mounted onto the car's axle, where the initial state has a flat tire on the axle and a good spare tire in the trunk.

There are just four actions: removing the spare from the trunk, removing the flat tire from the axle, putting the spare on the axle, and leaving the car unattended overnight.

We assume that the car is in a particularly bad neighbourhood, so that the effect of leaving it overnight is that the tires disappear.

$Init(At(Flat, Axle) \wedge At(Spare, Trunk))$
$Goal(At(Spare, Axle))$
$Action(Remove(Spare, Trunk),$
PRECOND: $At(Spare, Trunk)$
EFFECT: $\neg At(Spare, Trunk) \wedge At(Spare, Ground))$
$Action(Remove(Flat, Axle),$
PRECOND: $At(Flat, Axle)$
EFFECT: $\neg At(Flat, Axle) \wedge At(Flat, Ground))$
$Action(PutOn(Spare, Axle),$
PRECOND: $At(Spare, Ground) \wedge \neg At(Flat, Axle)$
EFFECT: $\neg At(Spare, Ground) \wedge At(Spare, Axle))$
$Action(LeaveOvernight,$
PRECOND:
EFFECT: $\neg At(Spare, Ground) \wedge \neg At(Spare, Axle) \wedge \neg At(Spare, Trunk)$
$\wedge \neg At(Flat, Ground) \wedge \neg At(Flat, Axle))$

The ADL description of spare tire problem is purely propositional.

It goes beyond STRIPS in that it uses a negated precondition, ¬At(Flat, Axle), for the PutOn(Spare, Axle) action.

EXAMPLE: THE BLOCKS WORLD

One of the most famous planning domains is known as the blocks world.

This domain consists of a set of cube-shaped blocks sitting on a table.

The blocks can be stacked, but only one block can fit directly on top of another.

A robot arm can pick up a block and move it to another position, either on the table or on top of another block.

The arm can pick up only one block at a time, so it cannot pick up a block that has another one on it.

The goal will always be to build one or more stacks of blocks, specified in terms of what blocks are on top of what other blocks.

For example, a goal might be to get block A on B and block C on D.

We will use $On(b, x)$ to indicate that block b is on x, where x is either another block or the table.

The action for moving block b from the top of x to the top of y will be $Move(b, x, y)$.

Now, one of the preconditions on moving b is that no other block be on it.

In first-order logic, this would be $\neg\exists x On(x, b)$ or, alternatively, $\forall x \neg On(x, b)$.

These could be stated as preconditions in ADL.

We can stay within the STRIPS language, however, by introducing a new predicate, $Clear(x)$, that is true when nothing is on x.

The action Move moves a block b from x to y if both b and y are clear.

After the move is made, x is clear but y is not.

A formal description of Move in STRIPS is:

- Action(Move(b, x, y)
- PRECOND: On(b, x) ∧ Clear(b) ∧ Clear(y),
- EFFECT: On(b, y) ∧ Clear(x) ∧ ¬On(b, x) ∧ ¬Clear(y))

Unfortunately, this action does not maintain Clear properly when x or y is the table.

When x = Table, this action has the effect Clear(Table), but the table should not become clear.

When y = Table, it has the precondition Clear(Table), but the table does not have to be clear to move a block onto it.

To fix this, we do two things.

First, we introduce another action to move a block b from x to the table:

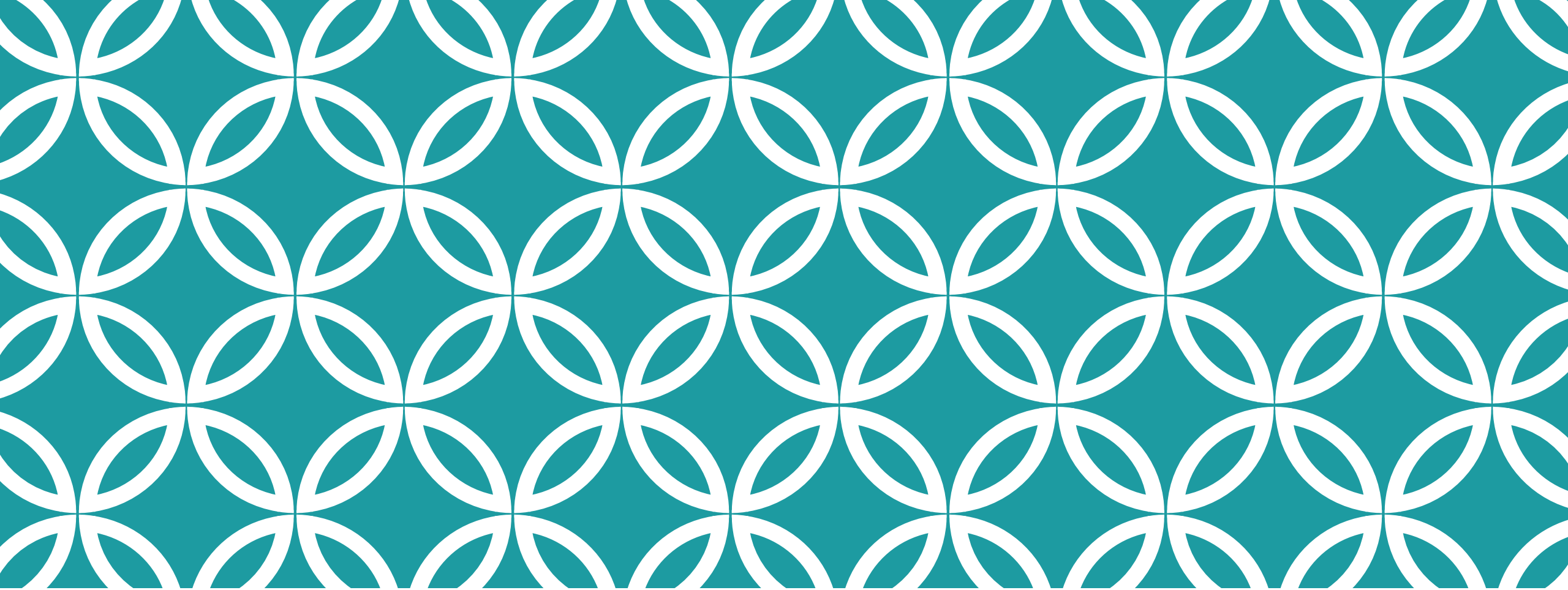
- Action(MoveToTable(b, x),
- PRECOND: On(b, x) ∧ Clear(b)),
- EFFECT: On(b, Table) ∧ Clear(x) ∧ ¬On(b, x)) .

Second, we take the interpretation of Clear(b) to be "there is a clear space on b to hold a block."

Under this interpretation, Clear(Table) will always be true.

The only problem is that nothing prevents the planner from using Move(b, x, Table) instead of MoveToTable(b, x).

We could live with this problem: it will lead to a larger-than-necessary search space, but will not lead to incorrect answers or we could introduce the predicate Block and add Block(b) $\wedge$ Block(y) to the precondition of Move.



UNDERSTANDING

UNDERSTANDING DEFINITION

To understand something is to transform it from one representation into another, where this second representation has been chosen to correspond to a set of available actions that could be performed and where the mapping has been designed so that for each event, an appropriate action will be performed.

EXAMPLE

If you say to an airline database system "I need to go to New York as soon as possible," the system would have understood it if it finds the first available plane to New York.

If you say the same thing to your best friend, who knows that your family lives in New York, he will have understood if he realizes that there may be a problem in your family and you may need some emotional support.

The success or failure of an “understanding” program can rarely be measured in an absolute sense but must instead be measured with respect to a particular task to be performed.

Computer understanding has so far been applied primarily to images, speech, typed language, and videos.

WHAT MAKES UNDERSTANDING HARD?

1. The complexity of target representation into which the matching is being done
2. The type of the mapping: one-one, many-one, one-many, or many-many
3. The level of interaction of the components of the source representation
4. The presence of noise in the input to the understander

COMPLEXITY OF THE TARGET REPRESENTATION

Suppose English sentences are being used for communication with a keyword-based data retrieval system. Then the sentence:

I want to read all about the last general elections.

would need to be translated into a representation such as:

(SEARCH KEYWORDS = ELECTION & GENERAL)

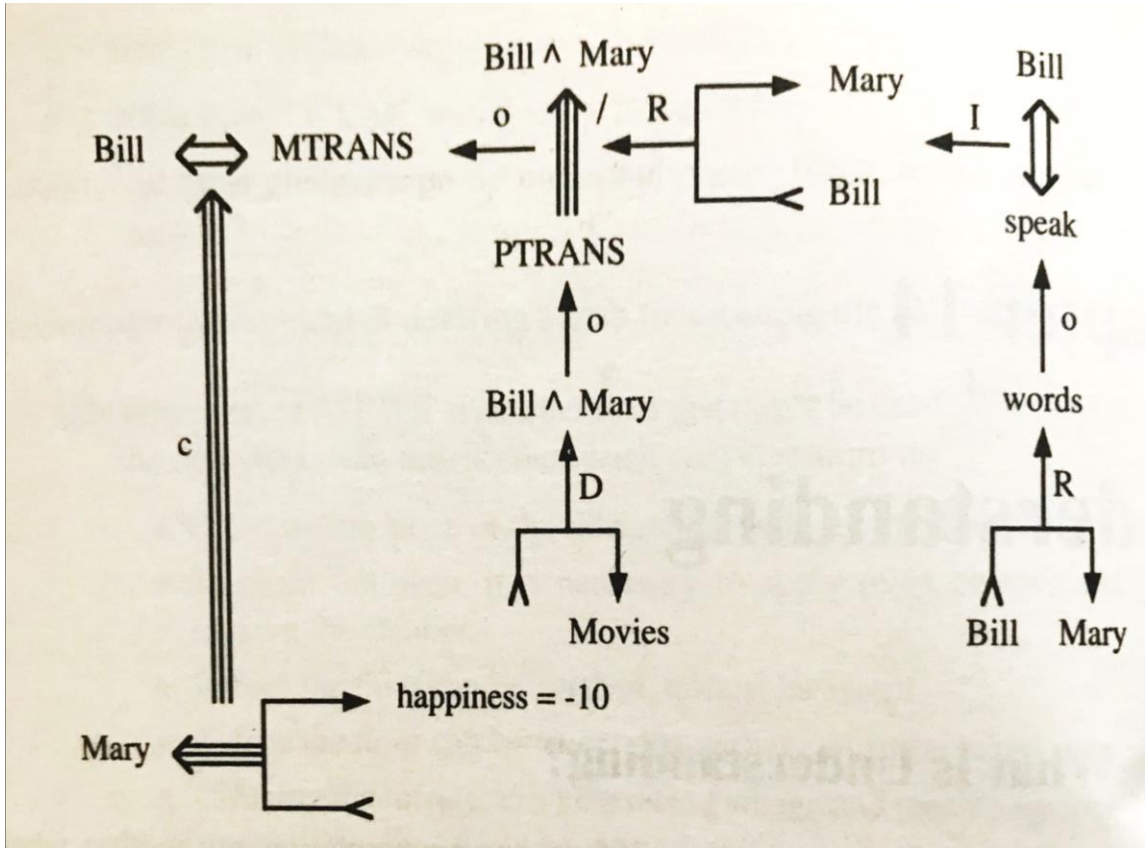
But now suppose that English sentences are being used to provide input to a program that records events so that it can answer a variety of questions about these events and their relationships. For example, consider the following story:

Bill told John he would not go to the movies with her.

His feelings were hurt.

The result of understanding his story could be represented using the conceptual dependency model.

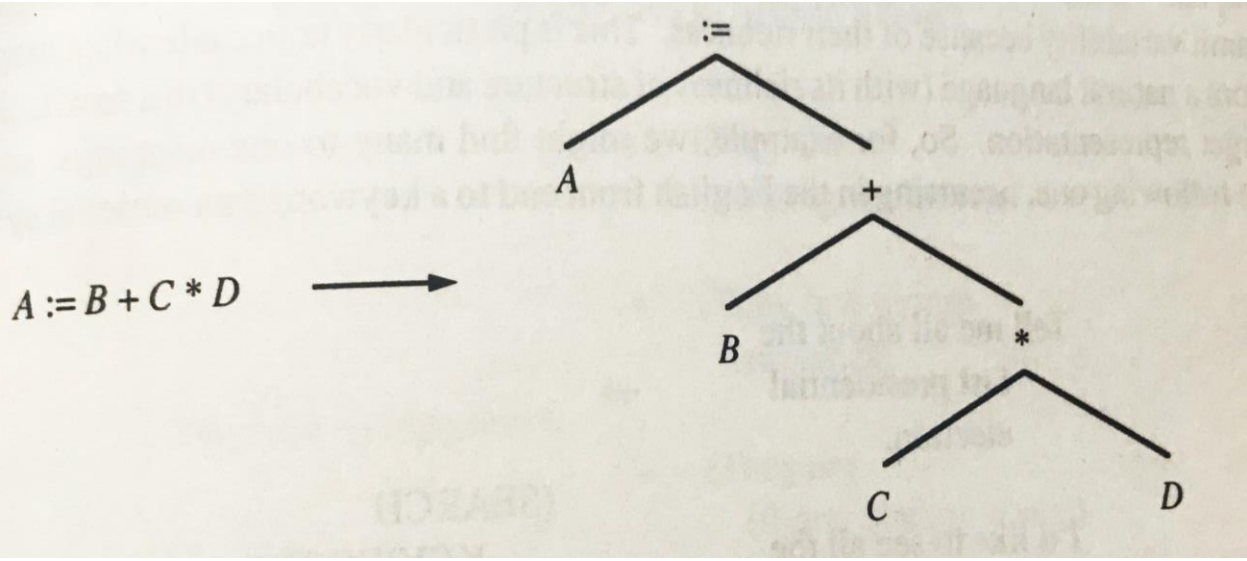
This representation is considerably more complex than that for the simple query.



TYPES OF MAPPINGS

Simplest kind of mapping is one-to-one.

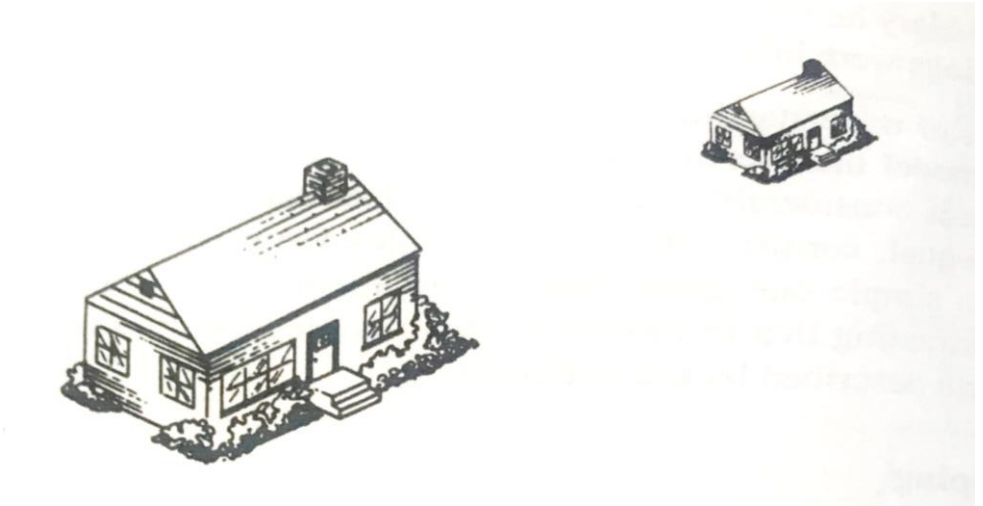
Example:



But very few input systems are totally one-one.

For example, when images are being interpreted, size and perspective will change as a function of the viewing position.

Thus a single object will look different in different images.

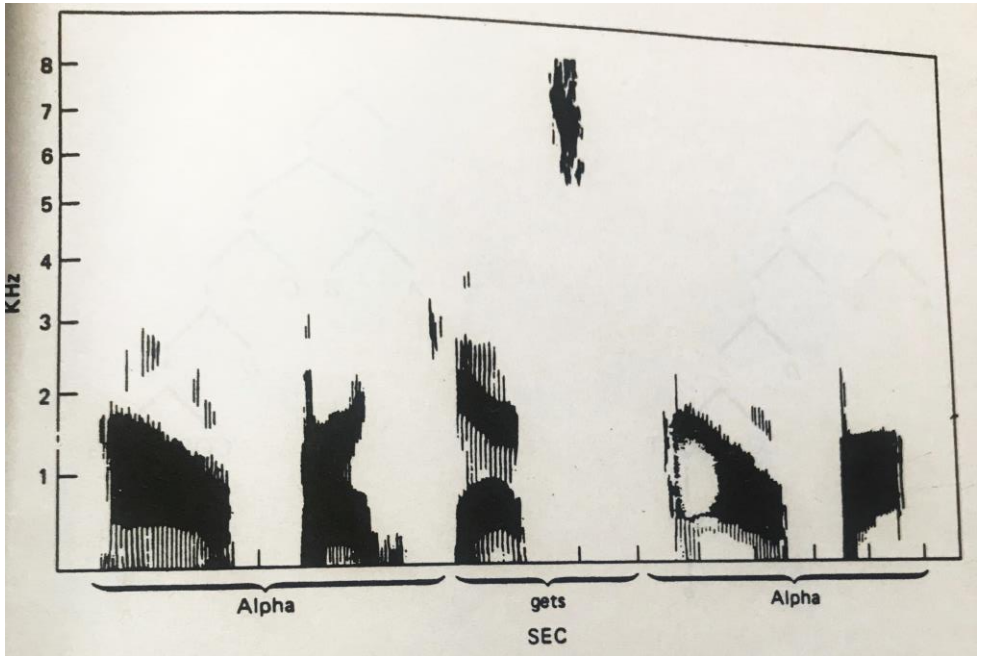


Another example:

No two people speak identically.

In fact, one person does not always say a given word the same way.

"Alpha gets alpha minus beta."



Natural languages admit variability because of their richness.

Example:

Tell me about the last general elections.

I would like to see all the stories on last general elections.

I am interested in the last general elections.

(SEARCH KEYWORDS = ELECTION & GENERAL)

Many-to-one mappings require that the understanding system know about all the ways that a target representation can be expressed in the source language.

As a result, they typically require a structured analysis of the input rather than simple, exact pattern match.

But they often do not require much other knowledge.

One-to-many mappings, on the other hand, often require a great deal of domain knowledge (in addition to input) in order to make the correct choice among the available target representations.

Example:

(They are (flying airplanes))

They are flying planes

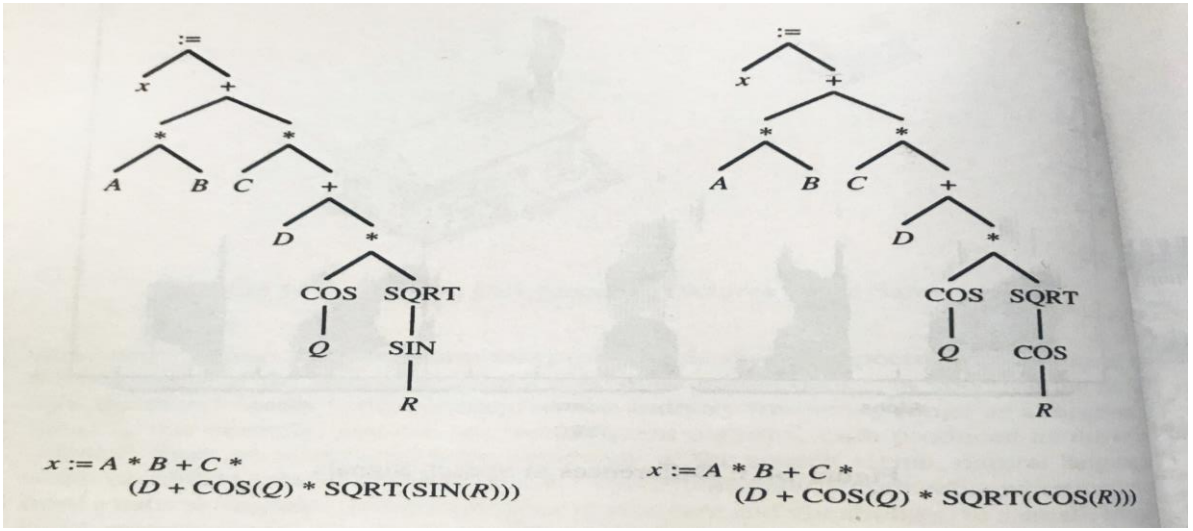
(They (are flying) airplanes))

LEVEL OF INTERACTION AMONG COMPONENTS

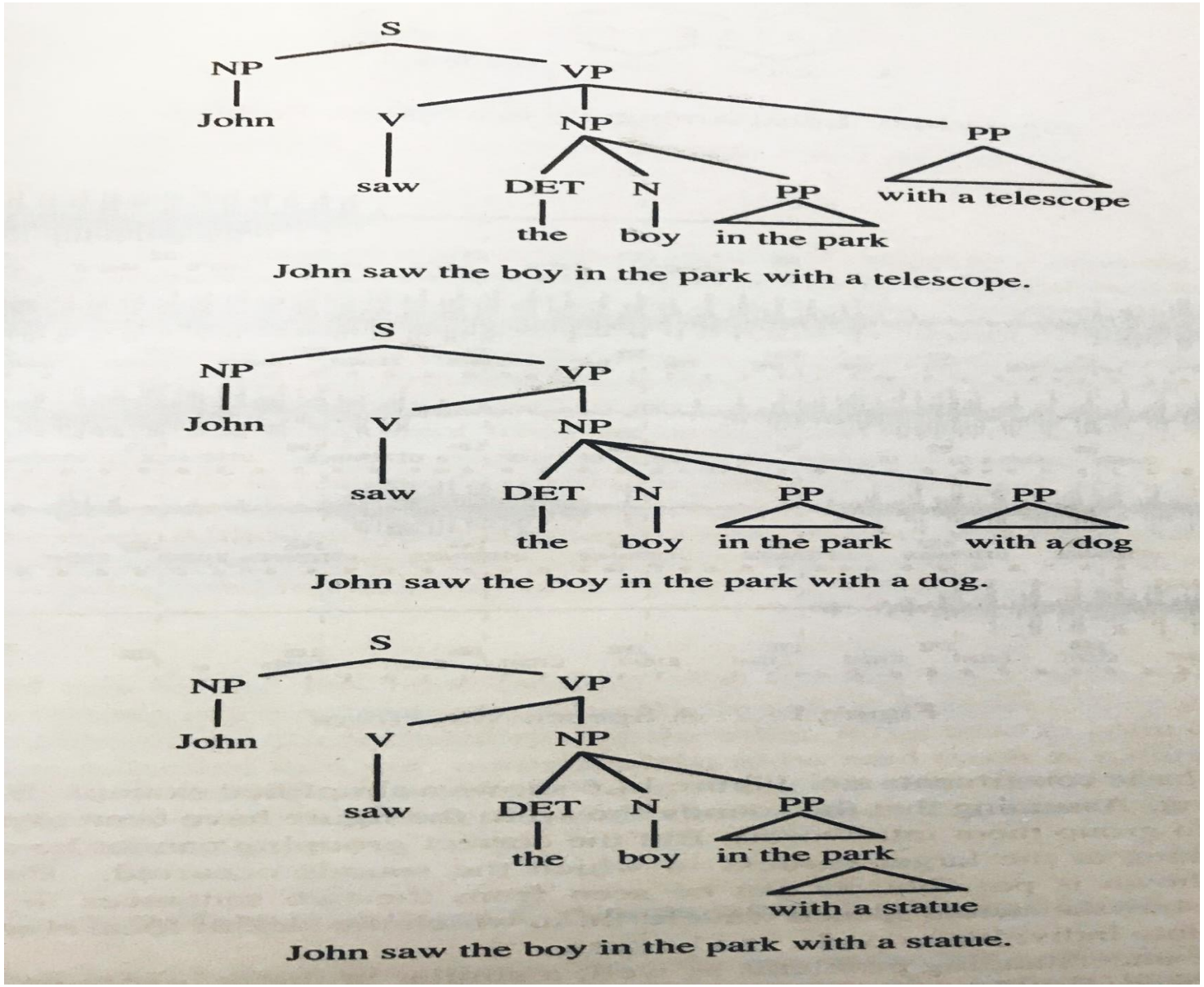
Each component is composed of several components (lines, words, symbols etc.).

Mapping process is the simplest if each component can be mapped without concern for other components of the statement.

Little interaction among components:



More Interaction Among Components



NOISE IN THE INPUT

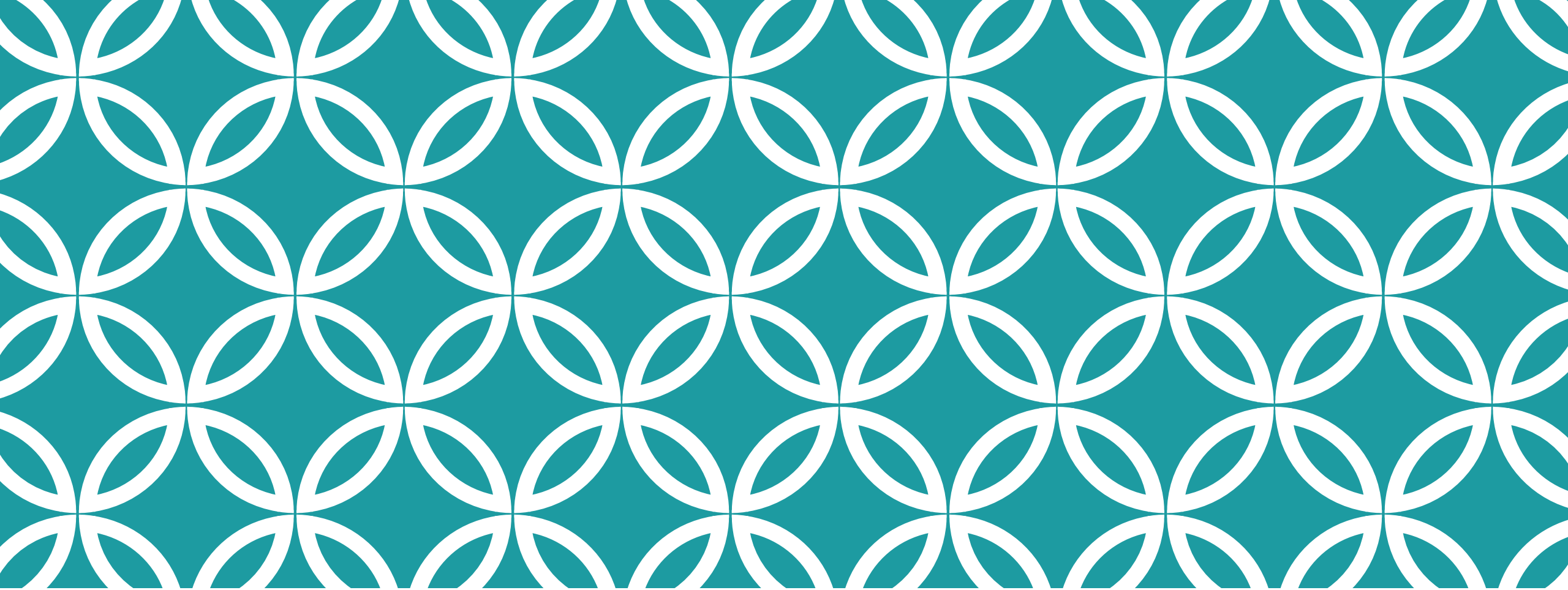
Unfortunately, in many understanding situations, the input to which the meaning should be assigned is not always the input that is presented to the understander.

Because of the complex environment in which understanding usually occurs, other things often interfere with the basic input before it reaches the understander.

In perceptual tasks such as speech and image understanding, this problem is common.

Although typed language is less susceptible to noise than is spoken language, noise is still a problem.

For example, typing errors are common, particulartly if language is being used interactively to communicate with a computer system.



LEARNING

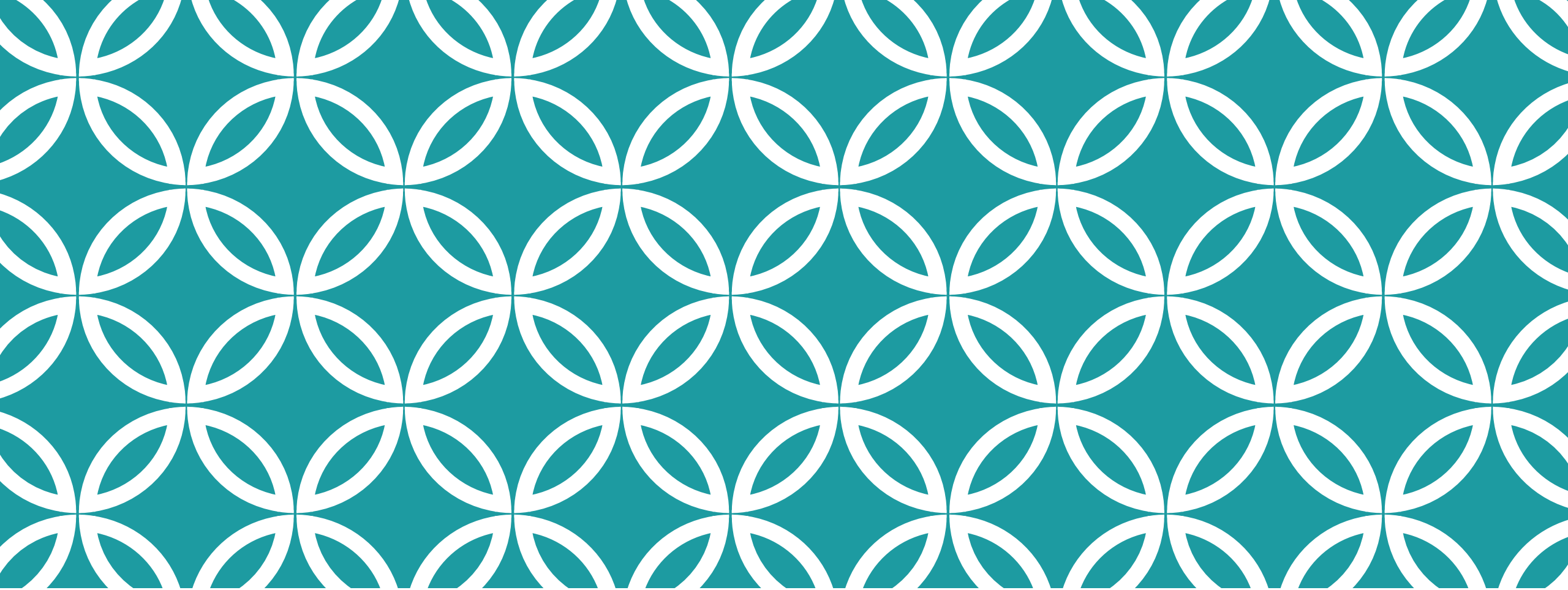
LEARNING

Learning is one of the fundamental building blocks of artificial intelligence (AI) solutions.

From a conceptual standpoint, learning is a process that improves the knowledge of an AI program by making observations about its environment.

From a technical/mathematical standpoint, AI learning processes focused on processing a collection of input-output pairs for a specific function and predicts the outputs for new inputs.

[Refer to Chapter 4]



COMMONSENSE

WHAT IS COMMONSENSE?

- Commonsense is the mental skills that most people share.
- Commonsense is ability to analyze a situation based on its context, using millions of integrated pieces of common knowledge.
- John McCarthy was the first to talk about commonsense reasoning in his paper in 1959, explains that a program has commonsense if it automatically deduces for itself sufficiently wide class of immediate consequences of any thing it is told and what it already knows.
- Commonsense is what people come to know in the process of growing and living in the world (R. Elio, 2002).
- Commonsense knowledge includes the basic facts about events and their effects, facts about knowledge and how it is obtained, facts about beliefs and desires. It includes the basic facts about material objects and their properties (John McCarthy, 1990).
- Currently, computers lack commonsense.

INTRODUCTION

Commonsense is ability to analyze a situation based on its context, using millions of integrated pieces of common knowledge.

Ability to use commonsense knowledge depends on being able to do commonsense reasoning. Commonsense Reasoning is a central part of intelligent behavior.

Formalizing the commonsense knowledge for even simple reasoning problem is a huge task. The reason is that, the most commonsense knowledge is implicit in contrast to expert/specialist knowledge, which is usually explicit. Therefore making commonsense reasoning system is making this knowledge explicit.

Example: Everyone knows that dropping a glass of water, the glass will break and water will spill on podium. However, this information is not obtained by formula or equation for a falling body or equations governing fluid flow.

The goal of the formal commonsense reasoning community is to encode this implicit knowledge using formal logic.

COMPUTERS AND ORDINARY REAL-LIFE ISSUES

Computers do many remarkable things.

Computer programs can play chess at the level of best players.

But no computer program match the capabilities of a three year old child at recognizing objects or can draw simple conclusions about ordinary life.

Building machines that can think the way any average person can is a distant reality.

WHY COMPUTERS CAN NOT THINK ABOUT THE WORLD AS ANY PERSON CAN?

Where the problem lies?

There are two basic types of knowledge.

- One is the specialist's knowledge which mathematicians, scientists and engineers possess.
- The other type is the commonsense knowledge which every one has, even a small 6-year–old child.

The need is to teach the computer to reason about the world (commonsense knowledge).

The researchers have not yet reached to any consensus on many related issues.

- McCarthy suggested to use logic to represent the knowledge.

Understanding commonsense capability is an active area of research in artificial intelligence.

COMMONSENSE KNOWLEDGE AND REASONING

Commonsense facts and methods are very little understood today.

Extending this understanding is the key problem the AI researchers are facing.

John McCarthy (1984) identified as commonsense as :

Commonsense knowledge - what every one knows.

Commonsense reasoning - ability to use commonsense knowledge.

COMMONSENSE KNOWLEDGE

What one can express as a fact using a richer ontology.

Examples:

Every person is younger than the person's mother

People do not like being repeatedly interrupted

If you hold a knife by its blade then the blade may cut you

If you drop paper into a flame then the paper will burn

You start getting hungry again a few hours after eating a meal

People go to parties to meet new people

People generally sleep at night

Here the problem is, how to give computers these millions of ordinary pieces of knowledge that every person learns by adulthood.

COMMONSENSE REASONING

What one builds as a reasoning method into his program.

Examples:

If you have a problem, think of a past situation where you solved a similar problem

If you take an action, anticipate what might happen next

If you fail at something, imagine how you might have done things differently

If you observe an event, try to infer what prior event might have caused it

If you see an object, wonder if anyone owns it

If someone does something, ask yourself what the person's purpose was in doing that.

Here the problem is, how to give computers the capacity for commonsense reasoning, the ways to use the commonsense knowledge to solve the various problems we encounter every day.

HOW TO TEACH COMMONSENSE TO A COMPUTER

There is no clear answer for to this question.

Presently, there is no program that can match the commonsense reasoning powers of a 5 year old child.

The problem was noticed long ago by John McCarthy. “We do not yet have enough ideas about how to represent, organize, and use much of commonsense knowledge, let alone build a machine that could learn automatically on its own".

- Some believe that, prior understanding is not necessary to build a machine, and intelligence can be made to emerge from some generic learning.
- Others feel that, unless we can acquire some experience in manually engineering systems with commonsense, we will not be able to build learning machines that can automatically learn commonsense.

BUILDING HUMAN COMMONSENSE KNOWLEDGE BASE

Stated below, the two ongoing AI projects for assembling comprehensive ontology and knowledge base of everyday commonsense knowledge with the goal of enabling AI applications to perform human-like reasoning.

Project CYC: A commonsense knowledge base, building since two decades, has collected 1.5 million pieces of commonsense knowledge, still far away from several hundred million required; the project faces a challenge because such a large database cannot be engineered by any one group.

Project Open Mind: A commonsense knowledge base, building since 1999, has accumulated more than 700,000 facts from over 15,000 contributors; the knowledge collected by has enabled many research projects at MIT and elsewhere.

The CYC and Open Mind projects are ensuring enough commonsense knowledge so as to work in a given environment enabling AI applications to perform human-like reasoning.

Next comes the commonsense reasoning :

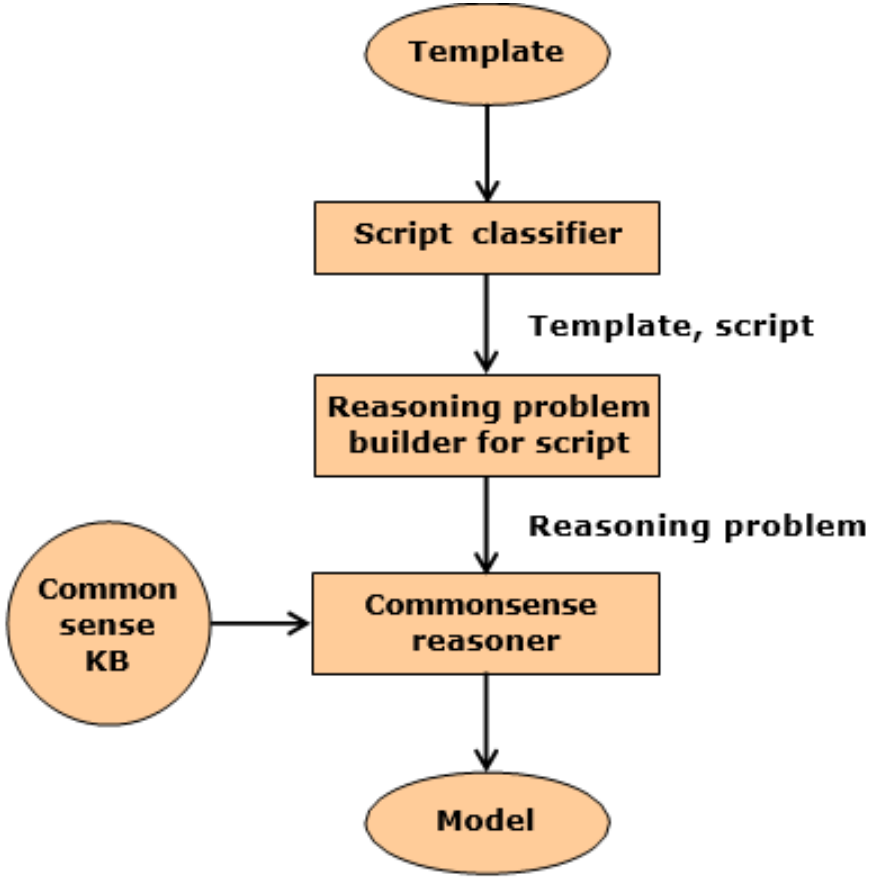
It is what one builds as a reasoning method into his program, a very complex task. We want computer to do reasoning as human does. Human does reasoning in different ways and the one which is logic reasoning (deductive, inductive, abductive), is of main concern in AI reasoning system. The logic reasoning can accomplish the task of commonsense reasoning. For instance:

- Predicate logic can represent knowledge about objects, facts, rules
- Frames can describe everyday objects
- Scripts can describe typical sequences of events
- Non-monotonic logics can support default reasoning

EXAMPLE OF COMMONSENSE SYSTEM ARCHITECTURE (MUELLER, 2004)

The system takes as input a template produced by information extraction system about certain aspects of a scenario.

- The template is a frame with slots and slots fillers
- The template is fed to a script classifier, which classifies what script is active in the template.
- The template and the script are passed to a reasoning problem builder specific to the script, which converts the template into a commonsense reasoning problem.
- The problem and a commonsense knowledge base are passed to a commonsense reasoner. It infers and fills in missing details to produce a model of the input text.
- The model provides a deeper representation of the input, than is provided by the template alone.



Commonsense System Architecture

FORMALIZATION OF COMMONSENSE REASONING

Commonsense reasoning is a central part of human behavior; no real intelligence is possible without it. The ultimate goal of artificially intelligent systems is that they exhibit commonsense behavior.

For the computers, the commonsense reasoning is not an easy task, indeed a very complex task, we all perform about every day world.

Example : There are chess-playing programs that beat champions, and there are expert systems that assist in clinical diagnosis, but there is no program that reason about how far one must bend over to put on one's socks.

The reason is expert knowledge is usually explicit, but most commonsense knowledge is implicit. Therefore, one of the prerequisites for developing commonsense reasoning systems is making this knowledge explicit.

John McCarthy (1990) identified commonsense reasoning as human ability to use commonsense knowledge.

Mueller (2006) defines commonsense reasoning as a process, taking information about certain aspects of a scenario in the world and making inference about other aspects of the scenario based on our commonsense knowledge or knowledge about how the world works.

To formalize commonsense reasoning, we need to construct representations for commonsense knowledge and inference algorithms to manipulate that knowledge.

McCarthy in 1959 was the first to put forward the idea of using a formal logic as the representation language for a commonsense reasoning system, with the reasoning done by deductive inference.

Robert C. Moore in his article, *"Automatic Deduction for Commonsense Reasoning: An Overview"*, explained the issues involved in drawing conclusions by means of deductive inference from bodies of commonsense knowledge represented by logical formulas.

This article contains first a review of initial attempts of late 60's and early 70's – failures and disappointments, and then the renewed attempts in late 70's and 80's to recent time - how domain-specific control information can offer a solution to the difficulties, the relationship of automatic deduction to the new field of "logic programming" and issues that arise while extending automatic-deduction techniques to nonstandard logic.

PHYSICAL WORLD

People know a great deal about how the physical world works. Most people, have no notion of the "laws of physics" that govern this world, yet they:

- can predict that a falling ball will bounce many times before come to halt.
- can predict the projection of cricket ball and even catch it.
- know a pendulum swings back and fore finally coming to rest in the middle.

How can we build a computer program to do such reasoning?

One answer is to program the equations governing the physical motion of the objects. But most people do not know these equations and also do not have exact numerical measures, yet they can predict what will happen in physical situations.

This means people seem to reason more abstractly that the equations would. Here comes qualitative physics, to understand how to build and reason with abstract, number less representation.

Researchers are therefore motivated towards :

- Modeling the qualitative world and
- Reasoning with qualitative information

MODELING THE QUALITATIVE WORLD

Qualitative physics seeks to understand physical processes by building models that may have following entities:

Variables	A restricted set of values, e.g. temperature as { frozen, between, boiling }.
Quantity Spaces	A small set of discreet values.
Rate of Change	Values at different times, modeled qualitatively, e.g. { decreasing, steady, increasing }.
Expressions	Combination of variables.
Equations	Assignment of expression to variables.
States	Sets of variables, whose values change over time.

EXAMPLE: QUALITATIVE ALGEBRA - ADDITION

Describe the volume of glass as **{empty, between, full}**.

When two qualitative values are added together then:

- empty + empty = empty
- empty + between = between
- empty + full = full
- between + between = { between, full, overflow }
- between + full = { between, over flow }
- full + full = { full, over flow }

REASONING WITH QUALITATIVE INFORMATION

Reasoning with qualitative information is often called qualitative simulation.

The basic idea is :

- Construct a sequence of discrete episodes that occur as qualitative variable.
- States are linked by qualitative rules that may be general.
- Rules may be applied to many objects simultaneously as they may all influence each other.
- Ambiguity may arise so split outcomes into different paths.
- A network of all possible states and transitions for a qualitative system is called an envisionment (mental images). There are often many paths through an envisionment. Each path is called history.
- Programs must know how to represent the behavior of many kinds of processes, materials and the world in which they act.

COMMONSENSE ONTOLOGIES

Some concepts are fundamental to commonsense reasoning.

A computer program that interacts with the real world must be able to reason about things like time, space and materials.

TIME

The most basic notion of time is events. Events occur during intervals over continuous spaces of time.

An interval has a start and end point and a duration between them.

Intervals can be related to one another as :

is-before, is-after, meets, is-met-by, starts, is-started-by, during, contains, ends, is-ended-by and equals.

We can build a axioms with intervals to describe events in time.

SPACE

Objects have a spatial extent while events have a temporal extent. We may try to extend of commonsense theory of time.

But, space is 3D and has many more relationships than those for time so it is not a good idea.

Another approach is view objects and space at various levels of abstraction.

For example, we can view most printed circuit boards as being a 2D object.

Choosing a representation means selecting relevant properties at particular levels of granularity.

For instance we can define relations over spaces such as inside, adjacent etc.

We can also define relations for curves, lines, surfaces, planes and volumes. e.g. along, across, perpendicular etc.

MATERIALS

Describe the properties of materials as :

You cannot walk on water.

If you knock a cup of coffee over what happens?

If you pour a full kettle into a cup what happens?

You can squeeze a sponge but not a brick.

The liquids provide many interesting points, such as, the space occupied by them. Thus we can define their properties such as:

Capacity - a bound to an amount of liquid.

Amount - volume occupied by a liquid.

Full - if amount equals capacity.

Other properties that materials can posses include:

Free - if a space is not wholly contained inside another object.

Surround - if enclosed by a very thin free space.

Rigid

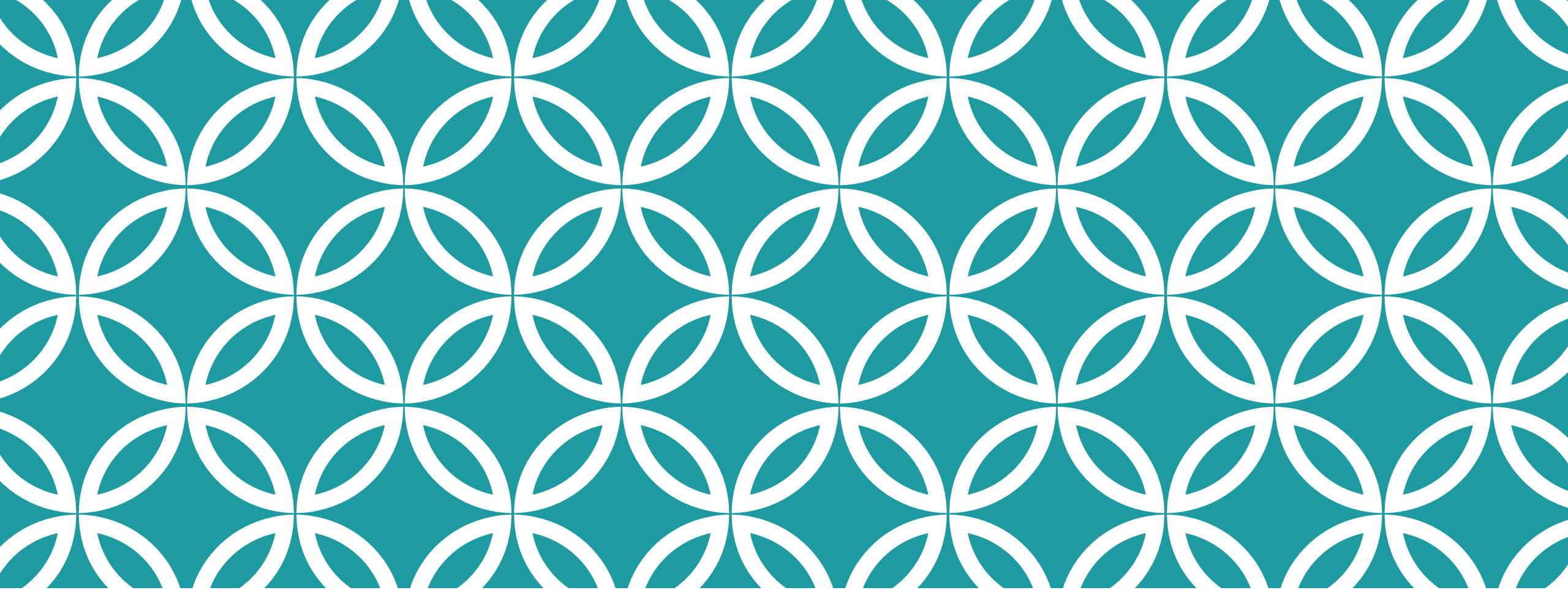
Flexible

Particulate - e.g. sand

MEMORY ORGANIZATION

Memory is central to commonsense behavior and also the basis for learning. Human memory is still not fully understood however psychologists have proposed several ideas.

- **Short term memory (STM):** Only a few items at a time can be held here; perceptual information are stored directly here.
- **Long term memory (LTM):** Capacity for storage is very large and fairly permanent; LTM is often divided further as :
  - Episodic memory: Contains information about personal experiences.
  - Semantic memory: General facts with no personal meaning, e.g. birds fly; useful in natural language understanding.



NATURAL LANGUAGE PROCESSING

WHAT IS NLP?

NLP is Natural Language Processing.

Natural languages are those spoken by people.

NLP encompasses anything a computer needs to understand natural language (typed or spoken) and also generate the natural language.

Natural Language Processing (NLP) is a subfield of Artificial intelligence and linguistics, devoted to make computers "understand" statements written in human languages.

NATURAL LANGUAGE

A natural language (or ordinary language) is a language that is spoken, written by humans for general-purpose communication.

Example: Punjabi, Hindi, English, French, and Chinese, etc.

A language is a system, a set of symbols and a set of rules (or grammar).

- The symbols are combined to convey new information.
- The rules govern the manipulation of symbols.

NATURAL LANGUAGE PROCESSING (NLP)

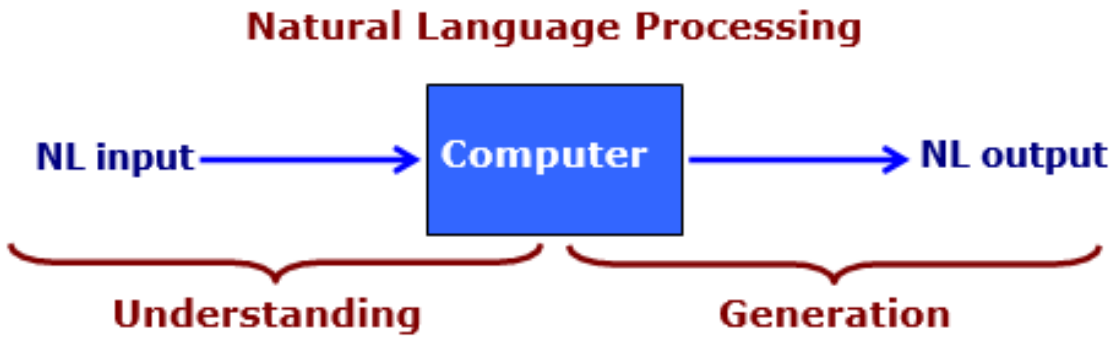
Natural Language Understanding (NLU):

The NLU task is understanding and reasoning while the input is a natural language.

Here we ignore the issues of natural language generation.

Natural Language Generation (NLG):

NLG is a subfield of natural language processing NLP. NLG is also referred to as text generation.



FORMAL LANGUAGE

Before defining formal language, we need to define symbols, alphabets, strings and words.

Symbol is a character, an abstract entity that has no meaning by itself. e.g., letters, digits and special characters.

Alphabet is finite set of symbols; an alphabet is often denoted by **Σ**(sigma).

For example, **B = {0, 1}** says **B** is an alphabet of two symbols, **0** and **1**.

C = {a, b, c} says **C** is an alphabet of three symbols, **a**, **b** and **c**.

String or a word is a finite sequence of symbols from an alphabet.

For example, **01110** and **111** are strings from the alphabet **B** above.

aaabccc and **b** are strings from the alphabet **C** above.

Language is a set of strings from an alphabet.

Formal language (or simply language) is a set **L** of strings over some finite alphabet **Σ**.

Formal language is described using formal grammars.

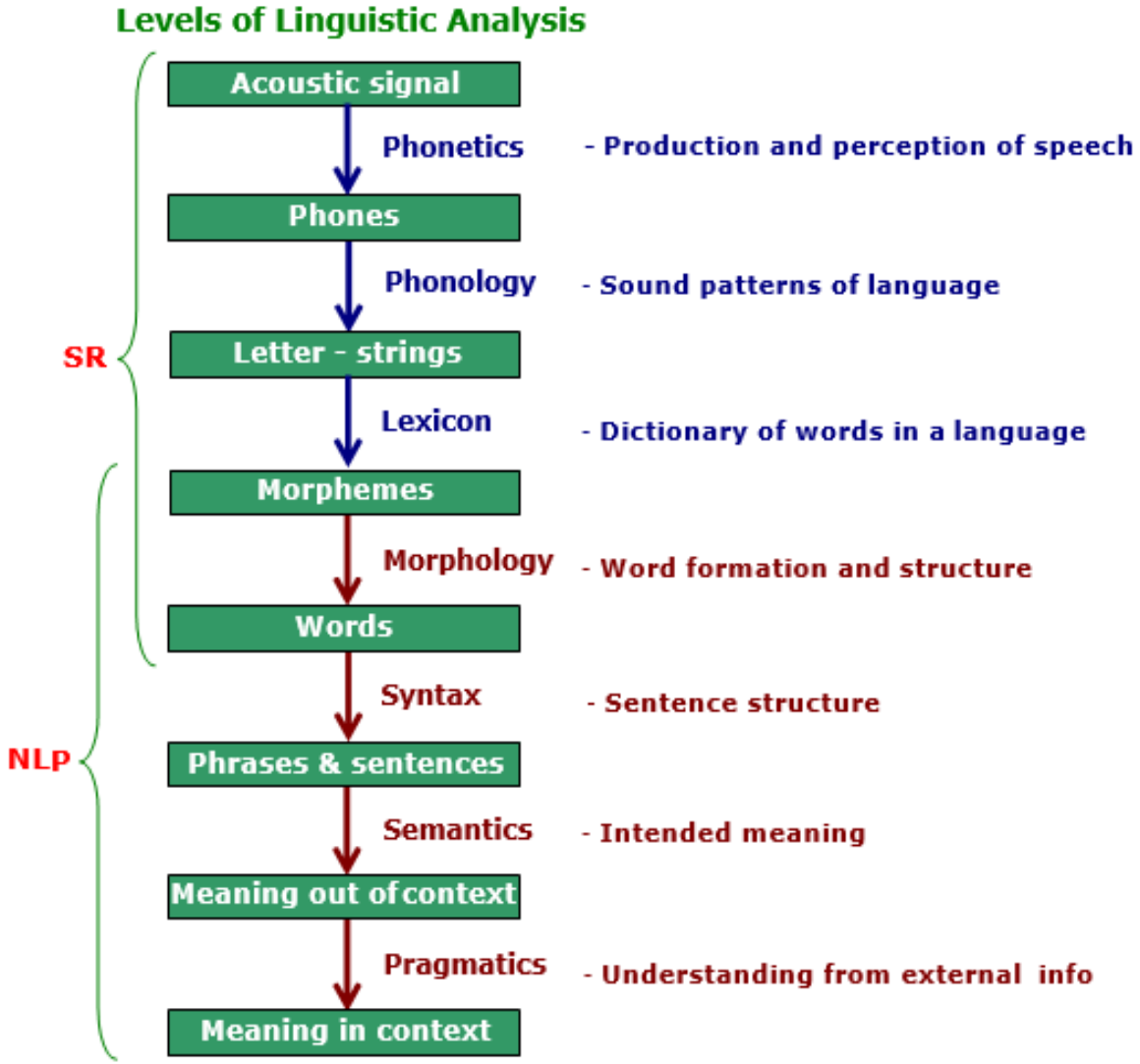
LINGUISTICS AND LANGUAGE PROCESSING

Linguistics is the science of language. study includes:

- sounds (phonology),
- word formation (morphology),
- sentence structure (syntax),
- meaning (semantics), and understar (pragmatics) etc.

The levels of linguistic analysis are:

- higher level corresponds to Speech Recognition (SR)
- lower levels corresponds to Natura Language Processing (NLP).



STEPS IN NATURAL LANGUAGE PROCESSING (NLP)

Natural Language Processing is done at 5 levels, as shown in the previous slide. These levels are briefly stated below.

- **Morphological and Lexical Analysis:**

- The lexicon of a language is its vocabulary, that include its words and expressions.
- Morphology is the identification, analysis and description of structure of words.
- The words are generally accepted as being the smallest units of syntax.
- The syntax refers to the rules and principles that govern the sentence structure of any individual language.

Lexical analysis: The aim is to divide the text into paragraphs, sentences and words.

- The lexical analysis can not be performed in isolation from morphological and syntactic analysis.

Syntactic Analysis:

- Here the analysis is of words in a sentence to know the grammatical structure of the sentence.
- The words are transformed into structures that show how the words relate to each others.
- Some word sequences may be rejected if they violate the rules of the language for how words may be combined.
- Example: An English syntactic analyzer would reject the sentence say:
  - "Boy the go the to store."

■ Semantic Analysis:

- It derives an absolute (dictionary definition) meaning from context; it determines the possible meanings of a sentence in a context.
- The structures created by the syntactic analyzer are assigned meaning.
- Thus, a mapping is made between the syntactic structures and objects in the task domain.
- The structures for which no such mapping is possible are rejected.
- Example: the sentence **"Colorless green ideas . . ."** would be rejected as semantically anomalous because colorless and green make no sense.

■ Discourse Integration:

- The meaning of an individual sentence may depend on the sentences that precede it and may influence the meaning of the sentences that follow it.
- Example : the word "it" in the sentence, "you wanted it" depends on the prior discourse context.

■ Pragmatic analysis:

- It derives knowledge from external commonsense information; it means understanding the purposeful use of language in situations, particularly those aspects of language which require world knowledge.
- The idea is, what was said is reinterpreted to determine what was actually meant. Example: the sentence "Do you know what time it is?" should be interpreted as a request.

DEFINING TERMS RELATED TO LINGUISTIC ANALYSIS

The following terms are explained in next few slides.

Phones, Phonetics, Phonology, Strings, Lexicon, Words, Determiner, Morphology, Morphemes, Syntax, Semantics, Pragmatics, Phrase, and Sentence.

- **Phones:** The phones are acoustic patterns that are significant and distinguishable in some human language.
  - Example: In English, the L-sounds at the beginning and end of the word "loyal", are termed "light L" and "dark L" by linguists.
- **Phonetics:** Tells how acoustic signals are classified into phones.
- **Phonology:** Tells how phones are grouped together to form phoneme in particular human languages.

- **Strings:** An alphabet is a finite set of symbols.
  - Example : English alphabets **{ a, b, c, d, e, f, g, h, i, j, k, l, m, n, o, p, q, r, s, t, u, v, w, x, y, z }**
  - A string is a sequence of symbols taken from an alphabet.

- **Lexicon:** Lexicon is collection of information about words of a language.
  - The information is about the lexical categories to which words belong.
  - Example: "**pig**" is usually a noun (**N**), but also occurs as a verb(**V**) and an adjective(**ADJ**).
  - Lexicon structure: a collection of lexical entries. Example: **("pig" N, V, ADJ)**

- **Words:** Word is a unit of language that carries meaning.
  - Example: Words like bear, car, house are very different from words like run, sleep, think, and are different from words like in, under, about.
  - These and other categories of words have names: nouns, verbs, prepositions, and so on.
  - Words build phrases, which in turn build sentences.
- **Determiners:** Determiners occur before nouns and indicate the kind of reference which the noun has.
  - Example below shows determiners marked by "bold letters".
  - **the** boy, **a** bus, **our** car, **these** children, **both** hospitals

- **Morphology:** Morphology is the analysis of words into morphemes, and conversely the synthesis of words from morphemes.
- **Morphemes:** A smallest meaningful unit in the grammar of a language.
  - A smallest linguistic unit that has semantic meaning.
  - A unit of language immediately below the 'word level'.
  - A smallest part of a word that can carry a discrete meaning.
  - Example: the word "unbreakable" has 3 morphemes:
    - Un - a bound morpheme
    - Break - a free morpheme
    - Able - a bound morpheme
  - Also "un-" is also a prefix; "-able" is a suffix; both are affixes.
  - Morphemes are of many types, stated in the next slide.

- **Types of morphemes:**
  - **Free Morphemes:** Can appear stand alone, or “free”.
    - Example: **“town”, “dog”, “town hall”, “dog house”.**
  - **Bound Morphemes:** Appear only together with other morphemes to form a lexeme.
    - Example : “**un-**” ; in general it tends to be prefix and suffix.
  - **Inflectional Morphemes:** Modify a word's tense, number, aspect, etc.
    - Example : **dog** morpheme with plural marker morpheme **s** becomes **dogs**.
  - **Derivational Morphemes:** Can be added to a word to derive another word.
    - Example : addition of “**-ness**” to “**happy**” gives “**happiness**”.

- **Root Morpheme:** It is the primary lexical unit of a word; roots can be either free or bound morphemes; sometimes "root" is used to describe word minus its inflectional endings, but with its lexical endings.
  - Example: word **chatters** has the inflectional root or lemma **chatter**, but the lexical root **chat**.
  - Inflectional roots are often called stems, and a root in the stricter sense may be thought of as a mono-morphemic stem.
- **Null Morpheme:** It is an "invisible" affix, also called zero morpheme represented as either the figure zero (**0**), or the empty set symbol **Ø**.
  - Adding a null morpheme is called null affixation, null derivation or zero derivation; null morpheme that contrasts singular morpheme with the plural morpheme.
  - For example,
    - **cat = cat + -0 = ROOT("cat") + SINGULAR**
    - **cats = cat + -s = ROOT("cat") + PLURAL**

Syntax:

- Syntax is the structure of language.
- It is the grammatical arrangement of words in a sentence to show its relationship to one another in a sentence.
- Syntax is finite set of rules that specifies a language.
- Syntax rules govern proper sentence structure.
- Syntax is represented by parse tree, a way to show the structure of a language fragment, or by a list.

- **Semantics:**
  - Semantic is meaning of words/phrases/sentences/whole texts.
  - Normally semantic is restricted to "meaning out of context" - that is, meaning as it can be determined without taking context into account.
- **Pragmatics**
  - Pragmatics tell how language is used; that is "meaning in context".
  - Example: if someone says "**the door is open**" then it is necessary to know which door "**the door**" refers to;
    - Need to know what the intention of the speaker: could be a pure statement of fact,
    - could be an explanation of how the cat got in, or
    - could be a request to the person addressed to close the door.

GRAMMATICAL STRUCTURE OF UTTERANCES

Here sentence, constituent, phrase, classification and structural rule are explained.

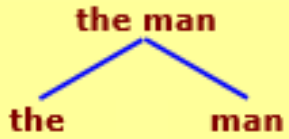
- **Sentence**
  - Sentence is a string of words satisfying grammatical rules of a language.
  - Sentences are classified as simple, compound, and complex.
  - Sentence is often abbreviated to "**S**".
  - **Sentence(S): "The dog bites the cat".**
- **Constituents**
  - Assume that a phrase is a construction of some kind.
  - Here construction means a syntactic arrangement that consists of parts, usually two, called "constituents".
  - Example: The phrase "the man" is a construction consisting of 2 constituents "the" and "man".

A FEW EXAMPLES OF CONSTITUENTS

Phrase : the man

Constituents : the and man

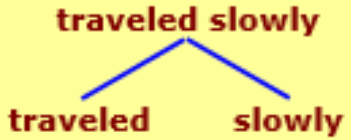
Construction :



Phrase : traveled slowly

Constituents : traveled and slowly.

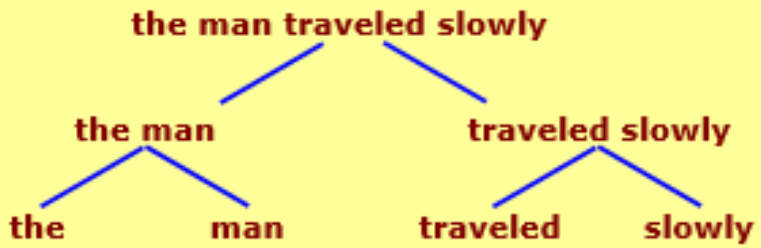
Construction :



Phrase : the man traveled slowly

Constituents four : the , man , traveled , slowly

Construction :



PHRASE

- A phrase is a group of words (minimum is two) that functions as a single unit in the syntax of a sentence.
  - For example, "**the house at the end of the street**" is a phrase, acts like noun.
  - "**end of the street**" is a phrase, acts like adjective;
- How phrases are formed is governed by phrase structure rules.
- Most phrases have a head or central word, which defines the type of phrase. Head is often the first word of the phrase. Some phrases, can be headless.
  - For example, "**the rich**" is a noun phrase composed of a determiner and an adjective, but no noun.
- Phrases may be classified by the type of head they take.

CLASSIFICATION OF PHRASES: NAMES (ABBREVIATION)

The most accepted classifications for phrases are stated below.

- **Sentence (S):** often abbreviated to "**S**".
- **Noun phrase (NP):** noun or pronoun as head, or optionally accompanied by a set of modifiers; the possible modifiers include: determiners: articles (the, a) or adjectives (the red ball) etc.; example: "**the** black cat", "**a** cat on the mat".
- **Verb phrase (VP):** verb as head, example: "**eat** cheese", "**jump** up and down".
- **Adjectival phrase (AP):** adjective as head, example: "**full** of toys".
- **Adverbial phrase (AdvP):** adverb as head, example: "**very carefully**".
- **Prepositional phrase (PP):** preposition as head, example : "**on** the rooftop", "**over** the rainbow".
- **Determiner phrase (DP):** determiner as head example: "**a** little dog", "**the** little dogs".
- In English, determiners are usually placed before the noun as a noun modifier that includes: articles (the, a), demonstratives (this, that), numerals (two, five, etc.), possessives (my, their, etc.), and quantifiers (some, many, etc.).

PHRASE STRUCTURE RULES

Phrase-structure rules are a way to describe language syntax.

Rules determine what goes into phrase and how its constituents are ordered.

They are used to break a sentence down to its constituent parts namely phrasal categories and lexical categories.

- Phrasal category includes: noun phrase, verb phrase, prepositional phrase;
- Lexical category includes: noun, verb, adjective, adverb, others.

Phrase structure rules are usually of the form **A → B C**,

- Meaning "constituent **A** is separated into two sub-constituents **B** and **C**" or simply "**A** consists of **B** followed by **C**".

Examples:

- **S → NP VP** reads: **S** consists of an **NP** followed by a **VP**; means a sentence consists of a noun phrase followed by a verb phrase.
- **NP → Det N1** reads: **NP** consists of an **Det** followed by a **N1**; means a noun phrase consists of a determiner followed by a noun.

PHRASE STRUCTURE RULES AND TREES FOR NOUN PHRASE (NP)

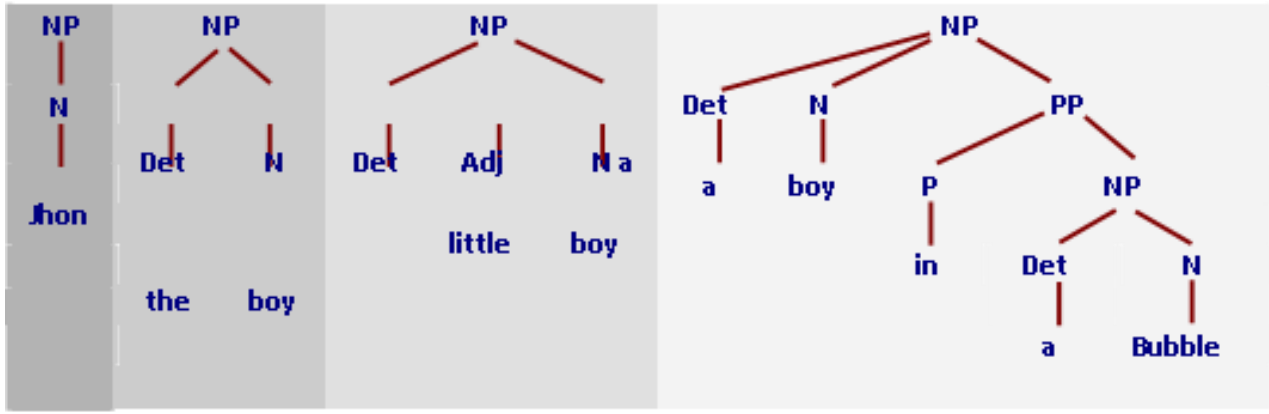
Noun Phrase (NP)

John	N
the boy	Det N
A little boy	Det Adj N
A boy in a bubble	Det N PP

Phrase Structure rules for NPs:

NP → (Det) (Adj) N (PP)

Phrase structure trees for NPs:



SYNTACTIC PROCESSING

Syntactic Processing converts a flat input sentence into a hierarchical structure that corresponds to the units of meaning in the sentence.

The Syntactic processing has two main components :

- one is called grammar, and
- other is called parser.

Grammar:

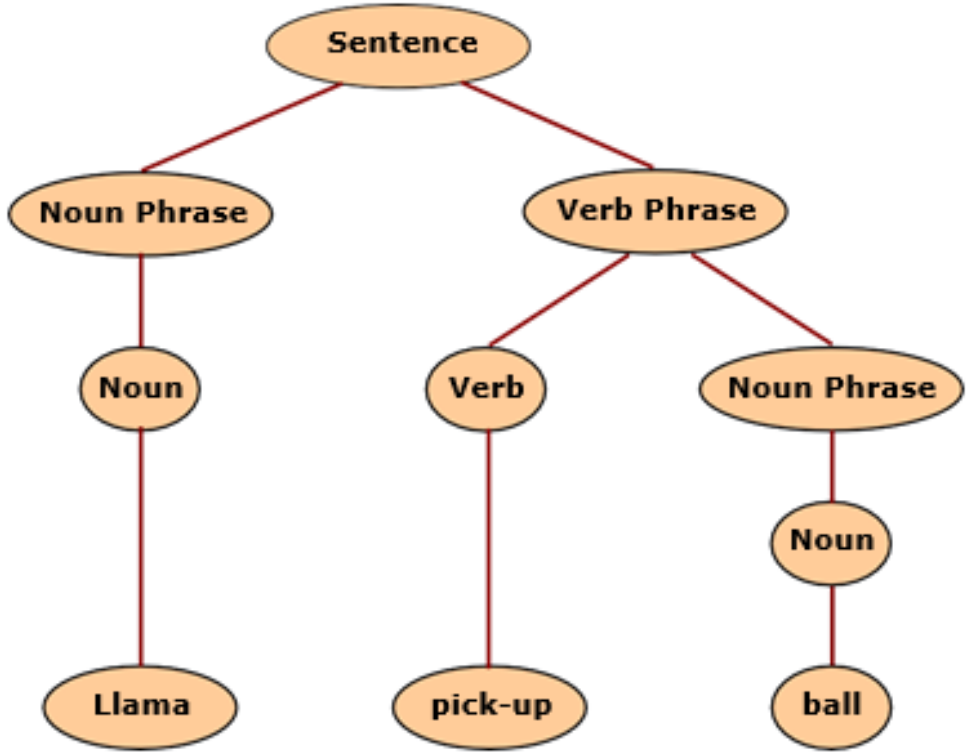
- It is a declarative representation of syntactic facts about the language. It is the specification of the legal structures of a language.
- It has three basic components : terminal symbols, non-terminal symbols, and rules (productions) .

Parser:

- It is a procedure that compares the grammar against input sentences to produce a parsed structures called **parse tree**.

PARSE TREE STRCUTURE

Example: Sentence “**Llama pickup ball**”.



CONTEXT FREE GRAMMAR (CFG)

In formal language theory, a context free grammar is a grammar where every production rules is of the form: **A** → **α** where **A** is a single symbol called **non-terminal**, and **α** is a **string** that is a sequence of symbols of terminals and/or non-terminals (possibly empty).

Terminal , Non-Terminal and Start Symbols

The terminal and non-terminal symbols are those symbols that are used to construct production rules in a formal grammar.

Terminal Symbol

Any symbol used in the grammar which does not appear on the left-hand-side of some rule (i.e. has no definition) is called a terminal symbol. Terminal symbols cannot be broken down into smaller units without losing their literal meaning.

Non-Terminal Symbol

Symbols that are defined by rules are called non-terminal symbols. Each production rule defines the non-terminal symbol. For example, the above rule states that "whenever we see an A, we can replace it with a".

A non-terminal may have more than one definition, in that case we use symbol "|" as the union operator.

Example 1: A → a | b states that "whenever we see A, we can replace it with a or with b".

Similarly, if a rule is **NP → Det N | Prop** then the vertical slash on the right side is a convention used to represent that the **NP** can be replaced either by **Det N** or by **Prop**. Thus, this is really two rules.

Example 2: **S → NP VP** states that the symbol **S** is replaced by the symbols **NP** and **VP**.

One special non-terminal is called **start symbol**, usually written **S**. The production rules for this symbol are usually written first in a grammar.

HOW GRAMMAR WORKS?

Grammar starts with the start symbol, then successively applies the production rules (replacing the L.H.S. with the R.H.S.) until reaches to a word which contains no non-terminals. This is known as a **derivation**.

Anything which can be derived from the start symbol by applying the production rules is called a **sentential form**.

Any grammar may have an infinite number of sentences.

The set of all such sentences is the language defined by that grammar.

Example of grammar :

S → X c **X → Y X** **Y → a | b**

The above grammar shows that it can derive all words which start arbitrarily and have many **a**'s or **b**'s and finish with a '**c**'.

This language is defined by the regular expression **(a | b) * c**.

The "*" indicates that the character immediately to its left may be repeated any number of times, including zero.

Thus **ab*c** would match "**ac**", "**abc**", "**abbc**", "**abbbc**", "**abbbbbbbbc**", and any string that starts with an "**a**", is followed by a sequence of **b**'s, and ends with a "**c**".

REGULAR EXPRESSION

Every regular expression can be converted to a grammar, but not every grammar can be converted back to a regular expression.

Any grammar which can be converted back to a regular expression is called a regular grammar; the language it defines is a regular language.

Regular Expression → Grammar

Regular Expression ← Regular Grammar

REGULAR GRAMMARS

A regular grammar is a grammar where all of the production rules are of one of the following forms:

$A \rightarrow \mathbf{a}\ B$ or $A \rightarrow \mathbf{a}$

where **A** and **B** represent any single non-terminal, and

a represents any single terminal, or the empty string.

PARSER

A parser is a program, that accepts as input a sequence of words in a natural language and breaks them up into parts (nouns, verbs, and their attributes), to be managed by other programming.

- Parsing can be defined as the act of analyzing the grammaticality an utterance according to some specific grammar.
- Parsing is the process to check, that a particular sequence of words in a sentence correspond to a language defined by its grammar.
- Parsing means show how we can get from the start symbol of the grammar to the sequence of words using the production rules.
- The output of a parser is *a* ***parse tree***.

PARSE TREE

Parse tree is a way of representing the output of a parser.

- Each phrasal constituent found during parsing becomes a branch node of the parse tree;
- the words of the sentence become the leaves of the parse tree;

There can be more than one parse tree for a single sentence.

PARSING

To parse a sentence, it is necessary to find a way in which the sentence could have been generated from the start symbol. There two ways to do: top-down parsing and bottom–up parsing.

- **Top-Down Parsing**
  - Begin with the start symbol and apply the grammar rules forward until the symbols at the terminals of the tree corresponds to the components of the sentence being parsed.
- **Bottom–Up Parsing**
  - Begin with the sentence to be parsed and apply the grammar rules backward until a single tree whose terminals are the wards of the sentence and whose top node is the start symbol has been produced.

MODELING A SENTENCE USING PHASE STRUCTURE

Every sentence consists of an internal structure which could be modeled with the phrase structure.

Algorithm:

- Apply rules on an proposition
- The base proposition would be:
  - S (the root, i.e. the sentence)
- The first production rule would be:
  - (NP = noun phrase, VP = verb phrase)
  - S → (NP, VP)
- Apply rules for the 'branches'
  - NP → noun VP → verb, NP
- The verb and noun have terminal nodes which could be any word in the lexicon for the appropriate category.
- The end is a tree with the words as terminal nodes, which is referred as the sentence.

EXAMPLE: PARSE TREE

Sentence: "He ate the pizza"

Apply the grammar with rules:

S ⟶ NP VP

NP ⟶ PRO

NP ⟶ ART N

VP ⟶ V NP

The lexicon structure is:

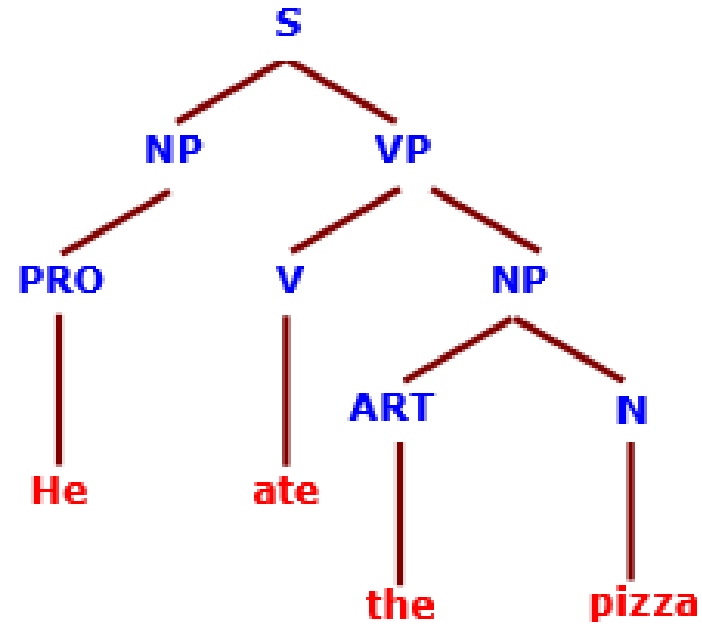
("ate" V)

("he" PRO)

("pizza" N)

("the" ART)

The parse tree is:



PRO, V, ART, N - lexical non-terminals

S, NP, VP - phrasal non-terminals

He, ate, the, pizza - words or terminals

SEMANTICS AND PRAGMATICS

The semantics and pragmatics, are the two stages of analysis concerned with getting at the meaning of a sentence.

- In the first stage (semantics), a partial representation of the meaning is obtained based on the possible syntactic structure(s) of the sentence and the meanings of the words in that sentence.
- In the second stage (pragmatics), the meaning is elaborated based on the contextual and the world knowledge.

For the difference between these stages, consider the sentence:

“He asked for the boss”.

From knowledge of the meaning of the words and the structure of the sentence we can work out that:

Someone (who is male) asked for someone who is a boss.

We can't say who these people are and why the first guy wanted the second.

If we know something about the context (including the last few sentences spoken/written), we may be able to work these things out.

Maybe the last sentence was **“Fred had just been sacked”.**

From our general knowledge that bosses generally sack people: if people want to speak to people who sack them it is generally to complain about it.

We could then really start to get at the meaning of the sentence :

“Fred wants to complain to his boss about getting sacked”.

REFERENCES

- *"Artificial Intelligence", by Elaine Rich and Kevin Knight, (2006), McGraw Hill companies Inc.*
- *"AI: A New Synthesis", by Nils J. Nilsson, (1998), Morgan Kaufmann Inc.*
- *"Artificial Intelligence: A Modern Approach" by Stuart Russell and Peter Norvig, (2002), Prentice Hall.*
- *"Artificial Intelligence: Structures and Strategies for Complex Problem Solving", by George F. Luger, (2002), Addison-Wesley.*
- *"Artificial Intelligence: Theory and Practice", by Thomas Dean, (1994), Addison-Wesley.*