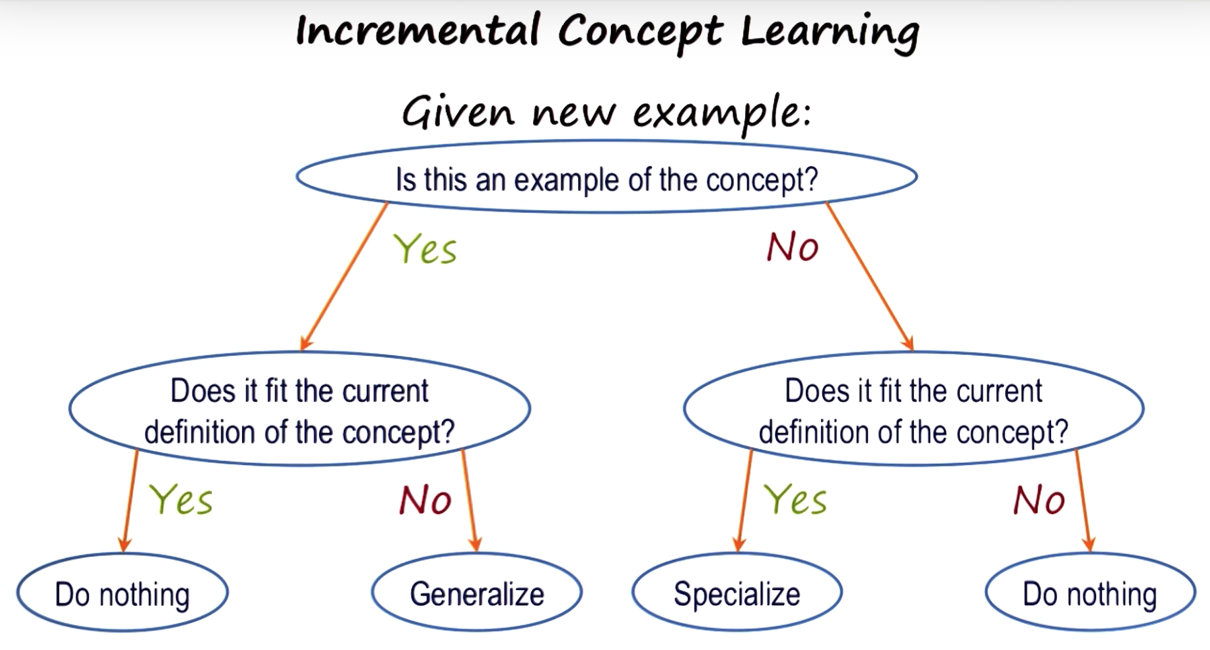
# Lesson 10: Incremental concept learning

**Quiz examples: Identifying a foo.**

Learning is often incremental.

1. We learn from one example at a time.
2. Often the examples are labeled as negative and positive example, this is called supervised learning.
3. The examples can come in a particular order. First one is typically a +ve.
4. This is different from case-based learning where we stored examples in their raw form in memory and reused them. Whereas here we abstract concepts from there.
5. The number of examples from which we are abstracting concept is very small.
6. While abstracting concepts, there is a tendency to often overgeneralize or overspecialize. Generalization and specialization are unknown to agent.

**Incremental Concept Learning:**

**Example 1:** Cats are black. Sees an orange cat. Is this example of cat? Yes. So, goes to left branch (Yes). Does it fit current definition (Cats are black)? No. So, **generalize** the concept that cats can be of any color.

**Example 2:** Dogs have fur, four legs and keeps as pet. Sees an orange cat. Is this an example of a dog? No. So, goes to the right branch (No). Does it fit current definition (Dogs fur, 4 legs, pet)? Yes. So, **specialize** the concept of dog to exclude this cat.

**Variabilization:**

With a single example, the AI program can only variabilize. Example of arch: 1st example is +ve. So, any brick can be placed in the spaces as long as relationships hold true. 2nd example is also +ve. Hence, AI program will generalize.

**Generalization to ignore features:** When there are 1 or 2 examples, learning space is potentially very large. Heuristics are applied to navigate through learning space. **Drop link heuristic** is used to generalize.

**Specialization to require features:** The 3rd example is -ve. AI program must refine its current definition in order to rule out -ve example. **Require-link heuristic**. Things not in common will become must be required.

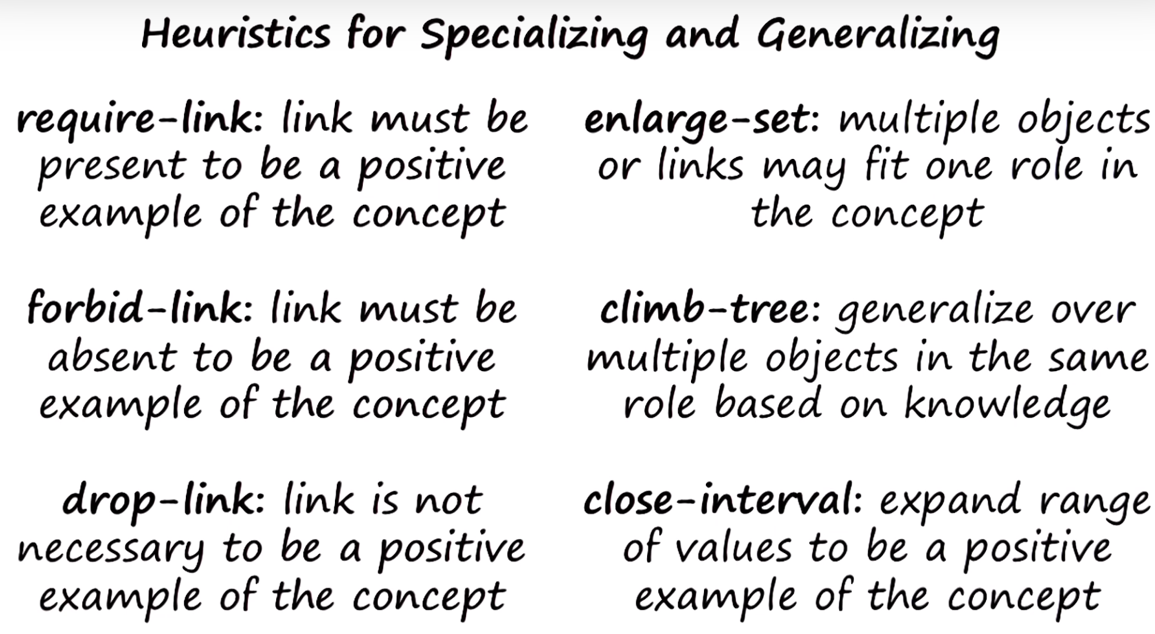
**Specialization to exclude features:** The 4th example is -ve. AI program must refine its current definition in order to rule out -ve example. **Forbid link heuristic** where some features are forbidden.

**Generalization to abstract features:** The 5th example is +ve. New example has wedge instead of brick, so include it with brick or wedge. This is called **enlarge set heuristic**.

**Generalization with background knowledge**: Brick and Wedge both are Block. So, replace brick or wedge with Block. This is called **climb tree heuristic**.

**An Alternate Visualization:**

Given a +ve example, AI agent comes up with a concept definition. When given a -ve example which is covered by current concept definition, it must be refined in such a way to exclude -ve example while still including the +ve example. Similarly when given a new +ve example, the definition should refine to include it.

**Heuristics for Concept Learning:**

**Final Concept of Foo:**

The final concept depends on both input examples and the background knowledge of the AI agent. When given large number of examples to begin with, then one can use statistical machine learning methods to find patterns of regularity in the input data. But if given a small number of examples and examples come one at a time, learning is incremental. In this case, the algorithm must use background knowledge to decide what to learn and how to learn it.

# Lesson 11: Classification

Classification is mapping sets of percepts in the world into equivalence classes, so that we can take actions in an effective manner. It is a type of Incremental concept Learning.

Example: Classifying Birds.

**Challenge of classification:**

**2^n percepts -> 2^m actions**

The number of combinations of percepts and actions are very large. And we have to map these percepts, combination of percepts, actions and combination of actions. This is a very complex mapping.

How can intelligent agent map percepts into actions? the intelligence is a lot of action selection.

**Equivalence Classes:**

k concept is equivalence classes.

**2^n percepts -> k concepts -> 2^m actions**

Instead of indexing my actions on the combinations of percepts, I index my actions on the equivalence classes that are called concepts. If there is no concept or classification, mapping 2^n percepts to 2^m actions will result in a very giant table. Classification breaks large table to number of smaller simpler problems.

**Concept Hierarchy:**

Concept organization is hierarchical. One advantage is going in a top down manner establishing and refining each category. This helps us to figure out which variables we need to actually key in on and focus on.

**Types of Concept:**

More formal, less formal

Axiomatic concepts, prototype concepts, exemplar concepts.

**Axiomatic concepts**: Concepts defined by a formal set of necessary and sufficient conditions. Example: a circle.

**Prototype concepts**: Base concepts defined by a typical example with overridable properties. Example: a chair. It can be represented using frames with defaults and inheritance.

**Exemplar concepts**: Concepts defined by implicit abstractions of instances, exemplars, of the concept. Example: beauty. It’s very hard to communicate to each other, and sometimes even individual specific.

Another less formal concept is called **QUALIA**. It refers to the raw sensations that we get from our sensors. Example: bitterness.

**Bottom-Up Search:**

AI agent knows the values of each leaf node and needs to predict the root node. Identify and abstract the values of each concept.

Classification is central to **cognition**.

# Lesson 12: Logic

Logic is basic for Planning. It is a formal language that helps us make assertions about the world in a very precise way.

**Why do we need formal logic?**

AI agent will consist of 2 parts:

1. **Knowledge base**: agent’s knowledge of the world. It will be represented in sentences in the language of logic.
2. **Inference engine**: apply rules of inference to the knowledge.
   1. **Soundness**: only valid conclusions can be proven.
   2. **Completeness**: all valid conclusions can be proven.

Example of bird:

If an animal has feathers then it is a bird

**Predicates:**

A function that maps object arguments to true or false. Example: Feathers (Bluebird) = True

If Feather (Animal): Then Bird (Animal) where Animal is object, Feather and Bird are predicates. This is an implicative relationship. It can also be read as Feather(animal) implies Bird(animal).

**Conjunctions and disjunctions:**

Conjunction ‘and’ is denoted by ^

Disjunction ‘or’ is denoted by v

**Implies:**

Implication is denoted by =>

Lays-eggs (animal) ^ Flies (animal) => Bird (animal)

**Notation Equivalency**:

and &, or |, NOT !, IMPLIES =,==

**Truth Tables:**

For and, if true & true = true, otherwise false

For or, if false | false = false, otherwise true

**Commutative Property:**

A & B == B & A

**Distributive Property:** works for any mixture of operators

A & (B | C) == A & B | A & C

**Associative Property:** needs both ors or both ands

A | (B | C) == (A| B) | C

**De Morgan’s Law:**

! (A & B) ==! A |! B

**Truth of implication:**

* **A B A=>B**
* true true true
* true false false
* false true true
* false false true

**Implication Elimination:**

It is useful to rewrite sentences to eliminate implications. Implication cannot occur in a conjunctive normal form.

Given: a=>b, Rewrite as: !a | b

**Rules of Inference:**

Instantiate general rules to prove specific claims.

1. **Modus Ponens**

Sentence 1: p => q

Sentence 2: p

Therefore, Sentence 3: q

1. **Modus Tollens**

Sentence 1: p => q

Sentence 2: !q

Therefore, Sentence 3: !p

S1: knowledge from bootstrap

S2: knowledge from percepts of the story

S3: knowledge from logical inference

These methods are not computationally feasible especially for complex tasks, when agents have limited computational resources and when knowledge base is very large.

**Universal Quantifier:** True for all values of x

It denotes all values that a variable can take.

For all animals,

∀x [Lays-eggs (x) & Flies (x) => Bird (x)]

**Existential Quantifier:** True for some or at least one value of y

For at least one animal,

∃y [Lays-eggs (y) & Flies (y) => Bird (y)]

**A Simple Proof: Resolution Theorem Proving**

S1: !can-move => !liftable

By implication elimination,

**S1: can-move | !liftable**

**S2: !can-move**

Here, prove that, S3: ! liftable

Assume, **S3: liftable**

From S1, S2, S3, liftable and !liftable cancels out, can-move and !can-move also cancels out. This results in Null condition which is a contradiction. Hence, we can infer that **S3: !liftable**

**A More Complex Proof:**

S1: !can-move & battery-full => !liftable

By implication elimination,

S1: !(!can-move & battery-full) | !liftable

By deMorgan’s law,

**S1: can-move | !battery-full) | !liftable**

**S2: !can-move**

**S3: battery-full**

Assume, **S4: liftable**

This results in Null condition and hence, we can infer **S4: !liftable**

A **horn clause**: disjunction that contains at most one positive literal. Example: S1.

# Lesson 13: Planning

**Goals to Propositional Logic**

Goals: The ceiling is painted and the ladder is painted

In propositional logic: Painted (Ceiling), Painted (Ladder)

**Goal State as Conjunction:**

Painted (Ceiling) & Painted (Ladder)

**Initial State:**

On (Robot, Floor) & Dry (Ladder) & Dry (Ceiling)

Next State: On (Robot, Ladder) & Painted (Ceiling)

**Operator**s:

In propositional logic:

climb-ladder:

Precondition: On (Robot, Floor) & Dry (Ladder)

Postcondition: On (Robot, Ladder)

Preconditions cannot have negative literals, while postconditions can have negative literals.

**Planning and State Spaces:**

It consists of series of states linking them with operators. So, in planning, intelligent agents map perpetual history into actions.

Planning is required to systematically organize the operators in order to avoid conflicts between various operators.

Linear Planner assumes the goals can be achieved in any order. It does not try to reason about the conflict between the goals. This results in goal clobbering.

**Partial Order Planning:**

It occurs when there are multiple goals and the plan for achieving one goal cobblers another goal.

Also known as Non-linear planner. **Example**:

On(Robot, Floor) & Dry(Ladder) & Dry(Ceiling)

climb-ladder

On(Robot, Ladder) & Dry(Ladder) & Dry(Ceiling)

paint-ceiling

On(Robot, Ladder) & Dry(Ladder) & Painted(Ceiling) & !Dry(Ceiling)

Here, we are working backwards using goal knowledge as control knowledge to select between operators. Here, agent identifies the sub-goals to accomplish the final goal.

**Detecting conflicts:**

There are 2 plans: Painted (Ladder) and Painted (Ceiling). We need to examine the relationship between the two plans and see if there are any conflicts between them.

For each precondition in current plan: If precondition for an operator in the current plan is clobbered by a state in another plan: Promote current plan’s goal above other plan’s goal.

Therefore, **Promote** Painted (Ceiling) **above** Painted (Ladder)

**Open Preconditions:**

When connecting 2 plans, if there is an open precondition, then select an operator whose postcondition will match the later state. and whose precondition will match the former state.

**Example**: Partial Order Plan for Blocks A, B, C, D on Table

Final Plan:

* Move(D, Table)
* Move(B, Table)
* Move(A, Table)
* Move(C, D)
* Move(B, C)
* Move(A, B)

**Hierarchical Task Network Planning:**

**Hierarchical Decomposition:**

Abstract out into a macro operator called unstack (1st three operations) and stack-ascending (last 3 operations).

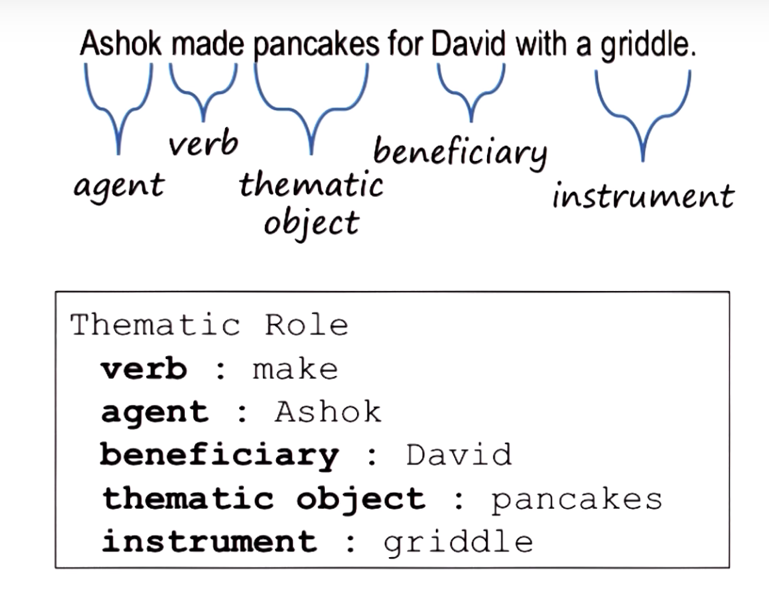
**Hierarchical Planning:**

complex problems are address by thinking at multiple levels of abstraction, so that at any one level of abstraction, the problem appears small and simple. In order to reason at multiple levels of abstraction, we need knowledge at these levels. In the case of multiple blocks example, there was knowledge not only at the level of move operations but also at the level of macro operations like unstack and stack-ascending.

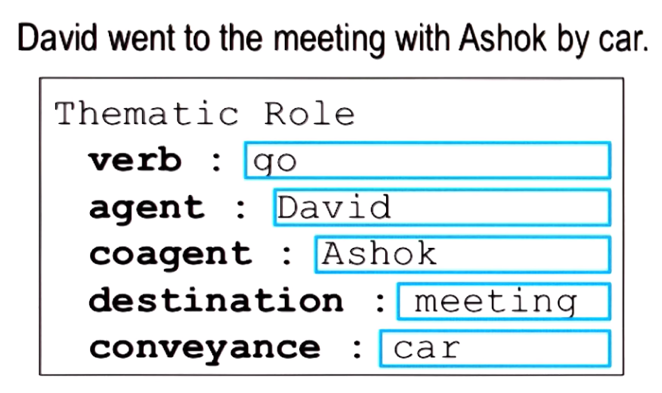
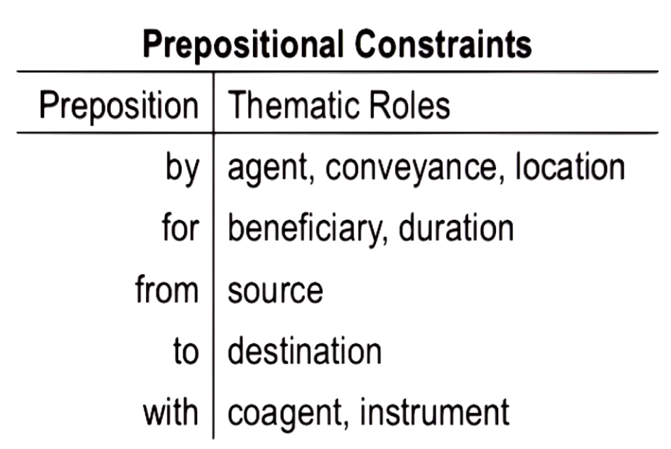
# Lesson 14: Understanding

Understanding comes under Common Sense Reasoning, and it comes after Frames.

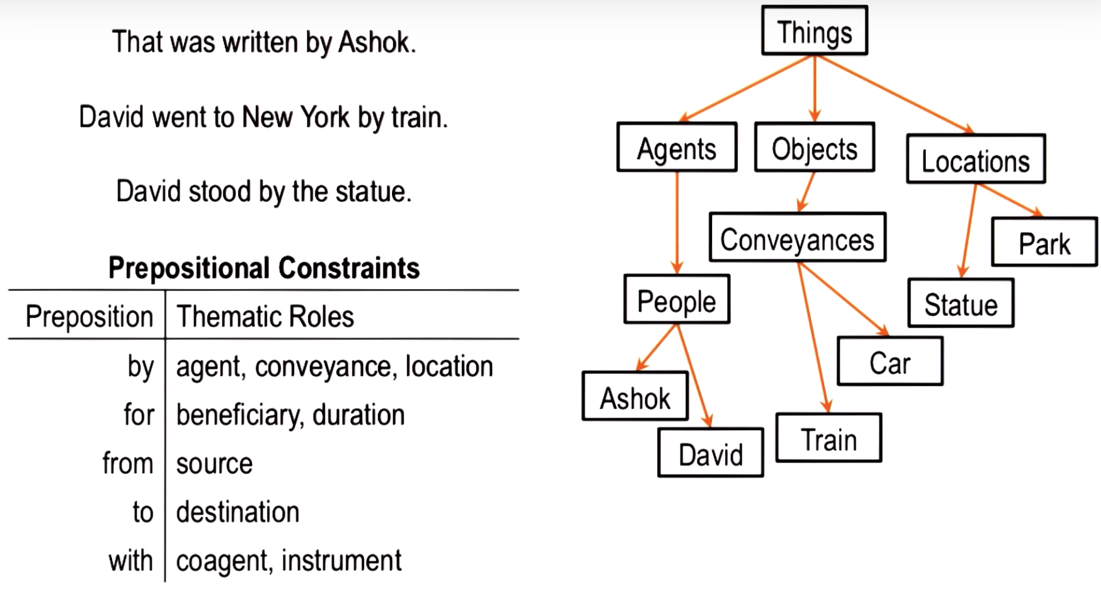
**Thematic Role Systems:**

Syntactic analysis and lexicon analysis will serve semantic analysis.

It generates expectations.

**Constraints:**

**Resolving Ambiguity in Propositions:**

Agent uses ontology or conceptualization of the world.

**Ambiguity in Verbs:**

Example: take

This verb has many interpretations, and each interpretation has a frame-like representation. This representation refers to the thematic roles that go with the particular meaning of take.