Module III

AI representational schemes- Semantic nets, conceptual dependency, scripts, frames, introduction to agent based problem solving, Machine learning-symbol based-a frame work for symbol based learning.

AI representational Schemes

Knowledge Representation

- **Problem**: represent human knowledge into computationally acceptable language
- **Desired Features**
  - *Exhaustiveness* ➔ All needed information is in KB.
  - *Modifiability* ➔ new information can be added without sacrificing consistency.
  - *Homomorphic mapping of objects* ➔ information organized in a natural and intuitive fashion
  - *Computational Efficiency*

Approaches to Problem Solving in AI

- **Different views**
  - **Weak Problem Solvers:** To create intelligent systems, we simply need to transform the syntactic form of the start state to match that of the desired goal state.
    - Example: General Problem Solver
  - **Strong Problem Solvers:** To create a system that acts intelligently we must represent world knowledge in a form accessible to the system.
    - Example: expert systems such as MYCIN.
  - **Subsumption Architecture:** “The world is its own model”.
  - **Genetic algorithms**

Explicit Representation of World Knowledge

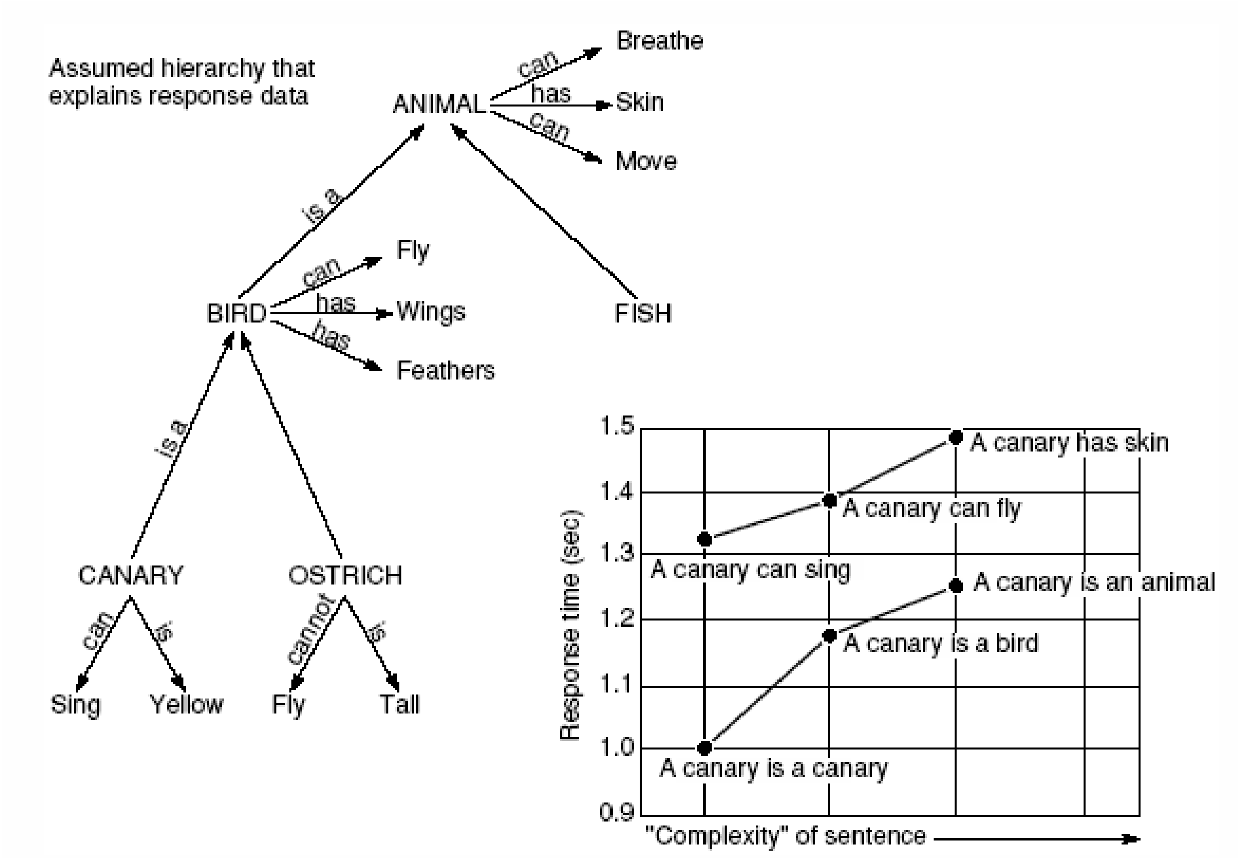
- Logic as a knowledge representation language
  - Propositional Logic
  - Predicate Logic (FOL)

- Semantic networks
- Frames
- Conceptual Dependency

AI Representational Schemes: Associationist Theories of Meaning

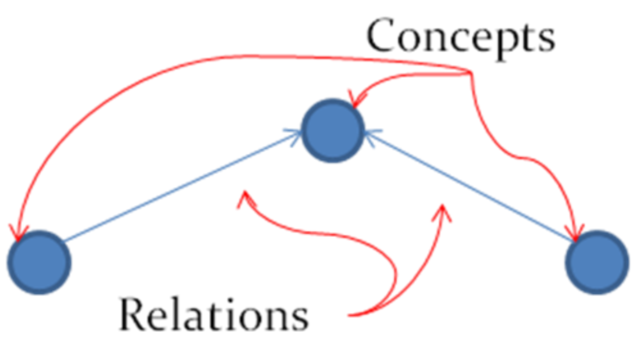
- Define the meaning of an object in terms of a network of associations with other objects.
- When humans perceive and reason about an object, the perception is first mapped into a concept.
- This concept is connected through appropriate relationships to other concepts.
- These relationships form an understanding of the properties and behavior of objects.
- For example, through experience, we associate the concept snow with other concepts such as cold, white, snowman, slippery, and ice.
- In addition to their ability to associate concepts, humans also organize their knowledge hierarchically, with information kept at the highest appropriate levels of the taxonomy.
- Collins and Quillian modeled human information storage and management using a semantic network (Figure).
- The structure of this hierarchy was derived from laboratory testing of human subjects.
- The subjects were asked questions about different properties of birds and the response-time varied.
- Graphs are ideal to formalize associationist theories of knowledge.
- Graphs provide a means of explicitly representing relations using arcs and nodes.
- A semantic network represents knowledge as a graph.

Fig 7.1 Semantic network developed by Collins and Quillian in their research on human information storage and response times (Harmon and King, 1985)



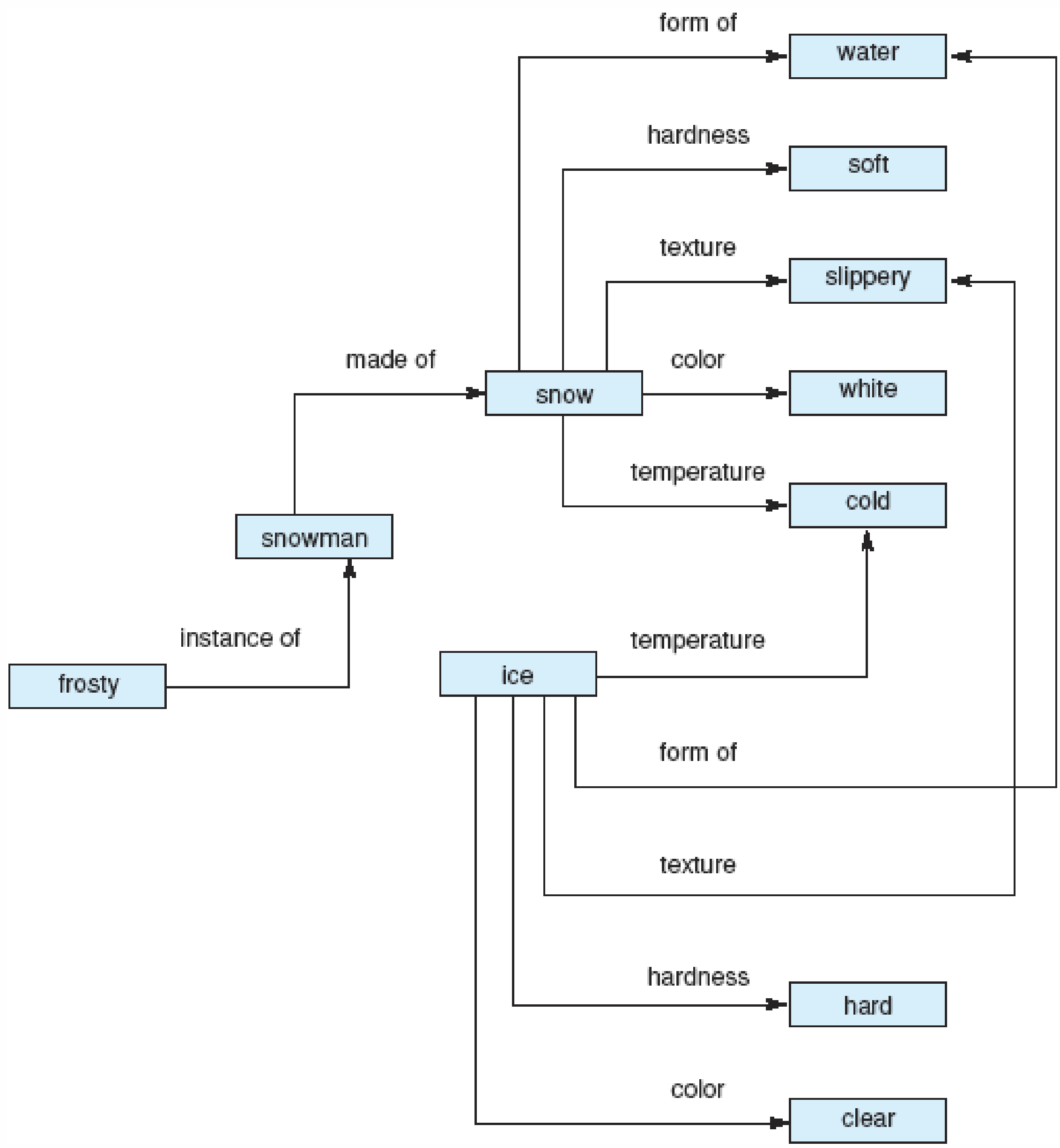
Semantic Networks

- Define objects in terms of their association with other objects
  - e.g. snow, white, snowman, ice, slippery.
- The nodes correspond to facts or concepts, and the arcs to relations or associations between concepts.
- Both nodes and links are generally labeled.
- Represent knowledge as a graph:
- Concepts at lower levels inherit characteristics from their parent concepts.

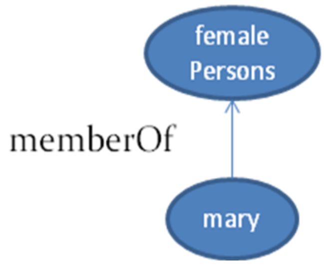


- Concepts at lower levels inherit characteristics from their parent concepts.

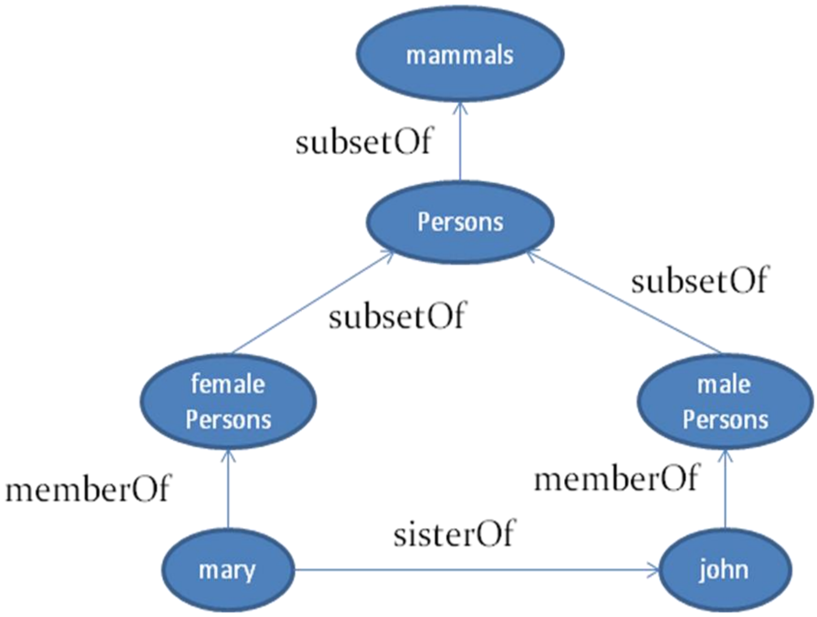
Semantic Network –example



- This network can be used to answer a range of questions about snow, ice, and snowman.
- These inferences are made by following the links to related concepts.
- Semantic networks also implement inheritance;
    - for example, frosty inherits all the properties of snowman.
- Well designed semantic networks are a form of logic.



memberOf(femalePersons, mary)



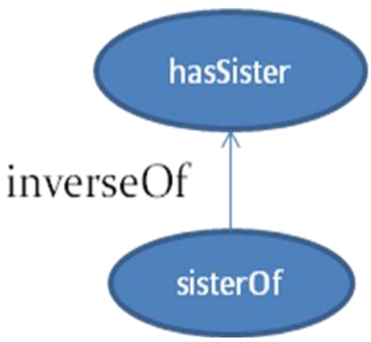
Inference Mechanism

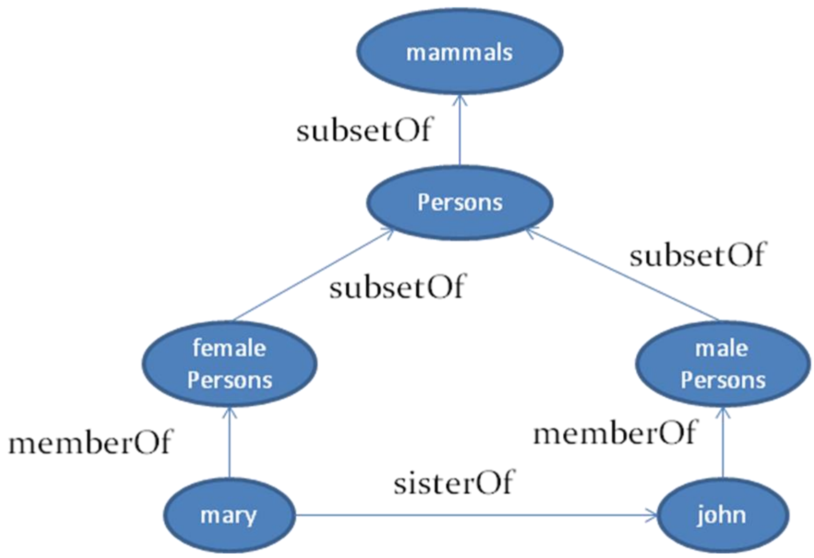
Inheritance

- e.g. Persons by default have 2 legs. How many legs does Mary have? John?

Use of Inverse Links

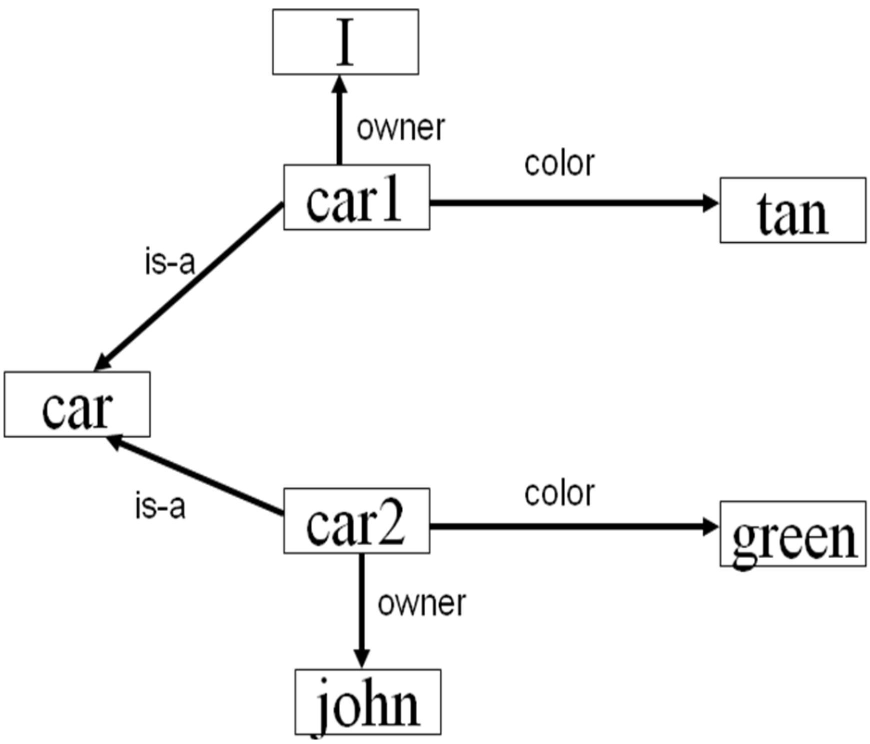
- e.g. hasSister(p, s) and sisterOf(s, p)





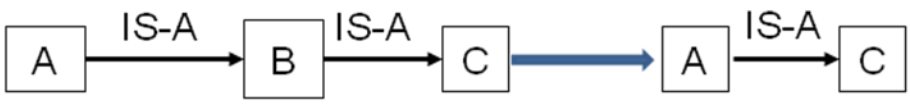
Examples of Semantic Net (2)

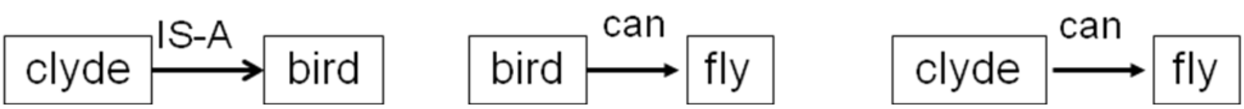
- My car is tan and John's car is green



Inference in a Semantic Net (1)

- **Inheritance**
  - the *is-a* and *instance-of* representation provide a mechanism to implement this.
  - Inheritance also provides a means of dealing with *default reasoning*





Semantic Networks Advantages

- Simple and transparent inference processes.
- Ability to assign default values for categories.
- Ability to include *procedural attachment*.

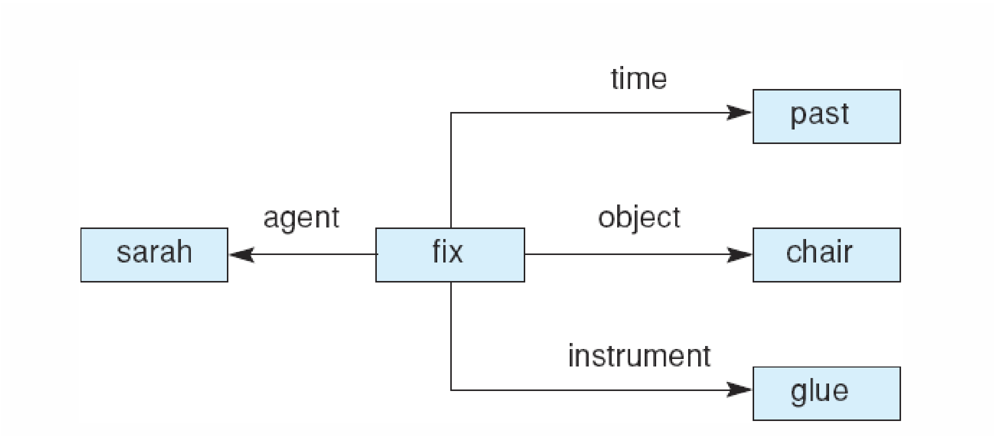
Semantic Networks - Disadvantages

- Simple query language may be too limiting to express complex queries.
- Does not represent full FOL since it does not provide means to use negation, disjunction, and existential quantification.
- n-ary functions must be mapped onto binary functions.

Standardization of Network Relationships

- Simmons addressed the need for standard relationships by focusing on the case structure of English verbs.
- In this verb-oriented approach, links define the roles played by nouns and noun phrases in the action of the sentence.
- Case relationships include agent, object, instrument, location, and time.
- A sentence is represented as a verb node, with various case links to nodes representing other participants in the action.
- This structure is called a case frame.
- In parsing a sentence, the program finds the verb and retrieves the case frame for that verb from its knowledge base.
- It then binds the values of the agent, object, etc., to the appropriate nodes in the case frame.
- Example: Sarah fixed the chair with glue

Fig 7.5 Case frame representation of the sentence "Sarah fixed the chair with glue."



- These built-in relationships indicate that Sarah is the person doing the fixing and that glue is used to put the chair together.
- These linguistic relationships are stored in a fashion that is independent of the actual sentence or even the language in which the sentence was expressed.

Conceptual Dependency (CD) theory

- CD theory was developed by Schank in 1973 to represent the meaning of NL sentences.
    - It helps in drawing inferences
    - It is independent of the language
    - CD representation of a sentence is not built using words in the sentence rather built using conceptual primitives which give the intended meanings of words.
- CD provides structures and specific set of primitives from which representation can be built.

Conceptual category

There are four primitive conceptual categories

- ACT Actions {one of the CD primitives}
- PP Objects {Picture Producers}
- AA Modifiers of actions {Action Aiders}
- PA Modifiers of PP's {Picture Aiders}

Primitive ACTs of CD theory

- ATRANS Transfer of an abstract relationship (i.e. give)

- PTRANS Transfer of the physical location of an object (e.g., go)
- PROPEL Application of physical force to an object (e.g. push)
- MOVEMovement of a body part by its owner (e.g. kick)
- GRASP Grasping of an object by an action (e.g. throw)
- INGEST Ingesting of an object by an animal (e.g. eat)
- EXPEL Expulsion of something from the body of an animal (e.g. cry)
- MTRANS Transfer of mental information (e.g. tell)
- MBUILD Building new information out of old (e.g decide)
- SPEAK Producing of sounds (e.g. say)
- ATTEND Focusing of a sense organ toward a stimulus (e.g. listen)
    - Primitives of meaning
        1. Actions
        2. Objects
        3. Modifiers of actions
        4. Modifiers of objects
    - *Conceptual syntax rules*
        1. Built using these primitives
        2. Constitute a grammar of meaningful semantic relationships.
    - *Conceptual dependency relationships*
        1. Are defined using the conceptual syntax rules
        2. Can be used to construct an internal representation of an english sentence.
- The first conceptual dependency describes the relationship between a subject and its verb and The third describes the verb-object relation in the following figure.

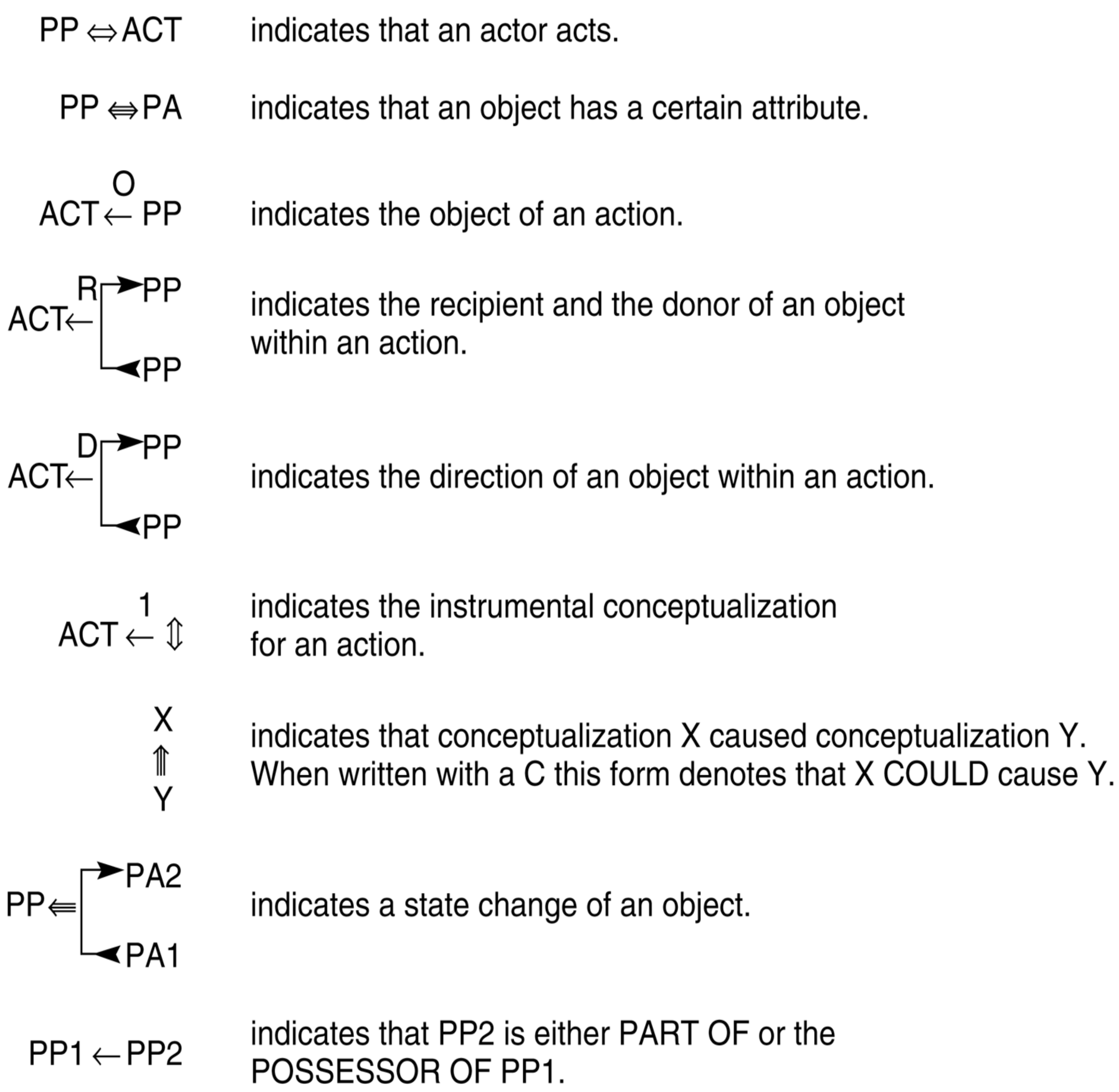


Fig. Basic Conceptual Dependency

Few conventions

- Arrows indicate directions of dependency
- Double arrow indicates two way link between actor and action.

  O – for the object case relation

  R – for the recipient case relation

  P – for past tense

  D – destination

  - Tense and mode are added.

  - Example: past, future, transition etc.
- Schank supplies a list of attachments or modifiers to the relationships.
- A partial list of these includes:

p	past
f	future
t	transition
k	continuing
t_s	start transition
?	interrogative
t_f	finish transition
c	conditional
/	negative
nil	present
delta?	timeless

Example:

"John throws the ball"

John $\Leftrightarrow$ *PROPEL* $\xleftarrow{O}$ Ball

"John threw the ball"

John $\overset{P}{\Leftrightarrow}$ *PROPEL* $\xleftarrow{O}$ Ball

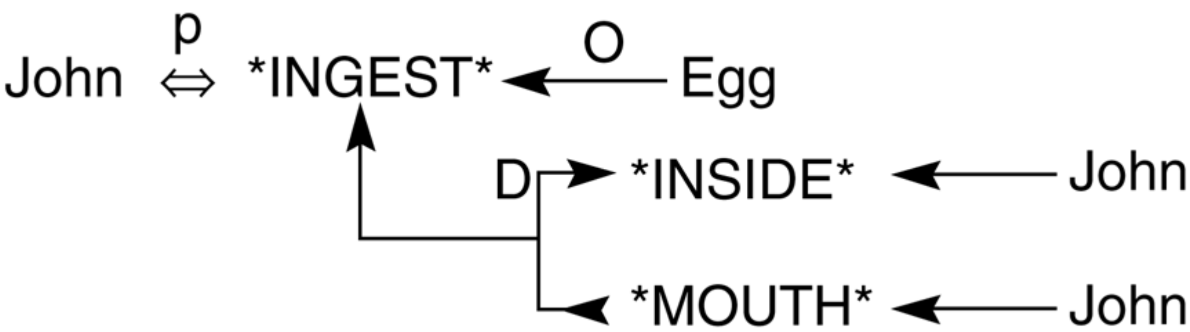
Fig 7.8 Some bacis conceptual dependencies and their use in representing more complex English sentences, adapted from Schank and Colby (1973).

1.	PP $\Leftrightarrow$ ACT	John $\overset{p}{\Leftrightarrow}$ PTRANS	John ran.
2.	PP $\Leftrightarrow$ PA	John $\Leftrightarrow$ height (>average)	John is tall.
3.	PP $\Leftrightarrow$ PP	John $\Leftrightarrow$ doctor	John is a doctor.
4.	PP $\uparrow$ PA	boy $\uparrow$ nice	A nice boy
5.	PP $\Uparrow$ PP	dog $\Uparrow$ POSS-BY John	John's dog

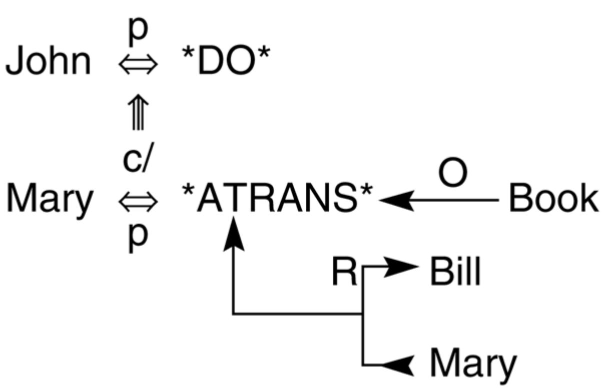
- The symbols have the following meanings:

$\leftarrow$	indicates the direction of dependency
$\Leftrightarrow$	indicates the agent–verb relationship
p	indicates past tense
INGEST	is a primitive act of the theory
O	object relation
D	indicates the direction of the object in the action

Conceptual dependency representing "John ate the egg"



Conceptual dependency representation of the sentence "John prevented Mary from giving a book to Bill"



- Example
- I gave a book to the man. CD representation is as follows:

P O R man (to)

I ⇔ ATRANS ← book

I (from)

- It should be noted that this representation is same for different saying with same meaning. For example
  - I gave the man a book,
  - The man got book from me,
  - The book was given to man by me etc.

Rule 1: PP ⇔ ACT

- It describes the relationship between an actor and the event he or she causes.
  - This is a two-way dependency, since neither actor nor event can be considered primary.
  - The letter P in the dependency link indicates past tense.
- Example: John ran

P

CD Rep: John ⇔ PTRANS

Rule 2: ACT ← PP

- It describes the relationship between a ACT and a PP (object) of ACT.
  - The direction of the arrow is toward the ACT since the context of the specific ACT determines the meaning of the object relation.
  - Example: John pushed the bike

O

CD Rep: John ⇔ PROPEL ← bike

Rule 3: PP ↔ PP

- It describes the relationship between two PP's, one of which belongs to the set defined by the other.

  Example: John is doctor

  CD Rep: John ↔ doctor

Rule 4: PP ← PP

- It describes the relationship between two PP's, one of which provides a particular kind of information about the other.
  - The three most common types of information to be provided in this way are possession (shown as POSS-BY), location (shown as LOC), and physical containment (shown as CONT).
  - The direction of the arrow is again toward the concept being described.
  - Example: John's dog

$$\text{dog} \xleftarrow{\text{poss-by}} \text{John}$$

CD Rep dog ← John

Rule 5: PP ⇔ PA

- It describes the relationship between a PP and a PA that is asserted to describe it.
  - PA represents states of PP such as height, health etc.
- Example: John is fat

CD Rep John ⇔ weight (> 80)

Rule 6: PP ← PA

- It describes the relationship between a PP and an attribute that already has been predicated of it.
  - Direction is towards PP being described.
  - Example: Smart John

CD Rep John ← smart

Rule 7: **ACT ←** **R** **→ PP (to)** / **← PP (from)**

- It describes the relationship between an ACT and the source and the recipient of the ACT
  - Example: John took the book from Mary

CD Rep: John ⇔ ATRANS ← R → John / ← Mary

O ↑

Book

Rule 8: PP ← → PA / ← PA

- It describes the relationship that describes the change in state.
  - Example: Tree grows

CD Rep: Tree ← → size > C / ← size = C

⇔ {x}

Rule 9: ⇑

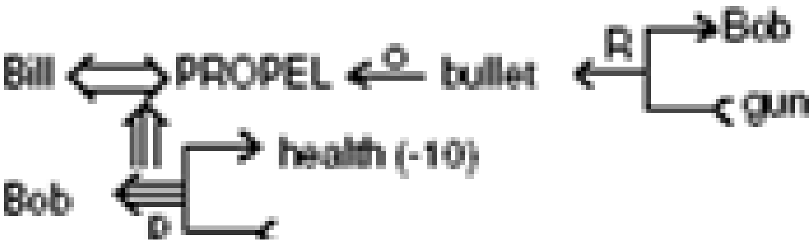
⇔ {y}

- It describes the relationship between one conceptualization and another that causes it.
  - Here {x} causes {y} i.e., if x then y
  - Example: Bill shot Bob

{x} : Bill shot Bob

⇑

{y} : Bob's health is poor



Rule 10: ⇔ {x}
↓
⇔ {y}

- It describes the relationship between one conceptualization with another that is happening at the time of the first.
  - Here {y} is happening while {x} is in progress.

- Example: While going home I saw a snake

  I am going home

  ↓

  I saw a snake

Conceptual Dependency theory

Advantages

- Provides a formal theory of natural language semantics
- Reduces problems of ambiguity.

- Representation directly captures much of the natural language semantics
- Sentences with similar meaning will have similar representations (*canonical form*).

Disadvantages:

- No program exists that can reliably reduce sentences to canonical form.
- Primitives not sufficient to represent more subtle concepts.

Scripts

- A script is a structured representation describing a stereotyped sequence of events in a particular context.(i.e. if the system isn't told some detail of what's going on, it assumes the "default" information is true).
- Scripts are used in natural language understanding systems to organize a knowledge base in terms of the situations that the system is to understand.

Why scripts?

1. Because real-world events do follow stereotyped patterns. Human beings use previous experiences to understand verbal accounts; computers can use scripts instead.
2. Because people, when relating events, do leave large amounts of assumed detail out of their accounts. People don't find it easy to converse with a system that can't fill in missing conversational detail.

- Scripts predict unobserved events.
- Scripts can build a coherent account from disjointed observations.

Applications

- This sort of knowledge representation has been used in intelligent front-ends, for systems whose users are not computer specialists.
- It has been employed in story-understanding and news-report-understanding systems.

Components of Scripts

- Script name (eg: restaurant)
  - Entry conditions: descriptors of the world that must be true for the script to be called.

Eg:- open restaurant, hungry customer

  - Roles: actions that the individual participants perform.

Eg:- waiter takes orders, customer orders, eats

- Props: the "things" that support the content of the script.

Eg:- tables, waiters

- Scenes: the script is broken into a sequence of scenes each of which presents a temporal aspect of the script.

Eg:- entering, ordering, eating, etc

- Results: facts that are true once the script has terminated.

Eg:- the customer is full and poorer

- The elements of the script are represented using conceptual dependency relationships.
- Placed together in a frame-like structure, they represent a sequence of meanings, or an event sequence.
- The program reads a small story about restaurants and parses it into an internal conceptual dependency representation.
- The program binds the people and things mentioned in the story to the roles and props mentioned in the script.
- The result is an expanded representation of the story contents, using the script to fill in any missing information and default assumptions.
- The program then answers questions about the story by referring to the script.
- Scripts

Example - Restaurant script.

Script: Restaurant

Track: Coffee Shop

Roles: S=Customer

W=Waiter

C=Cook

M=Cashier

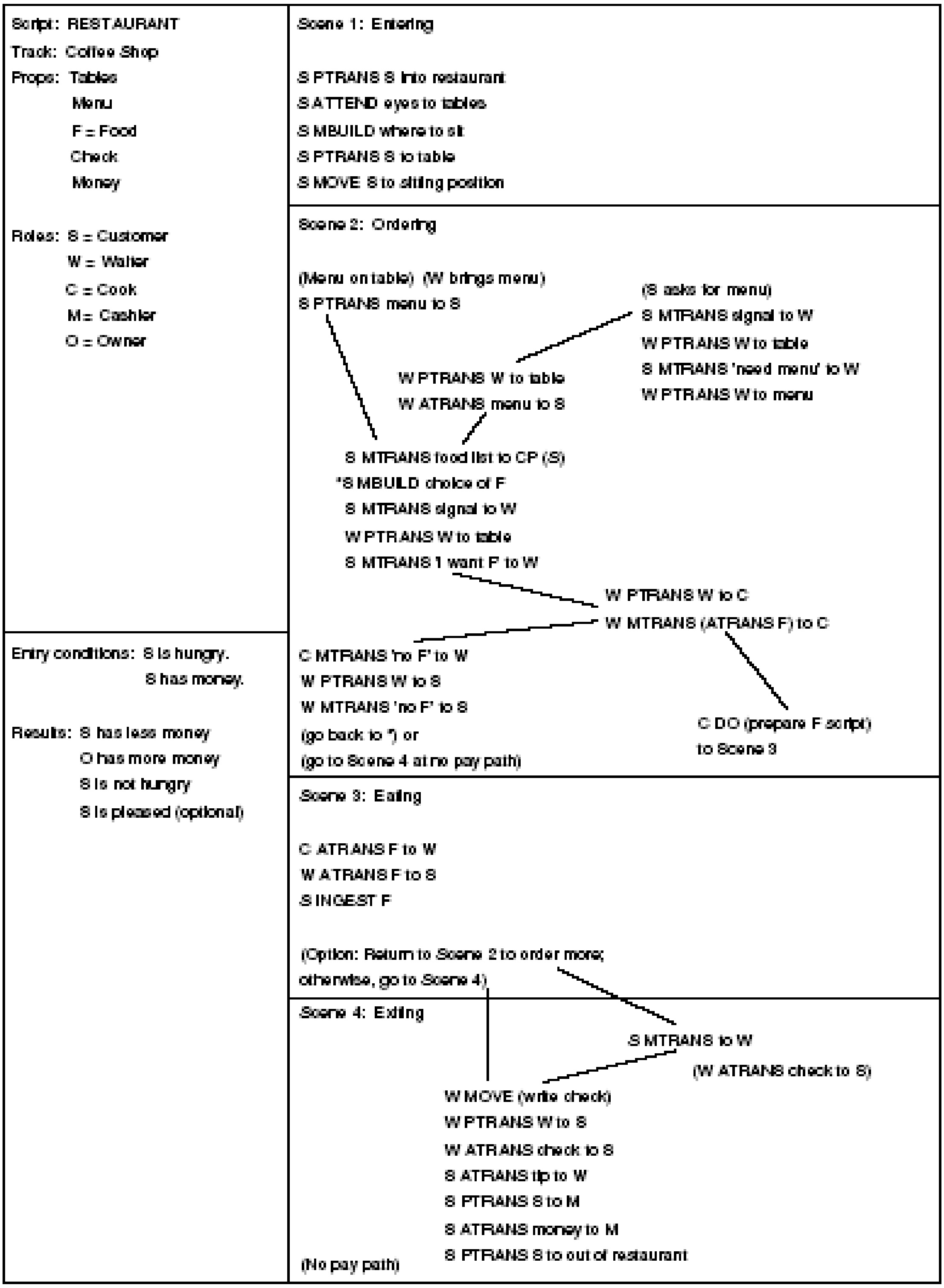
O=Owner

Props: Tables

F=Food

Check

Money & MENU



<u>Entry conditions :</u> S is hungry

S has money

<u>Results :</u> S has less money

O has more money

S is not hungry

S is pleased (optional)

Scene 1: Entering

S PTRANS S into restaurant

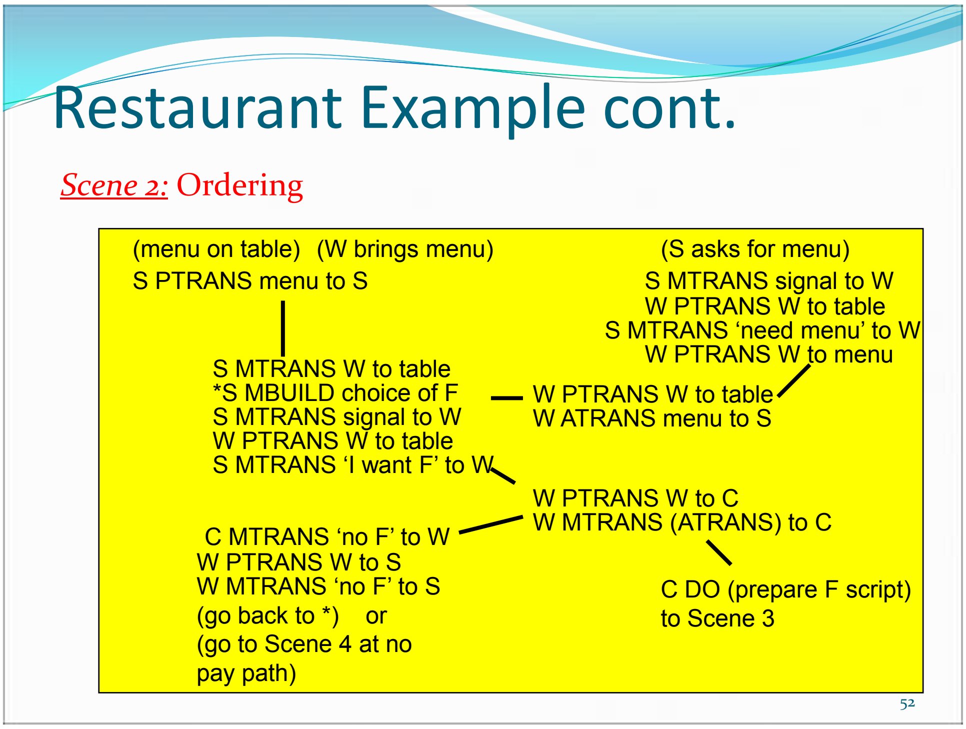
S ATTEND eyes to tables

S MBUILD where to sit

S PTRANS S to table

S MOVE S to sitting position

Scene 2: Ordering



Restaurant Example cont.

Scene 3 : Eating

C ATRANS F to W
W ATRANS F to S
S INGEST F
(Option : Return to Scene 2 to order more; otherwise go to Scene 4)

53

Restaurant Example cont.

Scene 4 : Exiting

S MTRANS to W
(W ATRANS check to S)
W MOVE (write check)
W PTRANS W to S
W ATRANS check to S
S ATRANS tip to W
S PTRANS S to M
S ATRANS money to M
S PTRANS S to out of
(No pay path)restaurant

54

Advantages / Disadvantages of Script

Advantages

- Capable of predicting implicit events
- Single coherent interpretation may be build up from a collection of observations.

Disadvantage

- More specific (inflexible) and less general than frames.
- Not suitable to represent all kinds of knowledge.
- To deal with inflexibility, smaller modules called memory organization packets (MOP) can be used.

- MOPs represent knowledge as smaller components along with rules for dynamically combining them to form a schema that is appropriate to the current situation.

Frames

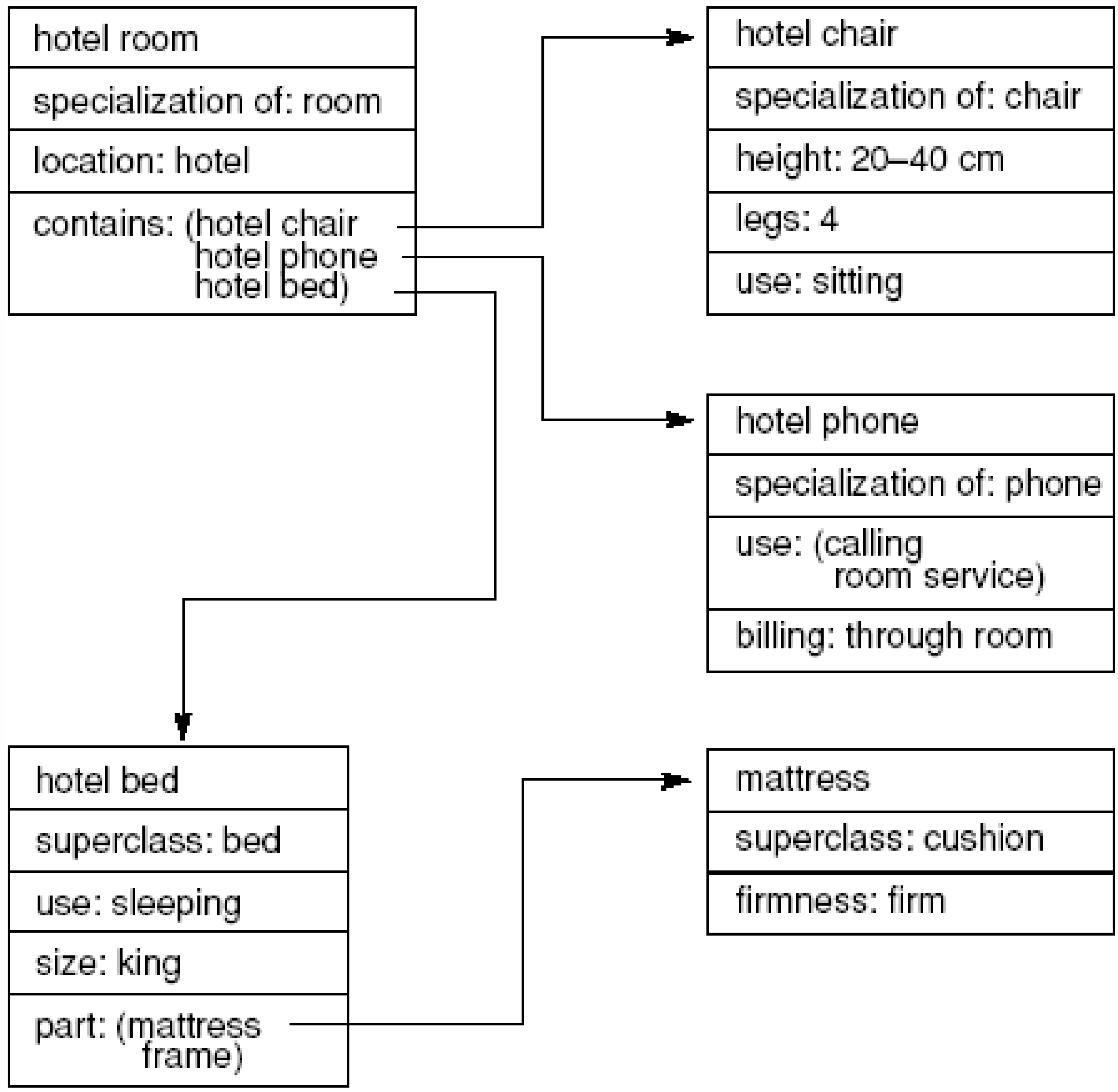
- Another representational scheme, in many ways similar to scripts, used to capture the implicit connections of information in a problem domain, was called frames.
- Frames support the organization of knowledge into more complex units reflecting the organization of objects in the domain.
- Can be viewed as a static data structure with values attached.
- Each individual frame may be seen as a data structure, similar to the traditional "record", that contains information relevant to stereotyped entities.
- The slots in the frame contain information such as:
    - Frame identification information.
    - Relationship of this frame to other frames.
    - Descriptors of requirements for a frame.
    - Procedural information on use of the structure described – attaches procedural code to a slot
    - Frame default information – slot values taken as true when no evidence is found. Eg:- chair has 4 legs
    - New instance information – slots may be left unspecified until needed. Eg:- color of bedspread

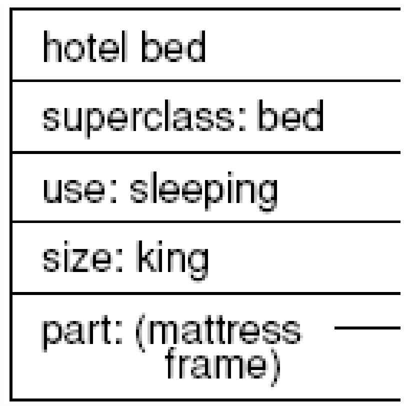
Advantages

- Frames add power and clarity to semantic nets by allowing complex objects to be represented as a single frame.
- Frames provide an easier framework to organize information hierarchically than semantic nets.
- Frames allow for procedural attachment which runs a *demon* (piece of code) as a result of another action in the KB (this has also been done to some semantic nets).

- Both frames and semantic nets support class inheritance.

Fig: Part of a frame description of a hotel room. "Specialization" indicates a pointer to a superclass.





- Frames provide an easier framework to organize information hierarchically than semantic nets.
- Frames allow for procedural attachment which runs a *demon* (piece of code) as a result of another action in the KB (this has also been done to some semantic nets).
- Both frames and semantic nets support class inheritance.

Agent-Oriented Problem Solving

An *agent* is a problem solver that is:

- *Situated* (interacts with its environment)
- *Autonomous* (makes its own decisions without external intervention)
- *Flexible* (responds to stimuli from the environment, and initiates actions based on situation).
- *Social* (can interact appropriately with other agents or with humans).

Multi-Agent Problem Solvers

- The term multi-agent system refers to all types of software systems composed of multiple semi-autonomous components.
- A particular problem can be solved by a number of modules (agents) which cooperate by dividing and sharing the knowledge about the problem and its evolving solution.
- Multi-agent systems are ideal for representing problems that include many problem-solving methods, multiple viewpoints, and multiple entities.
- Agents interact to
    - cooperate towards achieving a common goal.
    - coordinate in organizing the problem-solving activity.
    - negotiate sub-problem constraints to improve performance.
- Multi-agent systems form a “loosely coupled network of agents that work together” to achieve solutions to problems beyond the capabilities of any individual agent.
- Application domains where agent-based problem solving is appropriate include:
    1. Manufacturing.
    2. Automated Control.
    3. Telecommunications.

4. Transportation Systems,
5. Information Management.
6. E-Commerce.
7. Interactive Games and Theater.

Agent-Oriented Problem Solving Example: ROBOCUP

RoboCup is an annual international robotics competition founded in 1997.

"An international research and education initiative. "

Provides "a standard problem where wide range of technologies can be integrated and examined." (robocup.org)

Main domain: Soccer.

Format: Two teams of robots.

Robots compete in a soccer match on a standard platform.

Example: ROBOCUP

- Agents and objects (an instantiation of a class in OOP) share some similarities but are quite different.
- However, we can use objects to create agents.

Objects in OOP vs. Agents

- Similarities: Objects (like agents) have

- Systems with encapsulated states.
- Certain methods are associated with the object's state.
- Methods support interaction with the environment.
- Different objects communicate by message passing.

Differences:

- Objects do not usually control their own behaviour.
- Agents can initiate their own actions. Object generally do not.
- Objects do not have a *social behaviour.*
- Agents do not invoke methods in one another.
- Interacting agents usually have their own individual thread of control.
- Agents can use more than just simple messages to communicate.
- Objects are associated with their class. Agents can have multiple associations which may also change at any time.
- Emergence can occur from groups of agents but not from objects.

Machine Learning

- AI systems grow from a minimal amount of knowledge by learning
- Learning definition by Herbert Simon (1983):
    - Any change in a system that allows it to perform better the second time on repetition of the same task or on another task drawn from the same population
- Machine learning issues:
    - Generalization from experience
        - Induction – For large domains, a learner usually examines only a fraction; from this limited experience, the learner generalizes to the unseen instances of the domain.
        - Inductive biases – Learners generalize heuristically, that is, they select those aspects of their experience that are most likely to prove effective in the future.
    - Performance change: improve or degrade
- Machine Learning Categories

1. Symbol-based learning
   - Supervised learning
     - Inductive learning -- learning by examples
     - Inductive Bias
     - Explanation-based learning
   - Unsupervised learning
     - Clustering
     - Reinforcement learning: an agent is situated in an environment and receives feedback from that context
2. Neural/connectionist networks
3. Genetic/evolutionary learning

Machine learning: symbol-based

A framework for symbol-based learning

Factors for characterizing learning algorithms:

1. The data and goals of the learning task.
   - Describes the properties and quality of the training data.
   - The data may come from a teacher from the outside environment, or it may be generated by the program itself.
   - Data may be reliable or may contain noise.
   - It can be presented in a well-structured fashion or consist of unorganized data.
   - It may include both positive and negative examples or only positive examples.
   - Data may be readily available, the program may have to construct experiments, or perform some other form of data acquisition.
2. The representation of learned knowledge.
   - Concepts can be represented as
     - Logic expressions in predicate calculus
     - Structured representation such as frames or objects.

- Decision trees
- Rules

Eg: A simple formulation of the concept learning problem (Inferring the general definition of some concepts from specific positive and negative training examples) represents instances of a concept as conjunctive sentences containing variables.

Two instances of “ball”.

size(obj1,small)∧color(obj1,red) ∧shape(obj1,round)

size(obj2,large)∧color(obj2,red) ∧shape(obj2,round)

A more general concept of “ball” is

size(X,Y) ∧color(X,Z) ∧shape(X, round)

3. A set of operations.
    - Given a set of training instances, the learner must construct a generalization, heuristic rule, or plan that satisfies its goals.
    - ie manipulate representations
        - Generalizing or specializing symbolic expressions
        - Adjusting the weights in a neural network
        - Modifying the program’s representations.

Eg: Generalizing a definition by replacing constants with variables.

size(obj1,small)∧color(obj1,red) ∧shape(obj1,round)

Replacing a single constant with a variable produces the generalizations:

size(obj1, X) ∧ color(obj1, red) ∧ shape(obj1, round)

size(obj1, small) ∧ color(obj1, X) ∧ shape(obj1, round)

size(obj1, small) ∧ color(obj1, red) ∧ shape(obj1, X)

size(X, small) ∧ color(X, red) ∧ shape(X, round)

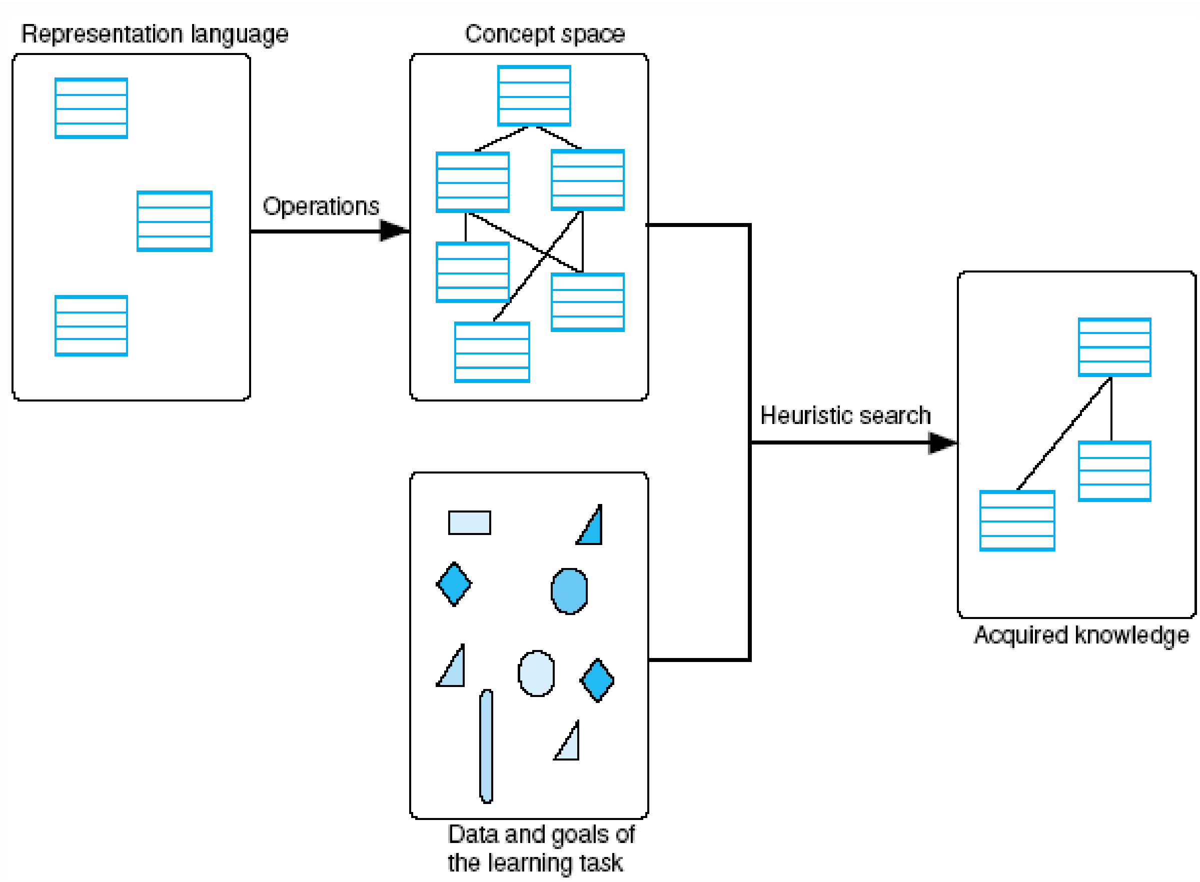
4. The concept space.
    - Search space: its representation, format
    - The learner searches this space to find the desired concept.

- The complexity of this concept space is a primary measure of the difficulty of a learning problem.

5. Heuristic search.

- Learning programs must select a direction and order of search as well as the use of available training data and heuristics to search efficiently.

A general model of the learning process

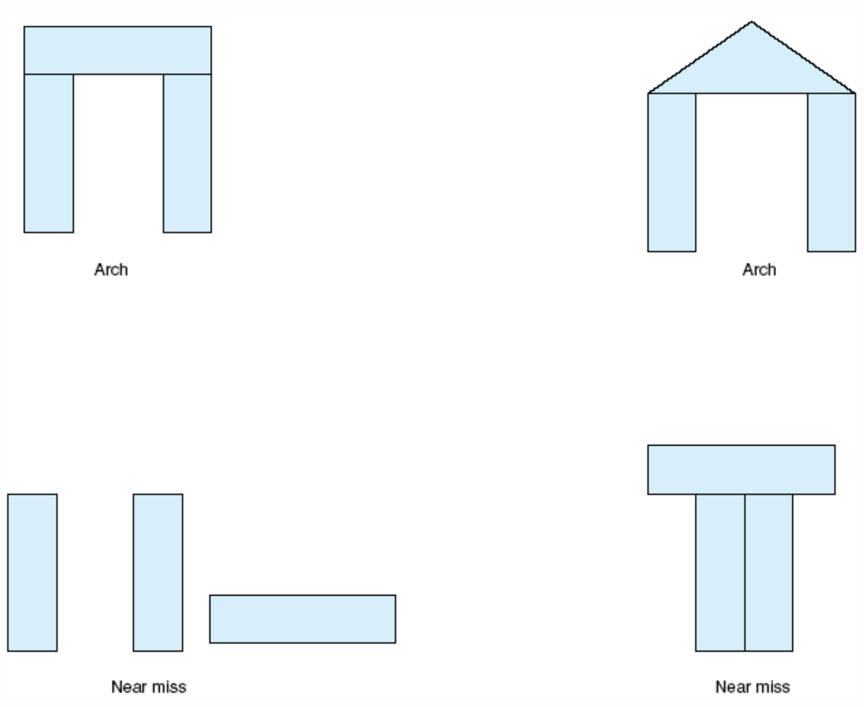


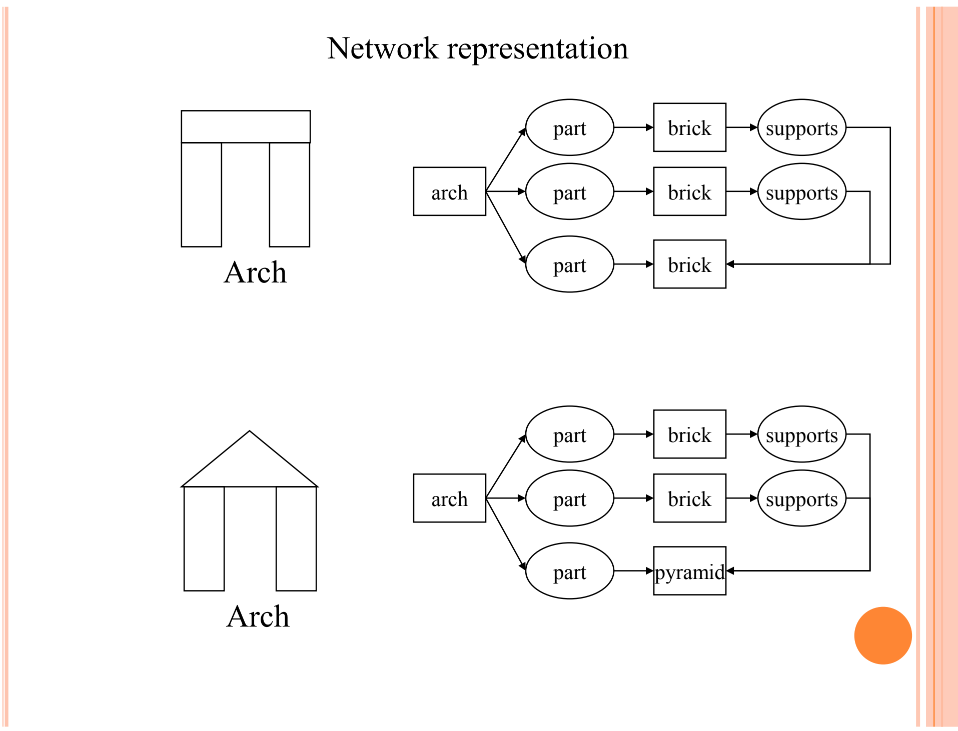
Learning By Examples – Inductive Learning

- Patrick Winston (1975)
  - Given a set of positive and a set of negative examples
  - Find a concept representation
  - Semantic network representation
- Example
  - Learn a general definition of structural concept, say “arch”

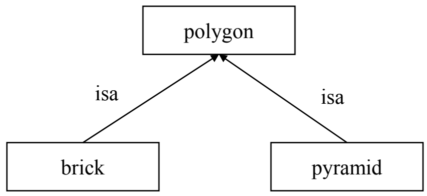
- Positive examples: examples of arch
  - What an arch looks like, to define the arch
- Negative examples: near misses
  - What an arch doesn't look like, to avoid the over-coverage of arch

Examples and near misses for the concept "arch."

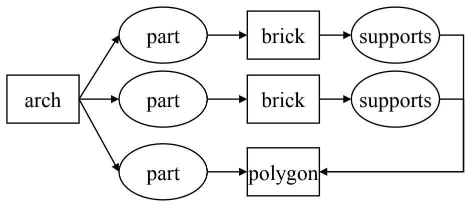




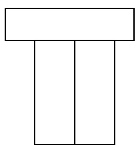
Background knowledge that bricks and pyramids are both types of polygons.



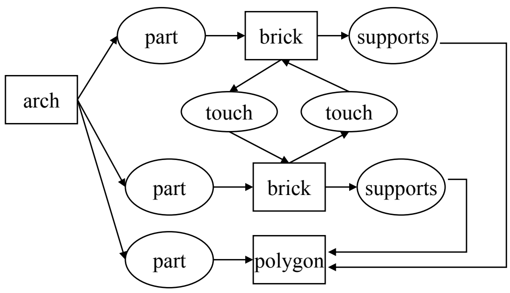
Arch description **generalized** to include both examples.



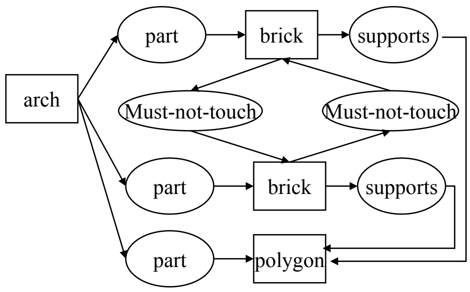
A near miss and its description



Near miss

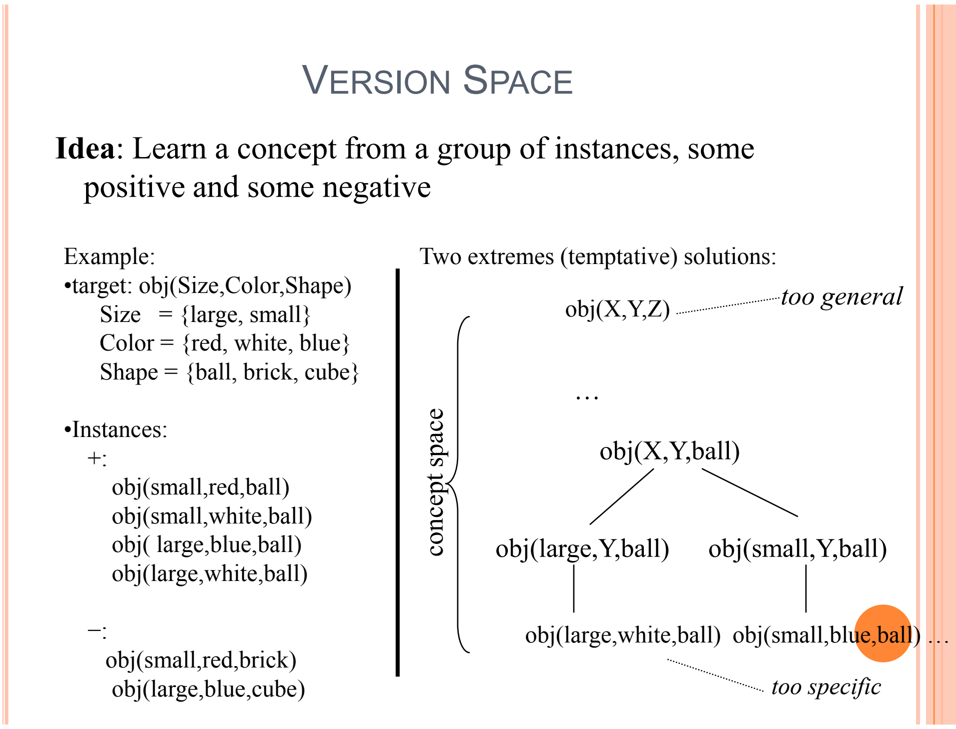


Arch description **specialized** to exclude the near miss

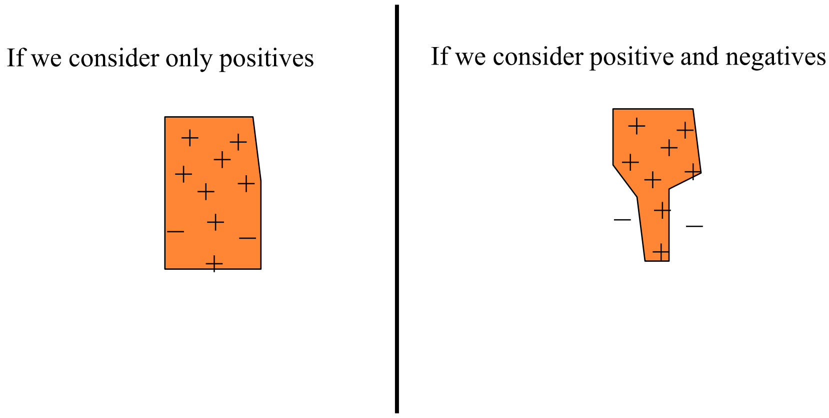


- Version Space
- A **version space** is a hierarchical representation of knowledge that helps to keep track of all the useful information supplied by a sequence of learning examples without remembering any of the examples.
- The **version space method** is a concept learning process accomplished by managing multiple models within a version space.

- A version space description consists of two complementary trees:
  - One that contains nodes connected to overly **general** models, and
  - One that contains nodes connected to overly **specific** models.
  - Node values/attributes are **discrete**.
- Fundamental Assumptions:
  - The data is correct; there are no erroneous instances.
  - A correct description is a conjunction of some of the attributes with values.
- Version Space



HOW VERSION SPACE WORKS



What is the role of the negative instances?

to help prevent over-generalizations

- Version Space Search
- Implements inductive learning as search through a concept space.
- Generalization imposes an ordering on the concepts in the space and uses the ordering to guide the search.
- Generalization & specialization are the most common types of operations for defining a concept space.
    - Extend the coverage of instances
    - Shorten/shrink the constrains
- Some generalization operators
- Replacing constants with variables
    - color(ball,red) *generalizes to* color(X,red)
    - Dropping conditions from a conjunctive expression
    - size(X,small) $\wedge$color(X,red) $\wedge$shape(X,round) *generalizes to*

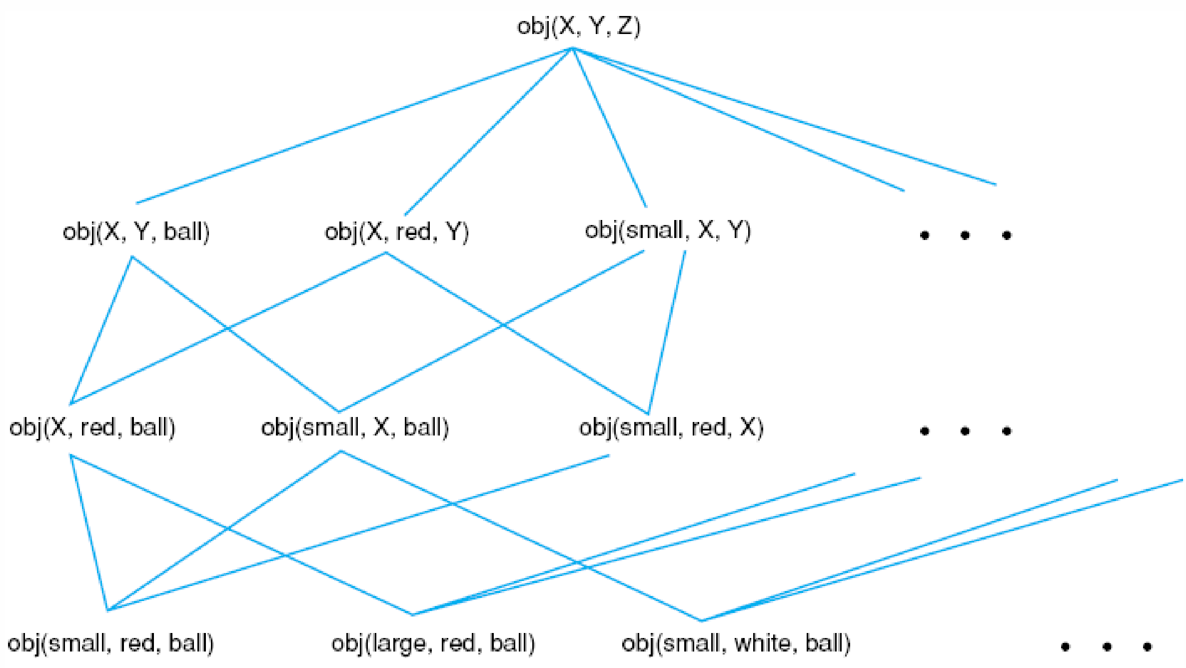
  color(X,red) $\wedge$shape(X,round)

    - Adding a disjunct to an expression
    - size(X,small) $\wedge$color(X,red) $\wedge$shape(X,round) *generalizes to* size(X,small) $\wedge$shape(X,round) $\wedge$(color(X,red)$\vee$color(X,blue))

- Replacing a property with its parent in a class hierarchy
- color(X,red) *generalizes to* color(X,primary_color) where primary_color is a superclass of red.

A concept space:

- Initial state obj(X, Y, Z) might cover all instances: too general
- As more instances are added, X, Y, Z will be constrained



- Version space
- A **version space** is the set of all concept descriptions consistent with the training examples.
- If concept p is more general than concept q, we say that p covers q ($p \geq q$) ie $\forall x\ p(x) \rightarrow positive(x)$ & $\forall x\ q(x) \rightarrow positive(x)$
  - p covers q iff $q(x) \rightarrow positive(x)$ is a logical consequence of $p(x) \rightarrow positive(x)$.

- Version Space Search Algorithms
- Characteristics of these algorithms
  - Data-driven
    - Positive examples to generalize the concept

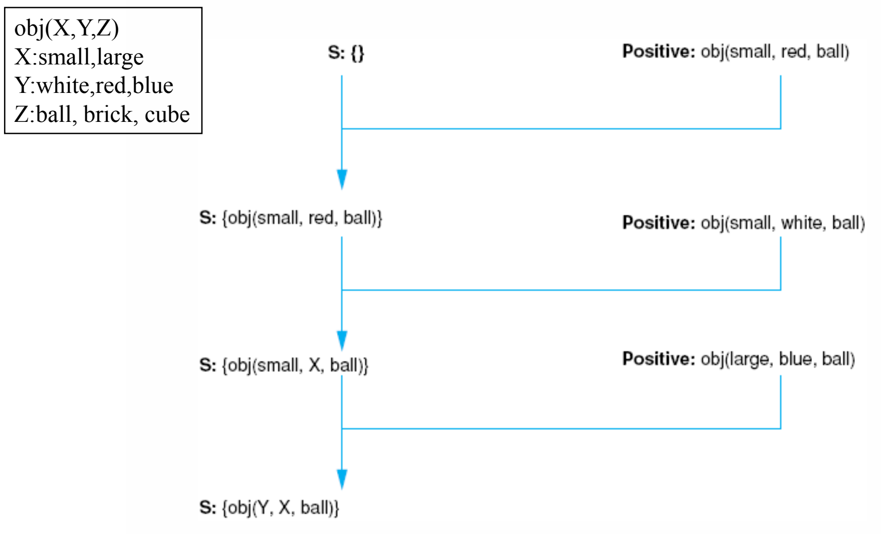
- Negative examples to constrain the concept (avoid overgeneralization)
- Procedure:
  - Starting from whole space
  - Reducing the size of the space as more examples included
  - Finding regularities (rules) in the training data
- Generalization on these regularities (rules)

- Three algorithms
  - Reducing the size of the version space in a specific to general direction
  - Reducing the size of the version space in a general to specific direction
  - Combination of above: candidate elimination algorithm
- Specific to General Search
- Maintains a set S of candidate concepts, the maximally specific generalizations from the training instances
- A concept c is maximally specific if it
  - covers all positive examples, none of the negative examples, and
  - for any other concept c' that covers the positive examples, c≤c'
  - The algorithm uses
  - Positive examples to generalize the candidate concepts
  - Negative example to avoid over generalization

```
Begin
Initialize S to the first positive training instance;
N is the set of all negative instances seen so far;

For each positive instance p
    Begin
    For every s ∈ S, if s does not match p, replace s with its most specific
        generalization that matchs p;
    Delete from S all hypotheses more general than some other hypothesis in S;
    Delete from S all hypotheses that match a previously observed negative
        instance in N;
    End;
For every negative instance n
    Begin
    Delete all members of S that match n;
    Add n to N to check future hypotheses for overgeneralization;
    End;
End
```

Specific to general search of the version space learning the concept “ball.”



General to Specific Search

- Maintains a set G of maximally general concepts
- A concept c is maximally general if it
    - covers none of the negative training examples, and
    - for any other concept c’ that covers no negative training examples, c≥c’

- The algorithm uses
- negative examples to specialize the candidate concepts
- Positive examples to eliminate overspecialization

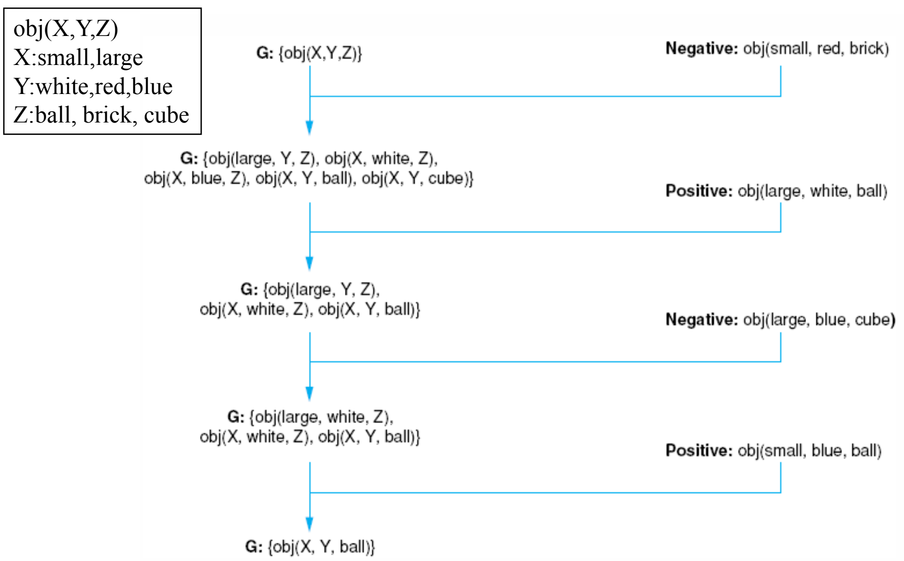
General to Specific Search Algorithm

Begin
Initialize G to contain the most general concept in the space;
P contains all positive examples seen so far;

For each negative instance n
  Begin
  For each g ∈ G that matches n, replace g with its most general specializations that do not match n;
  Delete from G all hypotheses more specific than some other hypothesis in G;
  Delete from G all hypotheses that fail to match some positive example in P;
  End;

For each positive instance p
  Begin
  Delete from G all hypotheses that fail to match p;
  Add p to P;
  End;
End

General to specific search of the version space learning the concept "ball."



Candidate Elimination Algorithm

- Combination of above two algorithms into a bi-direction search

- Maintains two sets of candidate concepts
  - G, the set of maximally general candidates
  - S, the set of maximally specific candidates
  - The algorithm specializes **G** and generalizes **S** until they converge on the target concept.

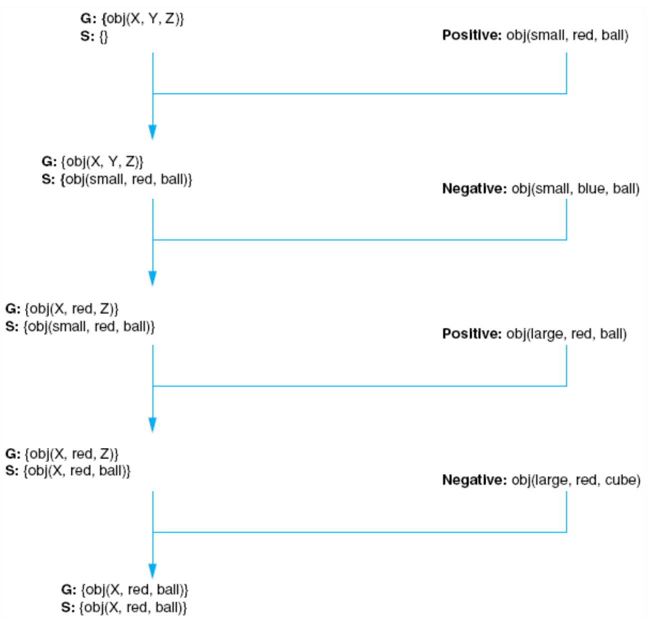
CANDIDATE ELIMINATION ALGORITHM

Begin
Initialize G to be the most general concept in the space;
Initialize S to the first positive training instance;

For each new positive instance p
  Begin
  Delete all members of G that fail to match p;
  For every s ∈ S, if s does not match p, replace s with its most specific
    generalizations that match p;
  Delete from S any hypothesis more general than some other hypothesis in S;
  Delete from S any hypothesis more general than some hypothesis in G;
  End;

For each new negative instance n
  Begin
  Delete all members of S that match n;
  For each g ∈ G that matches n, replace g with its most general specializations
    that do not match n;
  Delete from G any hypothesis more specific than some other hypothesis in G;
  Delete from G any hypothesis more specific than some hypothesis in S;
  End;

The candidate elimination algorithm learning the concept "red ball."



- The candidate elimination algorithm combines the specific-to-general search and the general-to-specific search into a bi-directional search.
- The candidate elimination algorithm is not noise resistant.
- The candidate elimination algorithm will converge toward the target concept if
    - There are no errors in the training samples and
    - There is some concept description in the concept space that correctly describes the target concept.
- The innermost circle encloses the set of known positive instances covered by the concepts in S
- The outermost circle encloses the instances covered by G; any instance outside this circle is negative.
- The shaded portion of the graphic contains the target concept, along with concepts that may be overly general or specific (the ?s).
- The search "shrinks" the outermost concept as necessary to exclude negative instances; it "expands" the innermost concept to include new positive instances.
- Eventually, the two sets converge on the target concept.
- In this fashion, candidate elimination can detect when it has found a single, consistent target concept.
- When both G and S converge to the same concept the algorithm may halt.
- If G and S become empty, then there is no concept that will cover all positive instances and none of the negative instances.

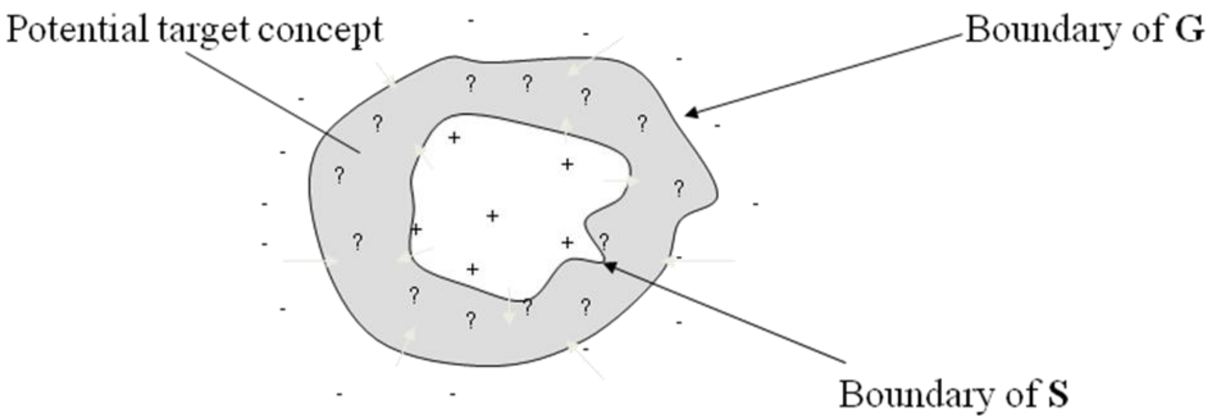


Fig: Converging boundaries of the G and S sets in the candidate elimination algorithm

- Decision Trees

- Learning algorithms of inducing concepts from examples
- Characteristics
  - A tree structure to represent the concept, equivalent to a set of rules
  - Entropy and information gain as heuristics for selecting candidate concepts
  - Handling noisy data
  - Classification – supervised learning
  - Typical systems: ID3, C4.5, C5.0

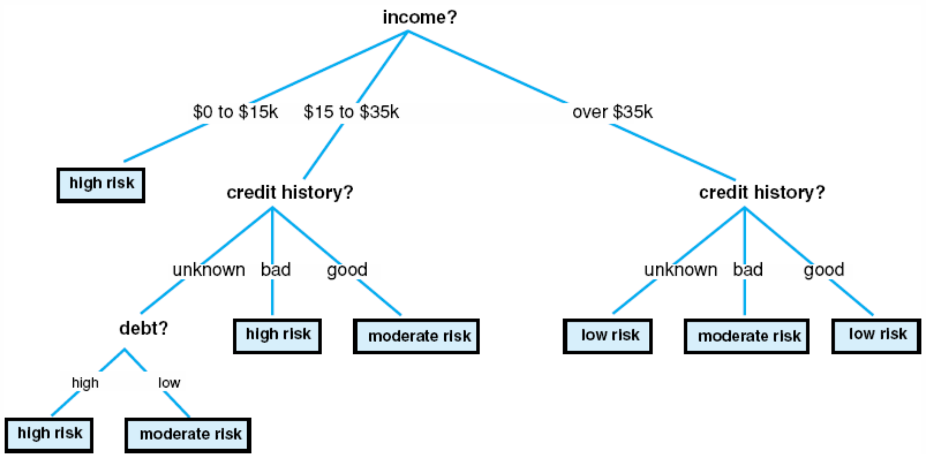
The ID3 decision tree induction algorithm

- ID3 induces concepts from examples.
- ID3 represents concepts as decision trees.
- For example, consider the problem of estimating an individual's credit risk on the basis of such properties as credit history, current debt, collateral, and income.
- Table below lists a sample of individuals with known credit risks.

Data from credit history of loan applications

NO.	RISK	CREDIT HISTORY	DEBT	COLLATERAL	INCOME
1.	high	bad	high	none	\$0 to \$15k
2.	high	unknown	high	none	\$15 to \$35k
3.	moderate	unknown	low	none	\$15 to \$35k
4.	high	unknown	low	none	\$0 to \$15k
5.	low	unknown	low	none	over \$35k
6.	low	unknown	low	adequate	over \$35k
7.	high	bad	low	none	\$0 to \$15k
8.	moderate	bad	low	adequate	over \$35k
9.	low	good	low	none	over \$35k
10.	low	good	high	adequate	over \$35k
11.	high	good	high	none	\$0 to \$15k
12.	moderate	good	high	none	\$15 to \$35k
13.	low	good	high	none	over \$35k
14.	high	bad	high	none	\$15 to \$35k

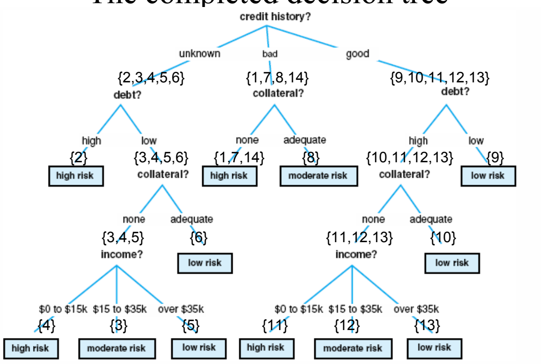
- In a decision tree, each internal node represents a test on some property
- Each possible value of that property corresponds to a branch of the tree
- Leaf nodes represents classification, such as low or moderate risk



ID3 Decision Tree

- ID3 constructs decision trees in a top-down fashion.
- ID3 selects a property to test at the current node of the tree and uses this test to partition the set of examples
- The algorithm recursively constructs a sub-tree for each partition
- This continues until all members of the partition are in the same class

The completed decision tree

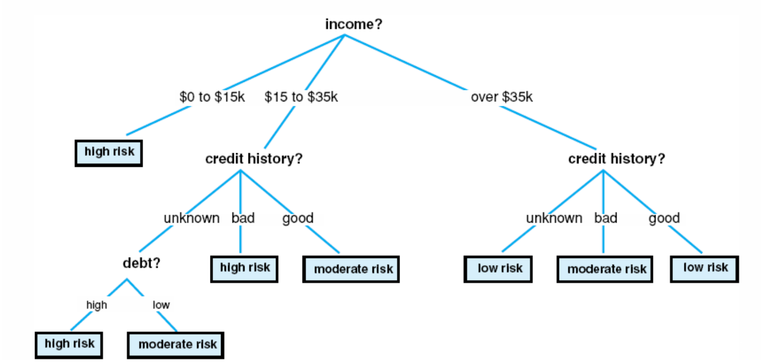


NO.	RISK	CREDIT HISTORY	DEBT	COLLATERAL	INCOME
1.	high	bad	high	none	$0 to $15k
2.	high	unknown	high	none	$15 to $35k
3.	moderate	unknown	low	none	$15 to $35k
4.	high	unknown	low	none	$0 to $15k
5.	low	unknown	low	none	over $35k
6.	low	unknown	low	adequate	over $35k
7.	high	bad	low	none	$0 to $15k
8.	moderate	bad	low	adequate	over $35k
9.	low	good	low	none	over $35k
10.	low	good	high	adequate	over $35k
11.	high	good	high	none	$0 to $15k
12.	moderate	good	high	none	$15 to $35k
13.	low	good	high	none	over $35k
14.	high	bad	high	none	$15 to $35k

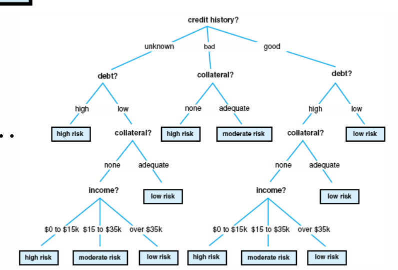
```
function induce_tree (example_set, Properties)

begin
if all entries in example_set are in the same class
   then return a leaf node labeled with that class
   else if Properties is empty
      then return leaf node labeled with disjunction of all classes in example_set
      else begin
         select a property, P, and make it the root of the current tree;
         delete P from Properties;
            for each value, V, of P,
               begin
                  create a branch of the tree labeled with V;
                  let partition_v be elements of example_set with values V for property P;
                  call induce_tree(partition_v, Properties), attach result to branch V
               end
      end
end
```

Completed decision tree for the alternative sequence

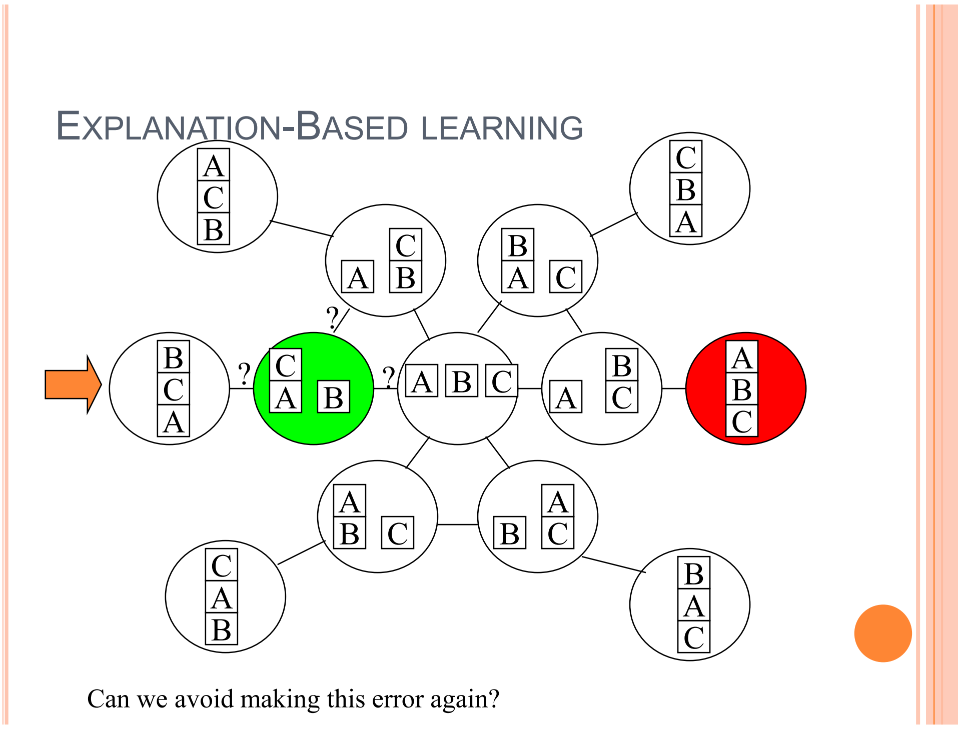


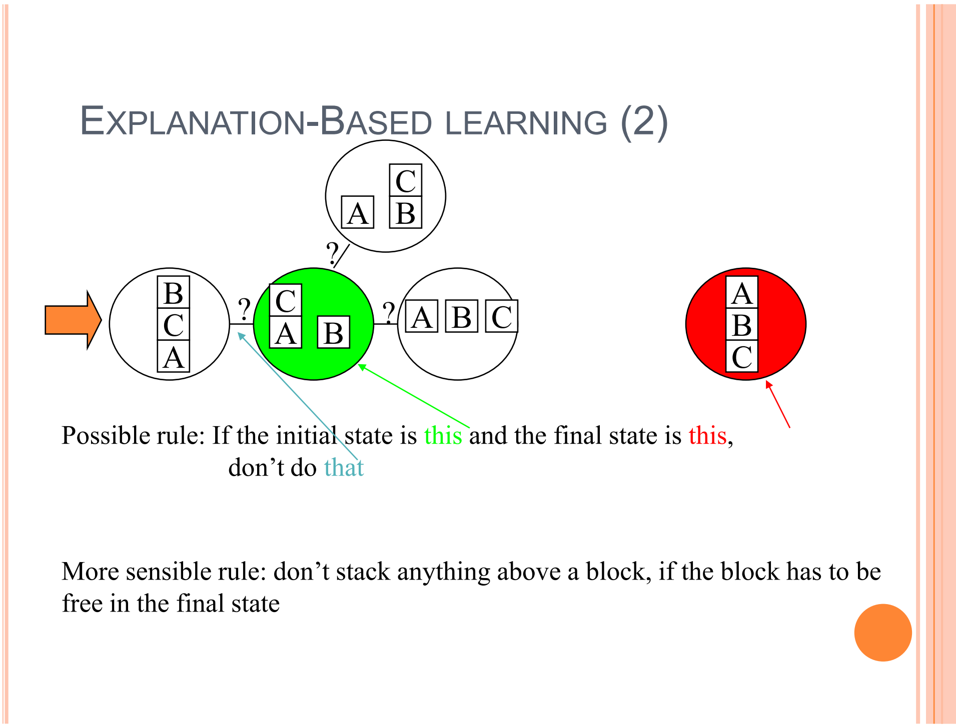
Compare…



Explanation-Based Learning

- Learner is to determine an effective definition for a *target concept* (e.g., a classification, theorem to be proved, plan for achieving a goal, heuristic of a problem solver).
- The learner is given:
    - *A target concept*
    - A *training example*.
    - A *domain theory* (i.e., a set of facts and rules used to explain how the training example is an instance of the goal concept).
    - Some *operationality criteria* (i.e., a means of describing the form that concept definitions may take.)
    - From this EBL computes a generalisation of the training example that is sufficient not only to describe the goal concept but also satisfies the operational criterion.
- This has two steps:
    - **Explanation** -- the domain theory is used to prune away all unimportant aspects of the training example with respect to the goal concept.
    - **Generalisation** -- the explanation is generalised as far possible while still describing the goal concept.





Unsupervised learning

- **Unsupervised Learning** eliminates the teacher and requires that the learners form and evaluate concepts their own.
- Science is perhaps the best example of unsupervised learning in humans.
- Scientists do not have the benefit of a teacher.
- Instead, they propose hypotheses to explain observations
- The clustering problem starts with (1) a collection of unclassified objects and (2) a means for measuring the similarity of objects.
- The goal is to organize the objects into classes that meet some standard of quality, such as maximizing the similarity of objects in the same class.

Unsupervised Learning: Clustering

- Numeric taxonomy
  - Objects represented as collection of numeric features (feature vector; a point in an multi-dimensional space)
  - A similarity function is defined for two objects according to Euclidean distance
  - e.g., agglomerative clustering; k-means
- Symbolic taxonomy

- Objects represented as collection of symbolic features
- Similarity could be proportion of features in common
- Conceptual clustering
- Produces general concept definitions
- Applies background knowledge to formation of categories

Reinforcement Learning

- In reinforcement learning, we design computational algorithms for transforming world situations into actions in a manner that maximizes a reward measure.
- Our agent is not told directly what to do or which action to take; rather, the agent discovers through exploration which actions offer the most reward.
- The agent acts on its environment, it receives some evaluation of its action (reinforcement), but is not told of which action is the correct one to achieve its goal.

Examples

- Game playing:

  Player knows whether it win or lose, but not know how to move at each step.
- Control:

  A traffic system can measure the delay of cars, but not know how to decrease it.
- Playing chess:

  Reward comes at end of game
- Animals:

  Hunger and pain - negative reward

  food intake – positive reward

RL is learning from interaction