DAYANANDA SAGAR COLLEGE OF ENGINEERING Dept. of Information Science & Engg
(An Autonomous Institute Affiliated to VTU, Belagavi)
Shavige Malleshwara Hills, Kumaraswamy Layout, Bengaluru-560078
Dr. Rajeshwari.J
Associate Professor
Dayananda Sagar College Engineerin

Knowledge representation

What is a knowledge?

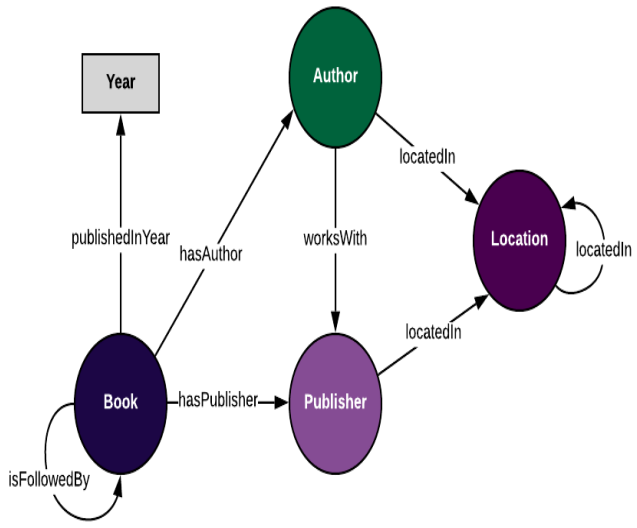
Knowledge is understanding of subject area. It includes Concepts and facts, properties of concepts, relationship between concepts, mechanism for how to combine concepts to solve problems.

4.1 Ontological engineering - representing abstract concepts (e.g. actions, time, physical objects, and beliefs that occur in different domains. Representing this abstract concept is sometimes called ontological engineering.

What is the need of ontological Engineering is when the agent has been assigned to an environment, it has lot of objects to deal with, it is not possible to represent each and every object the agent sees, instead we can give overall idea of the domain so that agents can learn the environment and reason about every action.

When we combine our classes and relationships, we can view our ontology in a graph format:

For Example.



Remember that our ontology is a general data model, meaning that we don't want to include information about *specific* books in our ontology. Instead, we want to create a reusable framework we could use to describe additional books in the future.

When we combine our classes and relationships, we can view our ontology in a graph format:

Dr. Rajeshwari.J
Associate Professor
Dayananda Sagar College Engineerin

Two major characteristics of general-purpose ontologies that distinguish them from special-purpose ontologies:

- a general-purpose ontology can be applicable in any special purpose domain (with the addition of domain-specific axioms)
- Different areas of knowledge must be **unified** because reasoning and problem solving involves several areas simultaneously. Example the circuit works in time domain as well as the labour works in time domain.

However, we need to give the general format using ontological engineering.

4.2 Categories and Objects:

a) Knowledge representation requires organisation of objects into categories:

-Although interaction of an agent takes place at the level of individual objects, much reasoning takes place at the level of categories. For example, a shopper would normally have the goal of buying a basketball, rather than a particular basketball such as BB9.

- Categories also serve to make predictions about objects once they are classified.

There are two choices for representing categories in first-order logic: predicates and objects.

a) Object is a member of category(reify)

We could then say Member (b, Basketballs), which we will abbreviate as b∈ Basketballs, to say that b is a member of the category of basketballs.

b) Category is a subclass of another category

We say Subset (Basketballs, Balls), abbreviated as Basketballs ⊂ Balls, to say that Basketballs is SUBCATEGORY a subcategory of Balls.

c) Categories serve to organize and simplify the knowledge base through inheritance.

If we say that all instances of the category Food are edible, and if we assert that Fruit is a subclass of Food and Apples is a subclass of Fruit, then we can infer that every apple is edible. We say that the individual apples inherit the property of edibility,

Dr. Rajeshwari.J
Associate Professor
Dayananda Sagar College Engineerin

d) All members of a category have some properties.

(x∈ Basketballs) ⇒ Spherical(x)

e) Members of a category can be recognized by some properties.

f)A category as a whole has some properties.

Dogs ∈ DomesticatedSpecies

fruits∈ Eatabes

Relationship between categories:

We say that two or more categories are DISJOINT disjoint if they have no members in common. Disjoint({Animals, Vegetables})

-Exhaustive decomposition: All the members of set s are covered by category c

Example: ExhaustiveDecomposition({Americans, Canadians , Mexicans}, NorthAmericans)

- A PARTITION disjoint exhaustive decomposition is known as a partition

(Note that the Exhaustive Decomposition of North Americans is not a Partition, because some people have dual citizenship.)

Partition({Males, Females}, Animals) is an Exhaustive Decomposition

-Categories can also be defined by providing necessary and sufficient conditions for membership. For example, a bachelor is an unmarried adult male:

x∈ Bachelors ⇔ Unmarried(x) ∧ x∈ Adults ∧ x∈ Males.

Natural kinds

Some categories have strict definitions: an object is a triangle if and only if it is a polygon with three sides. On the other hand, most categories in the real world have no clear-cut definition; these are called **natural kind** categories. For example, tomatoes tend to be a dull scarlet; roughly spherical; with an indentation at the top. There is, however, variation: some

Dr. Rajeshwari.J
Associate Professor
Dayananda Sagar College Engineerin

tomatoes are yellow or orange, unripe tomatoes are green, some are smaller or larger than average, and cherry tomatoes are uniformly small. Rather than having a complete definition of tomatoes, we have a set of features that serves to identify objects that are clearly typical tomatoes, but might not be able to decide for other objects.
This poses a problem for a logical agent. The agent cannot be sure that an object it has perceived is a tomato, and even if it were sure.

One useful approach is to separate what is true of all instances of a category from what is true only of typical instances. So in addition to the category Tomatoes , we will also have the category Typical (Tomatoes). Here, the Typical function maps a category to the subclass that contains only typical instances:
Typical (c) ⊆ c .
Most knowledge about natural kinds will actually be about their typical instances:
x∈ Typical (Tomatoes) ⇒ Red (x) ∧ Round (x) .

Thus, we can write down useful facts about categories without exact definitions

b**) Physical composition**

One object can be part of another object.

Eg Petal is a part of flower.

Objects can be grouped into part of hierarchies, reminiscent of the Subset hierarchy:

- PartOf (Bucharest , Romania)
- PartOf (Romania, EasternEurope)
- PartOf (EasternEurope, Europe)
- PartOf (Europe, Earth)

The PartOf relation is transitive and reflexive; that is,

PartOf (x, y) ∧ PartOf (y, z) ⇒ PartOf (x, z)

PartOf (x, x) .

Therefore, we can conclude PartOf (Bucharest , Earth).

An object is composed of the parts in its PartPartition and can be viewed as deriving some properties from those parts. For example, the mass of a composition object is the sum of the masses of the parts.

Dr. Rajeshwari.J
Associate Professor
Dayananda Sagar College Engineerin

It is also useful to define composite objects with definite parts but no particular structure. For example, if

the apples are Apple1, Apple2, and Apple3, then
BunchOf ({Apple1,Apple2,Apple3})

denotes the composite object with the three apples as parts (not elements).

We can define BunchOf in terms of the PartOf relation. Obviously, each element of s is part of BunchOf (s):

$\forall x\ x \in s \Rightarrow PartOf(x, BunchOf(s))$.

c) Measurements

a)Objects have height, mass, cost and so on. The values that we assign for these properties are called **measures**. Thus, the same length has different names in our language. We represent the length with a **units function** that takes a number as argument.

If the line segment is called L1, we can write
Length(L1)=Inches(1.5)=Centimeters(3.81)

Conversion between units is done by equating multiples of one unit to another: $Centimeters(2.54 \times d)=Inches(d)$.

b)Simple, quantitative measures are easy to represent. Other measures present more of a problem, because they have no agreed scale of values. Exercises have difficulty, desserts have deliciousness, and poems have beauty, yet numbers cannot be assigned to these qualities

The most important aspect of measures is not the particular numerical values, but the fact that measures can be ordered.

Although measures are not numbers, we can still compare them, using an ordering symbol such as >. For example, we might well believe that Norvig's exercises are tougher than Russell's, and that one scores less on tougher exercises:

$e1 \in Exercises \wedge e2 \in Exercises \wedge Wrote(Norvig, e1) \wedge Wrote(Russell, e2) \Rightarrow Difficulty(e1) > Difficulty(e2)$.

$e1 \in Exercises \wedge e2 \in Exercises \wedge Difficulty(e1) > Difficulty(e2) \Rightarrow ExpectedScore(e1) < ExpectedScore(e2)$.

Dr. Rajeshwari.J
Associate Professor
Dayananda Sagar College Engineerin

This is enough to allow one to decide which exercises to do, even though no numerical values for difficulty were ever used.

d) Objects: Things and stuff

Stuff is a composition of atomic particles where there is no clear cut distinction as individuation. Individuation—division into distinct objects.

- Scoop of ice cream forms things. Ice-cream as such form stuff.
- b∈ Butter ∧ Part Of (p, b) ⇒ p ∈Butter .

- We can now say that butter melts at around 30 degrees centigrade:
- b∈ Butter ⇒ MeltingPoint(b,Centigrade(30)) .

Some objects have intrinsic and extrinsic properties:

Some properties are intrinsic: they belong to the very substance of the object, rather than to the object as a whole. extrinsic properties—weight, length, shape, and so on.

4.3 EVENTS:

Situation calculus considers events to happen one at discrete time with actions and effects. It evaluates for action and effects one at a time. What is the state of the object before the event and what is the state after the effect.
E.g. Filling the bucket. Initial state will be empty bucket and final after filling the bucket with water. In situation calculus we are not dealing with the intermediate state.

Event calculus which is based on points of time rather than on situations.

- It deals with what is the state of object at particular time.
- Event calculus reifies fluents and events.
- The fluent At (Shankar, Berkeley) is an object that refers to the fact of Shankar being in Berkeley, but does not by itself say anything about whether it is true.
- To assert that a fluent is actually true at some point in time we use the predicate T, as in T (At (Shankar, Berkeley), t).
- Events are described as instances of event categories. The event E1 of Shankar flying from San Francisco to Washington, D.C. is described as
  E1 ∈ Flyings ∧ Flyer (E1, Shankar) ∧ Origin(E1, SF) ∧ Destination(E1,DC).
- We then use Happens(E1, i) to say that the event E1 took place over the time interval i.

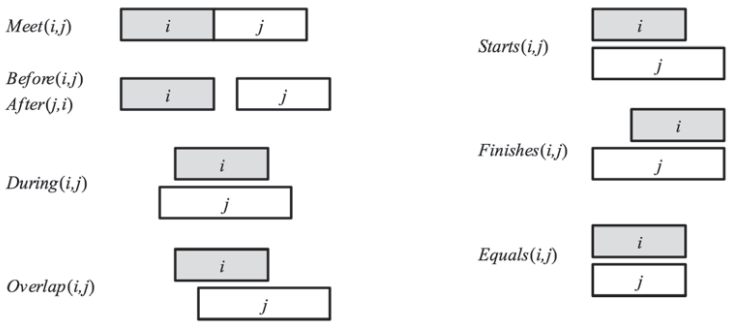
Dr. Rajeshwari.J
Associate Professor
Dayananda Sagar College Engineerin

a) Processes:

Any process e that happens over an interval also happens over any subinterval.
(e ∈ Processes) ∧ Happens(e, (t1, t4)) ∧ (t1 < t2 < t3 < t4) ⇒ Happens(e, (t2, t3)) .
Categories of events with this property are called process categories or liquid event categories.
If we resolve the above equation event e happens at t1 and t4 and t is in between t1 and t4, then event e happens at t2 and t3 also.

b) **Time intervals:**

- We will consider two kinds of time intervals: moments and extended intervals. The distinction is that only moments have zero duration:
  i∈Moments ⇔ Duration(i)=Seconds(0) .
  The function Duration gives the difference between the end time and the start time. Interval (i) ⇒ Duration(i)=(Time(End(i)) − Time(Begin(i))) .
  Date, which takes six arguments (hours, minutes, seconds, day, month, and year) and returns a time point:
  Time Interval relations:
  Meet(i, j) ⇔ End(i)=Begin(j)
  Before(i, j) ⇔ End(i) < Begin(j)
  After (j, i) ⇔ Before(i, j)
  During(i, j) ⇔ Begin(j) < Begin(i) < End(i) < End(j)
  Overlap(i, j) ⇔ Begin(i) < Begin(j) < End(i) < End(j)
  Begins(i, j) ⇔ Begin(i) = Begin(j)
  Finishes(i, j) ⇔ End(i) = End(j)
  Equals(i, j) ⇔ Begin(i) = Begin(j) ∧ End(i) = End(j)



4.4 MENTAL EVENTS AND MENTAL OBJECTS:

- Knowledge about one's own knowledge and reasoning processes is useful for controlling the inference.

Dr. Rajeshwari.J
Associate Professor
Dayananda Sagar College Engineerin

- The agent should know what are in its knowledge base and what are not
- Agent should have knowledge about beliefs or about deduction. Knowledge about one's own knowledge and reasoning processes is useful for controlling inference.
- **propositional attitudes** - the attitude an agent has towards mental objects (e.g. Believes, Knows, Wants, Intends, and Informs).

For example, to assert that Lois knows that Superman can fly:
Knows(Lois, CanFly(Superman))

one issue, is that if "Superman is Clark is true", then the inferential rules concludes that "Lois knows that Clark can fly" (this is an example of referential transparency).

But in reality Lois doesn't actually know that clark can fly.

- Superman is a clark
  Superman = Clark

(Superman = Clark) and (Knows(Lois, CanFly(Superman)) ⟹ Knows(Lois, CanFly(Clark))

Referential transparency

- an expression always evaluates to the same result in any context
- e.g. if agent knows that 2+2=4 and 4<5, then agent should know that 2+2<5
- built into inferential rules of most formal logic languages

Modal LOGIC

A modality is a 'mode of truth' of a proposition: about when that proposition is true, or in what way, or under which circumstances it would, could, or might have been true.

Designed to allow **referential opacity** into knowledge base.

Modal logic includes **modal operators** that takes sentences (rather than terms) as arguments (e.g. "A knows P" is represented with notation $\mathbf{K}_A P$ where **K** is the **modal operator** for knowledge, A is the agent, and P is a sentence).

syntax of modal logic is the same as first-order logic, with the addition that sentences can also be formed with **modal operators**

Dr. Rajeshwari.J
Associate Professor
Dayananda Sagar College Engineerin

semantics of modal logic is more complicated. In first-order logic a model contains a set of objects and an interpretation that maps each name to the appropriate object, relation, or function. In modal logic we want to be able to consider both the possibility that Superman's secret identity is Clark and that it isn't. Therefore, in modal logic a model consists of a collection of **possible worlds** (instead of 1 true world). The worlds are connected in a graph by **accessibility relations** (one relation for each modal operator). We say that world w_1 is accessible from world w_0 with respect to the modal operator $\mathbf{K}_A$ if everything in w_1 is consistent with what A knows in w_0, and we write this as Acc($\mathbf{K}_A$, w_0, w_1)
A knowledge atom $\mathbf{K}_A P$ is true in world w if and only if P is true in every world accessible from w. The truth of more complex sentences is derived by recursive application of this rule and the normal rules of first-order logic. That means that modal logic can be used to reason about nested knowledge sentences: what one agent knows about another agent's knowledge

.

Modal Logic's Tricky Interplay of Quantifiers and Knowledge

for example, the English sentence "Bond knows that someone is a spy" is ambiguous.

The first reading is that "there is a particular someone who Bond knows is a spy":

$\exists x\ \mathbf{K}_{Bond} Spy(x)$

which in modal logic means that there is an x that, in all accessible worlds, Bonds knows the same x to be a spy

The second reading is that "Bond just knows that there is at least one spy":

$\mathbf{K}_{\text{Bond}} \exists x\ Spy(x)$

which in modal logic means that in each accessible world there is an x that is a spy, but it need not be the same x in each world

Modal Logic - Writing Axioms With Modal Operators

assert that agents are able to draw deductions; "if an agent knows P and knows that P implies Q, then the agent knows Q":

$(\mathbf{K}_a P \wedge \mathbf{K}_a(P \Rightarrow Q)) \Rightarrow \mathbf{K}_a Q.$

4.5 Reasoning Systems for Categories

2 families of systems designed for organizing and reasoning with categories:

Dr. Rajeshwari.J
Associate Professor
Dayananda Sagar College Engineerin

- **semantic networks** - utilizing graphs and algorithms for inferring properties of an object on the basis of its category membership
- **description logics** - provides a formal language for constructing and combining category definitions and algorithms for deciding subset and superset relationships between categories

Semantic Networks

A typical graphical notation displays object or category names in ovals or boxes, and connects them with labelled links. For example, Figure 12.5 has a MemberOf link between Mary and FemalePersons, corresponding to the logical assertion Mary ∈ FemalePersons; similarly, the SisterOf link between Mary and John corresponds to the assertion SisterOf(Mary, John). We can connect categories using SubsetOf links, and so on. It is such fun drawing bubbles and arrows that one can get carried away. For example, we know that persons have female persons as mothers, so can we draw a HasMother link from Persons to FemalePersons? The answer is no, because HasMother is a relation between a person and his or her mother, and categories do not have mothers.

For this reason, we have used a special notation (the double-boxed link) in Figure 12.5. This link asserts that
$\forall x\ x \in Persons \Rightarrow [\forall y\ HasMother(x,y) \Rightarrow y \in FemalePersons]$

We might also want to assert that persons have two legs (the singly-boxed link) in Figure 12.5
$\forall x\ x \in Persons \Rightarrow Legs(x,2)$

As before, we need to be careful not to assert that a category has legs; the single-boxed link in Figure 12.5 is used to assert properties of every member of a category.

Dr. Rajeshwari.J
Associate Professor
Dayananda Sagar College Engineerin

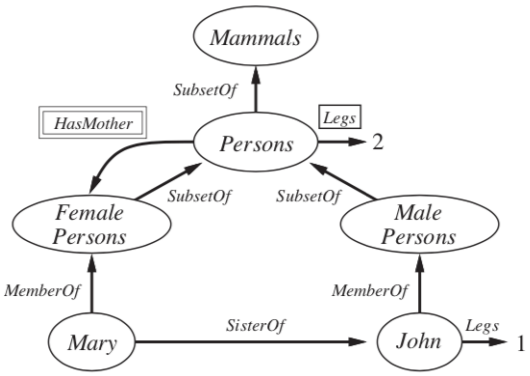


Figure 12.5 A semantic network with four objects (John, Mary, 1, and 2) and four categories. Relations are denoted by labeled links.

semantic network notation makes it convenient to perform inheritance reasoning. However, inheritance becomes complicated when an object can belong to more than one category or when a category can be a subset of more than one other category; this is called **multiple inheritance**

Semantic Networks - Default Values

semantic network also has the ability to represent **default values** for categories. we say the default value is **overridden** by the more specific value (e.g. figure 12.5 John has 1 leg even though he is a person and all persons have 2 legs)

Drawbacks of Semantic Networks

- one drawback of semantic network notation (compared to first-order logic) is that links between nodes/bubbles represent only binary relations.
  for example, to represent the following sentence in semantic networks is shown in figure 12.6 by reifying the proposition itself as an event belonging to an appropriate event category
  Fly(Shankar, NewYork, NewDelhi, Yesterday)

Dr. Rajeshwari.J
Associate Professor
Dayananda Sagar College Engineerin

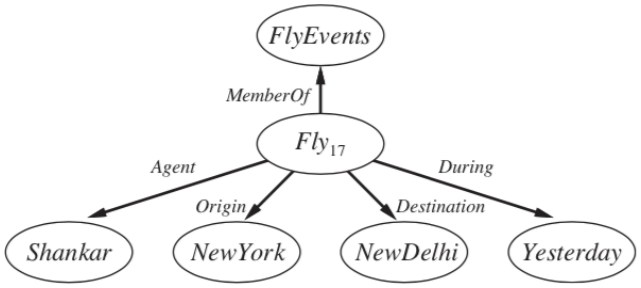


Figure 12.6 A fragment of a semantic network showing the representation of the logical assertion $Fly(Shankar, NewYork, NewDelhi, Yesterday)$.

- another drawback for this simple version of semantic networks, is that it still leaves out the full first-order logic such as representing negation, disjunction, nested function symbols, and existential quantification.

Description Logics

syntax of first-order logic makes it easy to say things about objects

description logics

- are notations that make it easier to describe definitions and properties of categories
- the principle inference tasks of description logics are:
  - **subsumption** - checking if one category is a subset of another by comparing their definition
  - **classification** - checking whether an object belongs to a category
  - **consistency** - some description logic system includes consistency of a category definition (whether the membership criteria are logically satisfiable)

the CLASSIC language (Borgida et al., 1989) is a typical description logic

syntax of CLASSIC descriptions shown in figure 12.7

Dr. Rajeshwari.J
Associate Professor
Dayananda Sagar College Engineerin

$$
\begin{aligned}
Concept &\rightarrow \textbf{Thing} \mid ConceptName \\
&\mid \textbf{And}(Concept, \ldots) \\
&\mid \textbf{All}(RoleName, Concept) \\
&\mid \textbf{AtLeast}(Integer, RoleName) \\
&\mid \textbf{AtMost}(Integer, RoleName) \\
&\mid \textbf{Fills}(RoleName, IndividualName, \ldots) \\
&\mid \textbf{SameAs}(Path, Path) \\
&\mid \textbf{OneOf}(IndividualName, \ldots) \\
Path &\rightarrow [RoleName, \ldots]
\end{aligned}
$$

Figure 12.7 The syntax of descriptions in a subset of the CLASSIC language.

for example, to say "bachelors are unmarried adult males":
Bachelor = And(Unmarried, Adult, Male).

the equivalent in first-order logic:
Bachelor(x) ⇔ Unmarried(x) ∧ Adult(x) ∧ Male(x)

any description in CLASSIC description logic can be translated into an equivalent first-order logic sentence. but some are more straightforward in CLASSIC description logic. for example, to describe the set of men with at least 3 sons who are all unemployed and married to doctors, and at most 2 daughters who are all professors in physics or math departments:
And(Man, AtLeast(3,Son), AtMost(2,Daughter),
All(Son, And(Unemployed, Married, All(Spouse, Doctor))),
All(Daughter, And(Professor, Fills(Department, Physics, Math))))

Note: Logical entailment: *The relation between a sentence and another sentence that follows from it.*
Mathematical notation: $\alpha \models \beta$: αentails the sentence β.
Formal definition of entailment:
$\alpha \models \beta$ if and only if $M(\alpha) \subseteq M(\beta)$
i.e. $\alpha \models \beta$ if and only if, in every model in which $\alpha$is true, β is also true.

Monotonic Reasoning:

In monotonic reasoning, once the conclusion is taken, then it will remain the same even if we add some other information to existing information in our knowledge base. To solve monotonic problems, we can derive the valid conclusion from the available facts only, and it will not be affected by new facts.

Dr. Rajeshwari.J
Associate Professor
Dayananda Sagar College Engineerin

Example:

- **Earth revolves around the Sun.**

It is a true fact, and it cannot be changed even if we add another sentence in knowledge base like, "The moon revolves around the earth".

Non-monotonic Reasoning

In Non-monotonic reasoning, some conclusions may be invalidated if we add some more information to our knowledge base.

Logic will be said as non-monotonic if some conclusions can be invalidated by adding more knowledge into our knowledge base.

Example: Let suppose the knowledge base contains the following knowledge:

- **Birds can fly**
- **Penguins cannot fly**
- **Pitty is a bird**

So from the above sentences, we can conclude that **Pitty can fly**.

However, if we add one another sentence into knowledge base "**Pitty is a penguin**", which concludes "**Pitty cannot fly**", so it invalidates the above conclusion.

4.6 Reasoning with Default Information

2 types of non-monotonic logics:

- **circumscription**
- **default logic**

Circumscription allows us to formalize non-monotonic reasoning directly in the language of classic logic. It is always the task of the user to specify which predicates to be minimized.

Dr. Rajeshwari.J
Associate Professor
Dayananda Sagar College Engineerin

To assume we have a predicate Ab that taking care of all the exceptional or abnormal cases where the default should not apply.

For instance, instead of saying that all birds fly we might

say: **∀x[Bird(x) ∧ ¬Ab(x) → Flies(x)]**

Meaning that all birds that are not in some way abnormal fly, that all normal birds fly.

Now imagine we have this fact in knowledge Base KB along with these facts:

KB=Bird(chilly), Bird(tweety), (tweety ≠ chily), ¬Flies(chilly)

now we would like to conclude that Tweety flies whereas Chilly doesn't.

But how we are concluding?

P*ROOF*: knowledge base doesn't have KB|≠ Flies(tweety). But in the previous example we saw that chilly is an abnormal bird but we have no such information about tweety.

Therefore in our last example we include chily but exclude tweety (in our extension), this is called circumscribing the predicate Ab. And the technique called circumscription

The minimal extension of predicate Ab in our example is:

- **∀x[x ≠ chilly→¬Ab(x)],**
- **∀x[Bird(x) ∧ x ≠ chilly →Flies(x)].**

Default Logic

default logic is a formalism in which default rules can be written to generate contingent, nonmonotonic conclusions

a default rule looks like this:

Bird(x) : Flies(x) / Flies(x)

this rules means that if *Bird(x)* is true, and if *Flies(x)* is ***consistent*** (causing no contradiction) with knowledge base, then *Flies(x)* may be concluded by default.

default rule has the form:

Dr. Rajeshwari.J
Associate Professor
Dayananda Sagar College Engineerin

P: $J_1, ..., J_n / C$

where:

- P is called the prerequisite
- C is the conclusion
- J_i are the justifications (if any one of them is proven false, then conclusion cannot be drawn).

Truth Maintenance Systems

Inferences drawn by the knowledge representation system will have only default status, rather than being absolutely certain. Inevitably, some of these inferred "facts" will turn out to be wrong and will have the be retracted(withdraw) in the face of new information. this process is called **belief revision**

The Difficulty of Belief Revisioning

Suppose that a knowledge base *KB* contains a sentence *P (perhaps a default conclusion recorded by a forward-chaining algorithm, or perhaps just an incorrect assertion*) and we want to execute TELL(KB, ¬P). To avoid creating a contradiction, we must first execute RETRACT(KB, P). This sounds easy enough. Problems arise, however, if any additional sentences were inferred from *P* and asserted(belief without any proof) in the *KB*. For example, the implication $P \Rightarrow Q$ might have been used to add *Q*. The obvious "solution"—retracting all sentences inferred from *P*—fails because such sentences may have other justifications besides *P*. For example, if *R* and $R \Rightarrow Q$ are also in the *KB*, then *Q* does not have to be removed after all.

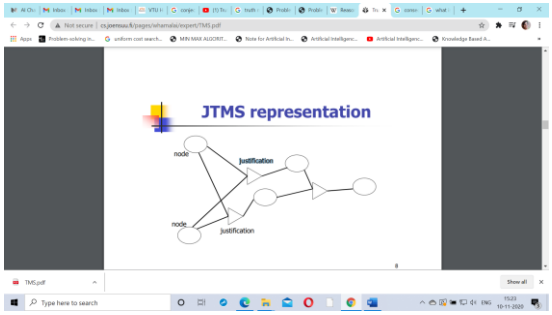
Truth Maintenance Systems (TMSs)

- are designed to handle the difficulty of belief revisioning
- types of TMSs:
  - justification-based truth maintenance system (JTMS)
  - assumption-based truth maintenance system (ATMS)

Justification-based truth maintenance system (JTMS)

- each sentence in the knowledge base is annotated with a **justification** consisting of the set of sentences from which it was inferred

Dr. Rajeshwari.J
Associate Professor
Dayananda Sagar College Engineerin



JTMS nodes

- each belief represented by a TMS node
- nodes are associated 1:1 with assertions
- the label of a node represents the belief status of the corresponding problem solver fact
- the relationships between beliefs are expressed by the justifications it participates n.

  - for example, *if the knowledge base already contains P ⇒ Q, then TELL(P) will cause Q to be added with the justification {P, P ⇒ Q}*
- a sentence can have any number of *justifications*
- justifications make retraction efficient. Given the call RETRACT(P), the JTMS will delete exactly those sentences for which P is a member of every justification
  - So:
    - if a sentence Q had the single justification {P, P ⇒ Q}, it would be removed
    - if it had the additional justification {P, P ∨ R ⇒ Q}, it would still be removed
    - but if it also had the justification {R, P ∨ R ⇒ Q}, then it would be spared
  - In this way, the time required for retraction of P depends only on the number of sentences derived from P rather than on the number of other sentences added since P entered the knowledge base.
- A truth maintenance system maintains consistency between old believed knowledge and current believed knowledge in the knowledge base (KB) through revision. If the current believed statements contradict the knowledge in the KB, then the KB is updated with the new knowledge. It may happen that the same data will again be believed, and the previous knowledge will be required in the KB. If the previous data are not present, but may be required for new inference. But if the previous knowledge was in the KB, then no retracing of the same knowledge is needed. The use of TMS

Dr. Rajeshwari.J
Associate Professor
Dayananda Sagar College Engineerin

avoids such retracing; it keeps track of the contradictory data with the help of a dependency record. This record reflects the retractions and additions which makes the inference engine (IE) aware of its current belief set.

- Each statement having at least one valid justification is made a part of the current belief set. When a contradiction is found, the statement(s) responsible for the contradiction are identified and the records are appropriately updated. This process is called dependency-directed backtracking.
- The TMS algorithm maintains the records in the form of a dependency network. Each node in the network is an entry in the KB
  -

assumption-based truth maintenance system (ATMS)

- Problem solvers need to explore multiple contexts at the same time, instead of a single one (the JTMS case)

4.7 The Internet Shopping World::

A Knowledge Engineering example:

An agent that helps a buyer to find product offers on the internet.

• IN = product description (precise or ¬precise)

• OUT = list of webpages that offer the product for sale.

• Environment = WWW

• Percepts = web pages (character strings)

• Extracting useful information required.

Shopping research agent that helps a buyer find product offers on the Internet. The shopping agent is given a product description by the buyer and has the task of producing a list of Web pages that offer such a product for sale, and ranking which offers are best.

The shopping agent's environment is the entire World Wide Web in its full complexity.

The agent's first task is to collect product offers that are relevant to a query. If the query is "laptops," then a Web page with a review of the latest high-end laptop would be relevant.

Dr. Rajeshwari.J
Associate Professor
Dayananda Sagar College Engineerin

a) Following links

The strategy is to start at the home page of an online store and consider all pages that can be reached by following relevant links. The agent will have knowledge of a number of stores,.

for example: Amazon ∈ OnlineStores ∧ Homepage(Amazon, "amazon.com") .

Ebay ∈ OnlineStores ∧ Homepage(Ebay, "ebay.com") .

ExampleStore ∈ OnlineStores ∧ Homepage(ExampleStore, "example.com") .

These stores classify their goods into product categories, and provide links to the major categories from their home page. Minor categories can be reached through a chain of relevant links, and eventually we will reach offers. In other words, a page is relevant to the query if it can be reached by a chain of zero or more relevant category links from a store's home page, and then from one more link to the product offer.

We can define relevance: Relevant(page, query) ⇔ ∃ store, home store ∈ OnlineStores ∧ Homepage(store, home) ∧ ∃ url, url2 RelevantChain(home, url2, query) ∧ LinkText(url2, url,text) ∧ page = Contents(url) .

First, we need to relate strings to the categories they name. This is done using the predicate Name(s, c), which says that string s is a name for category c—for example, we might assert that Name("laptops", LaptopComputers)

Here the predicate Link(from,to) means that there is a hyperlink from the from URL to the URL. To define what counts as a RelevantChain, we need to follow not just any old hyperlinks, but only those links whose associated anchor text indicates that the link is relevant to the product query. For this, we use LinkText(from,to,text) to mean that there is a link between from and to with text as the anchor text.

The existence of the chain itself is determined by a recursive definition, with the empty chain (start = end) as the base case: RelevantChain(start, end, query) ⇔ (start = end) ∨ (∃ u,text LinkText(start, u,text) ∧ RelevantCategoryName(query, text) ∧ RelevantChain(u, end, query)) .

Note: The text and query name the same category—e.g., "notebooks" and "laptops."

Name("laptops", LaptopComputers) Name("notebooks", LaptopComputers)

b) Comparing offers:

To compare those offers, the agent must extract the relevant information—price, speed, disk size, weight, and so on—from the offer pages. This can be a difficult task with real Web

Dr. Rajeshwari.J
Associate Professor
Dayananda Sagar College Engineerin

pages. A common WRAPPER way of dealing with this problem is to use programs called wrappers to extract information from a page.

we would like a wrapper to extract information such as the following: ∃ c, offer c∈ LaptopComputers ∧ offer ∈ProductOffers ∧ Manufacturer(c,IBM) ∧ Model(c, ThinkBook970) ∧ ScreenSize(c,Inches(14)) ∧ ScreenType(c, ColorLCD) ∧ MemorySize(c, Gigabytes(2)) ∧ CPUSpeed(c, GHz (1.2)) ∧ OfferedProduct(offer , c) ∧ Store(offer , GenStore) ∧ URL(offer , "example.com/computers/34356.html") ∧ Price(offer , $(399)) ∧ Date(offer , Today) .

The main point of this section is to show that some knowledge representation—in particular, the product hierarchy—is necessary for such an agent, and that once we have some knowledge in this form, the rest follows naturally.

1. **Reason** is the capacity for consciously making sense of things, applying logic, establishing and verifying facts, and changing or justifying practices, institutions, and beliefs based on new or existing information.
2. We use reasons or reasoning to form **inferences** which are basically conclusions drawn from propositions or assumptions that are supposed to be true.
3. **Deduction** is a general-to-specific form of reasoning that goes from known truths to specific instances. It starts with a hypothesis and examines the possibilities within that hypothesis to reach a conclusion. Example, in chemistry deducing a chemical equation basis two or more agents.
4. **Resolution** can be said to be anything which brings us to the truth of the knowledge. It is something which eventually is returned to the source and includes sources of knowledge and beings.