UNIT II

Knowledge Representation: Approaches and issues in knowledge representation- Propositional Logic –Predicate logic-Forward and backward reasoning - Unification- Resolution- Weak slot- filler structure – Strong slot-filler structure- Knowledge- Based Agent

# KNOWLEDGE REPRESENTATION

Let us first consider what kinds of knowledge might need to be represented in AI systems:

* 1. **Objects** -Facts about objects in our world domain. e.g. Guitars have strings, trumpets are brass instruments.
	2. **Events** - Actions that occur in our world. e.g. Steve Vai played the guitar in Frank Zappa's Band.
	3. **Performance** - A behavior like playing the guitar involves knowledge about how to do things.
	4. **Meta-knowledge**- Knowledge about what we know. e.g. Bobrow's Robot who plan's a trip. It knows that it can read street signs along the way to find out where it is.

# Mapping between facts and representations

Thus in solving problems in AI we must represent knowledge and there are two entities to deal with:

**Facts** - truths about the real world and what we represent. This can be regarded as the knowledge level.

**Representation** - which we manipulate. This can be regarded as the symbol level **of the facts** since we usually define the representation in terms of symbols that can be manipulated by programs.

We can structure these entities at two levels

* + **The knowledge level** - at which facts are described
	+ **The symbol level** - at which representations of objects are defined in terms of symbols that can be manipulated in programs



Figure: Mapping between facts and representations

English or natural language is an obvious way of representing and handling facts. Logic enables us to consider the following fact: spot is a dog as dog(spot) We could then infer that all dogs have tails with:

: dog(x) hasatail(x) We can then deduce:

hasatail(Spot)

Using an appropriate backward mapping function the English sentence Spot has a tail can be generated. The available functions are not always one to one but rather are many to many which is a characteristic of English representations.

The sentences All dogs have tails and every dog has a tail both say that each dog has a tail but the first could say that each dog has more than one tail try substituting teeth for tails. When an AI program manipulates the internal representation of facts these new representations should also be interpretable as new representations of facts.

# Intelligents agents should have following things

* + **Using Knowledge-** We have briefly mentioned where knowledge is used in AI systems. Let us consider a little further to what applications and how knowledge may be used.
	+ **Learning** – (Acquiring knowledge) This is more than simply adding new facts to a knowledge base. New data may have to be classified prior to storage for easy retrieval, etc. Interaction and inference with existing facts to avoid redundancy and replication in the knowledge and also so that facts can be updated.
	+ **Retrieval** - The representation scheme used can have a critical effect on the efficiency of the method. Humans are very good at it.
	+ **Reasoning** - Infer facts from existing data. If a system on only knows:

Miles Davis is a Jazz Musician.

All Jazz Musicians can play their instruments well.

If things like Is Miles Davis a Jazz Musician? or Can Jazz Musicians play their instruments well? are asked then the answer is readily obtained from the data structures and procedures.

However a question like Can Miles Davis play his instrument well? requires reasoning.

The above are all related. For example, it is fairly obvious that learning and reasoning involve retrieval etc.

The natural language reasoning requires inferring hidden state, namely, the intention of the speaker. When we say, "One of the wheel of the car is flat.", we know that it has three wheels left. Humans can cope with virtually infinite variety of utterances using a finite store of commonsense knowledge.

A logic consists of two parts, a language and a method of reasoning. The logical language, in turn, has two aspects, syntax and semantics. Thus, to specify or define a particular logic, one needs to specify three things:

* + **Syntax:** The atomic symbols of the logical language, and the rules for constructing well- formed, non-atomic expressions (symbol structures) of the logic.
	+ **Semantics:** The meanings of the atomic symbols of the logic, and the rules for determining the meanings of non-atomic expressions of the logic. It specifies what facts in the world a sentence refers to. Hence, also specifies how you assign a truth value to a sentence based on its meaning in the world.
	+ **Facts** are claims about the world that are True or False, whereas a **representation** is an expression (sentence) in some language that can be encoded in a computer program and stands for the objects and relations in the world.

There are a number of logical systems with different syntax and semantics. We list below a few.

* Propositional logic
* All objects described are fixed or unique

"John is a student" student(john) Here John refers to one unique person.

* First order predicate logic
* Objects described can be unique or variables to stand for a unique object "All students are poor" ForAll(S) [student(S) -> poor(S)]

Here S can be replaced by many different unique students. This makes programs much more compact:

eg. ForAll(A,B)[brother(A,B) -> brother (B,A)]

# APPROACHES TO KNOWLEDGE REPRESENTATION?

**Properties of a good system**

The following properties should be possessed by a knowledge representation system.

* **Representational Adequacy**- the ability to represent the required knowledge.
* **Inferential Adequacy** - the ability to manipulate the knowledge represented to produce new knowledge corresponding to that inferred from the original
* **Inferential Efficiency** - the ability to direct the inferential mechanisms into the most productive directions by storing appropriate guides;
* **Acquisitional Efficiency** - the ability to acquire new knowledge using automatic methods wherever possible rather than reliance on human intervention.

# Simple relational knowledge

The simplest way of storing facts is to use a relational method where each fact about a set of objects is set out systematically in columns. This representation gives little opportunity for inference, but it can be used as the knowledge basis for inference engines.

* + Simple way to store facts.
	+ Each fact about a set of objects is set out systematically in columns.
	+ Little opportunity for inference.
	+ Knowledge basis for inference engines.



**Figure:** Simple Relational Knowledge

We can ask things like:

* + Who is dead?
	+ Who plays Jazz/Trumpet etc.?

This sort of representation is popular in database systems.

# Inheritable knowledge

Relational knowledge is made up of objects consisting of

* + Attributes
	+ Corresponding associated values.

We extend the base more by allowing inference mechanisms:

* + Property inheritance
		- Elements inherit values from being members of a class.
		- Data must be organized into a hierarchy of classes.

o

Fig. Property Inheritance Hierarchy

* + Boxed nodes -- objects and values of attributes of objects.
	+ Values can be objects with attributes and so on.
	+ Arrows -- point from object to its value.
	+ This structure is known as a slot and filler structure, semantic network or a collection of frames.

**Algorithm: Property Inheritance** (to retrieve a value for an attribute of an instance object)

1. Find the object in the knowledge base
2. If there is a value for the attribute report it
3. Otherwise look for a value of instance if none fail
4. Otherwise go to that node and find a value for the attribute and then report it
5. Otherwise search through using isa until a value is found for the attribute.

# Inferential Knowledge

Represent knowledge as formal logic:

All dogs have tails: : dog(x) hasatail(x)

# Advantages:

* + A set of strict rules.
		- Can be used to derive more facts.
		- Truths of new statements can be verified.
		- Guaranteed correctness.
	+ Many inference procedures available to in implement standard rules of logic.
	+ Popular in AI systems. e.g Automated theorem proving.

# Procedural Knowledge Basic idea:

* + Knowledge encoded in some procedures
		- small programs that know how to do specific things, how to proceed.
		- e.g a parser in a natural language understander has the knowledge that a noun phrase may contain articles, adjectives and nouns. It is represented by calls to routines that know how to process articles, adjectives and nouns.

# Advantages:

1. Heuristic or domain specific knowledge can be represented.
2. Extended logical inferences, such as default reasoning facilitated.
3. Side effects of actions may be modelled. Some rules may become false in time. Keeping track of this in large systems may be tricky.

# Disadvantages:

1. Completeness - not all cases may be represented.
2. Consistency - not all deductions may be correct. e.g If we know that Fred is a bird we might deduce that Fred can fly. Later we might discover that Fred is an emu.
3. Modularity is sacrificed. Changes in knowledge base might have far-reaching effects.
4. Cumbersome control information.

# ISSUES IN KNOWLEDGE REPRESENTATION Overall issues

Below are listed issues that should be raised when using a knowledge representation technique:

* 1. Are any attributes of objects so basic that they occur in almost every problem domain?
	2. Are there any important relationships that exist among attributes of objects?
	3. At what level should knowledge be represented? Is there a good set of primitives into which all knowledge can be broken down?
	4. How should sets of objects be represented?
	5. Given a large amount of knowledge stored in a database, how can relevant parts be accessed when they are needed?

We will see each of these questions briefly in the next five sections.

# Important Attributes

Are there any attributes that occur in many different types of problem? There are two instance and isa and each is important because each supports property inheritance.

# Relationships among Attributes:

The attributes that we use to describe objects are themselves entities that we represent. What properties do they have independent of the specific knowledge they encode? There are four such properties that deserve are mentioned below.

* + - 1. Inverses.
			2. Existence in an isa hierarchy.
			3. Techniques for reasoning about values.
			4. Single valued attributes.

# Inverses

What about the relationship between the attributes of an object, such as, inverses, existence, techniques for reasoning about values and single valued attributes. We can consider an example of an inverse in

band(John Zorn,Naked City)

This can be treated as John Zorn plays in the band Naked City or John Zorn's band is Naked City.

Another representation is band = Naked City

band-members = John Zorn, Bill Frissell, Fred Frith, Joey Barron,

# Existence in an isa hierarchy:

Just as there are classes of objects and specialized subsets of those classes, there are attributes and specialization of attributes. Consider for example: the attribute height. In the case of attributes they support inheriting information about such things as constraints on the values that the attribute can have and mechanisms for computing those values.

# Techniques for reasoning about values:

Sometimes values of attributes are specified explicitly when a knowledge base is created. Several kinds of information can play a role in this reasoning including:

* + - 1. Information about the type of value- for (eg): the value of height must be a number measure in a unit of length.
			2. Constraints on the value, often stated in terms of related entities- for (eg): the age of the person cannot be greater than the age of either of that person’s parents.
			3. Rules for computing the values when it is needed.
			4. Rules that describe actions that should be taken if a value ever becomes known.

# Single valued attributes :

A specific but very useful kind of attribute is one that is guaranteed to take a unique value. For example: a baseball player can, at any one time, have only a single height and be a member of only one team.

# Choosing the granularity of representation:

At what level should the knowledge be represented and what are the primitives. Choosing the Granularity of Representation Primitives are fundamental concepts such as holding, seeing, playing and as English is a very rich language with over half a million words it is clear we will find difficulty in deciding upon which words to choose as our primitives in a series of situations. If Tom feeds a dog then it could become:

feeds(tom, dog)

If Tom gives the dog a bone like:

gives(tom, dog,bone) Are these the same?

In any sense does giving an object food constitute feeding? If give(x, food) feed(x) then we are making progress.

But we need to add certain inferential rules.

In the famous program on relationships Louise is Bill's cousin How do we represent this? louise

= daughter (brother or sister (father or mother( bill))) Suppose it is Chris then we do not know if it is Chris as a male or female and then son applies as well.

Clearly the separate levels of understanding require different levels of primitives and these need many rules to link together apparently similar primitives.

Obviously there is a potential storage problem and the underlying question must be what level of comprehension is needed.

# Representing set of objects:

It is important to be able to represent sets of objects for several reasons. One is that there are some properties that are true of sets that are not true of the individual members of a set.

**Example:** Consider the assertions that are being made in the sentences “There are more sheep than people in Australia” and “English speakers can be found all over the world.” The only way to represent the facts described in these sentences is to attach assertions to the sets representing people, sheep, and English speakers, since, for example, no single English speaker

can be found all over the world. The other reason that it is important to be able to represent sets of objects is that if a property is true of all elements of a set, then it is more efficient to associate objects is that if a property is true of all elements of a set.

# Finding the right structure as needed:

In order to have access to the right structure for describing a particular situation, it is necessary to solve all of the following problems.

* How to perform an initial selection of the most appropriate structure.
* How to fill in appropriate details from the current situation.
* How to find a better structure if the one chosen initially turns out not to be appropriate.
* What to do if none of the available structures is appropriate.
* When to create and remember a new structure.

# Selecting an initial structure

The selecting candidate knowledge structures to match a particular problem solving situation is a hard problem, there are several ways in which it can be done. Three important approaches are the following.

* Index the structures directly by the significant English words that can be used to describe them.
* Consider each major concept as a pointer to all of the structures in which it might be involved.
* Locate one major clue in the problem description and use it to select an initial structure.

# Revising the choice when necessary

Once the candidate knowledge structure is detected, we must attempt to do a detailed match of it to the problem at hand. Depending on the representation we are using the details of the matching process will vary.

When the process runs into a snag, though, it is often not necessary to abandon the effort and start over. Rather there are a variety of things that can be done. The following things can be done:

* Select the fragments of the current structure that do correspond to the situation and match them against candidate alternatives.
* Make an excuse for the current structure's failure and continue to use it.
* Refer to specific stored links between structures to suggest new directions in which to explore.

# PROPOSITIONAL LOGIC

**Logic:** The logic plays an important role in the design of almost all the systems in engineering and sciences. Designing the present days computer is complex task. This design involves two types of design namely

* 1. Hardware design
	2. Software design

These are based on mathematical logic called formal logic.

**Propositional logic:** The propositional logic deals with individual propositions, which are viewed as atoms, i.e these cannot be further broken into smaller constituents. Though propositional logic is not powerful than predicate logic but it has great importance in number of applications, particularly in the design of computers at hardware level.

For building a propostional logic, first we describe the logic with the help of a formula called Well-Formed Formula (wff). The propositional logic contains variables such as p, q, r, s, t, p1, p2, p3, q1, q2, q3, r1, r2,r3 etc…

The other symbols of propositional logic are

|  |  |  |
| --- | --- | --- |
| *¬p* | (read "not *p*") | the **negation** of *p* |
| *p∧q* | (read "*p* **and** *q*") | the **conjunction**of *p* and *q* |
| *p∨q* | (read "*p* **or** *q*") | the **disjunction**of *p* and *q* |
| *p→q* | (read "*p* **implies** *q*") | the **implication**of *q* from *p* |
| *p←q* | (read "*p* **if** *q*") | the **implication**of *p* from *q* |
| *p ↔q* | (read "*p* **if and only if** *q*" or "***p*****is equivalent to** *q*") |  |

The examples of propositions are given below. P= rama is student of second year.

Q= rama participates in tennis.

The following formulas can be constructed using above.

*p∧q= rama is student of second year* ***and*** *rama participates in tennis.*

*p∨q= rama is student of second year* ***or*** *rama participates in tennis.*

*¬p∧q= rama is not student of second year* ***and*** *rama participates in tennis.*

*¬p→q= if rama is not student of second year* ***then*** *rama participates in tennis.*

Note that we only talk about the truth value in an interpretation. Propositions may have different truth values in different interpretations.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| *p* | *q* | *¬p* | *p ∧q* | *p∨q* | *p ←q* | *p →q* | *p ↔q* |
| *True* | *True* | *false* | *True* | *true* | *true* | *True* | *true* |
| *True* | *False* | *false* | *False* | *true* | *true* | *False* | *false* |
| *False* | *True* | *true* | *False* | *true* | *false* | *True* | *false* |
| *False* | *False* | *true* | *False* | *false* | *true* | *True* | *true* |

Figure Truth table defining *¬*, *∧*, *∨*, *←*, *→*, and *↔*

If an expression is true, for all the rows (i.e. for all possible values of variables in that expression) then it is called as tautology, and we write it as *u.*

*The following are some of laws and its equivalences.*

|  |  |  |
| --- | --- | --- |
| S.No | Equivalence | Name of the Equivalence |
| 1. | *¬p(p ∧q)= ¬p∨¬q**¬(p∨q)= ¬p∧¬q* | Demorgans Law |
| 2. | p*∧Tp=p**p∨Cp=p* | Identity laws |
| 3. | p*∧¬p=Cp**p∨¬p=Tp* | Inverse laws |
| 4. | p*∨Tp=Tp**p∧ Cp =Cp* | Domination Law |
| 5. | p*∨p=p* | Idempotent laws |

|  |  |  |
| --- | --- | --- |
|  | *p∧p=p* |  |
| 6. | *p →q=¬p∨q* | Implication laws |
| 7. | *P∨q=q∨p**p∧q=q∧p* | Commutative laws |
| 8. | *P∨(q∨r)=(p∨q) ∨r)**p∧(q∧r)=(p∧q) ∧r* | Associative laws |
| 9. | *P∨(q∧r)=(p∨r) ∧(p∨r)**P∧ (q∨r)=(p∧r) ∨ (p∧r)* | Distributive laws |

**Example1: show that (***p∧q) →(p∨q) is a Tautology, i.e interpretation of this sentence is always true.*

**Solution:** *it can be proved that above logical expression is a tautology using rules of logical equivalences.*

**(***p∧q) →(p∨q)= ¬(p∧q) ∨(p∨q) (using implication laws)*

*= (¬p∨¬q) ∨(p∨q)*

*= (¬p∨p) ∨(¬q∨p) (by rearrangement of terms)*

*= Tp∨ Tp (using inverse law)*

*= T*

**Example2: Show that (p***∨q) ∧¬(¬p ∧q) and p are logically equivalence. Solution:* (p*∨q) ∧¬(¬p ∧q)*

*=*(p*∨q) ∧(¬¬p ∨¬q)*

*=(p∨q) ∧(p∨¬q)*

*=p∨(q ∧¬q)*

*=p∨F*

*=p*

# Do the following problems for practice

* + 1. **Find out using truth table whether implication is tautology.**
			1. (p*∧r) →p*

b. (p*∧q) →(p →q)*

**c.** ((p*∨(¬(q∧r))) →((p ↔q) ∨r)*

# Show that tautology without using Truth table.

a. (p*∧(p →q)) →q*

**b.** (*¬p∧(p∨q)) →q*

# Verify whether following are tautology.

a. (*¬p∧(p→q)) →¬q*

**b.** (*¬q∧(p∨q)) →¬p*

# Show that pairs of expressions are logically equivalent.

* + - 1. *¬p ↔q and p ↔¬q*

**b.** *¬(p∧q) and (¬p) ∨(¬q)*

**c.** *¬p→¬q and q→p*

# Inference rules:

Inference rules are used to infer new knowledge in the form of propositions from the existing knowledge. The knowledge propositions are logical implications of the existing knowledge.

# Modus Ponens: This rule is also called rule of detachment. Symbolically it is written as [p*∧(p→q)] →q*

*Or*

*P*

*p→q*

*q ( p and p implies q can be written as q)*

# Modus Tollens: The inference rule of modus tollens is a logical implication specified by [(p*→q) ∧¬q] →¬p*

Which can be written as

p*→q*

*¬q*

*¬p (p implies q and negation q can be written as negation p)*

# Law of syllogism: This rule of inference is expressed by the logical implication:

*[*(p*→q) ∧(q→r)] →(p→r)*

It can also be expressed in tabular form as: p*→q*

*q→r*

p*→r (p implies q and q implies r can be written as p implies r)*

**Example:** Prove whether the following argument is valid, contradiction or satisfied.

“rajini is preparing food in kitchen. If rajini is preparing food in kitchen then she is not playing violin. If she is not playing violin, then she is not learning music. Therefore, rajini is not learning music.”

**Solution:** The above statements can be specified in the form of logical implication if the following propositional symbols are assigned to statements:

p= rajini is preparing food in kitchen q= rajini is playing violin

r= rajini is learning music

now, the argument can be expressed in the form of a propositional formula:

p*∧((p→ ¬q) ∧(¬q→¬r)] →¬r*

the above wff can also be represented in the tabular form along with inference as follows

p p*→¬q*

*¬q→¬r*

*¬r (p and p implies negation q and negation q implies negation r*

*can be as ¬r)*

*To prove this logical implication, we carry out following steps.*

* 1. p already specified as premise
	2. p*→¬q* already specified as premise
	3. *¬q→¬r* already specified as premise
	4. p*→¬r by law of syllogism using 2 and 3 above*
	5. *¬r by modus ponnens rule, using 1 and 4*

**d.Rule of Conjunction: This rule states that if p and q are individually true statements, then the composite p***∧q is a true statements, i.e*

*[p∧q] →p∧q*

or

p

q

*p∧q ( p and q can be wriiten as p∧q)*

# Rule of Disjunctive Syllogism: It id defined as a logical implication given as follows.

[(p*∨q) ∧¬p] →q*

# Or

p*∨q*

*¬p*

q (p*∨q* and negation p can be written as q)

# Rule of Contradiction: it is defined as a logical implication,

*(¬p →Cp) →p*

# Or

*¬p →Cp*

# p ( it implies always p)

1. **Rule of Conjunctive Simplification: This rule states that conjuction of p and q logically implies p, i.e**

*(p∧q) →p Or*

*p∧q*

*p ( it implies p)*

# Rule of Disjunctive Amplification: This rule states that p*∨q* can be inferred from p, and p*∨q* is logical consequence of p, it is expressed as

*p→( p∧q)*

# this above can be expressed as,

p

p*∨q*

# Rule of End elimination: This rule infers p from the wff *p∧q, i.e*

*(p∧q*

p

# Rule of proof cases: it is stated in tabular form

*p→r q→r*

*(p∨q) →r*

**Example:** *Prove or disprove the following arguments:*

*“if the auditorium was not available or there were examinations, then the music programme was postponed. If the music programme gets postponed, then a new date was announced. No new date was announced. Therefore, auditorium was available.”*

**Solution:** *Let us assume that following are symbols for the statements (propositions) in the above argument.*

*P= auditorium was available Q= there were examination*

*R= music programme was postponed S= new date was announced*

*The statements can be expressed in the form of logical expressions given as follows.*

*(¬p∨q) →r*

*r→s*

*¬s*

p ( all 3 premise can be written as p) the logical implication for the above expression can be expressed as follows:

((*(¬p∨q) →r) ∧( r→s) ∧¬s) →p*

*Validity of the above arguments can be proved as follows.*

# Steps Inference Justification

1. *(¬p∨q) →r)* already specified as premise
2. *r→s* already specified as premise
3. *¬s* already specified as premise
4. *¬r by mobus tollens using 2,3*
5. *¬(¬p∨q) by 1 and 4 and modus tollens.*
6. *(p∧¬q) by 5, and de morgans’s rule*
7. *p by 6, and rule of end elimination.*

Thus, it is proved through various inference rules, which logically follow the premises, the argument as a whole is valid statement.

# 5 . PREDICATE LOGIC

**Basic idea:** The word “**predicate**” means to declare or affirm concerning the subject of a preposition. **For example**, in the sentence, “**He was a king**”, “**king**” is a predicate noun. Let us consider the following two statements represented as proposition.

# P= rama is a student, and Q= Krishna is a student.

Here, symbols p and q do not show anything common between them. However, the phrase “**is a student**” is Predicate, common in both sentences. In predicate logic these statements can be written as

# (First Level predicate)

*isstudent(Rama) and isstudent(Krishna)*

# (Second Level Predicate)

**(Third Level Predicate)**

*in addition it can be represented by*

*student(Rama) student(Krishna)*

**s***(Rama),*

s*(Krishna),*

*s(R),*

*s(K),*

the above representations in predicate form shows that there is some common feature is s(R)and s(K), because both have common predicate, i.e student. If all the students in a class are to be represented using this form, we use a variable for student name. Therefore, the statements, “x is a student” can be represented in predicate form as s(x).

**Predicate Formula:** A general form of predicate statement is,

P(a1, a2,… , an)

Where p is a predicate and a1, a2,……………..,an are terms. The predicate p(a1, a2,… , an) is

called atomic formula. A well formed formula (wff) defined in propositional calculus is also applicable in predicate calculus

|  |  |
| --- | --- |
| **For all** | tex2html_wrap_inline7176 |
| **There exists** | tex2html_wrap_inline7174 |
| **Implies** | tex2html_wrap_inline7156 |
| **Not** | tex2html_wrap_inline7182 |
| **Or** | tex2html_wrap_inline7184 |
| **And** | tex2html_wrap_inline7186 |

Connectives can be used in the predicate similar to those in propositions. Let us consider the sentences given below.

“Rama is a student ***and*** Rama plays cricket”. “Rama is a student ***or*** Rama plays cricket”.

“Rama is a student ***implies that*** Rama plays cricket”. “Rama is ***not*** student”.

These can be represented in the predicate forms in the same order as:

**s(R) *∧p(R,C)***

**s(R) *∨p(R,C)***

**s(R) *→p(R,C)***

***¬* s(R)**

In the above predicates, p(R,C) stands for “ Rama plays Cricket”, where p is predicate for “plays”, R for “Rama” is a subject and C for Cricket” is an object. P(R,C) is a two place predicate. Higher place predicates are also possible. Following are some of examples.

Rajan plays cricket and basketball = p(R,C,B).

# Functions:

The parameters a1, a2,….., an in a predicate p, given below, can be constants or variables or functions.

# P(a1,a2,…..,an)

Consider the following sentences:

“Rajan is father of Rohit.” “Sheela is mother of Rohit.” “Rajan and Sheela are spouse.”

Let the expressions – *fatherof(Rohit)*, and *motherof(Rohit)*, be functions and their values are

*“Rajan” and “Sheela”* respectively. Using above expressions, the predicate.

Spouse(Rajan, Sheela),

Can be written as

*spouse (fatherof*(Rohit*)*, *motherof*(Rohit)).

A function may have any number of objects, called arity of the functions. For example, if Rohit and Rajini are brother-sisters, then the functions.

Father of Rajni and Rohit, and Mother of Rajni and Rohit,

Can be written as,

# Example:

*fatherof* (Rajni, Rohit)=Rajan

*motherof* (Rajni, Rohit)=Sheela.

**“2 plus 2 is 4.”** Can be written as function formula as ***plus*(2, 2)=4**

**“50 divided by 10 is 5.”** Can be written as function formula as ***divided by*(50, 10)=5**

# REPRESENTING SIMPLE FACTS IN LOGIC:

We briefly mentioned how logic can be used to represent simple facts in the last lecture. Here we will highlight major principles involved in knowledge representation. In particular predicate logic will be met in other knowledge representation schemes and reasoning methods.

Symbols used the following standard logic symbols we use in this course are:

Let’s first explore the use of propositional logic as a way of representing the sort of world knowledge that an AI system might need. Propositional logic is appealing because it is simple to deal with and a decision procedure for it exists. Suppose we want to represent the obvious fact stated by the classical sentence.

It is raining.

RAINING

It is sunny

SUNNY

It is raining, then it is not sunny. RAINING SUNNY

Let’s now explore the use of predicate logic as a way of representing knowledge by looking at a

specific example. Consider the following set of sentences.

1. Marcus was a man
2. Marcus was a pompeian.
3. All Pompeians were romans
4. Caesar was a ruler.
5. All romans were either loyal to caesar or hated him.

The facts described by these sentences can be represented as a set of wff’s in predicate logic as follows:

1. Marcus was a man.

Man(Marcus)

This representation captures the critical fact of marcus being a man. It fails to capture some of the information in the english sentence, namely the notion of past tense.

1. Marcus was a Pompeian Pompeian(marcus)
2. All Pompeians were romans.

x: pompeians(x) *Roman(x)*

1. Caesar was a ruler. ruler(Casear)

Here we ignore the fact that proper names are often not references to unique individuals, since many people share the same name. Sometimes deciding which of several people of the same name being referred to in a particular statement may require a fair amount of knowledge and reasoning.

1. All romans were either loyal to caesar or hated him.

x:Roman(x) *loyalto(x, Caesar) V hate(x, Caesar)*

In English the word “or” sometimes means the logical inclusive or and sometimes means the logical exclusive or (XOR). Here we have used the inclusive interpretation. Some people argue however that this English sentence is really stating an, exclusive or. To express that, we would have to write.

x: roman(x) [(loyal to(x, Caesar) V hate(x, Caesar)) (loyalto(x, Caesar) hate(x, Caesar))]

# Variable and Quantifiers

To generalize the statement, “rama is student”, it is written as “x is student”, i.e s(X). if s(X) is true for a single case, then we say that the expression is satisfied.

Let us consider the following statements.

“x is human implies x is mortal.” “Socrates is human.”

When represented in predicate form, these become: h(x) ***→m(x), and***

## h(S)

the above two wffs have some resemblance to the premises required for the inference rule of modus ponens. To generalize the implication, the variable x applicable for the entire human domain is quantified using quantying operator  called universal quantifier. Above statements can be modified as follows after incorporating the effect of quantifiers.

“for all x, x is human implies that x is mortal”, and “Socrates is human”

Now, these are rewritten in symbolic form using quantifying operator  x (h(x) ***→m(x)), and***

## h(S)

*in this case the first statement x (h(x)* ***→m(x)) is true, when the statement is found to be true*** for entire range of x. because it says “for all x” or “for every x” or “for all possible values of x”, h(x) →m(x) is true. Still, it is not possible to infer m(S), i.e “mortal Socrates”. Because the statements still do not appear in the form such that the rule of modus ponens can be applied. The inference rule of universal instantiation, discussed in the next section, will help in resolving this problem.

# SYNTAX AND SEMANTICS of FOL:

Terms: First order logic has sentence, but it also has terms which represents objects. Constant symbols, variables and function symbols are used to build terms and quantifiers and predicate symbols are used to build sentences.

# SYNTAX of FOL in BNF (Backus – Naus Form):

Sentence  Atomic sentence

׀Sentence connective sentence

׀Quantifier variable,….sentence

׀7 sentence

׀(sentence).

Atomic sentence Predicate (Term,…) Term  Term

Term  Function (Term…)

׀constant

׀variable Connective  =>׀𝖠׀∨׀ Quantifier  ∀ ׀ ∃

Constant  𝐴׀ λcon׀ john׀

# QUANTIFIERS:

Quantifiers are used to express properties of entire collection of objects, rather than represent the object by names. FOL contains two standard quantifiers,

# Universal quantifier ( ) # Existential quantifier ( )

# UNIVERSAL QUANTIFIERS:

General Notation: “∀𝑋P” where,

P – Logical expression, X – Variable, - For all That is, P is true for all objects X in the Universe.

# Examples:

All cats are mammals => X cat (X)  mammals(X)

That is, all the cats in the universe belongs to the type of mammals and hence the variable X may be replaced by any of the cat name (object, Name)

# Examples:

Spot is a cat

Spot is a mammal Cat (spot) Mammal (spot)

Cat (spot)  mammal (spot) Spot – Name of the cat.

# Existential Quantifiers:

General Notification: X P, where

P – Logical Expression, X – Variable, - There exist That is P is true for some object X in the universe.

# Example:

Spot has a sister who is a cat. X sister (𝑋, 𝑠𝑝𝑜𝑡)  cat(X)

That is, the spot’s sister is a cat, implies spot is also a cat and hence X may be replaced by, sister of spot, if it exists.

# Example:

Felix is a cat.

Felix is a sister of spot Cat (Felix)

Sister (Felix, spot)

Sister (Felix, spot)  cat (Felix).

# NESTED QUANTIFIERS:

The sentences are represented using multiple quantifiers.

# Example:

# For all X and all Y, if x is the parent of Y then Y is the child of X. X, Y parent (X, Y)  child (Y, X).

# Everybody loves somebody

∀ 𝑋 ∃ Y loves (X, Y)

# There is someone who is loved by everyone Y X loves (X, Y)

Connection between and :

The two Quantifiers ( and ) are actually connected with each other through negation.

# Example:

Everyone likes ice cream

X likes (X, ice cream) is equivalent to 7 ∃ X 7 likes (X, ice cream) That is there is no one who does not like ice cream.

**Ground Term or Clause**: A term with no variable is called ground term.

Eg: cat (spot)

The De Morgan rules for quantified and unquantified sentences are as follows,

# # Quantified sentences:

X7P ≡ 7 XP

7 XP ≡ X7P XP ≡ 7∃ X 7P XP ≡ 7∀ X 7P

# #Unquantified sentence:

7(P𝖠Q) ≡ 7P ∨ 7Q

7P 𝖠 7Q ≡ 7(P∨Q) P 𝖠 Q ≡ 7(7P ∨ 7Q) P∨ Q ≡ 7(7P 𝖠 7Q)

**Inference rules :** All the inference rules applicable for propositional logic also apply for predicate logic. However, due to introduction of quantifiers, additional inference rules are there for the expressions using quantifiers. These are given below.

# Rule of universal instantiation

This rules states that if a universally quantified variable in a valid sentence is replaced by a term from the domain, then also the sentence is true. Thus is, if

*x (h(x) →m(x))*

Is true, and if x is replaced by “Socrates”. And quantifier is removed then the statement,

*(h(S) →m(S))*

*Is still true. This rule is called as rule of* universal instantiation and expressed as

***x P(x)***

# P(a) (for all x p(x), can be written as P(a) )

1. **Rule of universal generalization**

If a statement p(a) is true for each element a of universe, then the universal quanitifier may be prefixed, and *x P(x) can be inferred from p(a), i.e.*

# P(a), for all a Є U

***x P(x)*** *( (p(a) can be written as for all x, p(x))*

# Rule of Existential Instantiation

If x p(x) is true, and there is an element a in the universe of p, then we can infer p(a). i.e

# tex2html_wrap_inline7174x p(x)

**P(a) for some a** Є U **(There existx p(x), can be written as P(a))**

# Rule of Existential Generalization

**If p(a) is true for some a in the universe, then it can be inferred that **x p(x) is true, i.e

# P(a), for all a Є U

x ***P(x)*** *( (p(a) can be written as there exist x, p(x))*

# Example Problems:

*Prove the validity of the following statements*

*“All kings are men” (1)*

## Solution:

*“All men are falliable” (2)*

*“Therefore, all kings falliable”*

*The above statements could be rewritten without affecting the meanings they convey “For all x, x is king implies that x is man”*

*“For all y, y is man implies that y is falliable.” “Therefore, for all z, z is king implies that z is falliable.”*

*The above statements can be represented in the predicate form as follows.*

***x (k(x)*** *→m(x)) from (1)*

***y (m(y)*** *→f(y)) from(2)*

***z (k(z)*** *→f(z))*

*We can arrive to formal proof for the above using following steps.*

|  |  |  |
| --- | --- | --- |
|  | **STEPS** | **Justification** |
| *1.* | ***x (k(x)*** *→m(x))* | *Given the premise* |
| *2.* | k(a) *→m(a)* | *By rule of universal instantiation on (1)* |
| *3.* | ***y (m(y)*** *→f(y))* | *Given the premise* |
| *4.* | ***m(a)*** *→f(a)* | *By rule of universal instantiation on (3)* |
| *5.* | ***k(a)*** *→f(a)* | *By rule of syllogism using (2) and (4)* |
| *6.* | ***z (k(z)*** *→f(z))****Hence its proved*** | *By rule of universal generalization on (5)* |

# tex2html_wrap_inline7176 UNIFICATION

**Basic idea:** The require findings substitutions that make different logical expression. This provers is called unification and is a key component of all first order inference algorithms. The UNIFY algorithm takes two sentences and returns a unifier for them if one exists:

**Syntax**: Unify (P,Q) =  where SUBSET (,P)= SUBSET (, q)

Here are the results of unification with four different sentences that might be in knowledge base.

UNIFY ( knows (john,x), knows (john,jane)) = {x|jane} (1)

UNIFY ( knows (john,x), knows (y,bill)) = {x|bil,y|john} (2)

UNIFY ( knows (john,x), knows (y,mother(y))) = {y|john,x|mother(john)}……(3)

UNIFY ( knows (john,x), knows (x,Elizabeth)) = fail (4)

The last unification fails because x cannot take on the values john and Elizabeth at the sametime. Now remember that knows (x,Elizabeth) means “everyone knows Elizabeth”,

we should be able to infer that john knows Elizabeth. The problem arises only because the two sentences happen to use the same variable name, x. the problem can be avoided by standardizing apart one of the two sentences being unified, which means remaining its variable to avoid name clashes.

# p q θ

Knows(John,x) Knows(John,Jane) {x/Jane}} Knows(John,x) Knows(y,OJ) {x/OJ,y/John}}

Knows(John,x) Knows(y,Mother(y)) {y/John,x/Mother(John)}} Knows(John,x) Knows(x,OJ) {fail}

* Standardizing apart eliminates overlap of variables, e.g., Knows(z17,OJ) To unify *Knows(John,x)* and *Knows(y,z)*,

θ = {y/John, x/z } or θ = {y/John, x/John, z/John}

* The first unifier is more general than the second.

– There is a single most general unifier (MGU) that is unique up to renaming of variables.MGU = { y/John, x/z }

# Unification algorithm:



1. In propositional logic, it is easy to determine that two literals cannot both be true at the same time. Simply look for L and ¬L.
2. In predicate logic, this matching process is more complicated since the arguments of the predicates must be considered.

## For example: man(John) and ¬man(John) is a contradiction While

***man(John) and ¬man(Spot) is not***

thus in order to determine contradictions, we need a matching procedure that compares two literals and discovers whether there exists a set of substitutions that makes them identical. There is a straight forward recursive procedure, called the unification algorithm that does just in simple way.

**The basic idea of unification is very simple.:** To attempt to unify two literals, we first check if their initial predicate symbols are the same. If so, we can proceed. Otherwise, there is no way they can be unified regardless of their arguments. For example, the two literals.

# Try assassinate(Marcus, Caesar) hate(Marcus, Caesar)

Cannot be unified. If the predicate symbols match, then we must check the arguments, one pair at a time. If the first matches, we can continue with the second, and so on. To test each argument pair, we can simply call the unification procedure recursively.

# RULES:

The matching rules are simple. Different constants or predicates cannot match; identical ones can. A variable can match another variable, any constant, or a predicate expression, with the restriction that the predicate expression must not contain any instances of the variable being matched.

# Example:

**P(x,x) (1)**

# P(y,z) (2)

The two instances of P match fine. Next we compare x and y, and decide that if we substitute y for x, they could match. We will write that substitution as **x=y in (1)**

# y/.x

(We could, of course, have decided instead to substitute x for y, since they are both just dummy variable names. The algorithm will simply pick one of these two substitutions). But now, if we simply continue and match x and z, we produce the substitution z/x. but we cannot substitute both y and z for x, so we have not produced a consistent substitution.

What we can need to do after finding the first substitution y/x is to make that substitution y/x is to make that substitution throughout the literals, giving

# P(y,y)

**P(y,z)**

Now we can attempt to unify arguments y and z, which succeeds with the substitution z/y. The entire unification process has now succeeded with a substitution that is the composition of the two substitutions we found. We write the composition as **Y=z and x=y** unifications pass.

# (z/y) (y/x)

Following standard notation for function compostion. In general the substitution (a1/a2, a3/a4,..) (b1/b2,b3/b4…)…means to apply all the substitutions of the right most list, then take the result and apply all the ones of the ones of the next list, and so forth, until all substitutions have been applied.

The object of unification procedure is to discover at least one substitution that causes two literals to match.

# For example: the literals

hate(x,y) hate(Marcus, z)

could be unified with any of the following substitutions:

(marcus/x,z/y) (marcus/x,y/z) (marcus/x,Caesar/y,Caesar/z)

(marcus/x,polonius/y,polonius/z)

The first two of these are equivalent except for lexical variation. But the second two, although they produce a match, also produce a substitution that is more restrictive than absolutely necessary for the match.

# ALGORITHM:

* 1. If L1 or L2 are both variables or constants, then:
		1. If L1 and L2 are identical, then return NIL.
		2. Else if L1 is a variable, then if L1 occurs in L2 then return

{FAIL}, else return (L2/L1).

* + 1. Else if L2 is a variable then if L2 occurs in L1 then return {FAIL}, else return (L1/L2).
		2. Else return (FAIL}.
	1. If the initial predicate symbols in L1 and L2 are not identical, then return

{FAIL}

* 1. If L1 and L2 have a different number of arguments, then return {FAIL}.
	2. Set SUBST to NIL. (At the end of this procedure, SUBST will contain all the substitutions used to unify L1 and L20
	3. For I1 to number of arguments inL1:
		1. Call unify with the /th argument of L1 and the i th argument of L2, putting result in S.
		2. If S contains FAIL then return {FAIL}.
		3. If S is not equal to NIL then;
			+ Apply S to the remainder of both L1 and L2.
			+ SUBST := APPEND(S, SUBST)
	4. Return SUBST.

# WHAT IS RESOLUTION DISCUSS BRIEFLY IN PROPOSITIONAL AND PREDICATE LOGIC?

**Basic idea:** Resolution produces proofs by refutation. In others words, to prove a statement resolution attempts to show that negation of the statements produces a contradiction with the known statements (i.e that is unsatisfiable).

# The basis of resolution in propositional logic:

The resolution procedure is a iterative process at each step, two clauses called parent clauses are compared (resolved), yielding a new clause that has been inferred from them. The new clause represents ways that the two parent clauses interact with each other.

Suppose there are two literals in the system

**Winter** *∨ summer*

*¬***Winter** *∨ cold*

Recall that this means that both clauses must be true(i.e., the clauses, although they look independent, are really cojoined).

* We can observe precisely one of winter and *¬*winter will be true at any point.
* If winter is true , then cold must be true to guarantee the truth of the second clause.
* If *¬*winter is true, then summer must be true to gurantee the truth of the first clause.
* Thus we see that from these two clauses we can deduce Summer *∨ cold*

This is the deduction that the resolution procedure will make. Resolution operates by taking two

clauses that each contain the same literal, in this example , winter.

If the clause that is produced is the empty clause, then a contradiction has been found. For example, the two clauses Winter

*¬W*inter

Will produce an empty clause.

1. If contradiction exist then eventually it will be found.
2. If no contradiction exists, it is possible that the procedure will never terminate.

## Note: a→b is equivalent to ¬a∨b ( convert to clause form)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Example:** | **Given axioms** | **Converted to clause form** | **steps** |  |
|  | **P** | **P** | **1** |
|  | **(P*∧Q) →R*** | ***¬P∨¬Q∨R*** | ***2*** |
|  | ***(S∨T) →Q*** | ***¬S∨¬Q*** |  | ***3*** |
|  |  | ***¬T∨Q*** |  | ***4*** |
|  | ***T*** | ***T*** |  | ***5*** |



Step1: axiom1 and axiom2 are conjoined. Step2: *R and ¬R are true finally we get Axiom3.*

*Step3: Axiom 3 and Axiom 4 are conjoined where P and ¬P are found to be true, finally weget axiom5*

*Step 4: from Axiom 5 and Axiom6 weget axiom 7 where Q and ¬Q are found to be true. Step 5: Contradiction found.*

# Resolution in Predicate Logic:

In order to use resolution for expressions in predicate logic, we use the unification algorithm to locate pairs of literals that cancel out.

We also need to use the unifier produced by the unification algorithm to generate the resolvent clause.

For example:

Suppose we want to resolve two clauses;

1. *Man(Marcus)*
2. *¬Man(x1) ∨ mortal (x1)*
* The literal man/Marcus can be unified with the literal man{x\} with substituting

**x1=marcus** i.e Marcus/x1.

* While substituting **x1=marcus**, *¬Man(Marcus) is false. Where we cannot simply cancel out the two man literals as we did in propositional logic and generate the resolvent mortal(x1)*
* *Clause 2 says that for a given x1, either ¬Man(x1) or mortal (x1). So for it to be true, we can now conclude only that mortal(Marcus) must be true.*

# Algorithm: resolution

**Step1:** convert all the propositions of F to clause form.

**Step2:** Negate P & convert the result to clause form. Add it to set of clauses obtained in step1.

**Step3:** Repeat until either a contradiction is found or no progress can be made.

* 1. Select two clauses. Call the parent clauses.
	2. Resolve them together. The resulting clause, called the resolvent.

The resolvent will be the disjunction of all the literals of both parent clauses with appropriate substitutions performed and with the following exception. If there is one pair of literals T1 and T2 such that one of the parent clauses contains T2 and the other contains T1 and if T1 and T2 are unifiable, then neither T1 nor T2 should appear in the resolvent.

* 1. If the resolvent is empty clause then a contradiction has been found. , then a contradiction has been found. If it is not, then add it to the set of clauses available to the procedure.

# WEAK SLOT FILLER STRUCTURES

**Introduction**

* It enables attribute values to be retrieved quickly
	+ assertions are indexed by the entities
	+ binary predicates are indexed by first argument. *E.g. team(Mike-Hall , Cardiff)*.
* Properties of relations are easy to describe .
* It allows ease of consideration as it embraces aspects of object oriented programming. So called because:
* A *slot* is an attribute value pair in its simplest form.
* A *filler* is a value that a slot can take -- could be a numeric, string (or any data type) value or a pointer to another slot.
* A *weak* slot and filler structure does not consider the *content* of the representation. We will study two types:
* Semantic Nets.
* Frames.

# Semantic Nets

The major idea is that:

* The meaning of a concept comes from its relationship to other concepts, and that,
* The information is stored by interconnecting nodes with labeled arcs.



# Intersection search

One of the early ways that semantic nets were used was to find relationships among objects by spreading activation out from each of two nodes and seeing where the activation met. This process is called intersection search.

# Representing Non binary predicates

Semantic nets are a natural way to represent relationships that would appear as grpund instances of binary predicates in predicate logic. For example some of the arcs from the above figure, could be represented as logic as

isa(Person, Mammal)

instance (Pee-Wee-Reese, Person) team(Pee-Wee-Reese, Brooklyn-Dodgers) uniform-color (Pee-Wee-Reese, Blue)

These values can also be represented in logic as: *isa(person, mammal), instance(Mike- Hall, person) team(Mike-Hall, Cardiff)*

We have already seen how conventional predicates such as *lecturer(dave)* can be written as *instance (dave, lecturer)* Recall that *isa* and *instance* represent inheritance and are popular in many knowledge representation schemes. But we have a problem: *How we can have more than 2 place predicates in semantic nets? E.g. score(Cardiff, Llanelli, 23-6)* Solution:

* Create new nodes to represent new objects either contained or alluded to in the knowledge, *game* and *fixture* in the current example.
* Relate information to nodes and fill up slots.



Fig. A Semantic Network for *n*-Place Predicate

As a more complex example consider the sentence: *John gave Mary the book*. Here we have several aspects of an event.



Fig. A Semantic Network for a Sentence

# Making some important distinctions

Basic inference mechanism: *follow links between nodes*. Two methods to do this:

# Intersection search

-- the notion that *spreading activation* out of two nodes and finding their intersection finds relationships among objects. This is achieved by assigning a special tag to each visited node.

Many advantages including entity-based organisation and fast parallel implementation. However very structured questions need highly structured networks.

# Inheritance

-- the *isa* and *instance* representation provide a mechanism to implement this.

Inheritance also provides a means of dealing with *default reasoning*. *E.g.* we could represent:

* Emus are birds.
* Typically birds fly and have wings.
* Emus run.

in the following Semantic net:



Fig. A Semantic Network for a Default Reasoning

In making certain inferences we will also need to *distinguish between the link that defines a new entity and holds its value and the other kind of link that relates two existing entities*. Consider the example shown where the height of two people is depicted and we also wish to compare them.

We need extra nodes for the concept as well as its value.



Fig. Two heights

Special procedures are needed to process these nodes, but without this distinction the analysis would be very limited.

Fig. Comparison of two heights

# Partitioned Semantic Nets:

**Partitioned Networks** *Partitioned* Semantic Networks allow for:

* propositions to be made without commitment to truth.
* expressions to be quantified.

Basic idea: *Break network into* ***spaces*** *which consist of groups of nodes and arcs and regard each* ***space*** *as a node.*

Consider the following: *Andrew believes that the earth is flat.* We can encode the proposition *the earth is flat* in a *space* and within it have nodes and arcs the represent the fact (Fig. [15](http://www.cs.cf.ac.uk/Dave/AI2/node62.html#figpartition1%23figpartition1)). We can

the have nodes and arcs to link this *space* the the rest of the network to represent Andrew's

belief.  Fig. Partitioned network

# The Evolution into frames:

* The idea of a semantic net started out simply as a way to represent labeled connections among entities.
* But as we have just seen, as we expand the range of problem solving tasks that the representation must support, the representation itself necessarily begin to become more complex.
* In particular, it becomes useful to assign more structure to nodes as well as to links.

# Frames

A frame is a collection of attribute (usually called slots) and associated values (and possibly constraints on values) that describes some entity in the world. Sometimes a frame describes an entity in some absolute sense; sometimes it represents the entity from a particular point of view (as it did in the vision system proposal in which the term frame was first introduced).

A single frame taken alone is rarely useful, instead we build frame systems out collections of frames that are connected to each other by virtue of the fact that the value of an explore ways that frame systems can be used to encode knowledge and support reasoning.

# Frames as Sets and Instances:

Frames can also be regarded as an extension to Semantic nets. Indeed it is not clear where the distinction between a semantic net and a frame ends. Semantic nets initially we used to represent

labelled connections between objects. As tasks became more complex the representation needs to be more structured. The more structured the system it becomes more beneficial to use frames. A *frame* is a collection of attributes or slots and associated values that describe some real world entity. Frames on their own are not particularly helpful but frame systems are a powerful way of encoding information to support reasoning. Set theory provides a good basis for understanding frame systems. Each frame represents.

* a class (set), or
* an instance (an element of a class).

Consider the example first discussed in Semantics Nets

*Person*

*Adult-Male*

*Rugby-Player*

*Back*

*Mike-Hall*

*Isa : Mammal*

*Cardinality :*

*Isa : Person*

*Cardinality :*

*Isa : Adult-Male Cardinality :*

*Height :*

*Weight :*

*Position :*

*Team :*

*Team-Colours :*

*Isa : Rugby-Player*

*Cardinality :*

*Tries :*

*Instance : Back*

*Height : 6-0*

*Position : Centre*

*Team : Cardiff-RFC*

*Rugby-Team*

*Team-Colours : Black/Blue*

*Isa : Team Cardinality :*

*Team-size : 15*

*Coach :*



**Figure:** A simple frame system

Here the frames *Person, Adult-Male, Rugby-Player and Rugby-Team* are all **classes** and the frames *Robert-Howley* and *Cardiff-RFC* are instances.

# Note

* The *isa* relation is in fact the subset relation.
* The *instance* relation is in fact *element of*.
* The *isa* attribute possesses a transitivity property. This implies: *Robert-Howley* is a *Back*

and a *Back* is a *Rugby-Player* who in turn is an *Adult-Male* and also a *Person*.

* Both *isa* and *instance* have inverses which are called subclasses or all instances.
* There are attributes that are associated with the class or set such as cardinality and on the other hand there are attributes that are possessed by each member of the class or set.

# DISTINCTION BETWEEN SETS AND INSTANCES

It is important that this distinction is clearly understood.

*Cardiff-RFC* can be thought of as a set of players or as an instance of a *Rugby-Team*. If *Cardiff-RFC* were a *class* then

* its instances would be players
* it could not be a subclass of *Rugby-Team* otherwise its elements would be members of Rugby-Team which we do not want.

Instead we make it a subclass of *Rugby-Player* and this allows the players to inherit the correct properties enabling us to let the *Cardiff-RFC* to inherit information about teams.

This means that *Cardiff-RFC* is an instance of *Rugby-Team*.

**BUT** There is a problem here:

* A class is a set and its elements have properties.
* We wish to use inheritance to bestow values on its members.
* But there are properties that the set or class itself has such as the manager of a team.

This is why we need to view *Cardiff-RFC* as a subset of one class players and an instance of teams. We seem to have a CATCH 22. *Solution*: **MetaClasses**

A metaclass is a special class whose elements are themselves classes. Now consider our rugby teams as:



**Figure:** A Metaclass frame system

The basic metaclass is *Class*, and this allows us to

* define classes which are instances of other classes, and (thus)
* inherit properties from this class.

Inheritance of default values occurs when one element or class is an instance of a class.

# Slots as Objects

How can we to represent the following properties in frames?

* Attributes such as *weight, age* be attached and make sense.
* Constraints on values such as *age* being less than a hundred
* Default values
* Rules for inheritance of values such as children inheriting parent's names
* Rules for computing values
* Many values for a slot.

A slot is a relation that maps from its domain of classes to its range of values.

**NOTE** the following:

* Instances of *SLOT* are slots
* Associated with *SLOT* are attributes that each instance will inherit.
* Each slot has a domain and range.
* Range is split into two parts one the class of the elements and the other is a constraint which is a logical expression if absent it is taken to be true.
* If there is a value for default then it must be passed on unless an instance has its own value.
* The *to-compute* attribute involves a procedure to compute its value. *E.g.* in *Position*

where we use the dot notation to assign values to the slot of a frame.

* Transfers through lists other slots from which values can be derived from inheritance.

# Interpreting frames

A frame system interpreter must be capable of the following in order to exploit the frame slot representation:

* Consistency checking -- when a slot value is added to the frame relying on the domain attribute and that the value is legal using range and range constraints.
* Propagation of *definition* values along *isa* and *instance* links.
* Inheritance of default. values along *isa* and *instance* links.
* Computation of value of slot as needed.
* Checking that only correct number of values computed.

# Algorithm: Property Inheritance

To retrieve a value V for slot S of an instance F do.

1. Set CANDIDATES to empty.
2. Do breadth first or depth first search up the isa hierarchy from F, following all instance and isa links. At each step, see if a value for S or one f its generalizations is stored.
	1. If a value is found, add it to CANDIDATES and terminate that branch of the search.
	2. If no value is found but there are instance or isa links upward, follow them.
	3. Otherwise, terminate the branch.
3. for each element C of CANDIDATES do:
	1. see if there is any other element of CANDIDATES that was derived from a class closer to F than the class from which C came.
	2. If there is, then, remove C from CANDIDATES.
4. check the cardinality of CANDIDATES:
	1. if it is 0, then report that no value was found.
	2. If it is 1, then return the single element of CANDIDATES as V.
	3. If it is greater than 1, report a contradiction.

# Frame Languages:

The idea of a frame system as a way to represent declarative knowledge has been encapsulated in a series of frame oriented knowledge representation languages, whose features have evolved and been driven by an increased understating of the sort of representation issues.

Example: KRL, FRL, RLL, KL-ONE, Brachman and Schmolze, KRYPTON,NIKL.

* 1. **STRONG SLOT FILLER STRUCTURES**

# Introduction

Strong Slot and Filler Structures typically*:*

* Represent links between objects according to more **rigid** rules.
* Specific notions of what types of object and relations between them are provided.
* Represent knowledge about common situations.

# Conceptual Dependency (CD)

Conceptual Dependency originally developed to represent knowledge acquired from natural language input.

The goals of this theory are:

* To help in the drawing of inference from sentences.
* To be independent of the words used in the original input.
* That is to say: *For any 2 (or more) sentences that are identical in meaning there should be only one representation of that meaning.*

It has been used by many programs that portend to understand English (*MARGIE, SAM, PAM*). CD developed by Schank *et al.* as were the previous examples.

# CD provides:

* a structure into which nodes representing information can be placed
* a specific set of primitives
* at a given level of granularity.

Sentences are represented as a series of diagrams depicting actions using both abstract and real physical situations.

* The agent and the objects are represented
* The actions are built up from a set of primitive acts which can be modified by tense. Examples of Primitive Acts are:

# ATRANS

-- Transfer of an abstract relationship. *e.g. give*.

# PTRANS

-- Transfer of the physical location of an object. *e.g. go*.

# PROPEL

-- Application of a physical force to an object. *e.g. push*.

# MTRANS

-- Transfer of mental information. *e.g. tell*.

# MBUILD

-- Construct new information from old. *e.g. decide*.

# SPEAK

-- Utter a sound. *e.g. say*.

# ATTEND

-- Focus a sense on a stimulus. *e.g. listen, watch*.

# MOVE

-- Movement of a body part by owner. *e.g. punch, kick*.

# GRASP

-- Actor grasping an object. *e.g. clutch*.

# INGEST

-- Actor ingesting an object. *e.g. eat*.

# EXPEL

-- Actor getting rid of an object from body. *e.g. ????*.

Six primitive conceptual categories provide *building blocks* which are the set of allowable dependencies in the concepts in a sentence:

# Advantages of CD:

* Using these primitives involves fewer inference rules.
* Many inference rules are already represented in CD structure.
* The holes in the initial structure help to focus on the points still to be established.

# Disadvantages of CD:

* Knowledge must be decomposed into fairly low level primitives.
* Impossible or difficult to find correct set of primitives.
* A lot of inference may still be required.
* Representations can be complex even for relatively simple actions. Consider: *Dave bet Frank five pounds that Wales would win the Rugby World Cup*. Complex representations require a lot of storage

# Applications of CD:

**MARGIE**

(*Meaning Analysis, Response Generation and Inference on English*) -- model natural language understanding.

# SAM PAM

(*Script Applier Mechanism*) -- Scripts to understand stories. See next section.

(*Plan Applier Mechanism*) -- Scripts to understand stories.

Schank *et al.* developed all of the above.

# Scripts

A *script* is a structure that prescribes a set of circumstances which could be expected to follow on from one another.

It is similar to a thought sequence or a chain of situations which could be anticipated.

It could be considered to consist of a number of slots or frames but with more specialised roles.

Scripts are beneficial because:

* Events tend to occur in known runs or patterns.
* Causal relationships between events exist.
* Entry conditions exist which allow an event to take place
* Prerequisites exist upon events taking place. *E.g.* when a student progresses through a degree scheme or when a purchaser buys a house.

The components of a script include:

# Entry Conditions

-- these must be satisfied before events in the script can occur.

# Results

-- Conditions that will be true after events in script occur.

# Props Roles Track Scenes

-- Slots representing objects involved in events.

-- Persons involved in the events.

-- Variations on the script. Different tracks may share components of the same script.

-- The sequence of *events* that occur. *Events* are represented in *conceptual dependency*

form.

Scripts are useful in describing certain situations such as robbing a bank. This might involve:

* Getting a gun.
* Hold up a bank.
* Escape with the money. Here the *Props* might be
* Gun, *G*.
* Loot, *L*.
* Bag, *B*
* Get away car, *C*. The *Roles* might be:
* Robber, *S*.
* Cashier, *M*.
* Bank Manager, *O*.
* Policeman, *P*.

The *Entry Conditions* might be:

* *S* is poor.
* *S* is destitute. The *Results* might be:
* *S* has more money.
* *O* is angry.
* *M* is in a state of shock.
* *P* is shot.

There are 3 scenes: obtaining the gun, robbing the bank and the getaway. Some additional points to note on Scripts:

* If a particular script is to be applied it must be activated and the activating depends on its significance.
* If a topic is mentioned in passing then a pointer to that script could be held.
* If the topic is important then the script should be opened.
* The danger lies in having too many active scripts much as one might have too many windows open on the screen or too many recursive calls in a program.
* Provided events follow a known trail we can use scripts to represent the actions involved and use them to answer detailed questions.
* Different trails may be allowed for different outcomes of Scripts ( *e.g.* The bank robbery goes wrong).

The full Script could be described in the following figure

E



Fig. Simplified Bank Robbing Script Advantages of Scripts:

* Ability to predict events.
* A single coherent interpretation may be build up from a collection of observations. Disadvantages:
* Less general than frames.
* May not be suitable to represent all kinds of knowledge.

# CYC

What is CYC?

* An ambitious attempt to form a very large knowledge base aimed at capturing commonsense reasoning.
* Initial goals to capture knowledge from a hundred randomly selected articles in the En*CYC*lopedia Britannica.
* Both Implicit and Explicit knowledge encoded.
* Emphasis on study of underlying information (assumed by the authors but not needed to tell to the readers.

Example: Suppose we read that *Wellington learned of Napoleon's death*

Then we (humans) can conclude *Napoleon never new that Wellington had died*. How do we do this?

We require special implicit knowledge or commonsense such as:

* We only die once.
* You stay dead.
* You cannot learn of anything when dead.
* Time cannot go backwards. Why build large knowledge bases:

**Brittleness** - Specialised knowledge bases are *brittle*. Hard to encode new situations and non- graceful degradation in performance. Commonsense based knowledge bases should have a firmer foundation.

**Form and Content** - Knowledge representation may not be suitable for AI. Commonsense strategies could point out where difficulties in content may affect the form.

**Shared Knowledge** - Should allow greater communication among systems with common bases and assumptions. How is CYC coded?

* Special CYCL language:
	+ LISP like.
	+ Frame based
	+ Multiple inheritance
	+ Slots are fully fledged objects.
	+ Generalised inheritance -- any link not just *isa* and *instance*.