Integrated Knowledge Base: An Approach to Knowledge Extraction

Deepa Chaudhary, Praveen K.Yadav, Rakesh K. Singh, Subhojit Mitra, Siddharth
I.P.E.C Ghaziabad

ABSTRACT

This paper describes an approach to integrate knowledge base via converting predicates into Semantic networks and in frames. A knowledge base can be represented in a tabular form, a rule form, a tree form or any other form suitable for knowledge representation. Form conversion can be accomplished at all times. Unification of knowledge always overcome individual limitations and has synergetic effects in knowledge extraction. The graphical representation of knowledge base has more understandability than any other representation. Aim of this paper is to develop a system which accepts input from the user in the form of predicates and generates outputs with graphical representation of semantic networks as well as of frames.

Keywords: Knowledge Representation, Predicate Logic, Semantic Network, Frames, Ontology, Script and Production rule.

1. INTRODUCTION

Machines cannot be called intelligent until they are able to learn to do new things and to adapt to new situations, rather than simply doing as they are told to do.[Rich and Knight, 1991]

Knowledge is the information which represents long-term relationship i.e. ways of doing things, common sense, ideas, methods, skills, and so on. Knowledge is the backbone of Artificial Intelligence and so issues related to knowledge representation, understanding, knowledge designing and implementation are of relevance. In the creation of common sense knowledge base major information contents are the ontology of classes, instances and individuals; parts, properties, and materials of objects; functions and uses of objects; locations, durations, post and preconditions of events; behaviour; emotions; strategies; and context. To accommodate all these concepts in a single unit, a unique knowledge representation scheme is required [1].

Ontology is a specification of a conceptualization which defines a set of representational primitives with which to model a domain of knowledge or discourse. The representational primitives are typically classes (or sets), attributes (or properties), and relationships (or relations among class members). The definitions of the representational primitives include information about their meaning and constraints on their logically consistent application. Ontology defines (specifies) the concepts, relationships, and other distinctions that are relevant for modeling a domain. The specification takes the form of the definitions of representational vocabulary (classes, relations, and so forth), which provide meanings for the vocabulary and formal constraints on its coherent use. Ontologies are used in artificial intelligence, semantic web, software engineering, biomedical information, library science and information architecture as a form of knowledge representation about the world or some part of it. Ontology language can be classified by logic, graph and using frames [2].

Various knowledge representation schemes like Logic, Frame, Production Rules, Scripts, and Grids etc. have been employed in a number of successful practical systems. There are systems that employ different formalisms to knowledge representation like logic [3], semantic net [4], frame [5], production rules [6], script [7] separately or suitably combines some of these schemes and hence are more effective in subsequent use.

*Predicates***,** the language of logic, is one way of representing knowledge. Predicates can be used to illustrate all the basic concepts of logic. The atomic sentences (indivisible syntactic elements) consist of a single predicate followed by a parenthesized list of terms. The meaning of a concept comes from the ways in which it is connected to other concepts [1]. For example,

Facts can be written as:

Mammal has legs either 0 or 2 or 4.

Human is a mammal

Human eats food.

Humans have 2 legs.

There corresponding predicates are as follows:

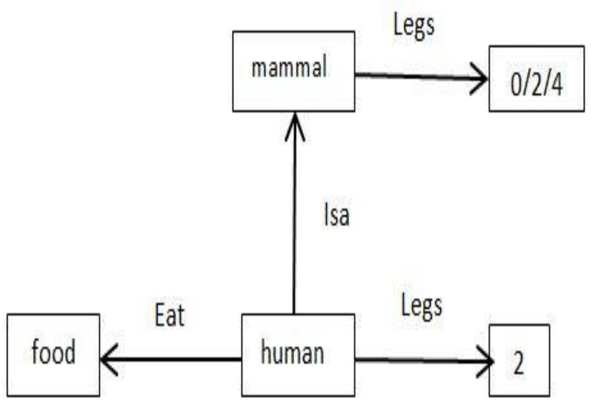
Legs(mammal,0/2/4)
Isa(human, mammal)

Eat(human ,food)

Legs(human,2)

Semantic networks are an alternative to predicate logic as a form of knowledge representation. The idea is that we can store our knowledge in the form of a graph, with nodes representing objects in the world, and arcs representing relationships between those objects. Semantic nets are natural way to represent relationships that would appear as ground instances of binary predicates in predicate logic (e.g. instance(Marcus , Man)). Semantic nets are graph based approach of ontology language.

This form of representation is closer to the way human structure knowledge by building mental links between things than the predicate logic we considered earlier. An important property of semantic nets, that they may be used for a form of inference known as inheritance. The idea of this is that if an object belongs to a class (indicated by an isa link) it inherits all the properties of that class. Another important aspect of semantic nets is their ability to represent default values for categories. Semantic nets have advantage of simplicity and transparency of the inference processes example:



Similarly *Frame* represents prototypical constellations of elements and attributes. Frame is a general record-like structure which represents an entity as a set of slots (attributes) and associated values. A frame can represent a specific entity, or a general concept A frame represents knowledge in an "object-oriented" manner, which means that facts are associated with the objects mentioned in the facts. A frame is an object with which facts are associated. [1].

Frame are said to contain slots, which may be used to hold several types of information. A slot is a mapping from a frame to a set of values. The values stored in slots are called Fillers. The slots may also contain procedures (i.e. procedural knowledge or descriptions of how to do things, not descriptions of things themselves). When slots contain procedural knowledge, this is often called "procedural attachment." This is an important distinction between frame and semantic networks. Unlike a record, structure or class, it is possible to add slots to a frame dynamically (i.e. while the program is executing) and the contents of the slot need not be a simple value. Frame are implicitly associated with one another because the value of a slot can be another frame [1]. A frame can inherit slots from another frame based on hierarchy regardless of the value of slot. Frame can represent procedures, so they are known as having a procedural nature as well as a declarative capability. For example:

MAMMAL

legs : 0/2/4

exist on : earth
have: lungs

____

HUMAN

isa : mammal

eat : food

legs : 2

____

2. INTEGRATION OF KNOWLEDGE BASE

In our algorithmic approach we are trying to develop a method to transform one form of knowledge representation into another.

A. Conversion of Predicates into Semantic Network

In Knowledge Base, predicates are used to store area specific knowledge. Predicates store knowledge in form of relationship between two objects or concepts while frame structures knowledge to associate mapping between chunks of knowledge. We can store data in a knowledgebase using Predicate. Suppose a user enters the following predicates:

Is a(mammal ,living thing)

Do(living thing ,respiration)

Have(mammal ,legs)

Legs(mammal,0/2/4)

Is a(human ,mammal)

Eat(human, food)

Legs(human,2)

Instance of(ram, human)

Eat(ram, ice-cream)

Instance of (ram ,mammal)

Instance of(mitthu ,bird)

Have(mammal ,lungs)

Exist on (mammal ,earth)

Exist on (living thing ,universe)

Is a (bird, living thing)

For the sake of simplicity, case sensitivity is not induced while storing knowledge derived from these predicates. Once we get the predicate as input we separate each predicate into three components i.e. attribute with two objects, object 1 having that attribute and object 2 characterizing that attribute. Linear search is used to gather all the knowledge related to particular object as object 1 whose attribute is not "instance of" and a list of unique values for Object1 is generated.

For all those predicates which have "instance of" attribute, its object 2 value is added in the list if it does not exist in the list of unique objects for which it is an instance of . In case of example given above the list of unique objects is as follows:

MAMMAL

LIVINGTHING

HUMAN

BIRD

In the above example any one of two cases for "instance of" attribute can happen:

Case 1: For predicate instance of (ram, mammal) "mammal" is already present in the list so mammal is not added to the list again.

Case 2: For the predicate Instance of (mitthu ,bird), mitthu is an "instance of" bird and bird is not present in the list of unique objects and hence Bird is added into the list of unique objects.

We will separately store all the information related to object 1 of predicate instance of. This will be treated differently as same name of object 1 can exists for the different object 2's.

Now the algorithm will first makes a list of attribute with their values for each object mentioned in the list. In this process if an attribute for any object occurs with same value at more than one place then consider it only once. If new attribute value is found for the same attribute then instead of creating a new attribute update its value. However in the case of those predicate which have "instance of" attribute do not update the values.

____

HUMAN

is a : mammal

eat : food

legs : 2

____

MAMMAL

is a : living thing

have : lungs

legs : 0/2/4

exist on : earth

____

BIRD

is a : living thing

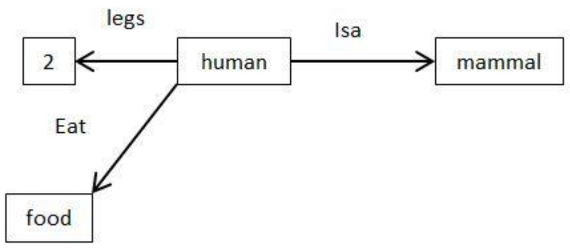
____

LIVING THING

do : respiration

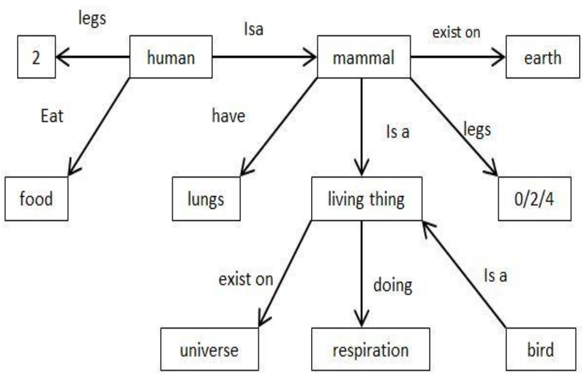
exist on : universe

After this step first object in the list HUMAN and its corresponding attribute values are drawn. They are connected through arrows labelled with corresponding attribute



Moving forward in the list, next object is MAMMAL. As it is already present, we do not represent it again. Its corresponding attribute values are drawn and connected through arrows labelled with attribute.

Same procedure is repeated for others, which produces final output as shown below.



B. Conversion of Predicates into Frames

Continuing this method further check for presence of "is a" slot. If "is a" slot is present with any object then check whether its filler is also an object. If so then inherit all its (filler object's) slots (with "*" to differentiate it from original slot) and filler. Only immediate parent class is searched for inheritance [8].

____

HUMAN

is a : mammal

eat : food

legs : 2

*is a : living thing

*have : lungs

*legs : 0/2/4

*exist on : earth

____

BIRD

is a : living thing

*do : respiration

*exist on : universe

____

MAMMAL

is a : living thing

have : lungs

legs : 0/2/4

exist on : earth

*do : respiration

*exist on : universe

____

LIVING THING

do : respiration

exist on : universe

____

The algorithm now checks for the next level of inheritance as discussed in the previous step. This algorithm is capable of inheriting all possible characteristics of its parent class.

____

HUMAN

is a : mammal

eat : food

legs : 2

*is a : living thing

*have : lungs

*legs : 0/2/4

*exist on : earth

*do : respiration

*exist on : universe

____

BIRD

is a : living thing

*do : respiration

*exist on : universe

____

MAMMAL

is a : living thing

have : lungs

legs : 0/2/4

exist on : earth

*do : respiration

*exist on : universe

____

LIVING THING

do : respiration

exist on : universe

____

This algorithm inherits only those slots from its parent object which are not defined explicitly for the current object. Here HUMAN will not further inherit legs attribute from its parent MAMMAL as it already have it with its filler value.

____

HUMAN

is a : mammal

eat : food

legs : 2

*have : lungs

*exist on : earth

*do : respiration

*exist on : universe

____

BIRD

is a : living thing

*do : respiration

*exist on : universe

____

MAMMAL

is a : living thing

have : lungs

legs : 0/2/4

exist on : earth

*do : respiration

*exist on : universe

____

LIVING THING

do : respiration

exist on : universe

____

If a slot is defined for parent object as well as its grand parent object then current object will inherit slots only from its parent object. Here HUMAN will inherit "exist on" from MAMMAL not from LIVING THING.

____

HUMAN

is a : mammal

eat : food

legs : 2

*have : lungs

*exist on : earth

*do : respiration

____

BIRD

is a : living thing

*do : respiration

*exist on : universe

____

MAMMAL

is a : living thing

have : lungs

legs : 0/2/4

exist on : earth

*do : respiration

*exist on : universe

____

LIVING THING

do : respiration

exist on : universe

____

If an entity has slot “has”/”have” then its filler is also added as another slot but is not shown unless its value is defined.

In above example slot lungs is not shown as slot unless lungs filler is not assigned.

ALGORITHM

TABLE: 2Dimensional array for storage of string with 3 columns for storing one element of predicate logic in each column.

TABLE1,TABLE2....TABLEi: 2Dimensional array for storage of string with 2 columns.

column1,column2,column3: Ist,IInd & III rd colum of table.

row I: Ith row of table.

Draw-arrow: function to create an arrow to join two nodes.

Draw-node: function to create a node.

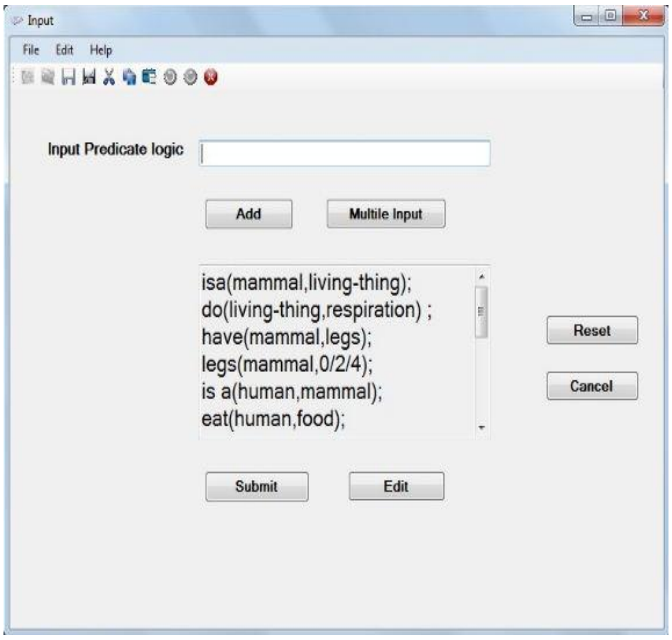
```
If ( first element of  predicate logic NOT=”instance of”
Store in TABLE( predicate logic).
      IF TABLE(column1) and TABLE(column2) exists
         Update TABLE(column3)
       END IF
     Else
        Store in separate location.
     END IF
CREATE  distinct LIST (TABLE(column 2))
Int n -> length of LIST
 FOR   I=0 to n
   FOR J=0 to end of TABLE
      Insert in TABLE2(list [I]),NULL)
      IF(TABLE(colum1)=list[I])
           Insert in TABLE2(TABLE2(column1),TABLE2
           (column3))
      END IF
      IF(TABLE(colum1)=”have” OR “has”)
           Insert in
           TABLE2(TABLE2(column1),<empty>)
      END IF
    END FOR
END FOR
FOR  I=0 TO end to TABLE2
 TABLE3(ROW I)=TABLE2(ROW I);
   WHILE(TABLE2(column3)!=NULL)
      IF(TABLE2(column1)=isa )
        Flag=true
        K=INDEXOF(TABLE2(column1)+1
     ENDIF
     END WHILE
IF( Flag=True)
   FOR X=K TO INDEXOF(NULL)
     TABLE3(column1)= * + TABLE2(column1)
     TABLE3(column2)=TABLE2(column2)
   END FOR
   Flag=False
 ENDIF
END FOR
REPEATE above FOR loop multiple times till it cover complete inheritance
FOR I =0 to LENGTHOF(TABLEi)
  FOR J=0 TO LENGTHOF(TABLEi)
     IF(TABLEi[I][1]=TABLEi[J][1] ignoring “*”)
       DELETE ROW J
     ENDIF
    END FOR
END FOR.
For( I=0 to  Length of TABLE2)
{
   If(Table2(I,column2)=NULL)
  {
      If(column1 is not present )
     {
        Draw central  node (TABLE2 (I,column1)
      }
        Current node=TABLE2(I,column1)
         I=I+1
 }
  Else  If( node bottom is empty)
  {
     Draw-arrow to bottom of current (TABLE2(I,column1))
     Draw-node to bottom of current (TABLE2(I,column2))
      I=I+1
   }
Else  If( node bottom-left  is empty)
  {
     Draw-arrow to bottom-left  of current
(TABLE2(I,column1))
```

```
    Draw-node to bottom-left of current (TABLE2(I,column2))
     I=I+1
  }
Else  If( node bottom-right  is empty)
  {
    Draw-arrow to bottom-right  of current (TABLE2(I,column1))
    Draw-node to bottom-right of current (TABLE2(I,column2))
     I=I+1
  }
Else  If( node left  is empty)
  {
    Draw-arrow to left  of current (TABLE2(I,column1))
    Draw-node to left of current (TABLE2(I,column2))
     I=I+1
  }
Else  If( node right  is empty)
  {
    Draw-arrow to right of current (TABLE2(I,column1))
    Draw-node to right of current (TABLE2(I,column2))
     I=I+1
  }
Else  If( node top-left  is empty)
  {
    Draw-arrow to top-left  of current (TABLE2(I,column1))
    Draw-node to top-left of current (TABLE2(I,column2))
     I=I+1
  }
Else  If( node top-right  is empty)
  {
    Draw-arrow to top-right  of current(TABLE2(I,column1))
    Draw-node to top-right of current(TABLE2(I,column2))
     I=I+1
  }
Else  If( node top  is empty)
  {
    Draw-arrow to top  of current (TABLE2(I,column1))
    Draw-node to top of current (TABLE2(I,column2))
     I=I+1
  }
```

3. IMPLEMENTATION DETAIL

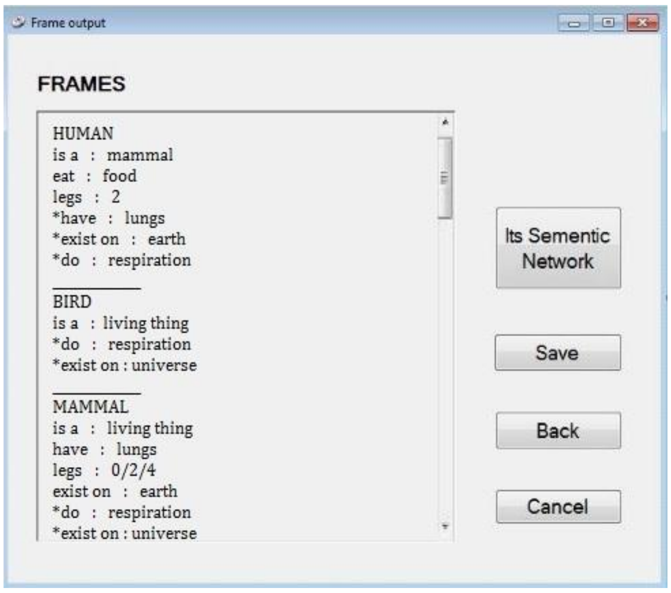
We have developed this module on .Net platform using c# to show working of this algorithm. A few screenshots to show inputs in the form of predicates and outputs in semantic network and in frame structure are shown.

Screen Shots:



isa(mammal,living-thing);
do(living-thing,respiration) ;
have(mammal,legs);
legs(mammal,0/2/4);
is a(human,mammal);
eat(human,food);

Fig. 1 Screen shot for Inputs as Predicates



```
HUMAN
is a  :  mammal
eat  :  food
legs  :  2
*have  :  lungs
*exist on  :  earth
*do  :  respiration

BIRD
is a  :  living thing
*do  :  respiration
*exist on : universe

MAMMAL
is a  :  living thing
have  :  lungs
legs  :  0/2/4
exist on  :  earth
*do  :  respiration
*exist on : universe
```

Fig. 2 Screen shot for Output as Frames

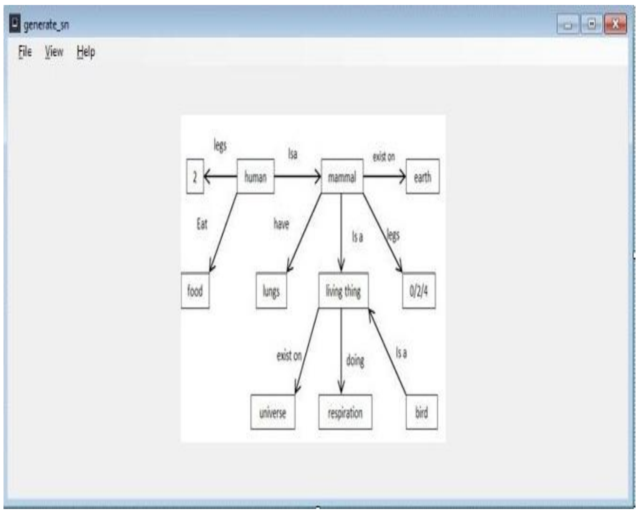


Fig. 3 Screen shot for outputs as Semantic network

4. CONCLUSION

We have successfully implemented the algorithm to convert predicates into Semantic networks and in frames. Work is going on to convert frames into semantic networks and vice-versa to give conversion of one Knowledge base into another a free hand. This experimental work gives us insights to convert one form of knowledge representation into other forms of knowledge representations to integrate all the existing knowledge bases.

5. REFERENCES

[1] Chaudhary Deepa, “Extracting EHCPRs Rules from Existing Knowledge Bases” International Conferences on Issues and Challenges in Network, Intelligence & Computing Technologies, 2-3 Sep.,2011.

[2] V. Maniraj, Dr. R Sivakumar, “Ontology Languages-A Review”.IACSIT.

[3] McDermott Drew, Doyle John, 1980, “Non-monotonic Logic”, Artificial Intelligence, vol. 13, pp. 41-72.

[4] Quillian, M.R 1968, “Semantic Memory”, in M. Minski, Ed., Semantic Information Processing, MIT Press, Cambridge, MA.

[5] Marvin Minsky, “A Framework for Representing Knowledge”, MIT-AI Laboratory, Memo 306, 1974.

[6] Davis R. and Buchanan B.G,”Production rules as a representation system for a knowledge based Consultation system”, Artificial intelligence, vol 8, pp. 15-45.

[7] Schank R.C and Abelson P.P, 1977, “Scripts Plans Goals and Understanding”,Hillsdale,N.J.

[8] Deepa Chaudhary, Praveen K.Yadaav, Rakesh K. Singh, Sudhanshu Mishra,Siddharth , “Enriching the Knowledgebase Using Unification Technioques”,”ARTCom 2012”.