St. Joseph’s College of Engineering & Technology Palai

Department of Computer Science & Engineering

S8 CS

RT804 Artificial Intelligence

Module 3

Website: http://sites.google.com/site/sjcetcssz

Artificial Intelligence - Module 3

Syllabus

Game playing and knowledge structures – Games as search problem –
Imperfect decisions – Evaluation functions – Alpha – Beta pruning
state of art game programs,
Introduction to frames and semantic nets.

Contents

Part I. Game Playing

[Russel2006] [Rich1991]

In many environments, there are multiple agents. In this, a given agent must consider the actions of other agents. If the agents are competing with each other, then such an environment is called a competitive environment. Competitive environments in which the agents goals are in conflict, results in problems known as games.

Consider the game of chess. If one player wins the game of chess, then utility function value is +1. in this, the opposite player loses. His utility function value is -1. if the game ends in a draw, the utility function value is 0. Like this, in AI, games are turn taking, 2 player, zero sum games.

For AI researchers, the nature of games makes them an attractive subject for study. The state of a game is easy to represent and agents are limited to a small number of actions. Precise rules are there for making these actions. By 1950, chess playing program was developed by Zuse, Shannon, Wiener and Turing. After that, systems for playing checkers, Othello, Backgammon and Go were developed. Games are interesting because they are too hard to solve. For example, chess has around 35^{100} states. It is impossible for a computer to generate all these states. Therefore game playing research has brought a number of interesting ideas on how to make the best possible use of time.

1 Minimax Search Procedure

We will consider games with 2 players. The 2 players in a game are referred to as MIN and MAX. MAX represents the player trying to win or to MAXimize his advantage. MIN is the opponent who attempts to MINimize MAX's score.

MAX moves first, and then they take turns moving until the game is over. At the end of the game, points are awarded to the winning player and penalties are given to the loser.

A game can be formally defined as a kind of search problem with the following components.

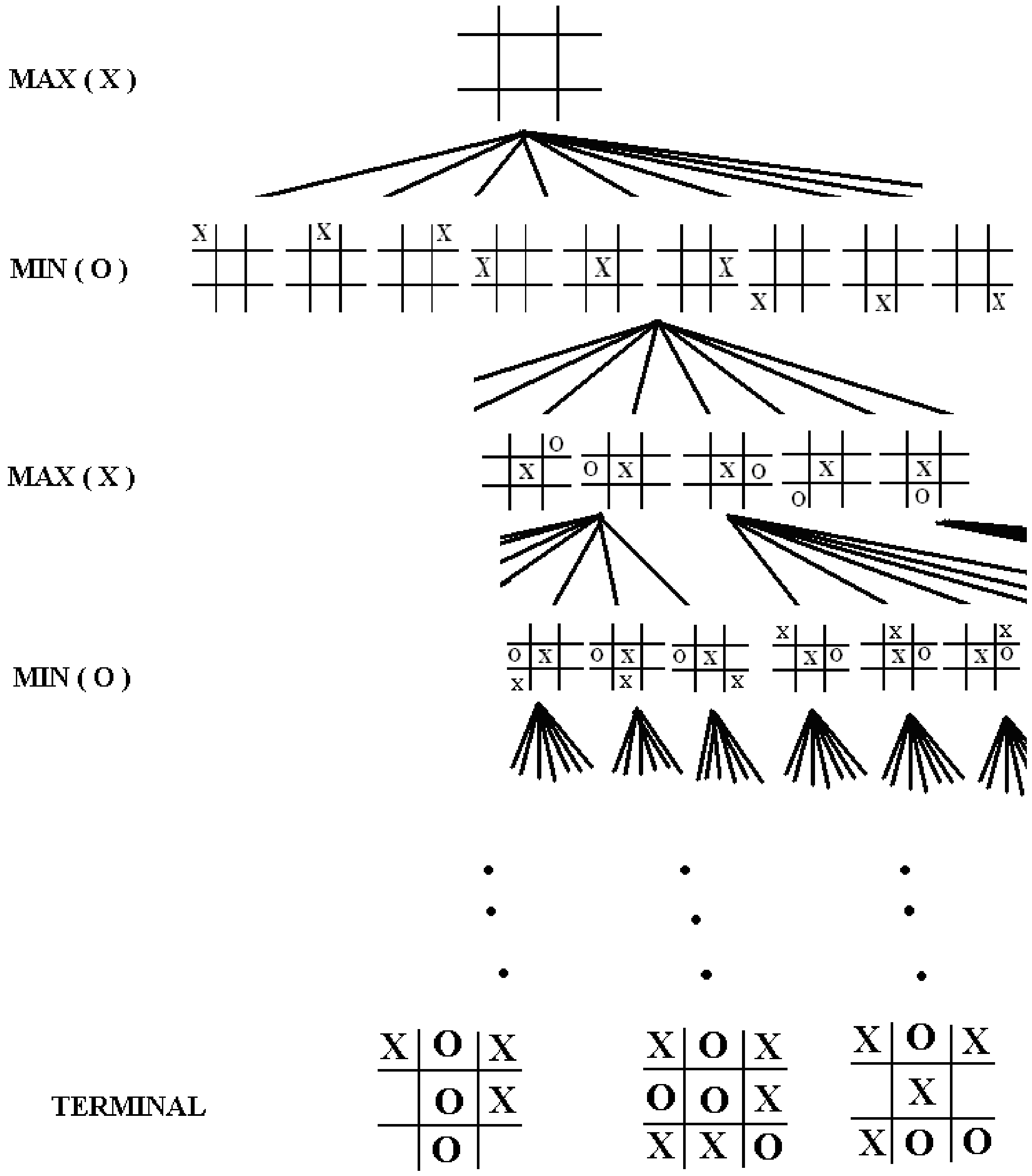
The initial state, which identifies the board position and identifies the player to move.

A successor function, which returns a list of (move, state) pairs, each indicating a legal move and the resulting state.

A terminal test, which determines when the game is over. States where the game has ended are called terminal states.

A utility function (also called an objective function) which gives a numeric value for the terminal states. In chess, the outcome is a win, loss or draw with values +1, -1 or 0. Some games have a wider variety of possible outcomes.

The initial state and the legal moves for each side define the game tree for the game. Figure below shows part of the game tree for tic-tac-toe.



The top node is the initial state, and MAX moves first, placing an X in an empty square. The figure shows part of the search tree giving alternating moves by MIN (O) and MAX (X), until we reach terminal states, which can be assigned utilities according to the rules of the game.

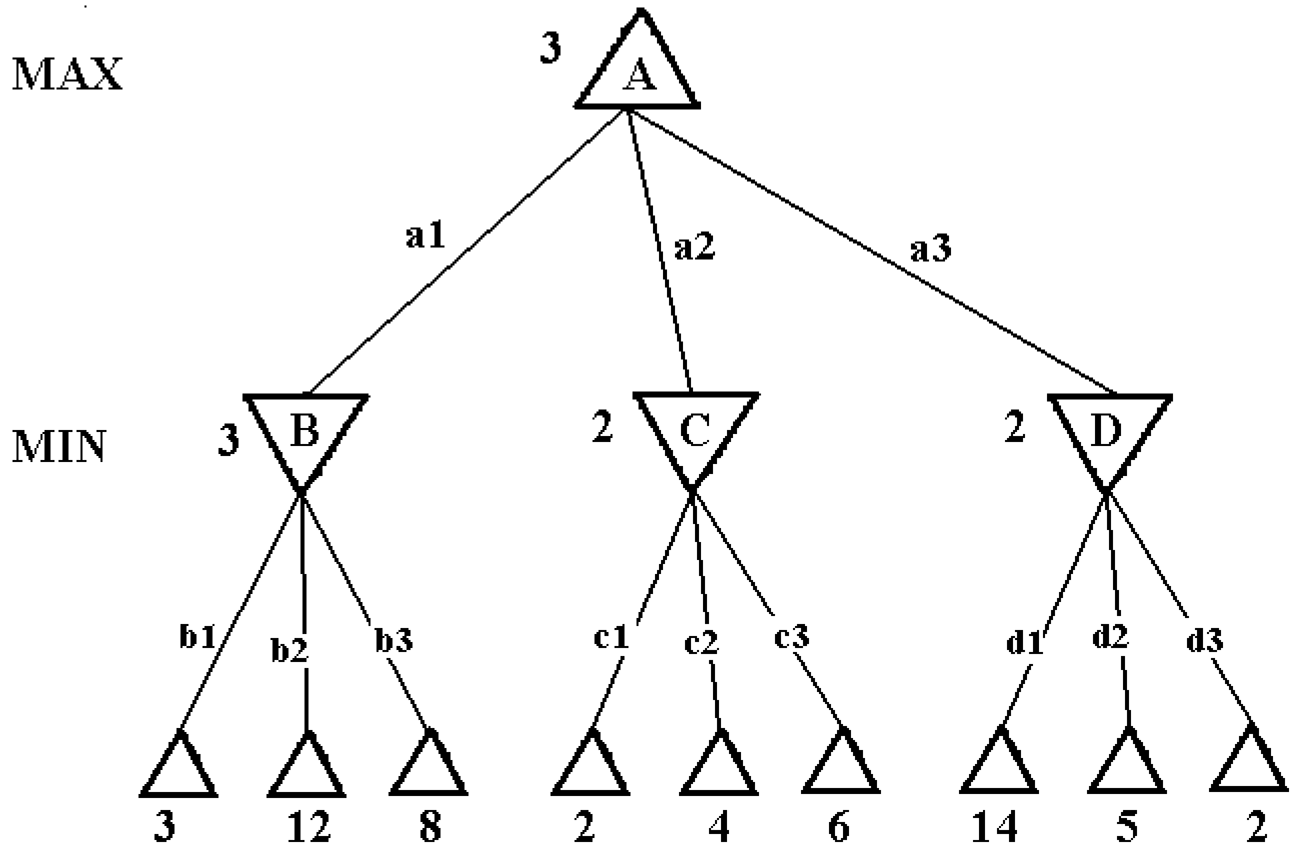
Play alternates between MAX's placing an X and MIN's placing an O until we reach leaf nodes corresponding to terminal states such that one player has there in a row or all the aquares are filled. The number on each leaf node indicates the utility value of the terminal state from the point of view of MAX.

High values are assumed to be good for MAX and bad for MIN. it is MAX's job to use the search tree to determine the best move.

Search Strategies

In a normal search problem, the best solution is a sequence of moves leading to a goal state. In a game, on the other hand, MIN has a role. MAX therefore must find a contingent strategy. We will see how to find this optimal strategy.

Even a simple game like tic-tac-toe is too complex for us to draw the entire game tree. So we will use a game tree as shown below.



The $\triangle$ nodes are MAX nodes, in which it is MAX's turn to move and the nodes are MIN nodes. The terminal states show the utility values for MAX.

The possible moves for MAX at the root node are labeled a1, a2, a3. the possible replies to a1 for MIN are b1, b2, b3 and so on. This game ends after one move each by MAX and MIN.

Given a game tree, the optimal strategy can be determined by examining the minimax value of each node, which we write as minimax-value (n). The minimax-value of a node is the utility for MAX of being in the corresponding state. The minimax-value of a terminal state is just its utility. Given a choice, MAX will prefer to move to a state of maximum value, whereas MIN prefers a state of minimum value. So we have

$$\text{MINIMAX-VALUE(n)} = \begin{cases} UTILITY(n) & \text{if n is a terminal state} \\ max_{S\epsilon successors(n)}\ \text{MINIMAX-VALUE(s)} & \text{if n is a MAX node} \\ min_{S\epsilon successors(n)}\ \text{MINIMAX-VALUE(s)} & \text{if n is a MIN node} \end{cases}$$

Let us apply these definitions to the above game tree. The terminal nodes on the bottom level are labeled with their utility values. The first MIN node, labeled B, has 3 successors wit hvalues3, 12 and 8. so its minimax-value is 3. similarly c has a minimax-value 2 and D ahs minimax-value 2. the root node is a MAX node; its successors have minimax values 3, 2 and 2. So it has a minimax value of 3.

That is,

$$MINIMAX-VALUE(A) = MAX[MIN(3,12,8), MIN(2,4,6), MIN(14,5,2)]$$
$$= MAX[3,2,2]$$
$$= 3$$

As a result, action a1 is the optimal choice for MAX because it leads to the successor with the highest minimax-value. This is the minimax decision at the root.

The definition of optimal play for MAX assumes that MIN also plays optimally – it maximizes the worst outcome for MAX. Let MIN does not play optimally. Then it is easy to show that MAX will do even better.

1.1 The Minimax Algorithm

The minimax algorithm given below computes the minimax decision from the current state.

```
function minimax-decision(state)
            returns an action
{
    v = max-value(state);
    return an action in successors(state) with value v;
}

function max-value (state)
            returns a utility value
{
    if terminal-test(state) then
```

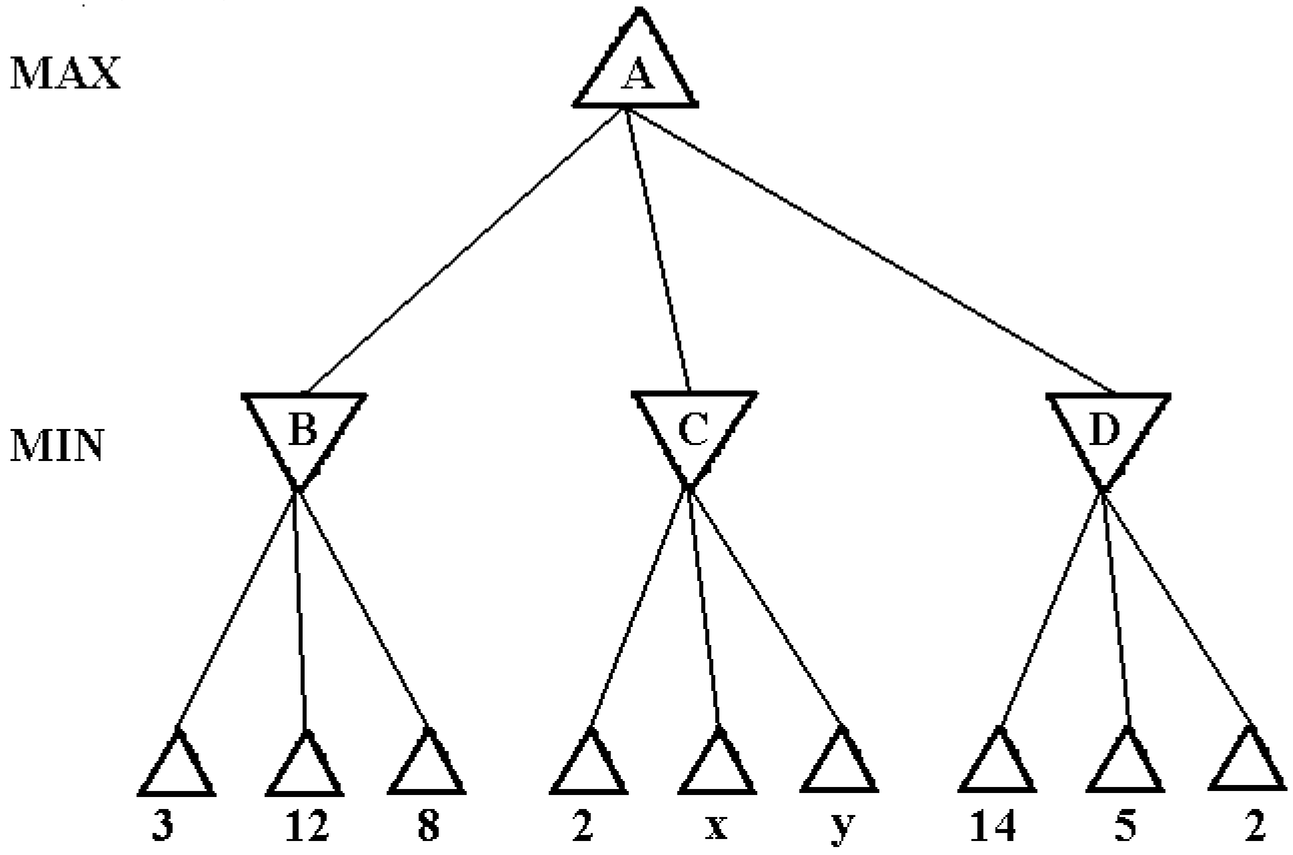
```
            return utility (state);
    v = -α;
    for a,s in successors(state)
    {
        v = max (v, min-value(s);
    }
    return v;
}

function min-value (state)
            returns a utility value
{
    If terminal-test (state) then
            return utility (state);
    v = +α;
    for a,s in successors (state)
    {
        v = min (v, max-value(s);
    }
    return v;
}
```

The above procedure uses a simple recursive computation of the minimax values of each successor state. The recursion proceeds all the way down to the leaves of the tree, and then the minimax values are backed up through the tree. For example, for the above game tree, the algorithm first recourses down to the three bottom left nodes, and uses the utility function on them to discover that their values are 3, 12 and 8 respectively. Then it takes the minimum of these values, 3, and returns it as the backed up value of node B. A similar process gives the backed up values of 2 for C and 2 for D. Finally, we take the maximum of 3, 2 and 2 to get the backed up value of 3 for the root node.

2 Alpha- Beta Pruning

The problem with minimax search is that a large number of states need to be examined. We can effectively cut it in half using a technique called alpha beta pruning. Here the trick is that it is possible to compute the correct minimax decision without looking at every node in the game tree.

Consider again the game tree shown below.



Let us calculate the minimax value at A. One way is to simplify the formula for minimax value. Let two successors of node C have values x and y. The value at the root node is given by

Minimax-value (A) = max [min (3, 12, 8), min (2, x, y), min (14, 5, 2)]

=max [3, min (2, x, y), 2)]

Suppose minimum of x and y is z.

Then,

Minimax-value (A) = max [3, min (2,z), 2]

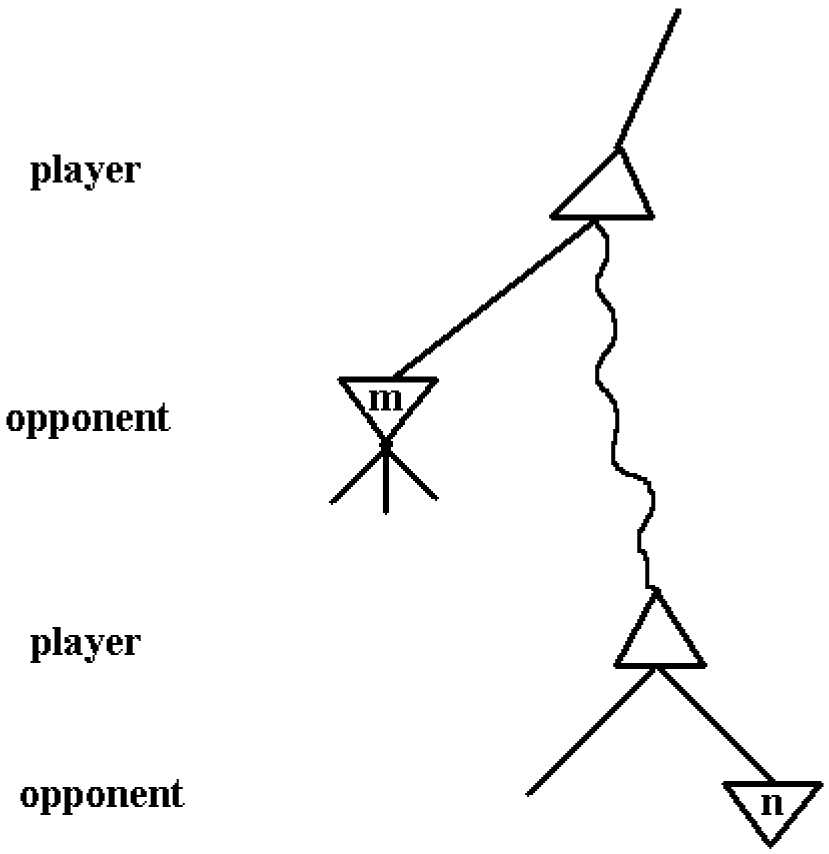
if z <=2.

= max [3, z, 2]

= 3

From this, it is clear that the value of the root node A and hence the minimax decision are independent of the values of the pruned leaves x and y.

Alpha beta pruning can be applied to trees of any depth, and it is often possible to prune entire sub trees rather than just leaves.



The general principle is this. Consider a node 'n' somewhere in the tree; such that player has a choice of moving to that node. If player has a better choice 'm' either at the parent node of 'n' or at any choice further up, then 'n' will never be reached in actual play. Alpha beta pruning gets its name from the following 2 parameters, α and β.

The algorithm for alpha beta search is given below.

```
function alpha-beta-search(state)
            returns an action
{
    v = max-value(state, -α, +α );
    return the action in successors (state) with value v;
}

function max-value(state,α,β )
            returns a utility value
{
    if terminal-test(state) then
```

```
                    return utility (state);
          v = -α;
          for a,s in successors(state)
          {
               v = max(v, min-value(s,α,β));
               if v >= β then
                    return v;
               α = max(α,v);
          }
          return v;
     }

     function min-value (state,α,β)
                    returns a utility value
     {
          if terminal-test(state) then
                    return utility (state);
          v = +α ;
          for a,s in successors(state)
          {
               v = min(v, max-value(s,α,β) );
               if v <= α then
                    return v;
               β = min(β,v);
          }
          return v;
     }
```

α – the value of the best (ie. Highest value) choice we have found so far at any choice point along the path for MAX.

β – the value of the best (ie. Lowest value) choice we have found so far at any choice point along the path for MIN.

Alpha-beta search updates the values of α and β as it goes along and prunes the remaining branches at a node as soon as the value of the current node is known to be worse than the current α or β value for MAX or MIN, respectively.

3 Imperfect Real Time Decisions

[Russel2006]

The minimax search searches the entire game tree, whereas the alpha beta algorithm helps us to prune large parts of it. But here also, alpha beta has to search all the way to terminal states for at least a portion of the search space. This is not practical. This is because moves in a two player game must be made within a reasonable amount of time.

It is proposed that programs should cut off the search earlier and apply a heuristic evaluation function to states in the search. Here the alpha beta search is altered in two ways.

1. The utility function is replaced by a heuristic evaluation function EVAL. It gives an estimate of the position's utility.
2. The terminal test is replaced by a cut off test.

3.1 Evaluation Functions

An evaluation function returns an estimate of the expected utility of the game from a given position. For centuries, chess players have developed ways of judging the value of a position. The performance of a game playing program is dependent on the quality of its evaluation function. How do we design good evaluation functions?

The computation of the evaluation function for a state must not take too long.

Consider the game of chess. If we cut off the search at non terminal states, the evaluation algorithm is uncertain about the final outcomes of those states.

Most evaluation functions work by calculating various features of the states. For example the number of pawns possessed by each side in a game of chess. Most evaluation functions compute separate numerical contributions from each feature and then combine them to find the total value.

For example, chess books give an approximate material value for each piece;

each pawn is worth 1,

a knight or bishop is worth 3,

a rook 5, and

the queen 9.

Other features such as “good pawn structure“ and “king safety” might be good. These feature values are then simply added up to obtain the evaluation of the position. This kind of evaluation function is called a weighted linear function because it is expressed as

$Eval(s) = w_1.f_1(s) + w_2.f_2(s) + \ldots\ldots\ldots\ldots\ldots\ldots.. + w_n.f_n(s)$

where each w_i is a weight and each f_i is a feature of the position.

For chess, the f_i could be the numbers of each kind of piece on the board, and w_i could be the values of the pieces. (1 for pawn, 3 for bishop etc..)

From this it is seen that the contribution of each feature is independent of the values of the other features. For example, we gave a value 3 to a bishop. It ignores the fact that bishops are more powerful in the end game. Because of this, current programs for chess and other games also use non linear combinations of features. For example, a pair of bishops might be worth more than twice the value of a single bishop, and a bishop is worth more in the end game than at the beginning.

Given the linear form of the evaluation, the feature and weights result in the best approximation to the true ordering of states by value.

3.1.1 Cutting Off Search

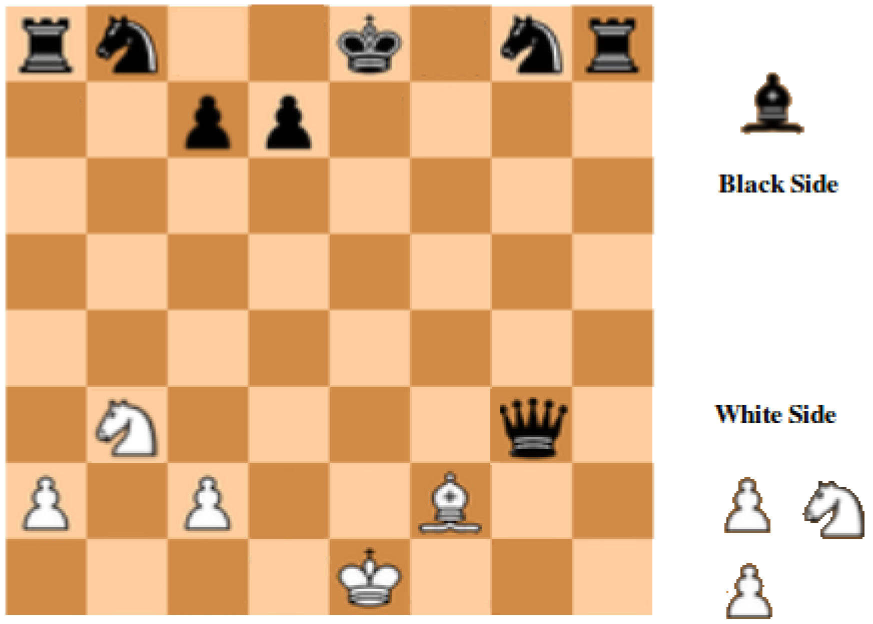
The next step is to modify alpha beta search so that it will call the heuristic EVAL function when it is appropriate to cut off the search. We use the following line,

```
if cutoff-test (state, depth) then
                                        return eval (state);
```

Here instead of terminal-test (state), we use cut-off-test (state, depth). The approach is to control the amount of search to set a fixed depth limit ’d’. The depth ’d’ is chosen so that the amount of time used will not exceed what the rules of the game allow.

Again consider the game of chess. An approach is to apply iterative deepening. When time runs out, the program returns the move selected by the deepest completed search.

But this approach can lead to errors. Consider again the simple evaluation function for chess. Let the program searches to the depth limit, reaching the position shown below.



white to move

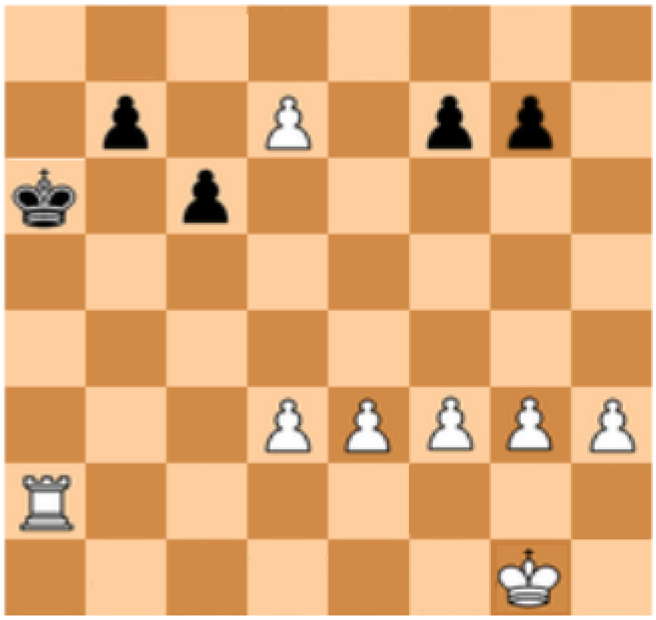
Here black is ahead by a knight and 2 pawns. It would generate the evaluation function value, and it declares that black is going to win. But from the fugure, white's next move will capture black queen. As a result, the position is really a won for white. But this can be seen only by looking ahead one more step.

Quiescent States

Hence, a more sophisticated cut off test is needed. The evaluation function should be applied only to quiescent positions. Quiescent position is a position which is unlikely to exhibit wild swings in value in the near future. Non-quiescent positions can be expanded further until quiescent positions are reached. This additional search is called quiescence search.

Horizon Effect

Another problem that can occur is horizon effect. Consider the following board configuration.



Horizon effect arises when the program is facing a move by the opponent that causes serious damage. In the above state, black is ahead in material, but if white advances its pawn from the 7^{th} row to the 8^{th}, the pawn will become a queen and create an easy win for white. The use of singular extensions has been quite effective in avoiding the horizon effect. A singular extension is a move that is clearly better than all other moves in a given position.

Forward Pruning

It is possible to do forward pruning. It means that some moves at a given node are pruned immediately without further consideration. Most humans playing chess only consider a few moves from each position. This is dangerous because sometimes the best move will be pruned away. So normally this is not applied near root. Forward pruning can be used safely in special situations. For example, when two moves are symmetric or equivalent, only one of them need be considered.

Combining all these techniques results in a program that can play chess. The branching factor for chess is about 35. If we used minimax search, we could look ahead only about 5 moves. Such a program can be defeated by an average human chess player who can plan 6 or 8 moves ahead. With alpha beta search, we get to about 10 moves. To reach grandmaster status, we need an extensively tuned evaluation function. We need a super computer to run the program.

4 State of the Art Game Programs

[Russel2006]

Some researchers believe that game playing has no importance in main stream AI. But game playing programs continue to generate excitement and a steady stream of innovations that have been adopted by a wider community.

4.1 Chess

Deep Blue Program

In 1997, the Deep Blue program defeated world chess champion, Garry Kasparov. Deep Blue was developed by Campbell, Hsu and Hoane at IBM. The machine was a parallel computer with 30 IBM processors and 480 VLSI chess processors. Deeep Blue used iterative deepening alpha beta search procedure. Deep Blue searched 126 million nodes per second on average. Search reached depth 14 routinely. The evaluation function had over 8000 features.

The success of Deep Blue shows that progress in computer game playing has come from powerful hardware, search extensions and good evaluation function.

Fritz

In 2002, the program Fritz played against world champion Vladimir Kramnik. The game ended in a draw. The hardware was an ordinary PC.

Checkers

Arthur Samuel of IBM, developed a checkers program. It learned its own evaluation function by playing itself thousands of times. In 1962, it defeated Nealy, a champion in checkers.

Chinook was developed by Schaeffer. Chinook played against world champion Dr. Tinsley in 1990. Chinook won the game.

Othello

Is a popular computer game. It has only 5 to 15 moves. In 1997, the Logistello program defeated human world champion, Murakami.

Backgammon

Garry Tesauro developed the program TD- gammon. It is ranked among the top 3 players in the world.

Go

It is the most popular board game in Asia. The programs for playing Go are Geomate and Go4++.

Bridge

Bridge is a multiplayer game with 4 players. Bridge Baron program won the 1997 bridge championship.

GIB program won the 2000 championship.

.

Discuss Alpha-Beta cut-off. (4marks)[MGU/Nov2010]

What are the game playing components? (4marks)[MGU/June2006]

What is minimax algorithm? How alpha-beta pruning improves the search process? (12marks)[MGU/June2006]

What is optimal strategy? (4marks)[MGU/Jan2007]

Implement the minimax algorithm, to play a tic-tac-toe game. (12marks)[MGU/Jan2007]

Bring out clearly the difference between game playing search methods and other AI search methods.

How does alpha-beta pruning improve the search process?

What is semantic net? (12marks)[MGU/June2007]

What is monotonicity and pruning? (4marks)[MGU/June2008]

Briefly explain Alpha-beta search algorithm. (4marks)[MGU/June2008]

Explain in detail Game playing and knowledge structures. (12marks)[MGU/June2008]

Compare Minimax search and Alpha-beta pruning.

Write notes on frames and semantic nets. (12marks)[MGU/June2008]

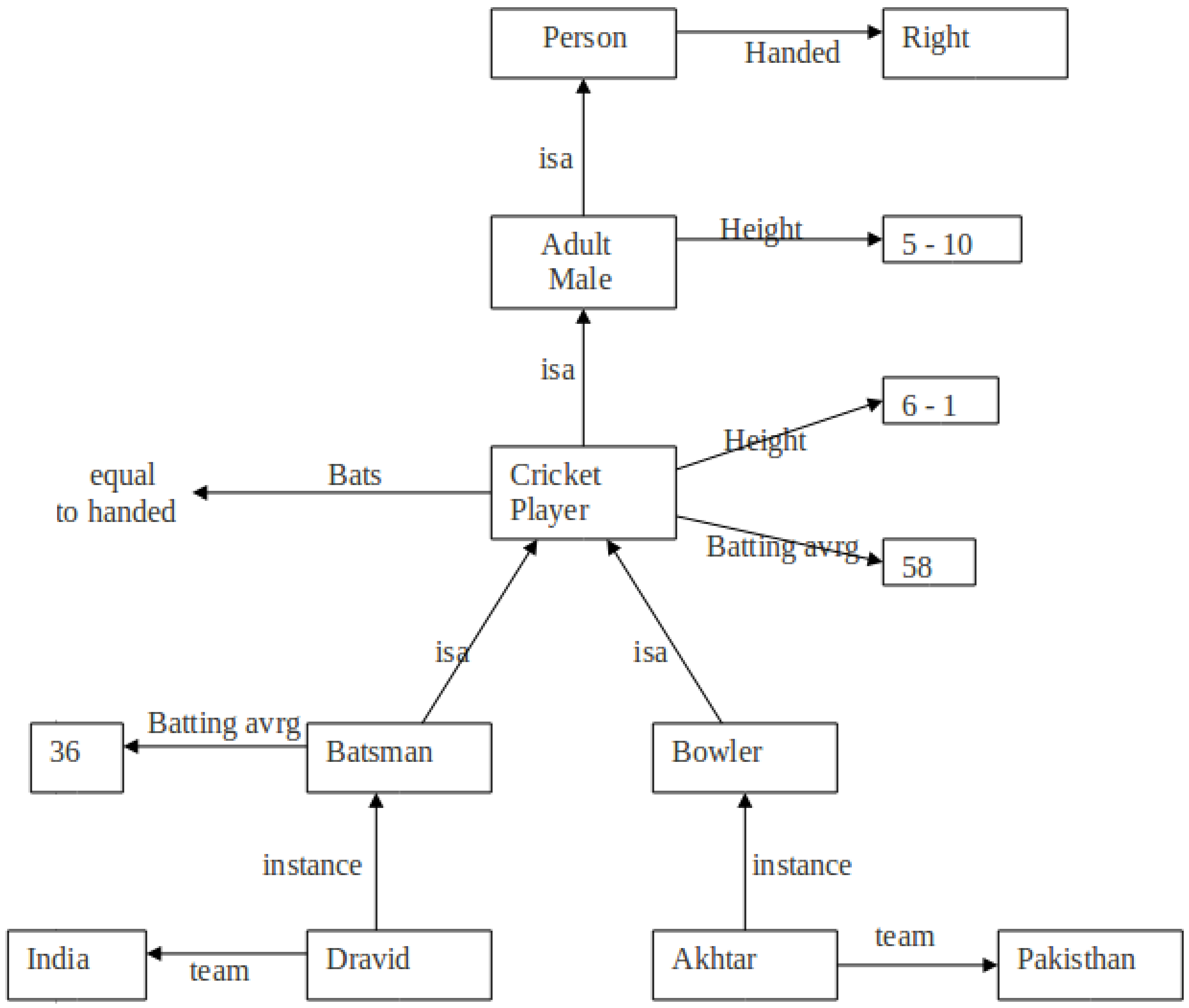
Part II. Knowledge Representation

Knowledge is very important in artificial intelligence systems. Knowledge is to be represented properly. It becomes clear that particular knowledge representation models allow for more powerful problem solving mechanisms that operate on them. Two of the knowledge representation schemes are semantic nets and frames.

5 Semantic Nets

[Rich1991] [Luger2005]

The following diagram shows a semantic network in which some knowledge is stored. Here it shows how some knowledge on cricket is represented.



Here boxed nodes represent objects and values of attributes of objects. These values can also be viewed as objects with attributes and values, and so on. The arrows on the lines point from an object to its value along the corresponding attribute line.

All of the objects and most of the attributes shown in this example have been chosen to correspond to the game of cricket. They have no general importance. The 2 exceptions to this are the attribute isa, which is being used to show class inclusion, and the attribute instance, which is being used to show class membership. Using this technique, the knowledge base can support retrieval of both of facts that have been explicitly stored and of facts that can be derived from those that are explicitly stored.

From the above semantic network, answers to the following questions can be derived.

Team (Dravid) = India

This attribute Dravid had a value stored explicitly in the knowledge base.

Batting average (Dravid) = 36

Since there is no value for batting average stored explicitly for Dravid, we follow the instance attribute to batsman and extract the value stored there.

Height (Akhtar) = 6-1

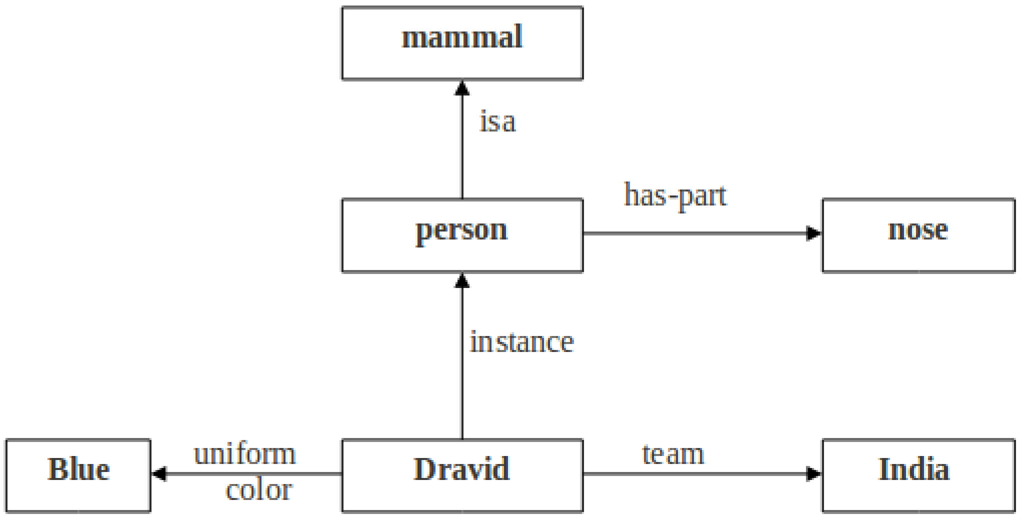
This represents another default inference. Notice here that because we get to it first, the more specific fact about the height of cricket players overrides a more general fact about the height of adult males.

Bats (Dravid) = Right

To get a value for the attribute 'Bats' required going up the isa hierarchy to the class 'Cricket Player'. But what we found there was not a value but a rule for computing a value. This rule required another value as input. So the entire process must be begun again recursively to find a value for 'handed'. This time it is necessary to go all the way up to 'person' to discover that the default value for handedness for people is 'right'. Now the rule for 'bats' can be applied, producing the result 'right'.

The main idea behind semantic nets is that the meaning of a concept comes from, the ways in which it is connected to other concepts. In a semantic net, information is represented as a set of nodes connected to each other by a set of labeled arcs, which represent relationships among the nodes.

Consider another example.



Intersection Search

One of the early ways that semantic nets were used to find relationships among objects by spreading activation out from each of 2 nodes and seeing where the activation met. This process is called intersection search.

Using this, it is possible to use the above semantic network to answer questions such as

"What is the connection between India and blue?"

Representing non-binary predicates

Some of the arcs from the above figure can be represented in logic as

isa (person, mammal)

instance (Dravid, person)

team (Dravid, India)

uniform-color (Dravid, Blue)

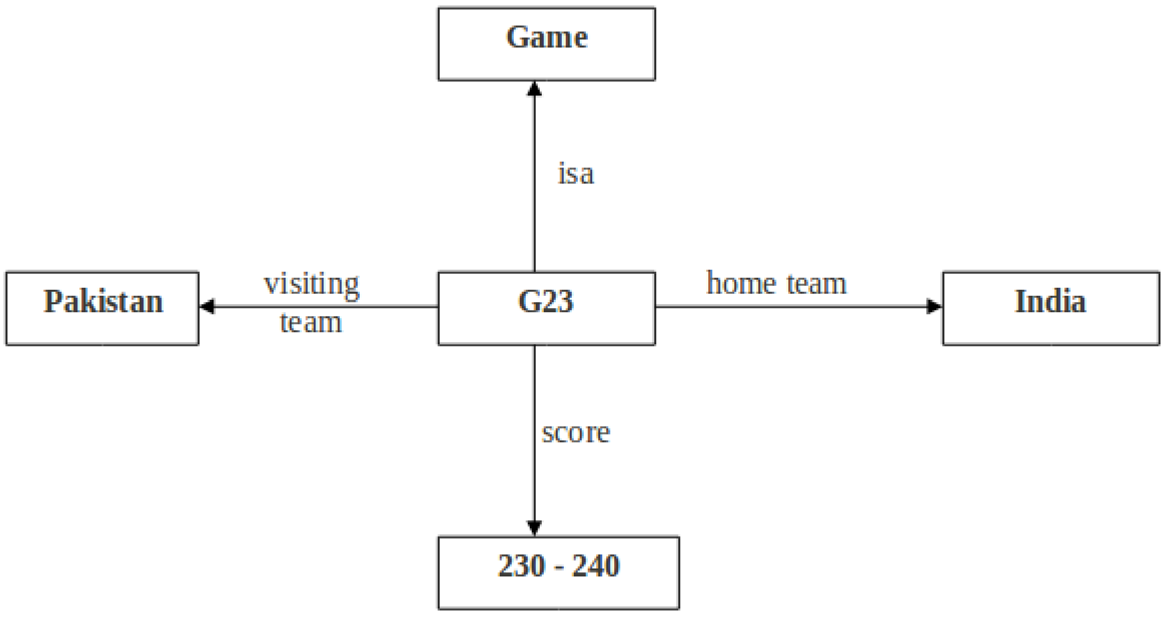
The above are binary predicates.

Three or more place predicates can also be converted to a binary form by creating one new object representing the entire predicate statement and then introducing binary predicates to describe the relationship to this new object of each of the original arguments.

For example, suppose we know that

score (Pakistan, India, 230-240)

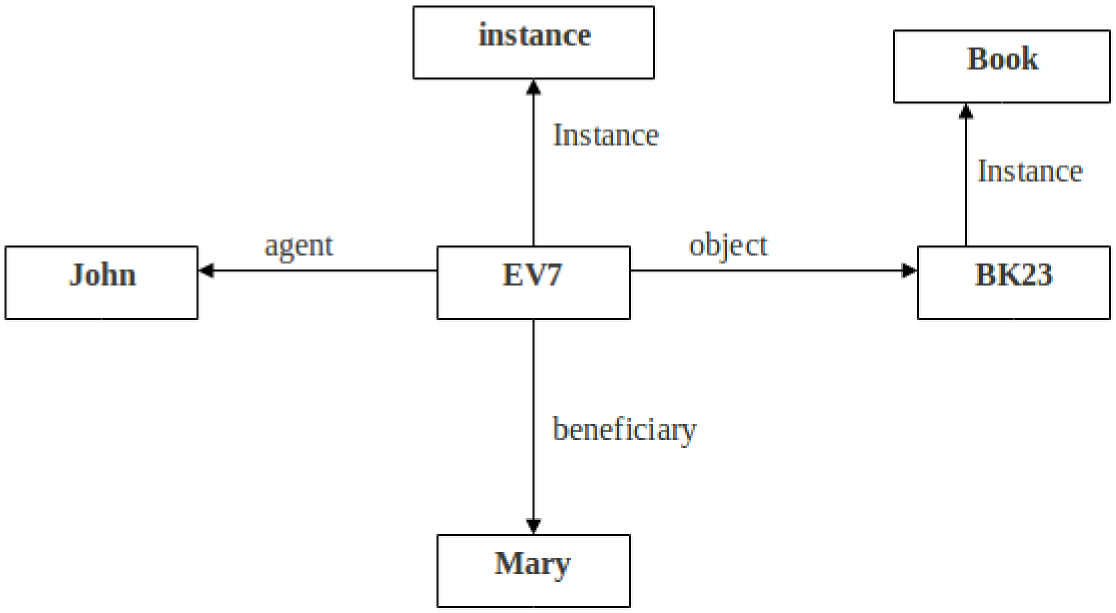
This can be represented as a semantic net by creating a node to represent the specific game and then relating each of the 3 pieces of information to it. This produces the semantic net shown below.



This technique is particularly useful for representing the contents of a sentence that describes several aspects of a particular event. The sentence

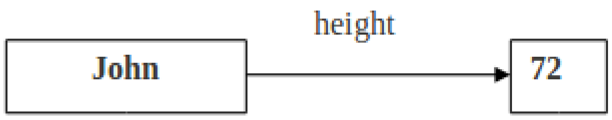
"John gave the book to Mary".

This is represented as follows.



Additional Points on Semantic Nets

There should be a difference between a link that defines a new entity and one that relates 2 existing entities. Consider the semantic net,

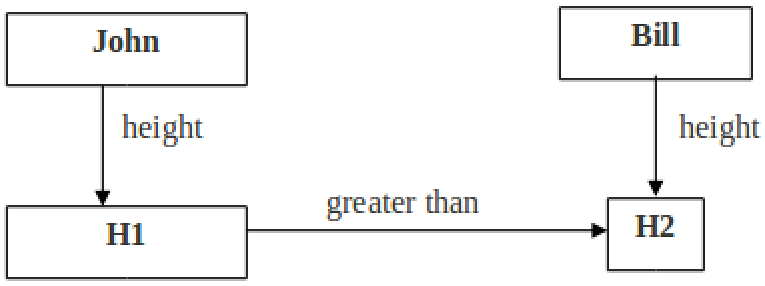


Both nodes represent objects that independently of their relationship to each other.

To represent the fact that

"John is taller than Bill"

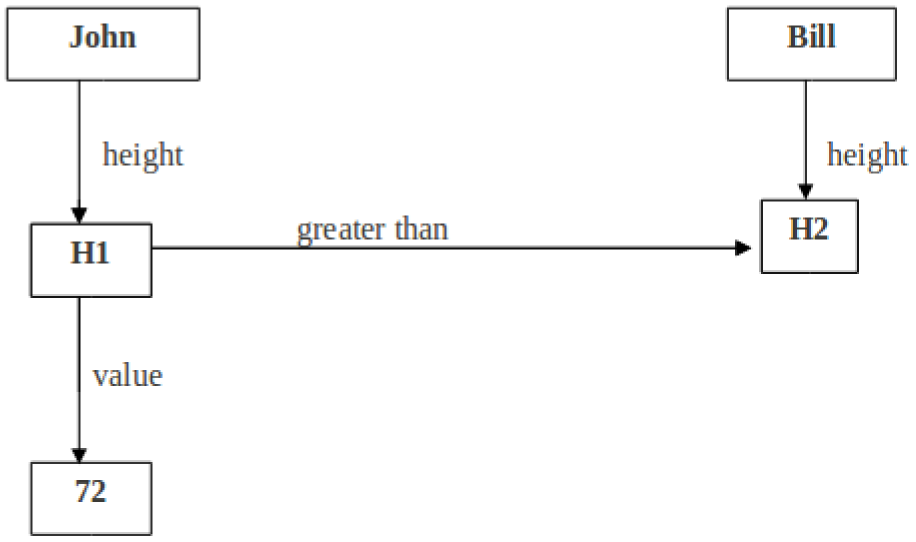
using the net



The nodes H1 and H2 are new concepts representing John's height and Bill's height respectively.

We may use the following net to represent the fact

"John is 6 feet tall and that he is taller than Bill".

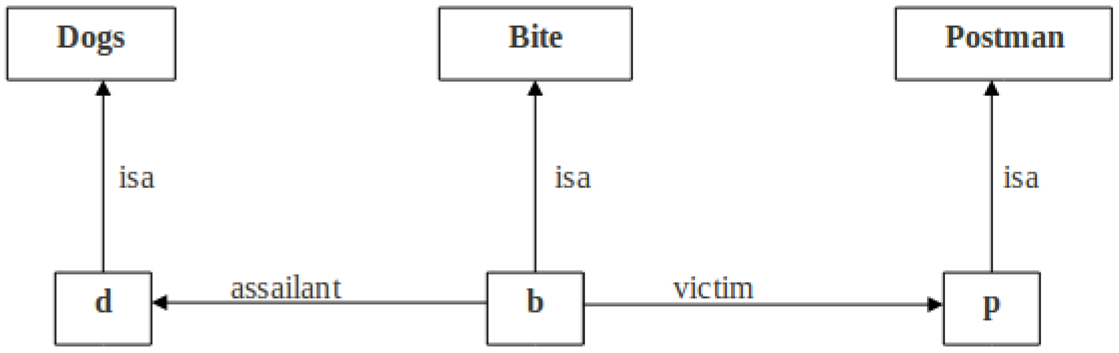


The operations on these nets can exploit the fact that some arcs such as height, define new entities, while others, such as greater than and value describe relationships among existing entities.

Partitioned Semantic Nets

Consider the fact

"The dog bit the postman".



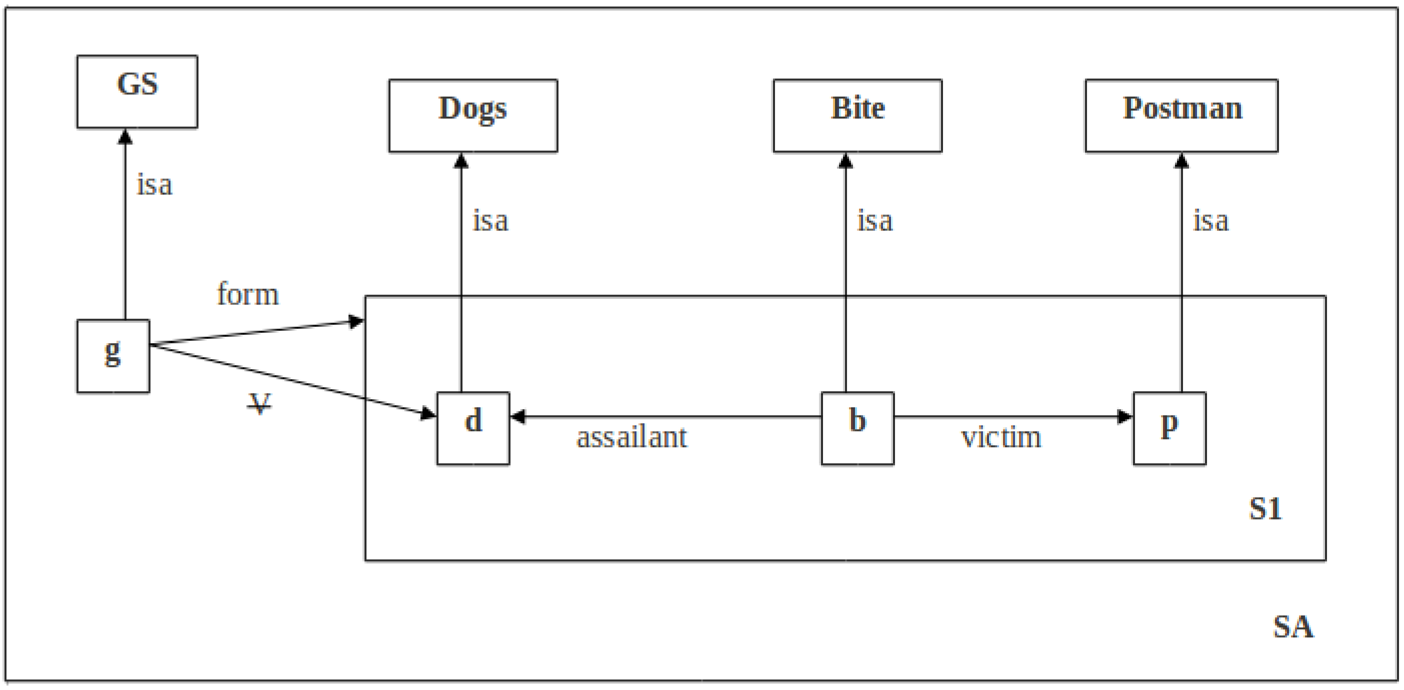
Consider another fact

“Every dog has bitten a postman”.

This is a quantified expression. That is

$$\forall x : Dog(x) \rightarrow \exists y : postman(y) \cap Bite(x, y)$$

Here we need to partition the semantic net into a set of spaces, each of which corresponds to the scope of one or more variables. To represent this fact, it is necessary to encode the scope of the universally quantified variable x. This can be done using partitioning as shown below.



Node g is an instance of special class GS of general statements about the world. (ie. With universal quantifier, ∀. Every element of GS has at least 2 attributes.

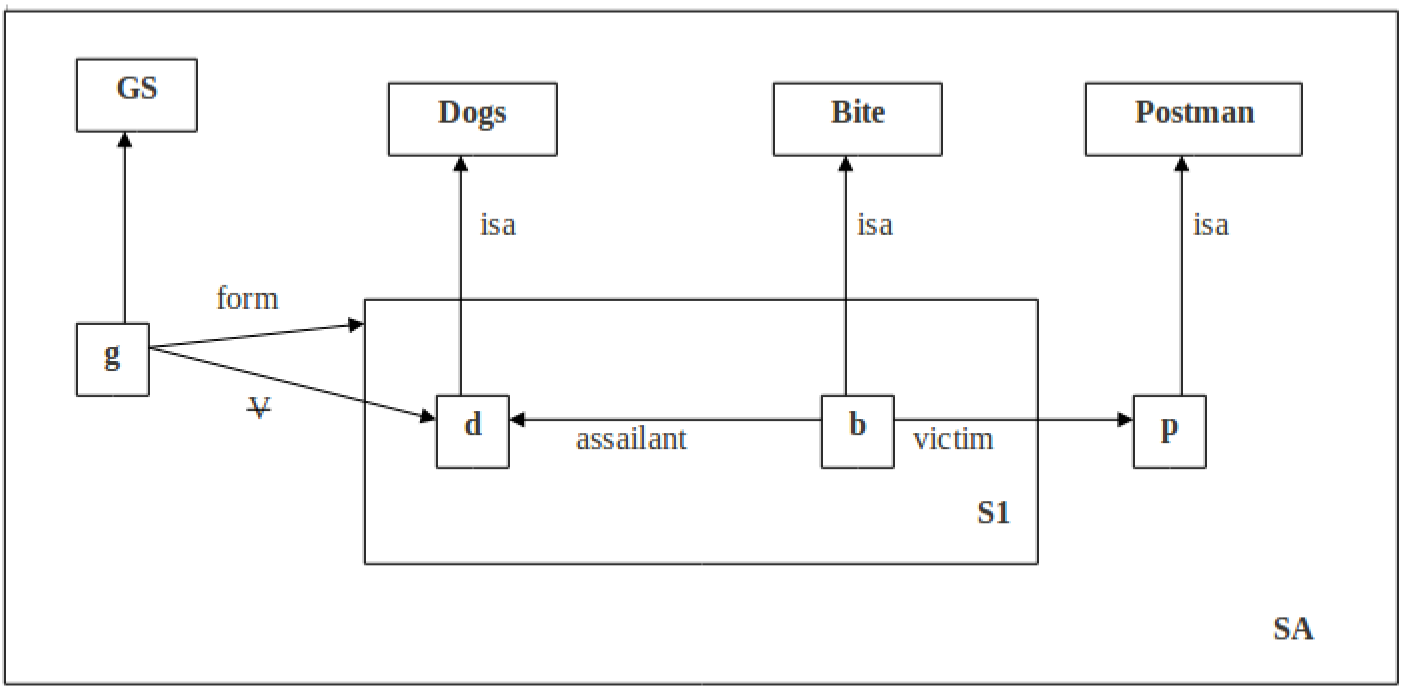
1. A form which states the relation to be asserted.

2. One or more ∀ connections, one for each of the universally quantified variables.

In this example, there is only one such universally quantified variable, ie d.

Consider another fact,

“Every dog has bitten the postman”.

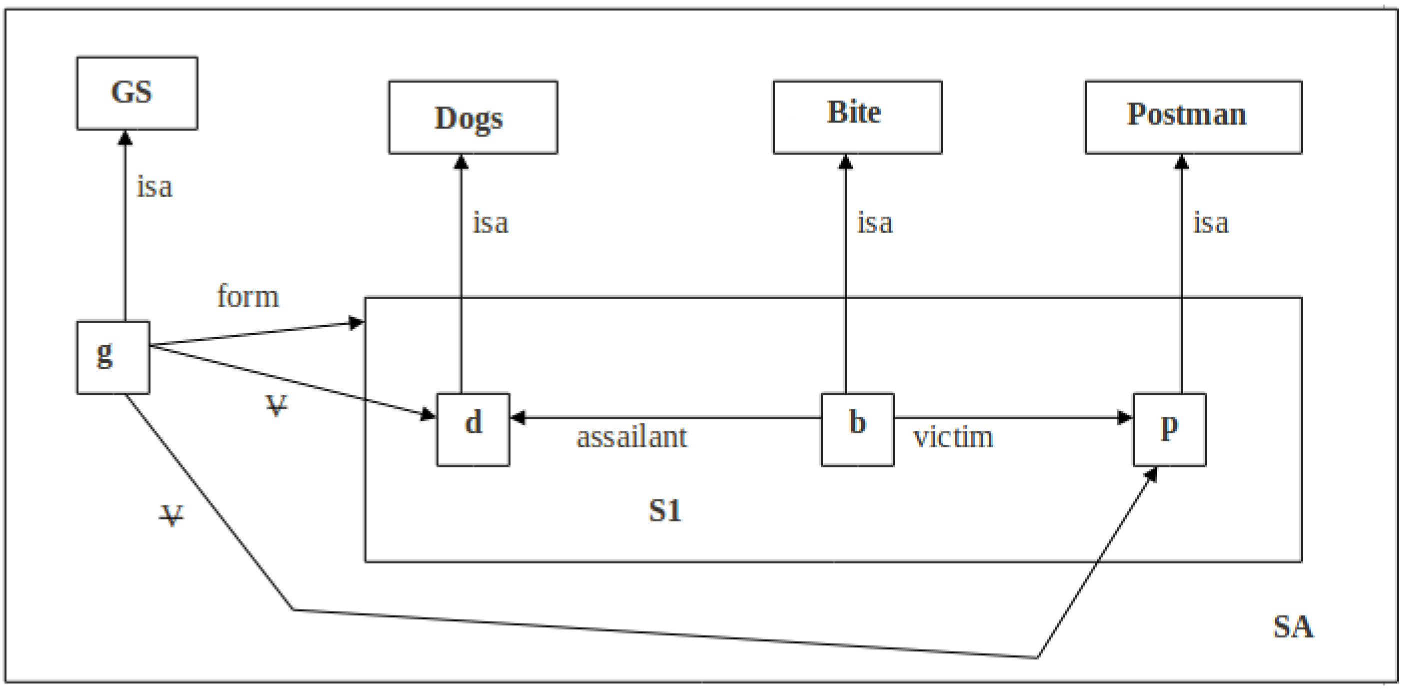


In this, node C representing the victim lies outside the form of the general statement. It is not viewed as an existentially quantified variable. Instead, it is interpreted as standing for a specific entity.

Consider another fact

“Every dog has bitten every postman”.

The semantic net for this is shown below.



In this case, g has 2 links, one pointing to d which represents any dog, and one pointing to m, representing any postman.

In the above net, space S1 is included in space SA.

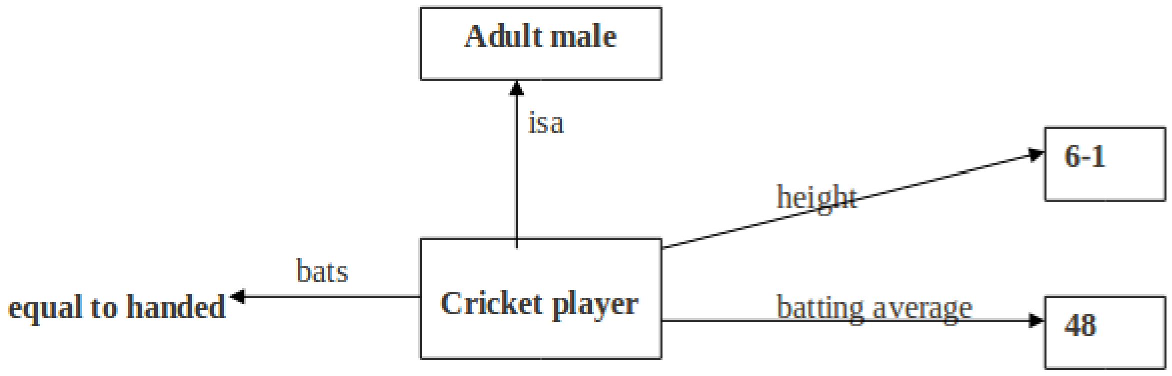
Conclusion The idea of semantic net is to simply represent labeled connections among entities. But as we expand the range of problem solving tasks, the semantic net begins to become more complex. It is necessary to assign more structure to nodes as well as to links. As a result, frames were evolved.

6 Frames

[Rich1991] [Luger2005]

A frame is a collection of attributes (slots) and associated values that describe some entity in the world.

Consider a node in the semantic network below.



The following is the frame corresponding to cricket player.

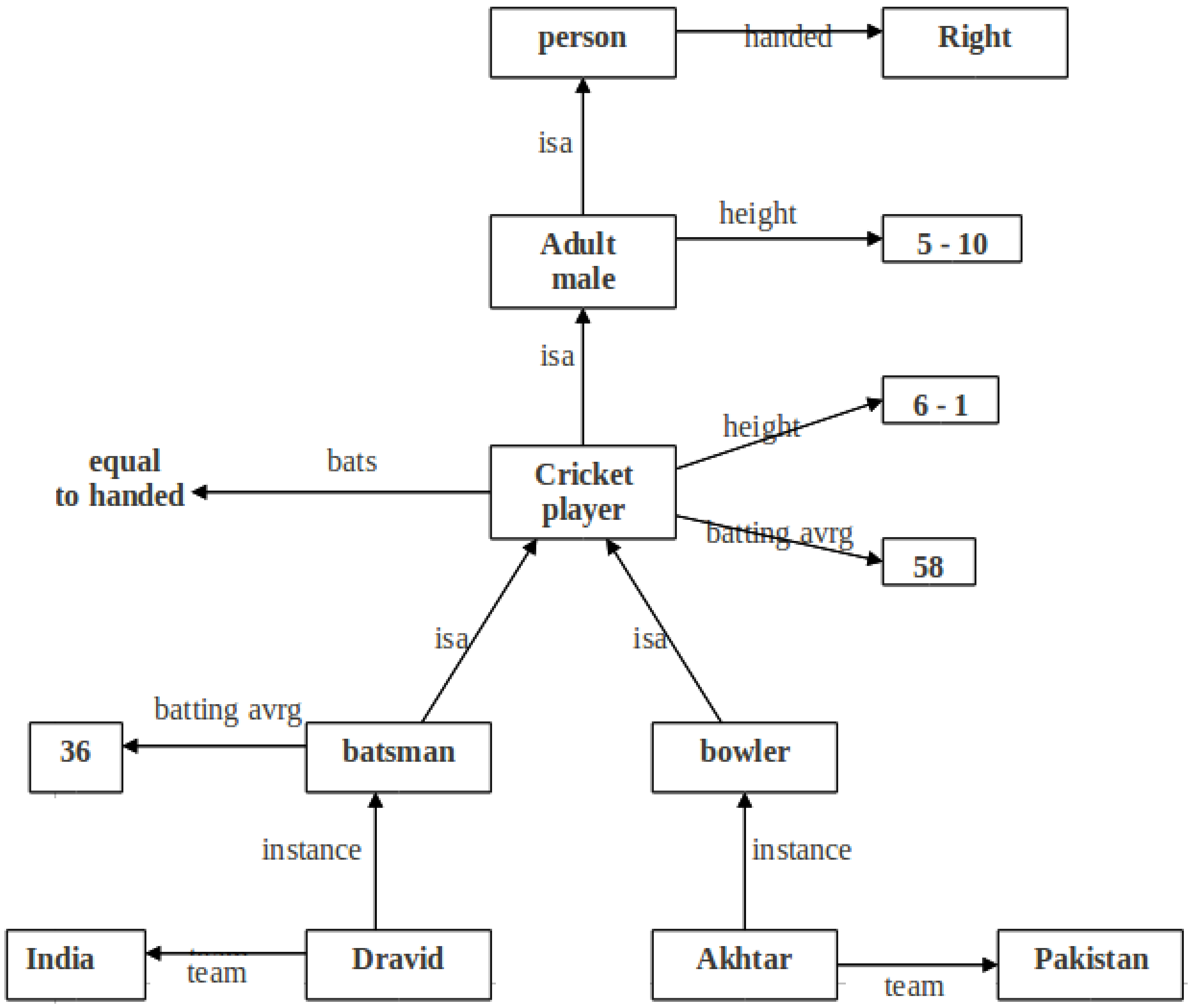
Cricket player

isa	:	Adult male
bats	:	(EQUAL handed)
height	:	6 1
batting average	:	48

There is so much flexibility in this frame representation, since this can be used to solve particular representation problems.

A single frame taken alone has not much use. Instead, we build frame systems out of collections of frames that are connected to each other. Frame systems are used to encode knowledge and support reasoning.

Consider the semantic network shown below.



The frame system corresponding to the above semantic network is shown below.

person

isa	mammal
*handed	right

adult male

isa	person
*height	5 – 10

cricket player

isa	adult male
*height	6 – 1
*bats	(equal to) handed
*batting avrg	48
*team	
*uniform color	

batsman

isa	cricket player
*batting avrg	52

Dravid

instance	batsman
team	India

bowler

isa	cricket player
batting avrg	40

Akhtar

instance	bowler
team	Pakistan

In this example, the frames ‘person’, ‘adult male’, ‘cricket player’, ‘bowler’, ‘batsman’ are all classes.

The frames ‘Dravid’, ‘Akhtar’, are instances.

Set theory provides a good basis for understanding frames. Here classes can be called as sets. Instances mean elements of a set.

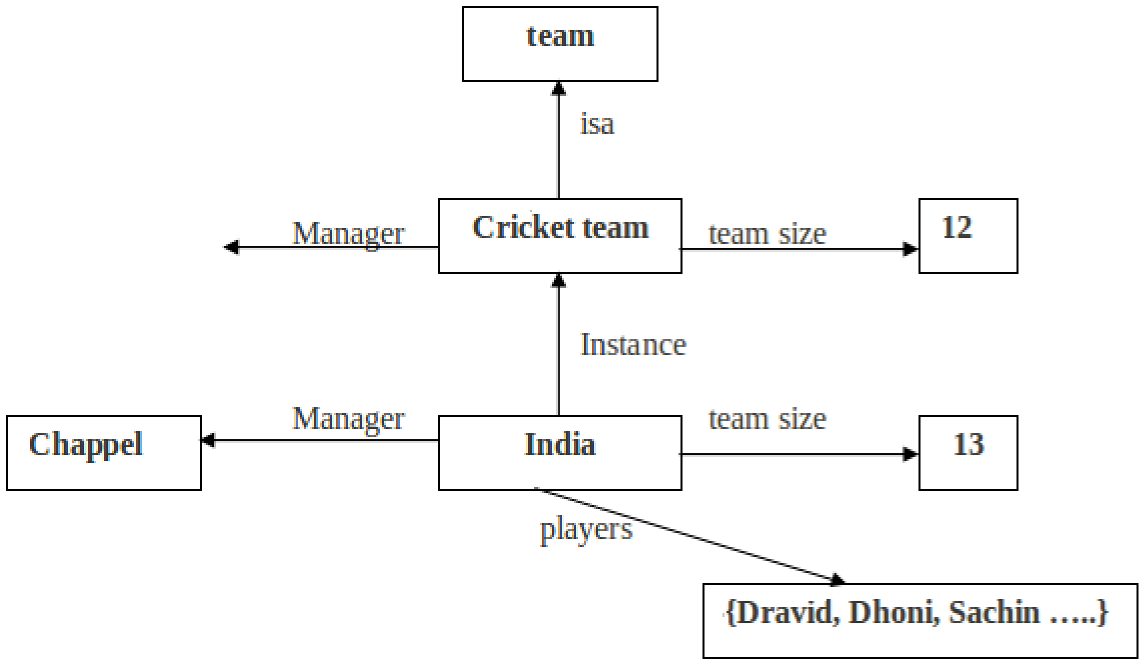
The 'isa' relation is in fact a subset relation. The set of adultmales is a subset of all persons. The set of cricket players is a subset of the set of adult males and so on.

Our instance relation corresponds to the relation element of. 'Dravid' is an element of the set of 'batsman'. Thus he is an element of all the supersets of 'batsman', including 'cricket player' and 'person'.

There are 2 kinds of attributes associated with a class (set). They are attributes about the set itself and attributes that are to be inherited by each element of the set.

The attributes that are inherited by each element of the set is indicated by an asterisk (*). For example, consider the class cricket player. Only one property of it as a set was seen. It is a subset of the set of 'adult male'. 5 properties are listed that all cricket players have (height, bats, batting average, team and uniform color), and default values for the first three of them were specified.

Consider the net shown below.



Here the distinction between a set and an instance may not seem clear. For example, 'India' is an instance of cricket team could be thought of as a set of players. Here notice that value of the slot 'players' is a set. Suppose we want to represent 'India' as a class instead of an instance. Then its instances would be the individual players.

A class can be viewed as 2 things simultaneously:

A subset (isa) of a larger class that also contains its elements and

An instance of a class of sets from which it inherits set level properties.

It is useful to distinguish between regular classes, whose elements are individual entities and meta classes, which are special classes whose elements are themselves classes. A class is now an element of (instance) some class as well as a sub class (isa) of one or more classes.

Consider the example shown below.

class

instance	:	class
isa	:	class

team

instance	:	class
isa	:	class
*team size	:	

cricket team

instance	:	class
isa	:	team
*team size	:	14
*manager	:	

India

instance	:	cricket team
isa	:	cricket player
team size	:	13
manager	:	chapel
*uniform color	:	blue

Dravid

instance	:	India
instance	:	batsman
uniform color	:	blue
batting avrg	:	53

The most basic meta class is the 'class' all classes are instances of it. In the example, 'team' is a sub class of class

'class' and 'cricket team' is a sub class of 'team'.

'India' is an instance of 'cricket team'. It is not an instance of 'class' because its elements are individuals, not sets. 'India' is a sub class of 'cricket player', since all of its elements are also elements of that set.

Finally, 'Dravid' is an instance of 'India'. This makes him also, by traversing 'isa' links, an instance of 'cricket player'. We had a class 'batsman', to which we attached the fact that 'batsman' have above average batting averages. To allow that here, we make 'Dravid' an instance of 'batsman' as well. He thus inherits properties from both 'India' and from 'batsman', as well as the classes above these.

Other class relations

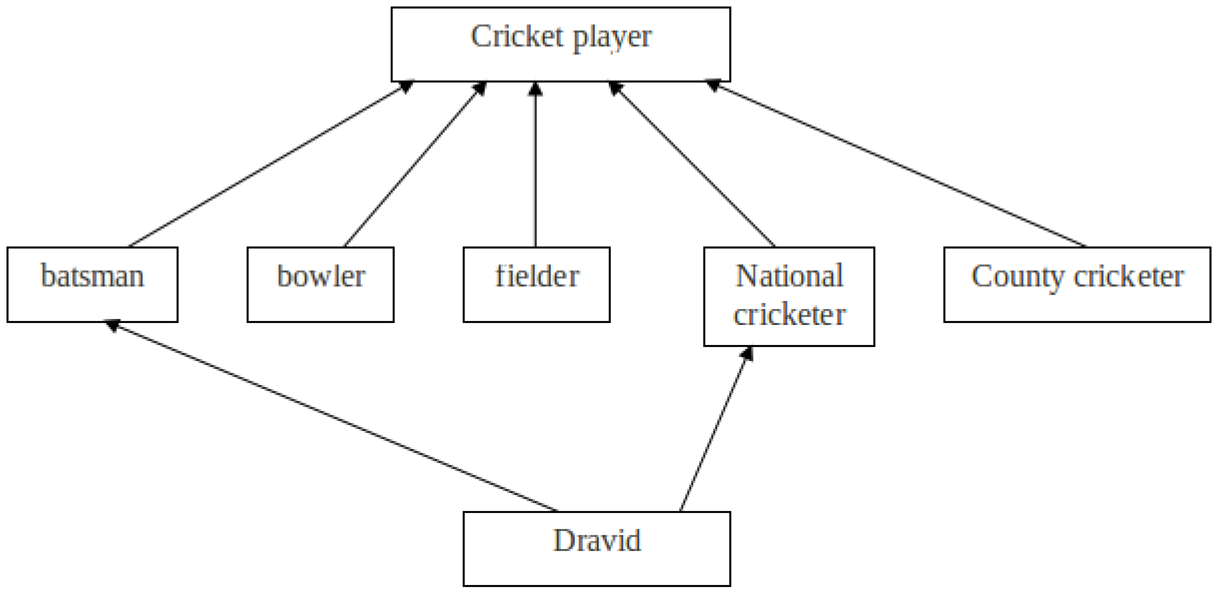
Classes can be related in many ways.

One such relationship is 'mutually disjoint with'. It relates a class to one or more other classes that are guaranteed to have no elements in common with it.

Another relationship is 'is covered by'. It relates a class to a set of subclasses, the union of which is equal to it. If a class 'is covered by' a set S of mutually disjoint classes, then S is called a partition of the class.

The examples are shown below.

Consider the classes in the following semantic net.



Cricket player

is covered by	{ batsman, bowler, fielder }
	{ national cricketer, county cricketer}

batsman

isa	:	cricket player
mutually disjoint with	:	{ bowler, fielder }

bowler

isa	:	cricket player
mutually disjoint with	:	{ batsman, fielder }

fielder

isa	:	cricket player
mutually disjoint with	:	{ batsman, bowler }

Dravid

instance	:	batsman
instance	:	national cricketer

6.1 Frame Languages

To represent knowledge, we are using frames. A number of frame oriented knowledge representation languages have been developed. Examples of such languages are

KRL,

FRL,

RLL,

KL-ONE,

KRYPTON,

NIKL,

CYCL,

Conceptual graphs,

THEO and

FRAMEKIT.

.

What is meant by semantic net? (4marks)[MGU/Nov2010]

Describe knowledge structures and imperfect decisions. (12marks)[MGU/Nov2010]

Construct semantic net representations for the following:

Pomepeian (Marcus)

Blacksmith (Marcus)

Mary gave the green flowered vase to her favourite cousin. (12marks)[MGU/Nov2010]

What is a semantic net? (4marks)[MGU/June2006]

What is a frame? How it is used to represent complex facts? (12marks)[MGU/June2006]

What is a frame? Give an example. (4marks)[MGU/Jan2007]

What is a semantic net? Construct partition semantic net representations for the following?

i. Every batter hit a ball.

ii. All the batters like the pitch. (12marks)[MGU/Jan2007]

Define and explain a Frame. (4marks)[MGU/June2007]

Construct semantic net representation for “Pompeian (Marcus), Blacksmith (Marcus). (4marks)[MGU/June2007]

Explain knowledge representation using frames. Give an example for a typical Frame. (12marks)[MGU/June2007]

References

.

1. Rich, E; Knight, K. (1991). Artificial Intelligence. TMH.
2. Luger, G. (2005). Artificial Intelligence. Pearson Education.
3. Russel, S; Norvig, P. (2006). Artificial Intelligence. Pearson Education.