THE USE OF FRAMES IN KNOWLEDGE-BASED SYSTEMS

A thesis
submitted in partial fulfilment
of the requirements for the Degree of
Master of Science in Computer Science
at Massey University by

WILLIAM JOHN TEAHAN

Massey University
1986

ABSTRACT

The general aim of this study was to investigate the use of frames as a means of representing knowledge in computer knowledge-based systems. This thesis examines the application of frames to two particular situations, the playing of an opening bid in Bridge, and the recognition of birds from field observations. The Frame Representation Language FRL was used in the implementation of the two different systems.

Three aspects of frames are investigated : the problems of matching two different frames; the problems of structuring frame systems for searching; and the problem of improving the interface between the frame system and the user of the knowldege base. A comparison is also made of frames with other methods of knowledge representation such as production systems and semantic networks. Finally, further areas of research into the use of frames are suggested such as the extension of frame matching, research into the aspects of knowledge representation and application of frames to specific problems.

ACKNOWLEDGEMENTS

Thanks to my supervisor Ray Kemp for providing help and suggestions throughout the preparation of this thesis.

CONTENTS

1 INTRODUCTION.

An area of increasing importance in Artificial Intelligence in recent years has been the problem of how to represent knowledge on the computer. Some examples of methods that have been developed are production systems and frames. This thesis investigates the features of frames in particular and how they can be used to construct knowledge bases which can be incorporated into expert systems.

Marvin Minsky, in "A Framework for Representing Knowledge", first proposed the theory of frames in 1974. He defined a frame as being "a data-structure for representing a stereotyped situation." Some examples of such situations are entering a room, driving to work and watching television. The frame representation language FRL was developed in 1977 by Goldstein and Roberts to implement the theory of frames. The language is mostly declarative in that it depends on data structures as opposed to procedures for the definition of the frames.

Each frame consists of various types of information described by slots. Each slot contains any number of facets which define how the information in the slot is to be used. Each facet in turn consists of values which contain the actual data or information that is being represented. Attached to each data item may also be several comments. Each comment consists of a label and a message.

Related frames are organised into frame systems. Each frame of the system share the same slots so that the same functions can be applied to all the frames in the system. The frame systems can also be structured into information retrieval networks which provide alternative frames to search when a frame fails to match a particular

situation. Diagram 1.1 illustrates the hierarchical structure of a frame system.

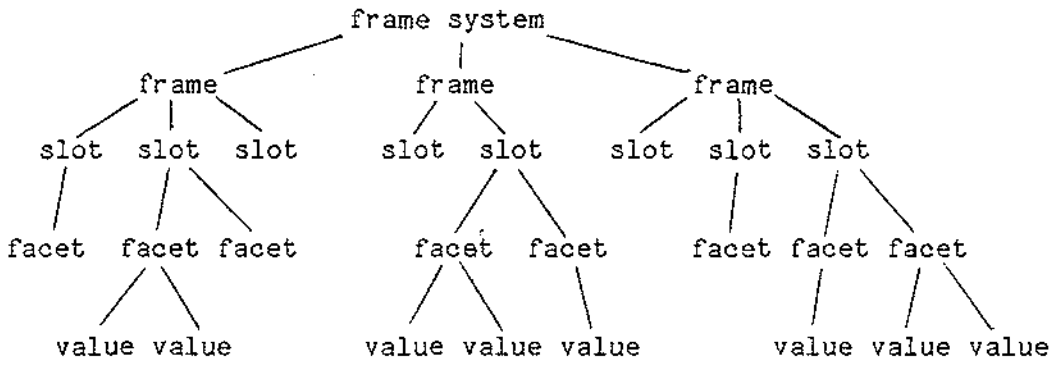

Diagram 1.1 Structure of a frame system.

Three basic instructions, FGET, FPUT and FREMOVE describe the main operations for obtaining, inserting and removing information stored in the frames. Four important features of frames are defaults, demons, inheritance and requirements.

Each slot in a frame can have a value facet which describes the data values associated with each slot. Alternatively, the slot can have a default facet which is used when there is no value facet exists. This allows for general assumptions about the information to be stored in the frames which can then be displaced at a later date when more specific data arrives that better fits the current situation.

Functions that are activated automatically when a specific situation occurs are called demons. The demons are expressed as functions attached to various facets in each slot of a frame. Examples of demons used in frames are if-added, if-removed and if-needed demons which are activated whenever the information in the slot is added to, removed or needed. These demons can activate further demons and hence a simple change or reference to a frame can initiate a whole series of

actions that may affect other frames in the system.

Another powerful feature of frames is the ability to use information from other frames through inheritance. Related frames in the system can be linked through an AKO (A-KIND-OF) slot which indicate that a particular frame has similar properties to the related frame. This means that information can be 'inherited' through the pathway and does not need to be stored in the frame itself.

Values within a slot may be restricted by certain requirements that describe the allowable values for the slot. Frames in an AKO hierarchy can therefore be classified as being generic, where general requirements are used to describe the frame, or individual, where more specific values are used.

The particular implementation of FRL used throughout this thesis was developed at MIT on the PRIME 750 computer and is incorporated into the V-mode LISP language available on the computer.

Two particular applications of frames have been investigated and are referred to throughout this work. These are :

1. finding an opening bid in bridge;
2. recognition of birds from field observations.

The first application relates to the problem of finding an appropriate opening bid in bridge such as 1 Spade or 2 Clubs given the cards in the player's hand. The bidding system used in the implementation is Acol. The second application involves the

recognition of birds from field observations of characteristics such as habitat and appearance. The set of birds used for this application is arbitrarily limited to the common town and pasture birds in New Zealand. The names of the frame systems developed for these two applications are, respectively, the Bridge System and the Bird Recognition System. The choice of the two different applications are sufficiently diverse to examine the versatility of frames when applied to different kinds of information.

The work described in the following chapters falls into three main areas of investigation :

1. matching two different frames;
2. structuring frame systems for search;
3. improving the interface between the knowledge base and the user.

The first area deals with the problem of comparing two separate frames with different information to see if they match. The use of requirements to express a generic frame against which an individual frame is matched is described. An alternative matching scheme is also proposed which uses matching functions attached to a generic frame to define how the frame is to be matched. The need for matching demons and a method to express how a frame is to be matched is also described.

The second area of research is the problem of structuring frame systems for search to find the particular frame or frames that match a given frame. This involves ordering the frame system in some manner or linking the frames in the system by reference to other frames. Types of structures investigated are : linear, set, hierarchical and network

structures.

The third area looks at how the frame systems can be organised so that the interface with the user can be improved. For improving the presentation of frames to the user, the use of attached functions to display and enter values in a frame is proposed. Various methods of improving the search of the knowledge base for the user are described, such as using matching demons for tracing and interactive match functions for querying the user. The problems associated with allowing the user to create or modify the frame systems are also examined.

After the description of these three areas of research, a comparison is made of frame-type structures with other systems of knowledge representation such as decision-trees, production systems and procedures. Following this is a description of improvements, current developments and useful lines of investigation on frame systems in general. Various further applications of the frame systems to other areas of knowledge are also explored.

2 MATCHING FRAMES.

2.1 Problems with Matching Frames.

There are many problems involved in trying to match two separate frames of information. Often it is necessary to match a set of specific information against a set with a more generalised framework. In bridge for example, the known information is the cards in the hand and from this can be determined further facts such as the division of the suits and the honour points. The player then tries to match these cards against more generalised requirements, such as a balanced hand distribution for no-trumps.

Some problems that arise from matching two different frames are:

1. Only parts of each frame are relevant to the match. A straight match between each slot in both frames is not possible since the number of slots in each is seldom equal.

2. A matching of specific data against more generalised data is required. The data may have to be in a particular range or be of a certain type for a match to occur.

3. Different matching functions between each slot of each frame are necessary because of the different information being matched.

4. In some cases, various logical combinations of the slots for matching is needed using or and and conditions.

5. More complex matching between slots is also required. This may involve using a function to express the types of data that are required.

For example, in playing bridge a player receives the set of cards shown in Figure 2.1.1.

spades	9 5 3 2
hearts	A Q 9
diamonds	K J 7
clubs	K Q 6

Figure 2.1.1 An opening hand in bridge.

From this he can build up a frame of known facts such as those shown below.

division of suits	4 spades, 3 hearts, 3 diamonds, 3 clubs
honour points	15
suit strength	no biddable suits
quick tricks	3
playing tricks	2
balance	well balanced

Figure 2.1.2 Analysis of bridge hand.

In searching for an opening bid, he has to match up the various conditions that are required for each bid. For a 'prepared' bid (which this hand fits), the conditions are :

(i) the distribution of the suits is 4-3-3-3

(ii) the longest suit is not biddable

(iii) the honour points are from 15 to 19.

The matching problems outlined above are demonstrated in this example. Only certain slots are relevant to the match, in this case, the division of the suits and the honour points. Also, the honour points have to be within a certain range while the suit distribution has to be of a certain type. More complicated requirements are demonstrated by the need to define what is 'biddable' by a function. The FRL frame devised for the prepared bid in the Bridge System is given in section 2.2.

Another example is the problem of bird recognition. Here, a description of an unknown bird is built up from characteristics observed in the field. A typical field description of a bird is shown in Figure 2.1.3.

-medium-sized bird

-found in the Wairarapa, near a marsh in open farmland

-the bird is mainly bluish, and has a prominent crimson bill

-laboured take-off with dangling legs during flight

-food consists mainly of plants and insects

Figure 2.1.3 Field description of a Pukeko.

To find out which bird fits these field observations, the bird watcher has to search through a list of known bird descriptions. The description for a Pukeko is shown in Figure 2.1.4. Reference : [Marshall, Kinsky and Robertson, 1972].

This example also demonstrates the problems outlined above. Only certain features of the bird are given in the field description. Each feature of the bird is expressed by different types of information. For example, the size of the bird may be stated as being small, medium or large, or as the actual size in cm. Features such as appearance and behaviour that are expressed by English descriptions require more complicated matching functions. The FRL frame devised for the Pukeko in the Bird Recognition System is given in section 3.3.

Another problem with matching frames occurs when the result of a match is indeterminate. In the Bridge System where the problem was restricted to finding an opening bid, the problem of unknown information does not arise. However, in the Bird Recognition System, gaps may occur in the field decription of a bird, and it is often necessary to match incomplete information.

There are various sources that might lead to an indeterminate match :

1. Missing slots in either of the two frames being matched;
2. Incomplete description for a slot value;
3. Indeterminate result because of the matching function.

Field Characteristics:

-size 51 cm.

-bright blue and black.

-red bill, frontal shield and legs

-often flicks tail to show prominent white under tail coverts

-runs fast, is a reluctant flier, flying heavily with dangling legs.

Distribution and Habitat:

-throughout New Zealand

-marshes, swamps, lagoons, lakes, riverbanks, with raupo and scrub cover

-often seen in open near wetlands

-feeds on wide variety of plant matter, snails, insects, sometimes eggs of other ground nesting birds

Figure 2.1.4 Known characteristics of a Pukeko.

In the Bird Recognition System, for example, many of the features of the bird such as nest location and number of eggs found in the nest may not have been included in the field observations. Also, the descriptions of some of the features such as appearance may also be incomplete with only a general description such as "yellow bill, black body" being given. Therefore, it is necessary when designing a matching scheme to take into account the effect of unknowns.

2.2 Use of $REQUIRE facets in Generic Frames.

Since frame matching is an integral part of a frame system, it is necessary to express the information relating to the matching within the frame itself. It is also important to have a general matching format that provides a straightforward and understandable means of describing a wide range of information.

Within the FRL structure, one method of matching frames is to use the $REQUIRE facet in a generic frame to express the conditions for the matching of each slot. The values related to each slot within the generic frame become expressed by functions or predicates which are evaluated by the matching procedure and return true, false or unknown depending on a successful match. The generic frame matches with an individual frame if all the requirements within the frame evaluate to true.

For the slot functions to be completely general and independent of the frame being matched, the source of the information needs to be specified within the matching conditions themselves. This is done by using the global variables :FRAME, :SLOT and :VALUE which contain the names of the frame and slot, and the slot values being matched. These variables can either be passed directly as explicit parameters, or be used implicitly within the functions themselves.

The FRL structure for a 'prepared' frame is shown in Figure 2.2.1.

```
(deframe prepared-frame
    (ako
         ($value
              (hand)))
    (division-of-suits
         ($require
              ((and (null (biddable-suit))
                    (division (4 3 3 3))))))
    (honour-points
         ($require
              ((points (from 15 19)))))
)
```

Figure 2.2.1 Frame definition for a prepared bid in bridge.

Each of the $REQUIRE facets in the frame shown are expressed by functions with the data being passed directly as arguments. Each of 'biddable-suit', 'division' and 'points' are separate functions which return true if the relevant information obtained from the ':FRAME' frame matches. The functions access the information being matched implicitly by referencing the global variables :SLOT and :VALUE.

The advantages of this matching system are many. By using functions to express the matching conditions, the full expressive power of Lisp can be employed. The problems outlined above in Section 2.1 are all easily overcome, and it is also possible to have understandable and readable slot values in the frames by the use of suitable function names. The knowledge within the frames is increased greatly since ideas may be considered as functions anyway.

One disadvantage of this system is the need to specify the source of the information to be matched within the conditions themselves through the use of global variables which produce less readable expressions. Another problem is that the frames being matched are not of the same structure, one frame being a generic description and the other being more specific, when in fact it would

be easier to describe the two frames being matched using the same format. Further, the matching functions and the data being matched have to be specified together within the requirements, and therefore are not independent of each other.

This method of matching was used to find the opening bid in the Bridge System. However, for the Bird Recognition System, a different method was devised for two main reasons :

1. To separate the matching process from the data being matched;
2. To describe the frames being matched using the same format.

2.3 Defining an Alternative Matching Scheme.

An alternative matching scheme, similar to that proposed by Rosenberg and Roberts [1979] is described below. Instead of specifying the matching requirements in a generic frame to be matched against an individual frame, the matching functions could be defined separately in a generic frame common to both of the frames being matched. That is, the two frames being matched are instances of the same generic frame and are linked to it through the AKO slot. The matching functions supplied in the generic frame would be separate from the $REQUIRE facet which would be used to specify the generic or global constraints for each slot, such as a list of valid words or a range of values. These constraints could also be used to provide help for a user in an interactive environment.

To perform the matching of the individual frames an FMATCH command would be added to the FRL language and would require only two parameters, the names of the frames to be matched. The primary task of this command would be to match the slot values of each frame by using matching functions that return true, false or unknown depending on the match.

Only those slots that occur in both frames are included in the match. This means that not all of the possible characteristics listed in the generic frame need to be included in the individual frames themselves. Hence, generalised frames with only one or two slots can be matched against other generalised frames or against more specific frames that include most of the characteristics.

The FMATCH of the two frames would return the following possible results :

true if at least one slot match returns true, and none return false

false if any of the slot matches return false

unknown if all the slot matches return unknown.

(In FRL, the logic values 'false' and 'unknown' are equivalent to the atoms NIL and ? respectively. The logic value 'true' is equivalent to the atom T or an s-expression which is not NIL or ?.)

The method of specifying the matching functions for each slot in the generic frame is outlined below.

2.4 Using $MATCH Facets to Specify the Matching Function.

The matching functions for each slot in the generic frame could be expressed by using a $MATCH facet that would specify the name of the match function. The match function would require two arguments, supplied by the FMATCH command during the matching process, and would return true, false or unknown if the two arguments matched or not. A major purpose of using the match functions would be to define the type of information that is to be used by the various individual frames that are linked to the generic frame. By specifying when two values match, the match functions in effect define the range and format of the information being matched, and therefore define its meaning. The generic frame that defines the matching functions for each bird in the Bird Recognition System is shown in Figure 2.4.1.

Using this method means that the frames and the matching functions can be defined naturally, with the information that is to be matched being passed as arguments only. The matching functions can also be defined independently of the data in the frames and provide a means of defining the types of information that is described by each slot. A sample bird described using this generic frame is given in section 3.3.

By allowing LAMBDA definitions as an alternative description of the function defined in the $MATCH facet, the flexibility of the matching system can be improved even further. For example, an alternative method of defining the 'size' slot in the generic bird frame in Figure 2.4.1 is shown in Figure 2.4.2.

```
(deframe bird
    (instance
         ($value
              ("black swan")
              ("paradise duck")
              ("pukeko") ...
         )
    )
    (size
         ($match
              (estimate)))
    (appearance
         ($match
              (appearance)))
    (distribution
         ($match
              (district)))
    (habitat
         ($match
              (same-nouns)))
    (food
         ($match
              (same-nouns)))
    (flight
         ($match
              (noun-description)))
    (behaviour
         ($match
              (noun-description)))
    (breeding
         ($match
              (season)))
    (nest-material
         ($match
              (same-nouns)))
    (nest-location
         ($match
              (location)))
    (number-of-eggs
         ($match
              (number)))
    (egg-colour
         ($match
              (noun-description)))
)
```

Fig. 2.4.1 Generic frame definition for the Bird Recognition System.

The two arguments described in the lambda definition correspond to the two values being matched between the frames. The matching function shown assumes a definite order in the information being matched; that is, the first argument passed to it is either 'large', 'medium' or 'small' and the second argument is a number.

```
(size
   ($match
      (lambda (guess size)
      /* Returns T if the GUESS (large, medium or small)
      /* corresponds to the actual size in cm.
          (cond
               ((equal guess 'small)
                    (lessp size 26)
               )
               ((equal guess 'medium)
                    (and (greaterp size 19) (lessp size 56))
               )
               (t   (greaterp size 50))
          )
      )
   )
)
```

Figure 2.4.2 Lambda definitions used to define the match functions.

The reason for this is that in the Bird Recognition System, the two frames being matched are the general field descriptions of the bird against the specific description of a bird and hence the matching functions can be used to define the meaning of the first frame in terms of the other. A more general match function could allow for any ordering of the information and would in effect be defining the meaning of the information specified by the slot in all the individual frames described by the generic frame.

One problem with using this system is that the matching function to be used is restricted to the one defined in the generic frame. In some cases, it can be difficult to define a function that can cater for all the different slot values since some values are unique or it can be much easier matched by other functions. For example, most birds have only one predominant size, but for the pheasant, the male is considerably larger than the female. Another example occurs in a bird's habitat, where for some birds it is easier to describe the habitats that a bird does NOT live in than to list all those that it does live in.

This problem can be overcome by adding an individual $MATCH facet for those slots where the value requires a special match function. The function defined in the generic frame can then be regarded as the default function which is used when no $MATCH facet occurs within the frame. Hence, the 'size' slot for a pheasant could be defined as shown in Figure 2.4.3.

```
(size
     ($value
          ((male 80 female 60)))
     ($match
          ((lambda (guess sizes)
               (or  (estimate guess (get sizes 'male))
                    (estimate guess (get sizes 'female))))
          )
     )
)
```

Figure 2.4.3 Use of the $MATCH facet to define a special match function for the two different sizes of a pheasant.

In the matching process, the size of the bird will then be matched against the two possible sizes of the bird obtained from the $VALUE facet.

Similarly, the 'habitat' slot for the harrier could be defined as in Figure 2.4.4 below.

```
(habitat
     ($value
          ((forest alpine)))
     ($match
          (different-nouns)
     )
)
```

Figure 2.4.4 Use of the $MATCH facet to allow a simpler description of the habitat of the harrier.

In this case, the matching function will match all those nouns which are NOT contained in the $VALUE list, and means that the bird is found in any habitat except in forest or alpine conditions.

More complicated frame matching schemes can easily be implemented within this structure. Sub-frame matching can be accomplished by specifying FMATCH as the matching function in the $MATCH facet with the names of the frames being passed as the arguments. Further, if the information in one of the frames is contained in a list whereas in the other frame it is contained within a sub-frame, then the required matching function can be defined using the list as the first argument and the name of the sub-frame as the second argument. For example, this occurs in the bird recognition frames defined in Figure 2.4.1 where the bird watcher's description of the bird's appearance needs to matched against the precise description of the bird listed in a subframe. Even more complicated matching of frames, where it is necessary to match different combinations of the slots using or and and conditions, can best be developed in a manner described in section 2.6.

2.5 Improving the Matching Structure using $IF-MATCHED demons.

An important feature of using $MATCH facets to specify the matching function is that the design of the matching process may be done independently of how the information in the knowledge base is structured. The problem of matching two separate frames of information simply becomes one of building up a library of functions that require only two parameters of information and perform the matching of the various types of information that may occur. These functions can easily be improved at any stage.

However, to achieve this goal, any desired effects of the match (such as displaying a message in a trace of the match) has to be separated from the matching process itself. This can be done by using $IF-MATCHED demons which are activated whenever the slot values are matched. An $IF-MATCHED facet would be added to the relevant slots, and would specify the function to be used when that particular slot is matched. The two slot values could be passed as arguments directly to the function in a similar manner to the $MATCH functions. The function could also access the information relevant to the match by using global variables that are bound during the matching process. The variables :FRAME1 and :FRAME2 could be set to the names of the frames being matched, :SLOT to the name of the slot and :MATCH to the result of the match. These variables could also be used within a $MATCH function if information other than the slot values were required in the match.

For example, in the Bird Recognition System, Figure 2.5.1 shows how an $IF-MATCHED demon may be used to indicate to the user of the system whether the distribution of the bird matches the description.

The above function is used to print a message whenever the distribution slot of a bird is matched. The global variable :MATCH is used to indicate the result of the match; :FRAME2 is the name of the bird being matched against.

The usefulness of the $IF-MATCHED demons is not limited to producing a trace of the matching process. A more complicated matching scheme may involve updating an associative network by adding a link to other frames, or adding or removing the information gained from the match to a short term memory that controls the

```
(defun show-district (district1 district2)
/* Prints a message if the DISTRIBUTION slot is matched.
    (cond
        ((true :match)
            (prin1 '"The ")
            (print-name :frame2)
            (prin1 '" is usually found in ")
            (print-names district2)
            (print '" which matches with where the bird was found.")
        )
        ((null :match)
            (prin1 '"The bird cannot be the ")
            (print-name :frame2)
            (prin1 '" which is usually found in ")
            (print-names district2)
            (print '".")
        )
    )
)
```

Figure 2.5.1 $IF-MATCHED demon for the distribution slot of a bird.

search. Another application could be to direct the search to another part of the frame system if the search fails or succeeds. Like the $IF-REMOVED, $IF-ADDED and $IF-NEEDED demons, the power of the language is improved by attaching procedures to the data stored in the frames.

2.6 Using the MATCH Slot to Specify the Frame Match.

A limitation of the matching scheme described above is that all the slots in the frame must match before FMATCH is successful. In most applications, this is sufficient because the nature of frames is to specify a stereotyped situation using a list of slots that describe that situation. However, in many cases a situation can be described in different ways which are all equally valid if they occur. For example, there are two situations for which a 2 clubs bid is an appropriate opening bid - when the honour points of the hand are above 22 or when the quick tricks are above 4. Within the

current frame format, it would not be possible to describe this situation by a unique frame. Therefore, it is necessary to devise some method of specifying alternatives in the match and also of specifying how the information in the frame is to be matched.

This information could be specified in a MATCH slot added to the description of the frame. The $VALUE facet of this slot would be used to describe the logical combinations of other frames that must match before the current frame matches. This could be done by specifying or, and and not conditions of the names of the other frames within the slot value. For example, the 2 clubs bid situation described above could be defined as shown in Figure 2.6.1.

In the frame shown, the 2 Clubs frame will match if either of the two frames 'high-points-hand' or 'high-tricks-hand' match. Note that in this example there are no other identifying slots in the frame which would also have to be matched if they were there. This method of specifying the match is particularly useful in describing conditions that should NOT occur if the frame is to match. Consequently, certain frames can automatically exclude the matching of other frames by using cross references to each other.

The first task of the FMATCH command must now be to check whether the frame match conditions specified in the MATCH slot are satisfied, and then to match the remaining slots in the frame if they are. So that common frames are not repetitively matched when the result is already known, a global list of results of the frame matches, called :MATCH-RESULTS, needs to be kept. This list could be a property list of frame names which contains a further list of frame names and the results of the matches between the respective

```
(deframe 2-clubs-hand
    (match
         ($value
              ((or high-points-hand high-tricks-hand))))
    (bid
         ($if-needed
              ( '(2 clubs))))
)
```

Figure 2.6.1 Definition of 2 Clubs hand using the MATCH slot.

frames. This list can be automatically updated the first time two particular frames are matched and accessed whenever they are matched at any subsequent time.

A major advantage of this system is that it is easier to define frames in terms of other frames which are more generalised or occur more regularly. Also, it is easier to describe complicated situations by using cross references to other frames. A substantial saving in the matching of the frames may also be possible because the frames are in effect matched only once.

Once the frame has been matched, it is often necessary to perform some task as a result of the match. This can be achieved by attaching an $IF-MATCHED facet to the MATCH slot. For example, in the 2 Clubs frame described above, the $IF-MATCHED facet could be used to indicate the bid for the frame instead of using a separate BID slot. Alternatively, the $IF-MATCHED facet could be used to display a message which indicates the result of the match.

The LISP definition of the FMATCH command defining the frame matching scheme described above is listed in Appendix A.

3 STRUCTURING FRAMES FOR SEARCH.

3.1 Different Frame Structures.

An important feature of any knowledge base is the ability to locate relevant information quickly and efficiently. In a knowledge base consisting of frames, the searching of individual frames that match a certain criterion can be improved by using different structuring techniques. A summary of various structures is given below.

Linear.

Each case or situation in a problem is represented by an individual frame in the knowledge base. The frame contains the features or slots that describe that situation. The linear structuring involves each frame being tried separately for a successful match.

Set.

The linear system can be improved by grouping common features in general set frames. By describing those frames that match or do not match each set frame, a smaller subset of the knowledge base can be obtained to improve the search.

Hierarchical.

Another method is to structure the knowledge base as a hierarchy or tree. Key features of each situation are placed at higher levels of the tree and these are tested first in order to

narrow down the search.

Network.

The network structure consists of linking frames by using references to other frames. In a search through the network, if a fit fails then the current frame indicates which one to try next. Alternatively, other frames in the network may have to be searched before a frame can be matched.

Sample frames devised for each type of structure are listed in Appendix B. A search trace for each structure devised for the Bird Recognition System is also listed in Appendix D.

3.2 Searching Strategies.

The structure of the knowledge base determines the type of searching strategy that is employed. Depending on the structure used, the individual frames in the knowledge base would contain further slots that indicate how to proceed with the search. These are described in greater detail for each structure in the following sections.

However, a further consideration independent of the various frame structures is the problem of multiple frame matches where more than one frame may be suitable for a given match. The method used to distinguish between these multiple matches is an important consideration in evaluating the efficiency of the frame structure.

A 'first fit' method involves searching through the frame structure for the first successful match. Any further searching is immediately stopped and the first frame is returned. The problem with this method is that the frame returned may not be the best possible match. The advantage is that further searching of the knowledge base is not required. In some applications, any match may be sufficient and hence the first fit strategy can be used.

However, it may be necessary to find all of the matches within a knowledge base. This involves a further problem of choosing which is the best frame amongst those that matched. In bridge bidding, for example, there may be various alternative opening bids for a hand that are feasible, but only one of these may be suitable. It is possible to allow for a more precise description in the frame to be matched so that the first fit method can be used, but this involves a trade off between generality of information and searching efficiency. Another possibility is to rank the alternatives, either by assigning a ranking to each frame or by having the matching process return information about how 'good' the match was (for example, the number of slot matches that returned true).

The problems involved with these different strategies are described in length along with the description of each frame structure following. The problems associated with ranking frame matches is also discussed as an area for further research in Chapter 6.

3.3 Linear Structure.

The linear system is the simplest of the searching strategies. It involves describing each situation in the knowledge base by an individual frame with each frame being tried in succession until one matches. The order that each frame is matched is fixed. Diagram 3.3.1 below illustrates the structure of a linear frame system.

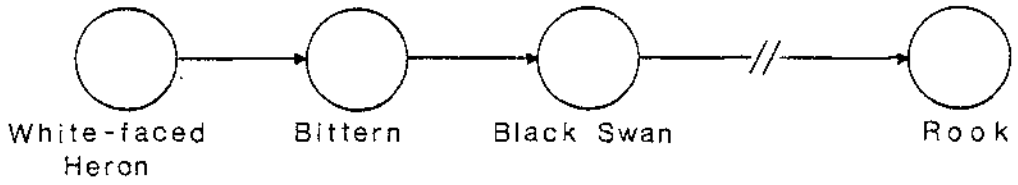

Diagram 3.3.1 Linear structure for the Bird Recognition System.

Each frame contains a list of slots that describe the situation that the frame represents. A successful match occurs only if the frame FMATCHes with the opposing frame.

An example of an individual frame using a linear structure in the Bird Recognition System described in section 2.4 is shown in Figure 3.3.1 below. Each slot in the frame shown describes a different feature of the bird. No links to any other frames are required. The AKO slot is used for inheriting the generic information related to the frame.

The search through the knowledge base can be achieved by defining the order that the frames are matched in the generic frame. In the Bird Recognition System, the linear ordering is described by the list in the INSTANCE slot of the generic BIRD frame. Each frame in this list is matched one after the other. If a first fit

```
(deframe "pukeko"
    (ako
         ($value
              ("bird")))
    (size
         ($value
              (51)))
    (appearance
         ($value
              ("pukeko appearance")))
    (distribution
         ($value
              ("new zealand")))
    (habitat
         ($value
              (marsh) (lagoon) (lake) (river)))
    (behaviour
         ($value
              ((walk (jaunty) run (fast) swim (comfortable)))))
    (flight
         ($value
              ((take-off (laboured) feet (dangling)
                flight (strong)))))
    (food
         ($value
              (plants) (snails) (insects) (eggs)))
    (breeding
         ($value
              ((august to november))))
    ("nest material"
         ($value
              (sticks) (rushes) (grass)))
    ("nest location"
         ($value
              ((in (vegetation marsh)))))
    ("number of eggs"
         ($value
              ((4 to 8))))
    ("egg colour"
         ($value
              ((mainly (red cream) spots (red brown)
                blotches (purple)))))
)
```

Figure 3.3.1 Example of the frame description of a bird in the linear system.

searching strategy is required, then the searching would stop when the first frame in the list matches. Consequently, for a more efficient search, the most likely frames should be placed at the beginning of the list. Alternatively, it might be more appropriate to rank the frames with the more important frames being at the

beginning of the list. This was done for the Bridge System so that the more relevant bids such as 2 clubs and 2 no trumps would be matched first. For a match all searching strategy, all the frames described in the list are tested and the names of those that matched are returned.

The advantages of this system are that the frame structure is simple and easy to set up. Each frame describes an individual situation with no relation to other frames. However, if there are many frames in the knowledge base, or if there are many generalised slot values within each frame that will match a large number of other frames, then this system can be inefficient for searching purposes. For a first fit strategy, the average number of tries is 1/(1+n) where n is the number of fits in the system and this may not be noticeable if the number of frames in the system is small. However, for a match all strategy the search requires that all the frames are matched.

3.4 Set Structure.

The set frame structure uses the properties of sets such as union and intersection to narrow down the search space required in the linear system. By identifying key slots in the knowledge base, sets of the most likely and unlikely frames may be constructed for the search.

The information relating to these sets can be grouped in general 'set' frames which are separate from the list of frames used for the linear system. These set frames contain three types of slots :

1. INSTANCE slot.

   Contains the set of frame names that positively match the frame.

2. REJECT slot.

   Contains the set of frame names that can definitely be rejected from the search.

3. Identifying slots.

   Describe the features of the frame.

For example, some set frames defined for the Bridge System are shown in Figure 3.4.1. The frames shown describe those sets of hands that have an average or low honour point count. Diagrams 3.4.1 and 3.4.2 illustrate the features of the set structure.

By matching each of the slots in the frames shown in Figure 3.4.1 a large number of frames can be automatically rejected from the search. Set frames may also be rejected from the search by including the names of other set frames in the REJECT slot.

The searching procedure requires that all the set frames are matched with two global sets constructed that contain the set of frames to search and the set of rejected frames. Only if a set frame matches are the frames in the INSTANCE and REJECT slots added to the respective global sets. After the last set frame has been processed, the most likely frames can be returned by removing any of

```
(deframe average-points-hands
    (instance
         ($value
              (1-of-suit-frame)))
    (reject
         ($value
              (light-opening-frame) (two-no-trumps-frame)
              (two-clubs-1-frame)))
    (honour-points
         ($require
              ((points (from 12 19)))))
)

(deframe low-points-hands
    (instance
         ($value
              (light-opening-frame) (pre-emptive-frame)))
    (reject
         ($value
              (1-of-suit-frame) (4-4-frame) (4-4-4-1-frame)
              (one-no-trumps-weak-frame) (prepared-frame)
              (two-clubs-frame) (two-no-trumps-frame)
              (strong-two-frame)))
    )
    (honour-points
         ($require
              ((points (below 12)))))
)
```

Figure 3.4.1. Two set frames used in the Bridge System.

the rejected frames from the set of frames to search. These frames can then be searched on the same basis as the linear system.

For this system to be reliable, it is necessary that the final set that is returned contains the names of all the frames in the knowledge base that could possibly match. However, this does not mean that all the frames that match a certain set frame need to be placed in the INSTANCE slot. If this were so, then the effectiveness of the system would be reduced since too many frames would be added to the final set. For example, most bidding hands match the 'average-points-hands' set frame above, but it is necessary that only the '1-of-suit-frame' be included in the INSTANCE slot because each of the possible bids are indicated by other set frames.

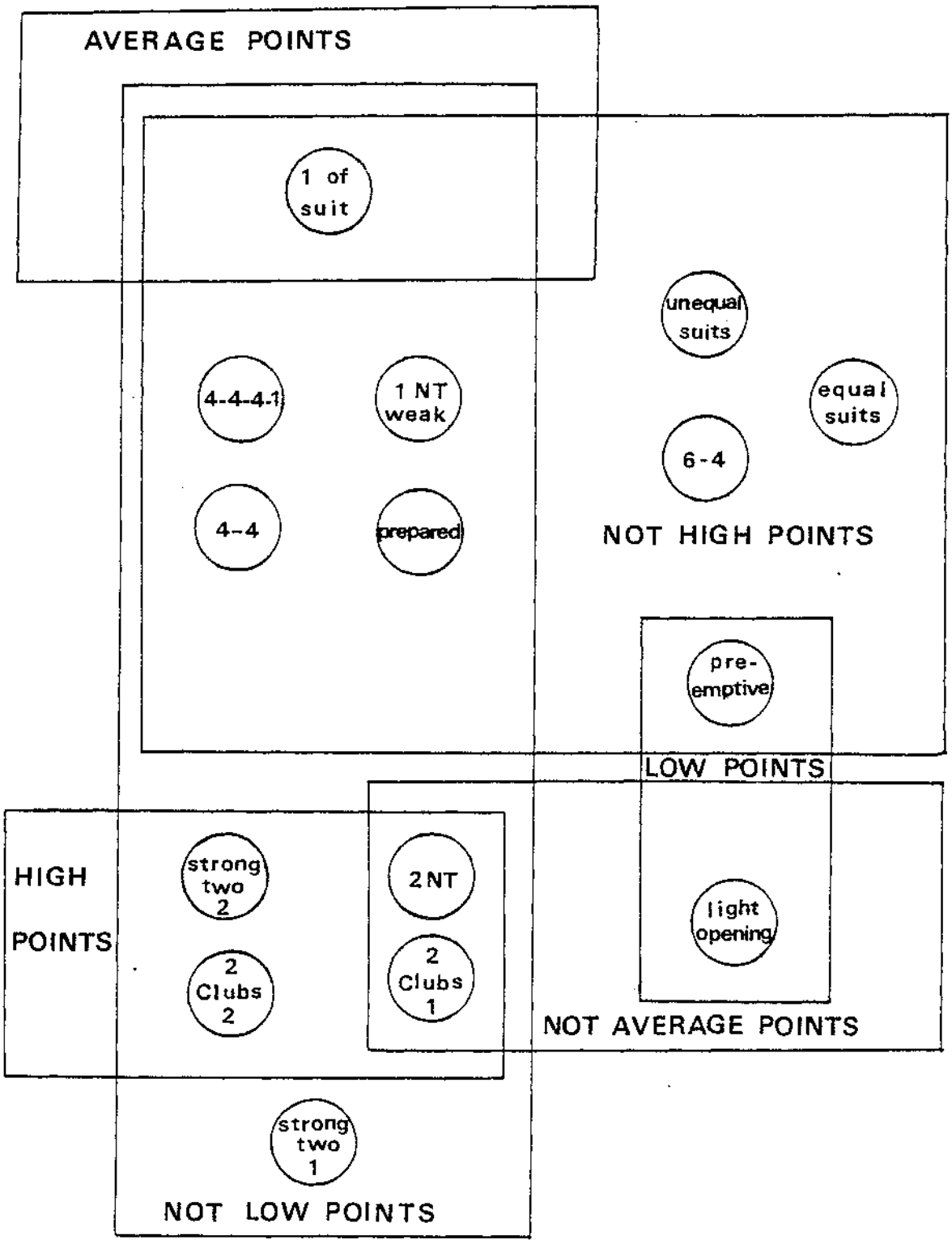

Diagram 3.4.1 Venn diagram of part of the Bridge set system.

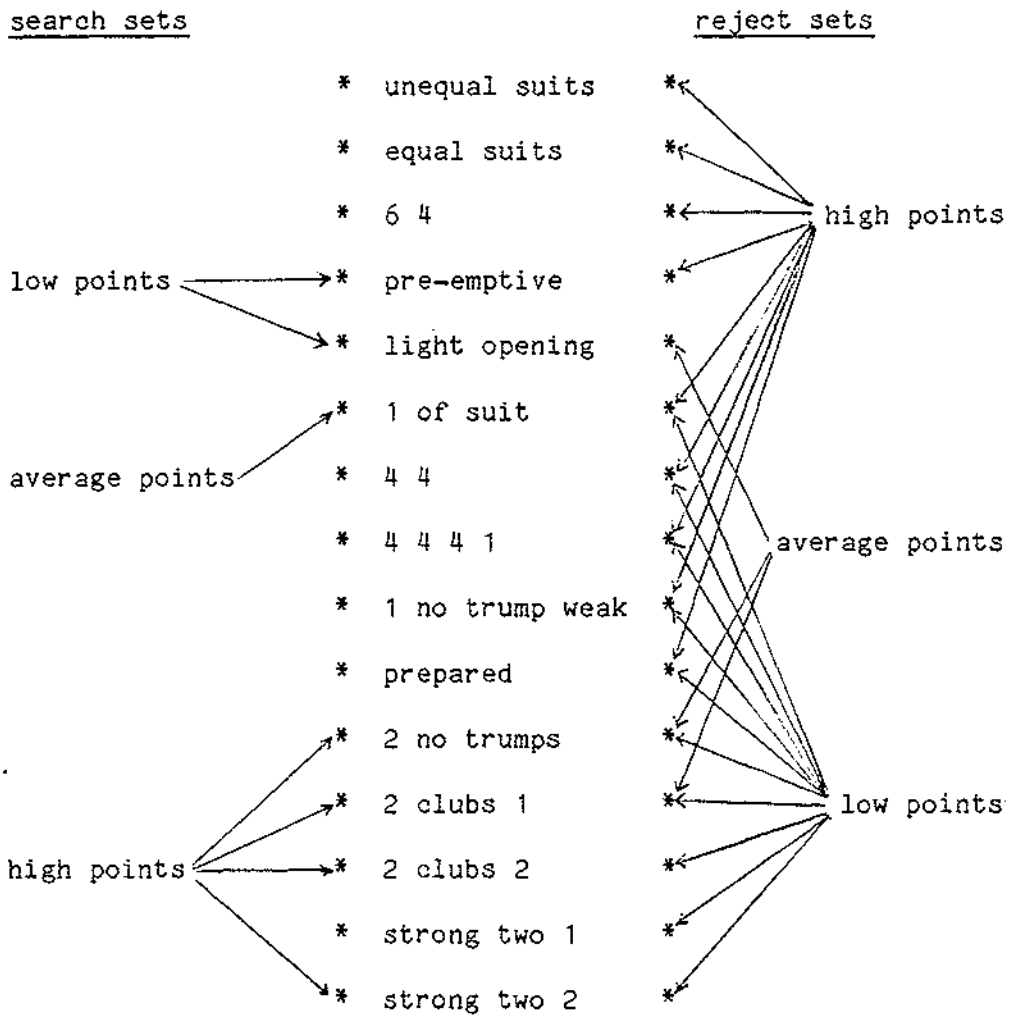

Diagram 3.4.2 Alternative representation of Diagram 3.4.1.

One disadvantage of this system is that it can often be difficult to recognize key slots in the knowledge base which can produce the most efficient rejection of the search space. This requires a careful balance between the time it takes to process all the set frames in the system and the size of the set that is finally returned. The extra overhead involved in matching the set frames can mean that this system is inefficient if the number of slots in the knowledge base is small and a first fit searching strategy is required. However, this system can offer a big improvement in search time if a match all strategy is required because a large

number of frames have automatically been rejected from the search. The advantages of the linear system also apply here since the set system uses the same frame structure for specifying individual frames.

3.5 Hierarchical Structure.

The hierarchical system involves setting up the knowledge base as a tree structure with key slots being grouped at higher levels of the tree. If these slots do not match at the higher level, then the whole sub-tree may be rejected from the search.

A node in the hierarchical system has 3 features :

1. A list of subframes using the INSTANCE slot. This list would contain the names of the 2 sub-nodes if a binary tree structure was required.

2. The name of the parent frame using the AKO slot. This means that the knowledge at higher levels of the tree can be inherited and need not be represented within the node itself.

3. The identifying characteristics of the frame such as honour points and suit distribution in bridge.

An example of a sub-tree of the bridge tree system is given in Figure 3.5.1 below. The frames shown represent the '4-4-suits' sub-tree which has two sub-nodes, the '4-4-4-1-tree' and the '4-4-tree'. The overall structure of the bridge hierarchy is

illustrated in Diagram 3.5.1.

```
(deframe 4-4-suits-tree
    (ako
         ($value
              (equal-suits-tree)))
    (instance
         ($value
              (4-4-4-1-tree)
              (4-4-tree)
         )
    )
    (division-of-suits
         ($require
              ((biddible (suits (4 4))))))
    (honour-points
         ($require
              ((points (above 12)))))
)


(deframe 4-4-4-1-tree
    (ako
         ($value
              (4-4-suits-tree)))
    (division-of-suits
         ($require
              ((division (4 4 4 1)))))
    (bid
         ($if-needed
              (4-4-4-1-bid)))
)


(deframe 4-4-tree
    (ako
         ($value
              (4-4-suits-tree)))
    (bid
         ($if-needed
              (equal-suits-bid)))
)
```

Figure 3.5.1 A sub-tree in the bridge bidding tree.

The sub-tree shown describes all those hands which have at least two four card suits. Knowledge at higher levels such as 'equal-suits' is assumed at lower levels of the tree through the AKO link. Notice that the '4-4-tree' has no identifying characteristics because of this.

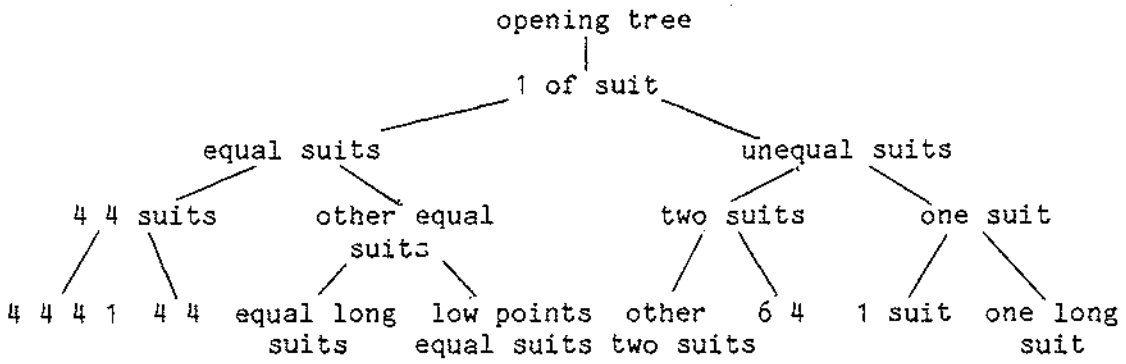

Diagram 3.5.1 Hierachical structure for 1 of suit bids in the Bridge System.

The searching through the tree is conducted on the same basis as a binary tree search. If at any stage of the search a node does not match, then the entire sub-tree may be rejected. The search continues down the tree until a terminal node matches; for the first fit strategy, the search immediately stops. If all the matches in the tree are required, then the rest of the tree is searched until all the nodes at any one level of the tree have been either matched or rejected. If the tree is reasonably balanced, then the search provides a very efficient search for both first fit and match all stategies since large parts of the search can be rejected at higher levels of the tree.

The major problem with this system is that the frame structure can often be difficult to set up. To obtain a balanced tree, it is necessary to organise the tree efficiently with the search being split in half at each node. Alternatively, an unbalanced tree can be constructed where the frequencies of the different properties are taken into consideration, with the most frequent properties being placed at higher levels of the tree. It may be possible to do this automatically, as in the Bridge System, for example, where the frequencies of each type of hand are already known.

It also can be difficult to structure the frames that make up a tree because it requires common features to be placed at higher levels of the tree. Often there is no connection between frames or it may become necessary to match odd frames that fit a certain requirement but are special cases at a lower level. This means that more branches in the tree have to be created, with a greater number of slots to be matched and consequent slower search time.

Another disadvantage is that the tree structure is not very flexible. Any additions or alterations may be difficult to implement within an existing tree and may require major changes or lead to further inefficiences in the search.

3.5.1 Search Tree

To overcome these difficulties, the hierarchical structure can be used solely for describing the search through a frame system with the actual description of each specific frame being separate. This means that, like the set system, the description of the search is independent of the individual frame descriptions, and therefore may be developed separately. The INSTANCE slot in the terminal nodes of the separate 'search tree' now contains the names of the relevant individual frames. Information about the individual frames may still be inherited through the AKO slots in the search tree. For example, part of the search tree designed for the Bird Recognition System is shown in Figure 3.5.1.1 below. The structure of the Bird search tree is also illustrated in Diagram 3.5.1.1.

```
(deframe "small white bird"
    (ako
         ($value
              ("other small bird")))
    (instance
         ($value
              ("red faced bird")
              ("not red faced bird")
         )
    )
    (appearance
         ($value
              ((mainly (white) body (white) underparts (white)
                upperparts (white)))))
)

(deframe "red faced bird"
    (ako
         ($value
              ("small white bird")))
    (instance
         ($value
              ("welcome swallow")
              ("goldfinch")
         )
    )
    (appearance
         ($value
              ((face (red) throat (red)))))
)
```

Figure 3.5.1.1 Two frames in the bird search tree.

In the two frames shown, "small white bird" and "red faced bird" are both nodes in the search tree, whereas "welcome swallow" and "goldfinch" are separate frames that describe individual birds. Note that it is often necessary to include the name of an individual frame in several terminal nodes of the search tree if that frame satisfies more than one pathway of the tree. This is to ensure that all the possible individual frames are searched.

This search method allows for a greater flexibility in designing the frame system since the search and the individual frame descriptions may be designed separately. Also, the

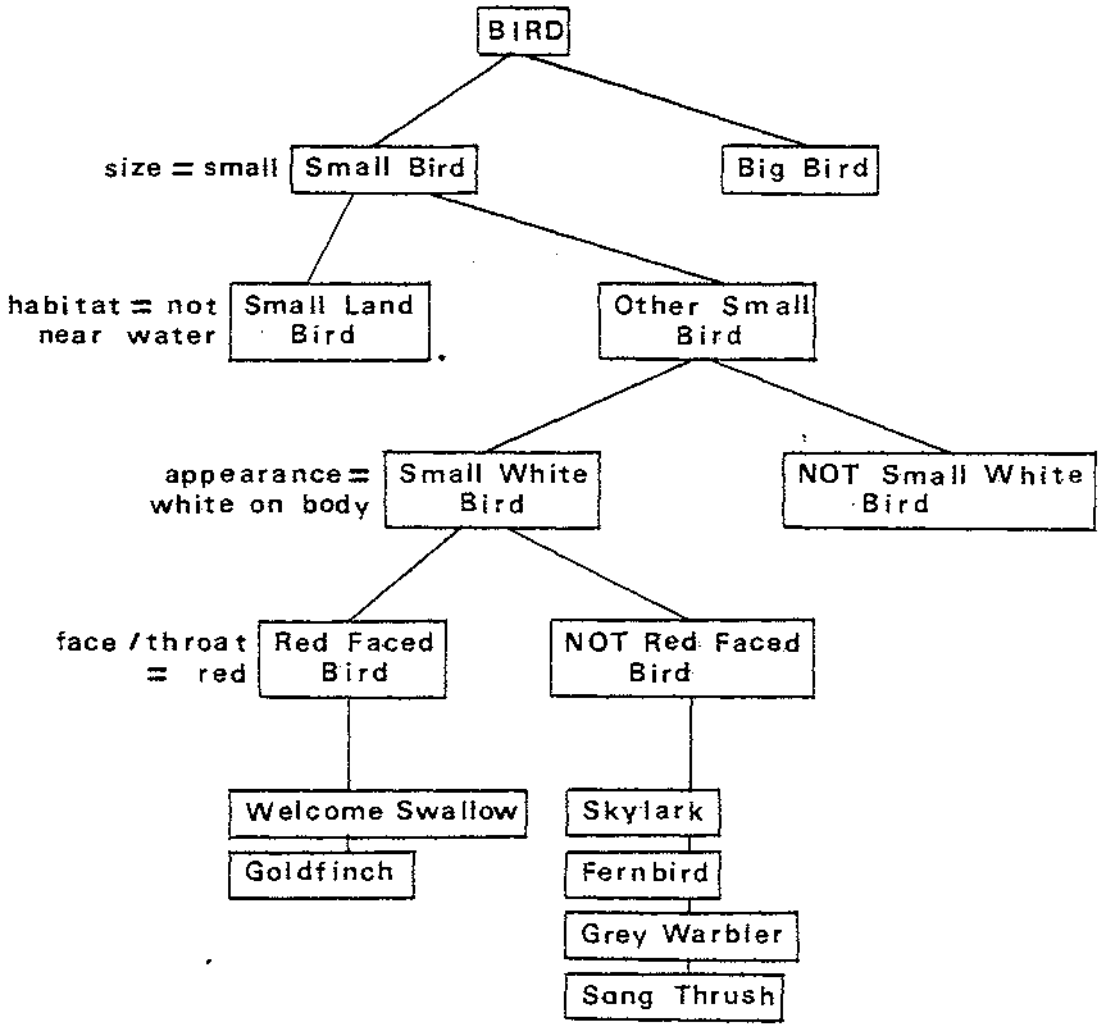

Diagram 3.5.1.1 Structure for the Bird search tree.

efficiency of the search may be improved since fewer nodes of the tree have to be defined. However, once the tree has been searched, there is the further need to match the relevant individual frames. To off-set this, the size of the search tree can be designed to take into account the trade-off between having a small search tree and having fewer individual frames to search.

A major limitation of the hierarchical approach (for both structures outlined above) is the effect of unknowns on the efficiency of the search. An ideal hierarchical structure will partition the search into two or more equal and disjoint sets at each level of the tree. Less than ideal situations include lop-sided partitions and situations where a particular slot is not present in some frames or the match function returns unknown.

There are two consequences of unknowns that affect the efficiency of the search :

1. Terminal nodes of the tree for which the partition at a certain level is unknown have to be included in all the sub-levels of the tree.
2. If the matching of a frame returns unknown at a certain level, then all the sub-levels of the tree have to be searched.

For a system such as the Bird Recognition System where much of the information being matched is unknown, careful consideration has to be taken in designing the descriptions at each level of the tree. By using common features in the descriptions such as appearance and size, the problems of multiple occurrences and the need for extra search can be lessened.

3.6 Network Structures.

A fourth method of structuring frames is to set up the knowledge base as a network. Each frame is linked to other frames by association paths with the information indicating which frame to try next being stored within the frame itself. Hence the pathway through the network a search takes is not fixed like the linear system but is dependent on the data within the frame being matched. And unlike the hierarchical system, the structure of the knowledge base is not constrained within a fixed format, allowing new frames to be added without any alteration to the existing structure.

There are various problems in designing network frame structures. One problem is to decide where to start the search since each node in the network is equivalent. Another problem is ensuring that the search of the network is systematic so that all the relevant nodes in the network are tried. A third problem is the problem of cycling, where the search keeps following the same pathways through the network. These problems are discussed below in relation to the various network structures devised.

The advantages of the network system are its flexibility and relative ease to set up. The network system can be expanded or altered quickly simply by adding a new node in the network or by changing an existing one. The system is also relatively straightforward to set up since the description of each frame is essentially independent. Another important feature is that the associative links in a network seem to correspond to human thought processes unlike other systems such as tree structures.

3.6.1 Slot network.

Since the only extra information needed to set up the network is the pathway the search takes through it, the frames can be set up in a similar fashion to the linear system. The links to further frames in the network can then be added as a list of names with the most likely frame being the first in the list.

There are two different methods of associating this list with the information contained in the frame. One method is to add to each frame a further SEARCH slot that contains the list of further frames to try. Therefore, the list is linked with the frame overall, with the searching through the network being done on a frame basis rather than at the slot level. However, this means that the whole frame has to be matched and does not take into account the information learnt from the slots which matched. It also means that the search could be just as inefficient as the linear system since the path in most cases would be fixed, except for very large knowledge bases.

The second method is to associate the set of names with the slots themselves. This can be done by having two facets for each slot :

1. $value the value of the slot
2. $search the set of frame names associated with the slot which indicates the most likely frames to search if a match of the slot value failed. The ordering of the set would indicate the order in which each frame is tried.

The structuring of such a slot network is illustrated in Diagram 3.6.1.1.

This system provides a method of allowing the search through the network to be associated with what has been previously learnt in the search. The ordering of the slots within the frame now becomes important since second and successive slots can use the information from the first slot being matched. By also specifying only those frames which are closely associated with the current frame in the $search set, the search can be guided throughout the network so that similar groups of frames are matched together.

Two examples of nodes in a bridge slot network are given in Figure 3.6.1.1 below.

The slot values of the two frames described are the same as used for the linear system. Associated with each slot, however, is the set of nodes in the network that are linked to the current frame. For example, the set of nodes linked to the 'honour-points' slot in the 'prepared-node' frame consists of two nodes - the 'one-no-trumps-weak-node' and the 'two-no-trumps-node'. This set would only be required in the search if the first slot in the frame matched while the second slot failed since the search is immediately directed to a new node if any match fails. Hence, the nodes in the specified set can use the information gained from the first match to direct the search.

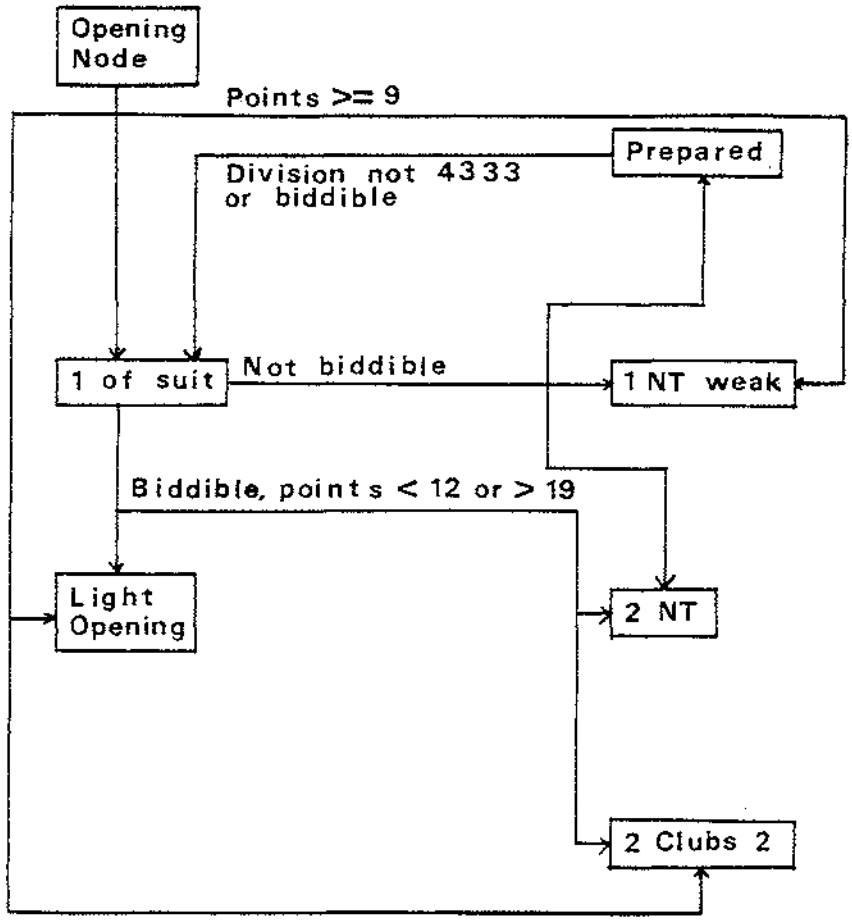

Diagram 3.6.1.1 Structure for the Bridge slot network.

```
(deframe prepared-node
    (division-of-suits
         ($require
              ((and (division (4 3 3 3))
                    (null (biddible-suit)))))
         ($search
              (1-of-suit-node))
    )
    (honour-points
         ($require
              ((points (from 15 19))))
         ($search
              (one-no-trumps-weak-node) (two-no-trumps-node))
    )
)

(deframe 1-of-suit-node
    (division-of-suits
         ($require
              ((biddible-suit)))
         ($search
              (prepared-node) (one-no-trumps-weak-node)
              (two-no-trumps-node))
    )
    (honour-points
         ($require
              ((points (from 12 19))))
         ($search
              (two-no-trumps-node) (light-opening-node)
              (two-clubs-2-node))
    )
)
```

Figure 3.6.1.1 Sample nodes in the bridge bidding network.

Notice that the two frames described above are cross-linked to each other forming a small loop in the network. The problem of cycling can easily occur in this type of network and it is necessary that some method is devised whereby the searching of previously tried nodes is eliminated. The simplest method is for the searching function to keep a set of names of those nodes that have already been tried so that if the first node in the $search set has already been tried then the second and subsequent nodes in the set will be searched. If all the nodes in the $search set have been tried then the search of the current path fails.

To ensure that the search is systematic, then it must be possible to reach every other node in the network from the starting node. This is easy to implement in the system used here by linking the starting nodes to other common nodes from where most parts of the network may be reached.

A further problem of efficiency occurs in deciding where to start the search to obtain the most efficient search through the network, since the pathways from some nodes may be considerably longer than others. However, the network can easily be structured around a common central node which efficiently splits the search by links to progressively less common nodes. The use of the $search sets provides an efficient search anyway, since the search is directed by what has been learnt in previous frames. The starting node in the bridge network, shown below, is used to reject a large percentage of the bids by noting that half of the possible hands in bridge do not have an opening bid. The set described in the $search set is also ordered with the most common node given first, and contains all those nodes that cannot be searched from any other node.

The overall searching strategy through the network is recursive with the immediate decision about which path to take being determined from the current slot being matched and its corresponding $search set. If the current path being investigated comes to a dead end, then various control mechanisms can be employed such as depth first, breadth first or even 'best first' for continuing the search through the network.

```
(deframe opening-node
    (honour-points
         ($require
              ((points (below 9))))
         ($search
              (1-of-suit-node) (equal-suits-node)
              (strong-two-2-node)  (strong-two-1-node)
              (one-no-trumps-weak-node) (two-clubs-two-node)
              (pre-emptive-node))
    )
)
```

Figure 3.6.1.2 Starting node for the search through the bridge network.

As the search is guided by the slots that match, a first fit strategy can be very efficient. For the match all strategy, the rejection of nodes from the search is carried out through the slot values that match, since there is no need to follow any $SEARCH links. Therefore, the effectiveness of the search is determined by the slot values that match.

3.6.2 AKO network.

An alternative network structure is an AKO network. The network consists of individual frames similar to the linear and slot network structures that express an idea or pattern. However, each frame does not have to be linked to the same generic frame and may have different slot names depending on the idea that the frame expresses. Generalised frames that describe common features are included in the network, and this information is inherited by other frames through the AKO slot. For a frame to match, the whole frame including all of the frames listed in the AKO slot have to match, and therefore the search through the network is conducted through the AKO slots. Diagram 3.6.2.1 illustrates the structure of the AKO network.

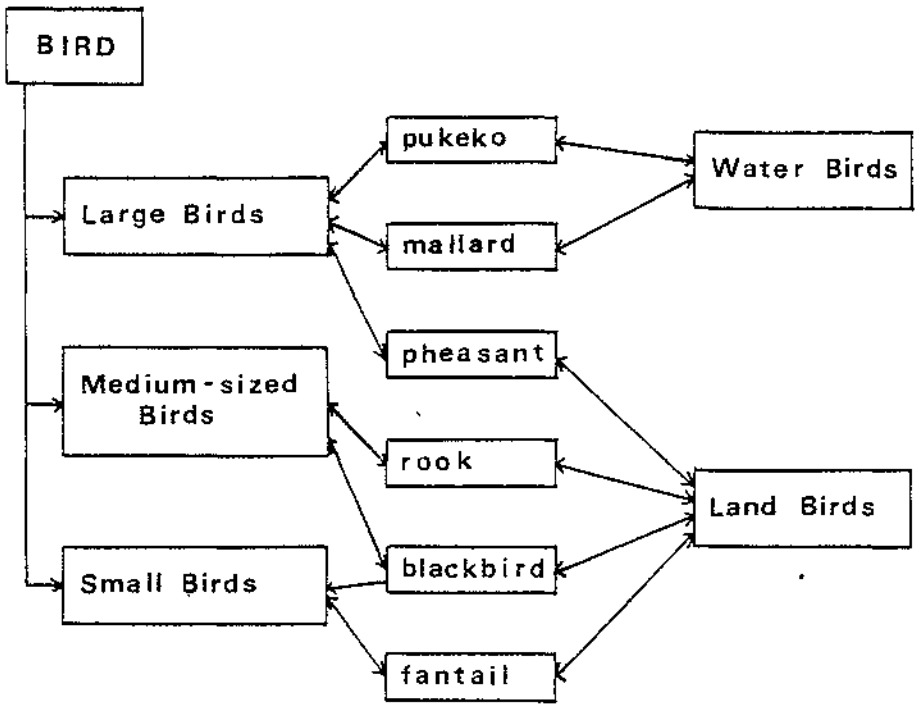

Diagram 3.6.2.1 Structure for the Bird AKO network.

The format of a frame in the AKO network requires 4 types of slots:

1. AKO slot.

This contains the names of the frames that must match if the matching of the current frame is to be successful. Information about the frame is also inherited from the AKO frames.

2. INSTANCE slot.

This contains the names of all the frames that satisfy the conditions described by the current frame. This slot also provides a list of further frames to try when searching through the network. The AKO and INSTANCE slots together provide forward and reverse links through inheritance.

3. REJECT slot.

The REJECT slot contains the names of all the frames that can be automatically rejected from the search if the current frame matches. This provides for a similar format to the set system meaning that the advantages of the set search may apply and also that a set frame can easily be linked into an AKO network.

4. Identifying slots.

These slots describe the features of the frame such as habitat and appearance. For example, Figure 3.6.3.1 below shows how the Spur Winged Plover is defined in the AKO network for the Bird Recognition System.

```
(deframe "spur winged plover"
    (ako
         ($value
              ("medium sized bird")
              ("water bird")
              ("colourful billed bird")
              ("light feet bird")
              ("light undersides bird")
              ("slow flight bird")
              ("South Island bird")
              ("animal eating bird")
              ("few eggs bird")
              ("coloured eggs bird")
              ("bird")))
    (size
         ($value
              (38)))
    (appearance
         ($value
              ("spur winged plover appearance")))
    (distribution
         ($value
              ("south island")))
    (habitat
         ($value
              (pasture) (crops) (marsh) (coast)))
    (behaviour
         ($value
              ((walk (sedate) run (nimble)))))
    (flight
         ($value
              ((wings (white dark) take-off (slow) rump (white)
                wingbeat (slow) tail (white black)))))
    (food
         ($value
              (worms) (insects)))
    (breeding
         ($value
              ((july to december))))
    ("nest material"
         ($value
              (hollow)))
    ("nest location"
         ($value
              ((in (ground hollow)))))
    ("number of eggs"
         ($value
              ((3 to 4))))
    ("egg colour"
         ($value
              ((mainly (muddy green) blotches (purple brown)))))
)
```

Figure 3.6.3.1 Frame definition for the Spur Winged Plover in the AKO network.

The frames listed in the AKO slot of the frame shown describe the kind of general features that the bird has. Each of these general features are defined by different frames of the network, and each may have further AKO, INSTANCE and REJECT slots that are used when searching the network.

The search of the network starts at a point in the network determined by its goal. The starting node is also a node in the network, since every node is included in the network structure, and would itself be a part of a more general search.

For example, if the Bird Recognition System was part of a larger network of animals, then the goal "Which bird?" would suggest searching the BIRD node which would then give the general requirements of what a bird was like. This frame would then suggest further frames to try for a more specific match. Since the Bird Recognition System is not part of a larger network, the identifying characteristics such as "has wings" and "lays eggs" need not be included in the BIRD node.

The search through the network requires that two list of frames be kept:

1. search-list - the list of nodes in the network suggested by the INSTANCE slot that have yet to be rejected from the search;

2. reject-list - the list of nodes in the network that have been rejected from the search either by the REJECT slot or a match returning false.

Nodes can be added and deleted from each list depending on the information found in the search. Alternatively, if the information in the REJECT slots can be assumed to be correct, then once a node is added to the reject list it can stay rejected therefore preventing any cycling that may occur.

The search algorithm consists of first setting the search list to a list of frames in the INSTANCE slot of the starting node, and then trying to match the first and successive frames in the list. There are three requirements for a frame to match :

1. The frame has not been rejected from the search.
2. All the AKO frames match.
3. The identifying slots of the frame match.

Since all the AKO frames must match before the current frame matches, the searching of the network is conducted primarily through the AKO slots. The search is also quickly narrowed down to a specific area by following the AKO slots. For a first fit search strategy, the search is complete when any of the frames in the INSTANCE slot of the starting node has been matched successfully. To find all the fits in the network, the search is completed when the search list is empty. The problem of systematic search can therefore be eliminated by ensuring that all the possible frames that may match are included in the INSTANCE slot of the starting node.

The REJECT slot provides the basic mechanism for rejection of most of the nodes in the search. Therefore, addition of general nodes at any point in the network facilitates a quicker search because of the rejection of the nodes. This means that general frames that describe certain features or groups of the network, such as a land bird or an omnivorous bird in the Bird Recognition System, may be included in the network to improve the search.

There are two different methods of searching the network. The first method described above is to match the current node and then add the nodes indicated by the INSTANCE slot to the global search list. The second method involves searching the nodes described by the INSTANCE slot immediately once the current node has been matched. However, this second method means that the search through the network is conducted in a haphazard manner with the search changing direction whenever a new node is matched. The algorithm for the first method for searching the network was implemented in the Bird Recognition System and is listed in the appendices.

There are two ways of improving a search through an AKO network. The first method involves ordering the frames listed in the AKO slot so that the most distinct or likely AKO frames are tried first when matching a particular frame. This is similar to human thought processes where the most distinct or interesting information is brought to mind first. The ordering of the AKO slot was done for the Bird Recognition System to ensure that the most distinct features such as size and appearance would be matched first. The second method involves ordering the frames

listed in the INSTANCE slot to ensure that the nodes with the most distinctive AKO information are tried first.

An advantage of using the AKO network structure is that, unlike the set system, obscure information (such as birds that are only found in the North or South Island) are only required when a particular frame needs to be matched. This means that for a very large knowledge base, the search efficiency may be improved by not having to process unnecessary frames. Another advantage of this type of frame structure is that the frame system is easy to set up with each frame being separate and distinct. This means that the frame structure is more flexible, making it easier to modify an existing system or design a new one.

One problem with searching through the AKO network is that the search is controlled only by frames that match and does not gain any information when a match fails. For example, when a general frame in the bird network such as a "medium sized bird" fails to match then further frames such as the "small sized bird" should be indicated by the frame. A SEARCH slot similar to the REJECT slot could be added to the frame structure which would contain the name of the frames that should be searched if the current frame does not match. However, although this feature provides an interesting area for future research, it was not implemented in the AKO network devised for the Bird Recognition System for three reasons. Firstly, the system being implemented was not very large, meaning that its effect would be difficult to ascertain and subsequent matches would turn up the respective nodes anyway. Secondly, the SEARCH slot would be less

straightforward to implement. And thirdly, the search would not be conducted through the inheritance links but instead through the SEARCH links (whose effect could be analysed later).

Another problem is that within the existing structure of FRL, there is no way of expressing a frame that has alternative AKO frames. This leads to problems when groups of frames in the network overlap. For example, in the bird AKO network, many birds are both plant and animal eating birds; similarly, many birds are medium to small in size, or are both land and water birds. If all of these features of a bird are included in the AKO slot, then the bird will fail to match a field description which includes only one of the features. Therefore, to overcome this problem, none of the alternative AKO frames can be placed in the AKO slot. This consequently leads to a reduction in the search efficiency and also to problems in trying to design or create the frame. However, additional frames, such as the "omnivorous bird" and "land and water bird" may be added to the network which match only those frames for which an overlap occurs. Alternatively, a method could be devised for defining the different logical combinations of frames in the AKO slot, and this is discussed in section 6.1 as an area for further research.

3.6.3 Further Network Structures

Further network structures can be designed by combining features of the other types of structures. For example, if the number of frames in the knowledge base is large, then the frames can be grouped into a network of separate frame systems with a linear search being used to search the individual frame systems. Alternatively, a set search can be used to determine which part of the network to search first, and then if the search fails further sets can be used to determine which parts of the network to try next. Since the basic structure of a network is to link relevant nodes together, then the various structures described above can also be combined together by linking the separate frame systems into a larger information retrieval network.

3.7 Analysis of the Different Frame Structures.

The above frame structures were implemented for the Bridge System. All the frame structures except for the hierarchical structure and slot network were also implemented for the Bird Recognition System. The hierarchical structure was not implemented for this system because of the difficulty of categorising the features of the bird into various separate sub-trees due to the large number of slots used in each bird frame and also due to the problems of unknowns. These problems were able to be overcome to some extent by using a search tree structure. The slot network was not implemented for the Bird Recognition System because the large number of slots in each individual frame would mean that a large number of links would be required, and this would take some time

unless done automatically.

The mean number of frame and slot matches for the search of the various structures using both a first fit and match all search strategy for the two systems are shown in Figures 3.7.1 and 3.7.2 below. An important point to note when analysing these statistics is that the number of individual frames used in both the systems is not large. Only 15 different types of hands are required for the Bridge System and the Bird Recognition System has a database of 45 birds. Also, the number of examples tried from which the means were obtained for both systems was not large (about 50), with the actual selection of the hands and bird descriptions used in the examples being random. Consequently, the results obtained provide only a rough guide to the effectiveness of each type of search and a larger number of frames and examples would be required for a more careful analysis.

Another important point is the problem of deciding which statistics to use to compare the searches of the different frame structures. Two statistics, the number of slots matched, and the number of frames matched, were obtained for the various structures. As the number of slot matches indicates the number of matching functions that were evaluated, then this statistic is perhaps more relevant for comparing the different structures. A comparison between the two statistics shows that on average less than 1.5 slots are matched per frame. (For the Bridge hierarchy, this figure is less than one because of the extra frames required to define the tree structure). Other statistics, such as comparing the processing time, or the time spent accessing the information in the frames as opposed to matching it, were not taken because of the size of the systems and the difficulty in obtaining the relevant information.

	LINEAR	SET	HIERARCHY	SEARCH TREE	SLOT NETWORK	AKO NETWORK
FIRST FIT STRATEGY						
SLOTS MATCHED	12.4	11.6	6.4	8.0	6.4	8.7
FRAMES MATCHED	8.5	10.1	7.0	7.6	3.9	6.0
MATCH ALL STRATEGY						
SLOTS MATCHED	22.1	13.1	14.5	14.2	16.2	16.8
FRAMES MATCHED	15.0	11.0	19.0	11.4	11.5	12.7

Figure 3.7.1 Mean search statistics for the different Bridge structures.

	LINEAR	SET	SEARCH TREE	AKO NETWORK
FIRST FIT STRATEGY				
SLOTS MATCHED	53.0	36.1	18.5	22.7
FRAMES MATCHED	31.4	34.3	14.7	18.4
MATCH ALL STRATEGY				
SLOTS MATCHED	78.4	45.2	27.0	48.2
FRAMES MATCHED	45.0	37.0	25.9	34.3

Figure 3.7.2 Mean search statistics for the different Bird structures.

The following summary of the effectiveness of each structure is made from the statistics shown :

1. The linear structure is the most inefficient structure.
2. The set structure is inefficient for a first fit strategy, but efficient for a match all strategy.
3. The hierarchical structure provides the most efficient search along with the slot network for a first fit strategy for the Bridge System. However, it is less efficient for a match all strategy.

4. The search tree structure is efficient for both search strategies.
5. The slot network structure is efficient for a first fit strategy but is less effective for a match all strategy.
6. The ako network structure is efficient, but less effective than the search tree structure.

4 IMPROVING THE USER INTERFACE.

4.1 Problems with Interfacing Between the Frame System and the User.

A major problem with any knowledge base is to provide an adequate interface between the information stored in the knowledge base and the user. The users of the knowledge base may require different formats for the information depending on the various purposes for which the information is intended. They may also require different operations such as searching and updating to be performed on the data as well. Therefore, a presentable and easy to work with interface has to be provided for all the various types of users that are likely to use the system.

The types of users of a knowledge base may be categorised into three main groups :

1. the programmer;
2. the expert;
3. the primary user.

The programmer wishes to use the knowledge base at the lowest level where the information is stored in the programming language format. The operations that are required are those that are associated with the maintenance and update of the database, and are provided by the programming language.

The expert needs to be able to create and modify the information in the knowledge base in the form that he is familiar with, whether it be scientific data, mathematical formulae, or English descriptions. In many cases, he is unfamiliar with how the information may be stored on the computer and therefore it is desirable that the expert works independently of the programmer.

The primary user wishes to refer to the expert knowledge stored in the knowledge base without having to understand the complex reasoning or research that has produced it (unless this is the purpose of the knowledge base). Often the user wishes only to enter some data and obtain a result.

Each of these three types of users has to be adequately provided for in a user interface system. Using the existing facilities available in the FRL language, only the programmer is supported to any great extent. Following are some of the methods developed to improve the interface to the expert and primary user.

4.2 Improving the Presentation by Using $ENTER and $DISPLAY Facets.

Presenting the information in the format the user requires is an important feature in the design of a knowledge base. Often the presentation is designed in an ad-hoc manner when a more structured approach can simplify the problem to a large extent.

The problem in presentation occurs because the internal programming representation of the data differs from the format that the expert or primary user wishes to have. For example, the user may wish the data to be presented in a table display or as

understandable English. If the user is required to enter any data, the problem of translating the data to the internal format also arises.

In a frame system, the format required may be different for each individual slot since each slot can describe different types of information. This means that for each slot the data has to be translated both from and to the internal data format. This can easily be accomplished by using $ENTER and $DISPLAY facets for each slot to describe how the data is to be entered and displayed to the user.

The $ENTER and $DISPLAY facets have a similar operation to the $MATCH facet. That is, they may be defined in a generic frame which is linked to the frame being presented through the AKO slot. Any individual frame may also have its own particular facets which overide any that were inherited from the generic frame. This provides a convenient method of presenting special or peculiar information to the user.

The use of these presentation functions would be provided by FENTER and FDISPLAY commands added to the FRL language. Each command would require only the name of the frame to be presented, similar to the FPRINT command. The FENTER command would FADD the values entered by the user into the relevant slots of the frame and hence may initiate IF-ADDED demons. The FDISPLAY command would access the information in each slot through the FNEED command and also may initiate IF-NEEDED demons. This provides a convenient method of maintaining security if there is any sensitive data in the knowledge base as the IF-NEEDED demons can check whether the

user is allowed to display the data. Definitions of the FENTER and FDISPLAY commands in LISP are given in the Appendix A.

An example of an $ENTER function used in the Bird Recognition System is shown below :

```
(defun ask-size ()
/* Reads in the size of a bird. (large medium small)
    (prog (size)
         loop
         (prin1 '"estimate of bird's size ")
         (print '"(large, medium or small) :")
         (prin1 '"(example : small = sparrow, ")
         (print '"medium = magpie, large = mallard duck)")
         (setq size (lower (read atom)))
         (cond
              ((member size '(large medium small ()))
                   (return size)
              )
              ((member size '(h help))
                   (help 'size)
                   (go loop)
              )
              (t   (prin1 '"invalid size - ")
                   (print '"must be large, medium or small")
                   (go loop)
              )
         )
    )
)
```

Figure 4.2.1 Example of function definition for the $ENTER facet for the SIZE slot of a bird.

The above function reads in the the size of the bird as estimated by the user. The only requirement of the function is to read and return the slot value in the relevant format. Interactive messages and references to helpful information (such as an explanation of the what needs to be entered or a list of words that are allowed to be used) are included in the function and help to guide the user. Checking of the data can also be accomplished here to ensure that the information is correct before it is entered into the knowledge base. Note that no arguments need to be passed to

the function. If no function is defined in the $ENTER facet, then the default LISP function READ is assumed by the FENTER command.

An example of an interactive session designed for the Bird Recognition System that uses the $ENTER functions is shown below.

```
Enter a name for the bird : BIRD1

Enter estimate of bird's size (large, medium or small) :
LARGE

Enter a description of the bird's appearance:
MAINLY WHITE, RED BILL

Enter the words that best describe the bird's distribution :
HELP

The distribution feature indicates where the bird is usually
found in New Zealand. Only the main districts of New Zealand are
used such as Hawkes Bay and Mid Canterbury.

The list of valid New Zealand districts are :

New Zealand        North Island       South Island
Auckland           North Auckland     South Auckland
Coromandel         Hauraki Gulf       Bay of Islands
Waikato            Rotorua            Bay of Plenty
Horowhenua         Manawatu           Taranaki
Wanganui           Hawkes Bay         Poverty Bay
Volcanic Plateau   King Country       Wairarapa
Wellington         Marlborough        Marlborough Sounds
Nelson             Kaikoura           West Coast
Canterbury         Mid Canterbury     South Canterbury
Otago              North Otago        South Otago
Southland          Fiordland

Enter the words that best describe the bird's distribution :
MANAWATU, WAIRARAPA

Enter the words that best describe the bird's habitat :
HELP
The habitat feature describes the habitats where the bird is
likely to be found. Common habitats are : open, alpine, coast,
gardens, habitation, forest.

The following are valid words :
alpine         clearings      cliffs         coast
crops          drains         forest         gardens
habitation     hedges         hill-country   lagoon
lake           marsh          open           orchards
pasture        plantations    pond           river
roadside       scrub

Enter the words that best describe the bird's habitat :
```

```
MARSH LAKE

Enter a description of the bird's behaviour:

Enter a description of the bird's flight:

Enter the words that best describe the bird's food :
PLANTS INSECTS

Was there a nest found ? YES

Enter the words that best describe the bird's nest material :
STICKS

Enter the location of the bird's nest :
ON GROUND, NEAR WATER

month in which nest was found :
JAN

How many eggs were found? 7

Enter a description of the colour of the egg:
MAINLY CREAM, RED SPOTS
```

Figure 4.2.2 Interactive session for entering a bird produced by the FENTER command.

More complicated interactive sessions with the user may also be designed by passing global information between the functions to indicate which of the succeeding questions are relevant. For example, in the session shown above, if the user reply to the question "Was there a nest found?" is NO, then the succeeding questions about the nest location and nest material are skipped by the use of a global flag.

In some applications, the user is not required to enter data for each slot. In the Bridge System, for example, the only information entered by the user is the cards in his hand from which the relevant data for each slot is obtained. However, the $ENTER facets can still be used as a means of obtaining this information since the operation required is still translation into the internal data format.

An example of a $DISPLAY function is shown Figure 4.2.3 below.

This function converts the breeding season passed to it in the internal LISP format into a form presentable to the user. Notice that special cases (such as a breeding season that continues throughout the year) can be checked for and a suitable display devised. The $DISPLAY function only requires one argument which is then set to the value of the slot to be displayed by the FDISPLAY command. If no function is found, then the default LISP function PRINT is used.

The display produced for the Starling in the Bird Recognition System is shown in Figure 4.2.4.

A major advantage of using the $ENTER and $DISPLAY facets is that the definition of the functions is independent of how the information in the knowledge base is accessed and stored. The problem simply becomes one of translating the information into a form relevant to the user and each function can be treated separately. Therefore, the design of the presentation can be easily structured with more elaborate functions provided as the need arises.

The use of the presentation functions in the generic bird frame of the Bird Recognition System is listed in the Appendix B. The frame demonstrates the usefulness of such functions in designing a knowledge base. A common function required of the Bird Recognition System is to enter and display a list of words that are associated with a particular feature. This is done in the frame listed by using the functions ASK-WORDS and PRINT-WORDS for each of

```
(defun print-season (months)
/* Writes out the MONTHS in the breeding season.
/* Eg.      (january to march) is converted to "January to March".
    (prog ()
          (setq months (capital months))
          (cond
               ((atom months)
                    (prin1 months)
               )
               ((equal (car months) (caddr months))
                    (prin1 (car months))
               )
               ((equal months '(January to December))
                    (prin1 '"All year")
               )
               (t   (map car cdr prin1 months))
          )
    )
)
```

Figure 4.2.3 Example of function definition for the $DISPLAY facet in the BREEDING slot of a bird.

the relevant slots. Another common function required is to enter a description of certain features of the bird. The problem of recognizing and interpreting an English description is complex and part of the problem of understanding natural language. However, various methods such as using a dictionary of relevant synonyms or restricting the vocabulary to dissimilar words is sufficient for the purpose required here. By having the ability to define separate functions for presentation, problems such as recognizing descriptions can be treated separately and improved at any stage.

```
Starling
     size
          21
     appearance
          overall
               mainly black green purple glossy, short tail,
               black body, short pointed wings, pointed bill
          winter plumage
               mainly black spotted streaked buff, black body,
               black pointed bill
          male breeding plumage
               mainly black purple green glossy, black body,
               yellow bill
          female breeding plumage
               mainly black buff, black body, yellow bill
          immature
               mainly dull brown buff, black body, white throat,
               dark bill
     distribution
          New Zealand
     habitat
          not forest
     food
          worms, insects, fruit
     breeding
          September to January
     nest material
          straw, grass
     nest location
          in hollow building tree cliff bank
     number of eggs
          4 to 6
     egg colour
          mainly plain pale blue glossy
```

Figure 4.2.4 Display produced for the Starling using the FDISPLAY command.

4.3 Improving the User Search Interface.

An important operation of a knowledge base required by the user is that of searching for the relevant information stored. This involves finding the list of frames in the frame system that match certain conditions.

4.3.1 Specifying the search using the FENTER command

The use of the FENTER command provides an easy means of allowing the user to enter the types of information that he wishes to search for. In the Bird Recognition System, the user wishes to find the list of birds that match various field observations which are entered through the $ENTER functions. However, by allowing the user to skip any of the features that are to be entered, any range or combination of information can also be specified. This means that a more elaborate query system can easily be developed where the user can select different subsets of the information stored in the knowledge base. For example, the user may wish to find all the birds that are small and have black bodies rather than trying to search for a bird that matches a more precise set of field observations. In this case, the user would enter the information about the size and appearance of the bird, and would skip any remaining questions. Hence, the use of the FENTER command provides a straightforward means for the user to specify the search.

4.3.2 Using $IF-MATCHED demons to provide a trace of the search

The user-search interface can also be improved by providing a trace of the search if the user requires. This can be accomplished by using $IF-MATCHED demons that are activated whenever certain slots are matched. These demons can display the result of the match and indicate (with reasons) which of the other frames in the frame system are likely or unlikely to be matched. An example of the use of this technique is shown in Figure 2.5.1. The traces produced for the Bird Recognition System are also shown in the Appendix D.

4.3.3 Using interactive matching functions

A further method of improving the user-search interface is to allow the user to guide the search. Whenever the search becomes doubtful, or the matching process uncertain, the user can be queried about certain aspects of the information required. This can be achieved by making the $MATCH functions interactive so that the functions themselves ask the user the relevant questions if the information provided is incomplete. The information obtained from the answers can be added to the frame of information provided by the user and can also determine if the match is successful or not.

For example, in a set structured system designed for the Bird Recognition problem, the matching of the sets could be performed by interrogating the user. The frame definition for an omnivorous bird could be as shown in Figure 4.3.3.1.

```
(deframe "omnivorous bird"
    (ako
         ($value
              (bird)))
    (food
         ($value
              (("Is the bird omnivorous? ")))
         ($match
              ((ask-question))))
    (reject
         ($value
              (("new zealand dabchick" "white faced heron"
                "bittern"...))))
)
```

Figure 4.3.3.1 Using an interactive matching function to match an omnivorous bird.

The actual content of the question asked may be constructed in a number of ways using the slot value and the match function. For example, in the frame defined above for the omnivorous bird, the question was passed directly to the match function ASK-QUESTION. Alternatively, the question could be constructed by the match function from the slot values passed to it.

An alternative method of defining the omnivorous bird is to use a non-interactive match function which checks that the food eaten indicated whether the bird was both plant eating and animal eating. However, because the description provided by the user may be incomplete or the matching process uncertain, it is much easier and more accurate to interrogate the user.

One problem with this technique is how to organise the format of the questions asked. Due to the nature of the match functions used, it is easiest to ask questions which expect one of three replies : YES, NO or DON'T KNOW (expressed by skipping the question). These answers are equivalent to the values TRUE,

FALSE and UNKNOWN respectively which are returned by the matching functions.

This method of improving the user search interface is particularly useful for a system which uses a search tree. By asking questions throughout the search tree, the search has a similar format to a decision tree, with the questions becoming more specific at lower levels of the tree. For example, by adapting the bird search tree to use interactive matching functions, the output shown in Figure 4.3.3.2 could be produced.

```
Is the bird a small sized bird ? NO
Is the bird a large sized bird ? YES
Is the habitat of the bird located near water or marsh ? YES
Are the underparts or upperparts of the bird blackish ? YES
Is the bill white or red ? YES

A list of possible birds that could match are :
White Faced Heron
Canada Goose
Mallard
Mute swan
```

Figure 4.3.3.2 Using an interactive search tree to find a bird.

A whole series of interactive matching functions could be used so that the user could guide the search of the knowledge base. However, there are various problems associated with this method. One problem is that if too many questions are asked, then the usefulness of the system will decrease. Another is that some method is required of remembering questions already answered so that the same question is not re-asked. This involves updating the frame of infomation supplied by the user from the ENTER functions using the information gained from the

answers during the matching process. A YES reply means that the information can be directly added to the slot value in the frame, but NO and DON'T KNOW replies must also be stored in the frame and are important for subsequent matches.

Another problem with using interactive matching functions is that the sequence of questions asked must be easy to follow for the user. For example, an alternative matching scheme for the Bird Recognition System was devised so that the user was interrogated about the appearance of the bird during the search by using the appearances of the birds stored in the knowledge base. A sample trace of the search for a white backed magpie is shown in Figure 4.3.3.3 below.

```
A medium-sized bird matches the size of the Banded Rail
which is 30 cm.
Is the upperparts olive brown black white spotted ? NO
The bird does not look like the Banded Rail.
The bird is not a Banded Rail.

A medium-sized bird matches the size of the Song Thrush
which is 23 cm.
Is the underparts pale grey brown white ? NO
The bird does not look like the Song Thrush.
The bird is not a Song Thrush.

A medium-sized bird matches the size of the Starling
which is 21 cm.
Is the tail short ? NO
Is the bird mainly blackish speckled buff ? NO
Is the body black purple green glossy ? NO
Is the throat white ?
The bird does not have the overall appearance of the Starling
nor does it look like the winter plumage, female breeding
plumage, immature or male breeding plumage Starling.
The bird is not a Starling.

A medium-sized bird matches the size of the Black Backed Magpie
which is 40 cm.
Is the bird mainly black white ? YES
Is the back black ? NO
Is the hindneck white ?
Is the hindneck mottled grey ?
Is the underparts mottled grey ? NO
The bird does not have the overall appearance of the Black
```

```
Backed Magpie nor does it look like the male, immature or female
Black Backed Magpie.
The bird is not a Black Backed Magpie.

A medium-sized bird matches the size of the White Backed Magpie
which is 42 cm.
Is the back white ? YES

The bird looks like the male White Backed Magpie.

Is the back grey ? NO

The bird does not look like the immature or female White Backed
Magpie.

The male White Backed Magpie matches the description.

A medium-sized bird matches the size of the Rook which is 41 cm.
Is the bird mainly bluish purplish ? NO
The bird does not look like the Rook.
The bird is not a Rook.


The list of birds that match the description are :
male White Backed Magpie
```

Figure 4.3.3.3 Trace of search for a white backed magpie using interactive matching functions.

The problems described above in implementing an interactive search are demonstrated here. The large number of questions needed to be answered detract from the usefulness of the system. Also, it is very easy for the user to produce unsatisfactory results if he does not wish to enter sensible replies. For example, by answering YES to all the questions then all the frames in the system will match. Another problem is that a complicated system of adding the information obtained from the answers has to be devised that prevents the questions being re-asked and also produces a satisfactory match for subsequent frames.

However, the main problem in using a technique such as this is that it is difficult to design the system so that the questions asked are relevant and easy to follow by the user. This is demonstrated in the trace above by the questions becoming too specific for some birds (such as the Starling) that eventually do not match. Also, some of the later questions (such as "Is the bird mainly purplish bluish?" when the question "Is the bird mainly black white?" has already been asked) seem unnecessary to the user even though it is possible, using a rudimentary matching scheme, that some birds might have both features. It is also more difficult to maintain relevance to the user for frame structures such as the set structure and the network structure.

Therefore, when trying to improve the user search interface by using interactive match functions, care has to be taken in designing the format of the interaction. For a straightforward and easy to follow system, emphasis should be placed on having an automatic matching process with only the more difficult matches queried.

4.4 Allowing the User to Modify the Knowledge Base.

An important goal of any user interface system is to make the user unaware of how the knowledge base is structured. For the problems of search and presentation where the user needs only to obtain the information already stored then this goal can easily be achieved by the methods outlined above. However, it is more difficult to achieve this goal for the expert user if he is also allowed to modify the knowledge base.

Modifications of the knowledge base involves three aspects :

1. Adding a frame to the frame system.
2. Removing a frame from the frame system.
3. Altering information in a frame.

4.4.1 Addition of frames to the knowledge base.

Additions of frames specified by the expert to the knowledge base may be accomplished by using the FENTER command to build up the new frame. The format of the information in the frame can be in the form that the user is familiar with by using different $ENTER functions to perform the translation of the data into the internal data format. The $ENTER functions can also ensure the integrity of the knowledge base by checking that the information being added is correct.

However, there are further problems associated with adding frames into the different types of frame systems. These are discussed below for the different structures.

1. Linear Structure.

As each frame in a linear structured frame system is separate, with no links to any other frames, the addition of any further frames to the frame system is easily accomplished by adding the name of the new frame to the list of all the frames in the system.

2. Set Structure.

The addition of a frame to a set structured frame system requires that the name of the frame is added to the relevant INSTANCE and REJECT sets. This may be accomplished in two ways:

1. Use the match functions to automatically decide whether a frame belongs to a set by matching the new frame against all the set frames and then adding the frame name to the relevant sets depending on the match.

2. Interrogate the user about whether the frame belongs to a particular set or not by using interactive match functions.

It is also possible to allow the expert to add set frames to the frame system using the methods described above. In this case, the user would add a general frame description of a bird, and the INSTANCE and REJECT slots would be automatically created

by matching the new frame against all the other frames in the system.

3. Hierarchical Structure.

New frames can be added to a frame hierarchy by using the frames in the tree to interrogate the user about the position the new frame should be placed in. If the hierarchical structure is used solely for defining a search tree, the new frame can simply be added as a separate frame and the search tree can remain unchanged except for updating the terminal nodes in the tree that match with the new frame.

4. Network Structures.

There are various problems associated with adding a new frame to the different network structures. For the slot-network structure, where each slot in a frame has a list of alternative frames to try, the new frame can be added into the network by adding links from common nodes to the new frame thus ensuring that the node is reached in the search. However, the problem of relevance in the link (where the information obtained in the slots already matched is used to specify a more relevant set of alternative nodes to try) means that the addition of nodes in a slot-network structure would affect the efficiency and the usefulness of the search.

The problem of adding a frame to an ako-network is much more easily overcome. This is because each frame is separate except for the values in the AKO slot. The AKO slot of the new frame can be automatically created by finding out which frames in the network match with the new frame. Any REJECT slots would also have to be updated during this process. However, a check needs to made to ensure that any AKO information is eliminated where there are overlapping groups (such as small and medium birds in the Bird Recognition System) as described above.

4.4.2 Removal of frames from the knowledge base.

The removal of frames from the knowledge base can be accomplished by using $IF-REMOVED demons to specify the removal operations for each slot of a frame. Each function can also ensure the integrity of the knowledge base by checking that the operation is allowable.

There are no major problems associated with the removal of frames from the linear and set frame structures. This is because each frame can be removed simply by removing any reference to the frame from the frame system. However, the efficiency of the search for the set system may be affected if arbritrary removal of set frames is allowed.

Removal of frames from a hierarchical structure is very difficult as information relating to lower levels of the tree may be contained in the node being removed. This can be overcome by using a static search tree where only the references

to the separate frames are altered when a frame is removed. A similar problem of removal from a hierarchy exists for the AKO network if arbitrary removal of the general AKO frames is allowed. However, like the search tree, removal of specific frames is possible.

The main problem with removing frames from a slot network is to ensure that every other frame in the network can still be reached from the starting node after the removal. For example, if a node can only be reached from another node by searching through a third node, then removing the third node from the network would also remove the link between the other two nodes. Therefore, the relevant pathways have to be maintained to ensure a systematic search for the slot network.

4.4.3 Alteration of a frame in the knowledge base.

The following method can be used to allow an expert to change an existing frame in the knowledge base :

1. Use FDISPLAY to display the frame.
2. Ask which slots are to changed.
3. Use $ENTER functions to enter the new information into the relevant slots.

A more elaborate frame alteration mechanism could provide editing facilities for the frame displayed but could still use the $ENTER functions by using a global flag to inhibit any interactive messages.

The problems associated with the effect of the alteration of the different frame structures are outlined below.

1. Linear Structure.

There is no effect of altering a frame on the linear structure as each frame definition is independent and the name of the frame remains the same.

2. Set Structure.

Once a frame has been altered, its membership in the INSTANCE and REJECT sets of a set structured system is affected. This means that to maintain the integrity of the knowledge base, first the old frame has to be removed from the knowledge base, and then the new frame added as described above. Therefore, the problems associated with the addition of a new frame into the set structure also apply to the alteration of an existing frame.

3. Hierarchical Structure.

The alteration of a frame in a hierarchical frame structure would be difficult unless a separate search tree structure is employed where the information in the tree is static except for the terminal nodes that specify which frames to match. The terminal nodes can be updated by removing any references to the old frame, and then the new frame can be matched against the search tree to find which terminal nodes should contain the name of the new frame.

4. Network Structures.

For the slot-network structure, the alteration of the slot value affects the search through the network. Therefore, it is difficult to allow any alteration of frames in a slot-network structure and still maintain the relevancy and correctness of the search.

However, for the ako-network, the frame structure may be maintained by matching the altered frame against the other frames in the network to find which frames should be included in AKO slot of the altered frame,. The AKO, INSTANCE and REJECT slots in the rest of the network would also have to be updated.

4.4.4 The effect of modifying the knowledge base on the efficiency of the search

Allowing the user to modify the knowledge base presents further problems in maintaining the efficiency of the search in the various frame structures. For the linear structure, a large number of additions of frames to the frame system will dramatically reduce the efficiency of the search.

For the set structure, the effectiveness of the search is dependent on the frames added or modified by the user since these operations affect the membership of the sets. If there are a large number of additions to the knowledge base then these need to be categorised by further general set frames added by the user if the search is to be efficient.

In the hierarchical system, modification of the knowledge base is only feasible if the search tree remains fixed. This means that the efficiency of the search will decrease with each new frame added.

For the slot network structure where modification of the frames affects the search pathways, the decreasing relevance of the search will consequently affect the search efficiency. For the ako-network, the problem of maintaining the efficiency of the search is the same as for the set structure; that is, the efficiency of the search is directly dependent on the information added or changed by the user.

4.5 Allowing the Expert to Create the Knowledge Base.

Allowing the expert to create the knowledge base is a far more difficult problem than modification of an existing knowledge base. The aim is to allow an expert who is completely unfamiliar with the computer (such as a bridge player) to be able to create and use his own knowledge base.

The main difficulty in achieving this aim is the problem of expressing procedural or functional information to the expert. The knowledge in a frame system consists of two aspects - the data structures, and the attached procedures. The problem of allowing the user to create the data structures is straightforward and involves entering the data the expert supplies into a frame type format using the methods described above. The problem of how to specify the attached procedures which are used in matching,

searching and presenting the frames is much more difficult as the final information must be expressed in the programming language.

Two methods for specifying the procedural information are proposed below.

4.5.1 Using lists of pre-defined functions to specify the procedural information.

One solution is to use a list or library of functions that have been pre-defined by the programmer. There would be separate libraries for the different operations required such as searching and presentation. For example, some useful matching functions that could be pre-defined are NUMBER, which matches number ranges such as ABOVE 5 and 1 TO 10, and NOUN-DESCRIPTION which matches an English description consisting of adjectives and nouns. The expert could then specify the functions he requires in a similar manner to that shown in Figure 4.5.1.1.

The system could also allow the user to partially define his own match functions. For example, a user could define a function called HABITATS which matches the list of words specified by the user that are valid habitats. Alternatively, the pre-defined match function provided could be an elaborate one which uses a large dictionary to match words of similar meaning.

For the definition of the presentation functions, the enter and display functions could be assumed by default to be those indicated by the match function, for example, READ-NUMBER and PRINT-NOUN-DESCRIPTION. A facility for using a library of

```
What is the name of the system ? BIRD RECOGNITION SYSTEM

What type of information do you wish to describe ? BIRD

What features do you wish to describe the BIRD with ?
SIZE, APPEARANCE, HABITAT, FOOD, DISTRIBUTION

Define the type of information each feature describes using any
of the following alternatives :
NUMBER        RANGE             NAME              WORDS
NOUNS         NOUN-DESCRIPTION  DISTRICT          ADDRESS
DATE          MONTH

size ? NUMBER
appearance ? NOUN-DESCRIPTION
habitat ? NOUNS
food ? NOUNS
distribution ? DISTRICT
```

Figure 4.5.1.1 Using pre-defined functions to allow a user to define a bird.

pre-defined presentation functions could be used as well. A library of edit functions such as CAPITALISE and JUSTIFY could also be supplied for allowing the user to design his own display. For the enter functions for which elaborate interactive sequences are required, the expert could be interrogated about the text of the questions to be asked and about the help information to be provided. Otherwise, the format of the text and help information can both be assumed by default from the name of the feature and the help available on the match function.

4.5.2 Interrogating the user about the procedural information

A further method of specifying the procedural information is to use an interactive approach where the user is interrogated about the procedural information required and the functions are then designed by the computer. For example, a function can be designed by the programmer for the Bridge System which allows the expert to design his own scoring system. Figure 4.5.2.1 shows a trace of how the Acol scoring system for honour points could be defined by the user.

More elaborate interrogation systems can be designed by the programmer where the expert can define his own variables and also define how the various features of each frame are produced from these variables by using a specific language format. The two methods of using libraries of pre-defined funcions and an interactive approach may also be combined together to provide for the expert a wide range of methods of specifying how the information in the frame system is to be presented and matched. Once this has been done, the expert can add individual frames to the frame system by using the enter functions already defined.

4.5.3 Problems with allowing the expert to create the knowledge base.

There are various problems associated with using the techniques described above to allow the expert to create his own knowledge base. One problem is that various structuring methods which might produce an efficient search (such as the hierarchical structure and the slot network structure) cannot be implemented because the information in the knowledge base needs

```
Enter the name of the feature : HONOUR POINTS

Which suits do you wish to count points for ?
HEARTS, SPADES, DIAMONDS, CLUBS

Which card(s) do you wish to allocate points for if found in any
of these suits ?

    which card(s) ? ACE
    how many points ? 4
    which card(s) ? KING
    how many points ? 3
    which card(s) ? QUEEN
    how many points ? 2
    which card(s) ? JACK
    how many points ? 1
    which card(s) ? TEN
    how many points ? 0.5
    which card(s) ?

Do you wish to add any more points for the length of each
suit ? YES

    length (eg. 3, above 4, below 5, 3 to 6) : ABOVE 5
    how many points : 2
    length (eg. 3, above 4, below 5, 3 to 6) : 5
    how many points : 1
    length (eg. 3, above 4, below 5, 3 to 6) :
```

Figure 4.5.2.1 Allowing the expert to define his own scoring scheme in a Bridge System.

to be pre-defined for these methods. The only structures that can be used are the linear, set and ako-network structures and even here the efficiency of the search depends on what or how many frames the expert decides to put in the system. Also, the search method provided by the programmer has to be a general search rather than one that is specifically designed for the purpose.

Another problem is that it is difficult to cater for all possible occurrences that may arise in the creation of a knowledge base. For example, the problem of special cases of information which require separate match functions (such as for different habitats of a bird) means that a method has to be

devised to allow the user to specify any special match function. Also, some types of functions are not suited to being defined by the user, an example being $IF-MATCHED functions which are used for tracing the search, and for providing an interactive matching scheme. Some systems such as geographical and scientific knowledge bases may also be difficult to design general functions for.

To get around specific problems, a facility could be provided for the programmer to define functions that are specially designed for certain users. However, rather than designing a general knowledge base format which can be used for any purpose, it is much easier to design a knowledge base specifically for a purpose such as playing Bridge and then allowing the expert to modify the existing frame system for a particular application.

5 COMPARISON OF KNOWLEDGE REPRESENTATION METHODS.

5.1 Features of Knowledge Representations.

Various aspects that determine the effectiveness of a particular method of representing knowledge are described below.

Modularity.

Davis and King [1977] define modularity of a program as "the degree of separation of its functional units into isolatable pieces". The separation of the formulation of the knowledge for a particular representation into distinct functional units or modules which may be modified individually without problems of interaction allows for a quicker and more structured approach to the design and use of knowledge bases.

Expressibility.

The expressive power of a knowledge representation is determined by the ease with which different types of information and their relationships may be formulated within it. Other considerations, such as how 'natural' the information can be expressed, and the problems involved in describing how information is to be matched are also determined to some extent by the expressive power of the representation.

Retrievability.

The ease with which a wide range of information may be retrieved determines the usefulness of a particular method of representing knowledge. Important aspects related to retrieval of information are the efficiency of the search, the range of information that can be obtained using general as well as specific queries, and the method of search such as forward or backward reasoning.

Modifiability.

Being able to easily modify existing information is an important aspect in the design of a knowledge base. Operations such as alteration, addition, deletion and replacement should all be possible without a major redesign of the system being required.

Consistency.

Maintaining the consistency of the information stored is another important consideration in the design of a knowledge base. Initially, the knowledge base has to be thoroughly tested to ensure that the information stored is relevant and consistent. The consistency of the information has to be subsequently maintained when the knowledge base is used or altered. The ease with which this is possible is an important criterion for selecting a particular method of knowledge representation.

Transparency.

The transparency of the information being represented is the ease with which its meaning may be understood by the designer and/or user of the information. A particular method of representing knowledge may provide a powerful means of formulating various

relations and facets, but if the representation is difficult to understand or difficult to use, then the expressive power of the system is wasted.

Adaptability.

The range of problems to which a particular method of knowledge representation may be applied to is a further means of comparing the effectiveness of the different methods. The relative ease to which various applications may be formulated by a representation also determines its effectiveness.

5.2 Methods of Representing Knowledge.

Various methods of knowledge representation on the computer such as production systems, frames and semantic networks have been researched extensively over the last decade. A comparison of the effectiveness of each of these different forms of knowledge representation is made below. The features described above such as modularity and transparency are considered for each representation.

5.2.1 Production Systems.

Productions Systems were first proposed by Post [Post, 1943]. They are made up of a list of 'if-then' rules which describe a set of conditions and a set of actions which occur if those conditions are met. Davis and King [1977] defined a 'pure' production system "as consisting of three basic components : a set of rules, a data base, and an interpreter for the rules." A simple production system might then consist of a data base of ideas such as "has wings" and "flies"; a set of if-then rules which contain an ordered pair of ideas; and an interpreter whose function it is to search the rules to find those whose ideas in the 'if' part of the rule are satisfied by the database, and then to replace these with the set of ideas listed in the 'then' part of the rule. Figure 5.2.1.1 illustrates part of a simple animal recognition production system.

Production systems have been researched extensively over the last decade. They have been applied to a wide range of applications including medical consulting [MYCIN : Shortliffe, 1976; Davis, Buchanan, and Shortliffe, 1977], human learning [Hedrick, 1976; Vere, 1977] and draw poker [Waterman, 1970]. As well as demonstrating an ability to represent many different types of knowledge, production systems have also proved useful in building interactive expert systems where knowledge has been previously acquired from human experts and an explanation of any reasoning is provided for human users to follow. Examples of such expert systems are MYCIN, DENDRAL [Feigenbaum et al., 1971] and PROSPECTOR [Duda et al., 1978].

The major advantage of using production systems to represent knowledge is its modularity. As each production rule is an independent part of the knowledge base, it is easy to add, modify or remove information from the system. Another important feature of production systems is that they seem to provide a natural means of expressing knowledge in a manner similar to that used by human experts.

Barr and Feigenbaum [1981] outline two important disadvantages to production systems : inefficiency and opacity. The advantages of having a modular and uniform representation is offset by the inefficiency of program execution as the state of the system is re-evaluated after a rule has been parsed. Further, the flow of control is not readily transparent to the designer, with algorithmic knowledge being difficult to incorporate into the rules.

Another limitation to production systems is that certain types of knowledge are not suited to being represented by it. Davis and King [1977] propose three areas where production systems are inappropriate : in domains where there is a concise unified theory (for example, mathematics); domains with "complex collections of multiple, parallel processes"; and domains where control flow and knowledge are combined together in the system.

In comparing the effectiveness of using production systems as a means of representing knowledge as opposed to using frames, the general features of each representation such as modularity and transparency may be considered. This is described below in section 5.2.3. However, another effective means of comparing the

```
IF "has feathers" THEN "is bird"

IF "flies", "lays eggs" THEN "is bird"

IF "is bird","does not fly","has long neck","is black and white"
THEN "is ostrich"

IF "is bird", "does not fly", "swims", "is black and white"
THEN "is penguin"

IF "is bird", "flies well"
THEN "is albatross"
```

Figure 5.2.1.1 Part of an Animal Production System.

two different methods of representation is to apply them to the same problem. For example, a frame system could be developed for a problem for which a production system has been applied to. Alternatively, a production system could be designed to play Bridge or recognize birds, the two areas to which frames have been applied in this thesis. By developing these similar systems, knowledge can be gained about the problems of representing certain types of information for each of the particular representations. The effectiveness of each system can also be readily determined.

Part of a production system for recognizing animals is listed in Figure 5.2.1.1. As a very simple production system, it is a good example to use for developing a similar frame system for comparing the two different methods of representation. A sample frame system that describes the same information is shown in Figure 5.2.1.2.

Certain limitations of using frames compared to using production systems are illustrated in this example. For example, first order logic such as the bird "flies" and "swims" can be easily incorporated into production systems. Modifiers can also

```
(deframe "bird1"
    (appearance
         ($value
              ((has feathers))))
)

(deframe "bird2"
    (behaviour
         ($value
              ((lays eggs) (flies))))

(deframe "bird"
    (match
         ($value
              ((or "bird1" "bird2"))))
)

(deframe "ostrich"
    (ako
         ($value
              (bird)))
    (appearance
         ($value
              ((long neck) (black and white))))
    (behaviour
         ($value
              ((does not fly))))
)

(deframe "penguin"
    (ako
         ($value
              (bird)))
    (appearance
         ($value
              ((black and white))))
    (behaviour
         ($value
              ((does not fly) (swims))))
)

(deframe "albatross"
    (ako
         ($value
              (bird)))
    (behaviour
         ($value
              ((flies well))))
)
```

Figure 5.2.1.2 FRL definition of the animal system.

be added such as the bird "flies well" or "does not fly", and the bird has a "long" neck. For frames, this is more difficult because the information has to be represented as an attribute and

its value. It is not satisfactory to place the idea that the bird "flies" under the slot for behaviour which is only a very general classification. Ideas such as "flies" as well as "flies well" need to be expressed in a uniform manner.

Another problem with frames is the difficulty in expressing alternative conditions. For example, in the animal system above, a bird can be described by "has feathers" or by "lays eggs" and "flies". The frame solution devised in Figure 5.2.1.2 where two separate bird frames are defined ("Bird1" and "Bird2") seems cumbersome. Again the problem of describing different alternative combinations of slots, and also the problem of describing alternative AKO paths limits the expressive power of the system.

As a second means of comparing production systems and frames, a production system could be developed along the same lines as the Bird Recognition System, thus bringing to light any further limitations of the two approaches. For example, a list of production rules could be defined to describe the features of a bird such as the Pukeko (shown in Figure 3.3.1). However, problems are encountered almost immediately. The large number of slots or features of a bird that need to be described mean that a very long-winded production rule is required. To use production rules to their best effect, the information should be split up into smaller sub-rules, and a complex system gradually built up from them. The information described in the slots such as breeding season and number of eggs which express information in various ranges, and also food and habitat, which use lists of words, makes it difficult to express the information into smaller

sub-categories, or to group the information into a smaller list of slots. The advantage of frame systems is that a whole series of features can be itemized and grouped together for a particular object or idea.

5.2.2 Semantic Networks and Property Lists.

Semantic Networks were first developed in 1968 [Quillan, 1968]. They consist of a network of nodes connected together by labelled arcs. The node from which an arc originates may be considered as being the object, with the arc being an attribute and the destination node being the value associated with it. Figure 5.2.2.1 gives an example of a small semantic network.

A semantic network is usually represented on the computer using property lists. A property list consists of a list of properties or attributes and their values. For example, a property list describing the appearance of a pukeko is shown in Figure 5.2.2.2. Semantic networks can be represented using such property lists by associating with each node in the network a property list containing the arcs as the properties and the destination nodes as the values.

Frames also are represented on the computer using property lists. Each slot in a frame is equivalent to a property in a property list. The value associated with each slot is another property list of facets and their values. Therefore, semantic networks and frames may be considered as being similar methods of

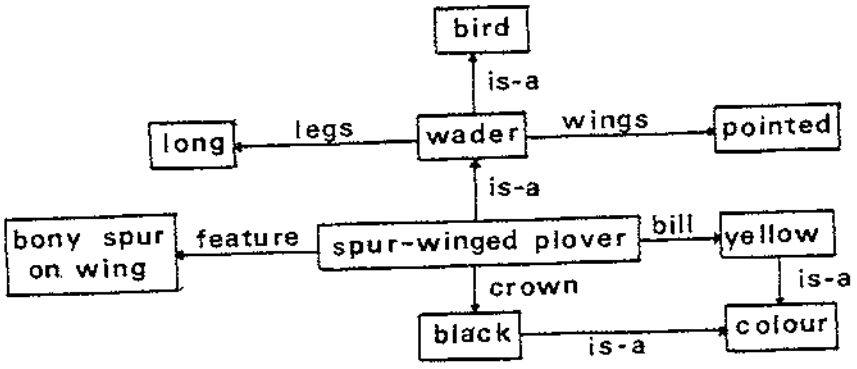

Diagram 5.2.2.1 A Semantic Network.

representing knowledge. In fact, semantic networks may be considered as being a subset of the frame method of representation where the slot values are used to represent the arcs and nodes linked to a particular node in the network. Aspects of semantic networks such as inheritance through the IS-A arc are provided in FRL through the AKO and INSTANCE slots. Frames also provide more comprehensive features such as procedural attachment and defaults which are not available in semantic networks (but could be included).

```
(    mainly      (bright blue black)
     bill        (large red)
     shield      (red)
     coverts     (white)
     thighs      (black)
     abdomen     (black)
     flanks      (purple blue)
     breast      (purple blue)
     body        (blue black)
     neck        (purple blue)
     throat      (purple blue)
     head        (black)
     upperparts (black green glossy)
     feet        (pale orange red)
     rump        (white)
)
```

Figure 5.2.2.2 Property list describing the Pukeko's appearance.

The term semantic network or semantic net has been used widely in AI research to describe a wide range of different systems. Often, the only similarites between some so-called semantic networks is the graphical representation of nodes and arcs used to describe them. Some forms of semantic networks try to represent aspects of predicate calculus through the use of quantifiers [Schubert,1975] and the use of 'partitioned' semantic networks [Hendrix, 1976] where groups of nodes of arcs are grouped together into a common unit. Other semantic networks, such as a 'procedural' semantic network [Levesque and Mylopoulous, 1979] which use classes, meta-classes and procedural information to represent the semantic information, have similarities more in common with frame-like representations.

As a means of expressing a simple relationship, semantic networks provide an easy to understand visual aid. However, as the complexity of the system increases, the usefulness of the graphical representation diminishes and it becomes increasingly

difficult to understand. The natural advantages of predicate calculus and frames is lost in the complexity of the graphical representation.

5.2.3 Frames.

The use of frames as a method of representing knowledge is described above in the previous chapters. Frames, as a form of knowledge representation, offer a wide variety of features with which to express knowledge. Features such as procedural attachments, inheritance and defaults, which are not readily available in production systems and semantic networks, offer a comprehensive range of tools for the knowledge expert to work with. In addition, frames provide a useful means of grouping together in a single place all the relevant information about a certain object or idea.

In comparing frames with production systems and semantic networks, I have described above aspects of representations such as modularity and transparency that determine the effectiveness of each method. I will now consider these aspects with frames in mind, as well as comparing them with the properties of production rules in particular.

Frames by their nature provide a modular environment, as all pieces of information related to a certain situation may be grouped together in a single place. Frame hierarchies also provide a means of progressively describing situations in a more

specific manner, with related ideas expressed through inheritance. In production systems, the system often becomes fragmented because the detailed information is described separately in individual rules. For example, if a production rule system of the Bridge System were designed, the information related to each hand would have to be built up from a large set of smaller underlying rules, and therefore the information would become spread throughout the system rather than grouped in one place.

Another advantage of frame systems over production systems is that they provide a means of separating control information from the data. In production systems, the action is imbedded in the rules. The design of the information being represented has to involve formulating at the same time what actions are to be taken if that information is to be matched. In frames, the design of the control structure can be done independently of the knowledge within the frames themselves. As well, the design of frame systems can be further modularized by using techniques such as matching functions and search trees described in Chapters 2 and 3 to separate out the problems of matching and searching.

The expressive power of frame systems is provided by the rich variety of features available. The matching facilities as described in Chapter 2 provide a comprehensive means of expressing when two situations are similar. Procedural attachment enables the full expressive power of the underlying language such as LISP to be easily incorporated into it. Frames also provide a means of building systems with different structures, such as a hierarchy or a network, so that the system

can be tailored to suit the type of information being described, whereas production systems are limited to a single format.

In-built functions such as FGET, FINHERIT and FNEED provide the basic method of retrieving information from a frame system. In addition, by the use of the FENTER command and different search techniques described in Chapter 4, a simple but powerful query system can easily be designed. In comparison, production systems also provide a powerful means of retrieving information, allowing both forward and backward reasoning, and also being useful for reproducing the actions of human experts in arriving at certain conclusions, for example in the medical consulting system MYCIN, where assumptions made and reasons for any conclusions can be listed.

Production systems provide a much easier means than frames of modifying the knowledge system. This is because the rules are indepèndent of each other with very little interaction between them, and a rule can be deleted or added easily. However, the effect of the addition or deletion on the system as a whole may be more difficult to assess. The interrelations in frames are imbedded within the frames themselves, with the different frame structures providing various problems when the frame system has to be modified. However, by separating out the design of the search, and by following the techniques described in section 4.4, these problems to some extent can be overcome.

Although a production system provides a uniform representation, there is the problem of maintaining consistency or of ensuring that there are no contradictions in the knowledge system. This is because each production rule is defined separately, and individual rules that may contradict already existing rules can be easily added to the system. In frames, it is easier to maintain consistency as all the information related to a certain object is gathered in a single place. However, other aspects such as maintaining references or links in the different frame structures means that there are different degrees of difficulty in maintaining consistency in a frame knowledge base.

The expression of knowledge in frames provides a format which is easy to design and also easy to understand. Production rules also provide a natural means of describing information, although for a complex system, the flow of control may be less transparent to the designer of the system.

Many languages such as FRL, KRL [Bobrow and Winograd, 1979] and KL-ONE [Brachman and Schmolze, 1985] have been devised to implement frames and these languages have been used to apply frame systems to a wide range of problems. Davis and King [1977] state the domains to which production systems are limited. Like production systems, frames would also be inappropiate for expressing mathematical theory, or for representing dependent subprocesses. However, procedural attachment allows control flow to be merged into each frame therefore providing a wider range of domains for which frames may be applied to.

6 FURTHER LINES OF RESEARCH.

The following areas provide useful lines of research for the use of frames:

1. Extensions to frame matching.
2. Research into aspects of representing knowledge.
3. Further application of frames to specific problems.

6.1 Extensions to Frame Matching.

The following possible extensions to the matching process for frames are proposed below :

1. Allow for approximate matches.
2. Extend matching to include AKO information.
3. Use ideas to match the frames and guide the search.
4. Implement a dynamic matching scheme.

6.1.1 Approximate frame matches.

A useful extension to frame matching is to allow for approximate or fuzzy matches to occur where not all of the slots in the frame match. For example, all the information in the frames may match except for one slot. In the Bird Recognition System, this may occur because the user may supply information about a feature of a bird which is incorrect.

There are various problems associated with incorporating an approximate matching scheme into a frame system. One problem is how to describe the result of the match, since the logic values true and false indicate only two possible results whereas an approximate result may occur anywhere between these extremes. Another problem is that the matching process is less efficient as further search is required to find all the frames that may approximately match in a frame system.

Three methods of implementing approximate frame matching are proposed below :

1. The match functions could be used to return a range of values such as true and false (if definitely true and false) and 'maybe', or a value (from 0 to 1, say) that indicates approximately how much weight can be placed on the result. For example, if a particular feature is very distinctive, then a higher weight may be returned by the matching function. For the whole frame to match, then at least one matching function has to return true or an average value has to maintained (of greater than 0.5 say). Alternatively, the FMATCH function could return a value which indicates how 'good' the match was from the values returned by the individual slot matches. During a search of the frame system, this value could then be used to rank the alternatives, with the most likely frame being the one that is considered first.

2. Instead of immediately returning false if one slot does not match, the match could continue until one or more further slots fail to match. The actual number of false slot matches allowed could be specified by a parameter or by a global flag.

3. The matching of the frames could be performed normally, except for storing the name of the relevant slot for those frames that fail to match. If the search of the frame system yields no results, then a further search of the system could be conducted starting from the slot that failed for each frame.

6.1.2 Using ideas to match the frames.

Any situation that may be described by a frame may be thought of as having two types of features - general and specific. The general features or ideas about a real situation are usually the first features that are brought to mind. This feature of human problem solving can be added to the frame matching scheme by attaching 'ideas' to each frame in the frame system. These ideas can simply take the form of a list of names to convey the general concepts being described. For example, in the Bird Recognition System, a list of ideas about the bird can be built up from the description such as the bird is "small sized", is "dark billed" and is a "land bird". The list of ideas can be created automatically when each frame is defined, and can be used whenever the frame is subsequently matched. If the list of ideas in the two frames being matched do not intersect, then a more specific match need not be undertaken.

The matching process can be improved even further by using two lists, a positive and negative idea list, attached to each frame. The negative list would contain the list of ideas that do NOT match, and would also be constructed automatically when a frame is defined. A frame matches if the positive idea lists intersect and the positive and negative lists do not. For example, if a bird is definitely a small sized bird, then the ideas "medium sized" and "large sized" could be placed in the negative list. Hence, a large number of frames would subsequently fail to match the bird. A further advantage of this technique is that in most cases a linear frame structure would be sufficient for search purposes, as most of the irrelevant frames in the frame system would be quickly rejected.

6.1.3 Matching the AKO slot.

Another possible improvement to the matching process is to include the AKO slot in the information that is being matched, similar to the AKO network search. The description of the AKO slot itself could also be improved by allowing logical conditions using AND, OR and NOT to be expressed. This would allow for cases such as the Starling in the Bird Recognition System which is either a medium or small sized bird.

The matching of the AKO slot and the frame itself would be conducted in a similar manner to the network search. However, there would be no need for separate MATCH and REJECT slots because the AKO slot could be used to express the same information. By also allowing the values of the AKO slot to be

evaluated, then a whole range of AKO values may be more easily expressed by using a list or function name.

The consequences of allowing the AKO slot to be expressed by logical relations is that the slot values throughout the rest of the frame also have to be expressed by logical relations. For example, if a Starling is defined as being either a medium or small bird, then the value of the SIZE slot would by default be either small or medium. Problems such as expressing the habitats of birds such as the Harrier (which is not alpine) would also be much easier solved by allowing values in the frames to be expressed by logical conditions. However, as the basis of FRL is to express a facet by a list of values instead of a single set of logical conditions, then the whole format of the language would need to be altered. Further research is required as to whether the alternative frame format is worthwhile or not.

6.1.4 Dynamic matching schemes.

Another area of research into the problem of matching frames, is the usefulness of providing a dynamic as opposed to a static matching process. The matching schemes previously described are primarily static in that the same information is being matched throughout the search of the frame system. In a dynamic matching process, information that is relevant to the match would be initially gained from the frame description being matched and would be subsequently updated as the search continued.

Some possibilities for dynamic frame matching are proposed below :

1. Information in the frames being matched can be added to or updated depending on what was found in the search.

2. Key features or ideas could be immediately 'hashed' to the relevant frames.

3. Various levels of categorisation of the frames into ideas could be conducted with the ideas being analysed becoming more specific during the search. For example, for the Bird Recognition System, the bird observations could be initially categorised into more distinctive ideas in relation to size and appearance. Ideas about less obvious features such as nest location and the colour of the egg can be included as the need arises.

4. More elaborate matching schemes can be designed using $IF-MATCHED functions to perform the automatic updating of ideas and information in the frames.

6.1.5 Using matching to perform the search.

The above extensions to frame matching pose a further question : Is an elaborate matching scheme better than using elaborate structuring techniques? This question is a variation on the procedural versus declarative knowledge debate - is it better to have a relatively simple structure with a sophisticated search algorithm or vice versa? By providing a matching scheme such as using ideas to guide the search, then the need to organise special structures for the search is eliminated. More importantly, the design of the search is separated from the design of the frames. The implementation of the above extensions to the matching of the frames will provide a useful means for comparing the two different techniques.

6.2 Research into Aspects of Knowledge Representation.

Another feature related to the bird and bridge systems is that they each provided different problems in applying the frames. For example, the problem of unknown information is a major problem in the Bird System. The need to specify multiple conditions of slot values occurs only in the Bridge System. Therefore, other forms of knowledge may provide further problems of a different nature. The following questions are raised :

1. Are certain types of information imcompatible with the frame format?
2. Which forms of information are most suited to it?
3. Which forms of information are most suited to other methods of knowledge representation?

4. Can other representations be incorporated into the frame format, or can frames be used to improve other systems? For example, CENTAUR [Aikens, 1983] demonstrates how frames and production rules may be combined together to exploit the best features of both forms of knowledge representation.

6.3 Application of Frames to Specific Problems.

Other specific applications, outlined below, provide further lines of research into the use of frames :

1. Implementation of improved user interfaces such as allowing a bridge expert to design a frame system.

2. Application of frames to other problems in Bridge. Some of the problems are :
   a) Implementing future bidding, such as rebids and responses to earlier bids.
   b) Distinguishing between multiple bids, where one bid is feasible, but another is better. For example, 1 Heart or 2 Hearts.
   c) Quantifying the hand, where a frame is built up of another person's hand from the information available, such as previous bids and the cards played.
   d) Playing the hand.

3. Implementation of software tools and expert systems that use frames.

4. Feasibility of constructing a database that uses the frame approach for retrieval and update as opposed to the network and relational database type systems.

7 SUMMARY.

This thesis has investigated the use of frames to represent knowledge on the computer. Two particular applications of frames, the first to the problem of finding an appropriate opening bid in bridge, and the second to the problem of recognising birds from field observations, were implemented in FRL, a frame representation language, to analyse important features of frames and to find out ways in which they can be improved.

The problems of matching similar pieces of information were discussed in Chapter 1. The existing matching scheme which matches a specific frame against a more generalised frame that contains various requirements for each slot, was used to implement the bridge system. The major limitation of this approach is that the matching process is merged in with the data being matched. To overcome this problem, an alternative matching scheme was proposed which involves the use of matching functions and if-matched demons to separate out the design of the matching process.

In Chapter 3, several frame structures were proposed to allow for different types of search. Four different structures, linear, set, hierarchical and network were analysed and the relative merits of each discussed. The different structures demonstrated the versatility of frames in expressing information in different ways, and offer a wide range from which to choose the best suited method of representation for a particular application. It was also shown how the different structures can be organised so that the design of the search may be separated out from the information contained in the frames.

The problem of interfacing between the frame system and the user was investigated in Chapter 4. It was shown that the existing interface was inadaquate for all types of user except for the programmer. The use of display and enter functions, and interactive matching functions, were proposed for improving the presentation to the users, and for providing a simple query and retrieval system. Aspects of allowing a user to create or modify a frame system were investigated, with the conclusion being that it is easier to design a frame system tailored to a specific purpose for the expert to modify, rather than design a general purpose frame system.

A comparison of frames with production systems and semantic networks was given in Chapter 5. It was stated that frame systems are a more general method of representing knowldege than semantic networks, which does not have extra features such as defaults and procedural attachments. The major advantage of production systems are their modularity, with the information being expressed by the same format, and also being easy to modify. The major advantage of frame systems are their ability to separate the control structure from the information being represented, and also that they enable information about a certain object to be grouped in a single place. It was also suggested that the range of problems to which frame systems may be applied is larger than for production systems because of the ability to attach control information to the frames.

Further areas of research are proposed in Chapter 6. These involve the extension of frame matching, and the application of frames to further problems such as playing bridge and building expert systems. In particular, a different method of search is proposed, whereby instead of searching a knowledge base for a match, information known

may be categorized into certain ideas from which further ideas or frames in the knowledge base are suggested.

APPENDIX A. FRL Commands.

Listed below is a summary of the additional FRL commands described in this thesis. The LISP definitions of each command is shown following the summary. For the standard FRL commands, refer to the FRL Manual.

FDISPLAY <frame>.

Displays the values of FRAME.

FDISPLAY-SLOT <frame> <slot>.

Displays the values of the SLOT using the attached display function. If no function is found, the function PRINT is used.

FENTER <frame> <generic-frame>.

Creates the frame FRAME and enters the values into the new frame using the enter functions defined in the GENERIC-FRAME.

FENTER-SLOT <frame> <slot>.

Enters the SLOT value using the attached enter function. If no function is found, the function READ is used. The local variable :ENTER-VALUES indicates whether a list of values or a single value is to be entered into the SLOT.

FMATCH <frame1> <frame2>.

Matches the frame FRAME1 against the frame FRAME2. Returns T if the two frames match, NIL if they do not match and ? if the result of the match is unknown. FMATCH-PUT is used to store the result of

the match. The local variables :FMATCH and :MATCH contain the results of the frame match and the slot match respectively. The variables :FRAME1, :FRAME2, :SLOT, :VALUES1 and :VALUES2 are bound to the frame and slot values during the matching process.

FMATCHED <frame1> <frame>.

Returns T if FRAME1 has already been matched against FRAME2.

FMATCHES <frame>.

Returns the names of the frames that have been matched against FRAME.

FMATCH-CLEAR <frame>.

Clears out the current match results of FRAME.

FMATCH-GET <frame1> <frame2>.

Gets the result of the match between FRAME1 and FRAME2. If the two frames have not already been matched, then NIL is also returned.

FMATCH-PUT <frame1> <frame2> <result>.

Stores the result of the match between FRAME1 and FRAME2 in the global list :MATCH-RESULTS.

FMATCH-SLOT <frame1> <frame2> <slot>.

Evaluates the match function attached to SLOT between FRAME1 and FRAME2. If an $IF-MATCHED function exists, then this is also evaluated.

FMATCH-SLOTS <frame1> <frame2>.

Matches the slots between FRAME1 and FRAME2. The following values are returned :

t if at least one slot match returns true, and none return false

f if any of the slot matches return false

? if all the slot matches return unknown.

FSEARCH <frame> <frame-list>.

Searches through the frames in FRAME-LIST and returns the names of the frames that match with FRAME. The global variable :MODE indicates whether the search stops at the first match or continues throughout the frame system.

FSEARCH-AKO-NETWORK <frame> <generic-frame>.

Searches the AKO network defined by GENERIC-FRAME and returns the names of the frames that match with FRAME. Uses the global variable :MODE as defined for FSEARCH.

FSEARCH-SET <frame> <frame-list>.

Returns the names of further frames to search after matching FRAME with the set frames specified by the FRAME-LIST. Uses the global variable :MODE as defined for FSEARCH.

FSEARCH-SLOT-NETWORK <frame> <node>.

Searches the slot network with starting node NODE and returns the names of the nodes in the network that match with FRAME. Uses the global variable :MODE as defined for FSEARCH.

FSEARCH-TREE <frame> <tree>.

Returns the names of the terminal nodes of TREE that match with FRAME. Uses the global variable :MODE as defined for FSEARCH.

FVALUES <frame> <slot> <facet>.

Appends and returns the list of values for the FACET of the SLOT of FRAME.

```
(defun fdisplay (frame)
/* Prints the $VALUE slots of FRAME.
    (prog ()
         (print (capital frame))
         (map car cdr
              (lambda (slot)
                   (spaces 5)
                   (print slot)
                   (fdisplay-slot frame slot)
              )
              (ldifference (fslots frame)
                   '(ako instance match reject search)
              )
         )
         (return frame)
    )
)

(defun fdisplay-slot (frame slot)
/* Displays the SLOT $VALUE using the display function in the $DISPLAY
/* facet of the slot. If no function exists, it uses the default
/* function(s) described through the AKO slot.
/* If one does not exist here, then PRINT is used.
    (prog (function value)
         (setq value (fvalues frame slot '$value))
         (setq function (fget frame slot '$display))
         (cond
              ((null (atom function))
                   (return ((caar function) value))
              )
              (t   (return (print value)))
         )
    )
)
```

```
(defun fenter (frame ako-frame)
/* Reads in the $VALUEs as indicated by the AKO-FRAME into FRAME.
    (prog (value :enter-values frame-name)
         (setq frame-name (fcreate frame))
         (fput frame-name 'ako '$value ako-frame)
         (map car cdr
              (lambda (slot)
                   (setq value (fenter-slot frame-name slot))
                   (cond
                        ((member value '(q quit Q QUIT))
                             (return frame-name)
                        )
                   )
              )
              (ldifference (fslots ako-frame)
                   '(ako instance match search reject)
              )
         )
         (return frame-name)
    )
)
(defun fenter-slot (frame slot)
/* Reads in the SLOT $VALUE using the enter function in the $ENTER
/* facet of the slot. If no function exists, it uses the default
/* function(s) described through the AKO slot.
/* If one does not exist here, then READ is used.
/* The local flag :ENTER-VALUES indicates whether the result returned
/* by the enter function to be put into the frame is a list of values
/* (if T) or a single value (if NIL).
    (prog (function value :enter-values)
         (setq function (fget frame slot '$enter))
         (cond
              ((null (atom function))
                   (setq value ((caar function)))
                   (cond
                        ((null value))
                        ((member value '(q quit Q QUIT)))
                        (:enter-values
                             (map car cdr
                                  (lambda (val)
                                       (fput frame slot '$value val)
                                  )
                                  value
                             )
                        )
                        (t   (fput frame slot '$value value))
                   )
                   (return value)
              )
         )
         (setq value (read))
         (cond
              ((null value))
              ((member value '(q quit Q QUIT)))
              (t   (fput frame slot '$value value))
         )
         (return value)
    )
)
```

```
(defun fmatch (:frame1 :frame2)
/* Matches the two frames :FRAME1 and :FRAME2.
/* The function first matches the MATCH slot of :FRAME2.
/* If this matches, then the function tries to match the slots
/* in each frame.
/* At the end of the match, the $IF-MATCHED facet is evaluated
/* if it exists.
    (prog (:fmatch fmatch-function)
          /* Match the two frames :
          (setq :fmatch (fmatch1 :frame1 :frame2))
          /* Evaluate any $IF-MATCHED functions :
          (setq fmatch-function (fget :frame2 'match '$if-matched))
          (cond
               (fmatch-function
                    ((caar fmatch-function) :frame1 :frame2)
               )
          )
          (return :fmatch)
    )
)

(defun fmatch1 (frame1 frame2)
/* Performs the matching of the frames without evaluating the
/* $IF-MATCHED functions.
    (prog (result flag ako1 ako2)
          (setq flag ?)
          (setq ako1 (fheritage frame1 'ako '$value))
          (setq ako2 (fheritage frame2 'ako '$value))
          (cond
               ((or (equal frame1 frame2)
                    (intersection (list (list frame2)) ako1)
                    (intersection (list (list frame1)) ako2))
                    /* Return T if one is the generic frame of the other
                    (return t)
               )
               ((null (intersection
                    (union ako1 (list (list frame1)))
                    (union ako2 (list (list frame2)))))
                    /* Return NIL if the heritages do not overlap
                    (return nil)
               )
               ((fmatched frame1 frame2)
                    (return (fmatch-get frame1 frame2))
               )
          )
          /* Evaluate the MATCH slot.
          (setq result (fmatch-match-slot frame1 frame2))
          (cond
               ((null result)
                    (fmatch-put frame1 frame2 nil)
                    (return nil)
               )
               (t   (setq flag result))
          )
          /* Match the slots in each frame.
          (setq result (fmatch-slots frame1 frame2))
          (cond
               ((unknown result)
                    (setq result flag)
```

```
            )
        )
        (fmatch-put frame1 frame2 result)
        (return result)
    )
)


(defun fmatch-slots (frame1 frame2)
/* Matches the slots in FRAME1 against those in FRAME2.
/* Returns T if at least one slot match returns "true"
/* (non NIL or non ?), and none return NIL.
/* Returns NIL if any of the slots return NIL.
/* Returns ? if all the slot matches return ?.
    (prog (slots1 slots2 result flag)
        (setq slots1
            (ldifference
                (fslots frame1) '(match reject search ako instance)
            )
        )
        (setq slots2
            (ldifference
                (fslots frame2) '(match reject search ako instance)
            )
        )
        (cond
            ((null (intersection slots1 slots2))
                (return ?)
            )
        )
        (map car cdr
            (lambda (:slot :values1 :values2)
                (setq :values1 (fvalues frame1 :slot '$value))
                (setq :values2 (fvalues frame2 :slot '$value))
                (cond
                    ((or (null :values1) (null :values2)))
                    (t
                        (setq result
                            (fmatch-slot frame1 frame2 :slot)
                        )
                        (cond
                            ((null result)
                                (return nil)
                            )
                            ((unknown result))
                            (t   (setq flag t))
                        )
                    )
                )
            )
            slots2
        )
        (cond
            (flag (return t))
            (t    (return ?))
        )
    )
)
```

```
(defun fmatch-slot (frame1 frame2 slot)
/* Matches the SLOT in FRAME1 against the SLOT in FRAME2.
/* If a $IF-MATCHED function exists, then this is also evaluated.
    (prog (:match)
         (setq :match (fmatch-eval frame1 frame2 slot '$match))
         (fmatch-eval frame1 frame2 slot '$if-matched)
         (return :match)
    )
)


(defun fmatch-eval (frame1 frame2 slot facet)
/* Evaluates the FACET function in FRAME2 using the values from
/* both of the frame's SLOTs as arguments.
    (prog (function default argument1 argument2)
         (setq function (fget frame2 slot facet))
         (cond
              ((null function)
                   (return nil)
              )
              (t    (setq function (caar function)))
         )
         (setq argument1 (fvalues frame1 slot '$value))
         (setq argument2 (fvalues frame2 slot '$value))
         (cond
              ((atom function)
                   (eval (list function 'argument1 'argument2))
              )
              (t   (function argument1 argument2))
         )
    )
)

(defun fmatch-match-slot (frame1 frame2)
/* Evaluates the MATCH slot of the frame FRAME2 against FRAME1.
    (prog (match-list)
         (cond
              ((member 'match (fslots frame2))
                   (setq match-list (fget frame2 'match '$value))
              )
         )
         (cond
              (match-list
                 (fmatch-match-list frame1 frame2 (caar match-list))
              )
              (t   ?)
         )
    )
)


(defun fmatch-match-list (frame1 frame2 match-list)
/* Matches the MATCH-LIST containing the logical combinations
/* (OR's, AND's and NOT's) of the frames in the MATCH slot.
    (prog (condition frames result flag)
         (setq condition (car match-list))
         (setq frames (cdr match-list))
         (map car cdr
```

```
                (lambda (frame)
                     (cond
                          ((null (atom frame))
                               (setq result
                                  (fmatch-match-list frame1 frame2 frame)
                               )
                          )
                          (t   (setq result (fmatch frame1 frame)))
                     )
                     (cond
                          ((equal result t)
                               (setq flag t)
                          )
                     )
                     (cond
                          ((equal condition 'not)
                               (cond
                                    ((unknown result)
                                         (return ?)
                                    )
                                    (t   (return (null result)))
                               )
                          )
                          ((and (equal condition 'or) result)
                               (return t)
                          )
                          ((and (equal condition 'and) (null result))
                               (return nil)
                          )
                     )
                )
                frames
           )
           (cond
                (flag (return t))
                (t    (return ?))
           )
      )
)

/* Extra definitions to support FMATCH :
(setq ? '?)
/* ? is the atom equivalent to the logic value UNKNOWN.

(setq :match-results nil)
/* :MATCH-RESULTS is the global list of results of each frame match.

(defun true (s)
/* Returns T if the s-expression S is non-NIL and not UNKNOWN.
    (null (member s '(() ?)))
)


(defun unknown (s)
/* Returns T if the s-expression S is equal to ?.)
    (equal s ?)
)
```

```
(defun fmatched (frame1 frame2)
/* Returns T if FRAME1 has already been matched against FRAME2.
    (member frame2 (get :match-results frame1))
)

(defun fmatch-get (frame1 frame2)
/* Gets the result of the frame match between FRAME1 and FRAME2.
/* If the two frames have not already been matched, then
/* NIL is returned.
    (get (get :match-results frame1) frame2)
)

(defun fmatch-put (frame1 frame2 result)
/* Puts the result of the match between FRAME1 and FRAME2 in the
/* global list :MATCH-RESULTS.
    (cond
         ((null :match-results)
              (setq :match-results (list frame1 (list frame2 result)))
         )
         ((null (get :match-results frame1))
              (put :match-results frame1 (list frame2 result))
         )
         (t   (put (get :match-results frame1) frame2 result))
    )
)


(defun fmatch-clear (frame)
/* Clears out the current match results of FRAME.
    (setq :match-results
         (reverse
              (ldifference
                   :match-results
                   (list frame (get :match-results frame))
              )
         )
    )
)


(defun fmatches (frame)
/* Returns the names of the frames that have been
/* matched against FRAME.
    (map car cddr (lambda (x) x) (get :match-results frame))
)
```

```
(defun fsearch (frame frame-list)
/* Matches the FRAME against all the frames in FRAME-LIST.
/* Returns the name(s) of the matching frames.
/* :MODE indicates whether to return ALL matches or the FIRST match.
    (prog (frames)
          (map car cdr
               (lambda (frame1)
                    (cond
                         ((fmatch frame frame1)
                              (cond
                                   ((equal :mode 'first)
                                        (return (list frame1))
                                   )
                              )
                              (setq frames (union frames (list frame1)))
                         )
                    )
               )
               frame-list
          )
          (return frames)
    )
)
```

```
(defun fsearch-ako-network (frame generic-frame)
/* Matches the FRAME in the against the AKO network specified by the
/* GENERIC-FRAME.
/* Returns the list of nodes in the network that match FRAME if
/* :MODE = ALL, or the first match if :MODE = FIRST.
    (prog (search-list reject-list frames nodes node result)
/*   REJECT-LIST is the list of nodes that have been rejected through
/*   the REJECT slot.
/*   SEARCH-LIST is the set of nodes that are suggested by the matching
/*   of the current NODE (through the INSTANCE slot).
/*   FRAMES is the list of matching frames.
        (fmatch-clear frame)
        (fmatch-put frame generic-frame t)
        (setq nodes (fvalues generic-frame 'instance '$value))
        (cond
            ((null nodes) (return nil))
            ((atom nodes) (setq nodes (list nodes)))
        )
        (setq search-list nodes)
        loop
        (setq node (car search-list))
        (setq result (fmatch-node frame node))
        (cond
            ((true result)
                (cond
                    ((equal :mode 'first)
                        (return (list node))
                    )
                )
                (setq frames (union frames (list node)))
                (setq reject-list
                    (union reject-list (fvalues node 'reject))
                )
            )
        )
/*      Update the SEARCH-LIST from the INSTANCE and REJECT slots.
        (setq search-list (reverse
            (ldifference
                (union search-list (fvalues node 'instance '$value))
                (union reject-list (fmatches frame))
            )
        ))
        (cond
            ((null search-list))
            (t   (go loop))
        )
        (return frames)
    )
)

(defun fmatch-node (frame node)
/* Returns T if the AKO nodes for NODE all match, and the FRAME matches
/* with NODE.
    (prog (result)
        (cond
            ((fmatched frame node)
                (return (fmatch-get frame node))
            )
        )
```

```
        (setq result (fmatch1-node frame node))
        (fmatch-put frame node result)
        (return result)
    )
)


(defun fmatch1-node (frame node)
/* Same as FMATCH-NODE except that :MATCH-RESULTS are not updated.
    (prog (result ako-function akos)
        /* Match the AKO frames :
        (setq akos (fvalues node 'ako '$value))
        (cond
            ((null akos))
            ((atom akos) (setq akos (list akos)))
        )
        (map car cdr
            (lambda (ako-frame)
                (cond
                    ((member ako-frame reject-list)
                        (return nil)
                    )
                )
                (setq result (fmatch-node frame ako-frame))
                (cond
                    ((null result)
                        (return nil)
                    )
                    ((and (true result) (equal :mode 'first)
                          (member ako-frame nodes))
                        (return result)
                    )
                )
            )
            akos
        )
        (setq ako-function (fget node 'ako '$if-matched))
        (cond
            (ako-function
                ((caar ako-function) frame node)
            )
        )
        (return (fmatch frame node))
    )
)
```

```
(defun fsearch-set (frame frames)
/* Returns the set of node-names specified by the INSTANCE slot sets
/* and the REJECT slot sets in those FRAMES that match with FRAME.
    (prog (matching-nodes rejected-nodes)
          (fmatch-clear frame)
          (map car cdr
               (lambda (frame1 search-list reject-list)
/*                  check to see if the slot value matched and add the
/*                  node lists if it does
                    (cond
                         ((member frame1 rejected-nodes))
                         ((true (fmatch frame frame1))
                              (cond
                                   ((member 'instance (fslots frame1))
                                        (setq search-list (fvalues
                                             frame1 'instance '$value)
                                        )
                                   )
                              )
                              (setq reject-list
                                   (fvalues frame1 'reject '$value)
                              )
                              (setq matching-nodes
                                   (union matching-nodes search-list)
                              )
                              (setq rejected-nodes
                                   (union rejected-nodes reject-list)
                              )
                         )
                    )
               )
               frames
          )
          (return
               (ldifference matching-nodes rejected-nodes)
          )
    )
)
```

```
(defun fsearch-slot-network (frame node)
/* Matches the FRAME against the slot network at NODE.
/* If the match fails, then it successively tries to match
/* the other nodes in the network.
/* :MODE indicates whether to return ALL matches or the FIRST match.
    (prog (result-list search-list result match-result flag)
         (setq flag ?)
         (cond
              ((fmatched frame node)
                   (return nil)
              )
         )
         (map car cdr
              (lambda (slot)
                   (setq match-result (fmatch-slot frame node slot))
                   (cond
                        ((true match-result)
                             (setq flag t)
                        )
                        ((unknown match-result))
                        (t
                           /* search the nodes described in the $SEARCH
                           /* facet if the slot does not match
                           (fmatch-put frame node nil)
                           (setq search-list (fget node slot '$search))
                           (map caar cdr
                              (lambda (node)
                                 (setq result
                                    (fsearch-slot-network frame node)
                                 )
                                 (cond
                                    ((and result (equal :mode 'first))
                                       (return result)
                                    )
                                    (result
                                       (setq result-list
                                          (union result result-list)
                                       )
                                    )
                                 )
                              )
                              search-list
                           )
                           (cond
                              ((equal :mode 'first)
                                 (return nil)
                              )
                              (t   (return result-list))
                           )
                        )
                   )
              )
              (reverse (fslots node))
         )
         (fmatch-put frame node flag)
         (return (list node))
    )
)
```

```
(defun fsearch-tree (frame tree)
/* Returns the names of the terminal nodes of TREE (nodes that have no
/* instantiation) that match with the FRAME;
/* Returns NIL if there is no match.
     (fmatch-clear frame)
     (fsearch1-tree frame tree)
)

(defun fsearch1-tree (frame tree)
     (prog (frames sub-nodes result)
          /* find the list of sub-nodes belonging to this tree
          (cond
               ((member 'instance (fslots tree))
                    (setq sub-nodes
                         (fget tree 'instance '$value)
                    )
               )
               (t   (setq sub-nodes nil))
          )
          (cond
               ((fmatched frame tree)
                    (setq result (fmatch-get frame tree))
               )
               (t   (setq result (fmatch frame tree)))
          )
          /* If there aren't any identifying slots in the node,
          /* or the match is non NIL,
          /* then the sub-nodes have to be searched.
          (cond
               ((and (ldifference (fslots tree)
                       '(reject ako instance search))
                     (null result))
                    (return nil)
               )
          )
          (cond
               ((null sub-nodes)
                    (return (list tree))
               )
               (t
                    (map caar cdr
                         (lambda (node)
                             (setq result (fsearch1-tree frame node))
                            (cond
                                 ((and result (equal :mode 'first))
                                      (return result)
                                 )
                                 (t   (setq frames
                                         (union frames result))
                                 )
                             )
                         )
                         sub-nodes
                    )
                    (return frames)
               )
          )
     )
)
```

```
(defun fvalues (frame slot facet)
/* Returns the values of the FACET of the SLOT of FRAME appended
/* together in a list if there are more than one value;
/* otherwise returns the value by itself.
    (prog (values)
         (setq values (fget frame slot facet))
         (cond
              ((equal (length values) 1)
                   (caar values)
              )
              (values
                   (map caar cdr (lambda (x) x) values)
              )
         )
    )
)
```

APPENDIX B. Frame Definitions.

Selected frame definitions for each different frame system are listed below for both the Bridge System and the Bird Recognition System. Sample attached functions are also listed.

B-1 Bridge System.

B-1-1 Definition of HAND.

```
(deframe hand
    (cards
         ($enter
              (enter-hand))
         ($display
              (display-cards)))
    ("honour points"
         ($enter
              (find-honour-points))
         ($display
              (display-honour)))
    ("division of suits"
         ($enter
              (find-suit-division))
         ($display
              (display-division)))
    ("suit strength"
         ($enter
              (find-suit-strength))
         ($display
              (display-strength)))
    ("playing tricks"
         ($enter
              (find-playing-tricks))
         ($display
              (display-value)))
    ("quick tricks"
         ($enter
              (find-quick-tricks))
         ($display
              (display-value)))
    (balance
         ($enter
              (find-balance))
         ($display
              (display-value)))
)
```

B-1-2 Sample Attached Functions.

REQUIRE functions :

```
(defun equal-suits ()
/* Returns T if the two longest suits in HAND are of equal length.
     (prog (hand-divisions)
          (setq hand-divisions
               (caar
                    (fget :frame '"division of suits" '$value)
               )
          )
          (cond
               ((equal
                    (caar hand-divisions)
                    (car (cadr hand-divisions))
               ))
          )
     )
)

(ndefun points number
/* Returns T if the honour points in the HAND matches the NUMBER.
    (match-number (car number)
         (caaar
             (fget :frame '"honour points" '$value)
         )
    )
)

(ndefun division divisions
/* Returns T if DIVISIONS matches against the division of suits
/* in HAND.
     (setq divisions (car divisions))
     (prog (hand-divisions)
          (setq hand-divisions
               (caar
                    (fget :frame '"division of suits" '$value)
               )
          )
          (map car cdr
               (lambda (number)
                  (cond
                     ((match-number number (caar hand-divisions))
                        (setq hand-divisions (cdr hand-divisions))
                     )
                     (t
                        (return nil)
                     )
                  )
               )
               divisions
          )
          (return t)
     )
)
```

Enter functions :

```
(defun enter-hand ()
/* Returns the global list CARDS entered by the user.
     (prog (spades hearts clubs diamonds count)
          loop
          (setq spades (list 'spades))
          (setq hearts (list 'hearts))
          (setq diamonds (list 'diamonds))
          (setq clubs (list 'clubs))
          (enter-suit spades)
          (enter-suit hearts)
          (enter-suit diamonds)
          (enter-suit clubs)
          (setq cards
               (list spades hearts diamonds clubs)
          )
          (setq count 0)
          (map car cdr
            (lambda (suit) (setq count (plus count (length suit))))
            cards
          )
          (cond
               ((equal count 17))
               (t (prin1 (minus count 4))
                  (prin1 '" cards were entered - please re-enter ")
                  (print '"the hand.")
                  (go loop)
               )
          )
          (return cards)
     )
)
(defun find-balance ()
/* Returns WELL BALANCED, BALANCED or UNBALANCED depending on the
/* CARDS.
     (prog (count)
          (setq count 0)
          (map car cdr
               (lambda (suit size)
                    (setq size (length suit))
                    (cond
                         ((lessp size 3)
                              (return 'unbalanced)
                         )
                         ((equal size 3)
                              (setq count (plus count 1))
                         )
                    )
               )
               cards
           )
           (cond
                ((equal count 0) (return '"well balanced"))
                ((lessp count 2) (return 'balanced))
                (t               (return 'unbalanced))
           )
     )
)
```

Display functions :

```
(defun display-strength (bids)
/* Prints out the strength of the bids.
    (prog (rebid bid flag)
         (spaces 10)
         (setq rebid (cdar bids))
         (setq bid (cdadr bids))
         (cond
              ((null rebid))
              (t
                   (setq flag t)
                   (print-words rebid)
                   (prin1 'rebiddible)
              )
         )
         (cond
              ((null bid)
                   (cond
                      ((null flag)
                        (prin1 '"No rebiddible or biddible suits")
                      )
                   )
              )
              (t
                   (cond
                        (flag (prin1 '"; "))
                   )
                   (print-words bid)
                   (prin1 'biddible)
              )
         )
         (print nullchar)
    )
)

(defun display-division (suits)
/* Prints out the division of the SUITS.
    (prog (flag)
         (spaces 10)
         (map car cdr
              (lambda (suit)
                   (cond
                        (flag (prin1 '", "))
                        (t    (setq flag t))
                   )
                   (prin1 (car suit))
                   (spaces 1)
                   (prin1 (cadr suit))
              )
              suits
         )
         (print nullchar)
    )
)
```

Bid functions :

```
(defun long-suit-bid (bid)
/* Returns a BID of the longest suit in HAND.
     (list bid
          (cadar
               (caar
                    (fget :frame '"division of suits" '$value)
               )
          )
     )
)

(defun two-long-suits-bid ()
/* Returns a bid of 1 of : longer suit first unless hand is weak and
/* shorter adjacent suit ranks higher in HAND.
     (prog (long-suit short-suit divisions)
          (setq divisions
               (caar
                    (fget :frame '"division of suits" '$value)
               )
          )
          (setq long-suit (cadar divisions))
          (setq short-suit (car (cdadr divisions)))
          (cond
               ((and (weak :frame)
                     (adjacent (list long-suit short-suit))
                     (greaterp short-suit long-suit)
                )
                    (return (list 1 short-suit))
               )
               (t   (return (list 1 long-suit)))
          )
     )
)

(defun prepared-opening-bid ()
/* Returns 1 clubs or 1 diamonds as a prepared opening bid for HAND.
     (cond
          ((greaterp
              (cadr
                 (assoc 'clubs
                    (cdaar (fget :frame '"honour points" '$value))
                 )
              )
              1
           )
               '(1 clubs)
          )
          (t   '(1 diamonds))
     )
)
```

B-1-3 Sample Linear Frames.

```
(deframe opening-frame
     (instance
          ($value
               (two-no-trumps-frame)
               (two-clubs-1-frame)
               (two-clubs-2-frame)
               (strong-two-1-frame)
               (strong-two-2-frame)
               (equal-suits-frame)
               (4-4-4-1-frame)
               (4-4-frame)
               (6-4-frame)
               (unequal-suits-frame)
               (1-of-suit-frame)
               (one-no-trumps-weak-frame)
               (prepared-frame)
               (light-opening-frame)
               (pre-emptive-frame)
          )
     )
)

(deframe unequal-suits-frame
     ("division of suits"
          ($require
               ((and (null (equal-suits))
                     (division ((above 4) (above 3)))))))
     ("honour points"
          ($require
               ((points (above 8)))))
     (bid
          ($if-needed
               (two-long-suits-bid)))
)

(deframe two-no-trumps-frame
     ("honour points"
          ($require
               ((no-trumps (from 20 22.5)))))
     (balance
          ($require
               ((balanced-hand))))
     (bid
          ($if-needed
               ( '(2 no-trumps))))
)
```

B-1-4 Sample Set Frames.

```
(deframe opening-set
    (instance
        ($value
            (high-points-set)
            (average-points-set)
            (low-points-set)
            (equal-suits-set)
            (two-long-suits-set)
            (one-long-suit-set)
            (no-trumps-set)
            (balanced-set)
        )
    )
)

(deframe high-points-set
    ("honour points"
        ($require
            ((or (points (above 19))
                 (q-tricks (above 4))))))
    (instance
        ($value
            (two-no-trumps-frame) (two-clubs-1-frame)
            (strong-two-2-frame) (two-clubs-2-frame)))
    (reject
        ($value
            (prepared-frame) (unequal-suits-frame) (6-4-frame)
            (1-of-suit-frame) (one-no-trumps-weak-frame)
            (4-4-frame) (4-4-4-1-frame) (equal-suits-frame)
            (pre-emptive-frame)))
    )
)

(deframe two-long-suits-set
    ("division of suits"
        ($require
            ((division ((above 4) (above 3))))))
    (instance
        ($value
            (unequal-suits-frame) (strong-two-2-frame)
            (strong-two-1-frame) (6-4-frame)))
    (reject
        ($value
            (4-4-frame) (4-4-4-1-frame) (prepared-frame)
            (two-no-trumps-frame) (1-of-suit-frame)
            (one-no-trumps-weak-frame) (pre-emptive-frame)))
    )
)
```

B-1-5 Sample Hierarchical Frames.

```
/*  level 1  */
(deframe opening-tree
    (instance
         ($value
              (1-of-suit-tree)
              (other-bid-tree)
         )
    )
)

/*  level 2  */
(deframe 1-of-suit-tree
    (ako
         ($value
              (opening-bid-tree)))
    (instance
         ($value
              (equal-suits-tree)
              (unequal-suits-tree)
         )
    )
    ("honour points"
         ($require
              ((points (from 8 19)))))
    ("division of suits"
         ($require
              ((or  (division ((above 4)))
                    (biddible-suit)
              ))))
)

/*  level 3  */
(deframe equal-suits-tree
    (ako
         ($value
              (1-of-suit-tree)))
    (instance
         ($value
              (4-4-suits-tree)
              (other-equal-suits-tree)
         )
    )
    ("division of suits"
         ($require
              ((equal-suits))))
)

/*  level 4  */
(deframe other-equal-suits-tree
    (ako
         ($value
              (equal-suits-tree)))
    (instance
         ($value
              (equal-long-suits-tree)
              (low-points-equal-suits-tree)
         )
```

```
    )
)

/*  level 5  */
(deframe equal-long-suits-tree
    (ako
         ($value
              (other-equal-suits-tree)))
    ("division of suits"
         ($require
              ((division ((above 4))))))
    (bid
         ($if-needed
              (equal-suits-bid)))
)
```

B-1-6 Sample Search Tree Frames.

```
(deframe opening-hands
    (instance
         ($value
              (no-bid-hand)
              (strong-hands)
              (no-trumps-hands)
              (one-hands)
         )
    )
)

(deframe no-bid-hand
    ("honour points"
         ($require
              ((points (below 9)))))
    (bid
         ($if-needed
              ( '() )))
)

(deframe strong-hands
    (instance
         ($value
              (two-clubs-hand)
              (two-no-trumps-frame)
              (strong-hand)
         )
    )
    (points
         ($require
              ((or (points (above 17))
                   (p-tricks (above 7))
                   (q-tricks (above 4))
              ))))
)

(deframe two-clubs-hand
    (points
         ($require
              ((or (points (above 22))
                   (q-tricks (above 4))))))
    (bid
         ($if-needed
              ( '(2 clubs) )))
)
```

B-1-7 Sample Slot Network Frames.

```
(deframe opening-node
    ("honour points"
         ($require
              ((points (below 9))))
         ($search
              (1-of-suit-node) (equal-suits-node)
              (strong-two-2-node) (strong-two-1-node)
              (one-no-trumps-weak-node) (two-clubs-2-node)
              (pre-emptive-node))
    )
    ("playing tricks"
         ($require
              ((p-tricks (below 8))))
         ($search
              (strong-two-1-node)))
    (bid
         ($if-needed
              ( '() )))
)

(deframe 1-of-suit-node
    ("suit strength"
         ($require
              ((biddible-suit)))
         ($search
              (prepared-node) (one-no-trumps-weak-node)
              (two-no-trumps-node))
    )
    ("honour points"
         ($require
              ((points (from 12 19))))
         ($search
              (two-no-trumps-node) (light-opening-node)
              (two-clubs-2-node))
    )
    (bid
         ($if-needed
              (light-opening-bid)))
)
```

B-1-8 Sample AKO Network Frames.

```
(deframe opening_node
     (instance
          ($value
               (no_bid_node) (strong_two_2_node)
               (two_no_trumps_node) (equal_suits_node)
               (unequal_suits_node) (two_clubs_1_node)
               (two_clubs_2_node) (strong_two_1_node) (4_4_4_1_node)
               (4_4_suits_node) (6_4_node) (1_of_suit_node)
               (one_no_trumps_weak_node) (prepared_node)
               (light_opening_node) (pre_emptive_node)
          )
     )
)

(deframe strong_node
    (ako
          ($value
               (opening_node)
          )
    )
    (reject
          ($value
               (light_opening_node) (1_of_suit_node) (4_4_suits_node)
               (4_4_4_1_node) (equal_suits_node) (unequal_suits_node)
               (6_4_node) (one_no_trumps_weak_node) (prepared_node)
               (pre_emptive_node) (no_bid_node) (4_4_node)
          )
    )
    (points
          ($require
               ((or (points (above 17))
                    (p-tricks (above 7))
                    (q-tricks (above 4))))))
)

(deframe strong_two_2_node
     (ako
          ($value
               (strong_node)
               (long_suit_node)
               (opening_node)
          )
     )
     ("honour points"
          ($require
               ((points (above 17)))))
     ("division of suits"
          ($require
               ((division ((above 4))))))
     (bid
          ($if-needed
               ((long-suit-bid 2))))
)
```

B-2 Bird Recognition System.

B-2-1 Definition of BIRD.

```
(deframe bird
    (ako
         ($if-matched
              (display-akos)))
    (instance
         ($value
              ("new zealand dabchick") ("white faced heron")
              ("bittern") ("black swan") ("mute swan")
              ("canada goose") ("paradise duck") ("mallard")
              ("grey duck") ("new zealand scaup") ("grey teal")
              ("new zealand shoveler") ("harrier")
              ("californian quail") ("brown quail") ("pheasant")
              ("banded rail") ("pukeko") ("australian coot")
              ("spur winged plover") ("rock pigeon") ("little owl")
              ("kingfisher") ("skylark") ("new zealand pipit")
              ("welcome swallow") ("hedge sparrow") ("fernbird")
              ("grey warbler") ("fantail") ("song thrush")
              ("blackbird") ("silvereye") ("yellowhammer")
              ("cirl bunting") ("chaffinch") ("greenfinch")
              ("goldfinch") ("redpoll") ("house sparrow")
              ("starling") ("indian myna") ("black backed magpie")
              ("white backed magpie") ("rook")
         )
    )
    (match
         ($if-matched
              (display-birds)))
    (size
         ($match
              (estimate))
         ($if-matched
              (show-estimate))
         ($enter
              (ask-size))
         ($display
              (print-list)))
    (appearance
         ($match
              (appearance))
         ($if-matched
              (show-appearance))
         ($enter
              (ask-noun-description))
         ($display
              (print-appearance)))
    (distribution
         ($match
              (district))
         ($if-matched
              (show-district))
         ($enter
              (ask-set))
         ($display
              (print-names)))
    (habitat
```

```
     ($match
          (same-nouns))
     ($if-matched
          (show-habitat))
     ($enter
          (ask-set))
     ($display
          (print-habitat)))
(behaviour
     ($match
          (noun-description))
     ($if-matched
          (show-behaviour))
     ($enter
          (ask-noun-description))
     ($display
          (print-noun-description)))
(flight
     ($match
          (noun-description))
     ($if-matched
          (show-flight))
     ($enter
          (ask-noun-description))
     ($display
          (print-noun-description)))
(food
     ($match
          (same-nouns))
     ($if-matched
          (show-food))
     ($enter
          (ask-set))
     ($display
          (print-words)))
("nest material"
     ($match
          (same-nouns))
     ($if-matched
          (show-nest))
     ($enter
          (ask-material))
     ($display
          (print-words)))
("nest location"
     ($match
          (location))
     ($if-matched
          (show-location))
     ($enter
          (ask-location))
     ($display
          (print-phrase-description)))
(breeding
     ($match
          (season))
     ($if-matched
          (show-season))
     ($enter
```

```
                (ask-season))
           ($display
                (print-season)))
      ("number of eggs"
           ($match
                (number))
           ($if-matched
                (show-number))
           ($enter
                (ask-eggs))
           ($display
                (print-eggs)))
      ("egg colour"
           ($match
                (noun-description))
           ($if-matched
                (show-colour))
           ($enter
                (ask-colour))
           ($display
                (print-noun-description)))
)
```

B-2-2 Sample Attached Functions.

Match functions :

```
(defun number (number range)
/* Returns T if NUMBER falls in the RANGE.
    (cond
         ((atom range)
              (equal number range)
         )
         ((lessp (length range) 3)
              (cond
                   ((equal (car range) 'above)
                        (greaterp number (cadr range))
                   )
                   ((equal (car range) 'below)
                        (lessp number (cadr range))
                   )
                   (t   (equal number (car range)))
              )
         )
         ((and (greaterp number (minus (car range) 1))
              (lessp number (plus (caddr range) 1)))
         )
    )
)

(defun same-nouns (noun1 noun2)
/* Returns non NIL if the nouns NOUN1 intersect with
/* the nouns NOUN2.
    (cond
        ((atom noun1) (setq noun1 (list noun1)))
    )
    (cond
        ((atom noun2) (setq noun2 (list noun2)))
    )
    (intersection noun1 noun2)
)
```

IF-MATCHED functions :

```
(defun show-number (number range)
/* Prints out a message if the NUMBER of eggs is matched
/* against RANGE.
    (prog (which)
         (cond
            ((true :match)
               (prin1 '"The ")
               (print-name :frame2)
               (prin1 '" usually has ")
               (print-eggs range)
               (prin1 '" eggs in its nest which matches with the ")
               (prin1 number)
               (print '" eggs that were found.")
            )
            ((null :match)
               (cond
                    ((atom range) (setq range (list range)))
               )
               (cond
                    ((lessp number (car range))
                         (setq which 'few)
                    )
                    (t   (setq which 'many))
               )
               (prin1 '"There are too ")
               (prin1 which)
               (prin1 '" eggs in the nest for the bird to be the ")
               (print-name :frame2)
               (print '.)
            )
         )
    )
)

(defun show-district (district1 district2)
/* Prints a message if DISTRICT is matched.
  (prog ()
     (cond
        ((true :match)
          (prin1 '"The ")
          (print-name :frame2)
          (prin1 '" is usually found in ")
          (print-names district2)
          (print '" which matches with where the bird was found.")
        )
        ((null :match)
          (prin1 '"The bird cannot be the ")
          (print-name :frame2)
          (prin1 '" which is usually found in ")
          (print-names district2)
          (print '.)
        )
     )
  )
)
```

Enter functions :

```
(defun ask-set ()
/* Asks and returns the WORDS that best describes the feature.
    (prog (words choices slot-help)
         (setq choices (caar (fget 'bird-words slot '$value)))
         loop
         (prin1 '"Enter the words that best describe the ")
         (prin1 '"bird's ")
         (prin1 slot)
         (print '" :")
         (cond
              ((equal slot 'distribution)
                   (setq words (read-names))
              )
              (t   (setq words (read-list)))
         )
         (cond
              ((member words '(q quit))
                   (return 'quit)
              )
              ((equal words 'help)
                   (cond
                        ((equal slot '"nest material")
                             (feature 'material)
                        )
                        (t   (feature slot))
                   )
                   (cond
                      ((equal slot 'distribution))
                      (t
                        (print '"The following are valid words : ")
                        (display-set choices)
                      )
                   )
                   (go loop)
              )
              ((ldifference words choices)
                   (prin1 clear-screen)
                   (prin1 '"The following are invalid words : ")
                   (print-list (ldifference words choices))
                   (print '" - please reenter or ask for help.")
                   (print nullchar)
                   (go loop)
              )
         )
         (setq words (intersection words choices))
         (setq :enter-values t) /* Enter more than one value
         (return words)
    )
)
```

Display functions :

```
(defun print-names (words)
/* Prints out the list of WORDS.
    (prog (flag)
          (setq flag nil)
          (cond
               ((atom words)
                    (print-name words)
                    (print nullchar)
                    (return)
               )
          )
          (map car cdr
               (lambda (word)
                    (cond
                         (flag     (prin1 '", "))
                         (t        (setq flag t))
                    )
                    (print-name word)
               )
               words
          )
          (print nullchar)
    )
)

(defun print-eggs (eggs)
/* Prints out the number of EGGS.
    (cond
          ((atom eggs)
               (prin1 eggs)
               (print nullchar)
          )
          ((lessp (length eggs) 3)
               (prin1 (car eggs))
               (print nullchar)
          )
          ((equal (car eggs) (caddr eggs))
               (prin1 (car eggs))
               (print nullchar)
          )
          (t   (print-list eggs))
    )
)
```

B-2-3 Sample Linear Frames.

```
(deframe "mallard"
    (ako
         ($value
              ("bird")))
    (size
         ($value
              (58)))
    (appearance
         ($value
              ("mallard appearance")))
    (distribution
         ($value
              ("new zealand")))
    (habitat
         ($value
              (river) (lake) (pond) (marsh) (lagoon)))
    (flight
         ($value
              ((flight (fast wheeling) take-off (sudden)
                sides (pale) tail (pale)))))
    (food
         ($value
              (plants) (insects)))
    (breeding
         ($value
              ((august to september))))
    ("nest material"
         ($value
              (grass) (vegetation) (down)))
    ("nest location"
         ($value
              ((near (water) in (vegetation)))))
    ("number of eggs"
         ($value
              ((5 to 20))))
    ("egg colour"
         ($value
              ((mainly (cream light)))))
)
```

```
(deframe "mallard appearance"
    ("male breeding plumage"
         ($value
              ((head (green) breast (purple brown) rump (black)
                bill (olive yellow green) body (grey brown)
                feet (orange) underparts (grey)))))
    ("male eclipse plumage"
         ($value
              ((mainly (brown) body (brown) feet (orange)
                bill (olive green) rump (brown black)
                crown (dark brown) upperparts (grey)
                speculum (shining blue white)))))
    ("female and immature"
         ($value
              ((mainly (brown) body (mottled brown)
                feet (orange) bill (brown orange)
                speculum (shining blue white)))))
)
```

B-2-4 Sample Set Frames.

```
(deframe "bird sets"
    (instance
         ($value
              ("small sized bird") ("medium sized bird")
              ("large sized bird")
              ("colourful bird")
              ("colourful billed bird") ("dark billed bird")
              ("dark feet bird") ("light feet bird")
              ("dark undersides bird") ("light undersides bird")
              ("North Island bird") ("South Island bird")
              ("water bird") ("land bird") ("land and water bird")
              ("animal eating bird") ("plant eating bird")
              ("omnivorous bird")
              ("slow flight bird") ("swift flight bird")
              ("grassy nest bird") ("branchy nest bird")
              ("tree nest bird") ("water nest bird")
              ("February to July breeding bird")
              ("many eggs bird") ("few eggs bird")
              ("coloured eggs bird") ("creamy eggs bird")
         )
    )
)

(deframe "large sized bird"
    (ako
         ($value
              (bird)))
    (instance
         ($value
              ("white faced heron") ("bittern") ("black swan")
              ("mute swan") ("canada goose") ("paradise duck")
              ("mallard") ("grey duck") ("harrier") ("pheasant")
              ("pukeko") ("australian coot")))
    (size
         ($value
              (large))
         ($match
              (equal))
         ($if-matched
              (show-set)))
    (reject
         ($value
              ("new zealand dabchick") ("new zealand scaup")
              ("grey teal") ("new zealand shoveler")
              ("californian quail") ("brown quail")
              ("banded rail") ("spur winged plover")
              ("rock pigeon") ("little owl") ("kingfisher")
              ("skylark") ("new zealand pipit")
              ("welcome swallow") ("hedge sparrow")
              ("fernbird") ("grey warbler")
              ("fantail") ("song thrush") ("blackbird")
              ("silvereye") ("yellowhammer")
              ("cirl bunting") ("chaffinch") ("greenfinch")
              ("goldfinch") ("redpoll") ("house sparrow")
              ("starling") ("indian myna") ("black backed magpie")
              ("white backed magpie") ("rook")))
)
```

```
(deframe "slow flight bird"
    (ako
         ($value
              (bird)))
    (flight
         ($value
              ((flight (slow laboured))))
         ($if-matched
              (show-set)))
    (reject
         ($value
              ("swift flight bird")
              ("canada goose") ("grey teal")
              ("new zealand shoveler")
              ("mallard") ("pukeko") ("skylark")
              ("welcome swallow") ("rook")))
)
```

B-2-5 Sample Search Tree Frames.

```
/* level 1
(deframe "bird tree"
    (ako
         ($value
              (bird)))
    (instance
         ($value
              ("small bird")
              ("big bird")
         )
    )
    (size
         ($match
              (equal))
         ($if-matched
              ((lambda (x y)))))
/* Do nothing if the slot is matched.
    (appearance
         ($match
              (any-noun))
         ($if-matched
              ((lambda (x y)))))
    (habitat
         ($if-matched
              ((lambda (x y)))))
    (match
         ($if-matched
              (display-birds)))
)

/* level 2
(deframe "small bird"
    (ako
         ($value
              ("bird tree")))
    (instance
         ($value
              ("small land bird")
              ("other small bird")
         )
    )
    (size
         ($value
              (small)))
)

/* level 3
(deframe "other small bird"
    (ako
         ($value
              ("small bird")))
    (instance
         ($value
              ("small white bird")
              ("not small white bird")
         )
    )
```

```
)

/* level 4
(deframe "small white bird"
    (ako
         ($value
              ("other small bird")))
    (instance
         ($value
              ("red faced bird")
              ("not red faced bird")
         )
    )
    (appearance
         ($value
              ((mainly (white) body (white) underparts (white)
                upperparts (white)))))
)

(deframe "not small white bird"
    (ako
         ($value
              ("other small bird")))
    (instance
         ($value
              ("brown quail")
              ("fantail")
              ("blackbird")
              ("starling")
         )
    )
    (match
         ($value
              ((not "small white bird"))))
)
```

B-2-6 Sample AKO Network Frames.

```
(deframe "mallard"
    (ako
         ($value
              ("large sized bird")
              ("water bird")
              ("colourful billed bird")
              ("light feet bird")
              ("omnivorous bird")
              ("swift flight bird")
              ("grassy nest bird")
              ("water nest bird")
              ("creamy eggs bird")
              ("bird")
         )
    )
    (size
         ($value
              (58)))
    (appearance
         ($value
              ("mallard appearance")))
    (distribution
         ($value
              ("new zealand")))
    (habitat
         ($value
              (river) (lake) (pond) (marsh) (lagoon)))
    (flight
         ($value
              ((flight (fast wheeling) take-off (sudden)
                sides (pale) tail (pale)))))
    (food
         ($value
              (plants) (insects)))
    (breeding
         ($value
              ((august to september))))
    ("nest material"
         ($value
              (grass) (vegetation) (down)))
    ("nest location"
         ($value
              ((near (water) in (vegetation)))))
    ("number of eggs"
         ($value
              ((5 to 20))))
    ("egg colour"
         ($value
              ((mainly (cream light)))))
)
```

APPENDIX C. Trace of Bridge System.

C-1 Presentation to the user.

Internal representation :

```
(hand1
     (ako
          ($value
               (hand)))
     (cards
          ($value
               (((spades A K J 10 9 8) (hearts A Q J 10) (diamonds 3 2)
                 (clubs A))))))
     (honour-points
          ($value
               (((19 (no-trumps 2.0000000000000E+01) (spades 8)
                  (hearts 7) (diamonds 0) (clubs 4))))))
     (division-of-suits
          ($value
               (((6 spades) (4 hearts) (2 diamonds) (1 clubs))))))
     (suit-strength
          ($value
               (((rebiddible spades) (biddible hearts))))))
     (playing-tricks
          ($value
               (8)))
     (quick-tricks
          ($value
               (4)))
     (balance
          ($value
               (unbalanced)))
)
```

Displaying a bridge hand :

```
Hand1
     balance
          unbalanced
     quick tricks
          4
     playing tricks
          8
     suit strength
          spades is rebiddible; hearts is biddible
     division of suits
          6 spades, 4 hearts, 2 diamonds, 1 clubs
     honour points
          19, (20 for no trumps)
          spades = 8, hearts = 7, diamonds = 0, clubs = 4
     cards
          spades A K J 10 9 8, hearts A Q J 10, diamonds 3 2, clubs A
```

Entering a bridge hand :

```
enter spades : a kj 10 j 9 8
Invalid card or multiple cards entered.
Please re-enter the cards in the suit.
enter spades : a k j 10 9 8
enter hearts : q j a 10
enter diamonds : 2 3
enter clubs : a
```

C-2 Sample Bridge Hands.

Hand1

balance
unbalanced
quick tricks
4
playing tricks
8
suit strength
spades is rebiddible; clubs is biddible
division of suits
6 spades, 4 clubs, 3 hearts, 0 diamonds
honour points
20, (20 for no trumps)
spades = 10, hearts = 5, diamonds = 0, clubs = 5
cards
spades A K Q J 9 8, hearts K Q 9, clubs K Q 9 8

1. Linear Search :

1 of suit frame	: 1 spades
unequal suits frame	: 1 spades
6 4 frame	: 1 spades
strong two 2 frame	: 2 spades
strong two 1 frame	: 2 spades

2. Set Search :

strong two 1 frame	: 2 spades
strong two 2 frame	: 2 spades

3. Hierarchical Search :

strong two 2 tree	: 2 spades
strong two 1 tree	: 2 spades

4. Search Tree Search :

strong hand	: 2 spades

5. Slot Network Search :

6 4 node	: 1 spades
strong two 2 node	: 2 spades
strong two 1 node	: 2 spades

6. AKO Network Search :

6 4 node	: 1 spades
strong two 1 node	: 2 spades
strong two 2 node	: 2 spades

```
Hand2
     balance
          balanced
     quick tricks
          2
     playing tricks
          2
     suit strength
          spades and diamonds are biddible
     division of suits
          4 spades, 4 diamonds, 3 hearts, 2 clubs
     honour points
          14, (14 for no trumps)
          spades = 8, hearts = 2, diamonds = 4, clubs = 0
     cards
          spades A K J 9, hearts Q 9 8, diamonds K J 9 2, clubs 6 5

1. Linear Search :

   one no trumps weak frame    : 1 no trumps
   1 of suit frame             : 1 spades
   4 4 frame                   : 1 spades

2. Set Search :

   1 of suit frame             : 1 spades
   4 4 frame                   : 1 spades
   one no trumps weak frame    : 1 no trumps

3. Hierarchical Search :

   1 no trumps weak tree       : 1 no trumps
   low points equal suits tree : 1 spades
   4 4 tree                    : 1 spades

4. Search Tree Search :

   1 of suit frame             : 1 spades
   4 4 hand                    : 1 spades
   one no trumps weak frame    : 1 no trumps

5. Slot Network Search :

   4 4 node                    : 1 spades
   1 of suit node              : 1 spades
   one no trumps weak node     : 1 no trumps

6. AKO Network Search :

   one no trumps weak node     : 1 no trumps
   1 of suit node              : 1 spades
   4 4 suits node              : 1 spades
```

Hand3
balance
balanced
quick tricks
2
playing tricks
2
suit strength
spades is biddible
division of suits
4 spades, 4 hearts, 3 diamonds, 2 clubs
honour points
10, (10.5 for no trumps)
spades = 5, hearts = 0, diamonds = 5, clubs = 0
cards
spades A J 10 9, hearts 7 6 5 4, diamonds K Q 8, clubs 3 2

1. Linear Search :

No bids

2. Set Search :

No bids

3. Hierarchical Search :

No bids

4. Search Tree Search :

No bids

5. Slot Network Search :

No bids

6. AKO Network Search :

No bids

```
Hand4
     balance
          unbalanced
     quick tricks
          2
     playing tricks
          10
     suit strength
          hearts and diamonds are rebiddible
     division of suits
          7 hearts, 6 diamonds, 0 clubs, 0 spades
     honour points
          12, (12 for no trumps)
          spades = 0, hearts = 3, diamonds = 9, clubs = 0
     cards
          hearts Q J 9 8 7 5 3, diamonds A K Q 9 8 2

1. Linear Search :

   1 of suit frame              : 1 hearts
   unequal suits frame          : 1 hearts
   strong two 1 frame           : 2 hearts

2. Set Search :

   strong two 1 frame           : 2 hearts

3. Hierarchical Search :

   strong two 1 tree            : 2 hearts
   1 suit tree                  : 1 hearts
   other two suits tree         : 1 hearts

4. Search Tree Search :

   1 of suit frame              : 1 hearts
   unequal suits hand           : 1 hearts
   strong hand                  : 2 hearts

5. Slot Network Search :

   unequal suits node           : 1 hearts
   1 of suit node               : 1 hearts
   strong two 1 node            : 2 hearts

6. AKO Network Search :

   1 of suit node               : 1 hearts
   strong two 1 node            : 2 hearts
   unequal suits node           : 1 hearts
```

APPENDIX D. Trace of Bird Recognition System.

D-1 Presentation of the birds

Internal frame representation :

```
(deframe "paradise duck"
    (ako
        ($value
            ("large sized bird")
            ("land and water bird")
            ("dark billed bird")
            ("dark feet bird")
            ("dark undersides bird")
            ("omnivorous bird")
            ("grassy nest bird")
            ("creamy eggs bird")
            ("bird")
        )
    )
    (size
        ($value
            (63)))
    (appearance
        ($value
            ("paradise duck appearance")))
    (distribution
        ($value
            ("new zealand")))
    (habitat
        ($value
            (hill-country) (river) (lake) (pond) (open) (pasture)))
    (behaviour
        ($value
            ((run (fast strong)))))
    (flight
        ($value
            ((wings (white) coverts (white)))))
    (food
        ($value
            (grass) (plants) (insects)))
    (breeding
        ($value
            ((august to january))))
    ("nest material"
        ($value
            (grass) (down)))
    ("nest location"
        ($value
            ((on (ground)))))
    ("number of eggs"
        ($value
            ((5 to 11))))
    ("egg colour"
        ($value
            ((mainly (cream)))))
)
```

Displaying a bird description :

Paradise Duck
- size
  - 63
- appearance
  - male and immature
    - mainly black, shining black head, white coverts, red brown abdomen, black bill, dark black long feet, short stout neck, green speculum
  - female
    - mainly bright brown chestnut, white head, dark back, green speculum, white coverts, brown underparts, black bill, dark black long feet, short stout neck
- distribution
  - New Zealand
- habitat
  - hill-country, river, lake, pond, open, pasture
- behaviour
  - fast strong run
- flight
  - white wings, white coverts
- food
  - grass, plants, insects
- breeding
  - August to January
- nest material
  - grass, down
- nest location
  - on ground
- number of eggs
  - 5 to 11
- egg colour
  - mainly cream

Entering a bird description :

Enter a name for the bird : BLACK AND WHITE BIRD

Enter estimate of bird's size (large, medium or small) :
MEDIUM

Enter a description of the bird's appearance:
MAINLY BLACK AND WHITE
Invalid adjective entered - please reenter.
If you are not sure, then enter 'help'.

Enter a description of the bird's appearance:
HELP
The appearance feature of a bird uses an english description to describe the appearance of the bird. The description takes the form of a list of phrases consisting of a noun modified by one or more adjectives. Some examples of acceptable phrases are: wide wings, colourful body, large eyes. The overall appearance of the bird can also be described by using the adverb "mainly" such as : mainly blue black brown, mainly dark.

There may be more than one type of appearance for a bird stored in the database (for example a male pheasant looks completely different from a female pheasant). When the bird is displayed, the different appearances are shown indented under the name of the type of bird.

The list of words allowed in the description are :

nouns :

abdomen	back	bill	body
breast	cheeks	chest	chin
claw	coverts	crest	crown
eyes	face	feet	flanks
forehead	head	mantle	mainly
nape	neck	plumage	rump
scapulars	shield	sides	speculum
tail	thighs	throat	underparts
underwing	upperparts	wings	

adjectives :

apple	barred	black	blue
bright	brown	buff	chalky
chestnut	clear	colourful	cream
curving	dark	dour	downy
dull	fanned	forked	freckled
glossy	green	grey	intensive
large	light	long	mottled
muddy	off	olive	orange
pale	pink	plain	plump
pointed	pure	purple	red
rounded	scaled	sharp	shining
short	small	spotted	silver
stout	streaked	thick	turquoise
untidy	white	wide	yellow

Enter a description of the bird's appearance:
MAINLY BLACK WHITE, GREY BACK

Enter the words that best describe the bird's distribution :

KAIKOURA

Enter the words that best describe the bird's habitat :
H
The habitat feature describes the habitats where the bird is likely to be found. Common habitats are : open, alpine, coast, gardens, habitation, forest.

The following are valid words :

alpine	clearings	cliffs	coast
crops	drains	forest	gardens
habitation	hedges	hill-country	lagoon
lake	marsh	open	orchards
pasture	plantations	pond	river
roadside	scrub		

Enter the words that best describe the bird's habitat :
OPEN

Enter a description of the bird's behaviour:
H
The behaviour of the bird describes the most noticeable idiosyncrosies of the bird in the field. Some examples are : cheerful attitude,fast swimmer, nodding head and frequent dives.

The list of words allowed in the description are :
nouns :

attitude	dives	head	rump
run	swim	tail	walk
wings			

adjectives :

bobbing	cheerful	comfortable	crouched
drooping	fast	flicking	frequent
freezing	frowning	high	jaunty
nimble	scavenging	secretive	sedate
skulking	slow	staring	strong

Enter a description of the bird's behaviour:
NIMBLE WALK

Enter a description of the bird's flight:
SWIFT FLIGHT
Invalid adjective entered - please reenter.
If you are not sure, then enter 'help'.

Enter a description of the bird's flight:
H
The flight feature is used to describe the predominant features of a bird's flight. Often a bird looks different from the air, and the appearance from the ground can also be entered here such as : black rump, white coverts, short wings. Other features such as the flight and take-off can also be described.

The list of words allowed in the description are :
nouns :

coverts	feathers	feet	flight
neck	rump	sides	tail
take-off	wingbeat	wings	

adjectives :

black	dangling	dark	drooping
fast	gliding	high	impulsive
laboured	large	low	outstretched
pale	retracted	short	slow
small	soaring	strong	sudden
wheeling	whirring	white	

Enter a description of the bird's flight:
FAST FLIGHT

Enter the words that best describe the bird's food :
INSECTS WORMS

Was there a nest found ? NO

Display produced from the above description :

Black And White Bird
- food
  - insects, worms
- flight
  - fast flight
- behaviour
  - nimble walk
- habitat
  - open
- distribution
  - Kaikoura
- appearance
  - mainly white black, grey back
- size
  - medium

D-2 Specific Search

Entering the bird

```
Enter a name for the bird : DUCK
Enter estimate of bird's size (large, medium or small) :
LARGE

Enter a description of the bird's appearance:
MAINLY BROWN BLACK WHITE, WHITE HEAD, DARK FEET

Enter the words that best describe the bird's distribution :
CANTERBURY

Enter the words that best describe the bird's habitat :
HILL-COUNTRY, POND
Enter a description of the bird's behaviour:

Enter a description of the bird's flight:
WHITE WINGS

Enter the words that best describe the bird's food :
GRASS INSECTS

Was there a nest found ? YES
Enter the words that best describe the bird's nest material :
GRASS DOWN

Enter the location of the bird's nest :
ON GROUND
month in which nest was found :

How many eggs were found? 11
Enter a description of the colour of the eggs:
MAINLY CREAM
```

Displaying the bird

```
Duck
     egg colour
          mainly cream
     number of eggs
          11
     nest location
          on ground
     nest material
          down, grass
     food
          grass, insects
     flight
          white wings
     habitat
          hill-country, pond
     distribution
          Canterbury
     appearance
          mainly white black brown, white head, dark feet
     size
          large
```

Linear Search :

Each frame in the INSTANCE slot of the BIRD frame are matched one after the other...

The New Zealand Dabchick is too small.
The bird is not a New Zealand Dabchick.

A large-sized bird matches the size of the White Faced Heron which is 66 cm.
The bird does not look like the White Faced Heron.
The bird is not a White Faced Heron.

A large-sized bird matches the size of the Bittern which is 71 cm.
The bird does not look like the Bittern.
The bird is not a Bittern.

A large-sized bird matches the size of the Black Swan which is 100 cm.
The bird looks like the adult Black Swan.
The Black Swan is usually found in New Zealand which matches with where the bird was found.
The bird's habitat was described as hill-country, pond so the bird cannot be the Black Swan because its habitat is coast, lake.
The bird is not a Black Swan.

A large-sized bird matches the size of the Mute Swan which is 150 cm.
The bird looks like the adult Mute Swan.
The bird looks like the immature Mute Swan.
The Mute Swan is usually found in Canterbury, Hawkes Bay which matches with where the bird was found.
The bird's habitat was described as hill-country, pond which matches with where the Mute Swan is usually found which is lake, pond.
The bird's food was described as grass, insects so the bird cannot be the Mute Swan because it eats plants.
The bird is not a Mute Swan.

A large-sized bird matches the size of the Canada Goose which is 100 cm.
The bird does not look like the immature or adult Canada Goose.
The bird is not a Canada Goose.

A large-sized bird matches the size of the Paradise Duck which is 63 cm.
The bird looks like the female Paradise Duck.
The Paradise Duck is usually found in New Zealand which matches with where the bird was found.
The bird's habitat was described as hill-country, pond which matches with where the Paradise Duck is usually found which is hill-country, river, lake, pond, open, pasture.
The description of the bird's flight was white wings which matches the flight description of the Paradise Duck which is white wings, white coverts.
The bird's food was described as grass, insects which matches

with what the Paradise Duck eats which is grass, plants, insects.
The bird's nest material was described as grass, down which matches with what the Paradise Duck uses to build its nest which is grass, down.
The description of the location of the nest was on ground which matches with the nest location of the Paradise Duck which is on ground.
The Paradise Duck usually has 5 to 11 eggs in its nest which matches with the 11 eggs that were found.
The colour of the eggs that were found was mainly cream which matches the colour of the eggs of the Paradise Duck which is mainly cream.

The female Paradise Duck matches the description.

A large-sized bird matches the size of the Mallard which is 58 cm.
The bird does not look like the male eclipse plumage, female and immature or male breeding plumage Mallard.
The bird is not a Mallard.

A large-sized bird matches the size of the Grey Duck which is 55 cm.
The bird does not look like the Grey Duck.
The bird is not a Grey Duck.

The New Zealand Scaup is too small.
The bird is not a New Zealand Scaup.

The Grey Teal is too small.
The bird is not a Grey Teal.

The New Zealand Shoveler is too small.
The bird is not a New Zealand Shoveler.

A large-sized bird matches the size of the Harrier which is 60 cm.
The bird looks like the old males Harrier.
The Harrier is usually found in New Zealand which matches with where the bird was found.
The bird's habitat was described as hill-country, pond which matches with where the Harrier is usually found which is not forest, alpine.
The bird's food was described as grass, insects which matches with what the Harrier eats which is mammals, mice, rabbits, rodents, insects, lizards, carrion.
The bird's nest material was described as grass, down which matches with what the Harrier uses to build its nest which is tussock, sticks, grass, rushes.
There are too many eggs in the nest for the bird to be the Harrier.
The bird is not a Harrier.

The Californian Quail is too small.
The bird is not a Californian Quail.

The Brown Quail is too small.
The bird is not a Brown Quail.

A large-sized bird matches the size of the Pheasant which is male 80 cm. and female 60 cm.
The bird looks like the female Pheasant.
The bird cannot be the Pheasant which is usually found in North Island.
The bird is not a Pheasant.

The Banded Rail is too small.
The bird is not a Banded Rail.

A large-sized bird matches the size of the Pukeko which is 51 cm.
The bird does not look like the Pukeko.
The bird is not a Pukeko.

A large-sized bird matches the size of the Australian Coot which is 101 cm.
The bird does not look like the immature or adult Australian Coot.
The bird is not an Australian Coot.

The Spur Winged Plover is too small.
The bird is not a Spur Winged Plover.

The Rock Pigeon is too small.
The bird is not a Rock Pigeon.

The Little Owl is too small.
The bird is not a Little Owl.

The Kingfisher is too small.
The bird is not a Kingfisher.

The Skylark is too small.
The bird is not a Skylark.

The New Zealand Pipit is too small.
The bird is not a New Zealand Pipit.

The Welcome Swallow is too small.
The bird is not a Welcome Swallow.

Similar output is displayed for the remaining medium and small sized birds which fail to match...

The bird(s) that match are :
female Paradise Duck

Set Search :

The set frames such as "medium sized bird" and "large sized bird" are matched first in the search. Only the set frames that match display a message...

The bird is a large sized bird.
The bird cannot be any of these birds which are not large sized birds:

Rook	White Backed Magpie	Black Backed Magpie
Indian Myna	Starling	House Sparrow
Redpoll	Goldfinch	Greenfinch
Chaffinch	Cirl Bunting	Yellowhammer
Silvereye	Blackbird	Song Thrush
Fantail	Grey Warbler	Fernbird
Hedge Sparrow	Welcome Swallow	New Zealand Pipit
Skylark	Kingfisher	Little Owl
Rock Pigeon	Spur Winged Plover	Banded Rail
Brown Quail	Californian Quail	New Zealand Shoveler
Grey Teal	New Zealand Scaup	New Zealand Dabchick

The bird is a South Island bird.
The bird cannot be any of these birds which are not South Island birds:

Indian Myna	Pheasant	Brown Quail
New Zealand Dabchick		

The bird is a water bird.
The bird cannot be any of these birds which are not water birds:

Rook	Indian Myna	House Sparrow
Redpoll	Greenfinch	Chaffinch
Cirl Bunting	Yellowhammer	Silvereye
Fantail	Hedge Sparrow	New Zealand Pipit
Little Owl	Pheasant	Californian Quail

The bird is a land and water bird.
The bird cannot be any of these birds which are not land and water birds:

Rook	Indian Myna	House Sparrow
Redpoll	Greenfinch	Chaffinch
Cirl Bunting	Yellowhammer	Silvereye
Fantail	Hedge Sparrow	New Zealand Pipit
Little Owl	Spur Winged Plover	Australian Coot
Pukeko	Banded Rail	Pheasant
Californian Quail	New Zealand Shoveler	Grey Teal
New Zealand Scaup	Grey Duck	Mallard
Mute Swan	Black Swan	Bittern
New Zealand Dabchick		

The bird is an animal eating bird.
The bird cannot be any of these birds which are not animal eating birds:

Rock Pigeon	Brown Quail	Californian Quail
Canada Goose	Black Swan	

The bird is a plant eating bird.
The bird cannot be any of these birds which are not plant eating birds:

Fantail	Grey Warbler	Fernbird
Welcome Swallow	New Zealand Pipit	Kingfisher
Little Owl	Spur Winged Plover	Harrier
Bittern	White Faced Heron	New Zealand Dabchick

The bird is an omnivorous bird.
The bird cannot be any of these birds which are not omnivorous birds:

White Backed Magpie	Black Backed Magpie	Grey Warbler
Fernbird	Welcome Swallow	New Zealand Pipit
Kingfisher	Little Owl	Rock Pigeon
Spur Winged Plover	Brown Quail	Californian Quail
Harrier	Canada Goose	Mute Swan
Black Swan	Bittern	White Faced Heron
New Zealand Dabchick		

The bird is a grassy nest bird.
The bird cannot be any of these birds which are not grassy nest birds:

Greenfinch	Fantail	Kingfisher
Little Owl	Rock Pigeon	Spur Winged Plover
White Faced Heron		

The bird is a many eggs bird.
The bird cannot be any of these birds which are not many eggs birds:

Rook	White Backed Magpie	Black Backed Magpie
Indian Myna	Redpoll	Cirl Bunting
Yellowhammer	Silvereye	Blackbird
Song Thrush	Fantail	Grey Warbler
Fernbird	Hedge Sparrow	Welcome Swallow
New Zealand Pipit	Kingfisher	Little Owl
Rock Pigeon	Spur Winged Plover	Harrier
White Faced Heron	New Zealand Dabchick	

The bird is a creamy eggs bird.
The bird cannot be any of these birds which are not creamy eggs birds:

Rook	White Backed Magpie	Black Backed Magpie
Indian Myna	Starling	Chaffinch
Cirl Bunting	Silvereye	Blackbird
Song Thrush	Hedge Sparrow	Spur Winged Plover
Banded Rail	Pheasant	White Faced Heron

The bird is a dark feet bird.
The bird cannot be any of these birds which are not dark feet birds:

Indian Myna	Rock Pigeon	Spur Winged Plover
Pukeko	Harrier	New Zealand Shoveler
Mallard	Bittern	White Faced Heron

The search now tries to match all the possible bird frames that were

returned by the set search. In this example, only the Paradise Duck was returned.

A large-sized bird matches the size of the Paradise Duck which is 63 cm.
The bird looks like the female Paradise Duck.
The Paradise Duck is usually found in New Zealand which matches with where the bird was found.
The bird's habitat was described as hill-country, pond which matches with where the Paradise Duck is usually found which is hill-country, river, lake, pond, open, pasture.
The description of the bird's flight was white wings which matches the flight description of the Paradise Duck which is white wings, white coverts.
The bird's food was described as grass, insects which matches with what the Paradise Duck eats which is grass, plants, insects.
The bird's nest material was described as grass, down which matches with what the Paradise Duck uses to build its nest which is grass, down.
The description of the location of the nest was on ground which matches with the nest location of the Paradise Duck which is on ground.
The Paradise Duck usually has 5 to 11 eggs in its nest which matches with the 11 eggs that were found.
The colour of the eggs that were found was mainly cream which matches the colour of the eggs of the Paradise Duck which is mainly cream.

The female Paradise Duck matches the description.

The bird(s) that match are :
female Paradise Duck

Search Tree Search :

The first terminal node of the search tree that matches is the "large coloured bill bird" frame which indicates three birds to try, the Australian Coot, the Pukeko and the Black Swan...

A large-sized bird matches the size of the Australian Coot which is 101 cm.
The bird does not look like the immature or adult Australian Coot.
The bird is not an Australian Coot.

A large-sized bird matches the size of the Pukeko which is 51 cm.
The bird does not look like the Pukeko.
The bird is not a Pukeko.

A large-sized bird matches the size of the Black Swan which is 100 cm.
The bird looks like the adult Black Swan.
The Black Swan is usually found in New Zealand which matches with where the bird was found.
The bird's habitat was described as hill-country, pond so the bird cannot be the Black Swan because its habitat is coast, lake.
The bird is not a Black Swan.

As the colour of the bill in the description being matched is unknown, the alternative "not large coloured bill bird" frame is also searched, and the Harrier, Grey Duck, Paradise Duck and Bittern are consequently searched...

A large-sized bird matches the size of the Harrier which is 60 cm.
The bird looks like the old males Harrier.
The Harrier is usually found in New Zealand which matches with where the bird was found.
The bird's habitat was described as hill-country, pond which matches with where the Harrier is usually found which is not forest, alpine.
The bird's food was described as grass, insects which matches with what the Harrier eats which is mammals, mice, rabbits, rodents, insects, lizards, carrion.
The bird's nest material was described as grass, down which matches with what the Harrier uses to build its nest which is tussock, sticks, grass, rushes.
There are too many eggs in the nest for the bird to be the Harrier.
The bird is not a Harrier.

A large-sized bird matches the size of the Grey Duck which is 55 cm.
The bird does not look like the Grey Duck.
The bird is not a Grey Duck.

A large-sized bird matches the size of the Paradise Duck which is 63 cm.
The bird looks like the female Paradise Duck.
The Paradise Duck is usually found in New Zealand which matches with where the bird was found.
The bird's habitat was described as hill-country, pond which matches with where the Paradise Duck is usually found which is hill-country, river, lake, pond, open, pasture.
The description of the bird's flight was white wings which matches the flight description of the Paradise Duck which is white wings, white coverts.
The bird's food was described as grass, insects which matches with what the Paradise Duck eats which is grass, plants, insects.
The bird's nest material was described as grass, down which matches with what the Paradise Duck uses to build its nest which is grass, down.
The description of the location of the nest was on ground which matches with the nest location of the Paradise Duck which is on ground.
The Paradise Duck usually has 5 to 11 eggs in its nest which matches with the 11 eggs that were found.
The colour of the eggs that were found was mainly cream which matches the colour of the eggs of the Paradise Duck which is mainly cream.

The female Paradise Duck matches the description.

A large-sized bird matches the size of the Bittern which is 71 cm.
The bird does not look like the Bittern.
The bird is not a Bittern.

The New Zealand Dabchick, Banded Rail and Spur Winged Plover are indicated by the "water black white bird" frame...

The New Zealand Dabchick is too small.
The bird is not a New Zealand Dabchick.

The Banded Rail is too small.
The bird is not a Banded Rail.

The Spur Winged Plover is too small.
The bird is not a Spur Winged Plover.

The bird(s) that match are :
female Paradise Duck

Network Search :

The search tries to match the AKO slots of the first frame in the INSTANCE slot of the BIRD frame. As the New Zealand Dabchick is an AKO "medium sized bird", it fails to match. The White Faced Heron is tried next...

The bird is a large sized bird.
The bird cannot be any of these birds which are not large sized birds:

Rook	White Backed Magpie	Black Backed Magpie
Indian Myna	Starling	House Sparrow
Redpoll	Goldfinch	Greenfinch
Chaffinch	Cirl Bunting	Yellowhammer
Silvereye	Blackbird	Song Thrush
Fantail	Grey Warbler	Fernbird
Hedge Sparrow	Welcome Swallow	New Zealand Pipit
Skylark	Kingfisher	Little Owl
Rock Pigeon	Spur Winged Plover	Banded Rail
Brown Quail	Californian Quail	New Zealand Shoveler
Grey Teal	New Zealand Scaup	New Zealand Dabchick

The bird is a water bird.
The bird cannot be any of these birds which are not water birds:

Rook	Indian Myna	House Sparrow
Redpoll	Greenfinch	Chaffinch
Cirl Bunting	Yellowhammer	Silvereye
Fantail	Hedge Sparrow	New Zealand Pipit
Little Owl	Pheasant	Californian Quail

The bird is an animal eating bird.
The bird cannot be any of these birds which are not animal eating birds:

Rock Pigeon	Brown Quail	Californian Quail
Canada Goose	Black Swan	

The White Faced Heron is not a "branchy nest bird" and so the Bittern is tried next...

The bird could be a Bittern because it is a :
water bird, large sized bird, and animal eating bird.
A large-sized bird matches the size of the Bittern which is 71 cm.
The bird does not look like the Bittern.
The bird is not a Bittern.

The bird is a plant eating bird.
The bird cannot be any of these birds which are not plant eating birds:

Fantail	Grey Warbler	Fernbird
Welcome Swallow	New Zealand Pipit	Kingfisher
Little Owl	Spur Winged Plover	Harrier

Bittern	White Faced Heron	New Zealand Dabchick

The Bittern fails to match, so the Black Swan is tried next...

The bird is a grassy nest bird.
The bird cannot be any of these birds which are not grassy nest birds:

Greenfinch	Fantail	Kingfisher
Little Owl	Rock Pigeon	Spur Winged Plover
White Faced Heron		

The bird could be a Black Swan because it is a :
plant eating bird, water bird, large sized bird, and grassy nest bird.
A large-sized bird matches the size of the Black Swan which is 100 cm.
The bird looks like the adult Black Swan.
The Black Swan is usually found in New Zealand which matches with where the bird was found.
The bird's habitat was described as hill-country, pond so the bird cannot be the Black Swan because its habitat is coast, lake.
The bird is not a Black Swan.

Similary, the Mute Swan, the Canada Goose and the Paradise Duck are tried...

The bird could be a Mute Swan because it is a :
plant eating bird, water bird, large sized bird, and grassy nest bird.
A large-sized bird matches the size of the Mute Swan which is 150 cm.
The bird looks like the adult Mute Swan.
The bird looks like the immature Mute Swan.
The Mute Swan is usually found in Canterbury, Hawkes Bay which matches with where the bird was found.
The bird's habitat was described as hill-country, pond which matches with where the Mute Swan is usually found which is lake, pond.
The bird's food was described as grass, insects so the bird cannot be the Mute Swan because it eats plants.
The bird is not a Mute Swan.

The bird is a land and water bird.
The bird cannot be any of these birds which are not land and water birds:

Rook	Indian Myna	House Sparrow
Redpoll	Greenfinch	Chaffinch
Cirl Bunting	Yellowhammer	Silvereye
Fantail	Hedge Sparrow	New Zealand Pipit
Little Owl	Spur Winged Plover	Australian Coot

Pukeko	Banded Rail	Pheasant
Californian Quail	New Zealand Shoveler	Grey Teal
New Zealand Scaup	Grey Duck	Mallard
Mute Swan	Black Swan	Bittern
New Zealand Dabchick		

The bird is a South Island bird.
The bird cannot be any of these birds which are not South Island birds:

Indian Myna	Pheasant	Brown Quail
New Zealand Dabchick		

The bird is a creamy eggs bird.
The bird cannot be any of these birds which are not creamy eggs birds:

Rook	White Backed Magpie	Black Backed Magpie
Indian Myna	Starling	Chaffinch
Cirl Bunting	Silvereye	Blackbird
Song Thrush	Hedge Sparrow	Spur Winged Plover
Banded Rail	Pheasant	White Faced Heron

The bird could be a Canada Goose because it is a :
grassy nest bird, plant eating bird, South Island bird, land and water bird, large sized bird, and creamy eggs bird.
A large-sized bird matches the size of the Canada Goose which is 100 cm.
The bird does not look like the immature or adult Canada Goose.
The bird is not a Canada Goose.

The bird is a dark feet bird.
The bird cannot be any of these birds which are not dark feet birds:

Indian Myna	Rock Pigeon	Spur Winged Plover
Pukeko	Harrier	New Zealand Shoveler
Mallard	Bittern	White Faced Heron

The bird is an omnivorous bird.
The bird cannot be any of these birds which are not omnivorous birds:

White Backed Magpie	Black Backed Magpie	Grey Warbler
Fernbird	Welcome Swallow	New Zealand Pipit
Kingfisher	Little Owl	Rock Pigeon
Spur Winged Plover	Brown Quail	Californian Quail
Harrier	Canada Goose	Mute Swan
Black Swan	Bittern	White Faced Heron
New Zealand Dabchick		

The bird could be a Paradise Duck because it is a :
grassy nest bird, omnivorous bird, dark feet bird, land and water bird, large sized bird, and creamy eggs bird.
A large-sized bird matches the size of the Paradise Duck which is 63 cm.
The bird looks like the female Paradise Duck.
The Paradise Duck is usually found in New Zealand which matches with where the bird was found.

The bird's habitat was described as hill-country, pond which matches with where the Paradise Duck is usually found which is hill-country, river, lake, pond, open, pasture.
The description of the bird's flight was white wings which matches the flight description of the Paradise Duck which is white wings, white coverts.
The bird's food was described as grass, insects which matches with what the Paradise Duck eats which is grass, plants, insects.
The bird's nest material was described as grass, down which matches with what the Paradise Duck uses to build its nest which is grass, down.
The description of the location of the nest was on ground which matches with the nest location of the Paradise Duck which is on ground.
The Paradise Duck usually has 5 to 11 eggs in its nest which matches with the 11 eggs that were found.
The colour of the eggs that were found was mainly cream which matches the colour of the eggs of the Paradise Duck which is mainly cream.

The female Paradise Duck matches the description.

As most of the birds have now been rejected or have invalid AKO slots, only the Australian Coot, Kingfisher and the Starling require further searching...

The bird could be a Australian Coot because it is a :
dark feet bird, water bird, large sized bird, and omnivorous bird.
A large-sized bird matches the size of the Australian Coot which is 101 cm.
The bird does not look like the immature or adult Australian Coot.
The bird is not an Australian Coot.

The bird could be a Kingfisher because it is a :
animal eating bird, dark feet bird, land and water bird, and creamy eggs bird.
The Kingfisher is too small.
The bird is not a Kingfisher.

The bird could be a Starling because it is a :
omnivorous bird, land and water bird, and grassy nest bird.
The Starling is too small.
The bird is not a Starling.

The bird(s) that match are :
female Paradise Duck

D-3 General Query

Entering the bird

```
Enter a name for the bird : HELP
Any name may be used for the bird's description

Enter a name for the bird : LARGE BLACK BIRD

Enter estimate of bird's size (large, medium or small) :
LARGE

Enter a description of the bird's appearance:
MAINLY BLACK, BLACK BODY

Enter the words that best describe the bird's distribution :
QUIT
```

Displaying the Bird

```
Large Black Bird
     appearance
          mainly black, black body
     size
          large
```

Linear Search :

The New Zealand Dabchick is too small.
The bird is not a New Zealand Dabchick.

A large-sized bird matches the size of the White Faced Heron which is 66 cm.
The bird does not look like the White Faced Heron.
The bird is not a White Faced Heron.

A large-sized bird matches the size of the Bittern which is 71 cm.
The bird does not look like the Bittern.
The bird is not a Bittern.

A large-sized bird matches the size of the Black Swan which is 100 cm.
The bird looks like the adult Black Swan.

The adult Black Swan matches the description.

A large-sized bird matches the size of the Mute Swan which is 150 cm.
The bird does not look like the immature or adult Mute Swan.
The bird is not a Mute Swan.

A large-sized bird matches the size of the Canada Goose which is 100 cm.
The bird does not look like the immature or adult Canada Goose.
The bird is not a Canada Goose.

A large-sized bird matches the size of the Paradise Duck which is 63 cm.
The bird looks like the male and immature Paradise Duck.

The male and immature Paradise Duck matches the description.

A large-sized bird matches the size of the Mallard which is 58 cm.
The bird does not look like the male eclipse plumage, female and immature or male breeding plumage Mallard.
The bird is not a Mallard.

A large-sized bird matches the size of the Grey Duck which is 55 cm.
The bird does not look like the Grey Duck.
The bird is not a Grey Duck.

The New Zealand Scaup is too small.
The bird is not a New Zealand Scaup.

The Grey Teal is too small.
The bird is not a Grey Teal.

The New Zealand Shoveler is too small.
The bird is not a New Zealand Shoveler.

A large-sized bird matches the size of the Harrier which is 60 cm.

The bird does not look like the immature, old males or adult Harrier.
The bird is not a Harrier.

The Californian Quail is too small.
The bird is not a Californian Quail.

The Brown Quail is too small.
The bird is not a Brown Quail.

A large-sized bird matches the size of the Pheasant which is male 80 cm. and female 60 cm.
The bird does not look like the female or male Pheasant.
The bird is not a Pheasant.

The Banded Rail is too small.
The bird is not a Banded Rail.

A large-sized bird matches the size of the Pukeko which is 51 cm.
The bird has the overall appearance of the Pukeko.

The overall appearance of the Pukeko matches the description.

A large-sized bird matches the size of the Australian Coot which is 101 cm.
The bird looks like the adult Australian Coot.
The bird looks like the immature Australian Coot.

The immature and adult Australian Coot matches the description.

The Spur Winged Plover is too small.
The bird is not a Spur Winged Plover.

The Rock Pigeon is too small.
The bird is not a Rock Pigeon.

The Little Owl is too small.
The bird is not a Little Owl.

The Kingfisher is too small.
The bird is not a Kingfisher.

Similar output is displayed for the remaining medium and small sized birds which fail to match...

The bird(s) that match are :
adult Black Swan
male and immature Paradise Duck
overall Pukeko appearance
immature Australian Coot
adult Australian Coot

Set Search :

Only one set frame, "large sized bird" matches the general query...

The bird is a large sized bird.
The bird cannot be any of these birds which are not large sized birds:

Rook	White Backed Magpie	Black Backed Magpie
Indian Myna	Starling	House Sparrow
Redpoll	Goldfinch	Greenfinch
Chaffinch	Cirl Bunting	Yellowhammer
Silvereye	Blackbird	Song Thrush
Fantail	Grey Warbler	Fernbird
Hedge Sparrow	Welcome Swallow	New Zealand Pipit
Skylark	Kingfisher	Little Owl
Rock Pigeon	Spur Winged Plover	Banded Rail
Brown Quail	Californian Quail	New Zealand Shoveler
Grey Teal	New Zealand Scaup	New Zealand Dabchick

All the large sized birds are returned by the set search and are subsequently searched...

A large-sized bird matches the size of the White Faced Heron which is 66 cm.
The bird does not look like the White Faced Heron.
The bird is not a White Faced Heron.

A large-sized bird matches the size of the Bittern which is 71 cm.
The bird does not look like the Bittern.
The bird is not a Bittern.

A large-sized bird matches the size of the Black Swan which is 100 cm.
The bird looks like the adult Black Swan.

The adult Black Swan matches the description.

A large-sized bird matches the size of the Mute Swan which is 150 cm.
The bird does not look like the immature or adult Mute Swan.
The bird is not a Mute Swan.

A large-sized bird matches the size of the Canada Goose which is 100 cm.
The bird does not look like the immature or adult Canada Goose.
The bird is not a Canada Goose.

A large-sized bird matches the size of the Paradise Duck which is 63 cm.
The bird looks like the male and immature Paradise Duck.

The male and immature Paradise Duck matches the description.

A large-sized bird matches the size of the Mallard which is 58 cm.
The bird does not look like the male eclipse plumage, female and immature or male breeding plumage Mallard.
The bird is not a Mallard.

A large-sized bird matches the size of the Grey Duck which is 55 cm.
The bird does not look like the Grey Duck.
The bird is not a Grey Duck.

A large-sized bird matches the size of the Harrier which is 60 cm.
The bird does not look like the immature, old males or adult Harrier.
The bird is not a Harrier.

A large-sized bird matches the size of the Pheasant which is male 80 cm. and female 60 cm.
The bird does not look like the female or male Pheasant.
The bird is not a Pheasant.

A large-sized bird matches the size of the Pukeko which is 51 cm.
The bird has the overall appearance of the Pukeko.

The overall appearance of the Pukeko matches the description.

A large-sized bird matches the size of the Australian Coot which is 101 cm.
The bird looks like the adult Australian Coot.
The bird looks like the immature Australian Coot.

The immature and adult Australian Coot matches the description.

The bird(s) that match are :
adult Black Swan
male and immature Paradise Duck
overall Pukeko appearance
immature Australian Coot
adult Australian Coot

Search Tree Search :

The "large coloured bill bird" frame indicates the Australian Coot, the Pukeko and the Black Swan...

A large-sized bird matches the size of the Australian Coot which is 101 cm.
The bird looks like the adult Australian Coot.
The bird looks like the immature Australian Coot.

The immature and adult Australian Coot matches the description.

A large-sized bird matches the size of the Pukeko which is 51 cm.
The bird has the overall appearance of the Pukeko.

The overall appearance of the Pukeko matches the description.

A large-sized bird matches the size of the Black Swan which is 100 cm.
The bird looks like the adult Black Swan.

The adult Black Swan matches the description.

The "not large coloured bill bird" frame is also searched as the colour of the bill is unknown, and therefore the Harrier, Grey Duck, Paradise Duck and Bittern are subsequently searched...

A large-sized bird matches the size of the Harrier which is 60 cm.
The bird does not look like the immature, old males or adult Harrier.
The bird is not a Harrier.

A large-sized bird matches the size of the Grey Duck which is 55 cm.
The bird does not look like the Grey Duck.
The bird is not a Grey Duck.

A large-sized bird matches the size of the Paradise Duck which is 63 cm.
The bird looks like the male and immature Paradise Duck.

The male and immature Paradise Duck matches the description.

A large-sized bird matches the size of the Bittern which is 71 cm.
The bird does not look like the Bittern.
The bird is not a Bittern.

The "coloured bill big black bird" indicates the Pukeko, Blackbird, Indian Myna and the Starling...

The Blackbird is too small.
The bird is not a Blackbird.

The Indian Myna is too small.
The bird is not an Indian Myna.

The Starling is too small.
The bird is not a Starling.

The "not coloured bill big black bird" is also searched as the colour of the bill is unknown, and therefore the Rook and the New Zealand Scaup are subsequently searched...

The Rook is too small.
The bird is not a Rook.

The New Zealand Scaup is too small.
The bird is not a New Zealand Scaup.

The bird(s) that match are :
immature Australian Coot
adult Australian Coot
overall Pukeko appearance
adult Black Swan
male and immature Paradise Duck

Network Search :

The White Faced Heron is the first "large sized bird" to be tried...

The bird is a large sized bird.
The bird cannot be any of these birds which are not large sized birds:

Rook	White Backed Magpie	Black Backed Magpie
Indian Myna	Starling	House Sparrow
Redpoll	Goldfinch	Greenfinch
Chaffinch	Cirl Bunting	Yellowhammer
Silvereye	Blackbird	Song Thrush
Fantail	Grey Warbler	Fernbird
Hedge Sparrow	Welcome Swallow	New Zealand Pipit
Skylark	Kingfisher	Little Owl
Rock Pigeon	Spur Winged Plover	Banded Rail
Brown Quail	Californian Quail	New Zealand Shoveler
Grey Teal	New Zealand Scaup	New Zealand Dabchick

The bird could be a White Faced Heron because it is a :
large sized bird.
A large-sized bird matches the size of the White Faced Heron which is 66 cm.
The bird does not look like the White Faced Heron.
The bird is not a White Faced Heron.

All the birds that are possibly large sized birds are now searched...

The bird could be a Bittern because it is a :
large sized bird.
A large-sized bird matches the size of the Bittern which is 71 cm.
The bird does not look like the Bittern.
The bird is not a Bittern.

The bird could be a Black Swan because it is a :
large sized bird.
A large-sized bird matches the size of the Black Swan which is 100 cm.
The bird looks like the adult Black Swan.

The adult Black Swan matches the description.

The bird could be a Mute Swan because it is a :
large sized bird.
A large-sized bird matches the size of the Mute Swan which is 150 cm.

The bird does not look like the immature or adult Mute Swan.
The bird is not a Mute Swan.

The bird could be a Canada Goose because it is a :
large sized bird.
A large-sized bird matches the size of the Canada Goose which is 100 cm.
The bird does not look like the immature or adult Canada Goose.
The bird is not a Canada Goose.

The bird could be a Paradise Duck because it is a :
large sized bird.
A large-sized bird matches the size of the Paradise Duck which is 63 cm.
The bird looks like the male and immature Paradise Duck.

The male and immature Paradise Duck matches the description.

The bird could be a Mallard because it is a :
large sized bird.
A large-sized bird matches the size of the Mallard which is 58 cm.
The bird does not look like the male eclipse plumage, female and immature or male breeding plumage Mallard.
The bird is not a Mallard.

The bird could be a Grey Duck because it is a :
large sized bird.
A large-sized bird matches the size of the Grey Duck which is 55 cm.
The bird does not look like the Grey Duck.
The bird is not a Grey Duck.

The bird could be a Harrier because it is a :
large sized bird.
A large-sized bird matches the size of the Harrier which is 60 cm.
The bird does not look like the immature, old males or adult Harrier.
The bird is not a Harrier.

The bird could be a Californian Quail.
The Californian Quail is too small.
The bird is not a Californian Quail.

The bird could be a Pheasant because it is a :
large sized bird.
A large-sized bird matches the size of the Pheasant which is male 80 cm. and female 60 cm.
The bird does not look like the female or male Pheasant.
The bird is not a Pheasant.

The bird could be a Pukeko.
A large-sized bird matches the size of the Pukeko which is 51 cm.
The bird has the overall appearance of the Pukeko.

The overall appearance of the Pukeko matches the description.

The bird could be a Australian Coot because it is a :
large sized bird.
A large-sized bird matches the size of the Australian Coot which is 101 cm.
The bird looks like the adult Australian Coot.
The bird looks like the immature Australian Coot.

The immature and adult Australian Coot matches the description.

Birds such as the Little Owl, Kingfisher, Song Thrush, Blackbird Starling and Indian Myna, which are medium or large sized birds, also have to be searched...

The bird could be a Little Owl.
The Little Owl is too small.
The bird is not a Little Owl.

The bird could be a Kingfisher.
The Kingfisher is too small.
The bird is not a Kingfisher.

The bird could be a Song Thrush.
The Song Thrush is too small.
The bird is not a Song Thrush.

The bird could be a Blackbird.
The Blackbird is too small.
The bird is not a Blackbird.

The bird could be a Starling.
The Starling is too small.
The bird is not a Starling.

The bird could be a Indian Myna.
The Indian Myna is too small.
The bird is not an Indian Myna.

The bird(s) that match are :
adult Black Swan
male and immature Paradise Duck
overall Pukeko appearance

immature Australian Coot
adult Australian Coot

BIBLIOGRAPHY

AIKENS, J.S. 1979. Prototypes and Production Rules : An Approach to Knowledge Representation for Hypothesis Formation. *IJCAI* 6, pp. 1-3.

AIKENS, J.S. 1983. Prototypical Knowledge for Expert Systems. *Artificial Intelligence* 20, pp. 163-210.

BARR, A. and FEIGENBAUM, E.A. 1981. *The Handbook of Artificial Intelligence*. Volume 1. William Kaufman, Inc., California.

BOBROW, D.G. and COLLINS, A. (Eds.). 1975. *Representation and Understanding : Studies in Cognitive Science*. Academic Press, New York.

BOBROW, D.G. and WINOGRAD, T. 1977. Experience with KRL-0, One Cycle of a Knowledge Representation Language. *IJCAI* 5, pp. 213-222.

BOBROW, D.G. and WINOGRAD, T. 1977. An Overview of KRL, a Knowledge Representation Language. *Cognitive Science* 1, pp. 3-46.

BOBROW, D.G., KAPLAN, R.M., KAY, M., NORMAN, D.A., THOMPSON, H. and WINOGRAD, T. 1977. GUS, A Frame-driven Dialog System. *Artificial Intelligence* 8, pp. 155-173.

BRACHMAN, R.J. and SCMOLZE, J.G. 1985. The Overview of the KL-ONE Knowledge Representation System. *Cognitive Science* 9, No. 2, pp. 171-216.

COHEN, B. and BARROW, R. 1968. *Basic Acol*. Allen and Unwin, London.

DAVIS, R. and KING, J. 1977. An Overview of Production Systems. *Machine Intelligence* 8, pp. 300-332.

DAVIS, R., BUCHANAN, B,G., SHORTLIFFE, E.H. 1975. Production Rules as a representation for a Knowledge-based Consultation Program. *Artificial Intelligence* 8, pp. 15-45.

DUDA, R.O., HART, P.E., NILSSON, N.J., and SUTHERLAND, G.L. 1978. Semantic Network Representations in Rule-based Inference Systems. In Waterman, D.A., and Hayes-Roth, F. (Eds.). 1978, *Pattern-directed Inference Systems*, pp. 203-221. Academic Press, New York.

FALLA, R.A., SIBSON, R.B. and TURBOTT, E.G. 1978. *The New Guide to the Birds of New Zealand*. Collins, Auckland.

FEIGENBAUM, E.A., et al. 1971. On Generality and Problem Solving - A Case Study Involving the DENDRAL program. *Machine Intelligence* 6, pp. 165-190.

FINDLER, N.V. (Ed.). 1979. *Associative Networks, Representation and Use of Knowledge by Computers*. Academic Press, New York.

FIKES, R. and KEHLER, T. 1985. The Role of Frame-based Representation in Reasoning. *Communications of the ACM*, Volume 28, pp. 905-920.

GEORGEFF, M.P. 1979. A Framework for Control in Production Systems. *IJCAI* 6, pp.328-334.

HAYES, P.J. 1977. On Semantic Nets, Frames, and Associations. *IJCAI* 5, pp. 99-107.

HAYES-ROTH, F. 1985. Rule-based Systems. *Communications of the ACM*, Volume 28, pp. 921-932.

HEDRICK, C. 1976. Learning Production Systems from Examples. *Artificial Intelligence* 7, pp. 21-49.

HENDRIX, G.G. 1976. Expanding the Utility of Semantic Networks Through Partitioning. *Artificial Intelligence* 7, pp. 21-49.

LEHNERT, W. and WILKS, Y. 1979. A Critical Perspective on KRL. *Cognitive Science* 3, pp. 1-28.

LEVESQUE, H. and MYLOPOULOS, J. 1979. A Procedural Semantics for Semantic Networks. In Findler, N.V. (Ed.), *Associative Networks*, pp. 93-120. Academic Press, New York.

MARSHALL, KINSKY, and ROBERTSON, 1972. *Common Birds in New Zealand - Town, Pasture and Freshwater Birds*. Mobil New Zealand Nature Series. A.H. and A.W. Reed LTD, Wellington.

MINSKY, M. (Ed.). 1968. *Semantic Information Processing*. MIT Press, Cambridge, Mass.

MINSKY, M. 1975. A Framework for Representing Knowledge. In Winston, P.H. (Ed.), *The Psychology of Computer Vision*, pp. 211-277. McGraw-Hill, New York.

POST E. 1943. Formal Reductions of the Combinatorial Problem. *American Journal of Mathematics* 65, pp. 197-268.

QUILLAN, M.R. 1968. Semantic Memory. In Minsky, M. (Ed.), *Semantic Information Processing*. pp. 227-270. MIT Press, Cambridge, Mass.

QUINLAN, J.R. 1979. A Knowledge-based System for Locating Missing High Cards in Bridge. *IJCAI* 6, pp. 705-707.

RAPHAEL, B. 1968. SIR : A Computer Program for Semantic Information Retrieval. In Minsky, M. (Ed.), *Semantic Information Processing*. pp. 33-145. MIT Press, Cambridge, Mass.

ROBERTS, R.B. and GOLDSTEIN, I.P. 1977. The FRL Primer. *AI Memo 408*, MIT.

ROBERTS, R.B. and GOLDSTEIN, I.P. 1977. The FRL Manual. *AI Memo 409*, MIT.

ROSENBERG, S. and ROBERTS, B. 1979. Co-reference in a Frame Database. *IJCAI* 6, pp. 729-734.

RYCHENER, M.D. 1976. *Production Systems as a Programming Language*

for Artificial Intelligence Applications. Computer Science Department, Carnegie-Mellon University.

RYCHENER, M.D. 1979. A Semantic Network of Production Rules in a System for Describing Computer Structures. *IJCAI* 6, pp. 738-743.

SCHANK, R., and ABELSON, R. 1977. *An Inquiry into Human Knowledge Structures.* Lawrence Erlbaum Associates, Hillsdale, New Jersey.

SCHUBERT, L.K. 1975. Extending the Expressive Power of Semantic Networks. *IJCAI* 4, pp. 158-164.

SHORTLIFFE, E.H. 1976. *Computer-based Medical Consultations : MYCIN.* North-Holland, New York.

STANIER, A.M. 1975. BRIBIP : A Bridge Bidding Program. *IJCAI* 4, pp. 374-378.

STEFIK, M. 1979. An Examination of a Frame-structured Representation System. *IJCAI* 6, pp. 845-852.

VAN MELLE, W. 1979. A Domain-independent Production-rule System for Consultation Programs. *IJCAI* 6, pp. 923-925.

VERE, S.A. 1977. Relational Production Systems. *Artificial Intelligence 8*, pp. 47-68.

WATERMAN, A. 1976. Adaptive Production Systems. *IJCAI* 3, pp. 296-303.

WATERMAN, D.A. 1970. Generalization Learning Techniques for Automating the Learning of Heuristics. *Artificial Intelligence 1*, pp. 121-170.

WATERMAN, D.A., and HAYES-ROTH, F. (Eds.). 1978. *Pattern-directed Inference Systems*. Academic Press, New York.

WINOGRAD, T. 1975. Frame Representations and the Declarative/Procedural Controversy. In Bobrow, D.G. and Collins, A. (Eds.), *Representation and Understanding : Studies in Cognitive Science*, pp. 185-210. Academic Press, New York.

WINSTON, P.H. 1977. *Artificial Intelligence*. Addison-Wesley. Reading, Mass.

WINSTON, P.H. and HORN, B.K.P. 1984. *Lisp*. Addison-Wesley. Reading, Mass.