Knowledge Selection with ANNs

Dimitris Karagiannis [*] Franz J. Kurfeß

Heinz-Wilhelm Schmidt[†]

ICSI – International Computer Science Institute, Berkeley, CA 94704

August 1991

Abstract

The access to information contained in possibly large knowledge bases is a crucial factor in the usability of such a knowledge base. In this paper, we present a method to select information relevant for a query in knowledge bases where the information is represented in a rule-based way. An approach based on artificial neural networks is used to pre-select the set of relevant rules, thus facilitating the task of the inference mechanism by restricting the search space to be traversed considerably. In addition to the information contained in the query itself, data derived from the environment in which the query is situated is used to further trim down the search space. Sources for this derivation process are data about the task under investigation as well as the history of user interactions.

We refer to the first way of diminishing the search space via the query as *identification*; the second one is referred to as *adaptation*, since the selection process is adapted to the current task. The third one, taking into account the history of interactions between user and knowledge base, is called *prediction*, aiming at a possible prediction of the next query, or a subset of rules relevant for the next query.

An implementation of the artificial neural networks used for these tasks is based on ICSIM, a connectionist simulator developed at ICSI.

[*]On leave from FAW-Ulm, P.O. Box 2060,W-7900 Ulm, Germany; E-mail: `karagian@dulfaw1a.bitnet`

[†]On leave from: Inst.f. Systemtechnik, GMD, Germany; E-mail: hws@icsi.berkeley.edu

1 Motivation

Knowledge based systems, which deal with complex tasks like knowledge acquisition, learning and in general with expert knowledge, have to contain **Knowledge Selection / Knowledge Partitioning** abilities. A typical functionality of such methods should include the possibility to identify the relevant part of the knowlede base with respect to the current query, to interpret additional information about the user, as well as to make use of the the knowledge is structured in a certain representation language. The aspects considered, such as attention, interpretation and selection, are studied in models which until now have belonged more to the cognitive science area. To map and connect these models is not only a difficult task per se, but with traditional computing techniques it seems only feasible to a restricted degree because of inherent limitations in these techniques, e.g. the representation of uncertain, incomplete or inconsistent information.

After extracting the functional requirements of these models we outline their realization based on a specific computational approach. The techniques examined here for implementing the selection process are based on Artificial Neural Networks (ANN's). Then the theoretical and conceptual methods as well as the necessary prerequisites, are given.

Suppose we have a knowledge based application realized with logical rules. After a user-query is analysed the knowledge base objects are instantiated and the problem solver mechanism started. In this case a two-step approach as shown in Fig. 1 is suggested. First, the query is used to instatiate the application knowledge base, leading to an *instatiated KB subset.* This state transition is described by the f_ϵ function. To train the network, user interactions are observed to relate a certain query to a particular subset of the knowledge base via the *FoA net*. Then f_σ is applied to the instantiated KB-subset to identify promising ones among different variations of the reasoning process to resolve the query. This process will be supported by the adaptation net. On the example later, we will show the influence of the *FoA* net on the function f_σ during the selection of the relevant facts/rules.

The information directly related to the current query can be identified immediately and should be selected in any case. Furthermore, an adequate inference system can be activated according to the internal structure of the knowledge base. In this context we refer to all the available information as the **universe** $\mathbf{U}_0$. The process of identifying a subset of the knowledge-base is referred to as **knowledge selection**. The information which is selected during this process contains the **context-sensitive knowledge**. It should be active in a specific situation depending on the *FoA*, which is determined by the actual KB instance and, together with the KB instance, forms the **active universe $\mathbf{AU}_0$**.

A conceptual approach is to analyze and to define a framework supported by the **selection net**. In general, this goal can be broken down in two steps:

- A framework to transform the cognitive model – *focus of attention* FoA – to an operational model – *context sensitive knowledge* –, and
- the intergration of metaprogramming and connectionist methods by prototyping this approach.

The knowledge identification function f_σ can be activated to

- structure the related KB (query independent);
- analyze the instatiated KB subset (query dependent);
- apply the selection rules according to the instantiated KB subset, the knowledge representation structure and the available information from the *focus of attention* and *selection net*;
- use the inference mechanism to formally derive the results.

For this study, we concentrate on showing the usability of such nets concerning the *selection* function. Because their functionality is application-independent, they can be used universally for selections tasks. In this paper, it will be shown how useful it is to support the acquisition and learning process with the knowledge selection features in the context of the L_0 project.

$$\begin{tikzcd}
\textit{user query} \arrow[rr, "f_\epsilon"] \arrow[dr] & & \textit{knowledge base subset} \arrow[dl] \arrow[dr] & & \textit{knowledge base} \arrow[ll, "f_\sigma"'] \arrow[dl] \\
& \textit{focus of attention net} & & \textit{selection net} & \\
& \textit{training} & & \textit{after learning} &
\end{tikzcd}$$

Figure 1: A two step approach

L_0 is a connectionist natural language learning project at the International Computer Science Institute, led by Jerry Feldman [Feldman et al., 1990].

In chapter 2, an idea how the Focus of Attention and the selection net can be used for interpreting the corresponding knowledge base set is worked out by using L_0 as an application. Based on an L_0 example we demonstrate part of the **knowledge selection** functionality. Also some aspects are discussed how a realistic procedure for the representation of an active universe U_0 suitable for L_0 might look like.

2 The L_0 Example

In this chapter we consider the L_0 language as an application example [Weber and Stolcke, 1990]. The realized version of L_0 is characterized by a small fixed set of rules dealing with a few geometrical shapes, some spatial relations between these shapes and the structures of utterances used to describe a two-dimensional scene in this space in English. Consider a scene in which we restrict the space:
We only allow a subset of the possible shapes and a subset of the spatial arrangements. Then of course, not *all* rules would be needed to reason about scenes in this restricted space, and an intelligent knowledge-based system should be able to take advantage of this restriction by reasoning only in terms of the semantical structure of the subspace. We are convinced that neural nets suitably configured for this type of problem can learn to focus on the 'right' set of rules and maybe even other reasoning characteristics like the inference strategy or a few meta-rules relevant for reasoning well in the subspace.

In the example used the following assumptions are made for simplifying the explanation of what the *Knowledge Selection* process should do.
The L_0 Application

- The knowledge representation structure and the inference mechanism is like the one in Prolog;
- The *FoA* is related to the active predicates concerning the relevant situation;
- The learning process partly also means a modification of an existing knowledge set.

A Simple Example: Case a
A set of objects is defineed as obj-set:= (square, circle, triangle, cube)
Query: Selection of three Objects e.g. triangle, circle, square
Goal : Identify the needed rules for the L_0 interaction
Suggested approach: *Metarule*; identify the rules which **refer** to the selected objects
A Simple Example: Case b
The selected objects are placed on the screen in the follow sequence: triangle,circle,square
Query: The *FoA* is pointed at the square; the L_0 interaction deals with objects neighboring the square. Goal: Eliminate the rules which are related to the triangle without erasing them from the screen invisible-objects
Suggested approach: *Knowledge selection*; identify the rules which are **related** to the query, i.e. the square in this case.

An idea how to build up the needed Active Universe AU_0 is given through the knowledge selection approach in the following chapter.

3 Knowledge selection using connectionist methods

Background knowledge aboput a certain task to be solved certainly has an influence on the learning process. As background knowledge in a computational environment we define the accessible knowledge, independent of the kind of its representation structure. This information space is the universe U_0. A subset of U_0 which contains the information used in a given situation is referred to as the Active Universe AU_0. In this context the question is how to identify and select the information which AU_0 should include. If that is possible, then the question of an adequate representation and the appropriate abstraction level has to be discussed.

3.1 Approach

In the following, we assume that the information of the knowledge base is represented in the form of rules and encoded through an enhanced occurrence-position scheme similar to [Hölldobler, 1990]. The encoding scheme marks the occurrence of a literal at a certain position in the program and uses an additional dimension to represent alternative definitions of rules. The x axis, or *knowledge axis*, identifies the available information items. The y

axis, or *relationship axis*, defines the structural relations of knowledge pieces and their composition. The z axis, or *alternative axis*, describes the possible alternatives in the structural relations.

Let's consider an example: Suppose a rule-base system has three alternatives to prove a query. Then the x axis corresponds to the composition of rules from head and body literals, identifying the literals which belong to the body of a certain rule; the y axis simply lists the symbols associated with the literals of the rules, and the z axis represents the three different alternatives. Section 5 contains more details and a longer example.

From a knowledge selection point of view, we can differentiate three levels associated with an appropriate order relation:

- **Level$_S$:** The *order relation$_S$* defines the *structure level*
- **Level$_A$:** The *order relation$_A$* defines the *application level*
- **Level$_I$**: The *order relation$_I$* defines the *interaction level*

This approach uses three kinds of nets which correspond to the levels.

- **Structure aspects** are represented through *identification nets*
- **Application aspects** are represented in the second level through *adaptation nets*
- **Interaction aspects** are represented in the third level through *prediction nets*

The following table gives a short overview of these three levels with the type of inference mechanism needed, examples for the type of network, and the appropriate filter function.

Knowledge selection process			
Functionality	*Inference Mechanism*	*Network Type*	*Filter Function*
identification	passive	assoc. memory	FF_ι
adaptation	active	back prop.	FF_α
prediction	active	recurrent	FF_π

Table 1: Knowledge selection process

The input to the network is provided in the form of three vectors $\vec{s}; \vec{a}; \vec{i}$. They contain information about the structure of the knowledge, application aspects and the user interaction.

The knowledge selection approach based on a three dimensional space uses different filter functions. The basic idea behind this approach is how knowledge pieces can be used in different applications, assuming a compatible representation.

How the filter functions would be combined is defined separately for three different levels. These levels of information are represented with different neural networks techniques. The levels and the information space are orthogonal to each other. The operational characteristics of a level are given by functions which operate in a specific level configuration. The functionality between the levels is defined by different functions, the so called *filter functions.*

3.2 Methodology

The scope of methodology is to describe how the knowledge selection process can be used for getting the results. Primarily we focus in this section on the description of the steps essential for finding a solution. The follow steps are suggested:

- Step 1: Define the **order relations** for the different levels
- Step 2: Extract the **input vector** from the query
- Step 3: Select the **system-input** relevant alternatives
- Step 4: Activate the **connectionist nets** for the different levels
- Step 5: Combine the **filtering functions** level results
- Step 6: Evaluate results (learning)

The techniques which can be used for the first level are already worked out and based primarily on different *meta-knowledge* theories. This work concentrates on the second level in which *connectionist* methods are more appropriate. This can be done by a two step process:

- identifying an *order-relation*, and
- constructing the *topology-space.*

One major problem which should be considered is how we can abstract information about the dependency on the cognitive model, the sequence of queries or application-specific attributes. This information should be used for training the net and extracting the order relation about possible knowledge dependencies.

This knowledge is represented e.g. in the L_0 prototype in Prolog in the form of logical rules. Assume that there is a set of queries, knowledge base generic objects, connection relations and a set of order relations, which are equivalent to the selection methods.
The set of queries: $Q = \{\lhd, \rhd, \triangle, \bigtriangledown\}$
The knowledge base: $KB = \{\circ := rule, \bullet := fact\}$
The connection set: $C = \{\equiv := direct, \doteq := indirect\}$
The order relation set: $OR = \{\vdash :=$ knowledge representation, $\models :=$ meta-programming, $\bowtie :=$ connectionist$\}$

3.3 Functionality

The existing knowledge partitioning approaches can be classified in two categories, the "structure" and the "semantical" one. Technical processes like construction, modeling or planning are often ill-structured. Approaches to computer assistance in these processes often rely on a purely formal symbolic approach to the underlying structure of knowledge and the reasoning processes involved. Unfortunately the better part of the processes are ill-structured while any formal approach by its very nature is an attempt to fully or partly capture the underlying structures. For that reason it is widely recognized that rather than supporting the construction of a goal 'product', 'model' or 'plan', respectively, the steps towards this goal need be supported. The change from 'goal-oriented' to 'process-oriented' approaches accounts for this understanding.

Focusing on a step in a construction or modeling process we soon realize that its outcome is an object, too, which can be considered the 'goal' of the step, and a number of 'goal-oriented' approaches can be transformed for reusing them on this meta-level. For instance the object of meta-reasoning in a knowledge-based system is to determine the next steps to take and all the known inferencing strategies can be used if the meta-rules have the form of rules.

But in a more philosophical sense, the steps have become the goal, due to the ill-structuring of the problem. The term 'planing-by-opportunity' has been coined elsewhere to reflect this understanding. With each step taken in unison by man and machine the 'world has changed'. Not only has the position of the observer and her/his problem changed a little leading to a slight shift in the knowledge relevant to help in the next step, but also the knowledge base itself may have, in general, changed due to the last step. As a result whatever the previous strategy/plan for subsequent steps may have been, it needs to be reconsidered in the context of the new opportunities.

Identification

To simplify the discourse and to focus from the general philosophical problem to the more technical problem, we restrict our concern to the one of rule selection which is this:

Given a KB consisting of a large set of rules, and a current query, what is the subset of rules relevant to deal with this query? Obviously this can be generalized in various ways to include other characteristics of a reasonining process including cognitive ones. But already the restricted problem above gives rise to a number of interesting and to our knowlege unsolved problems.

- How to identify semantically relevant rules?
- How can semantical relevance be characterized in an ill-structured domain?
- What are appropriate techniques to extract and describe semantical relevance?

This answer to these questions will help to cut down the number of rules to avoid blind alleys. Such a cut-down may be acceptable, even if we are only reasoning in a proper subset of the semantically relevant rules. That is, the gain in speed of reasoning may justify loss of precision and certainty in reasoning if the pruning of the rule sets considered does not eliminate relevant rules too often. We call this part of the approach *selective* because it is mainly centered around methods for selecting a subset of the universe of rules.

Adaptation

In this context a neural network interacts with the reasoning component of a knowledge-based system. While the reasoning component applies knowledge according to its syntactical structure, the net selects promising rules based on information outside the knowledge base. Instead the user provides feedback on the acceptance of steps and/or rules. And this feedback is automatically used to teach the neural net to pre-select properly in future steps. Consequently the *adaptation* combines two complementary approaches, classical reasoning techniques and the adaptive character of connectionist networks. This gives rise to an "adaptive methodology" of reasoning or to "adaptive knowledge-based systems", for short.

Prediction

A simple extension of the adaptive approach is the following that we call the 'predictive approach'. The neural net component of the reasoning system not only adapts to the query context, but is designed such that it adapts to the query history and evaluates its own selection proposals. Various approaches towards connectionist nets with such a predictive character are known for time-series prediction, including nets part of whose output is fed back to their input and/or nets with modified error terms.

It is unclear whether nets are able to predict the relevance of rules for a small number of steps ahead and if so how these nets or their learning algorithms look like. But if the adaptive methodology can be extended to a predictive one this would be of high practical relevance, because it would allow for various techniques of rule prefetch and might help increase the efficiency of reasoning considerably.
At this point just a classification in which the approach is described:

- **define-prediction** based primarily on predefined requirements. Their functionality for the specific application properties is mostly already programming. Learning and adaptive aspects are totally missing; e.g. associative memory nets.

- **acquire-prediction** based primarily on experience, which is reflected by the learning aspect. This kind of prediction could be partially realized through non-connectionist methods (i.e. methods which work more on a symbolic level) and with ones based on connectionist models (subsymbolic level).

With the three techniques of identification, adaptation and prediction, the efficiency of an inference mechanism to extract information from large knowledge bases can be improved considerably by restricting the search space to be traversed.

4 ICSIM: A connectionist simulation tool

ICSIM is a simulator for connectionist networks developed at ICSI. Its underlying idea is to provide the user with a collection of basic building blocks for the construction of connectionist networks while leaving the door open to problem-specific modifications and additions. ICSIM has been implemented in a number of object-oriented languages; the implementation referred to here is done in SATHER a language also developed at ICSI as a derivative of EIFFEL geared towards simplicity and higer efficiency [Omohundro, 1990, Omohundro, 1991]. The major goals of ICSIM are the support of novel artificial neural network concepts with an emphasis on modularity, shared structures and leaning, the provision of simple means for the modification, extension and addition of networks and units, and the possibility of constructing networks both in an incremental (during experimentation) and non-incremental way. In ICSIM, the conceptual focus is shifted from units to nets, and form global and sequential execution to local and asynchronous execution with the potential of using parallel and/or dedicated hardware.

The development of networks in ICSIM is centered around two aspects: one is the internal structure and behavior of a network, described by a MODEL; the other stresses the objects and features a user sees and manipulates, captured by VIEWS. On one hand views have the character of objects as mediums through which the models are seen; on the other hand, they are the tools which the user has at hand to manipulate and interact with the models. This separation of views and models reduces the complexity of networks consisting of several levels of subnets with a large number of components and intricate connection patterns by filtering different aspects of a particular model through appropriate views. In addition, the amount and type of information about the status and interactions of thousands of components presented to the user must be minimized and centered on essential aspects.

4.1 Models

Models for connectionist networks are built from units and nets; the functionality of units and nets is characterized by their structure and their behavior. Important aspects of the structure are the composition of nets into components (which again can be nets, or units), the interconnections between components, and their state. Behavior is characterized by transitions from states to other states as result of computations, and by modifications of the states through learning.

Structure The components of a network in general are nets, or in the basic case, units. Thus nets and units share a common functionality as computational objects, expressed in the class COMP_OBJECT. This class provides the framework for the interconnections and computations of nets and units. The state of a network is characterized by the combined state of its components, and thus ultimately by the state information comprised in the units. The state of a unit is determined by an internal activation level, called *potential*, and an activation level visible from the outside, called *output*. The potential usually is derived from the inputs through a real-valued function, a common example being the weighted sum of the inputs. Certain types of units may have additional attributes like modes or phases, and may take into account information about previous states to compute the current state. The output is a function of the potential, e.g. a sigmoid or threshold function. The separation of potential and output results in a more flexible simulation where a unit can perform internal computations of its potential without affecting simultaneous computations of other units. This is helpful in the very frequent case of synchronous execution, where all units compute their potential in the same step, and then change their output.

The input signals a unit receives come from the outputs of other units via directed connections, possibly with an associated weight representing the strength of a connection; these weights are typically modified during learning. The interconnection patterns in a network can vary greatly, from fully connected networks to networks with selective connections between single units or subnets; very often, however, some regularity can be found in the interconnection pattern. ICSIM provides the concept of *sites* to group different types of connections, thus partitioning the structure of the network. In order to facilitate the description of a network structure on a high level, ICSIM includes objects encapsulating connection specifications that parametrize various connection procedures, e.g. x_connect for cross-connected nets (all units of one net are connected to all units of a second net), or bus_connect, where a component c_i of one net is connected to its counterpart c'_i in another net. These connection procedures can be modified, or new ones defined, according to the requirements of a particular application.

Behavior The behavior of a network is founded in the behavior of the units it ultimately consists of. The behavior of a unit is described in terms of steps, which internally consist of two phases. In the first phase, the potential is computed as a function of the inputs received; in the second phase, the potential is posted to the output and becomes visible to the connected units.

The temporal organization of the steps in a network shows a considerable variety, ranging from synchronous operation where all components perform one step after the other simultaneously, over fairly asynchronous operation (where the units can be a certain number of steps 'apart') to unconstrained asynchronous operation. In addition, there might be some degree of sequentialization in the network, as in layered networks, where one layer completes its step before the next one starts.

During learning, the the behavior of the network is modified through changes in the weights of the connections. These changes are invoked by error signals which represent the deviation of the actual output of a net from its target output.

4.2 Views

The different views of a network are defined through objects which present the model to the user and allow the interaction between the model (and its evaluation) and the user. In the text view, the user controls the evaluation of the model and observes the activities in the network through a textual presentation. Typical interactions are the creation of a new net, resetting a net, changing the operational mode like synchronization or step size. The information about the progress of the evaluation can be easily configured according to the particular application, but typically shows the current potential or output of a particular set of units. With the tour view, more information about the objects and their activities during evaluation can be extracted, together with the creation of objects and execution of routines for test purposes. Some more views which represent information about a network and its evaluation in a graphical way are currently being developed for the XWindows interface.

4.3 An Example: ICSIM Building Blocks for a Selection Network

As an example for the use of ICSIM, let's have a closer look here at some of the facilities ICSIM provides for the construction of networks. We will examine some basic classes which will be used later for the construction of the "identification network" in Section 5.2. The units required are simple boolean units, which compute the potential as sum of their weighted inputs and set the output to 1 if the potential is equal to or greater than a certain threshold; otherwise the output is 0.

The definition of the corresponding class of ICSIM is as follows:

```
class BOOL_UNIT is  SINGLE_SITE_SUM; BOOL_THRESHOLD;
end;
```

BOOL_UNIT inherits from SINGLE_SITE_SUM and from BOOL_THRESHOLD. The first parent defines its input connections and accumulation, it computes the weighted sum of its inputs. The second parent defines its squashing function, here a simple linear threshold of 0.5 producing an output between 0 and 1. Various other features are inherited, including features to reset and step, and to connect or disconnect the unit in its environment.

Via BOOL_THRESHOLD, BOOL_UNIT inherits from TWO_PHASE_UNIT whose main purpose is the separation of the accumulated potential and the output visible to other units. This separation also provides the basis for a partially sequential simulation of real parallelism: a number of units can `compute` their potential and then all of them can `post` their output without any of them "seeing" intermediate results of this very computation.

```
   output:REAL;
   post is -- assign some function of the state to the output
      output := unit_fn(potential) end;
```

Below we list the main public features that are supported by all units (class ANY_UNIT).

```
   -- With respect to connection, units behave like atomic nets.
   -- All net connection routines are supported.
   -- Most of them take a connection specification as argument,
   -- a small object used to pass connection information to the various
   -- levels of hierarchical nets. At the unit level for instance it
   -- tells which sites are used to connect to and it tells what
   -- kind of weight initialization is to be chosen.

   connect(from_u:$ANY_UNIT; c:$ANY_CON_SPEC) is ...
   connect_to(to_ob:$ANY_MODEL; con:$ANY_CON_SPEC) is ...
   bus_connect(from_u:$ANY_MODEL; con:$ANY_CON_SPEC) is ...
   disconnect(from_u:$ANY_UNIT) is ...
   connectedp(from_u:$ANY_UNIT):BOOL is ...

   -- Computation is split in two separate steps, that may or may
   -- not represent serial steps.
   step is compute; post end;
   compute is ...
   post is ...
```

For the family of net classes, ANY_NET specifies the general protoocl. The main aspects here are the behavior of a net, which is characterized through the different computation modes (synchronous and asynchronous), and the interconnection patters, where some frequent ones are defined. In some cases it is not really possible to fully define a feature since there are certain requirements which are not known yet.

```
   -- parallel computation modes are characterized by a bound for
   -- the number of steps that one unit may get ahead of others. 0 means
   -- lock-step, or 'synchronous' computation.

   sync_distance:INT; -- 0, synchronous

   init is
      sync_distance := 2; -- loosely asynchronous by default.
   end;

--- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

-- Structure

   size:INT is -- number of components
   component(i:INT):$ANY_MODEL -- a unit or net

-- Connection
   bus_connect(from_object:$ANY_MODEL; c:$ANY_CON_SPEC) is
      -- Connect element to element by index such that the current net
      -- can get input from 'from_net'. c can be used to customize
      -- the connection structure.

   x_connect(from_ob:$ANY_MODEL; c:$ANY_CON_SPEC) is
      -- Cross connect: recursively connect all pairs of components.
      -- c can be used to customize the connection structure.

   complete_connect(c:$ANY_CON_SPEC) is
      -- Cross connect to oneself.
-- Provide memorizable access to computation modes

   select_sync_mode is -- Switch to synchronous computation mode
   select_async_mode(sync_bound:INT) -- Switch to asynchronous mode

-- Computation

   micro_step(n:INT) is
      -- Do n steps, each component step counting 1,
      -- obeying the sync_distance constraints.

   serial_step is
      -- Serially: let components do their step in some natural order
      -- defined by the type of net.

   sync_step is
      -- In Synch: do a single step (all components).
      compute; post
   end;

   random_serial_step is
      -- In Random Order: do a single step (all components on the average)

   parallel_step is
      -- In Parallel: do a single step (all components on the average)
      -- obeying sync_distance.

end; -- class any_net
```

The simulator has a number of more specialized classes including a meaningful set of intermediate classes from which it is fairly easy to derive new classes for experimental purposes in simulation and/or learning research project.

5 Knowledge selection in rule-based systems

In this section we discuss the process of selecting the relevant rules and facts for a query based on an example. Figure 2 shows the initial set of rules and facts; such a set of rules and facts will be referred to as (logic) program, and a computation based on a logic program usually is initiated by a query. For our purpose here, it is sufficient to concentrate on the predicate symbols only, neglecting the internal structure of objects as described by terms.

```
R1 <== R2 & R3.
R2 <== R4 & R5.
R4 <== R7 & F4.
R3 <== R6 & R6 & F1.
R6 <== F2 & F3.
R7 <== F5 & F6.
R5 <== R7 & F3 & R3.

F1.
F2.
F3.
F4.
F5.
F6.
F7.
```

Figure 2: Rules and facts as program

The structure of the set of rules and facts can also be represented as a tree, using the appearance of a predicate in the body of a rule to establish the subtree relation. This leads to the tree shown in Figure 3.

The nodes in the tree are numbered in a depth-first, left-to-right way in order to facilitate a linear representation of the positions. This leads to the matrix encoding of the rules and facts shown in Figure 4. In this matrix, only the positions of predicates and their relations according to the initial rules and facts are given; their symbols and the information they may stand for are completely obsolete on this level.

The matrix presented in Figure 4 is the blueprint for the network which selects the relevant rules according to a query. As an example, let's consider the query `<== R2 & R6`. The task of the network is to select all the rules and facts which might be used to answer the query; regarding the tree, these are the subtrees with `R2` and `R6`, respectively, as root. For `R2`, we get the set {`R4, R5, R7, F4, F5, F6, F1, R3, R6, F2, F3`}; `R6` yields {`F2, F3`}, which actually is a subset of the first one.

The network contains internal connections from a positions to its successors, thus mirroring the structure of the set of rules and facts. In a

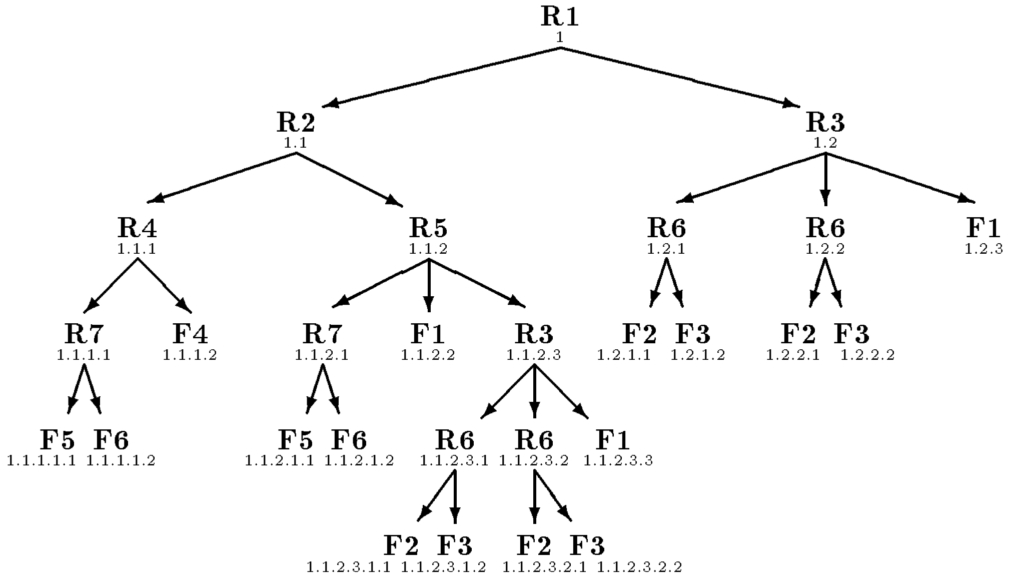

Figure 3: Rules and facts as tree

spreading activation scheme, the predicates initially used in the query, `R2` and `R6`, initiate the activation of the next two sets of predicates, {`R4, R5`} and {`F2, F3`}; the first set activates { `R7, F4`}, while the second does not propagate action any further since it consists of facts only. This continues until no more predicates can be activated. This spreading activation scheme corresponds to the computation of the transitive closure with respect to the logical implication, but in the reverse direction of the implication arrow.

5.1 Limitations

The scheme described above is only a rough outline, and suffers from a number of limitations. An obvious one is the size of the matrix required to represent the structure of the program. The size is determined by the number of different predicates in the programs, and how often they occur, so it is roughly quadratic with respect to the length of the program, measured as occurrences of predicates. With a *dag* (directed acyclic graph) representation instead of the tree, the size can be condensed considerably by representing each occurrence of a predicate only once; as a result, a node can have more than one incoming arcs. For our purpose here, this does not cause problems; if terms as arguments of predicates have to be taken inot consideration as well, we might have to differentiate between different occurrences of a predicate since its variables might be instantiated with different values.

Another limitation of the above scheme is the fact that it only makes sense to use it in situations where most queries activate only a relatively small subset of rules and facts. This is the case for applications like knowledge and data bases where the main purpose is to store large amounts of information in a structured way; it is not necessarily the case in theorem proving or logic programming where relatively few predicates may be densely interrelated to each other. In such a case it might be wiser to assume that most of the rules and facts will be used, and discard the ones which are not needed in a pre-processing phase through reduction techniques [Kurfeß, 1990, Letz et al., 1990, Bibel, 1987, Robinson, 1965].

Our above example has been constructed in a way that each activated predicate can result in the further activation of only one rule: there are no alternatives, or rules with the same predicate in the first position. This assumption has been made for the sake of simplicity, and certainly must be eliminated. Again we propose only a straighforward, not at all sophisticated solution: for each alternative definition of a rule, a separate matrix is constructed. The overhead for the representation of our program obviously increases considerably with the number of alternatives[1]; the separation of alternative solutions, however, allows us to treat them completely independent of each other.

Let us extend our example now by the alternative definition `R3 <== R4 & F5 & F6`. The enhanced tree is shown in Figure 3

The tree with alternatives actually does not show the whole picture: there is also the possibility to use the alternative definition of `R3` (designated by `R3'`) in `R5`. The addition of one alternative rule results in the appearance of four alternative solution candidates, which could be used as follows:

1. the first definition, `R3`, both in `R1` and in `R5`;
2. the second definition, `R3'`, both in `R1` and in `R5`;
3. the first definition, `R3`, in `R5` and the second definition, `R3'`, in `R1`;
4. the second definition, `R3'`, in `R5` and the first definition, `R3`, in `R1`.

[1]it does not only depend on the number of alternative rule definitions, but also on the number of occurences of the clause head literal

These four alternative solution candidates are represented in four different matrices. Alternative 1 already has been shown in Figure 4; the others are shown in Figures 6 - 8.

The result of the selection also gives us some clues to compare the required effort for the actual computation of a solution: one is the size of the set of selected predicates, the other the ration between the occurrences of predicates in rules versus the occurrences in facts. As a very broad rule of thumb, sets with fewer predicates might be 'easier' to solve, as well as sets with relatively many fact occurrences. This rule might be misleading, especially in the case of recursive programs. Recursive programs, however, are more apt to be found in theorem proving and logic programming applications, which are not th main goal of this proposal anyway. In addition, information like the rules involved in recursive parts can be derived from the matrix representation above, although it requires additional overhead.

The derivation of the network from the matrix representation is quite straightforward: a unit is used for each element of the matrix, and connections exist from each unit of a particular row to all the units of the rows which stand for sons of the father row; these connections are called *son connections*. Initially units are pre-charged if the predicate indicated by the column occurs at the position indicated by the row. In this state the network represents the structure of the knowledge base according to the relations between the occurrences of predicates in the heads and bodies of rules. For the selection process, the network requires additional connections from the input units (the predicate symbols) to all the units of the corresponding column, called *column connections*. A query is presented to the system by activating the predicate symbols occurring in the query. The column connections then activate all occurrences of these queries, and the son connections spread the activation to all predicates used in the body of rules whose heads appear in the query, and so on.

The network described here is very simplistic and inefficient; it is derived from a similar network used for connectionist unification [Hölldobler, 1990, Kohonen et al., 1991]. Its size is the number of predicate symbols times their occurrences, where each row only contains one entry. This is certainly not realistic for large knowledge bases, and a more condensed representation must be found. The functionality of the network is basically that of an associative memory, and optimization techniques from that area should be applicable.

Another inefficiency is induced by representing alternative solutions as different nets. Whereas it is conceptually very nice to treat alternatives independently, the overhead is prohibitive for realistic applications. It is important here that the number of networks required for alternatives is not just the sum of alternative rules; in addition, the occurrences of the predicate in the head of the rule must be taken into account. For one rule with n alternative definitions and m occurrences of the head predicate, the number of combinations is n^m, and for k rules the overall number of combinations is $\sum_{i=1}^{k} n_i^{m_i}$.

It is also questionable if a representation of knowledge as rules is adequate for all cases. The underlying scheme used to derive our network here is applicable to any representation which is based on items of knowledge and their relations, be they represented as rules, graphs, lists or some other way.

5.2 Identification Network

As an example, let us examine the ICSIM specification of a network[2] which recieves as input a set of rules appearing in the query, and identifies as output another set of rules which are relevant to solve the query; the output actually is not just the name of the rule, but information about the location of the rule (e.g. an address, or an entry to a hash table). Internally the network contains a matrix of units, the columns corresponding to the inputs, and the rows to the outputs. The units in the matrix can be in one of three states: present, absent and activated. An element is present if there is a relation between the input given by its column index and the output given by its row index; this indicates that a rule is associated with a certain location. A matrix unit gets activated if its input is part of the query, or if the rule corresponding to its input is relevant to solve the query. In the first case, the activation is pretty simple: If a unit is present, and it receives activation from the corresponding input unit, then it activates its output, which turns on the corresponding output unit. The necessary connections for this case go from a input unit to all the matrix units in the same column, and from each matrix unit to the output unit in the same row. For the second case, we need additional connections which indicate that a rule is relevant to solve the query. This information is contained in the structure of the rules and facts in the knowledge base: If a rule `R1` appears in the query, then all the rules and facts in its body, and all the rules and facts in the bodies of these rules, and so on, are relevant to solve the query. In our matrix, we encode this relation through additional connections from the head of a rule to all the parts of its body. There are anumber of ways to introduce

[2]The network described here is unnecessarily complex for our particular problem; it is used to demonstrate the facilities of ICSIM and is similar to the one used in [Kohonen et al., 1991], which takes into account the unification of term structures neglected here.

these connections into our network: between the input units, between the output units, between the rows of the matrix, and between the columns of the matrix. For demonstration purposes, let's use the connections between the rows of the matrix; this is also appropriate if the unification of term structures is taken into account as well, as in [Kohonen et al., 1991]. The first rule, for example, entails connections from the first row (location 1) to the second row (location 1.1) and to the twentyfirst row (location 1.2). These connections take care of the activation of rules R2 and R3 in the case that rule R1 has been activated.

```
class IDENTIFICATION_NET is
   ANY_NET;
   no_rules:INT; -- number of rules in the knowledge base
   no_facts:INT;    -- number of facts in the knowledge base
   no_positions:INT;     -- number of positions (nodes in the tree)
   inputs:NET1D{BOOL_UNIT};
   outputs: NET1D{BOOL_UNIT};
   matrix: NET2D{BOOL_UNIT};

   init is
      inputs.net1d_init_sized(no_rules + no_facts);
      outputs.net1d_init_sized(no_positions);
      matrix.net2d_init_sized(no_rules + no_facts, no_positions);
      set_occurrences; -- according to the knowledge base
      connect_inputs;
      connect_outputs;
      connect_rows; -- between head and body parts
   end; -- init

   crt_component(i:INT): BOOL_UNIT is -- called back by NET1D crt protocol
      res := BOOL_UNIT::crt;
   end;

   set_occurrences is
      -- set the potential of a matrix element to 'present'
      -- if the corresponding rule occurs at that position

   end; -- set_occurrences

   connect_inputs is
      -- connect element i of the input net to all matrix elements in column i

   end; -- connect_inputs
   connect_outputs is
      -- connect all matrix elements of row j to unit j in the output net

   end; -- connect_outputs

   connect_rows is
      -- connect all elements of row h to all elements of rows b1, ..., bn
      -- if h represents the head and b1, ..., bn the body parts of a rule

   end; -- connect_rows

end; -- class IDENTIFICATION_NET
```

5.3 Adaptation Network

The basic idea of adaptation is to use additional information from outside the knowledge base to narrow the number of relevant rules for a query. This information can be derived from the particular task being worked on, or from knowledge about the user (preferences, background).

5.4 Prediction Network

The information used for adaptation does not directly involve a notion of time; it just uses information presently available form outside the knowledge base. For prediction, time is very important. Here the information is derived form previous actions of the user, and again used to confine the number of relevant rules for a particular query. Short-term information (e.g. from the duration of the present session) can be used to focus on the context of the current task; long-term information may be derived from previous tasks treated by the same user or a group of users, and help to identify a particular area of interest. Neural network technology offers two techniques to take into account previous activities: one is *decay*, maintaining the activation of a particular node for some time and then decreasing it; the other is to identify *correlations* in the activations of groups of nodes. The first one modifies the state of a single node only, whereas the second can consist of strengthening or weakening connections between nodes.

6 Conclusions and Further Work

The central idea of this work is to facilitate the treatment of information contained in large knowledge bases in the form of rules and facts. We suggest a combination of neural network technologies and rule-based inference mechanisms. The neural networks are used to narrow the search space to be traversed by the inference mechanism. This is based on a three-step approach: First, the information in the knowledge base which is relevant for a query is identified according to syntactical criteria, basically corresponding to the computation of the transitive closure of the query. This step is realized by a network based on a matrix representation of the rules, facts, and their relations, which is functionally equivalent to an associative memory. Such a representation also opens the way to a treatment of semantical criteria in addition, e.g. the similarity of information in the knowledge base. For this purpose, networks like Kohonen's feature maps [Kohonen, 1990] can be used.

The next step is to take into account additional information available from the context of the query, aiming at an adaptation of the relevant information to the current task. Here we suggest the use of a network which learns to prefer certain rules to others, either through observations of previous activities, or through direct interaction with the user (unsupervised vs. supervised learning). Suitable architectures hera are backpropagation networks, or again feature maps.

The final step is to predict further actions (e.g. user queries) from the previous queries used to solve a particular task. In this step, temporal information must be learned, which is possible with recurrent networks, or networks incorporating delay components.

Considering a realization of the *selection* idea in knowledge based systems, the following activities/objectives have to be studied:

- Definition of the theoretical framework;
- approach how to design an operational *FoA* in an application;
- development of a prototype for this application and test it with the ICSIM simulator, and
- demonstration of results of the "knowledge-selection" approach by using the existing L_0-prototype.

The L_0 example belongs also to one of the possible applications which were suggested for the Applicable Artificial Neural Networks framework, short A^2N^2. This is a research proposal under preparation, in which three possible applications are discussed [Karagiannis and Schmidt, 1990]. L_0 has been chosen because the requirements which this language puts on the environment are adequate concerning the developement of the conceptual as well as the computational approaches.

Notice that if the *FoA* is not only related to the knowledge selection but also to the reasoning process over the selected knowledge, then this approach can only help in part. The next step is to improve the approach using paradigms from other areas, such as speech recognition by machine, engineering simulation and other semi-structured tasks.

Finally, in our opinion sheer computational power is not enough to achieve the semantic aspects requested for realizing the *FoA*. Therefore, we believe that this first step to develop a computational model, based on the **knowledge-selection process**, for realizing the *FoA* could be help to support the adequate knowledge environment, and guarantee situation-dependent and system response.

References

[Bibel, 1987] Bibel, W. (1987). *Automated Theorem Proving.* Vieweg, Braunschweig, Wiesbaden, second edition.

[Feldman et al., 1990] Feldman, J. A., Lakoff, G., Stolcke, A., and Weber, S. H. (1990). Miniature language acquisition: A touchstone for cognitive science. Technical Report TR-90-09, International Computer Science Institute, Berkeley, CA 94704-1105.

[Hölldobler, 1990] Hölldobler, S. (1990). A connectionist unification algorithm. Technical Report TR-90-012, International Computer Science Institute, Berkeley, CA 94704.

[Karagiannis and Schmidt, 1990] Karagiannis, D. and Schmidt, H.-W. (1990). The $A^2$2-Project: Applicable Artificial Neural Networks. Technical report, International Computer Science Institute, Berkeley, CA 94704-1105.

[Kohonen, 1990] Kohonen, T. (1990). Internal representation and associative memory. In Eckmiller, R., Hartmann, G., and Hauske, G., editors, *Parallel Processing in Neural Systems and Computers*, pages 177–182. Elsevier.

[Kohonen et al., 1991] Kohonen, T., Mäkisara, K., Simula, O., and Kangas, J., editors (1991). *International Conference on Artificial Neural Networks (ICANN-91)*, Espoo, Finland. North-Holland.

[Kurfeß, 1990] Kurfeß, F. (1990). *Parallelism in Logic — Its Potential for Performance and Program Development.* PhD thesis, Institut für Informatik, Technische Universität München. published as book by Vieweg Verlag, Wiesbaden (1991).

[Letz et al., 1990] Letz, R., Bayerls, S., Schumann, J., and Bibel, W. (1990). SETHEO - a high-performance theorem prover. *Journal of Automated Reasoning.*

[Omohundro, 1990] Omohundro, S. (1990). The Sather Language. Technical report, International Computer Science Institute.

[Omohundro, 1991] Omohundro, S. (1991). Differences between Sather and Eiffel. *Eiffel Outlook.*

[Robinson, 1965] Robinson, J. (1965). A machine-oriented logic based on the resolution principle. *Journal of the ACM*, 12:23–41.

[Weber and Stolcke, 1990] Weber, S. H. and Stolcke, A. (1990). L_0: A testbed for miniature language acquisition. Technical Report 90-10, International Computer Science Institute, Berkeley, CA 94704-1105.

```
R1 R2 R3 R4 R5 R6 R7  F1 F2 F3 F4 F5 F6 F7

#                                               1
   #                                            1.1
         #                                      1.1.1
                  #                             1.1.1.1
                                    #           1.1.1.1.1
                                       #        1.1.1.1.2
                                  #             1.1.1.2
            #                                   1.1.2
                  #                             1.1.2.1
                                    #           1.1.2.1.1
                                       #        1.1.2.1.2
                        #                       1.1.2.2
       #                                        1.1.2.3
               #                                1.1.2.3.1
                           #                    1.1.2.3.1.1
                              #                 1.1.2.3.1.2
               #                                1.1.2.3.2
                           #                    1.1.2.3.2.1
                              #                 1.1.2.3.2.2
                        #                       1.1.2.3.3
       #                                        1.2
               #                                1.2.1
                           #                    1.2.1.1
                              #                 1.2.1.2
               #                                1.2.2
                           #                    1.2.2.1
                              #                 1.2.2.2
                        #                       1.2.3
```

Figure 4: Matrix representation of rules and facts

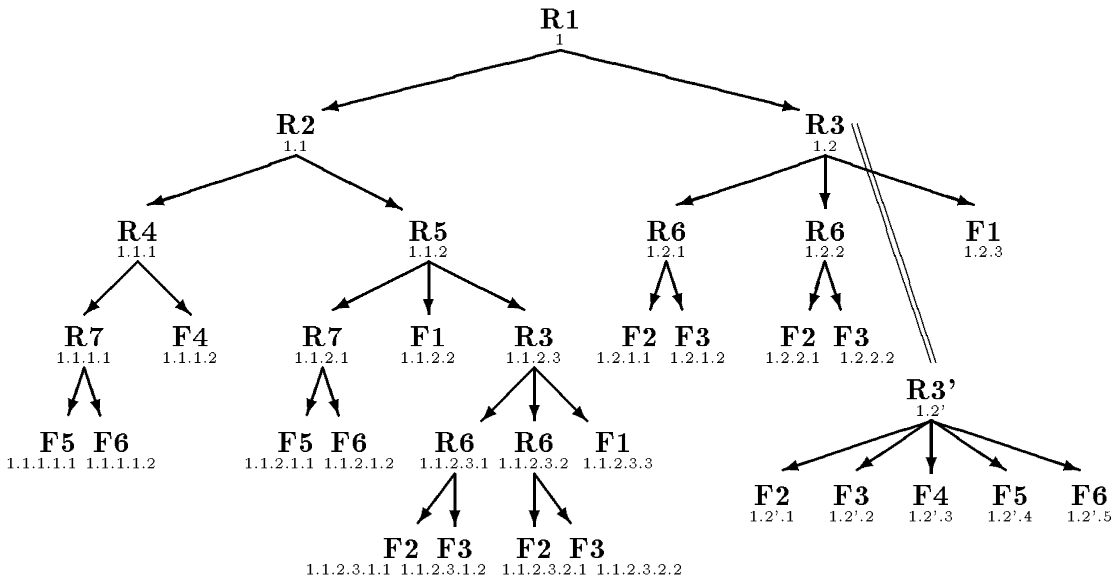

Figure 5: Tree representation with alternatives

```
R1 R2 R3 R4 R5 R6 R7  F1 F2 F3 F4 F5 F6 F7

#                                               1
   #                                            1.1
         #                                      1.1.1
                  #                             1.1.1.1
                                    #           1.1.1.1.1
                                       #        1.1.1.1.2
                                  #             1.1.1.2
            #                                   1.1.2
                  #                             1.1.2.1
                                    #           1.1.2.1.1
                                       #        1.1.2.1.2
                        #                       1.1.2.2
       #                                        1.1.2.3'
                           #                    1.1.2.3'.1
                              #                 1.1.2.3'.2
                                 #              1.1.2.3'.3
                                    #           1.1.2.3'.4
                                       #        1.1.2.3'.5
       #                                        1.2'
                           #                    1.2'.1
                              #                 1.2'.2
                                 #              1.2'.3
                                    #           1.2'.4
                                       #        1.2'.5
```

Figure 6: Matrix representation, Alternative 2

```
R1 R2 R3 R4 R5 R6 R7  F1 F2 F3 F4 F5 F6 F7

#                                               1
   #                                            1.1
         #                                      1.1.1
                  #                             1.1.1.1
                                    #           1.1.1.1.1
                                       #        1.1.1.1.2
                                  #             1.1.1.2
            #                                   1.1.2
                  #                             1.1.2.1
                                    #           1.1.2.1.1
                                       #        1.1.2.1.2
                        #                       1.1.2.2
       #                                        1.1.2.3
               #                                1.1.2.3.1
                           #                    1.1.2.3.1.1
                              #                 1.1.2.3.1.2
               #                                1.1.2.3.2
                           #                    1.1.2.3.2.1
                              #                 1.1.2.3.2.2
                        #                       1.1.2.3.3
       #                                        1.2'
                           #                    1.2'.1
                              #                 1.2'.2
                                 #              1.2'.3
                                    #           1.2'.4
                                       #        1.2'.5
```

Figure 7: Matrix representation, Alternative 3

```
R1 R2 R3 R4 R5 R6 R7  F1 F2 F3 F4 F5 F6 F7

#                                               1
   #                                            1.1
         #                                      1.1.1
                  #                             1.1.1.1
                                    #           1.1.1.1.1
                                       #        1.1.1.1.2
                                  #             1.1.1.2
            #                                   1.1.2
                  #                             1.1.2.1
                                    #           1.1.2.1.1
                                       #        1.1.2.1.2
                        #                       1.1.2.2
       #                                        1.1.2.3'
                           #                    1.1.2.3'.1
                              #                 1.1.2.3'.2
                                 #              1.1.2.3'.3
                                    #           1.1.2.3'.4
                                       #        1.1.2.3'.5
       #                                        1.2
               #                                1.2.1
                           #                    1.2.1.1
                              #                 1.2.1.2
               #                                1.2.2
                           #                    1.2.2.1
                              #                 1.2.2.2
                        #                       1.2.3
```

Figure 8: Matrix representation, Alternative 4