Data dictionary approach to the maintenance of expert systems: The Knowledge Dictionary

Bob Jansen* and Paul Compton†

The development and use of a Knowledge Dictionary, *a tool to facilitate the documentation and maintenance of rule based expert systems, is discussed. The Knowledge Dictionary may be used to record heuristics and their component parts, facts and rule actions, in such a way that a knowledge engineer, or end user, may determine the usage of any part of the knowledge, may easily add new parts, and may run the expert system to determine the effect of the maintenance. The Knowledge Dictionary utilizes the relational data model to store the heuristics in a data form rather than in an executable code form. Use of the relational model provides the knowledge engineer with all the power of relational calculus to interrogate the stored knowledge.*

Keywords: data dictionary, knowledge base, expert system, maintenance, design, database, database management system, modelling, heuristics, relational calculus

There are few knowledge-based systems today that have been in commercial production for more than a small amount of time[1]. Of these there is little documentation of the maintenance process, with the exception of Xcon/R1[2] and Garvan ES1[3,4]. However, there is no doubt that maintenance problems are widespread.

Xcon, an expert system for configuring DEC computers, has expanded considerably, and after four years in routine use knowledge addition was still involving four full time personnel[2].

Siratac[5], a cotton pest management decision support/expert system developed by the CSIRO Division of Plant Industry and the New South Wales and Queensland Departments of Agriculture, Australia, has been in routine use for a number of years. In this time, the knowledge base has expanded to incorporate new knowledge about the cotton plant, the pests, and the chemicals used to control those pests. This expansion has led to the knowledge becoming convoluted, leading to difficulties in maintenance. Siratac is currently being redesigned using software engineering tools and expert systems technology[6], and is discussed below.

Garvan ES1 is a medical expert system which provides automatic clinical interpretation of diagnostic reports from a pathology laboratory[4]. The report includes a brief discussion of what the results of the laboratory measurements mean, to advise the referring clinician. The current system is restricted to thyroid interpretation. It has been in production for three years, over which time the knowledge base has increased in complexity not due to carrying out new tasks, but merely in refining its existing knowledge base. It was introduced into routine use in mid 1984 when 96% of its interpretations were acceptable to domain experts. Currently, 99.7% of its interpretations are accepted and the rule base has doubled in size. Figure 1 provides an example of the growth of a single rule over this period, and provides a good illustration of the maintenance problem when a system is only being refined and not expanded.

The problem of maintenance is compounded in knowledge-based systems because of the increased complexity of knowledge over information, especially where the knowledge-based system is closely coupled with a conventional database system. Maintenance is still a large component of the software lifecycle for conventional systems, and will also be so for knowledge-based systems, as the experience with Xcon, Siratac and Garvan ES1 suggests. However, there are lessons to be learned from conventional systems maintenance which are applicable to the maintenance of knowledge-based systems.

We suggest that maintenance problems occur largely as a result of a lack of software engineering methodologies and tools used in the Artificial Intelligence (AI) area. We propose that many of the conventional software engineering tools used daily in the engineering of conventional database systems are also applicable to AI systems[7] (see also References 8, 9 and 10).

*CSIRO, Division of Information Technology, P.O. Box 1599,North Ryde, NSW 2113, Australia
†The Garvan Institute of Medical Research, St Vincents Hospital, Darlinghurst, NSW 2010, Australia

0950–7051/89/010014–13 $03.00

1984

```
RULE(22310.01)

if (bhthy or utsh_bhft4 or vhthy)
   and not on_t4
   and not surgery
   and (antithyroid or hyperthyroid)
 then DIAGNOSIS(" . . . consistent with thyrotoxicosis")
```

1987

```
RULE(22310.01)
if ((((T3 is missing) or (T3 is low and T3_BORD is low))
   and TSH is missing
   and vhthy
   and not (query_t4 or on_t4 or surgery or tumour
   or antithyroid or hypothyroid or hyperthyroid))
   or(
   (((utsh_bhft4 or
   (hithy and T3 is missing and TSH is missing))
   and (antithyroid or hyperthyroid))
   or
   utsh_vhft4
   or
   ((hithy or borthy)
   and T3 is missing
   and (TSH is undetect or TSH is low)))
   and
   not on_t4 and not (tumour or surgery)))
   and (TT4 isnt low or T4U isnt low)
then DIAGNOSIS(" . . .consistent with thyrotoxicosis")
```

Figure 1. Garvan ES1 single rule expanding due to the refinement of knowledge over time

GARVAN ES1 DICTIONARY

Data dictionary

The *data dictionary* is a concept developed to aid in the design, maintenance and documentation of conventional database systems[11]. In conventional systems, the data dictionary is used as the central repository for all design information for the system because conventional systems have grown so complex that it is difficult for any one person to understand the complex inter-relationships between the separate objects that comprise the system[12].

As discussed by Jansen[12] (and shown in Figure 2), the dictionary is used to store the business model of the system(s) and the computer implementation (or model) of the systems, in addition to maintaining links between objects in both these worlds to enable the exploration of the relationships between the system objects from the business and computer worlds. The data dictionary is increasingly recognized as *the* tool for systems design[11]. (Details about ANSI standards for the Information Resource Dictionary System, and projected savings to be made by the use of dictionary technology are given by Dolk and Kirsch[13].)

Knowledge dictionary

In knowledge-based systems, especially where those systems are coupled to conventional database systems, the maintenance problem is more involved because of the greater complexity of knowledge over information. The complexity is shown in Figure 3, where it may be seen that information is based on data, and knowledge is based on both information and data. Information may be viewed as the relationships between items of data, and knowledge as relationships between items of information, possibly incorporating data.

This may be illustrated in conventional systems by a database, where the *data* comprises the actual items of data stored in the database, *information* includes the schema, describing the relationship among the data, or the relational tables in a relational database, and *knowledge* includes the relationships among the schema items, e.g. validation rules, constraints, etc.

To aid in the control and understanding of this increasingly complex environment, tools are mandatory to aid the knowledge engineer to acquire the domain knowledge, perform efficient system(s) design, and perform the maintenance and documentation tasks required in such a way that they do not leave inconsistencies, corruptions, logic errors, etc.

An extension to the data dictionary concept has been developed to integrate and cater for knowledge-based systems' maintenance and documentation, a system known as *Knowledge Dictionary*.

There is much research currently taking place to couple expert and database systems, some of it using the dictionary concept, but this proposal differs from other similar proposals (e.g. References 8, 13–17) in that it is proposed to apply the dictionary concept to knowledge as well as to the information/data areas, and to store the rules as data rather than as directly executable code.

This system differs from existing object-oriented products such as NEXPERT OBJECT and SMALLTALK in that these products are still essentially stand alone applications, from a designers' point of view: they do have gateways to common database managers, but systems cannot be designed, documented and maintained in an integrated fashion, e.g. NEXPERT OBJECT has

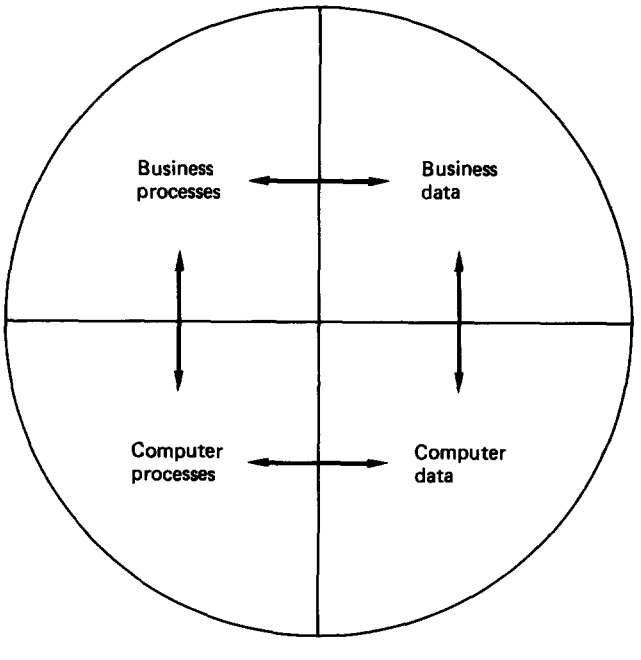

Figure 2. Data dictionary quadrants

a built-in interface to the Relational Database Manager (RDB) running under the DEC operating system VAX/VMS, but there is as yet no method for capturing the RDB database structure in NEXPERT without re-keying. This duplication of definitions then echoes throughout the length of the maintenance period. The proposal outlined here is for a knowledge dictionary acting as the central pivot for integrated design utilizing various software tools.

Similarly, these types of tools appear to address the knowledge acquisition phase of a project, but quite different capabilities may be required in the maintenance phase[3].

The knowledge dictionary has the equivalent functionality of a data dictionary for conventional systems, but is augmented to allow:

- the documentation of the knowledge base, analogous to the database;
- the validation of the total system(s), including the knowledge base component, utilizing the design information stored in the dictionary;
- the browsing of the knowledge to aid in the maintenance process;
- the provision of a maintenance environment through the inclusion of an inference engine that is able to process the rules stored as data; and
- the generation of the run-time knowledge base in a selected formalism, e.g. Prolog, frames, production rules, etc.*

Proposed model

The Garvan ES1 knowledge dictionary is based on the *Entity-Relationship* (ER) conceptual model (see Figure 4).

The conceptual model details the *object types* recognized by the dictionary, and the relationships between the object types, called *allowed relationships*. In the proposed dictionary, this conceptual model is user definable, as the schema used to store this model is a meta view of this model. Thus, the user can add any *object types/allowed relationships* to this model to cater for other requirements (causal models, fuzzy logic, temporal reasoning, etc.). (Note that this model is complete only in the area of the knowledge base and its interaction with conventional components.)

Further research is under way in the CSIRO Division of Information Technology Knowledge Base Engineering group to produce a dictionary model that is fully self-referential, and where the actual schema is a meta-meta view of this model[18]. This allows the schema of this model to be user definable, thus increasing the functionality of the dictionary as a whole. Therefore, the proposed dictionary can be viewed as a number of levels, each definable by (or visible to) a certain class of user.

The majority of this model is standard for a conventional data dictionary. The extensions which have been made are highlighted as the shaded entities and their relationships to other model entities†. It should be noted

*This function should be similar to that found with fourth generation programming systems, where after describing the application in detail, the generation of the run-time code is automatic and generally algorithmic.

†Note that in Figure 4 no attempt has been made to classify each entity in respect to membership of a particular dictionary quadrant.

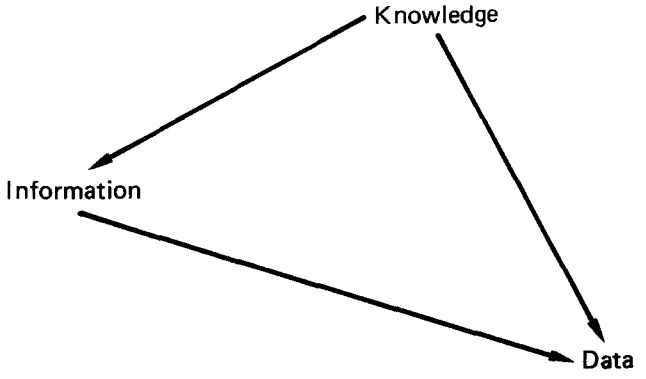

Figure 3. Taxonomy of knowledge, information and data

that the diagram shows what relationships an object type *may* make with its surroundings, called *allowed relationships*. The actual relationships for any object occurrence depend on its usage. In addition, the nature of the relationship is not shown on the diagram.

A relationship could be pointer- or set-based, as in a Codasyl database, relational- or value-based, as in the relational data model, or even function-based, where the membership of a relationship is dependent on the evaluation of some function, returning a true or false condition as appropriate. In each case, the relationship has properties defining the relationship type. For set based relationships, properties would include sort sequence and keys for sorted sets, set order, connect and disconnect requirements as mandatory or optional, etc.

The idea underlying the models is to treat each object type as a set of data or a table, thus allowing standard data manipulation operations upon it, e.g. as the knowledge dictionary is implemented in Prolog using the *Relational Data Model*[19], all the standard relational operators (union, intersect, difference, select, project, join, divide and HAS[20]) are potentially available to the knowledge engineer to browse and maintain the knowledge. As shown below, the use of the relational operators on the data representation of the rules allows a rich browsing and exploration capability. This is not normally available for an expert system. We hypothesize that this has occurred because attention has been focussed on the initial knowledge acquisition, and suggest that it is essential in the maintenance phase, where the knowledge engineer and the expert may no longer be intimately familiar with the system's knowledge, that flexible enquiry and exploration capabilities be available to ascertain the area and the effect of proposed maintenance. In addition, by adopting this approach the rules may be browsed, displayed, or entered using the commonly used **if...then** formalism, while hiding the run-time formalism from the user. It is anticipated that with the dictionary in place, other formalisms for representing, and thereby examining, knowledge will emerge.

A knowledge base, analogous to a database, can be considered as an instance of a knowledge base object stored in one or more files. The knowledge base may be subdivided into a number of worlds, or tasks, or be a collection of individual rules, each subset possibly stored in a different file, as in conventional database areas.

Each task is a collection of rules, one of which may be the trigger for another task to become active. The subdivision into tasks may require the use of a task precondition, as in OPS5, and therefore each task may have an associated task precondition. This is differentiated from the rule, as this is an implementation issue, and generally not required by the expert when viewing the knowledge. This is highlighted in Figure 5, where the item in italics is the task precondition, added to the implemented version but not mentioned in the expert's verbalization of the rule. (See Jansen[12] for a more detailed discussion involving this rule example.)

Rules themselves have a substructure, as shown in Figure 6. As shown in Figure 4, rules test for the presence or absence of a set of facts, and if the fact profile is matched then a set of rule actions are performed. Each rule action may assert a fact, retract a fact, display some data item, call a code module, or access a data record in some data store. It should be noted that the model enforces disjoint (or disjunctive) normal form on rule structure. Thus, any fact profile embodying OR conditions between individual facts must be separated into separate rules. This ensures that each rule is simpler to understand, in addition to excluding the problems associated with mixing NOT and OR conditions. However, the underlying dictionary allows groups of rules with the same action to be examined in concert.

While working in the area of facts and their meaning, it became clear that in most production systems facts can be considered as belonging to a *fact taxonomy*. The first classification is *simple* or *complex*. As illustrated in Figure 6, a complex fact consists of two facts related by an operator of AND or AND NOT, e.g. the complex fact *hithy_normal_tsh* in the Garvan ES1 thyroid domain relates the two 'simpler' facts *hithy* and *tsh is normal*. The latter fact is asserted if the blood sample *tsh level* is within the normal range, whilst the *hithy* complex fact is asserted by one of several rules comparing several

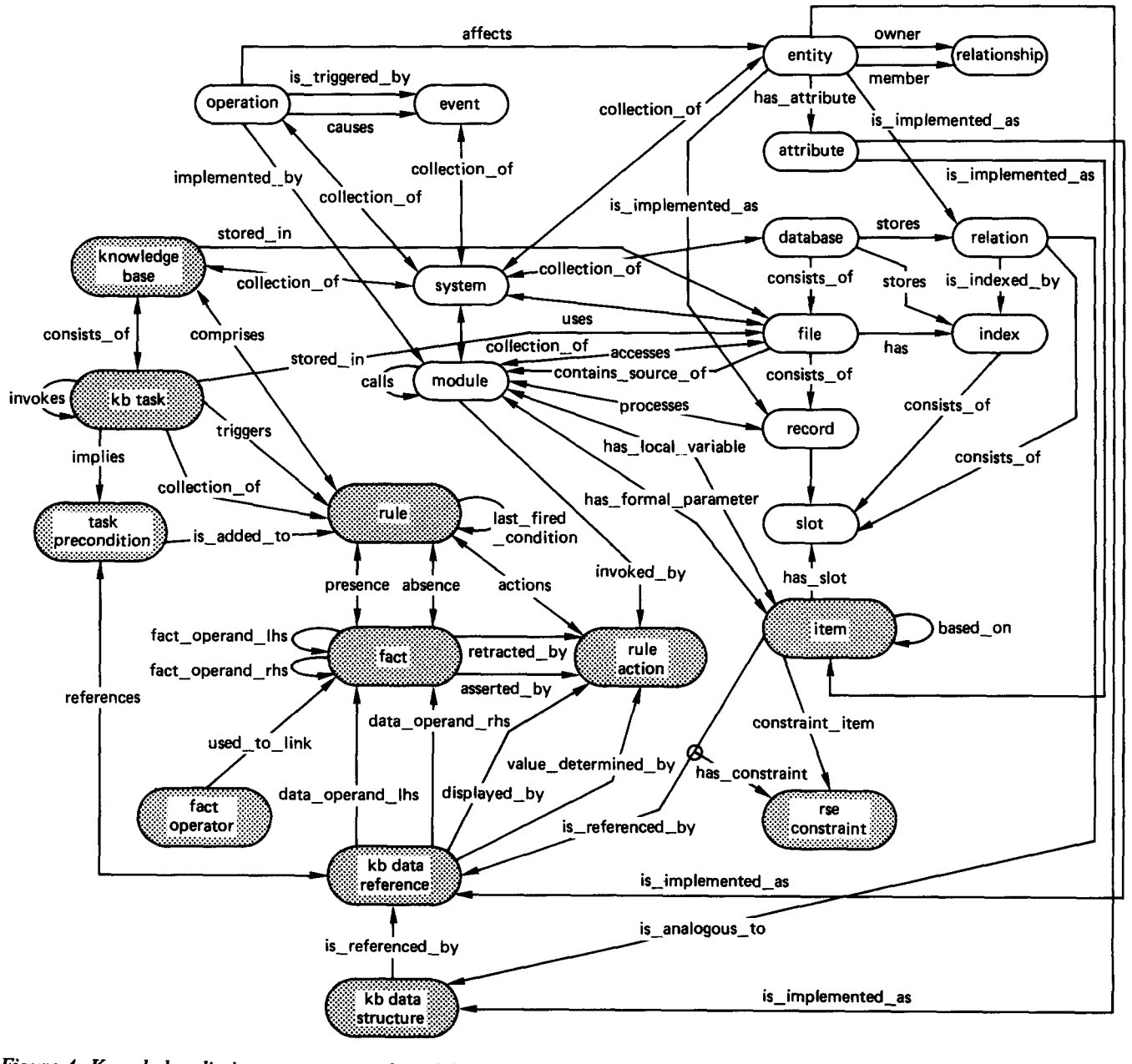

Figure 4. Knowledge dictionary conceptual model

Expert's verbalization of the rule

A spray will be required if heliothis brown eggs are over threshold for two days out of three.

OPS5 implementation of the rule

```
(p to_spray::brown_eggs_2_days
  (task ^is whether to spray)
  (threshold    ^pest  heliothis_brown_eggs
                ^level <thold>)

  -(spray_required)

  (pest_count)  ^pest  heliothis_brown_eggs
                ^day   <day1>
                ^level { <eggs_1> >= <thold> })

  (pest_count)  ^pest  heliothis_brown_eggs
                ^day   { <day2> <> <day1> }
                ^level { <eggs_2> >= <thold> })

  (make spray_required)
)
```

Figure 5. Sample OPS5 rule showing tasking (taken from the SIRATAC PLUS expert system)

Rule	::=	**if** antecedent **then** consequent
Antecedent	::=	fact_list
Consequent	::=	rule_action_list
Fact_list	::=	[operator] fact
	::=	fact[operator fact_list]
Fact	::=	simple_fact
	::=	complex_fact
Simple_fact	::=	data_item exists
	::=	data_item does not exist
	::=	data_item fact_operator data_item
Complex_fact	::=	fact operator fact
Fact_operator	::=	+ (addition)
	::=	− (subtraction)
	::=	* (multiplication)
	::=	/ (division)
	::=	> (greater than)
	::=	>= (greater than or equal to)
	::=	< (less than)
	::=	=< (less than or equal to)
	::=	= (equal to)
	::=	<> (not equal to)
	::=	contains
Operator	::=	AND
	::=	AND NOT
Rule_action_list	::=	rule_action[AND rule_action_list]
Rule_action	::=	assert fact
	::=	retract fact
	::=	display data_item
	::=	call module
	::=	determine value of data_item

Figure 6. Rule structure and substructure

blood hormone levels against threshold values for individual classes (e.g. see Figure 7). Thus, the 'simpler' facts may in themselves be complex.

The structure can be fully determined by working down each level until further decomposition is impossible. At this level, the leaf nodes are termed the simple facts. In most cases, simple facts will also be classed as *external*, where the validity of the facts is dependent on data values in the external, or real, world. *Internal* facts are those facts that are asserted by a heuristic within the knowledge domain. Thus, in our example, the fact *hithy* would also be classed as an internal fact.

In the knowledge dictionary, this taxonomy is stored as follows: the fact object is given a property of *operator*, which stores the operator used to relate the member facts or data items; complex facts are related to their 'simpler' facts by the *fact_operand_lhs* and *fact_operand_rhs* relationships; simple facts are related to their data items via the *data_operand_lhs* and *data_operand_rhs* relationships (see Figure 4).

Thus, by starting at the top level fact and storing the operator and the appropriate relationships, the complete taxonomy can be stored one level at a time. When requested, the structure can be evaluated or displayed by traversing the relationship linkages starting from any specified fact and applying or displaying the stored operator. In the case of simple facts, the appropriate data item may have to be retrieved from the data store, and so standard gateways should be invoked automatically.

This raises the question why have complex and internal facts? Why not replace their occurrences by the leaf nodes facts? The choice of using these fact types is dependant on the expert's visualization of the knowledge. If the heuristics included these higher level descriptions, then it is our belief that they should be included in the knowledge base. This maintains the expert's familiarity with the encoded knowledge. In addition, Clancey[21] introduced the concept of abstraction in the heuristic classification process using the Mycin expert

RULE(10500)
if FT4 is missing and
((FTI is high and TT4 is high) or
(FTI is high and TT4 is missing) or
(FTI is missing and TT4 is high))
then *hithy* NOW TRUE

RULE(10510)
if FTI is missing and
((FT4 is high and TT4 is high) or
(FT4 is high and TT4 is missing) or
(FT4 is missing and TT4 is high))
then *hithy* NOW TRUE

RULE(11100.01)
if T3 is high
and *hithy*
and TSH is missing
and TBG isnt high
and T4U isnt high)
and (not on_t4 surgery)
then DIAGNOSIS
("The T3 and THY are elevated consistent with thyrotoxicosis")

RULE(11110.02)
if T3 is high
and *hithy*
and TSH is missing
and (TBG is high or T4U is high)
and (not on_t4 or surgery)
then DIAGNOSIS
("The T3 and THY are elevated consistent with thyrotoxicosis and elevated binding protein")

RULE(11200)
if T3 is high and *hithy*
and (TSH is undetect or TSH is low)
then ht3t4_utsh NOW TRUE

Figure 7. Data abstraction process for the internal fact (or concept) hithy

Initial rule

RULE(11100.01)

if T3 is high
and hithy
and TSH is missing
and TBG isnt high
and T4U isnt high
and surgery
and not on_t4

then DIAGNOSIS("The T3 and THY are elevated consistent with thyrotoxicosis")

Constituent objects

a rule object named 11100.01

a number of fact objects, named
t3_high
hithy
tsh_missing
tbg_not_high
t4u_not_high
surgery
on_t4

a rule action object given an internal name of *interpretation_1* having as a *property* the string to display

Figure 8. Example rule showing its decomposition into constituent objects

system as an example. The abstraction concept is analogous to the complex and internal fact types, in that they represent a fact context of some complexity, but simplify the individual heuristics involved. The use of internal and/or complex facts is, in our opinion, justified on either of these grounds, e.g. Figure 7 shows the rules used to assert the abstract concept *hithy* in the Garvan ES1 expert system. If the rules 11100.01, 11110.02 and 11200 had their *hithy* fact replaced by the fact profiles from rules 10500 and 10510, the knowledge would become more complex — the 'wood for the trees' syndrome. Thus, for the sake of clarity, the *hithy* complex internal fact is used.

Implementation details

The prototype knowledge dictionary has been built in AAIS Prolog running on a Macintosh Plus/SE computer with 2.5 Mbyte of memory and a 20 Mbyte hard disc. Currently, the prototype knowledge dictionary contains approximately 624 knowledge domain rule objects, 217 fact objects, 185 rule action objects and 1940 tuples in the element_relationship table,constituting the re-implementation version of the Garvan ES1 thyroid interpretation expert system in disjoint (or disjunctive) normal form.

As stated above, the rules are stored as data using the relational model formalism. The following details how the rules are stored in the knowledge dictionary schema.

Take as an example the rule from the existing Garvan ES1 thyroid expert system, as shown in Figure 8. Prior to inserting this rule in the knowledge dictionary, the structure of the rule needs to be elicited. In this case, we have the constituent objects as shown in the lower

```
element(rule,'11100.01').
element(fact,t3_high).
element(fact,hithy).
element(fact,tsh_missing).
element(fact,tbg_not_high).
element(fact,t4u_not_high).
element(fact, surgery).
element(fact, on_t4)
element(rule_action,interpretation_1).
element(kb_data_reference,interpretation_1).
element_property(kb_data_reference,interpretation_1,
  kb_data_value,"The T3 and THY are elevated consistent
with thyrotoxicosis")
element_relationship(presence,rule,'11100.01',fact,t3_high).
element_relationship(presence,rule,'11100.01',fact,hithy).
element_relationship(presence,rule,'11100.01',fact,tsh_miss-
  ing).
element_relationship(presence,rule,'11100.01',fact,tbg_not_
  high).
element_relationship(presence,rule,'11100.01',fact,t4u_not_
  high).
element_relationship(presence,rule,'11100.01',fact,surgery).
element_relationship(absence,rule,'11100.01',fact, on_t4).
element_relationship(actions,rule,'11100.01',rule_action,inter-
  pretation_1).
element_relationship(displays,rule_action,interpretation_1, kb_
  data_reference,interpretation_1).
```

Figure 9. Example of object and relationship storage in the Knowledge Dictionary

section of Figure 8. Note that the naming convention in the knowledge dictionary model is based on a unique type/name doublet for any object. Thus, as in this case, objects of different types may have the same name. These objects would be stored as shown in Figure 9.

With the need to name each object we import the problem of naming conventions and 'meaningfulness'. From the above, it can be seen that if the names of the objects are meaningful in their own right, descriptions may not be required. However, they may be entered for each object occurrence to improve understanding. Similarly, the names of the allowed relationships from Figure 4 forming the first parameter in the element_relationship table are hopefully meaningful, so the purpose of the relationship is clear. The allowed relationship object occurrences may also have associated descriptions if required. In fact, any object occurrence may have an associated description. These descriptions are stored in flat files accessible through the standard AAIS Prolog 'edit window' facility (see the section below on the *Help* function).

Available functions

Using the above data declarations, a number of functions have been made available to manipulate the data. The more important of these functions are outlined below (in the accompanying figures, entries in italics show user input).

Add

The Add function allows for the addition of any occurrence of an allowed object type (see Figure 10). The result of the Add is the assertion of the valid object occurrence in the knowledge base if it does not already exist. This enforces the concept of each object type occurrence defined only once.

```
?add(fact,a_test_fact).
   yes
?add(fact,a_test_fact).
The fact a_test_fact has already been asserted.
Add aborted.
```

Figure 10. The Add function

```
?-usage(rule_action,interpretation_1).
Finding the usage in the knowledge domain

Relationship name - displays
Owner element type - rule_action
Owner element name - interpretation_1
Member element type - kb_data_reference
Member element name - interpretation_1

Relationship name - actions
Owner element type - rule
Owner element name - 11100.01
Member element type - rule_action
Member element name - interpretation_1
```

Figure 11. The Usage function

Usage

The Usage function is used to determine who uses what and how. It searches the *element_relationship* table extracting any tuples satisfying the criteria, and displays the result to the user (see Figure 11). This shows that the rule action *interpretation_1* is actioned in rule 11100.01, and the function of this rule action is to display the data item *interpretation_1*.

Similarly, the usage enquiry will display which rules assert or retract internal facts, thus identifying those rules that give the meaning of an internal fact. The rules 10500 and 10510 shown in Figure 7 describe when the fact *hithy* is true. Further meaning may be obtained by viewing the object's description file.

Show_rule

The Show_rule function displays the specified rule on the user's terminal in the familiar and accepted **if...then** form (see Figure 12). Note that although the stored form of the rule does not explicitly mention the actual text to be displayed, this function retrieves and displays the actual text followed by the internal rule action name in parentheses.

Add_rule

The Add_rule function allows the user to add a new rule specifying existing facts and rule actions. The function checks that the rule does not already exist, and that all specified facts and rule actions are known. If any are not known, the user is informed (see Figure 13). Note that since the action of the rule is to display an interpretation, the show_rule function extracts the actual text to be displayed.

Link_rule

The Link_rule function allows the user to link facts and rule actions to existing rules. The rules, facts and rule action occurrences must all be asserted in the knowledge base prior to calling this function (see Figure 14). Note

```
?- show_rule'11100.01'.

Rule - 11100.01

if
          t3_high
and       hithy
and       tsh_missing
and       surgery
and       t4u_not_high
and       tbg_not_high
and not   on_t4

then
  display "The T3 and THY are elevated consistent with thyro-
  toxicosis."
    (interpretation_1)

End of rule - 11100.01
```

Figure 12. The Show_rule function

```
?-add_rule testrule.

Please enter the names of facts used in this rule testrule.
The facts may be preceeded by one of the operators "and"
or "not".
The default is "and".
Type a ? to see a list of currently asserted facts.
When finished, type "end fact" on a new line.
lothy
t3_low
surgery
endfact
Input the rule action names one line at a time,
Type a ? to see the list of currently asserted rule actions ending
with a line starting "end rule action"
interpretation_24
end rule action
   yes
?-show_rule testrule.

Rule - testrule
if
       lothy
and    t3_low
and    surgery

then
     display "Low THY and T3 consistent with surgery."
     (interpretation_24)

End of rule - testrule
```

Figure 13. The Add_rule function

that no logic checking is done to the rule at this stage. It should also be noted that since the action of the rule is to display an interpretation, the show_rule function extracts the actual text to be displayed.

Help

As stated above, any object occurrence in the dictionary, be it a dictionary object or a knowledge domain object, may have associated textual description to allow the user to determine further the meaning of the object, if the name, or perusing the asserting rule, does not provide this information to their satisfaction. As stated above, this information is held in a standard text file, and may be accessed via the standard AAIS Prolog edit window facility. The display is in a Macintosh window

```
?-link_rule testrule.

Please enter the names of facts used in this rule testrule.
The facts may be preceeded by one of the operators "and"
or "not". The default is "and".
Type a ? to see a list of currently asserted facts
When finished, type "end fact" on a new line.
not t4_high
not tsh_high
end fact
Input the rule action names one line at a time,
Type a ? to see the list of currently asserted rule actions.
ending with a line starting end rule action

end rule action

  yes

?-show_rule testrule.

Rule - testrule
if
            lothy
and         t3_low
and         surgery
and not     t4_high
and not     tsh_high

then
    display "Low THY and T3 consistent with surgery."
      (interpreteation_24)

End of rule -testrule
```

Figure 14. The Link_rule function

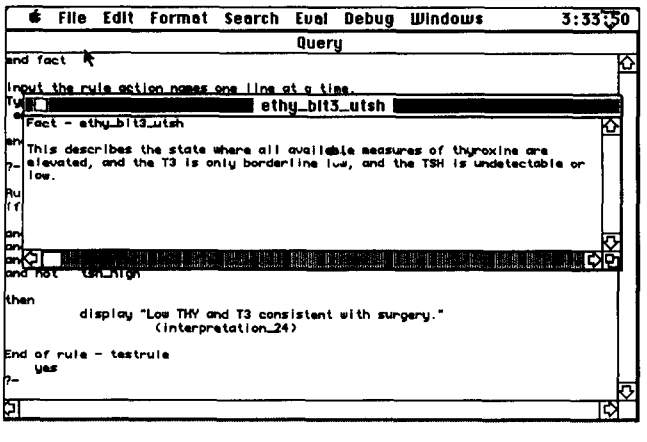

Figure 15. An example help window

overlaying any existing windows, and any number of windows, dependent on available memory, may be open at any time. Since this is independent of the current dictionary operation, this may be done any time the user requires (see Figure 15).

Note that this facility describes the display of a textual description that may be associated with any object occurrence. The other useful help facility is the Usage enquiry, which may be used to determine what rules assert a specified fact, and thus elicit the meaning of the internal fact.

Inferencing procedure

The above sections outline some functions currently supporting the maintenance activity. Having performed the required maintenance, it becomes necessary to check that the maintenance has had the desired effect, that the correct rule is called and the correct interpretation made. To assist with this, a first generation forward chaining inferencing procedure has been generated (see below).

As an implementation issue, the knowledge base has been split into a number of *worlds*, one for the maintenance function, and one for the inference function. This enables us to ensure that the inference function does not corrupt the knowledge domain world. Appendix 1 shows the output of an inferencing run, and the individual phases as described below.

Run/rerun
These functions cause the inference procedure to begin. The Run function initializes memory and prompts for new facts, while the Rerun function lists all currently asserted facts and prompts for more. In addition, Rerun can be directed to start at any inference step of the previous inference run. (An inference step is defined to be the firing of a single rule).

Input facts
This is the first phase of the inferencing procedure. The user is asked to enter a list of facts that are known to be true for a particular hormone profile. Currently, entry is via specific fact names rather than actual hormone sampling levels*.

Find matching rules
This phase uses the entered fact names, and for each produces a list of candidate rules, rules in which the fact is tested for as *present*. We assume that any test via the *absence* relationship, or a *not fact* test, is equivalent to the fact not being asserted and thus need not be tested. This is the binary logic concept, where a fact missing is equivalent to the fact being false and hence not asserted. Sometimes the knowledge might require the specific assertion of a negation, e.g. *not_on_t4*, which although logically equivalent to *not on_t4*, is a different concept. We recognize this distinction, as this is the way the experts relate to the knowledge (see the discussion above regarding Figure 7).

In addition, the rules that fired in previous inference steps of this inference run are excluded from the matching rule sets.

On completing all entered facts, all candidate rule sets are combined to find the resultant candidate rule set. This resultant rule set contains all rules where at least one element of the fact profile is tested for as being present. Note that the union procedure removes redundant rules, i.e. those rules that occur in more than one rule set. The above is analogous to the relational divide operator. The algorithm as implemented is shown in Figure 16.

*The existing Garvan expert system takes data from the automatic blood analysis in the form of values representing the levels of specific blood analytes in the sample. These values are abstracted to higher level descriptors, asserted as facts, prior to executing the main body of the knowledge base. The knowledge extracted from the experts is in the form of rules utilizing these high level descriptors, rather than exact blood hormone levels. Examples of these descriptors include *t3_high* and *tst_low*, which indicate the hormone and the relative level of the hormone in the sample.

```
until no more facts to be tested, do
  get asserted_fact(fact_name)
  not(fact_tested(fact_name)),
  for e in element_relationship table
    with  e.owner_type = 'rule',
          e.member_type = 'fact',
          e.member_name = fact_name,
    for p in previously_fired table
          with e.rule_name not = p.rule_name,
          append e.rule_name to fact_name.rule_set_list
          set fact_tested(fact_name),
    end_for
  end_for
end_do.
until end of rule_set_lists do
  get fact_name_1.rule_set_list
  get fact_name_2.rule_set_list
  % sorting in AAIS Prolog removes duplicate entries
  sort fact_name_1.rule_set_list, fact_name_2.rule_set_list
    fact_name_3.rule_set_list
  assert fact_name_3.rule_set_list
  retract fact_name_1.rule_set_list
  retract fact_name_2.rule_set_list
end_do
if no rule_set_list
then stop_inference
end_if.
```

Figure 16. Algorithm for Find matching rules function

```
until end of rule_set_list do
  get next rule_set_list.rule_name
  for e in element_relationship table
  % remove rules having an explicit fact not asserted test for
  one of the asserted facts
    for f in fact table
      with e.relationship_name = 'absence'
        e.owner_type = 'rule',
        e.owner_name = rule_set_list.rule_name,
        e.member_type = 'fact',
        e.member_name = f.fact_name
      remove rule_set_list.fact_name
    end for
  % remove rules having an explicit fact asserted test for a
  fact that has not been asserted
    for f in fact table
      with  e.relationship_name = 'presence'
            e.owner_type = 'rule'
            e.owner_name = rule_set_list.rule_name,
            e.member_type = 'fact',
            e.member_name not = f.fact_name
      remove rule_set_list.fact_name
    end_for
  end_for
end_do.
```

Figure 17. Algorithm for Check rule completeness function

Check rule completeness

The resultant rule set must now be checked so that any rules that are not completely satisfied are culled. The culling process removes any rules that test for a fact in the premise (it must be asserted), where that fact is not in the current fact profile, and those rules explicitly testing for *not fact*, where the fact is asserted in the current fact profile. The resultant rule set contains all rules that require a subset of the current fact profile to be satisfied as their premise. Figure 17 shows the algorithm used.

Conflict resolution

The candidate rule set from the check rule completeness phase is now passed to the conflict resolution phase. Minimal conflict resolution is undertaken to ensure that the knowledge dictionary has equivalent functionality to Garvan ES1. In the Garvan ES1 system no conflict arises because all rules found which satisfy the fact profile are executed. In addition, rules are ordered so that rules that manipulate the internal facts occur prior to those rules that display interpretations. With the dictionary-based system, in the case of an empty set, the user is informed, and the inferencing procedure ceases. In the case of multiple rules (a conflict condition), the user is informed of the conflicting rules and the first rule is chosen for firing.

Obey selected rule

The single rule extracted by conflict resolution is passed to this phase, and the action(s) to be performed are retrieved. Each action is obeyed in turn, be it asserting a fact, retracting a fact or displaying an interpretation. The rule names of rules that have fired are stored for later interrogation by the user. The task performed by the rule action is determined by determining which relationship the rule action partakes in, i.e. either asserted_by or retacted_by with a fact object, or displays with a kb_data_reference object. The algorithm for this phase is shown in Figure 18.

At the completion of the rule actions, the inferencing procedure recursively calls itself to determine if any other rules are now candidates to be obeyed. If so, the above procedure repeats, selecting an individual rule. If not, then the inferencing procedure ceases.

```
until no more rule_actions for this rule, do
  for e in element_relationship table % get a rule action
      with  e.relationship_name = 'actions',
            e.owner_type = 'rule'.
            e.owner_name = rule_to fire.rule_name,
            e. member_type = 'rule_action'
      for f in element_relationship table % determine action
      of rule action
        with  f.member_type = 'rule_action',
              f.member_name = e.member_name
        if f.relationship_name = 'asserted_by' % do assert
          then assert f.fact_name
        end_if
        if f.relationship_name = 'retracted_by' %do retract
          then retract f.fact_name
        end_if
        if  f.relationship_name  =  'displayed_by'  %display
        interpretation
          then for  k in element_relationship table
               with  k.relationship_name = 'displayed_by',
                     k.owner_type = 'kb_data_reference',
                     k.member_type = 'rule_action',
                     k.member_name = e.member_name
               %extract interpretation details from
               %element_property table and display
               display details k.kb_data_reference_name
               end_for
        end_if
      end_for_
    end_for
end_do
infer. %do next inference step
```

Figure 18. Algorithm for Obey selected rule function

Why_not
The Why_not function may be used to query why a rule did not fire in the inferencing process, in either all or a specified inference step. This is possible because the *check rule completeness* phase maintains a list of *all* rules in the candidate rule set that did not match the fact profile in some way. Why_not also provides for an examination of any rule not appearing in this set because they did not match the fact profile in any way. Figure 19 shows a sample output after the run shown in Appendix 1.

It should be noted that in this system the explanation facility does not rely on the rule trace facility, as implemented in most expert systems to cater for this facility. The user can be informed of exactly why a rule did not fire, either in a specified inference step or in all steps. Rule tracing is displayed by the Rules_fired function (described below).

This more detailed display is possible by treating the rule as data and storing the data in the relational model formalism. Once the fact profile current at an inference step can be determined, and the name of the rule to be explained is given, the step of determining which facts were missing is similar to the check rule completeness algorithm, and involves simple table searching to extract the fact names.

Rules_fired
The Rules_fired function allows the user to determine which rules have fired, and in which order, in the last run of the inferencing procedure. This is analogous to the rule trace implemented in most existing expert systems. Figure 20 shows a sample output after the run shown in Appendix 1.

```
?-why_not '21500.39H'.

The rule 21500.39H was not able to fire in inference step 1
as none of the facts matched the fact profile current then.

The rule 21500.39H could not fire in inference step 2 because
it did not match the fact profile. The differences were:

   vhthy was not asserted
   t3_not_missing was not asserted
   sick_euthy asserted but not used in rule
   comment_thyroid_surgery asserted but not used in rule
```

Figure 19. The Why_not function

```
-rules_fired.

The following rules have fired in the last run:

   Step number - 1 1400.0006B
   Step number - 2 20410.29
```

Figure 20. The Rules_fired function

Display fact profile
This function is used to display the fact profile that was current at an inference step. The user may request an individual inference step, or display the profile at each inference step in the last run. Comparing the fact profiles of consecutive inference steps enables the user to determine what facts were asserted or retracted, and thus gives an indication of the application of the knowledge by the inferencing procedure starting at the initial fact profile. Figure 21 displays the output of this function after the run shown in Appendix 1.

```
?-display_fact_profile all.

At step 1 the fact profile was as follows:
   sick_euthy
   comment_thyroid_surgery

At step 2 the fact profile was as follows:
   sick_euthy
   comment_thyroid_surgery
   surgery
```

Figure 21. The Display_fact_profile function

This shows that inference step *one* resulted in the fact 'surgery' being asserted into working memory. By analyzing Figure 21 in conjunction with Figure 20, it can be seen that rule 1400.0006B asserted the fact 'surgery'. This is borne out by looking at the run output in Appendix 1, where it shows indeed that rule 1400.0006B asserted the 'surgery' fact. As step *three* is not displayed, it is indicative that rule 20410.29 did not alter working memory, by asserting or retracting facts, and that inferencing ceased.

Future enhancements

Context
The dictionary approach allows for full documentation of all the knowledge, and provides a basis for setting up techniques for unrestricted browsing of the knowledge. We hypothesize that the most useful feature will be flexible alteration of the context in which the knowledge is examined. The user may start with a broad fact profile, and subsequently narrow the profile by the addition of extra facts. If this addition were done in the context of the existing facts and associated rule premises, then an orderly narrowing of the fact profile is possible, not necessarily relying on the knowledge of the expert in the respective knowledge domain and the experience of the knowledge engineer in how the system's knowledge has been organized. This is in strong contrast to current tools.

Context is similar to constraints in database theory. Context on the knowledge can be of benefit in several areas. First, the knowledge dictionary model allows the user to attach any number of classification terms to knowledge domain object occurences, and these classifications, if applied to rules, facts and rule actions, may be used to further narrow the search area of candidate rules. This is analogous to the grouped-by operation implemented in relational theory. Second, the production of the candidate rule set is based on a particular fact profile. The addition of further facts must reduce the candidate set even further. Thus it is feasible to only allow facts to be added to the fact profile within

the context of the existing candidate rule set, leading to a subset of the existing rule set, e.g. if the run shown in Appendix 1 were limited to the area of thyrotoxicosis, any rule with a fact profile not in this context would be weeded out in the *find_matching_rules* phase. If, on the other hand, the area of interest was hypothyroidism, then possibly none of the rules may have been candidates. And third, when adding rules or new elements, either facts or rule actions, to existing rules, the user should be warned if the addition results in any conflict with existing contextual knowledge established by the currently known relationships between facts, rule actions and rules. This is not necessarily an error condition, as the new information may expand the currently known horizons.

Knowledge base checking

The area of knowledge checking has not been mentioned up to now. Knowledge checking is seen as a matter of importance, especially when the automatic generation of the run-time system is produced from the knowledge dictionary desciption.

We currently have a set of functions to check each individual table for standard errors, e.g. duplicate entries, pointers to non-existent entries, etc. However, the majority of logic checking has not yet been implemented. It is envisaged that the standard logic checking will be implemented, including tests for subsumption, completeness, correctness, consistency, ambiguity, tautology, etc.[23-24].

One checking facility that is available to us, but is not yet implemented in the dictionary, is a *cornerstone cases* database containing all those cases and their interpretations that caused a change to the existing system. This database is available to be read and each case checked to ensure that correct interpretations are produced in all cases. Interestingly enough, although this checking procedure is frequently run on the existing knowledge base, logic errors in the rules have been found, implying that this type of checking is not foolproof. Such rules may arise through splitting and narrowing existing rules, which may then never be used.

Generate run time knowledge base

The automatic generation of the run-time knowledge base should be seen as analogous to the generation of the run-time conventional system, as now employed in several commercial data dictionary systems. If the dictionary contains a detailed specification of the domain, and has meta-knowledge regarding the implementation vehicle and the mapping of the domain functions into that vehicle, then the production of the run-time 'code' is relatively automatic. This enables the knowledge dictionary to become a truly active dictionary, and opens the path for trained experts to maintain the knowledge base themselves without having expertise in the field of AI languages.

The literature abounds with examples of the *Feigenbaum bottleneck*[21]. Several solutions are presented, including letting the experts do their own knowledge engineering. The problems raised are that experts are not knowledge engineers, and so the bottleneck may in fact be more constricted. In addition, the experts do not usually know the language of the implementation vehicle, and so they face a doubly daunting task. The knowledge dictionary environment may present the experts with an interface to express their knowledge in a way that facilitates the knowledge acquisition process, thereby relaxing the bottleneck.

Different knowledge formalisms

Once the knowledge engineer is able to generate the run-time system in the existing **if...then** formalism, the dictionary must be expanded to allow other formalisms to be stored and generated. The frame formalism[26] is high on the list of priorities. We suspect that since rules and frames are only formalisms for expressing the relationships in knowledge, the availability of the dictionary with its potential to use a wide range of techniques to express the underlying relationships will decrease the emphasis on particular knowledge formalisms.

Specialized hardware searching engines

Since the dictionary has been implemented in Prolog, the majority of its time is spent pattern matching and joining its internal tables, thereby making it processor intensive. In our case, the majority of the time is spent in traversing the element_relationship table (currently having 1940 entries). One avenue to speed up the process is the adoption of a hardware pattern matching engine (e.g. the Relational Algebra Accelerator[27], which uses the method of superimposed coding to speed up the pattern matching process undertaken by Prolog in the evaluation of the current goal).

CONCLUSION

The work to date suggests that the knowledge dictionary technology is suitable for use on expert systems, and that benefits are to be found similar to those for conventional data processing systems. The decomposition of heuristics and their storage in the relational data model makes available the power of the relational data model for knowledge maintenance and browsing. If the system is small enough, then the dictionary environment may be suitable in its own right as an expert system shell.

It has also been shown that heuristics may be decomposed into constituent parts, facts and rule actions. In doing so, any heuristic is implemented in its most understandable form, and automatically documented and cross referenced. Using the dictionary environment, the documentation and cross referencing is automatically kept up-to-date, in a form understandable to both knowledge engineers and knowledge domain experts.

The decomposed form of the knowledge is inferenceable using a simple inference engine obeying simple data manipulation rules. In addition, a number of future enhancements have been highlighted, including knowledge base checking and a context facility, which enables the expert system to emulate the domain expert's reasoning process more closely.

ACKNOWLEDGEMENTS

This project would not have been possible without the contributions of the following, ranging from the initial development of Garvan ES1, to important advice for the current project: Bob Colomb and Neil Ashburner of the CSIRO Division of Information Technology; Kim Horn of Telectronics (formerly at the Garvan); Ross Quinlan of the University of Sydney; Leslie Lazarus, Ken Ho, Rick Symons, Ross Vining and other domain experts of the Garvan Institute.

REFERENCES

1 **Buchanan, B** 'Expert systems: working system and the research literature' *Expert Syst.* Vol 3 No 1 (January 1986) pp 32–50

2 **Bachant, J and McDermott, J** 'RI revised: four years in the trenches' *AI Magazine* Vol 5 No 3 (Autumn 1984) pp 21–32

3 **Compton, P, Horn, K, Quinlan, J R , Lazarus, L and Ho, K** 'Maintaining an expert system' *Proc. 4th Australian Conf. on Appl. of Expert Syst.* Sydney, Australia (1988) pp 110–129

4 **Horn, K A, Compton, P, Lazarus, L and Quinlan, J R** 'An expert system for the interpretation of thyroid assays in a clinical laboratory' *Australian Comput. J.* Vol 17 No 1 (February 1985) pp 7–11

5 **Hearn, B, Brook, K, Ashburner, N and Colomb, R** 'SIRATAC, a cotton management expert system' *Proc. 1st Australian Artif. Intell. Congress* Melbourne, Australia (1986)

6 **Jansen, B** 'The redevelopment of the SIRATAC cotton management system' *CSIRO Division of Info. Technol. Technical Report — TR-FD-87-01* (April 1987)

7 **Jansen, B** 'A data dictionary approach to the software engineering of rule based expert systems' in **Gero, J S and Stanton, R (eds)** *Artificial Intelligence Developments and Applications* North Holland, Netherlands (1988) pp 101–117

8 **Debenham, J** 'Knowledge base design' *Australian Comput. J.* Vol 17 No 1 (February 1985) pp 42–48

9 **Debenham, J** 'Expert systems: an information processing perspective' *Proc. 2nd Austalian Conf. on Appl. of Expert Syst.* (May 1986) pp 230–248

10 **Duda, R O, Hart, P E, Reboh, R, Reiter, J and Risch, T** 'SYNTEL: using a functional language for financial risk assessment' *IEEE Expert* Vol 2 No 3 (Autumn 1987) pp 18–31

11 **Sprague, R H (Jr) and Carlson, E D** *Building effective decision support systems* Prentice Hall, USA (1982) pp 225–35

12 **Jansen, B** 'Applying software engineering concepts to rule based expert systems' in **Partridge, D (ed)** *AI and Software Engineering* Ablex, USA (in press)

13 **Dolk, D R and Kirsch, R A II** 'A relational information resource dictionary system' *Commun. ACM* Vol 30 No 1 (January 1987) pp 48–61

14 **Al-Zobaidie, A and Grimson, J B** 'Expert systems and database systems: how can they serve each other?' *Expert Syst.* Vol 4 No 1 (February 1987) pp 20–37

15 **Held, J P and Carlis, J V** 'Conceptual data modelling of an expert system' *Proc. 4th Int. Conf. on Entity-Relationship Approach* Chicago, USA (October 1985) pp 182–188

16 **Ishikawa, H, Izumida, Y, Yoshino, T, Hoshiai, T and Makinouchi, A** 'A knowledge-based approach to design on a portable natural language interface to database systems' *IEEE Proc. Conf. on Data Eng.* Los Angeles, USA (1986) pp 134–143

17 **Leung, C M R and Nijssen, G M (Shir)** 'Database oriented expert systems' *Proc. AI87 Conf.* Sydney, Australia (1987)

18 **Parle, A** 'Formal specification of a self referential data dictionary' *CSIRO Division of Info. Technol. Technical Report TR-FC-87-05* (1987)

19 **Codd, E F** 'A relational model for large shared data banks' *CADCM* Vol 13 No 6 (1970) pp 377–387

20 **Carlis, J V** 'HAS, a relational algebra operator, or divide is not enough to conquor' *Proc. IEEE Int. Conf. on Data Eng.* Los Angeles, USA (1986) pp 254–261

21 **Clancey, W J** 'Heuristic classification' *Artif. Intell.* Vol 27 (1985) pp 289–350

22 **Hodges, W** *Logic* Penguin, UK (1987)

23 **Liu, N K and Dillon, T** 'Detection of consistency and completeness in expert systems using numerical Petri nets' *Proc. AI87 Conf.* Sydney, Australia (1987) pp 170–188

24 **Nguyen, T A, Perkins, W A, Laffey, T J and Pecora, D** 'Knowledge base verification' *AI Magazine* Vol 8 No 2 (Summer 1987) pp 69–75

25 **Michie, D** 'Current developments in expert systems' *Proc. 2nd Australian Conf. on Appl. of Expert Syst.* Sydney, Australia (1986) pp 163–182

26 **Winston, P H** *Artificial Intelligence* (2nd edn) Addison-Wesley, USA (1984)

27 **Colomb, R M and Jayasooriah** 'A clause indexing system for Prolog based on superimposed coding' *Australian Comput. J.* Vol 18 No 1 (February 1986) pp 18–25

APPENDIX I — OUTPUT FROM THE INFERENCING PROCEDURE

This appendix shows a sample output from the inferencing procedure as described above in the section on the inferencing procedure. The user input is in italics. Tracing has been turned on to show the intermediate rule sets in the find_matching_rules phase. The 'check rule completeness' output has been edited to remove the bulk of the rules, but enough remains to demonstrate the output procedure.

```
?- run.
initialise_memory - completed ok.
Please enter the facts to be asserted. When finished, type 'end fact'.
enter fact - sick_euthy
enter fact - comment_thyroid_surgery
enter fact - end fact
starting find_matching_rules
find_matching_rules - all rule sets found.

/*rule_set/2*/

rule_set(sick_ euthy,['20200.26', '20201.32', '20205.32',
'20205.32A', '20205.32B', '20205.32C'.'20205.32D'.
'20205.32E'. '20210.32'. '20300.25'. '20400.49'. '20410.29'.
'20401.50'. '20411.50'. '20500.32'. '20600.46'. '20600.46A'.
'20601.41'. '20610.46'. '20610.46A'.

rule_set(comment_thyroid_surgery,['1400.0006B'].

The following rules are candidate rules to fire.

[1400.0006B, 20200.26, 20201.32, 20205.32, 20205.32A,
20205.32B, 20205.32C, 20205.32D, 20205.32E, 20210.32,
20300.25, 20400.49, 20401.50, 20410.29, 20411.50, 20500.32,
20600.46, 20600.46A, 20601.41, 20610.46, 20610.46A]
starting check_rule_completeness

checking rule 1400.0006B
```

```
checking rule 20200.26
  lothy is missing
  no_comment is missing
  comment_thyroid_surgery is asserted but not used

checking rule 20201.32
  lothy is missing
  no_comment is missing
  sick is missing
  comment_thyroid_surgery is asserted but not used

... Intervening rules removed to limit display

checking rule 20610.46
  hyperthyroid is missing
  comment_thyroid_surgery is asserted but not used

checking rule 20610.46A
  antithyroid is missing
  comment_thyroid_surgery is asserted but not used
check_rule_completeness - finished

/*rule_set/2*/

rule_set(candidate_set,['1400.0006B']).

starting check_rule_context
starting conflict resolution
starting obey_selected_rule
obey_selected_rule: obeying rule 1400.0006B
Obeying rule action - assert surgery (surgery_true)
starting find_matching_rules
find_matching_rules - all rule sets found.

/*rule_set/2*/

rule_set(sick_euthy,['20200.26'. '20201.32'. ' 20205.32'.
'20205.32A'. '20205.32B'. '20205.32C'. '20205.32D'.
'20205.32E'. '20210.32'. '20300.25'. '20400.49'. '20410.29'.
'20401.50'. '20411.50'. '20500.32'. '20600.46'.
'20600.46A'. '20601.41'. '20610.46'. '20610.46A']).
rule_set(surgery,[10000,'11100.01'. '11110.02'. '20400.49'.
'20410.29'. '21500.39'. '21500.39B'. '21500.39D'. '21500.39F'.
'21500.39H'. '22200.36A'. '22210.45A'. '22400.39A'. '43010.49'.
'43010.49A'. '43010.49B'. '43010.49C'. '43011.49'. '43011.49A'.
'43012.49'. '43700.49'. '43700.49A'. '44100.10']).

The following rules are candidate rules to fire.

[10000, 1100.01, 1110.02, 20200.26, 20201.32, 20205.32,
20205.32A, 20205.32B, 20205.32C, 20205.32D, 20205.32E,
20210.32, 20300.25, 20400.49, 20401.50, 20410.29, 20411.50,
20500.32, 20600.46, 20600.46A, 20601.41, 20610.46, 20610.46A,
21500.39, 21500.39B, 21500.39D, 21500.39F, 21500.39H,
22200.36A, 22210.45A, 22400.39A, 43010.49, 43010.49A,
43010.49B, 43010.49C, 43011.49, 43011.49A, 4312.49, 43700.49,
43700.49A, 44100.10]

starting check_rule_completeness

checking rule 10000
  goitre is missing

checking rule 11100.1
  t3_high is missing
  hithy is missing
  tsh_missing is missing
  t4u_not_high is missing
  tbg_not_high is missing
  sick_euthy is asserted but not used
  comment_thyroid_surgery is asserted but not used

checking rule 11110.02
  t3_high is missing
  hithy is missing
  tsh_missing is missing
  tbg_high is missing
  t4u_high is missing
  sick_euthy is asserted but not used
  comment_thyroid_surgery is aserted but not used

... Intervening rules removed to limit display

checking rule 43011.49A
  hypo_sick is missing

checking rule 43012.49
  lt3 is missing

checking rule 43700.49
  nt4t3_htsh is missing
  sick_euthy is asserted but not used
  comment_thyroid_surgery is asserted but not used

/*rule_set/2*/

rule_set(candidate_set,['20410.29'].

starting_check_rule_context
starting conflict resolution
starting obey_selected_rule
obey_selected_rule: obeying rule 20410.29

The interpretation for this blood sample is-
"Low T3 consistent with surgery."
(interpretation_26)
```