--

--

AN ADVANCED BINARY ENCODED MATRIX REPRESENTATION FOR RULEBASE VERIFICATION

Frans Coenen

Department of Computer Science,
Liverpool University,
Chadwick Building,
P.O. Box 147,
Liverpool L69 3BX,
England.
email: frans@uk.ac.liverpool.compsci
Tel: 051 794 3698
Fax: 051 794 3715

ABSTRACT

An advanced binary encoded matrix representation to support rulebase verification is described. The representation has two important advantages over traditional rulebase matrix representation techniques. Firstly the amount of storage capacity required is substantially less than that required using traditional techniques; the advantage gained is in the order of 97%. Secondly the binary representation offers a significant improvement in the processing of rulebase matrices, to identify various rulebase anomalies, through the use of logical "bitwise" comparators. Both techniques are described as implemented in the IMVER (Incidence Matrix VERification) systems and are fully analysed and compared using a number of complexity measures to illustrate the advantages gained using the binary encoded incidence matrix representation over the traditional representation.

KEYWORDS: Rulebase verification, Binary encoded incidence matrices

January 21, 1998

--

--

AN ADVANCED BINARY ENCODED MATRIX REPRESENTATION FOR RULEBASE VERIFICATION

Frans Coenen

Department of Computer Science,
Liverpool University,
Chadwick Building,
P.O. Box 147,
Liverpool L69 3BX,
England.
email: frans@uk.ac.liverpool.compsci
Tel: 051 794 3698
Fax: 051 794 3715

1. INTRODUCTION.

Rulebase verification systems based on incidence matrix techniques are well established. The technique revolves around the representation of rulebases using matrices (or tables) the elements of which represent propositions that may be present in a rule. The matrices can then be analysed to establish the existence (or non-existence) of various rulebase errors and anomalies. The technique was utilised in an experimental rulebase verification system, the IMVER-1 (Incidence Matrix Verification) system, developed at the University of Liverpool[1,2]. The principal criticisms directed at this system are:

1. That a considerable amount of storage space was required to store the matrices needed to represent any realistically sized rulebase.
2. The processing power needed to carry out the required matrix multiplications and the necessary comparisons was prohibitive.

These criticisms are applicable to all rulebase verification systems that use incidence matrix techniques and not just the IMVER-1 system. This paper represents a response to these criticisms. An advanced matrix representation technique that utilises a binary encoding to represent rulebase matrices is proposed. The technique has been built into an updated version of the IMVER-1 system, IMVER-2. The benefits offered by the advanced binary encoded matrix representation are:

1. A considerable saving in the space required to store rulebase matrices.
2. A significant reduction in the processing required to identify various rulebase errors and anomalies.

Both the traditional approach to incidence matrix KBS verification, as implemented in the IMVER-1 system, and the advanced binary encoded approach as advocated here, and built into the IMVER-2 system, are described in detail.

2. KBS VERIFICATION

The erroneous operation or failure of KBSs will at best result in lost time and hence increased costs. In a "worst case scenario" it may result in a complete system shutdown. Further, in some industries, such as chemical production, the ecological ramifications can be far reaching; consider the event where a KBS fails to apply an emergency shut down procedure. The verification and validation of KBS prior to delivery is therefore vital not only to prevent local financial losses but also to address the, far wider reaching, global consequences of failure. By verifying a KBS we cannot of course guarantee that it is error free. However, we can increase the confidence level that we may have in its operation; we may even attempt to qualify this numerically.

There are many definitions of KBS (rulebase) verification (Satre and Massey[3] and Balci and Sargent[4] both give extensive reviews). I have adapted the definition, with respect to traditional software systems, originally given by Boehm[5] and subsequently generally adopted by the software industry. Boehm defines software verification by posing the question:

Are we building the product right?

Thus verification is the process of checking that a system meets the requirements definition/specification. With regard to KBSs, where (in the traditional sense) no meaningful requirements specification exists, the process of verification described in these terms becomes meaningless. I have thus defined KBS verification as the process of checking that the system's operation is right, i.e. that results are arrived at in the correct manner. The work described here concentrates on KBS that use a rulebase approach to reasoning, we are thus primarily concerned with the verification of rulebases.

Verification is an integral part of the the development of all software systems and there are many techniques available, aimed at a variety of different types of system. Traditional software verification techniques are generally concerned with software testing. These techniques can be conveniently divided into *black box* and *white box* techniques. The first includes expediences such as function testing and equivalence testing; and the second techniques such as statement testing, branch testing and path testing. However traditional software verification techniques do not translate easily to KBSs. I can point to a number of reasons for this, two of the most significant are:

- The program flow and control in KBSs is hidden.
- The nature of KBSs is such that exhaustive verification is impractical.

At present there is a great deal of research activity concentrated in the field of rulebase (and KBS) verification; this is well illustrated by the attendance at the recent EUROVAV'91 and '93 conferences. The incidence matrix technique that forms the subject of this paper is one approach. Other well established approaches include normal form techniques[6], decision table methods[7], KB reduction[8], generic rule systems[9], and the use of static inspection tools[10].

3. OVERVIEW OF INCIDENCE MATRIX TECHNIQUES

Incidence matrix techniques have been used in a number of well documented KBS V&V systems. Agarwal and Tanniru[11] utilised the technique in their petri-net based system, Tsai and Jang[12] used it in their "framework" for KBS V&V and Botten and Raz[13] used a similar approach in their KBS V&V system which was later extended to support the verification of cooperating KBSs[14].

The deficiencies of the incidence matrix technique have been recognised since its conception and various methods have been investigated to (a) reduce the overall size of the matrices involved and (b) ease the associated processing. To date this has focused on the refining of the initial rule-proposition incidence matrix. One technique is to reduce the size of this matrix by multiplying it by the transpose of itself; the result is a rule-rule incidence matrix the elements of which indicate the number of propositions shared by pairs of rules. Such a matrix supports structural verification. However, the approach does not resolve the basic problems of storage and processing. Firstly the required matrix multiplication adds another level of processing to the technique. Secondly the resulting rule-proposition matrix does not support the identification of anomalies such as duplication, subsumption and inconsistency; for this it is necessary to return to the original matrix, hence the original matrix cannot be "thrown" away and the approach has the adverse effect of increasing the amount of storage space required not decreasing it.

Alternatively I can rearrange the initial rule-proposition matrix so that the columns and rows with the most instantiated elements appear towards the top left-hand corner of the incidence matrix. The effect of this is that, during processing, instantiated elements will be identified sooner than they might otherwise have been. This approach supports inference verification in that the connectivity of the rulebase can be established efficiently. However, the advantage gained tends to be outweighed by the processing required to rearrange the matrix and the observation that in many cases it is still necessary to process entire rows and columns.

In the IMVER-1 system no attempt at refinement of the incidence matrix was made. The resulting efficiency of the approach was considered to be no worse, and in some cases better, than other KBS V&V systems of its type. It is suggested that the IMVER-2 incidence matrix representation suggested here surpasses all previous attempts at addressing the problems of storage space and processing efficiency associated with the incidence matrix technique.

4. ERRORS AND ANOMALIES IN RULE-BASES

There is much discussion concerning the errors and anomalies that can occur in a rule-base with many authors presenting "definitive" lists[7,15]. I do not wish to repeat the exercise here, however to provide a common forum of understanding it is essential that I present definitions for the anomalies addressed by the IMVER systems.

Subsumption

Subsumption exists when one rule is a more specialised case of another. Thus if two rule antecedents are identical except that one antecedent includes one or more additional propositions, and the consequents are also identical, subsumption is considered to exist.

Duplication

The situation where the antecedents and consequents of two rules are identical (except perhaps in the ordering of propositions) is a specialised form of subsumption usually referred to as duplication.

Inconsistency

Inconsistency is defined as the situation where two rule antecedents are identical but their consequents are different (some authors use the term *contradiction* to define this anomaly). Where the antecedents of two rules are such that subsumption may exist but the consequents are different subsumption and inconsistency are considered to exist together. The presence of an inconsistent rule may also indicate the start of more general cases of subsumption resulting from chains of inferences.

Connectivity

A rule in a rulebase is connected if it has at least one downward connection and one upward connection. If a rule has neither it is unconnected, if a rule has only upward connections it is a leaf rule (or it is unconnected) and if it has only downward connections it is a root rule (or it is unconnected). A rule that has no upward or downward connections is sometimes referred to as a redundant rule; while a rule that is not a leaf rule but has no downward connection is sometimes referred to as a dead end rule. General cases of redundancy arising from chains of inferences are highlighted by the presence of unexpected root and leaf rules.

Circularity

Circularity exists when a rulebase contains a set of rules such that a non terminating loop can occur when the rules are fired. Circularity presents an urgent problem in backward chaining systems.

It is generally acknowledged[8,15] that the above five types of error/anomaly are the principal errors and anomalies that need to be identified when implementing rulebase verification. In this paper the identification of subsumption, duplication and inconsistency is collectively referred to as structural verification, while that of connectivity and circularity as inference verification. The above definitions have deliberately been kept short, readers requiring a more complete explanation for their derivation are referred to Coenen and Bench-Capon[16].

It should also be noted that there is a distinction between errors and anomalies. Errors are clearly undesirable and will require remedial action to remove them; anomalies in contrast may not necessarily represent problems in themselves, but rather symptoms of genuine errors. Further, in some cases the presence of certain anomalies may not be of significance, because not strictly correct, they may not impair the function of the system. There is much argument as to what constitutes an anomaly and what constitutes an error, much depends on the inference mechanism used; whatever the case both errors and anomalies are undesirable.

5. TRADITIONAL MATRIX REPRESENTATIONS

Using a traditional rulebase matrix representation each column in the matrix is used to represent a proposition, while each row represents a rule in which the proposition may appear. If a particular proposition appears in a rule the appropriate rule/proposition (row/column) intersection is instantiated with (say) a `1`. Otherwise intersections are instantiated with (say) a `0`. To demonstrate connectivity in a rulebase it is necessary to retain information concerning which propositions appear in the antecedents of rules and which in the consequents of rules. To achieve this either separate symbols for antecedent and consequent instantiations can be used or two matrices - an antecedent matrix and a consequent matrix - can be generated. Botten and Raz[13] in their KBS verification system adopted the first technique and used the symbols `I` and `T` to distinguish between antecedent and consequent propositions. In the IMVER-1 system the twin matrix technique was adopted.

Consider the balanced rulebase given in Table 1 this can be represented as an antecedent and a consequence matrix pair of the form given in Tables 2 and 3. In the IMVER systems rules are "read" from left to right thus in the above rulebases, the root proposition `aa` will be represented by the third column in each matrix. The matrix columns in Table 2 and 3 thus represent the propositions `ab`, `ac`, `aa`, `ad` and so on up to `be`.

*** Table 1 here ***

*** Table 2 here ***

*** Table 3 here ***

The storage capacity required for an antecedent and consequent matrix pair representing any rulebase can be calculated from the expression:

$$2 \times N \times Nr \times Np$$

Where:

- N = The number of bytes required to store a standard unsigned integer.
- Np = The number of unique propositions contained in the rulebase.
- Nr = The number of rules contained in the rulebase.

Given that both the IMVER-1 and IMVER-2 systems were implemented in the C programming language the number of bytes required to store a standard unsigned integer is equivalent to 4. Thus the above rulebase would require 3720 bytes of storage ($2 \times 4 \times 15 \times 31$), 1860 for each matrix.

6. THE ADVANCED BINARY ENCODED MATRIX REPRESENTATION

Inspection of the matrices presented in Tables 2 and 3 indicate that, assuming a single proposition will appear only once in any individual rule antecedent or consequent, the elements in the incidence matrices will always be instantiated with 1s and 0s. Thus the rows in the matrix can be considered to represent binary numbers. The maximum permitted size of an unsigned integer in the C programming language is 32 bits. Thus we can represent matrix rows of up to 32 propositions using a single 32 bit decimal integer. So that rulebases with more than 32 propositions can be represented we can support the notion of blocks of 32 propositions. Consider the rulebase given in Table 4 comprising 31 rules and 63 propositions, and designed to establish the root proposition `aa`. This rulebase can be represented as binary encoded antecedent and consequent matrices each consisting of two blocks (the antecedent matrix is given in Table 5).

*** Table 4 here ***

*** Table 5 here ***

The total storage capacity required to represent a rulebase using this binary encoded matrix format is then given by:

$$2 \times N \times Nr \times B$$

Where:

B = The number of proposition blocks (rounded up to the nearest whole number), i.e. Np/32.

Thus only 496 bytes ($2 \times 4 \times 31 \times 2$) would be required to represent the above rulebases, i.e. a considerable saving (96.8%) over the 15624 ($2 \times 4 \times 31 \times 63$) required using the above traditional representation. Inspection of the above expressions indicates that this advantage will hold for any realistically sized rulebases.

7. IMVER ARCHITECTURE AND OPERATION

Before continuing to analyse the advanced binary representation with respect to inference and structural verification it is appropriate to first consider the structure and operation of the IMVER systems. A block diagram illustrating their top level architecture is given in Figure 1. If the systems are viewed simply as "black boxs" the input will be a rulebase and the output a "commentary". If the system is considered as a "white box" a number of top-level modules can be identified, notably a translator and a number of rulebase verification modules. The translator comprises a lexical analyser, a parser and a matrix generator. Its overall purpose is to take a "raw" rulebase and translate it into the appropriate matrix format. During this process details of the rulebase to be investigated are also stored so that they can be reproduced as part of the commentary that will eventually result. At present IMVER rulebases are required to be in the following syntax:

*** Figure 1 here ***

(rule_base)	: (rule)
	: (rule) (rule_base)
	;
(rule)	: (propositions) (propositions) n;
(propositions)	: proposition
	: proposition (propositions)
	;

However, translators for other production rule formats can easily be created. For example the above syntax does not support disjunctions between propositions. A rulebase containing such disjunctions must therefore first be converted into a conjunctive form.

Once the appropriate incidence matrices have been created the two rulebase verification modules are applied in sequence and the commentary produced. The commentary consists of two parts. As a result of structural verification groups of rules which subsume one another or are duplicates or are inconsistent with one another are identified. A listing of the identified rules then forms the first part of the commentary. As a result of inference verification each rule is labelled according to whether it is identified as a root, body, leaf or redundant rule, or part of a circular rule set. The second part of the commentary thus comprises a complete list of rules each with its associate inference verification label.

The system has been tested using a number of genuine and artificial rulebases. In particular I have used a rulebase supplied by Merabti et al.[17] taken from their KBS for the selection of LAN architectures. I have also used the error seeded "animals" rulebase given in Coenen and Bench-Capon[16]. The rules in these rulebase are in the standard production rule format and hence are immediately compatible with the IMVER format. The algorithms used in both systems, successfully verified these rulebases identifying all errors and anomalies of the types identified in Section 4 above.

To examine the efficiency of the advanced IMVER-2 binary representation over the traditional approach incorporated into the IMVER-1 system artificial uniform balanced rulebases were used (see below).

8. UNIFORM BALANCED RULEBASES

The rulebases presented in Tables 1 and 4 are perfectly uniform; each rule has two conjuncted antecedent propositions and a single consequent proposition, and each rulebase can be presented as a perfectly balanced binary tree. A binary tree representing the rulebase given in Table 1 is presented in Figure 2. The tree has a depth of 5, a single root proposition, 16 leaf propositions and 14 body propositions. The advantage of uniform rulebases of this form is that they can be defined mathematically and, as a result, certain time complexity measures concerning the matrices used to represent the rulebases calculated.

*** Figure 2 here ***

I have define balanced rulebases according to the depth of the tree that may be used to represent them. Given the depth (D) of such a rulebase I can calculate the number of propositions (Np) from the expression:

$$Np = 2^D - 1$$

Of these there will always be one root proposition; I say that Nrp (number of root propositions) = 1. The number of leaf propositions (Nlp) and body propositions (Nbp) can then be calculated using the expressions:

$$Nlp = \frac{(Np + 1)}{2}$$
$$Nbp = \frac{(Np - 3)}{2}$$

I can also determine the number of antecedent and consequent proposition (Nap and Nac) from the expressions:

$$Nap = Np - Nrp$$
$$Ncp = Np - Nlp$$

The number of rules (Nr) in a balanced rulebase can be calculated using the expression:

$$Nr = \frac{2^D}{2} - 1$$

of which there will always be only one root rule, i.e. Nrr (number of root rules) = 1. The number of leaf rules (Nlr) and the number of body rules (Nbr) can then be found using the expressions:

$$Nlr = \frac{(Nr + 1)}{2}$$
$$Nbr = \frac{(Nr - 3)}{2}$$

9. INCIDENCE MATRIX PROCESSING

Incidence matrices represented using a traditional representation are usually processed by testing elements in individual rows to determine whether each element has been instantiated or not. Any instantiated elements are then compared with the elments in other rows or columns to establish the existence, or nonexistence, of various rulebase errors and anomalies. Using the advanced matrix representation blocks of 32 propositions can be tested simultaneously. If a block equates to nonzero one or more of its proposition elements are instantiated. To compare proposition blocks logical operators can be used. For example if the result of performing a logical "and" on two blocks is nonzero the blocks share one or more proposition elements. Examples:

```
(a)  1 1 1 0 (14)   (b)  0 1 1 1 (7)
```

```
  & 0 1 0 0 (4)       & 0 1 1 1 (7)
    -------             -------
    0 1 0 0 (4)         0 1 1 1 (7)


           (c)  1 0 1 0 (10)
              & 0 1 0 1 (5)
                -------
                0 0 0 0 (0)
```

In case (a) the blocks share a single proposition, in case (b) the blocks are identical and in case (c) no propositions are shared. To differentiate between result (a) and (b) an "exclusive or" operation can be used. For example:

```
(a)  1 1 1 0 (14)   (b)  0 1 1 1 (7)
   ^ 0 1 0 0 (4)       ^ 0 1 1 1 (7)
     -------             -------
     1 0 1 0 (10)        0 0 0 0 (0)
```

If the result is zero the blocks are identical. In this manner binary encoded matrices can be processed much more efficiently than when using a traditional incidence matrix representation.

10. STRUCTURAL VERIFICATION

Structural verification comprises the identification of anomalies such as subsumption, duplication and inconsistency. To implement structural verification every rule in the rulebase must be compared with every other rule in the rulebase. Thus, at the rule level, the time complexity of such an algorithm is:

$$\frac{Nr^2 - Nr}{2}$$

At the proposition level the algorithm requires that every proposition in each rule is compared with every proposition in every other rule until, for each rule pair, it can be ascertained that subsumption, duplication or inconsistency does or does not exist. To do this rule antecedent pairs are usually considered first followed by the associated rule consequents where appropriate. Initially there are three possible outcomes:

1. The antecedents are different therefore subsumption, duplication or inconsistency cannot exist.
2. One antecedent is subsumed by the other therefore subsumption or subsumption and inconsistency exists.
3. The antecedents are identical in which case the rules are duplicates or an inconsistency exists.

In the last two cases, to establish the exact nature of the anomaly identified, it is necessary to go on to consider the rows in the consequent matrix associated with the two antecedent matrix rows under consideration. If the consequent rows are identical subsumption or duplication, as appropriate, exists. If the consequents are different subsumption and inconsistency exist together or, if the antecedents were identical, a simple inconsistency exists. In the case of balanced rulebase where subsumption, duplication and inconsistency will not exist we only need to establish that the rows in an antecedent matrix are different. Using the traditional, IMVER-1, representation the number of comparisons that this will require can be given by the expression:

$$\sum_{i=1}^{i=N} 2(i^2 + i)$$

Where the value for N is arrived at using:

$$N = \frac{Np - 3}{2}$$

By induction this will reduce to:

$$\frac{NP^3 - 3Np^2 - Np + 3}{12}$$

In the case of the IMVER-2 representation, where logical operators are used to compare blocks of 32 antecedent proposition elements simultaneously, the number of comparisons is given by:

$$\frac{3}{2} \sum_{i=0}^{i=B-1} Nr^2 - Nr + 16i - 256i^2$$

which, again by induction, reduces to:

$$\frac{3BNr(Nr-1)}{2} - 4B(B-1)(32B-19)$$

Consider the balanced rulebase given in Table 4. The number of comparisons required to carry out structural verification using the traditional IMVER-1 representation will be:

$$C1 = \frac{63^3 - 3(63)^2 - 63 + 3}{12} = 19840$$

compared to:

$$C1 = \frac{(3\times2\times31)(31-1)}{2} - 8(64-19) = 2430$$

using the IMVER-2 representation.

From the above it is clear that implementing the structural verification algorithm using a binary encoding requires significantly fewer comparisons than when using a traditional approach. In Table 6 some empirical results are presented. The table shows the total number of comparisons required to carry out structural verification on a number of balanced rulebases of varying depths. From the table it can be seen that, (a) the results compare precisely with those produced using the above expressions and (b), for realistically sized rulebases, the advantages in efficiency gained using the advanced binary encoded representation are in the order of 90%.

*** Table 6 here ***

11. INFERENCE VERIFICATION

The term inference verification refers to the analysis of the interconnection between rules, i.e the connectivity of a the rulebase. Connectivity algorithms generally consist of two parts, one to establish upward connectivity and one to establish downward connectivity. To establish upward connectivity rows in the consequent matrix and columns in the antecedent matrix must be inspected. In the case of a root (or redundant) rule, by definition, there will be no upward connection. To establish downward connections rows in the antecedent matrix and columns in the consequent matrix are inspected; leaf rules will have no downward connections.

Circularity detection algorithms involve the tracing of inference chains either commencing with leaf rules and forward chaining to root rules or vice versa. If, as a result, the chain returns to the start rule, or leads up or down to a circular rule set, circularity exists. It is not necessary to test each individual rule specifically as chains of rules can be tested.

To test for circularity and upward connectivity using a traditional matrix representation, requires that every proposition element in each row in a consequent matrix is inspected. If the element is instantiated the indicated column in the antecedent matrix requires examination. For each instantiated element found in the antecedent matrix an upward connection for the rule in question exists which must be traced further. If no instantiated elements are found in the indicated antecedent column no upward connection exists for the rule under consideration. This process continues until either:

1. A rule with ni upward connection is reached (i.e. a root rule).
2. The start rule is returned to in which case that rule forms part of a circular rule set.
3. The chain leads up to a circular rule set in which case circularity exists higher up in the rulebase.

With respect to binary uniform rulebases where all rules are connected and no circularity exists, to carry out inference verification, every element in each consequence matrix row and each column representing a consequent proposition in the antecedent matrix must be examined. Using the IMVER-1 representation the number of matrix element comparisons that this will require will be equal to:

$$(Np \times Nr) + Nr^2 = Nr(NP + Nr)$$

The number of comparisons required using the IMVER-2 representation is then given by:

$$(B \times Nr) + Nr^2 = Nr(B + Nr)$$

Consider the balanced rulebase given in Table 4, the number of comparisons required to establish upward connectivity and the non-existence of circular rule sets using the IMVER-1 representation will be:

$$31(63 + 31) = 2914$$

Using the IMVER-2 representation the number of comparisons will be:

$$31(2 + 31) = 1023$$

Inspection of the above expressions indicates that the advantage gained using the IMVER-2 representation is in the order of:

$$\frac{31Np}{32(Np + Nr)}$$

To establish downward connectivity requires that antecedent matrix rows and consequent matrix columns are inspected. Leaf rules will have no downward connection, however, to establish this we must inspect the entire row representing each leaf rule and the columns associated with any identified instantiated elements. In the case of body and root rules, to establish downward connectivity, we need only establish that one antecedent proposition appears in at least one consequent, thus we do not need to inspect entire antecedent rows. The total number of comparisons required to establish downward connectivity can be calculated using the expression:

$$(Nlr \times Np) + (Aar1 \times Nbr) + 2(Nlr \times Nr) + (Acc1 \times Nbr) + 3$$

Where:

- Aar1 = The average number of comparisons required to identify an instantiated element in an antecedent matrix row representing a body rule.
- Acc1 = The average number of comparisons required to identify an instantiated element in a consequent matrix column representing a body proposition.

The number of comparisons required using the IMVER-2 representation is then given by:

$$(Nlr \times B) + (Aar2 \times Nbr) + Nr(Nlr + k) + (Acc2 \times Nbr) + 3$$

Where:

Aar2 = The average number of comparisons required to identify an instantiated element in an antecedent matrix row representing a body rule.

k = A constant expressing the number of occasions that the instantiated elements of leaf rule are spread over two blocks. This value is comparatively small and approximates to 0.667×B.

Acc2 = The average number of comparisons required to identify an instantiated element in a consequent matrix column representing a body proposition. This is equivalent to Acc1 (see above).

Returning to the balanced rulebase given in Table 4 the number of comparisons required to establish downward connectivity using the IMVER-1 representation will be:

$$(16 \times 63) + (29.57 \times 14) + 2(16 \times 31) + (15.79 \times 14) + 3 = 2638$$

Note that, from empirical data, Aar1 is given as 29.57 and Acc1 is given as 15.79 for a uniform rulebase of a depth of 6. The total number of comparisons to implement inference verification will then be equal to 5552 (2914+2638).

Using the IMVER-2 representation the number of comparisons required to check for downward connectivity will be:

$$(16 \times 2) + (1.5 \times 14) + 31(16+1) + (15.79 \times 14) + 3 = 804$$

Note that Aar2 is given as 1.5 and k as 1 for a uniform rulebase of a depth of 6 (Acc2 = Acc1 = 15.79). The total number of comparisons will then be 1827 (1023+804).

Again it is difficult to compare directly the complexity expressions given in the above two sub-sections, however, some conclusions can be drawn from the results. Further, in Table 7, some empirical results are presented. The table shows the number of comparisons required to carry out inference verification on a number of balanced rulebase of different depths represented using both the traditional and binary encoded representations. Inspection of these results shows that the algorithm as implemented using the IMVER-2 binary representation requires approximately 67% fewer comparisons than the same algorithm implemented using the traditional approach.

*** Table 7 here ***

12. REMEDIAL ACTION

Having established that one or more errors or anomalies exist remedial action must be taken. A suggested methodology is given in Coenen[18]. Broadly this comprises the following steps:

1. *Global Location*

Identify the section of the rulebase that will require attention.

2. *Remedial Action*

Determine the nature of the remedial action that will be required. In a rulebased system of the form under consideration here four distinct possibilities can be envisaged, (a) adding a rule, (b) removing a rule, (c) replacing a rule and (d) modifying a rule. The distinction between (c) and (d) is that the first consists of the removal of a rule and the adding of a new rule (b and a) while the second involves attention to an individual element of a rule, for example a proposition in the consequent. This over simplifies the task, in practice a number of these possibilities will require implementation in conjunction with one another, but for illustrative purposes this classification will suffice.

3. *Local Location*

Identify the specific elements in the rulebase that will require attention as a result of the proposed change. In the above case, if the remedial action identified consists of the removal of a single rule, the rules which call and are called by the rule to be removed will require attention. Alternatively if the rule is a leaf rule the mechanism for handling *askable* propositions will require investigation. When the specific rulebase elements that might require attention have been identified the next step is to consider any further remedial actions required with respect to these elements. In the above example of the removal of a rule this will require the modification of the rules that call it and are called by it. Thus a feature of the methodology is a loop where steps 2 and 3 are repeated until the extended sub-set of the part of the rulebase affected by the initial change required has been identified.

4. *Implementation*

Implementation should be carried out in a structured manner following the path mapped out by the looping process described in 3. In this manner the required remedial actions will be implemented in a logical and sequential manner.

5. *Further verification*

Once the necessary changes have been made the rulebase must be re-verified. This verification need only be implemented on the sub-set of the rulebase which will be affected by the change. This will not just comprise the individual elements on which some maintenance action has been implemented but will also involve those elements indirectly effected by the change (in some unfortunate cases this might involve the entire rulebase). I refer to this group of elements as the jeopardised sub-set. It may be that, during this testing stage, it is realised that further maintenance is required in which case a return to step 3 should be instigated.

For a more in depth discussion on the "maintenance" of rulebase systems interested readers are referred to Coenen and Bench-Capon[16].

13. CONCLUSIONS

In this paper I have described an advanced binary encoded matrix representation which will support rulebase verification. This has been implemented in an experimental rulebase verification system, IMVER-2, which is an upgrade of an earlier system, IMVER-1. Using this advanced matrix representation, significant storage and processing savings can be made. More specifically I have conclusively demonstrated that 97% less storage capacity is required using the binary encoded representation than that required using other incidence matrix representations. With respect to the efficient processing of incidence matrices supported by the advanced binary encoded matrix representation advantages of 90% and 67% for structural and inference verification respectively are attained using binary uniform rule bases. Further experiments indicate that this is in fact a "worst case" scenario. Where the ratio between the average number of antecedent and consequent propositions for the rules in a rulebase is greater than 2:1 (as is the case for most genuine rulebases) the processing advantage gained using the IMVER-2 representation increases proportionatly. This observation has been born out by successful tests using both genuine and artificial rulebases.

To date the advanced binary encoded incidence matrix representation described here has only been applied to propositional rulebases. Current investigations are concentrated on experiments to identify an enhanced binary encoded representation which will support the decomposition of propositions into their constituent parts. It is hoped that this avenue of research will allow for rulebase verification at the predicate, rather than the propositional, rule level.

REFERENCES

1 **Coenen, F P, Taleb-Bendiab, A and Forster, R** 'Verification of Rule-bases Using Incidence Matrices: The IMVER System' *in* **Rzevski, G, Pastor, J and Adey, R A (Eds.)** *Applications of Artificial; Intelligence in Engineering VIII, Vol 1: Design, Methods and Techniques, Proc. AIENG'93* Computational Mechanics Publications & Elsevier Applied Science, London (1993) pp248-260

2 **Coenen, F P and Forster, R (1993)** 'Matrix Techniques for Rulebase Verification' *in* **Bramer, M A and Macintosh, A L (Eds.)** *Research and Development in Expert Systems X, Proc. ES'93* (1993) pp235-247

3 **Satre, T W and Massey, J G** 'Expert System Verification and Validation, Part I: Defining the Concepts' *in* **Rzevski, G and Adey, RA (Eds.)** *Applications of Artificial Intelligence in Engineering VI,* Computational Mechanics Publications, Southampton, (1991) p859-872

4 **Balci, O and Sargent, R G** 'A Bibliography of the Credibility and Validation of Imulation and Mathematical Models' *Simuletter* Vol 15 No 3 (1984) p15-27

5 **Boehm, B W** *'Software Engineering Economics'* Prentice-Hall, New York (1981)

6 **Charles, E** 'Checking Knowledge Bases for Inconsistencies and Other Anomalies' *Knowledge-Based Systems Verification, Validation and Testing, Workshop Notes from the 9th National Conference on Artificial Intelligence, AAAI'91* Anaheim CA (1991)

7 **Cragen, B J and Steudel, H J** 'A Decision-Table-based Processor for Checking Completeness and Consistency in Rule-Based Expert Systems' *International Journal of Man-Machine Studies* Vol 26 (1987) pp633-648

8 **Ginsberg, A** 'A New Approach to Checking Knowledge Bases for Inconsistency and Redundancy' *Proc 3rd Annual Conference Expert Systems in Government* IEEE (1987) pp102-111

9 **Chang, C L, Combs, J B and Stachowitz, R A** 'A Report on the Expert Systems Validation Associate (EVA)' *Experts Systems with Applications* Vol 1 No 3 (1991) pp219-230

10 **Coenen, F P and Bench-Capon, T J M** 'KBS Maintenance Validation Using Simulation' *in* **Grierson, D E, Rzevski, G and Adey, R A (Eds.)** *Applications of Artificial Intelligence in Engineering VII,* Computational Mechanics Publications and Elsevier Applied Science, London (1992) pp215-228.

11 **Agarwal, R and Tanniru, M** 'A Petri-Net Based Approach for Verifying the Integrity of Production Systems' *Knowledge-Based Systems Verification, Validation and Testing, Workshop Notes from the 9th National Conference on Artificial Intelligence, AAAI'91* Anaheim CA (1991)

12 **Tsai, J J P and Jang, H C** 'A Framework for Knowledge-based Systems Verification' *Proc IEEE International Conference on Systems, Man and Cybernetics, Vol 2* IEEE New York (1992) pp1700-1705

13 **Botten, N and Raz, T** 'Knowledge Base Verification Using Matrices' *Proc 4th International Conference on Industrial and Engineering Applications of Artificial Intelligence and Expert Systems* (1993) pp91-98

14 **Botten, N** 'Complex Knowledge Base Verification Using Matrices' *in* **Belli, F and Radermacher, F J (Eds.)** *Lecture notes in Artificial Intelligence 604: Proceedings of the Fifth International Conference on Industrial and Engineering Applications of Artificial Intelligence and Expert Systems* Springer-Verlag, Berlin (1992) pp225-235

15 **Preece, A D and Shinghal, R** 'DARC: A Procedure for Verifying Rule-Based Systems' *in* **Liebowitz, J (Ed.)** *Expert Systems World Congress Proceedings, Volume 2* Pergamon Press (1991) pp971-979

16 **Coenen, F P and Bench-Capon, T J M** *Maintenance of Knowledge based Systems.* Academic Press, London (1993)

17 **Merabti, M, Hutchinson, D and Taleb-Bendiab, A** *Proc. International conference on Intelligent Systems* (1992) pp218-226

18 **Coenen, F P** 'A Methodology for the Maintenance of Knowledge based Systems' *in* **Niku-Lari, A (Ed.)** *EXPERSYS-92 (Proccedings)* IITT-International, Gournay sur Marne, France, (1992) pp171-176

```
(0)   ab ac => aa   (8)    ar as => ac
(1)   ad ae => ab   (9)    at au => ar
(2)   af ag => ad   (10)   av aw => at
(3)   ah ai => af   (11)   ax ay => au
(4)   aj ak => ag   (12)   az ba => as
(5)   al am => ae   (13)   bb bc => az
(6)   an ao => al   (14)   bd be => ba
(7)   ap aq => am
```

-- --

(0)	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
(1)	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
(2)	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
(3)	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
(4)	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
(5)	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
(6)	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
(7)	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
(8)	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0
(9)	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0
(10)	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0
(11)	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0
(12)	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0
(13)	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0
(14)	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1

-- --

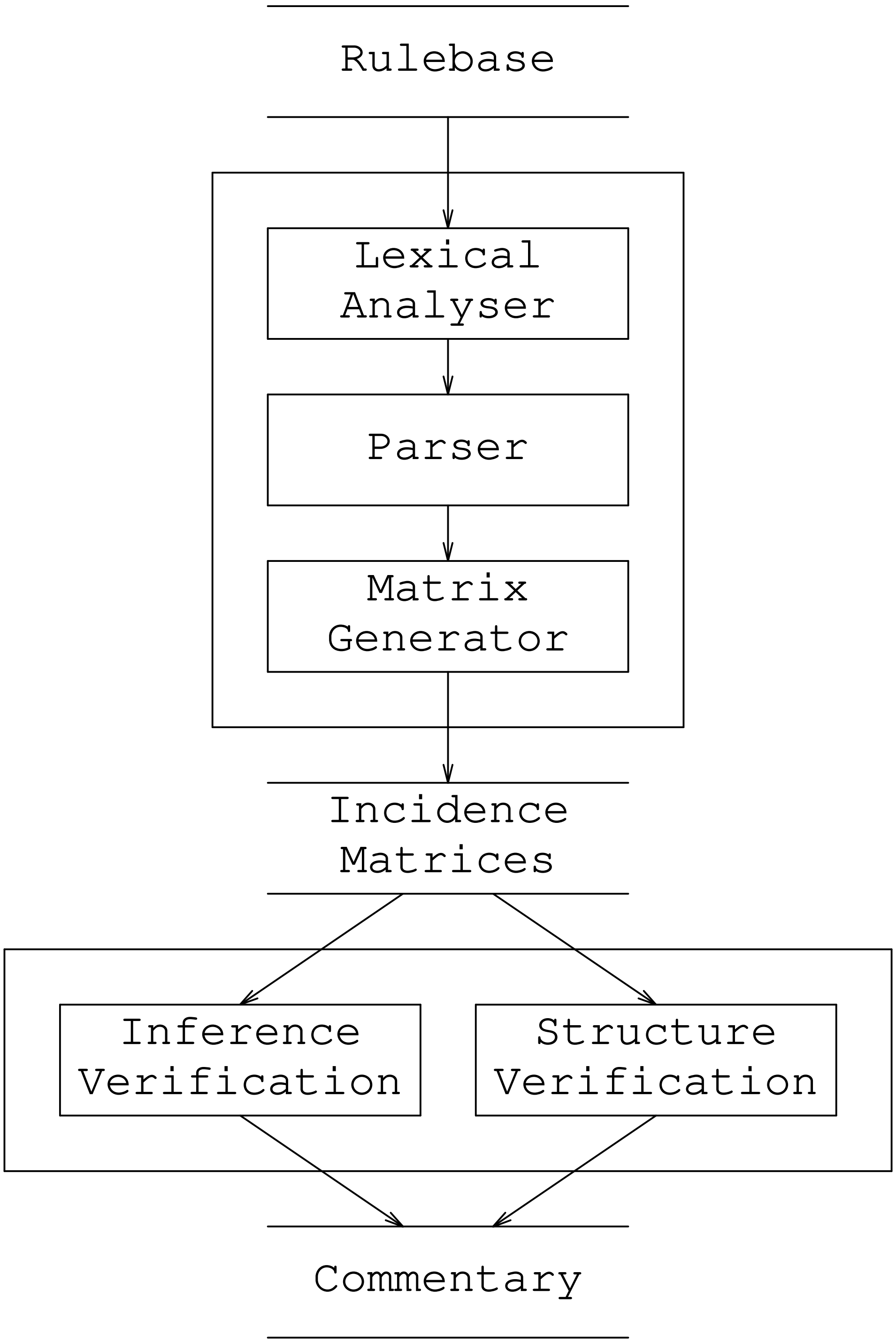
(0)	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
(1)	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
(2)	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
(3)	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
(4)	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
(5)	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
(6)	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
(7)	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
(8)	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
(9)	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0
(10)	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0
(11)	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0
(12)	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0
(13)	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0
(14)	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0

```
(0)    ab ac => aa   (16)   bh bi => ac
(1)    ad ae => ab   (17)   bj bk => bh
(2)    af ag => ad   (18)   bl bm => bj
(3)    ah ai => af   (19)   bn bo => bl
(4)    aj ak => ah   (20)   bp bq => bm
(5)    al am => ai   (21)   br bs => bk
(6)    an ao => ag   (22)   bt bu => br
(7)    ap aq => an   (23)   bv bw => bs
(8)    ar as => ao   (24)   bx by => bi
(9)    at au => ae   (25)   bz ca => bx
(10    av aw => at   (26)   cb cc => bz
(11)   ax ay => av   (27)   cd ce => ca
(12)   az ba => aw   (28)   cf cg => by
(13)   bb bc => au   (29)   ch ci => cf
(14)   bd be => bb   (30)   cj ck => cg
(15)   bf bg => bc
```

```
(0)  3221225472 0          = 11000000000000000000000000000000 00000000000000000000000000000000
(1)  402653184  0          = 00011000000000000000000000000000 00000000000000000000000000000000
(2)  100663296  0          = 00000110000000000000000000000000 00000000000000000000000000000000
(3)  25165824   0          = 00000001100000000000000000000000 00000000000000000000000000000000
(4)  6291456    0          = 00000000011000000000000000000000 00000000000000000000000000000000
(5)  1572864    0          = 00000000000110000000000000000000 00000000000000000000000000000000
(6)  393216     0          = 00000000000001100000000000000000 00000000000000000000000000000000
(7)  98304      0          = 00000000000000011000000000000000 00000000000000000000000000000000
(8)  24576      0          = 00000000000000000110000000000000 00000000000000000000000000000000
(9)  6144       0          = 00000000000000000001100000000000 00000000000000000000000000000000
(10) 1536       0          = 00000000000000000000011000000000 00000000000000000000000000000000
(11) 384        0          = 00000000000000000000000110000000 00000000000000000000000000000000
(12) 96         0          = 00000000000000000000000001100000 00000000000000000000000000000000
(13) 24         0          = 00000000000000000000000000011000 00000000000000000000000000000000
(14) 6          0          = 00000000000000000000000000000110 00000000000000000000000000000000
(15) 1          2147483648 = 00000000000000000000000000000001 10000000000000000000000000000000
(16) 0          1610612736 = 00000000000000000000000000000000 01100000000000000000000000000000
(17) 0          402653184  = 00000000000000000000000000000000 00011000000000000000000000000000
(18) 0          100663296  = 00000000000000000000000000000000 00000110000000000000000000000000
(19) 0          25165824   = 00000000000000000000000000000000 00000001100000000000000000000000
(20) 0          6291456    = 00000000000000000000000000000000 00000000011000000000000000000000
(21) 0          1572864    = 00000000000000000000000000000000 00000000000110000000000000000000
(22) 0          393216     = 00000000000000000000000000000000 00000000000001100000000000000000
(23) 0          98304      = 00000000000000000000000000000000 00000000000000011000000000000000
(24) 0          24576      = 00000000000000000000000000000000 00000000000000000110000000000000
(25) 0          6144       = 00000000000000000000000000000000 00000000000000000001100000000000
(26) 0          1536       = 00000000000000000000000000000000 00000000000000000000011000000000
(27) 0          384        = 00000000000000000000000000000000 00000000000000000000000110000000
(28) 0          96         = 00000000000000000000000000000000 00000000000000000000000001100000
(29) 0          24         = 00000000000000000000000000000000 00000000000000000000000000011000
(30) 0          6          = 00000000000000000000000000000000 00000000000000000000000000000110
```

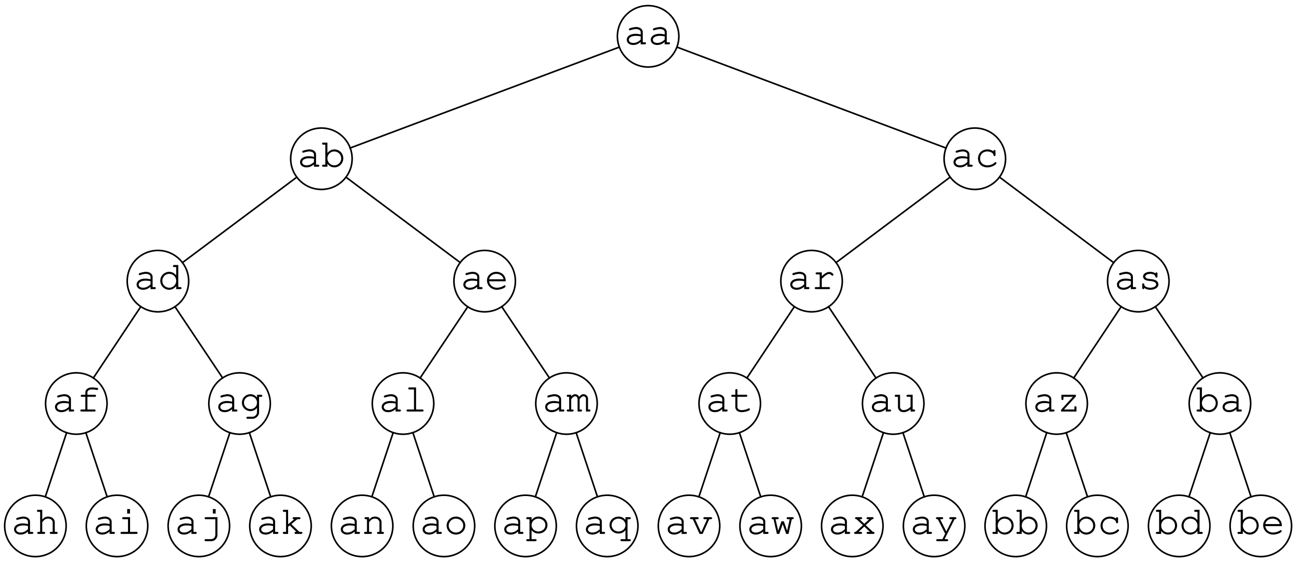
DEPTH	Nr	Np	IMVER-1 Represen-tation (K)	IMVER-2 Represen-tation(K)	Difference	% Differ-ence
5	15	31	2.2	0.3	1.9	85.94
6	31	63	19.8	2.4	17.4	87.75
7	63	127	166.7	18.2	148.5	89.08
8	127	255	1365.5	138.9	1226.6	89.83
9	255	511	11054.1	1081.2	9972.9	90.22
10	511	1023	88954.9	8521.4	80433.4	90.42
11	1023	2407	713732.1	67644.8	646087.2	90.52
12	2407	4095	5718237.2	539024.2	5179212.9	90.57
13	4095	8191	45779435.5	4303599.4	41475836.2	90.60
14	8191	16383	366369669.1	34394304.0	331975365.1	90.61

DEPTH	Nr	Np	IMVER-1 Represen-tation	IMVER-2 Represen-tation	Differ-ence	% Differ-ence
2	1	3	9	4	5	55.56
3	3	7	59	23	36	61.02
4	7	15	296	102	194	65.54
5	15	31	1316	426	890	67.63
6	31	63	5552	1827	3725	67.09
7	63	127	22832	7572	15260	66.84
8	127	255	92672	30725	61947	66.85
9	255	511	373568	123589	249979	66.92
10	511	1023	1500416	495872	1004544	66.95
11	1023	2047	6014720	1987819	4026901	66.95


Rulebase
Lexical Analyser
Parser
Matrix Generator
Incidence Matrices
Inference Verification
Structure Verification
Commentary

--

--


aa
ab
ac
ad
ae
ar
as
af
ag
al
am
at
au
az
ba
ah
ai
aj
ak
an
ao
ap
aq
av
aw
ax
ay
bb
bc
bd
be