Problem Solving Environment Based on Knowledge Based System Principles

A.M. Martinez-Enriquez[1], G. Escalada-Imaz[2], and Aslam Muhammad[3]

[1] Department of Computer Science, Centre of Research and Advanced Studies (CINVESTAV-IPN), Mexico D.F., Mexico
[2] Artificial Intelligence Research Institute (IIIA-CSIC), Spanish Council for Scientific Research (CSIC), Barcelona, Spain
[3] Department of CS & University of Engineering and Technology, Lahore, Pakistan
ammartin@cinvestav.mx, gonzalo@iiia.csic.es, maslam@uet.edu.pk

Abstract. This paper describes how to solve numerical problems by a non computer user based on Knowledge Based Systems (KBSs) principles. The aims of our approach is to handle numerical information of the problem, by using Backward Logical inferences, deducing whether or not the required mathematical composition functions exist. We implemented a backward inference algorithm which is bounded by $O(n)$, n being the size of KBS. Moreover the inference engine proceeds in a top-down way, so scans a small reduced search space compared to those of the forward chaining algorithms. Our experimental example deals with some statistical analysis and its application to clustering and concept formation.

Keywords: backward inference engine, concept formation.

1 Introduction

This paper focuses on the deduction and the composition of mathematical functions including in the inference rules along with an efficient inferences engine (IE) that access them to satisfy a user-query (Q). IE combines several deductive rules to resolve explicitly asked function including in the query (Q). In this way, we designed a knowledge based system including the binary logical relation of specific solution. The deduction of these relationships relies on an efficient Backward Inference Engine detailed in Section 3.

We have chosen as an experimental framework, the analysis and resolution of statistical problems that can be applied into different domains like clustering, pattern recognition. Our logical framework detailed in Section 2, is composed by: a base of facts (BF), a knowledge base of implication rules (BR), a user query noted (Q), and an Inference Engine (IE) responsible to determine/deduce whether or not a particular statistical function exists (Section 3).

The paper is organized as follows: Section 2 describes our main framework for discovering statistical functions using logical inference. Also, some examples of application are presented. Section 3 explains the inference mechanism. Section 4 analyzes the experimental results. Section 5 concludes and draws future work.

J.F. Martínez-Trinidad et al. (Eds.): MCPR 2014, LNCS 8495, pp. 81–91, 2014.

2 System Framework

This is the standard case where it is necessary to know if a client query (Q) can be deduced from BF and BR : $BF \wedge$ BR $\vdash Q$. The elements of Q, BF, and BR form a set of propositional Horn clauses. This classical deduction problem, in our studied case, can be formulated as follows:

Given

- *BF*: a set of propositions $\{mf_1, \ldots, mf_j, \ldots, mf_m\}$, where mf_j represents the confirmed existence of a particular mathematical function that has been implemented and tested, such that each proposition mf_j is evaluated to either True or False;
- *BR*: a set of implication rules $\{rf_1, \ldots, rf_i, \ldots, rf_n\}$, where a set of rf_i rules models a composition solution. Each rf_i, $mf_1 \wedge \ldots \wedge mf_n \rightarrow mf_i$, indicates that if the n mf in the antecedent exist then the existence of this statistical function mf_i in the consequent of this rule (rf_i) exist too. In fact, each registered rf_i into BR is true because the functionality it represents has been implemented, tested, and verified before; and
- *Q*: a user query representing whether a mf_s composition exists.
  This query is expressed by a disjunction of conjunctions of propositions: $Q = (q_{11} \wedge \ldots \wedge q_{1n1}) \vee \ldots \vee (q_{k1} \wedge \ldots \wedge q_{knk})$

Goal: To determine whether or not Q can be deduced from BF and BR performing sound inferences: BF $\wedge$ BR $\vdash$ Q.

In order to achieve this goal, we proposed and implemented a Backward Inference Engine being linear and logically correct, described later (Section 3).

2.1 Mathematical Functions

Each required useful functionality is provided by IE, which can be grouped as :

- **Input/Output.** In order to solve heterogeneous data mismatch, we implemented some data-format transforming functions for reading and exporting files containing numbers in different formats: ASCII, integer, float or double. ASCII files can have numbers laid out either with a number per line, a line with various numbers separated with a space, or several lines each with several numbers. From the inference point of view, this data transformation is represented by the following pattern. Next we write some particular rules.

  $$experimentDta \wedge treatDta \rightarrow stdDta$$

  Where: *experimentDta* stands for the existence of the input data in any layout format provided by the client user.
  treatDta is the existing mf_j which processes the input data : *experimentDta*
  stdDta refers to the existence of the transformed data. On the other hand, the proposition *stdDta* represents the existence of the output of the function (*treatDta*) which transforms *experimentDta* to the required format to be processed (if needed).

The following rules represent examples of the input data transformation (experimentDta) with numerical binary data into numerical ASCII string, which is the format used by our mathematical functions.

$$integerD \wedge binaryDoubleBigEndian \rightarrow ASCIIstring$$
$$integer4byts \wedge binaryIntegerBigEndian \rightarrow ASCIIstring$$
$$shortD \wedge binaryShortBigEndian \rightarrow ASCIIstring$$
$$floatD \wedge binaryshortLitleEndian \rightarrow ASCIIstring$$
$$doubleD \wedge binaryDoubleBigEndian \rightarrow ASCIIstring$$
$$long8bytD \wedge binaryLongBigEndian \rightarrow ASCIIstring$$
$$long8bytLittleED \wedge binaryLongLittleEndian \rightarrow ASCIIstring$$

As explained before $integerD$, $integer4byts, \ldots$ idem $ASCIIstring$ are propositions that test the existence of these input/output data type.

Similarly $binaryDoubleBigEndian$, $binaryIntegerBigEndian, \ldots$ are propositions testing the existence of functions mf_s registered in our infrastructure.

- **Plotting** draws data. For instance, giving two sets of input data, a regression line can be calculated and graphical representation can be displayed, as described by the following rule.

$$stdDta - ofX \wedge stdDta - ofY \wedge regressline \rightarrow plotRE$$
$$stdDta - ofX \wedge y - axisIntercpt \wedge regrCoeff \rightarrow regressline$$
$$stdDeviationofX \wedge stdDeviationofY \wedge correltCoeffofXY \rightarrow regrCoeff$$
$$meanofX \wedge meanofY \wedge regrCoeff \rightarrow y - axisIntercpt$$

Where $stdDta{-}ofX$, $stdDta{-}ofY$ represent the existence of two sets of input data, $regressline$ stands for the regression line equation, $y{-}axisIntercpt$ is the ordinate, $regrCoeff$ represents the correlation coefficient (see Table 1), and $plotRE$ the resulting plot.

- **Mathematical functions** compute numerical measures and analysis. The symbolic name associated to mf_s is a proposition stored in BF which validates the existence of an implemented and tested mf_s, such that $meanOpr$, $modeOpr$, $medianOpr$.

Our test bed environment solves problems using inference rules, e.g., giving the experimental data as required by $stdDta$, it is possible to calculate several numerical functions associated to the following classical implication rules, such that if a user asks for the standard Deviation (R_5), IE deduces that it is required to apply firstly $\{[R_4], [R_1]\}$:

$[R_1] stdDta \wedge meanOpr \rightarrow mean$ $\quad$ $[R_2] stdDta \wedge modeOpr \rightarrow mode$
$[R_3] stdDta \wedge mediaOpr \rightarrow media$ $\quad$ $[R_4] stdDta \wedge mean \wedge varianceOpr \rightarrow variance$
$[R_5] squareRoot \wedge variance \rightarrow stdDeviation$

Moreover, users are able to store an inference rule into BR with the set of deduced propositions, the rule can be named as user wants e.g., $ExperimentalApp1$. This rule can be executed as many times it is required for the same kind of experimentation analysis :

$$mean \wedge mode \wedge stdDeviation \rightarrow ExperimentalApp1$$

Giving two sets of data several rules take place, the semantic is as follows:

$$[R6] variance - ofX \wedge variance - ofY \wedge covarianceOpr \rightarrow covariance - ofXY$$

Semantic : Once $variance - ofX$, $variance - ofY$ have been calculated (they are True), and the function *covarianceOpr* has been implemented and tested (it is True), so the computation of $covariance - ofXY$ is launched.

2.2 Combining Logical and Numerical Information

A proposition P in KBS is evaluated to true iff its associated cells memory $[P]$ contains a numerical value x. If the cell memory $[P]$ is undefined, P is evaluated

Table 1. Some statistics functions and related inference rules. Input data are: d, X, and Y. Lowered indexes x and y refer to values in sets X and Y respectively, N is the number of data, and $N - a$ grades of freedom.

Statistics Equation	Related Rules
Mean $M = \frac{1}{N} \sum_{i=1}^{N} d_i$	$stdDta \wedge meanOpr \rightarrow mean$ $stdDta \wedge modeOpr \rightarrow mode$ $stdDta \wedge medianOpr \rightarrow median$
Standard deviation $SD = \sqrt{\frac{1}{N-1} \sum_{i=1}^{N} (d_i - M)^2}$	$stdDta \wedge mean \wedge varianceOpr \rightarrow variance$ $sqrt \wedge variance \rightarrow stdDeviaton$
Mean of X	$stndDta - ofX \wedge meanOpr \rightarrow meanofX$
Standard deviation of X	$stdDta - ofX \wedge meanofX \rightarrow stdDeviation - ofX$
Standard deviation of Y	$stdDta - ofY \wedge meanofY \rightarrow stdDeviation - ofY$
Correlation coefficient $r = \frac{\sum_{i=1}^{N} (X_i Y_i) - N M_x M_y}{(N-1) SD_x SD_y}$	$stdDeviatn - ofX \wedge stdDeviatn - ofY \wedge$ $grdsFreedom \wedge rOpr \rightarrow correltCoeff - ofXY$
Student's test t $t = r\sqrt{\frac{N-2}{1-r^2}}$	$stdDta - ofX \wedge stdDta - ofY \wedge$ $correltCoeff - ofXY \wedge tOpr \rightarrow stdentTtest$
Regression line $Y = a + bX$	$stdDta - ofX \wedge stdDta - ofY \wedge YOpr$ $y - axsIntercpt \wedge regrCoeff \rightarrow regressline$
Regr. coefficient(slope) $b = r\frac{SD_y}{SD_x}$	$stdDeviatn - ofX \wedge stdDeviatn - ofY \wedge$ $correltCoeff - ofXY \wedge coeRegOpr \rightarrow regrCoeff$
y-axisIntercpt(ordinate) $a = (M_y - bMx)$	$mean - ofX \wedge mean - ofY \wedge regrCoeff \wedge$ $aOpr \rightarrow y - axsIntercpt$
Residual Std. deviation $D_{res} = \sqrt{\frac{SD_y^2 (1-r^2)(n-1)}{n-2}}$	$stdDta - ofY \wedge stdnDeviaton - ofY \wedge DresOpr \wedge$ $correltCoeff - ofXY \rightarrow resStdDeviation$
Slope's std. error $SE_b = \frac{D_{res}}{\sqrt{\sum_{i=1}^{N} (X_i - M_x)^2}}$	$stndDta - ofX \wedge meanofX \wedge resStdDeviation \wedge$ $SEbOpr \rightarrow slopeStdErr$

to false. The consequent of an implication rule P is evaluated to true iff all its antecedent variables P' are true, i.e. [P'] contains a numerical value x, and iff the function f representing the logical implication is computationally calculable. The value assigned to its cell memory $[P]$ is derived from the computing function f applied upon the numerical values x, i.e.$f(x_1, x_2, \ldots, x_n) = y$ associated to its antecedent propositions. Once the computation of the function f has been performed, the truth of P is confirmed and its cell memory $[P]$ is set to y.

A rule within BR models a fragment of a solution. That means that generally the whole solution is represented by a set of rules and facts within KBS.

3 Inference Mechanism

We can distinguish two type of strategies in KBSs to deduce new facts:

- **Forward** chaining precedes from the knowledge base of facts (BF) and uses the knowledge base of rules (BR) to infer a client query (Q). During the inference process the consequent P of the rule is added to BF, since all propositions of the antecedent of the rule: $P1, \ldots, Pn$ are already in BF. It is means, a rule is triggered whenever its antecedent has been previously deduced, i.e. when all propositions in the antecedent have been added to BF. Case in which the consequent is added to BF.
- **Backward** chaining begins by Q. Instead of knowing if the consequent P is deduced from BF and BR, the problem is to know if the antecedent of the rule $P1, ...Pn$ whose consequent is P, can be deduced from BF and BR. In other words, in an inference step it is asked whether a proposition P can be deduced which leads to ask whether the propositions $P_1, P_2, \ldots, P_k$ in the antecedent of a rule whose consequent is P can be deduced.

Some linear algorithms to solve the problem of deducing a goal from BR and BF have been proposed in [1] [3] [4]. All of them rely on a bottom-up control strategy; hence they develop a larger search space than that required by a top-down search focused on the explicitly given goal. Indeed, Forward Chaining Algorithms deduce all the implicit knowledge that is logical consequence of the BF and BR. Contrary to this, Backward Chaining processes only inferences that are goal directed, i.e. they only scan rules and facts related to the current query.

The Backward Inference Engine algorithm (BackwardIE) employed here has a strict linear complexity and proceeds in top-down way [2]. Due to the top-down strategy, a goal is proved or disproved with a reduced search space.

The BackwardIE algorithms in KBSs remain however quite crude: usually a priori bounded depth-first or backtrack search strategies relying on the assumption of a loop-free search space. Because of the indirect-recursive problem (as in the two rules $Q \wedge R \rightarrow P$, $P \wedge S \rightarrow Q$) such an assumption is rarely met.

The only existing linear backward algorithm scanning a search space considering recursion in the BR is that presented in [2]. The formal description and proofs of the logical and complexity properties is beyond the space of the present article. Thus we will give a rough description of the strategy to scan the search space modeling the deduction backward problem by means of a given example.

The proof of a goal can be represented as a search problem in an And/Or graph G. As a running example, let us consider the following set of rules:

$P_2 \wedge P_8 \rightarrow P_1$, $P_9 \wedge P_{10} \rightarrow P_1$ $P_{11} \wedge P_{12} \rightarrow P_{10}$ $P_5 \wedge P_3 \wedge P_6 \rightarrow P_2$
$P_2 \wedge P_4 \rightarrow P_3$, $P_2 \rightarrow P_4$ $P_7 \wedge P_1 \rightarrow P_4$
$P_3 \wedge P_{14} \rightarrow P_{13}$ $P_4 \wedge P_{10} \rightarrow P_{13}$ $P_{15} \wedge P_{16} \rightarrow P_{17}$

Each proposition P in BR is associated with a node labeled P in the And/Or graph G and each rule is associated with a connector linking its consequent P with its set of antecedents $(P_1, P_2, \ldots, P_k)$. The propositions P in BF are marked as true nodes. Q is also a connector in G (see Figure 1). The proof of $Q = ((P_1 \wedge P_{13}) \vee P_{17})$ with $BF = \{P_5, P_7, P_9, P_{11}, P_{12}, P_{15}\}$ is a search in the graph shown in Figure 1. The below data structures are used to search in the And/Or graph. To deduce Q we need to search a tree T rooted at Q with the property that all its nodes P are either in BF or have a sub-tree routed in P in T. Its proof is in [2].

c(P) is the set of connectors issued from P.
r(C) is the node from which C is issued.
s(C) the set of successors nodes of r(C) along connector C.

IE is based on a recursive depth-first search. For each node P visited, the set c(P) of connectors outgoing from P is sequentially examined (Or-Expand) until one leads a solution. Visiting a connector C involves the expansion of all non leaf successor nodes in s(C)(And-Expand) until one fails to lead a solution. Initially nodes in BF are marked SOLVED, and all other nodes are marked OPEN. A node P being expanded is marked EXPANDING until either: **a)** P has a solved connector: one along which every successor node is SOLVED: In that case P is marked SOLVED; or **b)** is shown to have no such connector: P is marked FAILED.

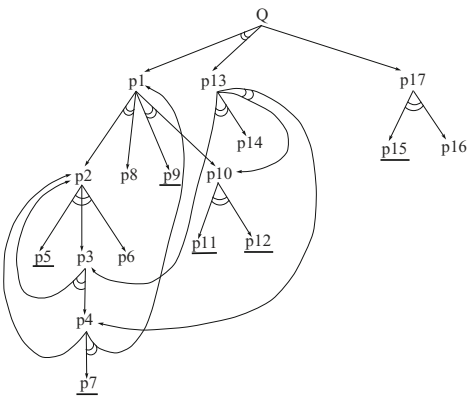

Fig. 1. And/Or graph G

There is a loop if during the expansion of a node P_i one of its successors P_j is found *EXPANDING*. The sequence of nodes in the loop $[P_j, \ldots, P_k, P_{k+1}, \ldots, P_j]$ is a last-in first-out stack; P_j will be called its *source* node. A node P_k is said to be *confined* on the loop if its connector on this loop is solved except for the successor node P_{k+1} in the loop (marked EXPANDING) and if P_k has no other solved connector. If all nodes in the loop are confined, then all will be marked FAILED. If P_i is confined and later on P_j is found SOLVED, then P_i will also be marked SOLVED. This marking of confined nodes as SOLVED is propagated backward up to to the source node. In general, a node P can have more than one EXPANDING successor: it may be on several overlapping loops. If it is confined for one or several loops, then the algorithm backtracks, leaving P EXPANDING. The final marking of P will depend on that of its EXPANDING successors. Two data structures will be needed for that:

- **Followers(P)**: nodes on which the marking of confined node P depends, arranged as a formula in disjunctive form, e.g. **Followers**$(P_4) = P_2 \vee P_1$; P_4 is solved iff either P_2 or P_1 is solved (Figure 1).
- **Leaders(P')**: list of predecessor nodes of P' that are confined on the loops, e.g. **Leaders**$(P_2) = (P_3, P_4)$

If P' appears in **Followers**(P) then P belongs to **Leaders**(P').

Let us illustrate the use of these structures on the running example (Figure 1). Node Q and then successively P_1, P_2, P_3 are expanded. At this step P_2 is found EXPANDING. Since P_3 is not yet proved to be confined on the loop, we proceed to the next recursion. Two successors of P_4 are EXPANDING, P_7 is SOLVED, and thus P_4 is confined. We set **Followers**(P_4)=$P_2 \vee P_1$, **Leaders**(P_1)=**Leaders**(P_2)=(P_4), and backtrack. Node P_3 is now found to be confined: **Followers**(P_3)=$P_2 \wedge P_4$. **Leaders**(P_2)=(P_3, P_4), **Leaders**(P_4)=(P_3). Back to P_2: we find P_6 FAILED; thus P_2 is also marked FAILED. **Leaders** of P_2 are not examined, and nothing will happen to pending nodes.

We backtrack to P_1, skip P_8 and visit successively P_9, P_{10}, P_{11} and P_{12}. P_{10} and then P_1 are marked SOLVED. Before backtracking to Q, node P_4 in **Leaders**(P_1) is examined:

Followers(P_4)=$(P_2 \vee P_4)$ leads us to mark P_4 SOLVED. Thus P_3 in **Leaders**(P_4) is also examined: **Followers**(P_3)=$P2 \wedge P_4$, so P_3 cannot be SOLVED; it remains EXPANDING. This ends the expansion of P_2. Since it is the last source node, all pending nodes that remain EXPANDING (only P_3) are marked FAILED.

Finally node P_{13} is expanded, its successor P_3 is FAILED, P_{14} is skipped and P_4 and P_10 are SOLVED; P_{13} is marked SOLVED. A solution is found, and the rest of the graph is not examined.

To summarize, the algorithm proceeds as follows: if the source node P_j of a loop is marked SOLVED, then nodes in **Leaders**(P_j) are sequentially visited. For every node P_i in this list, the formula **Followers**(P_i) is evaluated. If it leads to P_i SOLVED, then the list **Leaders**(P_i) is in turn examined. Nodes that remain EXPANDING after the end of the expansion of the last source node must be marked FAILED. For that, all nodes confined in a loop are put in a list called

Pending. The different evaluations of the formulas **Followers**(P_i) can be done in an efficient way with suitable data structure (see [2]).

The result obtained from IE is as follows:

Input

$Q = ((P_1 \wedge P_{13}) \vee P_{17})$, $BF = \{P_5, P_7, P_9, P_{11}, P_{12}, P_{15}\}$, P_i marked as SOLVED
$BR = \{R_0, \ldots, R_{11}\}$, where P_0 represents Q, i.e., $\{R_0, R_1\}$

$[R_0]$ $P_1 \wedge P_{13} \rightarrow P_0$ $[R_1]$ $P_{17} \rightarrow P_0$ $[R_2]$ $P_2 \wedge P_8 \rightarrow P_1$ $[R_3]$ $P_9 \wedge P_{10} \rightarrow P_1$
$[R_4]$ $P_5 \wedge P_3 \wedge P_6 \rightarrow P_2$ $[R_5]$ $P_2 \wedge P_4 \rightarrow P_3$ $[R_6]$ $P_2 \rightarrow P_4$ $[R_7]$ $P_7 \wedge P_1 \rightarrow P_4$
$[R_8]$ $P_4 \wedge P_{10} \rightarrow P_{13}$ $[R_9]$ $P_3 \wedge P_{14} \rightarrow P_{13}$ $[R_{10}]$ $P_{15} \wedge P_{16} \rightarrow P_{17}$ $[R_{11}]$ $P_{11} \wedge P_{12} \rightarrow P_{10}$

Output

Return from : $AND_{EXTENSION}(R_6$: $P_2 \rightarrow P_4)$
{ $Result_{AND} = Cycle; r_{UNDEFINED} = \{P_2\}$ }
Return from : $AND_{EXTENSION}(R_7 : P_7 \wedge P_1 \rightarrow P_4)$
{$Result_{AND} = Cycle; r_{UNDEFINED} = \{P_1\}$ }
call to: $OR_{EXTENSION}(P_4)$
{ $Val = EXT; Order = 4; PConnector(P_4) = \{R_6, R_7\}$; $C_{SUCCESSOR}(P_4)$={ } }
Return from : $AND_{EXTENSION}(R_5 : P2 \wedge P_4 \rightarrow P3)$
{$Result_{AND} = Cycle; r_{UNDEFINED} = \{P_4, P_2\}$}
callto : $OR_{EXTENSION}(P_3)$;
{$Val = EXT; Order = 3; PConnector(P_3) = \{R_5\}$; $C_{SUCCESSOR}(P_3)$={}}
call to: $OR_{EXTENSION}(P_6)$;
{$Val = F; Order = 5; PConnector(P_6) = \{\}$; $C_{SUCCESSOR}(P_6)$={ } }
Return from : $AND_{EXTENSION}(R_4 : P5 \wedge P_3 \wedge P_6 \rightarrow P_2)$ {$Result_{AND} = F$}
call to : $OR_{EXTENSION}(P_2)$;
{ $Val = F; Order = 2; PConnector(P_2) = \{R_4\}$; $C_{SUCCESSOR}(P_2) = \{R_6, R_5\}$ }
Return from : $AND_{EXTENSION}(R_2 : P_2 \wedge P_8 \rightarrow P1)$ {$Result_{AND} = F$}
Return from : $AND_{EXTENSION}(R_{11} : P_{11} \wedge P_{12} \rightarrow P_{10})$ {$Result_{AND} = V$}
call to : $OR_{EXTENSION}(P_{10})$
{ $Val = V; Order = 6; PConnector(P_{10}) = \{R_{11}\}$; $C_{SUCCESSOR}(P_{10}) = \{\}$ }
Return from : $AND_{EXTENSION}(R_3 : P_9 \wedge P_{10} \rightarrow P_1)$ {$Result_{AND} = V$ }
call to : $OR_{EXTENSION}(P_1)$
{ $Val = V; Order = 1; PConnector(P_1) = \{R_2, R_3\}$; $C_{SUCCESSOR}(P_1) = \{R_7\}$ }
Return from : $AND_{EXTENSION}(R_8 : P_4 \wedge P_{10} \rightarrow P_{13})$ { $Result_{AND} = V$}
call to : $OR_{EXTENSION}(P_{13})$;
{ $Val = V; Order = 7$; $PConnector(P_{13}) = \{R_8, R_9 : (P_3 \wedge P_{14} \rightarrow P_{13})\}$;
$C_{SUCCESSOR}(P_{13}) = \{\}$}
Return from : $AND_{EXTENSION}(R_0 : (P_1 \wedge P_{13} \rightarrow P_0) \vee (P_{17} \rightarrow P_0))$
{ $Result_{AND} = V$ }
call to : $OR_{EXTENSION}(P_0)$
{ $Val = V; Order = 0; PConnector(P_0) = \{R_0, R_1\}$; $C_{SUCCESSOR}(P_0) = \{\}$ }

Thus, P_0, i.e., Q is deduced from BR and BF: BR $\wedge$ BF $\vdash Q$

The BackwardIE algorithm was implemented in ANSI C, it runs either on Windows or Unix. The numerical calculation can be executed either in sequential or in concurrent, or in parallel or from different sites, according with the architecture of the problem solving environment.

4 Numerical Solution

The aim of our approach is to solve numerical problems for users who not necessary know how to treat them. Once IE deduces the sequence of mathematical functions (mf_s) that deduce the proposed (Q), this sequence forms a composition of services, ready to be computed. Suppose we have a cluster of observations that are characterized by several symbolic and numerical attributes-values, and a new observation arrives, but this new item has some missing values, because it has been impossible to measure them, so it could be interesting to calculate approximated values. Thus the system takes into account the maximum number of known attributes-values that makes similar the new observation with a cluster. Then the most similar cluster is processed statistically, for instance, the correlation is measured and the linear equation is used to predict missing attributes-values of the new observation.

Let be a cluster C1:{Cs,RB,Na,Li} and the new observation {Fr}. Table 2 shows the set of numerical attributes describing the members of C1 and the new observation [5].

Table 2. Description of the Numerical attributes-values of Cluster C1:{Li,Na, K,Rb,Cs}

Name	1 BL	2 ML	3 D	4 AW	7 CR	8 AR	9 AV	10 IE	11 SH	12 EL	13 HV	14 HF	15 EC	16 TC
Li_3	1330	180.5	0.53	6.939	1.23	1.55	13.1	124	0.79	1.0	32.48	0.72	0.108	0.17
Na_{11}	892	97.8	0.97	22.9898	1.54	1.90	23.7	119	0.295	0.9	24.12	0.62	0.218	0.32
K_{19}	760	63.7	0.86	39.102	2.03	2.35	45.3	100	0.177	0.8	18.9	0.55	0.143	0.23
Rb_{37}	688	38.9	1.53	85.47	2.16	2.48	55.9	96	0.080	0.8	18.1	0.55	0.080	NIL
Cs_{55}	690	28.7	1.90	132.905	2.35	2.67	70	90	0.052	0.7	16.3	0.50	0.053	NIL
Fr_{87}	677	27	NIL	223	NIL	NIL	NIL	NIL	NIL	0.7	NIL	NIL	NIL	NIL
AprxVal			2.74		2.5	3.3	72	67	0.02		6.7	0.5		

Firstly, IE deduces the sequence of mathematical functions as a composition of services. Secondly, the numerical calculation is launched. To simplify the obtained result, we only present here the necessary inference rules. The concept formation concerns only the relationship between Atomic Number (data X) vs Density (data Y), identified by A3 (i.e., proposition P3).

Input

$Q = (regressline, slopeStdErr)$
$BF = \{stsDta, stdDtaofX, stdDtaofY, meanOpr, modeOpr, varianceOpr, sqrt, grdsFreedom, rOpr, tOpr, YOpr, bOpr, aOpr, DresOpr\}$ marked as SOLVED
BR= $\{R_0, \ldots, R_{16}\}$, where P_0 represents Q, i.e., $\{R_0\}$
$[R_0] : regressline \wedge slopeStErr \rightarrow P_0$

$[R_6]stDtaofY \wedge meanOpr \rightarrow meanofY$ $[R_7]stDtaofX \wedge meanOpr \rightarrow meanofX$
$[R_8]sdtDtaofX \wedge varianceofX \rightarrow stdDeviationofX$
$[R_9]stDtaofY \wedge varianceofY \rightarrow stdDeviationofY$
$[R_{10}]stdDeviationofX \wedge stdDeviationofY \wedge$
$grdsFreedom \wedge rOpr \rightarrow correltCoeffofXY$
$[R_{11}]stdDtaofX \wedge stdDtaofY \wedge correltCoeffofXY \wedge tOpr \rightarrow stdentTtest$
$[R_{12}]stdDeviationofX \wedge stdDeviationofY \wedge$
$correltCoeffofXY \wedge bOpr \rightarrow regrCoeff$
$[R_{13}]meanofY \wedge meanofX \wedge regrCoeff \wedge aOpr \rightarrow y - axsIntercept$
$[R_{14}]stdDtaofY \wedge stdDtaofX \wedge YOpr \wedge yaxsIntercpt \wedge regrCoeff \rightarrow regressline$
$[R_{15}]stdDtaofX \wedge stdDtaofY \wedge stdDeviatonofY \wedge$
$correltCoeffofXY \wedge DresOpr \rightarrow resStdDeviation$
$[R_{16}]stdDtaofX \wedge stdDtaofY \wedge meanofX \wedge$
$resStdDeviation \wedge SEbOpr \rightarrow slopeStdErr$
$[R_{17}]stDtaofX \wedge meanofX \wedge varianceOpr \rightarrow varianceofX$
$[R_{18}]stDtaofY \wedge meanofY \wedge varianceOpr \rightarrow varianceofY$

Numerical Output

Return from : $Apply(R7, meanofX : stdDtaofX, meanOpr)\{Val(meanofX) => 25\}$;
Return from : $Apply(R17, varianceofX : stDtaofX, meanofX, varianceOpr)$
$\{Val(varianceofX) => 440\}$;
Return from : $Apply(R8, stdDeviationofX : sdtDtaofX, varianceofX)$
$\{Val(stdDeviationofX) => 20.9762\}$;
Return from : $Apply(R6, meanofY : stDtaofY, meanOpr)\{Val(meanofY) = 1.587\}$;
Return from: $Apply(R18, varianceofY : stDtaofY, meanofY, varianceOpr)$
$\{Val(varianceofY) => 0.30187\}$;
Return from : $Apply(R9, stdDeviationofY : stDtaofY, varianceofY)$
$\{Val(stdDeviationofY) => 0.549426974\}$;
Return from: $Apply(R10, correltCoeffofXY : stdDeviationofX, stdDeviationofY,$
$grdsFreedom, rOpr)\{Val(correlCoeffofXY) => 0.97528152\}$;
Return from: $Apply(R12, regrCoeff : stdDeviationofX, stdDeviationofY,$
$correlCoeffofXY, bOpr)\{Val(regrCoeff) => 0.0255455\}$;
Return from: $Apply(R13, y - axsIntercept : meanofY, meanofX, regrCoeff, aOpr)$
$\{Val(y - axsIntercept) => 0.51936363\}$;
Return from: $Apply(R14, regressLine : stDtaofY, stDtaofX, YOpr,$
$y - axsIntercept, regrCoeff)$
$\{Val(P17, regressLine) => Y = 0.519368 + 0.0255455X\}$;
Return from: $Apply(R15, resStdDeviation : stDtaofX, stDtaofY, stdDeviationofY,$
$correltCoeffofXY, DresOpr)\{Val(resStdDeviation) => 0.140186\}$;
Return from: $Apply(R16, slopStdErr : stDtaofX, stDtaofY,$
$meanofXresStdDeviation, SEbOpr)\{Val(lopeStdErr) => 0.00334155\}$;
Return from: $Apply(R0)\{Val(P0, slopeStdErr) => 0.00334155\}$;

Thanks to R_{14}, it was possible to approximate the density-value (A3) of the new observation (Fr), i.e 2.7. By creating the specific inference rule, it is possible to apply the same process automatically to all numerical attributes and to obtain the missing values when a correlation exists among the members of C1 and the new observation Fr (see last row of Table 2).

5 Conclusion

The principal contribution of our approach is performing inferences to solve problems in a numeric domain like clustering, as an example we used statistical rules. The implemented Backward Inference Engine algorithm is correct and linear. Besides, thanks to its top-down strategy, it only computes a small search space. This strategy assures real cases applications.

Our approach takes advantages of the well-know positives properties of KBSs. The rules can be created, modified, adapted, and re-organized, independent of the inference engine which remains the same. Each time a new rule is approved by users, it is added to BR. Currently we are increasing the statistical functionalities of our framework to be applied for different scientific domains. In addition we study other cases for re-using mathematical formulae.

References

1. Dowling, W.F., Gallier, J.H.: Linear-Time Algorithms for Testing the Satisfiability of Propositional Horn Formulae. J. Logic Programming 3, 267–284 (1984)
2. Ghallab, M., Escalada-Imaz, G.: A linear control algorithm for a class of rule-based systems. The Journal of Logic Programming (11), 117–132 (1991)
3. Gallo, G., Urbani, G.: Algorithms for Testing the Satisfiability of Propositional Formulae. J. Logic Programming 7(1), 45–61 (1989)
4. Itai, A., Makowsky, J.A.: Unification as a Complexity Measure for Logic Programming. J. Logic Programming 4, 105–177 (1987)
5. Scerri, E.R.: The periodic table: its story and its significance. Editor Oxford University Press (2007)