US 20200242150A1

(19) **United States**

(12) **Patent Application Publication** (10) **Pub. No.: US 2020/0242150 A1**

Abdelwahab (43) **Pub. Date: Jul. 30, 2020**

(54) **METHOD FOR CREATING AN EFFICIENT, LOGICALLY COMPLETE, ONTOLOGICAL LEVEL IN THE EXTENDED RELATIONAL DATABASE CONCEPT**

(71) Applicant: **Elnaserledinellah Mahmoud Elsayed Abdelwahab**, Cairo (EG)

(72) Inventor: **Elnaserledinellah Mahmoud Elsayed Abdelwahab**, Cairo (EG)

(21) Appl. No.: **16/596,147**

(22) Filed: **Oct. 8, 2019**

(30) **Foreign Application Priority Data**

Oct. 9, 2018 (DE) 10 2018 008 923.2

Publication Classification

(51) **Int. Cl.**

G06F 16/36	(2006.01)
G06F 16/28	(2006.01)
G10L 15/26	(2006.01)
G06N 5/00	(2006.01)

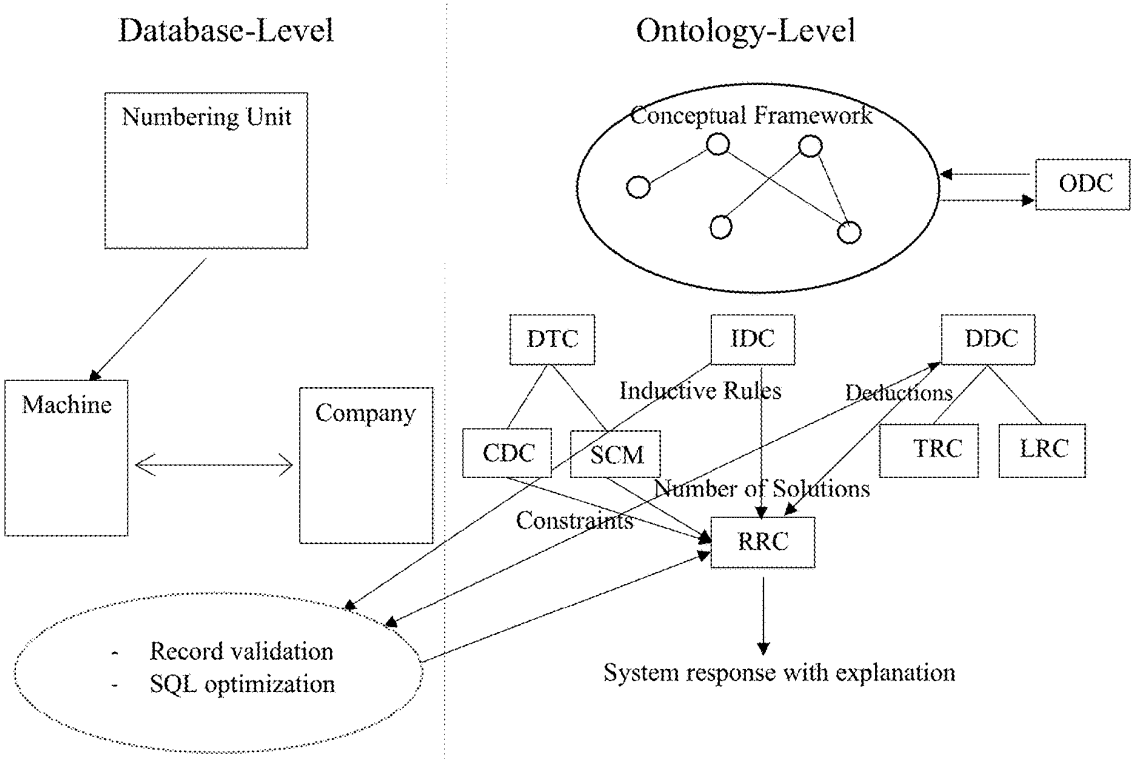
(52) **U.S. Cl.**

CPC ***G06F 16/367*** (2019.01); ***G06N 5/003*** (2013.01); ***G10L 15/265*** (2013.01); ***G06F 16/284*** (2019.01)

(57) **ABSTRACT**

In relational database concepts, query procedures suffer either from logically incomplete query results or lack of query time efficiency. The present invention provides efficient methods that optimize relational database systems in their query procedures so that the response procedure experiences an increase in efficiency in terms of speed and memory requirements without sacrificing logical conditions. In order to increase the efficiency of the query methods, a logically complete and at the same time efficient ontological level is introduced, which makes it possible to derive and/or evaluate application-specific constraints both, deductively and inductively.

It is the object of this invention to provide methods by which one can create a logically complete, efficient, ontological conceptual system in the catalog level of a relational database system that allows deduction as well as complete induction in its most general form insofar as that logical and natural language explanations of all system responses can be achieved. The inventive solution to this problem is specified in the claims 1-13.



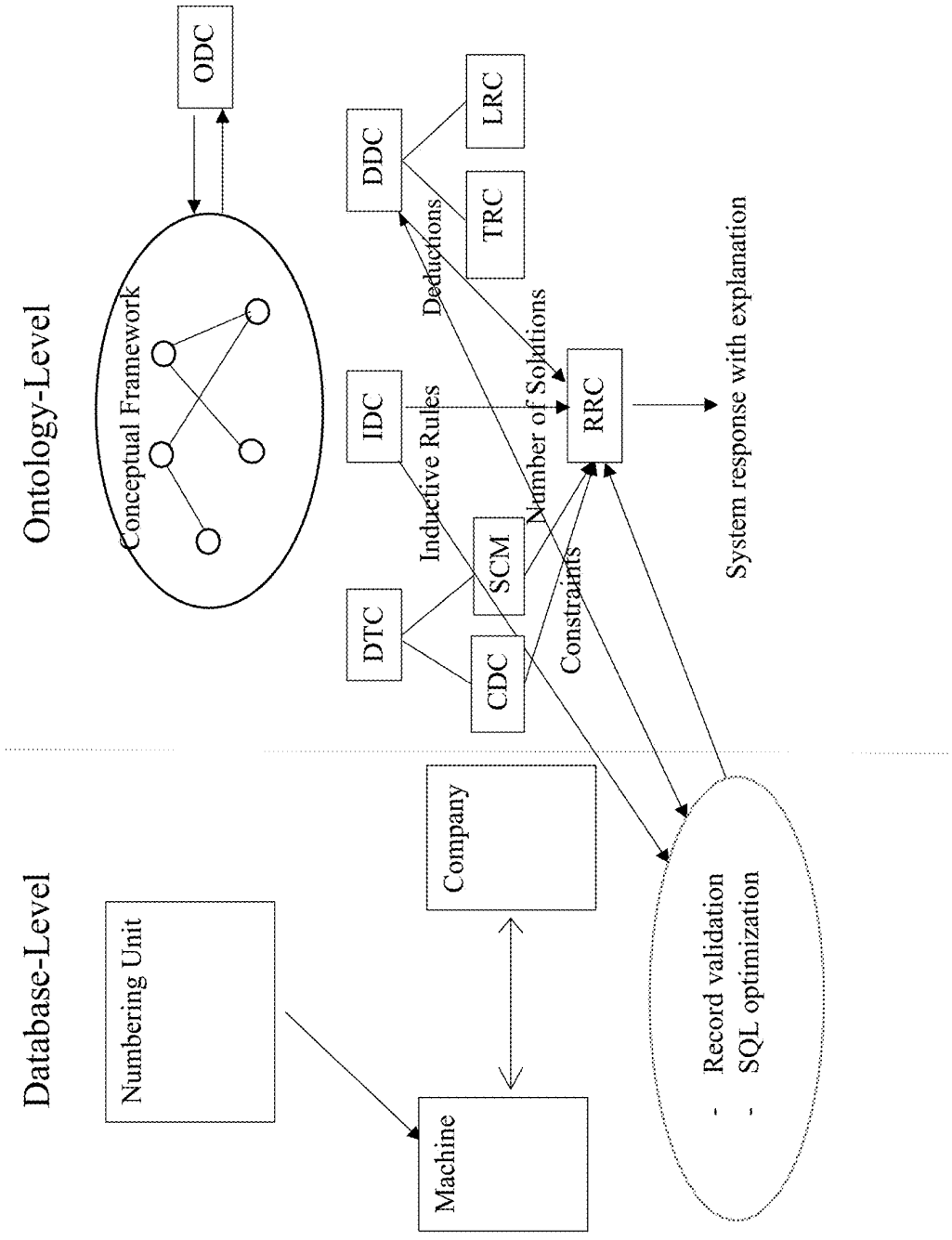

Ontology-Level
Database-Level
ODC
Conceptual Framework
DDC
LRC
TRC
Deductions
IDC
Inductive Rules
DTC
SCM
CDC
Number of Solutions
Constraints
RRC
System response with explanation
Numbering Unit
Company
Machine
- Record validation
- SQL optimization

FIG. 1

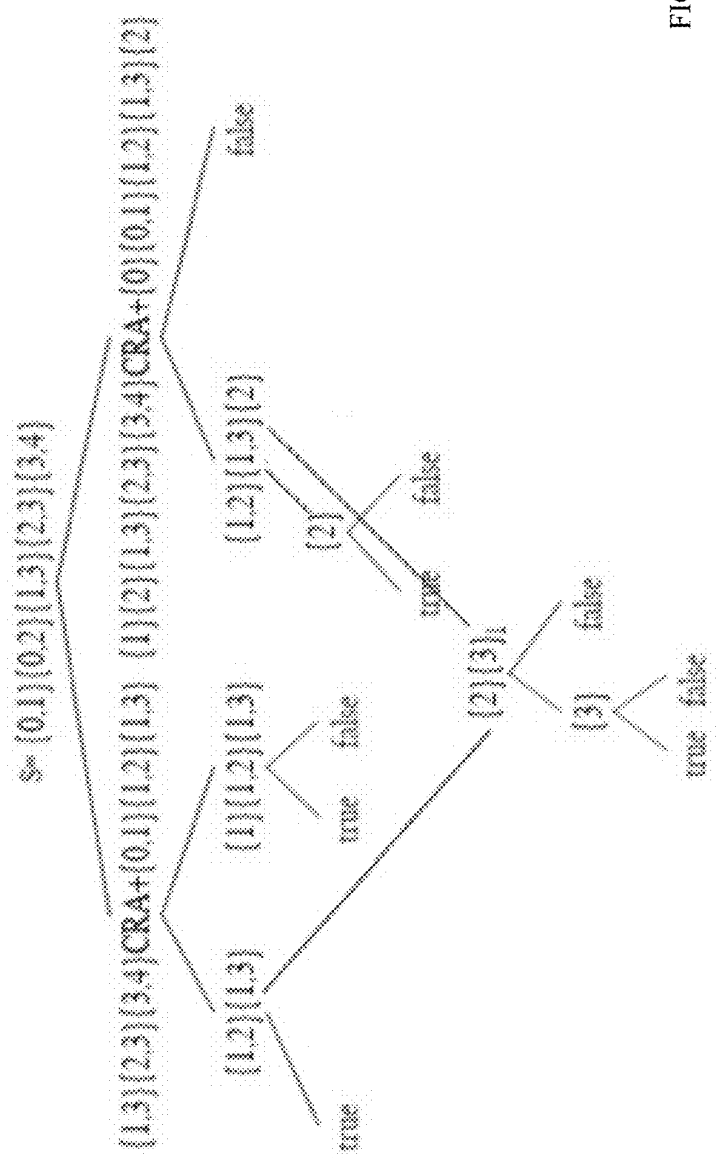

s= {0,1}{0,2}{1,3}{2,3}{3,4}
{1,3}{2,3}{3,4}CRA+{0,1}{1,2}{1,3}
{1,2}{1,3}
true
{1,1}{1,2}{1,3}
true false
{2}{3}
{3}
true false
{1,2}{1,3}{2}
{2}
true false
{1,1}{1,2}{1,3}{2,3}{3,4}CRA+{0}{0,1}{1,2}{1,3}{2}
false

FIG. 2

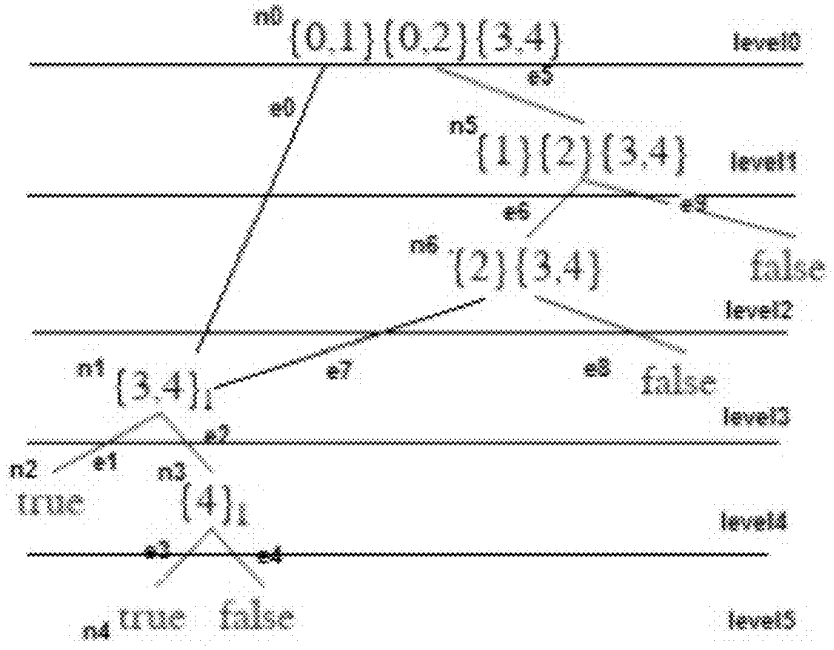

n0
{0,1}{0,2}{3,4}
level0
e5
e0
n5
{1}{2}{3,4}
level1
e6
e9
n6
{2}{3,4}
false
level2
n1
{3,4}1
e7
e8
false
level3
e2
n2
e1
n3
true
{4}1
level4
e3
e4
n4
true
false
level5

FIG. 3

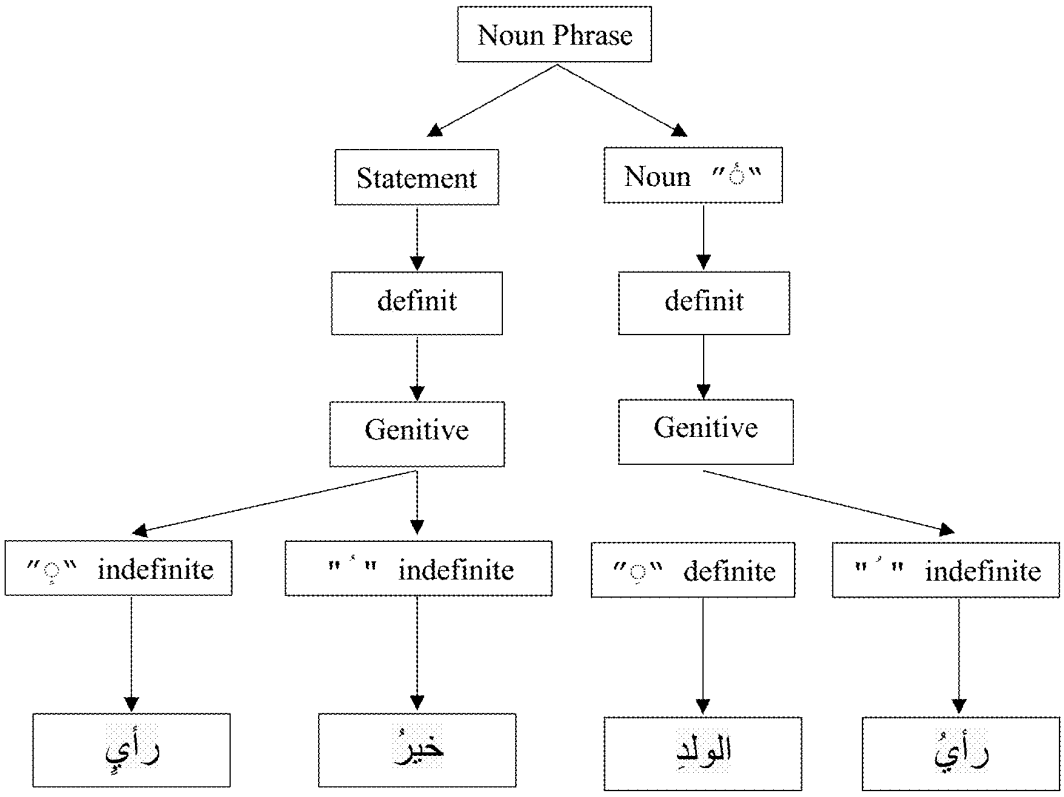

Noun Phrase
Statement
Noun ″ً″
definit
definit
Genitive
Genitive
″ٍ″ indefinite
″ٰ″ indefinite
″ٍ″ definite
″ٌ″ indefinite
رأيٍ
خيرُ
الولدِ
رأيُ

FIG. 4A

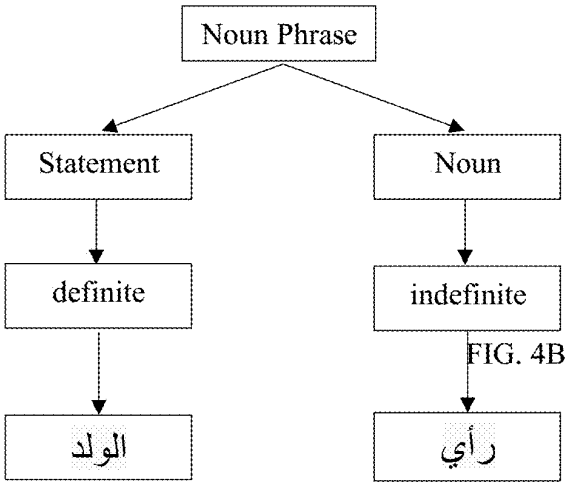


FIG. 4B

METHOD FOR CREATING AN EFFICIENT, LOGICALLY COMPLETE, ONTOLOGICAL LEVEL IN THE EXTENDED RELATIONAL DATABASE CONCEPT

TECHNICAL FIELD

[0001] The invention relates to a method for creating an ontological term system in the meta level of the relational database model, which is at the same time efficient and logically complete. In furtherance of the writing the entire system is called: Rational system (RS). The components of the ontological level of this system (listed in FIG. **1**) are as follows:

[0002] 1) Ontology Description Component (ODC), consisting of a graph- or logic-based editor of axioms (ontology structure) and facts.

[0003] 2) Decision Tree Component (DTC), which has the goal to provide selected constraints in CNF form as explicit Binary Decision Diagrams using SAT-solver methods. This component contains a CNF-based Constraints Definition Component (CDC) and a Solution Counting Procedure (SCP).

[0004] 3) Inductive Derivative Component (IDC), whose task is to generate combinatorics for selected parts of the overall system, for which no explicit constraints are known. In consequence, a complete induction procedure assists in inferring such constraints.

[0005] 4) Deductive Derivative Component (DDC) that applies syllogisms to selected parts of the overall system using a Language Recognition Component (LRC). A Translation Component (TRC) ensures that records from the database are rewritten into categorical statements.

[0006] 5) Rational Response Component (RRC), which can explain each response to a request made to the overall system by means of stored constraints (of DTC or IDC).

[0007] In contrast to known methods in deductive and relational databases, the RS not only allows linear processing times for entries and the improvement of database query procedures without endangering the logical consistency (cf. DE102015013593A1), but also allows a balanced application of known, efficient, logical procedures for the overall system: While DTC uses efficient methods of a priori completion for selected, difficult Boolean functions, DDC achieves fast response times regardless of the usual SQL machinery. According to the invention, this is achieved by simple, natural language-supported, syllogistic-based methods.

[0008] In addition, RRC offers the possibility of an intelligent system reaction, which justifies every answer using rules of logic (hence the property: rational). The constraints obtained by IDC do not require explicit verification and/or test procedures, since they originate from a mathematically stringent, complete induction. Using such constraints, the IDC also allows a compact representation of the combinatorics of selected parts of the overall system. The RS thus fulfills all the necessary criteria of a practically implementable logical system in its most general form, which can be used for terminological and logical control in most database applications.

PRIOR ART

[0009] Methods are known for generating ontology models from requirement documents and software as well as carrying out consistency checks between requirement documents and software code using ontology models. Terms are identified from the large number of requirement documents that are stored in a database. A processor assigns a term a word-tag. The word-tag indicates a grammatical use of each term in the requirement documents. The processor classifies each term based on word-tags. To form an ontology, the classification identifies whether each term is a part, symptom, action, event, or failure mode. The processor constructs an ontology-based consistency machine. A consistency check is carried out by applying the ontology-based consistency engine between ontologies extracted from two context documents. Inconsistent terms between the context documents are identified. At least one of the context documents with inconsistent terms is corrected (cf. DE102015121509A1). The disadvantage here is that the consistency and completeness of the ontology constructed in this way is disregarded.

[0010] Methods are known in which ambiguity is handled, which occurs when natural language information is combined with the knowledge represented by ontologies. Ambiguity is due to the fact that the same natural language identifier can denote several elements of the ontology. These procedures present a basic methodology of how the appropriate ontology element can be determined despite ambiguity. The approach is based on the human approach and the use of the context for monosemination. This represents the relationship between the entities mentioned in the text. These methods simulate the context of the text envelope based on the relationships within the ontology graph and disambiguate it by analyzing it. It is disadvantageous that the logical derivation (whether deduction or induction) is not affected (cf. Kleb, J.; Ontologie-basierte Monosemierung—Bestimmung von Referenzen im Semantic Web; KIT Scientific Publishing; 2015; DOI 10.5445/KSP/1000031500).

[0011] Methods are known for supporting the search for proven and existing solutions in the context of the development of technical products. Access to these solutions is made possible by considering the functions of technical systems from a usage perspective. The searcher is provided with a suitable solution space with relevant solutions using semantic networks. This can limit and expand the room according to various criteria. The disadvantage here is that in the case of more complex search tasks, the logical machinery is disregarded (cf. Gaag, A.; Entwicklung einer Ontologie zur funktionsorientierten Lösungssuche in der Produktentwicklung; Verlag Dr. Hut; 2010, ISBN 978-3-86853-731-4).

[0012] Methods are known for integrating semantic data processing in a device, in particular in a field device of automation technology. A generic description language scheme is used to define a semantic depot as a starting point. This description language scheme is enriched with contents of an ontology for the semantic representation of functioning of the device. Classes and/or subclasses of the ontology are taken from the ontology together with at least one property assigned to the classes and/or subclasses, converted into a corresponding schema declaration and finally this schema declaration is inserted into the description language schema. Then, one or more grammars are generated from the description language scheme, preferably grammars according to the

standardized data format Efficient XML Interchange, which are integrated in the device. A particular advantage of the method lies in the significantly more compact semantic data processing and data transmission. Hereby it is disadvantageous to disregard the logical features of the semantic data processing achieved and the associated ontology role (cf. WO2016110356A1).

[0013] Methods are known which give a user the opportunity to create databases and similar applications from imported ontologies. These databases can be configured specifically and come with error detection rules. The search in these databases is based on "meanings" instead of specific words. Ontology management guarantees consistent data integration, maintenance and flexibility, and also enables easy communication between multiple databases. Only the relevant ontology parts are considered for a specific application (sub-model). To ensure efficiency, the sub-model is then translated into an object-oriented, API-supporting Java application. The disadvantage is that no consistency or completeness criteria of the imported ontologies can be adopted and/or enforced (cf. U.S. Pat. No. 6,640,231).

[0014] Methods are known in which an ontology-related query is used to generate synonyms of words in database applications that could find relevant data records in addition to those used in the database queries. The disadvantage is that this search becomes even more complex with logically complex database queries (cf. U.S. Pat. No. 8,135,730).

[0015] Methods are known in which pairs of similar terms that exist in an OWL document are stored in a relational database and then used in database queries that establish semantic relationships between the two terms. It is disadvantageous that these methods cannot influence logically complex database queries (cf. U.S. Pat. No. 7,328,209).

[0016] General methods for handling constraints programming in the context of continuous or discrete variables for modeling mathematical or algorithmic problems are known (cf. DE4411514A1 and U.S. Pat. No. 5,216,593). The disadvantage is that these methods are not suitable for general database concepts.

[0017] Methods are known in which the most important referential integrity constraints are set in advance when creating an SQL execution plan (cf. U.S. Pat. No. 5,386,557). The disadvantage is that user-specific constraints are no longer possible. To ensure this, U.S. Pat. No. 5,488,722 discloses that a custom database restriction method depends on the likelihood of a consistency break. After that, the constraints that are probably not met are applied first. It is disadvantageous that this method is not generally applicable, since the establishment of a suitable hierarchy for the application priority of a restriction application requires the result of a database query, so that there are always non-ranked constraints when evaluating database queries.

[0018] In order to increase the efficiency of the query procedures, methods are known in which various predicates within a logical program are assigned a certain order rank (cf. EP0545090A2). Here, the assignment of the logical predicates affects the level of the term system, but not the logical deduction procedure (SLD resolution).

[0019] Methods are known for checking conditions for large amounts of data in a database (cf. EP0726537A1; Hirao, T.: Extension of the semantic processing model for relational databases, in: IBM Systems Journal, Volume 29, No. 4, 1990; p. 539 to 550 and Lippert, K.: Heterogene Kooperation, in: ix Multiuser Multitasking Magazin 7/1994, p. 140-147). The procedures specified therein are procedural and therefore have no logical-declarative form.

[0020] Methods according to DE19725965C2 are known which deal with general constraints of the extended relational database concept at the deductive catalog level. It is disadvantageous that the expansion of the logical theory can be very large, that is, exponential in the length of the logical formulas used.

[0021] Methods according to DE102015013593A1 are known in which the general restriction handling in extended relational database concepts is carried out in such a way that logical completeness methods at the catalog level can be used efficiently (i.e., not exponentially) in order to enable a maximum execution speed of logical queries. It is disadvantageous that term-related query processing is not dealt with explicitly. The possibility of using complete induction processes for the overall system is ignored. In addition, according to DE102015013593A1 no methods are specified with which the solutions of a set of CNF formulas can be counted efficiently. In addition, no syllogistic-based deduction applies. The answers of the system described in DE102015013593A1 lack a logical, natural explanatory component.

[0022] No methods are known with which a logically complete, efficient, ontological term system can be created at the catalog level of a relational database system, which enables the deduction and the complete induction in its most general form to the extent that logical (rational) explanations in a natural language of all system reactions can be achieved.

[0023] The Logical Completion

[0024] The rule sets occurring in a logical system must meet correctness and completeness criteria. In the context of this invention, it is said to be "complete" if axioms and rules of deduction explicitly derive everything that can be deduced.

[0025] In (Bancilhon, F.; Maier, D.; Sagiv, Y.; Ullman, J D: Magic sets and other strange ways to implement logic programs, Proc. ACM SIGMOD-SIGACT Symp. Of principles of database systems, Cambridge (Mass.), 1986), (Bayer, R.: Query Evaluation and Recursion in Deductive Database Systems, Manuscript, March 1985) or (Lozinskii, E L: Evaluation queries in deductive databases by generating, Proc. Int. Joint Conference on AI, 1: 173-177, 1985) there are alternative methods of completion.

[0026] These either concern the inference process itself, i.e., the way in which the rules are applied or the facts of the database relevant to the request. One method that deals with the inference process itself is the so-called semi-naive completion (cf. Bancilhon, F.; Ramakrishnan, R.: An Amateur's Introduction to Recursive Query Processing Strategies, Proc. Of the ACM SIGMOD-SIGACT Conference, Washington D.C., May 1986).

[0027] This tries to avoid the unnecessary repetition of generation steps by only using the incrementally generated facts, i.e. the facts that emerged in the last iteration are taken into account (cf. Chang, C L; Gallaire, H.; Minker, J.; Nicholas, M.: On the evaluation of queries containing derived relations in relational databases, Advances in database theory, Vol. I, 1981, or Marq-Puchen; Gallausiaux, M.; Jomien: Interfacing Prolog and Relational Database Management Systems, New applications of databases, Gardavin and Gelaube eds. Academic Press, London, 1984).

[0028] The assumption ΔR_i is that $\Delta R_i=(R_i \cup F(R_{i-1} \cup \Delta R_{i-1}))-R_i$ for every relation R_i (here the incremental change of

R_i and $F(R_i)$ is the functional expression that is deduced from the body of a rule). In general, one cannot simply calculate ΔR_i as a function of ΔR_{i-1}. In the case of linear recursive rules, however, this is possible because $F(R_{i-1} \cup \Delta R_{i-1}) = F(R_{i-1}) \cup F(\Delta R_{i-1}) = R_i \cup F(\Delta R_{i-1})$.

[0029] As long as one can assume that the rules are linear-recursive, semi-naive completion is an efficient method. However, if the flexibility is expanded and non-linear recursions are allowed, this method is no longer efficient (the early realizations of semi-naive approaches are not valid for this type of recursion). Furthermore, the exponential effort is only reduced by reducing the number of facts of the one pattern relevant for the last deduction step. In a link $A_1 \hat{} A_2 \hat{} \ldots \hat{} S \hat{} \ldots A_n$ where S corresponds to this pattern, the facts that unify S are taken into account, but links between the A_i must always be established again. It has been found that creating the AND operations is the most complex step in the completion procedure. The so-called APEX procedure (cf. Lozinskii, E L: Evaluation queries in deductive databases by generating, Proc. Int. Joint Conference on AI, 1: 173-177, 1985) is a procedure of a different kind. First, those for a definite query clause W (Target) relevant facts of a database are generated, only then to start the completion process. The relevant facts are calculated using so-called control system graphs. These contain all the logical links between rules of the database.

[0030] They are started with coupling a query generation process which, in the case of important AND operations, generates further queries W1?, W2? . . . . etc. The generation takes place through sideway information passing (SIP) between the query or queries and the facts of the respective link(s). Another method of this class is QSQ (cf. Vieille, L.; Recursive axioms in deductive databases: The Query-Subquery approach, Proc. First Int. Conf. On expert database systems, Kerschlag ed., Charlston, 1986). As in APEX, database rules are used for the generation of new queries. However, as in PROLOG, the relevant facts are searched for linearly and in depth using a backward chaining procedure. In the case of recursive predicates, the queries are generated using SIP using the facts already found.

[0031] The main difference between APEX and QSQ on the one hand and semi-naive completion on the other hand is that the solution of semi-naive completion deals with the general and principal problem of inferring basic facts, whereas the other two methods only try to optimize the usual inference mechanisms by taking the relevant facts into account.

[0032] Magic Sets (cf. Beeri, C.; Ramakrishnan; On the power of magic, Proc. Sixth ACM SIGMOD-SIGACT Symp. On principles of database systems, San Diego, Calif., March 1987) is a modification of QSQ, which the adds variable bindings in the form of new "magic" rules to a program or links them to the right side of a clause as constraints. Starting with the target clause, a lot of new predicates are generated. SIP succeeds in forwarding these adornments. The result is a new, modified version of the first program, which in some cases is more efficient. For example, the program:

[0033] anc(X,Y)←par(X,Y).

[0034] anc(X,Y)←anc(X,Z)^par(Z,Y).

and the query q(X)←anc(a,X).

the new "magic" program:

[0035] magic(a).

[0036] q(X)←anc(a,X).

[0037] anc(X,Y)←par(X, Y).

[0038] anc(X,Y)←magic(X)^anc(X, Z)^par(Z, Y).

[0039] magic(Z)←magic(X)^anc(X,Z).

[0040] The new magic predicate represents a restriction of the permissible substitutions. It systematically connects the program constants with each other.

[0041] Practical Considerations the Nature of a Logical Variable

[0042] The basic problem with constraint handling is to reduce the effort involved in applying a set of constraints. Solution approaches amount to generating instances of these rules before an adequate application of the constraints is activated. The fact that many solution approaches achieve a high degree of efficiency through variable instantiation calls for a fundamental discussion about the importance of a variable in the closed world of a deductive database and an RD model. The usual meaning of a variable in mathematical logic (and thus in logical programming) boils down to considering it as an entity detached from the domain of the application. The link between a variable instantiation and the domain is therefore unclear, because there are no explicit or implicit rules in the interpretation to describe these instantiation procedures. This access is therefore left to the implementation of a logical machine, which can lead to considerable problems.

[0043] DE19725965C2 solves this problem by introducing the Herbal Abstraction Structure. Here, variables are viewed as abstractions of terms and term relationships at the catalog level. This approach makes it possible to describe alternative completion methods that make it possible to move from a standard Herbrand interpretation to a "more complete" one using any degree of abstraction. If one reverses the "abstraction process", i.e. if one starts with un-instantiated clauses, the Herbal Abstraction Structure enables procedures that can split clauses of a logical program into a number of "more instanced" clauses. This in turn leads to the increase in efficiency described there (linearization). However, the method formalized in

[0044] DE19725965C2 (Alg. 2) does not specify a method for how the instantiation of the rules could be optimized. This could be achieved in a variety of ways in a Herbal Abstraction Structure. In addition, the main weakness of using the Herbal Abstraction Structure is that in the worst case it represents an exponential search space.

[0045] The method presented in DE102015013593A1, however, leads to complete evaluation methods depending on a new representation of variables as parts of the classic truth table, also called pattern strings or pattern trees. In contrast to the state-of-the-art resolution methods, these lead to small search spaces in which linear processing times of inputs can be realized. Here, the term "inputs" always means instantiations of logical formulas. A procedure is used to generate the extension that resolves model trees instead of clauses.

[0046] In this context, two types of resolution procedures for formulas/clauses (also known as Solvers) are known: complete and incomplete. A Solver is called complete if it can both determine that a formula can be fulfilled and that it cannot be fulfilled. Not all formulas that can occur in a Solver formula set fall into the same category. In practice, a distinction is made between three categories:

[0047] Random: formulas that are generated randomly according to a scheme called "fixed clause length

model" (one only specifies the number of variables and clauses and how long a clause should be, the rest is generated randomly)

[0048] Crafted: formulas derived from difficult combinatorial problems such as graph coloring

[0049] Application: formulas that are derived from applications in reality (e.g., circuit verification)

[0050] Not all known solver paradigms cope equally well with all formula categories. A distinction is made between four types of solvers, which are dealt with in DE102015013593A1. All four solver methods can be characterized by the following features and are therefore significantly different from DE102015013593A1 and from the method according to the invention presented below:

[0051] 1. They are an example of the application of Tarski's semantic concept of truth regarding formulas in mathematical logic. This term basically prescribes that variables exist separately from their meanings or values. These meanings are replaced in the formulas so that they are fulfilled. Thus, variables (and their associated literals) are only viewed as containers that do not allow structural information about the data stored in them to be derived or used.

[0052] 2. A by-product of this view is that algorithmic methods are forced to test different variable evaluations before they find a valid one. The term variable evaluation is therefore an integral part of this procedure.

[0053] 3. Information from the specific mathematical-logical formula, which relates to the concatenation of used variables (literals) and their mutual interactions, is not used or is used only inadequately (usually in the form of heuristics) in order to find the/a valid evaluation.

[0054] 4. All methods avoid the construction of the entire combinatorial space, since this construction is exponential with regard to the number of variables. Since the methods use variable evaluations iteratively, only a part of the space is constructed in each iteration, the formula is then evaluated and the next iteration is started, etc.

[0055] 5. Because the methods usually do not use general heuristics, their performance strongly depends on the type of formula (Tab. 1). "Good", "bad" and "neutral" are hereby rough indicators of the expected performance of a method in relation to a given type of formula. "SAT/UNSAT" stands for "satisfiable" or "unsatisfiable":

TABLE 1

Category	CDCL	Look-ahead	Message-passing	SLS
Random SAT	bad	neutral	good	good
Random UNSAT	bad	good	bad	bad
Crafted SAT	good	neutral	bad	neutral
Crafted UNSAT	neutral	neutral	bad	bad
Application SAT	good	bad	bad	bad
Application UNSAT	neutral	bad	bad	bad

[0056] Ultimately, a solver method is known that corresponds to the classic truth table method. It differs from the methods described in DE102015013593A1 as follows:

[0057] 1. Part of the method is the construction of the entire exponential space of all combinations of the variable values. After this space has been constructed, one can efficiently determine whether a particular evaluation of variables for the respective formula results in "true" or not.

[0058] 2. In contrast to all other methods, finding a solution does not include trying variable-values in the original formula, but the simple search in the generated space, i.e., in the truth table. This makes it possible to find the truth value of the instantiated formula without making use of the classic, logical operators (AND, OR, NOT), since the full extension of these operators, applied to the logical values "true" and "false", is material.

[0059] 3. The number of variable evaluations that one has to go through until a productive value is found is exponential in the worst case. This potential exponentiality is the main disadvantage.

[0060] 4. Because no assumptions are made about the formulas, the performance of the process is independent of the type of formula.

OBJECTIVE OF THE INVENTION

[0061] The objective of this invention, while maintaining the strictest logical conditions, is to optimize relational database systems in their most general concept, by means of a logically complete, ontological meta-level, which allows deductive as well as inductive reasoning in their logical query methods so that the response procedure experiences linear efficiency in terms of speed and storage requirements.

NATURE OF THE INVENTION

[0062] The invention is based on the objective of creating a method of the type mentioned at the outset which optimizes relational database systems in their query methods in such a way, that the response procedure experiences an increase in efficiency with regard to the speed and the memory requirement without having to give up logical conditions. This is achieved by introducing a logically complete and at the same time efficient ontological level, which allows application-specific constraints to be derived and/or evaluated deductively and inductively. This objective is achieved with process steps as specified in claims **1-13**.

Example of Accomplishment

Extension of the RDS Through the Concept of a Logically Complete, Efficient, Ontological Term System in the Catalog

[0063] The central method of this invention is based on the idea of either explicitly defining all the data and rules necessary for the terminological, logical and application-specific control or making them available in advance in the catalog by means of complete induction. As a result, not only is the constraint treatment dealt with more efficiently, but also some calculation requirements that are not possible in common relational systems. Consistency properties of databases are only logical in nature and therefore meta problems. The basic assumption of this invention is that the ontological term system of a database application remains sufficiently constant. We call this condition below: closed ontology. This should not be confused with the logical "closed world assumption", which in the context of logical systems expresses the fact that facts that are not explicitly stored in

the database are considered "wrong". The following example explains this procedure:

[0064] Be given a used printing database. This contains the tables "Machine", "Company" and "Numbering Unit" as shown in (FIG. **1**). The following describes possible conditions that are not easy to handle in conventional relational systems and that claim parts of the entire system shown here:

[0065] 1. Among other things, the table "Machine" has the fields "Machine type" and "Printing group". These are of particular interest because their combination models important constraints known in the printing industry. E.g., a Polar machine cannot belong to the group of 5-color printing machines, because the type "Polar" represents cutting machines. Similarly, a Heidelberg-Tiegel never has more than 2 inking units, so that the combination (type="Tiegel", group="3 colors") corresponds to no logic. We call such factual constraints of the term system: Type-1-Constraints

[0066] 2. Machines cannot have more than 5 numbering units. This condition can only be implemented by programming (usually: stored procedures), since neither entity nor table diagrams are able to express conditions via cardinalities of the relationships, except by "many" and "one". We call general conditions that have to do with cardinalities of the relations in the relational data model: Type-2-Constraints

[0067] 3. In the printing industry, machines are occasionally transported to and from different physical locations. Companies that intend to keep transport costs to a minimum therefore always need, among other things, calculations of the best transport routes. We call these types of calculation-intensive tasks that can be represented using general Boolean functions that can be expressed in CNF: Type-3-Constraints

[0068] 4. Sometimes it is necessary to know the factory standard configuration of a machine type in order to carry out a comparison with the used machine of the same type (e.g., whether the standard configuration is with or without a numbering unit in the factory). This request is related to both manufacturer specifications in the term system and the current used machine database. We call such constraints: Type-4-Constraints

[0069] 5. All inquiries that are exclusively related to the database are usually handled with common SQL components. This includes, for example: "Which companies currently supply Tiegel machines?". We call this type of constraint: Type-5-Constraints

[0070] 6. Inquiries such as: "Which parts are supplied with a Tiegel machine?" And "Which parts are supplied by a particular company with a Tiegel machine?" Are Type-1 and Type-4 constraints examples that are used for their execution require the calculation of the transitive envelope of the respective relationships. Stored procedures are used in common relational database contexts because SQL does not allow loops by default. We name queries that cannot be solved using SQL alone: Type-6-Constraints

[0071] 7. Derivation of formulas of the form: "All machines from company X are always completely overhauled and delivered with care" or "There are only a few spare parts that are compatible with a numbering system" are only possible if the entire data set and/or facts of the term system is taken into account. We call these: Type-7-Constraints

[0072] The following table (Tab. 2) shows the order of constraint types for different system components.

[0073] The descriptions and definitions following in Table 2 explain the functionalities according to the invention of the various components of the overall system with reference to FIG. **1**.

TABLE 2

Constraint	System-component	Comment
Type-1, Type-2	CDC, DTC, RRC, record-validation	Constraints related to term systems and relations are given in CDC, defined in CNF form, and evaluated by means of DTC. They are used in RRC for the generation of intelligent answers. Furthermore, they affect the customary record validation component.
Type-3	CDC, DTC, SCP, RRC	General Boolean functions are defined in CDC in CNF form. Solver method 1 in DTC converts to BDD (Binary Decision Diagram), whose information are made available to RRC. SCP provides for the number of different solution alternatives
Type-4, Type-5, Type-6	CDC, DTC, OBC, TRC, DDC, LRC	Constraints that have to do with both the database and the term system or those that are only database-related can be handled in two ways: either one first defines them in CNF form in CDC, then they are evaluated using DTC, or: selected records are in TRC initially translated into categorical statements, then used for deduction by means of DDC. As DDC allows recursion, this also covers Type 6 constraints. LRC provides that they are represented in the natural language.
Type 7	IDC, DDC, LRC	Database and term system provide a finite number of field/fact combinations for which the creation of complete combinatorial tables is possible. Rules can then be derived by means of complete induction (Method 9) and correspond to the "all" or "existence"-quantified formulas in Type 7.

[0074] Definition 1: Given the set B of all terms in an application to which all-quantified, existential, negated and indefinite terms, i.e., variables, include: A logical conclusion is called syllogism if two premises (prerequisites), called upper- and lower sentences, lead to a conclusion. In a categorical syllogism (also called assertory syllogism), premises and conclusions are categorical judgments, i.e., statements in which a term from B, the subject, another term from B, the predicate, is assigned or denied in a certain way.

[0075] Definition 2: Let kSyl be the set of all known, valid categorical syllogisms and hSyl be the set of all known, valid hypothetical syllogisms of the form: From ($\boldsymbol{P} \vdash \boldsymbol{Q}$) and ($\boldsymbol{Q} \vdash \boldsymbol{R}$) one derives ($\boldsymbol{P} \vdash \boldsymbol{R}$), where P, Q, R are categorical sentences and '$\vdash$' are syntactic derivation relations, then categorical sentence s is called the logical consequence of the categorical sentence set S ($S \rightarrow_{Syl} s$) ifs results from Syl=kSyl∪hSyl using Syllogisms. We call the list of deduction steps that lead to a sentence s from a sentence set S using rules from Syl: Derivation of s from S ($S\Delta_s$). If s contains no variables, it is called a fact. If $S\Delta_s$ is empty, it is called an axiom.

[0076] Definition 3: An ontology Ont=(B,R) is a tuple in which B is a set of all terms of an application and R is a set of all intended n-ary relationships between these terms. Alternatively, instead of R one can use the set of all

categorical sentences S of the form: <Term$_i$>is<Property$_j$>of<Relation$_k$> This is possible because:

$\forall r \in R, \forall b_i \in R: r(b_1,b_2,...b_n)$ iff {
$s_1=b_1\_is\_property_1\_of\_r$,
$s_2=b_2\_is\_property_2\_of\_r$,
....
$S_i=b_i\_is\_property_j\_of\_r$.
}

[0077] The sentence s_i is called the descriptive sentence of the relationship r. Descriptive sentences can only be facts.

[0078] Definition 4: An ontology Ont=(B,S), S set of all description sets of the relationships between terms in B, can also be represented in the form of a directed graph with marked nodes: A directed graph or digraph with node markings G=(V,E,M) consists of:

[0079] A set V of nodes

[0080] A set of ordered pairs of nodes $E \subseteq V \times V$ of edges

[0081] A set M of marks on the edges

[0082] An ontology is called consistent if:

[0083] the equivalent directed graph G is acyclic

[0084] $\forall s_1,s_2 \in S$, s_1,s_2: Axiom: $s_1 \neq \neg s_2$

[0085] Definition 5: A consistent ontology Ont=(B,S) is called complete if $\forall s \in S$, s follows logically from the axioms: $S\Delta_x$. Ont is called closed if the set B has a fixed, constant cardinality. This simple concept of completeness is possible because the set Syl contains known logically complete subsets of derivation methods (cf. Moss, L S; Completeness Theorems for Syllogistic Fragments, in F. Hamm and S. Kepser (eds.) Logics for Linguistic Structures, Mouton de Gruyter, 2008, 143-173). The correctness of the set Syl is also known and is assumed here.

[0086] Definition 6: In the consistent ontology Ont=(B,S), constraints of the form:
$(s_1 \& s_2 \& s_3 \ldots s_n)>c$,
where $s_1,s_2, \ldots s_n,c \in S$, the characters &: "and" and ">": mean material implication, called categorical constraints. The premises: $s_1,s_2, \ldots s_n$ are called categorical conjunctions. We call terms that are used in categorical conjunctions: decision terms. The term c is called: Conclusion.

[0087] Definition 7: A grammar is a 4-Tuple $G=(V_N, V_T,P,S)$ where:

[0088] 1. V_N is a finite, non-empty set, the set of non-terminal characters,

[0089] 2. V_T is a finite, non-empty set, the set of terminal characters,

[0090] 3. P is a finite subset of $V^* \times V^*$, the set of productions or rules,

[0091] 4. $S \in V_N$ is the starting signal.

[0092] A Diacritics-Grammar (DiaG) is a grammar that allows terminal characters with diacritics from V_T. We call terminal and non-terminal signs and productions that allow diacritics: dia-terminal/non-terminal and DiaProduction.

[0093] Example of a grammar for sentences in the English language:

[0094] $G=(V_N,V_T,P,S)$

[0095] V_N={noun phrase, verb phrase, proper name, article, noun, verb}

[0096] V_T={Susanne,cat,horse,hay,book,the,hunts,eats, reads}

[0097] P={(sentence->nominal phrase verb phrase.),

[0098] (Noun phrase->proper name|article noun),

[0099] (Verb phrase->verb|verb noun phrase),

[0100] (Proper name->Susanne),

[0101] (Noun->cat|horse|hay|book),

[0102] (Article->the),

[0103] (Verb->hunts|eats|reads)}.

[0104] S=sentence

[0105] Example of a DiaG for the Arabic language:

[0106] G=(VN, VT, P, S)

[0107] V_N={noun phrase, verb phrase, proper name, noun, verb}

[0108] V_T={ولدٌ , ولدا , ولدِ , رأيٌ , رأيٌ , رأيَ , خيرَ خيرٌ , خيرِ...

[0109] [all words/word patterns in the Arabic dictionary]}

[0110] P={(sentence->nominal phrase verb phrase.),

[0111] (Nominal phrase->statement "ٌ noun),

[0112] (Verb phrase->verb|"ٌ noun verb),

[0113] (Noun->certain|indefinite),

[0114] (Determined->AlNounPattern|Genitive|proper name),

[0115] (Proper name->{ . . . all known names . . . })

[0116] (Genitive->"ٌ certain "ٌ indefinite|" ٌ indefinite "ٌ indefinite)

[0117] (AlNounPattern->NounPattern "ال")

[0118] (Noun pattern->{ . . . all noun patterns in the dictionary . . . })

[0119] (Verb->{ . . . all verb patterns in the lexicon . . . })

[0120] (Statement->Determined|Undefined|Verb phrase)}.

[0121] S=sentence

[0122] 1. CDC The Constraints Definition Component consists of a simple text editor, in which one defines CNF formula sets.

[0123] 2. ODC Like CDC, the Ontology Description Component can consist of a text editor in which categorical description sentences and/or constraints can be defined, or a graph editor in which terms and their relationships can be expressed in the form of nodes and edges.

[0124] 3. DTC The Decision Tree Component is the central unit in which one evaluates Boolean functions f expressed in CNF. The result is a decision tree (BDD) that is equivalent to the truth table off. Method 1 is the central method in this component and uses the pattern property of the logical variables described in DE102015013593A1 in the following way:

[0125] Method 1:

[0126] Input: CNF clause set S

[0127] Output: BDD

[0128] Steps:

[0129] 1. Use Method 2 to rename the formula set S to an equivalent S'. This renaming takes into account sample lengths in the form described in Method 2.

[0130] 2. Select literal X of the first clause CE S' for the instantiation. X is the literal with the least index.

[0131] 3. Apply the evaluations: {X=TRUE} then {X=False} to S'. This application results in left and right clause sets S_1, S_2.

[0132] 4. If either S_1 or S_2 becomes TRUE or False, output TRUE/False nodes for the respective clause set.

[0133] 5. If neither S_1 nor S_2 are found in a container LCS (list of already processed clause sets): Initiate a recursive call first with S_1, then with S_2. This results in

a left (leftRes) and a right (rightRes) result. If not: Initiate a recursive call with only the clause set that was not found in LCS. Return the found BDD for the other.

[0134] 6. The end result of the resolution of clause set S' is a node with left child leftRes and right child rightRes.

[0135] Method 2:

[0136] Input: CNF clause set S

[0137] Output: CNF clause set S' that meets the following conditions:

[0138] 1. $\forall l_1,l_2,l_3, \ldots ,l_n \in C$ of S': l_i appears before l_j in C, if i<j, i.e., indexes of the literals are sorted in ascending order within the S' clauses.

[0139] 2. S' is sorted according to in ascending order, taking negation into account.

[0140] 3. Formally: ∀i,j: If i<j, then $L_i \in C$ appears before $L_j \in D$ in S', where L_i is head literal (i.e., first literal) of C and L_j head literal of D.

[0141] 4. $\forall x \in LIT(S')$, LIT(S') is the set of all literals in S', ∀C∈S'::

[0142] If $x \notin$ LEFT (x, C) then $\forall y \in$ LEFT(x,C): x>y. LEFT (x,C) is a function that returns all variable indices that exist before the variable x from clause C in the string representation of the formula set S'. (In other words, this condition stipulates that all new indices that appear in a clause for the first time must be larger than all those already used in S').

[0143] 5. S' is a set, i.e., clauses only appear once in it.

[0144] Steps:

[0145] 1. current set=Method3(S)

[0146] 2. while [current set is not sorted as in condition b)]

[0147] a) sort CurrentSet according to condition b)

[0148] b) CurrentSet=Method3(CurrentSet)

[0149] 3. S'=CurrentSet

[0150] 4. Return S'

[0151] Method 3:

[0152] Input: CNF clause set S

[0153] Output: CNF clause set S'

[0154] Steps:

[0155] 1. Number clauses in S in increasing order (start with 0).

[0156] 2. Set up a table with rows of literals in S and columns with clauses.

[0157] 3. For each clause C_i:

[0158] a) Sort literals in C_i in increasing order so that those that have not yet been renamed and appear in a larger number of clauses appear first.

[0159] b) For all literals in C_i: Create a new row and write down TRUE or False values, depending on whether the literal appears in the column clause or not.

[0160] 4. Rename all literal indexes in increasing order in the table. Start with 0.

[0161] 5. Construct all clauses of S using the new names/indices. The resulting set of clauses is S'.

[0162] 6. Return S'.

[0163] Example: If S={{0.5} {0.2} {1.3} {1.4} {2,3}}, the table in point 2 looks like this:

	C_0	C_1	C_2	C_3	C_4
0	TRUE	TRUE	False	False	False
5	TRUE	False	False	False	False
2	False	TRUE	False	False	TRUE
1	False	False	TRUE	TRUE	False
3	False	False	TRUE	False	TRUE
4	False	False	False	TRUE	False

[0164] According to point 4:

	C_0	C_1	C_2	C_3	C_4
0	TRUE	TRUE	False	False	False
1	TRUE	False	False	False	False
2	False	TRUE	False	False	TRUE
3	False	False	TRUE	TRUE	False
4	False	False	TRUE	False	TRUE
5	False	False	False	TRUE	False

[0165] The new set of clauses: S'={{0.1} {0.2} {3.4} {3.5} {2.4}}. This set does not meet all of the conditions in Method 2 and requires a new ordering and renaming loop. In this new loop the clause set is: S''={{0.1} {0.2} {2.4} {3.4} {3.5}} for S'''={{0.1} {0.2} {2, 3} {3,4} {4,5}} reformed. FIG. **2** shows an example execution of Method 1 on the CNF clause set: S={{0,1} {0,2} {1,3} {2,3} {3,4}}. The above Method 1 sets up the BDD for S, but cannot convey any information about the number of possible solutions. The following method fills this gap.

[0166] Method 4:

[0167] Input: BDD for CNF clause set S

[0168] Output: number of solutions

[0169] Steps: 1. numberedBDD=number nodes and edges in the BDD starting with 0 (Method 5)

[0170] 2. Set numberSolutions for nodes n_0=0, numberSolutions for all edges of the first BDD level=1

[0171] 3. For all levels i in numberedBDD

[0172] a) For all edges e_i, j is the index of the edge in plane i:

[0173] i. Set numberSolutions for e_{ij}=numberSolutions of the parent node

[0174] b) For all nodes n_{ik}, k is the index of the node in level i:

[0175] i. If n_{ik} TRUE (flower) is:

[0176] numberSolutions from $n_{ik}=(\Sigma e_x * 2^{i-Le}) * 2^{N-i}$,

[0177] x is the index of an edge that leads to n_{ik}, e_x numberSolutions of such an edge, L_e edge plane of x (given with: $L_e=L_{sr}+1$,

[0178] S_r parent node of e), N number of variables in S

[0179] ii. else: numberSolutions from $n_{ik}=\Sigma e_x * 2^{-Le}$

[0180] 4. Return numberSolutions=ΣTnd, Tnd is TRUE node (flower)

[0181] Method 5:

[0182] Input: BDD for CNF clause set S

[0183] Output: BDD with numbered nodes, edges and levels

[0184] Steps:

[0185] 1. Run the BDD in a recursive, depth-first manner. Number nodes and edges and create a linear, topological order. A topological order is basically an inequality that can be created linearly in the following

way: For every two nodes n_1, n_2, children of n: set the inequality $n<n_1<n_2$ and add it, recursively in a depth-first Way until the final inequality. The inequality is supplemented by recursively placing children of node n, before children of node

[0186] 2. For all u∈V (V node set of the BDD):

[0187] dist(u)=∞

[0188] dist(s)=0, s is root

[0189] 3. For all u∈V, in the linearized order:

[0190] dist(u)=Dist(u)

[0191] L_u=|dist(u)|

[0192] Dist:

[0193] Input: u∈V, BDD=(V,E), V node set, E edge set

[0194] Output: Integer representing the distance between u and the root

[0195] Note: l(u,v_i,) is the length of the edge from u to v_i (always: '−1').

[0196] Steps:

[0197] for all edges v_1,v_2, . . . v_n∈V such that: (u,v_i)∈E:

Dist (u) =
min {
[Dist (v_1) + l(u, v_1)], ...
[Dist (v_n) + l(u, v_n)]
}

[0198] FIG. **3** shows an execution of Method 5 on the BDD created for the clause set S={{0,1} {0,2} {0,4}} by means of Method 1. The following example sequence of operations shows the application of Method 4 to S:

[0199] a) Level-0: n_0=0

[0200] b) Level-1: e_0=1, e_s=1, n_s=e_s*2^{l-Le5}=1*2^{1-1}=1

[0201] c) Level-2: e_6=n_5=1, e_9=n_5=1, n_6=e_6*2^{l-Le6}=1*2^{2-2}=1

[0202] d) Level-3: e_7=n_6=1, e_8=n_6=1, n_1=e_0*2^{l-Le7}=1*2^{3-1}+1*2^{3-3}=5

[0203] e) Level-4: e_1=n_1=5, e_2=n_1=5, n_2=(e_1*2^{l-Le1}) *2^{N-l}=(5*2^{4-4})*2^{5-4}=10, n_3=e_2*2^{4-4}=5

[0204] f) Level-5: e_3=n_3=5, e_4=n_3=5, , n_4=(e_3*2^{l-Le3}) *2^{N-l}=(5*2^{5-5})*2^{5-5}=5

[0205] NumberSolutions=n_4+n_2=15

[0206] 4. DDC The central method in this component applies syllogisms of the set Syl until no new sentences can be derived.

[0207] Method 6:

[0208] Input: Categorical sentence set S

[0209] Output: Categorical sentence set S'Steps:

[0210] 1. NewSentence=TRUE, S'=S

[0211] 2. While (NewSentence=TRUE)

[0212] For all syllogisms sy of the set Syl:

[0213] a) Apply sy to S.

[0214] b) If a new sentence s has arisen:

[0215] Set NewSentence=TRUE, S'=S'∪s

[0216] else NewSentence=False

[0217] 3. Return S'

[0218] DDC contains a Translation Component (TRC), whose task is to convert selected data records into categorical statements. This is done using the following Method:

[0219] Method 7:

[0220] Input: Set D of selected data records

[0221] Output: Categorical sentence set S Steps:

1. For all data records r(b_1,b_2,...b_n) ∈ D:
   i. Apply Definition 3:
   ∀r∈ R, ∀b_i∈ R: iff{
   s_1=b_1_is_property$_1$_of_r,
   s_2=b_2_is_property$_2$_of_r,
   ....
   s_i=b_i_is_property$_j$_of_r.
   }
   ii. Set ∀i: S=S∪ s_i
2. Return S.

[0222] 5. LRC The Language Recognition Component has the task of converting sentences in natural language into categorical sentences. The opposite direction is trivial. To ensure this according to the invention, only noun sentences are taken into account. Different languages have different procedures in this regard, but all are based on being able to distinguish verbs, nouns and their connections at the word level. In Latin languages, this distinction is achieved by experimentally using a lexicon to look at each word in a sentence first as a verb and then as a noun. In Latin, there is generally ambiguity at the word level (at least between verb and noun). In Semitic languages and especially in the Arabic language, diacritics are used for precisely this task. Differences between verb, noun and other parts of speech are therefore recognizable at the syntactic level. This is the basic idea of the following Method, which is specially invented for the Arabic language. Its general definition also allows other languages that, similar to the Arabic language, contain syntactic structures that reflect semantic characteristics.

[0223] Method 8:

[0224] Input: Natural language sentence S, Diacritics-Grammar G, noun category sentence assignment list z, lexicon L

[0225] Output: Categorical sentence S'

[0226] Steps:

[0227] 1. Result structure={ }

[0228] 2. For all words w S:

[0229] i. Search w in L

[0230] ii. If w found:

[0231] Add verb/noun/determined/indefinite tags to the

[0232] Result structure on, else

[0233] cancellation

[0234] 3. Use G to find a correct derivative of S.

[0235] 4. If derivation found:

[0236] a) Search result structure in z

[0237] b) If the result structure is found:

[0238] Set S'=categorical sentence, else

[0239] cancellation

[0240] 5. Return S.

[0241] The following example explains how to perform this procedure for the Arabic language:

[0242] Given the sentence S="رأيُ الولدِ خيرُ رأيٍ"

[0243] (English: "The boy's opinion is the best opinion.")

[0244] Let G be the Diacritics-Grammar from Definition 7.

[0245] After Step 2 (always read from right to left)

[0246] ResultStructure="noun/ٍ indefinite noun/ٌ indefinite

[0247] Noun/ٍ determines noun/ٌ indefinitely".

[0248] The derivation in FIG. **4**A is then a correct derivation of S from G. FIG. **4**B, however, shows a failed derivation if the diacritics are not taken into account.

[0249] Assignment list z contains the following data records (read from right to left):

Noun sentence	Category-Sentence
Noun S_1 (determined), noun (indefinite) S_2	(S_2 is S_1)
Noun S_1 (undetermined), noun (determined) S_2, noun (indefinite) S_3	($S_3 + S_2$ is S_1)
Noun S_1 (undetermined), noun (undetermined) S_2, noun (determined) S_3, noun (indefinite) S_4	($S_4 + S_3$ is $S_2 + S_1$)

[0250] Method 8 outputs S'=[(S_4+S_3 is S_2+S_1)] as a category set for the above example.

[0251] 6. IDC This invention assumes that the ontology relevant to the application is closed. The consequence of this is that complete induction can easily be applied to parts of this ontology, since the combinatorics always remain constant. The aim is to enable the user to discover unknown constraints and thereby increase the coherence of the logic of the overall system. The following basic Method for this component is defined in such a way that it can also be used for database records.

[0252] Method 9:

[0253] Input: set M of the selected decision terms, V set of the values of these terms, set S of the selected conclusion terms, V' set of their values

[0254] Output: set M' of the categorical constraints

[0255] Steps:

[0256] 1. Establish a combinatorics table T for M and its values V

[0257] 2. For all alle s∈S, v_1,v_2, . . . v_n, V_i∈V' is the value of s:

[0258] a) For each combinatorics theorem in T

[0259] Set a suitable v_i (automatically, i.e., via recursive function, or manually)

[0260] 3. For all subsets T' of M with respective values t_1, . . .t, ∈V:

[0261] a) Verify whether there is an s∈S, v_i is the value of s, such that:

[0262] Each repeated appearance of the values t_1, . . . , t_n in T in the column s contains the value v_i

[0263] b) If yes: set new constraint=(t_1& . . . &t_n>v_i)

[0264] 4. Return all the constraints found

[0265] The following example illustrates the use of the above method in the context of the printing application.

[0266] Let M={printing group, type, numbering unit}, V={{1-color, multi-color}, {Heidelberg, Roland}, {available, not available}}. T (steps 1. and 2.) looks like this:

Printing group	Type	Numbering unit	Conclusion s = Parts on discount
1-color	Heidelberg	available	1
1-color	Heidelberg	unavailable	0
1-color	Roland	available	0
1-color	Roland	unavailable	0
multicolor	Heidelberg	available	1
multicolor	Heidelberg	unavailable	1
multicolor	Roland	available	1
multicolor	Roland	unavailable	1

[0267] The following subsets of M are formed in step 3:

[0268] (printing group)

[0269] (Type)

[0270] (Numbering unit)

[0271] (printing group, type)

[0272] (printing group, numbering unit)

[0273] (Type, numbering unit)

[0274] (printing group, type, numbering unit)

[0275] For (printing group) one finds the constraints: ((multicolor)>1)

[0276] There are no constraints for (type)

[0277] There are no constraints for (numbering unit)

[0278] For (printing group, type) one finds the constraints: ((1-color & Roland)>0), ((multicolor & Heidelberg)>1), ((multicolor & Roland)>1)

[0279] For (print group, numbering unit) one will find the constraints: ((1-color & not available)>0), ((multicolor & not available)>1), ((multicolor & available)>1)

[0280] For (type, numbering unit) one can find the constraints: ((Heidelberg & available)>1)

[0281] There are no constraints for (print group, type, numbering unit)

[0282] The constraints found reflect the following rules of the printing press industry:

[0283] 1) One gets a discount on spare parts from multicolored Heidelberg and/or Roland printing machines

[0284] 2) One does not get a discount for spare parts from Roland 1-color printing machines

[0285] 3) If no numbering unit was delivered with a 1-color printing machine, whether Roland or Heidelberg, then there is no discount for spare parts of this machine.

[0286] 4) One gets a discount for multi-colored machines, regardless of whether numbering units were delivered or not

[0287] 5) Spare parts for a Heidelberg machine whose numbering unit has been delivered are always subject to a discount

[0288] 7. RRC The Rational Response Component has the task of answering queries through logic-supported reactions. This is done on the assumption that there is a list of all categorical constraints, which is either explicitly defined in ODC or derived by Method 9 in the IDC.

[0289] Method 10:

[0290] Input: SQL query Qry, list of categorical constraints catCons

[0291] Output: Set M of all categorical constraints that belong to the query

[0292] Steps:

[0293] 1. Execute Qry. Name the resulting table ReT.

[0294] 2. Form the set M' of all decision terms that occur in ReT

[0295] 3. For each term b of M':

[0296] a) Search b in catCons

[0297] If found: add constraint to the list of results

[0298] 4. Return the list of results

[0299] As an alternative to SQL queries, categorical records can be searched directly in the RS. Since Method 6 in the DDC guarantees, by means of completeness and unity of the ontology, that every derivable sentence also exists in the extension of the ontology, a simple search procedure is sufficient for this type of query method.

[0300] Method 11:

[0301] Input: Categorical sentence s, list of all categorical sentences S, list of all categorical constraints const

[0302] Output: List of all categorical sentences/constraints that were involved in the derivation of s

[0303] Steps:

[0304] 1. Search s in S

[0305] 2. Search s in const

[0306] 3. If found: return S∪const, else

[0307] cancellation

DESCRIPTION OF THE DRAWINGS

[0308] The present invention may be better understood, and its numerous features and advantages made apparent to those skilled in the art by referencing the accompanying drawings.

[0309] FIG. **1** illustrates the various components of the overall system with ODC, DTC, CDC, IDC, DDC, LRC, TRC, and RRC;

[0310] FIG. **2** executes an example of Method 1 on the CNF clause set:

[0311] S={{0,1} {0,2} {1,3} {2,3} {3,4}} where Method 1 sets up the BDD for S, but cannot convey any information about the number of possible solutions;

[0312] FIG. **3** executes Method 5 on the BDD created for the clause set

[0313] S={{0,1} {0,2} {0,4}} by means of Method 1;

[0314] FIG. **4**A shows a correct derivation of S from G according to Method 8 with input of a natural language sentence S, Diacritics-Grammar G, noun category sentence assignment list z, and lexicon L;

[0315] FIG. **4**B provides a failed derivation if a Diacritics-Grammar is not taken into account;

1. A method for creating an efficient, logically complete, ontological level in the extended relational database concept, characterized in that the catalog level is extended to a logically complete and closed ontology, called a Rational System (RS).

2. Method according to claim **1**, characterized in that RS includes, inter alia, the following components:

a) Ontology Description Component (ODC), consisting of a graph- or logic-based editor of axioms (ontology structure) and facts.

b) Inductive Derivative Component (IDC), whose task is to generate combinatorics for selected parts of the overall system, for which no explicit constraints are known. In consequence, a complete induction procedure assists in inferring such constraints.

c) Deductive Derivative Component (DDC) that applies syllogisms to selected parts of the overall system using a Language Recognition Component (LRC). A Translation Component (TRC) ensures that records from the database are rewritten into categorical statements.

d) Rational Response Component (RRC), which can explain each response to a request made to the overall system by means of stored constraints.

3. Method according to claim **2**, characterized in that categorical constraints are derived by means of complete induction (Method 9) in the IDC via selected parts of the overall system.

4. Method according to claim **2**, characterized in that syllogisms and hypothetical syllogisms in the DDC are applied to selected categorical data sets of the entire system until no new sentences can be derived (Method 6). In addition, DDC contains a Translation Component (TRC) whose task is to convert selected data sets into categorical statements (Method 7). The language Recognition Component (LRC) of the DDC, however, has the task of converting sentences in natural language into categorical sentences.

5. Method according to claim **2**, characterized in that inquiries are answered by logic-assisted reactions. This is done by means of Method 10, which abstracts concepts and the associated categorical sentences/constraints from SQL-queries. Alternatively, a categorical sentence can be searched directly and the associated terms/constraints found (Method 11).

6. Method according to which a Decision Tree Component (DTC) explicitly makes available selected CNF-form constraints by means of SAT-solver methods (Methods 1, 2, 3) as Binary Decision Diagrams (BDDs).

7. Method according to claim **6**, characterized in that possible solutions of the CNF-formula are counted by means of Methods 4 and 5.

8. Method according to claim **6**, characterized in that the concept of a logical variable x is based on the truth pattern of x obtained from the truth table.

9. Method according to claim **6**, characterized in that a combinatorial space is generated with the resolution, which does not depend on the classical variable value combinatorics, but on the sequence and interaction of the truth pattern of the variables in the to be processed formula.

10. Method according to claim **6**, characterized in that by means of the combinatorial space, a canonical division of the clause set in smaller clause sets is carried out, whose entire final value depends on their respective truth values alone.

11. Method according to claim **6**, characterized in that the clause classification criteria of Method 2 (1-4) are met by applying as well as both, the resolution methods described in Methods 1 to 3 and the resulting CNF-formulas.

12. Method according to claim **6**, characterized in that the combinatorial space by use of this canonical partition is converted to an efficient decision tree (BDD), which is equivalent to the classical truth table, although it does not include all the truth table combinatorics.

13. Method of using a Dia-Grammar for automatic recognition of natural language sentences. The Dia-Grammar allows control of the sentence and/or word derivation method by the umlauts and/or meta-symbols known from the natural language syntax (Method 8).

* * * * *