**Verfahren zur Erstellung einer** **effizienten, logisch vollständigen, ontologischen Ebene im erweiterten relationalen Datenbankkonzept**

**Beschreibung**

**Anwendungsgebiet der Erfindung**

Die Erfindung betrifft ein Verfahren zur Erstellung eines ontologischen Begriffssystems in der Meta-Ebene des relationalen Datenbankmodells, das zugleich effizient und logisch vollständig ist. Das Gesamtsystem wird im weiteren Verlauf der Schrift: Rationales System (RS) genannt. Die Komponenten der ontologischen Ebene dieses Systems (aufgeführt in Fig. 1), sind die folgenden:

1. Ontologie-Beschreibungs-Komponente (OBK), bestehend aus einem Graph- oder Logik-basierten Editor von Axiomen (Ontologie-Struktur) und Fakten.
2. Entscheidungsbaumkomponente (EBK), die als Ziel hat, ausgewählte, in CNF-Form dargestellte Constraints, mittels SAT-Solver Methoden explizit als binäre Entscheidungsdiagramme zur Verfügung zu stellen. Diese Komponente enthält eine CNF-basierte Constraints-Definitions-Komponente (CDK) und ein Lösungszählungsverfahren (LZV).
3. Induktive Ableitungskomponente (IAK), dessen Aufgabe darin besteht, die Kombinatorik für ausgewählte Teile des Gesamtsystems zu generieren, wofür man keine expliziten Constraints kennt. Ein vollständiges Induktionsverfahren hilft dann beim Inferieren solcher Constraints.
4. Deduktive Ableitungskomponente (DAK), die Syllogismen, mittels einer Spracherkennungskomponente (SEK) auf ausgewählte Teile des Gesamtsystems anwendet. Eine Übersetzungskomponente (UEK) gewährleistet, dass Datensätze aus der Datenbank in kategorischen Aussagen umformuliert werden.
5. Rationale Antwortkomponente (RAK), die jede Antwort zu einer an das Gesamtsystem gestellten Anfrage mittels abgespeicherten Constraints (aus EBK oder IAK) erklären kann.

Das RS erlaubt im Unterschied zu bekannten Verfahren in deduktiven- und relationalen Datenbanken nicht nur lineare Verarbeitungszeiten von Eingaben und Verbesserung von Datenbankabfrageprozeduren ohne die logische Konsistenz zu gefährden (vgl. DE102015013593A1), sondern mittels EBK und DAK auch ein ausgeglichenes Anwenden von bekannten, effizienten, logischen Verfahren auf das Gesamtsystem: Während EBK effiziente Methoden der a-priori Vervollständigung für ausgewählte, schwierige boolesche Funktionen einsetzt, erzielt DAK unabhängig von der üblichen SQL-Maschinerie schnelle Antwortzeiten. Das wird erfindungsgemäß durch einfache, natursprachlich unterstützte, Syllogistik-basierte Verfahren realisiert. Darüber hinaus besteht durch RAK die Möglichkeit einer intelligenten Systemreaktion, die jede Antwort mittels Regeln der Logik begründet (daher die Eigenschaft: ***Rational***). Die durch IAK gewonnenen Constraints benötigen keine expliziten Verifizierungs- und/oder Testverfahren, da sie einer mathematisch stringenten, vollständigen Induktion entstammen. Mittels solcher Constraints erlaubt die IAK des weiteren eine kompakte Darstellung der Kombinatorik ausgewählter Teile des Gesamtsystems. Somit erfüllt das RS alle erforderlichen Kriterien eines praktisch umsetzbaren logischen Systems in seiner allgemeinsten Form, das für terminologische- wie auch logische Kontrolle in den meisten Datenbankanwendungen Einsatz finden kann.

**Charakteristik des bekannten Standes der Technik**

Bekannt sind Verfahren zur Erzeugung von Ontologie Modellen aus Anforderungsdokumenten und Software sowie die Durchführung von Konsistenzprüfungen zwischen Anforderungsdokumenten und Softwarecode unter Verwendung von Ontologie Modellen. Aus der Vielzahl von Anforderungsdokumenten die in einer Datenbank erhalten sind, werden Terme identifiziert. Ein Prozessor ordnet jedem Term ein Wortart-tag zu. Das Wortart-tag gibt eine grammatikalische Verwendung jedes Terms in den Anforderungsdokumenten an. Der Prozessor klassifiziert jeden Term auf der Grundlage der Wortart-tags. Um eine Ontologie zu bilden, identifiziert die Klassifizierung, ob jeder Term ein Teil, ein Symptom, eine Aktion, ein Ereignis oder ein Fehlermodus ist. Der Prozessor konstruiert eine Ontologie-basierte Konsistenzmaschine. Eine Konsistenzprüfung wird durch Anwenden der Ontologie-basierten Konsistenzmaschine zwischen Ontologien, die aus zwei Kontextdokumenten extrahiert werden, durchgeführt. Inkonsistente Terme zwischen den Kontextdokumenten werden identifiziert. Mindestens eines der Kontextdokumente mit inkonsistenten Termen wird korrigiert (vgl. DE102015121509A1). Nachteilig dabei, ist dass Konsistenz und Vollständigkeit der so konstruierten Ontologie außer Betracht gelassen wird.

Bekannt sind Verfahren, bei denen Ambiguität gehandhabt wird, die bei der Zusammenführung natürlich-sprachlicher Informationen mit dem durch Ontologien repräsentierten Wissen auftritt. Ambiguität ist darauf zurückzuführen, dass der gleiche natürlich-sprachliche Bezeichner mehrere Elemente der Ontologie bezeichnen kann. Diese Verfahren stellen eine grundlegende Methodik vor, wie trotz Ambiguität das passende Ontologie Element bestimmt werden kann. Der Ansatz orientiert sich am menschlichen Vorgehen und an der Verwendung des Kontextes zur Monosemierung. Dieser stellt den Zusammenhang zwischen den Entitäten dar, die im Text erwähnt werden. Diese Verfahren bilden Texthüllenkontext anhand der Zusammenhänge innerhalb des Ontologie Graphen nach, und nehmen durch dessen Analyse eine Disambiguierung vor. Nachteilig ist, dass die logische Ableitung (ob Deduktion oder Induktion) davon nicht tangiert ist (vgl. Kleb, J.; Ontologie-basierte Monosemierung Bestimmung von Referenzen im Semantic Web; KIT Scientific Publishing; 2015; DOI 10.5445/KSP/1000031500).

Bekannt sind Verfahren zur Unterstützung der Suche nach bewährten und existierenden Lösungen im Rahmen der Entwicklung technischer Produkte. Der Zugriff auf diese Lösungen wird über die Betrachtung der Funktionen technischer Systeme aus Anwender- und Gebrauchssicht ermöglicht. Dem Suchenden wird mittels semantischer Netze ein geeigneter Lösungsraum mit relevanten Lösungen bereitgestellt. Dieser kann den Raum nach verschiedenen Kriterien eingrenzen und erweitern. Nachteilig ist dabei, dass im Falle von komplexeren Suchaufgaben die logische Maschinerie außer Betracht gelassen wird (vgl. Gaag, A.; Entwicklung einer Ontologie zur funktionsorientierten Lösungssuche in der Produktentwicklung; Verlag Dr. Hut; 2010, ISBN 978-3-86853-731-4).

# Bekannt sind Verfahren zur Integration einer semantischen Datenverarbeitung in einem Gerät, insbesondere in einem Feldgerät der Automatisierungstechnik. Dabei wird ein generisches Beschreibungssprachenschemas zur Definition eines semantischen Depots als Ausgangsbasis verwendet. Dieses Beschreibungssprachenschemas wird mit Inhalten einer Ontologie zur semantischen Repräsentation einer Funktionsweise des Geräts angereichert. Dabei werden Klassen und/oder Subklassen der Ontologie zusammen mit mindestens einer den Klassen und/oder Subklassen zugeordneten Eigenschaft aus der Ontologie entnommen, in eine entsprechende Schemadeklaration umgeformt und schließlich diese Schemadeklaration in das Beschreibungssprachenschema eingefügt. Anschließend wird aus dem Beschreibungssprachenschema eine oder mehrere Grammatiken erzeugt, vorzugsweise Grammatiken gemäß dem standardisierten Datenformat *Efficient XML Interchange*, welche im Gerät integriert werden. Ein besonderer Vorteil der Verfahren liegt in einer wesentlich kompakteren semantische Datenverarbeitung und Datenübertragung. Nachteilig ist das Außerbetrachtlassen der logischen Merkmale der erreichten semantischen Datenverarbeitung und der damit verbundenen Ontologie Rolle (vgl. WO2016110356A1).

# Bekannt sind Verfahren, die einem Benutzer die Möglichkeit geben, aus importierten Ontologien Datenbanken und ähnliche Anwendungen zu kreieren. Diese Datenbanken können spezifisch konfiguriert werden und kommen mit Fehlererkennungsregeln. Die Suche in diesen Datenbanken geschieht nach „Bedeutungen” statt nach konkreten Wörtern. Ontologie-Verwaltung garantiert konsistente Datenintegration, Wartung und Flexibilität, und ermöglicht auch leichte Kommunikation zwischen mehreren Datenbanken. Dabei werden nur die relevanten Ontologie Teile für eine spezifische Anwendung in Betracht gezogen (Teilmodell). Um Effizienz sicher zu stellen, wird das Teilmodell dann in eine objektorientierte, API-unterstützende *Java* Anwendung übersetzt. Nachteilig ist, dass man keine Konsistenz oder Vollständigkeitskriterien der importierten Ontologien voraus- und/oder durchsetzen kann (vgl. US6640231).

# Bekannt sind Verfahren, bei denen eine Ontologie-bezogene Anfrage zur Hilfe genommen wird, um in Datenbankanwendungen Synonyme von Wörtern zu generieren, die zusätzlich zu den in den Datenbankanfragen benutzen, relevante Datensätze finden könnten. Nachteilig ist, dass diese Suche bei logisch komplexen Datenbankabfragen noch mehr Komplexität generiert (vgl. US8135730).

# Bekannt sind Verfahren, nach denen Paare ähnlicher Terme die in einem OWL-Dokument vorhanden sind, in einer relationalen Datenbank gespeichert und dann in Datenbankabfragen benutzt werden, die semantische Beziehungen der beiden Terme in Anspruch nehmen. Nachteilig ist, dass diese Verfahren logisch komplexe Datenbankabfragen nicht beeinflussen können (vgl. US7328209).

Bekannt sind allgemeine Verfahren zur Behandlung der Constrainst-Programmierung im Rahmen kontinuierlicher- oder diskreter Variablen zur Modellierung mathematischer- oder algorithmischer Probleme (vgl. DE4411514A1 und US5216593). Nachteilig ist, dass diese Verfahren sich nicht für allgemeine Datenbankkonzepte eignen.

Bekannt sind Verfahren, bei welchen die am wichtigsten erscheinenden referentiellen Integritätsbedingungen bei der Erstellung eines SQL-Ausführungsplanes im Voraus bestimmt werden (vgl. US5386557). Nachteilig ist, dass damit anwendungsbezogen keine benutzerdefinierten Constraints mehr möglich sind. Um dieses zu gewährleisten, wird bei US5488722 offengelegt, eine benutzerdefinierte Constraints-Methode für Datenbanken von der Wahrscheinlichkeit eines Konsistenzbruches abhängig zu machen. Danach werden diejenigen Constraints, welche wahrscheinlich nicht eingehalten werden, zuerst angewandt. Nachteilig ist, dass dieses Verfahren nicht allgemein anwendbar ist, da die Erstellung einer entsprechenden Ordnungshierarchie zur Anwendungspriorität einer Constraints-Anwendung das Ergebnis einer Datenbankabfrage voraussetzt, so dass auch bei Rekursion von Datenbankabfragen immer Constraints existieren, welche über keine Rangordnung verfügen. Somit besitzen sie zur Anwendung keinen Wahrscheinlichkeitswert.

Zur Effizienzsteigerung der Abfrageprozeduren sind Verfahren bekannt, bei welchen innerhalb eines logischen Programms verschiedenen Prädikaten ein bestimmter Ordnungsrang zugewiesen wird (vgl. EP 0545090A2). Dabei wird die Zuordnung der logischen Prädikate auf Ebene des Begriffssystems berührt, nicht aber das logische Vervollständigungsverfahren (SLD-Resolution).

Bekannt sind Verfahren, um Bedingungen für große Mengen von Daten einer Datenbank zu überprüfen (vgl. EP 0726537A1; Hirao, T.: Extension of the relational database semantic processing model, in: IBM Systems Journal, Vol. 29, No. 4, 1990, S.539 bis 550, und Lippert, K.: Heterogene Kooperation, in: ix Multiuser Multitasking Magazin 7/1994, S. 140 bis 147). Die darin angegebenen Verfahren sind prozedural angelegt, und entbehren damit die logisch-deklarative Form.

Bekannt sind Verfahren nach DE19725965C2, die allgemeine Constraints im erweiterten relationalen Datenbankkonzept auf der deduktiven Katalogebene behandeln. Nachteilig ist, dass die Extension der logischen Theorie sehr groß, d.i., exponentiell in der Länge der benutzten logischen Formeln sein kann.

Bekannt sind Verfahren nach DE102015013593A1, bei welchen allgemeine Constraints-Behandlung im erweiterten relationalen Datenbankkonzepten derartig durchgeführt wird, dass logische Vervollständigkeitsverfahren auf Katalogebene effizient (d.i., nicht exponentiell) angewandt werden können, um damit maximale Ausführgeschwindigkeit von logischen Abfragen zu gestatten. Nachteilig ist, dass Begriffssystem-bezogene Abfrageverarbeitung nicht explizit behandelt wird. Die Möglichkeit der Benutzung von vollständigen Induktionsverfahren auf das Gesamtsystem wird außer Acht gelassen. Auch werden nach DE102015013593A1 keine Verfahren angegeben, mit denen man die Lösungen einer CNF-Formelmenge effizient zählen kann. Hinzu kommt, dass Syllogistik-basierte Deduktion keine Anwendung findet. Antworten des in DE102015013593A1 beschriebenen Systems mangeln an einer logischen, natürlichen Erklärungskomponente.

Nicht bekannt sind Verfahren nach denen man ein logisch vollständiges, effizientes, ontologisches Begriffssystem in der Katalogebene eines relationalen Datenbanksystems erstellen kann, das Deduktion wie auch vollständige Induktion in ihrer allgemeinsten Form insofern ermöglicht, als dass logische (rationale) und natursprachliche Erklärungen aller Systemreaktionen erzielt werden können.

***Die logische Vervollständigung***

Die in einem logischen System vorkommenden Regelmengen müssen Korrektheits- und Vollständigkeitskriterien genügen. Im Kontext dieser Erfindung wird als „vollständig“ bezeichnet, wenn Axiome und Deduktionsregeln alles deduzierbare auch explizit ableiten.

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 (MA), 1986), (Bayer, R.: Qüry Evaluation and Recursion in Deductive Database Systems, Manuscript, March 1985) oder (Lozinskii, E.L.: Evaluation queries in deductive databases by generating, Proc. Int. Joint Conference on A.I., 1:173-177, 1985) existieren alternative Verfahren zur Vervollständigung.

Diese betreffen entweder den Inferenzprozess selber, d.h. die Art mit der man die Regeln anwendet, oder die für die Anfrage relevanten Fakten der Datenbasis. Eine Methode, welche sich mit dem Inferenzprozess selber befasst, ist die sogenannte semi-naive Vervollständigung (vgl. 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).

Diese versucht die unnötige Wiederholung von Generationsschritten zu vermeiden, indem sie nur die inkrementell-generierten Fakten, d.h. die Fakten, die in der letzten Iteration entstanden sind, berücksichtigt (vgl. 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, oder Marq-Puchen; Gallausiaux, M.; Jomien: Interfacing Prolog and Relational Database Management Systems, New applications of databases, Gardavin and Gelaube eds. Academic Press, London, 1984).

Die Annahme ist, dass  für jede Relation R*i*  
( ist hier die inkrementelle Änderung von R*i* und F(R*i*) der funktionale Ausdruck, den man aus dem Rumpf einer Regel deduziert). Im allgemeinen kann man  nicht einfach als Funktion von  berechnen. Im Falle von linear-rekursiven Regeln ist das aber möglich, weil .

Solange man davon ausgehen kann, dass die Regeln linear-rekursiv sind, ist die semi-naive Vervollständigung eine effiziente Methode. Wenn man aber die Flexibilität erweitert und nicht-lineare Rekursionen erlaubt, ist diese Methode nicht mehr günstig (die frühen Realisierungen von semi-naiven Ansätzen sind für diese Art von Rekursion nicht gültig). Des weiteren wird der exponentielle Aufwand nur durch die Reduktion der Anzahl der Fakten des einen für den letzten Deduktionsschritt relevanten Musters gemindert. In einer Verknüpfung , wo S diesem Muster entspricht, werden zwar jeweils die Fakten die S unifizieren in Betracht genommen, Verknüpfungen zwischen den A*i*´s müssen jedoch immer wieder hergestellt werden. Man hat festgestellt, dass die Erstellung der UND-Verknüpfungen den aufwendigste Schritt in der Vervollständigungsprozedur ausmacht.

Die sogenannte APEX-Prozedur (vgl. Lozinskii, E.L.: Evaluation queries in deductive databases by generating, Proc. Int. Joint Conference on A.I., 1:173-177, 1985) ist ein Verfahren anderer Art. Zuerst werden die für eine definite Abfrageklausel W? (Ziel) relevanten Fakten einer Datenbasis generiert, um erst dann den Vervollständigungsprozess zu beginnen. Die relevanten Fakten werden mit Hilfe von sogenannten Regelsystemgraphen berechnet. Diese beinhalten alle logischen Verknüpfungen zwischen Regeln der Datenbasis.

Sie werden mit einem Abfragegenerationsprozess, der im Falle von wichtigen UND-Verknüpfungen weitere Abfragen W1?,W2?....usw. generiert, gekoppelt. Die Generation erfolgt durch Konstantenaustausch (engl.: sideway information passing SIP) zwischen der Abfrage oder den Abfragen und den Fakten der jeweiligen Verknüpfung(en). Eine weitere Methode dieser Klasse ist QSQ (vgl. Vieille, L.; Recursive axioms in deductive databases: The Query-Subquery approach, Proc. First Int. Conf. on expert database systems, Kerschlag ed., Charlston, 1986). Wie in APEX werden die Datenbasisregeln dort für die Generation von neuen Abfragen benutzt. Die relevanten Fakten werden jedoch wie in PROLOG mit Hilfe eines „backward-chaining“ Verfahrens linear und in der Tiefe (engl.: depth-first) gesucht. Im Falle von rekursiven Prädikaten, werden die Abfragen mit Hilfe von den bereits gefundenen Fakten durch SIP generiert.

Der wesentliche Unterschied zwischen APEX und QSQ auf der einen Seite und der semi-naiven Vervollständigung auf der anderen Seite ist somit, *dass die Lösung der semi-naiven Vervollständigung das allgemeine und prinzipielle Problem der Inferenz von Grundfakten behandelt, wohingegen die beiden anderen Methoden lediglich versuchen, die üblichen Inferenzmechanismen über die Berücksichtigung der relevanten Fakten zu optimieren*.

Magic-sets (vgl. Beeri, C.; Ramakrishnan; On the power of magic, Proc. sixth ACM SIGMOD-SIGACT Symp. on principles of database systems, San Diego, CA, March 1987) ist eine Modifikation von QSQ, die die Variablenbindungen (engl. adornments) in Form von neuen „magischen“ Regeln zu einem Programm anfügt oder an der rechten Seite einer Klausel als Restriktionen anbindet. Beginnend mit der Zielklausel, wird eine Menge neuer Prädikate generiert. Durch SIP gelingt es, diese „adornments“ weiterzuleiten. Das Resultat ist eine neue, modifizierte Version des ersten Programmes, welche in manchen Fällen effizienter ist. Beispielsweise wird aus dem Programm:



und der Abfrage 

das neue „magische“ Programm:



Das neue magic-Prädikat stellt eine Restriktion der zulässigen Substitutionen dar. Es verbindet die Konstanten des Programmes systematisch miteinander.

***Pragmatische Betrachtungen zum Wesen einer logischen Variable***

Das Grundproblem der Constraints-Behandlung besteht darin, den bei der Anwendung einer Constraint-Menge von Regeln entstehenden Aufwand zu reduzieren. Lösungsansätze laufen darauf hinaus, Instanzen dieser Regeln zu generieren, bevor eine adäquate Anwendung der Constraints eingeschaltet wird. Die Tatsache, dass viele Lösungsansätze durch Variableninstanzierungen einen hohen Grad an Effizienz erreichen, fordert eine Grundsatzdiskussion über die Bedeutung einer Variablen in der geschlossen Welt einer deduktiven Datenbank und eines RD-Modells. Die in der mathematischen Logik (und damit in der logischen Programmierung) übliche Bedeutung einer Variablen läuft darauf hinaus, sie als eine vom Domain der Anwendung gelöste Entität zu betrachten. Damit ist der Zugang einer Variableninstanzierung zum Domain unklar, denn es existieren zur Beschreibung dieser Instanzierungsverfahren keine expliziten oder impliziten Vorschriften in der Interpretation. Dieser Zugang ist damit der Implementation einer logischen Maschine überlassen, was zu erheblichen Aufwandsproblemen führen kann.

DE19725965C2 löst dieses Problem durch die Einführung der Herbrandabstraktionenstruktur. Hierbei werden Variablen als Abstraktionen von Begriffen und Begriffsbeziehungen in der Katalogebene betrachtet. Dieser Ansatz erlaubt es, alternative Vervollständigungsverfahren zu beschreiben, die es ermöglichen, von einer Standard-Herbrandinterpretation zu einer „vollständigeren“ mittels eines beliebigen Abstraktionsgrades zu kommen. Kehrt man den „Abstraktionsprozess“ um, d.h., beginnt man mit uninstanzierten Klauseln, so ermöglicht die Herbrandabstraktionenstruktur Verfahren, die Klauseln eines logischen Programmes in eine Menge „instanzierterer“ Klauseln aufteilen können. Dies führt wiederum zu der dort beschriebenen Effizienzsteigerung (Linearisierung). Das in DE19725965C2 (Alg. 2) formalisierte Verfahren gibt jedoch keine Methode an, wie die Instanzierung der Regeln optimiert werden könnte. Dieses könnte in einer Herbrandabstraktionenstruktur mannigfaltig erreicht werden. Zudem besteht der wesentliche Schwachpunkt bei der Benutzung der Herbrandabstraktionenstruktur darin, dass sie im schlechtesten Falle einen exponentiellen Suchraum darstellt.

Das in DE102015013593A1 vorgestellte Verfahren führt hingegen in Abhängigkeit einer neuen Darstellung von Variablen als Teile der klassischen Wahrheitstabelle, auch Musterzeichenketten bzw. Musterbäume genannt, zu vollständigen Auswertungsmethoden. Diese führen im Unterschied der zum Stand der Technik zählenden Resolutionsverfahren, zu kleinen Suchräumen, in denen lineare Verarbeitungszeiten von Eingaben realisiert werden können. Mit dem Begriff „Eingaben“ sind vorliegend stets Instanziierungen von logischen Formeln gemeint. Für die Generierung der Extension wird ein Verfahren eingesetzt, das Musterbäume statt Klauseln auflöst.

In diesem Zusammenhang sind zwei Arten von Resolutionsverfahren von Formeln/Klauseln (auch: Solver genannt) bekannt: Vollständige und unvollständige. Dabei wird ein Solver vollständig genannt, wenn es sowohl feststellen kann, dass eine Formel erfüllbar, als auch dass sie unerfüllbar ist. Nicht alle Formeln, die in einer Solver Formelmenge vorkommen können, fallen in dieselbe Kategorie. Im Allgemeinen wird in der Praxis zwischen drei Kategorien unterschieden:

- *Random*: Formeln, die zufällig nach einem Schema erzeugt werden, dass sich „fixed clause length model” nennt (man gibt nur die Anzahl der Variablen und Klauseln an und wie lang eine Klausel sein soll, der Rest wird zufällig generiert)

- *Crafted*: Formeln, die aus schwierigen kombinatorischen Problemen, wie Graphenfärbung, abgeleitet werden

- *Application*: Formeln, die aus Anwendungen in der Realität abgeleitet werden (z.B. Schaltkreisverifikation)

Nicht alle bekannten Solver-Paradigmen kommen gleich gut mit allen Formelkategorien zurecht. Man unterscheidet zum heutigen Stand der Technik vier Solver-Arten welche in DE102015013593A1 dediziert behandelt werden. Alle vier Solver-Verfahren können durch folgende Merkmale charakterisiert werden und sind damit von DE102015013593A1 wie auch vom nachfolgend vorgestellten erfindungsgemäßen Verfahren signifikant verschieden:

1. Sie sind ein Beispiel für die Anwendung von *Tarskis* semantischem Wahrheitsbegriff’ auf Formeln der mathematischen Logik. Grundsätzlich schreibt dieser Begriff vor, dass Variablen getrennt von ihren Bedeutungen bzw. Werten existieren. Diese Bedeutungen werden in den Formeln zurückersetzt, damit diese erfüllt werden. Somit werden Variablen (und ihre dazugehörigen Literale) *lediglich wie Behälter betrachtet*, die nicht zulassen, dass man strukturelle Informationen über die in ihnen gespeicherten Daten ableiten bzw. nutzen kann.

2. Nebenprodukt dieser Sichtweise ist, dass algorithmische Verfahren gezwungenermaßen verschiedene Variablenauswertungen testen müssen, bevor sie eine gültige finden. Der Begriff einer *Variablenauswertung* ist demnach integraler Bestandteil dieses Verfahrens.

3. Informationen aus der konkreten mathematisch-logischen Formel, die die Aneinanderreihung von benutzten Variablen (Literalen) und ihren gegenseitigen Interaktionen betreffen, werden nicht oder nur unzulänglich angewandt (i.d.R. in Form von Heuristiken), um die/eine gültige Auswertung zu finden.

4. Alle Verfahren vermeiden die Konstruktion des gesamten kombinatorischen Raumes, da diese Konstruktion exponentiell bezüglich der Anzahl der Variablen ist. Da die Verfahren Variablenauswertungen iterativ verwenden, wird in jeder Iteration nur ein Teil des Raumes konstruiert, die Formel daraufhin ausgewertet und dann die nächste Iteration gestartet, usw.

5. Dadurch, dass die Methoden i.d.R. keine allgemeingültigen Heuristiken benutzen, ist ihre Leistung stark abhängig von der Art der Formel (Tab. 1). Dabei sind „gut”, „schlecht” und „neutral” grobe Indikatoren der erwarteten Leistung einer Methode bezogen auf eine gegebene Formelart. „SAT/UNSAT” steht für „erfüllbar” bzw. „nicht erfüllbar”:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Kategorie** | **CDCL** | **Look-ahead** | **Message-Passing** | **SLS** |
| Random SAT | schlecht | neutral | gut | gut |
| Random UNSAT | schlecht | gut | schlecht | schlecht |
| Crafted SAT | gut | neutral | schlecht | neutral |
| Crafted UNSAT | neutral | neutral | schlecht | schlecht |
| Application SAT | gut | schlecht | schlecht | schlecht |
| Application UNSAT | neutral | schlecht | schlecht | schlecht |

Tab. 1

Letztlich ist ein Solver-Verfahren bekannt, dass der klassischen Wahrheitstabellenmethode entspricht. Es unterscheidet sich von den in DE102015013593A1 beschriebenen Verfahren wie folgt:

1. Bestandteil der Methode ist die Konstruktion des gesamten exponentiellen Raumes aller Kombinationen der Variablenwerte. Nachdem dieser Raum konstruiert ist, kann man effizient feststellen, ob eine bestimmte Variablenauswertung für die jeweilige Formel „wahr“ ergibt oder nicht.

2. Dieses effiziente Feststellen benutzt im Unterschied zu allen anderen Methoden nicht das Ersetzen in der originalen Formel, sondern das einfache Suchen in dem generierten Raum, d.i., in der Wahrheitstabelle. Dadurch ist es möglich, den Wahrheitswert der instanzierten Formel zu finden, ohne von den klassischen, logischen Operatoren (AND, OR, NOT) Gebrauch zu machen, da die volle Extension dieser Operatoren, angewandt auf die logischen Werte „wahr“ und „falsch“, materiell vorliegt.

3. Die Anzahl der Variablenauswertungen die man durchgehen muss, bis ein ergiebiger Wert gefunden wird, ist im schlechtesten Falle exponentiell. Diese potentielle Exponentialität stellt den größten Nachteil dar.

4. Dadurch, dass keine Annahmen über die Formeln gemacht werden, ist die Leistung des Verfahrens, unabhängig von der Formelart.

**Ziel der Erfindung**

Diese Erfindung hat das Ziel unter Beibehaltung strengster logischer Bedingungen, relationale Datenbanksysteme in ihrem allgemeinsten Konzept, mittels einer logisch vollständigen, ontologischen Metaebene, die es erlaubt, deduktive wie auch induktive Argumentation einzusetzen, in ihren logischen Abfrageverfahren so zu optimieren, dass die Antwortprozedur lineare Effizienz in Bezug auf die Geschwindigkeit und den Speicherbedarf erfährt.

**Darlegung des Wesens der Erfindung**

Der Erfindung liegt die Aufgabe zugrunde, ein Verfahren der eingangs genannten Art zu kreieren, das relationale Datenbanksysteme in ihren Abfrageverfahren so optimiert, dass die Antwortprozedur eine Effizienzsteigerung in Bezug auf die Geschwindigkeit und den Speicherbedarf erfährt, ohne dass logische Bedingungen aufgegeben werden müssten. Dies wird durch die Einführung einer logisch vollständigen und zugleich effizienten ontologischen Ebene erzielt, die es erlaubt, anwendungsspezifische Constraints deduktiv wie auch induktiv abzuleiten und/oder auszuwerten. Diese Aufgabe wird mit Verfahrensschritten gelöst, wie sie in den Patentansprüchen 1-13 angegeben sind.

**Ausführungsbeispiel**

***Erweiterung des RDS durch das Konzept eines im Katalog sich befindlichen logisch-vollständigen, effizienten, ontologischen Begriffssystems***

Das zentrale Verfahren dieser Erfindung basiert auf der Idee, alle für die terminologische-, logische- und anwendungsspezifische Kontrolle notwendigen Daten und Regeln entweder explizit zu definieren oder mittels vollständiger Induktion im Katalog im Voraus zur Verfügung zu stellen. Dadurch wird nicht nur der Constraints-Behandlung effizienter nachgegangen, sondern auch manchen, in üblichen relationalen Systemen nicht möglichen Berechnungsanforderungen. Konsistenzeigenschaften von Datenbanken sind ausschließlich logischer Natur, und deshalb Metaprobleme. *Die Grundannahme dieser Erfindung ist, dass das ontologische Begriffssystem einer Datenbankanwendung im hinreichend konstant bleibt. Wir nennen diese Bedingung weiter unten: Geschlossene Ontologie.* Diese ist nicht mit der logischen „geschlossenen Welt Bedingung“ (Englisch: closed-world assumption) zu verwechseln, die im Rahmen logischer Systeme die Tatsache ausdrückt, dass Fakten, die nicht explizit in der Datenbank abgespeichert sind, als „falsch“ betrachtet werden. Das folgende Beispiel erläutert dieses Verfahren:

Sei eine Gebrauchtdruckmaschinendatenbank gegeben. Diese enthält die Tabellen „Maschine“,„Firma“ und „Nummeriereinheit“ wie in (Fig. 1) dargestellt. Im folgenden werden mögliche Bedingungen dargelegt, die in üblichen relationalen Systemen nicht leicht handhabbar sind und die Teile des gesamten, hier dargestellten Systems beanspruchen:

1. Unter anderem besitzt die Tabelle „Maschine“ die Felder „Maschinentyp“ und „Druckgruppe“. Diese sind von speziellem Interesse, da ihre Kombination wichtige, in der Druckindustrie bekannte Constraints modelliert. Z.B. kann eine Polar-Maschine nicht zur Gruppe 5-Farben Druckmaschinen gehören, da der Typ „Polar“ Schneidemaschinen darstellt. Ähnlich besitzt eine Heidelberg-Tiegel nie mehr als 2 Farbwerke, so dass die Kombination (Typ = „Tiegel“, Gruppe = „3-Farben“) keiner Logik entspricht. Wir nennen solche faktenbezogene Constraints des Begriffssystems: ***Typ-1-Constraints***
2. Maschinen können nicht mehr als 5 Nummeriereinheiten besitzen. Man kann diese Bedingung nur durch Programmierung (i.d.R.: Stored-procedures) implementieren, da weder Entitäts- noch Tabellendiagramm in der Lage sind, Bedingungen über Kardinalitäten der Beziehungen, außer durch „many“ und „one“, auszudrücken. Wir nennen allgemeine Bedingungen, die mit Kardinalitäten der Relationen im relationalen Datenmodell zu tun haben: ***Typ-2-Constraints***
3. In der Druckindustrie werden Maschinen gelegentlich von- und zu verschiedenen physikalischen Orten transportiert. Firmen, die intendieren Transportkosten minimal zu halten, benötigen deshalb unter anderem stets Kalkulationen der besten Transportrouten. Derart kalkulationsintensive Aufgaben, die mittels allgemeiner, in CNF-ausdrückbaren booleschen Funktionen dargestellt werden können, nennen wir hier: ***Typ-3-Constraints***
4. Manchmal ist es nötig, die werkseitige Standardkonfiguration eines Maschinentyps zu wissen um einen Abgleich mit der gebrauchten Maschine gleichen Typs durchzuführen (z.B. ob die Standardkonfiguration werkseitig mit- oder ohne Nummeriereinheit ist). Diese Anfrage steht sowohl mit Herstellerspezifikationen im Begriffssystem als auch mit der aktuellen Gebrauchtmaschinen-Datenbank in Zusammenhang. Wir nennen solche Constraints: ***Typ-4-Constraints***
5. Alle Anfragen, die ausschließlich mit der Datenbank in Zusammenhang stehen, werden i.d.R. mit üblichen SQL-Komponenten gehandhabt. Dazu gehört beispielsweise: „Welche Firmen liefern zur Zeit Tiegel-Maschinen?“. Wir nennen Constraints dieser Art: ***Typ-5-Constraints***
6. Anfragen wie: „Welche Teile werden mit einer Tiegel-Maschine geliefert?“ und „Welche Teile werden von einer bestimmten Firma mit einer Tiegel-Maschine geliefert?“ sind Typ-1- und Typ-4- Constraints-Beispiele, die für ihre Ausführung die Berechnung der transitiven Hülle der jeweiligen Beziehungen benötigen. In üblichen relationalen Datenbankkontexten werden hierzu Stored-Prozeduren benutzt, da SQL standardweise keine Schleifen zulässt. Wir nennen Anfragen, die mittels SQL alleine nicht gelöst werden können: ***Typ-6-Constraints***
7. Ableitungen von Formeln der Form: „Alle Maschinen der Firma X werden stets komplett überholt und gepflegt abgeliefert“ oder „Es gibt nur einige Ersatzteile, die kompatibel mit einem Nummerierwerk sind“ sind nur möglich, wenn der gesamte Datensatz- und/oder Faktenbestand des Begriffssystems in Betracht genommen wird. Diese nennen wir:

***Typ-7-Constraints***

Die folgende Tabelle (Tab. 2) zeigt die Ordnung von Constraints-Typen zu verschiedenen Systemkomponenten.

Die auf Tab. 2 folgenden Beschreibungen und Definitionen erläutern die erfindungsgemäßen Funktionalitäten der verschiedenen Komponenten des Gesamtsystems unter Bezugnahme auf Fig. 1.

|  |  |  |
| --- | --- | --- |
| Constraint | System-komponente | Kommentar |
| Typ-1, Typ-2 | CDK, EBK, RAK, Rekord-validierung | Begriffssystem- und Relationen bezogene Constraints werden in CDK in CNF-Form definiert, und mittels EBK ausgewertet. Sie werden in RAK für die Generierung von intelligenten Antworten benutzt. Darüber hinaus beeinflussen sie die übliche Rekordvalidierungskomponente. |
| Typ-3 | CDK, EBK, LZV, RAK | Allgemeine boolesche Funktionen werden in CDK in CNF-Form definiert. Solver-Verfahren 1 in EBK konvertiert sie zu BDDs (Binary Decision Diagram), dessen Informationen der RAK zur Verfügung gestellt werden. LZV liefert dazu die Anzahl der verschiedenen Lösungsalternativen |
| Typ-4, Typ-5, Typ-6 | CDK, EBK, OBK, UEK, DAK, SEK | Constraints, die sowohl mit Datenbank als auch mit Begriffssystem zu tun haben oder solche die nur datenbankbezogen sind, können in zweierlei Weisen gehandhabt werden: Entweder definiert man sie zuerst in CNF Form in CDK, dann werden sie mittels EBK ausgewertet, oder: Selektierte Datensätze werden in UEK anfangs in kategorischen Aussagen übersetzt, dann mittels DAK für die Deduktion benutzt. Da DAK Rekursion zulässt, werden hiermit auch Typ-6 Constraints abgedeckt. SEK sieht vor, dass sie in der natürlichen Sprache abgebildet werden. |
| Typ-7 | IAK, DAK, SEK | Datenbank und Begriffssystem liefern eine endliche Anzahl von Feld/Fakten- Kombinationen, für die das Erstellen von vollständigen Kombinatorik-tabellen möglich ist. Regeln können dann mittels vollständiger Induktion abgeleitet werden (Verfahren 9) und entsprechen den „all“ oder „Existenz“ – quantifizierten Formeln in Typ-7. |

Tab. 2

***Definition 1:*** Gegeben sei die Menge ***B*** aller Begriffe in einer Anwendung zu der auch allquantifizierte, existenzielle, negierte und unbestimmte Begriffe, d.h. Variablen, gehören: Ein logischer Schluss wird ***Syllogismus*** genannt, wenn jeweils zwei Prämissen (Voraussetzungen), genannt ***Obersatz*** und ***Untersatz***, zu einer Konklusion (Schlussfolgerung) führen. In einem ***kategorischen Syllogi*smus** (auch ***assertorischer Syllogismus*** genannt) sind Prämissen und Konklusion ***kategorische Urteile***, d.h. Aussagen, in denen einem Begriff aus ***B***, dem Subjekt, ein anderer Begriff aus ***B***, das Prädikat, in bestimmter Weise zu- oder abgesprochen wird.

***Definition 2:*** Sei ***kSyl*** die Menge aller bekannten, validen kategorischen Syllogismen und ***hSyl*** die Menge aller bekannten, validen hypothetischen Syllogismen der Form: ***Aus (P Ⱶ Q) und (Q Ⱶ R) wird (P Ⱶ R) abgeleitet***, wo P, Q, R kategorische Sätze und ‚Ⱶ’ syntaktische Ableitungsrelationen sind, dann wird ***kategorischer Satz s*** *logische Konsequenz der kategorischen Satzmenge S genannt (S →Syl s), wenn s mittels Anwendung von Syllogismen aus Syl=kSyl*∪*hSyl hervorgeht.*Die Liste der Deduktionsschritte die mittels Regeln aus *Syl* zu einem Satz s aus einer Satzmenge S führen, nennen wir hier: ***Ableitung von s aus S (S∆s)***. Wenn *s* keine Variablen beinhaltet, wird er ***Faktum*** genannt. Wenn S∆s leer ist, wird er ***Axiom*** genannt.

***Definition 3:*** Eine Ontologie ***Ont***=***(B,R)*** ist ein Tupel, in dem ***B*** Menge aller Begriffe einer Anwendung und ***R*** Menge aller beabsichtigten n-ären Beziehungen zwischen diesen Begriffen ist. Alternativ kann man statt ***R*** die Menge aller kategorischen Sätze ***S*** der Form: ***<Begriffi>ist<Eigenschaftj>von<Relationk>*** benutzen. Dies ist möglich, weil: ***∀rR, ∀biR: r(b1,b2,...bn) gdw {***



***s1=b1\_ist\_Eigenschaft1\_von\_r,***

***s2=b2\_ist\_Eigenschaft2\_von\_r,***

....

***Si=bi\_ist\_Eigenschaftj\_von\_r.***

***}***

Satz *si*wird ***Beschreibungssatz der Beziehung r***genannt. Beschreibungssätze können nur Fakten sein.

***Definition 4:*** Eine Ontologie ***Ont***=***(B,S)***, S Menge aller Beschreibungssätze der Beziehungen zwischen Begriffen in ***B***, kann auch in Form eines gerichteten Graphen mit markierten Knoten dargestellt werden: Ein gerichteter Graph oder Digraph mit Knotenmarkierungen G=(V,E,M) besteht aus:

* Einer Menge V von Knoten
* Einer Menge geordneter Knotenpaare E ⊆V x V von Kanten
* Einer Menge M von Markierungen der Kanten

Eine Ontologie wird ***konsistent*** genannt, wenn:

* der äquivalente gerichtete Graph G azyklisch ist
* ***∀s1,s2S***, *s*1,*s*2: Axiom: *s*1≠¬*s*2



***Definition 5:*** Eine konsistente Ontologie ***Ont***=***(B,S)*** wird vollständig genannt, wenn ***∀sS, s folgt logisch aus den Axiomen***: S∆s. ***Ont*** wird ***geschlossen*** genannt, wenn die Menge ***B*** eine festgesetzte, konstante Kardinalität besitzt. Dieser einfache Begriff der Vollständigkeit ist möglich, da die Menge ***Syl*** bekannter Weise logisch vollständige Teilmengen von Ableitungsmethoden beinhaltet (vgl. 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). Die Korrektheit der Menge ***Syl*** ist auch bekannt und wird hier angenommen.



***Definition 6:*** In der konsistenten Ontologie ***Ont***=***(B,S)*** werden Constraints der Form:

***(s1 & s2 & s3 .... sn) > c***,

wo *s1,s2*,...*sn*,c*S,* das Zeichen&: „and“ und „>“: Materielle Implikationbedeuten*,* ***kategorische Constraints*** genannt. Die Prämissen: *s1,s2*,...*sn* werden ***kategorische Konjunktionen*** genannt. Begriffe, die in kategorischen Konjunktionen benutzt werden, nennen wir: ***Entscheidungsbegriffe***. Der Begriff ***c*** wird: ***Konklusionsbegriff*** genannt.



***Definition 7:*** Eine **Grammatik** ist ein 4-Tupel

*G =* (*VN, VT, P, S*) wobei gilt:

1. VN ist eine endliche, nichtleere Menge, die Menge der nichtterminalen Zeichen,
2. VT ist eine endliche, nichtleere Menge, die Menge der terminalen Zeichen,
3. P ist eine endliche Teilmenge von V\* x V\*, die Menge der Produktionen oder Regeln,
4. S∈VN ist das Startzeichen.

Eine Umlaute-Grammatik (engl.: Diacritics-Grammer, DiaG) ist eine Grammatik, die terminale Zeichen mit Umlauten aus VT zulässt. Terminal- und Nicht-Terminalzeichen und Produktionen die Umlaute zulassen nennen wir: Dia-Terminal/NichtTerminal und DiaProduktion.

Beispiel einer Grammatik für umgangssprachliche Sätze der deutschen Sprache:

G = (VN,VT,P,S)

VN = {Nominalphrase,Verbalphrase,Eigenname,Artikel,Substantiv,Verb}

VT = {Susanne,Katze,Pferd,Heu,Buch,der,die,das,jagt,frisst,liest}

P = {(Satz -> Nominalphrase Verbalphrase .),

(Nominalphrase -> Eigenname | Artikel Substantiv),

(Verbalphrase -> Verb | Verb Nominalphrase),

(Eigenname -> Susanne),

(Substantiv -> Katze | Pferd | Heu | Buch),

(Artikel -> der | die | das),

(Verb -> jagt | frisst | liest)}.

S = Satz

Beispiel einer Dia-Grammatik für die arabische Sprache:

G = (VN,VT,P,S)

VN = {Nominalphrase,Verbalphrase,Eigenname,Substantiv,Verb}

VT = {ولدٌ,ولداً,ولدٍ,رأيُ,رأيٍٍ,رأيَ,خيرُ,خيرَ,خيرِ,…

[alle Wörter/Wortmuster im arabischen Lexikon]}

P = {(Satz -> Nominalphrase Verbalphrase .),

(Nominalphrase -> Aussage “ُ”Nomen),

(Verbalphrase -> Verb |“ُ”Nomen Verb),

(Nomen -> Bestimmt | Unbestimmt),

(Bestimmt -> AlNomenMuster | Genitiv | Eigenname),

(Eigenname -> {… alle bekannten Namen …})

(Genitiv-> “ُ”Bestimmt “ُ”Unbestimmt | “ُ”Unbestimmt “ُ”Unbestimmt)

(AlNomenMuster -> NomenMuster“ال”)

(NomenMuster -> {… alle NomenMuster im Lexikon …})

(Verb -> {… alle VerbMuster im Lexikon …})

(Aussage -> Bestimmt | Unbestimmt | Verbalphrase )}.

S = Satz

1. **CDK** Die Constraints-Definitionskomponente besteht aus einem einfachen Text Editor, in dem man CNF-Formelmengen definiert.
2. **OBK** Die Ontologie-Beschreibungskomponente kann ähnlich CDK aus einem Text Editor in dem man kategorische Beschreibungssätze und/oder Constraints definieren kann bestehen, oder aus einem Graph Editor in dem man Begriffe und ihre Beziehungen in Form von Knoten und Kanten ausdrücken kann.
3. **EBK** Die Entscheidungsbaumkomponente ist die zentrale Einheit, in der man in CNF ausgedrückte boolesche Funktionen *f* auswertet. Resultat ist ein Entscheidungsbaum (BDD), der der Wahrheitstabelle von *f* äquivalent ist. Verfahren 1 ist das zentrale Verfahren in dieser Komponente und benutzt die in DE102015013593A1 beschriebene Mustereigenschaft der logischen Variablen in folgender Weise:

**Verfahren 1:**

**Eingabe**: CNF Klauselmenge S

**Ausgabe**: BDD

**Schritte**:

1. Wende Verfahren 2 an, um die Formelmenge S in einer äquivalenten S’ umzubenennen. Diese Umbenennung berücksichtigt Musterlängen in der in Verfahren 2 beschriebenen Form.
2. Wähle Literal X der ersten Klausel C∈ S’ für die Instanzierung aus. X ist das Literal mit dem kleinsten Index.
3. Wende die Auswertungen: {X=TRUE} dann {X=FALSE} auf S’ an. Dieses Anwenden ergibt linke- und rechte-Klauselmengen S1, S2.
4. Wenn entweder S1 oder S2 TRUE oder FALSE werden, gib TRUE/FALSE-Knoten aus für die jeweilige Klauselmenge.
5. Wenn weder S1 noch S2 in einem Behälter LCS (Liste der schon bearbeiteten Klauselmengen) gefunden werden: Initiiere einen rekursiven Aufruf zuerst mit S1, dann mit S2. Das ergibt ein linkes (linksRes) und ein rechtes (rechtsRes) Resultat. Wenn nicht: Initiiere einen rekursiven Aufruf mit nur der Klauselmenge, die in LCS nicht gefunden wurde. Gib das gefundene BDD für die andere zurück.
6. Das Endresultat der Resolution von Klauselmenge S’ ist ein Knoten mit linkem Kind leftRes und rechtem Kind rightRes.

**Verfahren 2:**

**Eingabe**: CNF Klauselmenge S

**Ausgabe**: CNF Klauselmenge S’, die folgende Bedingungen erfüllt:

1. **∀*l1,l2,l3,...,lnC aus S’: li erscheint vor lj in C, wenn i<j,*** d.h. Indizes der Literale sind in aufsteigender Ordnung innerhalb der Klauseln von S’ sortiert.



1. **S’ ist nach *l1,l2,l3,...,ln* in aufsteigender Ordnung** unter Rücksichtnahme der Negation sortiert.

***Formal: ∀i,j: Wenn i<j, dann erscheint LiC vor LjD in S’, wo Li Kopfliteral (d.h. erster Literal) von C und Lj Kopfliteral von D ist***



1. **∀*x***∈**LIT(S’), LIT(S’) ist die Menge aller Literale in S’, ∀C**∈**S’:  
   Wenn *x*** ∉ **LEFT(*x*,C) dann**

**∀*y***∈ **LEFT(*x*,C): *x*>*y***.

LEFT(*x*,C) ist eine Funktion, die alle Variablen-Indizes zurückgibt, die vor der Variablen *x* aus Klausel Cin der String-Darstellung der Formelmenge S’ existieren.

(anders ausgedrückt schreibt diese Bedingung vor, dass alle neuen Indizes, die in einer Klausel zum ersten Mal vorkommen, grösser als alle schon benutzten in S’ sein müssen).

1. **S’ ist eine Menge, d.h. Klauseln kommen in ihr nur einmal vor.**

**Schritte**:

* 1. jetzigeMenge=Verfahren3(S)
  2. while [jetzigeMenge nicht wie in Bedingung b) sortiert ist]
     1. sortiere jetzigeMenge nach Bedingung b)
     2. jetzigeMenge=Verfahren3(jetzigeMenge)
  3. S’=jetzigeMenge
  4. Gib S’ zurück

**Verfahren 3:**

**Eingabe**: CNF Klauselmenge S

**Ausgabe**: CNF Klauselmenge S’

**Schritte**:

1. Nummeriere Klauseln in S in steigender Ordnung durch (fange mit 0 an).
2. Stell eine Tabelle auf, dessen Reihen Literalen-Indizien in S und dessen Spalten Klauseln darstellen.
3. Für jede Klausel C*i*:
   1. Sortiere Literale in C*i* in steigender Ordnung so, dass diejenigen, die noch nicht umbenannt wurden und in einer größeren Anzahl von Klauseln vorkommen zuerst erscheinen.
   2. Für alle Literale in C*i*:

Erstelle eine neue Reihe und schreib TRUE oder FALSE Werte auf, je nachdem, ob das Literal in der Spaltenklausel vorkommt oder nicht.

4. Nenn alle Literalen-Indizien in steigender Ordnung in der Tabelle um. Fang mit 0 an.

5. Konstruiere alle Klauseln von S unter Benutzung der neuen Namen/Indizien. Die resultierende Klauselmenge ist S’.

6. Gib S’ aus.

Beispiel: Wenn S = {{0,5} {0,2} {1,3} {1,4} {2,3}}, dann sieht die in Punkt 2 genannte Tabelle wie folgt aus:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | C0 | C1 | C2 | C3 | C4 |
| 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 |

Nach Punkt 4:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | C0 | C1 | C2 | C3 | C4 |
| **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 |

Die neue Klauselmenge: S’ = {{0,1}{0,2}{3,4}{3,5}{2,4}}. Diese Menge erfüllt nicht alle Bedingungen in Verfahren 2 und benötigt eine neue Ordnungs- und Umbenennungsschleife. In dieser neuen Schleife wird die Klauselmenge:

S’’ = {{0,1}{0,2}{2,4}{3,4}{3,5}} zu S’’’ = {{0,1}{0,2}{2,3}{3,4}{4,5}}

umformiert.

Figur 2 zeigt eine Beispielausführung von Verfahren 1 auf die CNF Klauselmenge: S={{0,1}{0,2}{1,3}{2,3}{3,4}}.

Das obige Verfahren 1 baut zwar das BDD für S auf, kann aber keine Information über die Anzahl der möglichen Lösungen vermitteln. Das folgende Verfahren füllt diese Lücke aus.

**Verfahren 4:**

**Eingabe**: BDD für CNF Klauselmenge S

**Ausgabe**: Anzahl der Lösungen

**Schritte**:

1. nummeriertesBDD = Nummeriere Knoten und Kanten im BDD beginnend mit 0 (**Verfahren 5** weiter unten)
2. Setze AnzahlLösungen für Knoten *n*0 = 0, AnzahlLösungen für alle Kanten der ersten BDD-Ebene =1
3. Für alle Ebenen *i* in nummeriertesBDD
   1. Für alle Kanten *eij*, j ist Index der Kante in Ebene *i*:
      1. Setze AnzahlLösungen für *eij*=AnzahlLösungen des Vaterknotens
   2. Für alle Knoten *nik*, *k* ist Index des Knotens in Ebene *i*:
      1. Wenn *nik* TRUE-Knoten (Blume) ist:

AnzahlLösungen von ***nik*=(∑e*x*\*2*i*-L*e*)\*2N-*i***,

*x* ist Index einer Kante, die zu *nik* führt, ***ex*** AnzahlLösungen einer solchen Kante, L*e* Kantenebene von *x* (gegeben mit: L*e*= LS*r*+1, S*r* Vaterknoten von *e*), N Anzahl der Variablen in S

* + 1. sonst: AnzahlLösungen von ***nik*=∑e*x*\*2*i*-L*e***

1. Gib AnzahlLösungen=∑ Tnd, Tnd ist TRUE-Knoten (Blume) aus

**Verfahren 5:**

**Eingabe**: BDD für CNF Klauselmenge S

**Ausgabe**: BDD mit nummerierten Knoten, Kanten und Ebenen

**Schritte**:

1. Durchlaufe das BDD in einer rekursiven, Tiefe-zuerst Weise. Nummeriere Knoten und Kanten und erstelle eine lineare, topologische Ordnung. Dabei ist eine topologische Ordnung im Grunde eine Ungleichung, die in der folgenden Weise linear erstellt werden kann: Für jede zwei Knoten *n*1, *n*2, Kinder von *n*: Setze die Ungleichung *n*<*n*1<*n*2 und ergänze sie, rekursiv in einer Tiefe-zuerst Weise, bis zur endgültigen Ungleichung. Die Ungleichung wird dadurch ergänzt, in dem man Kinder des Knotens *n*1 stets vor Kinder des Knoten *n*2 rekursiv aufstellt.
2. Für alle *u* ∈ V (V Knotenmenge des BDDs):

dist(*u*) = ∞

dist(*s*) = 0, *s* ist Wurzel

1. Für alle *u* ∈ V , in der linearisierten Ordnung:

dist(*u*)= **Dist(*u*)**

L*u*=|dist(*u*)|

**Dist:**

**Eingabe**: *u* ∈ V, BDD = (V,E), V Knotenmenge, E Kantenmenge

**Ausgabe**: Integerzahl die die Distanz zwischen *u* und der Wurzel darstellt

**Bemerkung**: *l*(*u*,*vi*,) ist die Länge der Kante von *u* nach *vi* (immer: ‘-1’).

**Schritte**:

für alle Kanten *v1,v2*,..*vn* ∈V sodass: (*u,vi*) ∈E:

**Dist** (*u*) =

min{

[**Dist** (*v*1) + l(*u*, *v*1)], ….

[**Dist** (*vn*) + l(*u, vn*)]

}

Fig. 3 zeigt eine Ausführung von Verfahren 5 auf das für die Klauselmenge S={{0,1}{0,2}{0,4}} mittels Verfahren 1 erstellte BDD. Die folgende Beispielsequenz von Operationen zeigt die Anwendung von Verfahren 4 auf S:

1. Ebene-0: *n*0=0
2. Ebene-1: *e*0=1,*e*5=1, *n*5=*e*5\*2*i*-L*e*5=1\*21-1=1
3. Ebene-2: *e*6=*n*5=1, *e*9=*n*5=1, *n*6= *e*6\*2*i*-L*e*6=1\*22-2=1
4. Ebene-3: *e*7=*n*6=1, e8=n6=1, *n*1=*e*0\*2*i*-L*e*0+*e*7\*2*i*-L*e*7=1\*23-1+1\*23-3=5
5. Ebene-4: *e*1=*n*1=5, *e*2=*n*1=5, *n*2=(*e*1\*2*i*-L*e*1)\*2N-*i*= (5\*24-4)\*25-4=10, n3=e2\*24-4=5,
6. Ebene-5: *e*3=*n*3=5, *e*4=*n*3=5, *n*4=(*e*3\*2*i*-L*e*3)\*2N-*i*=(5\*25-5) \* 25-5=5

AnzahlLösungen=*n*4+*n*2=**15**

1. **DAK** Das zentrale Verfahren in dieser Komponente wendet Syllogismen der Menge ***Syl*** so lange an, bis keine neuen Sätze ableitbar sind.

**Verfahren 6:**

**Eingabe**: Kategorische Satzmenge S

**Ausgabe**: Kategorische Satzmenge S’

**Schritte**:

1. NeuerSatz=TRUE, S’=S
2. While (NeuerSatz=TRUE)

Für alle Syllogismen *sy* der Menge ***Syl***:

* + 1. Wende *sy* auf S an
    2. Wenn ein neuer Satz s entstanden ist:

Setze NeuerSatz=TRUE, S’ = S’∪ *s*

ansonsten NeuerSatz=False

1. Gib S’ zurück

DAK enthält eine Übersetzungskomponente (UEK), dessen Aufgabe darin besteht, ausgewählte Datensätze in kategorischen Aussagen umzuwandeln. Dies geschieht mittels folgendem Verfahrens:

**Verfahren 7:**

**Eingabe**: Menge D von ausgewählten Datensätzen

**Ausgabe**: Kategorische Satzmenge S

**Schritte**:

1. Für alle Datensätze ***r(b1,b2,...bn)*** D:



* 1. Wende Definition 3 an:

***∀rR, ∀biR: gdw {***



***s1=b1\_ist\_Eigenschaft1\_von\_r,***

***s2=b2\_ist\_Eigenschaft2\_von\_r,***

....

***si=bi\_ist\_Eigenschaftj\_von\_r.***

***}***

* 1. Setze ***∀i:*** S=S∪ *si*

1. Gib S zurück
2. **SEK** Die Spracherkennungskomponente (SEK) hat die Aufgabe, Sätze in natürlicher Sprache in kategorischen Sätzen umzuwandeln. Die umgekehrte Richtung ist trivial. Um dies erfindungsgemäß zu gewährleisten, werden nur Substantiv-Sätze in Betracht gezogen.

Verschiedene Sprachen haben diesbezüglich unterschiedliche Verfahrensweisen, die aber alle darauf basieren, auf Wortebene in der Lage zu sein, Verb, Nomen und deren Verbindungen zu unterscheiden. In lateinischen Sprachen wird diese Unterscheidung dadurch erreicht, dass versuchsweise mit Hilfe eines Lexikons jedes Wort eines Satzes erst als Verb und dann als Substantiv betrachtet wird. In lateinischen Sprachen herrscht im allgemeinen Fall auf der Wortebene (zumindest zwischen Verb und Nomen) Ambiguität. In semitischen Sprachen und speziell in der arabischen Sprache werden Umlaute (Engl.: Diacritics) für genau diese Aufgabe benutzt. Unterschiede zwischen Verb, Nomen und anderen Wortarten sind deswegen schon auf syntaktischer Ebene erkennbar. Dies ist die Grundidee des folgenden Verfahrens, das speziell für die arabische Sprache erfunden ist. Seine allgemeine Definition lässt auch andere Sprachen zu, die ähnlich zur arabischen Sprache, syntaktische Gebilde beinhalten, die semantische Merkmale widerspiegeln.

**Verfahren 8:**

**Eingabe**: natursprachlicher Satz S, Dia-Grammatik *G*, Substantiv-Kategoriesatz Zuweisungsliste *z*, Lexikon L

**Ausgabe**: Kategorischer Satz S’

**Schritte**:

1. ResultatsStruktur={}
2. Für alle Wörter wS:



* 1. Suche *w* in L
  2. Wenn *w* gefunden:

Füge Verb/Nomen/bestimmt/unbestimmt-Tags der

ResultatsStruktur an, ansonsten

Abbruch

1. Benutze G, um eine korrekte Ableitung von S zu finden
2. Wenn Ableitung gefunden:
3. Suche ResultatsStruktur in *z*
4. Wenn ResultatsStruktur gefunden:

Setze S’=kategorischer Satz, ansonsten

Abbruch

1. Gib S’ aus

Das folgende Beispiel erklärt die Ausführung dieses Verfahrens für die arabische Sprache:

Gegeben sei der Satz S = „رأيُ الولدِ خيرُ رأيٍ“

(Deutsch: „*Die Meinung des Jungen ist die beste Meinung.*“)

Sei G die Dia-Grammatik aus Definition 7.

Nach Schritt 2 (immer von rechts nach links zu lesen) ist

ResultatsStruktur=“Nomen/ٍunbestimmt Nomen/ُunbestimmt

Nomen/ِbestimmt Nomen/ُunbestimmt“.

Die Ableitung in Fig. 4-a ist dann eine korrekte Ableitung von S aus G. Fig. 4-b zeigt hingegen eine fehlgeschlagene Ableitung, wenn man die Umlaute unberücksichtigt lässt.

Zuweisungsliste *z*enthält unter anderem folgende Datensätze (von rechts nach links zu lesen):

|  |  |
| --- | --- |
| **Nomen Satz** | **Kategoriesatz** |
| Nomen S1 (bestimmt), Nomen (unbestimmt) S2 | (S2 ist S1) |
| Nomen S1 (unbestimmt), Nomen (bestimmt) S2, Nomen (unbestimmt) S3 | (S3 + S2 ist S1) |
| Nomen S1 (unbestimmt), Nomen (unbestimmt) S2, Nomen (bestimmt) S3,Nomen (unbestimmt) S4 | (S4 + S3 ist S2+S1) |

Verfahren 8 gibt für das obige Beispiel S’=[ (S4 + S3 ist S2+S1)]als Kategoriesatz aus.

1. **IAK** Diese Erfindung nimmt an, dass die für die Anwendung relevante Ontologie geschlossen ist. Dies hat zur Folge, dass man vollständige Induktion auf Teile dieser Ontologie leicht anwenden kann, da die Kombinatorik stets konstant bleibt. Zielsetzung ist, dem Benutzer zu ermöglichen unbekannte Constraints zu entdecken und dadurch die Schlüssigkeit der Logik des Gesamtsystems zu steigern. Das folgende Basisverfahren dieser Komponente ist so definiert, dass es auch die Anwendung auf Datensätze der Datenbank erlaubt.

**Verfahren 9:**

**Eingabe**: Menge M der ausgewählten Entscheidungsbegriffe, W Menge der Werte dieser Begriffe, Menge S der ausgewählten Konklusionsbegriffe, W’ Menge ihrer Werte

**Ausgabe**: Menge M’ der kategorischen Constraints

**Schritte**:

* 1. Errichte eine Kombinatorik Tabelle T für M und ihre Werte W
  2. Für alle *s* S, *w*1,*w*2,...*wn*, *wi* W’ ist Wert von *s*:



* + 1. Für jeden Kombinatorik Satz in T

Setze ein geeignetes *wi* (automatisch, d.h. per rekursive Funktion, oder manuell)

1. Für alle Teilmengen T’ vom M mit jeweiligen Werten *t*1,...*tn* W:



* + 1. Verifiziere, ob es in T ein *s*S, *wi*ist Wert von *s*, gibt, sodass:



Jede mehrmalige Erscheinung der Werte *t*1,...*tn*in T in der Spalte *s* den Wert *wi* beinhaltet

* + 1. Wenn ja: Setze neuerConstraint = (*t*1&...&*tn*>*wi*)

1. Gib alle somit gefundenen Constraints aus

Das folgende Beispiel erläutert die Benutzung des obigen Verfahrens im Kontext der Druckmaschinenanwendung.

Sei M={Druckgruppe, Typ, Nummeriereinheit}, W={{1-farbig, mehrfarbig}, {Heidelberg, Roland},{vorhanden, nicht-vorhanden}}. T (Schritte 1. und 2.) sieht wie folgt aus:

|  |  |  |  |
| --- | --- | --- | --- |
| Druckgruppe | Typ | Nummeriereinheit | Konklusion  *s*=Ersatzteile auf Rabatt |
| 1-farbig | Heidelberg | Vorhanden | 1 |
| 1-farbig | Heidelberg | nicht-vorhanden | 0 |
| 1-farbig | Roland | Vorhanden | 0 |
| 1-farbig | Roland | nicht-vorhanden | 0 |
| mehrfarbig | Heidelberg | Vorhanden | 1 |
| mehrfarbig | Heidelberg | nicht-vorhanden | 1 |
| mehrfarbig | Roland | Vorhanden | 1 |
| mehrfarbig | Roland | nicht-vorhanden | 1 |

In Schritt 3. werden folgende Teilmengen von M gebildet:

(Druckgruppe)

(Typ)

(Nummeriereinheit)

(Druckgruppe, Typ)

(Druckgruppe, Nummeriereinheit)

(Typ, Nummeriereinheit)

(Druckgruppe, Typ, Nummeriereinheit)

***Für (Druckgruppe) findet man die Constraints:*** ((Mehrfarbig)>1)

***Für (Typ) findet man keine Constraints***

***Für (Nummeriereinheit) findet man keine Constraints***

***Für (Druckgruppe, Typ) findet man die Constraints:*** ((1-farbig & Roland)>0), ((Mehrfarbig & Heidelberg)>1), ((Mehrfarbig & Roland)>1)

***Für (Druckgruppe, Nummeriereinheit) findet man die Constraints:*** ((1-farbig & nicht vorhanden)>0), ((Mehrfarbig & nicht vorhanden)>1), ((Mehrfarbig & vorhanden)>1)

***Für (Typ, Nummeriereinheit) findet man die Constraints:*** ((Heidelberg & vorhanden)>1)

***Für (Druckgruppe, Typ, Nummeriereinheit) findet man keine Constraints***

Diese gefundenen Constraints spiegeln folgende Regeln der Druckmaschinenindustrie wider:

1. Man bekommt Rabatt auf Ersatzteile von mehrfarbigen Heidelberg- und/oder Roland-Druckmaschinen
2. Man bekommt keinen Rabatt für Ersatzteile von Roland 1-farbigen Druckmaschinen
3. Wenn keine Nummeriereinheit mit einer 1-farbigen Druckmaschine, ob Roland oder Heidelberg, geliefert wurde, dann gibt es keinen Rabatt für Ersatzteile dieser Maschine.
4. Für mehrfarbige Maschinen bekommt man Rabatt, unabhängig davon, ob Nummeriereinheiten geliefert wurden oder nicht
5. Ersatzteile einer Heidelberg Maschine, dessen Nummeriereinheit geliefert wurde, stehen immer unter Rabatt
6. **RAK** Die rationale Antwortkomponente hat die Aufgabe, Anfragen durch Logik-unterstützte Reaktionen zu beantworten. Dies geschieht in der Annahme, dass es eine Liste aller kategorischen Constraints gibt, die entweder explizit in OBK definiert oder per Verfahren 9 in der IAK abgeleitet wurde.

**Verfahren 10:**

**Eingabe**: SQL Anfrage ***Anf***, Liste der kategorischen Constraints ***katCons***

**Ausgabe**: Menge M aller kategorischen Constraints, die zur Anfrage gehören

**Schritte**:

1. Führe ***Anf*** aus. Nenne die entstandene Tabelle ***AusT***.
2. Bilde die Menge M’ aller Entscheidungsbegriffe, die in ***AusT*** vorkommen
3. Für jeden Begriff *b* aus M’:
   1. Suche *b* in ***katCons***

Wenn gefunden: Füge Constraint zur Resultatliste

1. Gib Resultatliste aus

Alternativ zur SQL-Anfragen kann man im RS kategorische Sätze direkt suchen. Da Verfahren 6 in der DAK mittels Vollständigkeit und Geschlossenheit der Ontologie garantiert, dass jeder ableitbare Satz in der Extension der Ontologie auch existiert, genügt für diese Art von Abfrageprozedur ein einfaches Suchverfahren

**Verfahren 11:**

**Eingabe**: Kategorischer Satz ***s***, Liste aller kategorischen Sätze ***S***, Liste aller kategorischen Constraints ***const***

**Ausgabe**: Liste aller kategorischen Sätze/Constraints, die in der Ableitung von *s* mitgewirkt haben

**Schritte**:

1. Suche ***s*** in ***S***
2. Suche ***s*** in ***const***
3. Wenn gefunden: Gib ***S*** ∪ ***const*** aus, ansonsten

Abbruch