(19) Deutsches Patent- und Markenamt

(10) **DE 10 2015 013 593 A1** 2017.04.20

(12) # Offenlegungsschrift

(21) Aktenzeichen: **10 2015 013 593.7**
(22) Anmeldetag: **15.10.2015**
(43) Offenlegungstag: **20.04.2017**

(51) Int Cl.: ***G06F 17/30*** (2006.01)

(71) Anmelder:
makmad.org e.V., 30159 Hannover, DE

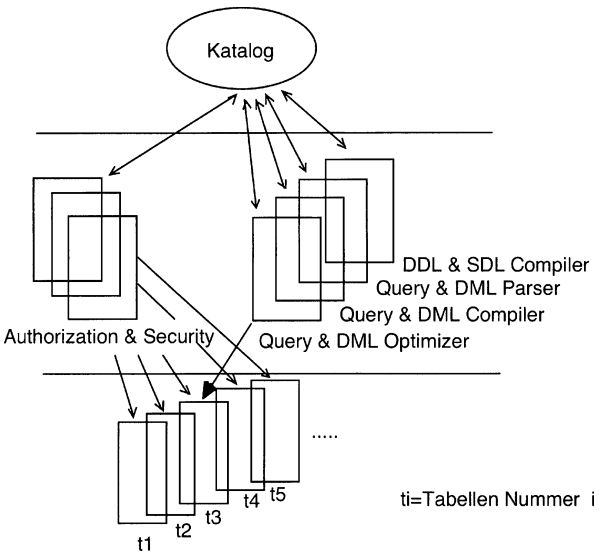
(72) Erfinder:
Abdelwahab, Naser, Dr., Kairo, EG

Prüfungsantrag gemäß § 44 PatG ist gestellt.

Die folgenden Angaben sind den vom Anmelder eingereichten Unterlagen entnommen

(54) Bezeichnung: **Effizientes Verfahren zur logischen Vervollständigung eines deduktiven Kataloges zur allgemeinen Constraintsbehandlung im erweiterten relationalen Datenbankkonzept**

(57) Zusammenfassung: Bei relationalen Datenbankkonzepten leiden Abfrageprozeduren entweder an logisch unvollständigen Abfrageergebnissen, oder mangeln an Ineffizienzen der Abfragezeiten. Die vorliegende Erfindung gibt effiziente Verfahren an, die relationale Datenbanksysteme in ihrem allgemeinsten und vollständigsten Konzept in ihren Abfrageverfahren so optimiert, dass die Antwortsprozedur eine lineare Effizienzsteigerung in Bezug auf die Geschwindigkeit erfährt, ohne dass strenge logische Bedingungen aufgegeben werden müssten.
Um die Abfrageverfahren zu optimieren, führt eine neue Darstellung von Variablen als Teile der klassischen Wahrheitstabelle zu vollständigen Auswertungsmethoden, die eine kompilierte, effiziente Version der logischen Ausdrücke generiert.
Das neue Verfahren eignet sich zum Einsatz in relationalen Datenbankanwendungen, bei welchen sowohl effiziente Abfragezeiten, als auch logische Vollständigkeit und Konsistenz durch allgemeine Constraintsbehandlung gefordert werden.
Es ist die Aufgabe dieser Erfindung, durch die Darstellung eines beliebigen Datenbankabfragekontextes mittels Formeln der Logik und von auf der effizienten Extension der logischen Theorie basierenden Verarbeitungsverfahren, Abfragen derart zu beschleunigen, dass Eingabe/Ausgabe-Operationen linear in der Länge der Eingabezeichenketten bleiben, unabhängig von der Komplexität der logischen Theorie. Die erfindungsgemäße Lösung dieser Aufgabe ist in den Patentansprüchen 1–13 angegeben. Erfindungsgemäß wird damit eine neues Verarbeitungsverfahren von Formeln als Basis für die Beschleunigung dargelegt. Um einen spezifischen Wahrheitswert zu finden führt dieses Verarbeitungsverfahren die effiziente Extension der logischen Theorie (materialisiert in logischen Musterbäumen vs. ihrer Intension materialisiert in Klauseln) ein.



Beschreibung

Anwendungsgebiet der Erfindung

[0001] Die Erfindung betrifft ein effizientes Verfahren zur logischen Vervollständigung eines deduktiven Kataloges zur allgemeinen Constraintsbehandlung im erweiterten relationalen Datenbankkonzept. Das Verfahren führt in Abhängigkeit einer neuen Darstellung von Variablen als Teile der klassischen Wahrheitstabelle zu vollständigen.

[0002] Auswertungsmethoden, die eine kompilierte, effiziente Version der logischen Ausdrücke generieren. Diese Neuerung erlaubt im Unterschied zu bekannten Verfahren in deduktiven Datenbanken, lineare Verarbeitungszeiten von Eingaben, die keine klassischen AND-, OR- oder NOT-Operatoren heranziehen. Damit führt die Erfindung in Abhängigkeit dieses neuen logischen Vervollständigungsverfahrens zu Abfrageprozeduren im Rahmen deduktiver Datenbanken, welche nicht mehr von der Komplexität der logischen Form abhängen, sondern von der im Voraus zur Verfügung gestellten, optimalen Extension der logischen Theorie. Sie dient der Verbesserung der Abfrageprozedur im allgemeinen, und der Effizienzsteigerung im speziellen, da die Antwortsprozeduren einer relationalen Datenbankabfrage, insbesondere bei feldübergreifenden Constraints, eine maximale Geschwindigkeitszunahme erfahren.

Charakteristik des bekannten Standes der Technik

[0003] Bekannt sind allgemeine Verfahren zur Behandlung der Constrainstprogrammierung im Rahmen kontinuierlicher oder diskreter Variablen zur Modellierung mathematischer oder algorithmischer Probleme (vgl. DE4411514A1 und US5216593).

[0004] 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 Constraintsmethode 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 Constraintsanwendung 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.

[0005] 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).

[0006] 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 hier sehr gross, d. i. exponentiell in der Länge der benutzten logischen Formeln sein kann.

[0007] Nicht bekannt sind Verfahren, bei welchen allgemeine Constraintsbehandlungen im erweiterten relationalen Datenbankkonzepten derartig durchgeführt werden, daß logische Vervollständigkeitsverfahren auf Katalogebene effizient (d. h. nicht exponentiell) angewandt werden, um damit maximale Ausführgeschwindigkeit von logischen Abfragen zu gestatten:

Das relationale Datenmodell

[0008] (vgl. Date: An introduction to database systems, Vol. I, Fourth Edition, Adison-Wesley, Massachusetts, 1986 und für den gesamten Überblick: DE19725965C2).

[0009] Das relationale Datenbank-Modell entstammt einer vereinfachten Form der Prädikatenlogik erster Ordnung. Die Begriffe: Beziehung (Relation), Domain, Attribut und Interpretation haben die dort üblichen Bedeutungen.

[0010] Die einzigen Objekte dieses Systems sind Tupel. Ein Tupel ist eine Sammlung von Attributen, die intensional-logischen Zwecke dienen.

[0011] Die Tupel werden in Tabellen zusammengefaßt. Die Tabellen beteiligen sich durch Tupel in verschiedenen Datenbankoperationen, welche Schemakonsistenz garantieren. Bezogen auf ihre Rolle in der Schemakonsistenz, werden die Tabellen in Grund- bzw. auxiliären Tabellen unterteilt.

[0012] Das RDS ist ein logisches Bild einer mit anderen Mitteln zu gewinnenden semantischen Darstellung der zu modellierenden Miniwelt. Von diesen Mitteln ist das ER-Diagramm am bekanntesten. Es liefert ein Schema, das die Basiskonsistenzbedingungen der Miniwelt enthält. Darunter versteht man die wichtigsten in der Miniwelt existierenden Beziehungen zwischen Objekten, welche z. T. durch Kardinalitätsklassen zu behandeln sind.

[0013] Schlüsselattribute haben eine zentrale Bedeutung als Garant der Schemakonsistenz. Sie gewähren die referentielle Integrität und bewerkstelligen damit, daß die Datenbankrelationen Normalformen genügen. Mit referentieller Integrität ist das Vorhandensein von Attributwerten in Grundtabellen gemeint, wenn diese Werte in auxiliaren existieren und umgekehrt.

[0014] Die Schnittstellensprache SQL, mit der man an das RDS Anfragen stellen kann, ist eine Logik-basierte Sprache, die es zwar erlaubt die Abfrage deklarativ zu beschreiben, aber nicht in der Lage ist, die volle Funktionalität einer Programmiersprache wiederzugeben. Es fehlt die Rekursion.

[0015] Durch Rekursion modellierbare allgemein-logische Konsistenzbedingungen (wie die Kalkulation der transitiven Hüllen von Relationen z. B.) werden vom RDS nicht unterstützt und bedürfen der externen Programmierungsarbeit. Einige Ansätze dieser Programmierung sind prozedural und haben den Nachteil der Komplexität und der Unübersichtlichkeit, andere sind deklarativ und haben den Nachteil der Ineffizienz und der Beschränktheit auf bestimmten Modellierungssituationen.

[0016] Da das RD-Modell aus der Prädikatenlogik entstammt, verfügt es – in einer beschränkten Weise – über die in der Logik übliche Aufteilung eines Systems in Ebenen bzw.

[0017] Metaebenen. Das RD kennt davon nur zwei: Die untere enthält die Daten in Form von Tupeln. Die obere hingegen verfügt über alle für die Implementierung des Systems wichtigen Informationen über Beziehungen und andere abstrakte Datentypen. Die in der Literatur übliche Bezeichnung dieser letzteren Ebene lautet: Katalog.

[0018] Wenn ein Katalog mit Detailinformationen über andere Aspekte der Anwendung wie z. B. organisationsspezifische Merkmale, die mit der Realisierung der Operationen eines RDS nichts zu tun haben, verfügt, so spricht man von einem Data Dictionary.

[0019] Ein reales RDS enthält die in **Fig.** 1 aufgeführten Komponenten. Ti (j = 1 ... n) steht für Tabellennamen. Die Pfeile symbolisieren den Datentransfer.

[0020] Der Katalog wird u. a. benutzt, um die Spezifikationen eines Datenbankschemas abzuspeichern (DDL-compiler) und die Ausdrücke einer Abfrage gegebenenfalls mit diesen Daten zu vergleichen (DML-parser). Gleichzeitig wird beim Kompilieren (DML-Compiler) auf reale Dateiorganisationsinformationen mit dem Ziel zugegriffen, Maschinencodes zu generieren. Abfrageoptimierungen (DML-Optimizer) benutzen Feldbeschreibungen, um z. B. Indexe und Hashfunktionen bei dem Entwurf eines Ausführungsplanes zu berücksichtigen. Dazu werden auch heuristische Optimierungskriterien wie Selektivitätsabschätzungen (selectivity-estimates) benötigt und im Katalog abgelegt. Andere Funktionen eines Kataloges bestehen darin, Sicherheitsstufen zu definieren und die Übersetzung der Datenbankviews in das originale Schema zu garantieren.

Constraintsbehandlung

[0021] Abgesehen von der oben erwähnten Schlüsselintegrität, existiert in einem RDS die Möglichkeit, auch explizite Konsistenzkriterien in Form von Programmeinträgen im Katalog zu definieren. Dies hat gewöhnlich

die Form einer SQL-Klausel, die mit eingebauten Konsistenzverifikationskomponenten versehen ist. Beispielsweise erreicht die Befehlssequenz

```
create assertion salary
check (not exists (select * from angestellter A,
                                  angestellter B
                   where A.gehalt >  B.gehalt
                   &     A.rang <=   B.rang))
```

eine Beschränkung der Tupel der Datenbasis auf diejenigen, bei denen das Gehalt eines Angestellten niemals das seines ranghöheren übertrifft. Das Anbinden der Constraintsdefinition an SQL hat den Nachteil der Übernahme mangelnder Ausdruckskraft. Das oben erwähnte Fehlen der Rekursion macht es unmöglich, transitive Abhängigkeiten z. B. in einem Constraint zu berücksichtigen.

[0022] Im Beispiel wurde angenommen, daß die Ordnung der Tupel unter Berücksichtigung des Aspekts „Rang" bereits existiert. Ein explizit aufgeführtes Attribut (Rang) realisierte diese Ordnung. Im Falle des Fehlens einer solchen Ordnung, die streng sein muß, würde diese Art von Konsistenzüberwachung nicht möglich sein. Wenn eine Datenmenge quantitativ so groß ist, daß z. B. eine manuelle Eingabe der Werte aus pragmatischer Sicht unrealistisch wird, dann tritt ein solcher Fall auf. Externe Programmierungsmaßnahmen, welche die Tupelbeziehung(en) rekursiv oder iterativ bearbeiten, wären diesbezüglich die einzige Lösung.

[0023] Der Katalog enthält aber nicht nur die Beschreibung der Constraints, sondern auch ihre Realisierungsinformation. In der Praxis einer heutigen RD-Umgebung hat man drei mögliche Realisierungsstrategien: Transaktionsprozeduren, Zusicherungen und Trigger.

[0024] Mit Transaktion ist gemeint:
Man faßt eine Menge von Basisoperationen in Form einer Prozedur zusammen, und betrachtet diese als eine eigene Konsistenzeinheit. Damit garantiert das System die Unteilbarkeit dieser Einheit. Zusicherungen sind hingegen logische Prädikate, die auf Datenbasiszustände angewandt werden, um die Konsistenzbedingungen zu erfüllen. Schließlich sind Trigger automatisch ausgelöste Prozeduren, die als Folgeoperationen zusammen eine Konsistenzeinheit bilden können.

Ansätze zur Constraintsbehandlung

[0025] Wenn große Quantitäten von Daten in eine Datenbank importiert werden müssen, dann entsteht das Problem, die Constraintmenge möglichst effizient zu aktivieren.

[0026] Dabei ist vor allem das Erhalten der referentiellen Integrität das aufwendigste Vorhaben, da die Insert- bzw. Delete-Anweisungen diese Integrität verletzen könnten. SQL verfügt über die CASCADE-Funktion, mit dessen Hilfe alle „Waisentupel", d. h. alle mit Fremdschlüssel versehenen Tupel von auxiliären Tabellen, die keine in den Grundtabellen vorkommenden Schlüsselwerte besitzen, beseitigt werden können. Um lange zeitliche Verzögerungen zu vermeiden, muß die CASCADE-Funktion jedoch möglichst prägnant mit den im letzten Absatz erwähnten Konsistenzerhaltungsstrategien kombiniert werden. **Fig.** 2 zeigt den üblichen Befehlsfluß, wenn eine SQL-Anweisung bearbeitet wird.

[0027] Nach der syntaktischen und semantischen Bearbeitung in (1) und (2), wird eine Optimierung eingeführt. Die Grundidee ist, daß möglichst viele SQL-Befehlsvorkommen eliminiert (vgl. Query Graph Model und IBM Research-Report RJ-6367, IBM Almada, San Jose, CA., August 1988). Die Planoptimierung (4) ergänzt dieses Verfahren, indem reale Dateiorganisationsparameter benutzt werden, um die bestmögliche Realisierung zu garantieren. Im Falle der Existenz von Constraints wird durch den Constraintscompiler (6) ein Maschinencode generiert, der die Constraints darstellt. Das hier vorgestellte Verfahren gestaltet den in **Fig.** 2 dargestellten Befehlsfluß heuristisch neu.

[0028] Einige Ansätze zur Realisierung von effizienten Constraint-Checks (vgl. EP0726537A1) ersetzen den normalen SQL-Compiler, der im Falle eines INSERT-Commandos aktiviert wird, durch einen anderen. Dessen Funktion ist es, vor der INSERT-Maschinencodesequenz eine SELECT-Anweisung zu generieren. Diese entfernt alle Tupel aus der Datenbank, die die Constraints verletzen, bevor ein neuer Tupel angefügt werden kann.

Damit garantiert er die referentielle Integrität. Demgegenüber ersetzt das nachstehend neue Verfahren den Constraintscompiler durch einen unten vorgestellten Constraints-Abfrage-Generator. Dieser hat die Aufgabe, Constraintchecks in Form von prädikativen Anfragen an dem Katalog zu stellen.

Deduktive Datenbanken

[0029] Die Problematik der Constraintsbehandlung kann in ihrer notwendigen Allgemeinheit nur im Rahmen der logischen Programmierung untersucht werden. Dabei spielt es keine Rolle, ob die Constraints allgemeine oder referentielle Integrität modellieren, da die referentielle Integrität eine spezielle Art von Variablenbindungsproblemen darstellt, wie sie in der logischen Programmierung üblich ist.

[0030] Auf dem Gebiet von Datenbanken benutzt man deduktive Datenbanken immer dann, wenn Lösungsansätze angestrebt werden, die gleichzeitig deklarativ und mathematisch-logisch vollständig sein sollen. Die übliche deduktive Datenbanksprache ist DATALOG. Sie ist eine beschränkte prädikative Sprache ohne Funktionen und ohne Negation.

[0031] DATALOG entspricht den wesentlichen Bedürfnissen einer Datenbank, weil sie das logische Bild des RD-Modells darstellt. Im RDS werden – wie oben erwähnt – auf Tupelebene keine komplexen Datenstrukturen benötigt, welche mit Funktionen vergleichbar wären. Zudem wird die Semantik der Negation als Scheitern betrachtet, die gewünschten Tupel in der geschlossenen und endlichen Welt der Datenbank zu finden (engl.: Negation as Failure, Closed World Assumption).

[0032] Um den Rahmen der hier beschriebenen Erfindung streng einzugrenzen, wir im folgenden der DATALOG-Formalismus mathematisch-logisch ausgeführt.

[0033] Formale Definitionen sind mit dem Index Def. versehen, wogegen allgemein-sprachliche Vereinbarungen mit Ver. gekennzeichnet sind:
Ver. 1: Eine intensionale Datenbasis (IDB) ist die Menge aller Tupel aus einer Datenbank, die mittels Regeln der Logik aus einer vorgegebenen (extensionalen) Menge (EDB) ableitbar sind.
Ver. 2: Die Regeldefinitionssprache ist eine Sprache, mit welcher man die ableitbaren Beziehungen der IDB bestimmen kann.

[0034] Diese Sprache erlaubt es dem Programmierer, logische Verknüpfungen zwischen den verschiedenen Sachverhalten in der Datenbasis zu definieren. Im allgemeinen werden von ihr folgende Eigenschaften vorausgesetzt:

A. Rekursive Definitionen, d. h. Regeln können selbstbezogen sein.
B. Negation, d. h., daß die Möglichkeit ein nicht existierendes Faktum in der logischen Verknüpfung zu berücksichtigen gewährleistet ist.
C. Beliebige benutzerdefinierte Funktionen sind gestattet.

[0035] Sie ist folglich eine weitreichend reduzierte Version der Horn-Klausel-Sprache von PROLOG (vgl. Chang, C. L.; Lee, R. C.: Symbolic logic and mechanical theorem proving, Academic Press, 1977 Edition, 1973 und Sterling, Leon; Ehud, Shapiro, The Art of Prolog, MIT Press series in logic programming, 1986).
Def. 1: Literale sind negative oder positive logische Aussagen.
Def. 2: Horn-Klauseln sind Regeln der Form: $Q \leftarrow A1 \wedge A2 \ldots \wedge An$, wo Q und Ai nicht-negative Literale sind.
Def. 3: Eine Regel der Form $Q \leftarrow A1 \wedge A2 \ldots n\ An$ wird definit genannt, wenn Q und Ai Atome sind (für alle Ai).

[0036] Eine der bekanntesten Regeldefinitionssprachen ist DATALOG (vgl. Ullman, Jeffery D., Principles of database & knowledge-base systems, Volume I & II, Computer science press, 1988). DATALOG ist eine Sprache ohne Funktionssymbole, welche jedoch über verschiedene Extensionen (z. B. $DATALOG^{fun}$, $DATALOG^{neg}$) verfügt, die teilweise Funktionsdefinitionen erlauben. Die Syntax von DATALOG enthält folgende Elemente:

A.	Konstanten	a, b, c, d ...
B.	Variablen	x, y, z, ...
C.	Relationale Prädikate	R1, R2 ...
D.	Logische Verknüpfungen	v, ∧,
E.	Gleichheitsprädikate	=, <, > ...
F.	Punktuations- und Trennsymbole	„,”, ”.”

[0037] Formeln und Regeln werden rekursiv mit Symbolen aus A bis F definiert.
Def. 4: Ein Term ist entweder eine Konstante oder eine Variable.
Def. 5: Ein Atom ist ein Ausdruck der Form P(t1, t2, ... tn), wo P n-äres Prädikat ist und ti Terme. Die Menge aller Atome wird A geschrieben.
Def. 6: Ein Grundterm (Atom) ist ein Term (Atom) ohne Variablen.

[0038] In einer Regel Q ← A1 ∧ A2 ... ∧ An, wird Q Konklusion (Kopf) und A Prämisse genannt. Jedes Ai ist Teilziel und die gesamte Konjunktion ist ein Ziel (Rumpf).
Def. 7: Eine DATALOG-Regel enthält nur Atome. Sie wird rekursiv genannt, wenn die Konklusion auch als Prämisse in der Regel enthalten ist. Sie wird darüber hinaus linear rekursiv genannt, wenn die Konklusion nur einmal unter den Prämissen vorkommt.
Def. 8: Eine DATALOG-Regel der Form Q ←., wo Q Grundatom ist, wird Faktum genannt.

[0039] Beispiele von DATALOG-Regeln sind
Vorfahren(x, y) ← Eltern(x, z) ∧ Vorfahren(z, y). oder
Vorfahren(x, y) ← Vorfahren(x, z) ∧ Vorfahren(z, y). Die erste ist linear und die zweite nicht-linear rekursiv.
Def. 9: Ein DATALOG-Programm ist eine Menge von DATALOG-Regeln.

[0040] Die Semantik von DATALOG-Programmen kann entweder deklarativ (modell-basiert) oder prozedural (beweis-basiert) sein. Im folgenden wird die modellbasierte Variante beschrieben.
Def. 10: Eine Interpretation 1 einer DATALOG-Formel ist ein Tupel (D, PA, CA), in dem D ein Datendomain, CA eine Abbildung von Konstanten auf Elemente des Domains und PA eine Abbildung von n-ären Prädikaten (D^n) auf die Menge (richtig, falsch) ist.
Def 11: Eine Variablenauswertung Π (in einer bestimmten Interpretation) ist eine Abbildung, die jeder Variablen in einer DATALOG-Formel ein Element aus dem Domain D zuschreibt.

[0041] Eine bestimmte Interpretation und eine bestimmte Variablenauswertung definieren daher den Wahrheitswert einer DATALOG-Formel eindeutig.
Def 12: Ein Modell eines DATALOG-Programms ist eine Interpretation, in der alle Regeln und Fakten den Wahrheitswert „Richtig" haben. Dies bedeutet:

1) Für alle Tupel (t1 ... tn), die in der Relation P vorkommen, ist P(t1 ... tn) „Richtig".
2) Für alle Regeln und alle Variablenauswertungen Π gilt, daß wo immer π(A1 ∧ ... An) „Richtig" ist (die Ai's sind sämtliche Prämissen einer Regel), die Konklusion auch den Wahrheitswert „Richtig" besitzen .

[0042] Ein Modell ist also eine Menge von Prädikatinstanzen, die alle Tupeln der intensionalen Datenbasis IDB enthält.

[0043] Da es mehrere Modelle eines DATALOG-Programmes geben kann, spricht man von der Semantik und meint das minimale Modell. In (Van Emden, Kowalski, The semantics of predicate logic programming languages, Journal of the ACM, Oktober 1976) wurde bewiesen, daß es für DATALOG-Programmkonstrukte ein solches Modell gibt.

[0044] Man gelangt zu folgender Definition:
Def. 13: Die kanonische, deklarative Semantik eines DATALOG-Programms ist nur und ausschließlich die Menge aller Prädikatinstanzen, die aus dem Programm abgeleitet werden können. Diese Menge wird minimales Modell genannt.

[0045] Diese Definition impliziert eine Modellgenerationsprozedur. Man nimmt die Fakten der Datenbasis als Anfangspunkt und wendet solange eine Regeln nach der anderen an, bis keine neuen Fakten mehr generiert werden. Diese Prozedur wird naive Vervollständigung genannt, welche erfindungsgemäß zentraler Punkt des neuen Verfahrens ist.

Die logische Vervollständigung

[0046] 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. Das erreicht man am leichtesten mit Hilfe von sogenannten Vervollständigungsprozeduren:
Alg. 1 (naive Vervollständigung):

Input: Ein funktionsfreies, definites und endliches logisches Programm (mit DATALOG als Regeldefinitonssprache) .
Output: Die vollständige Extension aller Relationen in D (geschrieben ext(D)).
Schritt 1: M = Alle Fakten in D.
Schritt 2: Repeat
Schritt 2.1 Setze M_{alt} = M;
Schritt 2.2 Für jede Regel Q ← A1 ∧ A2 ... ∧ An; Begin
Schritt 2.2.1 Berechne jede Variablenauswertung Π, so daß
π(A1 ∧ A2 ... ∧ An) „Richtig" in Mist;
Schritt 2.2.2. Wenn $\prod(Q) \notin M$, füge es zu M.
End
until ($M = M_{alt}$)

[0047] Zwei wesentliche Schwachpunkte dieses Verfahrens sind die unkontrollierte Wiederholung von Deduktionsschritten und die Generation von Extensionen aller Relationen. Was die Komplexität betrifft ist klar, daß man bezogen auf die Länge der Formeln, mit einem exponentiellen Aufwand bei der Generierung rechnen muß.
Def. 14: Eine Prämisse Ai einer Regel Q ← Al ∧ A2 ... ∧ An wird als Muster bezeichnet, wenn es mindestens ein Faktum F und eine Variablenauswertung Π in der Datenbasis gibt, so daß F = ΠP(Ai) (man sagt: Ai unifiziert F).
Proposition 1: Angenommen jedes Muster M einer durchschnittlichen Regel Q ← Al ∧ A2 ... ∧ An unifiziert m Fakten von ext(D) (im Durchschnitt), dann ist der Aufwand von Alg. 1 im schlechtesten Falle $O(m^n)$.

Beweis:

[0048] Die Berechnung aller Variablenauswertungen Π in Schritt 2.2.1. benötigt einen Algorithmus, welcher die Schnittmenge der Ais (in A1 ∧ A2 ... ∧ An) bildet. Der Aufwand dieses Algorithmus – angenommen die Mengen sind nicht sortiert – ist $c_1 \cdot m^n$, wo c_1 eine Konstante ist. Im Falle einer geeigneten Sortierung wäre der Aufwand $c_2 \cdot n \cdot m \cdot \log(m)$, wobei $c_2 > c_1$. Dies ist aber in Alg. 1 nicht der Fall, da man keine Ordnung der Fakten voraussetzt.

[0049] Schritt 2.2.1. ist nicht der einzige Schritt, der einen Engpaß bildet. Der Suchvorgang in Schritt 2.2.2. ist sehr aufwendig, wenn keine Sortierordnung existiert. Im iterativen Prozeß wird das neue Faktum erst dann in die Datenbasis eingefügt, wenn man die ganze Datenbasis nach ihm durchsucht hat. Angenommen der Prozeß generiert k neue Fakten und t Fakten existieren bereits in der Datenbasis, dann sind im Durchschnitt k·t Vergleiche notwendig. Wenn aber k sehr groß wird (das zehnfache von t z. B.), dann ist mit der langen Kette $t + [1/\alpha_0] \cdot k \cdot t + [1/\alpha_1] \cdot k \cdot t + ... + k \cdot t$ zu rechnen, wobei die Koeffizienten a0, a1 ... usw. von der logischen Form abhängen. Diese entscheidet in welchen iterativen Schritten welche „Portion" von k gebildet wird.

[0050] Trotz diesen Schwierigkeiten bleibt Alg. 1 eine sichere Methode um das minimale Modell eines DATALOG-Programms zu generieren. Dies aufgrund folgender Eigenschaft:
Proposition 2: Alg. 1 terminiert – bei korrektem Input – immer.

[0051] Für den Beweis benötigt man einige Definitionen und eine Proposition.
Def. 15: Das Herbrand-Universum eines logischen Programmes U_p ist die Menge aller Grundterme, die aus den Konstanten und Funktionssymbolen in P erstellt werden können.
Def. 16: Die Herbrand-Basis eines logischen Programmes B_p ist die Menge aller Grundatome, die aus den Prädikatsymbolen von P und der Grundterme in U_p erstellt werden können.
Def. 17. Eine Herbrand-Interpretation eines logischen Programmes ist irgendeine Teilmenge von B_p.
Def. 18: Eine Herbrand-Interpretation 1 ist Herbrand-Modell eines definiten logischen Programmes P, gdw. alle definiten Klauseln in P den Wert „Richtig" haben.

[0052] Eine definite Klausel hat den Wert „Richtig", wenn mindestens eine atomare Prämisse „Falsch" oder die Konklusion „Richtig" ist. Ein Atom A hat den Wert „Richtig" in I, wenn A ∈ I und andernfalls „Falsch".

[0053] Das Herbranduniversum eines Programmes P, das nur die Konstante {0} und die Funktion {s} enthält, ist {0, s(0), s(s(0)), s(s(s(0))) ...}. Angenommen es enthält auch ein Prädikatensymbol „>„, dann ist die Herbrandbasis
{> (s0, 0), > (s(s0), s(0)), > (s(s(s(0))), s(s(0))) ...}.

[0054] Im Beispiel sind U_p und B_p unendlich, da man durch s beliebig lange Terme bilden kann. Wenn man aber keine Funktionssymbole zuläßt, dann sind U_p und B_p endlich.

[0055] Dies ist der zentrale Gedanke der folgenden Proposition:
Proposition 3: Die Herbrandbasis eines definiten, funktionsfreien und endlichen logischen Programmes P, ist endlich.

Beweis

[0056] Aus der Endlichkeit und der Funktionsfreiheit von P folgt unmittelbar die Endlichkeit des Herbranduniversums. Darüber hinaus ist die Prädikatenmenge von P endlich. Daraus folgt, daß B_p endlich ist.

Beweis von Proposition 2

[0057] Falls Alg. 1 bei einem korrekten Input nicht terminiert, dann gibt es unendliche Ketten von Grundatomen, die entweder unendlich neue Instanzen von Grundtermen des Herbranduniversums, oder bereits immer wieder generierte Atome aus B_p enthalten.

[0058] Erstere Möglichkeit ist ausgeschlossen, da B_p und U_p endlich sind. Die zweite Möglichkeit tritt nicht auf, da Schritt 2.2.2. garantiert, daß nur neugenerierte Atome zur Datenbasis ergänzt werden.

Alternative Vervollständigungsverfahren

[0059] 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.: Query Evaluation and Recursion in Deductive Database Systems, Manuscript, March 1985) oder (Lozinskii: Evaluation queries in deductive databases by generating, Proc. Int. Joint Conference on A. I., 1985) existieren verschiedene alternative Verfahren zur naiven Vervollständigung.

[0060] 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, Proc. of the ACM SIGMOD-SIGACT Conference, Washington D. C., May 1986).

[0061] Diese versucht die unnötige Wiederholung von Generationsschritten (Schritt 2.2.1. in Alg. 2) zu vermeiden, indem sie nur die inkremental-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).

[0062] Die Annahme ist, daß $\Delta R_i = (R_i \cup F(R_{i-1} \cup \Delta R_{i-1})) - R_i$ für jede Relation R_i (ΔR_i ist hier die inkrementale Änderung von R_i und $F(R_i)$ der funktionale Ausdruck, den man aus dem Rumpf einer Regel deduziert). Im allgemeinen kann man ΔR_i nicht einfach als Funktion von ΔR_{i-1} berechnen. Im Falle von linear-rekursiven Regeln ist das aber möglich, weil $F(R_{i-1} \cup \Delta R_{i-1}) = F(R_{i-1}) \cup F(\Delta R_{i-1}) = RU \cup F(\Delta R_{i-1})$.

[0063] Solange man davon ausgehen kann, daß die Regeln linear-rekursiv sind, ist die semi-naive Vervollständigung eine effizientere Alternative zu Alg. 1. 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).

[0064] Desweiteren wird der exponentielle Aufwand nur durch die Reduktion der Anzahl der Fakten des einen für den letzten Deduktionsschritt relevanten Musters gemindert. In einer Verknüpfung $A_1 \wedge A_2 \wedge ... \wedge S \wedge ... A_n$, 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 gesehen (Proposition 1), daß die Erstellung der UND-Verknüpfungen den aufwendigste Schritt in der Vervollständigungsprozedur ausmacht.

[0065] Die sogenannte APEX-Prozedur [Lozinskii 1985] ist ein Verfahren der zweiten Art. Es werden zuerst die für eine definite Abfrageklausel W? (Ziel) relevanten Fakten einer Datenbasis generiert, um erst dann den Vervollständigungsprozeß zu beginnen.

[0066] Die relevanten Fakten werden mit Hilfe von sogenannten Regelsystemgraphen berechnet. Diese beinhalten alle logischen Verknüpfungen zwischen Regeln der Datenbasis.

[0067] 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 die Abfragen und den Fakten der jeweiligen Verknüpfung(en).

[0068] 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). Dort werden die Datenbasisregeln (wie in APEX) für die Generation von neuen Abfragen benutzt. Die relevanten Fakten werden jedoch mit Hilfe eines „backward-chaining" Verfahrens linear und in der Tiefe (engl.: depth-ferst) (wie in PROLOG) gesucht. Im Falle von rekursiven Prädikaten, werden die Abfragen mit Hilfe von den bereits gefundenen Fakten durch SIP generiert.

[0069] 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 die üblichen Inferenzmechanismen über die Berücksichtigung der relevanten Fakten nur zu optimieren versuchen.

[0070] 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:
anc(X, Y) ← par(X, Y).
anc(X, Y) ← anc(X, Z) ∧ par(Z, Y).
und der Abfrage q(X) ← anc(a, X).
das neue „magische" Programm:
magic(a).
q(X) ← anc(a, X).
anc(X, Y) ← par(X, Y).
anc(X, Y) ← magic(X) ∧ anc(X, Z) ∧ par(Z, Y).
magic(Z) ← magic(X) ∧ anc(X, Z).

[0071] Das neue magic-Prädikat stellt eine Restriktion der zulässigen Substitutionen dar. Es verbindet die Konstanten des Programmes systematisch miteinander.

Semantische Betrachtungen zur Bedeutung von Variablen im RD-Modell

[0072] Es existieren Datalog-basierte Ansätze zur Constraintsbehandlung.

[0073] Das Grundproblem besteht darin, den bei der Anwendung einer Constraintmenge von DATALOG-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.

[0074] DE19725965C2 löst dieses Problem durch die Einführung der Herbrandabstraktionenstruktur. Hierbei werden Variablen als Abstraktionen von Begriffen und Begriffsbeziehungen in der Katalogsebene 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 den 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 Alg. 2 formalisierte Verfahren in DE19725965C2 gibt jedoch kein Verfahren an, wie die Instanzierung der Regeln optimiert werden könnte. Dieses könnte in einer Herbrandabstraktionenstruktur manigfaltig erreicht werden. Zudem besteht der wesentliche Schwachpunkt bei der Benutzung der Herbrandabstraktionenstruktur darin, dass sie im schlechtesten Falle einen exponentiellen Suchraum darstellt.

[0075] Das hier vorgestellte Verfahren führt 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 realisieren werden. 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.

[0076] 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. In der Praxis wird im Allgemeinen 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)

[0077] Nicht alle Solverparadigmen kommen gleich gut mit allen Formelkategorien zurecht. Man unterscheidet vier Solver-Arten welche einzeln ausgeführt werden. Um eine klare Übersicht der heutigen Methoden zu gewährleisten, wird jede Art durch ihre Implementierung der folgenden Merkmale gekennzeichnet:
Randomisierung, Vollständigkeit, Algorithmenklasse, Suchstrategie, Variablenauswahlheuristik, Werteauswahlheuristik

Verfahrensbegriffe:

[0078] Branch and Bound: Verfahren des Operations Research (OR), bei dem ein zu lösendes kombinatorisches Optimierungsproblem (endliche Anzahl unabhängiger Variablen mit diskretem Wertevorrat) keiner effektiven analytischen Behandlung zugänglich ist oder Enumerationsverfahren (Entscheidungsbaumverfahren) wegen des Rechenaufwandes ausscheiden. Lässt sich das Problem durch n diskrete Variablen, die jeweils k mögliche Werte annehmen können formulieren, dann liegt eine Interpretation als qualitatives Entscheidungsproblem nahe.

[0079] Vorgehensweise: Das Lösungsverfahren verwendet das Prinzip der Aufteilung und Begrenzung des Lösungsraumes, um eine vollständige Enumeration zu entbehren.

Schritte:

a) Branch (Verzweigung): Einer der Variablen wird ein bestimmter zulässiger Wert zugeordnet, wodurch ein neues Unterproblem entsteht, dessen Umfang um eine Variable geringer ist. Bei k möglichen Werten für die ausgewählte Variable entstehen so k-„einfachere" Unterprobleme. Es bleibt festzustellen, welches der Unterprobleme die optimale Lösung enthält.
b) Bound (Schranke): Nach der Fixierung einer Variablen wird ermittelt, wie die Lösung für die restlichen Variablen günstigenfalls ausfallen kann. Hat man für alle möglichen Werte einer ausgewählten Variablen die Bounds ermittelt, so wählt man die Alternative mit dem günstigsten Bound, um zur nächsten Verzweigung weiterzugehen. Wird nach mehrmaligem Branching und Bounding eine zulässige Lösung erreicht,

so können alle Fälle mit ungünstigeren Bounds gestrichen werden. Das Optimum ist erreicht, wenn keine günstigere zulässige Lösung mehr zu erwarten ist.
c) Unit-Propagation: Die Formel wird auf Unit-Klauseln durchsucht. Eine Unit-Klausel ist eine Klausel, in der alle Variablen, bis auf eine, bereits belegt sind und die Klausel noch nicht erfüllt ist. Eine solche Klausel kann nur erfüllt werden, wenn die letzte, nicht belegte, Variable so belegt wird, dass die Klausel wahr wird.

[0080] Die vier Solver-Arten sind nun wie folgt zu klassifizieren:

1. DPLL/Look-Ahead

[0081] Benannt nach Davis, Putnam, Logeman, Loveland (siehe: M. Davis, G. Logemann, and D. Loveland. A Machine Program for Theorem Proving. CACM, 5(7): 394–397, 1962.), ursprünglich DP (DP-Resolution), dann DLL, auch DPLL.

Randomisiert:	Nein
Vollständig:	Ja
Algorithmenklasse:	Branch-and-Bound (chronologisches Backtracking)
Suchstrategie:	systematisch, binärer Suchbaum
Variablen-Auswahlheuristik:	VAR, wählt eine Variable die als nächstes belegt werden soll (decision).

Werte-Auswahlheuristik:

– VAL, wählt den Wert, der dieser Variablen als erstes zugewiesen werden soll (direction).
– Belegen einer Variable + unit propagation (= Branch).
– Konflikt (also eine der Klauseln wird leer) (= Bound): Belegungen rückgängig machen. Decision variable in anderer Richtung belegen, falls möglich, ansonsten Backtracking.
– Falls backtracking für aller erste Variable nötig: Formel ist unerfüllbar.

[0082] Die Güte eines Branch-and-Bound-Verfahrens hängt hier wesentlich von der Auswahl der Bounds ab. Diese Auswahl kann nur heuristisch erfolgen; es ist deshalb nicht möglich, über die Konvergenz des Algorithmus Aussagen zu machen. Nachteilig beim DPLL ist aber vor allem der potentiell exponentiell große Suchbaum und die Tatsache, dass es sich nur bei speziellen Formelarten gut verhält (siehe Tabelle 1 im weiteren).

2. SLS

[0083] Benannt nach: Stochastic Local Search (kurz SLS, siehe beispielsweise: Balint, A., Fröhlich, A.: Improving stochastic local search for SAT with a new probability distribution. In: SAT-2010. LNCS, Vol. 6175, pp. 10–15 (2010))

Randomisiert:	Ja
Vollständig:	Nein
Algorithmenklasse:	Las Vegas Algorithmus (randomisierter Algorithmus, der immer ein korrektes Ergebnis liefert, wenn er terminiert)
Suchstrategie:	randomisiert, Optimierung einer Zielfunktion (objective function)

a. Wähle zufällig vollständige Startbelegung α
b. Verifiziere, ob α(F) = 1. Falls ja: Ende.
c. Falls nein: U = {c|c ε F: α(c) = 0}. Wähle u ε U zufällig
d. In u, wähle eines der Literale bzw. die dazugehörige Variable (PICKVAR) mittels 'objective Function'
e. Invertiere die Belegung (FLIPVAR) und gehe zu b.

[0084] Die Variablenauswahlheuristik PICKVAR wählt eine Variable mit Hilfe einer 'objecive function', die optimiert werden soll.
– Variante 1: Minimiere die Anzahl der unerfüllten Klauseln (betrachte alle Variablen in u und flippe diejenige Belegung, bei der |U| nach dem Flip minimal)
– Variante 2: Minimiere die Anzahl der Klauseln, die durch den Flip unerfüllt werden.

[0085] Nachteilig ist, dass falls Varianten 1 und 2 in ein lokales Minimum geraten sie sehr ineffizient werden. Die einzige Lösung dagegen ist eine Randomisierung (Zufallsflips). Auch kann nicht feststellen, ob abgefragte Belegung vorher schon mal abgefragt wurden. Das grösste Defizit von SLS liegt in der Unvollständigeikt der Mehtode.

3. MP

[0086] Benannt nach: Message Passing, Ansatz motiviert durch statistische Physik (Ising Modell, siehe: W. Gropp, E. L. Lusk, N. Doss, and A. Skjellum. A High-Performance, Portable Implementation of the MPI Message Passing Interface Standard. Parallel Computing, 22(6): 789–828, 1996.)

Randomisiert:	Ja
Vollständig:	Nein
Algorithmenklasse:	Message passing
Suchstrategie:	randomisiert, Klauseln und Variablen verhandeln über die Belegung (Faktorgraph)

– Eine Klausel fragt bei Variablen an, mit welcher Wahrscheinlichkeit sie eine günstige Belegung annehmen
– Je schlechter die Gesamtsituation für die Klausel, desto stärker fordert die Klausel die Variablen ihre Belegung zu ändern
– Forderungen (Klauseln an Variablen) und Zustimmung (Variablen an Klauseln) werden als Nachrichten aufgefasst
– Klauseln reden nur mit den Variablen, deren Literale sie auch enthalten.
Variablen reden mit allen Klauseln, in denen sie vorkommen (Bipartiter Graph mit zwei Kantentypen: Faktorgraph)
– Sind sich alle einig (kaum Veränderungen in den Nachichten): Variablen mit stärkestem Bias belegen

[0087] Hierbei kann es kann sein, dass MP zwar konvergiert, aber alle Variablen einen Bias nahe 0 haben. In dem Fall wird bspw. SLS verwendet, um die verbleibenen Variablen zufällig zu belegen, so dass eine erfüllbare Belegung entsteht. Weiter kann es sein, dass MP nicht konvergiert oder dass MP konvergiert und nicht-triviale Biases entstehen, aber das Belegen in einem Konflikt endet. MP funktioniert sehr gut auf erfüllbaren großen (Variablenanzahl > 100000) random Formeln mit Ratio an die 4. 2 (Klauseln/Variablen Ratio).

[0088] Nachteilig ist, dass MP nicht feststellen kann, ob eine Formel unerfüllbar ist (ist also grundsätzlich unvollständig).

4. CDCL

[0089] Benannt nach: Conflict Driven Clause Learning, (siehe: E. Goldberg and Y. Novikov. BerkMin: a fast and robust SATsolver. In Design, Automation and Testing in Europe Conference, pages 142–149, March 2002)

Randomisiert:	Nein
Vollständig:	Ja
Algorithmenklasse:	Backtracking (nicht-chronologisch, = Backjumping)
Suchstrategie:	systematisch, Klausel-lernen um Unit Propagation (UP) zu verbessern

– VAR und VAL (wie bei DPLL).
– Belegung einer Variablen und Überprüfung, ob UP in einem Konflikt endet (vgl. DPLL).
– Konstruktion eines Suchbaums jedoch nicht per Rekursion.
– Entscheidungen werden gezählt und gespeichert sowie alle Belegungen, die mit UP daraus gefolgt sind.
– Läuft der Algorithmus in einen Konflikt, so analysiert er diesen mittels eines Konfliktgraph (FirstUIP).
– Dabei entsteht eine neue Klausel: diese wird in die Formel eingefügt.
– Danach findet backjumping statt, und zwar derart, dass die neue Klausel eine unit Klausel ist.

[0090] Nachteilig ist, wie bereits zum DPLL notiert: Potentiell exponentiell grosser Suchbaum, Leistungs Steigerung nur bei speziellen Formelarten (siehe Tabelle 1 unten).

[0091] Alle vier Solver-Verfahren können durch folgende Merkmale charakterisiert werden und sind damit vom nachfolgend vorgestellten erfindungsgemäßen Verfahren signifikant verschieden:

1. Sie sind ein Beispiel für die Anwendung von Torskis 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) lediglisch wie Behälter betrachtet, die nicht zulassen, dass 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 benutzen, wird in jeder Iteration nur ein Teil des Raumes konstruiert, die Formel darauf ausgewertet, 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”:

[0092]

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

[0093] Letztlich ist ein Solver-Verfahren bekannt, dass der klassischen Wahrheitstabellenmethode entspricht. Es unterscheidet sich von den oben beschriebenen Verfahren in den Punkten 4 und 5 wie folgt:

1. Bestandteil der Methode ist die Konstruktion des gesamten exponentiellen Raumes aller Kombinationen der Variablenwerte. Nachdem dieser Raum konstruiert wurde 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 Methden 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. Die Leistung des Verfahrens ist, dadurch dass keine Annahmen über die Formeln gemacht werden, unabhängig von der Formelart.

Ziel der Erfindung

[0094] Die Erfindung hat das Ziel, relationale Datenbanksysteme in ihrem allgemeinsten und vollständigsten Konzept, unter Beibehaltung strengster logischer Bedingungen, in ihren logischen Abfrageverfahren so zu optimieren, daß die Antwortsprozedur lineare Effizienz in Bezug auf die Geschwindigkeit und den Speicherbedarf erfährt.

Darlegung des Wesens der Erfindung

[0095] Der Erfindung liegt die Aufgabe zugrunde, ein Verfahren der eingangs genannten Art zu kreieren, das relationale Datenbanksysteme in ihrem allgemeinsten und vollständigsten Konzept in ihren Abfrageverfahren so optimiert, daß die Antwortsprozedur eine Effizienzsteigerung in Bezug auf die Geschwindigkeit und den Speicherbedarf erfährt, ohne daß logische Bedingungen aufgegeben werden müßten.

[0096] 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 befindlichen logisch-vollständigen, effizienten Faktensuchraumes

[0097] Das Verfahren dieser Erfindung basiert auf der Idee, alle für die terminologische, logische und anwendungsspezifische Kontrolle notwendigen Daten, im Katalog im Voraus zur Verfügung zu stellen. Dadurch wird der Constraintsbehandlung effizienter nachgegangen. Konsistenzeigenschaften von Datenbanken (bezüglich Mengen von Constraints), sind ausschließlich logischer Natur, und deshalb Metaprobleme.

[0098] Das folgende Beispiel erläutert dieses Verfahren:
Sei eine Druckmaschinendatenbank gegeben. Diese enthält die Tabellen „Maschine” und „Firma”, wie in (**Fig.** 3) ersichtlich ist. Die Tabelle „Maschine” besitzt – unter anderem – 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”) auch keinen Sinn macht. Würde man diese Information in Form von allgemeinen DATALOG-Regeln auffassen, so stößt man auf das Problem, solche Regeln faktenbezogen definieren zu müssen, da sie nur sehr schwer Allgemeingültigkeit besitzen. Der übliche formal-logische Weg führt zwangsweise zur nicht-monotonen Logik und damit zu erheblichen Effizienzbeschränkungen. Der in Patentschrift DE19725965C2 verfolgte Ansatz besteht darin, zulässige Faktenkombinationen im Voraus bereitzustellen indem man die Grundterme bzw. Beziehungen des Kataloges logisch vervollständigt. Somit erreicht man:

1. Eine strikte terminologische Kontrolle, d. h. es dürfen in den Feldern keine neuen, für das System unbekannte Werte eingegeben werden.
2. Die Terme stehen untereinander in genau den Beziehungen, die im vollständigen Katalog aufgelistet sind.

[0099] Im Unterschied zu DE19725965C2 wir jetzt durch die Einführung neuer Resolutionssverfahren ein effizienter Faktensuchraum aufgebaut. **Fig.** 7 zeigt das neue Verfahren einer SQL-Klauselbearbeitung:
Die in Punkt (6) zusammengestellten Anfragen werden in (7) verifiziert. Falls das Resultat der Suche positiv war, läuft die Befehlssequenz wie üblich weiter. Ansonsten wird eine Systemwarnung (8) erfolgen. Die Begriffskombination ist in diesem Falle ungültig.

[0100] Beispielsweise würde in der obigen Druckmaschinendatenbank bei einer Dateneingabe (Typ = „Polar” und Gruppe = „5-Farben”) der Abfragengenerator das Faktum „hat_gruppe(polar,5-Farben)” zusammenstellen. Dieser erweist sich als nicht existent und schaltet den Warnprozeß in (8) ein.

[0101] Die folgenden Schritte erläutern Aspekte der Erfindung unter Bezugnahme auf **Fig.** 8:

[0102] Schritt 1: Die Datenbankabfrage wird zu Klauseln übersetzt. Dies geschieht mit Hilfe von bekannten Verfahren.

[0103] Schritt 2: Benutzte Literale werden in Musterbäume (Musterzeichenketten) konvertiert, die jeweils das Vielfache (genannt: Harmonie) einer Grundfrequenz von Einsen und Nullen („Wahr” und „Falsch”) darstellen. Diese Musterbäume haben eine konstante, von der Anzahl der Variablen unabhängige Länge.

[0104] Schritt 3: Die Musterbäume verschiedener Literale einer Klausel werden mittels einer speziellen Art von OR-Operation (genannt: PatternOr) zu einem einzigen binären Baum zusammengefügt, der jeweils die Klausel entspricht.

[0105] Schritt 4: Musterbäume, die Klauseln darstellen, werden mittels einer speziellen Art von AND-Operation (genannt: PatternAND) resolviert. Dieses neue Resolutionsverfahren generiert einen binären Baum (genannt: Resultats-Baum, siehe **Fig.** 9), der in einen kompilierten Entscheidungsbaum für die logische Theorie transformiert werden kann. Wichtigstes Merkmal dieses letzteren Baumes ist die Tatsache, dass er die Wahrheitstabelle der gegebenen Klauselmenge vollständig ersetzt, d. h. eine Klauselmenge ist wahr genau dann wenn der Wert „Wahr" aus diesem Baum inferiert werden kann. Der Baum wird benutzt, um den gesamten Wahrheitswert aller Klauseln zu finden. Schritt 5: Weiter bearbeitet der Baum die Abfrage derart (übersetzt in instanzierten Formeln der Logik), dass nur lineare Baumsuchverfahren notwendig sind, um den Wahrheitswert zu finden. Dieser Wahrheistwert wiederum stellt – in der benutzten Übersetzung – ihre Ausgaben dar.

Bestimmen des Wahrheitswertes einer Abfrage mittels eines
Verfahrens zur Verarbeitung von logischen Musterbäumen

Verfahren 1: Das zentrale Verfahren dieser Erfindung bestimmt den
Wahrheitswert einer Formelmenge mittels der unten ausgeführten Schritte
(Patentanspruch 1). Die Klauselmenge stellt eine beliebige Datenbankabfrage dar.

Schritt 1 – Konvertieren der Abfrage in Klauseln: In diesem Schritt werden bekannte Verfahren angewandt, um zur logischen Darstellung der Abfrage zu gelangen.
Schritt 2 – Konvertieren aller Literale zu Musterzeichenketten: Angenommen folgende Klauselmenge entspricht einer Abfrage: C = {{x0, x2}, {nx0, nx1}{x1, x2}} mit N = 3 (Anzahl der Variablen) und M = 3 (Anzahl der Klauseln), dann ist, wie bekannt, die klassische Wahrheitstabelle in Tab. 2 gegeben:

[0106]

X0	X1	X2	C1	C2	C3
0	0	0	0	1	0
0	0	1	1	1	1
0	1	0	0	1	1
0	1	1	1	1	1
1	0	0	1	1	0
1	0	1	1	1	1
1	1	0	1	0	1
1	1	1	1	0	1

Tab. 2

[0107] Das folgende Verfahren generiert eine Liste (genannt: Musterzuweisungsliste), die jedem Literal und seiner Negation eine eindeutige Musterzeichenkette wie folgt zuweist (siehe **Fig.** 9).

[0108] Die Liste ist eine Aufreihung aller möglichen Literale der N Variablen (d. h. sie ist 2·N lang) und jedem Literal wird eine Zeichenkette zugewiesen, die Harmonien von Einsen und Nullen darstellt. Diese Harmonien werden durch die im Verfahren beschriebenen Gleichungen gegeben. Das Literal nx0 beispielsweise erhält die Musterzeichenkette: 2(1)2(0). In einer Musterzeichenkette wird der Faktor 2^i Mustermultiplikationsfaktor genannt.

Verfahren 2: Konstruktionsverfahren einer Musterzuweisungsliste
für ein beliebiges N (Patentansprüche 2–7):

Schritt 1: Starte mit einer leeren Liste
Schritt 2: für i = 0 to N – 1
a) Füge ein –ves Literal an, nenne es nX<i>
b) Weise ihm die Musterzeichenkette 2^i(2^[N – i – 1](1) 2^[N – 1 – i](0)) zu
c) Füge ein +ves Literal an, nenne es X<i>
d) Weise ihm die Musterzeichenkette 2^i(2^[N – i – 1](0) 2^[N – 1 – i](1)) zu

Schritt 3 – Ausführen der PatternOr-Operation: Die durch das Verfahren 2 gewonnene Musterzuweisungsliste wird dazu verwendet, Musterbäume von Klauseln zu kreieren. Zur Kreierung eines logischen Musterbaumes aus einer Musterzuweisungsliste ist ein logischer Operator (hier PatternOr genannt) notwendig der im Unterschied zum klassischen OR die Muster/Untermustermultiplikationsfaktoren in Betracht nimmt, indem er gleiche Operationen zwischen Unterzeichenketten, die Multiplikationsfaktoren besitzen, nur einmal ausführt. Wenn PatternOr beispielsweise zwischen x0 = 4(0)4(1) und x2 = 4(1(0)1(1)) angewandt wird, hat man (siehe **Fig.** 10a und b.) nur folgende Operationen durchzuführen:
a) Die Unterzeichenkette 4(1) wird vollständig kopiert. Sie stellt den rechten Teil des Resultates dar.
b) Die Unterzeichenkette 4(0) wird rekursiv mit der Hälfte von 4(1(0)1(1)) weiterbearbeitet, d. h. die Operation PatternOr(4(0), 2(1(0)1(1))) wird durchgeführt.
c) Der Ausdruck in b) kann in folgender Weise gekürzt werden: PatternOr(4(0), 2(1(0)1(1))) = 2·PatternOr (2(0), (1(0)1(1))), was bedeutet: Der PatternOr Operator wird nur einmal auf eine halb so lange Zeichenkette angewandt und dann mit einem Faktor von 2 multipliziert (d. h. die Zeichenkette wird mit sich selbst einmal konkateniert), um dasselbe Resultat zu erreichen, das in b. angestrebt wird.
d) Die Zeichenkette 1(1) wird vollständig kopiert und stellt den rechten Teil der Unterkette dar.
e) PatternOr(1(0), 1(0)) ergibt 1(0).

[0109] Die für PatternOr benötigten Schritte um eine Musterzeichenkette für C1 zu generieren lauten: 2 Kopierschritte (Kopieren von 4(1) und 1(1)), 2 Teilungsschritte (genannt: Splits), bei denen die Zeichenketten zur Hälfte geteilt werden und schliesslich zwei Divisions/Multiplikationsschritte. Es ist klar, dass die obige Verfahrensweise nicht mehr von N (oder M) abhängt, weil sie von der harmonischen Darstellung der Musterzeichnketten – unabhängig von ihrer Länge – Gebrauch macht.

Verfahren 3: Erstellen eines logischen Musterbaumes für Klauseln (PatternOR) (Patentanspruch 8):

[0110] Es wird angenommen,
1. dass die Abfrage bereits in Klauseln K<a>, 0 <= a <= M, übersetzt wurde,
2. dass Musterzeichenketten aller Literale der Klausel K<a> (der Form K<a> = {Xi, Xj, Xk} oder K<a> = {nXi, nXj, nXk}) mittels dem obigen Verfahren 2 schon bestimmt wurden,
3. dass das Verfahren bei der Eingabe beliebige solche Musterzeichenketten Z1 und Z2 annimmt,
4. dass je nachdem, wieviele Literale in der Klausel existieren (1, 2 oder 3), das Verfahren gar nicht, einmal oder zweimal hintereinander benutzt wird, und
5. dass das Verfahren einen Musterbaum liefert, der das Resultat der logischen OR Operation zwischen den Eingabemusterzeichenketten einer einzigen Klausel darstellt.
6. Benutze die Zeichenketten Z1 und Z2 in der folgenden Weise:
Schritt 1: Wenn beide Zeichenketten Multiplikationsfaktoren beinhalten, dann dividiere den grössten Faktor durch den kleinsten (sei f der kleinste) und setze PatternOr(z1, z2) = f·PatternOr(Z1/f, Z2/f), d. h. wiederhole Schritt 1 rekursiv mit den reduzierten Ketten, wobei Z1/f eine Zeichenkette ist, die einen durch f dividierten Multiplikationsfaktor beinhaltet. Ähnlich verhält es sich mit Z2/f.
Schritt 2: Wenn nur eine Zeichenkette (Z2 beispielsweise) einen Multiplikationsfaktor enthält, dann teile die andere (Z1) in zwei Hälften (split operation) auf. Setze PatternOr(Z1, Z2) = PatternOr(Z1links, Z2/2) & PatternOr(Z1rechts, Z2/2). Mit anderen Worten: Wiederhole Schritt 1 rekursiv zweimal, wobei beim ersten Mal die linke Seite und beim zweiten Mal die rechte Seite von Z1 benutzt wird. Verknüpfe das Resultat der beiden Rekursionen.
Schritt 3: Wenn keines der beiden Zeichenketten einen Faktor beinhaltet, setze PatternOr(Z1, Z2) = PatternOr(Z1links, Z2links) & PatternOr(Z1rechts, Z2rechts). Anders ausgedrückt: Wiederhole rekursiv Schritt 1 zuerst mit den beiden linken Seiten dann mit den beiden rechten Seiten von Z1 und Z2. Verknüpfe das Resultat wie oben.
Schritt 4: Wenn Z1 = 2^x(0) ist, dann soll das Verfahren Z2 ausgeben und terminieren. Wenn Z2 = 2^x(0) ist, dann soll das Verfahren Z1 ausgeben und terminieren
Schritt 5: Wenn Z1 = 2^x(1) ist, dann soll das Verfahren Z1 ausgeben und terminieren. Wenn Z2 = 2^x(1) ist, dann soll das Verfahren Z2 ausgeben und terminieren.

[0111] Verfahren 3 wurde in den **Fig.** 10 auf zwei Literale angewandt, da die Beispielklausel C1 nur zwei hatte. Nehmen wir an, dass C1' = {nx0, nx1, nx3} z. B. ist, dann zeigt **Fig.** 11 den Prozess der Musterbaumgenerierung für C1' nach Verfahren 3.

[0112] Verfahren 3 dient dem Zwecke der Generierung von logischen Musterbäumen, die die Klauseln darstellen. So gesehen ist die maximale Anzahl der Schritte dieses Verfahrens die notwendig sind um den Mus-

terbaum einer einzigen Klausel zu konstruieren, stets konstant, da sie von der Anzahl der Variablen N und der Klauseln M unabhängig ist.

[0113] Logische Musterbäume, die durch Verfahren 3 (PatternOr) gewonnen wurden, haben folgende Eigenschaften (siehe **Fig.** 11):

1. Sie enthalten maximum k (in der Figur k = 3) ineinander verschachtelte, sich wiederholende Zeichenketten (genannt: Muster-Symbole oder Symbole), die mit einem Wiederholungsfaktor (Multiplikationsfaktor) versehen sind. Das Wurzelsymbol [C<i>s3] in der Figur (genannt: Basissymbol) stellt die grösste sich wiederholende Zeichenkette dar. Ihre Unterzeichenketten, die sich auch mit einem Faktor wiederholen und keine Blätter sind (genannt: Knoten oder Untersymbole), werden je nach Grösse [C<i>s2] und [C<i>s1] bezeichnet. Die Anzahl der eindeutigen, sich wiederholenden, ineinander verschachtelten Mustersymbole eines Baumes wird Wiederholungstiefe genannt. Die Wiederholungstiefe der durch Verfahren 3 gewonnenen Musterbäume ist maximal k.
2. Blätter sind harmonische Wiederholungen von Einzen oder Nullen
3. Mustersymbole enthalten jeweils nur den Namen einer einzigen Klausel, <Ci>, da sie keine Verknüpfungen zwischen Klauseln darstellen. Diese Eigenschaft wird Kardinalität der Mustersymbole genannt. Die Kardinalität der durch Verfahren 3 gewonnenen Mustersymbole beträgt 1.
4. Mit der Verzweigungstiefe (oder einfach „Tiefe") eines logischen Musterbaumes wird die maximale Anzahl der Verzweigungen – bis zu den Blättern – bezeichnet. Ein durch Verfahren 3 gewonnener Baum hat deswegen eine maximale konstante Tiefe von k, die unabhängig von N oder M ist. Sie ist äquivalent mit der maximalen Wiederholungstiefe.

[0114] Schritt 4 – Im nächsten Schritt ist zu zeigen wie man gemäss dieser Erfindung die Musterbäume benutzt um das logische AND zwischen den Klauseln zu realisieren. Ähnlich zu PatternOr zeigt das folgende Verfahren 4 (PatternAnd oder Resolve) wie dies erfolgen kann. **Fig.** 14a und b beinhaltet die wichtigsten Fallunterscheidungen dieses Verfahrens.

Verfahren 4: Erstellen eines logischen Musterbaumes für die Gesamtheit der Klauseln einer logischen Formel (PatternAND oder Resolve) (Patentanspruch 9).

[0115] Es wird angenommen,

1. dass die Abfrage bereits in Klauseln K<a>, 0 <= a <= M, übersetzt wurde,
2. dass logische Musterbäume, die jede Klausel K<a> der Formel darstellen, mittels Verfahren 3 bereits vorliegen,
3. Verfahren 4 für beliebige Musterbäume Z1 und Z2 definiert ist und nicht nur für die, die einzelnen Klauseln entsprechen,
4. dass das Verfahren M – 1 Mal eingesetzt wird um das endgültige Resultat zu erreichen. In jedem Schritt i wird der bis dahin erreichte Musterbaum (Zwischenresultat Nummer i – 1) mit der nächsten Klausel Ci + 1 resolviert, und
5. dass das Verfahren einen Musterbaum liefert, der das Resultat der logischen AND Operation zwischen allen Klauseln darstellt.

Benutze die Zeichenketten Z1 und Z2 in folgender Weise:

[0116] Schritt 1: Wenn beide Zeichenketten Multiplikationsfaktoren beinhalten, dann dividiere den grössten Faktor durch den kleinsten (sei f der kleinste) und setze PatternAnd(z1, z2) = f·PatternAnd(Z1/f, Z2/f), d. h. wiederhole rekursiv Schritt 1 mit den reduzierten Ketten, wobei Z1/feine Zeichenkette ist, die einen durch f dividierten Multiplikationsfaktor beinhaltet. Ähnlich verhält es sich mit Z2/f.

[0117] Schritt 2: Wenn nur eine Zeichenkette (Z2 beispielsweise) einen Multiplikationsfaktor enthält, dann teile die andere (Z1) in zwei Teilen (split operation, siehe **Fig.** 12a und b, Fall 1 und 3) auf. Setze PatternAnd(Z1, Z2) = PatternAnd(Z1links, Z2/2) & PatternAnd(Z1rechts, Z2/2). Mit anderen Worten wiederhole Schritt 1 rekursiv zweimal, wobei im ersten Mal die linke und im zweiten Mal die rechte Seite von Z1 benutzt wird. Konkateniere das Resultat der beiden Rekursionen

[0118] Schritt 3: Wenn keines der beiden Zeichenketten einen Faktor beinhaltet (siehe **Fig.** 12a und b, Fall 2), dann setze PatternAnd(Z1, Z2) = PatternAnd(Z1links, Z2links) & PatternAnd (Z1rechts, Z2rechts). Anders ausgedrückt wiederhole rekursiv Schritt 1 zuerst mit beiden linken Seiten dann mit beiden rechten Seiten von Z1 und Z2. Verknüpfe das Resultat wie oben.

[0119] Schritt 4: Wenn Z1 = 2^x(0) ist, dann soll das Verfahren Z1 ausgeben und terminieren. Wenn Z2 = 2^x (0) ist, dann soll das Verfahren Z2 ausgeben und terminieren.

[0120] Schritt 5: Wenn Z1 = 2^x(1) ist, dann soll das Verfahren Z2 ausgeben und terminieren. Wenn Z2 = 2^x(1) ist, dann soll das Verfahren Z1 ausgeben und terminieren. **Fig.** 13a zeigt das Resultat von PatternAnd angewandt auf Klauselmenge C (auch Resolutionsresultat genannt).

[0121] Der wesentliche, praktische Unterschied zwischen Verfahren 3 und Verfahren 4 (ausser dass beide logisch verschiedene Operationen realisieren) ist der, dass Eingabemusterbäume, die konstante Tiefen im Falle von Verfahren 3 hatten für Verfahren 4 lineare Tiefen (in M) haben können. Der allgemeinste Fall eines logischen Eingabebaumes für Verfahren 4 wird in **Fig.** 13a gezeigt. **Fig.** 13b illusitriert die theoretische Expansion dieses Baumes, die bis zu M^M (d. h. faktoriell viele) Symbole zulassen kann. Praktisch kann man zeigen, dass die Anzahl der durch dieses Verfahren generierten, eindeutigen Musterbäume polynomial in der Anzahl der Literale der Abfrage bleiben kann, wenn man Literale geeignet substituiert.

[0122] Dieser Baum (genannt: allgemeiner Verknüpfungsbaum, Resolutionsresultat) hat folgende Eigenschaften:

a) Alle Mustersymbole (inklusive dem Basissymbol) sind – im Unterschied zu Klauselmusterbäumen – aus Symbolen mehrerer Klauseln zusammengesetzt, weil der Baum den logischen Verknüpfungen dieser Klauseln entspricht. Ihre Kardinalität ist also im allgemeinen > 1

b) Die Tiefe eines allgemeinen Verknüpfungsbaumes ist linear in M.

c) Die Kardinalität des Basissymbols und aller Untersymbole ist maximal M.

d) Blätter des Baumes sind selbstständige Musterbäume, die dieselbe Struktur wie die des allgemeinen Verknüpfungsbaumes aufweisen, aber eine maximale Kardinalität von M – 1 besitzen (d. h. sie stellen Resultate von maximal M – 1 Klausel Verknüpfungen dar)

e) Symbole des Baumes (als Mengen von Untersymbolen und Blättern gesehen) sind mittels einer nach oben beschränkten Halbordnung – bezogen auf ihre Länge – organisiert. Obere Schranke ist das Mte Basissymbol, untere Schranken sind die Symbole mit der Kardinalität 1.

Verfahren 5: Erstellen eines Entscheidungsbaumes für eine Klauselmenge (Patentanspruch 10):

[0123] Bei näherer Untersuchung von Verfahren 4 stellt sich heraus, dass es eine kannonische Aufteilung der Klauselmengen – als Nebenprodukt – zulässt (siehe **Fig.** 14a). Diese Aufteilung ist das Resultat der Ausführung von hintereinanderfolgenden Resolutionsschritten und wird dazu benuzt, den logischen Entscheidungsbaum zu konstruieren. Darüber hinaus kann man Musterlängen der Literale dafuer benutzen, geeignete Ordnungskriterien für die durch die Umbenennung der Variablen ermöglichte effiziente Herstellung dieses Baumes zu liefern.

[0124] **Fig.** 14b zeigt anhand des angegebenen Beispiels den Übergang vom Resultatsbaum zum Entscheidungsbaum. Um diesen Übergang zu gewährleisten, ist das folgende Verfahren notwendig:

[0125] Es wird angenommen,

1. dass die Abfrage bereits in Klauseln K<a>, 0 <= a <= M, übersetzt wurde,
2. dass ein Resolutionsresultat für die Klauselmenge bereits existiert,
3. dass Verfahren 5 als Eingabe einen Musterbaum reskursiv bearbeitet, und
4. dass dieser Musterbaum am Anfang mit dem Resolutionsresultat gleich gesetzt wird.

[0126] Konstruiere den Entscheidungsbaum wie folgt:

[0127] Schritt 1: Kreiere einen Knoten (= K) im Entscheidungsbaum. Kreiere zwei Knotenals linke und rechte Knoten für K(LK, RK).

[0128] Schritt 2: Betrachte die Klauselmenge des obersten Symboles im Musterbaum (= Menge) und vergleiche sie jeweils mit den Klauslmengen des linken (= LMenge) und des rechten Untersymboles (= RMenge) im selben Baum, wenn die Kardinalität dieser Symbole >= 1 ist. Bestimme im letzteren Falle die Variable die aus den unteren Klauselmengen verschwunden ist. Setze diese Variable als Namen von K ein.

[0129] Schritt 3: Wenn linke oder rechte Untersymbole des obersten Symbols im Musterbaum eine Kardinalität < 1 und die Form 2^x(0) haben, dann ist der linke bzw. rechte Knoten (LK oder RK) 'falsch'. Wenn sie dagegen die Form 2^i(1) haben, dann ist derselbe 'richtig'.

[0130] Schritt 4: Bestimme die partielle Variablenauswertung, die aus der originalen Klauselmenge Menge die untere linke Menge (LMenge) macht und schreibe sie auf einen Pfeil, der aus dem neuen Knoten K links zu LK rausgeht.

[0131] Schritt 5: Wiederhole Schritt 4 für die untere rechte Menge (RMenge).

[0132] Schritt 6: Setze Musterbaum = linker Unterbaum des jetztigen Musterbaumes, wenn LK nicht 'falsch' oder 'richtig' ist, und rufe dich rekursiv auf, ansonsten halte an.

[0133] Schritt 7: Setze Musterbaum = rechter Unterbaum des jetztigen Musterbaum, wenn RK nicht 'falsch' oder 'richtig' ist, und rufe dich rekursiv auf, ansonsten halte an.

[0134] Schritt 8: Setze das Resultat des linken und des rechten rekursiven Aufrufs, wenn die geschehen, als linker bzw. rechter Unterknoten des Knotens K ein (an der Stelle von LK und/oder RK).

[0135] Zu Schritt 4: Wenn beispielsweise die obere Klauselmenge {X0, X2}{nX0, nX1}{X1, X2} und die untere {X2}{X1, X2} war, dann schreibe die Auswertung X0 = 0 auf den linken Pfeil der aus dem Knoten X0 herrauskommt (siehe **Fig.** 14b). Beachte, dass diese Variablenauswertung stets nur eine Variable beinhaltet, nämlich die, die als Namen des oberen Knotens benutzt wurde.

[0136] Verfahren 6: Auffinden des gesamten Wahrheitswertes einer Klauselmenge (Patentanspruch 11): Der Entscheidungsbaum wird dazu benuzt, den gesamten Wahrheitswert der Klauseln zu finden.

[0137] Es wird angenommen,

1. dass die Abfrage bereits in Klauseln K<a>, 0 <= a <= M, übersetzt wurde,
2. dass bereits ein Entscheidungsbaum für die Klauselmenge existiert.

[0138] Finde den Wahrheitswert wie folgt (Navigation im binären Baum):

[0139] Schritt 1: Setze den Zeiger auf dem Basisknoten im Baum.

[0140] Schritt 2: Wenn dieser Knoten ein Blatt ist, terminiere mit der Ausgabe 'wahr' oder 'falsch' je nachdem, ob der Wert des Blattes 'wahr' oder 'falsch' ist.

[0141] Schritt 3: Wenn der Knoten kein Blatt ist, rufe dich rekursiv auf, zuerst mit dem linken, dann mit dem rechten Knoten.

[0142] Schritt 4: Wenn der linke oder der rechte rekursive Aufruf den Wert 'wahr ergibt, dann terminiere den Basisaufruf mit dem Wert 'wahr', ansonsten terminiere mit dem Wert 'falsch'.

Verfahren 7: Benutzung des Entscheidungsbaumes für
die Verarbeitung der Abfrage (Patentanspruch 12 und 13):

[0143] Der letzte Schritt besteht darin, dass man den generierten Entscheidungsbaum benutzt. Das folgende Vefahren 7 beschreibt das detaillierte Vorgehen. **Fig.** 14b illustriert anhand des angegebenen Entscheidungsbaumes und der Wahrheitstabelle leicht nachvollziehbar, wie man das im Folgenden beschriebene Verfahren 7 auf konkrete Eingaben anwendet.

[0144] Es wird angenommen,

1. dass bereits ein Entscheidungsbaum B für die Klauselmenge generiert wurde,
2. dass die Abfrage in Form von instanzierten Klauseln K'<a> existieren, wobei K'<a> aus K<a> gewonnen wird, indem man alle Literale mit Werten aus der Menge {wahr, falsch} ersetzt.

[0145] Für die gegebene Eingabe der Turingmaschine finde den Wahrheitswert wie folgt (Navigation im binären Baum):

[0146] Schritt 1: Setze den Zeiger auf dem Basisknoten im Baum B.

[0147] Schritt 2: Lies den Namen der Variable, der in dem Knoten abgespeichert wurde.

[0148] Schritt 3: Stelle fest, welchen Wert die Variable in der zu bearbeitenden Eingabe annimmt.

[0149] Schritt 4: Wenn dieser Wert 'wahr' ist, dann benutze den Pfeil mit der Markierung <VariablenName>= 'wahr', um zum nächsten Knoten K zu gelangen.

[0150] Schritt 5: Wenn dieser Wert 'falsch' ist, dann benutze den Pfeil mit der Markierung <VariablenName>= 'falsch', um zum nächsten Knoten K zu gelangen.

[0151] Schritt 6: Wenn ein Blatt im Baum erreicht wurde, gib den Wert des Blattes aus. Das ist der Wert, der die Ausgabe der Turingmaschine (bezogen auf der gegbenen Eingabe) entspricht.

[0152] Schritt 7: Ansonsten setze B = Unterbaum von B angefangen beim Knoten K.

[0153] Schritt 8: Rufe dich rekursiv auf.

ZITATE ENTHALTEN IN DER BESCHREIBUNG

Diese Liste der vom Anmelder aufgeführten Dokumente wurde automatisiert erzeugt und ist ausschließlich zur besseren Information des Lesers aufgenommen. Die Liste ist nicht Bestandteil der deutschen Patent- bzw. Gebrauchsmusteranmeldung. Das DPMA übernimmt keinerlei Haftung für etwaige Fehler oder Auslassungen.

Zitierte Patentliteratur

- DE 4411514 A1 [0003]
- US 5216593 [0003]
- US 5386557 [0004]
- US 5488722 [0004]
- EP 0545090 A2 [0005]
- EP 0726537 A1 [0006, 0028]
- DE 19725965 C2 [0006, 0008, 0074, 0074, 0098, 0099]

Zitierte Nicht-Patentliteratur

- Hirao, T.: Extension of the relational database semantic processing model, in: IBM Systems Journal, Vol. 29, No. 4, 1990, S. 539 bis 550 [0006]
- Lippert, K.: Heterogene Kooperation, in: ix Multiuser Multitasking Magazin 7/1994, S. 140 bis 147 [0006]
- Date: An introduction to database systems, Vol. I, Fourth Edition, Adison-Wesley, Massachusetts, 1986 [0008]
- Query Graph Model und IBM Research-Report RJ-6367, IBM Almada, San Jose, CA., August 1988 [0027]
- Chang, C. L.; Lee, R. C.: Symbolic logic and mechanical theorem proving, Academic Press, 1977 Edition, 1973 [0035]
- Sterling, Leon; Ehud, Shapiro, The Art of Prolog, MIT Press series in logic programming, 1986 [0035]
- Ullman, Jeffery D., Principles of database & knowledge-base systems, Volume I & II, Computer science press, 1988 [0036]
- Van Emden, Kowalski, The semantics of predicate logic programming languages, Journal of the ACM, Oktober 1976 [0043]
- 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 [0059]
- Bayer, R.: Query Evaluation and Recursion in Deductive Database Systems, Manuscript, March 1985 [0059]
- Lozinskii: Evaluation queries in deductive databases by generating, Proc. Int. Joint Conference on A. I., 1985 [0059]
- Bancilhon, F.; Ramakrishnan, R.: An amateur's introduction to recursive query processing, Proc. of the ACM SIGMOD-SIGACT Conference, Washington D. C., May 1986 [0060]
- 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 [0061]
- Marq-Puchen; Gallausiaux, M.; Jomien: Interfacing Prolog and Relational Database Management Systems, New applications of databases, Gardavin and Gelaube eds. Academic Press, London, 1984 [0061]
- Lozinskii 1985 [0065]
- Vieille, L.; Recursive axioms in deductive databases: The Query-Subquery approach, Proc. First Int. Conf. on expert database systems, Kerschlag ed., Charlston, 1986 [0068]
- Beeri, C.; Ramakrishnan; On the power of magic, Proc. sixth ACM SIGMOD-SIGACT Symp. on principles of database systems, San Diego, CA, March 1987 [0070]
- M. Davis, G. Logemann, and D. Loveland. A Machine Program for Theorem Proving. CACM, 5(7): 394–397, 1962 [0081]
- Balint, A., Fröhlich, A.: Improving stochastic local search for SAT with a new probability distribution. In: SAT-2010. LNCS, Vol. 6175, pp. 10–15 (2010) [0083]
- Ising Modell, siehe: W. Gropp, E. L. Lusk, N. Doss, and A. Skjellum. A High-Performance, Portable Implementation of the MPI Message Passing Interface Standard. Parallel Computing, 22(6): 789–828, 1996 [0086]
- E. Goldberg and Y. Novikov. BerkMin: a fast and robust SATsolver. In Design, Automation and Testing in Europe Conference, pages 142–149, March 2002 [0089]

Patentansprüche

1. Effizientes Verfahren zur logischen Vervollständigung eines deduktiven Kataloges zur allgemeinen Constraintsbehandlung im erweiterten relationalen Datenbankkonzept **dadurch gekennzeichnet,** dass Variablen durch ihre Literale als Musterzeichenketten dargestellt werden, die Mustererscheinungen von Wahrheitswerten in der klassischen Wahrheitstabelle widerspiegeln.

2. Verfahren nach Anspruch 1, **dadurch gekennzeichnet,** dass Musterzeichenketten mittels musterorientierten OR-Operationen zu logischen Musterbäumen zusammengeführt werden.

3. Verfahren nach Anspruch 1, **dadurch gekennzeichnet,** dass logischen Musterbäume mittels musterorientierten AND-Operationen resolviert werden, wobei das Auffinden des gesamten Wahrheitswertes der Klauselmenge dem Resultat dieser Resolution entspricht.

4. Verfahren nach Anspruch 1, **dadurch gekennzeichnet,** dass mit der Resolution ein kombinatorischer Raum generiert wird, der nicht von der klassischen Variablenwertkombinatorik abhängt, sondern von der Aneinanderreihung und Interaktion der Wahrheitsmuster der Variablen in der zu bearbeitenden Formel.

5. Verfahren nach Anspruch 1, **dadurch gekennzeichnet,** dass mittels des kombinatorischen Raums eine kanonische Aufteilung der Klauselmenge in kleineren Klauselmengen durchgeführt wird, von deren jeweiligen Wahrheitswerten alleine der gesamte Endwert abhängt.

6. Verfahren nach Anspruch 1, **dadurch gekennzeichnet,** dass Längen von Musterzeichenketten dazu verwendet werden, geeignete Ordnungskriterien für die durch die Umbenennung der Variablen ermöglichte effiziente Herstellung des kombinatorischen Raums zu liefern.

7. Verfahren nach Anspruch 1, **dadurch gekennzeichnet,** dass der kombinatorische Raum unter Benutzung dieser kanonischen Aufteilung zu einem Entscheidungsbaum konvertiert wird, der mit der klassischen Wahrheitstabelle äquivalent ist, obwohl er nicht die gesamte Wahrheitstabellenkombinatorik beinhaltet.

8. Verfahren nach Anspruch 1, **dadurch gekennzeichnet,** dass negative Literale einer logischen Formel eine Musterzeichenkette der Form 2^i(2^[N – i – 1](1)2^[N – 1 – i](0)) und positive Literale der Form 2^i(2^[N – i – 1](0)2^[N – 1 – i](1)) mit i Indexnummer des jeweiligen Literales, 1 = 'wahr', 0 = 'falsch zugewiesen werden.

9. Verfahren nach Anspruch 2, **dadurch gekennzeichnet,** das OR-Operationen nicht bit-orientiert, sondern musterorientiert erfolgen, wobei Multiplikationsfaktoren von Teilzeichenketten der Muster dazu verwendet werden, ähnliche Operationen nur einmal auszuführen und das Resultat dann mit diesen Faktoren zu versehen, und wobei eine Musterzeichenkette die einen Multiplikationsfaktor f besitzt, die f-malige Verknüpfung mit sich selbst darstellt.

10. Verfahren nach Anspruch 3, **dadurch gekennzeichnet,** dass AND-Operationen nicht bit-orientiert, sondern musterorientiert erfolgen, wobei Multiplikationsfaktoren von Teilzeichenketten der Muster dazu verwendet werden, ähnliche Operationen nur einmal auszuführen und das Resultat dann mit diesen Faktoren zu versehen und wobei eine Musterzeichenkette die einen Multiplikationsfaktor f besitzt, die f-malige Verknüpfung mit sich selbst darstellt.

11. Verfahren nach Anspruch 5, **dadurch gekennzeichnet,** dass eine kanonische Aufteilung der Klauselmenge in kleineren Klauselmengen, von denen die erstere abhängt, dazu verwendet wird, partielle Variablenauswertungen zu kreieren, die die Knoten und Pfade des Baumes bestimmen.

12. Verfahren nach Anspruch 6, **dadurch gekennzeichnet,** dass nur Such- und Navigationsoperationen im Entscheidungsbaum verwendet werden um den Wahrheitswert der Klauselmenge zu finden.

13. Verfahren nach Anspruch 11, **dadurch gekennzeichnet,** dass die Steuerung der Navigation im Entscheidungsbaum durch die aus der Eingabe gewonnenen Variablenwerte unter ausschließlicher Verwendung von Such- und Navigationsoperationen im Entscheidungsbaum zum Auffinden des Wahrheitswertes erfolgt.

Es folgen 24 Seiten Zeichnungen

Anhängende Zeichnungen

Zeichnungen:

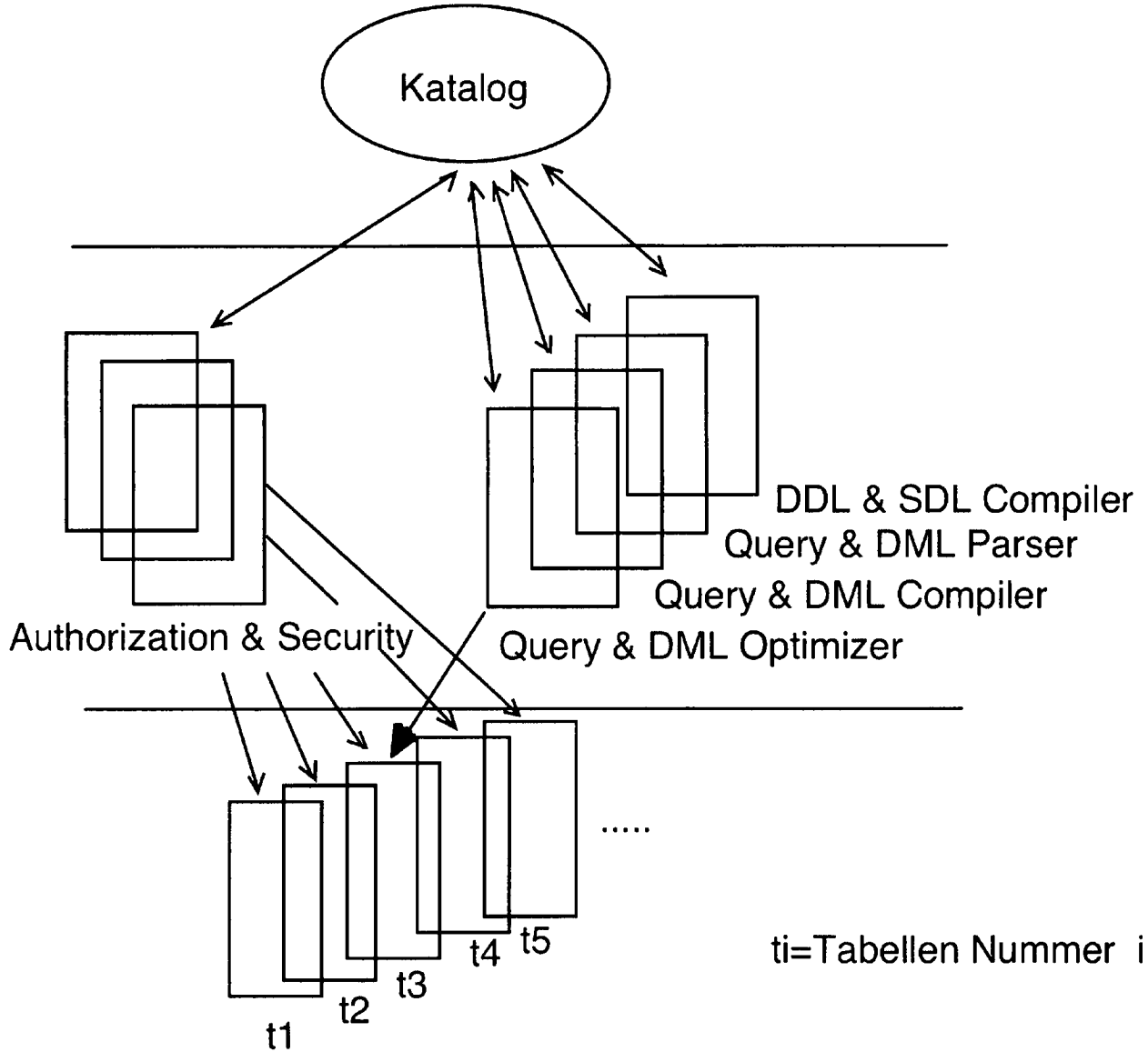


Fig. 1

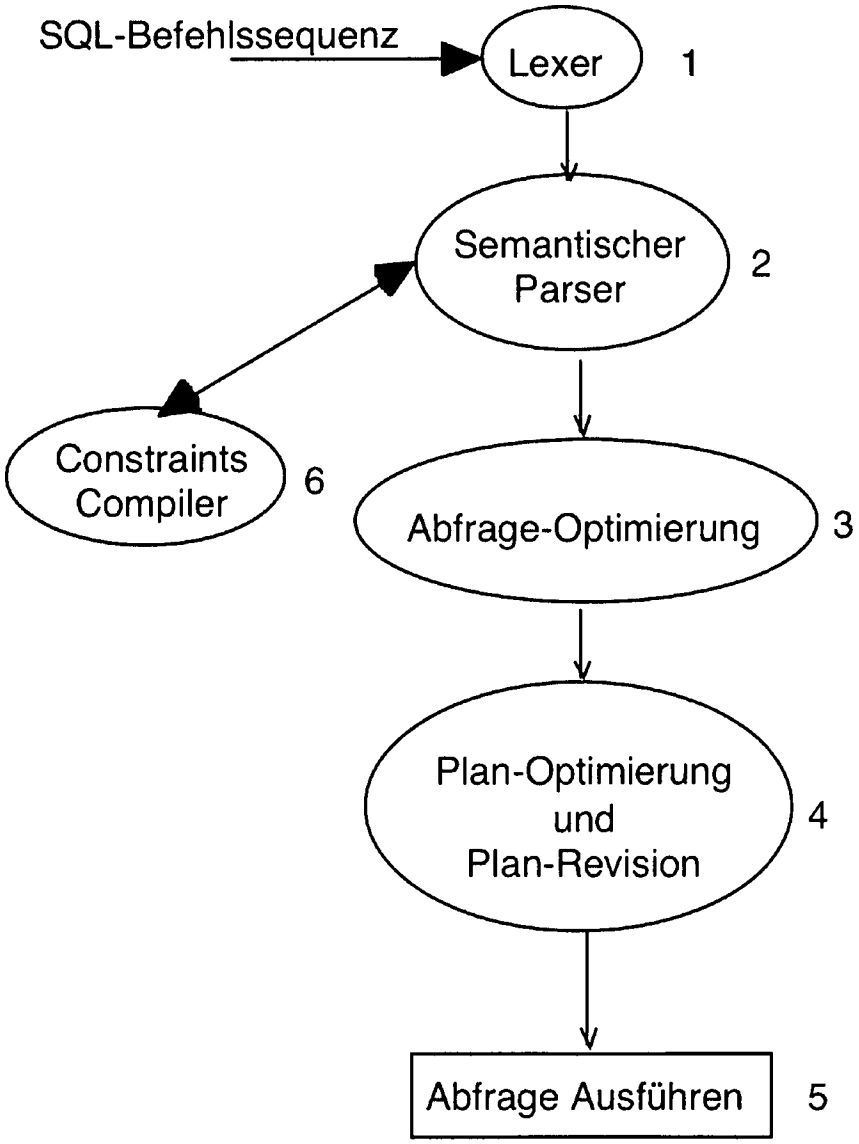
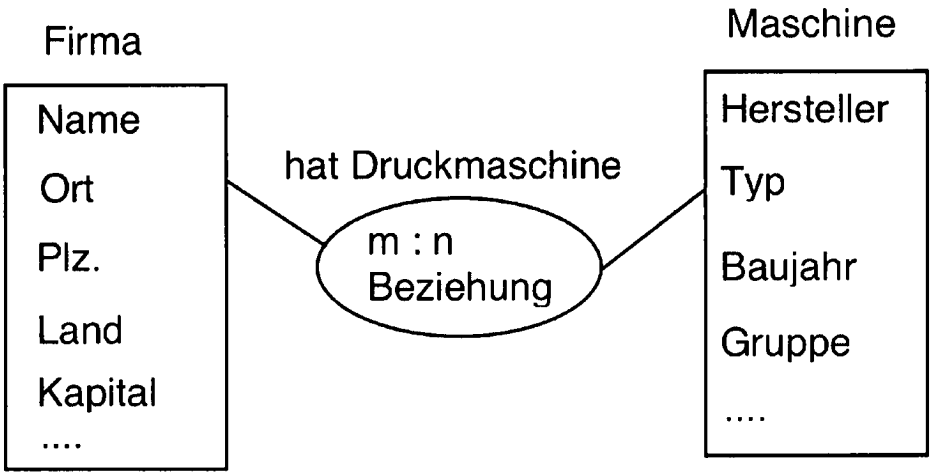

SQL-Befehlssequenz
Lexer
1
Semantischer Parser
2
Constraints Compiler
6
Abfrage-Optimierung
3
Plan-Optimierung und Plan-Revision
4
Abfrage Ausführen
5

Fig. 2



ER-Diagramm

Fig. 3

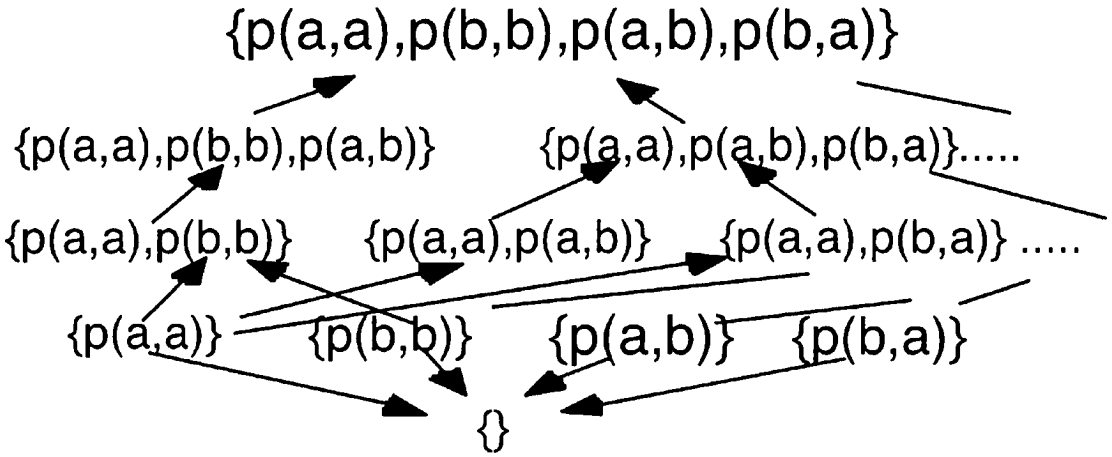


Fig. 4

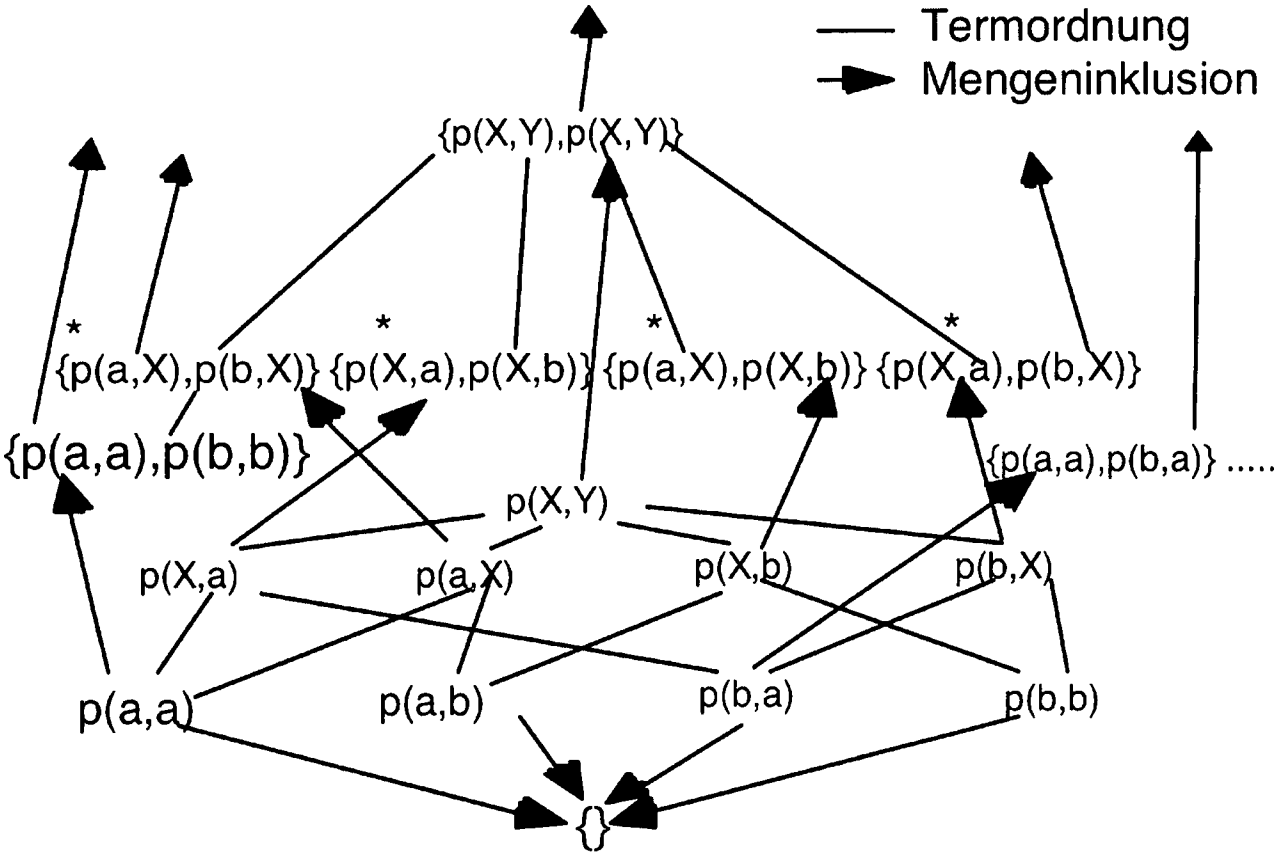

Termordnung
Mengeninklusion
{p(X,Y),p(X,Y)}
{p(a,X),p(b,X)}
{p(X,a),p(X,b)}
{p(a,X),p(X,b)}
{p(X,a),p(b,X)}
{p(a,a),p(b,b)}
{p(a,a),p(b,a)} .....
p(X,Y)
p(X,a)
p(a,X)
p(X,b)
p(b,X)
p(a,a)
p(a,b)
p(b,a)
p(b,b)
{}

Fig. 5

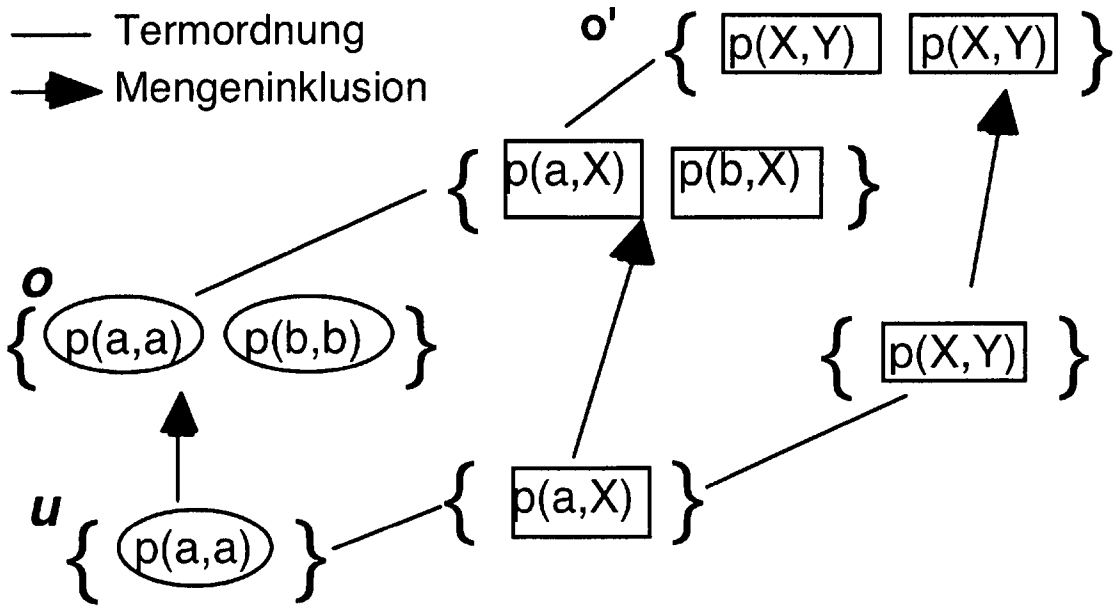

Termordnung
Mengeninklusion
o'
p(X,Y)
p(X,Y)
p(a,X)
p(b,X)
o
p(a,a)
p(b,b)
p(X,Y)
u
p(a,a)
p(a,X)

Fig. 6

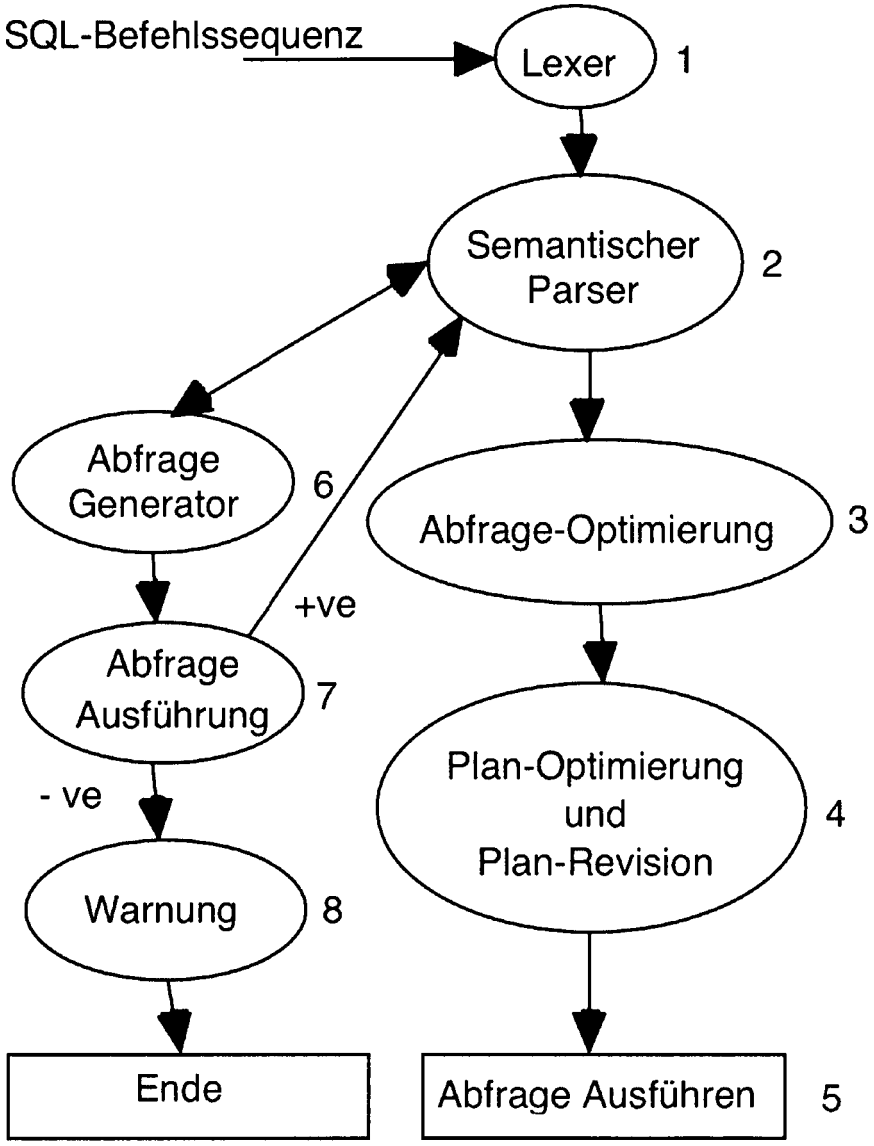
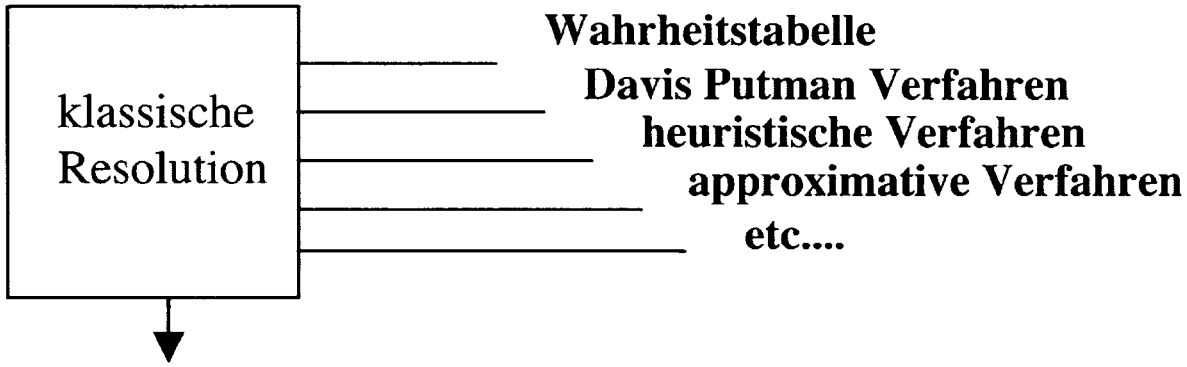

SQL-Befehlssequenz
Lexer
1
Semantischer Parser
2
Abfrage Generator
6
Abfrage-Optimierung
3
+ve
Abfrage Ausführung
7
- ve
Plan-Optimierung und Plan-Revision
4
Warnung
8
Ende
Abfrage Ausführen
5

Fig. 7

A. Klassische Verfahren (beliebige Turingmaschine simuliert in einem logischen System):

TM wird uebersetzt in die Logik zur Klauselmenge (3SAT monoton CNF):

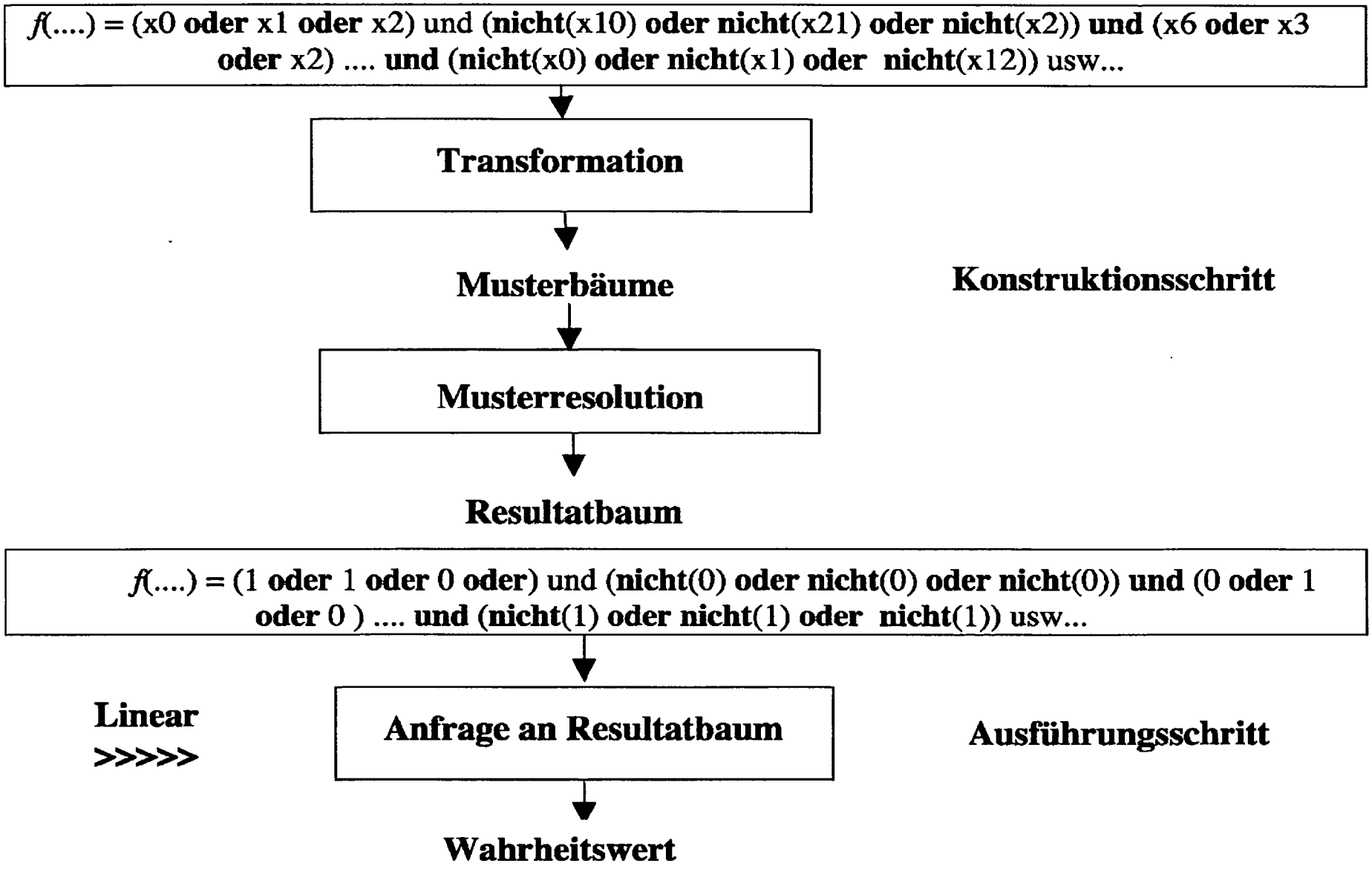
f(....) = (x0 **oder** x1 **oder** x2) und (**nicht**(x10) **oder nicht**(x21) **oder nicht**(x2)) **und** (x6 **oder** x3 **oder** x2) .... **und** (**nicht**(x0) **oder nicht**(x1) **oder nicht**(x12)) usw...



Merkmale:

- Komplexität der Verarbeitung = Konstruktionsaufwand der Kombinatorik + Ausführungsaufwand
- Benutzung der klassischen AND, OR, NOT Operatoren

B. Neues Verfahren:



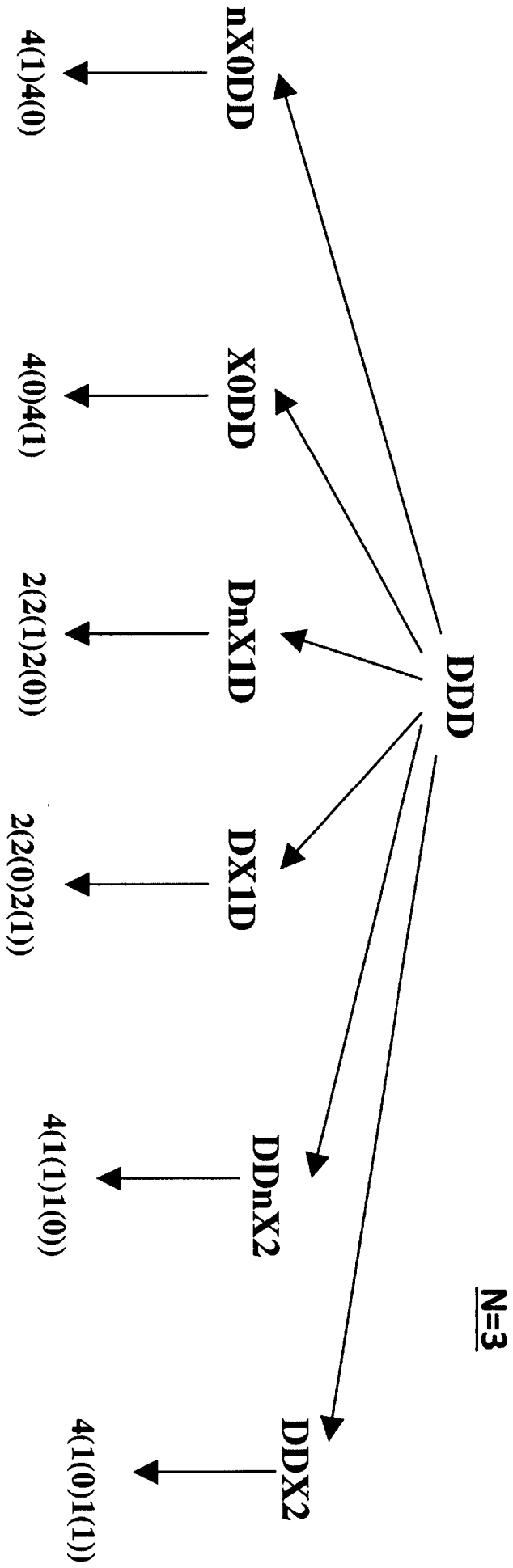
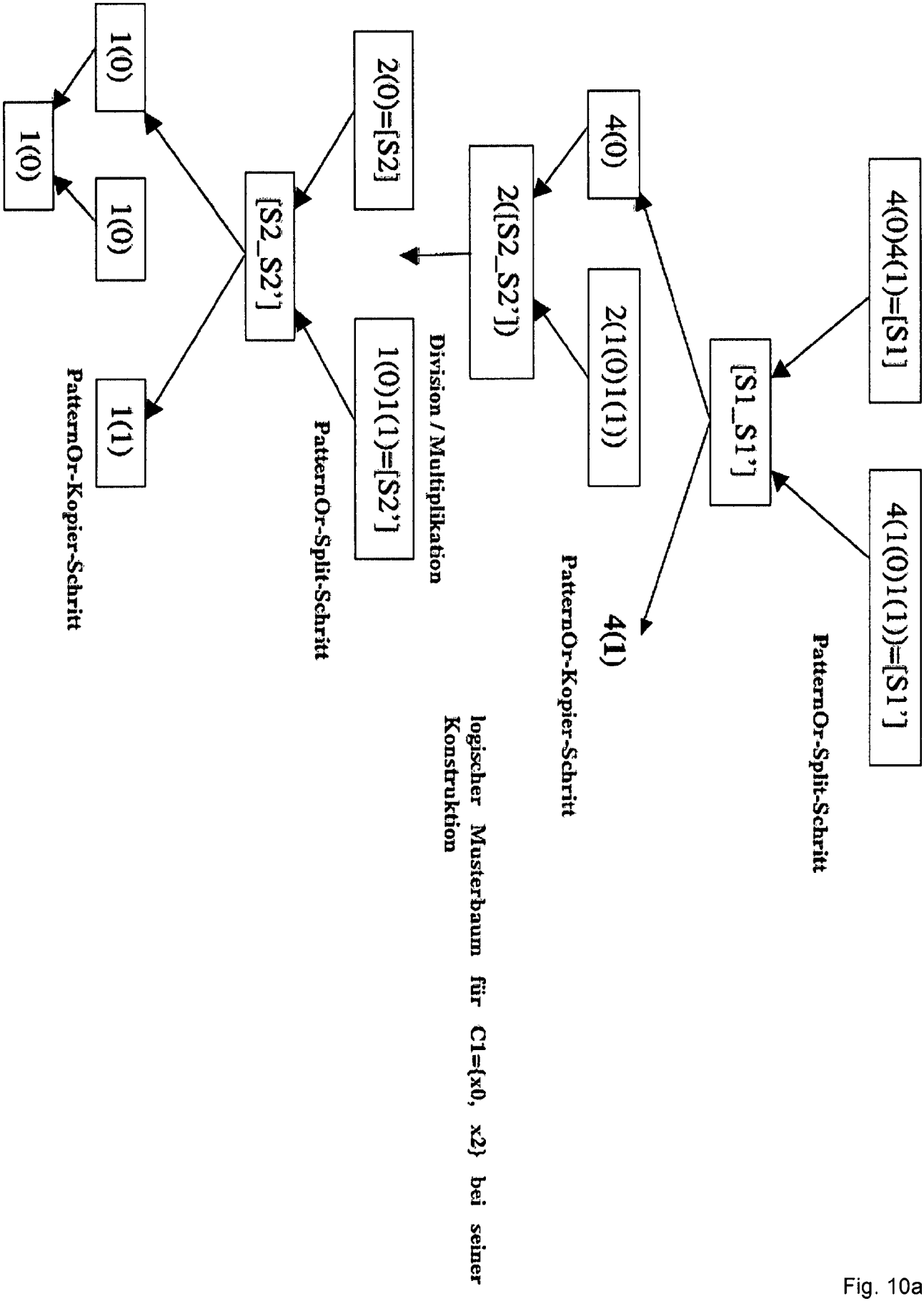

DDD
nX0DD
X0DD
DnX1D
DX1D
DDnX2
DDX2
4(1)4(0)
4(0)4(1)
2(2(1)2(0))
2(2(0)2(1))
4(1(1)1(0))
4(1(0)1(1))
N=3

Fig. 9



logischer Musterbaum für C1={x0, x2} bei seiner Konstruktion

Fig. 10a

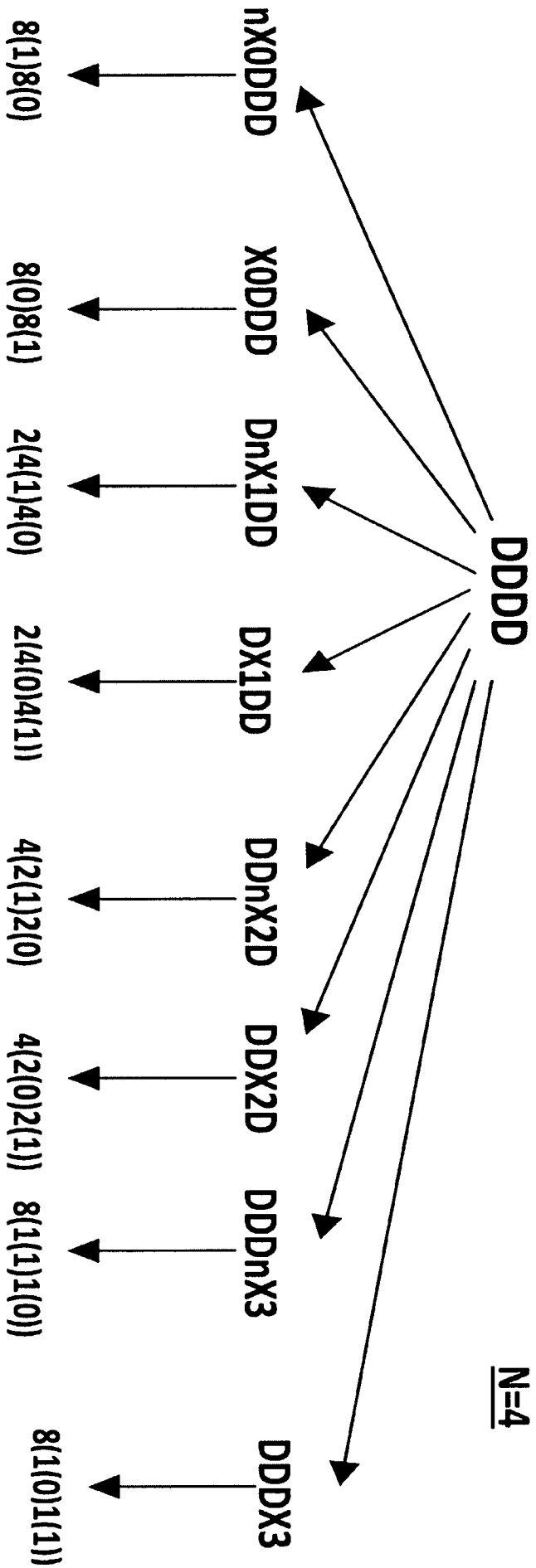

DDDD
nXDDD
XDDD
DnX1DD
DX1DD
DDnX2D
DDX2D
DDDnX3
DDDX3
N=4
8(1)8(0)
8(0)8(1)
2(4)1(4)0)
2(4)0(4)1))
4(2(1)2(0)
4(2(0)2(1))
8(1(1)1(0))
8(1(0)1(1))

Fig. 10b

8(1)8(0)=[S1]
2(4(1)4(0))=[S2
[S1_S1']
PatternOr - SplitSchritt
8(1)
8(0)=[S1']
4(1)4(0)=[S2'
PatternOr - KopierSchritt
[S1'_S2']
4(1)
4(0)
Generierung des Musterbaumes fuer
C1'={nX0,nX1,nX3}, Musterzuweisungsliste
Figur 10, N=4

Fig. 11-1

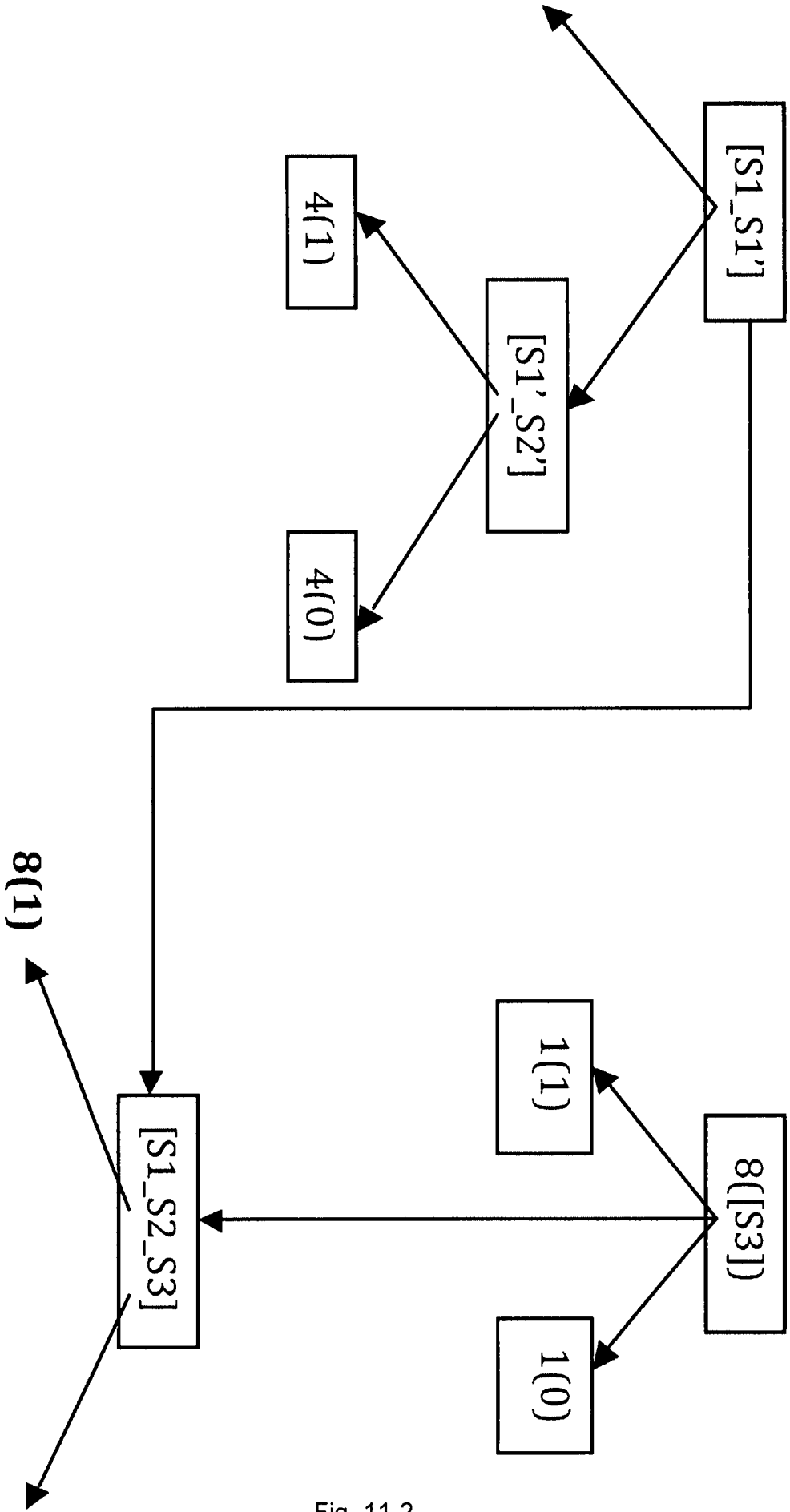


Fig. 11-2

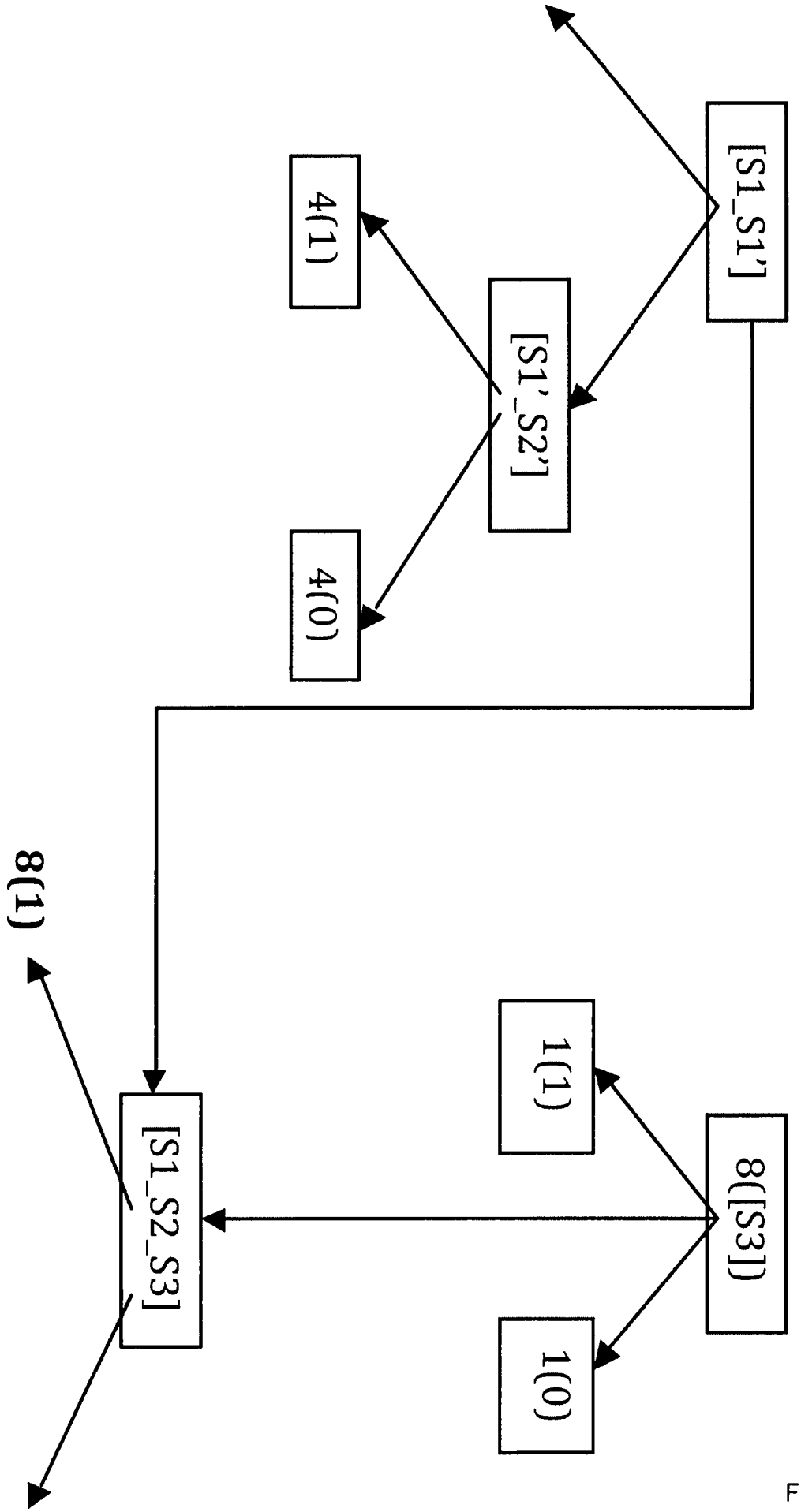


Fig. 11-3

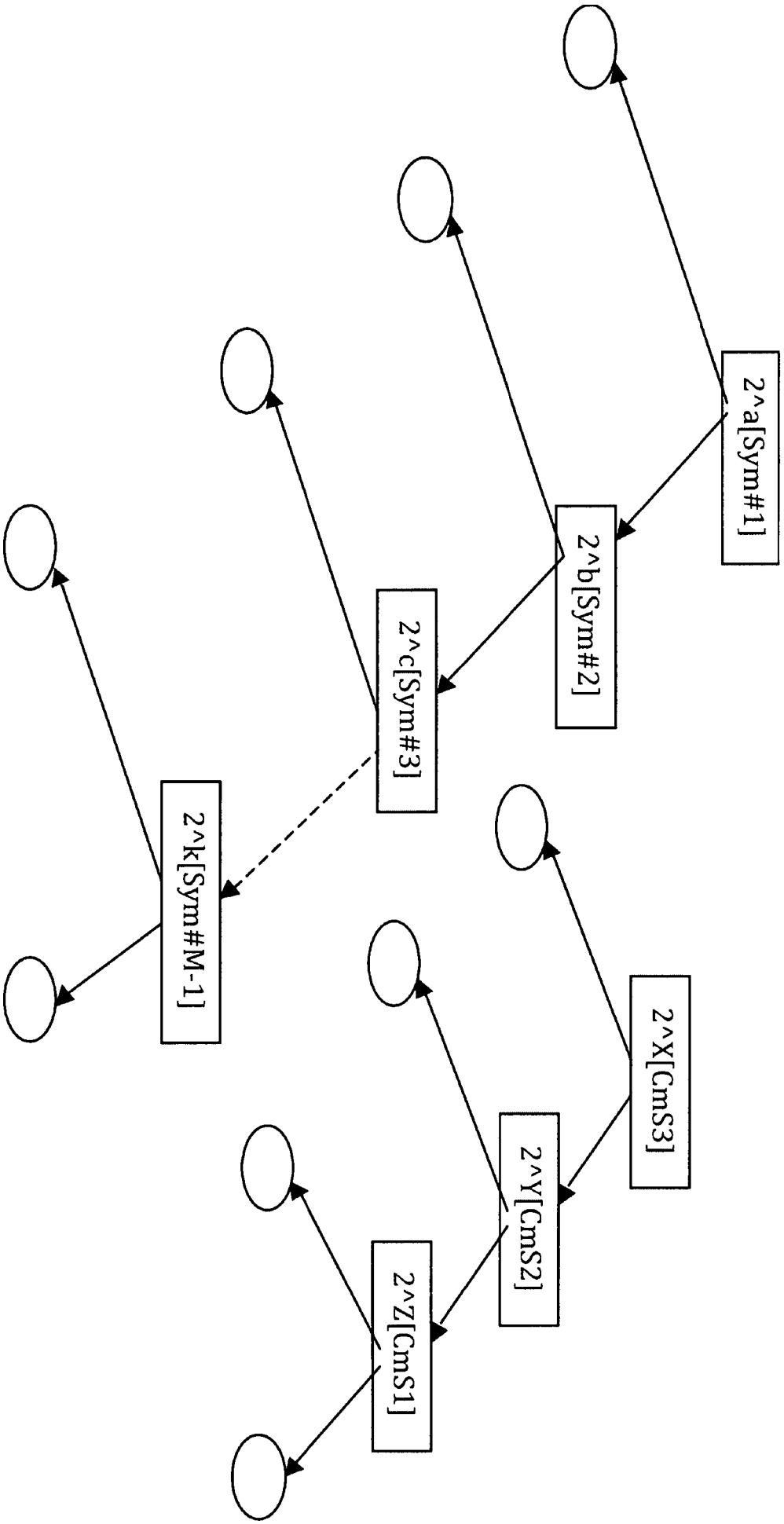


Fig. 12a-1

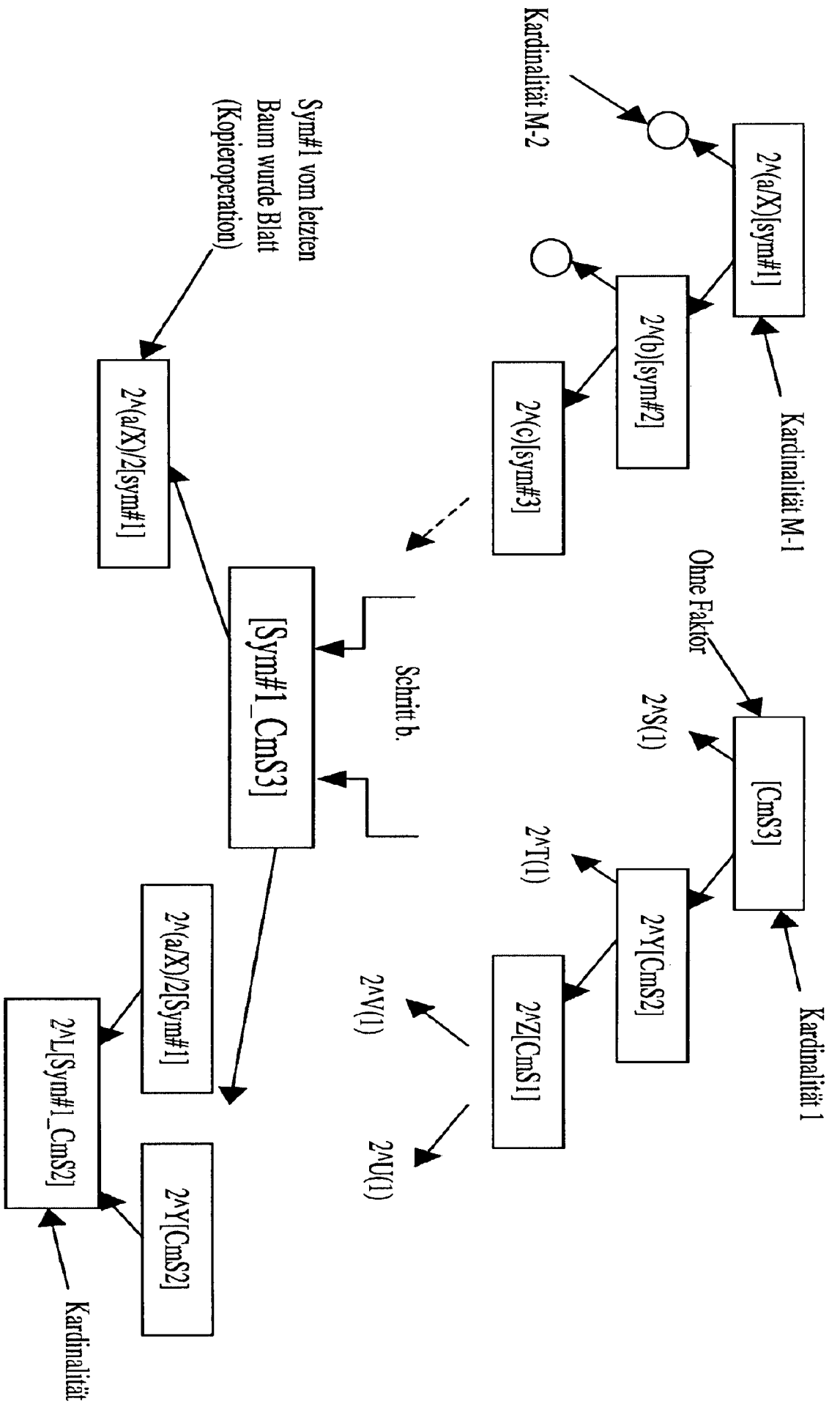

Fall 1, a>>X, d.h. [CmS3]>>[Sym1], Verfahren 4 Schritte nach Division:
2^(a/X)[sym#1]
Kardinalität M-1
Kardinalität M-2
2^(b)[sym#2]
2^(c)[sym#3]
Ohne Faktor
[CmS3]
Kardinalität 1
2^S(1)
2^Y[CmS2]
2^T(1)
2^Z[CmS1]
2^V(1)
2^U(1)
Schritt b.
[Sym#1_CmS3]
Sym#1 vom letzten Baum wurde Blatt (Kopieroperation)
2^(a/X)/2[sym#1]
2^(a/X)/2[Sym#1]
2^L[Sym#1_CmS2]
2^Y[CmS2]
Kardinalität

Fig. 12a-2

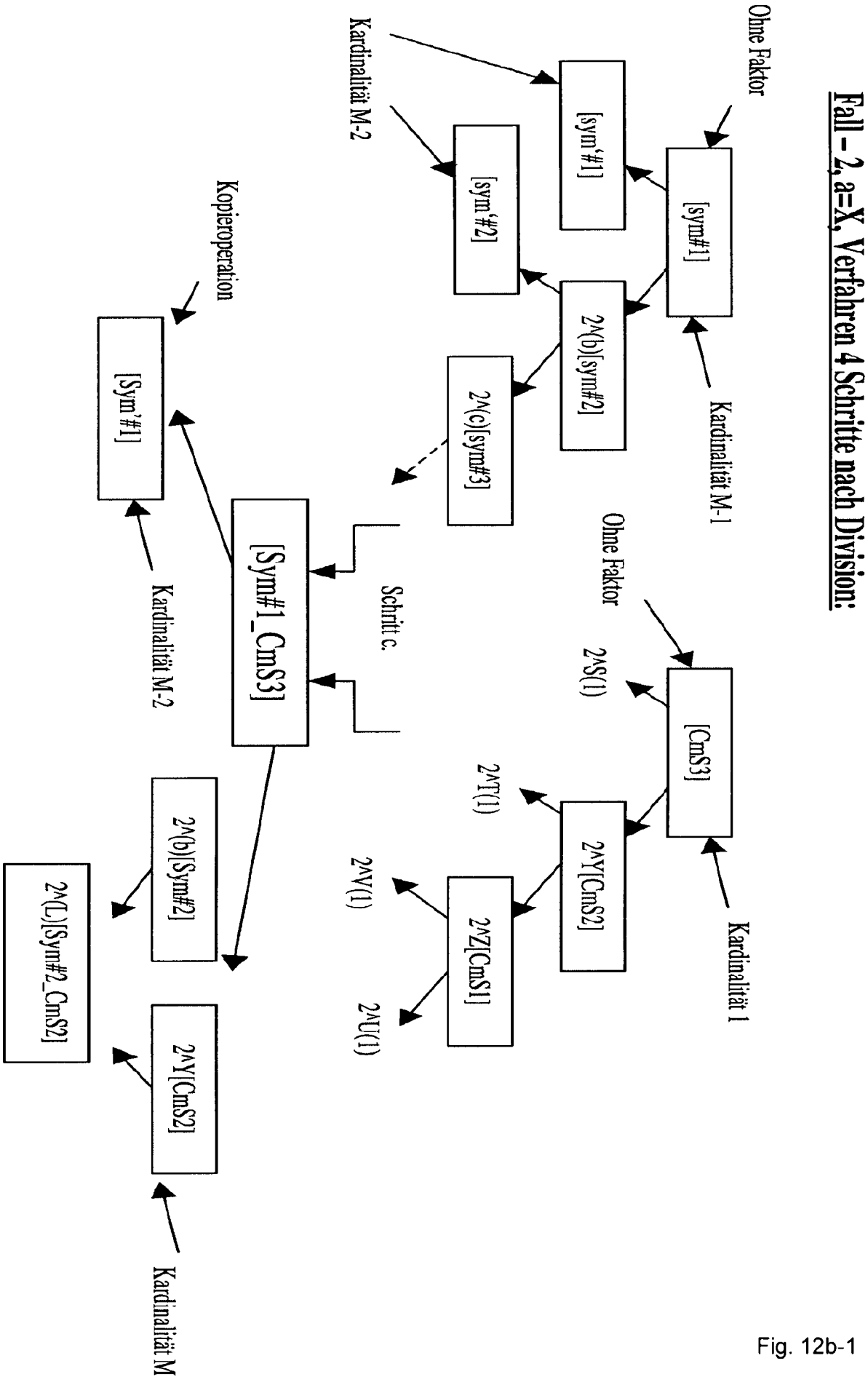

Fall – 2, a=X, Verfahren 4 Schritte nach Division:
Ohne Faktor
[sym#1]
Kardinalität M-1
[sym'#1]
[sym'#2]
Kardinalität M-2
2^(b)[sym#2]
2^(c)[sym#3]
Schritt c.
Ohne Faktor
[CmS3]
2^S(1)
2^T(1)
2^Y[CmS2]
2^V(1)
2^Z[CmS1]
2^U(1)
Kardinalität 1
Kopieroperation
[Sym'#1]
Kardinalität M-2
[Sym#1_CmS3]
2^(b)[Sym#2]
2^(L)[Sym#2_CmS2]
2^Y[CmS2]
Kardinalität M

Fig. 12b-1

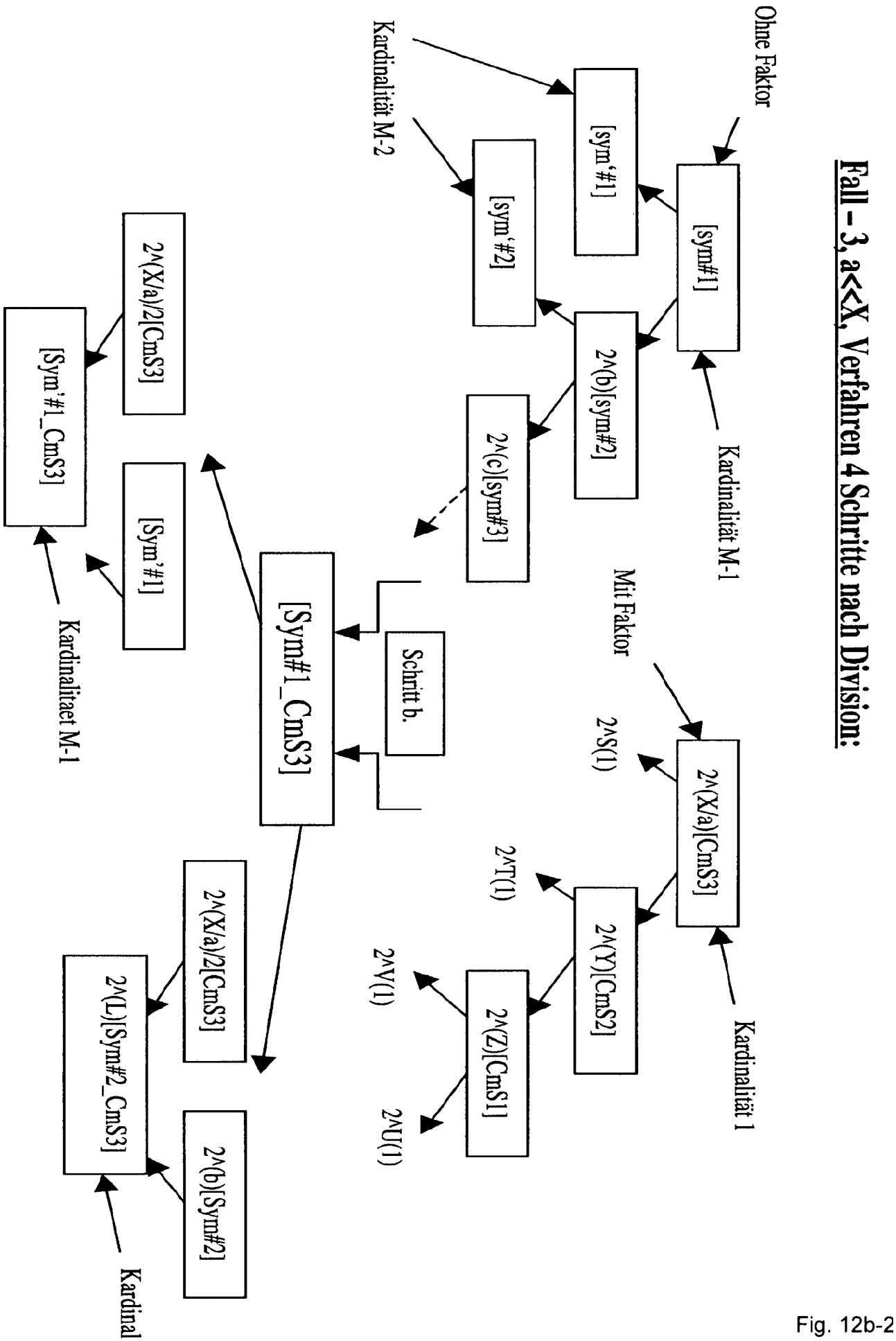

Fall – 3, a<<X, Verfahren 4 Schritte nach Division:
Ohne Faktor
Kardinalität M-2
Kardinalität M-1
[sym'#1]
[sym#1]
[sym'#2]
2^(b)[sym#2]
2^(c)[sym#3]
Schritt b.
[Sym#1_CmS3]
Mit Faktor
2^S(1)
2^T(1)
2^V(1)
2^U(1)
2^(X/a)[CmS3]
2^(Y)[CmS2]
2^(Z)[CmS1]
Kardinalität 1
2^(X/a)/2[CmS3]
[Sym'#1_CmS3]
[Sym'#1]
Kardinalitaet M-1
2^(X/a)/2[CmS3]
2^(L)[Sym#2_CmS3]
2^(b)[Sym#2]
Kardinal

Fig. 12b-2

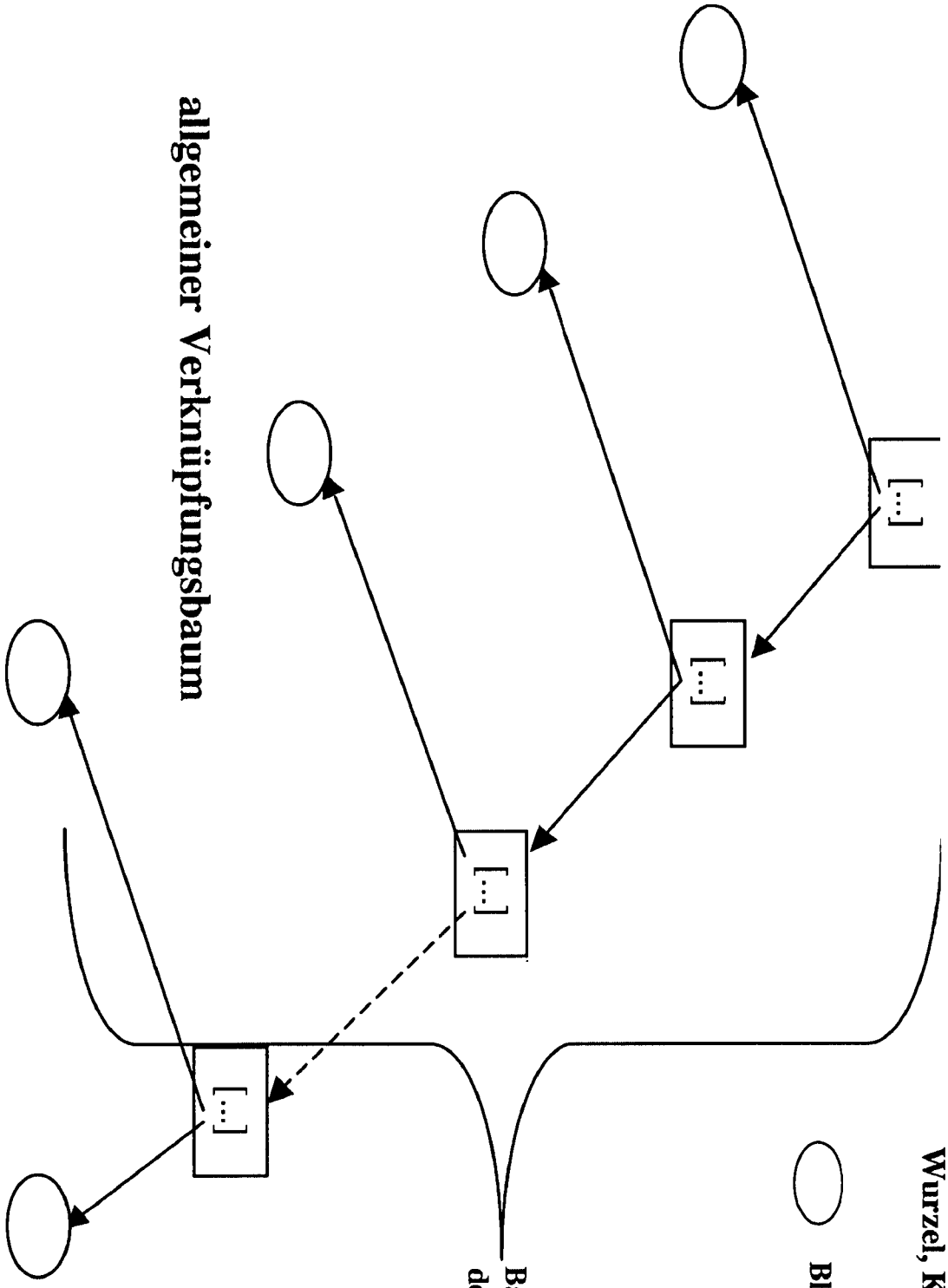
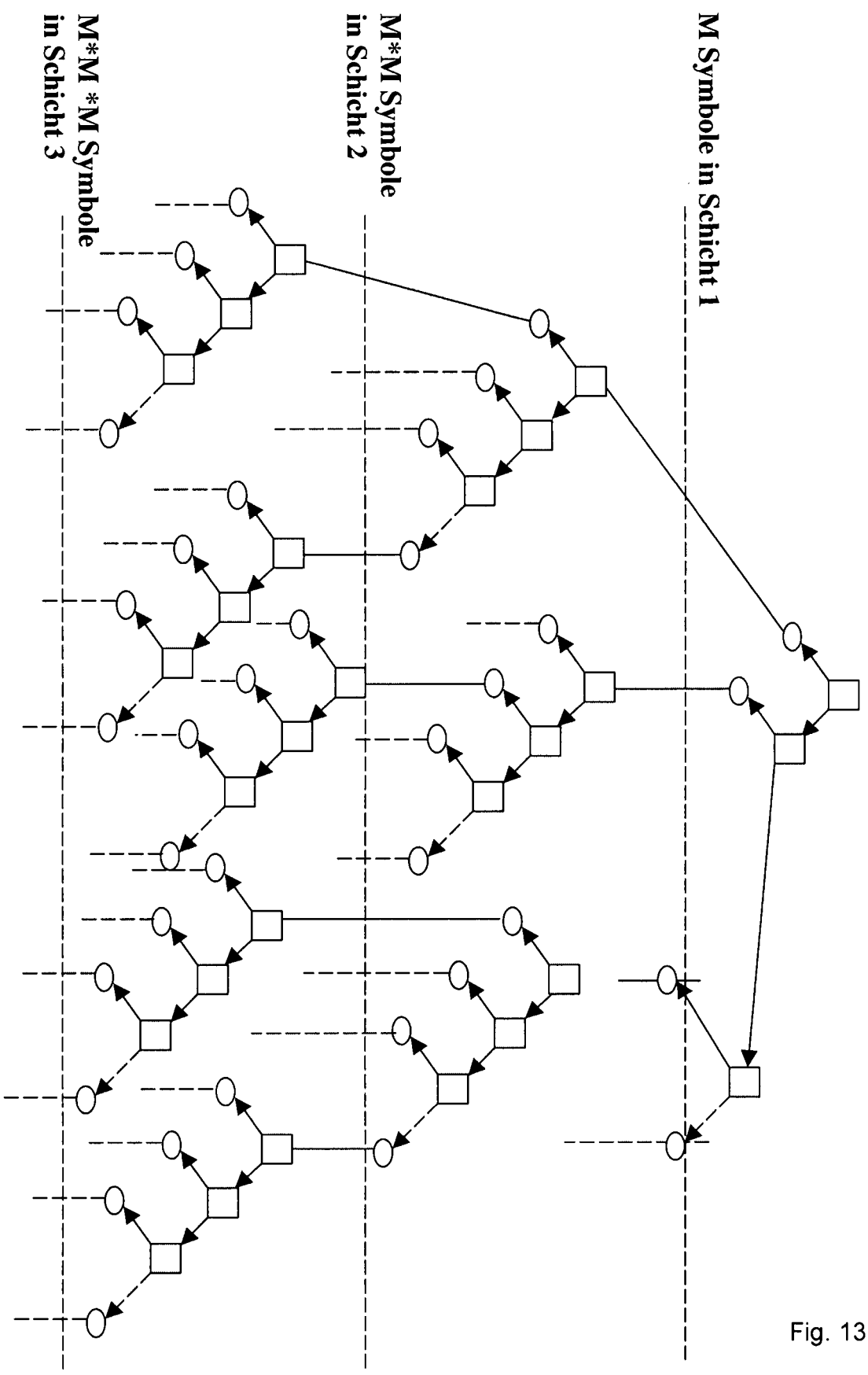

allgemeiner Verknüpfungsbaum
[...]
[...]
[...]
[...]
[..] = Intermediärer Knoten oder Wurzel, Kardinalitaet M
Blatt, Kardinalitaet max M-1
Baumtiefe= O(M), d.h linear in der Anzahl der Klauseln

Fig. 13a



Theoretische Expansion des allgemeinen Verknüpfungsbaumes: Anzahl der totalen Symbole <= M^M

Fig. 13b

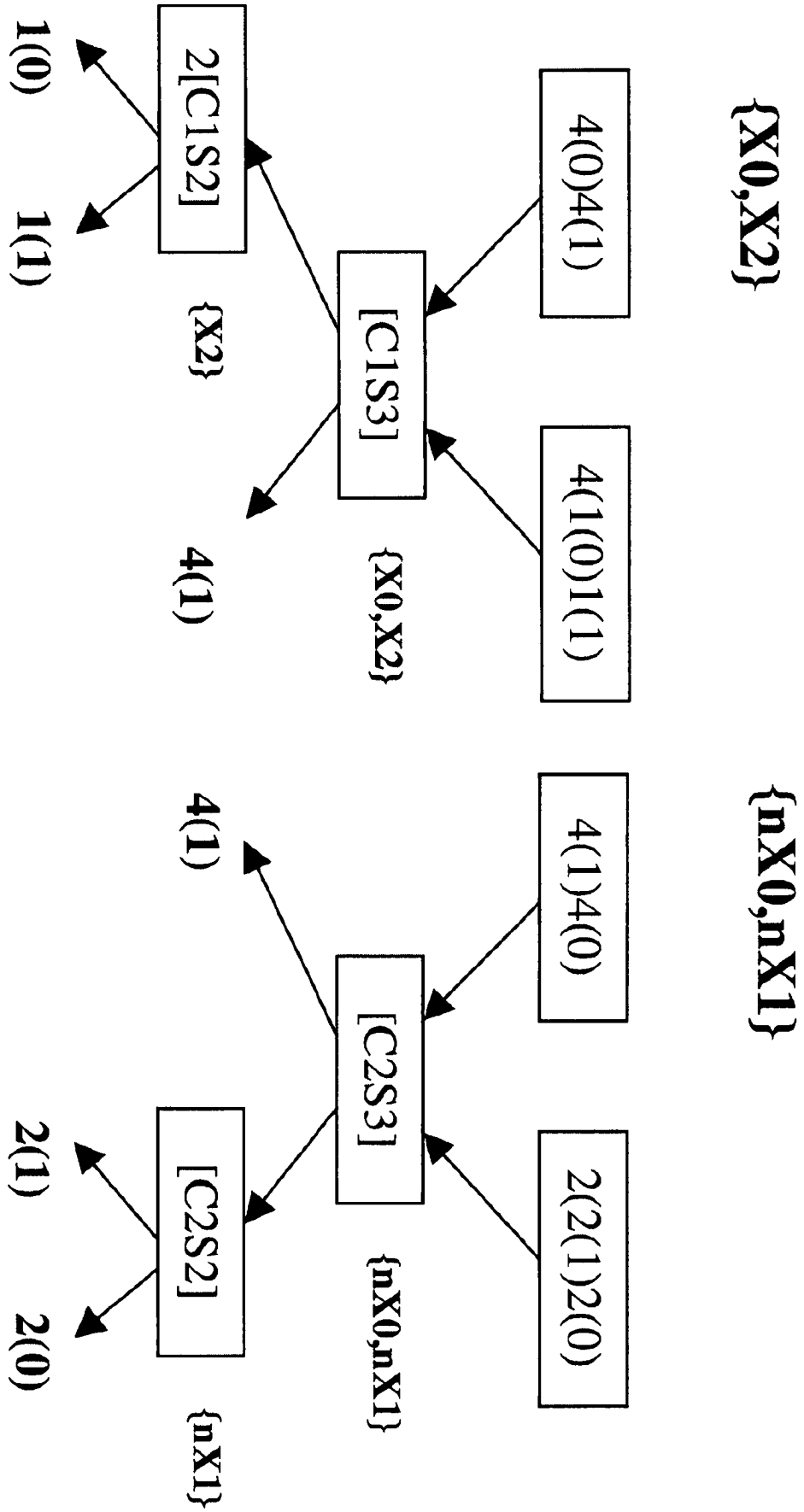

1(0)
1(1)
2[C1S2]
{X2}
4(1)
[C1S3]
{X0,X2}
4(0)4(1)
4(1(0)1(1)
{Z,X2}
2(1)
2(0)
[C2S2]
{nX1}
4(1)
[C2S3]
{nX0,nX1}
4(1)4(0)
2(2(1)2(0)
{nX0,nX1}

Fig. 14a-1

Resolving C={{x0,x2},{nx0,nx1},{x1,x2}},
Musterzuweisungsliste Fig. 9, N=3

➔ PatternOr

➔ PatternAnd

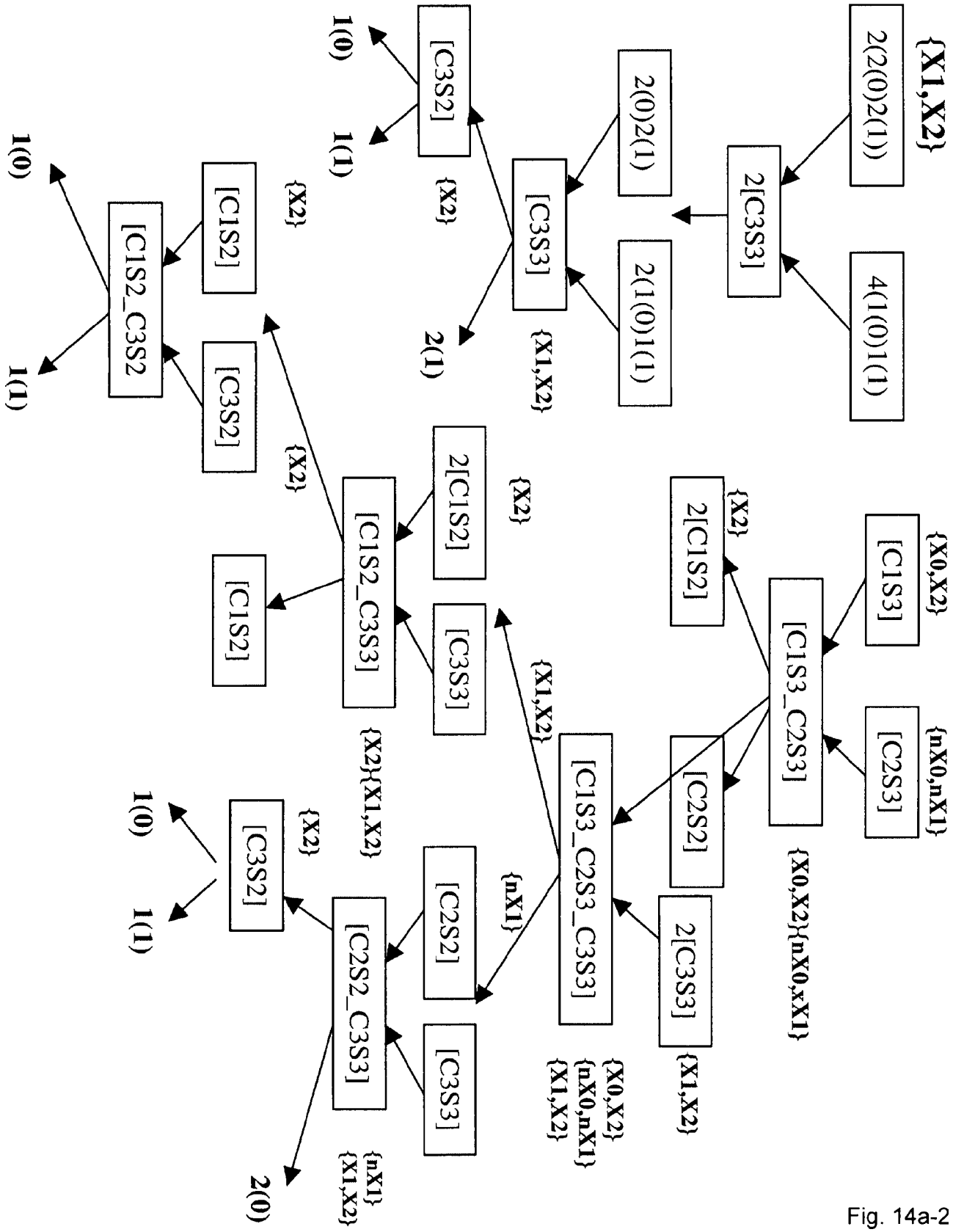


Fig. 14a-2

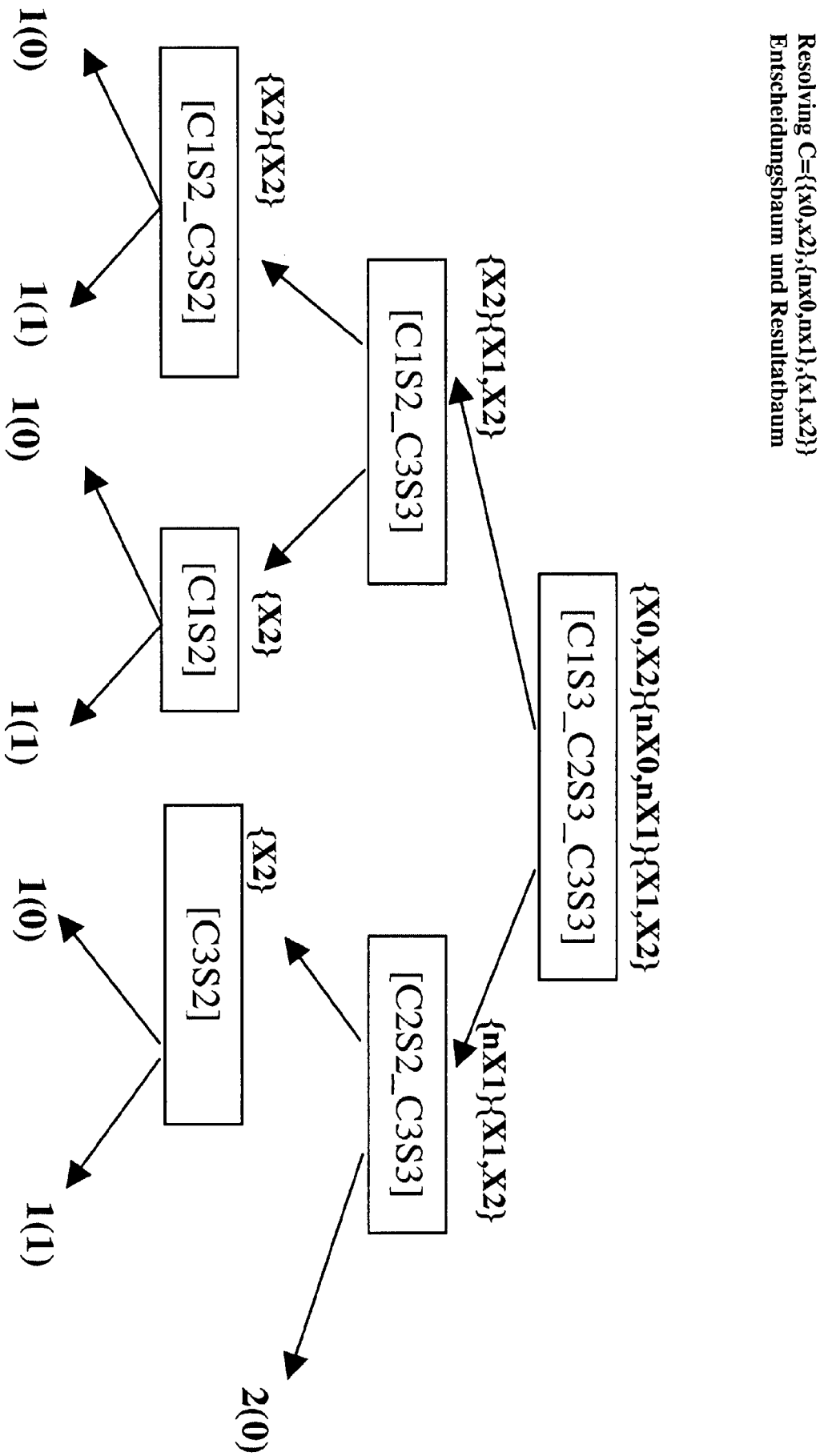

Resolving C={{x0,x2},{nx0,nx1},{x1,x2}}
Entscheidungsbaum und Resultatbaum
[C1S3_C2S3_C3S3]
{X0,X2}{nX0,nX1}{X1,X2}
[C1S2_C3S3]
{X2}{X1,X2}
[C2S2_C3S3]
{nX1}{X1,X2}
[C1S2_C3S2]
{X2}{X2}
[C1S2]
{X2}
[C3S2]
{X2}
1(0)
1(1)
1(0)
1(1)
1(0)
1(1)
2(0)

Fig. 14b-1

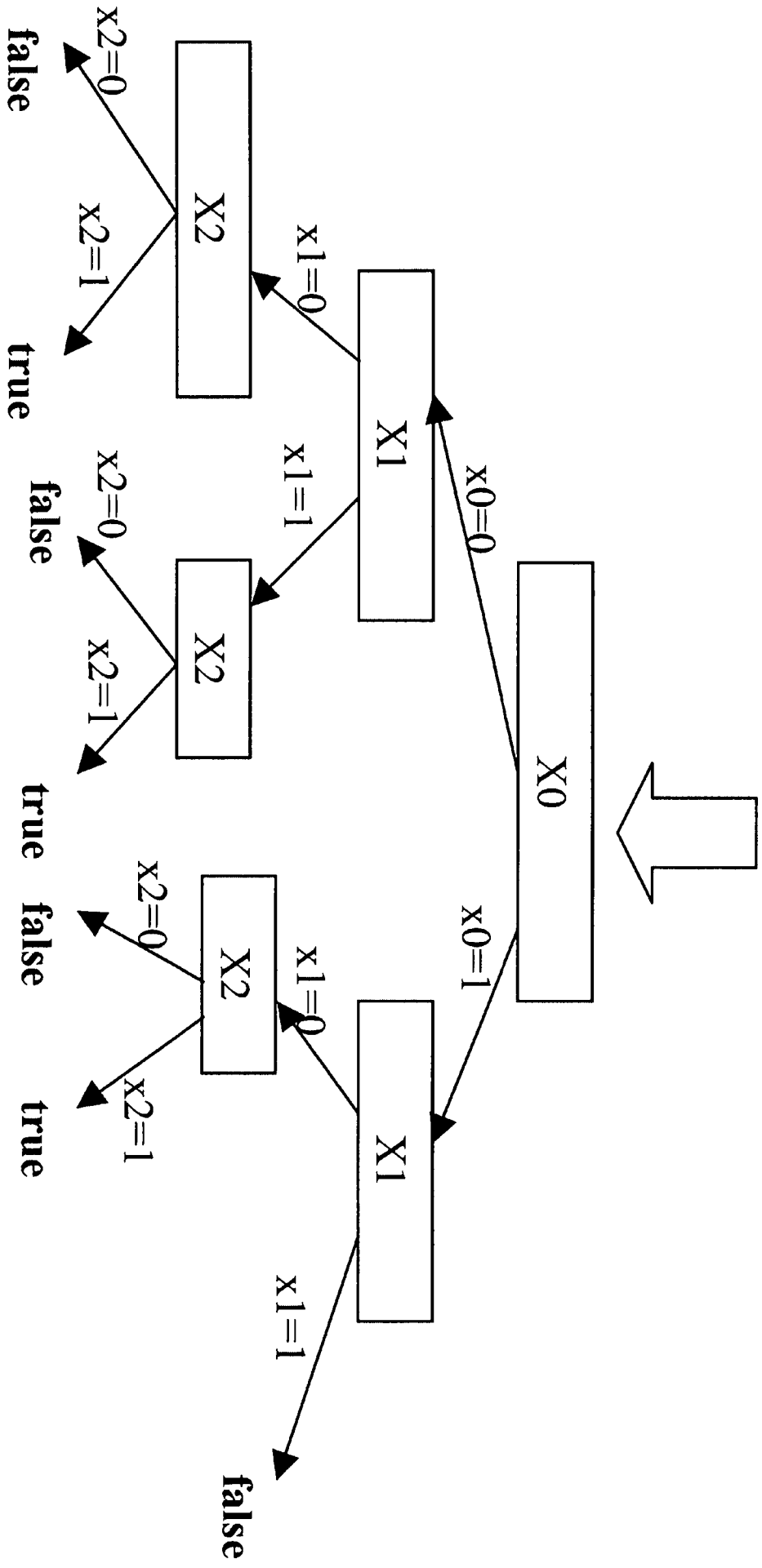

X0
x0=0
x0=1
X1
x1=0
x1=1
X1
x1=0
x1=1
X2
x2=0
x2=1
X2
x2=0
x2=1
X2
x2=0
x2=1
false
true
false
true
false
true
false

Fig. 14b-2

X0	X1	X2	Klassisch	Entscheidungsbaum
0	0	0	**0**	**0**
0	0	1	**1**	**1**
0	1	0	**0**	**0**
0	1	1	**1**	**1**
1	0	0	**0**	**0**
1	0	1	**1**	**1**
1	1	0	**0**	**0**
1	1	1	**0**	**0**

Wahrheitswertsfindung zwischen klassischem Ersetzen in den SAT-Klauseln und dem Benutzen des Entscheidungsbaumes

Fig. 14b-3