Diplomarbeit

Wissensentdeckung in Datenbanken mit dynamischer Anpassung des Hypothesentests

Dirk Münstermann

Diplomarbeit
am Fachbereich Informatik
der Universität Dortmund

30. April 2002

Betreuer:

Prof. Dr. Katharina Morik
Dipl.-Inform. Oliver Ritthoff

Zusammenfassung

Die Wissensentdeckung in Datenbanken (Knowledge Discovery in Databases – KDD) ist ein umfassender Prozeß, der in seinem Analyseschritt Verfahren des maschinellen Lernens einsetzt. Von besonderem Interesse sind hierbei die Lernverfahren der induktiven logischen Programmierung (ILP), die die Entdeckung komplexer Regeln erlauben. Das erste ILP-Lernverfahren, das direkt mit einer kommerziellen Datenbank zusammenarbeitet, ist RDT/DB (Rule Discovery Tool / Database), welches in MOBAL integriert ist. Dieses Lernverfahren ermöglicht das Lernen von Regeln; es ist eine Lernaufgabe, die schwieriger als das Begriffslernen ist.
Aufbauend auf RDT/DB stellen wir in dieser Arbeit die Erweiterung zum Lernverfahren RDT/DM (Rule Discovery Tool/ Data Mining) vor. Dieses Lernverfahren verwendet weitere Möglichkeiten des Datenbank-Systems zur Optimierung des Lernens und ist im Gegensatz zu RDT/DB auf einfache Weise in eigene Anwendungen zu integrieren.

Danksagung

Zunächst möchte ich Katharina Morik für die zahlreichen Gespräche und Diskussion danken, die nicht unwesentlich diese Arbeit beeinflußt haben. Weiterhin danke ich Oliver Ritthoff für hilfreiche Kommentare. Abschließend geht mein besonderer Dank an Christine Baer, die mich besonders in den letzten Wochen in vielfacher Hinsicht unterstützt hat.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

1. Einleitung

Das Forschungsgebiet des *maschinellen Lernens* beschäftigt sich mit der computergestützten Modellierung und Realisierung von Lernphänomenen. Diese Definition (vgl. [MORIK et al. 2000]) beschreibt das Gebiet zwar umfassend, aber nicht eindeutig. Insbesondere ist der zentrale Schlüsselbegriff „Lernen“ nicht klar definiert. In der Literatur zu maschinellem Lernen sind im Laufe der Zeit verschiedene Definitionen des Lernens erstellt worden. Allen Definitionen ist jedoch gemeinsam, daß sie unserem intuitiven Verständnis von Lernen nicht entsprechen, zu unpräzise oder zu allgemein sind. Eine Betrachtung des Lernens als *induktiver Schluß*[1], gemeint ist eine der drei klassischen logischen Schlußformen (Deduktion, Induktion und Abduktion), bietet eine eindeutige, aber nicht sehr umfassende Darstellung. Eine Definition des Lernens, die beide Eigenschaften besitzt, ist zur Zeit nicht bekannt. Allerdings können bestimmte Teilgebiete des Forschungsgebietes „maschinelles Lernen“ durch die explizite Beschreibung der *Lernaufgabe* definiert werden. Diese einzelnen Lernaufgaben können sehr präzise festgelegt werden und erlauben dadurch zu entscheiden, ob ein bestimmtes Phänomen ein Lernphänomen ist oder nicht. In [MORIK et al. 2000] wird allgemein festgelegt, wie solche Lernaufgaben zu beschreiben sind. Insbesondere muß jede Lernaufgabe eindeutig spezifizieren, wann ein System erfolgreich gelernt hat. Ein klassisches Beispiel für eine Lernaufgabe ist das Funktionslernen aus Beispielen, das bei einer Zielmenge, die genau zwei diskrete Werte enthält, auch als Begriffslernen aus Beispielen bezeichnet wird. Die Repräsentation von Begriffen erfolgt durch Attribute oder mit Hilfe der (eingeschränkten) Prädikatenlogik. In der Prädikatenlogik können wir Relationen und ihre Verknüpfungen darstellen. Die induktive logische Programmierung (ILP) beschäftigt sich mit dem Lernen von logischen Programmen, die eine eingeschränkte Prädikatenlogik darstellen. Eine Besonderheit der ILP ist die Einbeziehung von Hintergrundwissen beim Lernen. Die ILP, wie auch andere Lernaufgaben, kann man sich als eine Suche nach geeigneten Hypothesen innerhalb des Hypothesenraumes vorstellen. Dieser Ansatz wurde von Mitchell in [MITCHELL 1982] beschrieben. Bei der ILP wird eine geordnete Suche durch den Hypothesenraum durchgeführt. Die Ordnung zwischen den einzelnen Hypothesen wird durch eine Generalisierungsbeziehung (h_1 ist genereller als h_2) beschrieben. Die Suche durch den Hypothesenraum erfolgt schrittweise von den speziellen zu den generellen Hypothesen (bottom-up) oder umgekehrt von den generellen zu den speziellen Hypothesen (top-down). Zusätzlich erlaubt die Generalisie-

[1]Die Induktion schließt vom Besonderen auf das Allgemeine. Ein typisches Beispiel ist: Aus „Sokartes ist ein Mensch.“ und „Sokrates ist sterblich.“ schließt „Alle Menschen sind sterblich.“

rungsbeziehung ein sicheres Beschneiden des Hypothesenraumes. Ein Lernverfahren für die Lernaufgabe der induktiven logischen Programmierung ist RDT (Rule Discovery Tool) [KIETZ und WROBEL 1992]. RDT ist in die Werkbank MOBAL [MORIK et al. 1993] integriert. Dieses Lernverfahren ist ein zentraler Bestandteil dieser Arbeit. Insbesondere werden wir die Anwendung dieses Lernverfahrens auf Daten in einer Datenbank betrachten.

In den Anfängen der elektronischen Datenverarbeitung verwalteten die einzelnen Applikationen ihre Dateien selbständig. Die Daten waren den Anwendungen direkt zugeordnet. Bei der Weiterverarbeitung oder der Verwendung der gleichen Daten durch verschiedene Programme wurden diese jeweils kopiert. Dadurch sind in der Regel die gleichen Daten mehrmals gespeichert worden (redundante Datenspeicherung). Außerdem konnten zu einem Zeitpunkt unterschiedliche Versionen der Daten im System vorhanden sein. Bedingt durch die verteilte Datenhaltung konnten Anwendungen, die aufeinander aufbauen, nur sequentiell nacheinander ausgeführt werden und arbeiteten zudem auf veralteten Daten. Die Menge der verwalteten Daten wuchs ständig an, ebenso wurden die Datenstrukturen komplexer. Der Wunsch nach einem „zentralisierten Datenverwaltungsprogramm“ kam auf, woraufhin die ersten Ansätze eines Datenbank-Systems[2]entwickelt wurden. Im Laufe der Zeit wurden aus diesen einfachen Programmen die heute bekannten Datenbank-Systeme. Diese modernen Systeme verwalten neben den Daten auch die Beziehungen zwischen den Daten und übernehmen weitere Tätigkeiten wie die Überwachung vor unberechtigten Zugriffen, die Einhaltung der Datensicherheit oder die Kontrolle des gleichzeitigen Zugriffs mehrerer Anwendungen. Durch die Bereitstellung der Daten an einer zentralen Stelle vereinfacht sich die Anwendungsentwicklung, da die Verwaltung der Daten entfällt und der Zugriff auf die Daten durch feste Schnittstellen spezifiziert ist. Eine Änderung oder Neuimplementierung einer Anwendung ist somit unabhängig von der Datenverwaltung. Die Datenbeschreibung erfolgt einmalig in dem Datenbank-System, und eine Änderung der Datenstruktur bleibt für alle Anwendungen unproblematisch. Die vorherrschenden Datenbank-Systeme beruhen derzeit auf dem relationalen Modell (siehe [CODD 1970, CODD 1990]) und werden als *relationale Datenbanken* bezeichnet. Allerdings erfüllt kein heutiges Datenbank-System die von Codd aufgestellten Kriterien an ein relationales Datenbank-System vollständig.

Die interaktive Analyse von angesammelten Datenbeständen (z.B. in Datenbanken) ist in den letzten Jahren eine wichtige praktische Anwendung des maschinellen Lernens geworden. Als Stichworte sind in diesem Rahmen Data Mining und Knowledge Discovery in Databases (KDD) zu nennen. Die Wissensentdeckung in Datenbanken wird als umfassender Prozeß verstanden, der von der ersten Betrachtung der Domäne bis zur Nutzung der Ergebnisse reicht. Data Mining ist in diesem Prozeß der eigentliche Analyseschritt, in dem Hypothesen gesucht und bewertet werden. In diesem Analyseschritt werden verschiedene Verfahren des maschinellen Lernens eingesetzt. Die Verfahren der induktiven logischen Programmierung sind

[2]Ein Datenbank-System besteht aus der Datenbank und einem Datenbank-Management-System, das die Verwaltung der Daten der Datenbank durchführt.

im Data Mining von besonderem Interesse, da sie die Entdeckung komplexer Regeln erlauben. Das erste ILP-Lernverfahren, das direkt mit einer kommerziellen (relationalen) Datenbank verbunden wurde, ist RDT/DB, eine Weiterentwicklung von RDT (vgl. [Brockhausen und Morik 1996]). Bedingt durch die Anbindung sind die Prädikatengenerierung und der Hypothesentest für die Interaktion mit der Datenbank modifiziert worden. Aus dem Datenbankschema werden mit Hilfe von Abbildungsvorschriften Prädikate generiert, die dann bei der Hypothesengenerierung verwendet werden. Für den Hypothesentest wird die generierte Hypothese in eine SQL-Anfrage umgewandelt und diese zur Auswertung an die Datenbank weitergegeben. Die Datenbank wird bei diesem Verfahren ausschließlich gelesen. Eine Umrepräsentierung, Veränderung oder Erweiterung von Daten in der Datenbank erfolgt nicht.

Die in dem Lernverfahren RDT/DB gewählte Verwendung der Datenbank schöpft die Möglichkeiten eines Datenbank-Systems nicht vollständig aus. Wir werden im Rahmen dieser Arbeit weitere Aspekte des Datenbank-Systems in das Lernverfahren RDT/DM (Rule Discovery Tool / Data Mining), eine Erweiterung aus dem Lernverfahren RDT/DB, einbinden.

Das „data dictionary“ eines Datenbank-Systems beinhaltet Informationen zu den Daten, den Datentypen und den Beziehungen zwischen den Daten. Insbesondere sind die logischen Verbindungen – gemeint ist damit die Referenzierung von Fremdschlüsseln auf Primärschlüssel – zwischen Daten dort gespeichert. Weiterhin bietet das Datenbank-Management-System durch die Erzeugung von Views und ohne Veränderung der ursprünglichen Daten in der Datenbank eine Möglichkeit, neue Sichten auf die Daten zu definieren. Diese Views benötigen in der Datenbank nur einen geringen zusätzlichen Speicherplatz (es werden keine Daten kopiert), bieten aber einen schnelleren Zugriff auf ausgewählte Daten sowie eine Reduzierung der Komplexität von Zugriffen auf die Datenbank.

In dem Lernverfahren RDT/DM werden wir sowohl die zusätzlichen Informationen zu den logischen Verbindungen zwischen den Daten in der Datenbank als auch Views verwenden. Durch die logischen Verbindungen reduziert sich die Anzahl der Hypothesen bei der Hypothesengenerierung und dadurch der betrachtete Hypothesenraum. Bedingt durch die Views wird im Gegensatz zu dem Lernverfahren RDT/DB schreibend auf das Datenbank-System zugegriffen. Die Views werden in dem Lernverfahren RDT/DM in unterschiedlichen Bereichen eingesetzt. Einerseits verwenden wir sie bei der Prädikatengenerierung und andererseits im Rahmen des Hypothesentests. Die Verwendung im Hypothesentest auf der Datenbank führt dazu, daß eine dynamische Anpassung der SQL-Anfrage, die zum Test der generierten Hypothese erzeugt wird, notwendig ist.

Neben diesen konzeptionellen Erweiterungen des Lernverfahrens steht die einfache Verwendbarkeit im Vordergrund. Die Lernverfahren RDT und RDT/DB sind beide in der Werkbank MOBAL integriert. Eine direkte Nutzung dieser Lernverfahren durch eigene Anwendungen ist problematisch. Deshalb wurde das Lernverfahren RDT/DM als eigenständige Applikation konzipiert, die neben einer direkten Nutzung auch durch eine Programmierschnittstelle in eigene Anwendungen integriert werden kann.

Die vorliegende Arbeit gliedert sich grob in drei Bereiche:

- Erarbeitung der Grundlagen zu Datenbanken (Kapitel 2) und der Wissensentdeckung in Datenbanken durch Regellernen (Kapitel 3),
- Einführung der konzeptionellen Erweiterungen (Kapitel 4) und eine Übersicht der relevanten Implementierungsdetails (Kapitel 5) sowie
- Evaluation der durchgeführten Versuche (Kapitel 6).

In Kapitel 2 wenden wir uns den Datenbanken zu. Wir führen kurz die für den weiteren Verlauf der Arbeit notwendigen Begriffe und Konzepte ein. Insbesondere stellen wir das relationale Modell nach [Codd 1990] vor, definieren die relationale Datenbank und beschreiben die Standard Query Language (SQL). Zusätzlich führen wir eine Darstellung von Relationen und logische Verbindungen zwischen diesen ein. Im Anschluß betrachten wir die herstellerunabhängige Datenbankschnittstelle JDBC.

Kapitel 3 beschäftigt sich mit der Wissensentdeckung in Datenbanken durch Regellernen. Wir definieren dort zunächst die Lernaufgabe des Regellernens und legen die grundlegenden Begriffe fest. Im weiteren Verlauf beschreiben wir den Regellerner RDT (Rule Discovery Tool), um danach auf die Kopplung von RDT mit einer relationalen Datenbank einzugehen. Diese Erweiterung von RDT wird mit RDT/DB bezeichnet.

Der Schwerpunkt unserer Arbeit liegt auf den Kapiteln 4 und 5. In Kapitel 4 führen wir die konzeptionellen Erweiterungen im Vergleich zum Lernverfahren RDT/DB ein, die zu dem neuen Lernverfahren RDT/DM (Rule Discovery Tool / Data Mining) führen. Dazu beziehen wir die logischen Verbindungen zwischen Relationen der Datenbank in den Algorithmus ein und nutzen die durch das Datenbank-Management-System gegebene Funktionalität zur Erzeugung von Views. Anschließend erfolgt in Kapitel 5 eine Beschreibung der Konzepte der Implementierung.

Der dritte Teilbereich ist durch Kapitel 6 abgedeckt. Dort beschreiben wir zunächst die von uns benutzte Versuchsumgebung und diskutieren im Anschluß daran eine Auswahl der durchgeführten Versuche und die dadurch erlangten Ergebnisse.

Den Abschluß dieser Arbeit bildet Kapitel 7 mit einer Zusammenfassung der Inhalte und einem Resümee zu den Ergebissen dieser Arbeit. Desweiteren stellen wir in einem Ausblick einige Ideen für die Weiterentwicklung des Algorithmus vor und diskutieren weitere Untersuchungsmöglichkeiten.

2. Datenbanken

In diesen Kapitel werden wir die grundlegende Begriffswelt zu Datenbanken beschreiben. Dazu führen wir in Abschnitt 2.1 das relationale Modell ein, um damit ein relationales Datenbanksystem zu definieren. In Abschnitt 2.3 beschreiben wir für ein solches System die Datenbeschreibung- und Datenmanipulationssprache SQL. Zum Schluß dieses Kapitels führen wir in Abschnitt 2.4 JDBC ein. Dabei handelt es sich um eine datenbankunabhängige Schnittstelle zum Zugriff von Java auf eine beliebige relationale Datenbank.

(Relationale) Datenbanken sind in der Industrie seit einiger Zeit die dominierende Speicherungsmethode von strukturierten Datenbeständen, was hauptsächlich durch das Prinzip der *Datenunabhängigkeit* zu erklären ist. Charakterisiert ist dieses Prinzip durch die Trennung der Anwendungsentwicklung von der optimierten Datenhaltung. Einerseits kann sich dadurch der Anwendungsentwickler vollständig auf seine Applikationen konzentrieren und hat die Möglichkeit, über eine definierte Schnittstelle auf die Daten zuzugreifen. Andererseits bestehen die Aufgaben des Datenbankentwicklers in der Optimierung der Speicherung und des Zugriffs auf die Daten. Weiterhin ist hierdurch die vollständige Entkopplung einer Anwendung von den benutzen Daten gegeben, so daß eine Änderung der Anwendung oder ein Wechsel der Datenbank nicht zu unüberwindbaren Hindernissen führt. Neben der Datenunabhängigkeit ist der Aspekt der *Datenintegration*, d.h. die Nutzung der Daten aus mehreren Anwendungen und die damit verbundene unnötige Mehrfachspeicherung der Daten, ein wichtiger Grund[1] für die Verwendung von Datenbanken. Nach diesen ersten Vorüberlegungen definieren wir nun die Begriffe Datenbank und Datenbank-Management-System.

Definition 2.1 (Datenbank)

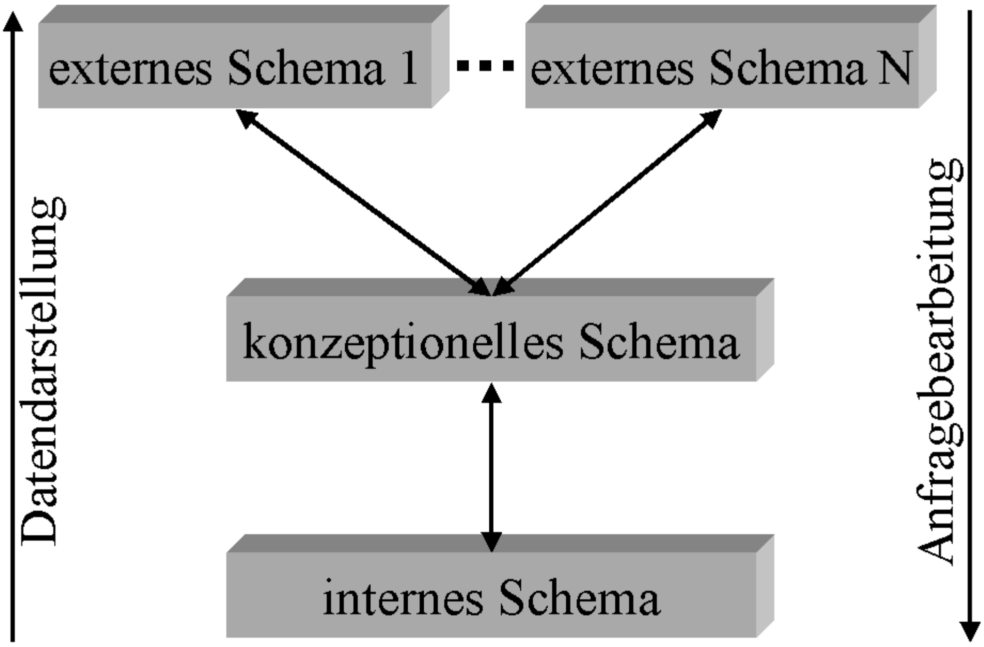
> *Eine* Datenbank *ist eine Sammlung von nicht-redundanten Daten, die von mehreren Applikationen benutzt werden. [*Howe *1983]*

Definition 2.2 (Database-Management-System (DBMS))

> *Als* Datenbank-Management-System (DBMS) *wird eine Software bezeichnet, die die Verwaltung von Daten in der Datenbank übernimmt. Insbesondere unterstützt das DBMS den Zugriff der Anwendungen auf diese Daten.*

[1] Natürlich gibt es noch weitere Gründe, wie zum Beispiel der Datenschutz oder das Transaktionskonzept.

Die oben beschriebene Datenunabhängigkeit teilt sich in die physische und die logische Datenunabhängigkeit auf. Die physische Unabhängigkeit bedeutet, daß die konzeptionelle Sicht auf den Datenbestand unabhängig von der gewählten Datenstruktur ist. Mit der logischen Unabhängigkeit wird die Abkopplung der Anwendungsschnittstelle von dem Datenbestand beschrieben. Aus dieser Datenunabhängigkeit wurde das Modell der Drei-Ebenen-Schema-Architektur[2] hergeleitet (siehe Abbildung 2.1).



Quelle: [Saake und Sattler 2000]

Abbildung 2.1.: Drei-Ebenen-Schema-Architektur

Hierbei stellt das interne Schema die systemspezifische Realisierung der Datenbank dar. Das konzeptionelle Schema beinhaltet eine implementierungsunabhängige Modellierung der gesamten Datenbank und das externe Schema eine anwendungsspezifische (Teil-)Sicht auf diese. In der Regel sind die Modellierungen der einzelnen externen Schemata mit der Modellierung des konzeptionellen Schemas identisch.

Bei relationalen Datenbanken ist die Modellierung der konzeptionellen und externen Schemata durch das relationale Modell spezifiziert.

[2]Diese Architektur wird auch als Dreischichtenmodell nach ANSI/SPARC bezeichnet. Die Ebenen sind dort mit Benutzersicht, logische Gesamtsicht und physische Sicht benannt.

2.1. Das relationale Modell

Die erste Version eines relationalen Modells wurde von E. F. Codd definiert (vgl. [Codd 1970]). Aufbauend auf dieser Arbeit wurde das relationale Modell kontinuierlich verbessert und detaillierter beschrieben (siehe [Codd 1979]). Die zweite Version wurde schließlich in [Codd 1990] veröffentlicht.

Für die Einführung in das relationale Modell beziehen wir uns auf [Sauer 1998]. Wir legen zunächst eine prinzipielle Übersicht zu den Bestandteilen des relationalen Modells fest und werden im weiteren Verlauf nur die für diese Arbeit relevanten Punkte konkret definieren.

Ein *relationales Modell* besteht aus (relationalen) Objekten, Operationen und Regeln.
Die im relationalen Modell verwendeten *(relationalen) Objekte* sind Domain (Wertebereich), Relation (Tabelle), Degree (Ausdehnungsgrad der Tabelle), Attribut (Spalte), Tuple (Datensatz), Candidate-Key (eindeutiger Schlüssel), Primary-Key (Primärschlüssel), Alternate-Key (Alternativschlüssel) und Foreign-Key (Fremdschlüssel).
Auf diese Objekte können die *(relationalen) Operatoren* Restriction (Zeilenselektion), Projection (Spaltenselektion), Product (Kartesisches Produkt), Union (Vereinigung), Intersection (Schnittmenge), Difference (Differenz), Join (Verbindung) und Division angewendet werden. Diese Operatoren definieren eine relationale Algebra, mit der die Objekte bearbeitet werden können.
In dem Modell müssen die zwei *(relationalen) Integrationsregeln* Entity-Integrität und referenzielle Integrität gelten.

Die oben aufgeführten Objekte, Operatoren und Regeln beschreiben das relationale Modell vollständig. Für diese Arbeit benötigen wir allerdings nur einen Teilbereich, den wir im folgenden genauer definieren.

In dem relationalen Modell werden alle Daten in Form von Relationen gespeichert.

Definition 2.3 (Relation/Tabelle)

> *Eine* Relation (Tabelle) *stellt eine logisch zusammenhängende Einheit von Informationen dar. Sie besteht aus einer festen Anzahl von* Attributen (Spalten) *und einer variablen Anzahl von* Tupeln (Zeilen). *Die Relation besitzt einen eindeutigen Namen, den* Relationsnamen.

Jedem Attribut ist ein Wertebereich zugeordnet. Besitzt ein Attribut innerhalb eines Tupels keinen Wert, ist es mit $NULL$ belegt. $NULL$ ist kein spezieller Wert, sondern ein Bezeichner für einen nicht gesetzten Wert. Insbesondere sind $NULL$-Einträge nicht miteinander vergleichbar. Den Relationsnamen zusammen mit den Attributen der Relation bezeichnet man auch als *Relationsschema*. In Abbildung 2.2 wird ein Beispiel für ein Relation mit allen Bezeichnungen dargestellt. Die Anzahl der Attribute wird als *Degree/Ausdehnungsgrad* der Relation bezeichnet. Eine Relation mit n Attributen wird als *n-ary*, mit genau einem Attribut als *unär* und mit genau zwei Attributen als *binär* bezeichnet.

Die Begriffe Relation, Attribut und Tupel des relationalen Modells entsprechen im SQL-Sprachgebrauch den Begriffen Tabelle, Spalte und Zeile. Wir werden in dieser Arbeit keine Unterscheidung zwischen diesen Begriffen durchführen und beide Begriffswelten synonym zueinander benutzen.

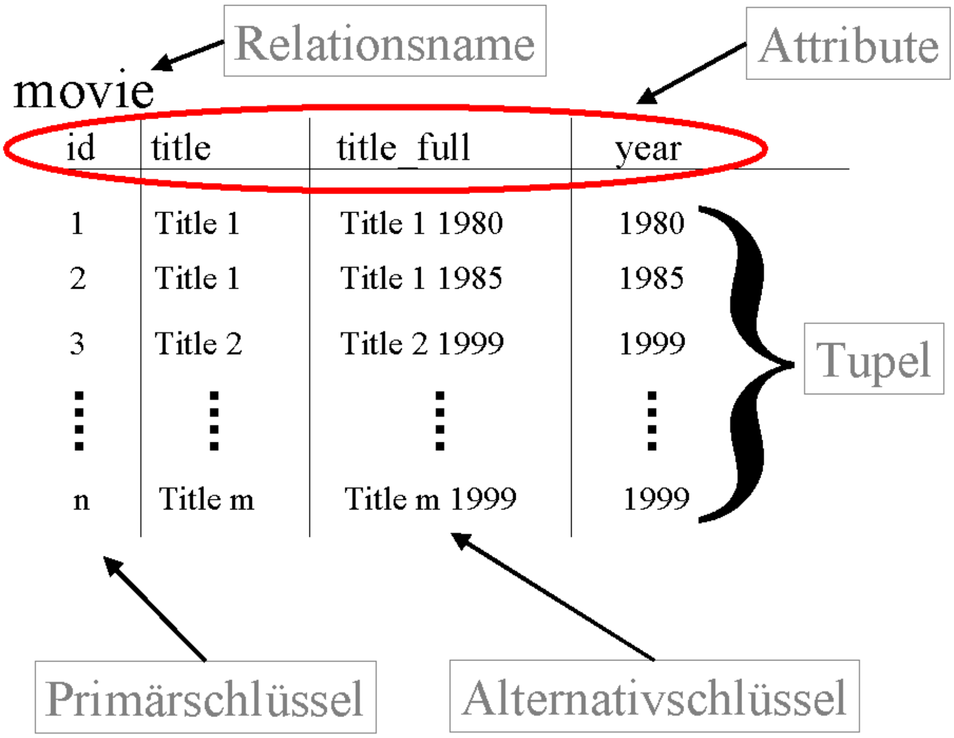


Abbildung 2.2.: Beispiel einer Relation mit allen Bezeichnungen

Eine Relation kann einen oder mehrere eindeutige Schlüssel besitzten:

Definition 2.4 (Schlüssel)

Eine Menge von Attributen $\mathcal{A}$ wird als Schlüssel *bezeichnet, wenn die folgenden Eigenschaften gelten:*

- *Alle Werte dieser Attributmenge sind eindeutig (unique).*
- *$\mathcal{A}$ identifiziert jedes Tupel der Relation eindeutig.*
- *Alle Attribute der Menge $\mathcal{A}$ sind notwendig, d.h. wird ein Attribut dieser Menge entfernt, so ist die so entstandene Menge kein Schlüssel mehr.*

Jeder Relation kann ein ausgewählter Schlüssel als Primärschlüssel zugewiesen werden.

Definition 2.5 (Primärschlüssel)

> *Ein* Primärschlüssel *einer Relation ist ein markierter Schlüssel der Relation.*

Zwischen verschiedenen Relationen kann durch die Festlegung eines Fremdschlüssels eine logische Verbindung erzeugt werden.

Definition 2.6 (Fremdschlüssel)

> *Eine Attributmenge wird als* Fremdschlüssel *einer Relation bezeichnet, wenn eine identische Attributmenge (gleiche Wertebereiche) in einer anderen Relation ein Primärschlüssel ist.*

Durch die Festlegung von Fremdschlüsseln und die dadurch definierten logischen Verbindungen zwischen Relationen ergibt sich die Frage nach der Integrität von Daten in der Datenbank. Integrität meint hierbei, daß eine Datenbank unversehrt ist, also keine widersprüchlichen Daten enthält. Codd führt dazu die Entity-Integrität und referenzielle Integrität als relationale Regeln ein.

Definition 2.7 (Entity-Integrität)

> *Ist A ein Attribut des Primärschlüssels einer Relation, so darf A zu keinem Zeitpunkt den Wert $NULL$ besitzen.*

Diese Regel stellt eine notwendige Bedingung an die Datenbank dar, da $NULL$-Werte nicht eindeutig sind und alle Tupel einer Relation durch den Primärschlüssel eindeutig identifiziert sein müssen.

Die zweite Regel bezieht sich auf die logische Verbindung zwischen Relationen durch Fremdschlüssel.

Definition 2.8 (Referenzielle Integrität)

> *Sei R_1 eine Relation mit einem Fremdschlüssel, der auf den Primärschlüssel der Relation R_2 referenziert. Dann muß gelten: Jeder Wert des Fremdschlüssels aus R_1 existiert als identischer Wert des Primärschlüssels in R_2 oder der Wert des Fremdschlüssels ist $NULL$.*

Die referenzielle Integrität wird durch die Überwachung bestimmter Operationen auf der Datenbank garantiert. Insbesondere muß diese Regel bei dem Hinzufügen, Ändern oder Löschen von Tupeln eingehalten werden. Das Löschen eines Tupels aus der Relation mit einem Fremdschlüssel ist im allgemeinen problemlos möglich. Beim Löschen eines Tupels aus der assoziierten Relation mit dem Primärschlüssel sind hingegen Probleme unausweichlich (sofern der Wert im Fremdschlüssel existiert). Aus diesem Grund wird die Reaktion des DBMS in solchen Fällen bei der Definition des Fremdschlüssels festgelegt. Die möglichen Reaktionen bestehen in der Abweisung der Anweisung, in der gleichzeitigen Löschung aller assoziierten Tupel der Fremdschlüsselrelation oder im Setzen des Fremdschlüssels auf $NULL$.

Zur vollständigen Beschreibung des relationalen Modells fehlen noch die relationalen Operatoren. Durch die Anwendung der Operatoren auf eine oder mehrere Relationen erhalten wir eine neue Relation.

person

id	name	geschlecht
1	Peter Müller	m
2	Eva Schmidt	w
3	Fritz Walter	m

→ **Sel(person, geschlecht='m')**

id	name	geschlecht
1	Peter Müller	m
3	Fritz Walter	m

(a) Selektion

person

id	name	geschlecht
1	Peter Müller	m
2	Eva Schmidt	w
3	Fritz Walter	m

→ **Proj(person, <id, name>)**

id	name
1	Peter Müller
2	Eva Schmidt
3	Fritz Walter

(b) Projektion

Abbildung 2.3.: Anwendungsbeispiel der relationalen Operatoren „Selektion“ und „Projektion“

Durch die *Selektion (Restriktion)* wird aus einer Relation durch eine oder mehrere Bedingungen an ihre Attribute eine neue Relation mit einer kleineren Tupelmenge extrahiert. Bei der *Projektion* wird eine neue Relation durch die Teilmenge der Attribute der Ursprungsrelation bestimmt. In Abbildung 2.3 sind in (a) und (b) Beispiele zu diesen Operatoren aufgeführt.

person

id	name	geschlecht
1	Peter Müller	m
2	Eva Schmidt	w
3	Fritz Walter	m

actor

pid	film
1	Das Boot
1	Star Wars
3	Man in Black

→ **Join(person, actor, person.id = actor.pid)**

id	name	film
1	Peter Müller	Das Boot
1	Peter Müller	Star Wars
3	Fritz Walter	Man in Black

Abbildung 2.4.: Anwendungsbeispiel des relationalen Operators „(natürlicher) Join“

Neben diesen Operatoren, die auf einer einzigen Relation arbeiten, verbindet der Join-Operator zwei Relationen zu einer neuen. Diese Verbindung wird durch eine

Beziehung zwischen zwei Attributwerten der beiden Relationen bestimmt. In dem Beispiel in Abbildung 2.4 ist die Beziehung zwischen *person.id* und *actor.pid* durch die Gleichheit der Attributwerte festgelegt.

Definition 2.9 (Relationales DBMS)

> *Unter einem* relationalen Datenbank-Management-System *versteht man ein DBMS, welches im konzeptionellen Schema zur Modellierung der Datenbank das relationale Modell einsetzt.*

Typische relationale Datenbanken sind z.B. Oracle, mySQL und PostgreSQL.

2.2. Darstellung von logischen Verbindungen

Im weiteren Verlauf der Arbeit werden wir zur Darstellung von logischen Verbindungen in Datenbanken eine an UML angelehnte Form verwenden. Dabei werden Relationen als zweigeteilte Rechtecke mit dem Relationsnamen im oberen Teil und allen Attributen im unteren Teil dargestellt. Die Attribute des Primärschlüssel sind unterstrichen.

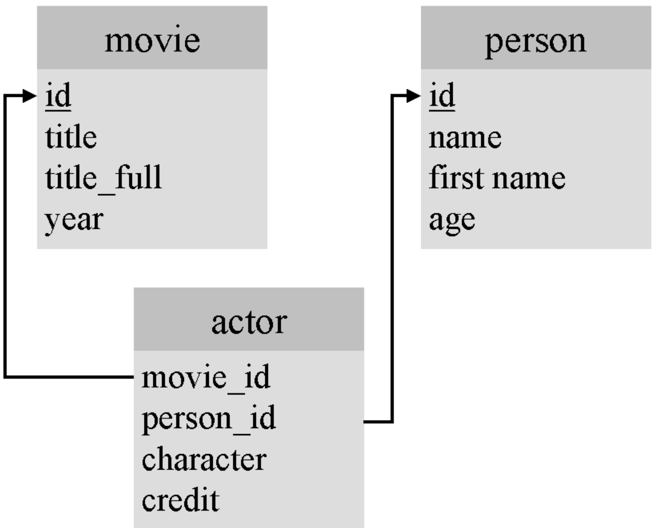


Abbildung 2.5.: Beispiel der Darstellung logischer Verbindungen in einer Datenbank

Die logischen Verbindungen zwischen diesen Relationen entsprechen den Verbindungslinien zwischen den Fremdschlüsselattributen und der zugehörigen Relation mit dem Primärschlüssel. In Abbildung 2.5 stellen wir die drei Relationen *movie*, *person* und *actor* dar. Die logische Verbindung durch Fremdschlüssel besteht einerseits durch den Fremdschlüssel *movie_id* der Relation *actor* mit dem Primärschlüssel *id* der Relation *movie* und andererseits durch den Fremdschlüssel *person_id* der Relation *actor* mit dem Primärschlüssel *id* der Relation *person*.

Die in UML übliche Darstellung von Objekten und Abhängigkeiten zwischen diesen Objekten wollen wir hier vernachlässigen, auch von einer konkreten Einführung in UML sehen wir ab, da diese für unsere Darstellungen unnötig ist.

2.3. Kurze Einführung in SQL

Bei relationalen Datenbanksystemen werden die Datenbankanweisungen in zwei Gruppen aufgeteilt:

1. die Datendefinitionssprache (DDL – Data Definition Language) und
2. die Datenmanipulationsprache (DML – Data Manipulation Language)

Zu der Datendefinitionssprache gehören alle Datenbankanweisungen, die die logische Struktur bzw. Tabellen beschreiben, verändern oder löschen. Insbesondere sind das die Befehle CREATE TABLE und CREATE/DROP VIEW. Zu der Datenmanipulationsprache gehören alle datenverarbeitenden Anweisungen, wie z.B. SELECT und UPDATE. In SQL (Standard Query Language) sind beide Sprachen vereint worden.

Für den weiteren Verlauf dieser Arbeit sind einerseits die Anfragen an die Datenbank mit Hilfe des SQL-Befehls SELECT und andererseits die Erzeugung von Views auf die Daten wichtig.

Zur Darstellung der ausgewählten Anweisungen werden wir die erweiterte Backus-Naur-Form (EBNF) verwenden. In dieser Notation werden reservierte Wörter groß geschrieben, Alternativen durch | getrennt, optionale Bestandteile (keinmal oder genau einmal) durch [ ] und wiederholbare Bestandteile (kein-, einmal oder mehrmals) durch { } gekennzeichnet.

2.3.1. Anfragen in SQL

Anfragen werden in SQL generell durch den Befehl SELECT durchgeführt. Dieser Befehl besitzt dadurch eine zentrale Bedeutung innerhalb der SQL-Befehle. Durch den SELECT-Befehl ist es möglich, aus einer oder mehreren Tabellen Daten zu verarbeiten und die extrahierten Daten in Form einer Tabelle anzuzeigen. Die Daten der involvierten Tabellen oder Views werden aussschließlich gelesen und nicht verändert. In Tabelle 2.1 ist ein Teil der Syntax des SELECT-Befehls dargestellt.

Durch den SELECT-Befehl können unter anderem die relationalen Operationen Selektion, Projektion und Join durchgeführt werden. D.h. man hat die Möglichkeit, aus einer Relation Attribute und Tupel zu extrahieren und zusätzlich mehrere Relationen miteinander zu verbinden.

Innerhalb des SQL-Befehls können Funktionen auf Spalten oder auf Tupel dieser Spalten angewendet werden. Eine für unsere Anwendung wichtige Funktion ist *COUNT*, die die Anzahl der durch den SELECT-Befehl beschriebenen Tupel liefert.

```
SELECT [ ALL | DISTINCT [ ON ( expression [, ...] ) ] ]
    * | expression [ AS output_name ] [, ...]
    [ FROM from_item [, ...} ]
    [ WHERE condition ]
    [ GROUP BY expression [, ...] ]
    [ HAVING condition [, ...] ]
    [ { UNION | INTERSECT | EXCEPT [ ALL ] } select ]
    [ ORDER BY expression { ASC | DESC | USING operator } [, ...] ]
    [ FOR UPDATE [ OF tablename [, ...] ] ]
    [ LIMIT { count | ALL } [ { OFFSET | , } start ]]

from_item :- [ ONLY ] table_name [ * ]
             [ [ AS ] alias [ ( column_alias_list ) ] ] |
             ( select ) [ AS ] alias [ ( column_alias_list ) ]
             | from_item [ NATURAL ] join_type from_item
             [ ON join_condition | USING ( join_column_list ) ]
```

Tabelle 2.1.: Spezifikation des SQL-Befehls SELECT

2.3.2. Definition von Views

Unter einem View versteht man eine logische Sicht auf eine oder mehrere Relationen (Tabellen oder Views). Ein View wird durch einen SELECT-Befehl erzeugt. Die Syntax zu diesem Befehl sieht folgendermaßen aus:

```
CREATE VIEW view [ column [, ...] ]
    AS SELECT expression [ AS colname ] [, ...]
    FROM table [ WHERE condition ]
    [ WITH [ CASCADE | LOCAL ] CHECK OPTION ]
```

Tabelle 2.2.: Spezifikation des SQL-Befehls CREATE VIEW

Durch die Erzeugung eines Views wird keine neue Tabelle in der Datenbank angelegt, sondern nur eine View-Definition im Systemkatalog der Datenbank gespeichert. Bei dem Zugriff auf einen View wird diese Definition zum Zugriff auf die Daten verwendet.

2.4. JDBC

Zum Zugriff auf Datenbanken aus einer Programmiersprache heraus bietet jeder DBMS-Hersteller eine eigene Programmierschnittstelle an. Diese herstellerspezifischen Bibliotheken führen zu einer Abhängigkeit von dem Anbieter und erschweren zusätzlich durch ihre Unterschiede die Entwicklung DBMS-unabhängiger Anwendungen sowie die Portierung auf andere DBMS.

JDBC ist vergleichbar mit ODBC (Open Database Connectivity), einer allgemeinen Datenbankschnittstelle, die ursprünglich für Windows entwickelt wurde. Insbesondere baut die Konzeption von JDBC auf ODBC auf.

Bei JDBC handelt es sich eine Schnittstellendefinition (API) von Sun für den einheitlichen (herstellerunabhängigen) Zugriff auf unterschiedliche Datenbanken mittels Java (siehe [JDBC 2002]). Für den konkreten Zugriff auf die jeweiligen Datenbanken ist ein Treiber, eine spezielle Implementierung der Schnittstelle, für diese Datenbank notwendig. Durch die festgelegte Schnittstelle ist der Austausch der Datenbank ohne Neuübersetzung oder Änderungen an dem Quellcode möglich.

Einen guten Überblick über Datenbanken und Java sowie insbesondere über JDBC bietet [Saake und Sattler 2000].

2.4.1. Treiberkonzept

JDBC stellt einen sogenannten Treibermanager sowie abstrakte Klassen (Schnittstellen) für den Datenbankzugriff bereit. In Verbindung mit dem datenbankspezifischen Treiber erfolgt damit der Zugriff auf die Datenbank. Der Treibermanager wählt anhand der Verbindungsinformationen den geeigneten Treiber aus.

Die Realisierung der Treiber kann in vier Kategorien (vgl. Abbildung 2.6) eingeteilt werden:

1. JDBC-ODBC-Bridge (Typ 1)

   Hier wird ODBC zur Interaktion mit der Datenbank benutzt. D.h. alle JDBC-Aufrufe werden in ODBC-Aufrufe übersetzt.

2. Native-API-Treiber (Typ 2)

   Hierbei benutzt der JDBC-Treiber die DBMS-spezifischen Programmierschnittstellen der jeweiligen Datenbank.

3. JDBC-Net-Treiber (Typ 3)

   Zwischen dem JDBC-Treiber und dem Datenbanksystem wird eine DBMS-unabhängige Komponente (Middleware) eingefügt, die die Übersetzung zwischen dem DBMS-unabhängigen Protokoll und dem DBMS-speziellen Protokoll übernimmt.

4. Native-Protokoll-Treiber (Typ 4)

   Dieser Treiber übersetzt den JDBC-Aufruf direkt in das spezielle Datenbankprotokoll, ohne dazu die Programmierschnittstelle der Datenbank zu benutzen.

Die Treibertypen 1 und 2 sind eher als Übergangslösungen anzusehen, da sie durch die Benutzung von externen (betriebssystemspezifischen) Bibliotheken einerseits gegen die Idee der Unabhängigkeit von einem Betriebssystem und bei der Verwendung innerhalb von Applets andererseits gegen die Sicherheitsrestriktionen (Sandbox) verstoßen.

Bei den Treibertypen 3 und 4 treten diese Einschränkungen nicht mehr auf. Insbesondere der Typ 4 bietet durch seine direkte Implementierung des speziellen Datenbankprotokolls einen effizienten Zugriff auf die Datenbank.

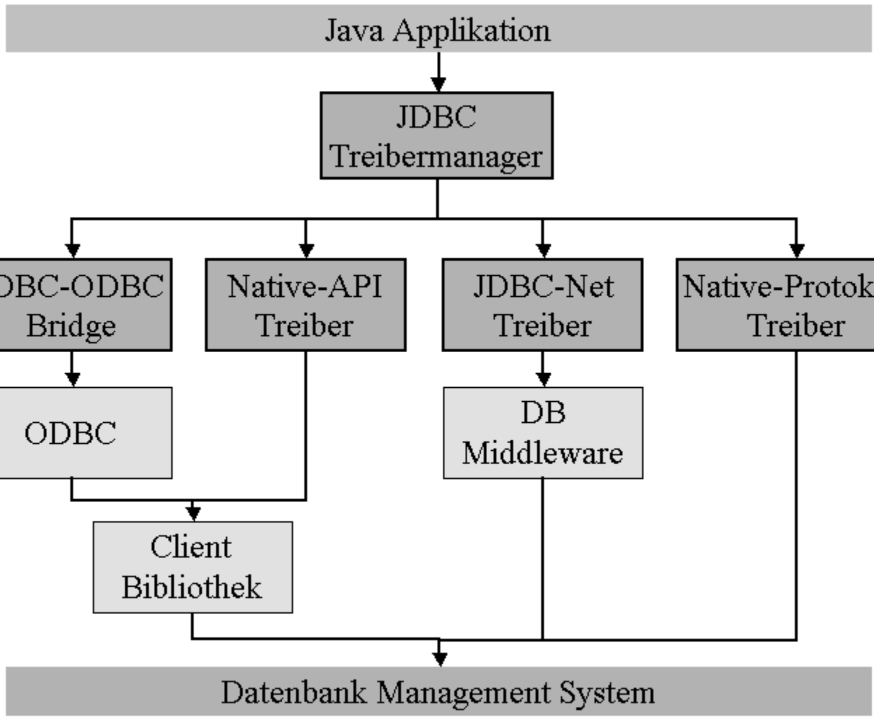


Abbildung 2.6.: Übersicht der Implementierungtypen für JDBC-Treiber

2.4.2. Programmierschnittstelle (API)

Die Programmierschnittstelle von JDBC bietet aufgrund einer großen Auswahl von Methoden die Möglichkeit, Informationen zu der Datenbank (Kataloge, Schemata, Datentypen und spezielle Optionen der Datenbank) und zu den Tabellen (Name, Spalten, Datentypen und Schlüssel) direkt auszulesen, und erlaubt die Ausführung beliebiger SQL-Befehle. Für diese Ausführung sind unterschiedliche Varianten vorgesehen:

1. die direkte Ausführung einzelner SQL-Befehle,
2. die Sammlung mehrerer SQL-Befehle und sukzessive Ausführung dieser Befehle durch einen Befehl (Batch),
3. die Erzeugung von vorkompilierten Templates zu SQL-Befehlen, in denen nur die Variablen ersetzt werden, so daß dann eine schnelle Ausführung möglich ist.

Die aktuelle JDBC-Version bietet bereits die vollständige Unterstützung der verschiedenen SQL-Normen bis hin zu SQL3.

3. Regellernen und Wissensentdeckung in Datenbanken

Im wissenschaftlichen Bereich wird die Wissensentdeckung in Datenbanken (Knowledge Discovery in Databases – KDD) wie folgt beschrieben:

> Wissensentdeckung in Datenbanken ist der nichttriviale Prozeß der Identifikation gültiger, neuer, potenziell nützlicher und schlußendlich verständlicher Muster in (großen) Datenbanken. [Fayyad et al. 1996]

Die Wissensentdeckung in Datenbanken wird als ein Prozeß verstanden, der aus mehreren Schritten besteht. Dadurch wird die umfassende Sichtweise der Wissensentdeckung auf die Daten und den Prozeß der Datenanalyse hervorgehoben. Innerhalb des KDD-Prozesses wird der eigentliche Analyseschritt, die Suche nach Hypothesen und deren Bewertung, mit Data Mining bezeichnet. In diesem Schritt werden verschiedene Verfahren des maschinellen Lernens eingesetzt. In [Wrobel 1998] und [Morik et al. 2000] werden die Unterschiede und Gemeinsamkeiten zwischen KDD und maschinellem Lernen diskutiert sowie eine Einführung in die Begriffswelt und in verschiedene im KDD-Prozeß benutzte Verfahren des maschinellen Lernens gegeben.
In dem Analyseschritt des KDD-Prozesses sind die Verfahren der induktiven logischen Programmierung von besonderem Interesse, da sie die Entdeckung komplexer Regeln erlauben. In der induktiven logischen Programmierung erfolgt die Repräsentation durch eingeschränkte Prädikatenlogik. Die ILP kann als Suche nach geeigneten Hypothesen in dem Hypothesenraum aufgefaßt werden. Wir werden in diesem Kapitel das Lernverfahren RDT und seine Erweiterung RDT/DB betrachten. Beide sind Lernverfahren der induktiven logischen Programmierung. Sie führen das Lernen von Regeln durch. Diese Lernaufgabe ist schwieriger als das Begriffslernen (vgl. [Kietz 1996]). Beim Regellernen muß das Lernverfahren diejenigen Prädikate selbständig finden, über die es Regeln lernen will. Im Gegensatz dazu werden beim Begriffslernen dem Lernverfahren ein oder mehrere Zielbegriffe vorgegeben.

Zunächst werden wir im folgenden Abschnitt die Lernaufgabe des Regellernens definieren und die im weiteren Verlauf dieses Kapitels wichtigen Begriffe einführen. Aufbauend darauf beschreiben wir in Abschnitt 3.2 den Regellerner RDT (Rule

Discovery Tool). Die Anbindung dieses Regellerners an eine relationale Datenbank und die damit verbundenen Erweiterungen führen wir in Abschnitt 3.3 ein.

3.1. Das Regellernproblem und einige wichtige Begriffe

Das Regellernproblem ist in [Kietz 1996, Seite 23] wie folgt definiert worden:

Definition 3.1 (Regellernen/Wissensentdeckung)

Gegeben sei eine Menge von Beobachtungen E in einer Sprache $\mathcal{L}_E$, Hintergrundwissen B in einer Sprache $\mathcal{L}_B$ sowie eine Sprache $\mathcal{L}_H$ für die Hypothesen.
Gesucht wird eine Menge H von Hypothesen (Regeln) in der Sprache $\mathcal{L}_H$ mit den folgenden Eigenschaften:

Gültigkeit:	$\mathcal{M}^+(B \cup E) \subseteq \mathcal{M}(H)$, *d.h. H ist in allen minimalen Modellen von B und E wahr.*
Notwendigkeit:	$\forall h \in H : \exists e \in E : B, E \setminus \{e\} \not\models e \,\text{und}\, B, E \setminus \{e\}, h \models e$, *d.h. alle Hypothesen enthalten neue Informationen über die Beobachtungen.*
Vollständigkeit:	$\forall h \in H$ *, die gültig und notwendig sind* $: H \models h$, *d.h. alle in B und E wahren Hypothesen folgen aus H.*
Minimalität:	$\nexists G \subset H :$ *G gültig und vollständig*

In [Kietz 1996, Seite 84] wird die Lernaufgabe für das im folgenden betrachtete Lernverfahren RDT über funktionsfreien[1], k-literalen[2] Hornklauseln als Sprache der Hypothesen festgelegt. Eine ausführliche Einführung in die Notationen und Definitionen der Logik bietet [Görz 1995, Kapitel 2]. Wir beschränken uns hier auf eine kurze informelle Einführung der von uns benötigten Begriffe.

Eine *Klausel c* ist eine endliche Menge von Literalen. Dargestellt wird sie als Menge

$$\{\neg l_1, \ldots, \neg l_i, l_{i+1}, \ldots, l_n\}$$

oder alternativ als Regel

$$l_1, \ldots, l_i \rightarrow l_{i+1}, \ldots, l_n.$$

Eine Klausel heißt *Hornklausel*, wenn sie höchstens ein positives Literal enthält, und sie heißt *(Horn)Regel*, wenn sie genau ein positives und mindestens ein negatives Literal enthält. Desweiteren werden Klauseln, die nur genau ein positives

[1] Damit ist die Funktionsfreiheit der Formeln gemeint. Eine Formel heißt funktionsfrei, wenn sie nur Prädikatensymbole, Variablen und Konstanten enthält.

[2] Eine Klausel heißt k-literal, wenn maximal k Literale im Klauselköper vorkommen.

Literal enthalten, als *Fakt* bezeichnet. Innerhalb einer Hornklausel werden die negativen Literale als *Prämissen* und das positive Literal als *Konklusion* oder Kopf der Regel bezeichnet. Eine korrekte Hornregel ist zum Beispiel:

$$l_1, \ldots, l_i \rightarrow l_{i+1}$$

In diesem Beispiel sind $l_1, \ldots, l_i$ die Prämissen und l_{i+1} die Konklusion der Regel.

Ein Literal $l = p(t_1, \ldots, t_n)$ besteht aus einem Prädikatensymbol p und Termen $t_1, \ldots, t_n$. Bei funktionsfreien Formeln bestehen die Terme nur aus Variablen- und Konstantensymbolen. Die Variablensymbole werden wir zur Unterscheidung von den später auftretenden Prädikatenvariablen als *Termvariablen* bezeichnen. Im weiteren Verlauf der Arbeit werden wir alle Variablen in Klauseln/Regeln durch einen großen Anfangsbuchstaben kennzeichnen und im Gegensatz dazu alle Prädikatensymbole durch einen kleinen Anfangsbuchstaben.

Um Aussagen über verschiedene Klauseln zu treffen, führen wir eine Ordnung auf Klauseln ein. Dabei verwenden wir nicht die Implikation „aus $c1$ folgt logisch $c2$“, da diese im allgemeinen Fall nicht entscheidbar ist. Stattdessen verwenden wir eine Subsumtionsbeziehung, die eine korrekte, aber unvollständige Ableitungsrelation darstellt. Für diese Subsumtionsbeziehung benötigen wir zunächst die Θ-Substitution:

Definition 3.2 (Substitution)

> *Eine* Substitution $\Theta = \{X_1 \mid t_1, \ldots, X_n \mid t_n\}$ *ist eine eindeutige Abbildung aus der Menge der Variablen in die Menge der Terme ohne zu unifizieren, d.h. sie darf weder verschiedene quantifizierte Variablen noch quantifizierte und freie Variablen unifizieren.*

Durch die Anwendung einer Substitution Θ auf eine Klausel C, bezeichnet mit $C\Theta$, werden alle Termvariablen gleichzeitig durch die entsprechenden Terme ersetzt.

Mit Hilfe dieser Substitution sind wir in der Lage, die Θ-Subsumtion auf Klauseln zu definieren:

Definition 3.3 (Θ-Subsumtion $\vdash_\Theta$)

> *Eine Klausel $C1$ Θ-subsumiert eine Klausel $C2$ ($C1 \vdash_\Theta C2$) genau dann, wenn eine Substitution Θ auf Termvariablen existiert, so daß $C1\Theta \subseteq C2$ gilt.*

Nach [Robinson 1965] gilt:

Satz 3.4

> *Θ-Subsumtion ist eine korrekte Ableitungsrelation, d.h. es gilt:*
>
> $$D \vdash_\Theta C \quad \Rightarrow \quad D \models C$$

Für Hornklauseln kann die Θ-Subsumtion wie folgt definiert werden:

Definition 3.5 (Θ-Subsumtion $\geq_\Theta$ für Hornklauseln)

Seien $C1$ und $C2$ Hornklauseln der Form $C_{kopf} \leftarrow C_{körper}$. Dann gilt:

$$C1 \geq_\Theta C2 \quad \Leftrightarrow \quad \exists\Theta \; : \; C1_{kopf}\Theta = C2_{kopf} \wedge C1_{körper}\Theta \subseteq C2_{körper}$$

Die durch das Lernverfahren RDT generierten Hypothesen sind generative Klauseln, d.h. jede Termvariable der Konklusion ist auch Termvariable in der Prämisse.

3.2. Rule Discovery Tool (RDT)

In diesem Abschnitt werden wir das Regelentdeckungswerkzeug RDT (Rule Discovery Tool) [Kietz und Wrobel 1992] behandeln. Bei RDT handelt es sich um ein Verfahren zum effizienten Lernen von Regeln auf der Basis von Fakten. Es ist in der Werkbank MOBAL [Morik et al. 1993] integriert. Innerhalb dieser Werkbank sieht das Anwendungsszenario die Erweiterung der MOBAL-Wissensbasis durch die Induktion zusätzlicher Regeln vor.

Das Lernverfahren RDT benutzt eine explizite, syntaktische Sprachbeschränkung bzgl. der Hypothesensprache $\mathcal{L}_H$. Diese Sprachbeschränkung ist durch den Anwender deklarierbar und veränderbar und erfolgt durch die explizite Angabe von Modellwissen. Man spricht in diesem Fall von Lernen mit *deklarativem Bias*. Dabei wird der deklarative Bias unterteilt in den syntaktischen und den sematischen Bias. Der syntaktische Bias legt die Form der Hypothesen fest, wohingegen der sematische Bias den logischen Aufbau der Hypothesen beschreibt. Der Begriff des deklarativem Bias ist als Charakteristikum der induktiven logischen Programmierung bekannt (vgl. [Muggleton und De Raedt 1994]) und stellt im maschinellen Lernen einen modellbasierten Ansatz dar.

3.2.1. Modellwissen

RDT benutzt verschiedene Arten von Modellwissen zur Einschränkung des Hypothesenraumes. Die wichtigste Einschränkung stellen Regelschemata[3]dar, die die syntaktische Form der Hypothesen festlegen (syntaktischer Bias). Zusätzlich wird eine sematische Einschränkung (semantischer Bias) durch eine Prädikatentopologie ([Klingspor 1991], [Morik et al. 1993, Kapitel 5]) und eine Sortentaxonomie ([Morik et al. 1993, Kapitel 4],[Kietz 1988]) vorgenommen.

Regelschemata

Die folgende Einführung zu Regelschemata ist angelehnt an [Morik et al. 1993, Kapitel 6] und [Kietz 1996, Kapitel 5].

[3]Regelschemata werden auch als Regelmodelle oder Meta-Prädikate bezeichnet.

Durch Regelschemata wird die syntaktische Form der Regelhypothesen auf bestimmte Formen festgelegt. Ursprünglich sind Regelschemata von [EMDE 1987] im METAXA-System eingeführt worden.

Definition 3.6 (Regelschema)

Ein Regelschema $\mathcal{RS}$ *hat die Form einer Regel, in der mindestens ein Prädikatensymbol durch eine Prädikatenvariable ersetzt wurde. Zusätzlich besitzt jedes Regelschema einen Kopf, der eine Bezeichnung, alle benutzten Prädikatenvariablen und Konstanten enthält.*

Veranschaulichen läßt sich ein Regelschema an folgendem Beispiel:

Beispiel 3.7

Das Regelschema

$$mp(C, P1, P2, P3, Q) : P1(X, Y) \ \& \ P2(X, C) \ \& \ P3(Y) \rightarrow Q(Y)$$

hat die Bezeichnung mp*, benutzt die Prädikatenvariablen* $P1, P2, P3$ *und* Q *und beinhaltet eine Konstante* C*.*

Der später eingeführte RDT-Algorithmus benutzt eine Ordnung der Prämissen in einem Regelschema, um gemäß dieser Reihenfolge die schrittweise Instanziierung durchzuführen. Damit eine solche Ordnung definiert werden kann, benötigen wir zunächst die Relationsketten und die Tiefe.

Für die Termvariablen aus einer Regel bzw. einem Regelschema definiert die Relationskette eine Verbindung der Termvariablen mit der Konklusion.

Definition 3.8 (Relationskette $rc(X)$)

Eine Relationskette $rc(X)$ *ist eine Liste von Prädikatenvariablen, die die Verbindung einer Termvariable* X *mit der Konklusion darstellt, und ist wie folgt (rekursiv) definiert:*

1. *Ist* X *Termvariable der Konklusion, gilt:* $rc(X) = \emptyset$.

2. *Ist* X_i *mit* $(1 \leq i \leq n)$ *Termvariable einer Prämisse* $P(X_1, \ldots, X_n)$*, dann gilt:*

$$rc(X_i) = \{P\} \circ rc(X_j) \quad \Leftrightarrow \quad X_j \text{ mit } (1 \leq j \leq n), j \neq i \text{ ist durch } rc(X_j) \text{ mit der Konklusion verbunden.}$$

Es kann ggf. mehrere (unterschiedliche) Verbindungen zwischen Termvariable und Konklusion geben. Dadurch bedingt können zu einer Termvariablen X mehrere Relationsketten $rc(X)$ existieren.

Mit Hilfe der Relationsketten zu einer Termvariablen definieren wir die Tiefe einer Termvariablen, die die minimale Anzahl von Prädikaten zum Erreichen der Konklusion angibt.

Definition 3.9 (Tiefe $\delta(X)$)

Sei X eine Termvariable. Die Tiefe $\delta(X)$ *ist die Länge der minimalen Relationskette:*

$$\delta(X) = \min(\{\, length(rc(X)) \mid rc(X) \text{ ist Relationskette von } X\})$$

Betrachten wir zur Veranschaulichung das folgende Beispiel:

Beispiel 3.10

Für das in Beispiel 3.7 eingeführte Regelschema ergeben sich für die Termvariablen Y, X, C die folgenden Relationsketten und zugehörigen Tiefen:

$$\begin{array}{rclcrcl} rc(Y) & = & \emptyset & \quad & \delta(Y) & = & 0 \\ rc(X) & = & \{P1\} \circ rc(Y) = \{P1\} & & \delta(X) & = & 1 \\ rc(C) & = & \{P2\} \circ rc(X) = \{P2, P1\} & & \delta(C) & = & 2 \end{array}$$

Auf der Menge der Prämissen eines Regelschemas wird mit Hilfe dieser Tiefe eine Ordnung definiert.

Definition 3.11 (Prämissenordnung $\leq_{\mathcal{P}}$)

Seien P und P' Prämissen eines Regelschemas. Die Prämissenordnung $\leq_{\mathcal{P}}$ *ist wie folgt definiert:*

$$P \leq_{\mathcal{P}} P' \;\Leftrightarrow\; \min(\{\delta(X) | X \textit{ in } P\}) \leq \min(\{\delta(X) | X \textit{ in } P'\})$$

Diese Ordnung auf den Prämissenprädikaten wird im RDT-Algorithmus bei der Instanziierung der Regelschemata benutzt. Die Instanziierung der Prämissen erfolgt gemäß dieser Ordnung, d.h. wenn $P \leq_{\mathcal{P}} P'$ gilt, wird P vor P' instanziiert. Durch die inkrementelle Instanziierung (nur der instanziierte Teil wird betrachtet) wird eine Folge von Hypothesen $h_1, h_2, \ldots, h_n$ erzeugt, die gemäß Θ-Subsumtion geordnet sind, folglich gilt $h_1 \vdash_\Theta h_2 \vdash_\Theta \ldots \vdash_\Theta h_n$.
Wenn somit eine teilinstanziierte Hypothese keine Instanzen in den Daten besitzt, brauchen alle daraus resultierenden teilinstanziierten Hypothesen weder instanziiert noch getestet werden. Zudem werden durch die Prämissenordnung Hypothesen vermieden, in denen Prädikate instanziiert sind, die niemals durch andere Prädikate mit der Konklusion verbindbar sind.
Zu dem obigen Beispiel 3.10 ist die Prämissenordnung durch

$$\begin{array}{ccc} P1 & \leq_{\mathcal{P}} & P3 \\ P3 & \leq_{\mathcal{P}} & P1 \end{array} \leq_{\mathcal{P}} P2$$

gegeben. Ein mögliches Regelschema mit der Anordnung der Prämissen gemäß $\leq_{\mathcal{P}}$ ist somit:

$$mp(C, P2, P1, P3, Q) : P3(Y) \ \& \ P1(X, Y) \ \& \ P2(X, C) \rightarrow Q(Y)$$

Durch den Einsatz von Regelschemata in der hier beschrieben Form ist die Form der Hypothesen vorgegeben.

Prädikatentopologie

Eine sematische Einschränkung des Hypothesenraumes erhalten wir durch die Einbeziehung des Verwendungszwecks des Lernergebnisses. Da jedes Lernverfahren zu einem bestimmten Zweck eingesetzt wird, sollen seine Ergebnisse für eine spezielle Aufgabe nützlich sein. Ein (erfahrener) Benutzer kann dem Lernverfahren relevantes Hintergrundwissen zu Verfügung stellen. In RDT wird dazu die Prädikatentopologie eingesetzt. Die Menge der Prädikate $\mathcal{P}$ wird in ggf. nicht-disjunkte Mengen T_i, die Topologieknoten, einer Gruppierung $\mathcal{T} = \{T_1, \ldots, T_m\}$ zerlegt. Diese Topologieknoten erhalten durch den Benutzer beschreibende Bezeichnungen und werden durch gerichtete Kanten zu einer Hierarchie verbunden. Diese Hierarchie wird unter Ausnutzung der Topologiekompatibilität zur Beschränkung des Hypothesenraumes eingesetzt.

Definition 3.12 (Topologiekompatibilität)

Sei h eine Hypothese mit Konklusion q und den Prämissen p_j $(1 \leq j \leq n)$, $q \in T_i$, und $T_l, \ldots, T_k$ seien direkte Vorgänger von T_i. Dann gilt:

$$h \quad \text{topologiekompatibel} \quad \Leftrightarrow \quad \forall p_j : p_j \in T_i \ \vee \ p_j \in T_l \cup \ldots \cup T_k$$

Der Benutzer kann mit Hilfe diese Topologie den Bereich der Suche auf einen bestimmten Zweck oder eine bestimmte Ausrichtung beschränken und durch die Hierarchie die gewünschte Suchrichtung in dem Bereich vorgeben.

In RDT wird durch die Benutzung der Prädikatentopologie die Suche nach Beispielen bzw. Instanzen auf die Elemente einiger weniger Knoten der Topologie beschränkt. Eine solche Topologie kann in MOBAL durch den Benutzer vorgegeben oder durch die Systemkomponente PST (Predicate Structering Tool) bestimmt werden.

Sortentaxonomie

Eine weitere semantische Einschränkung erfolgt durch den Einsatz von Sorten. Dabei sind die Termvariablen eines Prädikats einer Sorte zugewiesen. Während der Instanziierung eines Regelschemas werden durch das Binden eines Prädikates die Sorten der Termvariablen festgelegt. Die Auswahl der weiteren Prädikate mit einer dieser Termvariablen ist damit auf Prädikate mit der korrekten Sorte eingeschränkt. MOBAL bietet die Systemkomponente STT (Sort Taxonimie Tool) zur automatischen Generierung von Argumentsorten an.

3.2.2. Hypothesentest

Bei RDT wird neben dem Test auf Akzeptanz einer (vollinstanziierten) Hypothese ein weiterer Test, ein inkrementeller Test, auf (teil-)instanziierten Regelschemata durchgeführt. Durch den zweiten Test werden (gemäß dem Beschneidungskriterium) solche Hypothesen aussortiert, die zu speziell sind. Da bei der inkrementellen Instanziierung die erzeugten Hypothesen gemäß der Θ-Subsumtion geordnet sind, muß dieses teilinstanziierte Regelschema nicht weiter instanziiert und getestet werden.

Bewertungsmaße

Zur Definition des Akzeptanzkriteriums sowie des Beschneidungskriteriums werden die folgenden Bewertungsmaße verwendet.

Sei im folgenden $h = P(X_1, \ldots, X_m) \rightarrow q(X_1, \ldots, X_n)$ eine Hypothese mit $m \geq n$, $P(X_1, \ldots, X_m)$ die Konjunktion der Prämissen $p_1, \ldots, p_m$, die die Termvariablen $X_1, \ldots, X_m$ beinhalten, und $q(X_1, \ldots, X_n)$ die Konklusion mit den Termvariablen $X_1, \ldots, X_n$.

Auf einer solchen Hypothese h sind die folgenden Mengen definiert:

- Die Menge der positiven Beispiele der Hypothese h ist durch

  $$\mathbf{pos}(h) := \{(c_1, \ldots, c_n) \mid P(c_1, \ldots, c_m) \,\&\, q(c_1, \ldots, c_n)\}$$

  gegeben.

- Die Menge der Beispiele, für die die Prämissen und die Negation der Konklusion wahr sind, ist durch

  $$\mathbf{neg}(h) := \{(c_1, \ldots, c_n) \mid P(c_1, \ldots, c_m) \,\&\, \mathit{not}\,(q(c_1, \ldots, c_n))\}$$

  gegeben.

- Die Menge der Beispiele, für die die Konklusion unbekannt ist, d.h. Beispiele, die weder wahr noch falsch beweisbar sind, ist durch

  $$\mathbf{pred}(h) := \{(c_1, \ldots, c_n) \mid P(c_1, \ldots, c_m) \,\&\, \mathit{unknown}\,(q(c_1, \ldots, c_n))\}$$

  gegeben.

- Die Menge der Gesamtbeispiele der Prämissen von h ist durch

  $$\mathbf{total}(h) := \{(c_1, \ldots, c_n) \mid P(c_1, \ldots, c_m)\} \;=\; \mathit{pos}(h) \cup \mathit{neg}(h) \cup \mathit{pred}(h)$$

  gegeben.

- Die Menge der Beispiele der Konklusion, die von h nicht abgedeckt sind, ist durch

  $$\mathbf{unc}(h) := \{(c_1, \ldots, c_n) \mid q(c_1, \ldots, c_n)\} \setminus \{(c_1, \ldots, c_n) \mid P(c_1, \ldots, c_m)\}$$

  gegeben.

- Die Menge der Beispiele der Konklusion q ist durch

  $$\mathbf{concl}(h) := \{(c_1, \ldots, c_n) \mid q(c_1, \ldots, c_n)\}$$

  gegeben.

Die Bewertungsmaße einer Hypothese sind durch die Kardinalität der obigen Mengen gegeben. Für diese Bewertungsmaße gelten die folgenden Relationen:

Satz 3.13

Seien h, h' zwei Hypothesen mit $h \geq_\Theta h'$. Dann gelten die folgenden Beziehungen:

$$\begin{aligned} &pos(h) \supseteq pos(h'), && neg(h) \supseteq neg(h') \\ &pred(h) \supseteq pred(h'), && total(h) \supseteq total(h') \\ &unc(h) \subseteq unc(h'), && concl(h) = concl(h') \end{aligned}$$

Beweis:

Der Beweis der einzelnen Relationen ist mit Hilfe der Definition zur Θ-Subsumtion offensichtlich. □

Akzeptanzkriterium

Das Akzeptanzkriterium wird durch den Benutzer mit Hilfe der Kardinalitäten der obigen Bewertungsmaße sowie mit arithmetischen Vergleichsoperatoren (d.h. $=, <, \leq, >, \geq$) und arithmetischen Veränderungsoperatoren (d.h. $+, -, \cdot, /$) definiert. Dabei können einzelne Ausdrücke durch Konjunktion und Disjunktion zusammengefasst werden.

Beispiel 3.14 (Beispiel eines Akzeptanzkriteriums)

$$|pos(h)| > 5\,,\ |neg(h)| < 2\,,\ |pos(h)|/|total(h)| > 0.7\,,$$
$$|pred(h)|/|pos(h)| < 0.3\,,\ |unc(h)|/|concl(h)| < 0.5$$

Durch die Gestaltung des Akzeptanzkriteriums kann der Benutzer die Zuverlässigkeit bzw. Unzuverlässigkeit der Regeln festlegen. Insbesondere hat er die Möglichkeit, eine gewisse Menge von negativen Beispielen zuzulassen oder auch ein minimale Anzahl von positiven Beispielen (größer als 1) zu verlangen.

Eine akzeptierte Hypothese h ist ein vollständig instanziiertes Regelschema und somit eine gefundene Regel. Bedingt durch die Relationen aus Satz 3.13 ist eine weitere Betrachtung der Spezialisierungen von h nicht sinnvoll, da dadurch lediglich redundante Regeln gefunden werden.

Beschneidungskriterium

Das Beschneidungskriterium lehnt im Rahmen der inkrementellen Instanziierung alle Hypothesen ab, deren instanziierter Teil dem Kriterium nicht genügt. Da die Hypothesen gemäß Θ-Subsumtion geordnet erzeugt werden, müssen diese teilinstanziierten Regelschemata nicht weiter instanziiert und getestet werden.

Das Beschneidungskriterium wird mit Hilfe der Relationen aus Satz 3.13 und unter Beachtung der Bewertungsmaße aus dem Akzeptanzkriterium aufgestellt.

Beispiel 3.15

Zu dem Akzeptanzkriterium aus Beispiel 3.14 leitet sich das Beschneidungskriterium

$$|pos(h)| \leq 5, \ |unc(h)|/|concl(h)| \geq 0.5$$

ab.

Bei „einfachen“ Akzeptanzkriterien wird das Beschneidungskriterium durch eine einfache Negation hergeleitet. Allerdings sind innerhalb dieses Kriteriums die Bewertungsmaße $|neg(h)|$ und $|pred(h)|$ nicht sinnvoll, da Bedingungen an eine minimale Anzahl von negativen oder unbekannten Beispielen für die Aussortierung nicht relevant sind.

3.2.3. Algorithmus

Der Algorithmus von RDT beruht auf der Hierarchie der Regelschemata, der Hypothesengenerierung und dem Test der Hypothesen. Wir wenden uns zunächst der Regelschematahierarchie zu, um im Anschluß die Hypothesengenerierung zu spezifizieren. Aufbauend darauf kommen wir zu der konkreten Ablaufbeschreibung des Algorithmus. Den Test der Hypothesen werden wir in der Ablaufbeschreibung erneut aufgreifen.

Da in RDT innerhalb der Regelschemata Konstanten und Vergleichsoperatoren verwendbar sind, erläutern wir deren Nutzung und Verarbeitung innerhalb des Algorithmus.

Hierarchie der Regelschemata

Auf der Menge der Regelschemata kann eine (Halb-)Ordnung definiert werden.

Definition 3.16 (Generalisierungsbeziehung $\geq_{\mathcal{RS}}$)

Seien R und R' Regelschemata, σ eine Substitution bezüglich Termvariablen und Σ eine Substitution bzgl. Prädikatenvariablen, so daß Σ verschiedenen Prädikatenvariablen nicht gleichsetzt. Dann heißt R genereller als *R' ($R \geq_{\mathcal{RS}} R'$) genau dann, wenn gilt*

$$\exists \sigma, \Sigma : R\sigma\Sigma \subseteq R'$$

Im allgemeinen sind „kürzere“ Regelschemata (mit einer kleineren Anzahl von Prämissen) genereller als „längere“ Regelschemata. Mit Hilfe dieser Ordnung kann eine Hierarchie der Regelschemata erzeugt werden.

Das folgende Beispiel beschreibt eine Menge von Regelschemata, in Abbildung 3.1 wird die zugehörige Hierarchie dargestellt. Dieses Beispiel ist aus [Morik 1999] entnommen.

Beispiel 3.17 (Hierarchie der Regelschemata)

Gegeben seien die folgenden Regelschemata:

$$\begin{aligned}
mp1(Q) :&\quad Q(X) \leftarrow \quad .\\
mp2(Q,P) :&\quad Q(X) \leftarrow \quad P(X).\\
mp3(Q,R) :&\quad Q(X) \leftarrow \quad R(X,Y).\\
mp4(Q,P1,P2) :&\quad Q(X) \leftarrow \quad P1(X),\ P2(X).\\
mp5(Q,R,P1) :&\quad Q(X) \leftarrow \quad R(X,Y),\ P1(Y).\\
mp6(Q,R1,R2) :&\quad Q(X) \leftarrow \quad R1(X,Y),\ R2(X,Z).\\
mp7(Q,P2,R,P1) :&\quad Q(X) \leftarrow \quad P2(Y),\ R(X,Y),\ P1(Y).\\
mp8(Q,R1,R2,P1) :&\quad Q(X) \leftarrow \quad R1(X,Y),\ R2(X,Z),\ P1(Y).\\
mp9(Q,P2,R1,P1,R2,P3) :&\quad Q(X) \leftarrow \quad P2(Y),\ R1(X,Y),\ P1(Y),\\
&\qquad\qquad\quad R2(X,Z),\ P3(Z).
\end{aligned}$$

Dann gilt beispielsweise: $mp3 \geq_{\mathcal{RS}} mp6$ mit $\Sigma_1 = \{R|R2\}$ und $\sigma_1 = \{Y|Z\}$ oder $\Sigma_2 = \{R|R1\}$ und $\sigma_2 = \emptyset$.

Die Hierarchie der Regelschemata und die Prämissenordnung in jedem Regelschema hängen ausschließlich von den vorhandenen Regelschemata ab und sind insbesondere unabhängig von der Lernaufgabe.

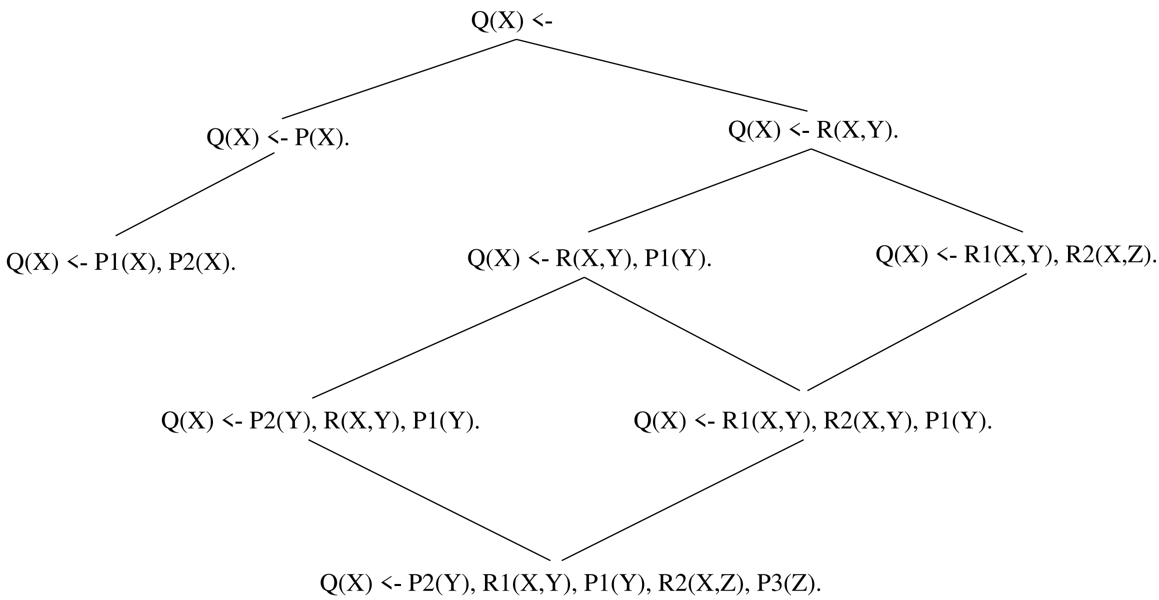


Abbildung 3.1.: Beispiel einer Hierarchie von Regelschemata gemäß $\geq_{\mathcal{RS}}$

Hypothesengenerierung aus einem Regelschema

Um aus einem Regelschema eine Regel zu erzeugen, wird eine Instanziierung des Regelschemas durchgeführt.

Definition 3.18 (Instanziierung)

> *Sei P eine Prädikatenvaribale, p ein Prädikatensymbol und $\mathcal{RS}$ ein Regelschema. Eine* Instanziierung Σ *ist eine endliche Menge von Paaren $P|p$ wobei P und p die gleiche Anzahl von Termvariablen besitzen. Mit $\mathcal{RS}\Sigma$ wird die Anwendung der Instanziierung auf das Regelschema bezeichnet. Dabei werden alle Pädikatenvariablen durch die zugeordneten Prädikatensymbole ersetzt.*

Sind in einem Regelschema Konstanten enthalten, werden diese erst nach der vollständigen Instanziierung der Prädikatenvariablen betrachtet. Dazu werden alle möglichen konsistenten Belegungen der Konstanten in den jeweiligen Prädikaten eingesetzt.

Im Algorithmus werden die Regelschemata schrittweise instanziiert und getestet. Die Reihenfolge der Prädikate zur Instanziierung ist durch die Prämissenordnung $\geq_{\mathcal{P}}$ gegeben. Durch diese Reihenfolge der Instanziierung wird sichergestellt, daß immer das Prädikat als nächstes instanziiert wird, welches über die bereits instanziierten Prädikate mit der Konklusion verbunden ist. Dadurch wird zum einen garantiert, daß keine unnötigen Instanziierungen durchgeführt werden, wenn die verbindenden Prädikate nicht existieren, und zum anderen, daß die Auswahl der möglichen Instanziierungen durch die verbindenen Prädikate eingeschränkt wird. Zusätzlich werden die teilinstanziierten Hypothesen gemäß $\vdash_{\Theta}$ geordnet erzeugt.

Erfüllt eine der inkrementell erzeugten Hypothesen – genauer der instanziierte Teil davon – den Test nicht, so müssen alle daraus resultierenden Hypothesen nicht mehr untersucht werden, da sie den Test auch nicht bestehen können.

Das folgende Beispiel veranschaulicht die Instanziierung:

Beispiel 3.19

Gegeben sei das Regelschema

$$mp(P1, P2, Q) : P2(X, Y) \,\&\, P1(Y) \quad \rightarrow \quad Q(X),$$

die Prädikatenvariable Q sei durch das Prädikat $reich$ gebunden und die Prädikate $verheiratet(A, B)$ und $prinz(C)$ seien in der Wissensdatenbank von MOBAL enthalten. Dann ist eine mögliche Instanziierung:

$$\Sigma = \{Q \mid reich, P2 \mid verheiratet, P1 \mid prinz\}$$

Die resultierende Regel lautet:

$$verheiratet(X, Y) \,\&\, prinz(Y) \quad \rightarrow \quad reich(X)$$

Bei der schrittweisen Instanziierung werden die einsetzbaren Prädikatensymbole durch die gleiche Stelligkeit, die Prädikatenkompatibilität und die Sortenkompatiblität bestimmt.

Konstanten und Vergleichsprädikate

Bei bestimmten Anwendungen kann es sinnvoll sein, Regeln mit Konstanten und/oder mit Vergleichsoperatoren zu lernen.
Mögliche Regeln sind zum Beispiel (vgl. [Kietz 1996, Seite 97f]):

$$place\_description(X, bus\_lane) \rightarrow no\_parking(X)$$

$$human(X) \,\&\, body\_temp(X, T) \,\&\, gt(T, 37.5) \rightarrow fever(X)$$

Das Konstantenlernen wird in RDT auf markierten Termvariablen des Regelschemas durchgeführt. Die dazu benötigten Termvariablen werden durch die Aufnahme in den Kopf des Regelschemas markiert. Der Algorithmus in RDT führt das Lernen der Konstanten erst dann durch, wenn alle Prädikatenvariablen bereits durch Prädikatensymbole ersetzt sind, d.h. das Regelschema ist bzgl. der Prädikatenvariablen vollständig instanziiert und hat bereits das Beschneidungskriterium bestanden. Dadurch wird das Konstantenlernen nur auf einem kleinen Teil des Hypothesenraumes angewandt.

Bei der Verwendung von Vergleichsoperatoren bzw. -prädikaten werden diese direkt in das Regelschema aufgenommen. Da sie keine Prädikatenvariablen darstellen und insbesondere bei der Instanziierung nicht berücksichtig werden, erscheinen sie nicht im Regelschemakopf. Zusätzlich müssen sie dem System als Vergleichsprädikate bekannt sein, so daß RDT diese Prädikate auswertet und nicht in der Wissensbasis nach ihnen sucht.

Um die obigen Regeln in RDT zu lernen, müssen die folgenden Regelschemata definiert sein:

$$mp1(R, T, C) : R(X, T) \rightarrow C(X)$$

$$mp2(C, R, T, C2) : C(X) \,\&\, R(X, Y) \,\&\, gt(Y, T) \rightarrow C2(X)$$

In MOBAL können die folgenden Vergleichsprädikate verwendet werden:

$$\begin{array}{rr} \mathbf{eq}(X, Y) \doteq X = Y & \mathbf{ne}(X, Y) \doteq X \neq Y \\ \mathbf{lt}(X, Y) \doteq X < Y & \mathbf{gt}(X, Y) \doteq X > Y \\ \mathbf{le}(X, Y) \doteq X \leq Y & \mathbf{ge}(X, Y) \doteq X \geq Y \end{array}$$

Ablaufbeschreibung

Nachdem wir in den vorangegangenen Abschnitten einige wichtige Aspekte des RDT-Algorithmus hervorgehoben haben, werden wir nun den vollständigen Ablauf beschreiben. Der in diesem Abschnitt verwendete Pseudo-Programmiercode ist mit

Algorithmus 1 Äußere RDT-Schleife

```
rdt(Q):
  set RS to ∅
  set LEAVES to ∅
  for all known rule models R do
    if the conclusion C of R is unifiable with Q then
      push RΣ, where Σ = {C|Q} onto RS
    end if
  end for
  while RS ≠ ∅ do
    pop a most general (with respect to ≥_RS) rule model R from RS
    instantiate-and-test( {R}, TOO-GENERAL)
    for all X ∈ RS which are specializations of R do
      pop X from RS
      for all Y ∈ TOO-GENERAL do
        for all really different Σ : Yσ ≥_RS XΣ do
          push XΣ onto RS
        end for
      end for
    end for
  end while
```

Algorithmus 2 RDT-Instanziierung

```
instantiate-and-test(HYPO,TOO-GENERAL)
  while HYPO ≠ ∅ do
    pop a hypothesis R from HYPO
    if there are a premises with uninstantiated predicates in R then
      let PREM be the smallest (with respect to ≤_P) of these premises
         and P its predicatevariable
      for all predicates p which are
               (arity-compatible with P) and
               (topology-compatible with the conlusion of R) and
               (sort-compatible with R) do
        set R to RΣ with Σ = {R|p}
        test(R)
      end for
    else
      select a constant T to learn with smallest δ(T)
      for all terms t suitable for T do
        set R to Rσ with σ = {T|t}
        test(R)
      end for
    end if
  end while
```

leichten Änderungen aus [Kietz 1996, Seite 99f] entnommen.

Durch den Benutzer wird ein Zielprädikat Q vorgegeben. Aus allen bekannten Regelschemata werden diejenigen ausgewählt, deren Konklusion durch das Zielprädikat substituierbar ist. Über diesen Regelschemata wird eine Hierachie aufgebaut. Ausgehend von einem generellsten Regelschema wird die Hierarchie Top-Down per Breitensuche untersucht. Diese äußere Schleife ist als Pseudo-Programmcode in Algorithmus 1 dargestellt. Wichtig ist dabei, daß bei dem Übergang zu dem jeweils nächsten Regelschema nur die als zu generell festgelegten Hypothesen weiter betrachtet werden. Der aktuelle Suchstatus des Algorithmus ist durch die Hypothesenschlange RS und die Blätterliste $LEAVES$ gegeben.

Die Hypothesengenerierung wird wie in Abschnitt 3.2.3 beschrieben durchgeführt. Algorithmus 2 stellt das dort beschriebene Vorgehen als Pseudo-Programmcode dar. Der Test der Hypothesen erfolgt schrittweise analog zu den Ausführungen in Abschnitt 3.2.2 auf Seite 24. Dabei wird bei teilinstanziierten Hypothesen zunächst überprüft, ob bereits eine generellere Regel in der Liste $LEAVES$ (d.h. ein Regel, die bereits akzeptiert oder aussortiert wurde) enthalten ist. Ist dies der Fall, so ist eine weitere Betrachtung der Hypothese unnötig. Sonst wird der bereits instanziierte Teil der Hypothese gegen das Beschneidungskriterium getestet und ggf. als zu speziell aussortiert. Der darauffolgende Akzeptanztest wird nur auf vollinstanziierte Hypothesen angewendet. Wird die Hypothese akzeptiert, haben wir eine Regel gefunden, sonst ist die Hypothese als zu generell eingestuft und muß

Algorithmus 3 RDT-Test

```
test(R):
  if the instantiated part of R is not a specialization (⊢Θ) of a element of LEAVES
  then
    test the instantiated part of R against the acceptance criterion
    if R is not to special {i.e. is acceptable with respect to pos, total and unc}
    then
      if all premises of R are instantiated then
        if R is acceptable with respect to all criteria then
          push R onto LEAVES
          assert R in the KB
        else
          push R onto TOO-GENERAL
        end if
      else
        push R onto HYPO
      end if
    else
      push R onto LEAVES
    end if
  end if
```

mit einem spezielleren Regelschema weiter untersucht werden. Dieses Vorgehen ist in Algorithmus 3 als Pseudo-Code dargestellt.

3.3. Rule Discovery Tool für Datenbanken (RDT/DB)

RDT/DB ist eine Weiterentwicklung von RDT mit einem direkten Zugriff auf die kommerzielle Datenbank Oracle in der Version 7 und 8. Dabei ist RDT/DB weiterhin in der Werkbank MOBAL integriert. Motiviert war diese Erweiterung durch eine Zielsetzung im Prozeß der Wissensentdeckung in Datenbanken: die Analyse einer sich in Benutzung befindenden Datenbank.

Der Hypothesenraum bei RDT/DB wird durch die gleiche deklarative Beschreibung der Hypothesensprache wie in RDT eingeschränkt. Im Unterschied zu RDT wird der Hypothesentest allerdings auf der Datenbank durchgeführt. Dazu müssen Abbildungen zwischen den Tabellen und Prädikaten erzeugt werden. In Abschnitt 3.3.1 werden wir diese Prädikatenbildung beschreiben. Zusätzlich müssen die Anfragen an die Datenbank, d.h. die SQL-Befehle zur Bestimmung der Bewertungsmaße, bestimmt werden.

Die erste Implementierung zu RDT/DB wurde im Rahmen der Diplomarbeit von [Lindner 1994] durchgeführt. Aufbauend auf dieser Arbeit wurde am Lehrstuhl für Künstliche Intelligenz der Universität Dortmund ein neue Version implementiert, die eine erweiterte Funktionalität bietet. In [Brockhausen und Morik 1998] wird diese Version im Überblick beschrieben. Eine genauere Beschreibung, insbesondere im Bezug auf die Prädikatenbildung und die Generierung der SQL-Anweisungen aus diesen Prädikaten, wird in [Morik und Brockhausen 1997] vorgestellt.

Eine weitere Erweiterung in RDT/DB stellt die Ausnutzung von funktionalen Abhängigkeiten und unären Inklusionsabhängigkeiten dar. Die grundlegenden Ideen wurden in [Brockhausen 1994] erarbeitet. Auf diese Erweiterungen werden wir in dieser Arbeit nicht eingehen, da sie im weiteren Verlauf nicht benötigt werden.

3.3.1. Prädikatenbildung

Die Prädikatenbildung bei RDT/DB erfolgt durch die Definition von Abbildungen zwischen den Tabellen in der Datenbank und den Prädikaten für das Lernverfahren. Dazu wird zunächst die Struktur der Datenbank, welche alle Tabellen mit zugehörigen Spalten und Primärschlüssel beinhaltet, ausgelesen und für die weitere Verwendung in RDT/DB gespeichert. Dabei werden ausschließlich die Informationen zu einer Tabelle und nicht die Inhalte (Tupel) der Tabellen gespeichert. Aufbauend auf diesen Informationen erfolgt in einem weiteren Schritt die Erzeugung der Prädikate gemäß vorgegebener oder durch den Benutzer definierter Abbildungen. Für diese erzeugten Prädikate werden wiederum Informationen zu den zugehörigen Tabellen und der benutzten Abbildung gespeichert. Durch diese Form der Informationsspeicherung ist es zu jeder Zeit möglich, Tabellen mit Prädikaten und umgekehrt Prädikate mit Tabellen in Relation zu stellen. Zusätzlich ist der verwendete Abbildungstyp transparent.

In dieser Arbeit werden wir lediglich auf die vier Standard-Abbildungen zwischen Tabellen (Relationen) einer Datenbank und Prädikaten zum Lernen in RDT eingehen. Im allgemeinen können in RDT/DB beliebige Abbildungen festgelegt werden oder bestehende Abbildungen beliebig kombiniert werden.

Alle hier aufgeführten Abbildungen entsprechen einer bestimmten Form der Abbildung:

Definition 3.20 (Allgemeine Abbildung)

Seien $\mathcal{T}$ die Menge aller Tabellen der Datenbank und $\mathcal{P}$ die Menge aller Prädikate des Lernverfahrens. Dann wird eine Abbildung der folgenden Form als Mapping *bezeichnet:*

$$map : \mathcal{T} \to \mathcal{P}$$

Die einfachste Abbildung einer Tabelle erfolgt 1 zu 1, d.h. eine Tabelle wird mit allen Attributen auf ein Prädikat mit den Attributnamen als Termvariablen abgebildet.

Definition 3.21 (Abbildung Typ 1)

Sei R eine Relation mit den Attributen $A_1, \ldots, A_n$ und dem Namen rn. Dann ist die Abbildung vom Typ 1 definiert als:

$$R \quad \mapsto \quad rn\,(A_1, \ldots, A_n)$$

Bei dieser Form der Abbildung besteht die Gefahr, daß wir sehr allgemeine Regeln erhalten können, da alle Variablen der Konklusion allquantifizierbar sind.

Das folgende Beispiel veranschaulicht diese Abbildung.

Beispiel 3.22

Als Beispiel benutzen wir die Relation/Tabelle person *mit den Attributen* id, name *und* geschlecht. *Mit einem Mapping vom Typ 1 ergibt sich die folgenden Abbildung:*

person → *person(id,name,geschlecht)*

id	*name*	*geschlecht*
1	*Peter Müller*	*m*
⋮	⋮	⋮

Bilden wir allerdings jedes Attribut einer Tabelle auf ein eigenes Prädikat ab, vermeiden wir solche allquantifizierten Variablen. Betrachten wir zusätzlich den Primärschlüssel mit seinen Attributen einer Tabelle, so sind die folgenden Abbildungen sinnvoll.

Eine Abbildung, die sowohl jedes (Nichtschlüssel-)Attribut auf ein eigenes Prädikat abbildet als auch die Primärschlüsselattribute berücksichtigt, ist durch den folgenden Typ 2 gegeben.

Definition 3.23 (Abbildung Typ 2)

Seien R eine Relation mit den Attributen $A_1, \ldots, A_n$ und dem Namen rn, a_x der Attributname für A_x und $A_j, \ldots, A_l$ die Primärschlüsselattribute der Relation. Dann ist die Abbildung vom Typ 2 definiert als:

$$\forall x \in [1, \ldots, n] \setminus [j, \ldots, l] : R \quad \mapsto \quad rn\_a_x\,(A_j, \ldots, A_l, A_x)$$

Die Anzahl der erzeugten Prädikate mit der Abbildung vom Typ 2 ist durch die Kardinalität der Menge der Nichtschlüsselattribute gegeben.

Beispiel 3.24

Für die Tabelle aus Beispiel 3.22 und dem Primärschlüssel id *erfolgt bei einem Mapping Typ 2 die folgende Abbildung:*

person → *person_name(id,name)*, *person_geschlecht(id,geschlecht)*

id	*name*	*geschlecht*
1	*Peter Müller*	*m*
⋮	⋮	⋮

In den obigen Abbildungen sind ausschließlich Informationen zu der Tabelle, d.h. Tabellenname und Attributnamen sowie der Primärschlüssel einer Tabelle, berücksichtigt worden. Im folgenden werden die konkreten Werte in der betrachteten Spalte einer Tabelle bei den Abbildungen berücksichtigt. Dadurch reduziert sich die Ausdrucksmächtigkeit auf die Aussagenlogik. Ein Lernen von Konstanten ist somit hinfällig.

Zunächst betrachten wir alle unterschiedlichen Werte eines Nichtschlüsselattributes zur Definition einer Abbildung. Diese spezielle Belegung wird im Prädikatnamen aufgenommen.

Definition 3.25 (Abbildung Typ 3)

Seien R eine Relation mit den Attributen $A_1, \ldots, A_n$ und dem Namen rn, a_x der Attributname für A_x mit den Werten $a_{x1}, \ldots, a_{xm}$ und $A_j, \ldots, A_l$ die Primärschlüsselattribute der Relation. Dann ist die Abbildung vom Typ 3 definiert als:

$$\forall x \in [1, \ldots, n] \setminus [j, \ldots, l] : \forall i \in [1, \ldots, m] : R \quad \mapsto \quad rn\_a_x\_a_{xi}\,(A_j, \ldots, A_l)$$

Eine solche Abbildung ist nur bei einer überschaubaren Anzahl von Werten für die Nichtschlüsselattribute sinnvoll. In unserem Beispiel ist die Anwendung der Abbildung vom Typ 3 auf das Attribut *geschlecht* durch die geringe Anzahl von Werten „m“ und „w“ äußerst sinnvoll.

Beispiel 3.26

Für die Tabelle aus Beispiel 3.24 und dem Primärschlüssel id *liefert ein Mapping vom Typ 3 die folgende Abbildung:*

person ⟶ *person_name_Peter_Müller(id)*, *person_name_Eva_Schmidt(id)*, *person_geschlecht_m(id)*, *person_geschlecht_w(id)*, ⋮

id	*name*	*geschlecht*
1	*Peter Müller*	*m*
2	*Eva Schmidt*	*w*
⋮	⋮	⋮

Zusätzlich wird bei RDT/DB durch den Algorithmus NUM_INT eine Diskretisierung von numerischen Werten durchgeführt. Mit Hilfe dieser Diskretisierung wird die Abbildung vom Typ 4 definiert:

Definition 3.27 (Abbildung Typ 4)

Seien R eine Relation mit den Attributen $A_1, \ldots, A_n$ und dem Namen rn, a_x der Attributname für A_x mit den Werten $a_{x1}, \ldots, a_{xm}$ und $< a_{x1p}, a_{x1q} >, \ldots, < a_{xmp}, a_{xmq} >$ eine Intervallzerlegung dieser Werte und $A_j, \ldots, A_l$ die Primärschlüsselattribute der Relation. Dann ist die Abbildung vom Typ 4 definiert als:

$$\forall x \in [1, \ldots, n] \setminus [j, \ldots, l] : \forall i \in [1, \ldots, m] :$$
$$R \mapsto rn\_a_x\_ < a_{xip}, a_{xiq} > (A_j, \ldots, A_l)$$

Beispiel 3.28

Wir verändern unsere Tabelle aus Beispiel 3.26, indem wir das Attribut geschlecht *durch* alter *ersetzen. Dann ist die Abbildung vom Typ 4 (mit der Intervallzerteilung $< 10, 20 >, < 20, 35 >, \ldots, < 60, 72 >$) wie folgt gegeben:*

person

id	*name*	*alter*
1	*Peter Müller*	*35*
2	*Eva Schmidt*	*16*
⋮	⋮	⋮
n	*Fritz Walter*	*72*

$\longrightarrow$ *person_alter_<10,20>(id)*
person_alter_<20,35>(id)
⋮
person_alter_<60,72>(id)

Zusammen mit der Bestimmung einer Menge von Regelschemata ist durch die Wahl einer Abbildung der Hypothesenraum bestimmt. Insbesondere läßt sich eine obere Schranke für die Größe des Hypothesenraumes angeben (siehe Abschnitt 3.3.3).

Durch die Abbildungsvorschriften der Typen 2-4 werden eine große Menge von Prädikaten mit der gleichen Stelligkeit erzeugt. Dadurch reduziert sich die Anzahl der zu konstruierenden Regelschemata.

3.3.2. Hypothesentest auf der Datenbank

Der Test der Hypothesen wird bei RDT/DB auf der Datenbank durchgeführt. Dazu wird aus dem bereits instanziierten Teil des Regelschemas zusammen mit der Abbildung der Prädikate je eine SQL-Anfrage für jedes Bewertungsmaß generiert.

Bewertungsmaße

Bei RDT/DB reduzieren sich die Bewertungsmaße auf vier Primitive, die durch die Kardinalität der folgenden Mengen gegeben sind (vgl. Abschnitt 3.2.2):

- Die Menge der Beispiele, für die die Hypothese h wahr ist, ist durch

$$\mathbf{pos}(h) := \{(c_1, \ldots, c_n) \mid P(c_1, \ldots, c_m) \,\&\, q(c_1, \ldots, c_n)\}$$

  gegeben.

- Die Menge der Beispiele, für die die Prämissen und die Negation der Konklusion wahr sind, ist durch

$$\mathbf{neg}(h) := \{(c_1, \ldots, c_n) \mid P(c_1, \ldots, c_m) \,\&\, \mathit{not}\,(q(c_1, \ldots, c_n))\}$$

  gegeben.

- Die Menge der Beispiele, für die die Konklusion q wahr ist, ist durch

$$\mathbf{concl}(h) := \{(c_1, \ldots, c_n) \mid q(c_1, \ldots, c_n)\}$$

  gegeben.

- Die Menge der Beispiele, für die die Negation der Konklusion q wahr ist, ist durch

$$\mathbf{negconcl}(h) := \{(c_1, \ldots, c_n) \mid \mathit{not}\,(q(c_1, \ldots, c_n))\}$$

  gegeben.

Die Bestimmung der Werte für $|concl(h)|$ und $|negconcl(h)|$ ist nicht für jede Hypothese notwendig, sondern nur einmalig für alle Hypothesen mit der gleichen Konklusion. Bei RDT/DB sind somit zwei SQL-Anfragen je Hypothese (für $|pos(h)|$ und $|neg(h)|$) notwendig. Hinzu kommen zusätzlich zwei Anfragen bei der Instanziierung der Konklusion.

Generierung der SQL-Anfragen

Die Generierung der SQL-Anweisung zum Auslesen der Bewertungsmaße aus der Datenbank erfolgt mit Hilfe der instanziierten Prädikate einer Hypothese. Es sei noch einmal erwähnt, daß bei RDT/DB der Hypothesentest bereits auf teilinstanziierten Regelschemata durchgeführt wird. Zusätzlich wird die SQL-Anweisung durch den zugehörigen Abbildungstyp der beteiligten Literale bestimmt.

Um auf die zu einem abgebildeten Prädikat zugehörigen Tabellen zu referenzieren, definieren wir zunächst eine Hilfsabbildung von einem Prädikat auf die zugehörige Tabelle und zusätzlich eine Abbildung zum Auslesen der Primärschlüsselattribute eines Prädikates. Diese Abbildungen benutzen die zu einem Prädikat gespeicherten Informationen, die assoziierte Tabelle und den verwendeten Abbildungstyp. Wir wollen diese Menge an Informationen mit $\mathcal{D}$ bezeichnen.

Definition 3.29

Seien p ein Prädikat, P eine Konjunktion von Prädikaten $p_1, \ldots, p_n$, t ein Tabellenname, T eine durch Kommata getrennte Auflistung mehrerer Tabellennamen und A eine durch Kommata getrennte Auflistung von Spaltennamen. Dann definieren wir die folgenden Abbildungen:

- $\mathbf{tab(p)} : p \overset{\mathcal{D}}{\mapsto} t$

  Eine Abbildung, die ein Prädikat p auf einen Tabellennamen t der zugehörigen Tabelle abbildet.

- $\mathbf{tab(P)} : P \overset{\mathcal{D}}{\mapsto} T$

  Eine Abbildung, die eine Prädikatenkonjunktion P auf eine Tabellenauflistung T der zugehörigen Tabellen der einzelnen Prädikate abbildet.

- $\mathbf{pk(p)} : p \overset{\mathcal{D}}{\mapsto} A$

  Eine Abbildung, die ein Prädikat p auf eine Spaltennamenauflistung A abbildet, so daß die Spaltennamen den Namen der Primärschlüsselattributen der zugehörigen Tabelle entsprechen.

Zusätzlich benötigen wir eine Funktion, die aus einem einzelnen Prädikat bzw. einer Konjunktion von Prädikaten die SQL-Bedingung für die SELECT-Anweisung generiert.

Definition 3.30

Seien p, P wie oben festgelegt und C eine durch AND verknüpfte Auflistung von (SQL-)Bedingungen. Dann definieren wir die folgenden Abbildungen:

- $\mathbf{cond(p)} : p \overset{\mathcal{D}}{\mapsto} C$

  Eine Abbildung, die ein Prädikat p auf eine Bedingungsauflistung C der zugehörigen Tabelle abbildet.

- $\mathbf{cond(P, q)} : P, q \overset{\mathcal{D}}{\mapsto} C$

  Eine Abbildung, die eine Prädikatenkonjunktion P und ein zusätzliches Prädikat q auf eine Bedingungsauflistung C der zugehörigen Tabellen der einzelnen Prädikate abbildet.

Die Bedingungsauflistung ist abhängig von der Anzahl der Prädikate und deren gemeinsamen Termvariablen sowie von den gewählten Abbildungstypen der einzelnen Prädikate. Sind zusätzlich Symbole anstelle von Termvariablen in den Prädikaten enthalten, so werden diese ebenfalls berücksichtigt.

1. Eine Termvariable kann in unterschiedlichen Prädikaten der Hypothese enthalten sein. Diese mehrmalige Verwendung muß durch Bedingungen erfaßt

werden. Dazu werden Gleichheitsbeziehungen zwischen den Spalten der assoziierten Tabellen aufgestellt.

2. Bei einigen der obigen Abbildungstypen ergeben sich zusätzliche Bedingungen zur Einschränkung der betrachteten Tupel der zugehörigen Tabelle. In der relationalen Algebra wird dabei von Selektion gesprochen. Bei dem Abbildungstyp 3 wird eine Einschränkung auf einen bestimmten Wert der Spalte und bei Abbildungstyp 4 auf ein bestimmtes Intervall der Werte einer Spalte durchgeführt.

3. Bei Prädikaten, die bereits konkrete Symbole anstelle von Termvariablen aufweisen, werden zusätzliche Bedingungen an die entsprechende Spalte der Tabelle gestellt.

Werden zusätzlich in einem Regelschema Vergleichsprädikate (siehe Abschnitt 3.2.3 auf Seite 29) verwendet, so werden hierfür entsprechende Bedingungen auf Spaltenbasis erzeugt.

Mit Hilfe der obigen Abbildungen definieren wir die SQL-Anweisungen zu den Bewertungsmaßen für RDT/DB in einer allgemeineren Form.

Definition 3.31 (Allg. SQL-Anfragen für *pos*, *neg*, *concl* und *negconcl*)

Sei $h = P(X_1, \ldots, X_m) \rightarrow q(X_1, \ldots, X_n)$ eine Hypothese mit $m \geq n$, $P(X_1, \ldots, X_m)$ die Konjunktion der Prämissen $p_1, \ldots, p_m$, die die Termvariablen $X_1, \ldots, X_m$ verwenden, und $q(X_1, \ldots, X_n)$ die Konklusion mit den Termvariablen $X_1, \ldots, X_m$. Dann gilt:

$$
\begin{aligned}
|concl(h)| &= \text{SELECT COUNT(*) FROM } tab(q) \\
&\quad\ \text{WHERE } cond(q); \\
|negconcl(h)| &= \text{SELECT COUNT(*) FROM } tab(q) \\
&\quad\ \text{WHERE } cond\,(not(q)); \\
|pos(h)| &= \text{SELECT COUNT(*) FROM } tab(q),\, tab(P) \\
&\quad\ \text{WHERE } cond(P, q); \\
|neg(h)| &= \text{SELECT COUNT(*) FROM } tab(q),\, tab(P) \\
&\quad\ \text{WHERE } cond\,(P, not(q))\,;
\end{aligned}
$$

Betrachten wir hierzu ein Beispiel:

Beispiel 3.32 (SQL-Anweisung)

Wir betrachten die Relationen person *und* actor *aus Abbildung 2.4 auf Seite 10. Zusätzlich sei eine Relation* movie *mit genau zwei Attributen* name *und* top *gegeben. Aus diesen Relationen sind durch die Prädikatengenerierung neben anderen die Prädikate* movie_top(name), actor(pid,film) *und* person_name_Peter_Mueller(id) *erzeugt worden.*

Betrachten wir die folgende Hypothese

$$h = movie\_top(X) \leftarrow actor(Y, X), person\_name\_Peter\_Mueller(Y).$$

Die SQL-Anfragen für die einzelnen Bewertungsmaße sind wie folgt gegeben:

$|concl(h)|$ = *SELECT COUNT(*) FROM movie m*
WHERE m.top = 1;

$|negconcl(h)|$ = *SELECT COUNT(*) FROM movie m*
WHERE NOT m.top = 1;

$|neg(h)|$ = *SELECT COUNT(*)*
FROM movie m, actor a, person p
WHERE NOT m.top = 1 AND m.name = a.film
AND a.pid = p.id AND p.name = 'Peter Müller';

$|pos(h)|$ = *SELECT COUNT(*)*
FROM movie m, actor a, person p
WHERE m.top = 1 AND m.name = a.film
AND a.pid = p.id AND p.name = 'Peter Müller';

Bewertungskriterien

Das Akzeptanz- und das Beschneidungskriterium sind in gleicher Weise wie bei RDT durch den Benutzer frei definierbar. Allerdings können nur die Bewertungsmaße $|neg(h)|$, $|pos(h)|$, $|concl(h)|$ und $|negconcl(h)|$ verwendet werden.

Beispiel 3.33

Ein Beispiel für ein Akzeptanzkriterium für den Fall, daß es zwei Klassen zu Unterscheidung gibt, ist durch

$$\frac{|pos(h)|}{|pos(h)| + |neg(h)|} \geq \frac{|concl(h)|}{|concl(h)| + |negconcl(h)|}$$

*gegeben (vgl. [*Brockhausen *und* Morik *1998]).*

3.3.3. Größe des Hypothesenraumes

Die intuitive Annahme, daß die Größe des Hypothesenraumes durch die Anzahl der Tupel in der Datenbank bedingt ist, trifft nicht zu. Die Größe des Hypothesenraumes hängt von

- der Anzahl r der Regelschemata,
- der Anzahl p der Prädikate, die für die Instanziierungen der Prädikatvariablen zur Verfügung stehen sowie
- der maximalen Anzahl k der Literale in einem Regelschema ab.

Eine obere Schranke für den Hypothesenraum ist durch

$$r \cdot p^k$$

gegeben. Werden Konstanten gelernt, müssen dazu alle Werte für die jeweilige Termvariable durchprobiert werden. Dadurch ist die Größe des Hypothesenraumes zusätzlich von

- der Anzahl c der Konstanten und
- der maximalen Anzahl i der möglichen Werte für diese Konstanten abhängig.

Es ergibt sich somit eine obere Schranke für die Größe des Hypothesenraumes mit Konstantenlernen:

$$r \cdot (p \cdot i^c)^k \tag{3.1}$$

Die tatsächliche Größe des Hypothesenraumes ist abhängig von der durch den Benutzer vorgegebenen Hypothesensprache, bleibt aber grundsätzlich endlich. Insbesondere variiert die Größe des Hypothesenraumes bedingt durch die Wahl der Abbildung zwischen der Datenbank und den Prädikaten zum Teil sehr gravierend, so daß man diese sehr sorgfältig wählen sollte.

3.3.4. Algorithmus

Die wesentlichen Änderungen des Algorithmus bestehen in dem Hypothesentest auf der Datenbank und der Generierung der Prädikate aus dem Datenbankinhalt. Außerdem ist der Test auf Sortenkompatibilität durch einen Test auf Kompatibilität der Datenbanktypen ersetzt worden. Zusätzlich sind die oben erwähnten Erweiterungen bzgl. funktionaler Abhängigkeiten und unärer Inklusionsabhängigkeiten ergänzt worden, auf die wir hier nicht eingehen wollen, da sie im weiteren Verlauf der Arbeit nicht vom Interesse sind.

Datentypkompatibilität

Entsprechend der Sortenkompatibilität ist bei der Verwendung von Datenbanken die Ausnutzung der Datentypen einer Tabelle bzw. der jeweils betrachteten Spalte möglich. Dazu wird bei Literalen mit gleichen Termvariablen der Datentyp der assoziierten Tabellenspalte verglichen. Formal können wir die Datentypkompatibilität wie folgt definieren:

Definition 3.34 (Datentypkompatibilität)

Sei h eine Hypothese, $termvars(h)$ die Menge aller Termvariablen aus h, $preds(X)$ die Menge aller instanziierten Prädikate, die die Termvariable X enthalten, und $type(p, X)$ eine Funktion, die für das Prädikat p und die Termvariable X den Datentyp der Spalte der assoziierten Tabelle zurückliefert. Dann gilt:
h ist datentypkompatibel genau dann, wenn

$$\begin{aligned}
&\forall X \in termvars(R) : \\
&\quad |preds(X)| \leq 1 \ \vee \\
&\quad \forall p1, p2 \in preds(X) : p1 \neq p2 : type(p, X) \equiv type(p2, X)
\end{aligned}$$

4. RDT/DM - Konzeptionelle Erweiterungen

In RDT/DB wird die Datenbank bei der Prädikatengenerierung und beim Hypothesentest eingesetzt. Zur Generierung der Prädikate durch die in Abschnitt 3.3.1 beschriebenen Abbildungen benutzt das Verfahren die Informationen Tabellenname, Spaltenname, Werte einer Spalte und Primärschlüsselattribute zu den Tabellen. Der Hypothesentest erfolgt direkt auf den durch die Abbildungen festgelegten Tabellen. Alle weiteren Informationen aus der Datenbank wie z.B. die Fremdschlüssel werden nicht berücksichtigt. Ebenso werden die Möglichkeiten des DBMS zur Festlegung von Views auf die Daten und die damit möglichen Einschränkungen der Attribute und der Tupel sowie die Möglichkeit der Zusammenstellung von Daten nicht benutzt.

Die Fremdschlüssel einer Relation stellen – wie in Abschitt 2.1 beschrieben – eine logische Verbindung zwischen verschiedenen Relationen dar. Diese Verbindungen können bei der Hypothesengenerierung zur semantischen Einschränkung des Hypothesenraumes benutzt werden. In [Wrobel 1997] werden Fremdschlüssel in ähnlicher Weise zu Generierung von Hypothesen eingesetzt (Foreign-Links).

Innerhalb des Algorithmus werden relationale Operatoren (Selektion, Projektion oder Join) auf den gleichen Relationen und mit den gleichen Parametern mehrmals durchgeführt. Diese Mehrfachausführung ist einerseits zeitintensiv und andererseits entstehen dadurch unnötig komplexe Datenbankzugriffe. Werden hingegen für die Ergebnisrelationen dieser relationalen Operatoren einmalig Views in der Datenbank erzeugt, sind die gewünschten Tupel mit den gewünschten Attributen direkt aus der Datenbank abgreifbar.

Aus diesen Überlegungen heraus ergeben sich Teilbereiche des Algorithmus, die wir verändern bzw. erweitern:

1. Beim Auslesen des Datenbankschemas werden zusätzliche Informationen zu den Fremdschlüsseln der Relationen ausgelesen und die dadurch resultierenden logischen Verbindungen zwischen Relationen verwaltet. D.h. die von uns verwendete interne Datenbank $\mathcal{D}$ speichert diese Informationen zusätzlich neben den anderen Informationen zu dem gegebenen Datenbankschema.

2. Die Abbildungen zur Generierung der Prädikate aus den Datenbankrelationen müssen so angepaßt werden, daß die Attribute der Fremdschlüssel berücksichtigt werden.

3. Bei der Prädikatenbildung sind gemäß der Abbildungsvorschrift Views auf die Daten zu erzeugen. D.h. eine Relation wird durch Selektion und Projektion auf bestimmte Tupel und Attribute eingeschränkt, die so erzeugte neue Relation wird als View in der Datenbank gespeichert.

4. Bei der sukzessiven Instanziierung eines Regelschemas werden die Kompatibilitätstests um einen Test auf Schlüsselkompatibilität erweitert.

5. Innerhalb des Algorithmus werden wir Views an den inneren Knoten der Regelschemata-Hierarchie für alle zu generellen Hypothesen erzeugen. Diese Views werden dann bei der weiteren Betrachtung dieser Hypothesen in den spezielleren Regelschemata benutzt.

Das in Punkt 1 beschriebene Auslesen und Speichern der gesamten Informationen zu dem Datenbankschema inklusive aller Fremdschlüsselrelationen nehmen wir als gegeben an. Im weiteren Verlauf des Kapitels werden wir uns zunächst der Prädikatenbildung unter Berücksichtigung der Punkte 2 und 3 zuwenden. Im Anschluß daran gehen wir in Abschnitt 4.2 auf die Hypothesengenerierung unter Berücksichtigung der Schlüsselkompatibilität ein (Punkt 4). Danach wird im Abschnitt 4.3 die Verwendung von Views innerhalb des Lernlaufes (Punkt 5) beschrieben. Im Abschnitt 4.4 stellen wir aufbauend auf die vorigen Abschnitte den Algorithmus RDT/DM vor. Anschließend leiten wir in 4.5 eine obere Schranke zu den erzeugten Views her. Am Ende dieses Kapitels in Abschnitt 4.6 betrachten wir eine zusätzliche Erweiterung des Algorithmus, die sich aus [Morik 2002] ergibt.

4.1. Prädikatenbildung

Bei der Prädikatenbildung orientieren wir uns an der aus RDT/DB bekannten Herleitung der Prädikate durch Abbildungen und nutzen das DBMS zur Generierung von Views auf die Daten der Datenbank. In dem folgenden Abschitt werden wir zunächst erweiterte Abbildungen unter Berücksichtigung von Fremdschlüsseln besprechen, um dann in Abschnitt 4.1.2 auf die Erzeugung der Views für die jeweiligen Abbildungen einzugehen. In Abschnitt 4.1.3 werden wir einige view- und fremdschlüsselbasierte Standardabbildungen einführen.

4.1.1. Erweiterte Abbildungen

Mit Berücksichtigung der Fremdschlüssel bei der Generierung von Prädikaten wollen wir eine weitere Möglichkeit zur Beschränkung des Hypothesenraumes einführen. Dazu definieren wir zunächst eine Prädikatengenerierung mit Fremdschlüsselattributen und kommen dann in Abschnitt 4.2 auf die Kompatibilität von Hypothesen bezüglich Fremdschlüsselrelationen zu sprechen.

Bei der Abbildung einer Tabelle, die keinen Fremdschlüssel besitzt, greifen ohne Einschränkung die in Abschnitt 3.3.1 aufgelisteten Abbildungen. Werden hingegen die Fremdschlüssel einer Tabelle berücksichtigt, müssen wir die Definition der Abbildungen erweitern.

Bei der Einbeziehung von Fremdschlüsseln stellen sich die folgenden Fragen:

1. Wie werden Tabellen abgebildet, die sowohl einen Primärschlüssel als auch Fremdschlüssel besitzen?

2. Ist es sinnvoll, bei Tabellen mit mehreren Fremdschlüsseln die Fremdschlüsselattribute aller Fremdschlüssel gemeinsam abzubilden oder sollten jeweils getrennte Abbildungen durchgeführt werden?

Im Rahmen der ersten Frage bewerten wir den Primärschlüssel höher als die Fremdschlüssel und führen die Abbildung ausschließlich mit dem Primärschlüssel durch. Insbesondere ist die Modellierung in vielen Datenbanken in der Form geregelt, daß bei der Existenz von Fremdschlüsseln deren Attribute direkt oder unter Erweiterung um zusätzliche Attribute als Primärschlüssel der Relation festgelegt werden.

Die zweite Frage bezieht sich ausschließlich auf Relationen ohne Primärschlüssel. Haben solche Relationen mehrere Fremdschlüssel, bilden wir alle zugehörigen Attribute gemeinsam ab. Wir beziehen uns dabei auf das Entity-Relationship-Modell, in dem es die Unterscheidung in Objekt- und Beziehungstypen gibt. Dabei sind die Beziehungstypen durch Fremdschlüssel mit mehreren (in der Regel zwei) Objekttypen verbunden. Diese Form der Darstellung von Beziehungen zwischen Tabellen ist auch innerhalb von Regelschemata bzw. Regeln durchaus gewünscht, da dadurch zwei Prädikate über ein Verbindungsprädikat zueinander in Beziehung gesetzt werden.

4.1.2. Ausnutzung von Views in den Abbildungen

Bei RDT/DB sind die Abbildungen auf Tabellen bezogen, d.h. für alle Prädikate, die nur eine einzelne Spalte, einen bestimmten Wert innerhalb einer Spalte oder einen bestimmten Wertebereich in einer Spalte betreffen, werden trotzdem die Zugriffe über die gesamte Tabelle durchgeführt. Durch die Benutzung von Views auf die Daten können wir sowohl die Anzahl der betrachteten Tupel bzw. Attribute einschränken als auch die für den Hypothesentest zu generierenden SQL-Befehle vereinfachen.

Die Abbildung von Tabellen auf Prädikate wird bei RDT/DM zweistufig durchgeführt. Zunächst wird aus einer Tabelle ein View generiert, erst danach wird dieser View auf ein Prädikat abgebildet. Der View wird dabei gemäß der erweiterten Abbildungsvorschrift erzeugt und ermöglicht dadurch eine einfache Abbildung auf das Prädikat. Eine gesonderte Verwaltung der Abbildungen bleibt durch die unterschiedliche Generierung der SQL-Anweisungen weiterhin notwendig.

Zusätzlich bietet diese zweistufige Abbildung die Möglichkeit, aus verschiedenen Tabellen einen neuen View zu generieren und erst diesen View auf ein oder mehrere Prädikate abzubilden.

4.1.3. Beispielabbildungen

Wir werden an dieser Stelle nur drei der vier Standardabbildungen erweitern. Die Abbildung vom Typ 1 werden wir hier nicht betrachten, da bei diesem Typ die Generierung eines Views unnötig ist, denn dieser View entspräche der Tabelle selbst. Außerdem sind alle Attribute in dem generierten Prädikat enthalten, dies gilt insbesondere auch für die Fremdschlüsselattribute. Neben den hier betrachteten Standardabbildungen sind beliebige Kombinationen und weitere Abbildungen in RDT/DM möglich. Die beschriebenen Abbildungsvorschriften halten sich weiterhin an die in Definition 3.20 festgelegte allgemeine Abbildung.
Das generelle Vorgehen bei diesen erweiterten Abbildungsvorschriften besteht aus zwei Schritten:

1. Aus den Relationen werden gemäß der „normalen“ Abbildungsvorschrift des jeweiligen Typs eine Menge von Views erzeugt.

2. Diese Views werden dann ohne weitere Einschränkungen auf Prädikate abgebildet.

Definition 4.1 (Erw. Abbildung Typ 2)

Sei R eine Relation mit den Attributen $A_1, \ldots, A_n$ und dem Namen rn, a_x der Attributname für A_x, und seien $A_j, \ldots, A_l$ die Primärschlüsselattribute, wenn die Relation einen Primärschlüssel besitzt, sonst die Fremdschlüsselattribute aller Fremdschlüssel der Relation. Sei zusätzlich V_x ein View mit dem Namen $vn_x = rn\_a_x$. Dann ist die erweiterte Abbildung vom Typ 2 für alle $x \in [1, \ldots, n] \setminus [j, \ldots, l]$ definiert als:

$$\begin{aligned} R &\mapsto V_x(A_j, \ldots, A_l, A_x) \\ &\mapsto vn_x\,(A_j, \ldots, A_l, A_x) \end{aligned}$$

Definition 4.2 (Erw. Abbildung Typ 3)

Sei R eine Relation mit den Attributen $A_1, \ldots, A_n$ und dem Namen rn, a_x der Attributname für A_x mit den Werten $a_{x1}, \ldots, a_{xm}$, und seien $A_j, \ldots, A_l$ die Primärschlüsselattribute, wenn die Relation einen Primärschlüssel besitzt, sonst die Fremdschlüsselattribute aller Fremdschlüssel der Relation. Sei zusätzlich V_{xi} ein View mit dem Namen $vn_{xi} = rn\_a_x\_a_{xi}$. Dann ist die erweiterte Abbildung vom Typ 3 für alle $x \in [1, \ldots, n] \setminus [j, \ldots, l]$ definiert als:

$$\begin{aligned} \forall i \in [1, \ldots, m] : R &\mapsto V_{xi}(A_j, \ldots, A_l) \\ &\mapsto vn_{xi}\,(A_j, \ldots, A_l) \end{aligned}$$

Definition 4.3 (Erw. Abbildung Typ 4)

Sei R eine Relation mit den Attributen $A_1, \ldots, A_n$ und dem Namen rn, a_x der Attributname für A_x mit den Werten $a_{x1}, \ldots, a_{xm}$, und $< a_{x1p}, a_{x1q} >, \ldots, < a_{xmp}, a_{xmq} >$ eine Intervallzerlegung dieser Werte. Weiterhin seien $A_j, \ldots, A_l$ die Primärschlüsselattribute, wenn die Relation einen Primärschlüssel besitzt, sonst die Fremdschlüsselattribute aller Fremdschlüssel der Relation. Sei zusätzlich V_{xipq} ein View mit dem Namen $vn_{xipq} = rn\_a_x\_ < a_{xip}, a_{xiq} >$. Dann ist die erweiterte Abbildung vom Typ 4 für alle $x \in [1, \ldots, n] \setminus [j, \ldots, l]$ definiert als:

$$\begin{aligned} \forall i \in [1, \ldots, m] : R(A_1, \cdots, A_n) &\mapsto V_{xipq}(A_j, \ldots, A_l) \\ &\mapsto vn_{xipq}(A_j, \ldots, A_l) \end{aligned}$$

Die aufgeführten Abbildungen können in RDT/DM beliebig kombiniert und auf alle Tabellen und Spalten oder auf ausgewählten Tabellen bzw. Spalten von Tabellen angewendet werden.

Alle Informationen zu den Tabellen (Tabellenname, Spalten und Schlüssel), zu dem Primärschlüssel und den Fremdschlüsseln sowie zu den generierten Views werden innerhalb von RDT/DM verwaltet und gespeichert. Zusätzlich wird die bijektive Abbildung zwischen den Tabellen/Views und den Prädikaten verwaltet. Es ist somit zu jedem Zeitpunkt möglich, aus einem Prädikat den View und umgekehrt aus einem View das zugehörige Prädikat zu bestimmen. Insbesondere ist der Abbildungstyp ersichtlich. Alle diese Informationen sind in der internen Datenbank $\mathcal{D}$ abgelegt.

4.1.4. Auswirkung der Abbildungen auf die SQL-Anfragen

Wie wir in Abschnitt 3.3.2 für RDT/DB gesehen haben, sind für den Hypothesentest auf der Datenbank zur Bestimmung der Bewertungsmaße SQL-Anweisungen notwendig. Insbesondere sind diese SQL-Anweisungen von dem gewählten Abbildungstyp abhängig.

Bei RDT/DM treten folgende Änderungen aufgrund des Einsatzes von Views bei der Prädikatengenerierung auf:

1. Durch die Erzeugung von Views existieren bei allen Prämissenprädikaten keine Bedingungen, die durch den Abbildungstyp vorgegeben sind. Dieses gilt für die nicht negierte Konklusion ebenso.

2. Bei der Negation der Konklusion ändert sich die assoziierte Tabelle ($tab(q)$), und es sind zusätzliche abbildungstypabhängige Bedingungen ($cond(p)$) notwendig.

4.2. Hypothesengenerierung

Die Hypothesengenerierung verläuft analog zu der Beschreibung zu RDT/DB, d.h. die Prädikatenvariablen innerhalb des Regelschemas werden sukzessive gemäß der Prämissenordnung $\geq_{\mathcal{P}}$ substituiert. Dabei werden die möglichen Prädikatensymbole zur Substitution durch die Kompatibilität zur Stelligkeit der Prädikatenvariable und zu den Datentypen der Termvariablen eingeschränkt.

In RDT/DM schränken wir diese möglichen Prädikate weiter ein. Wie in Abschnitt 4.1 beschrieben, enthalten die aus der Datenbank generierten Prädikate grundsätzlich die Attribute des Primärschlüssels bzw. der Fremdschlüssel. Zusätzlich werden die Informationen zu dem Primärschlüssel und den Fremdschlüsseln der Relation und zu den assoziierten Prädikaten ebenfalls in unserer Informationsdatenbank $\mathcal{D}$ für die Abbildungen und dem Datenbankschema gespeichert.

Mit Hilfe dieser Informationen können wir unseren Hypothesenraum durch die Kompatibilität der Schlüssel einer Hypothese zusätzlich semantisch einschränken.

Bevor wir uns mit der Schlüsselkompatibilität von (teil-)instanziierten Regelschemata beschäften, müssen wir festlegen, unter welchen Voraussetzungen zwei Schlüssel (Primär- oder Fremdschlüssel) kompatibel sind:

Definition 4.4 (Kompatibilität zweier Schlüssel)

Seien $k1$, $k2$ Schlüssel, Primär- oder Fremdschlüssel, innerhalb einer Datenbank.
Dann ist $k1$ kompatibel *zu $k2$ ($k1 \equiv k2$), wenn eine der folgenden Aussagen zutrifft:*

1. *$k1 = k2$, die beiden Schlüssel sind identisch.*
2. *$k1$ ist Primärschlüssel und $k2$ zugehöriger Fremdschlüssel oder umgekehrt.*
3. *$k1$ und $k2$ sind Fremdschlüssel eines gemeinsamen Primärschlüssels.*

Mit Hilfe der Kompatibilität zweier Schlüssel einer Datenbank läßt sich die Schlüsselkompatibilität einer Hypothese festlegen:

Definition 4.5 (Schlüsselkompatibilität)

> *Seien h eine Hypothese, $termvars(h)$ die Menge aller Termvariablen aus h, $preds(X)$ die Menge aller instanziierten Prädikate, die die Termvariable X enthalten, und $key(p, X)$ eine Funktion, die für das Prädikat p und die Termvariable X den Schlüssel (hier nur den Primär- oder einen Fremdschlüssel), zu dem das Attribut/die Spalte der assoziierten Tabelle gehört, zurückliefert. Dann gilt:*
> *h ist* schlüsselkompatibel *genau dann, wenn*
>
> $$\begin{aligned}&\forall X \in termvars(h):\\&\quad |preds(X)| \leq 1 \;\vee\\&\quad \forall p \in preds(X): \nexists key(p, X) \;\vee\\&\quad \forall p1, p2 \in preds(X), p1 \neq p2\; \exists key(p1, X), key(p2, X):\\&\qquad key(p1, X) \equiv key(p2, X)\end{aligned}$$

In den von uns durchgeführten Versuchen ist durch diese zusätzliche Einschränkung des Hypothesenraumes eine starke Reduzierung der betrachteten Hypothesen ersichtlich (siehe Abschnitt 6.2).

4.3. Hypothesentest

In diesem Abschnitt beschreiben wir den Einsatz von Views innerhalb der Traversierung der Regelschemata-Hierarchie. Dadurch muß der Hypothesentest auf der Datenbank – genauer die Erzeugung der SQL-Anfragen für die Hypothesen — dynamisch angepaßt werden. Während der Traversierung werden Hypothesen durch das Akzeptanzkriterium abgelehnt. Diese Hypothesen sind zu generell und müssen im weiteren Verlauf in einem spezielleren Regelschema weiter instanziiert werden. An dieser Stelle erzeugen wir Views, die die durch die abgelehnten Hypothesen beschriebene Tupelmenge enthalten. Diese Views verwenden wir im weiteren Verlauf der Suche für den Hypothesentest auf der Datenbank.
Zunächst werden wir die notwendigen mehrdimensionalen Abbildungen dieser Hypothesen auf einer Menge von Relationen beschreiben und aufbauend darauf die Erzeugung der SQL-Anfragen festlegen.

4.3.1. Mehrdimensionale Abbildungen zwischen Relationen und Prädikaten

Bei der Generierung von Prädikaten haben wir bisher eindimensionale Abbildungen der Form $\mathcal{T} \rightarrow \mathcal{P}$ betrachtet, die ausgehend von einer Relation je nach Abbildungsvorschrift ein oder mehrere Prädikate erzeugen. Wollen wir allerdings mehrere unterschiedliche Relationen auf eine Konjunktion von Prädikaten abbilden, benötigen wir eine mehrdimensionale Abbildung zwischen den Relationen und den Prädikaten.

Definition 4.6 (mehrdimensionale Abbildung (Multimapping))

Sei $\mathcal{T}$ die Menge aller Tabellen der Datenbank und $\mathcal{P}$ die Menge aller Prädikate des Lernverfahrens. Dann bezeichnen wir Abbildungen der folgenden Form als Multimapping*:*

$$multimap : \mathcal{T} \times \cdots \times \mathcal{T} \to \mathcal{P} \times \cdots \times \mathcal{P}$$

Im Gegensatz zur Prädikatengenerierung, bei der wir ausgehend von einer Relation ein oder mehrere Prädikate erzeugen, gehen wir hier von einer gegebenen Regel mit mehrerern (mindestens zwei) Literalen aus. D.h. wir müssen zu einer Regel alle assoziierten Relationen (Tabellen oder Views) bestimmen und aus diesen mit Hilfe der durch die Regel vorgegebenen relationalen Operatoren eine neue Relation (View) erzeugen.

Definition 4.7 (Multimapping)

Seien h eine vollinstanziierte Hypothese (eine Regel), $p_i \in \mathcal{P}, 1 \leq i \leq m$ die Prädikate in h, $q = p_m \in \mathcal{P}$ die Konklusion, $r_1, \ldots, r_n \in \mathcal{T}, n \leq m$ die zu den Literalen assoziierten Relationen, $X_1, \ldots, X_l$ die in h verwendeten Termvariablen und $col(p, X)$ eine Funktion, die für ein Prädikat p und eine Termvariable X die Spalte der assoziierten Relation liefert. Dann definieren wir ein Multimapping durch:

$$\begin{aligned} r_1, \ldots, r_n &\mapsto v\,(col(h, X_1), \ldots, col(h, X_l)) \\ &\mapsto (p_1 \,\&\, \ldots \,\&\, p_{m-1} \to q) \end{aligned}$$

mit

$$col(h, X) := \begin{cases} col(q, X) & , \quad q \in preds(X) \\ col(p_i, X) & , \quad p_i \in preds(X), p_i \neq q \end{cases}$$

Die obige Abbildungsvorschrift legt die Attribute des Views eindeutig fest. Die Anzahl der Attribute ist durch die Anzahl der Termvariablen in der Regel bestimmt. Die Spalte ergibt sich aus den assoziierten Relationen zu den Prädikaten der Regel. Allerdings sind mehrere Spalten in unterschiedlichen Relationen für eine Termvariable möglich, da die Regel in unterschiedlichen Prädikaten dieselbe Termvariable verlangt (Termvariablen aus den Prämissen müssen mit Termvariablen der Konklusion verbunden sein). Wenn die Termvariable in mehreren Prädikaten vertreten ist, wobei die Konklusion eines dieser Prädikate ist, wird grundsätzlich das zu der Konklusion und das zu der Termvariablen assoziierte Attribut in dem View aufgenommen. Ist die Konklusion nicht in den Prädikaten enthalten, so wird ein beliebiges assoziiertes Attribut in den View eingebunden.

Durch diese Form der Viewgenerierung ist garantiert, daß alle Termvariablen für den weiteren Verlauf des Algorithmus mit den entsprechenden Attributen in der Datenbank erhalten bleiben und keine redundanten Attribute in den View aufgenommen werden.

Neben den Attributen werden durch die Regel auch die Tupel in dem View beschrieben. Die Tupel ergeben sich aus dem Verbinden mehrerer Relationen (relationale Join-Operationen) der ursprünglichen Relationen sowie durch zusätzliche Bedingungen, die durch den Typ der Abbildung zur Prädikatenbildung festgelegt sind.
Die Generierung von Views in der oben beschriebenen Form wird innerhalb des RDT/DM-Algorithmus auf alle Hypothesen angewendet, die durch das Akzeptanzkriterium abgelehnt werden (zu generelle Hypothesen) und die durch die Hierarchie der Regelschemata weiter spezialisiert werden können (innere Knoten, kein Blatt). Für jede dieser Hypothesen werden zwei Views mit der obigen Abbildungsvorschrift erzeugt, wobei wir die Konklusion einmal positiv und einmal negiert betrachten. Im Gegensatz zu der Prädikatengenerierung ist an dieser Stelle die Generierung von zwei Views sinnvoll, da durch die zusätzlichen Bedingungen die Menge der betrachteten Tupel stark eingeschränkt wird (auch bei der Negation) und wir im weiteren Lernlauf diesen Vorteil bei der Abfrage beider Bewertungsmaße $pos(h)$ und $neg(h)$ ausschöpfen wollen.

4.3.2. Reduktion der betrachteten Tupel

Durch die Erzeugung von Views während der Traversierung der Regelschemata-Hierarchie werden sowohl die in einer SQL-Anweisung involvierten Relationen als auch die Anzahl der zu betrachtenden Tupel reduziert. In diesem Abschnitt werden wir die Reduktion der Tupel veranschaulichen. In Abbildung 4.1 wird die Redu-

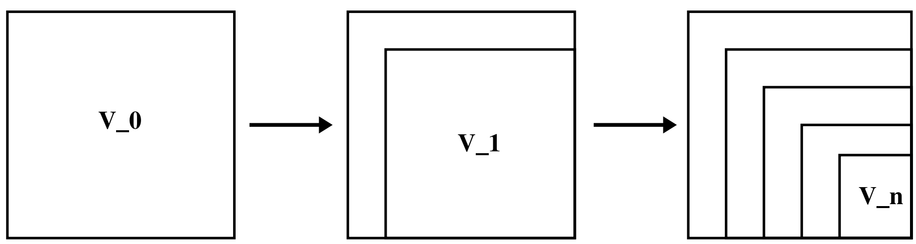


Abbildung 4.1.: Reduktion der Tupel durch Multimappings

zierung der betrachteten Tupelmengen dargestellt. V_0 ist die gesamte Tupelmenge, die wir betrachten. Alle weiteren V_i, $1 \leq i \leq n$, sind Teilmengen, die durch die Views v_i beschrieben werden. Genaugenommen sind die jeweiligen Views noch in zwei Views für die negativen und die positiven Beispiel unterteilt. Wir betrachten die durch beide Views gemeinsam beschriebene Tupelmenge. Die Reduktion im i-ten Schritt können wir durch die Formel

$$e_i = V_{i-1} - V_i$$

beschreiben.

4.3.3. Generierung der SQL-Anfragen

Bei der Erzeugung der SQL-Anweisungen müssen wir die Multimappings bei den Anfragen für $pos(h)$ und $neg(h)$ berücksichtigen. Dadurch ergibt sich, daß die von uns in Definition 3.31 dargestellte Bestimmung der Tabellen für die Konklusion ($tab(q)$) und die Konjunktion der Prämissen ($tab(P)$) hier nicht mehr ausreichend ist. Wir benötigen somit die wie folgt definierte Funktion $tab(p, P)$:

Definition 4.8

Seien p ein Prädikat, P eine Konjunktion von Prädikaten $p_1, \ldots, p_n$, r eine Relation, R eine durch Kommata getrennte Auflistung mehrerer Relationen. Dann definieren wir die folgende Abbildung:

$$\mathbf{tab(p, P)} : p, P \overset{\mathcal{D}}{\mapsto} R$$

Hierdurch wird eine Prädikatenkonjunktion P und ein zusätzliches Prädikat p auf eine Auflistung R der zugehörigen Relationen abgebildet. Dabei ist es möglich, daß mehrere Prädikate auf genau eine einzige Relation r abgebildet werden.

Mit Hilfe dieser Funktion und den Funktionen aus Abschnitt 3.3.2 können wir die erweiterten SQL-Anfragen definieren:

Definition 4.9 (Erw. SQL-Anfragen für *pos*, *neg*)

Sei $h = P(X_1, \ldots, X_m) \rightarrow q(X_1, \ldots, X_n)$ eine Hypothese mit $m \geq n$, $P(X_1, \ldots, X_m)$ die Konjunktion der Prämissen $p_1, \ldots, p_m$, die die Termvariablen $X_1, \ldots, X_m$ enthalten, und $q(X_1, \ldots, X_n)$ die Konklusion mit den Termvariablen $X_1, \ldots, X_m$. Dann gilt:

$$\begin{aligned} |pos(h)| &= \textit{SELECT COUNT(*) FROM } tab(q, P) \\ & \quad\ \textit{WHERE } cond(P, q); \\ |neg(h)| &= \textit{SELECT COUNT(*) FROM } tab(q, P) \\ & \quad\ \textit{WHERE } cond\,(P, not(q))\,; \end{aligned}$$

Durch diese Form der Erzeugung der SQL-Abfragen zu den Bewertungsmaßen werden die Multimappings berücksichtigt.

4.4. Algorithmus

Der Algorithmus zu RDT/DM orientiert sich an dem Algorithmus zu RDT bzw. RDT/DB. In der Vorverarbeitung ist die Prädikatenbildung (vgl. Abschnitt 4.1) erweitert worden. Dadurch werden Hypothesen, die nicht schlüsselkompatibel sind, direkt aussortiert (vgl. Abschnitt 4.2).

Die Hauptänderung des Algorithmus besteht allerdings in der Verwendung von Multimappings. Um die dabei erzeugten Views effektiv nutzen zu können, haben wir die Traversierung der Hierarchie der Regelschemata von Breitensuche auf Tiefensuche verändert.

Die Idee ergibt sich aus den Überlegungen, daß nicht akzeptierte Hypothesen nur in einem Teilgraphen ausgehend von dem betrachteten Regelschema weiter spezialisiert werden können und daß erzeugte Views nur innerhalb dieses Teilgraphen sinnvoll sind. Das bedeutet, wir erzeugen die benötigten Views und arbeiten den Teilgraphen ab, um danach direkt die Views wieder aus unserer Datenbank zu löschen.

Durch dieses Vorgehen existieren nur die für die weitere Bearbeitung notwendigen Views in der Datenbank. In Abschnitt 4.5 werden wir eine obere Schranke an die Anzahl der Views herleiten.

4.5. Eine obere Schranke an die erzeugten Views

Bei der Verwendung von Views innerhalb eines Algorithmus stellt sich die Frage, wieviele Views im schlechtesten Fall (worst case) erzeugt werden können.

In diesem Abschnitt wollen wir eine obere Schranke an die Anzahl der erzeugten Views herleiten. Wir verwenden dazu die in Abschnitt 3.3.3 definierten Bezeichnungen. Sei im folgenden r die Anzahl der Regelschemata, p die Anzahl der Prädikate und k die maximale Anzahl der Literale in einem Regelschema.

Betrachten wir zunächst die Views, die durch die Prädikatenbildung erzeugt werden. Mit Ausnahme der Prädikate, die durch die Abbildung vom Typs 1 erzeugt werden, wird je Relation grundsätzlich mindestens ein View angelegt. Vernachlässigen wir alle Prädikate des Abbildungstyps 1, so ist die Anzahl der Views bei der Prädikatenbildung kleiner oder gleich der Anzahl der generierten Prädikate p:

$$V_{\mathrm{pgen}} \quad \leq \quad p$$

Wenden wir uns nun den Views zu, die während des Lernens erzeugt werden. Damit ein View generiert werden kann, müssen zwei Bedingungen gelten:

1. Eine Hypothese wird nicht akzeptiert, und
2. diese Hypothese kann durch ein spezielleres Regelschema weiter spezialisiert werden.

Im schlechtesten Fall gehen wir davon aus, daß alle Hypothesen zu einem Regelschema nicht akzeptiert werden, also p^k Hypothesen. Da pro Hypothese genau zwei Views erzeugt werden, erhalten wir pro Regelschema $2 \cdot p^k$ Views. Allerdings werden diese Views nur dann erzeugt, wenn es speziellere Regelschemata gibt, die eine Spezialisierung der Regel generieren können. In unserer Regelschemata-Hierarchie tritt dieser Fall bei allen inneren Knoten auf. Wenn r die Anzahl der Regelschemata ist, dann soll n die Anzahl der innerern Knoten und l die Anzahl der Blätter sein, so daß $r = n + l$ gilt. Für die Anzahl der während des Lernens erzeugten Views gilt:

$$V_{\text{multi}} \leq 2 \cdot n \cdot p^k$$

Diese Schranke gilt insbesondere bei der Breitensuche durch die Hierarchie der Regelschemata. Durch den Einsatz der Tiefensuche und dem Löschen von nicht mehr benötigten Views (wie im vorherigen Abschnitt beschrieben), können wir eine bessere obere Schranke bestimmen. Dazu betrachten wir einen Pfad, also eine Folge von Knoten von einer Wurzel zu einem Blatt gemäß $\geq_\Theta$, in unserer Regelschemata-Hierarchie. Sei t_{max} der maximale (längste) Pfad, der in unserer Hierarchie vorkommt. Dann erhalten wir die obere Schranke:

$$V_{\text{multi}} \leq 2 \cdot (t_{max} - 1) \cdot p^k,$$

wobei $t_{max} - 1$ genau die Anzahl der inneren Knoten auf dem Pfad festlegt. Insbesondere gilt: $t_{max} - 1 \leq n$

Eine obere Schranke für die in RDT/DM erzeugten Views ist somit:

$$\begin{aligned} V_{\text{total}} &= V_{\text{pgen}} + V_{\text{multi}} \\ &\leq p + 2 \cdot (t_{max} - 1) \cdot p^k \end{aligned} \tag{4.1}$$

Treten in den Regelschemata zusätzlich Konstanten auf, verändert sich diese Formel vergleichbar mit der Formel (3.1) zu:

$$V_{\text{total}} \leq p + 2 \cdot (t_{max} - 1) \cdot (p \cdot i^c)^k \tag{4.2}$$

In der Realität wird die Anzahl der erzeugten Views erheblich unter dieser oberen Schranke liegen. Die Ausnutzung der Kompatibilitätbedingungen (Stelligkeit, Datentyp und Schlüssel) führt bereits zu einer erheblichen Reduktion der möglichen Hypothesen. Durch das Beschneidungskriterium werden zusätzlich Hypothesen abgelehnt, die dann alle spezielleren Hypothesen ausschließen. Zuletzt sollen Regeln gelernt werden, so daß das Akzeptanzkriterium mindestens einige Hypothesen akzeptieren sollte, durch die wiederum speziellere Hypothesen nicht mehr betrachtet werden.

4.6. Suche nach „interessanten“ Regeln

Die Suche nach Regeln in sehr großen und hochdimensionierten Datenbanken ergibt zum größten Teil eine Menge von Regeln, die dem Benutzer bereits bekannt sind, da diese dem dominierenden Modell innerhalb der Datenbank entsprechen. Dieses dominierende Modell unterdrückt die interessanten lokalen Muster in der Datenbank. Gerade zu diesen Mustern sind die Regeln für den Benutzer interessant, da sie Ausnahmen beschreiben.

In [Morik 2002] wird ein Verfahren zum Lernen solcher interessanten lokalen Muster beschrieben. Der grundlegende Ansatz ist dabei, zunächst die allgemeinen Regeln zu lernen und dann in einem zweiten Schritt, in dem „Rest“, das lokale Muster zu betrachten.

Der im zweiten Schritt betrachtete „Rest“ besteht aus den folgenden drei Beispielmengen (weitere sind natürlich möglich), die gegeben sind durch:

1. alle Ausnahmen zu den akzeptierten Regeln,
2. Beispiele, die von keiner der gefundenen Regeln abgedeckt werden,
3. negative Beispiele, die die Akzeptanz einer Regel verhindern.

Die Ergebnisse in [Morik 2002] sind mit dem Lernverfahren RDT/DM gelernt worden. Dazu ist der Algorithmus um die Sammlung der für den zweiten Lauf relevanten Daten erweitert worden. Mit Hilfe dieser Daten ist dann das Lernverfahren jeweils auf einer eingeschränkten Menge von Beispielen erneut gestartet worden. Die Ergebnisse zu den Lernläufen werden in Abschnitt 6.2 aufgeführt und diskutiert.

Als Nebeneffekt bestimmt RDT/DM im Anschluß an einen Lernlauf die Abdeckung der positiven und negativen Beispiele durch die gefundenen Regeln.

5. RDT/DM – Implementierung

Bei der Reimplementierung des Regellerners RDT/DB waren neben den konzeptionellen Erweiterungen weitere Aspekte von Interesse. Zunächst einmal stand der Wunsch nach einer stand-alone-Version im Raum, die außerhalb der Werkbank MOBAL läuft und dadurch eine leichtere Anbindung bzw. Einbindung in andere Software möglich macht. Weiterhin ist die in RDT/DB benutzte Anbindung der Datenbank ausschließlich für ORACLE konzipiert und somit nicht ohne erheblichen Aufwand auf eine andere Datenbank übertragbar. Eine Anbindung beliebiger Datenbanken ist generell wünschenswert.

Zur Umsetzung dieser Aufgaben standen die beiden folgenden Möglichkeiten zur Auswahl:

1. Reimplementierung in Prolog unter Verwendung des Originalquellcodes

2. Neuimplementierung in einer anderen Programmiersprache

Wir haben uns für die zweite Möglichkeit entschieden, die Gründe dafür legen wir im folgenden dar.
Die Implementierung in Prolog hat den Vorteil, daß ein Großteil des Prolog-Quellcodes weiterverwendbar ist und der Hauptaufwand in den konzeptionellen Erweiterungen liegt. Allerdings ist RDT/DB in vielen Bereichen mit anderen Komponenten der Werkbank MOBAL verzahnt. Zusammen mit der sehr umfangreichen, über Jahre hinweg gewachsenen Struktur von MOBAL ist die Identifizierung der für den Betrieb von RDT/DB relevanten Teile sehr aufwendig. MOBAL und dadurch auch RDT/DB laufen zur Zeit unter Quintus-Prolog [Quintus Prolog 2002], einem kommerziellen Prolog-System, wodurch der Einsatz auf verschiedenen Betriebssystemen mit verschiedenen Datenbanken nur stark eingeschränkt möglich ist. Die Portierung von RDT/DB und den notwendigen Teilen aus MOBAL auf ein anderes Prolog-System (z.B. SWI-Prolog [SWI Prolog 2002]) ist durch den Umfang des Quellcodes und der zum Teil verwendeten speziellen Befehle von Quintus-Prolog sehr komplex und aufwendig.

Bei einer Neuimplementierung des Algorithmus läßt sich die benutzte Programmiersprache anhand unserer Vorgaben und Wünsche auswählen und somit die Vorteile einer modernen Programmiersprache ausnutzen. Zusätzlich läßt sich die Implementierung auf die relevanten und wichtigen Teile des Algorithmus beschränken, wodurch ein „schlankes“ Programm erzielt wird.

Bleibt noch die Wahl der Programmiersprache zu klären. Die wichtigsten Anforderungen an diese Sprache sind:

- leichte Anbindung beliebiger Datenbanken
- umfangreiche Standardbibliotheken
- Betrieb auf unterschiedlichen Plattformen
- moderne (objektorientierte) Programmiersprache

Diese Anforderungen werden zum Beispiel durch die Programmiersprachen C++ und Java erfüllt. Insbesondere bietet Java mit seiner JDBC-Schnittstelle eine sehr flexible und datenbankunabhängige Anbindung verschiedener (relationaler) Datenbanken (vgl. Abschnitt 2.4), so daß wir diese Programmiersprache gewählt haben. Wenn wir zusätzlich die Anforderung nach der leichten Einbindung bzw. Anbindung des Verfahrens in eigene Anwendungen berücksichtigen, bietet Java zur Zeit die besseren Möglichkeiten.

In diesem Kapitel werden wir zunächst einen Überblick über die Implementierung geben, um dann in den folgenden Abschnitten bestimmte Konzepte besonders zu behandeln.

5.1. Übersicht

Das Zusammenspiel der einzelnen Teile des RDT/DM-Algorithmus werden wir anhand der Abbildung 5.1 erläutern.

In der Vorverarbeitung wird aus den durch den Benutzer angegebenen Regelschemata eine Hierarchie dieser Regelschemata unter Verwendung der Ordnung $\geq_{\mathcal{RS}}$ erzeugt. Desweiteren werden Prädikate generiert. Die Generierung erfolgt auf der durch den Benutzer eingeschränkten Menge von Relationen und verwendet die vorgegebenen Abbildungsvorschriften, die durch den Benutzer ggf. auf bestimmte Relationen oder Attribute eingeschränkt werden.

In das Regellernen fließen die Regelschematahierarchie, die generierten Prädikate und die durch den Benutzer vorgegebenen Bewertungskriterien sowie das Zielprädikat, wenn es durch den Benutzer vorgegeben wurde, ein. Die Hierachie der Regelschemata wird top-down per Tiefensuche durchlaufen, solange speziellere Regelschemata existieren und Hypothesen noch zu generell sind. Für jedes betrachtete Regelschema werden alle syntakisch und sematisch korrekten Hypothesen gebildet und einem Test unterzogen, der die zugehörigen Bewertungsmaße durch Zugriff auf die Datenbank bestimmt.

Nach dem Lernlauf wird die Abdeckung der positiven Beispiele und der negativen Beispiele durch die gefundenen Regeln bestimmt.

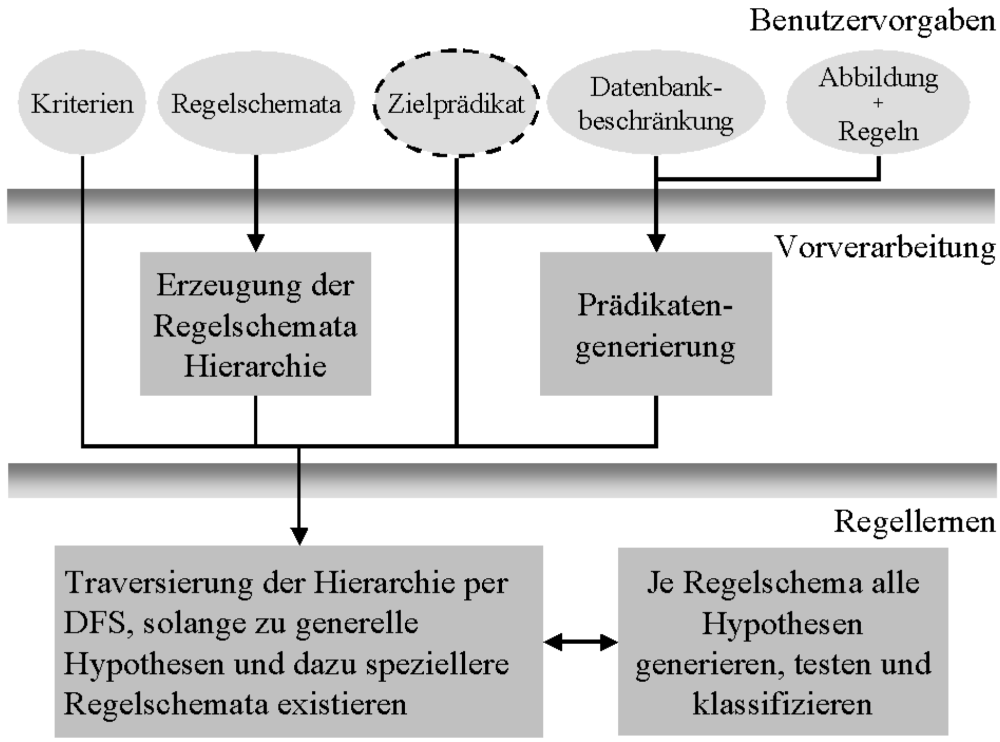


Abbildung 5.1.: Übersicht der Teilbereiche des RDT/DM-Algorithmus

5.2. Vorverarbeitung und Lernen

Die grundlegende Aufteilung des RDT/DM-Algorithmus erfolgt in die Bereiche Vorverarbeitung und Regellernen.

Die Vorverarbeitung enthält neben der Prädikatenbildung incl. der Erzeugung der Views auch die Generierung der Regelschemata-Hierarchie. Dadurch beschränkt sich die im Lernlauf notwendige Spezialisierung der Regelschemata gemäß $\geq_{\mathcal{RS}}$ auf eine einfache Traversierung dieser Hierarchie. Insbesondere kann die Hierarchie per Tiefensuche und Breitensuche durchlaufen werden.

Das Regellernen greift auf die durch die Vorverarbeitung erzeugte Hierarchie und die erzeugten Prädikate zu und führt die Hypothesengenerierung und den Hypothesentest durch.

5.3. Modularität und Erweiterbarkeit durch die Plug-In-Technologie

Eine sehr nützliche Funktionalität von Java besteht in dem Nachladen von Klassen zur Laufzeit. Verwendet wird dieses unter anderem bei der Datenbankanbindung durch JDBC. Unter Verwendung dieser Funktionalität haben wir in großen Teilen der RDT/DM-Klassenbibliothek Plug-In-Schnittstellen zum dynamischen Einbinden von eigenen Klassen definiert.

Wir greifen hier zwei der wichtigsten Schnittstellen heraus:

- Die Schnittstelle für beliebige Bewertungskriterien, die eine beliebige Anzahl von zusätzlichen Parametern unterstützt.
- Die Abbildungsschnittstelle, die für die Prädikatengenerierung eine Abbildung zwischen Relationen in der Datenbank und Prädikaten des Lernverfahren definiert. Diese Schnittstelle bietet eine beliebige Parameterisierung und eine konkrete Festlegung der Anwendbarkeit der Abbildungsvorschrift. Diese Anwendbarkeit bezieht sich auf die Wertebereiche der involvierten Spalten und auf die durch den Benutzer vorgegebenen Relationen sowie auf einzelne Attribute bestimmter Relationen der Datenbank, auf denen die Abbildung vollzogen werden soll.

Die gesamte Klassenbibliothek ist vollständig modular entworfen, d.h. die einzelnen Java-Pakete beruhen in der Regel auf einer definierten Schnittstelle und können unter Beibehaltung dieser jederzeit durch eine andere Implementierung ersetzt werden.

5.4. Benutzung von RDT/DM

Der Benutzer kann RDT/DM als direkt ausführbare Anwendung einsetzen oder unter Verwendung der Klassenbibliothek die gesamte oder einen Teil der Funktionalität in seine eigenen Programme integrieren. Diese Möglichkeiten werden in Abbildung 5.2 dargestellt.

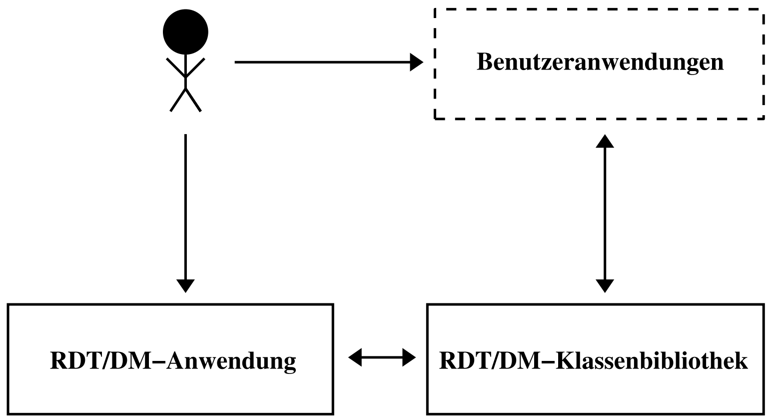


Abbildung 5.2.: Möglichkeiten der Benutzung von RDT/DM

Bei der direkten Ausführung werden die benötigten Einstellungen dem Programm durch zwei Konfigurationsdateien übergeben. Diese Dateien verwenden das XML-Format und sind durch den Benutzer einfach lesbar und mit Hilfe eines Standard-XML-Editors leicht zu bearbeiten. Die Ausgabe erfolgt in eine durch den Benutzer bestimmbare Datei. Die Menge der Details der Ausgaben ist stufenweise einstellbar.

Die Aufteilung in zwei Dateien folgt dabei der Idee, grundsätzliche Einstellungen für den Betrieb in eine Systemkonfigurationsdatei und versuchs- bzw. benutzerbezogene Einstellungen in eine Benutzerkonfigurationsdatei zu schreiben. In der Systemkonfigurationsdatei sind zum Beispiel die relevanten Einstellungen zu der Datenbank (ohne Benutzer und Passwort), der verwendete Datenbank-Treiber und die Adresse des Datenbankservers gespeichert. Zusätzlich werden dort Bewertungskriterien und Abbildungsvorschriften registriert (Plug-Ins), die dann in der Benutzerkonfigurationsdatei verwendbar sind. Die Benutzerkonfigurationsdatei beinhaltet die für den Lernlauf notwendigen Informationen. Dort werden zum Beispiel die relevanten Relationen der Datenbank und die konkreten Abbildungsvorschriften auf bestimmte Relationen oder Attribute festgelegt. Zusätzlich erfolgt die Bestimmung der verwendeten Regelschemata und Bewertungskriterien sowie der Versuchsmodalitäten und ggf. des Zielprädikates.

Nach der Installation und der Grundkonfiguration wird die Systemkonfigurationsdatei in der Regel nur noch bei der Registrierung von durch den Benutzer erstellten Bewertungskriterien und Abbildungsvorschriften verändert. Die Benutzerkonfigurationsdatei hingegen ist für jeden Lernlauf anzupassen.

Die Benutzung der Programmierschnittstelle (API) erlaubt den Zugriff auf die einzelnen Module der Klassenbibliothek und erlaubt dem Anwendungsentwickler die vollständige Kontrolle der einzelnen Teilschritte während der Vorverarbeitung und dem Regellernen. Der Austausch ganzer Module bzw. die Erweiterung oder Modifikation von Modulen ist bei der Benutzung der API ohne großen Aufwand möglich.

6. Versuche

In diesem Kapitel werden wir einige der durchgeführten Versuche beschreiben und diskutieren. Dazu führen wir zunächst unsere Versuchsumgebung ein, indem wir die verwendete Datenbank und Rechnerumgebung beschreiben. Aufbauend darauf werden wir die einzelnen Versuche diskutieren.

6.1. Versuchsumgebung

Die Versuchsumgebung besteht aus der von uns für unsere Versuche verwendeten Datenbasis sowie aus der verwendeten Hardwareausstattung unseres Versuchssystems.

6.1.1. Datenbasis für die Versuche

Für die Durchführung von Versuchen verwenden wir eine Datenbank, die wir aus einer Teilmenge der Internet Movie Database [IMDb 2002] und den freiverfügbaren Daten des MovieLens-Systems [MovieLens 2002] erstellt haben.

Internet Movie Database

Bei der Internet Movie Database handelt es sich eine Sammlung von Informationen zu Filmen sowie Videospielen. Unter den Filmen befinden sich Kino-, Video- und Fernsehfilme sowie Fernsehserien.
Die Datenbank enthält eine Reihe von Informationen zu den Filmen. Neben dem Titel (in verschieden Sprachen) und dem Produktionsjahr sind weitere Informationen wie mitspielende Schauspieler, Regisseure, Kameramänner, Produzent, Komponist, Maskenbildner, Produktionsfirma, Spezial-Effekte-Firma etc. enthalten.
Insbesondere sind auch Bewertungen zu den einzelnen Filmen vorhanden.

Der Zugriff auf die Internet Movie Database erfolgt im Regelfall über das WWW mit Hilfe eines Browsers. Auf der komfortablen Webseite ist die Suche nach verschiedenen Optionen möglich. Desweiteren wird die Internet Movie Database in regelmäßigen Abständen in Textdateien exportiert und auf einigen FTP-Servern veröffentlicht.

Movielens

Bei dem MovieLens-System werden dem Benutzer mit Hilfe von kollaborativen Filtern Filme vorgeschlagen. In der Datenbank werden Filme mit dem Titel, Produktionsjahr, Genre und einer Referenz zum zugehörigen Film in der IMDb gespeichert. Zusätzlich werden Benutzerdaten (Alter und Beschäftigung) und die Bewertung der Benutzer zu den Filmen gespeichert.

Eine Teilmenge dieser Datenbank ist in Form von Textdateien auf einem FTP-Server veröffentlicht worden. Es handelt sich dabei um Datensätze zu 900 Benutzern, die mindestens je 20 Filme bewertet haben.

Versuchsdatenbank

Da wir den direkten Zugriff auf die Datenbank benötigen, haben wir unsere Versuchsdatenbank aus den Textdateien zu den obigen Datenbanken aufgebaut. Die Generierung der Datenbank erfolgte durch ein Java-Programm, welches zunächst durch Michael Wurst entwickelt und von mir erweitert und verbessert wurde.

Diese Datenbank besteht zur Zeit aus 19 Tabellen und hat bis zu 743416 Tupel in den Tabellen.
Zusätzlich zu den aus den Textdateien erzeugten Tabellen haben wir je nach durchgeführten Versuchen die Datenbank um weitere Tabellen erweitert bzw. eine andere Form der Repräsentierung der Daten erzeugt.

Insbesondere haben wir aus den Daten der IMDb eine Rangfolge der Filme bestimmt. Dazu ist die Funktion des IMDb-Projektes

$$rank = \frac{v}{v+k}X + \frac{k}{v+k}C \tag{6.1}$$

verwendet worden. Dabei steht X für die durchschnittliche Bewertung des Films, v für Anzahl der Bewertungen des Films, k für die notwendige Anzahl von Bewertungen und X für die durchschnittliche Bewertung aller betrachteten Filme.

6.1.2. Hardwareausstattung

Alle Versuche sind auf einem PC (Athlon 800 MHz mit 512 MByte RAM) unter Linux 2.2.18 durchgeführt worden. Als Datenbank verwenden wir [PostgreSQL 2002] in der Version 7.1.3 sowie den JDBC-Treiber [jxDBCon-PostgreSQL 2002]. Die Ausführung des RDT/DM-Programms erfolgt mit der Java 2 Standard Edtion 1.3.1 für Linux. Zur Bestimmung der Laufzeiten benutzen wir die Standard-API-Methoden von Java.

6.2. Versuchsbeschreibung

In unserem Versuch schränken wir die Versuchsdatenbank in mehrern Bereichen ein. Zunächst betrachten wir in diesem Versuch ausschließlich Kinofilme. Alle anderen Filmtypen wie Videoproduktionen, Fernsehfilme und -serien sowie Computerspiele werden in diesem Versuch nicht miteinbezogen. Aus dieser Menge von Kinofilmen bestimmen wir mit Hilfe der Funktion (6.1) die einhundert besten und die einhundert schlechtesten Filme aus der Datenbank. Dabei schränken wir die Kinofilme durch die Anzahl der abgegebenen Bewertungen ein. Die besten einhundert Filme müssen mindestens 400 ($k = 400$) und die schlechtesten einhundert Filme müssen mindestens 50 ($k = 50$) Bewertungen erhalten haben. Die beiden Gruppen zu je 100 Filmen bezeichnen wir als gute bzw. als schlechte Filme. Für unsere Versuchsdurchführung verwenden wir genau diese zweihundert Kinofilme. Zu diesen Filmen nutzen wir die Informationen über das Produktionsland des Filmes und erzeugen daraus eine Einteilung in Kontinente, in denen die Filme produziert wurden. Die Einteilung erfolgt in die Gruppen Amerika, Europa und Asien, weitere Kontinente tauchen nicht auf. Zusätzlich verwenden wir das Genre des Films, den Regisseur und zehn Schauspieler aus dem Film. Die Schauspieler sind anhand der Position aus der Credit-Liste ausgewählt worden, dabei berücksichtigten wir die Positionen 1-10. Wir erweitern die Datenbank um eine Tabelle, die für die mitwirkende Person (Schauspieler oder Regisseur) angibt, ob sie in mindestens zwei guten bzw. mindestens zwei schlechten Film mitgespielt hat. Die Lernaufgabe besteht in der Bestimmung von Regeln für die Einteilung in gute und schlechte Filme.

Im Vergleich zu der in Abschnitt 6.1.1 beschriebenen Datenbank haben wir zur Unterstützung des Lernverfahrens und zur leichteren Lesbarkeit der Ergebnisse eine andere Repräsentierung der Datenbank gewählt. Abbildung 6.1 stellt die für diesen Versuch benutzte Datenbank dar.

6.2.1. Abbildungen der Datenbank

Die Abbildung der Datenbanktabellen auf logische Prädikate erfolgt durch die Anwendung der Abbildungsvorschriften aus Abschnitt 4.1.1. Die folgende Tabelle gibt die Zuordnung der Relationen zu den Abbildungsvorschriften an.

Tabelle	Spalte	Erw. Abbildungstyp
ranking	top	Typ 3
country	alle	Typ 3
continent	alle	Typ 3
genre	alle	Typ 3
director	alle	Typ 1
actor	alle	Typ 1
tbperson	alle	Typ 3

Wie in Abschnitt 4.1.1 beschrieben, erfolgt bei RDT/DM durch die erweiterten Abbildungen zunächst die Erzeugung von Views und danach die Bildung von Prädikaten. Die folgende Tabelle listet die erzeugten Views und die zugehörigen Prädikate auf. Zur einfachen Lesbarkeit der Tabelle haben wir die vollständigen Prädikatnamen durch bedeutungsgleiche, kürzere Bezeichnungen ersetzt.

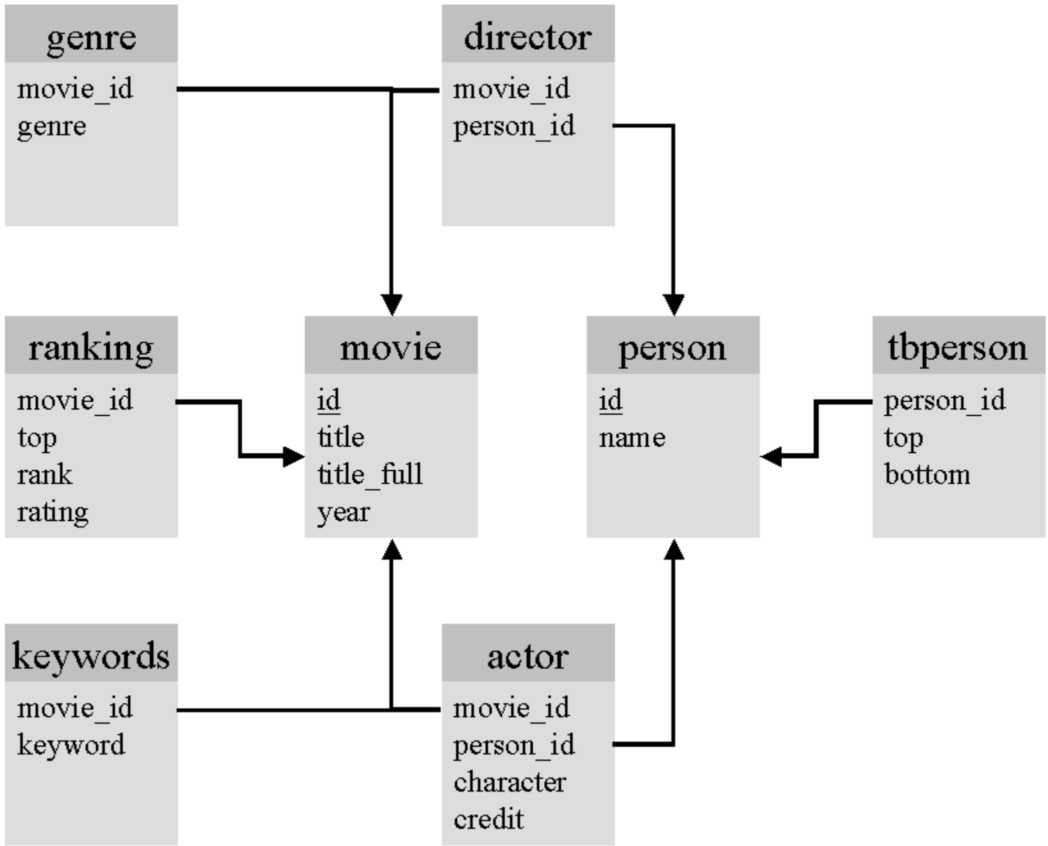


Abbildung 6.1.: Die logischen Verbindungen in der Datenbank der Versuchsdurchführung

Tabelle 6.1.: Abbildung der Tabellen auf Prädikate für die Versuchsdurchführung

Tabelle	View	Prädikat
ranking	ranking_top	top(movie_id)
country	country_country_1	country_West_Germany(movie_id)
	country_country_2	country_Japan(movie_id)
	country_country_3	country_UK(movie_id)
	country_country_4	country_France(movie_id)
	country_country_5	country_Spain(movie_id)
	country_country_6	country_Canada(movie_id)
	country_country_7	country_China(movie_id)
	country_country_8	country_Norway(movie_id)
	country_country_9	country_Sweden(movie_id)
	country_country_10	country_Italy(movie_id)
	country_country_11	country_Denmark(movie_id)

	country_country_12 country_country_13 country_country_14 country_country_15	country_Germany(movie_id) country_USA(movie_id) country_South_Korea(movie_id) country_Argentina(movie_id)
continent	continent_continent_1 continent_continent_2 continent_continent_3	continent_Europe(movie_id) continent_Asia(movie_id) continent_America(movie_id)
genre	genre_genre_1 genre_genre_2 genre_genre_3 genre_genre_4 genre_genre_5 genre_genre_6 genre_genre_7 genre_genre_8 genre_genre_9 genre_genre_10 genre_genre_11 genre_genre_12 genre_genre_13 genre_genre_14 genre_genre_15 genre_genre_16 genre_genre_17 genre_genre_18 genre_genre_19	genre_Fantasy(movie_id) genre_Action(movie_id) genre_Mystery(movie_id) genre_Sci-Fi(movie_id) genre_Western(movie_id) genre_Family(movie_id) genre_Thriller(movie_id) genre_Crime(movie_id) genre_Romance(movie_id) genre_Short(movie_id) genre_Adventure(movie_id) genre_Musical(movie_id) genre_Horror(movie_id) genre_Film-Noir(movie_id) genre_Comedy(movie_id) genre_Drama(movie_id) genre_War(movie_id) genre_unknown(movie_id) genre_Animation(movie_id)
director		director(movie_id,person_id)
actor		actor(movie_id,person_id)
tbperson	tbperson_top_1 tbperson_top_0 tbperson_bottom_1 tbperson_bottom_0	topperson(person_id) notopperson(person_id) botperson(person_id) nobotperson(person_id)

Wir erhalten somit für den Lernlauf $p = 44$ Prädikate.

6.2.2. Regelschemata

Für den Lernlauf haben wir die folgenden Regelschemata dem Verfahren zur Verfügung gestellt:

1. $ma1(Q, P1, P2) : Q(X) \leftarrow P1(X), P2(X).$

2. $ma2(Q, R, P) : Q(X) \leftarrow R(X, Y), P(Y).$

3. $ma3(Q, P1, R1, R2) : Q(X) \leftarrow P1(X), R1(X, Y), R2(Y).$

Die Größe des Hypothesenraumes ist somit unter Anwendung der Formel (3.1) nach oben durch $r \cdot p^k = 3 \cdot 44^3 = 255552$ beschränkt.

6.2.3. Akzeptanz- und Beschneidungskriterium

Als Akzeptanzkriterium verwenden wir

$$\frac{|pos(h)|}{|concl(h)|} - \frac{|neg(h)|}{|concl(h)|} \geq 0.3 \tag{6.2}$$

Durch dieses Kriterium wird das Verhältnis der Differenz der positiven und der negativen abgedeckten Beispiele der untersuchten Hypothese zu allen positiven Beispielen dargestellt. Wir erhalten somit bezogen auf alle positiven Beispiele (sind hier zufälligerweise genau 100) den Prozentsatz der mehr abgedeckten positiven Beispiele. Es müssen durch die betrachtete Hypothese mindestens 30% mehr positive als negative Beispiele abgedeckt sein.

Durch das Beschneidungskriterium wollen wir eine minimale Anzahl von abgedeckten positiven Beispielen garantieren und wählen deshalb:

$$\frac{|pos(h)|}{|concl(h)|} \leq 0.1 \tag{6.3}$$

Dadurch müssen mindestens 10% aller positiven Beispiele durch die untersuchte Hypothese abgedeckt werden.

6.2.4. Ergebnisse

Als Ergebnis dieser Suche erhalten wir die Regeln:

1. $top(X) \leftarrow country\_USA(X), genre\_Drama(X).$
2. $top(X) \leftarrow continent\_America(X), genre\_Drama(X).$
3. $top(X) \leftarrow director(X, Y), topPerson(Y).$
4. $top(X) \leftarrow actor(X, Y), topPerson(Y).$
5. $top(X) \leftarrow director(X, Y), noBotPerson(Y).$
6. $top(X) \leftarrow actor(X, Y), noBotPerson(Y).$

Diese Regeln decken 85% aller positiven und 14% aller negativen Beispiele innerhalb der Datenbank ab. Im Rahmen der Versuchsdurchführung haben wir diesen Lernlauf mit und ohne Ausnutzung der Fremdschlüsselrelationen durchgeführt. Werden diese logischen Verbindungen bei der Hypothesengenerierung berücksichtigt, reduziert sich die Anzahl der zu testenden Hypothesen um 18% von 771 auf 629.

6.2.5. Suche nach „interessanten“ Regeln

Wie wir in Abschitt 4.6 beschrieben haben, sind die obigen Regeln die zu erwartenden Regeln. Bei einer oberflächlichen Betrachtung der Datenbank ist die starke Ausrichtung auf amerikanische Dramen direkt offensichtlich. Interessanter sind hingegen die Regeln, die nicht offensichtlich aus den Daten hervorgehen, d.h. die sich bei der Betrachtung der nicht akzeptierten positiven Beispiele ergeben oder die sich aus den noch abgedeckten negativen Beispielen lernen lassen. In dem folgenden Abschnitt werden wir zunächst die nicht abgedeckten positiven Beispiele und darauffolgend in Abschnitt 6.2.5 die durch die obigen Regeln zusätzlich abgedeckten negativen Beispiele untersuchen.

Für die durchgeführten Lernläufe haben wir den Hypothesenraum im Vergleich zum ersten Lernlauf vergrößert, um RDT/DM zusätzliche Möglichkeiten zum Lernen von Regeln zu bieten. Diese Vergrößerung erfolgt einerseits durch die Erweiterung bzw. Veränderung der Regelschemata und andererseits durch Hinzunahme von zusätzlichen Prädikaten.

Die Menge der Regelschemata erweitern wir um ein längeres Schema mit drei einstelligen Prädikaten ($ma4(Q, P1, P2, P3)$) und verändern das Schema $ma2(Q, R, P)$ so, daß hier Konstanten gelernt werden müssen. Es ergeben sich somit die neuen Regelschemata:

1. $ma1(Q, P1, P2) : Q(X) \leftarrow P1(X), P2(X).$
2. $ma2c(Q, R, P; C) : Q(X) \leftarrow R(X, C), P(C).$
3. $ma3(Q, P1, R1, R2) : Q(X) \leftarrow P1(X), R1(X, Y), R2(Y).$
4. $ma4(Q, P1, P2, P3) : Q(X) \leftarrow P1(X), P2(X), P3(X).$

Die möglichen Belegungen von C ergeben sich aus der Anzahl der Schauspieler (2379) und der Regisseure (165) aus der Datenbank. In den Regelschemata sind somit $c = 1$ Konstanten enthalten, die maximale Anzahl der Belegung ist $i = 2379 + 165 = 2544$.

Für die 29 Filme fügen wir eine zusätzliche Tabelle *keywords* ein, die verschiedene Schlüsselwörter zu diesen Filmen beinhaltet. Mit Hilfe der erweiterten Abbildung vom Typ 3 werden aus dieser Tabelle 330 zusätzliche Prädikate generiert. Wir verwenden somit für die Suche insgesamt $p = 374$ Prädikate.

Eine obere Schranke an die Größe des Hypothesenraumes für diese Lernaufgabe ist folglich mit Hilfe der Formel (3.1) gegeben durch:

$$r \cdot (p \cdot i^c)^k = 4 \cdot \left(374 \cdot 2544^1\right)^3 = 4 \cdot 951456^3 < 3.5 \cdot 10^{18}$$

Für die weiteren Lernläufe passen wir das Akzeptanzkriterium so an, daß wir die Differenz der positiven und negativen Beispiele mit der Summe dieser in Verhältnis setzen:

$$\frac{|pos(h)| - |neg(h)|}{|pos(h)| + |neg(h)|} \geq 0.6 \tag{6.4}$$

Durch diese Form des Akzeptanzkriteriums sind wir nicht mehr mit der Gesamtheit aller positiven oder negativen Beispielen verbunden. Die Bewertung der Hypothesen basiert aussschließlich auf den von dieser Hypothese abgedeckten positiven und negativen Beispielen. Der gewählte Schwellwert von 0.6 garantiert, daß entweder nur positive Beispiele existieren oder daß eine erheblich größere Anzahl positiver als negativer Beispiele durch die Hypothese abgedeckt werden.

Lernlauf mit den nicht abgedeckten positiven Beispielen

In der Tabelle 6.2 sind die Filme aufgelistet, die durch die obigen Regeln nicht abgedeckt sind, obwohl sie zu den guten Filmen gehören.

movie_id	title
31118	Boot, Das
80533	Festen
87878	Fucking Åmål
132908	Ladri di biciclette
140438	Lola rennt
141194	Lord of the Rings: The Fellowship of the Ring, The
144157	Léon
159076	Monty Python and the Holy Grail
172408	Nuovo cinema Paradiso
219479	Shrek
221975	Sjunde inseglet, Det
231168	Straight Story, The
260797	Vita è bella, La
263038	Wallace & Gromit: The Wrong Trousersy
269387	Wo hu cang long

Tabelle 6.2.: Die nicht abgedeckten guten Filme

Auf dieser kleinen Menge von Beispielen führen wir einen weiteren Lernlauf durch, dabei verwenden wir den vergrößerten Hypothesenraum wie oben beschrieben.

Das Verfahren lernt die folgenden Regeln:

1. $top(X) \leftarrow country\_Italy(X), genre\_Drama(X).$
2. $top(X) \leftarrow continent\_Europe(X), keyword\_family(X).$
3. $top(X) \leftarrow continent\_Europe(X), keyword\_independent - film(X).$
4. $top(X) \leftarrow continent\_Europe(X), keyword\_bicycle(X).$
5. $top(X) \leftarrow genre\_Drame(X), keyword\_love(X).$
6. $top(X) \leftarrow genre\_Drame(X), keyword\_bicycle(X).$

7. $top(X) \leftarrow continent\_Europe(X), genre\_Romance(X), genre\_Drame(X).$

Mit diesen Regeln decken wir von den obigen 15 Filmen 11 ab. Durch das Beschneidungskriterium (6.3) mit dem Schwellwert 0.2 wird garantiert, daß die gefundenen Regeln jeweils mindestens zwei der Filme abdecken. Die auch weiterhin nicht abgedeckten Filme sind:

movie_id	title
31118	Boot, Das
141194	Lord of the Rings: The Fellowship of the Ring, The
219479	Shrek
221975	Sjunde inseglet, Det

Diese Filme sind anhand der Informationen in der Datenbank nicht in einer sinnvollen Weise durch Regeln beschreibbar. Möglicherweise könnten diese Filme bei einer Suche auf einer erweiterten Datenbank (erweitert um signifikante Informationen zu den Filmen) ebenfalls abgedeckt werden.

Untersuchen wir die Laufzeiten und die Anzahl der betrachteten Hypothesen während der Suche und vergleichen dabei die Suche mit und ohne Ausnutzung der Fremdschlüsselrelationen, so erhalten wir:

Fremdschlüssel	Laufzeit der Suche	Betrachtete Hypothesen
nein	28 : 08	22220
ja	5 : 22	11118

In diesem direkten Vergleich zwischen einem Lernlauf mit und ohne Ausnutzung der Fremdschlüsselrelationen auf einem relativ großen Hypothesenraum sind die Vorteile der Variante mit Fremdschlüsselrelationen offensichtlich. Sowohl die Laufzeit der Suche verringert sich um 82% als auch die Anzahl der zu testenden Hypothesen wird um ca. 50% reduziert.

Lernlauf mit den abgedeckten negativen Beispielen

In der Tabelle 6.3 sind diejenigen Filme aufgelistet, die durch die obigen Regeln abgedeckt sind, obwohl sie zu den schlechten Filmen gehören.
Die Lernaufgabe hat sich hierbei geändert. Im Gegensatz zu den anderen Lernläufen versuchen wir hier, Regeln für schlechte Filme zu finden. Diese Regeln sollen uns eine Möglichkeit zur Unterscheidung zwischen guten und schlechten Filmen geben, die beide durch die Regeln des ersten Lernlaufes abgedeckt sind.

Bei der Durchführung der Lernläufe ergeben sich 21 Regeln, die eine Abdeckung von 11 der 14 Filme erreichen. Die gefundenen Regeln sind dabei sehr unterschiedlich und geben keine für den Benutzer einheitliche Vorgabe für schlechte Filme, die durch die Regeln des ersten Lauf mitabgedeckt sind. Da die Anzahl der Regeln

movie_id	title
20853	Barbaric Beast of Boggy Creek, Part II, The
30455	Bolero
50405	Cool as Ice
93002	Girl in Gold Boots
93904	Glitter
126601	Kazaam
134046	Laserblast
155225	Mighty Morphin Power Rangers: The Movie
157360	Mitchell
169071	Night Train to Mundo Fine
181702	Parts: The Clonus Horror
188962	Police Academy 5: Assignment: Miami Beach
188963	Police Academy 6: City Under Siege
217363	Shanghai Surprisey

Tabelle 6.3.: Die zusätzlich abgedeckten schlechten Filme

zusätzlich größer als die Anzahl der abgedeckten negativen Filme ist, macht es mehr Sinn, diese Filme direkt als Ausnahmen aufzulisten.

Bei einer Abschwächung des Beschneidungskriterium (6.3) auf einen Schwellwert von 0.1, d.h. es müssen mindestens zwei der 14 positiven Beispiele abgedeckt sein, liefert RDT/DM 59 Regeln. Insbesondere wird hierbei auch das Konstantenlernen durchgeführt und wir erhalten die Regeln:

1. $notop(X) \leftarrow actor(X, 532370), notopPerson(532370).$
2. $notop(X) \leftarrow actor(X, 174119), notopPerson(174119).$
3. $notop(X) \leftarrow actor(X, 308990), notopPerson(308990).$
4. $notop(X) \leftarrow actor(X, 532370), botPerson(532370).$
5. $notop(X) \leftarrow actor(X, 174119), botPerson(174119).$
6. $notop(X) \leftarrow actor(X, 308990), botPerson(308990).$

Dadurch haben wir drei Schauspieler bestimmen können, die in genau zwei schlechten Filmen mitgespielt haben. Bei der Auswertung dieser Personen ergibt sich, daß alle drei Schauspieler in den „Police Academy“-Filmen mitgespielt haben.

Ein weiterer Lernlauf ohne die zusätzlichen Prädikate aus der Tabelle *keywords* liefert eine überschaubare Anzahl von 8 Regeln, die neben den obigen Konstanten zusätzlich feststellt, daß Filme, die sowohl Musicals als auch Dramen sind, zu den schlechten Filmen gehören. Allerdings führen diese 8 Regeln nur zu einer Abdeckung von 42% (6 von 14).

Die folgende Tabelle listet für die verschiedenen Beschneidungskriterien abhängig von der Nutzung der erweiterten Prädikatenmenge und abhängig von der Ausnutzung der Fremdschlüsselrelationen die jeweiligen Laufzeiten der Suche sowie die Anzahl der betrachteten Hypothesen auf:

Pruning	Keywords	Fremdschlüssel	Laufzeit	Betr. Hypothesen
≤ 0.2	ja	nein	38 : 37	27652
≤ 0.2	ja	ja	16 : 53	20214
≤ 0.1	ja	nein	1 : 13 : 42	53488
≤ 0.1	ja	ja	25 : 06	39258
≤ 0.1	nein	nein	13 : 18	13548
≤ 0.1	nein	ja	11 : 04	12278

In diesen Versuchsläufen ist erneut der starke Einfluß der Fremdschlüsselrelationen ersichtlich, der zu einer starken Reduzierung der Laufzeit führt.

6.2.6. Dynamische Viewgenerierung

In diesem Abschnitt untersuchen wird die Auswirkungen der dynamischen Erzeugung von Views auf die Geschwindigkeit der Suche innerhalb des Hypothesenraumes. Dazu haben wir eine Reihe von Versuchen sowohl mit als aus auch ohne diese Viewgenerierung durchlaufen lassen. Für die durchgeführten Versuche verwenden wir die in Abbildung 6.1 dargestellte Datenbank. Die folgende Tabelle zeigt die Ergebnisse zu einer Auswahl der von uns durchgeführten Versuche auf. In der Tabelle ist die obere Schranke an die Größe des Hypothesenraumes des speziellen Versuchs sowie die Laufzeit mit und ohne Viewgenerierung aufgeführt.

$\lvert\mathcal{L}_H\rvert \leq$	Viewerzeugung	Laufzeit
4205	nein	00 : 00 : 08
4205	ja	00 : 03 : 20
511104	nein	00 : 00 : 23
511104	ja	00 : 05 : 52
$3.5 \cdot 10^{18}$	nein	00 : 11 : 04
$3.5 \cdot 10^{18}$	ja	00 : 27 : 18
$3.5 \cdot 10^{18}$	nein	00 : 13 : 18
$3.5 \cdot 10^{18}$	ja	00 : 42 : 30

Aus diesen Versuchen wird ersichtlich, daß sich durch die Verwendung von Views die Laufzeiten der Lernläufe gravierend verlängert haben. Eine erste Untersuchung legt die Vermutung nahe, daß sich der Aufwand für die Erzeugung und die Zerstörung der Views nur bei einer großen Datenbank mit vielen Attributen und sehr vielen Tupeln lohnt. Zusätzlich besteht die Möglichkeit, daß die verwendete Datenbank die Laufzeit stark beeinflußt. Um diese Vermutungen abschließend zu klären, sind eine Reihe weiterer Versuche mit komplexeren Datenbanken und unterschiedlichen Datenbank-Systemen notwendig. Einen direkten Vorteil für die Laufzeit haben wir durch die dynamische Viewgenerierung mit der in diesen Versuchen benutzten Datenbank und dem PostgreSQL-Datenbank-System nicht erhalten.

7. Zusammenfassung und Ausblick

Zum Abschluß der Arbeit werden wir kurz die Ergebnisse der Arbeit zusammenfassen und danach im Ausblick einige Ideen für Erweiterungen des Algorithmus und zusätzliche Untersuchungen diskutieren.

7.1. Zusammenfassung

Ziel der Arbeit war es, aufbauend auf dem Algorithmus zu RDT/DB ein Lernverfahren zu entwickeln, das den folgenden Anforderungen genügt:

- Ausgliederung aus MOBAL, stand-alone-Version
- Verwendung des Lernverfahrens als direkte Anwendung sowie als Einbindung in eigene Programme
- Anbindung an beliebige Datenbanken
- Verstärkte Nutzung der Datenbank zur Optimierung des Lernverfahrens

Diese Zielsetzungen sind in dem Lernverfahren RDT/DM vollständig umgesetzt worden. Insbesondere erfolgte die Implementierung von RDT/DM in der objektorientierten Sprache Java im Gegensatz zu RDT/DB, welches als Bestandteil von MOBAL in Prolog implementiert ist. Ein wesentlicher Vorteil der Sprache Java besteht in der Nutzung der herstellerunabhängigen Datenbankschnittstelle JDBC, wodurch die Anbindung an beliebige Datenbanken gewährleistet wird. Die durchgeführte Implementierung ermöglicht unter Verwendung von Konfigurationsdateien die direkte Nutzung des Lernverfahrens, aber auch die Einbindung in eigene Anwendungen durch die Benutzung der Klassenbibliothek. Daß zur Ausgliederung von RDT/DB aus MOBAL eine komplette Neuimplementierung vorgenommen worden ist, hängt neben den oben aufgeführten Gründen auch mit der starken Verzahnung des Lernverfahrens RDT/DB mit anderen Komponenten aus MOBAL zusammen.

Die konzeptionellen Erweiterungen bestehen in der verstärkten Nutzung des Datenbank-Management-Systems. Hervorzuheben sind hierbei zwei wesentliche Punkte:

- die Verwendung von Views und

- die Verwendung von logischen Verbindungen zwischen Relationen

Zu beachten ist allerdings, daß im Unterschied zu RDT/DB auch schreibend auf die Datenbank zugegriffen wird.
Die logischen Verbindungen werden bei der Hypothesengenerierung zur semantischen Einschränkung der Menge der Hypothesen eingesetzt. Dazu definieren wir die Schlüsselkompatibilität bezogen auf eine Hypothese und integrieren diese als notwendige Bedingung in die Hypothesengenerierung.

Der Einsatz von Views erfolgt bei der Prädikatengenerierung und während der Traversierung der Regelschemata-Hierarchie. Für die Generierung der Prädikate erweitern wir die Abbildungsvorschriften von RDT/DB durch einen zweistufigen Ansatz. Im ersten Schritt werden Views für die speziellen Attribute und Tupel der Relation erzeugt. Danach erfolgt die Abbildung dieser Views auf Prädikate. Durch diese Konzeption verschieben wir die Komplexität in die Datenbank. D.h. beim Zugriff auf die zu einem Prädikat assoziierte Relation (View) ist die Einschränkung auf bestimmte Attribute und/oder Tupel unnötig, da diese durch das DBMS beim Zugriff auf den View automatisch erfolgt. Zusätzlich ist gerade dieser Zugriff (Viewdefinition auf Relation) innerhalb des Datenbank-Systems in der Regel performanter, als eine vergleichbare Anfrage von außen. Bei der Traversierung der Regelschemata-Hierarchie setzen wir die Views bei Hypothesen ein, die durch das Akzeptanzkriterium verworfen worden sind und die zusätzlich mit Hilfe von spezielleren Regelschemata weiter spezialisiert werden können. Dies führt zu einer schrittweisen Reduktion der betrachteten Tupel beim Hypothesentest. Zur effektiven Nutzung und zur Beschränkung der in der Datenbank erzeugten Views ändern wir die Suche in der Regelschemata-Hierarchie von Breitensuche auf Tiefensuche. Dadurch können wir Views für die Nutzung innerhalb eines Teilgraphen der Hierarchie erzeugen und diese danach wieder löschen, da sie für andere Bereiche der Hierarchie nicht benötigt werden. Für die Anzahl der erzeugten Views haben wir eine obere Schranke bestimmt.

Bei den von uns durchgeführten Versuchen hat sich die Ausnutzung der logischen Verbindungen zur semantischen Hypothesenraumeinschränkung als sehr sinnvoll erwiesen. Die Laufzeit des Algorithmus reduzierte sich bei allen durchgeführten Versuchen erheblich. Allerdings ist diese Steigerung der Performanz nur bei Datenbanken zu erreichen, die Fremdschlüssel unterstützen und bei denen bei der Modellierung diese verwendet werden.

Die Verwendung von Views hat die Vorteile, daß die Prädikatengenerierung flexibler gestaltet werden kann und die Komplexität bei der Erzeugung der SQL-Anfragen erheblich reduziert wird. Auf der anderen Seite ist das DBMS bei der Auswertung der Viewdefinitionen in der Datenbank gefordert. Sind diese internen Auswertungen in der Datenbank nicht optimiert und reduziert sich durch die Views nicht die Zugriffszeit auf die Daten, kann durch die Benutzung kein zeitlicher Vorteil für das Lernverfahren entnommen werden. Insbesondere ist dadurch der Verwaltungsaufwand für die Erzeugung und die Zerstörung der Views zu groß. Eine Untersuchung des Verhaltens des Lernverfahrens bei der Nutzung verschiedener Datenbanken ist sinnvoll. Desweiteren liegt die Vermutung nahe, daß durch die

Anwendung von RDT/DM auf eine komplexe Datenbank mit zahlreichen Relationen und vielen Attributen und Tupeln die Vorteile der dynamischen Viewgenerierung besser zum Tragen kommen. Auch hier ist eine weiterführende Untersuchung sinnvoll.

7.2. Ausblick

Diesen Abschnitt unterteilen wir in die vier Bereiche Datenbank-System, Akzeptanzkriterium, zweistufiges Lernen und Implementierung.

1. Neben den beschriebenen Aspekten des Datenbank-Systems sind weitere Nutzungen denkbar. Zunächst besteht natürlich die Möglichkeit, die Ergebnisse oder Teilergebnisse des Lernverfahrens zur weiteren Verarbeitung in die Datenbank zu schreiben. Auch die Auslagerung von Teilbereichen des Algorithmus in die Datenbank (stored procedures) könnte die Ausführungsgeschwindigkeit erhöhen. Zusätzlich könnte die Untersuchung von modernen Datenbanken (z.B. von objektorientierten oder objektrelationalen Datenbanken) für das Lernen relevante Informationen ersichtlich machen, so daß der Einsatz eines solchen Datenbank-System sinnvoll wird.

2. Eine sinnvolle Untersuchung stellen Akzeptanzkriterien dar, die auf statistischen Überlegungen und Theorien beruhen.

3. Wie wir bei der Suche nach „interessanten“ Regeln aufgezeigt haben, ist eine zweistufige Suche zur Bestimmung von lokalen Mustern in der Datenbank sinnvoll. Eine Automatisierung eines solchen Vorgehens stellt eine interessante Erweiterung des bestehenden Algorithmus dar.

4. Zuletzt sind auch bei der Implementierung des RDT/DM - Algorithmus einige Erweiterungen möglich. Der Entwurf und die Implementierung eines graphischen Benutzerinterface würde die Nutzung des Programms durch den Anwender erheblich vereinfachen. Zusätzlich könnte man dadurch den Benutzer auf Konfigurationsfehler aufmerksam machen und ihn grundsätzlich bei der Durchführung von Versuchen unterstützen. Auch bei den Abbildungsvorschriften zur Prädikatengenerierung bestehen Verbesserungsmöglichkeiten, z.B. könnte man durch die Definition von Abbildungen über mehrere Relationen bereits in der Vorverarbeitung relevante Attribute miteinander verbinden. Ebenso sind entsprechend der Diskretisierung von Werten weitere Vorverarbeitungen innerhalb der Abbildungen möglich.

Literaturverzeichnis

[Brockhausen 1994] Brockhausen, Peter (1994). *Entdeckung von funktionalen und unären Inklusionsabhängigkeiten in relationalen Datenbanken*. Diplomarbeit, Fachbereich Informatik, Universität Dortmund. in german.

[Brockhausen und Morik 1996] Brockhausen, Peter und K. Morik (1996). *Direct Access of an ILP Algorithm to a Database Management System*. In: Pfaringer, Bernhard und J. Fürnkranz, Hrsg.: *Data Mining with Inductive Logic Programming (ILP for KDD)*, MLnet Sponsored Familiarization Workshop, S. 95–110, Bari, Italy.

[Brockhausen und Morik 1998] Brockhausen, Peter und K. Morik (1998). *Wissensentdeckung in relationalen Datenbanken: Eine Herausforderung für das maschinelle Lernen*. In: Nakhaeizadeh, Gholamreza, Hrsg.: *Data Mining, theoretische Aspekte und Anwendungen*, Wirtschaftsinformatik, S. 193–211. Physica Verlag.

[Codd 1970] Codd, Edgar F. (1970). *A Relational Model of Data for Large Shared Data Banks*. Communications of the ACM, 13(6):377–387.

[Codd 1979] Codd, Edgar F. (1979). *Extending the Database Relational Model to Capture more Meaning*. ACM Transactions on Database Systems, 4(4):397–434.

[Codd 1990] Codd, Edgar F. (1990). *The relational model for database management version 2*. Addison-Wesley, Reading, Mass.

[Emde 1987] Emde, Werner (1987). *Non-Cumulative Learning in METAXA.3*. In: *IJCAI-87*, S. 208–210, Los Altos, CA. Morgan Kaufman. An extended version appeared as KIT-Report 56, Techn. Univ. Berlin.

[Fayyad et al. 1996] Fayyad, Usama M., G. Piatetsky-Shapiro und P. Smyth (1996). *From Data Mining to Knowledge Discovery: An Overview*. In: Fayyad, Usama M., G. Piatetsky-Shapiro, P. Smyth und R. Uthurusamy, Hrsg.: *Advances in Knowledge Discovery and Data Mining*, S. 1–36. AAAI Press/The MIT Press, Menlo Park, California.

[Görz 1995] Görz, Günther, Hrsg. (1995). *Einführung in die künstliche Intelligenz*. Addison-Wesley, 2 Aufl.

[HOWE 1983] HOWE, D. R. (1983). *Data Analysis for Data Base Design*. Edward Arnold.

[IMDb 2002] IMDB (2002). *Internet Movie Database*. URL `http://www.imdb.org`.

[JDBC 2002] JDBC (2002). *Schnittstellen Spezifikation zwischen bel. Datenbanken und Java*. URL `http://java.sun.com/products/jdbc`.

[jxDBCon-PostgreSQL 2002] JXDBCON-POSTGRESQL (2002). *JDBC-Treiber für die PostgreSQL-Datenbank*. URL `http://www.postgresql.org/`.

[KIETZ und WROBEL 1992] KIETZ, J.-U. und S. WROBEL (1992). *Controlling the Complexity of Learning in Logic through Syntactic and Task–Oriented Models*. In: MUGGLETON, STEPHEN, Hrsg.: *Inductive Logic Programming.*, Nr. 38 in *The A.P.I.C. Series*, Kap. 16, S. 335–360. Academic Press, London [u.a.].

[KIETZ 1988] KIETZ, JÖRG-UWE (1988). *Incremental and Reversible Acquisition of Taxonomies*. In: BOOSE, JOHN M., Hrsg.: *Proceedings of EKAW–88*, Kap. 24, S. 1–11. GMD, Sankt Augustin. Also as KIT-Report 66, Technical University Berlin.

[KIETZ 1996] KIETZ, JÖRG UWE (1996). *Induktive Analyse relationaler Daten*. Doktorarbeit, Technische Universität Berlin, Berlin.

[KLINGSPOR 1991] KLINGSPOR, VOLKER (1991). *Abstraktion von Inferenzstrukturen in MOBAL*. Diplomarbeit, Univ. Bonn.

[LINDNER 1994] LINDNER, GUIDO (1994). *Anwendung des Lernverfahrens RDT auf eine relationale Datenbank*. Diplomarbeit, Universität Dortmund, Fachbereich Informatik, Lehrstuhl VIII. Eine Zusammenfassung ist auch als Forschungsbericht Nr. 12 des Lehrstuhls Informatik VIII, Universität Dortmund, erschienen.

[MITCHELL 1982] MITCHELL, TOM M. (1982). *Generalization as Search*. Artificial Intelligence, 18(2):203–226.

[MORIK 1999] MORIK, KATHARINA (1999). *Maschinelles Lernen*. Skript zur gleichnamigen Vorlesung, 3.Auflage.

[MORIK 2002] MORIK, KATHARINA (2002). *Detecting Interesting Instances*.

[MORIK und BROCKHAUSEN 1997] MORIK, KATHARINA und P. BROCKHAUSEN (1997). *A Multistrategy Approach to Relational Knowledge Discovery in Databases*. Machine Learning Journal, 27(3):287–312.

[MORIK et al. 2000] MORIK, KATHARINA, S. WROBEL und T. JOACHIMS (2000). *Maschinelles Lernen und Data Mining*. In: GÖRZ, GÜNTHER, C.-R. ROLLINGER und J. SCHNEEBERGER, Hrsg.: *Handbuch der Künstlichen Intelligenz*. Oldenbourg, München, 3 Aufl.

[Morik et al. 1993] Morik, Katharina, S. Wrobel, J.-U. Kietz und W. Emde (1993). *Knowledge Acquisition and Machine Learning - Theory, Methods, and Applications*. Knowledge Based Systems. Academic Press, Harcourt Brace & Company, Publishers London, SanDiego, New York, Boston, Sydney, Tokyo.

[MovieLens 2002] MovieLens (2002). *MovieLens-System*. URL `http://www.movielens.org`.

[Muggleton und De Raedt 1994] Muggleton, S. und L. De Raedt (1994). *Inductive Logic Programming: Theory and Methods*. Journal of Logic Programming, 19/20:629–679.

[PostgreSQL 2002] PostgreSQL (2002). *Datenbank v. 7.1.3*. URL `http://www.postgresql.org/`.

[Quintus Prolog 2002] Quintus Prolog (2002). *Kommerzielle Prolog Version des Swedish Institut of Computer Science*. URL `http://www.sics.se/quintus`.

[Robinson 1965] Robinson, J. (1965). *A Machine-oriented Logic Based on the Resolution Principle*. Journal of the ACM, 12(1):23–41.

[Saake und Sattler 2000] Saake, Gunter und K.-U. Sattler (2000). *Datenbanken & Java – JDBC, SQLJ und ODMG*. IX-Edition. dpunkt, Heidelberg, 1 Aufl.

[Sauer 1998] Sauer, Hermann (1998). *Relationale Datenbanken: Theorie und Praxis – mit einem Beitrag zu SQL-3*. Addision Wesley Longman, Bonn; Reading, Massachusetts u.a., 4 Aufl.

[SWI Prolog 2002] SWI Prolog (2002). *Freie Prolog Version des Dept. of Social Science Informatics der Universität Amsterdam*. URL `http://www.swi.psy.uva.nl/projects/SWI-Prolog`.

[Wrobel 1997] Wrobel, Stefan (1997). *An Algorithm for Multi–relational Discovery of Subgroups*. In: Komorowski, J. und J. Zytkow, Hrsg.: *Principles of Data Minig and Knowledge Discovery: First European Symposium (PKDD 97)*, S. 78–87, Berlin, New York. Springer.

[Wrobel 1998] Wrobel, Stefan (1998). *Data Mining und Wissensentdeckung in Datenbanken*. Künstliche Intelligenz, 12(1):6–10.