TECHNISCHE UNIVERSITÄT BRAUNSCHWEIG
INFORMATIK-SKRIPTEN
33

Grundlagen von Entity-Relationship-Modellen

Martin Gogolla

Braunschweig
Juli 1994

Inhaltsverzeichnis

1 Motivation

1.1 Datenbanksysteme

Ein Datenbanksystem (DBS) besteht wie in Abbildung 1 dargestellt aus einer Datenbank (DB), d. h. einem integrierten, einheitlich strukturierten Datenbestand und einem Datenbank-Managementsystem (DBMS), d. h. einem Software-System zur Verwaltung der Datenbank.

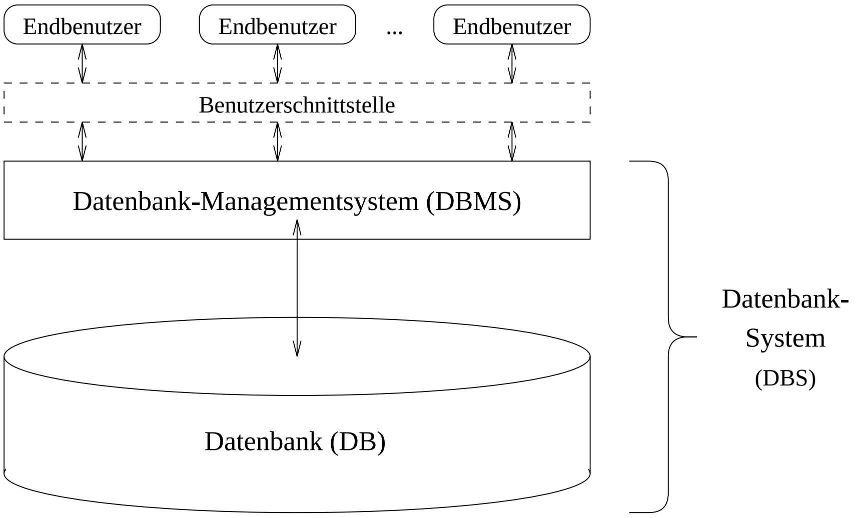


Abbildung 1: Struktur eines Datenbanksystems

Für Datenbanksysteme gibt es unterschiedliche Benutzerklassen. Zum einen Endbenutzer, die i. w. fertige Programmpakete verwenden, zum anderen Anwendungsprogrammierer, die eben die gerade genannten Programmpakete erstellen und schließlich Datenbank-Administratoren, die den Anwendungsprogrammierern ermöglichen mit der Datenbank zu arbeiten.

Ein Datenbanksystem stellt eine Reihe von Grundoperationen zur Verfügung. Zu diesen Operationen gehören insbesondere

- Adhoc-Anfragen,
- Zusammenstellungen, Berichte, Auswertungen und
- Änderungen, Ergänzungen, Löschungen.

Die Architektur von Datenbanksystemen wird von zwei wesentlichen Grundprinzipien beeinflußt: (1) Trennung von Daten und Metadaten und (2) Gewährleistung von Datenunabhängigkeit.

- 1. Prinzip: Trennung von Daten und Metadaten

  Die Daten charakterisieren den Datenbank-Zustand, der auch Datenbank-Ausprägung oder Datenbank-Inhalt genannt wird. Dagegen stellen die sogenanten Meta-Daten die Beschreibung des Datenbank-Schemas dar und geben so eine Datenbeschreibung der vorliegenden Daten. Unter einem Datenbank-Schema versteht man hierbei sowohl die Beschreibung der möglichen Datenbankzustände als auch der möglichen Übergänge.

  Die Metadaten charakterisieren somit

  - die Struktur der Datenbankzustände,
  - die Integritätsbedingungen, d. h. die Regeln für zulässige Datenbank-Zustände und Datenbank-Änderungen,
  - die möglichen Datenbank-Operationen, also die Basiszugriffe auf die Datenbank-Zustände und
  - die Autorisierungsregeln, d. h. die Regeln für erlaubte Datenbank-Zugriffe.

- 2. Prinzip: Gewährleistung von Datenunabhängigkeit

  Unter dem Prinzip der Datenunabhängigkeit versteht man die Stabilität der Benutzerschnittstelle gegenüber Änderungen. Man unterscheidet zwei Fälle, nämlich die physische Datenunabhängigkeit und die logische Datenunabhängigkeit.

  - Die physische Datenunabhängigkeit gewährleistet die Stabilität der Benutzerschnittstelle gegenüber Änderungen der internen Datenorganisation.
  - Die logische Datenunabhängigkeit gewährleistet die Stabilität der Benutzerschnittstelle gegenüber Änderungen der externen Anwendungen.

Aus diesen beiden Prinzipien leitet sich das im folgenden skizzierte Drei-Schichten-Modell aus Abbildung 2 zur Architektur eines Datenbanksystems ab.

- Die externen Sichten sind jeweils für spezifische Benutzergruppen eingerichtet (z. B. in einer betrieblichen Anwendung für Bertrieb, Buchhaltung, Personal oder Lager).
- Die Gesamt-Sicht ist der einheitlicher Bezugspunkt für die Übersetzung zwischen den genannten externen Sichten und der folgenden internen Sicht.
- Die interne Sicht beschreibt die Art und Struktur der Speicherung der Daten.

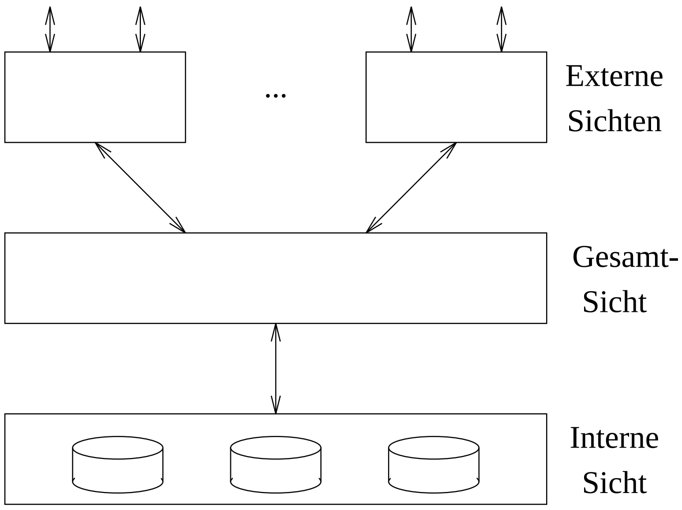


Abbildung 2: Drei-Schichten-Modell eines Datenbanksystems

Hieraus ergibt sich eine mehrstufige Datenbank-Definition. Zunächst werden die externen Schemata, dann das Gesamtschema und schließlich das internes Schema definiert. Die Übergänge zwischen den Schematas werden mit zugehörigen Transformationsregeln festgelegt.

Jedes Schema in dem oben beschriebenen Schichten-Modell wird nun in einem bestimmten Datenmodell beschrieben und daher wird nun der Schwerpunkt auf der Betrachtung des Begriffes Datenmodell liegen.

1.2 Datenmodelle

Unter einem Datenmodell wird ein System von Konzepten zur Strukturierung und Modellierung von Datenbeständen verstanden. Beispiele für Datenmodelle sind das hierarchische Datenmodell, das Netzwerk-Datenmodell oder das relationale Datenmodell.

Zu einem Datenmodell gehört eine Daten(bank)sprache, d. h. bestimmte Darstellungsmittel zur Beschreibung von Datenbeständen und Datenmanipulationen. Bei der Daten(bank)sprache unterscheidet man die Anteile zur Datendefinition (DDL $\approx$ Data Definition Language) und die Anteile zur Datenmanipulationen (DML $\approx$ Data Manipulation Language).

Daten(bank)schemata geben somit eine Beschreibung der invarianten Strukturen eines zeitveränderlichen Datenbestandes in einer Daten(bank)sprache wieder. Sie charakterisieren die möglichen Zustände sowie die möglichen Anfragen und Änderungen.

Nach dieser ersten Annäherung soll nun der Begriff Datenmodell genauer beleuchtet werden. Nach [Cod81] besitzt eine Datenmodell drei Komponenten: eine Strukturkomponente, eine Integritätsbeschreibungskomponente und eine Datenmanipulationskomponente.

Strukturkomponente: Der zu modellierende Weltausschnitt besteht in der Regel aus einer Ansammlung von Objekten und einer Vielzahl natürlicher Beziehungen zwischen ihnen. Demzufolge müssen geeignete Strukturen zur Modellierung der Objekte und ihrer Beziehungen angeboten werden.

Integritätsbeschreibungskomponente: Allein Strukturen reichen häufig aber nicht aus, um komplexe Beziehungen darzustellen. Deshalb werden so nicht darstellbare Beziehungen durch Integritätsbedingungen beschrieben.

Datenmanipulationskomponente: Ein zu modellierender Weltausschnitt ist nicht statisch sondern er ändert sich. Gewünscht wird das Arbeiten mit gespeicherten Daten und Änderungen am Datenbestand. Aus diesem Grund müssen Möglichkeiten zur Handhabung der Datenbankobjekte angeboten werden. Unter Handhabung der Datenbankobjekte sollen dabei sowohl Anfragen als auch Änderungsoperationen verstanden werden.

Diese drei Komponenten eines Datenmodells sollen nun am Beispiel des Relationenmodells erläutert werden.

Strukturkomponente: Die grundlegende Strukturen im Relationenmodell sind Relationen im mathematischen Sinne. Sie werden durch Relationenschemata beschrieben.

Beispiel: GENUG-Schema (Gesellschaft für Nahrungs- und Genußmittel)

- KUNDE(KName, KAdr, Kto)
- AUFTRAG(KName, Ware, Menge)
- LIEFERANT(LName, LAdr, Ware, Preis)

Integritätsbeschreibungskomponente: Integritätsbedingungen werden im Relationenmodell häufig innerhalb der Relationenschemata notiert.

- KUNDE($\underline{\text{KName}}$, KAdr, Kto)
- AUFTRAG($\underline{\text{KName}}$ → KUNDE.KName, $\underline{\text{Ware}}$, Menge)
- LIEFERANT($\underline{\text{LName}}$, LAdr, $\underline{\text{Ware}}$ → AUFTRAG.Ware, Preis)
  LName → LAdr

Im obigen Beispiel sind als Integritätsbedingungen Schlüsselattribute, Fremdschlüsselbeziehungen und funktionale Beziehungen angegeben. Die unterstrichenen Schlüsselattribute dienen zur eindeutigen Identifikation eines Tupels in einer Relation. Fremdschlüssel sind durch Pfeile in den Relationenschemata angedeutet. Beispielsweise bedeutet „Ware → AUFTRAG.Ware“ in „LIEFERANT“, daß jede in der Relation LIEFERANT vorkommende Ware auch in der Relation AUFTRAG erscheinen muß. Funktionale Beziehungen sind durch Pfeile verdeutlicht. Beispielsweise bedeutet „LName → LAdr“, daß in der Relation LIEFERANT der Lieferantenname eindeutig die Lieferantenadresse festlegt.

Schlüsselattribute können auch als Forderungen nach funktionalen Beziehungen aufgefaßt werden. D. h. statt der obigen Unterstreichungen könnte man auch explizit „KName → KAdr Kto“, „KName Ware → Menge“ und „LName Ware → Preis LAdr“ fordern.

Datenmanipulationskomponente: Datenmanipulationen sollen für das relationale Datenmodell an Hand der Sprache SQL aufgezeigt werden.

- Es sollen zunächst zwei einfache Anfragen in SQL formuliert werden.

  Welche Lieferanten liefern Milch oder Mehl?

  ```
  SELECT LName, LAdr
  FROM   LIEFERANT
  WHERE  Ware = 'Milch' OR Ware = 'Mehl'
  ```

  Welche Lieferanten aus Braunschweig liefern von Weiss bestellte Waren?

  ```
  SELECT LName, LIEFERANT.Ware
  FROM   LIEFERANT, AUFTRAG
  WHERE  LIEFERANT.LAdr LIKE '%Braunschweig%' AND
         LIEFERANT.Ware = AUFTRAG.Ware AND
         KName = 'Weiss'
  ```

- Eine weitere Möglichkeit in SQL besteht darin, Änderungsoperationen durchzuführen.

  Füge den Kunden TU BS ein.

  ```
  INSERT INTO KUNDE VALUES ('TU BS', '38106 Braunschweig', 0)
  ```

  Füge alle Lieferanten als Kunden ein.

  ```
  INSERT INTO KUNDE (SELECT LName, LAdr, 0
                     FROM   LIEFERANT)
  ```

  Lösche alle Aufträge von schlechten Kunden.

```
DELETE AUFTRAG
WHERE  KName in (SELECT KName FROM KUNDE WHERE Kto < 0)
```

Halbiere alle Auftragsmengen von Weiss.

```
UPDATE AUFTRAG
SET    Menge = Menge*0.5
WHERE  KName = 'Weiss'
```

Man sieht daß Anfragen, d. h. Konstrukte der Art `SELECT FROM WHERE`, auch in Änderungsoperationen verwendet können. Damit kann man die Modifikation nicht nur einzelner Tupel durchführen, sondern sogar die Modifikation von Tupel-Mengen.

Nun sollen die Funktionen eines Datenbank-Managementsystem nach [Cod90] in Beziehung gestellt werden zur obigen Erklärung des Begriffs Datenmodell.

Integration: Alle Daten sollen einheitlich verwaltet werden. Dies wird durch die Einrichtung der Relationenschemata erreicht.

Operationen: Es sind explizit Operationen zur Datenspeicherung, zur Datensuche und zur Datenänderung vorgesehen.

Katalog: Durch die Angabe der Relationenschemata und der Attributnamen hat man Zugriff auf die Datenbeschreibung.

Benutzersichten: Hiermit kann ein eingeschränkter Zugang zu den Daten realisiert werden. Im Prinzip könnte man die Einrichtung von Sichten, in SQL mit `CREATE VIEW`, als Datenmanipulation auffassen.

Konsistenzüberwachung: Die Gewährleistung korrekter Datenbank-Inhalte und Datenbank-Änderungen wird durch die Integritätsbeschreibungskomponente explizit unterstützt.

Datenschutz: Um unautorisierte Zugriffe auszuschliessen, können Rechte an Relationenschemata und Attributen vergeben werden. Im Prinzip betrifft dies die Strukturkomponente in der außer den Relationennamen auch die Benutzer und deren Rechte mit vermerkt werden könnten.

Transaktionen: Dies betrifft die Zusammenfassung von Datenbank-Änderungen zu Einheiten, um die Konsistenz der Datenbank-Inhalte zu gewährleisten.

Synchronisation: Um konkurrierende Transaktionen mehrerer Benutzer zu synchronisieren gibt es spezielle Maßnahmen in Datenbank-Managementsystemen. Auch diese Funktion bezieht sich auf die Integritätsbeschreibungskomponente.

Datensicherung: Zur Wiederherstellung von Daten nach Systemfehlern werden Funktionen zur Datensicherung angeboten. Dies betrifft wieder die Forderung nach Aufrechthaltung der Datenbank-Integrität.

1.3 Semantische Datenmodelle

Bevor auf spezielle Eigenarten von semantischen Datenmodellen eingegangen wird bedarf es einer Begriffsklärung: Was versteht man unter einem semantischen Datenmodell? In [HM81] wird dieser Begriff wie folgt umschrieben:

> A semantic database model is designed to capture more meaning of an application environment than is possible with contemporary database models

Die Bezeichnung semantisches Datenmodell soll also andeuten, daß mehr Wissen über die Anwendung (mehr Bedeutung bzw. mehr Semantik) in einem Datenbank-Schema ausgedrückt werden kann als in einem konventionellen Datenmodell. Generell besteht in semantischen Datenmodellen der Wunsch, unterschiedliche Gegebenheiten in der Diskurswelt auch mit unterschiedlichen Konzepten des Datenmodells darzustellen.

Der Unterschied zwischen einem semantischen und einem konventionellen Datenmodell wird an Hand eines Beispiels aus [Hoh93] erläutert.

> Modelliert wird ein einfaches Stadt-Land-Fluß-Beispiel, in dem Städte (STADT, HAFENSTADT), Länder (LAND), verschiedene Arten von Gewässern (FLUSS, SEE, MEER, GEWAESSER) und führende Personen (PERSON) eines Landes modelliert werden.
>
> Zwischen diesen Objekt-Typen bestehen folgende Beziehungen: (1) Jede STADT liegt in genau einem LAND. (2) Eine HAFENSTADT liegt an einem GEWAESSER. (3) Ein FLUSS mündet in genau ein GEWAESSER und kann durch mehrere LAENDER fließen.
>
> Neben den üblichen Eigenschaften, wie den Namen (StName) einer STADT, der Einwohnerzahl (LEinw) eines Landes oder Adressen (Adr) einer PERSON, sollen insbesondere die Geometrien der geographischen Objekte in einem Koordinatensystem von Interesse sein: (1) Ein LAND wird durch eine Menge von Polygonzügen (Attribute Regionen mit Datentyp regions $\approx$ set(polygon)), (2) jede Stadt und jeder See durch einen Kreis (Attribute StGeo bzw. SGeo mit Datentyp circle), jedes Meer durch ein Polygon (MGeo mit Datentyp polygon) und jeder FLUSS durch einen nicht-geschlossenen und nicht-überlappenden Linienzug (Attribut FGeo mit Datentyp lines) dargestellt.

Die in einer separaten Datenschicht spezifizierten Datentypen werden hier als Wertebereiche der Attribute verwendet. Dabei soll address (Adresse einer PERSON) ein weiterer benutzer-definierter Datentyp sein.

Dieser Sachverhalt wird nun durch das folgende erweiterte Entity-Relationship-Diagram aus Abbildung 3 modelliert.

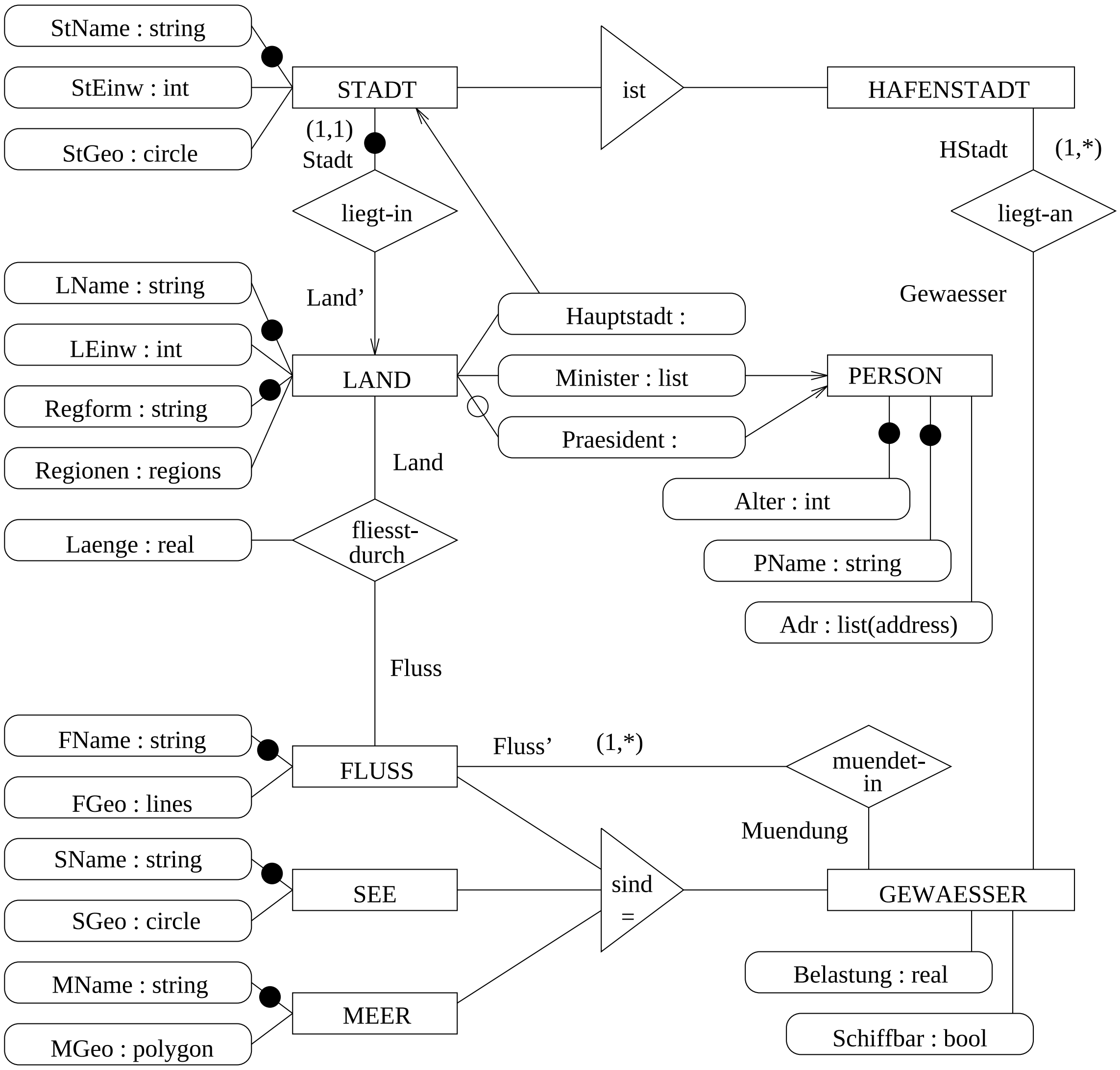


Abbildung 3: Stadt-Land-Fluß-Modellierung im Braunschweiger EER-Modell

Im Gegensatz dazu sieht die in Abbildung 4 dargestellte Modellierung dieses Sachverhalts als relationales Schema völlig anders aus.

Auffällig in der relationalen Modellierung ist die Überladung des Konstrukts Relation mit ganz unterschiedlichen Intentionen. Relationen werden verwendet um

```
R(STADT) ( Stadt$:STADTsurrogate, StName:string, StEinw:int,
           StGeo:circle )
R(LAND) ( Land$:LANDsurrogate, LName:string, LEinw:int,
          Regform:string, Regionen:regions,
          Hauptstadt:STADTsurrogate, Praesident:PERSONsurrogate )
R(FLUSS) ( Fluss$:FLUSSsurrogate, FName:string, FGeo:lines )
R(SEE) ( See$:SEEsurrogate, SName:string, SGeo:circle )
R(MEER) ( Meer$:MEERsurrogate, MName:string, MGeo:polygon )
R(PERSON) ( Person$:PERSONsurrogate, PName:string, Alter:int )
R(GEWAESSER) ( Gewaesser$:GEWAESSERsurrogate, Schiffbar:bool,
               Belastung:real, origin$:surrogate,
               Is-FLUSS:bool, Is-SEE:bool, Is-MEER:bool )
R(HAFENSTADT) ( Hafenstadt$:HAFENSTADTsurrogate,
                origin$:surrogate, Is-Stadt:bool )
R(muendet-in) ( Fluss'$:FLUSSsurrogate,
                Muendung$:GEWAESSERsurrogate )
R(fliesst-durch) ( Fluss$:FLUSSsurrogate, Land$:LANDsurrogate,
                   Laenge:real )
R(liegt-in) ( Stadt$:STADTsurrogate, Land$:LANDsurrogate )
R(liegt-an) ( HStadt$:HAFENSTADTsurrogate,
              Gewaesser$:GEWAESSERsurrogate )
R(Minister) ( Minister$:LANDsurrogate, Minister:PERSONsurrogate,
              Minister#:int )
R(Adr) ( Adr$:PERSONsurrogate, Adr:address, Adr#:int )
```

Abbildung 4: Stadt-Land-Fluß-Modellierung im relationalen Datenmodell

Objekttypen, allgemeinen Beziehungen, Ist-Beziehungen und mehr-wertige Attribute darzustellen. Beachtenswert ist weiter, daß die Zustandsräume für Entity-Relationship-Schema und das relationale Schema nicht isomorph sind: Das Entity-Relationship-Schema beinhaltet mehr modell-inhärente Integritätsbedingungen. Beispielsweise ist das (1,1)-Constraint von liegt-in im relationalen Schema nicht beachtet und für das objekt- und listen-wertige Attribut Minister von LAND nach PERSON ist es im relationalen Schema nicht unbedingt garantiert, daß die einzelnen Minister mit fortlaufenden Nummern versehen werden.

Aufgabe der konzeptionellen Modellierung einer Diskurswelt ist es, die für eine Datenbankanwendung relevanten Sachverhalte und Gesetzmäßigkeiten eines Anwendungsgebiets in einen formalen Rahmen zu bringen. Aus den obigen Beispiel wird bereits deutlich, daß für die konzeptionelle Datenmodellierung z. B. das relationale Datenmodell wenig geeignet ist, da der Abstand zwischen den Strukturen der Diskurswelt und den zur Modellierung verwendeten Relationen zu groß ist. Solche Überlegungen haben zur Entwicklung von semantischen Datenmodellen geführt. Einen guten Überblick zu diesem Thema bieten die Artikel [STW84, UD86, HK87, PM88, PT88].

Semantische Datenmodelle bieten bestimmte Modellierungskonzepte bzw. Standardabstraktionen [Lip88, Her94], die an Gegebenheiten der Diskurswelt orientiert sind: Klassifikation, Komponentenbeziehungen, IST-Beziehungen, allgemeine Beziehungen, explizite Integritätsbedingungen, Ableitungsregeln und Meta-Klassifikation.

Klassifikation: Unter Klassifikation versteht man die Zusammenfassung von Objekten der Diskurswelt zu Objekttypen mit dem Ziel Gemeinsamkeiten zu betonen und von Individualeigenschaften zu abstrahieren. Als Beispiel nennen wir den Entitytyp STADT aus dem obigen Stadt-Land-Fluß-Diagramm.

Komponentenbeziehungen: Bei der Zusammensetzung von Objekten der Diskurswelt aus anderen Objekten oder Werten spricht man von Komponentenbeziehungen. Es lassen sich zwei Spezialfälle unterscheiden: Aggregation und Assoziation.

- Unter Aggregation versteht man die Zusammensetzung eines Objekts aus anderen Objekten oder Werten verschiedenen Typs. Als Beispiel betrachte man den Entitytyp LAND mit seinen Eigenschaften LName, LEinw, Regform, Regionen, Hauptstadt, Minister und Praesident. Jedes Land stellt somit eine Aggregation über den zugrunde liegenden Entity- und Daten-Typen dar.
- Man spricht von Assoziation, wenn die Zusammensetzung eines Objekts aus anderen Objekten oder Werten gleichen Typs gemeint ist.

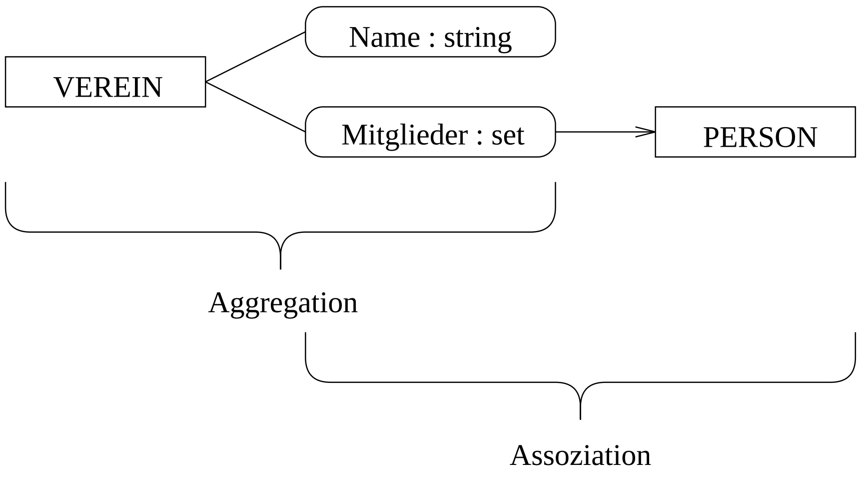


Abbildung 5: Beispiel für Aggregation und Assoziation

Im Beispiel in Abbildung 5 bildet der Entitytyp VEREIN zusammen mit den Eigenschaften Name und Mitglieder eine Aggregation. Die Eigenschaft Mitglieder ist eine Assoziation über dem Entitytyp PERSON.

Komponentenbeziehungen lassen sich nach folgenden Kriterien einteilen.

Exklusiv: Es gibt exklusive und nicht-exklusive Komponentenbeziehungen. Beispielsweise gehört ein Fahrgestell exklusiv zu genau einem Auto, während ein Modul nicht-exklusiv Bestandteil mehrerer Programme sein kann.

Existenzabhängig: Ein weiteres Kriterium zur Einteilung von Komponentenbeziehungen ist die Existenzabhängigkeit. Ein Fahrgestell ist in seiner Existenz abhängig von dem zugehörigen Auto während ein Motor existenzunabhängig vom Auto ist.

Identitätspropagierend: Komponentenbeziehungen können identitätspropagierend sein. Beispielsweise ergibt sich die Identität einer Gruppe aus der ihrer Mitglieder, d. h. die Identität wird vom Teil auf das Ganze weitergegeben. Auf der anderen Seite kann die Identität sich auch vom Ganzen auf das Teil fortpflanzen. Man denke an Dokumente in einer Bibliothek, in der sich die Identität eines Dokumentes aus der Bibliothek ergibt, da gleiche Dokumentnummern für unterschiedliche Dokumente in verschiedenen Bibliotheken denkbar sind. Letztendlich sind auch unabhängige Identitäten vom Teil und vom Ganzen möglich.

Eigenschaftspropagierend: Das letzte Kriterium betrifft die Propagation von Eigenschaften. Beispielsweise bestimmt das Alter des-Fahrgestells auch das Alter eines Autos, d. h. eine Eigenschaft vom Teil überträgt sich auf das Ganze. Auf der anderen Seite enthalten in einem Werkzeugschrank die Einschübe Werkzeug, d. h. eine Eigenschaft des Ganzen bestimmt auch Eigenschaften des Teils. Man spricht von selektiver Propagation von Eigenschaften wenn nicht alle Eigenschaften übertragen werden. Z. B. überträgt sich die Geschwindigkeit eines Autos auf die Geschwindigkeit seiner Teile, nicht aber sein Gewicht.

Aus dem obigen Spektrum an Möglichkeiten für Komponentenbeziehungen wird klar, daß es schwierig ist ein allgemeines Konzept für derartige Beziehungen zu finden. In Spezialfällen bleibt manchmal nur die Modellierung mit allgemeinen Beziehungen (s. u.).

Ist-Beziehungen: Bei Ist-Beziehungen geht es um die gleichzeitige Erfassung eines Objekts durch mehrere Konzepte. Solche Beziehungen führen in der Regel zu Objekttyphierarchien. Man unterscheidet zwei Spezialfälle: Generalisierungshierarchien und Rollenhierarchien.

- Bei Generalisierungshierarchien handelt es sich um Hierarchien mit statischer bottom-up Instanziierung. An der Spitze steht das allgemeinste Konzept, das in mehreren Schritten weiter verfeinert wird. Unterklassen erben in der Regel Eigenschaften der Oberklassen und können neue Eigenschaften hinzufügen. Oberklassen beschreiben damit gemeinsame Eigenschaften von Unterklassen. Die Instanziierung geschieht bottom-up. Eine Instanz einer Unterklasse ist gleichzeitig Instanz einer Oberklasse. Wenn die Generalisierungshierarchie ein Baum ist und die Extensionen an den Blättern disjunkt sind, dann sind auch die Extensionen aller Unterklassen disjunkt und hinsichtlich der Oberklassen überdeckend. Sie bilden dann eine Partition.

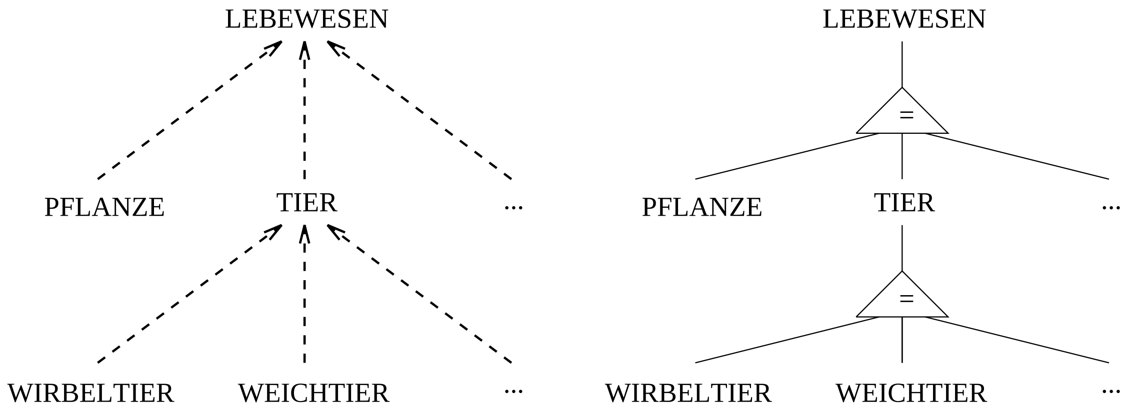


Abbildung 6: Beispiel für eine Generalisierungshierarchie

Im Beispiel in Abbildung 6 aus der Biologie ist rechts auch die Repräsentation im erweiterten Entity-Relationship-Modell angegeben. Ein weiteres Beispiel für eine Generalisierungshierarchie ist die Typkonstruktion „sind“ zwischen den Entitytypen FLUSS, SEE, MEER und GEWAESSER. Im erweiterten Entity-Relationship-Modell werden Generalisierungshierarchie mit =-Typkonstruktionen und jeweils einer Ausgangsklasse dargestellt.

- Bei Rollenhierarchien sind Hierarchien mit dynamischer top-down Instanziierung angesprochen. Klassen in einer Rollenhierarchie entspringen häufig verschiedenen Kontexten einer Diskurswelt. Unterklassen erben auch von Oberklassen. Aber die Instanziierung erfolgt top-down und vor allen Dingen dynamisch. Extensionen von Unterklassen sind damit nicht notwendigerweise disjunkt oder überdeckend.

  Wie bei Generalisierungshierarchien ist im Beispiel in der Abbildung 7 auch jeweils die Repräsentation im erweiterten Entity-Relationship-Modell angegeben. Generalisierungshierarchien entsprechen

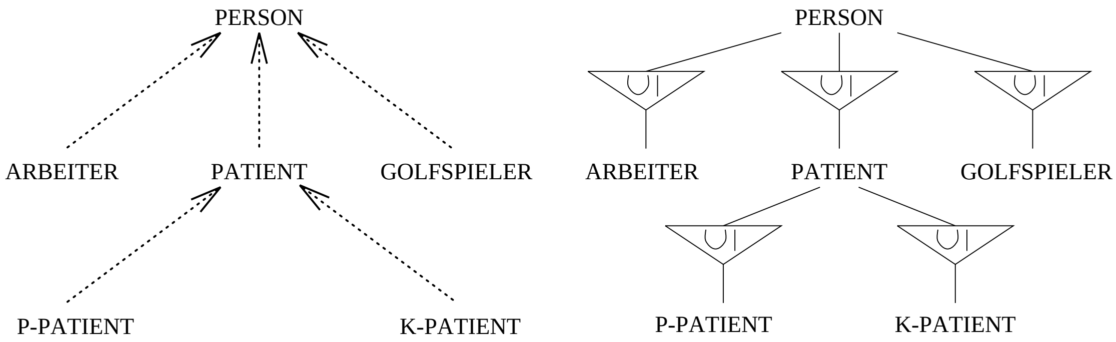


Abbildung 7: Beispiel für eine Rollenhierarchie

$\subseteq$-Typkonstruktionen mit jeweils einer Eingangsklasse. Ein anderes Beispiel für eine Rollenhierarchie ist die Typkonstruktion „ist“ zwischen den Entitytypen STADT und HAFENSTADT aus dem Stadt-Land-Fluß-Diagramm.

Das Konstrukt Typkonstruktion aus dem erweiterten Entity-Relationship-Modell ist bei der Modellierung von Generalisierungshierarchien und Rollenhierarchien also semantisch überladen.

Allgemeine Beziehungen: Der allgemeinste Ansatz zur Modellierung von Beziehungen zwischen Objekten beruht auf der Definition von mathematischen Relationen. Mit solchen Beziehungen läßt sich prinzipiell sehr viel modellieren. Man nimmt dann aber in Kauf, daß unterschiedliche Gegebenheiten der Diskurswelt gleich dargestellt werden. Als Beispiel sei die Beziehung fliesst-durch zwischen den Entitytypen FLUSS und LAND aus dem Stadt-Land-Fluß-Diagramm genannt.

Explizite Integritätsbedingungen: Komplexe Sachverhalte werden durch Integritätsbedingungen modelliert, insbesondere wenn sie sich über mehrere Schemabestandteile erstrecken. Sie werden in einer Form der Prädikatenlogik formuliert.

Als Beispiel wird gefordert, daß Hauptstädte in den Ländern liegen müssen, mit denen sie auch über die Beziehung liegt-in verbunden sind:

$\forall$ l:LAND liegt-in(Hauptstadt(l),l)

Statt dieser eher mathematischen Formulierung könnte man sich die Bedingung auch SQL-ähnlich, d. h. mit Schlüsselworten abgereichert, vorstellen.

```
FORALL ( LAND l ) liegt-in(Hauptstadt(l),l)
```

Ableitungsregeln: Ein weiteres Modellierungskonzept sind abgeleitete Schemakomponenten, d. h. abgeleitete Attribute, abgeleitete Beziehungstypen und abgeleitete Objekttypen. Dieses Konzept findet Anwendung wenn sich Information aus anderen Schemateilen gewinnen läßt. Abgeleitete Objekttypen entsprechen Sichten.

Als Beispiel im Stadt-Land-Fluß-Diagramm läßt sich als abgeleitetes Attribut die Gesamtflußlänge durch folgende Ableitungsregel definieren.

FLaenge(f) = SUM (-[Laenge(fd) | fd:fliesst-durch $\wedge$ fd.Fluss=f]-)

Die obige Regel ist im erweiterten Entity-Relationship-Kalkül formuliert. Man könnte sie aber auch mit einer etwas gesprächigeren Syntax in SQL-artiger Weise formulieren.

```
FLaenge(f) = SUM ( SELECT Laenge(fd)
                   FROM   fliesst-durch fd
                   WHERE  fd.Fluss=f       )
```

Meta-Klassifikation: Unter Meta-Klassifikation versteht man die Zusammenfassung von Objekttypen zu einer Klasse von Objekttypen. Dieses Konzept wird an Hand des Kartographie-Beispiels in Abbildung 8 erläutert. Das Attribut Bezeichung ist in diesem Beispiel ein abgeleitetes Schlüsselattribut.

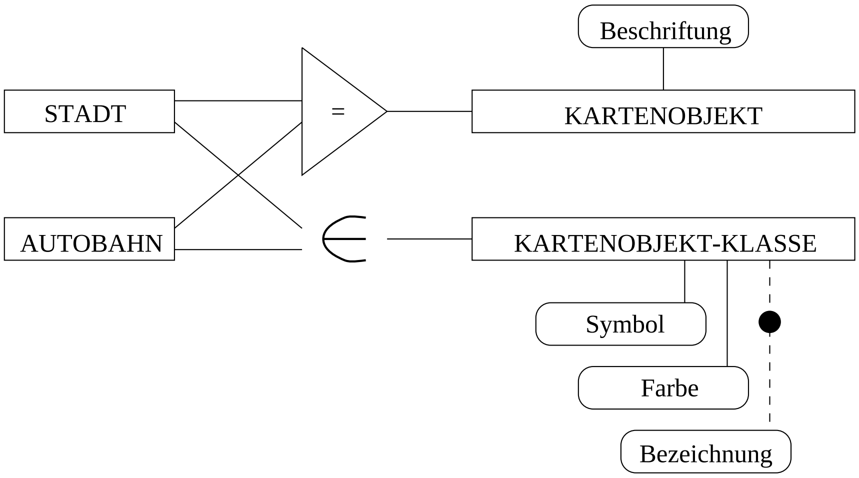


Abbildung 8: Meta-Klassifikation

Es wird nun zunächst eine Instanz zu dem Schema beschrieben. Es mögen genau die Städte st_1, st_2 und st_3 und die Autobahnen ab_1 und ab_2 im gegebenen Datenbank-Zustand existieren. Dann muß es die Kartenobjekte st_1, st_2, st_3, ab_1 und ab_2 geben. Weiter gibt es dann zwei Objekte kok_1 und kok_2 von Typ KARTENOBJEKT-KLASSE für die gilt: Bezeichnung(kok_1)=„STADT“ und

Bezeichnung(kok_2)=„AUTOBAHN“. Es läßt sich dann weiter z. B. festsetzen: Symbol(kok_1)=„Kreis“, Symbol(kok_2)=„Strich“, Farbe(kok_1)=„rot“ und Farbe(kok_2)=„schwarz“.

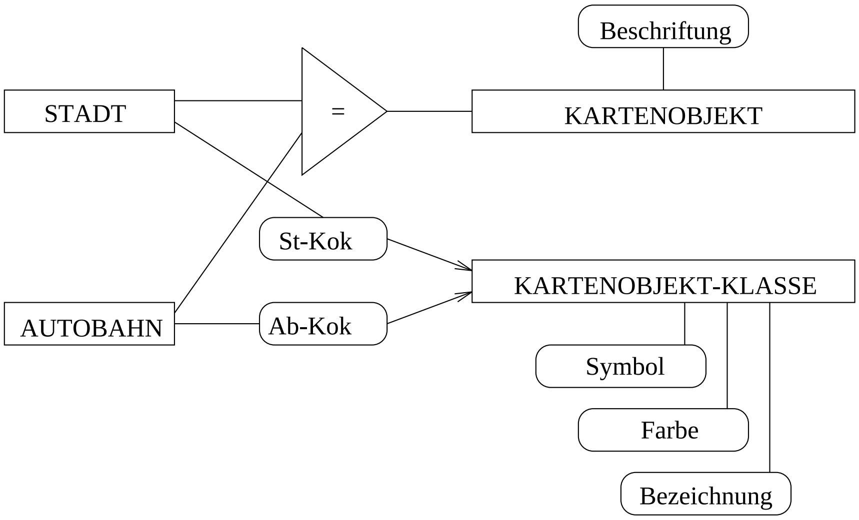


Abbildung 9: Meta-Klassifikation im Entity-Relationship-Modell

Dieser Sachverhalt läßt sich wie in Abbildung 9 dargestellt bedingt im erweiterten Entity-Relationship-Modell wiedergeben. Die Meta-Klassifikation wird durch zwei objekt-wertige Attribute St-Kok und Ab-Kok modelliert. Man benötigt aber noch explizite Integritätsbedingungen, die sicherstellen, daß die neu eingeführten Attribute auch die richtigen Werte annehmen.

∀ st:STADT Bezeichnung(St-Kok(st)) = „STADT“
∀ ab:AUTOBAHN Bezeichnung(Ab-Kok(ab)) = „AUTOBAHN“
∀ kok:KARTENOBJEKT-KLASSE ∃ st:STADT St-Kok(st) = kok
∨
∃ ab:AUTOBAHN Ab-Kok(ab) = kok

Als Konsequenz ergibt sich, daß Kartenobjekt-Klassen entweder die Bezeichnung „STADT“ oder „AUTOBAHN“ haben, denn das obige Constraint läßt sich quasi mit dem Attribut Bezeichnung multiplizieren.

∀ kok:KARTENOBJEKT-KLASSE Bezeichnung(kok) = „STADT“
∨
Bezeichnung(kok) = „AUTOBAHN“

Meta-Klassifikation ist ein Abstraktionskonzept, das nur für dynamische (und hier nicht weiter betrachtete) Schemata relevant ist.

2 Der Braunschweiger EER-Ansatz

In diesem Kapitel wird nun ein typisches semantisches Datenmodell vorgestellt, nämlich das in Braunschweig entwickelte erweiterte Entity-Relationship-Modell [HNSE87, HG88, GH91, EGH+92, Hoh93, Gog94].

2.1 Das EER-Modell

Um sowohl die Syntax als auch die Semantik von erweiterten Entity-Relationship-Schematas zu diskutieren, benötigt man eine Sprache mit der man über eben eine solche Syntax und Semantik sprechen kann. Als Sprache wird hier die Mengentheorie gewählt. Es sollen die folgenden axiomatischen Konventionen gelten.

2.1 Axiomatische Konventionen

|SET| ist Klasse der Menge, |FISET| Klasse der endlichen Mengen, |FUN| Klasse der totalen Funktionen und |REL| Klasse der Relationen. Es gilt $|\text{FISET}| \subseteq |\text{SET}|$ und $|\text{FUN}| \subseteq |\text{REL}| \subseteq |\text{SET}|$.

Es seien Mengen S, S_1, ..., $S_n \in |\text{SET}|$ gegeben. $\mathcal{F}(S)$ ist die Einschränkung der Potenzmenge $\mathcal{P}(S)$ von S auf endliche Mengen, S^* die Menge der endlichen Listen über S, S^+ die Menge der nicht-leeren endlichen Listen über S, und $S_1 \times ... \times S_n$ das kartesiche Produkt über $S_1,...,S_n$. Die Notation $\mathcal{B}(S)$ für Multimengen ist dabei analog zur Notation $\mathcal{F}(S)$ für Mengen zu sehen. Die Menge der endlichen Multimengen oder Bags über S ist $\mathcal{B}(S)$. Ein Bag ist eine endliche Menge S zusammen mit einer Zählfunktion occur : $S \rightarrow \mathcal{N}$, die für jedes Element die Anzahl der Duplikate des Elements angibt.

Endliche Mengen werden notiert als $\{c_1,...,c_n\}$, Listen als $<c_1,...,c_n>$, Elemente des kartesischen Produkts als $(c_1,...,c_n)$, und Multimengen als $\{\{c_1,...,c_n\}\}$. Für Mengen $\{c_1,...,c_n\}$ gilt $i \neq j$ impliziert $c_i \neq c_j$. Dies gilt nicht notwendigerweise für Multimengen: $\{\{c_1,...,c_n\}\}$ mit occur(c) = k impliziert es gibt k unterschiedliche Indizes $i_1,...,i_k \in 1..n$ mit $c_{i_j} = c$ für $j \in 1..k$.

2.2 Beispiel: Multimenge

Die Multimenge {{ A, B, A, C }} wird dargestellt als Menge { A, B, C } zusammen mit der Zählfunktion occur : $\{ A, B, C \} \rightarrow \mathcal{N}$ mit occur(A)=2, occur(B)=1 und occur(C)=1. Wenn man {{ A, B, A, C }} nach der obigen Notation als $\{\{ c_1, c_2, c_3, c_4, \}\}$ darstellt, gibt es zwei Indizes (1 und 3) mit $c_1 = c_3 = A$.

In erweiterten Entity-Relationship-Schemata tauchen sowohl Standard-Datentypen wie int oder string als auch Nicht-Standard-Datentypen wie point oder circle auf. Daher muß man sich zunächst darüber klar werden, wie der Begriff Datentyp zu interpretieren ist.

2.3 Definition: Datensignatur

Die Syntax einer Datensignatur DS ist gegeben durch

a. die Mengen DATA, OPNS, PRED $\in$ |FISET|,

b. eine Funktion source $\in$ |FUN| so daß source : OPNS $\rightarrow$ DATA*,

c. eine Funktion destination $\in$ |FUN| so daß destination : OPNS $\rightarrow$ DATA, und

d. eine Funktion arguments $\in$ |FUN| so daß arguments : PRED $\rightarrow$ DATA$^+$.

Wenn $\sigma \in$ OPNS, source(σ) = $<$d$_1$,...,d$_n$$>$, und destination($\sigma$) = d, wird dies dargestellt durch σ : d$_1$$\times$...$\times$d$_n$ $\rightarrow$ d. Wenn $\pi \in$ PRED mit arguments(π) = $<$d$_1$,...,d$_n$$>$, wird dies dargestellt durch π : d$_1$$\times$...$\times$d$_n$.

Die Semantik einer Datensignatur DS ist gegeben durch

a. eine Funktion μ[DATA] $\in$ |FUN| so daß μ[DATA] : DATA $\rightarrow$ |SET| und $\perp \in \mu$[DATA](d) für jedes d $\in$ DATA,

b. eine Funktion μ[OPNS] $\in$ |FUN| so daß μ[OPNS] : OPNS $\rightarrow$ |FUN| und σ : d$_1$$\times$...$\times$d$_n$ $\rightarrow$ d impliziert μ[OPNS](σ) : μ[DATA](d$_1$)$\times$...$\times$$\mu$[DATA](d$_n$) $\rightarrow$ μ[DATA](d) für jedes $\sigma \in$ OPNS, und

c. eine Funktion μ[PRED] $\in$ |FUN| so daß μ[PRED] : PRED $\rightarrow$ |REL| und π : d$_1$$\times$...$\times$d$_n$ impliziert μ[PRED](π) $\subseteq$ μ[DATA](d$_1$)$\times$...$\times$$\mu$[DATA](d$_n$) für jedes $\pi \in$ PRED.

Die Menge OPNS^{BIN} besteht aus allen binären Operationen $\sigma \in$ OPNS mit source(σ)=$<$d,d$>$ und destination(σ)=d für d $\in$ DATA. Die zugehörige Funktion μ[OPNS](σ) muß kommutativ und assoziativ sein.

2.4 Bemerkung: Vordefinierte Sorten, Operationen und Prädikate

Man geht davon aus aus, daß es eine Reihe von vordefinierten Datentypen, also Datensorten einschließlich der Operationen und Prädikate gibt.

DATA $\supseteq$ { int, real, string }

OPNS $\supseteq$ { +, −, $*$: int $\times$ int $\rightarrow$ int,
+, −, $*$: real $\times$ real $\rightarrow$ real,
/ : int $\times$ int $\rightarrow$ real,
/ : real $\times$ real $\rightarrow$ real,
$\uparrow$: int $\times$ int $\rightarrow$ int,
$\uparrow$: real $\times$ int $\rightarrow$ real,

$\text{sqrt} : \text{real} \to \text{real}$
$\text{concat} : \text{string} \times \text{string} \to \text{string},$
$\text{length} : \text{string} \to \text{int} \}$

$\text{PRED} \supseteq \{ =, \neq, <, \leq, \geq, > : \text{int} \times \text{int},$
$=, \neq, <, \leq, \geq, > : \text{real} \times \text{real},$
$=, \neq, <, \leq, \geq, > : \text{string} \times \text{string},$
$\text{substr} : \text{string} \times \text{string} \}$

Wie in Programmiersprachen setzt man also die gebräuchlichen Datentypen Int, Real und String als gegeben voraus. Als Semantik bzw. Interpretation nimmt man die „übliche" Semantik an: $\mu[\text{DATA}](\text{int}) = \mathcal{Z} \cup \{\perp\}$, $\mu[\text{DATA}](\text{real}) = \mathcal{R} \cup \{\perp\}$ und $\mu[\text{DATA}](\text{string}) = \text{A}^* \cup \{\perp\}$ für ein fest gegebenes Alphabet A. Die Bedeutung der Operationen und Prädikate sei ebenfalls wie „üblich" festgelegt.

Es wird also explizit zwischen dem Sortensymbol int $\in$ DATA und der Bedeutung des Symbols int $\mu[\text{DATA}](\text{int}) := \mathcal{Z} \cup \{\perp\}$ unterschieden. Ganz analog differenziert man auch beispielsweise zwischen den Funktionssymbolen abs, square : int $\to$ int $\in$ OPNS und zwischen der Bedeutung dieser Symbole. Z. B. ist die Bedeutung von square die folgende Funktion:

$$\begin{array}{llcl} \mu[\text{OPNS}](\text{square}) : & \mathcal{Z} \cup \{\perp\} & \to & \mathcal{Z} \cup \{\perp\} \\ & \text{i} & \mapsto & \texttt{if } \text{i} \in \mathcal{Z} \texttt{ then } \text{i}*\text{i} \texttt{ else } \perp \texttt{ fi} \end{array}$$

Später, im Abschnitt über den erweiterten Entity-Relationship-Kalkül, werden unterschiedliche, syntaktisch verschiedene Terme (wie z. B. abs(-4), square(2) $\in$ TERM) mit gleicher Bedeutung (μ[TERM](abs(-4))=μ[TERM](square(2))=4) auftauchen.

Der spezielle Wert $\perp$ wird als Ausnahmewert zur Fehlerbehandlung verwendet. Z. B. soll $\mu(1/0) = \perp$ gelten. Zur Vereinfachung der Schreibweise läßt man die syntaktische Kategorie bei der Semantikfunktion μ oft weg: Statt $\mu[\text{TERM}](1/0)$ schreibt man kurz $\mu(1/0)$.

2.5 Exkurs: Algebraische Spezifikation von Datentypen

Die Bedeutung der Datentypen Int, Real und String wurde oben explizit festgesetzt. Es gibt aber auch Ansätze, die die Semantik von Datentypen implizit über Axiome festlegen. Ein solcher Ansatz ist die algebraische Spezifikation von Datentypen. Bespielsweise liessen sich die ganzen Zahlen durch folgende Gleichungsspezifikation beschreiben.

```
DATA TYPE Int
SORTS int
OPNS  0 : -> int
      _+1, _-1 : int -> int
```

```
      _+_ : int int -> int
EQNS  (i+1)-1 = i
      (i-1)+1 = i
      i+0 = i
      i+(j-1) = (i+j)-1
      i+(j+1) = (i+j)+1
END
```

Für die obige Spezifikation sind mehrere, ganz unterschiedliche Interpretationen (oder Algebren, daher die Bezeichnung „Algebraische Spezifikation“) denkbar. Es werden beispielhaft zwei Algebren genannt.

1. μ(int) = $\mathcal{Z}$, μ(0) sei die Konstante 0, μ(+1) die Nachfolgerfunktion, μ(-1) die Vorgängerfunktion und μ(+) die Addition auf ganzen Zahlen.
2. μ(int) = $\{0,1\}^*$, μ(0)=„0110“, μ(+1) und μ(-1) seien die Identität und μ(x+y) = x. Auch diese Algebra erfüllt die obigen Axiome.

Es gibt im Gebiet der algebraischen Spezifikation jedoch Methoden, die sicherstellen, daß stets die erste der beiden obigen Interpretationen gewählt wird.

Im Kontext von Datentypen tauchen gewisse Standardkonstruktionen, wie z. B. die Konstruktion einer Liste list(sort) über einer gegebenen Datensorte sort, immer wieder auf. Deswegen macht es Sinn, diese Konstruktionen einmal zu definieren und sie dann immer wieder zu verwenden. Es werden Möglichkeiten zur Bildung von Mengen, Listen, Multimengen und Records vorgesehen.

2.6 Definition: Sortenausdrücke

Es sei eine Datensignatur DS und eine Menge S $\in$ |SET| mit DATA $\subseteq$ S mit einer (semantischen) Funktion μ[S] : S $\rightarrow$ |SET| gegeben, so daß μ[S](d) = μ[DATA](d) für d $\in$ DATA und $\perp \in \mu$[S](s) für jedes s $\in$ S. Die Syntax der Sortenausdrücke über S ist gegeben durch die Menge EXPR(S):

a. Wenn s $\in$ S, dann s $\in$ EXPR(S).
b. Wenn s $\in$ EXPR(S), dann set(s) $\in$ EXPR(S).
c. Wenn s $\in$ EXPR(S), dann list(s) $\in$ EXPR(S).
d. Wenn s $\in$ EXPR(S), dann bag(s) $\in$ EXPR(S).
e. Wenn $s_1,...,s_n$ $\in$ EXPR(S) (n>1), dann record($s_1,...,s_n$) $\in$ EXPR(S).

Die Semantik der Sortenausdrücke ist eine Funktion μ[EXPR(S)] : EXPR(S) $\rightarrow$ |SET| festgelegt durch die folgenden Regeln.

a. μ[EXPR(S)](s) := μ[S](s)

b. μ[EXPR(S)](set(s)) := $\mathcal{F}(\mu$[EXPR(S)](s)) $\cup \{\perp\}$

c. μ[EXPR(S)](list(s)) := (μ[EXPR(S)](s))* $\cup \{\perp\}$

d. μ[EXPR(S)](bag(s)) := $\mathcal{B}(\mu$[EXPR(S)](s)) $\cup \{\perp\}$

e. μ[EXPR(S)](record(s_1,...,s_n)) :=
(μ[EXPR(S)](s_1)$\times$...$\times\mu$[EXPR(S)](s_n)) $\cup \{\perp\}$

$\hat{\mu}$[EXPR(S)] ist eine Abkürzung für $\cup_{s \in EXPR(S)}$ μ[EXPR(S)](s), d. h., die Menge aller Instanzen von Sortenausdrücken über der Menge S. $\hat{\mu}$[EXPR(S)] stellt somit quasi den Wertevorrat oder das Wertuniversum dar, das durch die Sortenausdrücke festgelegt wird.

2.7 Beispiel: Sortenausdrücke

Beispiele für Sortenausdrücke sind record(real,real), list(record(real,real)) oder record(record(real,real),real). Der erste Sortenausdruck entpricht etwa der Datensorte point mit x- und y-Koordinate, der zweite der Sorte lines als Liste von Punkten und der dritte der Datensorte circle mit Kreismittelpunkt und Radius.

Allein Sortenausdrücke reichen aber zur Beschreibung von Nicht-Standard-Datentypen nicht aus. Man benötigt auch Operationen und Prädikate um Berechnungen durchführen zu können.

2.8 Definition: Durch Sortenausdrücke induzierte Operationen und Prädikate

Die Syntax der durch Sortenausdrücke induzierten Operationen und Prädikate ist gegeben durch die folgenden Mengen und Funktionen. Hierbei beziehen sich s,s_1,...,s_n auf beliebige Sortenausdrücke und σ auf ein Element aus OPNS^{BIN} mit σ : d$\times$d $\rightarrow$ d.

OPNS(S)/PRED(S) source+destination/arguments

$\text{CNT}_{set(s)}$:	set(s) $\rightarrow$ int
$\text{IND}_{set(s)}$:	set(s) $\rightarrow$ set(int)
$\text{APL}_{\sigma,set(d)}$:	set(d) $\rightarrow$ d
$\text{UNION}_{set(s),set(s)}$:	set(s) $\times$ set(s) $\rightarrow$ set(s)
$\text{CUT}_{set(s),set(s)}$:	set(s) $\times$ set(s) $\rightarrow$ set(s)
$\text{IN}_{set(s)}$:	set(s) $\times$ s
$\text{CNT}_{list(s)}$:	list(s) $\rightarrow$ int
$\text{IND}_{list(s)}$:	list(s) $\rightarrow$ set(int)

$\mathrm{LTB}_{list(s)}$:	list(s) $\rightarrow$ bag(s)
$\mathrm{SEL}_{list(s)}$:	list(s) $\times$ int $\rightarrow$ s
$\mathrm{POS}_{list(s)}$:	list(s) $\times$ s $\rightarrow$ set(int)
$\mathrm{APL}_{\sigma,list(d)}$:	list(d) $\rightarrow$ d
$\mathrm{IN}_{list(s)}$:	list(s) $\times$ s
$\mathrm{CNT}_{bag(s)}$:	bag(s) $\rightarrow$ int
$\mathrm{IND}_{bag(s)}$:	bag(s) $\rightarrow$ set(int)
$\mathrm{BTS}_{bag(s)}$:	bag(s) $\rightarrow$ set(s)
$\mathrm{OCC}_{bag(s)}$:	bag(s) $\times$ s $\rightarrow$ int
$\mathrm{APL}_{\sigma,bag(d)}$	bag(d) $\rightarrow$ d
$\mathrm{IN}_{bag(s)}$:	bag(s) $\times$ s
$\mathrm{PRJ}_{record(s_1,\ldots,s_n),i}$:	record(s_1,...,s_n) $\rightarrow$ s_i (i $\in$ 1..n)

Die Semantik der durch Sortenausdrücke induzierten Operationen und Prädikate wird folgendermaßen festgelegt.

- **Mengen-Operationen**

  $\mu(\mathrm{CNT}_{set(s)})$: $\{c_1,\ldots,c_n\} \mapsto n$

  $\mu(\mathrm{IND}_{set(s)})$: $\{c_1,\ldots,c_n\} \mapsto \{1,\ldots,n\}$

  $\mu(\mathrm{APL}_{\sigma,set(d)})$: $\{c_1,\ldots,c_n\} \mapsto$

  $$\begin{cases} \perp & \text{wenn } n = 0 \\ c_1 & \text{wenn } n = 1 \\ \mu(\sigma)(c_1, \mu(\mathrm{APL}_{\sigma,set(d)})(\{c_2,\ldots,c_n\})) & \text{wenn } n \geq 2 \end{cases}$$

  $\mu(\mathrm{UNION}_{set(s)})$: $(\{c_1,\ldots,c_n\},\{c'_1,\ldots,c'_m\}) \mapsto$

  $\{c_1,\ldots,c_n\} \cup \{c'_1,\ldots,c'_m\}$

  $\mu(\mathrm{CUT}_{set(s)})$: $(\{c_1,\ldots,c_n\},\{c'_1,\ldots,c'_m\}) \mapsto$

  $\{c_1,\ldots,c_n\} \cap \{c'_1,\ldots,c'_m\}$

  $\mu(\mathrm{IN}_{set(s)}) := \{ (\{c_1,\ldots,c_n\} , c) \mid c \in \{c_1,\ldots,c_n\} \}$

- **Listen-Operationen**

  $\mu(\mathrm{CNT}_{list(s)})$: $<c_1,\ldots,c_n> \mapsto n$

  $\mu(\mathrm{IND}_{list(s)})$: $<c_1,\ldots,c_n> \mapsto \{1,\ldots,n\}$

  $\mu(\mathrm{LTB}_{list(s)})$: $<c_1,\ldots,c_n> \mapsto \{\{c_1,\ldots,c_n\}\}$

  $\mu(\mathrm{SEL}_{list(s)})$: $(<c_1,\ldots,c_n> , i) \mapsto \begin{cases} c_i & \text{wenn } 1 \leq i \leq n \\ \perp & \text{sonst} \end{cases}$

  $\mu(\mathrm{POS}_{list(s)})$: $(<c_1,\ldots,c_n>,c) \mapsto$

  $$\begin{cases} \emptyset & \text{wenn } n = 0 \\ \mu(\mathrm{POS}_{list(s)})(< c_1,\ldots,c_{n-1} >, c) \cup \{n\} & \text{wenn } n \geq 1 \text{ und } c_n = c \\ \mu(\mathrm{POS}_{list(s)})(< c_1,\ldots,c_{n-1} >, c) & \text{wenn } n \geq 1 \text{ und } c_n \neq c \end{cases}$$

$\mu(\mathrm{APL}_{\sigma,list(d)})$: <c$_1$,...,c$_n$> $\mapsto$

$$\begin{cases} \perp & \text{wenn } n = 0 \\ c_1 & \text{wenn } n = 1 \\ \mu(\sigma)(c_1, \mu(\mathrm{APL}_{\sigma,list(d)})(< c_2, ..., c_n >)) & \text{wenn } n \geq 2 \end{cases}$$

$\mu(\mathrm{IN}_{list(s)})$:= { (<c$_1$,...,c$_n$> , c) | c $\in$ {c$_1$,...,c$_n$} }

- **Bag-Operationen**

  $\mu(\mathrm{CNT}_{bag(s)})$: {{c$_1$,...,c$_n$}} $\mapsto$ n

  $\mu(\mathrm{IND}_{bag(s)})$: {{c$_1$,...,c$_n$}} $\mapsto$ {1,...,n}

  $\mu(\mathrm{BTS}_{bag(s)})$: {{c$_1$,...,c$_n$}} $\mapsto$

  $$\begin{cases} \emptyset & \text{wenn } n = 0 \\ \{c_1\} \cup \mu(\mathrm{BTS}_{bag(s)})(\{\{c_2, ..., c_n\}\}) & \text{wenn } n \geq 1 \end{cases}$$

  $\mu(\mathrm{OCC}_{bag(s)})$: ({{c$_1$,...,c$_n$}} , c) $\mapsto$

  $$\begin{cases} 0 & \text{wenn } n = 0 \\ \mu(\mathrm{OCC}_{bag(s)})(\{\{c_2, ..., c_n\}\}, c) + 1 & \text{wenn } n \geq 1 \text{ und } c_1 = c \\ \mu(\mathrm{OCC}_{bag(s)})(\{\{c_2, ..., c_n\}\}, c) & \text{wenn } n \geq 1 \text{ und } c_1 \neq c \end{cases}$$

  $\mu(\mathrm{APL}_{\sigma,bag(d)})$: {{c$_1$,...,c$_n$}} $\mapsto$

  $$\begin{cases} \perp & \text{wenn } n = 0 \\ c_1 & \text{wenn } n = 1 \\ \mu(\sigma)(c_1, \mu(\mathrm{APL}_{\sigma,bag(d)})(\{c_2, ..., c_n\})) & \text{wenn } n \geq 2 \end{cases}$$

  $\mu(\mathrm{IN}_{bag(s)})$:= { ({{c$_1$,...,c$_n$}} , c) | c $\in$ {c$_1$,...,c$_n$} }

- **Record-Operationen**

  $\mu(\mathrm{PRJ}_{record(s_1,...,s_n),i})$: (c$_1$,...,c$_n$) $\mapsto$ c$_i$

CNT steht für COUNT, IND für INDICES, APL für APPLY, LTB für LIST TO BAG, SEL für SELECT, POS für POSITIONS, BTS für BAG TO SET, OCC für OCCURS und PRJ für PROJECT. {...} $\cup$ {...} bezieht sich auf die Vereinigung von Mengen ({c} $\cup$ {c} = {c}).

2.9 Beispiel: Operationen und Prädikate für Sortenausdrücke

Es sei die Menge M = { 2, 4, 5 } gegeben. Dann gilt $\mathrm{CNT}_{set(int)}$(M) = 3, $\mathrm{IND}_{set(int)}$(M) = { 1, 2, 3 }, $\mathrm{APL}_{+,set(int)}$(M) = 11, $\mathrm{APL}_{max,set(int)}$(M) = 5, (M,2) $\in$ $\mathrm{IN}_{set(int)}$ und (M,3) $\notin$ $\mathrm{IN}_{set(int)}$.

Für die Liste L = < 1, 2, 1 > gilt $\mathrm{POS}_{list(int)}$(L,1) = { 1, 3 }, $\mathrm{IND}_{list(int)}$(L) = { 1, 2, 3 }, $\mathrm{LTB}_{list(int)}$(L) = {{ 1, 1, 2, }}, $\mathrm{APL}_{+,list(int)}$(L) = 4 und $\mathrm{APL}_{*,list(int)}$(L) = 2.

Die obige Schreibweise für Operationen und Prädikate ist wegen der vielen Indizes recht aufwendig. Es werden daher folgende Abkürzungen und Konventionen getroffen.

2.10 Bemerkung: Konvertierungen und Aggregationsfunktionen

- list-, set-, bag- or record-Indizes kann man bei Operations-Symbolen weglassen und die Klammersetzung ist wie üblich.
- Statt UNION(x_1,x_2) wird auch x_1 UNION x_2 geschrieben. Bei der Operation CUT verfährt man analog.
- LTS(x) entspricht BTS(LTB(x)).
- SUM entspricht APL_+ abhängig von Kontext.
- MAX entspricht APL_{max} abhängig von Kontext mit max(x,y) = `if` x>y `then` x `else` y. MIN ist analog definiert.
- $AVG_{set(d)}$(x) steht für $APL_{+,set(d)}$(x)/$CNT_{set(d)}$(x). Bei Listen und Bags wird analog verfahren. Es gilt dann AVG($\emptyset$) = $\perp$.
- Anstelle von $PRJ_{record(s_1,\ldots,s_n),i}$(x) oder $PRJ_{record(s_1,\ldots,s_n),i}$(($x_1$,...,$x_n$)) wird auch x.i oder ($x_1$,...,$x_n$).i verwendet, wobei i eine Konstante zwischen 1 und n ist.

2.11 Beispiel: Aggregationsfunktionen

Um noch einmal explizit mit Multimengen zu rechnen, werden die folgenden Beispielrechnungen in denen Aggregationsfunktionen vorkommen genannt.

SUM {{1,2,1,3,2}} = SUM {{3,2,2,1,1}} = 9

SUM (BTS {{1,2,1,3,2}}) = SUM (BTS {{1,1,2,2,3}}) =
SUM {1,2,3} = 6

AVG {{1,2,2,3,1}} = SUM {{1,2,2,3,1}} / CNT {{1,2,2,3,1}} = 1.8

AVG {1,2,3} = AVG (BTS {{1,2,2,3,1}}) = 2

AVG <1,2,3,2,1> = AVG (LTB <1,2,3,2,1>) = 1.8

2.12 Beispiel: Spezifikation von Datentypen

Für unser STADT-LAND-FLUSS-Beispiel benötigt man abgesehen von den Standard-Datentypen wie int oder string noch Nicht-Standard-Datentypen um Punkte, Kreise, Polygone, Regionen, Adressen, Linienzüge und Mengen von Regionen darzustellen. Diese Datentypen werden nun spezifiziert indem eine entsprechende Signatur und zugehörige Gleichungen und Klauseln angegeben werden.

```
SORTS       point, circle, polygon, region, address,
            lines, regions
DEFINITIONS point = record(real,real)
            circle = record(point,real)
```

```
            address = record(int,string,string,int)
            polygon = list(point)
            lines = list(point)
            region = record(string,polygon)
            regions = set(region)
OPERATIONS  x , y : point → real
                 /* x- und y-Koordinate eines Punktes
            pdist : point × point → real
                 /* Abstand zweier Punkte
            start , end : lines → point
                 /* Start- und End-Punkt einer Linie
            center : circle → point
            radius : circle → real
                 /* Mittelpunkt und Radius eines Kreises
            name : region → string
            geo : region → polygon
                 /* Name und Geometrie einer Region
            pno : address → int
            city : address → string
            street : address → string
            no : address → int
                 /* Komponenten einer Adresse
            ...
PREDICATES  cccut : circle × circle
                 /* Kreise mit gemeinsamen Punkten
            ppcut : polygon × polygon
                 /* Polygone mit gemeinsamen Punkten
            lpcut : lines × polygon
                 /* Linien und Polygone mit gemeinsamen Punkten
            cpcut : circle × polygon
                 /* Kreise und Polygone mit gemeinsamen Punkten
            ...
VARIABLES   p, p1, p2 : point; l : lines
            c : circle; r : region; a : address
EQUATIONS  x(p)          = p.1 /* = PRJ_RECORD(REAL,REAL),1 (p)
           y(p)          = p.2 /* = PRJ_RECORD(REAL,REAL),2 (p)
           pdist(p1,p2)  = sqrt((x(p1)-x(p2))↑2+(y(p1)-y(p2))↑2)
           start(l)      = SEL(l,1) /* = SEL_LIST(RECORD(REAL,REAL)) (CNT(l),1)
           end(l)        = SEL(l,CNT(l)) /* = SEL_LIST(RECORD(REAL,REAL)) (l,1)
           center(c)     = c.1 /* = PRJ_RECORD(RECORD(REAL,REAL),REAL),1 (c)
           radius(c)     = c.2 /* = PRJ_RECORD(RECORD(REAL,REAL),REAL),2 (c)
           name(r)       = r.1 /* = PRJ_RECORD(STRING,LIST(RECORD(REAL,REAL))),1 (r)
           geo(r)        = r.2 /* = PRJ_RECORD(STRING,LIST(RECORD(REAL,REAL))),2 (r)
```

```
            pno(a)       = a.1 /* = PRJ_RECORD(INT,STRING,STRING,INT),1(a)
            city(a)      = a.2 /* = PRJ_RECORD(INT,STRING,STRING,INT),2(a)
            street(a)    = a.3 /* = PRJ_RECORD(INT,STRING,STRING,INT),3(a)
            no(a)        = a.4 /* = PRJ_RECORD(INT,STRING,STRING,INT),4(a)
            ...
CLAUSES     ...
```

Die zentrale Definition dieses Abschnittes ist die Definition der Syntax und der Semantik eines erweiterten Entity-Relationship-Schemas. Ein solches Schema führt zunächst in der Syntax Namen für Entity- und Relationshiptypen, Attribute, Komponenten (d. h. objektwertige Attribute) und Typkonstruktionen ein. Die Semantik assoziiert mit einem solchen Schema einen konkreten Datenbankzustand, d. h. eine Interpretation der syntaktischen Einheiten.

2.13 Definition: Erweitertes Entity-Relationship-Schema

Es sei eine Datensignatur DS gegeben. Die Syntax eines erweiterten Entity-Relationship-Schema EER(DS) über DS ist gegeben durch

a. die Mengen ENTTYPE, RELTYPE, ATTRIBUTE, COMPONENT, CONSTRUCTION $\in$ |FISET| und
b. die Funktionen participants, asource, adestination, csource, cdestination, input, output $\in$ |FUN| so daß
   - participants : RELTYPE $\rightarrow$ ENTTYPE$^+$,
   - asource : ATTRIBUTE $\rightarrow$ ENTTYPE $\cup$ RELTYPE,
   - adestination : ATTRIBUTE $\rightarrow$ { d,set(d),list(d),bag(d) | d$\in$DATA },
   - csource : COMPONENT $\rightarrow$ ENTTYPE,
   - cdestination :
     COMPONENT $\rightarrow$ { e,set(e),list(e),bag(e) | e$\in$ENTTYPE },
   - input : CONSTRUCTION $\rightarrow$ $\mathcal{F}$(ENTTYPE)-{$\emptyset$}, und
   - output : CONSTRUCTION $\rightarrow$ $\mathcal{F}$(ENTTYPE)-{$\emptyset$}.

Wenn r $\in$ RELTYPE mit participants(r) = $<e_1,...,e_n>$, wird dies dargestellt durch r(e_1,...,e_n). Wenn a $\in$ ATTRIBUTE mit asource(a)=e oder asource(a)=r und adestination(a)=d, wird dies dargestellt durch a : e $\rightarrow$ d oder a : r $\rightarrow$ d. Wenn c $\in$ COMPONENT mit csource(c)=e und cdestination(c)=e', wird dies dargestellt durch c : e $\rightarrow$ e'. Wenn c $\in$ CONSTRUCTION mit input(c) = $\{i_1,...,i_n\}$ und output(c) = $\{o_1,...,o_m\}$, wird dies dargestellt durch c(i_1,...,i_n;o_1,...,o_m).

Für zwei unterschiedliche Konstruktionen c_1,c_2 $\in$ CONSTRUCTION müssen die folgenden Bedingungen gültig sein:

a. output(c_1) $\cap$ output(c_2) = $\emptyset$

b. Es ist nicht erlaubt, daß connection$^+$(e,e) gilt für irgendein e $\in$ ENTTYPE, wobei connection$^+$ der transitive Abschluß von connection ist: Wenn $e_{in} \in$ input(c) und $e_{out} \in$ output(c) für c $\in$ CONSTRUCTION, dann gilt connection(e_{in},e_{out}).

Die Semantik eines erweiterten Entity-Relationship-Schemas EER(DS) ist gegeben durch

a. eine Funktion μ[ENTTYPE] $\in$ |FUN| so daß μ[ENTTYPE] : ENTTYPE $\to$ |FISET| und $\perp \in \mu$[ENTTYPE](e) für jedes e $\in$ ENTTYPE,

b. eine Funktion μ[RELTYPE] $\in$ |FUN| so daß μ[RELTYPE] : RELTYPE $\to$ |FISET|, und r(e_1,...,e_n) impliziert μ[RELTYPE](r) $\subseteq$ (μ[ENTTYPE](e_1)-$\{\perp\}$ $\times ... \times$ μ[ENTTYPE](e_n)-$\{\perp\}$) $\cup$ $\{\perp\}$, und $\perp \in$ μ[RELTYPE](r) für jedes r $\in$ RELTYPE,

c. eine Funktion μ[ATTRIBUTE] $\in$ |FUN| so daß μ[ATTRIBUTE] : ATTRIBUTE $\to$ |FUN| und a : e $\to$ d impliziert μ[ATTRIBUTE](a) : μ[ENTTYPE](e) $\to$ μ[EXPR(DATA)](d), a : r $\to$ d impliziert μ[ATTRIBUTE](a) : μ[RELTYPE](r) $\to$ μ[EXPR(DATA)](d),

d. eine Funktion μ[COMPONENT] $\in$ |FUN| so daß μ[COMPONENT] : COMPONENT $\to$ |FUN| und c : e $\to$ e' impliziert μ[COMPONENT](c) : μ[ENTTYPE](e) $\to$ μ[EXPR(ENTTYPE)](e'), und

e. eine Funktion μ[CONSTRUCTION] $\in$ |FUN| so daß μ[CONSTRUCTION] : CONSTRUCTION $\to$ |FUN| und c(i_1,...,i_n;o_1,...,o_m) impliziert μ[CONSTRUCTION](c) : $\cup_{j\in 1..m}$ μ[ENTTYPE](o_j) $\to$ $\cup_{j\in 1..n}$ μ[ENTTYPE](i_j), wobei die Funktion μ[CONSTRUCTION](c) injektiv ist.

μ[ATTRIBUTE](a), μ[COMPONENT](c), und μ[CONSTRUCTION](c) geben den undefinierten Wert weiter: μ[ATTRIBUTE](a)($\perp$) = $\perp$, μ[COMPONENT](c)($\perp$) = $\perp$, und μ[CONSTRUCTION](c)($\perp$) = $\perp$.

2.14 Beispiel: Syntax eines erweiterten Entity-Relationship-Schemas

Die obige Definition wird nun anhand des STADT-LAND-FLUSS-Beispiels erläutert. Es liegen die folgenden syntaktischen Einheiten vor.

- ENTTYPE = { STADT, HAFENSTADT, LAND, PERSON, FLUSS, SEE, MEER, GEWAESSER }
- RELTYPE = { liegt-in, liegt-an, fliesst-durch, muendet-in }
- ATTRIBUTE = { StName, StEinw, StGeo, LName, LEinw, Regform, Regionen, Alter, PName, Adr, Laenge, FName, FGeo, SName, SGeo, MName, MGeo, Belastung, Schiffbar }

- COMPONENT = { Hauptstadt, Minister, Praesident }
- CONSTRUCTION = { ist, sind }

Hierauf sind die weiteren, die Syntax betreffenden Funktionen wie folgt definiert.

- participants : liegt-in $\mapsto$ <STADT,LAND>, ...
- asource : StName $\mapsto$ STADT, ...
- adestination : StName $\mapsto$ string, ...
- csource : Minister $\mapsto$ LAND, ...
- cdestination : Minister $\mapsto$ list(PERSON), ...
- input : ist $\mapsto$ { STADT }, sind $\mapsto$ { FLUSS, SEE, MEER }
- output : ist $\mapsto$ { HAFENSTADT }, sind $\mapsto$ { GEWAESSER }

2.15 Beispiel: Semantik eines erweiterten Entity-Relationship-Schemas

Wie oben schon erwähnt ordnet die Semantik eines erweiterten Entity-Relationship-Schemas dem Schema eine Ausprägung zu, die hier auch Datenbankzustand genannt wird.

- Entitytypen:
  μ(STADT) = { st_1, st_2, st_3, $\perp$ } (eigentlich μ[ENTTYPE](STADT))
  μ(FLUSS) = { f_1, f_2, $\perp$ }
  μ(GEWAESSER) = { g_1, g_2, g_3, g_4, $\perp$ }
  ...
- Relationshiptypen und Attribute:

$$\begin{array}{llllll}
\mu(\text{liegt-an}) & = \{ & <st_1,f_1>, & <st_2,f_1>, & <st_2,f_2> & \} \\
\ldots & & & & & \\
\mu(\text{laenge}) & & \downarrow & \downarrow & \downarrow & \\
 & & 17 & 3 & 12 &
\end{array}$$

  ...
- Komponenten:
  μ(Praesident) : $l_1 \mapsto p_{17}$, $l_4 \mapsto p_6$, ...
  ...
- Typkonstruktionen:

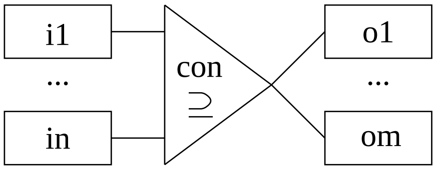
$$\begin{array}{ll}
\mu(\text{sind}): & \mu(\text{GEWAESSER}) \rightarrow \mu(\text{FLUSS}) \cup \mu(\text{SEE}) \cup \mu(\text{MEER}) \\
\mu(\text{sind}): & g_1 \mapsto s_7, \\
 & g_2 \mapsto f_1, \\
 & g_3 \mapsto f_2, \\
 & g_4 \mapsto m_9, \\
 & \perp \mapsto \perp
\end{array}$$

  ...

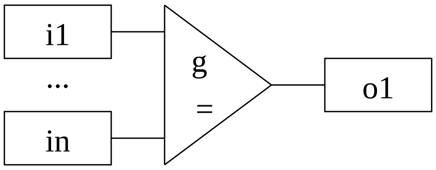
Die Funktion μ(sind) muß, da sie im Diagram mit = gekennzeichnet ist, nicht nur injektiv sondern auch zuätzlich surjektiv und insgesamt damit bijektiv sein, d. h. jedem Ausgangsentity muß genau ein Eingangsentity zugeordnet werden. Für den obigen Datenbankzustand bedeutet dies, daß es nur ein SEE-Entity und ein MEER-Entity geben darf: μ(SEE) = { s_7, $\perp$ } und μ(MEER) = { m_9, $\perp$ }.

2.16 Bemerkung: Typkonstruktion

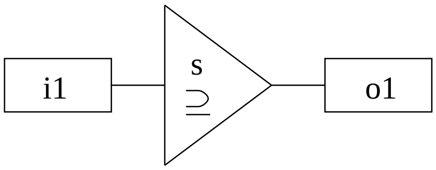
Wie in Abbildung 10 dargestellt sieht der allgemeine Fall der Typkonstruktion vor, daß es eine beliebige, endliche Anzahl von Eingangsentitytypen i1, ..., in und eine beliebige, endliche Anzahl von Ausgangsentitytypen o1, ..., om gibt. Die "Pfeilrichtung" des Dreiecks soll dabei andeuten, daß die Eingangsentities auf die Ausgangsentitytypen aufgeteilt werden. Intention dabei ist, die Eingangsentities disjunkt auf die Ausgangsentitytypen zu verteilen.



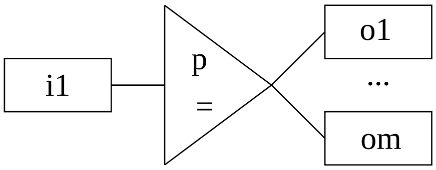
Allgemeiner Fall der Typkonstruktion



Generalisierung



Spezialisierung



Partition

Abbildung 10: Spezialfälle für Typkonstruktionen

Aus dem allgemeinen Fall der Typkonstruktion ergeben sich eine Reihe von Spezialfällen: Unter einer Generalisierung versteht man die Zusammenfassung von n Eingangsentitytypen zu einem Ausgangsentitytyp mit der zusätzlichen =-Forderung. Ein Beispiel hierfür ist die Typkonstruktion "sind". Mit Spezialisierung bezeichnet man Typkonstruktionen mit einem Eingangstyp und einem Ausgangstyp. Die Typkonstruktion "ist" stellt dafür ein Beispiel dar. Unter einer Partition versteht man die disjunkte, vollständige Aufteilung von Eingangsentities auf unterschiedliche Ausgangstypen. Ein Beispiel hierfür wäre die Aufteilung von Personen (Entitytyp PERSON) nach dem Geschlecht (Entitytypen MANN und FRAU).

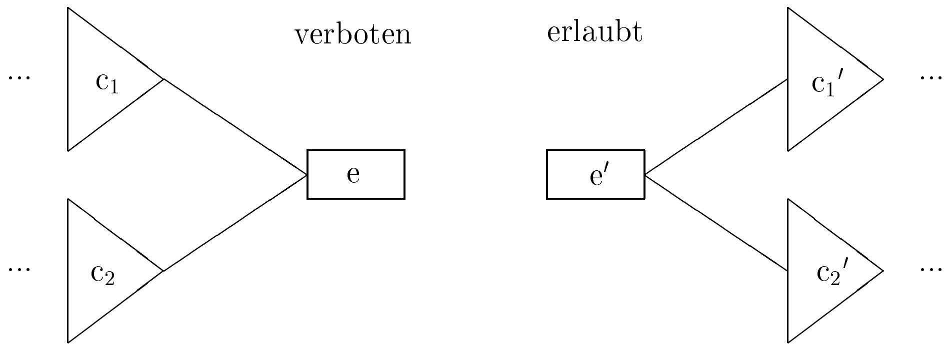


Abbildung 11: Erlaubte und verbotene Typkonstruktionen

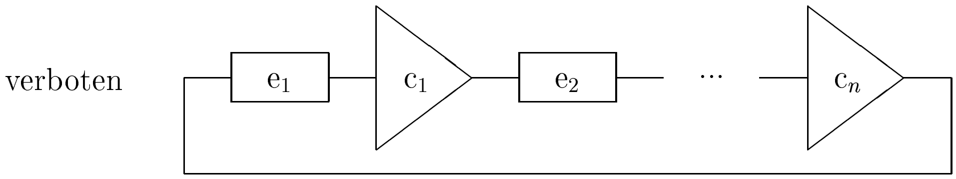


Abbildung 12: Azyklizität der Typkonstruktion

Wie in der formalen Definition gefordert, gibt es zwei wesentliche Einschränkungen an Typkonstruktionen: (a) in Abbildung 11 ist skizziert, daß es nicht erlaubt ist, daß ein Entitytyp Ausgangsentitytyp von zwei unterschiedlichen Typkonstruktionen ist; (b) nach Abbildung 12 ist es nicht erlaubt durch Typkonstruktionen Zyklen einzuführen. Diese beiden Bedingungen stellen die Eindeutigkeit der konstruierten Entities, d. h. der Ausgangsentities, sicher. Zu jedem konstruierten Entity $\underline{e}$ gibt es eindeutig Typkonstruktionen c_1, ..., c_n, andere konstruierte Entities $\underline{e}_1$ vom Typ e_1, ..., $\underline{e}_{n-1}$ vom Typ e_{n-1} und ein nicht-konstruiertes Entity $\underline{e}_n$ vom Typ e_n, so daß die

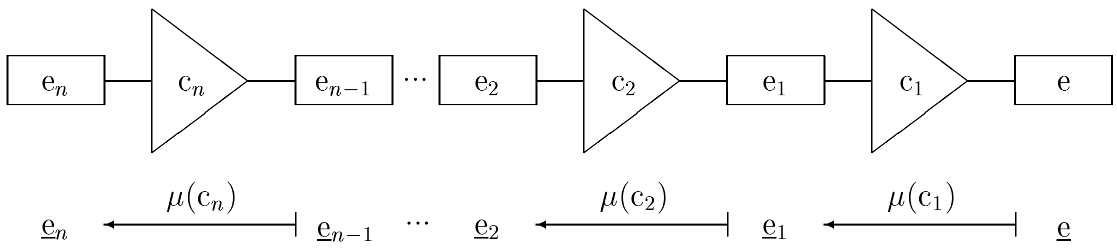


Abbildung 13: Eindeutigkeit der konstruierten Entities

Entities (wie in Abbldung 13 dargestellt) durch die semantischen Funktionen μ[CONSTRUCTION](c_i) (bzw. kurz $\mu(c_i)$) miteinander verbunden sind.

Allgemeine Typkonstruktionen ohne =-Einschränkung entsprechen dem was im Abschnitt über semantische Datenmodelle Rollenhierarchien genannt worden waren. Typkonstruktionen mit =-Einschränkung können mit Generalisierungshierarchien verglichen werden.

Ein weiteres Konzept im erweiterten Entity-Relationship-Modell sind sogenannte Rollennamen, die mit Relationshiptypen verbunden sind. Rollennnamen stellen quasi einen formalen Kommentar dar, der Auskunft über die "Rolle" gibt, die ein Entitytyp in einer Beziehung spielt.

2.17 Definition: Rollennamen

Es sei ein erweitertes Entity-Relationship-Schema EER(DS) und eine Menge NAMES gegeben. Rollen werden dargestellt durch eine Funktion roles $\in$ |FUN| so daß roles: RELTYPE $\rightarrow$ (NAMES $\cup$ $\{\perp\}$ $)^+$ und r(e_1,...,e_m) $\in$ RELTYPE impliziert roles(r) = <n_1,...,n_m>, wobei $n_i \neq n_j$ für i$\neq$j (mit Ausnahme von $\perp$). Falls der Rollenname n_i (i=1..m) für r(e_1,...,e_m) $\in$ RELTYPE $\perp$ ist und falls die Entity Sorte e_i nur einmal in <e_1,...,e_m> auftritt, wird angenommen, daß e_i als Rollenname benutzt wird. Rollennamen müssen angegeben werden, falls ein Entitytyp mehr als einmal in einem Relationshiptyp auftaucht.

2.18 Beispiel: Rollennamen im STADT-LAND-FLUSS-Beispiel

Im laufenden Beispiel sind durch das Diagramm die folgenden Rollennamen festgelegt worden.

roles :	liegt-in	$\rightarrow$	<Stadt,Land'>
	liegt-an	$\rightarrow$	<HStadt,Gewaesser>
	fliesst-durch	$\rightarrow$	<Land,Fluss>
	muendet-in	$\rightarrow$	<Fluss',Muendung>

2.19 Beispiel: Notwendigkeit von Rollennamen

Ein Beispiel in dem Rollennamen angegeben werden müssen ist die in Abbildung 14 dargestellte Diskurswelt zur Modellierung von Autobahnen, Auffahrten und Autobahnkreuzen.

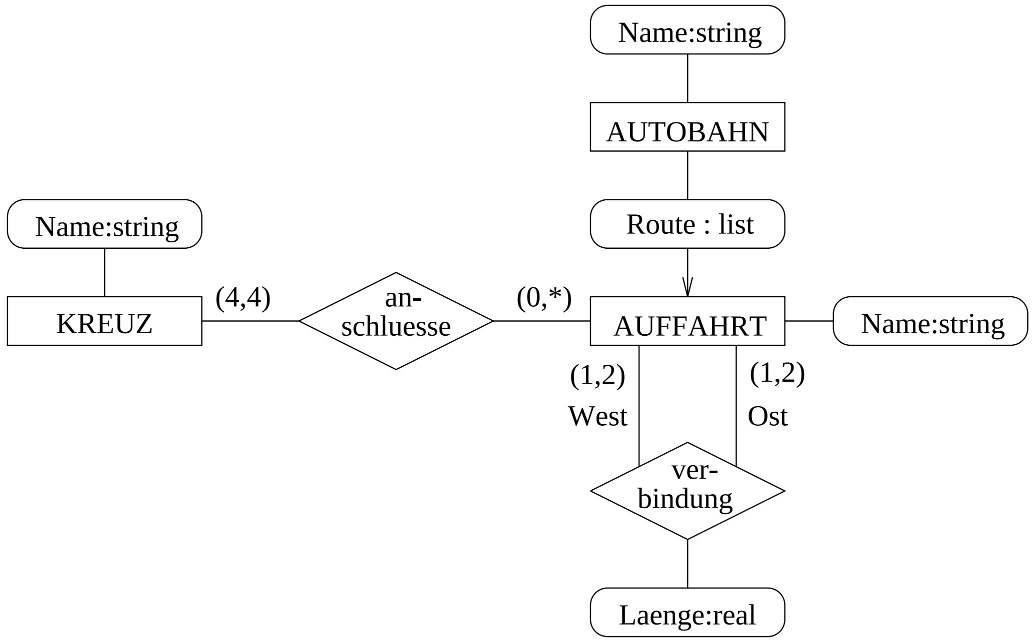


Abbildung 14: Modellierung von Autobahnen

Um eindeutig auf die beteiligten Auffahrten zugreifen zu können, werden für den Relationshiptyp verbindung die Rollennamen West und Ost angegeben. Somit wird vb.West die westliche Auffahrt und vb.Ost die östliche Auffahrt eines Relationships vb $\in \mu$(verbindung) kennzeichnen.

2.2 Der erweiterte Entity-Relationship-Kalkül

Wie schon im Abschnitt über semantische Datenmodelle dargestellt, können komplexe Sachverhalte der Diskurswelt nicht allein mit der strukturellen Komponente eines Datenbank-Schemas notiert werden. Für komplexe Sachverhalte benötigt man explizite Integritätsbedingungen, die im Braunschweiger Ansatz als Formeln des erweiterten Entity-Relationship-Kalküls formuliert werden.

2.20 Motivation

Als erstes Beispiel für eine Integritätsbedingung wird die Forderung formuliert, daß das Durchschnittsalter der Minister eines Landes kleiner gleich 65 sein muß.

$\forall$ (l:LAND) AVG -[Alter(p) | (p:PERSON) $\wedge$ p IN Minister(l)]- $\leq$ 65

Der Kern des Kalküls sind Anfragen, d. h. Terme der Form -[t | d $\wedge$ f]- wobei t der Ergebnis-Term, d Variablen-Deklarationsfolge und f eine qualifizierende (d. h. einschränkende) Formel ist. Diese Anfrageterme sind in Analogie zu Anfragen in SQL zu sehen. Eine SQL-ähnliche Notation für diese Terme wäre `(SELECT` t `FROM` d `WHERE` f `)`.

Das Ergebnis eines solchen Terms ist i. a. eine Multimenge (oder Bag). Im obigen Beispiel könnte beispielsweise die Zahl 42 dreimal im Ergebnis der Unteranfrage auftauchen, wenn es in einem Land 3 Minister im Alter von 42 Jahren gibt.

Der erweiterte Entity-Relationship-Kalkül ist im Gegensatz zu SQL streng orthogonal, d. h. die obigen Terme sind wieder in den Einheiten t, d und f erlaubt, sofern die Sorte paßt. Hierzu werden später noch eine Reihe von Beispielen vorgestellt werden. Die Motivation für den Kalkül ergibt sich aber nicht nur aus der Formulierung von Integritätsbedingungen. Da Anfrageterme essentieller Bestandteil des Kalküls sind, lassen sich natürlich auch Anfragen mit ihm formulieren.

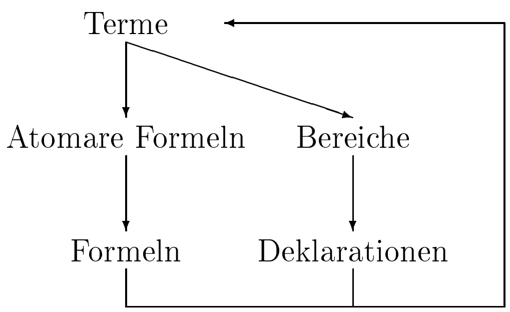


Abbildung 15: Nicht-hierarchische Struktur des Kalküls

Der erweiterte Entity-Relationship-Kalkül hat wie in Abbildung 15 dargestellt eine nicht-hierarchische Struktur, d. h. mit Termen lassen sich Formeln bilden und mit Formeln und Deklarationen wiederum Terme. Dies ist im Gegensatz zu klassischen Datenbank-Kalkülen wie dem Tupel- oder Bereichs-Kalkül. Dort werden aus Termen zwar auch Formeln gebildet und schließlich zu Anfragen zusammengesetzt, aber diese Anfragen können dort nicht wieder in anderen Termen auftreten. Eine Konsequenz daraus ist, daß z. B. Aggregationsfunktionen in diesen Kalkülen nicht dargestellt werden können, wohl aber im erweiterten Entity-Relationship-Kalkül. Dieser nicht-hierarchische Aufbau wird nun anhand der obigen Integritätsbedingung erläutert: Die Formel "p IN Minister(l)", die die Terme "p", "l" und "Minister(l)" enthält, ist Teil des Terms "-[Alter(p) | (p:PERSON) ∧ p IN Minister(l)]-", welcher wiederum Teil der gesamten Formel ist. Man sieht hier also, daß z. B. Formeln Bestandteil von Termen und diese wieder Teil von Formeln sein können. Die Formel läßt sich wie in Abbildung 16 dargestellt weiter syntaktisch aufgliedern. Die Begriffe Term, Formel und Deklaration sind hier mit T, F und D abgekürzt.

$$\underbrace{\forall\ \underbrace{\text{(l:LAND)}}_{D}\ \underbrace{\underbrace{\text{AVG -[}\ \underbrace{\underbrace{\text{Alter(p)}}_{T} \mid \underbrace{\text{(p:PERSON)}}_{D} \wedge \underbrace{\underbrace{\text{p}}_{T}\ \text{IN}\ \underbrace{\text{Minister(l)}}_{T}}_{F}\ \text{]}}_{T}\text{-}}_{T} \le \underbrace{65}_{T}}_{F}}_{F}$$

Abbildung 16: Struktur einer Integritätsbedingung

Bevor der Kalkül nun in Syntax und Semantik definiert wird, wird die syntaktische Situation mit den gegebenen Sorten, Operationen und Prädikaten in der folgenden Definition zusammengefaßt.

2.21 Notation: SORT, OPNS, und PRED

Es sei ein erweitertes Entity-Relationship-Schema EER(DS) gegeben. SORT bezieht sich auf die Vereinigung von allen Daten-, Entity-, und Relationship-Sorten.

SORT = DATA ∪ ENTTYPE ∪ RELTYPE

Die Erweiterung von OPNS und PRED bezüglich eines gegebenen Entity-Relationship-Schema ist folgendermaßen festgelegt.

$\text{OPNS}_{EER} = \text{ATTRIBUTE} \cup \text{COMPONENT}$

$\text{PRED}_{EER} = \text{RELTYPE}$

$\text{OPNS} = \text{OPNS}_{DS} \cup \text{OPNS}_{EER} \cup \text{OPNS(SORT)}$

$\text{PRED} = \text{PRED}_{DS} \cup \text{PRED}_{EER} \cup \text{PRED(SORT)}$

Die semantischen Festlegungen für μ[SORT], μ[OPNS$_{EER}$], μ[PRED$_{EER}$], μ[OPNS] und μ[PRED] werden durch die korrespondierenden μ's auf der rechten Seite festgelegt. Jedes Element im Wertuniversum $\hat{\mu}$[EXPR(SORT)] (bis auf $\perp$) besitzt nun eine eindeutige Sorte, d. h. es gibt eine eindeutige Funktion

sort: $\hat{\mu}$[EXPR(SORT)]-$\{\perp\}$ $\rightarrow$ EXPR(SORT).

2.22 Beispiel: SORT, OPNS, und PRED

- SORT = { int, ... } $\cup$
{ LAND, ... } $\cup$
{ liegt-in, ... }
- EXPR(SORT) = SORT $\cup$
{ record(LAND,int), set(liegt-in), ... }
- OPNS = { + : int $\times$ int $\rightarrow$ int, ...} $\cup$
{ StName : STADT $\rightarrow$ string,
Minister : LAND $\rightarrow$ set(PERSON), ... } $\cup$
{ CNT : set(PERSON) $\rightarrow$ int, ...}
- PRED = { $\leq$: int $\times$ int, ...} $\cup$
{ liegt-in : STADT $\times$ LAND, ... } $\cup$
{ IN : set(PERSON) $\times$ PERSON, ...}

Wie im obigen Beispiel schon angedeutet, benötigt man in Integritätsbedingungen und Anfragen Variablen. Es wird eine fest vorgegeben Menge von Variablen vorgesehen, die alle eine feste Sorte besitzen. Belegungen sind dann Zuweisungen an die Variablen mit Werten die verträglich mit der Variablen-Sorte sind.

2.23 Definition: Variablen und Belegungen

Es sei ein erweitertes Entity-Relationship-Schema EER(DS), eine Menge VAR von Variablen und eine Funktion type : VAR $\rightarrow$ EXPR(SORT) gegeben (v $\in$ VAR$_s$ steht für v $\in$ VAR und type(v)=s). Die Menge der Belegungen ASSIGN ist definiert durch

ASSIGN := { $\beta \in$ |FUN| | β : VAR$_s$ $\rightarrow$ $\hat{\mu}$[EXPR(SORT)](s) für s $\in$ EXPR(SORT) }

Die Belegung ϵ : VAR $\rightarrow$ $\{\perp\}$ wird leere Belegung genannt.

2.24 Beispiel: Variablen

Es Variablennamen verwendet, die Abkürzungen der entsprechenden zugrundeliegenden Bereiche sind: l, l_1, l_2, ... für LAND; st, st_1, st_2, ... für STADT; la, la_1, la_2 für liegt-in; etc.

Variablen werden durch Deklarationen an endliche Mengen gebunden. Es werden zunächst nur einfache Deklarationen, d. h. Variablen für Entity- und Relationshiptypen vorgesehen.

2.25 Definition: Einfache Form der Deklarationen

Die Syntax von Deklarationen ist gegeben durch eine Menge DECL und Funktionen free, decl : DECL $\rightarrow \mathcal{F}$(VAR). Wenn v $\in$ VAR_s für s $\in$ ENTTYPE or s $\in$ RELTYPE, dann (v:s) $\in$ DECL, free((v:s)) := $\emptyset$, und decl((v:s)) := {v}.

Die Semantik einer Deklaration ist eine Relation μ[DECL] $\subseteq$ DECL $\times$ ASSIGN. ((v:s),β) $\in$ μ[DECL], gdw. β(v) $\in$ μ[SORT](s).

2.26 Beispiel: Deklarationen

- (l:LAND) ist eine Deklaration der Variablen l; decl(l:LAND) = { l }; $\beta(l) \in \mu(LAND)$.
- (la:liegt-an) ist eine Deklaration der Variablen la; decl(la:liegt-an) = { la }; $\beta(la) \in \mu$(liegt-an) $\subseteq$ μ(STADT) $\times$ μ(GEWAESSER).

Zentraler Begriff des erweiterten Entity-Relationship-Kalküls ist der Begriff des Terms. Terme werden zu Werten ausgewertet. Durch die reiche Sortenstruktur, die durch die Sortenausdrücke induziert wird, können insbesondere sehr komplexe Anfrage-Terme gebildet werden.

2.27 Definition: Terme

Die Syntax von Termen ist gegeben durch eine Menge TERM, und Funktionen sort : TERM $\rightarrow$ EXPR(SORT) und free : TERM $\rightarrow \mathcal{F}$(VAR) (t $\in$ $TERM_s$ steht für t $\in$ TERM und sort(t)=s).

i. Wenn v $\in$ VAR_s, dann v $\in$ $TERM_s$ und free(v) := {v}.

ii. Wenn v $\in$ VAR_r, r(e_1,...,e_n) $\in$ RELATION und roles(r) = $< n_1,...,n_n >$, dann v.i $\in$ $TERM_{e_i}$ (i=1..n) und v.n_i $\in$ $TERM_{e_i}$ (i=1..n) mit free(v.i) := {v} und free(v.n_i) := {v}.

iii. Wenn c $\in$ CONSTRUCTION, s_{out} $\in$ output(c) und t_{in} $\in$ $TERM_{s_{in}}$, dann $s_{out}(t_{in})$ $\in$ $TERM_{s_{out}}$ mit free($s_{out}(t_{in})$) := free(t_{in}).

iv. Wenn c $\in$ CONSTRUCTION, s_{in} $\in$ input(c) und t_{out} $\in$ $TERM_{s_{out}}$, dann $s_{in}(t_{out})$ $\in$ $TERM_{s_{in}}$ mit free($s_{in}(t_{out})$) := free(t_{out}).

v. Wenn $\omega : s_1 \times ... \times s_n \rightarrow s$ $\in$ OPNS und t_i $\in$ $TERM_{s_i}$ (i=1..n), dann $\omega(t_1,...,t_n)$ $\in$ $TERM_s$ mit free($\omega(t_1,...,t_n)$) := free(t_1) $\cup$ ... $\cup$ free(t_n).

vi. Es seien $t_i \in TERM_{s_i}$ (i=1..n), $d \in DECL$ und $f \in FORM$ gegeben.
Dann -[$t_1,....,t_n$ | $d \wedge f$]- $\in$ TERM.
sort(-[$t_1,....,t_n$ | $d \wedge f$]-) =
`if` n=1 `then` bag(s_1) `else` bag(record($s_1,...,s_n$)).
free(-[$t_1,....,t_n$ | $d \wedge f$]-) :=
(free(d) $\cup$ free(t_1) $\cup$ ... $\cup$ free(t_n) $\cup$ free(f)) - decl(d).

Die Semantik von Termen ist eine Funktion μ[TERM] : TERM $\times$ ASSIGN $\rightarrow \hat{\mu}$[EXPR(SORT)].

i. μ[TERM](v,β) := β(v).

ii. μ[TERM](v.i,β) := `if` μ[TERM](v,β)=($x_1,...,x_n$) `then` x_i `else` $\perp$.

iii. μ[TERM](s_{out}(t_{in}),β) :=

`if` $\exists$ e$\in$ μ[ENTTYPE](s_{out}) mit
μ[CONSTRUCTION](c)(e)=μ[TERM](t_{in},β)
`then` e `else` $\perp$.

iv. μ[TERM](s_{in}(t_{out}),β) :=

`if` μ[CONSTRUCTION](c)(μ[TERM](t_{out},β))
$\in$ μ[ENTTYPE](s_{in})
`then` μ[CONSTRUCTION](c)(μ[TERM](t_{out},β)) `else` $\perp$.

v. μ[TERM](ω($t_1,...,t_n$),β) :=
μ[OPNS](ω)(μ[TERM](t_1,β),...,μ[TERM](t_n,β)).

vi. μ[TERM](-[$t_1,...,t_n$ | $d \wedge f$]-, β) :=

{{ (μ[TERM](t_1,β'),...,μ[TERM](t_n,β')) |
es gibt β' $\in$ ASSIGN mit β'(v)=β(v) für v $\in$ VAR-decl(d)
und (d,β') $\in$ μ[DECL] und (f,β') $\in$ μ[FORM];
sprich: β'(v) $\neq$ β(v) erlaubt falls v $\in$ decl(d)
}}.

2.28 Beispiel: Terme

1. l (nach i.)
2. la.1, la.HStadt, la.HAFENSTADT (nach ii.)
3. GEWAESSER(f) (nach iii.)
4. FLUSS(g) (nach iv.)
5. LName(l) (nach v.)
6. PName(Praesident(l)) (nach v.)
7. Minister(l) (nach v.)

8. CNT(Minister(l)) (nach v.)
9. 3.14 $*$ radius(StGeo(st)) (nach v.)
10. pdist(center(StGeo(st_1)),center(StGeo(st_2))) (nach v.)
11. -[LName(l), PName(Praesident(l)) | (l:LAND)]- (nach vi.)
12. PName(Minister(l)) ist kein korrekter Term sehr wohl aber
    -[PName(p) | (p:PERSON) $\wedge$ p IN Minister(l)]- (nach vi.)
    Dieser Term hat die freie Variable l.
13. -[LName(l), -[StName(st) | (st:STADT) $\wedge$ liegt-in(st,l)]- |
    (l:LAND) $\wedge$ Regform(l) = 'DEMOKRATIE']- (nach vi.)

2.29 Bemerkung: Multimengen

μ[TERM](-[t_1,...,t_n | d $\wedge$ f]-,β) kann eine echte Multimenge ergeben obwohl β fest gegeben ist. Die Auswertung der t_i erfolgt bezüglich aller Belegungen β' die sich von β in den in d deklarierten Variablen unterscheiden dürfen. Man betrachte folgende Auswertung.

μ[TERM](-[Alter(p) | (p:PERSON)]- , β)

Unter der Annahme, daß nur 4 Personen { p_1, p_2, p_3, p_4 } im aktuellen Datenbank-Zustand vorhanden sind, könnte das Ergebnis z. B. $\{\{$ 25, 42, 42, 60 $\}\}$ sein, da die Variable p mit p_1, p_2, p_3, p_4 belegbar ist.

Unter der weiteren Annahme, daß auch nur 2 Städte { st_1, st_2 } in aktuellen Zustand sind, würde die Auswertung von

μ[TERM](-[Alter(p) | (p:PERSON; (st:STADT))]- , β)

das Ergebnis $\{\{$ 25, 25, 42, 42, 42, 42, 60, 60 $\}\}$ liefern, da $<p,st>$ mit $<p_1,st_1>$, $<p_1,st_2>$, ..., $<p_4,st_2>$ belegbar ist, d. h. es gibt acht unterschiedliche Belegungen für die Variablen p und st.

In -[t | d $\wedge$ f]- legen die unterschiedlichen Belegegungen der deklarierten Variablen die Duplikate in der Resultatsmultimenge fest, sprich t wird bezüglich aller dieser Belegegungen ausgewertet.

Formeln, d. h. logische Aussagen formuliert in einer Spielart des Prädikatenkalküls dienen zum einen zur Formulierung von Integritätsbedingungen, zum anderen in Anfrage-Termen zur zusätzlichen Einschränkung von Resultaten.

2.30 Definition: Formeln

Die Syntax von Formeln ist gegeben durch eine Menge FORM und eine Funktion free : FORM $\rightarrow \mathcal{F}$(VAR).

i. Wenn $\pi : s_1 \times ... \times s_n \in$ PRED und $t_i \in TERM_{s_i}$ (i=1..n), dann
   $\pi(t_1,...,t_n) \in$ FORM und free($\pi(t_1,...,t_n)$) := free(t_1) $\cup ... \cup$ free(t_n).

ii. Wenn t_1,t_2 $\in$ $TERM_s$, dann t_1=t_2 $\in$ FORM und free(t_1=t_2) := free(t_1) $\cup$ free(t_2).

iii. Wenn t $\in$ TERM, dann UNDEF(t) $\in$ FORM und free(UNDEF(t)) := free(t).

iv. Wenn c $\in$ CONSTRUCTION, s_{in} $\in$ input(c), s_{out} $\in$ output(c), t_{in} $\in$ $TERM_{s_{in}}$, und t_{out} $\in$ $TERM_{s_{out}}$, dann t_{in} IS t_{out} $\in$ FORM mit free(t_{in} IS t_{out}) := free(t_{in}) $\cup$ free(t_{out}).

v. Wenn f $\in$ FORM, dann $\neg$(f) $\in$ FORM und free($\neg$(f)) := free(f).

vi. Wenn f_1, f_2 $\in$ FORM, dann ($f_1 \vee f_2$) $\in$ FORM und free(($f_1 \vee f_2$)) := free(f_1) $\cup$ free(f_2).

vii. Wenn f $\in$ FORM und d $\in$ DECL, dann $\exists$d(f) $\in$ FORM und free($\exists$d(f)) := (free(f)-decl(d)) $\cup$ free(d).

Die Semantik von Formeln ist eine Relation μ[FORM] $\subseteq$ FORM $\times$ ASSIGN.

i. (π(t_1,...,t_n),β) $\in$ μ[FORM] gdw. (μ[TERM](t_1,β),...,μ[TERM](t_n,β)) $\in$ μ[PRED](π).

ii. (t_1=t_2,β) $\in$ μ[FORM] gdw. μ[TERM](t_1,β)=μ[TERM](t_2,β).

iii. (UNDEF(t),β) $\in$ μ[FORM] gdw. μ[TERM](t,β)=$\perp$.

iv. (t_{in} IS t_{out},β) $\in$ μ[FORM] gdw. μ[CONSTRUCTION](c)(μ[TERM])(t_{out},β) = μ[TERM](t_{in},β).

v. ($\neg$(f),β) $\in$ μ[FORM] gdw. nicht (f,β) $\in$ μ[FORM].

vi. (($f_1 \vee f_2$),β) $\in$ μ[FORM] gdw. (f_1,β) $\in$ μ[FORM] oder (f_2,β) $\in$ μ[FORM].

vii. ($\exists$d(f),β) $\in$ μ[FORM] gdw. es gibt β' $\in$ ASSIGN mit β'(v)=β(v) für v $\in$ VAR-decl(d) und (f,β') $\in$ μ[FORM] und (d,β') $\in$ μ[DECL].

Die in der Definition nicht genannten logischen Konnektive und der Allquantor lassen sich mit Hilfe der definierten Konnektive und mit dem Existenzquantor festlegen.

2.31 Bemerkung: Andere Konnektive und Allquantor

($f_1 \wedge f_2$) := $\neg$ ($\neg$ f_1 $\vee$ $\neg$ f_2),

($f_1 \Rightarrow f_2$) := ($\neg$ f_1 $\vee$ f_2),

($f_1 \Leftrightarrow f_2$) := ($f_1 \Rightarrow f_2$) $\wedge$ ($f_2 \Rightarrow f_1$),

$\forall$ d (f) := $\neg$ $\exists$ d($\neg$(f)).

DEF(t) := $\neg$ UNDEF(t).

2.32 Beispiel: Formeln

1. liegt-an(st,g)
2. ppcut(MGeo(m_1),MGeo(m_2))
3. p IN Minister(l)
4. p=Praesident(l)
5. LTB(Minister(l)) = -[p | (p:PERSON) $\wedge$ Alter(p) > 50]-
6. UNDEF(addr(p))
7. m IS g
8. $\forall$ (st:STADT) (StEinw(st) > 100,000) $\Rightarrow$
   $\exists$ (p:PERSON) (buergermeister(p,st))

   In dieser Integritätsbedingung wird zusätzlich ein Relationshiptyp buergermeister(PERSON,STADT) vorausgesetzt.

Bereiche sind spezielle mengen-wertige Terme. Sie dienen zur Bindung von Variablen und werden stets zu endlichen Mengen ausgewertet. Daher ist es auch möglich, noch einen Potenzmengen-Operator quasi als Bereichs-Konstruktor mitaufzunehmen.

2.33 Definition: Bereiche

Die Syntax von Bereichen ist gegeben durch eine Menge RANGE und Funktionen domain : RANGE $\rightarrow$ EXPR(SORT) und free : RANGE $\rightarrow$ $\mathcal{F}$(VAR) ($r_s \in$ RANGE steht für r $\in$ RANGE und domain(r)=s).

i. Wenn s $\in$ ENTTYPE or s $\in$ RELTYPE, dann s $\in$ RANGE_s und free_{RANGE}(s) := $\emptyset$.
ii. Wenn t $\in$ $\text{TERM}_{set(s)}$ mit s $\in$ EXPR(SORT), dann t $\in$ RANGE_s und free_{RANGE}(t) := free_{TERM}(t).
iii. Wenn r $\in$ RANGE_s, dann set(r) $\in$ $\text{RANGE}_{set(s)}$ und free_{RANGE}(set(r)) := free(r).

Die Semantik von Bereichen ist eine Funktion μ[RANGE] : RANGE $\times$ ASSIGN $\rightarrow$ $\hat{\mu}$[EXPR(SORT)].

i. μ[RANGE](s,β) := μ[SORT](s).
ii. μ[RANGE](t,β) := μ[TERM](t,β).
iii. μ[RANGE](set(r),β) := $\mathcal{F}$(μ[RANGE](r,β)).

2.34 Beispiel: Bereiche

1. PERSON ist Bereich vom Typ PERSON (PERSON $\in$ RANGE_{PERSON}).

2. liegt-an ist Bereich vom Typ liegt-an.
3. LTS(Minister(l)):set(PERSON) ist Bereich vom Typ PERSON.
4. Regionen(l):set(region) ist Bereich vom Typ region.
5. set(GEWAESSER) ist Bereich vom Typ set(GEWAESSER).

Mit Hilfe von Bereichen kann nun die endgültige Form der Deklarationen bestimmt werden. Eine Deklaration ist eine Folge von Variablen-Bindungen an (zu endlichen Mengen auswertbaren) Bereiche.

2.35 Definition: Endgültige Form der Deklarationen

Die Syntax von Deklarationen ist gegeben durch eine Menge DECL und Funktionen free, decl : DECL $\rightarrow \mathcal{F}$(VAR).

i. Wenn v $\in$ VAR$_s$, r $\in$ RANGE$_s$, und v $\notin$ free(r), dann (v : r) $\in$ DECL, free(v : r) := free(r), und decl(v : r) := $\{$v$\}$.

ii. Wenn v $\in$ VAR$_s$, r $\in$ RANGE$_s$, d $\in$ DECL mit
v $\notin$ free(r) $\cup$ free(d) $\cup$ decl(d), dann (v : r; d) $\in$ DECL,
free(v : r; d) := (free(r) - decl(d)) $\cup$ free(d), und
decl(v : r; d) := $\{$v$\}$ $\cup$ decl(d).

Die Semantik von Deklarationen ist eine Relation
μ[DECL] $\subseteq$ DECL $\times$ ASSIGN.

i. (v : r,β) $\in$ μ[DECL], gdw. β(v) $\in$ μ[RANGE](r,β).

ii. (v : r; d,β) $\in$ μ[DECL], gdw. β(v) $\in$ μ[RANGE](r,β) und (d,β) $\in$ μ[DECL].

Es gilt immer free(d) $\cap$ decl(d) = $\emptyset$.

2.36 Beispiel: Deklarationen

1. (p:PERSON)
2. (l:LAND;(st:STADT))
3. (st:STADT;(l:LAND))
4. (a:LTS(Adr(p)))
5. (p:LTS(Minister(l)))
6. (str : BTS -[StName(st) | (st:STADT)]- UNION
   BTS -[city(a) | (a:LTS(Adr(p)));(p:PERSON)]-)

7. -[str | (str : BTS -[StName(st) | (st:STADT)]- UNION
   BTS -[city(a) | (a:LTS(Adr(p)))]- ;
   (p:PERSON))]-
8. (f:BTS-[Regforma(l) | (l:LAND)]-)
9. (a:LTS(Adr(p));(p:LTS(Minister(l))))
10. (st:BTS-[st | (st:STADT) $\wedge$ buergermeister(p,st)]-;(p:LTS(Minister(l))))
11. (p:LTS(Minister(l));(rg:area(l);(l:LAND)))
12. (p:LTS(Ministers(l));(rg:area(l)))

2.37 Übersicht über die Features des Kalküls

Die reichhaltigen Möglichkeiten zur Bildung von Deklarationen, Termen, Formeln und Bereichen sind in Abblidung 17 zusammengefaßt.

2.38 Eigenschaften des erweiterten Entity-Relationship-Kalküls

Ohne in Details zu gehen, werden noch nun einige wichtige Eigenschaften des erweiterten Entity-Relationship-Kalküls gennant.

- Der Kalkül ist sicher, d. h. die Auswertung eines jeden Terms liefert stets ein endliches Ergebnis [Gog94].
- Der Kalkül ist relational vollständig, d. h. jeder Term der relationalen Algebra kann durch einen äquivalenten Kalkül-Term dargestellt werden [Gog94]. Der Kalkül ist sogar genauso mächtig wie vorgeschlagene Erweiterungen der relationalen Algebra für sogenannte Non-First-Normal-Form-Datenmodelle (NF^2-Datenmodelle) [Her94].
- Der Kalkül ist Dank des Potenzmengen-Operators mächtiger als die relationale Algebra und die klassischen relationalen Kalküle (Bereichs- und Tupelkalkül), da die transitive Hülle einer zweistelligen Relation berechnet werden kann [Gog94].

2.3 Anwendungen des Kalküls

Integritätsbedingungen dienen im Datenbankentwurf zur Festlegung von erwünschten Datenbankzuständen. Sie sind geschlossene Formeln des erweiterten Entity-Relationship-Kalküls.

2.39 Definition: Integritätsbedingungen

Die Syntax von Integritätsbedingungen ist gegeben durch eine Menge CONSTRAINT $\in$ |SET|. Wenn f $\in$ FORM mit free(f)=$\emptyset$, dann f $\in$ CONSTRAINT.

Die Semantik von Integritätsbedingungen ist gegeben durch eine Relation μ[CONSTRAINT] $\subseteq$ CONSTRAINT mit f $\in$ μ[CONSTRAINT] gdw. (f,ϵ) $\in$ μ[FORM].

Syntaktische Kategorie	Feature	Beispiel
Terme	Variablen	l(:LAND), la(:liegt-an)
	Komponenten einer Relationshiptyp Variablen	la.HStadt, la.HAFENSTADT, la.1
	Typumwandlungen	GEWAESSER(f), FLUSS(g)
	Datenoperationen	pdist(center(StGeo(st_1)),center(StGeo(st_2)))
	Attribute	Alter(p)
	Komponenten	Praesident(l)
	Operationen der Sortenausdrücke	CNT(Minister(l))
	Multimengen-Terme	-[Alter(p) | (p:PERSON) ∧ Alter(p)>50]-
Formeln	Datenprädikate	ppcut(MGeo(m_1),MGeo(m_2))
	Relationshiptypen	liegt-in(st,l)
	Prädikate der Sortenausdrücke	p IN Minister(l)
	Gleichheit	Alter(p)=42
	Test auf Definiertheit	UNDEF(Adr(p))
	Test für Typkonstruktionen	f IS g
	Negation	¬ UNDEF(Alter(p))
	Logisches oder	Alter(p)≤18 ∨ Alter(p)≥65
	Logisches und	Alter(p)=42 ∧ city(SEL(Adr(p),1))='Rome'
	Implikation	p IN Minister(l) ⇒ Alter(p)≥18
	Äquivalenz	muendet-in(f,GEWAESSER(m)) ⇔ lpcut(FGeo(f),MGeo(m))
	Existenzquantor	∃ (p:PERSON) Alter(p)>99
	Allquantor	∀ (fd:fliesst-durch) fd.Laenge>100
Bereiche	Entity- oder Relationship-Typen	LAND, fliesst-durch
	Mengenwertige Terme	BTS -[p | (p:PERSON) ∧ Alter(p)>50]-
	Potenzmengen	set(LAND)
Deklarationen	Bereiche	(p : LTS(Minister(l)) UNION BTS-[p | (p:PERSON) ∧ Praesident(l)=p]-)
	Sequenzen	(p : LTS(Minister(l)); (l;LAND))

Abbildung 17: Übersicht über die Features des Kalküls

2.40 Beispiel: Integritätsbedingungen

1. „Hauptstädte müssen in den Ländern liegen, mit denen sie auch über die Beziehung liegt-in verbunden sind“
   ∀ (l:LAND) liegt-in(Hauptstadt(l),l)
   oder
   CNT -[l | (l:LAND)]-
   =
   CNT -[l | (l:LAND) ∧ liegt-in(Hauptstadt(l),l)]-
2. „Minister sind mindestens 18 Jahre alt“
   ∀ (p:PERSON) (∃ (l:LAND) p IN Minister(l)) ⇒ Alter(p)>18
   versus
   ∀ (p:PERSON) p IN Minister(∗) ⇒ Alter(p)>18
   versus
   p IN Minister(∗) ⇒ Alter(p)>18
3. „Der Relationshiptyp liegt-in zwischen den Entitytypen STADT und LAND stimmt mit den entsprechenden geometrischen Attributen überein“

   ∀ (st:STADT) ∀ (l:LAND)
     liegt-in(st,l) ⇔ ∃ (rg:Regionen(l)) cpcut(StGeo(st),geo(rg))
4. „Eine Person ist Präsident von höchstens einem LAND“
   ∀ (l_1, l_2:LAND) Praesident(l_1)=Praesident(l_2) ⇒ l_1=l_2
   oder
   ∀ (p:PERSON) CNT -[l | (l:LAND) ∧ Praesident(l)=p]- ≤ 1
5. „Das Durchschnittsalter der Minister eines Landes ist kleiner gleich 65“
   ∀ (l:LAND) AVG -[Alter(p) | (p:LTS(Minister(l)))]- ≤ 65
6. „Regionen verschiedener Länder überschneiden sich nicht“

   ∀ (l_1,l_2:LAND) $l_1 \neq l_2$ ⇒
     ∀ (rg_1:Regionen(l_1)) (rg_2:Regionen(l_2)) ¬ ppcut(geo(rg_1),geo(rg_2))
7. „Ein Fluß kann nicht in sich selbst münden“
   Fluss-muendet-in-Fluss(f_1,f_2) :⇔ muendet-in(f_1,GEWAESSER(f_2)).
   ∀ (f:FLUSS) ¬ Fluss-muendet-in-Fluss$^+$(f,f)
   Fluss-muendet-in-Fluss$^+$ steht für die transitive Hülle von Fluss-muendet-in-Fluss.
8. „Ein Minister muß mindestens eine Adresse in einer Stadt seines Landes besitzen“

   ∀ (p:PERSON) ∀ (l:LAND)
     (p IN Minister(l) ⇒
       ∃ (a:LTS(Adr(p))) ∃ (st:STADT) StName(st) = city(a))

9. „Die Summe der Stadteinwohnerzahlen ist kleiner gleich der Landeinwohnerzahl“

   $\forall$ (l:LAND)
   SUM -[StEinw(st) | (st:STADT) $\wedge$ liegt-in(st,l)]- $\leq$ LEinw(l)

10. „Städte eines gegebenen Landes haben ‘fast’ die gleiche Einwohnerzahl, d. h. in einem gegebenen Land ist die maximale Stadteinwohnerzahl höchstens doppelt so groß wie die minimale Stadteinwohnerzahl“

    $\forall$ (l:LAND) MIN -[StEinw(st) | (st:STADT) $\wedge$ liegt-in(st,l)]- $*$ 2
    $\geq$
    MAX -[StEinw(st) | (st:STADT) $\wedge$ liegt-in(st,l)]-

2.41 Graphische Darstellung von Integritätsbedingungen

Im STADT-LAND-FLUSS-Beispiel in Abbildung 3 sind einige Integritätsbedingungen graphisch dargestellt.

Kardinalitäten: Die Kardinalitätsangabe für den Relationshiptyp liegt-in und den Entitytyp STADT entspricht folgender Formel des Kalküls.
$\forall$ (st:STADT) 1 $\leq$ CNT -[li | (li:liegt-in) $\wedge$ li.Stadt=st]- $\wedge$
CNT -[li | (li:liegt-in) $\wedge$ li.Stadt=st]- $\leq$ 1
Im allgemeinen können Kardinalitäten durch zwei natürliche Zahlen $(low, high)$ mit $low \leq high$ spezifiziert werden. Die Angabe $high$ darf dabei auch als $*$ angegeben werden und steht dann für eine beliebige Zahl.

Schlüsselattribute: Die Attributemenge {LName, Regform} ist ein Schlüssel für den Entitytyp LAND.
$\forall$ (l_1, l_2:LAND)
(LName(l_1)=LName(l_2) $\wedge$ Regform(l_1)=Regform(l_2)) $\Rightarrow$ l_1=l_2
Schlüsselattribute werden mit dicken Punkten gekennzeichnet.

Funktionale Beziehungen: Der Relationshiptyp liegt-in ist funktional von STADT nach LAND.
$\forall$ (st:STADT) $\forall$ (l_1,l_2:LAND) liegt-in(st,l_1) $\wedge$ liegt-in(st,l_2) $\Rightarrow$ l_1=l_2
Funktionale Beziehungen werden mit einem Pfeil gekennzeichnet.

Optionale Attribute: Grundsätzlich sind alle Attribute und Komponenten im erweiterten Entity-Relationship-Modell optional. In der graphischen Darstellung wählt man jedoch den häufigeren Fall des nicht-optionalen Attributs bzw. der nicht-optionalen Komponente als Voreinstellung. D. h. für alle Attribute und Komponenten, die durch eine ununterbrochene Gerade mit ihrem Entitytyp verbunden sind, muß eine Integritätsbedingung eingeführt werden, die die Definiertheit fordert. Für das Attribute StName wäre das beispielsweise:

$\forall$ (st:STADT) DEF(StName(st))

Um nun optionale von nicht-optionalen Attributen und Komponenten graphisch unterscheiden zu können, kennzeichnet man nicht-optionale mit einem Kreis, wie z. B. beim Attribut Praesident des Entitytyps LAND.

=-Typkonstruktionen: Der allgemeine Fall der Typkonstruktion ist der Fall einer $\subseteq$-Typkonstruktion. Die =-Forderung der Typkonstruktion 'sind' entspricht folgender Integritätsbedingung.

$\forall$ (m:MEER) DEF(GEWAESSER(m)) $\wedge$
$\forall$ (s:SEE) DEF(GEWAESSER(s)) $\wedge$
$\forall$ (f:FLUSS) DEF(GEWAESSER(f))

Anfragen werden nun als spezielle daten-wertige Terme eingeführt. Die Einschränkung auf daten-wertige Terme ist darin begründet, daß Anfragen Auskunft über einen Datenzustand und daher ein ausdruckbares Ergebnis liefern sollen. Anfragen werden als geschlossene Terme, d. h. Terme ohne freie Variablen definiert.

2.42 Definition: Queries

Die Syntax von Queries ist gegeben durch eine Menge QUERY $\in$ |SET|. Wenn t $\in$ TERM_s mit s $\in$ EXPR(DATA) und free(t)=$\emptyset$, dann t $\in$ QUERY.

Die Semantik von Queries ist gegeben durch eine Funktion μ[QUERY] : QUERY $\to$ μ[EXPR(DATA)] so daß μ[QUERY](t) := $\hat{\mu}$[TERM](t,ϵ).

2.43 Beispiel: Queries

1. „Namen von italienischen Flüsse"

   -[FName(f) | (f:FLUSS) $\wedge$
   $\exists$ (l:LAND) (LName(l)='Italien' $\wedge$ fliesst-durch(l,f))]-

   oder

   -[FName(f) | (f:FLUSS) $\wedge$
   $\exists$ (fd:fliesst-durch) (LName(fd.1)='Italien' $\wedge$ fd.2=f)]-

   oder

   -[FName(fd.FLUSS) | (fd:fliesst-durch) $\wedge$ LName(fd.LAND)='Italien']-

2. „Namen von Flüssen, die durch alle demokatrischen Länder fliessen"

   -[FName(f) | (f:FLUSS) $\wedge$
   $\forall$ (l:LAND) (Regform(l)='Demokratie' $\Rightarrow$ fliesst-durch(l,f))]-

3. „Namen von Flüssen, die durch Frankreich fliessen zusammen mit ihrer Durchflußlänge"

-[FName(f), Laenge(fd) | (f:FLUSS; (fd:fliesst-durch)) $\wedge$
$\exists$ (l:LAND) (LName(l)=‘Frankreich’ $\wedge$ fd.FLUSS=f $\wedge$ fd.LAND=l)]-

oder

-[FName(fd.FLUSS), Laenge(fd) | (fd:fliesst-durch) $\wedge$
LName(fd.LAND)=‘Frankreich“]-

4. „Namen von Ländern, die nur Städte mit weniger als 100,000 Einwohner haben“

   -[LName(l) | (l:LAND) $\wedge$
   $\neg$ $\exists$ (st:STADT) (StEinw(st)$\geq$100,000 $\wedge$ liegt-in(st,l))]-

   oder

   -[LName(l) | (l:LAND) $\wedge$
   $\forall$ (st:STADT) (StEinw(st)$\geq$100,000 $\Rightarrow$ $\neg$ liegt-in(st,l))]-

   oder

   -[LName(l) | (l:LAND) $\wedge$
   $\forall$ (st:STADT) (liegt-in(st,l) $\Rightarrow$ StEinw(st) $<$ 100,000)]-

5. „Namen und Gebiet von Städten am Rhein mit einer Fläche grösser als 5 km^2“

   -[StName(st), 3.14$*$radius(StGeo(st))$\uparrow$2 | (st:STADT) $\wedge$
   $\exists$ (f:FLUSS) (FName(f)=‘Rhein’ $\wedge$ liegt-an(HAFENSTADT(st),f) $\wedge$
   3.14$*$radius(StGeo(st))$\uparrow$2$>$ 5,000,000)]-

6. „Namen der britischen Regionen“
   -[name(rg) | (rg:Regionen(l));(l:LAND) $\wedge$ LName(l)=‘Groß-Britanien’]-

7. „Addressen von Ministern demokratischer Länder“
   (a)

   -[LTB(Adr(p)) | (p:LTS(Minister(l)));(l:LAND) $\wedge$
   Regform(l)=’Demokratie“]-

   (b)

   -[a | (a:LTS(Adr(p)));(p:LTS(Minister(l)));(l:LAND) $\wedge$
   Regform(l)=’Demokratie’]-

   (c)

   -[-[a | (a:LTS(Adr(p)))]- | (p:LTS(Minister(l)));(l:LAND) $\wedge$
   Regform(l)=’Demokratie’]-

(d)

-[-[Adr(p) | (p:LTS(Minister(l)))]- | (l:LAND) ∧
    Regform(l)='Demokratie']-

(e)

-[Adr(p) | (p:LTS(Minister(l)));(l:LAND) ∧
    Regform(l)='Demokratie']-

8. „Namen von Ministern, die nur Adressen in ihrem Land haben"

   -[PName(p) | (p:LTS(Minister(l));(l:LAND)) ∧
       ∀ (a:LTS(Adr(p)))
         ∃ (st:STADT) (liegt-in(st,l) ∧ StName(st)=city(a))]-

9. „Namen von Flüssen, die in die Nordsee fliessen"

   -[FName(f) | (f:FLUSS) ∧
     ∃ (m:MEER) (MName(m)='Nordsee' ∧
         ∃ (g:GEWAESSER) (m IS g ∧ muendet-in(f,g)))]-

   oder

   -[FName(f) | (f:FLUSS) ∧
     ∃ (m:MEER) (MName(m)='Nordsee' ∧ muendet-in(f,WATERS(m)))]-

10. „Name und Alter des Präsidenten der UDSSR"

   -[PName(p), Alter(p) | (p:PERSON) ∧
       ∃ (l:LAND) (LName(l)='UDSSR' ∧ p=Praesident(l))]-

   oder

   -[PName(Praesident(l)), Alter(Praesident(l)) | (l:LAND) ∧
       LName(l)='USSR']-

11. „Namen von Personen mit unbekannter Adresse"
   -[PName(p) | (p:PERSON) ∧ UNDEF(Adr(p))]-

12. „Anzahl der gespeicherten Länder"
   CNT -[c | (l:LAND)]-

13. „Sume der Einwohnerzahlen der gespeicherten Länder"
   SUM -[LEinw(l) | (l:LAND)]-

14. „Durchschnittliche Landeinwohnerzahl"
   AVG -[LEinw(l) | (l:LAND)]-
   oder
   SUM -[LEinw(l) | (l:LAND)]- / CNT -[LEinw(l) | (l:LAND)]-
   oder

$APL_{+,bag(int)}$ -[LEinw(l) | (l:LAND)]- /
$CNT_{bag(int)}$ -[LEinw(l) | (l:LAND)]-

15. „Landnamen und Durchschnittsalter der Minister"

    -[LName(l),
      AVG -[Alter(p) | (p:PERSON) ∧ p IN Minister(l)]- |
      (l:LAND)]-

    versus

    -[LName(l),
      AVG -[Alter(p) | (i:POS(Minister(l),p));(p:LTS(Minister(l)))]- |
      (l:LAND)]-

16. „Landnamen und durchschnittliche Stadteinwohnerzahl"

    -[LName(l), AVG -[StEinw(st) | (st:STADT) ∧ liegt-in(st,l)]- |
      (l:LAND)]-

    versus

    -[LName(l),
      AVG -[StEinw(st) | liegt-in(st,l)]- | (l:LAND;(st:STADT))]-

17. „Durchschnitte der maximalen Landeinwohnerzahlen"

    AVG -[MAX-[LEinw(l) | (l:LAND) ∧ Regform(l)=f]- |
      (f:BTS-[Regform(l) | (l:LAND)]-)]-

    versus

    AVG -[MAX -[LEinw(l_1) |
      (l_1:LAND) ∧
      Regform(l_1)=Regform(l_2)]- | (l_2:LAND)]-

18. „Namen und Landeinwohnerzahlen der Länder deren Landeinwohnerzahl gleich der Summe der Stadteinwohnerzahlen ist"

    -[LName(l), LEinw(l) | (l:LAND) ∧
      LEinw(l) = SUM-[StEinw(st) | (st:STADT) ∧ liegt-in(st,l)]-]-

19. „Attribute von Ministern"
    -[p | (p:LTS(Minister(l)));(l:LAND)]-
    versus
    -[PName(p), Adr(p), Alter(p) | (p:LTS(Minister(l)));(l:LAND)]-

20. „Landnamen zusammen mit den Stadtnamen und der Differenz der Landeinwohnerzahl und der Summe der Stadteinwohnerzahl"

-[LName(l),
  -[StName(st) | (st:STADT) ∧ liegt-in(st,l)]-,
  LEinw(l)-(SUM-[StEinw(st) | (st:STADT) ∧ liegt-in(st,l)]-) |
  (l:LAND)]-

21. „Ministernamen der USA zusammen mit der Positionsnummer in der Ministerliste“

    -[PName(SEL(Minister(l),i)), i |
      (i:IND(Minister(l)));(l:LAND) ∧ LName(l)=‘USA’]-

    oder

    -[PName(p), i |
      (i:POS(minister(l),p));(p:LTS(minister(l)));(l:LAND) ∧
      LName(l)=‘USA’]-

22. „Minister der UDSSR mit wenigstens zwei Ministerämtern“

    -[PName(p) | (p:LTS(Minister(l)));(l:LAND) ∧
      CNT(POS(Minister(l),p))>2 ∧ LName(l)=‘UDSSR’]-

    oder

    -[PName(p) | (p:LTS(Minister(l)));(l:LAND) ∧
      OCC(LTB(Minister(l)),p)>2 ∧ LName(l)=‘UDSSR’]-

3 Weitere semantische Datenmodelle

In diesem Kapitel werden drei aus der Literatur bekannte semantische Datenmodelle anhand eines Beispiels kurz vorgestellt. Anschließend wird jeweils das Beispiel auch im erweiterten Entity-Relationship-Modell modelliert, um so einen Vergleich zu ermöglichen.

3.1 IFO

Das semantische Datenmodell IFO [AH87] war eines der ersten Datenmodelle, zu dem auch eine formale Semantik gegeben wurde. IFO wird anhand einer kleinen Fahrzeug-Diskurswelt in Abbildung 18 erläutert.

Knoten in IFO entsprechen Typen und Kanten Beziehungen zwischen Typen. In erster Annäherung zum ER-Modell könnte man also IFO-Knoten mit Entitytypen und IFO-Kanten mit Relationshiptypen vergleichen. IFO unterscheidet die spezifizierten Typen danach, ob sie atomar oder konstruiert sind.

Atomare Typen: Die atomaren Typen werden weiter in druckbare, abstrakte und freie Typen aufgeteilt.

- **Druckbare Typen:** Druckbare Typen werden in der graphischen Darstellung mit Quadraten dargestellt. Es handelt sind hierbei um vordefinierte Typen zur Ein- und Ausgabe.
- **Abstrakte Typen:** Abstrakte Typen sind Typen deren Instanzen keine zugrunde liegende Struktur besitzen. Sie werden mit Rauten graphisch notiert.
- **Freie Typen:** Freie Typen treten immer im Zusammenhang mit IST-Beziehungen, also im Zusammenhang mit Rollen- und Generalisierungshierarchien, auf. Ihre graphische Repräsentation sind leere Kreise.

Konstruierte Typen: Für konstruierte Typen stehen zwei Möglichkeiten zur Gestaltung zur Verfügung.

- **Mengen-Typen:** Mengen-Typen werden graphisch mit einem von einem Kreis umschlossenen Stern dargestellt. Die Instanzen von diesen Typen sind Mengen von Instanzen des zugrundeliegenden Typs, der über einen Strich mit dem Mengen-Typ verbunden ist.
- **Produkt-Typen:** Ganz analog zu Mengen-Typen sind Produkt-Typen zu sehen. Ihre Instanzen sind Elemente des kartesischen Produkts der zugrundeliegenden Typen.

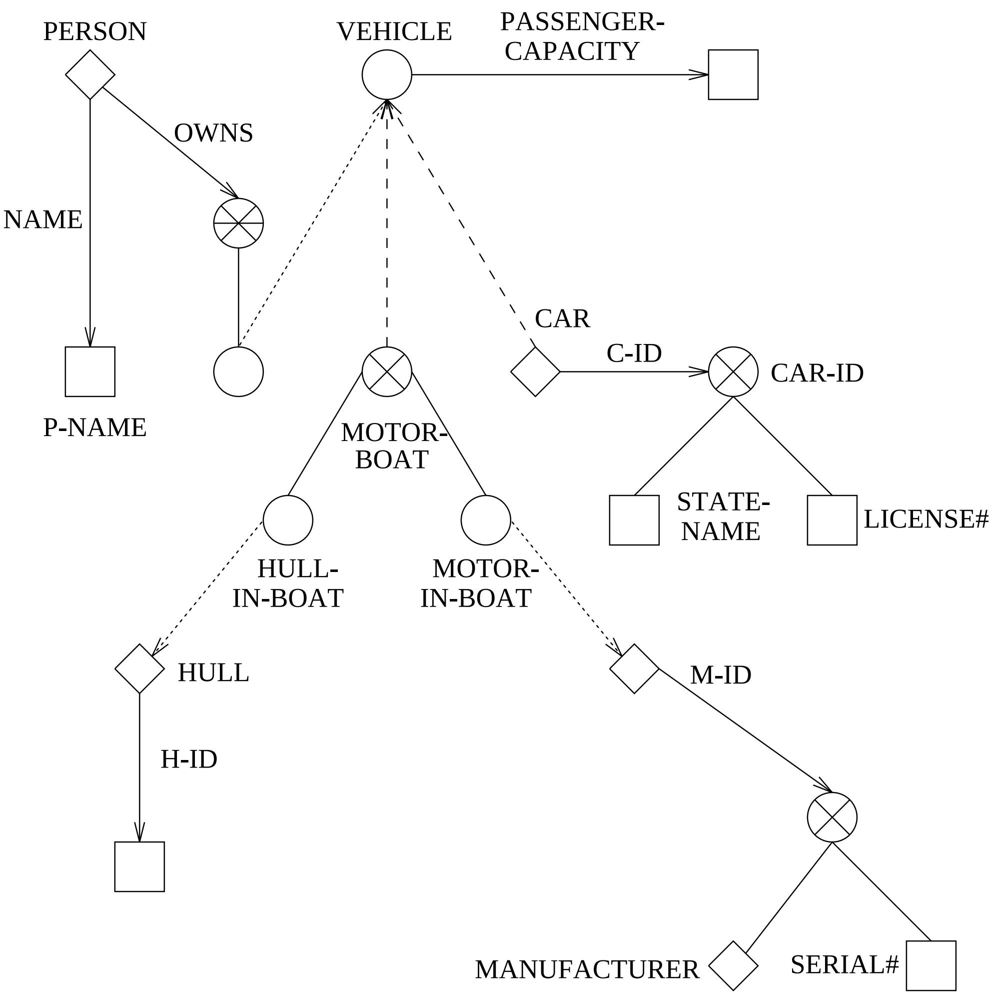


Abbildung 18: Beispiel-Modellierung in IFO

IFO ist eines der wenigen Datenmodelle, das zwischen Rollen- und Generalisierungshierarchien unterscheidet, die im allgemeinen Abschnitt über semantische Datenmodelle vorgestellt wurden.

Rollenhierarchien: Diese Hierarchien werden graphisch mit einem gepunkteten Pfeil markiert.

Generalisierungshierarchien: Die graphische Notation für diese Hierarchien sind gestrichelte Pfeile.

Das obige Beispiel-Schema wird nun wie in Abbildung 19 angegeben im erweiterten Entity-Relationship-Modell modelliert.

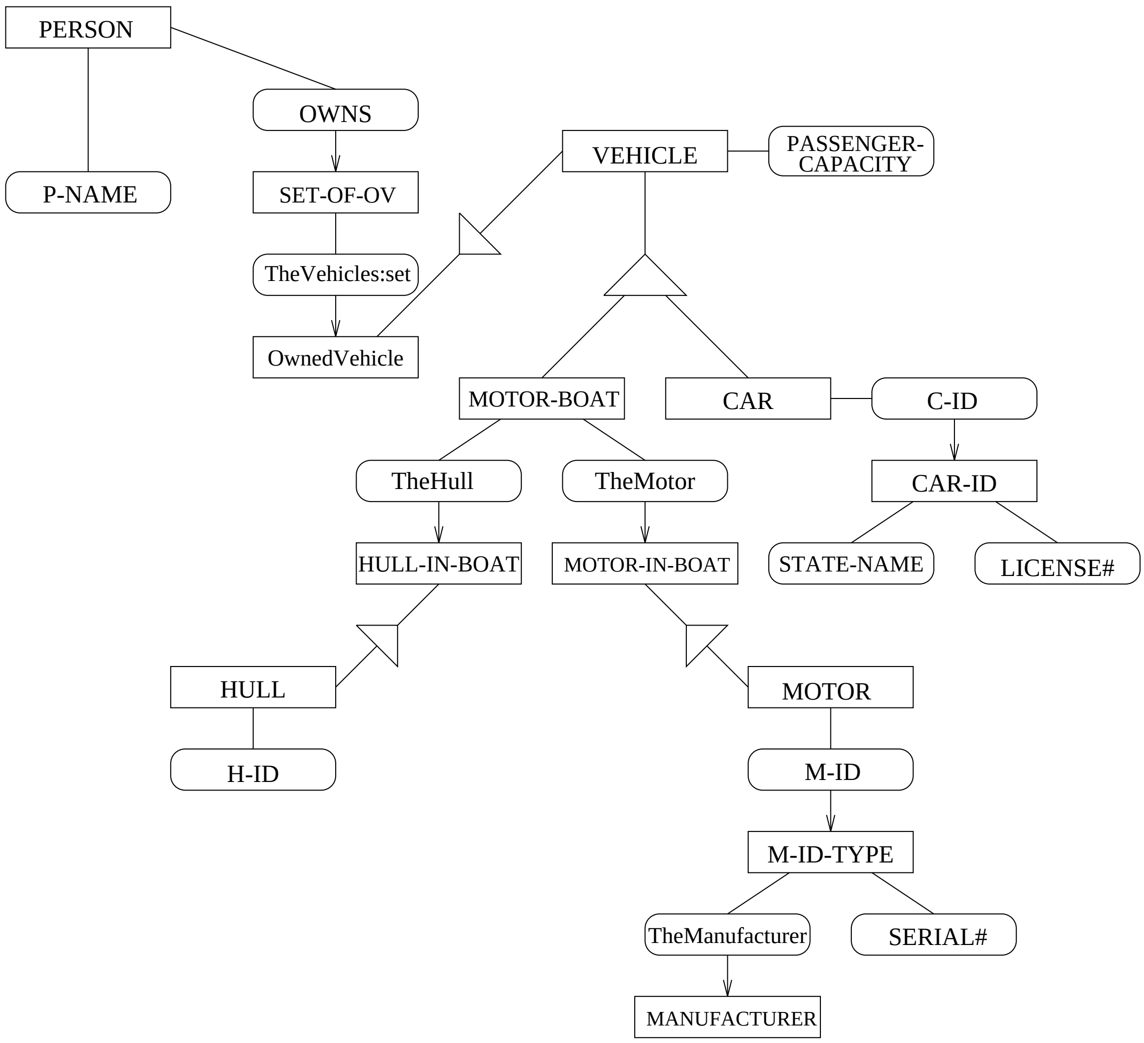


Abbildung 19: IFO-Modellierung im EER-Modell

Im wesentlichen findet sich die durch das IFO-Schema gegebene Strukur auch im ER-Schema wieder. Für druckbare Typen werden Attribute verwendet. Abstrakte und freie Typen entsprechen Entitytypen. Für konstruierte Typen werden explizit neue Entitytypen eingeführt. Rollen- und Generalisierungshierarchien werden mittels Typkonstruktionen repräsentiert, wobei die Richtung des Dreiecks bei Rollenhierarchien genau entgegengesetzt zur IFO-Pfeilrichtung ist. Man beachte aber, daß die Pfeilrichtungen der Typkonstruktionen insofern die natürlicheren sind, da sie auch der Konstruktion der Entities entsprechen: Basisentities sind Rümpfe (HULL) und Motoren aus denen Motorboote entstehen können; Motorboote und Autos werden zu Fahrzeugen generalisiert, die wiederum zu Fahrzeugen spezialisiert werden, die im Besitz von Personen sind.

3.2 ERC

Von den hier betrachten semantischen Datenmodellen steht das ERC-Modell (Entity-Relationship-Complex) [PRYS89] dem erweiterten Entity-Relationship-Modell naturgemäß am nächsten. In Abbildung 20 wird eine Diskurswelt mit Autos und Versicherungen betrachtet.

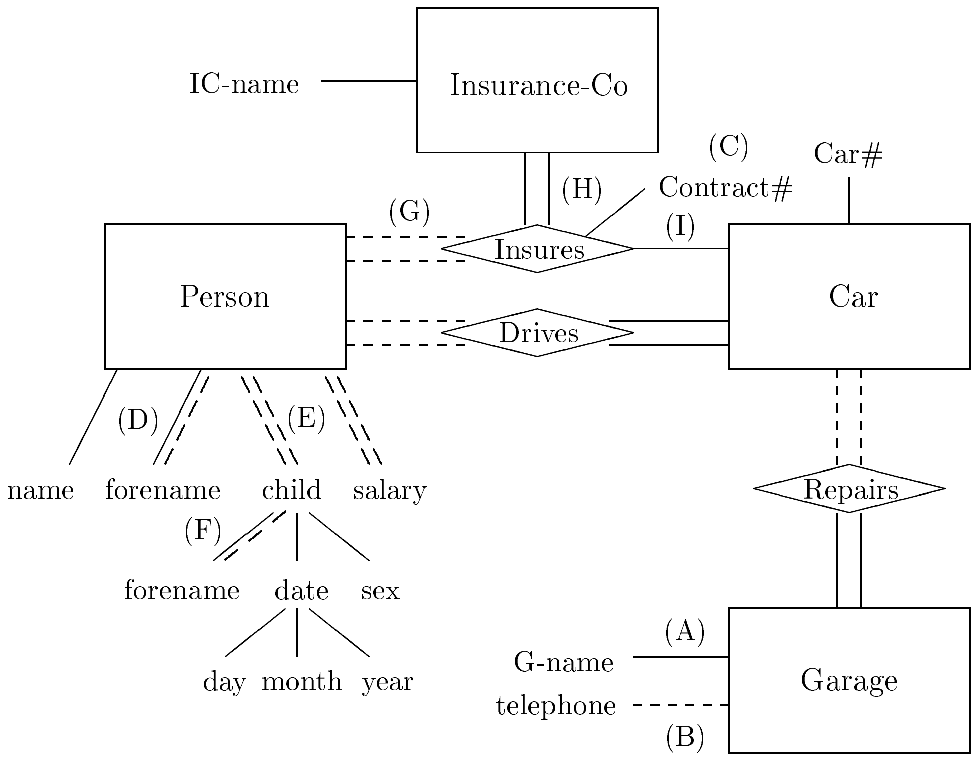


Abbildung 20: Beispiel-Modellierung in ERC

Im wesentlichen kann man in ERC strukturierte Datentypen (wie z. B. den Typ Child) und Kardinalitätsangaben graphisch repräsentieren. Die unterschiedlichen Linien haben in ERC folgende Bedeutung.

- Eine nicht-durchbrochene, einzelne Linie hat Kardinalität 1 [in EER-Notation (1,1)].
- Eine gestichelte, einzelne Linie hat Kardinalität 0 oder 1 [(0,1)].
- Eine nicht-durchbrochene, doppelte Linie hat Kardinalität größer gleich 2 [(2,∗)].
- Eine gestichelte zusammen mit einer nicht-durchbrochenen Linie hat Kardinalität größer gleich 1 [(1,∗)].

- Eine gestichelte, doppelte Linie hat Kardinalität größer gleich 0 [(0,$*$)].

Für die ER-Modellierung wird zunächst eine entsprechende Datentypdefinition gegeben.

```
SORTS       dateSort, childSort
DEFINITIONS dateSort = record(1..31,1..12,0..3000)
            childSort = record(list(string),dateSort,{female,male})
OPERATIONS  day : dateSort -> 1..31
            month : dateSort -> 1..12
            year : dateSort -> 0..3000
            forename : childSort -> list(string)
            date : childSort -> dateSort
            sex : childSort -> {female,male}
EQUATIONS   day(d) = d.1
            month(d) = d.2
            year(d) = d.3
            forename(c) = c.1
            date(c) = c.2
            sex(c) = c.3
```

Auf der Basis dieser Datentypen läßt sich nun das entsprechende Entity-Relationship-Diagramm in Abbildung 21 spezifizieren.

Die im Diagramm fehlenden weiteren Angaben zu Attributen sind wie folgt zu setzen: IC-name : Insurance-Co $\rightarrow$ string, name : Person $\rightarrow$ string, forename : Person $\rightarrow$ list(string), child : Person $\rightarrow$ list(childSort), salary : Person $\rightarrow$ list(integer), telephone : Garage $\rightarrow$ string, G-name : Garage $\rightarrow$ string, Contract# : Insures $\rightarrow$ string und Car# : Car $\rightarrow$ string. Die Besonderheiten von ERC bezüglich der Kardinalitätsangaben, die im ERC-Diagram mit (A) bis (I) gekennzeichnet worden waren, werden durch Integritätsbedingungen wiedergegeben.

(A) $\forall$ (g:Garage) DEF(G-name(g))
(B) Voreingestellt für Attribute ist die Erlaubis undefiniert zu sein.
(C) $\forall$ (ins:Insures) DEF(Contract#(ins))
(D) $\forall$ (p:Person) CNT(forename(p)) ≥ 1
(E) Voreingestellt für Listen ist die Erlaubis die Länge 0 zu haben.
(F) $\forall$ (p:Person) $\forall$ (i:IND(child(p))) CNT(forename(SEL(child(p),i))) ≥ 1
(G) Voreingestellt für Beziehungen ist die Erlaubis mehrmals an der Beziehung teilzunehmen.
(H) $\forall$ (ic:Insurance-Co)
(CNT -[ins | (ins:Insures) $\wedge$ ins.Insurance-Co=ic]-) ≥ 2
(I) $\forall$ (c:Car) (CNT -[ins | (ins:Insures) $\wedge$ ins.Car=c]-) $= 1$

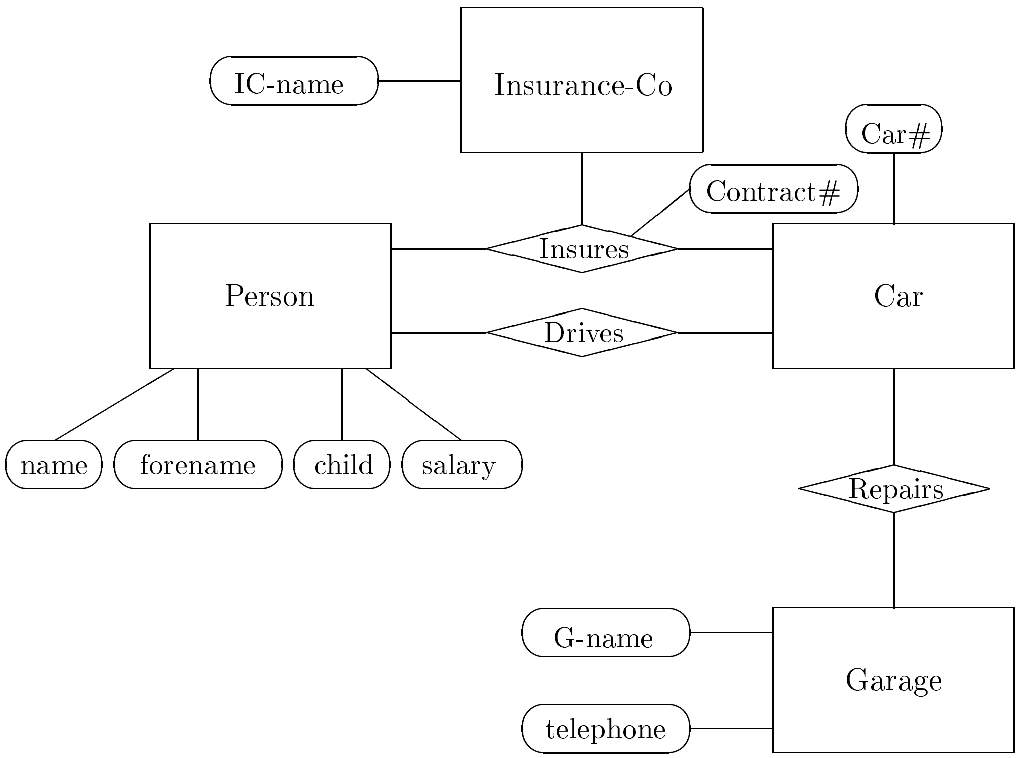


Abbildung 21: ERC-Modellierung im EER-Modell

3.3 OMT

Die Object Modelling Technique (OMT) [RBP+91] ist eine neuere, populäre Methode zur Software-Entwicklung. Unter anderen Aspekten stellt sie auch Sprachmittel zur Beschreibung von Struktur zur Verfügung, die hier etwas näher betrachtet werden sollen. In Abbildung 22 wird eine Diskurswelt mit Firmen and Angestellten vorgestellt.

Der Sachverhalt läßt sich wie in Abbildung 23 im erweiterten Entity-Relationship-Modell wiedergeben.

Kardinalitäten werden in OMT an den Enden der einfach durch Striche dargestellten Beziehungstypen notiert: Kein spezielles Symbol steht für eine Kardinalität 1, ein leerer Kreis für Kardinalität 0 oder 1 und ein dicker Punkt für eine Kardinalität größer oder gleich 0.

(A) $\forall$ (p:Person) $0 \leq$ (CNT -[m | (m:Manages) $\wedge$ m.worker=p]-) ≤ 1
(B) $\forall$ (p:Person) (CNT -[wf | (wf:Works-for) $\wedge$ wf.Person=p]-) = 1

Diese speziellen Forderungen werden (wie oben skizziert) durch Integritätsbedingungen repräsentiert. Das zweite OMT-Beispiel in Abbildung 24 stellt eine Informatik-Anwendung dar.

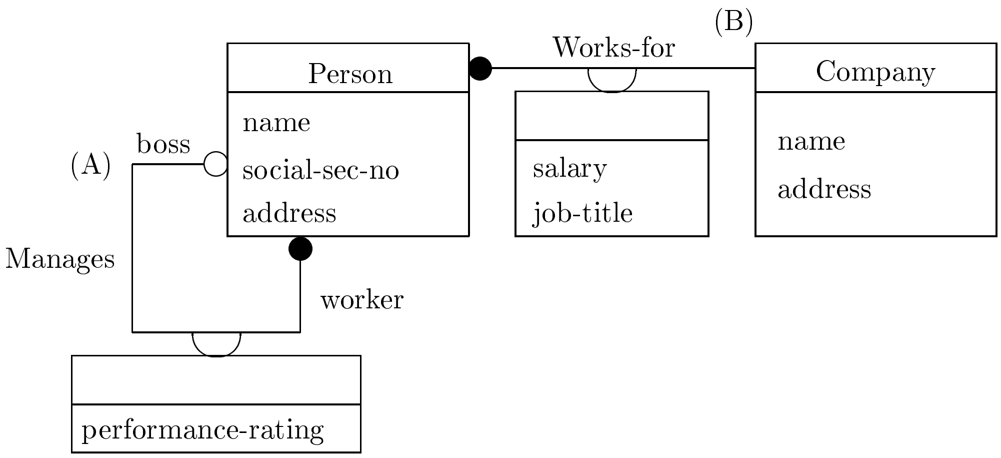


Abbildung 22: Beispiel-Modellierung (a) in OMT

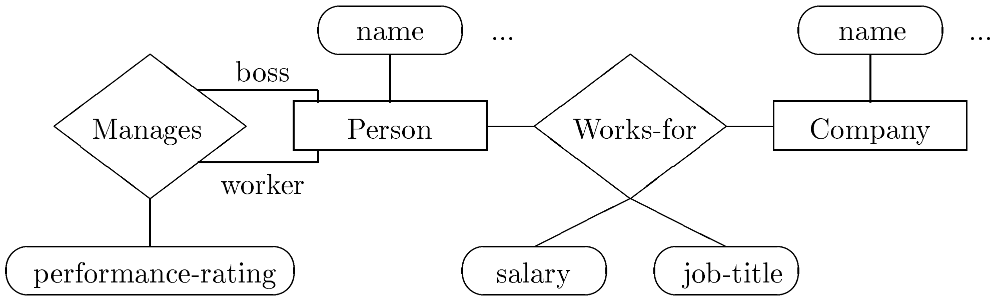


Abbildung 23: OMT-Modellierung (a) im EER-Modell

Komponentenbeziehungen werden in OMT mit kleinen Rauten gekennnzeichnet. IST-Beziehungen sind in OMT (wie erweiterten Entity-Relationship-Modell) durch Dreiecke graphisch dargestellt. Das zugehörige ER-Diagramm ist in Abbildung 25 wiedergegeben.

Die zugrundeliegende Arbeit über OMT [RBP$^+$91] beschreibt OMT nur informell und daher ist z. B. im allgemeinen nicht klar, ob Komponentenbeziehungen exklusiv sind oder nicht. Das Beispiel suggeriert natürlich exklusive Komponentenbeziehungen. Man kann daher davon ausgehen, daß zur Modellierung im ER-Modell noch die folgenden Integritätsbedingungen benötigt werden.

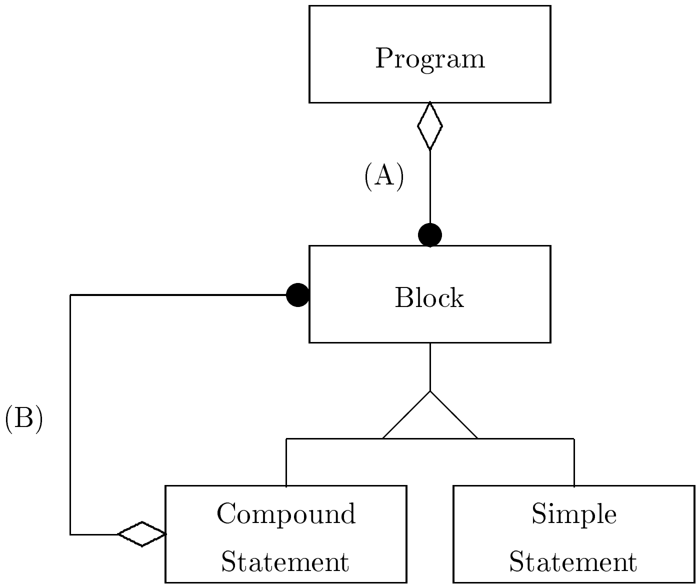


Abbildung 24: Beispiel-Modellierung (b) in OMT

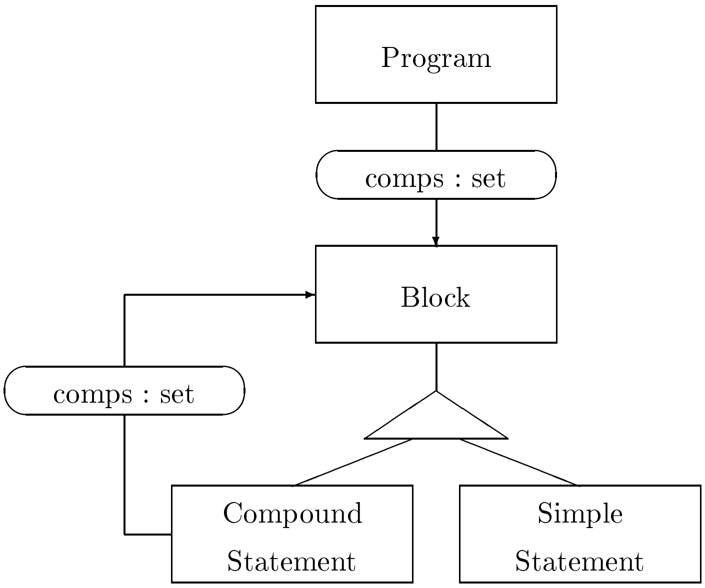


Abbildung 25: OMT-Modellierung (b) im EER-Modell

(A) $\forall$ (b:Block) $\forall$ (p_1,p_2:Program)
(b IN comps(p_1) $\wedge$ b IN comps(p_2)) $\Rightarrow$ p_1=p_2

(B) is-part-of(b_1:Block,b_2:Block) :$\Leftrightarrow$ b_1 IN comps(CompoundStatement(b_2))
$\forall$ (b:Block) $\forall$ (b_1,b_2:Block) (is-part-of(b,b_1) $\wedge$ is-part-of(b,b_2)) $\Rightarrow$ b_1=b_2
$\forall$ (b:Block) $\neg$ is-part-of$^+$(b,b)

Bedingung (A) bezieht sich darauf, daß ein Block nicht gleichzeitig Teil von mehreren Programmen sein kann. Bedingung (B) fordert analoges für zusammengesetzte Anweisungen und Blöcke. Zusätzlich wird in (B) noch gefordert, daß ein Block nicht Teil von sich selbst sein kann. Diese Problemstellung wird in [RBP+91] jedoch nicht angesprochen.

Literatur

[AH87] S. Abiteboul and R. Hull. IFO — A Formal Semantic Database Model. *ACM Trans. on Database Systems*, 12(4):525–565, 1987.

[Cod81] E.F. Codd. Data Models in Database Management. *ACM SIGMOD Record*, 11(2):112–114, 1981.

[Cod90] E.F. Codd. *The Relational Model for Database Management.* Addison-Wesley, Reading, 1990.

[EGH+92] G. Engels, M. Gogolla, U. Hohenstein, K. Hülsmann, P. Löhr-Richter, G. Saake, and H.-D. Ehrich. Conceptual Modelling of Database Applications Using an Extended ER Model. *Data & Knowledge Engineering*, 9(2):157–204, 1992.

[GH91] M. Gogolla and U. Hohenstein. Towards a Semantic View of an Extended Entity-Relationship Model. *ACM Trans. on Database Systems*, 16(3):369–416, 1991.

[Gog94] M. Gogolla. *An Extended Entity-Relationship Model — Fundamentals and Pragmatics.* Springer, Berlin, LNCS 767, 1994.

[Her94] R. Herzig. *Zur Spezifikation von Objektgesellschaften mit TROLL* light. Dissertation, Naturwissenschaftliche Fakultät, Technische Universität Braunschweig, 1994.

[HG88] U. Hohenstein and M. Gogolla. A Calculus for an Extended Entity-Relationship Model Incorporating Arbitrary Data Operations and Aggregate Functions. In C. Batini, editor, *Proc. 7th Int. Conf. on Entity-Relationship Approach*, pages 129–148. ER Institute, Pittsburgh (CA), 1988.

[HK87] R. Hull and R. King. Semantic Database Modelling: Survey, Applications, and Research Issues. *ACM Computing Surveys*, 19(3):201–260, 1987.

[HM81] M. Hammer and D. McLeod. Database Description with SDM: A Semantic Database Model. *ACM Trans. on Database Systems*, 6(3):351–386, 1981.

[HNSE87] U. Hohenstein, L. Neugebauer, G. Saake, and H.-D. Ehrich. Three-Level Specification of Databases Using an Extended Entity-Relationship Model. In R. Wagner, R. Traunmüller, and H.C. Mayr, editors, *Proc. GI-Fachtagung "Informationsermittlung und -analyse für den Entwurf von Informationssystemen"*, pages 58–88, Linz, 1987. Informatik-Fachbericht 143, Springer, Berlin, 1987.

[Hoh93] U. Hohenstein. *Formale Semantik eines erweiterten Entity-Relationship-Modells.* Teubner, Stuttgart, 1993.

[Lip88] U.W. Lipeck. Datenbankentwurf. Vorlesung an der TU Braunschweig Sommer'88, 1988.

[PM88] J. Peckham and F. Maryanski. Semantic Data Models. *ACM Computing Surveys*, 20(3):153–189, 1988.

[PRYS89] C. Parent, H. Rolin, K. Yetongnon, and S. Spaccapietra. An ER Calculus for the Entity-Relationship Complex Model. In F.H. Lochovski, editor, *Proc. 8th Int. Conf. on Entity-Relationship Approach*, pages 75–98. ER Institute, Pittsburgh (CA), 1989.

[PT88] W.D. Potter and R.P. Trueblood. Traditional, Semantic and Hyper-Semantic Approaches to Data Modeling. *IEEE Computer*, 21(6):53–63, 1988.

[RBP$^+$91] J. Rumbaugh, M. Blaha, W. Premerlani, F. Eddy, and W. Lorensen. *Object-Oriented Modeling and Design.* Prentice-Hall, Englewood Cliffs (NJ), 1991.

[STW84] M. Schrefl, A M. Tjoa, and R.R. Wagner. Comparison Criteria for Semantic Data Models. In C.V. Ramamoorthy, editor, *Proc. 1st Int. Conf. on Data Engineering (ICDE)*, pages 120–125. IEEE, Silver Spring (MD), 1984.

[UD86] S. D. Urban and L. Delcambre. An Analysis of the Structural, Dynamic, and Temporal Aspects of Semantic Data Models. In *Proc. Int. Conf. on Data Engineering*, pages 382–387. ACM, New York, 1986.[1]

[1] Die wichtigsten Referenzen sind fett gedruckt: Für Kapitel 1 ist [HK87], für Kapitel 2 [EGH$^+$92, GH91] und für Kapitel 3 [AH87, PRYS89, RBP$^+$91] maßgebend.