Teil 2: Mathematische Logik mit Datenbank-Anwendungen

Literatur:

- Bergmann/Noll: Mathematische Logik mit Informatik-Anwendungen. Springer, 1977.
- Ebbinghaus/Flum/Thomas: Einführung in die mathematische Logik. Spektrum Akademischer Verlag, 1996.
- Tuschik/Wolter: Mathematische Logik, kurzgefasst. Spektrum Akademischer Verlag, 2002.
- Schöning: Logik für Informatiker. BI Verlag, 1992.
- Chang/Lee: Symbolic Logic and Mechanical Theorem Proving. Academic Press, 1973.
- Fitting: First Order Logic and Automated Theorem Proving. Springer, 1995, 2. Auflage.
- Ulf Nilson, Jan Małuszyński: Logic, Programming, and Prolog (2. Auflage). 1995, [http://www.ida.liu.se/~ulfni/lpp]

Lernziele

Nach diesem Kapitel sollten Sie Folgendes können:

- grundlegende Begriffe erklären: Signatur, Interpretation, Variablenbelegung, Term, Formel, Modell, Konsistenz, Implikation.
- Integritätsbedingungen und Anfragen als Formeln aufschreiben
- gebräuchliche Äquivalenzen anwenden, um logische Formeln zu transformieren
- prüfen, ob eine Formel in einer Interpretation erfüllt ist, ggf. passende Variablenbelegungen finden

Inhalt

Einführung, Motivation (1)

Wichtige Ziele mathematischer Logik sind:

- den Begriff einer Aussage über gewisse "Miniwelten" zu formalisieren (logische Formel),
- Begriffe der logischen Implikation und des logischen Beweises präzise zu definieren,
- Algorithmen zu finden, um zu testen, ob eine Aussage von anderen logisch impliziert wird.

  So weit das möglich ist. Es hat sich herausgestellt, dass diese Aufgabe im allgemeinen unentscheidbar ist. Auch der Begriff der Entscheidbarkeit, heute ein Kernbegriff der Informatik, wurde von Logikern entwickelt (damals gab es noch keine Computer und keine Informatik).

Einführung, Motivation (2)

Anwendung mathematischer Logik in Datenbanken I:

- Allgemein ist das Ziel von mathematischer Logik und von Datenbanken
  - ◇ Wissen zu formalisieren,
  - ◇ mit diesem Wissen zu arbeiten.
- Zum Beispiel benötigt man Symbole, um über eine Miniwelt sprechen zu können.
  - ◇ In der Logik sind diese in Signaturen definiert.
  - ◇ In Datenbanken sind sie im DB-Schema definiert.

    Es ist daher keine Überraschung, dass ein Datenbankschema in der Hauptsache eine Signatur definiert.

Einführung, Motivation (3)

Anwendung mathematischer Logik in Datenbanken II:

- Um logische Implikationen zu formalisieren, muss die mathematische Logik mögliche Interpretationen der Symbole untersuchen,
- d.h. mögliche Situationen der Miniwelt, über die Aussagen in logischen Formeln gemacht werden.
- Der DB-Zustand beschreibt auch mögliche Situationen eines gewissen Teils der realen Welt.
- Im Grunde sind logische Interpretation und DB-Zustand dasselbe (aus "modelltheoretischer Sicht").

Einführung, Motivation (4)

Anwendung mathematischer Logik in Datenbanken III:

- SQL-Anfragen sind Formeln der mathematischen Logik sehr ähnlich. Es gibt theoretische Anfragesprachen, die nur eine Variante der Logik sind.
- Die Idee ist, dass
  - ◇ eine Anfrage eine logische Formel mit Platzhaltern ("freien Variablen") ist,
  - ◇ und das Datenbanksystem dann Werte für diese Platzhalter findet, so dass die Formel im gegebenen DB-Zustand erfüllt wird.

Einführung, Motivation (5)

Warum sollte man mathematische Logik lernen I:

- Logische Formeln sind einfacher als SQL, und können leicht formal untersucht werden.
- Wichtige Konzepte von DB-Anfragen können schon in dieser einfachen Umgebung erlernt werden.
- Erfahrung hat gezeigt, dass Studenten oft logische Fehler in SQL-Anfragen machen.

  Vor allem, wenn ihnen Logik nicht speziell erklärt wird. Es wird interessant sein zu sehen, ob sich das in diesem Semester ändert.

Einführung, Motivation (6)

Warum sollte man mathematische Logik lernen II:

- SQL verändert sich und wird zunehmend komplexer (Standards: 1986, 1989, 1992, 1999, 2003).

  Irgendwann wird vermutlich jemand eine wesentlich einfachere Sprache vorschlagen, die das gleiche kann. Datalog war schon ein solcher Vorschlag, hat sich aber bisher noch nicht durchsetzen können. Aber man denke an Algol 68 und PL/I, gefolgt von Pascal and C.

- Es werden neue Datenmodelle vorgeschlagen (z.B. XML), mit noch schnelleren Änderungen als SQL.
- Zumindest ein Teil dieser Vorlesung sollte auch in 30 Jahren noch gültig sein.

Geschichte des Gebietes (1)

~322 v.Chr.	Syllogismen [Aristoteles]
~300 v.Chr.	Axiome der Geometrie [Euklid]
~1700	Plan der mathematischen Logik [Leibniz]
1847	"Algebra der Logik" [Boole]
1879	"Begriffsschrift" (frühe logische Formeln) [Frege]
~1900	natürlichere Formelsyntax [Peano]
1910/13	Principia Mathematica (Sammlung formaler Beweise) [Whitehead/Russel]
1930	Vollständigkeitssatz [Gödel/Herbrand]
1936	Unentscheidbarkeit [Church/Turing]

Geschichte des Gebietes (2)

1960	Erster Theorembeweiser [Gilmore/Davis/Putnam]
1963	Resolutionsmethode zum automatischen Beweisen [Robinson]
~1969	Frage-Antwort-Systeme [Green et.al.]
1970	Lineare Resolution [Loveland/Luckham]
1970	Relationales Datenmodell [Codd]
~1973	Prolog [Colmerauer, Roussel, et.al.] (begann als Theorembeweiser für das Verstehen natürl. Sprache) (Vgl.: Fortran 1954, Lisp 1962, Pascal 1970, Ada 1979)
1977	Konferenz "Logik und Datenbanken" [Gallaire, Minker]

Inhalt

1. Einführung, Motivation, Geschichte
2. Signaturen, Interpretationen
3. Formeln, Modelle
4. Formeln in Datenbanken
5. Implikation, Äquivalenz

Alphabet (1)

Definition:

- Sei $ALPH$ eine unendliche, aber abzählbare Menge von Elementen, die Symbole genannt werden.

  Formeln sind Wörter über $ALPH$, d.h. Folgen von Symbolen.

- $ALPH$ muss zumindest die logischen Symbole enthalten, d.h. $LOG \subseteq ALPH$, wobei

$$LOG = \{(,\ ),\ ,\ ,\ \top,\ \bot,\ =,\ \neg,\ \wedge,\ \vee,\ \leftarrow,\ \rightarrow,\ \leftrightarrow,\ \forall,\ \exists\}.$$

- Zusätzlich muss $ALPH$ eine unendliche Teilmenge $VARS \subseteq ALPH$ enthalten, die Menge der Variablen. Diese ist disjunkt zu LOG (d.h. $VARS \cap LOG = \emptyset$).

  Einige Autoren betrachten Variablen als logische Symbole.

Alphabet (2)

- Z.B kann das Alphabet bestehen aus
  - ◇ den speziellen logischen Symbolen LOG,
  - ◇ Variablen, beginnend mit einem Grobuchstaben, und bestehend aus Buchstaben, Ziffern und "_",
  - ◇ Bezeichnern, beginnend mit einem Kleinbuchstaben, bestehend aus Buchstaben, Ziffern und "_".
- Man beachte, dass Wörter wie "`father`" als Symbole angesehen werden (Elemente des Alphabets).

  Vgl.: lexikalischer Scanner vs. kontextfreier Parser in einem Compiler.

- In der Theorie sind die exakten Symbole unwichtig.

Alphabet (3)

Mögliche Alternativen für logische Symbole:

Symbol	Alternative	Alt2	Name
$\top$	`true`	`T`	
$\bot$	`false`	`F`	
$\neg$	`not`	`~`	Negation
$\wedge$	`and`	`&`	Konjunktion
$\vee$	`or`	`|`	Disjunktion
$\leftarrow$	`if`	`<-`	
$\rightarrow$	`then`	`->`	
$\leftrightarrow$	`iff`	`<->`	
$\exists$	`exists`	`E`	Existenzquantor
$\forall$	`forall`	`A`	Allquantor

Signaturen (1)

Definition:

- Eine Signatur $\Sigma = (\mathcal{S}, \mathcal{P}, \mathcal{F})$ enthält:
  - ◇ Eine nichtleere, endliche Menge $\mathcal{S}$. Die Elemente heißen Sorten (Datentypnamen).
  - ◇ Für jedes $\alpha = s_1 \ldots s_n \in \mathcal{S}^*$, eine endliche Menge (Prädikatssymbole) $\mathcal{P}_\alpha \subseteq ALPH - (LOG \cup VARS)$.
  - ◇ Für jedes $\alpha \in \mathcal{S}^*$ und $s \in \mathcal{S}$, eine Menge (von Funktionssymbolen) $\mathcal{F}_{\alpha,s} \subseteq ALPH - (LOG \cup VARS)$.
- Für jedes $\alpha \in \mathcal{S}^*$ und $s_1, s_2 \in \mathcal{S}$, $s_1 \neq s_2$ muss gelten, dass $\mathcal{F}_{\alpha,s_1} \cap \mathcal{F}_{\alpha,s_2} = \emptyset$.

Signaturen (2)

- Sorten sind Datentypnamen, z.B. `string`, `person`.
- Prädikate liefern für gegebene Eingabewerte wahr oder falsch, z.B. `<`, `substring_of`, `female`.
- Ist $p \in \mathcal{P}_\alpha$, dann werden $\alpha = s_1, \ldots, s_n$ die Argumentsorten von p genannt.

  s_1 ist der Typ des ersten Arguments, s_2 der des zweiten, usw.
- Beispiele (formal und übliche Syntax):
  - ◇ $< \, \in \mathcal{P}_{\texttt{int int}}$, oder anders geschrieben `<(int, int)`.
  - ◇ $\texttt{female} \in \mathcal{P}_{\texttt{person}}$, oder `female(person)`.

Signaturen (3)

- Die Anzahl der Argumentsorten (Länge von α) nennt man Stelligkeit des Prädikatsymbols, z.B.:
  - ◇ `<` ist ein Prädikatsymbol der Stelligkeit 2.
  - ◇ `female` ist ein Prädikatsymbol der Stelligkeit 1.
- Prädikate der Stelligkeit 0 nennt man aussagenlogische Konstanten/Symbole. Z.B.:
  - ◇ `the_sun_is_shining`,
  - ◇ `i_am_working`.
- Die Menge $\mathcal{P}_\epsilon$ enthält alle aussagenlogischen Konstanten (ϵ ist das leere Wort).

Signaturen (4)

- Das gleiche Symbol p kann Element verschiedener $\mathcal{P}_\alpha$ sein (überladenes Prädikat), z.B.
  - ◇ $< \, \in \mathcal{P}_{\texttt{int int}}$.
  - ◇ $< \, \in \mathcal{P}_{\texttt{string string}}$ (lexikographische/alphabetische Ordnung).
- Es kann also mehrere verschiedene Prädikate mit gleichem Namen geben.

  Die Möglichkeit überladener Prädikate ist nicht wichtig. Man kann stattdessen auch verschiedene Namen verwenden, etwa `lt_int` und `lt_string`. Überladene Prädikate komplizieren die Definition und sind in der mathematischen Logik normalerweise ausgeschlossen. Sie erlauben aber natürlichere Formulierungen.

Signaturen (5)

- Eine Funktion liefert für gegebene Eingabewerte einen Ausgabewert. Beispiele: `+`, `age`, `first_name`.

  Es sei angenommen, dass Funktionen für alle Eingabewerte definiert sind. Das muss nicht sein, z.B. `telefax_no(peter)` könnte nicht existieren und `5/0` ist nicht definiert. SQL verwendet eine dreiwertige Logik zur Behandlung von Nullwerten (Aussagen können wahr, falsch oder undefiniert sein). Man muss Kompromisse schließen zwischen exakter Beschreibung der Realität und einem genügend einfachen Modell.
- Ein Funktionssymbol in $\mathcal{F}_{\alpha,s}$ hat die Argumentsorten α und die Ergebnissorte s, z.B.
  - ◇ $+ \in \mathcal{F}_{\texttt{int int},\,\texttt{int}}$, übersichtlicher: `+(int, int): int`.
  - ◇ $\texttt{age} \in \mathcal{F}_{\texttt{person},\,\texttt{int}}$, oder: `age(person): int`.

Signaturen (6)

- Eine Funktion ohne Argumente heißt Konstante.
- Beispiele für Konstanten:
  - ◇ $1 \in \mathcal{F}_{\epsilon,\texttt{int}}$, oder übersichtlicher: `1: int`.
  - ◇ $\texttt{'Ann'} \in \mathcal{F}_{\epsilon,\texttt{string}}$, oder: `'Ann': string`.
- Bei Datentypen (z.B. `int`, `string`) kann üblicherweise jeder mögliche Wert durch eine Konstante bezeichnet werden.

  Im Allgemeinen sind die Menge der Werte und die der Konstanten dagegen verschieden. Zum Beispiel wäre es möglich, dass es keine Konstanten vom Typ `person` gibt.

Signaturen (7)

- Zusammengefasst legt eine Signatur anwendungsspezifische Symbole fest, die verwendet werden, um über die entsprechende Miniwelt zu sprechen.
- Hier: mehrsortige (typisierte) Logik.
  Alternative: unsortierte/einsortige Logik.

  Dann wird $\mathcal{S}$ nicht benötigt, und $\mathcal{P}$ und $\mathcal{F}$ haben als Index nur die Stelligkeit. Z.B. Prolog verwendet eine unsortierte Logik. Dies ist auch in Lehrbüchern über mathematische Logik gebräuchlich (die Definitionen sind dann etwas einfacher). Da man Sorten durch Prädikate der Stelligkeit 1 simulieren kann, ist eine einsortige Logik keine echte Einschränkung. Allerdings sind Formeln mit Typfehlern in einer mehrsortierten Logik syntaktisch falsch, während die entsprechende Formel in einer einsortierten Logik legal, aber logisch falsch (inkonsistent) ist.

Signaturen (8)

Beispiel:

- $\mathcal{S} = \{\texttt{person}, \texttt{string}\}$.
- $\mathcal{F}$ besteht aus
  - ◇ Konstanten der Sorte `person` (`arno`, `birgit`, `chris`).
  - ◇ unendlich vielen Konstanten der Sorte `string`, z.B. `''`, `'a'`, `'b'`, ..., `'Arno'`, ...
  - ◇ Funktionssymbolen `first_name(person): string` und `last_name(person): string`.
- $\mathcal{P}$ besteht aus
  - ◇ dem Prädikat `married_to(person, person)`.
  - ◇ den Prädikaten `male(person)` und `female(person)`.

Signaturen (9)

Übung:

- Definieren Sie eine Signatur über
  - ◇ Bücher (mit Autoren, Titel, ISBN)

    Es reicht aus, die Liste der Autoren eines Buches als einen String zu behandeln. Fortgeschrittene Übung: Behandeln Sie Bücher mit mehreren Autoren, indem Sie Listen von Strings modellieren.
  - ◇ Buchbesprechungen (mit Kritiker, Text, Sternen).

    Jede Besprechung ist für genau ein Buch.
  - ◇ "Sterne" können keiner, einer, zwei oder drei sein.

Signaturen (10)

Definition:

- Eine Signatur $\Sigma' = (\mathcal{S}', \mathcal{P}', \mathcal{F}')$ ist eine Erweiterung der Signatur $\Sigma = (\mathcal{S}, \mathcal{P}, \mathcal{F})$, falls
  - ◇ $\mathcal{S} \subseteq \mathcal{S}'$,
  - ◇ für jedes $\alpha \in \mathcal{S}^*$:
    $\mathcal{P}_\alpha \subseteq \mathcal{P}'_\alpha$,
  - ◇ für jedes $\alpha \in \mathcal{S}^*$:
    und $s \in \mathcal{S}$: $\mathcal{F}_{\alpha,s} \subseteq \mathcal{F}'_{\alpha,s}$.
- D.h. eine Erweiterung von Σ fügt neue Symbole zu Σ hinzu.

Interpretationen (1)

Definition:

- Sei die Signatur $\Sigma = (\mathcal{S}, \mathcal{P}, \mathcal{F})$ gegeben.
- Eine Σ-Interpretation $\mathcal{I}$ definiert:
  - ◇ eine Menge $\mathcal{I}(s)$ für jedes $s \in \mathcal{S}$ (Wertebereich),
  - ◇ Eine Relation $\mathcal{I}(p, \alpha) \subseteq \mathcal{I}(s_1) \times \cdots \times \mathcal{I}(s_n)$ für jedes $p \in \mathcal{P}_\alpha$ und $\alpha = s_1 \ldots s_n \in \mathcal{S}^*$.

    Im Folgenden schreiben wir $\mathcal{I}(p)$ an Stelle von $\mathcal{I}(p, \alpha)$ wenn α für die gegebene Signatur Σ klar ist (d.h. p nicht überladen ist).
  - ◇ eine Funktion $\mathcal{I}(f, \alpha)$: $\mathcal{I}(s_1) \times \cdots \times \mathcal{I}(s_n) \rightarrow \mathcal{I}(s)$ für jedes $f \in \mathcal{F}_{\alpha,s}$, $s \in \mathcal{S}$ und $\alpha = s_1 \ldots s_n \in \mathcal{S}^*$.
- Im Folgenden schreiben wir $\mathcal{I}[\ldots]$ anstatt $\mathcal{I}(\ldots)$.

Interpretationen (2)

Beachte:

- Leere Wertebereiche verursachen Probleme, deshalb werden sie normalerweise ausgeschlossen.

  Einige Äquivalenzen gelten nicht, wenn die Bereiche leer sein können. Zum Beispiel, kann die Prenex-Normalform nur unter der Annahme erreicht werden, dass die Bereiche nicht leer sind.
- In Datenbanken kann dies aber vorkommen.

  Z.B. Menge von Personen, wenn die Datenbank gerade erst erstellt wurde. Auch bei SQL kann es Überraschungen geben, wenn eine Tupelvariable über einer leeren Relation deklariert ist.
- Statt Wertebereich sagt man auch Individuenbereich oder Domäne (engl. Domain).

Interpretationen (3)

- Die Relation $\mathcal{I}[p]$ wird auch die Extension von p genannt (in $\mathcal{I}$).
- Formal gesehen sind Prädikat und Relation nicht gleiche, aber isomorphe Begriffe.

  Ein Prädikat ist eine Abbildung auf die Menge $\{true, false\}$ boolescher Werte. Eine Relation ist eine Teilmenge des kartesischen Produkts $\times$.
- Zum Beispiel ist $\texttt{married\_to}(\texttt{X}, \texttt{Y})$ genau dann wahr in $\mathcal{I}$, wenn $(\texttt{X}, \texttt{Y}) \in \mathcal{I}[\texttt{married\_to}]$.
- Weiteres Beispiel: $(3, 5) \in \mathcal{I}[<]$ bedeutet $3 < 5$.

  Im Folgenden werden die Wörter “Prädikatsymbol” und “Relationssymbol” austauschbar verwendet.

Interpretationen (4)

Beispiel (Interpretation für Signatur auf Folie 2-23):

- $\mathcal{I}[\texttt{person}]$ ist die Menge mit Arno, Birgit und Chris.
- $\mathcal{I}[\texttt{string}]$ ist die Menge aller Strings, z.B. `'a'`.
- $\mathcal{I}[\texttt{arno}]$ ist Arno.
- Für Stringkonstanten c ist $\mathcal{I}[c] = c$.
- $\mathcal{I}[\texttt{first\_name}]$ bildet z.B. Arno auf `'Arno'` ab.
- $\mathcal{I}[\texttt{last\_name}]$ liefert für alle drei Personen `'Schmidt'`.
- $\mathcal{I}[\texttt{married\_to}] = \{(\text{Birgit}, \text{Chris}), (\text{Chris}, \text{Birgit})\}$.
- $\mathcal{I}[\texttt{male}] = \{(\text{Arno}), (\text{Chris})\}$, $\mathcal{I}[\texttt{female}] = \{(\text{Birgit})\}$.

Relationale Datenbanken (1)

- Ein DBMS definiert eine Menge von Datentypen, wie Strings oder Zahlen, mit Konstanten, Datentypfunktionen (z.B. $+$) und Prädikaten (z.B. $<$).
- Für diese definiert das DBMS Namen (in der Signatur Σ) und Bedeutung (in der Interpretation $\mathcal{I}$).
- Für jeden Wert $d \in \mathcal{I}[s]$ gibt es mindestens eine Konstante c mit $\mathcal{I}[c] = d$.

  D.h. alle Datenwerte sind durch Konstanten benannt. Das wird auch Bereichsabschlußannahme genannt und ist z.B. zur Ausgabe von Datenwerten im Anfrageergebnis wichtig. Im Allg. können verschiedene Konstanten den gleichen Datenwert bezeichnen, z.B. `0`, `00`, `-0`.

Relationale Datenbanken (2)

- Das DB-Schema im relationalen Modell fügt dann Prädikatsymbole (Relationssymbole) hinzu.

  Diese sind die formale Entsprechung der "Tabellen", die in Kapitel 1 genannt wurden.

- Der DB-Zustand interpretiert diese durch endliche Relationen.

  Während die Interpretation der Datentypen festgeschrieben und in das DBMS eingebaut ist, kann die Interpretation der zusätzlichen Prädikatsymbole (DB-Relationen) durch Einfügen, Löschen und Updates verändert werden. Dafür müssen die DB-Relationen aber eine endliche Erweiterung haben. Datentypprädikate sind durch Prozeduren im DBMS implementiert, während die Prädikate des DB-Schemas durch Dateien auf der Platte implementiert werden.

Relationale Datenbanken (3)

- Die wesentlichen Einschränkungen des relationalen Modells sind also:
  - ◇ Keine neuen Sorten (Typen),
  - ◇ Keine neuen Funktionssymbole und Konstanten,
  - ◇ Neue Prädikatsymbole können nur durch endliche Relationen interpretiert werden.
- Zusätzlich müssen Formeln "bereichsunabhängig" oder "bereichsbeschränkt" sein (siehe unten).

  Es sind nicht beliebige Formeln erlaubt, um sicherzustellen, dass Formeln einer gegebenen Interpretation in endlicher Zeit auswertet werden können (obwohl z.B. `int` als unendliche Menge interpretiert wird).

Relationale Datenbanken (4)

Beispiel:

- In einer relationalen DB zur Speicherung von Hausaufgabenpunkten könnte es drei Prädikate geben:
  - ◇ `student(int SID, string FName, string LName)`

    Argumentnamen erklären die Bedeutung der Argumente. Das erste ist eine eindeutige Nummer ("student ID"), das zweite der Vorname des Studenten mit dieser ID und das dritte der Nachname. Z.B. könnte `student(101, 'Ann', 'Smith')` wahr sein.
  - ◇ `exercise(int ENO, int MaxPoints)`

    Z.B. bedeutet `exercise(1, 10)`: für Übung 1 gibt es 10 Punkte.
  - ◇ `result(int SID, int ENO, int Points)`

    Z.B. bedeutet `result(101, 1, 9)`, dass Ann Smith (die Studentin mit Nummer 101) 9 Punkte für Übung 1 bekommen hat.

Relationale Datenbanken (5)

- Hier behandeln wir die "Bereichskalkül"-Version des relationalen Modells.

  Es gibt auch eine "Tupelkalkül"-Version des relationalen Modells, die SQL noch ähnlicher ist. Die Unterschiede sind nicht wesentlich, z.B. können Anfragen automatisch in beide Richtungen umgewandelt werden. Die Definition des Tupelkalküls ist etwas komplizierter, da Variablen im Tupelkalkül über ganze Tupel (Tabellenzeilen) laufen. Deshalb werden in Lehrbüchern beide Versionen in getrennten Kapiteln als zwei verschiedene logische Formalismen behandelt. Wenn man jedoch einen Formalismus verstanden hat, ist es sehr leicht den anderen zu lernen. Außerdem kann der obige Formalismus den in SQL verwendeten Teil des Tupelkalküls behandeln (das ist wie das ER-Modell ohne Relationships, siehe unten). Der schwierige Teil sind Tupelvariablen, die nicht direkt an DB-Relationen gebunden sind (SQL verbietet dieses und verwendet stattdessen `UNION`).

Entity-Relationship-Modell (1)

- Im Entity-Relationship-Modell kann das DB-Schema folgendes einführen (bei gegebenen Datentypen):
  - ◇ neue Sorten ("Entity-Typen"),
  - ◇ neue Funktionen der Stelligkeit 1 von Entity-Typen zu Datentypen ("Attribute"),
  - ◇ neue Prädikate zwischen Entity-Typen, evtl. beschränkt auf Stelligkeit 2 ("Relationships").
  - ◇ neue Funktionen, die auf gleichen Entity-Typen wie Relationships definiert sind, aber einen Datentyp zurückgeben ("Relationship-Attribute").

Entity-Relationship-Modell (2)

- Die Interpretation von Entity-Typen (im Datenbankzustand) muss immer endlich sein.

  Damit sind auch Attribute und Relationships endlich.
- Funktionen für Relationship-Attribute müssen bei Eingabewerten, für die das Relationship falsch ist, einen festen Dummywert als Ausgabe haben.

  Anfragen sollten so geschrieben sein, dass der genaue Dummywert für das Anfrageergebnis unwichtig ist. Z.B. wenn f ein Attribut des Relationships p ist, würde eine Formel der Form $p(X,Y) \wedge f(X,Y) = Z$ diese Eigenschaft haben. Für eine saubere Behandlung von Relationship-Attributen müsste man die Logik erweitern (um partiell definierte Funktionen), aber das ist wohl den Aufwand nicht wert.

Entity-Relationship-Modell (3)

Beispiel (Hausaufgabenpunkte):

- Sorten: `student` und `exercise`.
- Funktionen:
  - ◇ `sid(student): int`
  - ◇ `first_name(student): string`
  - ◇ `last_name(student): string`.
  - ◇ `eno(exercise): int`
  - ◇ `maxpoints(exercise): int`
- Prädikat: `solved(student, exercise)`.
  Funktion: `points(student, exercise): int`

Inhalt

Variablendeklaration (1)

Definition:

- Sei die Signatur $\Sigma = (\mathcal{S}, \mathcal{P}, \mathcal{F})$ gegeben.
- Eine Variablendeklaration für Σ ist eine partielle Abbildung $\nu: \mathit{VARS} \rightarrow \mathcal{S}$.

  Sie ist nur für eine endliche Teilmenge von VARS definiert. Das ist keine Einschränkung, weil jede Formel nur endlich viele Variablen enthält.

Bemerkung:

- Die Variablendeklaration ist nicht Teil der Signatur, da sie lokal durch Quantoren geändert wird (s.u.).

  Die Signatur ist für die gesamte Anwendung fest, die Variablendeklaration ändert sich schon innerhalb einer der Formel.

Variablendeklaration (2)

Beispiel:

- Variablendeklarationen definieren, welche Variablen verwendet werden können, und was ihre Sorten sind:

ν	
Variable	Sorte
`SID`	`int`
`Points`	`int`
`E`	`exercise`

  Jede Variable muss eine eindeutige Sorte haben.

- Variablendeklarationen werden auch in der Form $\nu = \{$`SID/int`, `Points/int`, `E/exercise`$\}$ geschrieben.

Variablendeklaration (3)

Definition:

- Sei ν eine Variablendeklaration, $X \in \mathit{VARS}$, und $s \in \mathcal{S}$.
- Dann schreiben wir $\nu\langle X/s\rangle$ für die lokal modifizierte Variablendeklaration ν' mit

$$\nu'(V) := \begin{cases} s & \text{falls } V = X \\ \nu(V) & \text{sonst.} \end{cases}$$

Bemerkung:

- Beides ist möglich: ν kann für X schon definiert sein, oder an dieser Stelle bisher undefiniert sein.

Terme (1)

- Terme sind syntaktische Konstrukte, die zu einem Wert ausgewertet werden können (z.B. zu einer Zahl, einer Zeichenkette, oder einer Person).
- Es gibt drei Arten von Termen:
  - ◇ Konstanten, z.B. `1`, `'abc'`, `arno`,
  - ◇ Variablen, z.B. `X`,
  - ◇ zusammengesetzte Terme, bestehend aus Funktionssymbolen angewandt auf Argumentterme, z.B. `last_name(arno)`.
- In Programmiersprachen: Term = Ausdruck.

Terme (2)

Definition:

- Sei eine Signatur $\Sigma = (\mathcal{S}, \mathcal{P}, \mathcal{F})$ und eine Variablendeklaration ν für Σ gegeben.
- Die Menge $\mathit{TE}_{\Sigma,\nu}(s)$ der Terme der Sorte s ist folgendermaßen rekursiv definiert:
  - ◇ Jede Variable $V \in \mathit{VARS}$ mit $\nu(V) = s$ ist ein Term der Sorte s (dafür muss ν definiert sein für V).
  - ◇ Jede Konstante $c \in \mathcal{F}_{\epsilon,s}$ ist ein Term der Sorte s.
  - ◇ Wenn t_1 ein Term der Sorte s_1 ist, ..., t_n ein Term der Sorte s_n, und $f \in \mathcal{F}_{\alpha,s}$ mit $\alpha = s_1 \ldots s_n$, $n \geq 1$, dann ist $f(t_1, \ldots, t_n)$ ein Term der Sorte s.

Terme (3)

Definition, fortgesetzt:

- Jeder Term kann durch endlich häufige Anwendung obiger Regeln konstruiert werden. Nichts anderes ist ein Term.

  Diese Bemerkung ist formal wichtig, da die obigen Regeln nur festlegen, was ein Term ist, und nicht was, kein Term ist. Dazu muss die Definition abgeschlossen werden. (Selbst wenn man die obigen Regeln als Gleichungssystem auffasst, müßten unendliche Baumstrukturen als Lösungen ausgeschlossen werden.)

Definition:

- $\mathit{TE}_{\Sigma,\nu} := \bigcup_{s \in \mathcal{S}} \mathit{TE}_{\Sigma,\nu}(s)$ sei die Menge aller Terme.

Terme (4)

- Bei einigen Funktionen ist Infixnotation üblich, also z.B. X+1 statt der "offiziellen" Notation +(X, 1).

  Wenn man damit anfängt, muss man eigentlich auch Rangfolgen (Prioritäten) der Operatoren definieren, und explizite Klammerung erlauben. Die formalen Definitionen werden dadurch komplizierter.

- Aufrufe von Funktionen der Stelligkeit 1 kann man auch in Punktnotation (objektorientiert) schreiben, z.B. "X.last_name" für "last_name(X)".

Terme (5)

- "Syntaktischer Zucker" wie Infix- und Punktnotation ist in der Praxis sinnvoll, aber für die Theorie der Logik nicht wichtig.

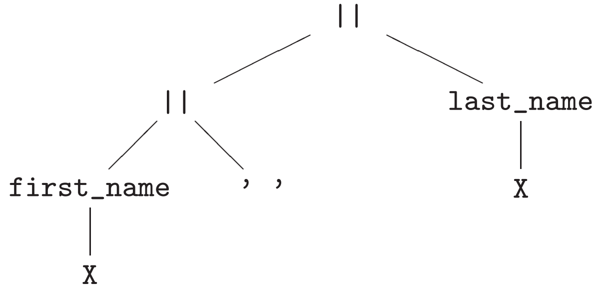
  In Programmiersprachen gibt es manchmal Unterschiede zwischen der "konkreten Syntax" und der "abstrakten Syntax" (Syntaxbaum). Die abstrakte Syntax läßt viele Details weg und beschreibt eher die internen Datenstrukturen des Compilers.

- Im Folgenden nutzen wir die obigen Abkürzungen in praktischen Beispielen, aber die formalen Definitionen behandeln nur die Standardnotation.

  Die Übersetzung von Abkürzungen in Standardnotation sollte offensichtlich sein.

Terme (6)

- Terme kann man in Operatorbäumen visualisieren ("||" bezeichnet in SQL die Stringkonkatenation):



- Übung: Wie kann man diesen Term mit "||" als Infixoperator und mit Punktnotation schreiben?

Terme (7)

Übung:

- Welche der folgenden Ausdrücke sind korrekte Terme (bezüglich der Signatur auf Folie 2-23 und einer Variablendeklaration ν mit $\nu(\mathtt{X}) = \mathtt{string}$)?
  - ☐ arno
  - ☐ first_name
  - ☐ first_name(X)
  - ☐ firstname(arno, birgit)
  - ☐ married_to(birgit, chris)
  - ☐ X

Atomare Formeln (1)

- Formeln sind syntaktische Ausdrücke, die zu einem Wahrheitswert ausgewertet werden können, z.B

$$1 \leq X \ \wedge X \leq 10.$$

- Atomare Formeln sind die grundlegenden Bestandteile zur Bildung dieser Formeln (Vergleiche etc.).
- Atomare Formeln können folgende Formen haben:
  - ◇ Ein Prädikatsymbol, angewandt auf Terme, z.B. `married_to(birgit, X)`.
  - ◇ Eine Gleichung, z.B. `X = chris`.
  - ◇ Die log. Konstanten $\top$ (wahr) und $\bot$ (falsch).

Atomare Formeln (2)

Definition:

- Sei eine Signatur $\Sigma = (\mathcal{S}, \mathcal{P}, \mathcal{F})$ und und eine Variablendeklaration ν für Σ gegeben.
- Eine atomare Formel ist ein Ausdruck von einer der folgenden Formen:
  - ◇ $p(t_1, \ldots, t_n)$ mit $p \in \mathcal{P}_\alpha$, $\alpha = s_1 \ldots s_n \in \mathcal{S}^*$, und $t_i \in TE_{\Sigma,\nu}(s_i)$ für $i = 1, \ldots, n$.
  - ◇ $t_1 = t_2$ mit $t_1, t_2 \in TE_{\Sigma,\nu}(s)$, $s \in \mathcal{S}$.
  - ◇ $\top$ und $\bot$.
- $AT_{\Sigma,\nu}$ sei die Menge der atomaren Formeln für Σ, ν.

Atomare Formeln (3)

Bemerkungen:

- Für einige Prädikate verwendet man traditionell Infixnotation, z.B. $X > 1$ statt $>(X, 1)$.

  Wir werden natürlich diese bekannte Notation im Beispiel verwenden.

- Bei aussagenlogischen Konstanten kann man die Klammern weglassen, z.B. p statt $p()$.
- Es ist möglich, "=" als normales Prädikat zu behandeln, wie es einige Autoren auch tun.

  Die obige Definition sichert, dass zumindest die Gleichheit für alle Sorten verfügbar ist, und wir werden unten sicherstellen, dass es auch immer die Standardinterpretation hat.

Formeln(1)

Definition:

- Sei eine Signatur $\Sigma = (\mathcal{S}, \mathcal{P}, \mathcal{F})$ und eine Variablendeklaration ν für Σ gegeben.
- Die Mengen $FO_{\Sigma,\nu}$ der (Σ, ν)-Formeln sind folgendermaßen rekursiv definiert:
  - ◇ Jede atomare Formel $F \in AT_{\Sigma,\nu}$ ist eine Formel.
  - ◇ Wenn F und G Formeln sind, so auch $(\neg F)$, $(F \wedge G)$, $(F \vee G)$, $(F \leftarrow G)$, $(F \rightarrow G)$, $(F \leftrightarrow G)$.
  - ◇ $(\forall\, s\, X\colon F)$ und $(\exists\, s\, X\colon F)$ sind in $FO_{\Sigma,\nu}$ falls $s \in \mathcal{S}$, $X \in VARS$, und F eine $(\,\Sigma,\ \nu\langle X/s\rangle\,)$-Formel ist.
  - ◇ Nichts anderes ist eine Formel.

Formeln (2)

- Die intuitive Bedeutung von Formeln ist wie folgt:
  - $\neg F$: "nicht F" (F ist falsch).
  - $F \wedge G$: "F und G" (beide sind wahr).
  - $F \vee G$: "F oder G" (eine oder beide sind wahr).
  - $F \leftarrow G$: "F falls G" (falls G wahr ist, muss F wahr sein).
  - $F \rightarrow G$: "wenn F, dann G"
  - $F \leftrightarrow G$: "F genau dann, wenn G".
  - $\forall\, s\, X \colon F$: "für alle X (der Sorte s), ist F wahr".
  - $\exists\, s\, X \colon F$: "es gibt ein X (aus s), so dass F gilt".

Formeln (3)

- Bisher wurden viele Klammern gesetzt, um eine eindeutige syntaktische Struktur zu sichern.

  Für die formale Definition ist das eine einfache Lösung, aber für Formeln in realen Anwendungen wird diese Syntax unpraktisch.

- Regeln zur Klammersetzung:
  - Die äußeren Klammern sind niemals notwendig.
  - $\neg$ bindet am stärksten, dann $\wedge$, dann $\vee$, dann $\leftarrow$, $\rightarrow$, $\leftrightarrow$ (gleiche Stärke), und als letztes $\forall$, $\exists$.
  - Da $\wedge$ und $\vee$ assoziativ sind, werden z.B. für $F_1 \wedge F_2 \wedge F_3$ keine Klammern benötigt.

    Beachte, dass $\rightarrow$ und $\leftarrow$ nicht assoziativ sind.

Formeln (4)

Formale Behandlung der Bindungsstärke:

- Eine Formel der Stufe 0 (Formel-0) ist eine atomare Formel oder eine in (...) eingeschlossene Formel-5.

  Die Stufe einer Formel entspricht der Bindungsstärke des äußersten Operators (kleinste Nummer bedeutet höchste Bindungsstärke). Man kann jedoch eine Formel-i wie eine Formel-j verwenden mit $j > i$. In entgegengesetzter Richtung werden Klammern benötigt.

- Eine Formel-1 ist eine Formel-0 oder eine Formel der Form $\neg F$, wobei F eine Formel-1 ist.
- Eine Formel-2 ist eine Formel-1 oder eine Formel der Form $F_1 \wedge F_2$, wobei F_1 eine Formel-2 ist, und F_2 eine Formel-1 (implizite Klammerung von links).

Formeln (5)

Formale Behandlung der Bindungsstärke, fortgesetzt:

- Eine Formel-3 ist eine Formel-2 oder eine Formel der Form $F_1 \vee F_2$ mit einer Formel-3 F_1 und einer Formel-2 F_2.
- Eine Formel-4 ist eine Formel-3 oder eine Formel der Form $F_1 \leftarrow F_2$, $F_1 \rightarrow F_2$, $F_1 \leftrightarrow F_2$, wobei F_1 und F_2 Formeln der Stufe 3 sind.
- Eine Formel-5 ist eine Formel-4 oder eine Formel der Form $\forall s\ X \colon F$, $\exists\, s\ X \colon F$ mit einer Formel-5 F.
- Eine Formel ist eine Formel der Stufe 5 (Formel-5).

Formeln (6)

Abkürzungen für Quantoren:

- Wenn es nur eine mögliche Sorte für eine quantifizierte Variable gibt, kann man sie weglassen, d.h. $\forall X\colon F$ statt $\forall\, s\, X\colon F$ schreiben (entsprechend für $\exists$).

  Oft ist der Typ der Variablen durch ihre Verwendung eindeutig festgelegt (z.B. Argument eines Prädikates mit eindeutiger Argumentsorte).

- Wenn ein Quantor direkt auf einen anderen Quantor folgt, kann man den Doppelpunkt weglassen.

  Z.B. $\forall X\, \exists Y\colon F$ statt $\forall X\colon \exists Y\colon F$.

- Statt einer Sequenz von Quantoren gleichen Typs, z.B. $\forall X_1 \ldots \forall X_n\colon F$, schreibt man $\forall X_1, \ldots, X_n\colon F$.

Formeln (7)

Abkürzung für Ungleichheit:

- $t_1 \neq t_2$ kann als AbKürzung für $\neg(t_1 = t_2)$ verwendet werden.

Übung:

- Sei eine Signatur gegeben mit $\leq\, \in \mathcal{P}_{\text{int int}}$ und $1, 10 \in \mathcal{F}_{\epsilon,\text{int}}$, und eine Variablendeklaration mit $\nu(X) = \texttt{int}$.
- Ist $1 \leq X \leq 10$ eine syntaktisch korrekte Formel?

Formeln(8)

Übung:

- Welche der folgenden Formeln sind syntaktisch korrekt (bezüglich der Signatur von Folie 2-23)?
  - ☐ $\forall\, \texttt{X}, \texttt{Y}\colon \texttt{married\_to}(\texttt{X}, \texttt{Y}) \rightarrow \texttt{married\_to}(\texttt{Y}, \texttt{X})$
  - ☐ $\forall\, \texttt{person P}\colon\ \vee\, \texttt{male}(\texttt{P}) \vee \texttt{female}(\texttt{P})$
  - ☐ $\forall\, \texttt{person P}\colon\ \texttt{arno} \vee \texttt{birgit} \vee \texttt{chris}$
  - ☐ $\texttt{male}(\texttt{chris})$
  - ☐ $\forall\, \texttt{string X}\colon\ \exists\, \texttt{person X}\colon\ \texttt{married\_to}(\texttt{birgit}, \texttt{X})$
  - ☐ $\texttt{married\_to}(\texttt{birgit}, \texttt{chris})$ $\wedge\ \vee\ \texttt{married\_to}(\texttt{chris}, \texttt{birgit})$

Geschlossenene Formeln (1)

Definition:

- Sei eine Signatur Σ gegeben.
- Eine geschlossene Formel (für Σ) ist eine (Σ, ν)-Formel für die leere Variablendeklaration ν.

  D.h. die Variablendeklaration, die überall undefiniert ist.

Übung:

- Welche der folgenden Formeln sind geschlossen?
  - ☐ $\texttt{female}(\texttt{X}) \wedge \exists\, \texttt{X}\colon \texttt{married\_to}(\texttt{chris}, \texttt{X})$
  - ☐ $\texttt{female}(\texttt{birgit}) \wedge \texttt{married\_to}(\texttt{chris}, \texttt{birgit})$
  - ☐ $\exists\, \texttt{X}\colon \texttt{married\_to}(\texttt{X}, \texttt{Y})$

Geschlossenene Formeln (2)

Bemerkung:

- Um festzulegen, ob eine Formel wahr oder falsch ist, braucht man außer einer Interpretation auch Werte für die Variablen, die nicht durch Quantoren gebunden sind (freie Variablen).
- Bei geschlossenen Formeln: Interpretation reicht.

  Alle in der Formel verwendeten Variablen sind in der Formel selbst durch einen Quantor eingeführt. Es gibt keine Variablen, die schon von außen kommen (gewissermaßen als Parameter).

Variablen in einem Term

Definition:

- Die Funktion $vars$ berechnet die Menge der Variablen, die in einem gegebenem Term t auftreten.
  - ◇ Wenn t eine Konstante c ist:
  $$vars(t) := \emptyset.$$
  - ◇ Wenn t eine Variable V ist:
  $$vars(t) := \{V\}.$$
  - ◇ Wenn t die Form $f(t_1, \ldots, t_n)$ hat:
  $$vars(t) := \bigcup_{i=1}^{n} vars(t_i).$$

Freie Variablen einer Formel

Definition:

- $free(F)$ ist die Menge der freien Variablen in F:
  - ◇ Ist F atomare Formel $p(t_1, \ldots, t_n)$ oder $t_1 = t_2$:
  $$free(F) := \bigcup_{i=1}^{n} vars(t_i).$$
  - ◇ Ist F logische Konstante $\top$ oder $\bot$: $free(F) := \emptyset$.
  - ◇ Wenn F die Form $(\neg G)$ hat: $free(F) := free(G)$.
  - ◇ Wenn F die Form $(G_1 \land G_2)$, $(G_1 \lor G_2)$, etc. hat: $free(F) := free(G_1) \cup free(G_2)$.
  - ◇ Wenn F die Form $(\forall\, s\, X\colon G)$ oder $(\exists\, s\, X\colon G)$ hat: $free(F) := free(G) - \{X\}$.

Variablenbelegung (1)

Definition:

- Eine Variablenbelegung $\mathcal{A}$ für $\mathcal{I}$ und ν ist eine partielle Abbildung von $VARS$ auf $\bigcup_{s \in \mathcal{S}} \mathcal{I}[s]$.
- Sie definiert für jede Variable V, für die ν definiert ist, einen Wert aus $\mathcal{I}[s]$, wobei $s := \nu(V)$.

Bemerkung:

- D.h. eine Variablenbelegung für $\mathcal{I}$ und ν definiert Werte von $\mathcal{I}$ für die Variablen, die in ν deklariert sind.

Variablenbelegung (2)

Beispiel:

- Betrachte die folgende Variablendeklaration ν:

ν	
Variable	Sorte
`X`	`string`
`Y`	`person`

- Eine mögliche Variablenbelegung ist

$\mathcal{A}$	
Variable	Wert
`X`	abc
`Y`	Chris

Variablenbelegung (3)

Definition:

- $\mathcal{A}\langle X/d\rangle$ kennzeichnet die Variablenbelegung $\mathcal{A}'$, die bis auf $\mathcal{A}'(X) = d$ mit $\mathcal{A}$ übereinstimmt.

Beispiel:

- Wenn $\mathcal{A}$ die Variablendeklaration von der letzten Folie ist, so ist $\mathcal{A}\langle$`Y`/Birgit$\rangle$:

$\mathcal{A}'$	
Variable	Wert
`X`	abc
`Y`	Birgit

Wert eines Terms

Definition:

- Sei eine Signatur Σ, eine Variablendeklaration ν für Σ, eine Σ-Interpretation $\mathcal{I}$, und eine Variablenbelegung $\mathcal{A}$ für $(\mathcal{I}, \nu)$ gegeben.
- Der Wert $\langle\mathcal{I}, \mathcal{A}\rangle[t]$ eines Terms $t \in TE_{\Sigma,\nu}$ ist wie folgt definiert (Rekursion über die Termstruktur):
  - ◇ Ist t eine Konstante c, dann ist $\langle\mathcal{I}, \mathcal{A}\rangle[t] := \mathcal{I}[c]$.
  - ◇ Ist t eine Variable V, dann ist $\langle\mathcal{I}, \mathcal{A}\rangle[t] := \mathcal{A}(V)$.
  - ◇ Hat t die Form $f(t_1, \ldots, t_n)$, mit t_i der Sorte s_i: $\langle\mathcal{I}, \mathcal{A}\rangle[t] := \mathcal{I}[f, s_1 \ldots s_n](\langle\mathcal{I}, \mathcal{A}\rangle[t_1], \ldots, \langle\mathcal{I}, \mathcal{A}\rangle[t_n])$.

Wahrheit einer Formel (1)

Definition:

- Der Wahrheitswert $\langle\mathcal{I}, \mathcal{A}\rangle[F] \in \{0, 1\}$ einer Formel F in $(\mathcal{I}, \mathcal{A})$ ist definiert als (0 bedeutet falsch, 1 wahr):
  - ◇ Ist F eine atomare Formel $p(t_1, \ldots, t_n)$ mit den Termen t_i der Sorte s_i:

$$\langle\mathcal{I}, \mathcal{A}\rangle[F] := \begin{cases} 1 & \text{falls } (\langle\mathcal{I}, \mathcal{A}\rangle[t_1], \ldots, \langle\mathcal{I}, \mathcal{A}\rangle[t_n]) \\ & \quad \in \mathcal{I}[p,\, s_1 \ldots s_n] \\ 0 & \text{sonst.} \end{cases}$$

  - ◇ (auf den nächsten 3 Folien fortgesetzt ...)

Wahrheit einer Formel (2)

Definition, fortgesetzt:

- Wahrheitswert einer Formel, fortgesetzt:
  - ◇ Ist F eine atomare Formel $t_1 = t_2$:

$$\langle \mathcal{I}, \mathcal{A}\rangle[F] := \begin{cases} 1 & \text{falls } \langle \mathcal{I}, \mathcal{A}\rangle[t_1] = \langle \mathcal{I}, \mathcal{A}\rangle[t_2] \\ 0 & \text{sonst.} \end{cases}$$

  - ◇ Ist F "true" $\top$: $\langle \mathcal{I}, \mathcal{A}\rangle[F] := 1$.
  - ◇ Ist F "false" $\bot$: $\langle \mathcal{I}, \mathcal{A}\rangle[F] := 0$.
  - ◇ Hat F die Form $(\neg G)$:

$$\langle \mathcal{I}, \mathcal{A}\rangle[F] := \begin{cases} 1 & \text{falls } \langle \mathcal{I}, \mathcal{A}\rangle[G] = 0 \\ 0 & \text{sonst.} \end{cases}$$

Wahrheit einer Formel (3)

Definition, fortgesetzt:

- Wahrheitswert einer Formel, fortgesetzt:
  - ◇ Hat F die Form $(G_1 \wedge G_2)$, $(G_1 \vee G_2)$, etc.:

G_1	G_2	$\wedge$	$\vee$	$\leftarrow$	$\rightarrow$	$\leftrightarrow$
0	0	0	0	1	1	1
0	1	0	1	0	1	0
1	0	0	1	1	0	0
1	1	1	1	1	1	1

Z.B. falls $\langle \mathcal{I}, \mathcal{A}\rangle[G_1] = 1$ und $\langle \mathcal{I}, \mathcal{A}\rangle[G_2] = 0$ dann $\langle \mathcal{I}, \mathcal{A}\rangle[(G_1 \wedge G_2)] = 0$.

Wahrheit einer Formel (4)

Definition, fortgesetzt:

- Wahrheitswert einer Formel, fortgesetzt:
  - ◇ Hat F die Form $(\forall\, s\, X\colon G)$:

$$\langle \mathcal{I}, \mathcal{A}\rangle[F] := \begin{cases} 1 & \text{falls } \langle \mathcal{I}, \mathcal{A}\langle X/d\rangle\rangle[G] = 1 \\ & \quad \text{für alle } d \in \mathcal{I}[s] \\ 0 & \text{sonst.} \end{cases}$$

  - ◇ Hat F die Form $(\exists\, s\, X\colon G)$:

$$\langle \mathcal{I}, \mathcal{A}\rangle[F] := \begin{cases} 1 & \text{falls } \langle \mathcal{I}, \mathcal{A}\langle X/d\rangle\rangle[G] = 1 \\ & \quad \text{für mindestens ein } d \in \mathcal{I}[s] \\ 0 & \text{sonst.} \end{cases}$$

Modell (1)

Definition:

- Ist $\langle \mathcal{I}, \mathcal{A}\rangle[F] = 1$, so schreibt man auch $\langle \mathcal{I}, \mathcal{A}\rangle \models F$.
- Sei F eine (Σ, ν)-Formel. Eine Σ-Interpretation $\mathcal{I}$ ist ein Modell der Formel F (geschrieben $\mathcal{I} \models F$), falls $\langle \mathcal{I}, \mathcal{A}\rangle[F] = 1$ für alle Variablendeklarationen $\mathcal{A}$.

  D.h. freie Variablen werden als $\forall$-quantifiziert behandelt. Die Variablendekl. ist natürlich unwichtig, falls F eine geschlossene Formel ist.
- Ist $\mathcal{I} \models F$, so sagt man $\mathcal{I}$ erfüllt F.

  Oder: F ist in $\mathcal{I}$ wahr. Das gleiche gilt für $\langle \mathcal{I}, \mathcal{A}\rangle \models F$.
- $\mathcal{I}$ ist ein Modell einer Menge Φ von Σ-Formeln (geschrieben $\mathcal{I} \models \Phi$), gdw. $\mathcal{I} \models F$ für alle $F \in \Phi$.

Modell (2)

Definition:

- Eine Formel F oder eine Menge von Formeln Φ nennt man konsistent, wenn sie ein Modell hat.
- Eine Formel F nennt man erfüllbar, wenn es eine Interpretation $\mathcal{I}$ und eine Variablendeklaration $\mathcal{A}$ gibt, so dass $(\mathcal{I}, \mathcal{A}) \models F$.

  Sonst nennt man sie unerfüllbar. Oft werde ich inkonsistent sagen, wenn ich eigentlich unerfüllbar meine.
- Eine (Σ, ν)-Formel F nennt man Tautologie, gdw. für alle Σ-Interpretationen $\mathcal{I}$ und (Σ, ν)-Variablenzuweisungen $\mathcal{A}$ gilt: $(\mathcal{I}, \mathcal{A}) \models F$.

Modell (3)

Übung:

- Man betrachte die Interpretation auf Folie 2-29:
  - ◇ $\mathcal{I}[\texttt{person}] = \{\text{Arno}, \text{Birgit}, \text{Chris}\}$.
  - ◇ $\mathcal{I}[\texttt{married\_to}] = \{(\text{Birgit}, \text{Chris}),\ (\text{Chris}, \text{Birgit})\}$.
  - ◇ $\mathcal{I}[\texttt{male}] = \{(\text{Arno}),\ (\text{Chris})\}$, $\mathcal{I}[\texttt{female}] = \{(\text{Birgit})\}$.
- Welche der folgenden Formeln ist in $\mathcal{I}$ wahr?
  - ☐ $\forall\, \texttt{person X}\colon \texttt{male(X)} \leftrightarrow \neg\texttt{female(X)}$
  - ☐ $\forall\, \texttt{person X}\colon \texttt{male(X)} \vee \neg\texttt{male(X)}$
  - ☐ $\exists\, \texttt{person X}\colon \texttt{female(X)} \wedge \neg\exists\, \texttt{person Y}\colon \texttt{married\_to(X,Y)}$
  - ☐ $\exists\, \texttt{person X},\ \texttt{person Y},\ \texttt{person Z}\colon \texttt{X}=\texttt{Y} \wedge \texttt{Y}=\texttt{Z} \wedge \texttt{X}\neq\texttt{Z}$

Inhalt

Formeln in Datenbanken (1)

- Wie oben erklärt, definiert das DBMS eine Signatur $\Sigma_{\mathcal{D}}$ und eine Interpretation $\mathcal{I}_{\mathcal{D}}$ für die eingebauten Datentypen (`string`, `int`, ...).
- Dann erweitert das DB-Schema diese Signatur $\Sigma_{\mathcal{D}}$ zu der Signatur Σ aller Symbole, die z.B. in Anfragen verwendet werden können.

  Jedes Datenmodell legt gewisse Restriktionen für die Art der neuen Symbole, die eingeführt werden können, fest. Z.B. kann im klassischen relationalen Modell das Datenbankschema nur neue Prädikatsymbole definieren.

Formeln in Datenbanken (2)

- Der DB-Zustand ist dann eine Interpretation $\mathcal{I}$ für die erweiterte Signatur Σ.

  Das System speichert natürlich nur die Interpretation der Symbole von "$\Sigma - \Sigma_\mathcal{D}$" explizit ab, da die Interpretation der Symbole in $\Sigma_\mathcal{D}$ schon in das DBMS eingebaut ist und nicht verändert werden kann. Außerdem können nicht beliebige Interpretationen als DB-Zustand verwendet werden. Die genauen Restriktionen hängen vom Datenmodell ab, aber die neuen Symbole müssen eine endliche Interpretation haben.

- Formeln werden in Datenbanken verwendet als:
  - ◇ Integritätsbedingungen (Constraints)
  - ◇ Anfragen (Queries)
  - ◇ Definitionen abgeleiteter Symbole (Sichten).

Integritätsbedingungen (1)

- Nicht jede Interpretation ist als DB-Zustand zulässig.

  Der Zweck einer DB ist, einen Teil der realen Welt zu modellieren. In der realen Welt existieren gewisse Restriktionen. Dafür sollen Interpretationen, die diese Restriktionen verletzen, ausgeschlossen werden.

- Z.B. muß eine Person in der realen Welt männlich oder weiblich sein, aber nicht beides. Daher sind folgende Formeln in jedem sinnvollen Zustand erfüllt:
  - ◇ $\forall\,\mathtt{person}\ \mathtt{X}\colon\ \mathtt{male(X)} \vee \mathtt{female(X)}$
  - ◇ $\forall\,\mathtt{person}\ \mathtt{X}\colon\ \neg\,\mathtt{male(X)} \vee \neg\,\mathtt{female(X)}$
- Dies sind Beispiele für Integritätsbedingungen.

Integritätsbedingungen (2)

- Integritätsbedingungen sind geschlossene Formeln.

  Das Datenmodell muss nicht beliebige Formeln zulassen. Eine typische Einschränkung ist die Bereichsunabhängigkeit (siehe unten).

- Eine Menge von Integritätsbedingungen wird als Teil des DB-Schemas spezifiziert.
- Einen DB-Zustand (Interpretation) nennt man zulässig, wenn er alle Integritätsbedingungen erfüllt.

  Im Folgenden betrachten wir nur zulässige DB-Zustande. Dabei sprechen wir dann wieder nur von "DB-Zustand".

Integritätsbedingungen (3)

Schlüssel:

- Objekte werden oft durch eindeutige Datenwerte (Zahlen, Namen) identifiziert.
- Beispiel (Punkte-DB, ER-/Tupelkalkülvariante, Folie 2-37): Es soll keine zwei verschiedene Objekte des Typs `student` mit der gleichen `sid` geben:

  $$\forall\,\mathtt{student}\ \mathtt{X},\ \mathtt{student}\ \mathtt{Y}\colon\ \mathtt{sid(X)} = \mathtt{sid(Y)}\ \rightarrow\ \mathtt{X} = \mathtt{Y}$$

- Alternative, äquivalente Formulierung:

  $$\neg\,\exists\,\mathtt{student}\ \mathtt{X},\ \mathtt{student}\ \mathtt{Y}\colon\ \mathtt{sid(X)} = \mathtt{sid(Y)}\ \wedge\ \mathtt{X} \neq \mathtt{Y}$$

Integritätsbedingungen (4)

Schlüssel, Forts.:

- In der relationalen Variante (Bereichskalkül, Folie 2-33) wird ein Prädikat der Stelligkeit 3 verwendet, um die Studentendaten zu speichern.
- Das erste Argument (SID) identifiziert die Werte der anderen (first name, last name) eindeutig:

$$\forall\, \texttt{int ID},\ \texttt{string F1},\ \texttt{string F2},\ \texttt{string L1},\ \texttt{string L2}\colon$$
$$\texttt{student(ID, F1, L1)} \land \texttt{student(ID, F2, L2)} \rightarrow$$
$$\texttt{F1} = \texttt{F2} \land \texttt{L1} = \texttt{L2}$$

- Da Schlüssel in der Praxis häufig vorkommen, hat jedes Datenmodell eine spezielle Notation dafür.

Integritätsbedingungen (5)

Übung:

- Man betrachte das Schema auf Folie 2-33:
  - ◇ `student(int SID, string FName, string LName)`
  - ◇ `exercise(int ENO, int MaxPoints)`
  - ◇ `result(int SID, int ENO, int Points)`
- Formulieren Sie folgende Integritätsbedingungen:
  - ◇ Die `Points` in `result` sind stets nichtnegativ.
  - ◇ Jede Übungsnummer `ENO` in `result` existiert auch in `exercise` (keine "broken links").

    Dies ist ein Beispiel einer "Fremdschlüsselbedingung".

Anfragen (1)

- Eine Anfrage (Form A) ist ein Ausdruck der Form

$$\{s_1\, X_1, \ldots, s_n\, X_n \mid F\},$$

  wobei F eine Formel bzgl. der gegebenen Signatur Σ und Variablendeklaration $\{X_1/s_1, \ldots, X_n/s_n\}$ ist.

  Auch hier kann es Restriktionen für die möglichen Formeln F geben, vor allem die Bereichsunabhängigkeit (siehe unten).

- Die Anfrage fragt nach allen Variablenbelegungen $\mathcal{A}$ für die Ergebnisvariablen $X_1, \ldots, X_n$, für die die Formel F im gegebenen DB-Zustand $\mathcal{I}$ wahr ist.

  Um sicherzustellen, dass die Variablenbelegungen ausgegeben werden können, sollten die Sorten s_i der Ergebnisvariablen Datentypen sein.

Anfragen (2)

Beispiel I:

- Man betrachte das Schema auf Folie 2-33:
  - ◇ `student(int SID, string FName, string LName)`
  - ◇ `exercise(int ENO, int MaxPoints)`
  - ◇ `result(int SID, int ENO, int Points)`
- Wer hat mindestens 8 Punkte für Hausaufgabe 1?

$$\{\texttt{string FName},\ \texttt{string LName} \mid \exists\, \texttt{int SID},\ \texttt{int P}\colon$$
$$\texttt{student(SID, FName, LName)} \land$$
$$\texttt{result(SID, 1, P)} \land \texttt{P} \geq 8\}$$

Anfragen (3)

Beispiel II:

- Geben Sie alle Ergebnisse von Ann Smith aus:

  $\{$`int ENO`, `int Points` $\mid \exists$ `int SID`:
  `student(SID, 'Ann', 'Smith')` $\wedge$
  `result(SID, ENO, Points)`$\}$

- Wer hat Übung 2 bisher noch nicht eingereicht?

  $\{$`string FName`, `string LName` $\mid$
  $\exists$ `int SID`: `student(SID, FName, LName)` $\wedge$
  $\neg\,\exists$ `int P`: `result(SID, 2, P)`$\}$

Übung:

- Geben Sie die Studenten aus, die 10 Punkte in Übung 1 und 10 Punkte in Übung 2 haben.

Anfragen (4)

- Eine Anfrage (Form B) ist ein Ausdruck der Form

  $$\{t_1, \ldots, t_k\ [s_1\, X_1, \ldots, s_n\, X_n] \mid F\},$$

  wobei F eine Formel und die t_i Terme für die gegebene DB-Signatur Σ und die Variablendeklaration $\{X_1/s_1, \ldots, X_n/s_n\}$ sind.

- In diesem Fall gibt das DBMS die Werte $\langle \mathcal{I}, \mathcal{A}\rangle[t_i]$ der Terme t_i für jede Belegung $\mathcal{A}$ der Ergebnisvariablen $X_1, \ldots, X_n$ aus, so dass $\langle \mathcal{I}, \mathcal{A}\rangle \models F$.

  Diese Form der Anfrage ist vor allem dann praktisch, wenn die Variablen X_i von Sorten sind, die sonst nicht ausgegeben werden können.

Anfragen (5)

Beispiel:

- Man betrachte das Schema auf Folie 2-37:
  - ◇ Sorten `student`, `exercise`.
  - ◇ Funktionen `first_name(student): string`, ...
  - ◇ Prädikat: `solved(student, exercise)`.

    Funktion: `points(student, exercise): int`

- Wer hat mindestens 8 Punkte für Hausaufgabe 1?

  $\{$`S.first_name`, `S.last_name` `[student S]` $\mid$
  $\exists$ `exercise E`:
  `E.eno` $= 1 \wedge$ `solved(S, E)` $\wedge$ `points(S, E)` $\geq 8\}$

Anfragen (6)

- Eine Anfrage (Form C) ist eine geschlossene Formel F.
- Das System gibt "ja" aus, falls $\mathcal{I} \models F$ und "nein" sonst.

Übung:

- Angenommen, Form C ist nicht verfügbar. Kann sie mit Form A oder B simuliert werden?
- Offensichtlich ist Form A Spezialfall von Form B: $\{X_1, \ldots, X_n\ [s_1\, X_1, \ldots, s_n\, X_n] \mid F\}$. Ist es umgekehrt auch möglich, Form B auf Form A zurückzuführen?

Bereichsunabhängikkeit (1)

- Man kann nicht beliebige Formeln als Anfragen verwenden. Einige Formeln würden unendliche Antworten generieren:

$$\{\texttt{int SID} \mid \neg\texttt{student(SID, 'Ann', 'Smith')}\}$$

- Andere Formeln verlangen, dass man unendlich viele Werte für quantifizierte Variablen testet:

$$\exists\, \texttt{int X, int Y, int Z:}\ \ \texttt{X} * \texttt{X} + \texttt{Y} * \texttt{Y} = \texttt{Z} * \texttt{Z}$$

- Ist eine Formel bereichsunabhängig, so genügt es, nur endlich viele Werte für jede Variable zu betrachten. Dann treten obige Probleme nicht auf.

Bereichsunabhängigkeit (2)

- Für einen DB-Zustand (Interpretation) $\mathcal{I}$ und eine Formel F sei der "aktive Bereich" die Menge der Werte aus $\bigcup_{s \in \mathcal{S}} \mathcal{I}[s]$, die
  - ◇ in DB-Relationen in $\mathcal{I}$,
  - ◇ als Wert einer DB-Sorte oder DB-Funktion,
  - ◇ oder als variablenfreier Term (Konstante) in F auftreten.
- Der aktive Bereich ist endlich, enthält aber für bereichsunabhängige Formeln alle relevanten Werte.

Bereichsunabhängigkeit (3)

- Eine Formel F ist bereichsunabhängig, gdw. es für alle möglichen Datenbankzustände ausreicht (im Sinne gleicher Antwortmengen), wenn man für die Variablen in der Formel nur Werte aus dem aktiven Bereich einsetzt.

  Genauer: (1) F muß falsch sein, wenn ein Wert außerhalb des aktiven Bereichs für eine freie Variable eingesetzt wird. (2) Für jede Teilformel $\exists X\colon G$ muß G falsch sein, wenn X einen Wert außerhalb des aktiven Bereichs annimmt. (3) Für jede Teilformel $\forall X\colon G$ muß G wahr sein, wenn X einen Wert außerhalb des aktiven Bereichs annimmt.

Bereichsunabhängigkeit(4)

- Da der aktive Bereich (im Zustand abgespeicherte Werte) endlich ist, können bereichsunabhängige Anfragen in endlicher Zeit ausgewertet werden.
- Zum Beispiel ist die Formel

$$\exists\, \texttt{int}\, X\colon\ X \neq 1$$

  nicht bereichsunabhängig: Der Wahrheitswert ist abhängig von anderen Integers als 1, aber schon im leeren DB-Zustand gibt es solche nicht.

  Die exakte Menge möglicher Werte (Bereich) ist manchmal unbekannt. Es treten nur die Werte auf, die in der DB bekannt sind. Dann ist es günstig, wenn der Wahrheitswert nicht vom Bereich abhängt.

Bereichsunabhängigkeit (5)

- "Bereichsbeschränkung" ist eine syntaktische Bedingung, die Bereichsunabhängigkeit impliziert.

  Für jede Formel definiert man eine Menge beschränkter Variablen im positiven und negativen Kontext.

  Z.B. wenn F eine atomare Formel $p(t_1, \ldots, t_n)$ mit der DB-Relation p ist, dann ist $posres(F) := \{X \in VARS \mid t_i \text{ ist die Variable } X\}$ und $negres(F) := \emptyset$. Für andere atomare Formeln, sind beide Mengen leer, außer wenn F die Form $X = t$ hat, wobei t variablenfrei ist oder eine DB-Funktion als äußerste Funktion hat. Dann gilt $posres(F) := \{X\}$. Ist F gleich $\neg G$, dann $posres(F) := negres(G)$ und $negres(F) := posres(G)$.
  Hat F die Form $G_1 \wedge G_2$, dann $posres(F) := posres(G_1) \cup posres(G_2)$ und $negres(F) := negres(G_1) \cap negres(G_2)$. Etc.

  Eine Formel F ist bereichsbeschränkt, wenn $free(F) = posres(F)$ und für jede Teilformel $\forall X\colon G$ gilt, dass $X \in negres(G)$, und für jede Teilformel $\exists X\colon G$ gilt, dass $X \in posres(G)$.

Bereichsunabhängigkeit (6)

- Bereichsbeschränkung erzwingt, daß jede Variable an eine endliche Menge von Werten gebunden ist.

  Z.B. durch das Vorkommen in einer DB-Relation im positiven Kontext (außer es ist universell quantifiziert, dann im negativen Kontext).

- Bereichsunabhängigkeit ist im allgemeinen unentscheidbar (nicht algorithmisch zu testen). Bereichsbeschränkung ist entscheidbar.

  Z.B. $\exists X\colon X \neq 1 \wedge F$ wäre bereichsunabhängig, wenn F stets falsch ist. Die Konsistenz dieser Formel (auch bereichsbeschränkter Formeln) ist unentscheidbar.

  Trotzdem ist die Bereichsbeschränkung ausreichend allgemein: Z.B. können alle Anfragen der relationalen Algebra in bereichsbeschränkte Formeln übersetzt werden.

Inhalt

Implikation

Definition/Notation:

- Eine Formel oder Menge von Formeln Φ impliziert (logisch) eine Formel oder Menge von Formeln G, gdw. jedes Modell von Φ auch ein Modell von G ist. In diesem Fall schreibt man $\Phi \vdash G$.
- Viele Autoren schreiben $\Phi \models G$.

  Der Unterschied ist wichtig, wenn man über Axiome und Deduktionsregeln spricht. Dann steht $\Phi \vdash G$ für syntaktische Deduktion, $\Phi \models G$ für durch Modelle definierte Implikation. Ein Deduktionssystem ist korrekt und vollständig, wenn beide Relationen übereinstimmen.

Lemma:

- $\Phi \vdash G$ gdw. $\Phi \cup \{\neg\forall(G)\}$ ist inkonsistent.

Äquivalenz (1)

Definition/Lemma:

- Zwei Σ-Formeln oder Mengen von Σ-Formeln F_1 und F_2 sind äquivalent, gdw. sie die gleichen Modelle haben, d.h. für jede Σ-Interpretation $\mathcal{I}$ gilt:
$$\mathcal{I} \models F_1 \iff \mathcal{I} \models F_2.$$
- F_1 und F_2 sind äquivalent gdw. $F_1 \vdash F_2$ und $F_2 \vdash F_1$.

Lemma:

- "Äquivalenz" von Formeln ist eine Äquivalenzrelation, d.h. sie ist reflexiv, symmetrisch und transitiv.
  Dies gilt auch für strenge Äquivalenz (vgl. nächste Folie).

Äquivalenz (2)

Definition/Lemma:

- Zwei (Σ, ν)-Formeln F_1 und F_2 sind stark äquivalent gdw. für jede Σ-Interpretation $\mathcal{I}$ und jede $(\mathcal{I}, \nu)$-Variablendeklaration $\mathcal{A}$:
$$(\mathcal{I}, \mathcal{A}) \models F_1 \iff (\mathcal{I}, \mathcal{A}) \models F_2.$$
- Starke Äquivalenz von F_1 und F_2 wird geschrieben als: $F_1 \equiv F_2$.
- Entsteht G_1 aus G_2 durch Ersetzen der Teilformel F_1 durch F_2, und sind F_1 und F_2 stark äquivalent, so sind auch G_1 und G_2 stark äquivalent.

Einige Äquivalenzen (1)

- Kommutativität (für und, oder, gdw):
  - ◇ $F \wedge G \equiv G \wedge F$
  - ◇ $F \vee G \equiv G \vee F$
  - ◇ $F \leftrightarrow G \equiv G \leftrightarrow F$
- Assoziativität (für und, oder, gdw):
  - ◇ $F_1 \wedge (F_2 \wedge F_3) \equiv (F_1 \wedge F_2) \wedge F_3$
  - ◇ $F_1 \vee (F_2 \vee F_3) \equiv (F_1 \vee F_2) \vee F_3$
  - ◇ $F_1 \leftrightarrow (F_2 \leftrightarrow F_3) \equiv (F_1 \leftrightarrow F_2) \leftrightarrow F_3$

Einige Äquivalenzen (2)

- Distributivgesetz:
  - ◇ $F \wedge (G_1 \vee G_2) \equiv (F \wedge G_1) \vee (F \wedge G_2)$
  - ◇ $F \vee (G_1 \wedge G_2) \equiv (F \vee G_1) \wedge (F \vee G_2)$
- Doppelte Negation:
  - ◇ $\neg(\neg F) \equiv F$
- De Morgan'sche Regeln:
  - ◇ $\neg(F \wedge G) \equiv (\neg F) \vee (\neg G)$.
  - ◇ $\neg(F \vee G) \equiv (\neg F) \wedge (\neg G)$.

Einige Äquivalenzen (3)

- Ersetzung des Implikationsoperators:
  - ◇ $F \leftrightarrow G \;\equiv\; (F \rightarrow G) \wedge (F \leftarrow G)$
  - ◇ $F \leftarrow G \;\equiv\; G \rightarrow F$
  - ◇ $F \rightarrow G \;\equiv\; \neg F \vee G$
  - ◇ $F \leftarrow G \;\equiv\; F \vee \neg G$
- Zusammen mit den De Morgan'schen Regeln bedeutet dies, dass z.B. $\{\neg, \vee\}$ ausreichend sind, weil die anderen logischen Junktoren $\{\wedge, \leftarrow, \rightarrow, \leftrightarrow\}$ durch diese ausgedrückt werden können.

  Wir werden sehen, dass auch nur einer der Quantoren benötigt wird.

Einige Äquivalenzen (4)

- Ersetzung von Quantoren:
  - ◇ $\forall\, s\, X\colon F \;\equiv\; \neg(\exists\, s\, X\colon (\neg F))$
  - ◇ $\exists\, s\, X\colon F \;\equiv\; \neg(\forall\, s\, X\colon (\neg F))$
- Logische Junktoren über Quantoren bewegen:
  - ◇ $\neg(\forall\, s\, X\colon F) \;\equiv\; \exists\, s\, X\colon (\neg F)$
  - ◇ $\neg(\exists\, s\, X\colon F) \;\equiv\; \forall\, s\, X\colon (\neg F)$
  - ◇ $\forall\, s\, X\colon (F \wedge G) \;\equiv\; (\forall\, s\, X\colon F) \wedge (\forall\, s\, X\colon G)$
  - ◇ $\exists\, s\, X\colon (F \vee G) \;\equiv\; (\exists\, s\, X\colon F) \vee (\exists\, s\, X\colon G)$

Einige Äquivalenzen (5)

- Quantoren bewegen: Sei $X \notin \mathit{free}(F)$:
  - ◇ $\forall\, s\, X\colon (F \vee G) \;\equiv\; F \vee (\forall\, s\, X\colon G)$
  - ◇ $\exists\, s\, X\colon (F \wedge G) \;\equiv\; F \wedge (\exists\, s\, X\colon G)$

  Falls zusätzlich $\mathcal{I}[s]$ nicht leer sein kann:
  - ◇ $\forall\, s\, X\colon (F \wedge G) \;\equiv\; F \wedge (\forall\, s\, X\colon G)$
  - ◇ $\exists\, s\, X\colon (F \vee G) \;\equiv\; F \vee (\exists\, s\, X\colon G)$
- Eliminierung überflüssiger Quantoren:

  Ist $X \notin \mathit{free}(F)$ und kann $\mathcal{I}[s]$ nicht leer sein:
  - ◇ $\forall\, s\, X\colon F \;\equiv\; F$
  - ◇ $\exists\, s\, X\colon F \;\equiv\; F$

Einige Äquivalenzen (6)

- Vertauschung von Quantoren: Ist $X \neq Y$:
  - ◇ $\forall\, s_1\, X\colon (\forall\, s_2\, Y\colon F) \;\equiv\; \forall\, s_2\, Y\colon (\forall\, s_1\, X\colon F)$
  - ◇ $\exists\, s_1\, X\colon (\exists\, s_2\, Y\colon F) \;\equiv\; \exists\, s_2\, Y\colon (\exists\, s_1\, X\colon F)$

    Beachte, dass Quantoren verschiedenen Typs ($\forall$ und $\exists$) nicht vertauscht werden können.
- Umbenennung gebundener Variablen:

  Ist $Y \notin \mathit{free}(F)$ und F' entsteht aus F durch Ersetzen jedes freien Vorkommens von X in F durch Y:
  - ◇ $\forall\, s\, X\colon F \;\equiv\; \forall\, s\, Y\colon F'$
  - ◇ $\exists\, s\, X\colon F \;\equiv\; \exists\, s\, Y\colon F'$

Normalformen (1)

Definition:

- Eine Formel F ist in Pränex-Normalform gdw. sie geschlossen ist und die folgende Form hat:

  $$\Theta_1\, s_1\, X_1\ \ldots\ \Theta_n\, s_n\, X_n\!:\ G,$$

  wobei $\Theta_1, \ldots, \Theta_n \in \{\forall, \exists\}$ und G quantor-frei sind.

- Eine Formel F ist in disjunktiver Normalform gdw. sie in Pränex-Normalform ist, und G die Form hat

  $$(G_{1,1} \wedge \cdots \wedge G_{1,k_1})\ \vee \cdots \vee\ (G_{n,1} \wedge \cdots \wedge G_{n,k_n}),$$

  wobei jedes $G_{i,j}$ eine atomare Formel oder eine negierte atomare Formel ist.

Normalformen (2)

Bemerkung:

- Konjunktive Normalform ist wie disjunktive Normalform, aber G muss die folgende Form haben:

  $$(G_{1,1} \vee \cdots \vee G_{1,k_1})\ \wedge \cdots \wedge\ (G_{n,1} \vee \cdots \vee G_{n,k_n}).$$

Theorem:

- Kann man nichtleere Bereiche $\mathcal{I}[s]$ voraussetzen, so kann jede Formel äquivalent in Pränex-Normalform, disjunktive Normalform, und konjunktive Normalform transformiert werden.