Einführung in Datenbanken

Kapitel 5: Tupelkalkül und SQL

Prof. Dr. Stefan Brass

Martin-Luther-Universität Halle-Wittenberg

Wintersemester 2018/19

http://www.informatik.uni-halle.de/~brass/db18/

Inhalt

Relationale Datenbanken (1)

- In relationalen Datenbanken werden die Daten in Tabellen abgespeichert, z.B.

STUDENTEN		
SID	VORNAME	NACHNAME
101	Lisa	Weiss
102	Michael	Grau
103	Daniel	Sommer
104	Iris	Winter

Zur Vereinfachung wurde die Spalte "EMAIL" hier weggelassen.

- Statt Zeilen spricht man formal auch von "Tupeln".

Bei einer Tabelle mit drei Spalten wie im Beispiel entsprechen die Tabellenzeilen Tripeln (3-Tupel), z.B. (101, 'Lisa', 'Weiss').

Relationale Datenbanken (2)

- In der Logik kann man den Zugriff auf Tabellenzeilen auf zwei verschiedene Arten formalisieren:
  - Bereichskalkül (BK): Eine Tabelle mit n Spalten entspricht einem n-stelligen Prädikat: $p(t_1, \ldots, t_n)$ ist wahr gdw.

t_1	$\cdots$	t_n

    eine Zeile der Tabelle ist.
  - Tupelkalkül (TK): Eine Tabelle mit n Spalten entspricht einer Sorte mit n Zugriffsfunktionen, die die Werte der Spalten liefern.

    Alternativ statt Sorte auch einstelliges Prädikat, siehe unten.
- SQL basiert auf dem Tupelkalkül, Datalog auf dem Bereichskalkül.

Relationale Datenbanken (3)

- Im Beispiel würde der Bereichskalkül ein Prädikat `STUDENTEN` einführen.
  - `STUDENTEN(101, 'Lisa', 'Weiss')` wäre wahr.
  - `STUDENTEN(200, 'Martin', 'Mueller')` wäre falsch.

    Im Bereichskalkül laufen Variablen über Datentypen (`int`, `string`).

    Natürlich würde man besser Tabellennamen im Singular wählen.

- Der Tupelkalkül würde eine Sorte `STUDENTEN` einführen, sowie Zugriffsfunktionen `SID`, `VORNAME`, `NACHNAME`.
  - Für ein `X` der Sorte `STUDENTEN` gilt dann: `SID(X) = 101`, `VORNAME(X) = 'Lisa'`, und `NACHNAME(X) = 'Weiss'`.

    Im Tupelkalkül laufen Variablen über ganzen Tupeln.

    Man führt dann die alternative Notation `X.SID` für `SID(X)` ein.

Relationale Datenbanken (4)

- Tatsächlich ist die obige Tupelkalkül-Variante schon auf SQL angepasst.
- Im theoretischen Tupelkalkül kann man Variablen über beliebigen Tupeln deklarieren, z.B.

  $\langle$`SID: int, VORNAME: string, NACHNAME: string`$\rangle$

  Dieser ganze Ausdruck ist eine Sorte. Das Alphabet ist ja ohnehin unendlich, insofern ist das kein Problem. Natürlich kann man eine Abkürzung für den Tupel-Typ einer Tabelle einführen, aber Variablen können auch über Tupel-Typen laufen, die in der Datenbank nicht vorkommen (z.B. für das Anfrage-Ergebnis).
- Die Tabellen entsprechen dann einstelligen Prädikaten. Z.B. kann man mit der Bedingung `STUDENTEN(S)` fordern, dass der Wert einer Variablen `S` der Tupelsorte tatsächlich als Zeile der Tabelle vorkommt.

Relationale Datenbanken (5)

- Vorteil von SQL gegenüber theoretischem TK:
  - Es werden keine Einschränkungen auf den Formeln benötigt, um zu garantieren, dass Variablen nur über einem endlichen Bereich laufen.

    Damit die Formeln effektiv auswertbar sind, muss sichergestellt werden, dass es ausreicht, nur endlich viele Werte für jede Variable auszuprobieren. Dafür wurden die Bedingungen "Bereichsunabhängigkeit" und "Bereichsbeschränkung" eingeführt. In SQL kann man keine Formeln aufschreiben, die dagegen verstoßen würden.
- Nachteil von SQL:
  - Für Ergebnis-Spalten, in denen Werte aus verschiedenen Datenbank-Spalten vorkommen, wird ein `UNION`-Operator benötigt. Logische Formeln reichen hier nicht.

Tupelkalkül — SQL-Variante (1)

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

  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 Allgemeinen können verschiedene Konstanten den gleichen Datenwert bezeichnen, z.B. `0`, `00`, `-0`.

Tupelkalkül — SQL-Variante (2)

STUDENTEN

SID	VORNAME	NACHNAME	EMAIL
101	Lisa	Weiss	···
102	Michael	Grau	NULL
103	Daniel	Sommer	···
104	Iris	Winter	···

BEWERTUNGEN

SID	ATYP	ANR	PUNKTE
101	H	1	10
101	H	2	8
101	Z	1	12
102	H	1	9
102	H	2	9
102	Z	1	10
103	H	1	5
103	Z	1	7

AUFGABEN

ATYP	ANR	THEMA	MAXPT
H	1	ER	10
H	2	SQL	10
Z	1	SQL	14

Tupelkalkül — SQL-Variante (3)

- Aufgrund der Angaben im DB-Schema wird die Signatur $\Sigma_{\mathcal{D}}$ zur Signatur Σ für Anfragen erweitert, und zwar um:
  - Sorten (eine für jede Relation/Tabelle).

    Während die Interpretation der Datentypen fest in das DBMS eingebaut ist, kann die Interpretation der zusätzlichen Sorten durch Einfügungen, Löschungen und Updates verändert werden.
    Dafür müssen diese Sorten einen endlichen Wertebereich haben.
    Diese Sorten werden ja durch Dateien auf der Platte implementiert, während die Datentypen durch Programmcode im DBMS implementiert sind und daher unendliche Wertebereiche möglich wären, z.B. beliebig lange Zeichenketten.
  - einstellige Funktionen, jeweils von einer der neuen Sorten in einen Datentyp (für jede Spalte).

    Dies sind Zugriffsfunktionen für die Attribute/Komponenten der Zeilen/Tupel/Records.

Tupelkalkül — SQL-Variante (4)

- In der Punkte-DB gibt es z.B. die Sorte `STUDENTEN` mit den Funktionen
  - `SID(STUDENTEN): int`
  - `VORNAME(STUDENTEN): string`
  - `NACHNAME(STUDENTEN): string`

    Die Werte der Sorte `STUDENTEN` sind die Tabellenzeilen.
    Jede der Funktionen liefert den Wert der entsprechenden Spalte für die als Argument gegebene Tabellenzeile. Diese Zugriffsfunktionen tun also nichts anderes, als jeweils eine Komponente des Tupels auszuwählen.
- Beispiel für Überladung: `SID(BEWERTUNGEN): int`.

Tupelkalkül — SQL-Variante (5)

- Z.B. enthält $\mathcal{I}[\texttt{STUDENTEN}]$ das Tupel

$$t = (101, \text{'Lisa'}, \text{'Weiss'})$$

- Dann ist $\mathcal{I}[\texttt{SID}](t) = 101$.
- Selbstverständlich müssen die neuen Sorten als endliche Mengen interpretiert werden (ggf. auch leer).

  Man fordert auch, dass es keine zwei verschiedenen Tupel geben kann, die in den Werten aller Zugriffsfunktionen übereinstimmen. Dies ist für die Äquivalenz zum Bereichskalkül wichtig, da dort ein Prädikat nicht zweimal wahr sein kann. In der Praxis (SQL) könnte es tatsächlich solche Tupel geben, die in allen Komponenten übereinstimmen. Fast immer werden aber Schlüssel für eine Relation/Tabelle definiert, und dann kann dieser Fall wieder nicht auftreten.

Inhalt

Variablendeklaration (1)

Definition:

- Sei die Signatur $\Sigma = (\mathcal{S}, \mathcal{P}, \mathcal{F})$ gegeben.
- Eine Variablendeklaration für Σ ist eine partielle Abbildung $\nu : \mathit{VARS} \to \mathcal{S}$, so dass
  - der Definitionsbereich von ν endlich ist, und

    Das ist keine Einschränkung, weil jede Formel nur endlich viele Variablen enthält.
  - $\nu(c)$ für $c \in \mathcal{F}_\epsilon \cap \mathit{VARS}$ undefiniert ist.

    Wenn ein Bezeichner schon als Konstante verwendet wird, kann er nicht gleichzeitig als Variable benutzt werden, weil sonst Vorkommen des Bezeichners in Formeln mehrdeutig wären.
- Variablendeklarationen werden auch in der Form $\nu = \{\mathtt{S/STUDENTEN},\ \mathtt{B/BEWERTUNGEN}\}$ geschrieben.

Variablendeklaration (2)

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

ν	
Variable	Sorte
S	STUDENTEN
B	BEWERTUNGEN

  Jede Variable muss eine eindeutige Sorte haben.

- In SQL werden Variablendeklarationen unter `FROM` definiert:

```
FROM STUDENTEN S, BEWERTUNGEN B
```

- Im Tupelkalkül und in SQL kann man Variablen nur für Tupel-Sorten deklarieren (über Tabellenzeilen laufen lassen).

  Nicht für Datensorten (wie `int`). Sichert endliche Auswertbarkeit von SQL.

Variablendeklaration (3)

Definition:

- Sei ν eine Variablendeklaration, $X \in VARS$, 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.
- In SQL können Variablen durch eine Variable gleichen Namens in Unteranfragen "verschattet" werden.

Terme (1)

- Terme sind syntaktische Konstrukte, die zu einem Wert ausgewertet werden können (z.B. zu einer Zahl, einer Zeichenkette, oder zu einer Zeile in einer Tabelle).
- Es gibt drei Arten von Termen:
  - Konstanten, z.B. `1`, `'abc'`,
  - Variablen, z.B. `X`,
  - zusammengesetzte Terme, bestehend aus Funktionssymbolen angewandt auf Argumentterme, z.B. `NACHNAME(S)`.

    Zusammengesetzte Terme können beliebig weit verschachtelt werden, z.B. `(PUNKTE(B)/MAXPT(A)) * 100)`.
- Terme sollten aus Programmiersprachen bekannt sein. Dort sagt man (Wert-)Ausdruck/Expression statt Term.

Terme (2)

Definition:

- Sei eine Signatur $\Sigma = (\mathcal{S}, \mathcal{P}, \mathcal{F})$ und eine Variablendeklaration ν für Σ gegeben.
- Die Menge $TE_{\Sigma,\nu}(s)$ der Terme der Sorte s bezüglich (Σ, ν) ist folgendermaßen rekursiv definiert:
  - Jede Variable $V \in VARS$ 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:

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

Terme (4)

- Einige Funktionen werden üblicherweise als Infix-Operatoren zwischen ihre Argumente (Operanden) geschrieben, also z.B. `X+1` statt der "offiziellen" Notation `+(X, 1)`.

  Wenn man damit anfängt, muss man 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. "`S.NACHNAME`" für "`NACHNAME(S)`".

Terme (5)

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

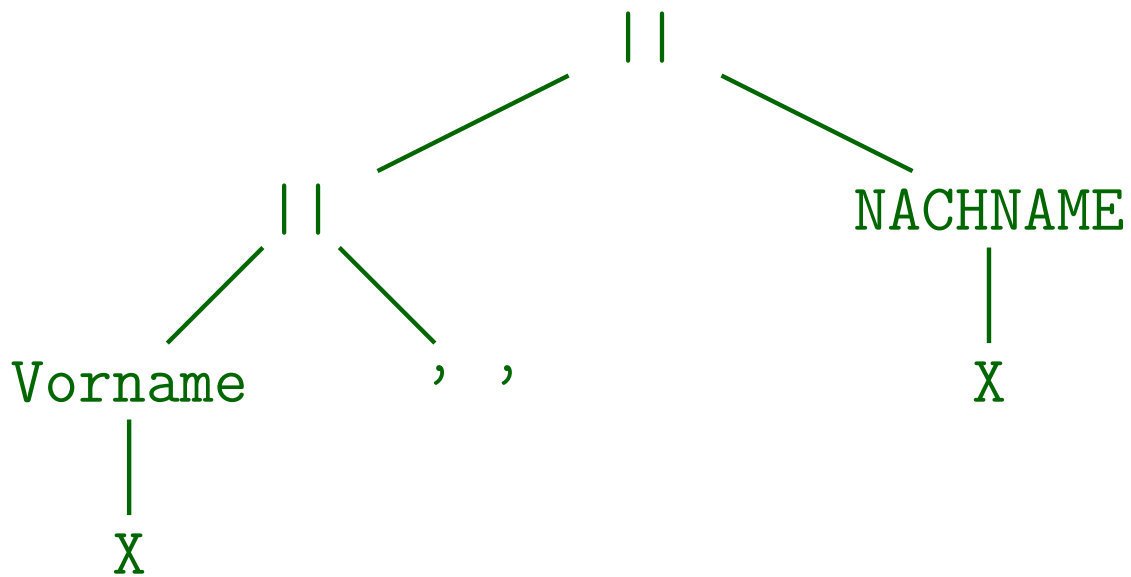
  In Programmiersprachen gibt es manchmal Unterschiede zwischen der "konkreten Syntax" und der "abstrakten Syntax" (Syntaxbaum).
  Die abstrakte Syntax läßt viele Details weg und beschreibt eher die internen Datenstrukturen des Compilers.

- 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):



- Aufgabe: Wie kann man diesen Term mit “||” als Infixoperator und mit Punktnotation schreiben?

  Die meisten zweistelligen Operationssymbole sind “linksassoziativ”, man braucht hier also keine Klammern: `a||b||c` wird verstanden als `(a||b)||c`.

Inhalt

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:
  - Prädikatsymbol, angewandt auf Terme, z.B. `B.PUNKTE > 5`.
  - Gleichung, z.B. `S.NACHNAME = 'Weiss'`.

    Durch die Sonderbehandlung steht "=" unabhängig von der Signatur zur Verfügung und hat eine feste Interpretation als Gleichheit.
  - Logischen Konstante: $\top$ (wahr) und $\bot$ (falsch).

    Diese gibt es nicht in SQL. Man kann z.B. `1=1` für "wahr" schreiben.

Atomare Formeln (2)

Definition:

- Sei eine Signatur $\Sigma = (\mathcal{S}, \mathcal{P}, \mathcal{F})$ und eine Variablendeklaration ν für Σ gegeben.
- Eine atomare Formel ist ein Ausdruck der Form:
  - $p(t_1, \ldots, t_n)$ mit $p \in \mathcal{P}_\alpha$, $\alpha = s_1 \ldots s_n \in \mathcal{S}^*$, $n > 0$ und $t_i \in TE_{\Sigma,\nu}(s_i)$ für $i = 1, \ldots, n$.

    Für einige Prädikate (z.B. <) ist die Infixnotation üblich.
  - p mit $p \in \mathcal{P}_\epsilon$,
  - $t_1 = t_2$ mit $t_1, t_2 \in TE_{\Sigma,\nu}(s)$, $s \in \mathcal{S}$.
  - $\top$ oder $\bot$.
- $AT_{\Sigma,\nu}$ sei die Menge der atomaren Formeln für Σ, ν.

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 der 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 wenn G" (ist G wahr, so auch F)
  - $F \rightarrow G$: "wenn F, dann G"
  - $F \leftrightarrow G$: "F genau dann, wenn G".
  - $\forall\, s\, X : F$: "für alle X (der Sorte s) gilt F".
  - $\exists\, s\, X : F$: "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 äußersten Klammern sind nie 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 $(\ldots)$ 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 $\forall s\ X : F$, $\exists s\ X : F$ 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. $\forall X : F$ statt $\forall s\, X : F$ schreiben (entsprechend für $\exists$).

  Oft ist der Typ der Variablen durch ihre Verwendung eindeutig festgelegt.

- Wenn ein Quantor direkt auf einen anderen Quantor folgt, kann man den Doppelpunkt weglassen, z.B. $\forall X\, \exists Y : F$.

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

Abkürzung für Ungleichheit:

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

Formeln in SQL (1)

- Der Aufbau einer einfachen SQL-Anfrage ist:

  SELECT $t_1, \ldots, t_k$
  FROM $s_1\ X_1, \ldots, s_n\ X_n$
  WHERE F

- Sei ν die Variablendeklaration aus der FROM-Klausel: $\nu = \{X_1/s_1, \ldots, X_n/s_n\}$.
- Sei Σ die Signatur, die sich aus dem Datenbank-Schema und den in das DBMS eingebauten Datentypen ergibt.
- Dann sind $t_1, \ldots, t_k$ Terme bezüglich (Σ, ν).

  Die Ergebnissorte des Terms muss eine Datensorte sein (keine Tupelsorte).

- F ist eine Formel bezüglich (Σ, ν).

  SQL versteht zum Spaltenzugriff nur die Notation `X.A`, nicht `A(X)`.

Formeln in SQL (2)

- Es gibt in SQL nur drei logische Junktoren:
  - `AND` wird für $\wedge$ geschrieben (Konjunktion).
  - `OR` wird für $\vee$ geschrieben (Disjunktion).
  - `NOT` wird für $\neg$ geschrieben (Negation).
- Statt $\exists\, s\, X : F$ schreibt man

```
EXISTS(SELECT *
       FROM   s X
       WHERE  F)
```

  Man kann also testen, ob das Ergebnis der Unteranfrage nicht leer ist.

  Wie bei der logischen Formel kann man in der Bedingung F auch die Variablen der äußeren Anfrage verwenden, weil es eine Formel bezüglich $(\Sigma, \nu\langle X/s\rangle)$ ist.

Formeln in SQL (3)

Beispiel:

- Gesucht sind Studenten (Vorname und Nachname), die Hausaufgabe 1 nicht abgegeben haben.

  D.h. es existiert kein Eintrag in der Bewertungstabelle mit der Nummer (`SID`) dieses Studenten für Hausaufgabe 1 (d.h. `'H'` als Aufgabentyp und `1` als Nr.).

- Logik: $\neg\exists\,\texttt{BEWERTUNGEN B}:\ \texttt{S.SID} = \texttt{B.SID} \wedge \texttt{B.ATYP} = \texttt{'H'} \wedge \texttt{B.ANr} = 1$

- SQL:

```
SELECT S.VORNAME, S.NACHNAME
FROM   STUDENTEN S
WHERE  NOT EXISTS (SELECT *
                   FROM   BEWERTUNGEN B
                   WHERE  S.SID  = B.SID
                   AND    B.ATYP = 'H'
                   AND    B.ANR  = 1)
```

Geschlossenene Formeln

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.

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 (solche Variablen nennt man “freie Variablen”, s.u.).
- Bei geschlossenen Formeln reicht die Interpretation.

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 $(G_1 \wedge G_2)$, $(G_1 \vee G_2)$, etc. hat: $free(F) := free(G_1) \cup free(G_2)$.
  - Wenn F die Form $(\forall\, s\, X :\, G)$ oder $(\exists\, s\, X :\, G)$ hat: $free(F) := free(G) \setminus \{X\}$.

Anfragen

- Geschlossene Formeln werden in Datenbanken als Integritätsbedingungen verwendet.
- Anfragen sind dagegen typischerweise keine geschlossenen Formeln: Man möchte Werte für die freien Variablen ("Antwortvariablen") berechnen, die die Formel wahr machen.
- Beim Bereichskalkül laufen die Variablen über Datenwerte, dort sind Anfragen einfach Formeln mit freien Variablen.
- Beim Tupelkalkül, SQL und einem Kalkül für das ER-Modell laufen die Variablen über Tupel bzw. Objekte, man braucht dann Terme zur Umrechnung in druckbare Datenwerte:

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

Dies entspricht genau SELECT t_1, ..., t_k FROM s_1 X_1, s_n X_n WHERE F.

$t_1, \ldots, t_k$ sind Terme und F eine Formel bzgl. (Σ, ν) mit $\nu := \{X_1/s_1, \ldots, X_n/s_n\}$.

Inhalt

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 $s := \nu(V)$.

  Für Variablen V, die in ν nicht deklariert sind (d.h. ν ist undefiniert für V) ist auch $\mathcal{A}$ undefiniert. Damit ist auch eine Variablenbelegung endlich aufschreibbar, z.B. wie Variablendeklarationen: $\{X_1/d_1, \ldots, X_n/d_n\}$.

Bemerkung:

- D.h. eine Variablenbelegung legt für alle Variablen, die in ν deklariert sind, Werte aus $\mathcal{I}$ fest (passender Sorte).

Variablenbelegung (2)

Beispiel:

- Es sei folgende Variablendeklaration ν gegeben:

Variable	Sorte
X	STUDENTEN

- Eine mögliche Variablenbelegung ist:

X $\longrightarrow$

STUDENTEN			
SID	VORNAME	NACHNAME	EMAIL
101	Lisa	Weiss	...
102	Michael	Grau	NULL
103	Daniel	Sommer	...
104	Iris	Winter	...

Mathematisch: $\mathcal{A}(X) = (101, \text{'Lisa'}, \text{'Weiss'}, \text{'...'}) \in \mathcal{I}[\texttt{STUDENTEN}]$

Variablenbelegung (3)

Definition:

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

Bemerkung:

- Wie bei den Variablendeklarationen braucht man auch bei den Variablenbelegungen die Möglichkeit zur lokalen Veränderung der Abbildung für eine einzelne Variable (zur Behandlung von Quantoren).

Wert eines Terms

Definition:

- Seien 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: $\langle \mathcal{I}, \mathcal{A} \rangle [t] := \mathcal{I}[f, s_1 \ldots s_n](\langle \mathcal{I}, \mathcal{A} \rangle [t_1], \ldots, \langle \mathcal{I}, \mathcal{A} \rangle [t_n])$.

    D.h. man wertet die Argumentterme aus und wendet auf die Ergebnisse die Funktion an, durch die das Funktionssymbol f interpretiert wird.

Inhalt

Wahrheit einer Formel (1)

Definition:

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

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

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

Wahrheit einer Formel (2)

Definition, fortgesetzt:

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

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

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

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

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	$\wedge$	$\vee$	$\leftarrow$	$\rightarrow$	$\leftrightarrow$
f	**f**	**f**	**f**	**w**	**w**	**w**
f	**w**	**f**	**w**	**f**	**w**	**f**
w	**f**	**f**	**w**	**w**	**f**	**f**
w	**w**	**w**	**w**	**w**	**w**	**w**

Beispiel zum Lesen der Tabelle:

Falls $\langle \mathcal{I}, \mathcal{A} \rangle[G_1] = \mathbf{w}$ und $\langle \mathcal{I}, \mathcal{A} \rangle[G_2] = \mathbf{f}$

dann $\langle \mathcal{I}, \mathcal{A} \rangle[(G_1 \wedge G_2)] = \mathbf{f}$.

Wahrheit einer Formel (4)

Definition, fortgesetzt:

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

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

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

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

Modell (1)

Definition:

- Ist $\langle \mathcal{I}, \mathcal{A} \rangle[F] = \mathbf{w}$, so schreibt man auch $\langle \mathcal{I}, \mathcal{A} \rangle \models F$. Dann heißt $\langle \mathcal{I}, \mathcal{A} \rangle$ ein Modell von F.

  Man sagt dann auch $\langle \mathcal{I}, \mathcal{A} \rangle$ erfüllt F bzw. F ist in $\langle \mathcal{I}, \mathcal{A} \rangle$ wahr.

  Viele Autoren wenden den Beriff "Modell" allerdings nur auf Interpretationen allein an (ohne Variablenbelegung), wie im folgenden Punkt beschrieben.

- Sei F eine (Σ, ν)-Formel. Gilt $\langle \mathcal{I}, \mathcal{A} \rangle[F] = \mathbf{w}$ für alle Variablenbelegungen $\mathcal{A}$ (für $\mathcal{I}$ und ν), so schreibt man $\mathcal{I} \models F$ und nennt $\mathcal{I}$ ein Modell von F.

  D.h. freie Variablen werden als $\forall$-quantifiziert behandelt.

  Die Variablenbelegung ist aber irrelevant, falls F eine geschlossene Formel ist.

Modell (2)

Definition:

- Eine Formel F heißt konsistent gdw. es $\mathcal{I}$ und $\mathcal{A}$ gibt mit $\langle \mathcal{I}, \mathcal{A} \rangle \models F$ (d.h. wenn sie ein Modell hat).

  Entsprechend für Mengen von Formeln.

  Manche Autoren nennen eine Formel nur dann konsistent, wenn sie in einer Interpretation $\mathcal{I}$ für alle Variablenbelegungen $\mathcal{A}$ wahr ist. Wenn sie nur für mindestens eine Variablenbelegung wahr ist, würde die Formel "erfüllbar" heißen.

- F ist inkonsistent gdw. F ist nicht konsistent.

  D.h. F ist immer falsch, egal welche Interpretation und Variablenbelegung man nimmt. Mit anderen Worten: F hat kein Modell.

  (Man beachte wieder, dass es in manchen Büchern "unerfüllbar" heißt, und inkonsistent etwas schwächer ist.)

Modell (3)

Definition:

- $\langle \mathcal{I}, \mathcal{A} \rangle \models \Phi$ für eine Menge Φ von Σ-Formeln gdw. $\langle \mathcal{I}, \mathcal{A} \rangle \models F$ für alle $F \in \Phi$.

  Analog wird $\mathcal{I} \models \Phi$, Modell und Konsistenz einer Menge Φ von Formeln definiert: Man fordert die entsprechende Eigenschaft jeweils für alle Formeln $F \in \Phi$.

Definition:

- Eine (Σ, ν)-Formel F nennt man Tautologie gdw. $(\mathcal{I}, \mathcal{A}) \models F$ für alle Σ-Interpretationen $\mathcal{I}$ und (Σ, ν)-Variablenbelegungen $\mathcal{A}$ gilt.

  D.h. Tautologien sind immer wahr.

Antwort auf eine Anfrage

Definition:

- Sei die folgende Anfrage gegeben:

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

  Oder in SQL: `SELECT` t_1 `,` $\ldots$ `,` t_k `FROM` $s_1\ X_1$ `,` $s_n\ X_n$ `WHERE` F.

  $t_1, \ldots, t_k$ sind Terme und F eine Formel bzgl. (Σ, ν) mit $\nu := \{X_1/s_1, \ldots, X_n/s_n\}$.

- Sei $\mathcal{I}$ die Interpretation, die dem Datenbank-Zustand entspricht.

  Der Anteil für die Symbole der Datensignatur $\Sigma_{\mathcal{D}}$ ist natürlich die in das DBMS eingebaute Interpretation $\mathcal{I}_{\mathcal{D}}$.

- Dann ist die Antwort die Menge

$$\{(\langle \mathcal{I}, \mathcal{A} \rangle[t_1], \ldots, \langle \mathcal{I}, \mathcal{A} \rangle[t_k]) \mid \mathcal{A} \text{ ist Variablenbelegung für } \mathcal{I} \text{ und } \nu \text{ mit } \langle \mathcal{I}, \mathcal{A} \rangle \models F\}$$

Inhalt

Implikation

Definition:

- Eine Formel oder Menge von Formeln Φ impliziert (logisch) eine Formel oder Menge von Formeln G gdw. jedes Modell $\langle \mathcal{I}, \mathcal{A} \rangle$ von Φ auch ein Modell von G ist. In diesem Fall schreibt man $\Phi \models G$.

  In der Behandlung von freien Variablen unterscheiden sich die Definitionen verschiedener Autoren. Z.B. würde $X = Y \wedge Y = Z \models X = Z$ nach obiger Definition gelten. Betrachtet man freie Variablen aber als implizit allquantifiziert, so gilt das nicht.

  Beachte: Das Zeichen $\models$ ist überladen. Steht links eine Interpretation bzw. ein Paar aus Interpretation und Variablenbelegung, bedeutet es "ist Modell von" (d.h. die Interpretation macht die Formel rechts wahr). Steht links eine Formelmenge, bedeutet es die logische Implikation: Jedes Modell der Formelmenge links macht die Formel rechts wahr.

Äquivalenz (1)

Definition:

- Zwei (Σ, ν)-Formeln oder Mengen solcher Formeln F_1 und F_2 heißen (logisch) äquivalent gdw. sie die gleichen Modelle haben, d.h. wenn für jede Σ-Interpretation $\mathcal{I}$ und jede $(\mathcal{I}, \nu)$-Variablenbelegung $\mathcal{A}$ gilt:

$$\langle \mathcal{I}, \mathcal{A} \rangle \models F_1 \iff \langle \mathcal{I}, \mathcal{A} \rangle \models F_2.$$

  Man schreibt dann: $F_1 \equiv F_2$.

  Wie schon beim Modellbegriff und der logischen Implikation behandeln manche Autoren freie Variablen als implizit allquantifiziert.

  Bei Datenbanken bezieht sich "Äquivalenz" häufig auf eine gegebene Menge von Integritätsbedingungen: Dann werden nicht beliebige Σ-Interpretationen $\mathcal{I}$ betrachtet, sondern nur solche, die die Integritätsbedingungen erfüllen.

Äquivalenz (2)

Lemma:

- F_1 und F_2 sind äquivalent gdw. $F_1 \models F_2$ und $F_2 \models F_1$.
- "Äquivalenz" von Formeln ist eine Äquivalenzrelation, d.h. sie ist reflexiv, symmetrisch und transitiv.

  Reflexiv: $F \equiv F$.

  Symmetrisch: Wenn $F \equiv G$, dann $G \equiv F$.

  Transitiv: Wenn $F_1 \equiv F_2$ und $F_2 \equiv F_3$, dann $F_1 \equiv F_3$.

- Entsteht G_1 aus G_2 durch Ersetzen der Teilformel F_1 durch F_2, und gilt $F_1 \equiv F_2$, so gilt auch $G_1 \equiv G_2$.
- Wenn $F \models G$, dann $F \wedge G \equiv F$.

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)

- Entfernung der Negation:
  - $\neg(t_1 < t_2) \equiv t_1 \geq t_2$
  - $\neg(t_1 \leq t_2) \equiv t_1 > t_2$
  - $\neg(t_1 = t_2) \equiv t_1 \neq t_2$
  - $\neg(t_1 \neq t_2) \equiv t_1 = t_2$
  - $\neg(t_1 \geq t_2) \equiv t_1 < t_2$
  - $\neg(t_1 > t_2) \equiv t_1 \leq t_2$

  Zusammen mit dem De'Morganschen Gesetz kann man die Negation bis zu den atomaren Formeln herunterschieben, und dann durch Umdrehen der Vergleichsoperatoren eliminieren.
  Das geht auch mit Quantoren, aber man braucht dann $\exists$ und $\forall$.
  Da es in SQL nur $\exists$ gibt, kann man dort $\neg$ vor $\exists$ nicht entfernen.

Einige Äquivalenzen (5)

- Prinzip des ausgeschlossenen Dritten:
  - $F \vee \neg F \equiv \top$ (immer wahr)
  - $F \wedge \neg F \equiv \bot$ (immer falsch)
- Vereinfachung von Formeln mit den logischen Konstanten $\top$ (wahr) und $\bot$ (falsch):
  - $F \wedge \top \equiv F$ $\quad$ $F \wedge \bot \equiv \bot$
  - $F \vee \top \equiv \top$ $\quad$ $F \vee \bot \equiv F$
  - $\neg\top \equiv \bot$ $\quad$ $\neg\bot \equiv \top$

Einige Äquivalenzen (6)

- 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 (7)

- 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 (8)

- 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:
  - $\forall\, s\, X\colon\ F \equiv \forall\, s\, Y\colon\ F'$
  - $\exists\, s\, X\colon\ F \equiv \exists\, s\, Y\colon\ F'$

Einige Äquivalenzen (9)

- Gleichheit ist Äquivalenzrelation:
  - $t = t \equiv \top$ (Reflexivität)
  - $t_1 = t_2 \equiv t_2 = t_1$ (Symmetrie)
  - $t_1 = t_2 \wedge t_2 = t_3 \equiv t_1 = t_2 \wedge t_2 = t_3 \wedge t_1 = t_3$ (Transitivität)
- Verträglichkeit mit Funktions-/Prädikatsymbolen:
  - $f(t_1, \ldots, t_n) = t \wedge t_i = t_i' \equiv$ $f(t_1, \ldots, t_{i-1}, t_i', t_{i+1}, \ldots, t_n) = t \wedge t_i = t_i'$
  - $p(t_1, \ldots, t_n) \wedge t_i = t_i' \equiv$ $p(t_1, \ldots, t_{i-1}, t_i', t_{i+1}, \ldots, t_n) \wedge t_i = t_i'$