Fall 3: `x = x1::xs` $\wedge$ `y = y1::ys` $\wedge$ `x1` $\leq$ `y1`.

Wir schließen:

```
sorted (merge x y)   =  sorted (merge (x1::xs) (y1::ys))
                     =  sorted (x1 :: merge xs y)
                     =  ...
```

Fall 3.1: `xs = []`

Wir schließen:

```
                ...  =  sorted (x1 :: merge [] y)
                     =  sorted (x1 :: y)
                     =  sorted y
                     =  true     :-)
```

Fall 3.2: `xs = x2::xs'` $\wedge$ `x2` $\leq$ `y1`.

Insbesondere gilt: `x1` $\leq$ `x2` $\wedge$ `sorted xs`.

Wir schließen:

```
...  =  sorted (x1 :: merge (x2::xs') y)
     =  sorted (x1 :: x2 :: merge xs' y)
     =  sorted (x2 :: merge xs' y)
     =  sorted (merge xs y)
     =  true   nach Induktionsannahme   :-)
```

Fall 3.3: `xs = x2::xs'` $\wedge$ `x2` $>$ `y1`.

Insbesondere gilt: `x1` $\leq$ `y1` $<$ `x2` $\wedge$ `sorted xs`.

Wir schließen:

```
...  =  sorted (x1 :: merge (x2::xs') (y1::ys))
     =  sorted (x1 :: y1 :: merge xs ys)
     =  sorted (y1 :: merge xs ys)
     =  sorted (merge xs y)
     =  true   nach Induktionsannahme   :-)
```

Fall 4: `x = x1::xs` $\wedge$ `y = y1::ys` $\wedge$ `x1` $>$ `y1`.

Wir schließen:

```
sorted (merge x y)   =   sorted (merge (x1::xs) (y1::ys))
                     =   sorted (y1 :: merge x ys)
                     =   ...
```

Fall 4.1: `ys = []`

Wir schließen:

```
                ...  =   sorted (y1 :: merge x [])
                     =   sorted (y1 :: x)
                     =   sorted x
                     =   true     :-)
```

Fall 4.2: `ys = y2::ys'` $\wedge$ `x1` $>$ `y2`.

Insbesondere gilt: `y1` $\leq$ `y2` $\wedge$ `sorted ys`.

Wir schließen:

```
...  =  sorted (y1 :: merge x (y2::ys'))
     =  sorted (y1 :: y2 :: merge x ys')
     =  sorted (y2 :: merge x ys')
     =  sorted (merge x ys)
     =  true   nach Induktionsannahme   :-)
```

Fall 4.3: `ys = y2::ys'` $\wedge$ `x1` $\leq$ `y2`.

Insbesondere gilt: `y1` $<$ `x1` $\leq$ `y2` $\wedge$ `sorted ys`.

Wir schließen:

```
...  =  sorted (y1 :: merge (x1::xs) (y2::ys'))
     =  sorted (y1 :: x1 :: merge xs ys)
     =  sorted (x1 :: merge xs ys)
     =  sorted (merge x ys)
     =  true   nach Induktionsannahme   :-))
```

Diskussion:

- Wieder steht der Beweis unter dem Vorbehalt, dass alle Aufrufe der Funktionen `sorted` und `merge` terminieren :-)
- Als zusätzliche Technik benötigten wir Fallunterscheidungen über die verschiedenen Möglichkeiten für Argumente in den Aufrufen :-)
- Die Fallunterscheidungen machten den Beweis länglich :-(

  // Der Fall $n = 0$ ist tatsächlich überflüssig,

  // da er in den Fällen 1 und 2 enthalten ist :-)

8 Datalog: Rechnen mit Relationen

Beispiel 1: Das Lehrangebot einer TU

$\Longrightarrow$ Entity-Relationship Diagram

Diskussion:

- Viele Anwendungsbereiche lassen sich mit Hilfe von Entity-Relationship-Diagrammen beschreiben.
- Entitäten im Beispiel: Dozent, Vorlesung, Student.
- Die Menge aller vorkommenden Entitäten d.h. Instanzen lassen sich mit einer Tabelle beschreiben ...

Dozent :

Name	Telefon	Email
Esparza	17204	`esparza@in.tum.de`
Nipkow	17302	`nipkow@in.tum.de`
Seidl	18155	`seidl@in.tum.de`

Vorlesung:

Titel	Raum	Zeit
Diskrete Strukturen	MI 1	Do 12:15-13, Fr 10-11:45
Perlen der Informatik III	MI 3	Do 8:30-10
Einführung in die Informatik II	MI 1	Di 16-18
Optimierung	MI 2	Mo 12-14, Di 12-14

Student:

Matr.nr.	Name	Sem.
123456	Hans Dampf	03
007042	Fritz Schluri	11
543345	Anna Blume	03
131175	Effi Briest	05

Diskussion (Forts.):

- Die Zeilen entsprechen den Instanzen.
- Die Spalten entsprechen den Attributen.
- Annahme: das erste Attribut identifiziert die Instanz

  $\Longrightarrow$ Primärschlüssel

  Folgerung: Beziehungen sind ebenfalls Tabellen ...

liest:

Name	Titel
Esparza	Diskrete Strukturen
Nipkow	Perlen der Informatik III
Seidl	Einführung in die Informatik II
Seidl	Optimierung

hört:

Matr.nr.	Titel
123456	Einführung in die Informatik II
123456	Optimierung
123456	Diskrete Strukturen
543345	Einführung in die Informatik II
543345	Diskrete Strukturen
131175	Optimierung

Mögliche Anfragen:

- In welchen Semestern sind die Studierenden der Vorlesung "Diskrete Strukturen" ?
- Wer hört eine Vorlesung bei Dozent "Seidl" ?
- Wer hört sowohl "Diskrete Strukturen" wie "Einführung in die Informatik II" ?

$\Longrightarrow$ Datalog

Idee: Tabelle $\Longleftrightarrow$ Relation

Eine Relation R ist eine Menge von Tupeln, d.h.

$$R \subseteq \mathcal{U}_1 \times \ldots \times \mathcal{U}_n$$

wobei $\mathcal{U}_i$ die Menge aller möglicher Werte für die i-te Komponente ist. In unserem Beispiel kommen etwa vor:

`int`, `string`, möglicherweise Aufzähldatentypen

// Einstellige Relationen sind Mengen :-)

Relationen können durch Prädikate beschrieben werden ...

Prädikate können wir definieren durch Aufzählung von Fakten ...

... im Beispiel:

```
liest ("Esparza", "Diskrete Strukturen").
liest ("Nipkow", "Perlen der Informatik III").
liest ("Seidl", "Einführung in die Informatik II").
liest ("Seidl", "Optimierung").

hört (123456, "Optimierung").
hört (123456, "Einführung in die Informatik II").
hört (123456, "Diskrete Strukturen").
hört (543345, "Einführung in die Informatik II").
hört (543345, "Diskrete Strukturen").
hört (131175, "Optimierung").
```

Wir können aber auch Regeln benutzen, mit denen weitere Fakten abgeleitet werden können ...

... im Beispiel:

```
hat_Hörer (X,Y) :- liest (X,Z), hört (M,Z), student (M,Y,_).
semester (X,Y) :- hört (Z,X), student (Z,_,Y).
```

- `:-` bezeichnet die logische Implikation "$\Leftarrow$".
- Die komma-separierte Liste sammelt die Voraussetzungen.
- Die linke Seite, der Kopf der Regel, ist die Schlussfolgerung.
- Die Variablen werden groß geschrieben.
- Die anonyme Variable `_` bezeichnet irrelevante Werte :-)

An die Wissensbasis aus Fakten und Regeln können wir jetzt Anfragen stellen ...

... im Beispiel:

```
?- hat_Hörer ("Seidl", Z).
```

- Datalog findet alle Werte für `Z`, für die die Anfrage aus den gegebenen Fakten mit Hilfe der Regeln beweisbar ist :-)
- In unserem Beispiel ist das:

```
Z = "Hans Dampf"
Z = "Anna Blume"
Z = "Effi Briest"
```

Weitere Anfragen:

```
?- semester ("Diskrete Strukturen", X).
   X = 2
   X = 4

?- hört (X, "Einführung in die Informatik II"),
   hört (X, "Diskrete Strukturen").
   X = 123456
   X = 543345
```

Weitere Anfragen:

```
?- semester ("Diskrete Strukturen", X).
   X = 2
   X = 4

?- hört (X, "Einführung in die Informatik II"),
   hört (X, "Diskrete Strukturen").
   X = 123456
   X = 543345
```

Achtung:

Natürlich kann die Anfrage auch gar keine oder mehr als eine Variable enthalten :-)

Ein Beispiel-Beweis:

Die Regel:

```
hat_Hörer (X,Y) :- liest (X,Z), hört (M,Z), student (M,Y,_).
```

gilt für alle `X`, `M`, `Y`, `Z`.

Ein Beispiel-Beweis:

Die Regel:

```
hat_Hörer (X,Y) :- liest (X,Z), hört (M,Z), student (M,Y,_).
```

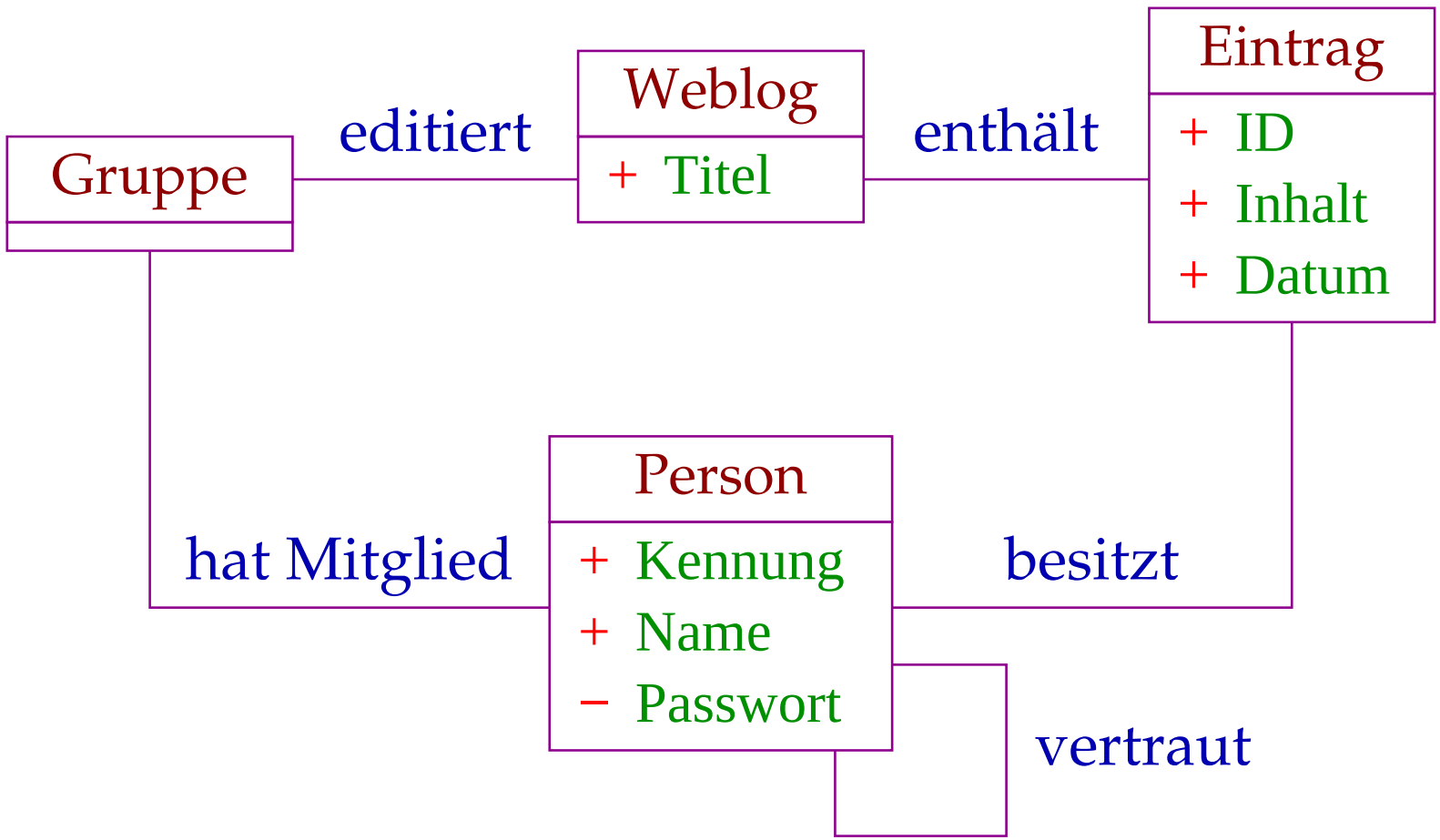
gilt für alle `X`, `M`, `Y`, `Z`. Mit Hilfe der Substitution:

`"Seidl"/X` `"Einführung ..."/Z` `543345/M` `"Anna Blume"/Y`

können wir schließen:

```
liest ("Seidl", "Einführung ...")
                hört (543345, "Einführung ....")
                student (543345, "Anna Blume", 3)
-------------------------------------------------
          hat_Hörer ("Seidl", "Anna Blume")
```

Beispiel 2: Ein Weblog



Aufgabe: Festlegung der Zugriffsberechtigung

- Jedes Mitglied der editierenden Gruppe darf einen neuen Eintrag hinzufügen.
- Nur die Besitzerin eines Eintrags darf ihn löschen.
- Modifizieren darf ihn jeder, dem die Besitzerin traut.
- Lesen darf ihn jedes Mitglied der Gruppe und jeder ihrer mittelbar Vertrauten ...

Spezifikation in Datalog:

```
darf_hinzufügen (X,W) :- editiert (Z,W),
                         hat_Mitglied (Z,X).
darf_löschen (X,E) :- besitzt (X,E).
darf_modifizieren (X,E) :- besitzt (X,E).
darf_modifizieren (X,E) :- besitzt (Y,E),
                           vertraut (Y,X).
darf_lesen (X,E) :- enthält (W,E),
                    darf_hinzufügen (X,W).
darf_lesen (X,E) :- darf_lesen (Y,E),
                    vertraut (Y,X).
```