6. Modellierung von Strukturen

In Kapitel 4 haben wir bereits Graphen und Bäume als Möglichkeiten kennengelernt, mit denen sich Objekte sowie Beziehungen zwischen je 2 Objekten gut modellieren lassen.
In Kapitel 5.2 wurden Verallgemeinerungen davon eingeführt, die so genannten Σ-Strukturen, wobei Σ eine Signatur ist. Abgesehen von Graphen und Bäumen kann man damit beispielsweise auch die natürlichen (oder die rationalen) Zahlen mit arithmetischen Operationen $+, \cdot$ etc. modellieren oder – wie in Kapitel 5.6 gesehen – auch relationale Datenbanken, die z.B. Informationen über Kinofilme und das aktuelle Kinoprogramm enthalten.
In Kapitel 6 werden wir nun zwei weitere Kalküle kennenlernen, mit denen man strukturelle Eigenschaften von Systemen beschreiben kann: das Entity-Relationship-Modell und kontextfreie Grammatiken.

6.1 Das Entity-Relationship-Modell

Das Entity-Relationship-Modell (kurz: ER-Modell) geht zurück auf einen grundlegenden Artikel von P.P. Chen aus dem Jahr 1976:

P.P. Chen: "The Entity-Relationship-Model — Towards a Unified View of Data". ACM Transactions on Database Systems, Band 1, Nr. 1, Seiten 9-36, 1976.

Es wird heute praktisch als Standardmodell für frühe Entwurfsphasen in der Datenbankentwicklung eingesetzt. Darüber hinaus basiert auch die Spezifikationssprache UML ("Unified Modeling Language"), die zur Spezifikation von Strukturen und Beziehungen in Software-Systemen eingesetzt wird, auf dem ER-Modell.

In dieser Vorlesung werden wir einige grundlegende Züge des ER-Modells vorgestellt. Details können Sie z.B. in der Veranstaltung "Datenbanksysteme I" kennenlernen.

Das ER-Modell basiert auf den 3 Grundkonzepten

- Entity: "zu modellierende Informationseinheit" (deutsch: "Objekt", "Ding", "Entität")
- Relationship: zur Modellierung von Beziehungen zwischen Entities (deutsch: "Beziehung", "Relation")
- Attribut: Eigenschaft von einem Entity oder einer Beziehung.

Genauer:

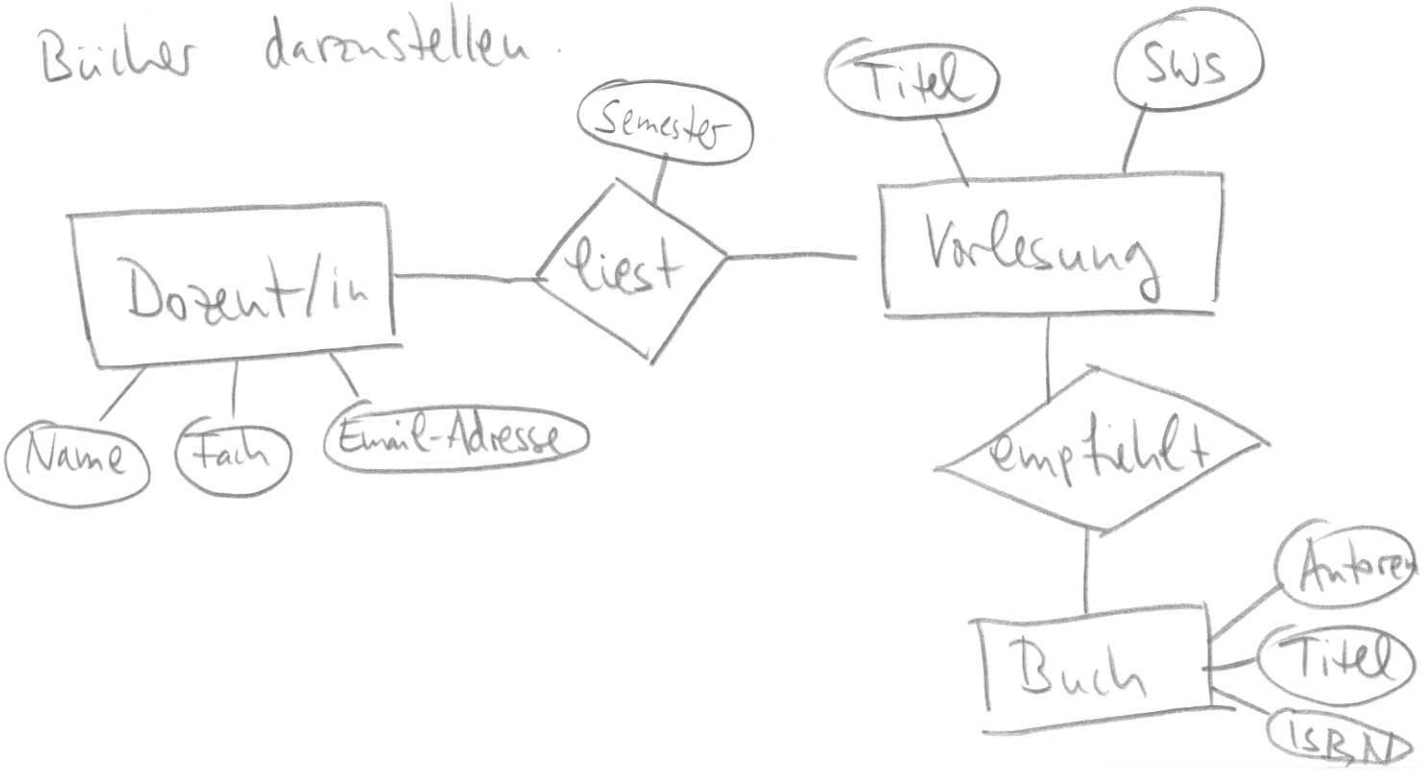
- Entity: Objekt der realen oder der Vorstellungswelt, über das Informationen zu speichern sind
  z.B. eine Vorlesungsveranstaltung, ein Buch oder eine/n Dozent/in.
  Auch Informationen über Ereignisse wie Klausuren können Objekte im Sinne des ER-Modells sein.
- Entity-Typ (bzw. Entity-Menge): eine Zusammenfassung von Entities, die im Modell als "gleichartig" angesehen werden.
  z.B.: "Vorlesung", "Buch", "Dozent/in"

Im Modell steht ein Entity-Typ für die Menge aller in Frage kommenden Objekte dieser Art.

- Relationship: Beziehung zwischen Entities,
  z.B.: welche Dozenten/innen welche Vorlesungen halten
- Attribut: Eigenschaft von Entities oder Relationships,
  z.B. die ISBN eines Buchs, der Titel einer Vorlesung oder die Semester, in denen Vorlesung X von Dozent/in Y gehalten wird.

Beispiel 6.1:

Graphische Darstellung für eine Modellierung im ER-Modell — im Beispiel geht es darum, Vorlesungen, Dozenten und für die Vorlesung empfohlene Bücher darzustellen.



- Entity-Typen werden als Rechtecke dargestellt

(hier: Dozent/in, Vorlesung, Buch)

- Eigenschaften von Entities, sog. Attribute, werden durch Ellipsen dargestellt, die mit dem Rechteck des zugehörigen Entity-Typs verbunden sind

(im Beispiel hat jede/r Dozent/in die Attribute "Name", "Fach" und "Email-Adresse")

Ein Attribut ordnet jedem Entity des entsprechenden Entity-Typs einen Wert zu.

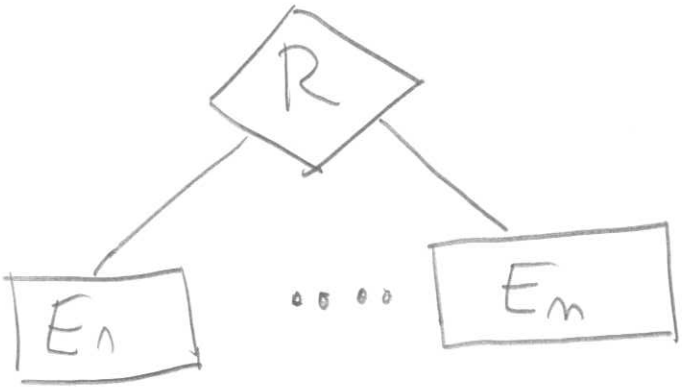
Ein Attribut, dessen Wert jedes Entity eindeutig bestimmt (z.B die ISBN von Büchern), heißt Schlüsselattribut.
Um Schlüsselattribute im ER-Modell explizit zu kennzeichnen, werden sie unterstrichen.
Auch mehrere Attribute zusammen können einen Schlüssel bilden, z.B.

- Typen von Relationships, sog. Relationen-Typen, werden durch Rauten dargestellt, die mit den betreffenden Entity-Typen durch Striche verbunden sind.
(z.B. ist in Bsp 6.1 "liest" ein Relationen-Typ, der angibt, welche/r Dozent/in welche Vorlesung liest).

Allgemein gilt: Ein Relationen-Typ modelliert Beziehungen zwischen den Entities der betroffenen Entity-Typen.

Ein n-stelliger Relationen-Typ R (für $n \geq 2$) verknüpft Entities aus n Entity-Typen $E_1, \ldots, E_n$.

Er wird graphisch repräsentiert durch

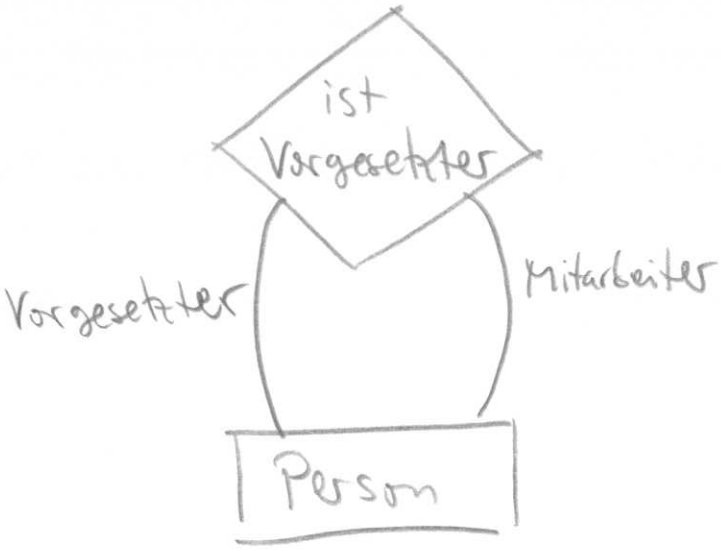


Eine konkrete Ausprägung des Relationen-Typs R ist eine Menge von n-Tupeln $(e_1, \ldots, e_n)$, wobei für jedes $i \in \{1, \ldots, n\}$ gilt: e_i ist ein Entity des Entity-Typs E_i.

- Auch Relationen-Typen können Attribute haben (z.B. hat der Relationen-Typ "liest" in Bsp 6.1 ein Attribut "Semester", das angibt, in welchen Semestern Dozent/in X die Vorlesung Y hält).

Allgemein gilt: Ein Attribut ordnet jedem Tupel des entsprechenden Relationen-Typs einen Wert zu.
Beispielsweise ordnet das Attribut "Semester" jedem Tupel (X,Y) der "liest"-Relation die Liste aller Semester zu, in denen Dozent/in X die Vorlesung Y hält.

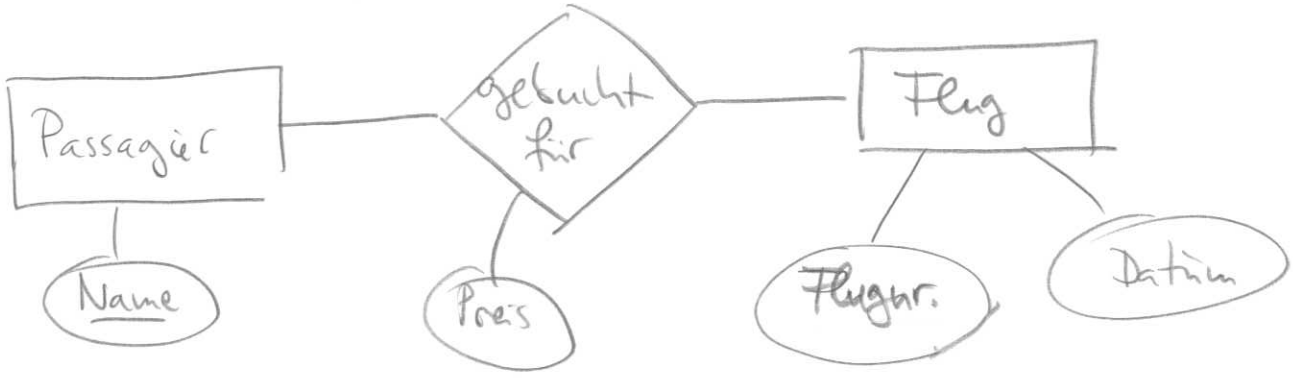
- Für manche Relationen-Typen wird aus ihrem Namen und der graphischen Darstellung zunächst nicht klar, welche Bedeutung die einzelnen Entity-Typen in der Relation haben — insbes. wenn ein Entity-Typ mehrfach am Relationen-Typ beteiligt ist. Es können dann Rollennamen vergeben werden, etwa um die Beziehung "Person X ist Vorgesetzter von Person Y" darzustellen:



Beispiel 6.2

Man beachte die Auswirkungen von Modellierungsentscheiden beim Entwickeln eines ER-Modells:

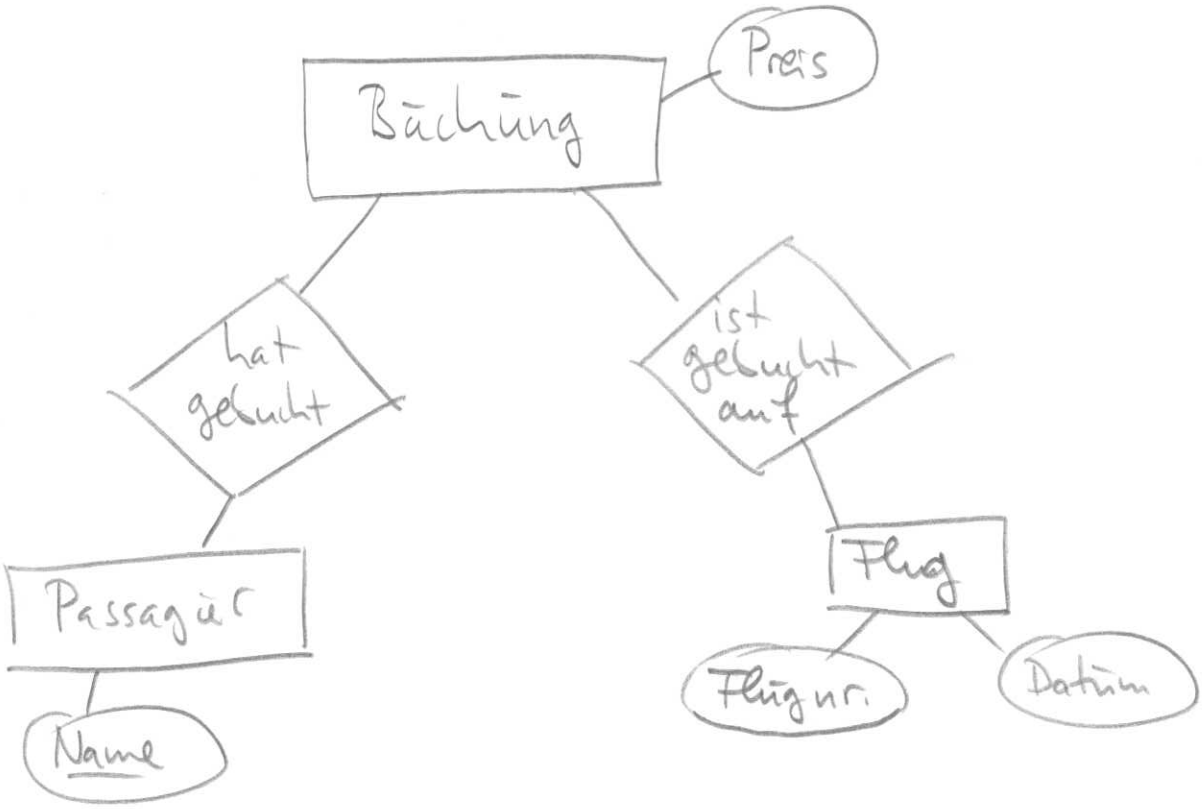
Nutzt ein Reisebüro das ER-Modell



so besteht eine konkrete Ausprägung des Relationen-Typs "gebucht für" aus einer Menge von Tupeln (X, Y), die angibt, dass Person X ein Ticket für Flug Y gebucht hat – und zwar zum Preis $Preis(X, Y)$. Insbesondere heißt dies aber, dass

Passagier X für Flug Y nicht zwei verschiedene Buchungen getätigt haben kann.

Wenn man solche "Mehrfachbuchungen" zulassen will, kann man das folgende ER-Modell benutzen:

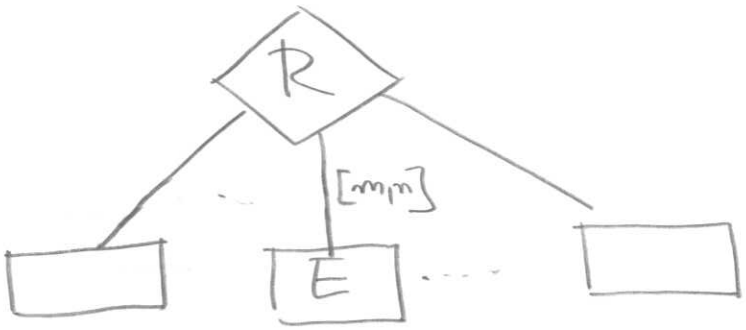


Ein weiterer Bestandteil von ER-Modellen

Kardinalität von Relationen-Typen:

Relationen-Typen in der Form, wie wir sie bisher eingeführt haben, sagen über konkrete Ausprägungen nur aus, dass einige Entities aus den beteiligten Entity-Typen in der angegebenen Beziehung stehen können. Oft will man aber genauere Angaben (bzw. Einschränkungen) machen — z.B., dass jeder Angestellte durch eine Relation des Relationen-Typs "arbeitet in" mit genau einer Abteilung verbunden ist. Dies kann graphisch folgendermaßen dargestellt werden:

Allgemein besagt ein Relationen-Typ der Form



dass für jede konkrete Ausprägung dieses Typs gelten muss:
Jedes Entity e der konkreten Ausprägung des Entity-Typs E kommt in mindestens m und höchstens n Tupeln vor.

Spezialfälle für [m,n]:

- [1,1] bedeutet: "in genau einem Tupel"
- [0,1] bedeutet: "in höchstens einem Tupel"
- [0,*] bedeutet: "in beliebig vielen Tupeln"
  Die Angabe [0,*] wird oft auch einfach weggelassen.

Kurznotation für 2-stellige Relationen-Typen:

bedeutet

d.h.: "jedes Entity a des Typs A kommt in höchstens einem Tupel von R vor, und jedes Entity b des Typs B kommt in beliebig vielen Tupeln von R vor."

Beispiel 6.3:

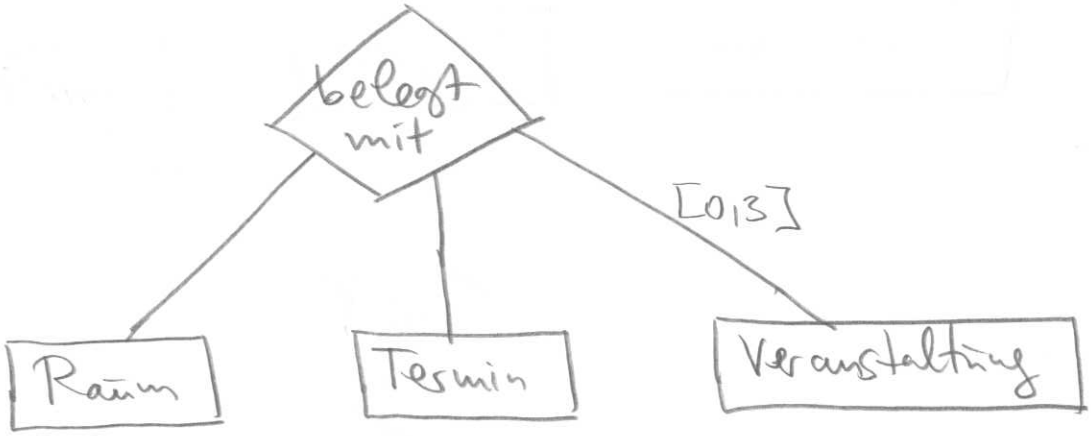
(a)

bedeutet: "Jedes Buch wird von mindestens einer Person geschrieben"

(b)

bedeutet: "Jeder Termin im Stundenplan ist mit höchstens einer Veranstaltung belegt"

(c)



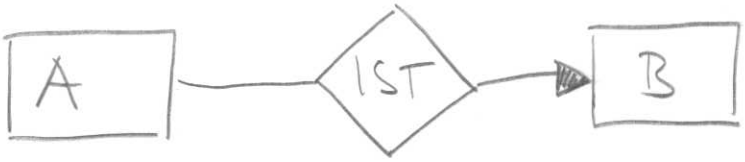
bedeutet: "Jede Veranstaltung wird höchstens dreimal (pro Woche) angeboten

Noch ein Bestandteil von ER-Modellen:

Die IST-Beziehung (englisch "is-a"):

Der spezielle Relationen-Typ IST definiert eine Spezialisierungs-Hierarchie.

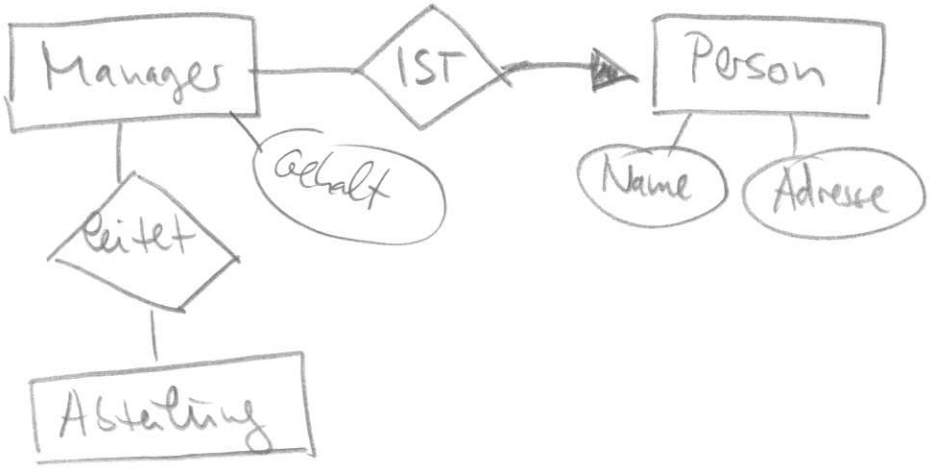
Graphische Darstellung:



Bedeutung: Einige Entities des Typs B sind auch Entities des Typs A
(d.h. A ist eine Spezialisierung des Typs B).

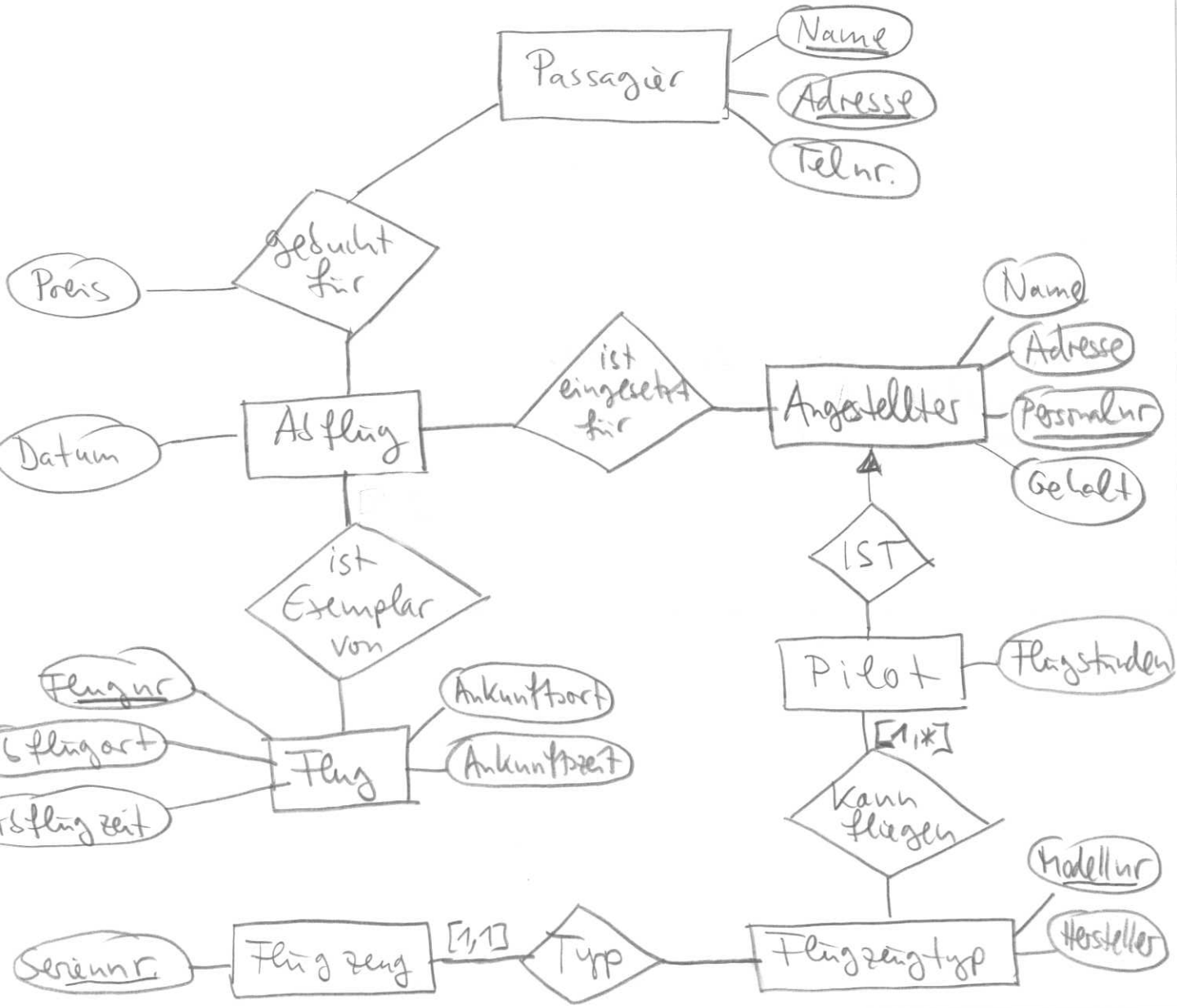
Andere Formulierung: Jedes Entity des Typs A ist auch ein Entity des Typs B.

Beispiel:



Allgemein gilt:
Die Entities des Typs A "erben" alle Attribute von B und können außerdem noch weitere Attribute haben, die spezielle "A-Eigenschaften" beschreiben.
Auch Schlüsselattribute werden als solche geerbt.

Beispiel 6.4: Ein umfangreiches ER-Modell, das einige Aspekte einer Fluggesellschaft modelliert:



In diesem ER-Modell wird u.a. folgendes modelliert:

1) Es kommt eine IST-Spezialisierung vor, die besagt, dass Piloten spezielle Angestellte sind.
Dies wird insbes. benötigt, um den Relationen-Typ "kann fliegen" hinreichend präzise formulieren zu können und das Attribut "Flugstunden" nicht allen Angestellten zuordnen zu müssen.

2) Entities aller hier aufgeführten Typen (bis auf Abflug) werden durch Schlüsselattribute eindeutig identifiziert. Bei den Passagieren wird angenommen, dass Name und Adresse den jeweiligen Passagier eindeutig festlegen. Die Namen der übrigen Schlüsselattribute deuten an, dass man jeweils eine Nummerierung eingeführt hat, um die Eindeutigkeit zu erreichen (z.B. Personalnr, Flugnr etc).

3) Der Relationen-Typ "Typ" verbindet konkrete Flugzeuge mit Flugzeugtypen, indem sie jedem Flugzeug genau einen Flugzeugtypen zuordnet.
Solche Unterscheidungen zwischen "Typ" und "Exemplar" werden oft in Modellierungen verwendet und sind wichtig, damit Relationen und Attribute sachgerecht zugeordnet werden können. Beispielsweise sind die

Fähigkeiten eines Piloten dadurch bestimmt, welche Flugzeugtypen er fliegen kann – und nicht durch die konkreten Flugzeuge(templare), die er fliegen kann.

Vorsicht: Beim ersten Hinsehen mag es verlockend erscheinen, Typ-Exemplar-Beziehungen durch die IST-Spezialisierung zu modellieren.
Dies ist aber ein schwerer Entwurfsfehler:
"Typen" und "Exemplare" bezeichnen verschiedenartige Entity-Typen, zwischen denen i.d.R. keine Teilmengen-Beziehung (wie bei der IST-Spezialisierung) bestehen kann.

6.2 Kontextfreie Grammatiken

Kontextfreie Grammatiken (kurz: KFGs)
eignen sich besonders gut zur Modellierung von
beliebig tief geschachtelten baum-artigen Strukturen.

KFGs können gleichzeitig

- hierarchische Baumstrukturen und
- Sprachen und deren textuelle Notation

spezifizieren.

KFGs werden z.B. angewendet zur Definition von

- Programmen einer Programmiersprache
  und deren Struktur, z.B. Java, C, Pascal
  (KFGs spielen z.B. beim "Compilerbau" eine wichtige Rolle)
- Datenaustauschformaten, d.h. Sprachen als
  Schnittstelle zwischen Software-Werkzeugen,
  z.B. HTML, XML
- Bäumen zur Repräsentation strukturierter Daten, z.B. XML
- Strukturen von Protokollen beim Austausch von
  Nachrichten zwischen Prozessen oder Geräten

KFGs sind ein grundlegender Kalkül für die formale
Definition von Sprachen eingesetzt wird

In dieser Veranstaltung werden nur die Grundbegriffe und einige Beispiele vorgestellt; im Detail werden KFGs in der Veranstaltung "GL-2" behandelt.

Definition des Begriffs "Kontextfreie Grammatik":

Es gibt 2 Sichtweisen auf KFGs:

1) Eine KFG ist ein spezielles Ersetzungssystem. Seine Regeln geben an, auf welche Art man ein Symbol durch eine Folge von Symbolen ersetzen kann. Auf diese Weise definiert eine KFG eine Sprache, d.h. eine Menge von Worten über einem bestimmten Alphabet, die mit den durch die KFG gegebenen Regeln erzeugt werden können

2) Gleichzeitig definiert eine KFG eine Menge von Baumstrukturen, die sich durch schrittweises Anwenden der Regeln erzeugen lassen.

Für die Modellierung von Strukturen ist die zweite Sichtweise besonders interessant. Aber es ist oft sehr nützlich, dass dasselbe Kalkül auch gleichzeitig

eine textuelle Notation für die Baumstrukturen liefern kann und dass Eigenschaften der zugehörigen Sprache untersucht werden können.

Definition 6.5: (KFG)

Eine kontextfreie Grammatik $G = (T, N, S, P)$ besteht aus

- einer endlichen Menge T, der so genannten Menge der Terminalsymbole (die Elemente aus T werden auch Terminale genannt)
- einer endlichen Menge N, der sog. Menge der Nichtterminalsymbole (die Elemente aus N werden auch Nichtterminale genannt).

  Die Mengen T und N sind disjunkt, d.h. $T \cap N = \emptyset$.

  Die Menge $V := T \cup N$ heißt Vokabular; die Elemente in V nennt man auch Symbole.
- einem Symbol $S \in N$, dem sog. Startsymbol
- einer endlichen Menge $P \subseteq N \times V^*$, der sog. Menge der Produktionen.

  Für eine Produktion $(A, x) \in P$ schreiben wir meistens $A \to x$ (bzw. $A ::= x$).

Beispiel 6.6:

Als erstes Beispiel betrachten wir eine KFG für "Wohlgeformte Klammerausdrücke":

$G_K := (T, N, S, P)$ mit

- $T := \{ (,) \}$, d.h.

  G_K hat als Terminale die Symbole "(" ("öffnende Klammer") und ")" ("schließende Klammer")

- $N := \{$ Klammerung, Liste $\}$, d.h.

  G_K hat als Nichtterminale die Symbole "Klammerung" und "Liste"

- $S :=$ Klammerung, d.h.

  "Klammerung" ist das Startsymbol

- $P := \{$ Klammerung $\rightarrow$ (Liste),
  Liste $\rightarrow$ Klammerung Liste,
  Liste $\rightarrow \varepsilon$ $\}$

(zur Erinnerung: ε bezeichnet das leere Wort).

Bedeutung der Produktionen / Semantik von KFGs

Jede Produktion einer KFG, etwa die Produktion
$A \to x$ kann man auffassen als

- "Strukturregel", die besagt "Ein A besteht aus x"

oder als

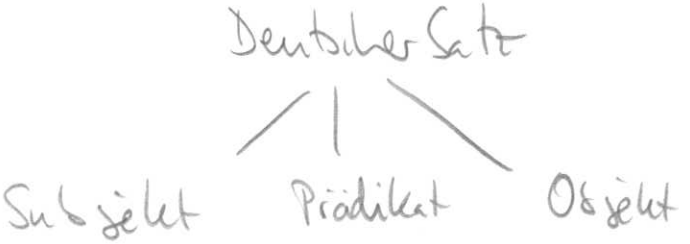
- "Ersetzungsregel", die besagt "A kann man durch x ersetzen"

Beispielsweise kann man die Produktion

DeutscherSatz → Subjekt Prädikat Objekt

verstehen als Aussage, die besagt:
"Ein DeutscherSatz ist aufgebaut aus Subjekt Prädikat Objekt"

Graphische Darstellung:



Das Grundkonzept für die Anwendung von Produktionen einer KFG ist die "Ableitung":

Definition 6.7: (Ableitung)

Sei $G = (T, N, S, P)$ eine KFG.

Falls $A \to x$ eine Produktion in P ist und $u \in V^*$ und $v \in V^*$ beliebige Worte über der Symbolmenge $V = T \cup N$ sind,

so schreiben wir

$$uAv \underset{G}{\Rightarrow} uxv \quad \left(\text{bzw kurz: } uAv \Rightarrow uxv\right)$$

und sagen, dass uAv in einem Ableitungsschritt zu uxv umgeformt werden kann.

Eine Ableitung ist eine Folge von hintereinander angewendeten Ableitungsschritten.

Für Worte $w \in V^*$ und $w' \in V^*$ schreiben wir

$$w \underset{G}{\Rightarrow}^* w' \quad (\text{bzw kurz: } w \Rightarrow^* w'),$$

um auszusagen, dass es eine endliche Folge von Ableitungsschritten gibt, die w zu w' umformt.

Spezialfall: diese Folge darf auch aus 0 Ableitungsschritten bestehen, d.h. f.a. $w \in V^*$ gilt: $w \Rightarrow^* w$.

Beispiel 6.8

Sei G_K die "Klammer-Grammatik" aus Bsp 6.6:

Beispiele für einzelne Ableitungsschritte:

- (Liste) $\Rightarrow$ (Klammerung Liste)
- ((Liste) Liste) $\Rightarrow$ (() Liste)

Ein Beispiel für eine Ableitung in G_K:

Klammerung
$\Rightarrow$ (Liste)
$\Rightarrow$ (Klammerung Liste)
$\Rightarrow$ (Klammerung Klammerung Liste)
$\Rightarrow$ (Klammerung (Liste) Liste)
$\Rightarrow$ ((Liste) (Liste) Liste)
$\Rightarrow$ (() (Liste) Liste)
$\Rightarrow$ (() () Liste)
$\Rightarrow$ (() ())

In jedem Schritt wird jeweils eine Produktion auf ein Nichtterminal der vorangehenden Symbolfolge angewandt. Obige Kette von Ableitungsschritten zeigt, dass

Klammerung $\Rightarrow^*$ (() ())

Definition 6.9 (Sprache einer KFG)

Sei $G = (T, N, S, P)$ eine KFG.

Die von G erzeugte Sprache $L(G)$ ist die Menge aller Worte über dem Alphabet T, die aus dem Startsymbol S ableitet werden können. D.h.:

$$L(G) := \{ w \in T^* : S \overset{*}{\underset{G}{\rightarrow}} w \}$$

Beispiel 6.10

Die KFG G_K aus Bsp. 6.6 definiert die Sprache $L(G_K)$, die aus allen "wohlgeformten Klammerausdrücken" besteht, die von einem Klammerpaar umschlossen werden.

Beispielsweise gehören folgende Worte zu $L(G_K)$:

$()$, $(())$, $(()())$, $((()())())$,

aber die Worte $)($, $((()$, $(()()$, $(Liste)$

gehören nicht zu $L(G_K)$.

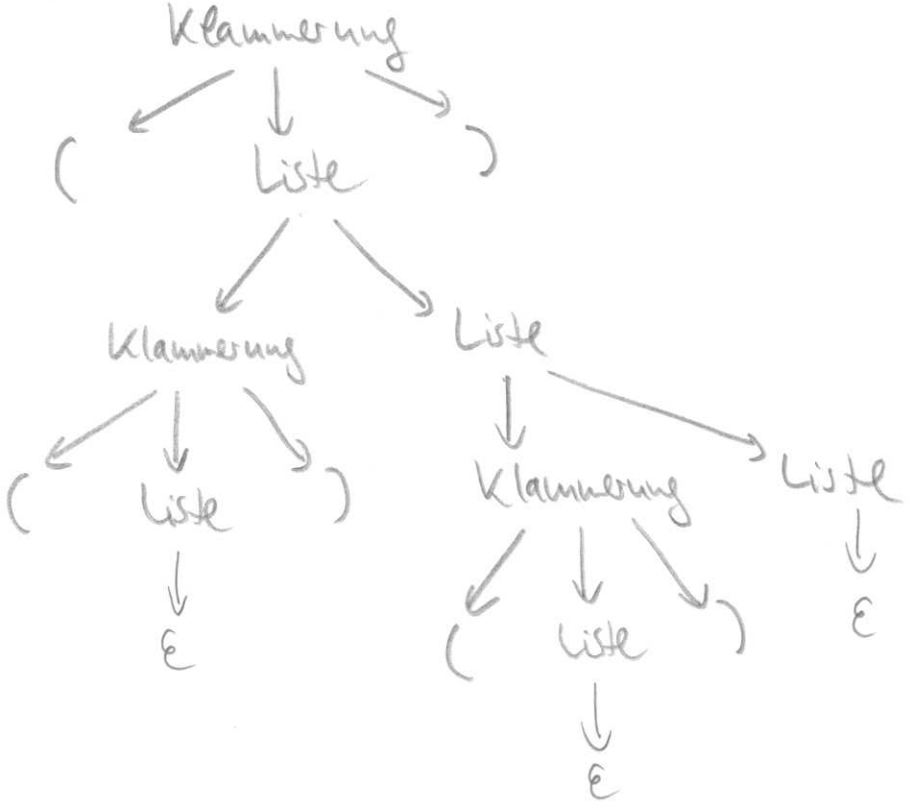
Beispiel 6.11:

Sei G_K die "Klammer-Grammatik" aus Bsp 6.6.

Die Ableitung

$$\text{Klammerung} \Rightarrow (\text{Liste}) \Rightarrow (\text{Klammerung Liste})$$
$$\Rightarrow ((\text{Liste})\ \text{Liste}) \Rightarrow (()\ \text{Liste})$$
$$\Rightarrow (()\ \text{Klammerung Liste}) \Rightarrow (()\ \text{Klammerung})$$
$$\Rightarrow (()\ (\text{Liste})) \Rightarrow (()())$$

wird durch den folgenden Ableitungsbaum dargestellt:



Ableitungsbäume:

Sei $G = (T, N, S, P)$ eine KTG.

Jede Ableitung $S \Rightarrow_G^* w$ lässt sich als gerichteter Baum darstellen, bei dem

- jeder Knoten mit einem Symbol aus $T \cup N \cup \{\varepsilon\}$ markiert ist und
- bei dem die Kinder jedes Knotens eine festgelegte Reihenfolge haben (in der Zeichnung eines Ableitungsbaums werden von links nach rechts zunächst das "erste Kind" dargestellt, dann das zweite, dritte etc.)

Die Wurzel des Baums ist mit dem Startsymbol S markiert. Jeder Knoten mit seinen Kindern repräsentiert die Anwendung einer Produktion aus P: Die Anwendung einer Produktion der Form $A \to x$ (mit $A \in N$ und $x \in V^*$) wird im Ableitungsbaum repräsentiert durch einen Knoten, der mit dem Symbol A markiert ist und der $|x|$ viele Kinder hat, so dass das i-te Kind mit dem i-ten Symbol von x markiert ist (f.a. $i \in \{1, \ldots, |x|\}$. Spezialfall: Ist x das leere Wort ε, so wird eine Anwendung der Produktion $A \to \varepsilon$ repräsentiert durch einen mit A markierten Knoten, der genau ein Kind hat, das mit ε markiert ist.

Beachte:

Ein Ableitungsbaum kann mehrere Ableitungen repräsentieren

Beispielsweise repräsentiert der obige Ableitungsbaum auch die Ableitung

Klammerung $\Rightarrow$ (Liste) $\Rightarrow$ (Klammerung Liste)

$\Rightarrow$ (Klammerung Klammerung Liste)

$\Rightarrow$ ((Liste) Klammerung Liste)

$\Rightarrow$ ((Liste) (Liste) Liste)

$\Rightarrow$ (() (Liste) Liste) $\Rightarrow$ (() () Liste)

$\Rightarrow$ (() ()),

in der gegenüber der ursprünglichen Ableitung aus Bsp 6.11 einige Ableitungsschritte vertauscht sind.

Im Ableitungsbaum wird von der konkreten Reihenfolge, in der die einzelnen Ableitungsschritte vorkommen, abstrahiert.

Im Folgenden betrachten wir einige weitere Beispiele für kontextfreie Grammatiken.

Beispiel 6.12 (Aussagenlogik)

Wir konstruieren eine KFG

$$G_{AL} = (T, N, S, P),$$

deren Sprache $L(G_{AL})$ gerade die Menge aller aussagenlogischen Formeln ist, in denen nur Variablen aus $\{V_0, V_1, V_2\}$ vorkommen:

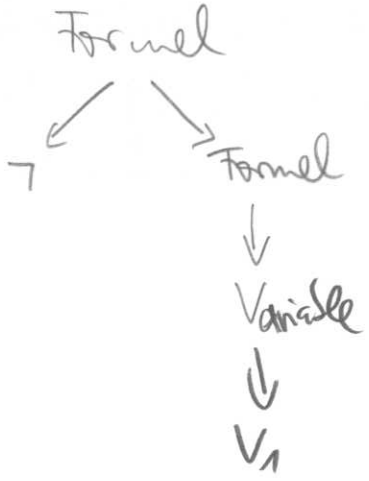
- Terminalsymbole
  $T := \{ V_0, V_1, V_2, 0, 1, \neg, \wedge, \vee, \rightarrow, \leftrightarrow, (,) \}$

- Nichtterminalsymbole
  $N := \{\text{Formel}, \text{Variable}, \text{Junktor}\}$

- Startsymbol $S := \text{Formel}$

- Produktionen $P := \{$
  $\text{Formel} \to 0$, $\text{Formel} \to 1$, $\text{Formel} \to \text{Variable}$
  $\text{Formel} \to \neg\,\text{Formel}$,
  $\text{Formel} \to (\,\text{Formel}\ \text{Junktor}\ \text{Formel}\,)$
  $\text{Variable} \to V_0$, $\text{Variable} \to V_1$, $\text{Variable} \to V_2$,
  $\text{Junktor} \to \wedge$, $\text{Junktor} \to \vee$, $\text{Junktor} \to \rightarrow$, $\text{Junktor} \to \leftrightarrow$ $\}$

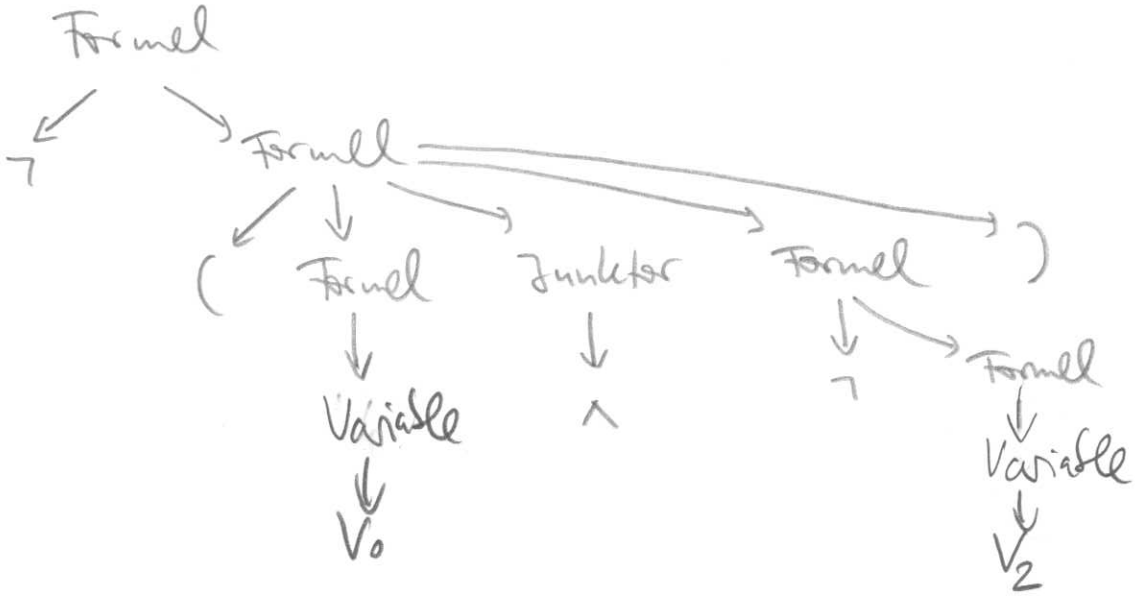
Beispiele für Ableitungsbäume:



Dieser Ableitungsbaum repräsentiert die Ableitung

Formel ⇒ ¬Formel ⇒ ¬Variable
⇒ ¬V1

Das durch diese(n) Ableitungsbaum) erzeugte Wort in der Sprache L(GAL) ist die Formel ¬V1



Dieser Ableitungsbaum repräsentiert die Ableitung

Fomel ⇒ ¬Formel ⇒ ¬(Formel Junktor Formel)

⇒ ¬(V0 Junktor Formel) ⇒ ¬(V0 ∧ Formel)

⇒ ¬(V0 ∧ ¬Formel) ⇒ ¬(V0 ∧ ¬V2)

Das durch diese(n) Ableitung(sbaum) erzeugte Wort in der Sprache L(GAL) ist die Formel ¬(V0 ∧ ¬V2)

Beispiel 6.13 (Menü-Strüktur)

In der graphischen Benützungsoberfläche von vielen Software-Systemen werden oftmals "Menüs" verwendet. Ein Menü besteht aus einem Menünamen und einer Folge von Einträgen. Jeder einzelne Eintrag besteht dabei aus einem Operationsnamen oder selbst wieder einem Menü.

Beispiel:

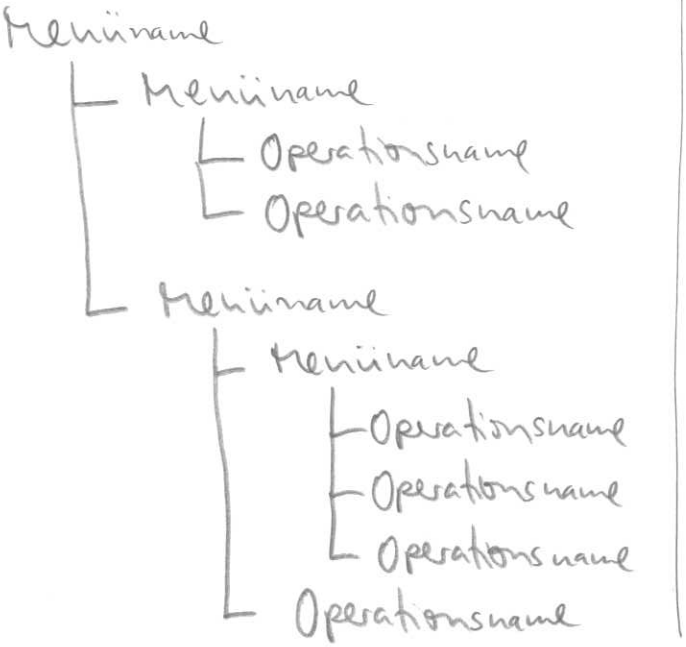
Zur Spezifizierung der Grund-Strüktur solcher Menüs kann man folgende Grammatik $G_{Menü}$ verwenden:

$$G_{Menü} = (T, N, S, P) \quad \text{mit}$$

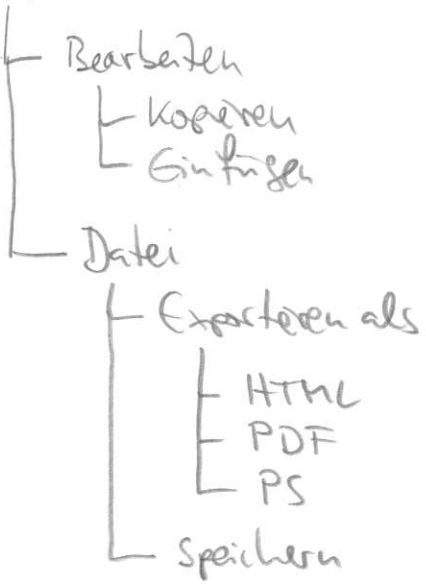
- $T := \{$ Menüname, Operationsname $\}$
- $N := \{$ Menü, Eintragfolge, Eintrag $\}$
- $S :=$ Menü

- $P :=$ { Menü → Menüname Eintragsfolge,
  Eintragsfolge → Eintrag,
  Eintragsfolge → Eintrag Eintragsfolge,
  Eintrag → Operationsname
  Eintrag → Menü }

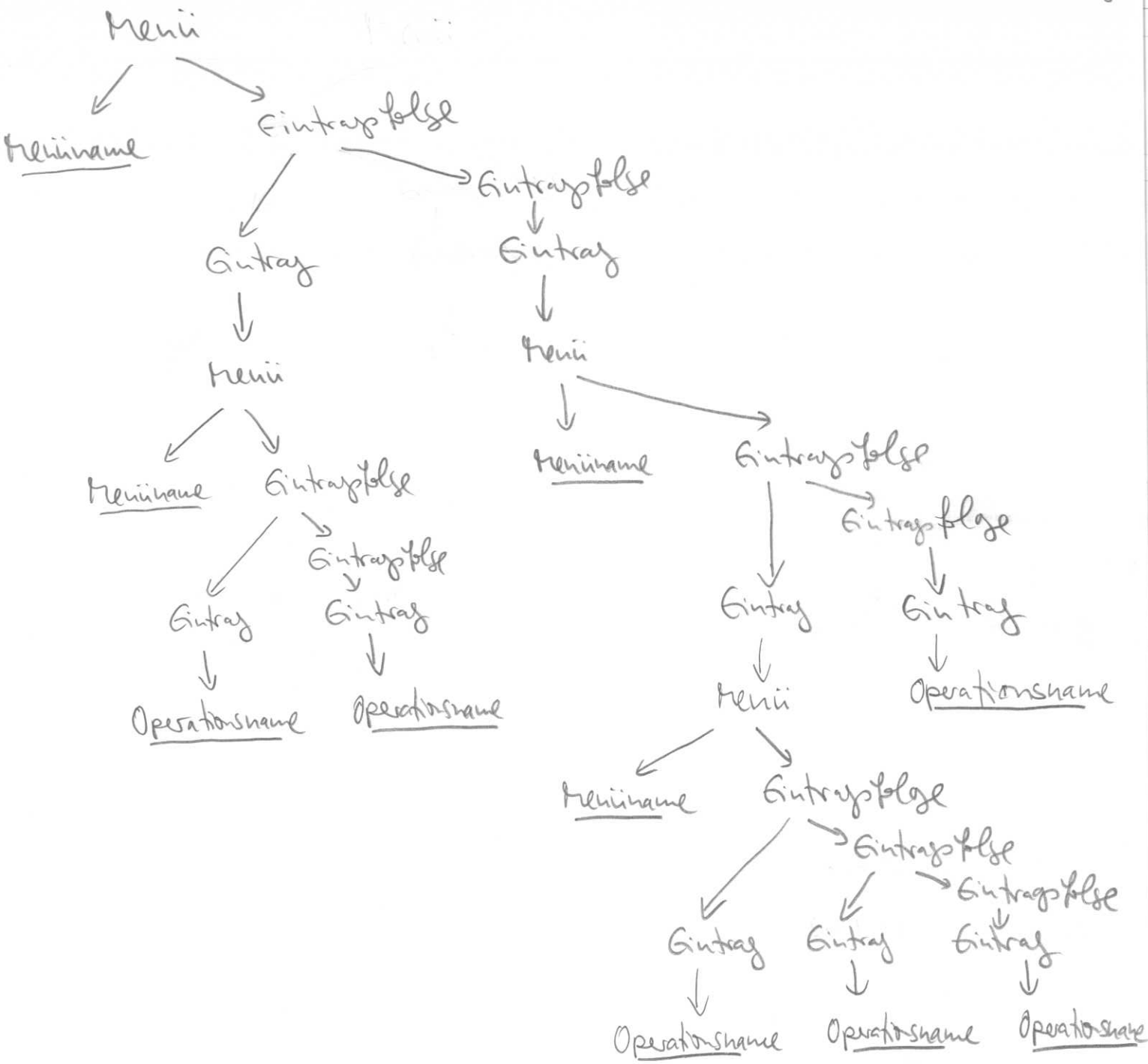
Jeder Ableitungsbaum repräsentiert die Struktur eines Menüs. Ein Menü der Struktur



Bsp: Hauptmenü



wird z.B. von folgendem Ableitungsbaum repräsentiert:


Menü
Menüname
Eintragfolge
Eintrag
Menü
Menüname
Eintragfolge
Eintrag
Eintragfolge
Eintrag
Operationsname
Eintragfolge
Eintrag
Operationsname
Menü
Menüname
Eintragfolge
Eintrag
Menü
Menüname
Eintragfolge
Eintragfolge
Eintrag
Operationsname
Eintragfolge
Eintrag
Eintrag
Operationsname
Operationsname
Operationsname

Beispiel 6.14: (HTML-Tabellen)

HTML: Format für Beschreibung von verzweigten Dokumenten im WWW.
Ein Bestandteil, der oft im Quell-Code von Internet-Seiten vorkommt, sind Tabellen. Z.B. wird der Eintrag

Tag	Zeit	Raum
Mi	8:00 – 11:00	Magnus-Hörsaal
Do	14:00 – 16:00	NM 10

durch HTML-Quelltext der folgenden Form erzeugt:

```
<table>
  <tr>
    <td> Tag </td>
    <td> Zeit </td>
    <td> Raum </td>
  </tr>
  <tr>
    <td> Mi </td>
    <td> 8:00 – 11:00 </td>
    <td> Magnus-Hörsaal </td>
  </tr>
  <tr>
    <td> Do </td>
    <td> 14:00 – 16:00 </td>
    <td> NM 10 </td>
  </tr>
</table>
```

<tr> steht für den Anfang einer Zeile der Tabelle
</tr> steht für das Ende einer Zeile der Tabelle

<td> steht für den Anfang eines Eintrags,
</td> für das Ende eines Eintrags

Als Einträge in einzelnen Zellen der Tabelle kann z.B. Text stehen oder eine weitere Tabelle.

Im Folgenden konstruieren wir eine Grammatik

$$G_{\text{HTML-Tabellen}} = (T, N, S, P),$$

so dass die von $G_{\text{HTML-Tabellen}}$ erzeugte Sprache aus (möglicherweise geschachtelten) HTML-Tabellen besteht:

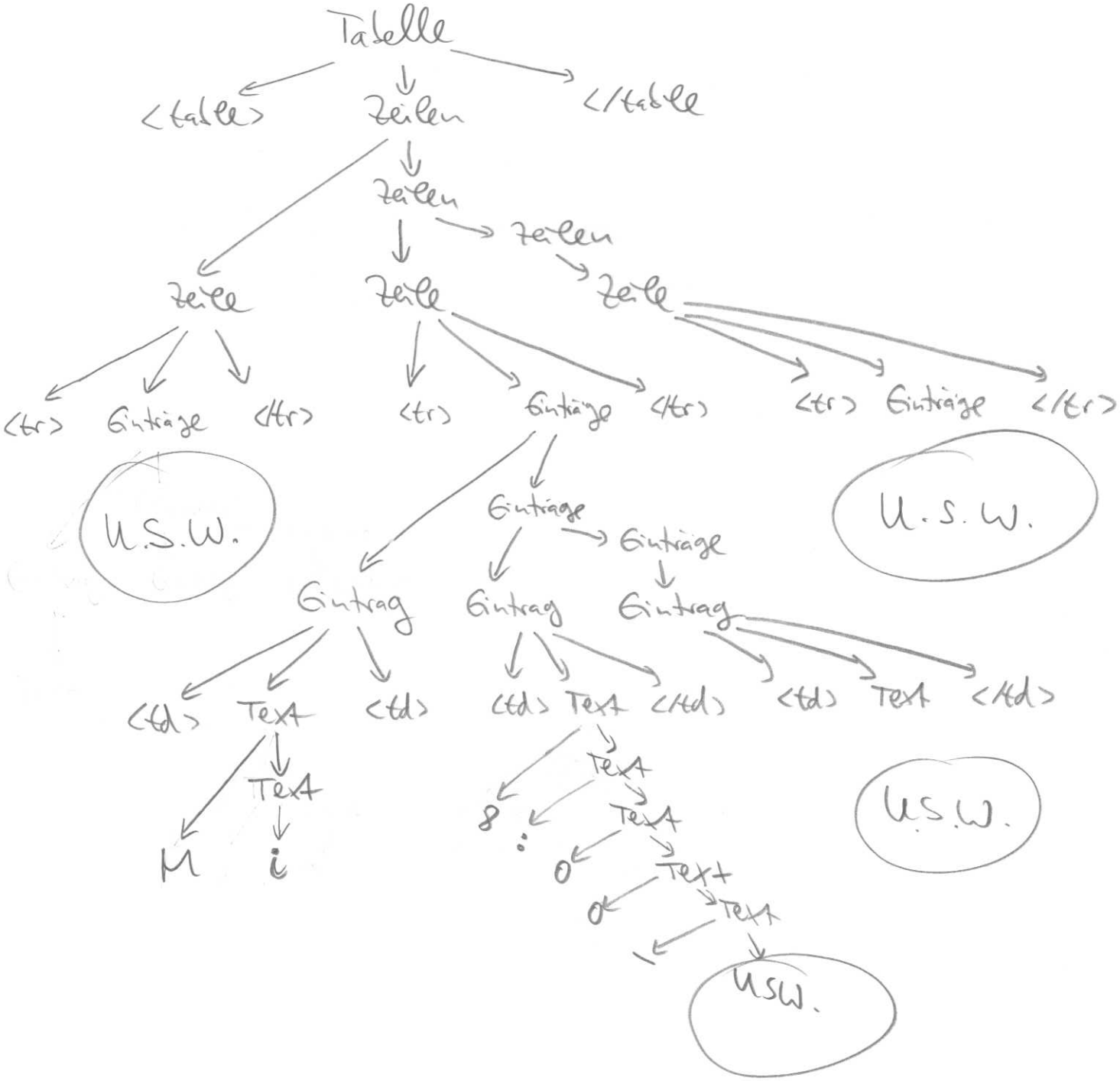
$T :=$ { <table>, </table>,
<tr>, </tr>, <td>, </td>,
a, ..., z, A, ..., Z, 0, 1, ..., 9, :, -, ␣,
ö, ä, ü, ß, Ö, Ä, Ü }

$N :=$ { Tabelle, Zeile, Eintrag, Text, Zeilen, Einträge }

$S :=$ Tabelle

$P :=$ { Tabelle → <table> Zeilen </table>,
Zeilen → Zeile, Zeilen → Zeile Zeilen,
Zeile → <tr> Einträge </tr>,
Einträge → Eintrag, Einträge → Eintrag Einträge,
Eintrag → <td> Text </td>, Eintrag → <td> Tabelle </td>
Text → a, ..., Text → z, Text → A, ..., Text → Ü,
Text → aText, ..., Text → zText, Text → AText, ..., Text → ÜText }

Die oben angegebene Beispiel-HTML-Tabelle wird z.B. durch eine Ableitung erzeugt, die durch folgenden Ableitungsbaum repräsentiert wird:



Bemerkung 6.15

Typische Fragen bzgl. kontextfreien Grammatiken:

(a) Welche Sprachen können prinzipiell durch KFGs erzeugt werden, welche nicht?

Bsp: Die Sprache $L_1 = \{ a^n b^n : n \in \mathbb{N} \}$ wird von der KGF $G_1 = (T, N, S, P)$ mit $T = \{a, b\}$, $N = \{S\}$ und $P = \{ S \to aSb, \quad S \to \varepsilon \}$ erzeugt, d.h. $L_1 = L(G_1)$.

Notation: $a^n b^n$ ist eine Abkürzung für $\underbrace{aa\cdots a}_{n}\underbrace{bb\cdots b}_{n}$

Satz: Es gibt keine KFG, die genau die Sprache $L_2 := \{ a^n b^n c^n : n \in \mathbb{N} \}$ erzeugt

Beweis: In der Vorlesung GL-2.

(b) Gegeben sei eine KFG $G = (T, N, S, P)$ und ein Wort $w \in T^*$. Wie kann man herausfinden, ob $w \in L(G)$ ist, d.h. ob das Wort w in der von G erzeugten Sprache gehört?

Dies ist das so genannte Wortproblem.

Einen Algorithmus zum Lösen des Wortproblems für CFGs werden Sie in der Vorlesung GL-2 kennenlernen (den so genannten CYK-Algorithmus, der nach seinen Erfindern Cocke, Younger und Kasami benannt ist).