Modellierung

Prof. Dr. Uwe Kastens

WS 2001/2002

Begründung der Vorlesung

- Das **Modellieren** ist eine für das Fach **Informatik typische Arbeitsmethode**.
- Mit der Modellierung einer **Aufgabe** zeigt man, ob und wie sie **verstanden** wurde.
- Ein zutreffendes Modell ist **Voraussetzung** und Maßstab **für eine systematische Lösung.**
- Als **Ausdrucksmittel** muss man **passende Kalküle und Notationen** anwenden können.

Ziele

Die Teilnehmer sollen

- einen Überblick über **grundlegende Modellierungsmethoden und -kalküle** bekommen,
- den **konzeptionellen Kern der Kalküle** beherrschen,
- erste Erfahrungen an **typischen Beispielen** sammeln und
- die für die Methoden **typischen Techniken** erlernen.

Insgesamt sollen sie lernen,

- sich bei der Analyse von Problemen **präzise auszudrücken**,
- die **Scheu vor formalen Kalkülen verlieren** und
- den **praktischen Wert von präzisen Beschreibungen** erkennen.

Durchführung

Zu jedem **Modellierungskalkül** soll(en)

- mit einigen typischen kleinen **Beispielen motivierend eingeführt** werden,
- der **konzeptionelle Kern** des Kalküls vorgestellt werden,
- **Anwendungstechniken und Einsatzgebiete** an Beispielen gezeigt und in den Übungen erfahren werden,
- an einem **durchgehenden Beispiel** größere Zusammenhänge gelernt werden,
- auf **weiterführende Aspekte** des Kalküls, seine Rolle in Informatikgebieten und -vorlesungen sowie auf algorithmische Lösungsverfahren **nur verwiesen** werden,

Inhalt

Thema	Semesterwoche	Abschnitte im Buch
1. Einführung	1	
2. Grundlegende Strukturen		
Wertebereiche	2	Anhang A
Terme, Algebren	3, 4	3.1 - 3.8
3. Logik		
Aussagenlogik	5	4.1
Verifikation von Algorithmen	6	
Prädikatenlogik	7, 8	4.2
4. Graphen	9, 10	2.2
Verbindung, Zuordnung, Anordnung		
5. Modellierung von Strukturen		
Kontextfreie Grammatiken,	11	
Entity-Relationship Modell	12, 13	
6. Modellierung von Abläufen		
endliche Automaten,	14	2.4
Petri-Netze	15	2.5
7. Projekte, Zusammenfassung	16	

Literaturhinweise

elektronisches Vorlesungsmaterial, entsteht schrittweise aus der Version des Vorjahres:

- **U. Kastens: Vorlesung Modellierung WS 2001 / 2002**
  http://www.uni-paderborn.de/cs/ag-kastens/model

zum Nachlernen und Nachschlagen:

- **G. Goos: Vorlesungen über Informatik, Band 1, 3. Auflage, Springer-Lehrbuch, 2000**
- **T. Scheurer: Foundations of Computing, System Development with Set Theory and Logic, Addison-Wesley, 1994**

elektronisches Vorlesungsmaterial aus älteren Jahrgängen:

- **H. Kleine Büning: Vorlesung Modellierung WS 1999 / 2000**
  http://www.uni-paderborn.de/cs/ag-klbue/courses/ws99/modellierung

Weitere Hinweise im Material zur Vorlesung im WWW

Bezüge zu anderen Vorlesungen

Modellierung

Problem verstehen bevor implementieren

Software-Entwicklung

Eigenschaften von D & A formal beschreiben

Datenstrukturen & Algorithmen

Spezifikation von Software-Aufgaben und -Lösungen

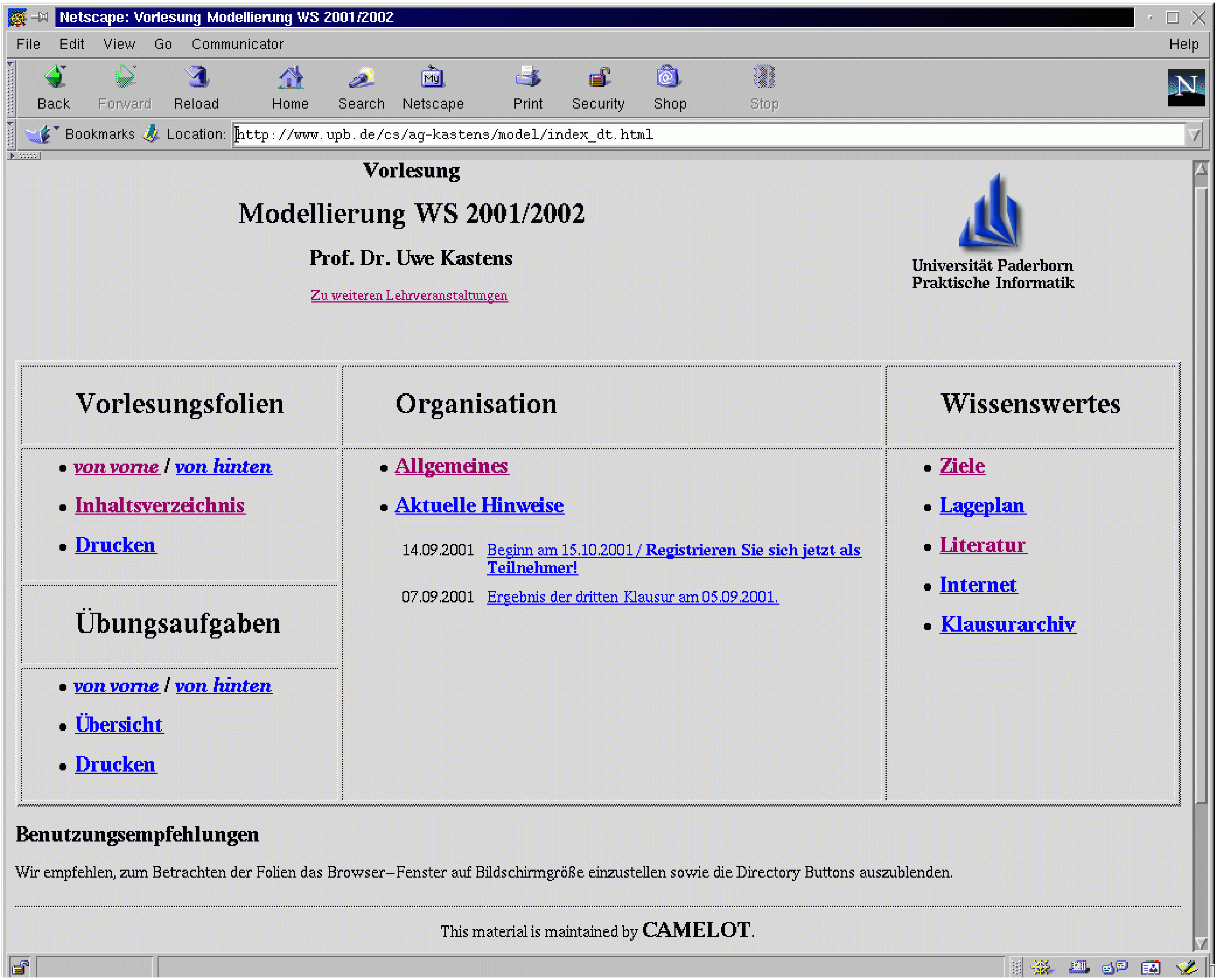
Techniken des Software-Entwurfs

auf grundlegenden Kalkülen aufbauen

Berechenbarkeit & formale Sprachen Algorithmen & Komplexität

Probleme und Aufgaben präzise beschreiben

Elektronisches Skript: Startseite



Elektronisches Skript: Organisation

Veranstalter:

Dozent: Prof. Dr. Uwe Kastens

Sprechstunde:

- Di 11.00 - 12.00, F2.308
- Mi 15.00 - 16.00, F2.308

Übungsbetreuer:

- Claudia Brak (cbrak@upb.de)
- Kim von Grawert (kim@upb.de)
- Mathias Handorf (hane@upb.de)
- Marc Jesse (jesse@math.upb.de)
- Jochen Kreimer (jotte@upb.de)
- Dinh Khoi Le (le@upb.de)
- Thomas Lücking (luck@upb.de)
- Dominik Niehus (nicke@upb.de)
- Marc Pannenberg (orionite@upb.de)
- Carsten Schmidt (cschmidt@upb.de)
- Björn Schwerdtfeger (strauss@upb.de)
- Eike Schwindt (schwindt@upb.de)
- Sven Walther (karotte@upb.de)

Termine:

Vorlesung:

- Mo 11.00 - 12.30, AudiMax
- Fr 09.00 - 10.30, AudiMax

Beginn: Montag, 15.10.2001.

Zentralübung:

- Di 13.00 - 13.45, AudiMax

Beginn: Dienstag, 30.10.2001.

Elektronisches Skript: Übungstermine

Übungsgruppen:

In den Übungsgruppen werden Aufgaben in Kleingruppen gelöst und Fragen zur Vorlesung und zu den Hausaufgaben besprochen.

Wir bieten wöchentlich an zwölf unterschiedlichen Terminen(*) insgesamt 26 Übungsgruppen an:

- A: Mo 07-09, (01) D1.312,[NN]
- (02) D1.320,[NN]
- (03) D1.328,[NN]
- B: Mo 09-11, (04) J2.130,[NN]
- (05) N3.206,[NN]
- (06) N3.237,[NN]
- C: Mo 16-18, (07) D1.328,[NN]
- (08) E2.304,[NN]
- (09) J2.130,[NN]
- D: Di 11-13, (10) E2.316,[NN]
- (11) N3.206,[NN]
- E: Di 16-18, (12) D1.328,[NN]
- (13) J2.331,[NN]
- (14) N3.206,[NN]
- F: Mi 11-13, (15) J2.130,[NN]
- G: Mi 14-16, (16) N3.206,[NN]
- H: Mi 16-18, (17) N3.206,[NN]
- I: Do 07-09, (18) D1.303,[NN]
- (19) D1.328,[NN]
- (20) D1.338,[NN]
- J: Do 09-11, (21) N3.206,[NN]
- K: Fr 07-09, (22) E2.310,[NN]
- (23) N3.206,[NN]
- L: Fr 14-16, (24) D1.303,[NN]
- (25) E2.304,[NN]
- (26) E2.310,[NN]

Beginn: Montag, 22.10.2001.

Sichern Sie sich Ihren Platz bis Donnerstag, 18.10.2001:
Hierzu *registrieren* Sie sich bitte bei unserer Teilnehmer–Datenbank (siehe unten). Dabei können Sie *zwei Wunschtermine* angeben, zu denen Sie eine Übungsgruppe besuchen möchten. Vor Beginn der Übungen erfahren Sie in Ihrem Teilnehmerkonto, welcher Übungsgruppe Sie *zugeordnet* wurden.

(*) Die Übungen von 7–9 Uhr beginnen nicht vor 7:30 Uhr. Die genaue Anfangszeit jeder Übungsgruppe wird zu Beginn mit dem Tutor und allen Teilnehmern festgelegt.

Teilnehmer:

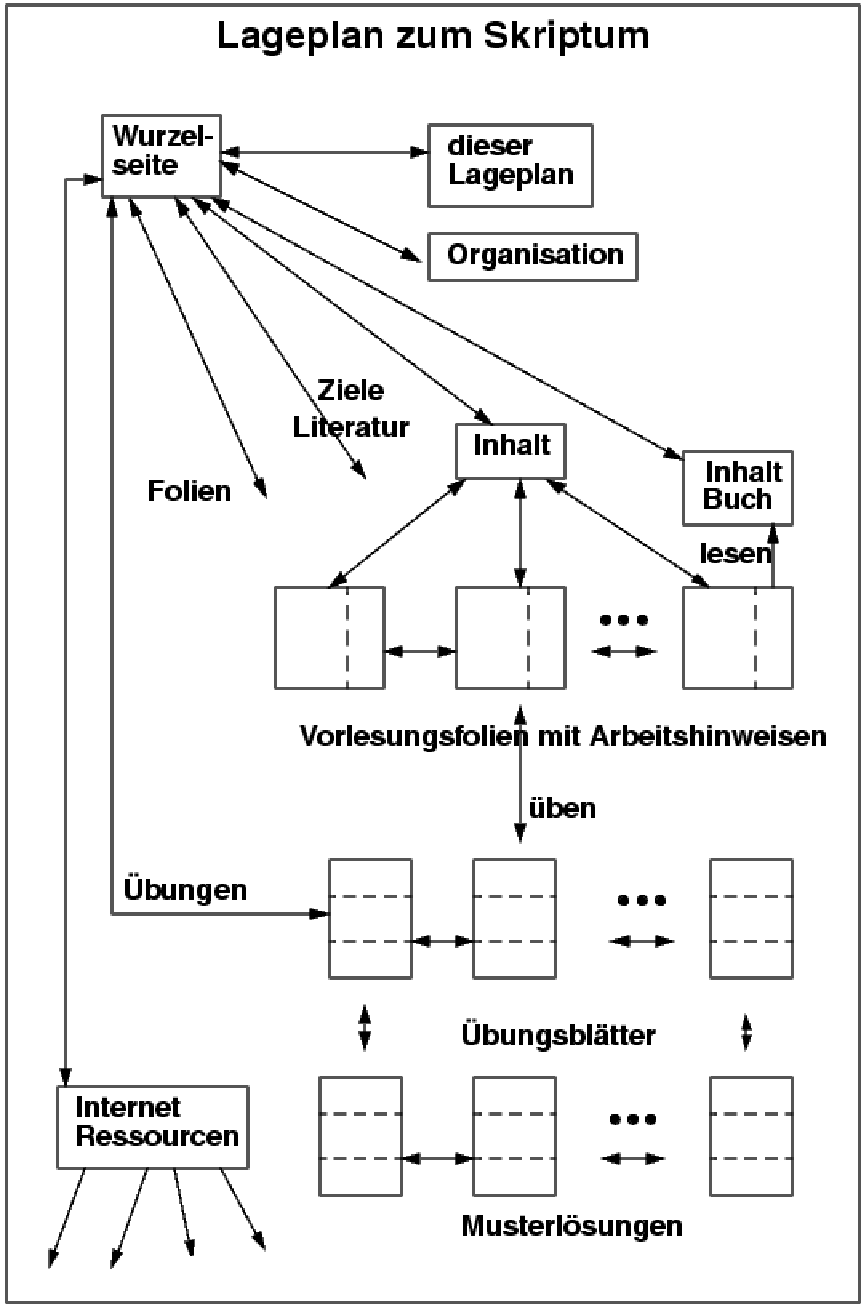
In der Teilnehmer–Datenbank können Sie jederzeit die aktuell über Sie gespeicherten Informationen abrufen:

Mein Konto

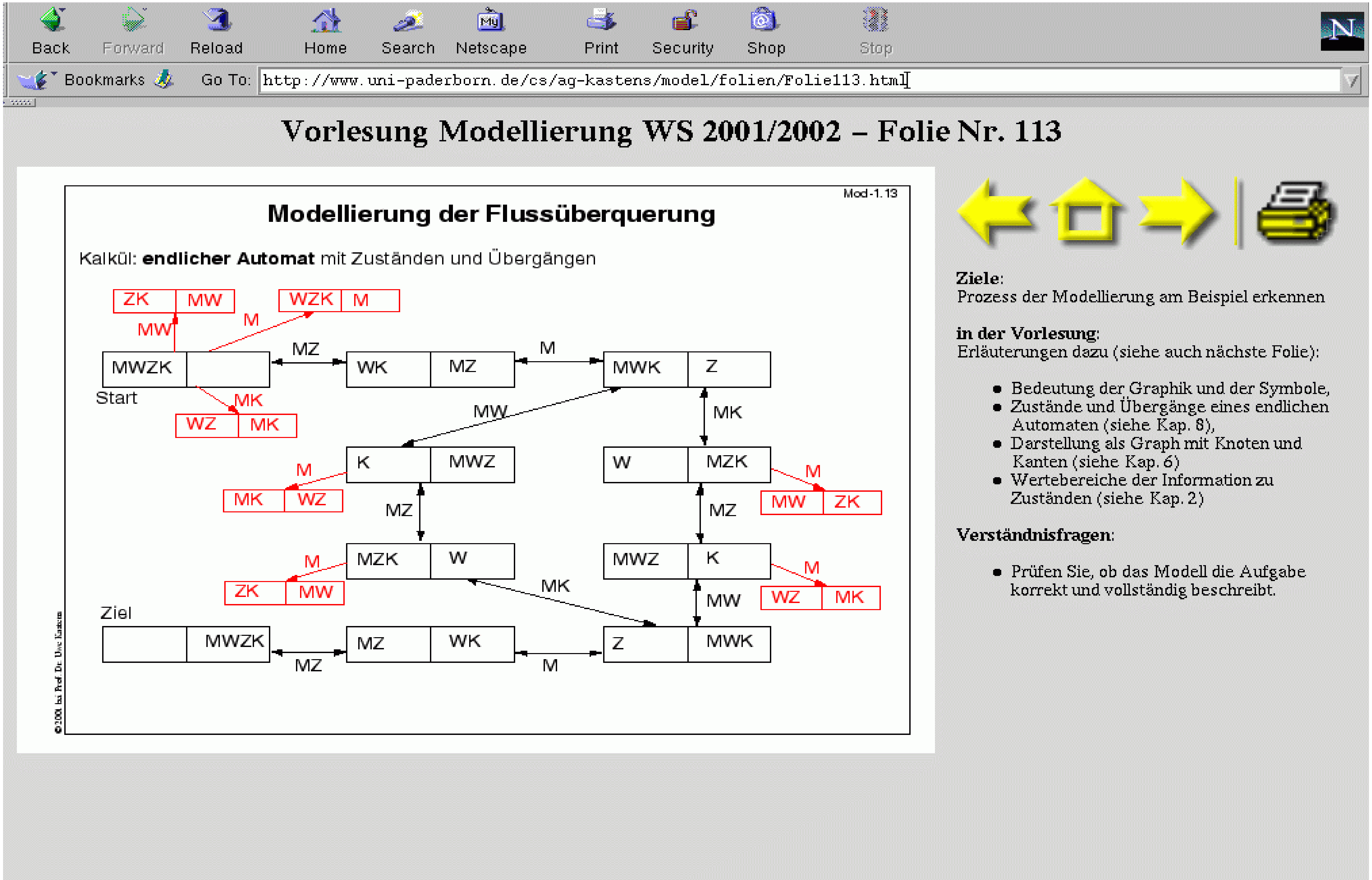
Zu Beginn der Veranstaltung müssen Sie sich hier registrieren! Danach können Sie Ihr Teilnehmerkonto benutzen.

Sie können Ihre persönlichen Informationen, sowie die Zugehörigkeit zu einer Übungsgruppe und Ihre Punkte aus den Hausaufgaben einsehen. Später finden Sie hier die Klausuranmeldung und erfahren Ihr Klausurergebnis.

Elektronisches Skript: Lageplan



Elektronisches Skript: Kommentierte Folien



Beispiel: Die Flussüberquerung

Aufgabe:

Ein Mann steht mit einem Wolf, einer Ziege und einem Kohlkopf am linken Ufer eines Flusses, den er überqueren will. Er hat ein Boot, das groß genug ist, ihn und ein weiteres Objekt zu transportieren, so dass er immer nur eins der drei mit sich hinübernehmen kann.

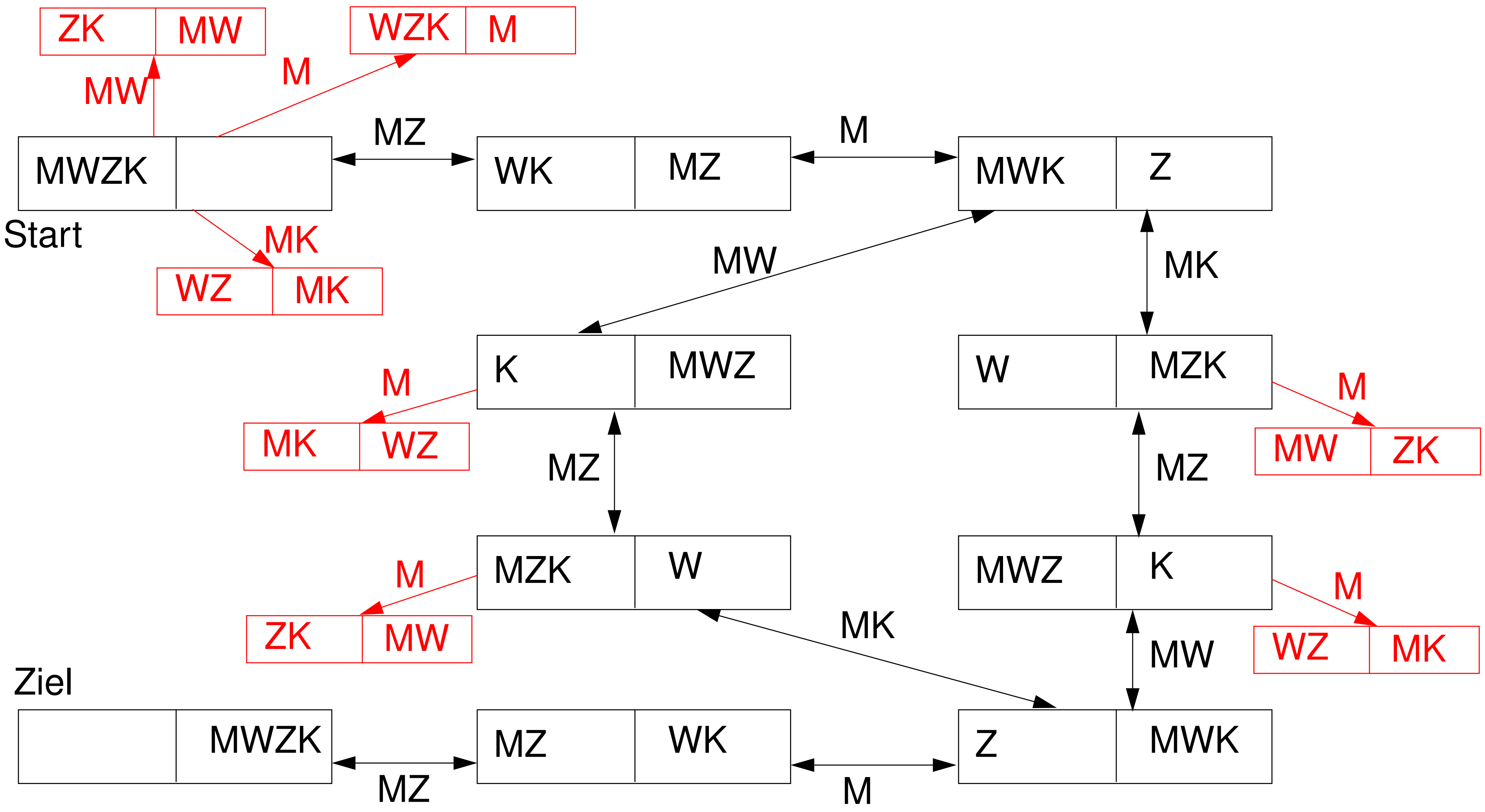
Falls der Mann allerdings den Wolf und die Ziege oder die Ziege und den Kohlkopf unbewacht an einem Ufer zurücklässt, so wird einer gefressen werden.

Ist es möglich, den Fluss zu überqueren, ohne dass die Ziege oder der Kohlkopf gefressen werden?

Quelle: Hopcroft, Ullman: Einführung in die Automatentheorie, formale Sprachen und Komplexitätstheorie, S. 14, 15

Modellierung der Flussüberquerung

Kalkül: **endlicher Automat** mit Zuständen und Übergängen



Diskussion des Modellierungsbeispiels

- Modellierung von **Abläufen**, Folgen von Schritten: Kalkül endlicher Automat
- **Abstraktion**: nur die Zustände und Übergänge interessieren
- **relevante Objekte benannt**: M, W, Z, K
- jeder **Zustand** wird charakterisiert durch ein **Paar von Mengen** der Objekte, (linkes Ufer, rechtes Ufer); jedes Objekt kommt genau einmal vor
- zulässige und **unzulässige Zustände**
- **Übergänge** werden mit den transportierten Objekten beschriftet

Hier liefert die Modellierung die Lösung direkt: Es gibt Wege vom Start zum Ziel.
Meist kann man am Modell Lösungen entwickeln.

Besonders wichtig ist, was **nicht modelliert** wurde, da es **für die Aufgabe irrelevant** ist!
z. B. die Länge des Bootes, die Breite und Tiefe des Flusses, usw.

Modellierungsbeispiel: Getränkeautomat

Die **Bedienung eines Getränkeautomaten** soll modelliert werden. Das Gerät soll Getränke wie Kaffee, Tee, Kakao gegen **Bezahlung mit Münzen** abgeben. Man soll **Varianten der Getränke** wählen können, z. B. mit oder ohne Milch oder Zucker. Die Modellierung soll berücksichtigen, dass im Gerät nur **begrenzte Vorräte** untergebracht werden können.

Im Rahmen der **Übungen** werden **präzisere Beschreibungen** der Bedienung und der Funktionen des Getränkeautomaten entwickelt.

Im Laufe des Semesters werden wir die jeweils gelernten **Kalküle zur Modellierung des Getränkeautomaten anwenden**. Daran werden wir erkennen, welche Kalküle sich für welche Aspekte gut eignen.

Allgemeiner Modellbegriff

- **Abbild** eines vorhandenen Originals (z. B. Schiffsmodell)
- **Vorbild** für ein herzustellendes Original (Gebäude in kleinem Maßstab; Vorbild in der Kunst)
- **konkretes** oder **abstraktes Modell** (Schiffsmodell, Rentenmodell)
- konkretes oder abstraktes **Original** (Schiff, Bevölkerungsentwicklung)

davon abweichende Bedeutungen:

- Fotomodell: führt Mode (oder sich) vor
- Automodell: Typreihe
- in der Logik: Eine Struktur S ist ein Modell der Formeln F, wenn alle F für S gelten.

hier in der Informatik:

- **abstraktes Abbild oder Vorbild zu abstrakten oder konkreten Originalen**

Modellbegriff im allgemeinen Lexikon

Modell [italien., zu lat. modulus „Maß, Maßstab"], allg. Muster, Vorbild, Entwurf.

▷ Mensch (auch Tier), der (das) als Vorbild für künstler. Studien oder Kunstwerke dient („sitzt").

▷ in der *Bildhauerei* meist in verkleinerter Form ausgeführter Entwurf einer Plastik oder Tonarbeit, die in Bronze gegossen werden soll. – ↑Architekturmodell.

▷ in der *Modebranche* Bez. für 1. ein nur einmal oder in eng begrenzter Anzahl hergestelltes Kleidungsstück *(M.kleid);* 2. die Vorlage für eine Vervielfältigung; 3. svw. Mannequin.

▷ im *Sprachgebrauch verschiedener Wiss.* (Philosophie, Naturwiss., Soziologie, Psychologie, Wirtschaftswiss., Politikwiss., Kybernetik u. a.) ein Objekt materieller oder ideeller (Gedanken-M.) Natur, das von einem Subjekt auf der Grundlage einer Struktur-, Funktions- oder Verhaltensanalogie für ein anderes Objekt *(Original)* eingesetzt und genutzt wird, um Aufgaben zu lösen, deren Durchführung unmittelbar am Original selbst nicht möglich bzw. zu aufwendig ist (z. B. Flugzeug-M. im Windkanal). Die **Modellmethode** vollzieht sich in vier Schritten: 1. Auswahl (Herstellung) eines dem [geplanten] Original entsprechenden M.; 2. Bearbeitung des M., um neue Informationen über das M. zu gewinnen (**Modellversuch;** ↑Ähnlichkeitsgesetze); 3. Schluß auf Informationen über das Original (meist Analogieschluß); ggf. 4. Durchführung der Aufgabe am Original. Infolge der Relationen zw. Subjekt, Original und M. **(Modellsystem)** ist ein M. einsetzbar u. a. zur Gewinnung neuer Informationen über das Original (z. B. Atom-M.), zur Demonstration und Erklärung (z. B. Planetarium), zur Optimierung des Originals (z. B. Netzplan), zur Überprüfung einer Hypothese oder einer techn. Konstruktion (z. B. Laborversuch). – Abweichend von diesem M.begriff versteht die *mathemat. Logik* unter M. eine Interpretation eines Axiomensystems, bei der alle Axiome dieses Systems wahre Aussagen darstellen. Diese **Modelltheorie** liefert grundlegende Verfahren zur Behandlung von Fragen der Vollständigkeit, Widerspruchsfreiheit und Definierbarkeit.

Wissenschaften
einschließlich
Informatik

mathematische
Logik

aus
Meyers Neues Lexikon, in zehn Bänden,
Meyers Lexikonverlag, 1993

Modellbegriff im Lexikon der Informatik

Modell (allgemeiner Begriff)

Teilgebiet: Modellierung

model (in general)

Während wir in den Formalwissenschaften wie Mathematik oder Physik einen präzisen Gebrauch des Wortes „Modell" (→ *Gegenstandsraum*) vorfinden, wird das Modell-Denken in den Sozialwissenschaften weitgehend durch einen vagen Gebrauch des Ausdrucks „Modell" gekennzeichnet. Folgende Begriffe, die sich in ihrer Intention oft stark unterscheiden, dürften die gebräuchlichsten Verwendungsweisen sein:

1. *Modell in der mathematischen Logik*
2. Modell als Bezeichnung für Theorien schlechthin
3. Modell als Resultat der Abbildung der Wirklichkeit.

Weitere Klassifizierungskriterien (→ *Klassifizierung²*) lassen sich nach dem Zweck, der mit den einzelnen Modellen verfolgt wird angeben (siehe Abb. S. 512).

Modell als Theorie schlechthin (2) findet sich häufig im verbalen Sprachgebrauch der Sozialwissenschaften. Insbesondere jene Teilklassen von Theorien, die mathematisiert, quantifiziert bzw. formalisiert sind, werden allgemein als Modell bezeichnet. Beispiele sind Preismodell, Rentenmodell.

Modelle als Abbild der Realität (3) stellen eine umfangreiche, sehr heterogene Klasse dar. Hierbei bilden die Beschreibungen ohne Verwendung einer Sprache, meist auf ein handliches Maß verkleinerten Nachbildungen eines vorgestellten Originals, die bekannteste Art von Modellen. Diese werden, wie z.B. der Globus, auch als ikonische oder materiale Modelle bezeichnet. *Stübel*

Teilgebiet: Logik

model

Es gibt zwei unterschiedliche Definitionen für Modelle der mathematischen *Logik*:

a) Eine Struktur Σ heißt Modell einer Formelmenge X, wenn jede Formel aus X in Σ gültig ist.

b) Das Paar (I,ζ), bestehend aus einer Interpretation I und einer *Belegung* ζ, heißt Modell einer Formelmenge X, wenn jede Formel aus X bei I und ζ wahr ist.

Für Mengen X von Aussagen, also Formeln ohne freie Variablen, sind beide Definitionen gleichwertig, da dann die Belegung keine Rolle spielt.

Die Modelltheorie beschäftigt sich mit gegenseitigen Beziehungen zwischen Aussagen formalisierter Theorien und mathematischen Strukturen, in denen die Aussagen gelten. *Müller; Stübel*

Modell, abstrakt symbolisches

Teilgebiet: Modellierung

abstract symbolic model

Eine vor allem in der Betriebswirtschaft sehr verbreitete Klasse von Modellen bilden die abstrakt symbolischen Abbilder eines Realitätskomplexes. Dabei kann es sich sowohl um rein verbale Reproduktionen eines Systems handeln als auch um ein künstliches Sprachsystem, das durch zunächst inhaltsleere symbolische Zeichen und syntaktische (→ *Syntax von Programmiersprachen*) Regeln gekennzeichnet ist. *Stübel*

aus
H-J. Schneider: Lexikon der Informatik und Datenverarbeitung, 3. Aufl., Oldenbourg Verlag, 1991

Zweck des Modells

Der **Verwendungszweck** bestimmt die Art des Modells! z. B.

- Gebäudemodell: optischer Eindruck
- Grundriss: Einteilung des Grundstückes und der Räume
- Kostenplan: Finanzierung
- Gewerkeplan: Bauabwicklung

Nur was **für den Zweck relevant** ist, wird modelliert!

Vollständige Modellierung des Originals ist nicht sinnvoll.

Für den Zweck die jeweils passende Modellierungsmethode (Kalkül) verwenden!

Arbeiten mit dem Modell

- **Operationen, die man am Original nicht durchführen kann**
  z. B. neue Flügelform im Windkanal oder in der Computer-Simulation erproben
- Bestimmte Aspekte eines **komplexen Gebildes untersuchen und verstehen**,
  z. B. Geschäftsabläufe in einer Firma
- **Verständigung zwischen Auftraggeber und Hersteller** des Originals,
  z. B. Hausbau, Software-Konstruktion
- Fixieren von **Anforderungen für die Herstellung** des Originals,
  Software: Requirements, Spezifikation

Modell validieren:

Nachweisen, dass die
relevanten Eigenschaften des Originals korrekt und vollständig im Modell erfasst sind
und darüber Einvernehmen herstellen.

Bezug zwischen Original und Modell

$$\begin{tikzcd}[column sep=huge]
\text{Original} & O \arrow[r, "\text{Modellabbildung}", "a"'] \arrow[dd, "\text{opO}"'] & M \arrow[dd, "\text{opM}"] & \text{Modell} \\
\text{Operation auf Original} & & & \text{Operation auf Modell} \\
\text{geändertes Original} & O' & M' \arrow[l, "\text{Interpretation}"', "I"] & \text{geändertes Modell}
\end{tikzcd}$$

Für alle relevanten Operationen muss das Diagramm kommutieren, d. h

$$opO\ (O) = I\ (opM\ (a\ (O)))$$

Die Operation auf dem Original entspricht der Interpretation der Operation auf dem Modell.

Modellierte Aspekte

Ein Modell beschreibt nur bestimmte Aspekte des Originals und seiner Teile:

- **Struktur**, Zusammensetzung des Originals (z. B. Organisationsschema einer Firma)
- **Eigenschaften** von Teilen des Originals (z. B. Farbe und Wert einer Spielkarte)
- **Beziehungen** zwischen Teilen des Originals (z. B. Abhängigkeiten der Gewerke beim Hausbau)
- **Verhalten** des Originals unter Operationen (z. B. Zugfolge bei der Flussüberquerung)

Zur Modellierung bestimmter Aspekte eignen sich bestimmte Methoden und Kalküle:

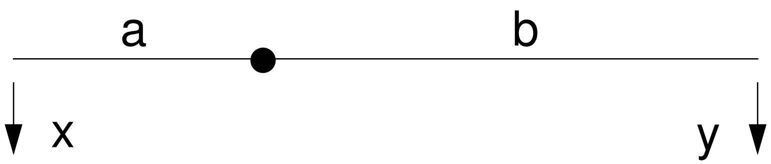
- **Struktur**: Wertebereiche, Entity-Relationship, KFG, Klassifikation, Typen
- **Eigenschaften**: Logik, Relationen
- **Beziehungen**: Graphen, Relationen, Logik, Entity-Relationship
- **Verhalten**: endliche Automaten, Petri-Netze, Algebren, Graphen

Deklarative oder Operationale Beschreibung

Deklarative Beschreibung des Modells
macht Aussagen über Aspekte des Originals.

Operationale Beschreibung des Modells
gibt an, wie sich das Original unter bestimmten Operationen verhält.

Beispiel Balkenwaage:



deklarativ:
Die Waage ist im Gleichgewicht, wenn sich die Gewichte umgekehrt proportional zu den Längen der Balken verhalten: x * a = y * b.

operational:
Wenn ich auf den Balken der Länge a ein Gewicht x auflege, muss ich auf den Balken der Länge b ein Gewicht y = x * a / b auflegen, damit die Waage im Gleichgewicht ist.

deklarativ:
Aussagen meist allgemein gültig,
auf die Aufgabe bezogen,
ohne redundante Abläufe

operational:
häufig nur Beispiele, unvollständig,
legt eine Lösung nahe (fest),
erzwingt Nachvollziehen von Abläufen

2. Grundlegende Strukturen
2.1 Wertebereiche beschrieben durch Mengen

In der Modellierung von Systemen, Aufgaben, Lösungen kommen **Objekte unterschiedlicher Art und Zusammensetzung** vor.

Für Teile des Modells wird angegeben, **aus welchem Wertebereich sie stammen**, aber noch offen gelassen, welchen Wert sie haben.
Beispiel: Gegeben 3 Karten aus einem Kartenspiel; welches ist die höchste?

Die Beschreibung des Modells wird präzisiert durch **Angabe der Wertebereiche**, aus denen die Objekte, Konstanten, Werte von Variablen, Eingaben, Ausgaben, Lösungen, usw. stammen.

Wertebereich: eine **Menge gleichartiger Werte**
Wertebereiche werden aus Mengen und Strukturen darüber gebildet.

Beispiel: Modellierung von Kartenspielen

Wertebereich	**einige Elemente daraus**
KartenSymbole := {7, 8, 9, 10, Bube, Dame, König, Ass}	8 Dame
KartenArten := {Kreuz, Pik, Herz, Karo}	Pik
Karten := KartenArten $\times$ KartenSymbole Menge aller Paare aus KartenArten und KartenSymbole	(Kreuz, 8) (Herz, Dame)

Übersicht über Begriffe

Wertebereich: eine **Menge gleichartiger Werte**

Grundlegender Kalkül: **Mengenlehre** (halbformal);
Mengen und Mengenoperationen

Strukturen über Mengen zur Bildung **zusammengesetzter Wertebereiche**

- Potenzmengen
- Kartesisches Produkt, Tupel
- Folgen
- Relation
- Funktion
- disjunkte Vereinigung

Verwendung des Kalküls:

Modellierung von Strukturen und Zusammenhängen

Grundlage für alle anderen formalen Kalküle

abstrakte Grundlage für Typen in Programmiersprachen

Einführendes Beispiel 2.1

Internationale Arbeitsgruppen

Bei der UNO sollen Arbeitsgruppen aus Delegierten der drei Nationen A, B und C gebildet werden. Jede Nation hat vier Delegierte. Jede Gruppe besteht aus drei Personen, eine aus jeder Nation. Die Sprachen der drei Nationen sind verschieden; wir nennen sie auch A, B, C. Die Mitglieder jeder Arbeitsgruppe sollen eine gemeinsame Sprache sprechen.

aus [T. Scheurer S. 155]

Beispiel 2.1 Modellierung mit Wertebereichen

Beschreibung	**formale Angaben**
Menge der Nationen	Nationen := {A, B, C}
Indexmenge zur Unterscheidung der Delegierten	Ind := {1, 2, 3, 4}
ein Delegierter modelliert durch ein **Paar**	(a, i) mit $a \in$ Nationen, $i \in$ Ind
Wertebereich der Delegierten	Delegierte := Nationen $\times$ Ind

Wertebereich der Arbeitsgruppen

3-Tupel, kartesischesProdukt AGn := $\{(A, i) \mid i \in Ind\} \times \{(B, j) \mid j \in Ind\} \times \{(C, k) \mid k \in Ind\}$

Wertebereich für **Teilmengen** von Sprachen	SprachMengen := Pow (Nationen)
	Pow (M) ist die **Potenzmenge** von M
Eine **Funktion** Sp gibt an, welche Sprachen ein	
Delegierter spricht:	Sp $\in$ DSpricht
Wertebereich solcher Funktionen	DSpricht := Delegierte -> SprachMengen
Wertebereich der gemeinsamen Sprachen einer AG	
Wertebereich	AGSpricht := AGn -> SprachMengen
GemSp ist eine Funktion daraus	GemSp $\in$ AGSpricht

N := M bedeutet „**Der Name N ist definiert als M**“.

Mengen

Menge: Zusammenfassung M von verschiedenen Objekten, den Elementen der Menge.
a ist Element aus M wird notiert $a \in M$.

Definition von Mengen durch

- **Aufzählen der Elemente (extensional)**: $M := \{1, 4, 9, 16, 25\}$
- **Angabe einer Bedingung (intensional)**: $M := \{a \mid a \in \mathbb{N}, a \text{ ist Quadratzahl und } a \leq 30\}$

Mengen können aus **atomaren oder zusammengesetzten** Elementen gebildet werden,
z. B. Menge von Paaren {(Pik, 10), (Herz, Dame)}

Eine Menge kann auch **verschiedenartige Elemente** enthalten, z. B. {1, (Pik, 10), M, 9}
aber **nicht bei der Modellierung mit Wertebereichen**

Die Existenz von atomaren Objekten des jeweiligen Modellierungsbereiches wird vorausgesetzt, z. B. die natürlichen Zahlen, Arten und Werte von Spielkarten.

Wichtige grundsätzliche Eigenschaften von Mengen:

- **Alle Elemente einer Menge sind verschieden.**
- **Die Elemente einer Menge sind nicht geordnet.**
- **Dieselbe Menge kann auf verschiedene Weisen definiert werden.**

Russels Paradoxon

Man muss prinzipiell entscheiden können, ob ein Wert a **Element einer Menge** M ist, „$a \in M$?“

Russels Paradoxon:

Sei P die Menge aller Mengen, sie sich nicht selbst als Element enthalten,
also $P := \{ x \mid x \notin x\}$.

Dann führt die Frage „Ist P Element von P?“ zum **Widerspruch**.

Um solche Anomalien auszuschließen, geben wir in **intensionalen Mengendefinitionen** an, aus welchem größeren, **schon definierten Wertebereich** die Elemente stammen:

$M := \{a \mid a \in \mathbb{N}, \text{a ist Quadratzahl und } a \leq 30\}$

hier also „$a \in \mathbb{N}$“.

Mengenoperationen

die leere Menge	$\emptyset$	
Teilmenge von	$M \subseteq N$	aus $a \in M$ folgt $a \in N$
echte Teilmenge von	$M \subset N$	$M \subseteq N$ und $M \neq N$
Vereinigung	$M \cup N$	$:= \{x \mid x \in M \text{ oder } x \in N\}$
Durchschnitt	$M \cap N$	$:= \{x \mid x \in M \text{ und } x \in N\}$
Differenz	$M \setminus N$	$:= \{x \mid x \in M \text{ und } x \notin N\}$
relatives Komplement	$U \setminus N$	mit $N \subseteq U$ Trägermenge

M und N sind **disjunkt** genau dann, wenn gilt $M \cap N = \emptyset$

$|M|$ oder Card(M) ist die **Kardinalität** von M, d. H. die Anzahl der Elemente in M

Potenzmengen

Potenzmenge (powerset) einer Grundmenge U ist die **Menge aller Teilmengen** von U,
geschrieben Pow (U) oder $\wp$ (U).
Pow (U) := $\{M \mid M \subseteq U\}$

Beispiel: Grundmenge U := {a, b} Potenzmenge Pow (U) = {∅, {a}, {b}, {a, b}}

Kardinalität: $|\text{Pow}(U)| = 2^n$ wenn $|U| = n$

Wenn die **Werte Teilmengen von U** sind, ist ihr **Wertebereich die Potenzmenge von U**.

Beispiel 2.1: Wertebereich der Sprachen, die ein Delegierter spricht
SprachMengen := Pow (Nationen), $\{A, B\} \in$ SprachMengen

Modellierungstechnik: Menge von Lösungen statt einer Lösung
Manche Aufgaben haben nicht immer genau eine Lösung, sondern je nach Daten mehrere oder keine Lösung. Dann kann man nach der Menge aller Lösungen fragen.
Der Wertebereich der Antwort ist die **Potenzmenge** des Wertebereiches der Lösungen.

Vergleiche auch **Mengentyp** in Pascal:

```
type Sprachen = set of {A, B, C};
var spricht: Sprachen;
spricht := {A, B};
```

Kartesisches Produkt

Kartesisches Produkt der Mengen M und N:

Menge **aller geordneten Paare** mit erster Komponente aus M und zweiter aus N

$M \times N := \{z \mid z = (x, y) \text{ und } x \in M \text{ und } y \in N\}$

oder kürzer $M \times N := \{(x, y) \mid x \in M \text{ und } y \in N\}$

Zusammengesetzte Wertebereiche, z. B. Delegierte := Nation $\times$ Ind

Verallgemeinert zu (geordneten) **n-Tupeln**:

$M_1 \times M_2 \times \ldots \times M_n := \{(a_1, a_2, \ldots, a_n) \mid a_i \in M_i \text{ und } i \in I\}$
mit **Indexmenge** $I := \{1, \ldots, n\}$ und $n > 1$

Beispiel:
Daten := Tage $\times$ Monate $\times$ Jahre

Notation bei **gleichen Mengen M_i** : $M \times M \times \ldots \times M = M^n$ mit $n > 1$

Beispiel:
Wertebereich der Ergebnisse 3-maligen Würfelns: DreiWürfe := $\{1, 2, 3, 4, 5, 6\}^{\mathbf{3}}$

Kardinalität: $| M_1 \times M_2 \times \ldots \times M_n | = \prod_{i \in I} |M_i|$ mit $I = \{1,\ldots, n\}$ mit $n > 1$

Folgen

Endliche Folgen von Elementen aus A: n-Tupel über A von beliebiger Länge n > 0

$\mathbf{A^+} := \{ x \mid x \in A^{\mathbf{i}} \text{ und } i \geq 1 \}$ d. h. $(a_1, ..., a_n) \in A^+$ für $n \geq 1$ und $a_i \in A$

Notationen:

ε oder ()	**die leere Folge**
$A^{\mathbf{0}} := \{ \varepsilon \}$	Menge, die nur die leere Folge über A enthält
$A^{\mathbf{1}} := \{ (a) \mid a \in A \}$	Menge einelementiger Folgen über A
$\mathbf{A^* := A^+ \cup A^0}$	der Wertebereich von Folgen über A, der auch die leere Folge enthält.

Beispiele:

$(1, 1, 2, 5, 5, 10, 20) \in \mathbb{N}^+$

$(m, o, d, e, l, l) \in \text{Buchstaben}^+$

$\text{neueAufträge} := \text{Auftrag}^*$

$\text{gezogeneKarten} := \text{Karten}^*$

Relationen

Relationen sind Teilmengen aus kartesischen Produkten.

n-stellige Relation: $R \subseteq M_1 \times M_2 \times \ldots \times M_n$

Eine solche Relation R stammt also aus dem **Wertebereich Pow** $(\mathbf{M_1} \times \mathbf{M_2} \times \ldots \times \mathbf{M_n})$

Seine **Kardinalität** ist $|\mathbf{Pow}\ (\mathbf{M_1} \times \mathbf{M_2} \times \ldots \times \mathbf{M_n})| = 2^{\prod_{i \in I} |\mathbf{Mi}|}$, falls alle $\mathbf{M_i}$ endlich sind.

Eine Relation R definiert eine **Aussage über Tupel**.
Wir sagen auch: „Eine Relation R gilt für die Tupel in R.“

Beispiele: NationenKleiner := {(A, C), (C, B), (A, B)} $\subseteq$ Nationen^2

Oder := {(w, w), (w, f), (f, w)} $\subseteq$ Bool^2

GültigeDaten $\subseteq$ Daten = Tage $\times$ Monate $\times$ Jahre
(20, Oktober, 2000) $\in$ GültigeDaten, (30, Februar, 2000) $\notin$ GültigeDaten

alternative Schreibweisen für Elemente aus Relationen:
R(a) für a $\in$ R, z. B. GültigeDaten(20, Oktober, 2000)

bei 2-stelligen Relationen auch mit Operatoren:
x R y für (x, y) $\in$ R, z. B. $x \leq y$, $a \neq b$, $p \rightarrow q$

Eigenschaften 2-stelliger Relationen

Für zweistellige Relationen $R \subseteq M \times M$ sind folgende Begriffe definiert:

- **reflexiv**, wenn für alle $x \in M$ gilt $x\ R\ x$;
- **irreflexiv**, wenn für kein $x \in M$ gilt $x\ R\ x$;
- **symmetrisch**, wenn für alle $x, y \in M$ gilt: aus $x\ R\ y$ folgt $y\ R\ x$;
- **antisymmetrisch**, wenn für alle $x, y \in M$ gilt: aus $x\ R\ y$ und $y\ R\ x$ folgt $x = y$;
- **asymmetrisch**, wenn für alle $x, y \in M$ gilt: aus $x\ R\ y$ folgt, $y\ R\ x$ gilt nicht;
- **transitiv**, wenn für alle $x, y, z \in M$ gilt, aus $x\ R\ y$ und $y\ R\ z$ folgt $x\ R\ z$;
- **alternativ**, wenn für alle $x, y \in M$ gilt $x\ R\ y$ oder $y\ R\ x$;
- **Äquivalenzrelation**, wenn R reflexiv, symmetrisch und transitiv ist.

Hinweise zum Anwenden der Definitionen:

- „**x R y**“ bedeutet „**(x, y)** $\in$ **R**“
- **„**für alle $x \in M$ gilt ...“: der **gesamte Wertebereich M** muss geprüft werden
- „für alle $x, y \in M$ gilt ...“: alle Paare von Werten aus M prüfen, auch solche mit $x = y$
- „A **oder** B“ ist wahr, wenn **mindestens eins von beiden wahr** ist
- „**aus A folgt B**“ ist gleichwertig zu „**(nicht A) oder B**“.

Ordnungsrelationen

Eine zweistellige Relationen $R \subseteq M \times M$ ist eine

- **partielle Ordnung** oder **Halbordnung**, wenn R reflexiv, antisymmetrisch und transitiv ist;
- **strenge Ordnung** oder **strenge Halbordnung**, wenn R irreflexiv und transitiv ist;
- **Quasiordnung**, wenn R reflexiv und transitiv ist;
- **totale** oder **lineare Ordnung**, wenn R eine alternative Halbordnung ist,
  also alternativ, reflexiv, antisymmetrisch und transitiv.

Funktionen

Eine **Funktion f ist eine 2-stellige Relation f $\subseteq$ D $\times$ B** mit folgender Eigenschaft:
Aus (x, y) $\in$ f und (x, z) $\in$ f folgt y = z, d. h. zu einem Urbild x gibt es nur ein Bild y.

D ist der **Definitionsbereich** von f; B ist der **Bildbereich** von f
D und B können beliebige, auch zusammengesetzte Wertebereiche sein.

Der **Wertebereich D -> B** ist die **Menge aller Funktionen, die von D auf B abbilden**.
Es gilt **D -> B $\subseteq$ Pow (D $\times$ B).**
D -> B enthält als Elemente alle Mengen von Paaren über **D $\times$ B**, die Funktionen sind

Für eine Funktion f $\in$ D -> B gilt f $\subseteq$ D $\times$ B.
Statt **f $\in$ D -> B** sagt man auch **f hat die Signatur D -> B** oder kurz **f: D -> B**

Schreibweisen für (x, y) $\in$ f auch y = f (x) oder f (x) = y oder x f y

Die Menge der Paare (x, y) $\in$ f heißt **Graph von f**.

Beispiele:

Funktion	**Wertebereich**
not := {(w, f), (f, w)}	Bool -> Bool
id := {(w,w), (f, f)}	Bool -> Bool
Quadrat := {(a, b) | a $\in \mathbb{N}$ und b = a*a}	$\mathbb{N}$ -> $\mathbb{N}$
ggt := {((a,b),c) | a, b, c $\in \mathbb{N}$ und c ist größter gemeinsamer Teiler von a und b}	$\mathbb{N} \times \mathbb{N}$ -> $\mathbb{N}$
Sp := {((A, 1), {A,B}), ((B, 2), {B})}	Delegierte -> SprachMengen

Eigenschaften von Funktionen

Eine Funktion **f : D -> B** heißt

- **total**, wenn es für jedes $x \in D$ ein Paar $(x, y) \in f$ gibt,
- **partiell**, wenn f für einige $x \in D$ nicht definiert sein könnte,
- **surjektiv**, wenn es zu jedem $y \in B$ ein Paar $(x, y) \in f$ gibt,
- **injektiv**, wenn es zu jedem $y \in B$ höchstens ein Paar $(x, y) \in f$ gibt,
- **bijektiv**, wenn f zugleich surjektiv und injektiv ist.

n-stellige Funktion: f: D -> B; der Definitionsbereich D ist ein Wertebereich von n-Tupeln

0-stellige Funktion k: -> B; der Definitionsbereich ist der **leere Wertebereich**;
k ist eine **konstante Funktion**; b = k() ist ein fester Wert

Kardinalität des Wertebereiches, aus dem Funktionen stammen $| D \to B | = (| B |+1)^{|D|}$

Anzahl der totalen Funktionen in $| D \to B |$ ist $| B |^{|D|}$

Anzahl der Möglichkeiten für unterschiedliche Funktionen mit dieser Signatur
z. B. $|\{A, B, C\} \to \{w, f\}| = 3^3 = 27$ insgesamt; $2^3 = 8$ totale Funktionen in $|\{A, B, C\} \to \{w, f\}|$

Spezielle Funktionen

Identitätsfunktion

$\mathbf{id_M}$: M -> M mit $id_{\mathbf{M}}$:= { (x, x) | $x \in M$ }

Charakteristische Funktion $\chi_{\mathbf{M}}$ einer Menge $M \subseteq U$, mit der Trägermenge U

gibt an, welche Elemente der Trägermenge U in M enthalten sind.

$\chi_{\mathbf{M}}$: U -> Bool mit $\chi_{\mathbf{M}}$:= { (x, b) | $x \in$ U und b = ($x \in M$)}

$\chi_{\mathbf{M}}$ ist eine totale Funktion

Funktionen mit Bildbereich Bool heißen **Prädikate**.

z. B. <= : ($\mathbb{N}_0 \times \mathbb{N}_0$) -> Bool

Funktionen zur Modellierung von **mehrfachen Vorkommen**:

b: V -> $\mathbb{N}_0$ gibt für jeden Wert aus V an, wie oft er vorkommt

z. B. geldBeutel: brdMünzen -> $\mathbb{N}_0$

mit geldBeutel := {(1,3), (2, 0), (5,0), (10, 2), (50, 1), (100, 3), (200, 2), (500, 1)}

So kann man sog. **Multimengen (bags)**, in denen Elemente mehrfach vorkommen können, beschreiben.

Funktionen auf Indexmengen

Indexmengen dienen zur Unterscheidung von Objekten des Modellbereiches
z. B. Ind = {1, ..., n}, KartenSymbole := {7, 8, 9, 10, Bube, Dame, König, Ass}

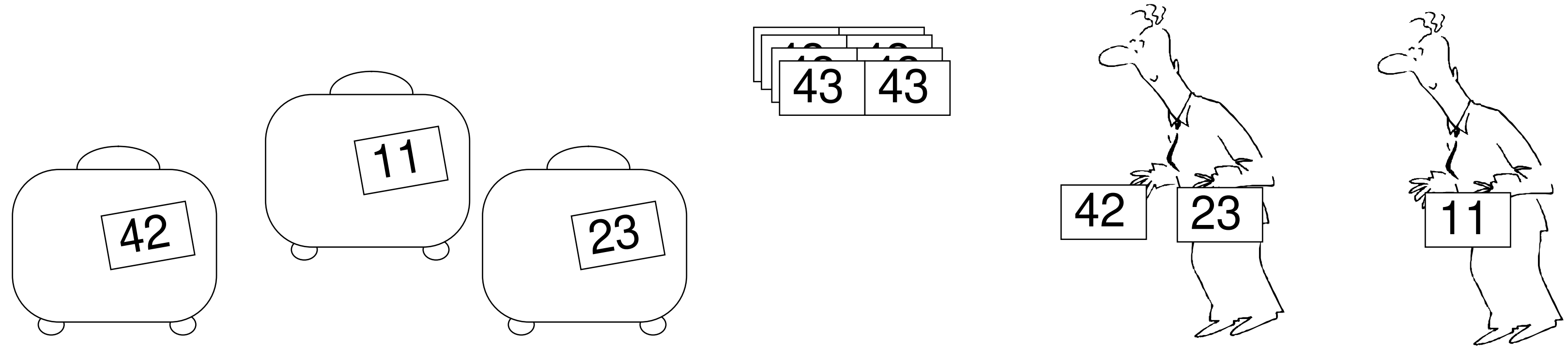
Funktionen auf Indexmengen modellieren ...

das Auftreten von Werten in Folgen:

Beispiel:

eine Folge	F := (w, e, l, l, e)
Indexmenge dazu	FPositionen := {1, 2, 3, 4, 5}
Werte in der Folge	FWerte := {w, e, l}
Auftreten von Werten in der Folge	FAuftreten := {(1, w), (2, e), (3, l), (4, l), (5, e)}
Wertebereich	FAuftreten $\in$ FPositionen -> FWerte

Zuordnungen zwischen Mengen:
z. B. Gepäckstücke ihren Eigentümern zuordnen durch ein Funktionenpaar
Marke1 $\in$ Ind -> Gepäckstücke (injektiv)   Marke2 $\in$ Ind -> Eigentümer



Disjunkte Vereinigung

Die allgemeine **disjunkte Vereinigung** fasst n Wertebereiche (Mengen) A_1, A_2, ..., A_n zu einem **vereinigten Wertebereich V** zusammen, wobei $I := \{1, ..., n\}$.
Die Herkunft der Elemente aus A_i wird in den Paaren von V gekennzeichnet:

$$\mathbf{V := \{ (i, a_i) \mid a_i \in A_i \}}$$

Die erste Komponente der Paare ist eine **Kennzeichenkomponente** (tag field).
Die A_i brauchen nicht paarweise disjunkt zu sein.

Kardinalität: $|V| = \sum_{i \in I} |A_i|$

Anwendungsmuster:

V ist ein allgemeinerer Wertebereich, abstrahiert von den spezielleren A_i

Beispiele:

Kunden := {Siemens, Benteler, Unity}   Lieferanten := {Orga, Siemens}
Ind := {Kunde, Lieferant}
GeschäftsPartner := { (Kunde, Siemens), (Kunde, Benteler), (Kunde, Unity),
(Lieferant, Orga), (Lieferant, Siemens)}

Buchstaben := {a, b, ..., z}   Ziffern := {0, 1, ..., 9}   Ind := {Buchstabe, Ziffer}

Zeichen = { (Buchstabe, b) | b $\in$ Buchstaben} $\cup$ { (Ziffer, z) | z $\in$ Ziffern}

Hinweise zum Modellieren mit Wertebereichen

- Erst Grundmengen festlegen, dann Strukturen darüber bilden.
- Typische Elemente eines Wertebereiches angeben - der Wertebereich ist eine Menge davon.
- Wertebereichen ausdruckskräftige Namen geben.
- Zusammengesetzte Wertebereiche schrittweise aufbauen (oder zerlegen).
- Entwürfe prüfen: Wertebereiche in Worten erklären.
- Nur gleichartige Elemente in einem Wertebereich.
- Mengen, Tupel und Folgen beliebiger Länge nicht verwechseln.
- Alle Klammern haben Bedeutung - zusätzliche verändern das Modell.

Wertebereiche zur Modellierung des Getränkeautomaten

Folgende Aspekte des Getränkeautomaten
können durch Wertebereiche Modelliert werden:

- Getränkevarianten
- Vorrat an Getränken und Zutaten
- Vorrat an Wechselgeld
- Eingeworfene Münzen
- Betätigte Wahltasten
- Anzeige des Automaten
- Zustand des Automaten
- weitere Aspekte ...

2.2 Terme Übersicht

In allen formalen Kalkülen benutzt man **Formeln als Ausdrucksmittel**.
Hier betrachten wir **nur ihre Struktur - nicht ihre Bedeutung**. Wir nennen sie **Terme**.

Terme bestehen aus **Operationen**, **Operanden**, **Konstanten und Variablen**:

a + 5   blau ? gelb = grün   ♥ > ♦

Terme werden nicht „ausgerechnet“.
Operationen, Konstanten und Variablen werden als **Symbole ohne Bedeutung** betrachtet.

Notation von Termen:

Infix-, **Postfix**-, **Präfix**- und **Baum**-Form

Umformung von Termen:

Grundlage für die Anwendung von Rechenregeln, Gesetzen

Für **Variable** in Termen werden Terme **substituiert:**

in   a + a = 2∗a   substituiere   a durch 3∗b   3∗b + 3∗b = 2∗3∗b

Unifikation: Terme durch Substitution von Variablen gleich machen,
z. B. um die Anwendbarkeit von Rechenregeln zu prüfen

Sorten und Signatur

Terme werden zu einer **Signatur** gebildet.
Sie legt die verwendbaren Symbole und die Strukturierung der Terme fest.

Signatur Σ := (S, F), S ist eine Menge von **Sorten**, F ist eine Menge von **Operationen**.

Sorte s $\in$ **S: Name** für einen Wertebereich; er braucht nicht weiter definiert zu sein.
z. B. STACK, ELEM, BOOL

Operation f $\in$ **F**: **Name** einer Funktion (Operatorsymbol), beschrieben durch
Anzahl der Operanden (**Stelligkeit)**, **Sorten der Operanden** und des **Ergebnisses**

Beispiele:	**Name**	**Operandensorten**	**Ergebnissorte**	
	push:	STACK x ELEM	-> STACK	2-stellig
	createStack:		-> STACK	0-stellig
	$\wedge$:	BOOL x BOOL	-> BOOL	2-stellig
	$\neg$:	BOOL	-> BOOL	1-stellig
	T:		-> BOOL	0-stellig

0-stellige Operationen sind Konstante, z. B. T, F, createStack

Beispiel für eine Signatur: Σ_{BOOL} := (S_{BOOL}, F_{BOOL}) mit
S_{BOOL} := { BOOL },
F_{BOOL} := { T: -> BOOL, F: -> BOOL, $\wedge$: BOOL x BOOL -> BOOL, $\neg$: BOOL-> BOOL}

Korrekte Terme

In **korrekten Termen** muss jeweils die Zahl der Operanden mit der **Stelligkeit** der Operation und die **Sorten** der Operandenterme mit den Operandensorten der Operation übereinstimmen.

Die **Menge τ der korrekten Terme der Sorte s** wird induktiv definiert:
Zu einer Signatur $\Sigma = (S, F)$ ist ein **korrekter Term der Sorte $s \in S$**:

- der **Name einer Variablen** der Sorte s, oder
- die **Anwendung einer n-stelligen Operation**
  $f: s_1 \times s_2 \times ... \times s_n \text{ -> } s \in \Sigma$ auf Terme t_i: $f(t_1, t_2, .., t_n)$
  wobei jedes **t_i ein korrekter Term der Sorte s_i** ist
  mit $n \geq 0$ (einschließlich Konstante bei $n = 0$) und $i \in \{1, ..., n\}$

Nur die nach diesen Regeln gebildeten Terme gehören zur Menge τ der korrekten Terme.

$f(t_1,..., t_n)$ ist ein **n-stelliger Term**; die t_i sind seine **Unterterme**.

Korrekte Terme, die **keine Variablen** enthalten, heißen **Grundterme**.

Beispiele: korrekte Terme zur Signatur Σ_{BOOL}: $F \quad \neg T \quad T \wedge x \quad \neg(a \wedge b) \quad x \wedge \neg y$

nicht korrekt: $a \neg b \quad \neg(\wedge b)$

Notationen für Terme

Notation eines n-stelligen Terms mit Operation (Operator) f und Untertermen $t_1, t_2, ..., t_n$:

Bezeichnung	Notation	Beispiele
Funktionsform:	Operator **vor der geklammerten Folge** seiner Operanden $f\ (t_1, t_2, ..., t_n)$	 empty (push (createStack, c)) $\wedge$ (< (0, a), < (a, 10))
Präfixform:	Operator **vor** seinen Operanden $f\ \ t_1\ \ t_2 \ldots t_n$	 $\wedge$ < 0 a < a 10
Postfixform:	Operator **nach** seinen Operanden $t_1\ \ t_2 \ldots t_n\ \ f$	 0 a < a 10 < $\wedge$
Infixform	2-stelliger Operator **zwischen** seinen (beiden) Operanden $t_1\ f\ \ t_2$	 0 < a $\wedge$ a < 10

Die **Reihenfolge der Operanden** ist in allen vier Notationen **gleich**.

Präzedenzen und Klammern

Die **Infixform** benötigt Klammern oder Präzedenzen, um Operanden an ihren Operator zu binden: Ist in x + 3 * y die 3 rechter Operand des + oder linker Operand des * ?

Klammern beeinflussen die Struktur von Termen in der Infixform:
  z. B. (x + 3) * y oder x + (3 * y)
Redundante Klammern sind zulässig.
Ein Term ist **vollständig geklammert**, wenn er und alle seine Unterterme geklammert sind:
  z. B. ((x) + ((3) * (y)))

Für die **Infixform** können den Operatoren unterschiedliche **Bindungsstärken (Präzedenzen)** zugeordnet werden, z. B. bindet * seine Operanden vereinbarungsgemäß stärker an sich als +, d. h. * hat **höhere Präzedenz** als +.
  Damit sind x + 3 * y und x + (3 * y) verschiedene Schreibweisen für denselben Term.

Für **aufeinanderfolgende Operatoren gleicher Präzedenz** muss geregelt werden, ob sie ihre Operanden links-assoziativ oder rechts-assoziativ binden:

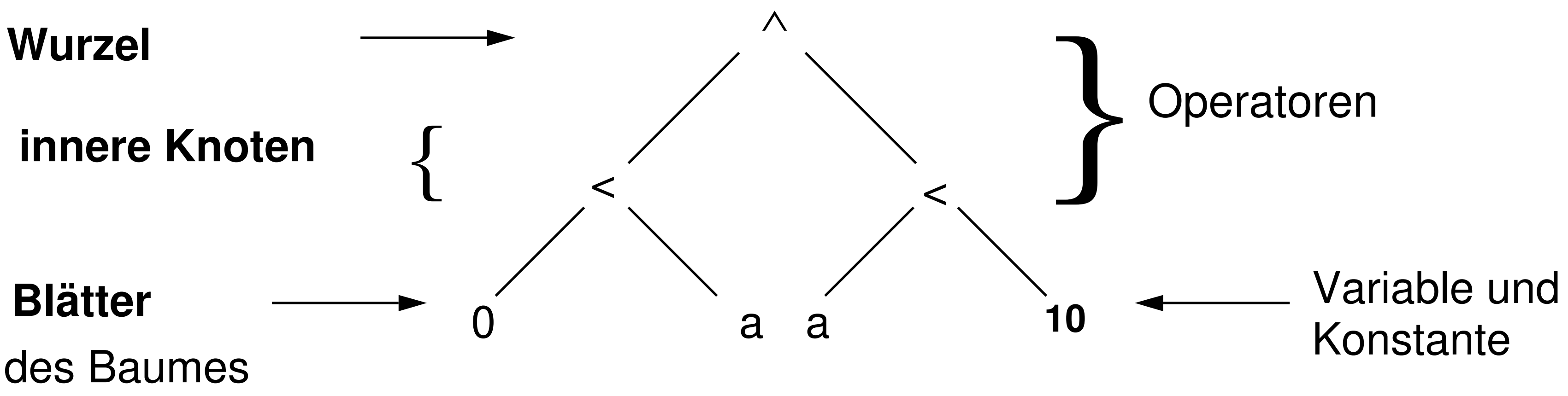
links-assoziativ:	x + 3 + y	steht für (x + 3) + y
rechts-assoziativ:	x ** 3 ** y	steht für x ** (3 ** y)

Funktionsform, Präfixform, Postfixform benötigen
  weder Präzedenzen noch zusätzliche Klammern!

Terme als Bäume

Terme kann man als Bäume darstellen (Kantorowitsch-Bäume):

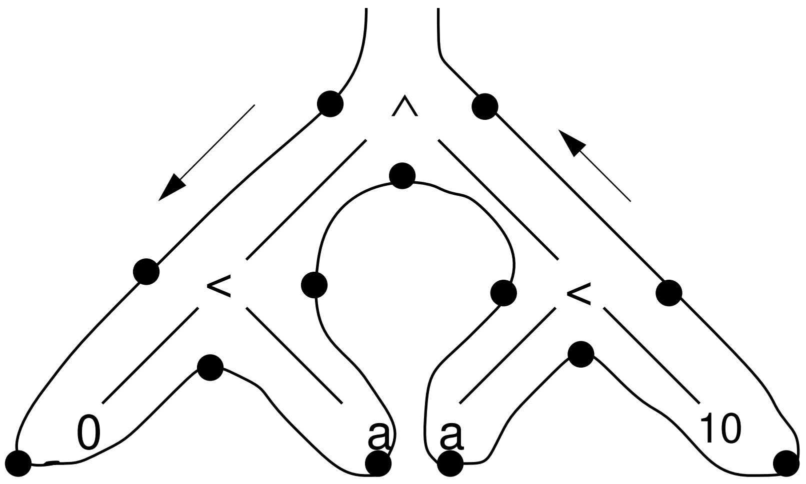
Term: $0 < a \wedge a < 10$



Aus einem Durchlauf des Baumes in Pfeilrichtung erzeugt man

- **Präfixform**, wenn man beim **ersten Besuch**
- **Postfixform**, wenn man beim **letzten Besuch**
- **Infixform**, wenn man beim **vorletzten Besuch** (bei 2-stelligen Operatoren)

den Operator aufschreibt.



Substitution

Eine **Substitution** beschreibt, wie in einem Term vorkommende **Variable durch Terme ersetzt** werden.

Eine **einfache Substitution** σ **= [v / t]** ist ein Paar aus einer Variablen v und einem Term t zur Signatur ∑. v und t müssen **dieselbe Sorte** s haben.
Beispiel: σ = [x / 2*b]

Die **Anwendung einer Substitution** σ **auf einen Term** u schreibt man **u** σ, z. B. (x+1) [x / 2*b].

Die **Anwendung einer Substitution u** σ mit σ **= [v / t]**, ist **definiert** durch

- u σ = t , falls u die zu ersetzende Variable v ist,
- u σ = u , falls u ≠ v und u eine Konstante oder eine andere Variable ist,
- u σ = f (u_1σ, u_2σ, ..., u_nσ) , falls u = f (u_1, u_2, ..., u_n)

D. h. in u werden **alle Vorkommen der Variablen v gleichzeitig durch den Term t ersetzt**.

Kommt v auch in t vor, so wird es nicht nochmals ersetzt!

Beispiele: (x + 1) [x / 2*b] = (2*b + 1)

(x - x) [x / 3] = (3 - 3)

(x + y) [y / y*y] = (x + y*y)

Mehrfache Substitution

Eine **mehrfache Substitution** $\sigma = [\, \mathbf{x_1} / \mathbf{t_1}, ..., \mathbf{x_n} / \mathbf{t_n}]$ mit paarweise verschiedenen Variablen $\mathbf{x_i}$ steht für die
gleichzeitige Substitution aller Vorkommen aller Variablen $\mathbf{x_i}$ jeweils durch die Terme t_i.

Beispiele: $\sigma = [\, x / 2*b, y / 3 \,]$

$(x + y)\, \sigma = (2*b + 3)$
$(y + a*y)\, \sigma = (3 + a*3)$
$(x * y)\, [\, x / y,\ y / y*y \,] = (y * (y * y))$

Auf einen Term können **mehrere Substitutionen hintereinander** ausgeführt werden,
z. B. $u\, \sigma_1\, \sigma_2\, \sigma_3 = ((u\, \sigma_1)\, \sigma_2)\, \sigma_3$

$(x+y)\, [x/y*x]\, [y/3]\, [x/a] = (y*x+y)\, [y/3]\, [x/a] = (3*a+3)$

Mehrere einfache **Substitutionen hintereinander** kann man **in eine mehrfache Substitution** mit gleicher Wirkung umrechnen:
Die Hintereinanderausführung $[\, x_1 / t_1, ..., x_n / t_n \,]\, [\, y / r \,]$

hat auf jeden Term die gleiche Wirkung wie

falls y unter den x_i vorkommt $[\, x_1 / (t_1[\, y / r \,]), ..., x_n / (t_n\, [\, y / r \,])]$
falls y nicht unter den x_i vorkommt $[\, x_1 / (t_1[\, y / r \,]), ..., x_n / (t_n\, [\, y / r \,]), y / r \,]$

Beispiel: $[\, x / y*x \,]\, [\, y / 3 \,]\, [\, x / a \,] = [\, x / 3*x, y / 3]\, [\, x / a \,] = [\, x / 3*a, y / 3 \,]$

Umfassende Terme

Rechenregeln werden mit **allgemeineren Termen** formuliert, die auf **speziellere Terme** angewandt werden,

z. B. Distributivgesetz:	$a * (b + c)$	$=$	$a * b + a * c$
angewandt auf	$2 * (3 + 4*x)$	$=$	$2 * 3 + 2 * 4*x$

Ein **Term s umfasst einen Term t**, wenn es eine Substitution σ gibt, die s in t umformt: $s\ \sigma = t$

s ist ein **allgemeineres** Muster; es umfasst den **spezielleren** Term t.

s umfasst t, ist eine **Quasiordnung**, d. h. die Relation **umfasst** ist

transitiv:	sei $r\ \sigma_1 = s$, $s\ \sigma_2 = t$, dann ist $r\ (\sigma_1\ \sigma_2) = t$
reflexiv:	$t\ [\,] = t$, mit der leeren Substitution [ ]
nicht antisymmetrisch:	Terme, die sich nur in den Variablennamen unterscheiden, kann man ineinander umformen, z. B. $2*x\ [\ x\ /\ y\ ] = 2*y$ und $2*y\ [\ y\ /\ x\ ] = 2*x$

Unifikation

Bei der Relation „umfasst“ wird ein Term mit einer Substitution in einen anderen transformiert.

Bei der **Unifikation** werden **zwei Terme so substituiert, dass sie gleich werden**.

Zwei Terme s und t sind unifizierbar, wenn es eine Substitution σ **gibt mit s** $\sigma =$ **t** σ.
σ heißt **Unifikator von s und t**.

Beispiel: Terme: s = (x + y) t = (2 + z)
Unifikatoren: $\sigma_1 = [\, x / 2, y / z \,]$ $\sigma_2 = [\, x / 2, z / y \,]$,
$\sigma_3 = [\, x / 2, y / 1, z / 1 \,]$ $\sigma_4 = [\, x / 2, y / 2, z / 2 \,]$ ...

Ist σ **ein Unifikator** von s und t und τ **eine Substitution**, dann ist auch die Hintereinanderausführung $\sigma\,\tau = \sigma'$ auch ein Unifikator von s und t.
Wir sagen σ **ist allgemeiner als** σ'**,** denn t σ umfasst t σ'.

Im Beispiel sind σ_1 und σ_2 allgemeiner als σ_3 und σ_4 .

Ein Unifikator σ heißt **allgemeinster Unifikator** der Terme s und t, wenn es zu allen anderen Unifikatoren σ' eine Substitution τ gibt mit $\sigma\,\tau = \sigma'$.

Im Beispiel sind σ_1 und σ_2 allgemeinste Unifikatoren, z. B. $\sigma_1 [\, z / 1 \,] = \sigma_3$

Es kann **mehrere allgemeinste Unifikatoren** geben. Sie können durch **Umbenennen von Variablen** ineinander überführt werden, z. B. $\sigma_1 [\, z / y \,] = [\, x / 2, y / y, z / y \,] = \sigma_2$

Unifikationsverfahren

Unifikation zweier Terme s und t nach Robinson:

Seien s und t Terme in **Funktionsschreibweise**.
Dann ist das **Abweichungspaar A(s, t) = (u, v)** das erste Paar unterschiedlicher, korrespondierender Unterterme u und v, das man beim Lesen von links nach rechts antrifft.

Algorithmus:

1. Setze σ = [ ] (leere Substitution)
2. Solange es ein Abweichungspaar A(s σ, t σ) = (u, v) gibt wiederhole:
   a. ist **u eine Variable x**, die in v nicht vorkommt, dann ersetze σ durch σ [x / v], oder
   b. ist **v eine Variable x**, die in u nicht vorkommt, dann ersetze σ durch σ [x / u],
   c. **sonst** sind die Terme s und t **nicht unifizierbar; Abbruch** des Algorithmus.
3. Bei Erfolg gilt s σ = t σ und σ **ist allgemeinster Unifikator**.

Beachte, dass bei jeder Iteration die bisherige Substitution auf die vollständigen Terme s, t angewandt wird.

Beispiel für Unifikationsverfahren

Unifikation zweier Terme s und t nach Robinson:

$s = +(*(2, x), 3)$
$t = +(z, \quad x)$

$\sigma = [\,]$

Schritt

↓ Abweichungspaar

1 $s\sigma = +(*(2, x), 3)$
$t\sigma = +(z, \quad x)$

Fall 2b: $\sigma = [\,]\,[z / *(2, x)]$

↓ Abweichungspaar

2 $s\sigma = +(*(2, x), 3)$
$t\sigma = +(*(2, x), x)$

Fall 2b: $\sigma = [\,]\,[z / *(2, x)]\,[x / 3]$

3 $s\sigma = +(*(2, 3), 3)$
$t\sigma = +(*(2, 3), 3)$

allgemeinster Unifikator: $\sigma = [z / *(2, x)]\,[x / 3] = [z / *(2, 3), x / 3]$

2.3 Algebren

Eine **Algebra** ist eine **formale Struktur** definiert durch eine Trägermenge, Operationen drüber und Gesetze zu den Operationen.

In der Modellierung der Informatik spezifiziert man mit Algebren
Eigenschaften veränderlicher Datenstrukturen und dynamische Systeme,
z. B. Datenstruktur *Keller* oder die Bedienung eines Getränkeautomaten.

Wir unterscheiden 2 Ebenen: abstrakte Algebra und konkrete Algebra:

Eine **abstrakte Algebra** spezifiziert Eigenschaften abstrakter Operationen,
definiert nur duch eine Signatur - Realisierung durch Funktionen bleibt absichtlich offen

Trägermenge: korrekte Terme zu einer Signatur

Gesetze identifizieren Terme mit gleicher Bedeutung:
z. B. $\neg$ false $\equiv$ true pop (push (k, t)) $\equiv$ k

Eine **konkrete Algebra** zu einer abstrakten Algebra

definiert **konkrete Funktionen** zu den Operationen der Signatur,
so dass die Gesetze auch für die Funktionsterme gelten.

Sie beschreibt so eine **Implementierung** der spezifizierten Datenstruktur,
bzw. des Systems

Abstrakte Algebra

Eine **abstrakte Algebra $\mathbf{A} = (\tau, \Sigma, Q)$** ist definiert durch die
Menge korrekter Terme τ zur **Signatur** Σ und einer **Menge von Axiomen Q**.

Ein **Axiom** hat die Form $t_1 \equiv t_2$, wobei t_1, t_2, **korrekte Terme gleicher Sorte** sind und **Variable** enthalten können.

Die Algebra definiert alle Paare von Termen als **gleichbedeutend**, die man mit den Axiomen ineinander umformen kann.

Mit Axiomen umformen heißt:
Unter Anwenden eines Axioms $t_1 \equiv t_2$ kann man einen Term s_1 in einen Term s_2 oder s_2 in s_1 umformen, wenn gilt:

- s_1 und s_2 stimmen in ihren „äußeren" Strukturen überein und unterscheiden sich nur durch die Unterterme r_1 und r_2 an entsprechenden Positionen in s_1 und s_2, und
- es gibt eine Substitution σ , sodass gilt $t_1\, \sigma = r_1$ und $t_2\, \sigma = r_2$

$$
\begin{array}{llcccccc}
\text{Terme} & s_1 & = & \ldots\ldots\ldots\ r_1\ \ldots\ldots\ldots & & \ldots\ldots\ldots\ r_2\ \ldots\ldots\ldots & = & s_2 \\
 & & & \shortparallel & & \shortparallel & & \\
 & & & t_1\, \sigma & & t_2\, \sigma & & \\
\text{Axiom} & & & t_1 & \equiv & t_2 & &
\end{array}
$$

Beispiel: abstrakte Algebra Bool

Signatur Σ = ({BOOL}, F)

Operationen F:

true:		-> BOOL
false:		-> BOOL
$\wedge$:	BOOL x BOOL	-> BOOL
$\vee$:	BOOL x BOOL	-> BOOL
$\neg$:	BOOL	-> BOOL

Axiome Q: für alle x,y der Sorte BOOL gilt

Q_1:	$\neg$ true	$\equiv$ false
Q_2:	$\neg$ false	$\equiv$ true
Q_3:	true $\wedge$ x	$\equiv$ x
Q_4:	false $\wedge$ x	$\equiv$ false
Q_5:	x $\vee$ y	$\equiv \neg (\neg x \wedge \neg y)$

Es wird außerdem gefordert, dass true und false **verschiedene** Konstante sind.

Konkrete Algebra

Zu einer abstrakten Algebra $\mathrm{A} = (\tau, \Sigma, Q)$, kann man
konkrete Algebren $\mathrm{A} = (W_A, F_A, Q)$
angeben, wobei
W_A eine **Menge von Wertebereichen**, je einer **für jede Sorte** aus S von Σ
F_A eine **Menge von Funktionen**, je eine **für jede Operation** aus F von Σ.

Die Definitions- und Bildbereiche der Funktionen müssen konsistent den Sorten der Operationen zugeordnet werden.

Die **Axiome Q** müssen mit den Funktionen F_A **für alle entsprechenden Terme aus τ gelten**.

Es können in der konkreten Algebra noch weitere Gesetze gelten.

Eine konkrete Algebra heißt auch **Modell der abstrakten Algebra**.

Beispiel für eine konkrete Algebra

Beispiel: eine konkrete Algebra FSet zur abstrakten Algebra Bool:

	konkrete Algebra FSet	**abstrakte Algebra Bool**
W_A:	$\{\emptyset, \{1\}\}$	Sorte BOOL
F_A:	$\{1\}$	true
	$\emptyset$	false
	Mengendurchschnitt $\cap$	$\wedge$
	Mengenvereinigung $\cup$	$\vee$
	Mengenkomplement bezüglich $\{1\}$	$\neg$

Axiome Q:
Man kann zeigen, dass die Axiome für die entsprechenden Terme mit F_A gelten:
z. B. $\emptyset \cap x \equiv \emptyset$ entspricht false $\wedge$ x $\equiv$ false

Die übliche boolsche Algebra ist natürlich auch eine konkrete Algebra zur abstrakten Algebra Bool.

Beispiel 2.2: Datenstruktur Keller

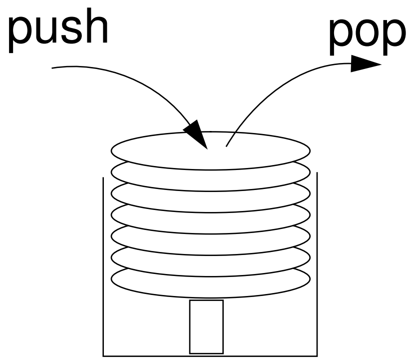
Die Eigenschaften einer **Datenstruktur Keller** beschreiben wir zunächst informell.
Folgende **Operationen** kann man mit einem Keller ausführen:

create Stack:	liefert einen leeren Keller
push:	fügt ein Element in den Keller ein
pop:	entfernt das zuletzt eingefügte Element
top:	liefert das zuletzt eingefügte und nicht wieder entfernte Element
empty:	gibt an, ob der Keller leer ist.

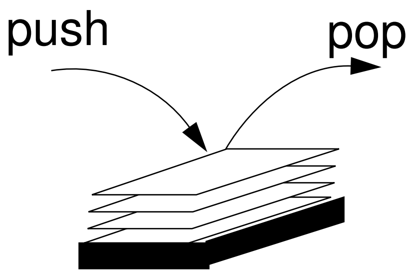
Die Eigenschaften der Datenstruktur Keller sollen präzise durch eine abstrakte Algebra spezifiziert werden.

Beispiele

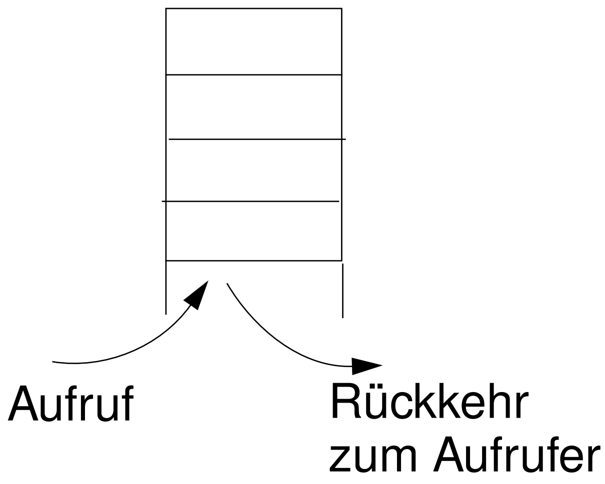
Tellerstapel



Aktenstapel



Laufzeitkeller



Beispiel: Abstrakte Algebra spezifiziert Keller

Abstrakte Algebra Keller:

Signatur Σ = (S, F),
Sorten S = {Keller, Element, BOOL},
Operationen F:

createStack:		-> Keller
push:	Keller x Element	-> Keller
pop:	Keller	-> Keller
top:	Keller	-> Element
empty:	Keller	-> BOOL

Axiome Q: für beliebige Terme t der Sorte Element und k der Sorte Keller gilt:

K1:	empty (createStack)	$\equiv$ true
K2:	empty (push (k, t))	$\equiv$ false
K3:	pop (push (k, t))	$\equiv$ k
K4:	top (push (k, t))	$\equiv$ t

Keller ist die Sorte, deren Terme Kellerinhalte modellieren.
Element und BOOL sind **Hilfssorten** der Algebra.

Implementierungen der abstrakten Algebra Keller können durch **konkrete Algebren** dazu beschrieben werden.

Klassifikation von Operationen

Die Operationen einer Algebra, die die **definierte Sorte als Ergebnissorte** (hier: Keller) haben, kann man in 2 disjunkte Mengen einteilen: **Konstruktoren** und **Hilfskonstruktoren**.

Die Konstruktoren müssen so gewählt sein, dass man **jeden Grundterm mit den Axiomen** in einen gleichwertigen **umformen** kann, der **nur Konstruktoren** enthält. (Ausnahme s. u.)

z. B. in der Keller-Algebra:
Konstruktoren: push, createStack; Hilfskonstruktor: pop
mit dem Axiom K3 kann man Vorkommen der Operation pop aus Termen entfernen

Wir sagen: Grundterme, in denen nur **Konstruktoren** vorkommen, sind in **Normalform**.
z. B. push (.... push (createStack, n_1) , ...), n_m), mit $m \geq 0$

Eigenschaften von Termen in Normalform, kann man auf gleichwertige Terme **übertragen**.

Operationen, die eine **andere Ergebnissorte** haben, heißen **Projektionen**. (hier: top, empty).

Undefinierte Terme:

Terme der definierten Sorte, die man **nicht in eine Normalform** umformen kann, werden als **undefiniert** angesehen. Sie modellieren eine **Fehlersituation**, z. B. pop (createStack)

Manche **Projektionen** sind nicht für jeden Term in Normalform definiert.
Dies modelliert auch **Fehlersituationen**, z. B.top (createStack)

Anwendungen algebraischer Spezifikationen: Eigenschaften aus den Axiomen erkennen

Beispiel: Keller

1. K3: pop (push (k, t)) ≡ k

   Keller-Prinzip: zuletzt eingefügtes Element wird als erstes wieder entfernt (last-in-first-out, LIFO)

2. top: Keller -> Element
   K4: top (push (k, t)) ≡ t

   top ist die einzige Operation, die Keller-Elemente liefert:
   Nur auf das zuletzt eingefügte, nicht wieder entfernte Element kann **zugegriffen** werden.

3. push (.... push (createStack, n_1) , ...), n_m), mit $m \geq 0$
   K3: pop (push (k, t)) ≡ k

   Die **Anzahl der Elemente im Keller** ist
   die Anzahl der push-Operationen abzüglich der Anzahl der pop-Operationen im Term.
   Begründung: Rückführung auf Normalform, eine push-Operation für jedes Element im Keller.

Spezifikation um Operationen erweitern

Erweitere die Keller-Spezifikation um eine **Operation size**.
Sie soll die **Anzahl der Elemente im Keller** liefern.

1. Operation **size in die Signatur einfügen:**
   size: Keller -> N

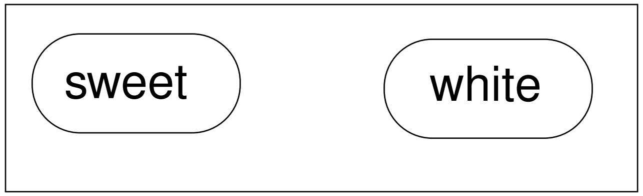
2. Ergebnis-Sorte **N zu den Sorten zufügen**:
   S = {Keller, Element, BOOL, N}

3. **Axiome zufügen**, so dass size für jeden Keller-Wert definiert ist:
   K7: size (createStack) ≡ 0
   K8: size (push (k, t)) ≡ size (k) + 1

Weil in der **Normalform** nur createStack und push vorkommen, braucht size nur für solche Terme definiert zu werden.

Es wird vorausgesetzt, dass für die Sorte N die Konstanten 0 und 1 sowie die Operation + definiert sind.

Abstrakte Algebra für Teilaspekt des Getränkeautomaten



Knöpfe des Getränkeautomaten
zur Auswahl von Zutaten

Signatur Σ = (S, F);

Sorten S := {Add, Choice}

Operationen F:

sweet:	-> Add
white:	-> Add
noChoice:	-> Choice
press: Add x Choice	-> Choice

Die Sorte **Choice** modelliert die Auswahl;
Add ist eine Hilfssorte

Bedeutung der Axiome:

Q_1: Knopf nocheinmal drücken macht Auswahl rückgängig.

Q_2: Es ist egal, in welcher Reihenfolge die Knöpfe gedrückt werden.

Axiome Q: für alle a, b der Sorte Add und
für alle c der Sorte Choice gilt:

Q_1: press (a, press (a, c)) $\equiv$ c

Q_2: press (a, press (b, c)) $\equiv$ press (b, press (a, c)

Beispiel-Terme: press (white, noChoice)
press (sweet, press (white, press (sweet, noChoice)))

Realisierung der Spezifikation durch eine konkrete Algebra

Beispiel: eine Realisierung von Kellern durch **Funktionen auf Folgen** von natürlichen Zahlen:

Zuordnung der Sorten:

konkret	**abstrakt**
Bool	BOOL
$\mathbb{N}_0$	Element
N-Folge = $\mathbb{N}_0{}^*$	Keller

Signatur und **Zuordnung von Funktionen**

konkret			**abstrakt**
newFolge:		-> N-Folge	createStack
append:	N-Folge x $\mathbb{N}_0$	-> N-Folge	push
remove:	N-Folge	-> N-Folge	pop
last:	N-Folge	-> $\mathbb{N}_0$	top
noElem:	N-Folge	-> Bool	empty

Definition der Funktionen

newFolge( )	-> ()
append $((a_1, ..., a_n), x)$	-> $(a_1, ..., a_n, x)$
remove $((a_1, ..., a_{n-1}, a_n))$	-> $(a_1, ..., a_{n-1})$
last $((a_1, ..., a_n))$	-> a_n
noElem (f)	-> f = ()

Gültigkeit der Axiome zeigen

Keller in Algorithmen einsetzen

Aufgabe: Terme aus **Infix-Form in Postfix-Form** umwandeln
gegeben: Term t in Infix-Form, mit 2-stelligen Operatoren unterschiedlicher Präzedenz, zunächst ohne Klammern
gesucht: Term t in Postfix-Form

Eigenschaften der Aufgabe und der Lösung:

1. **Reihenfolge der Variablen und Konstanten bleibt unverändert**
2. **Variablen und Konstanten werden vor ihrem Operator ausgegeben**, also sofort
3. In der Infix-Form aufeinander folgende **Operatoren echt steigender Präzedenz** stehen in der Postfix-Form **in umgekehrter Reihenfolge**; also kellern.
4. **Operatorkeller enthält Operatoren echt steigender Präzedenz**.
   Es gilt die **Kellerinvariante KI**:
   Sei push (... push (CreateStack, opr_1), opr_2), ...) dann gilt
   Präzedenz ($opr_{\mathbf{i}}$) < Präzedenz ($opr_{\mathbf{i+1}}$)

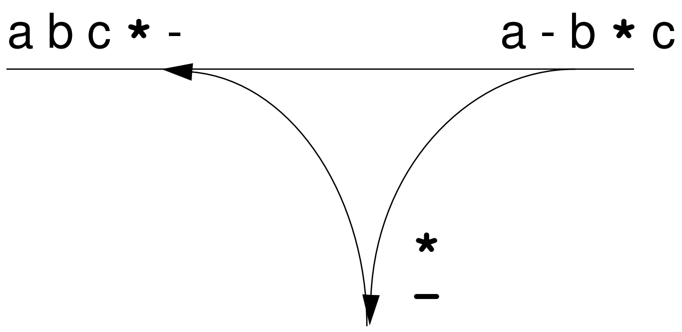
Algorithmus: Infix- in Postfix-Form wandeln

Die Eingabe enthält einen Term in Infix-Form;
die Ausgabe soll den Term in Postfix-Form enthalten

```
Variable: keller ∈ Keller; symbol ∈ Operator ∪ ElementarOperand

keller = createStack();
solange Eingabe nicht leer wiederhole                   {KI}
            lies symbol
            falls symbol ∈ ElementarOperand
                    gib symbol aus
            falls symbol ∈ Operator                     {KI}
                    solange   not empty (keller) ∧
                              Präzedenz (top (keller)) ≥ Präzedenz (symbol)
                    wiederhole                          {KI}
                              gib top (keller) aus;
                              keller = pop (keller);
                    keller = push(keller, symbol);      {KI}

solange not empty (keller) wiederhole
            gib top(keller) aus;
            keller = pop(keller);
```



An den Stellen {KI} gilt die Kellerinvariante.

3. Logik
3.1 Aussagenlogik

Kalkül zum **logischen Schließen**. Grundlagen: Aristoteles 384 - 322 v. Chr.

Aussagen: Sätze, die prinzipiell als wahr oder falsch angesehen werden können.
z. B.: „Es regnet.“, „Die Straße ist nass.“
aber „Dieser Satz ist falsch.“ ist in sich widersprüchlich, ist keine Aussage.

Junktoren verknüpfen Aussagen: „Es regnet nicht, **oder** die Straße ist nass.“

Aussagenlogische Formeln als Sätze einer formale Sprache:

z. B.	regen	$\rightarrow$	straßeNass	$\leftrightarrow$	$\neg$	regen	$\vee$	straßeNass
Belegung der Aussagen mit **Wahrheitswerten**:	f		w			f		w
Interpretation der Formel liefert Wahrheitswert:		w			w			
							w	
				w				

Formales Schließen im Gegensatz zur empirischen Beurteilung, z. B. ob „die Straße nass ist.“

Aus „Wenn es regnet, ist die Straße nass.“ **und** „Es regnet.“ **folgt** „Die Straße ist nass.“

Aussagen in der **Spezifikation**, in der **Modellierung** von Aufgaben

Vorschau auf Begriffe

- **Aussagenlogische Formeln** definiert durch **Signatur der boolschen Algebra**
- **Belegung von Variablen** mit Wahrheitswerten
- **Interpretation** aussagenlogischer Formeln
- **Gesetze der boolschen Algebra** zur Umformung von Formeln
- **erfüllbare** und **allgemeingültige** Formeln
- **logischer Schluss:** Folgerung aus einigen Annahmen

Beispiel: Aussagenlogik in der Spezifikation

Unfall durch fehlerhafte Spezifikation:

Airbus A320, Warschau (1993). Der zuständige Rechner blockiert bei der Landung die Aktivierung von Schubumkehr und Störklappen, wodurch das Flugzeug über das Landebahnende hinausschießt. Es herrschen starker Wind von schräg hinten und Aquaplaning auf der Landebahn.

Beabsichtigte Spezifikation der Störklappenfreigabe:

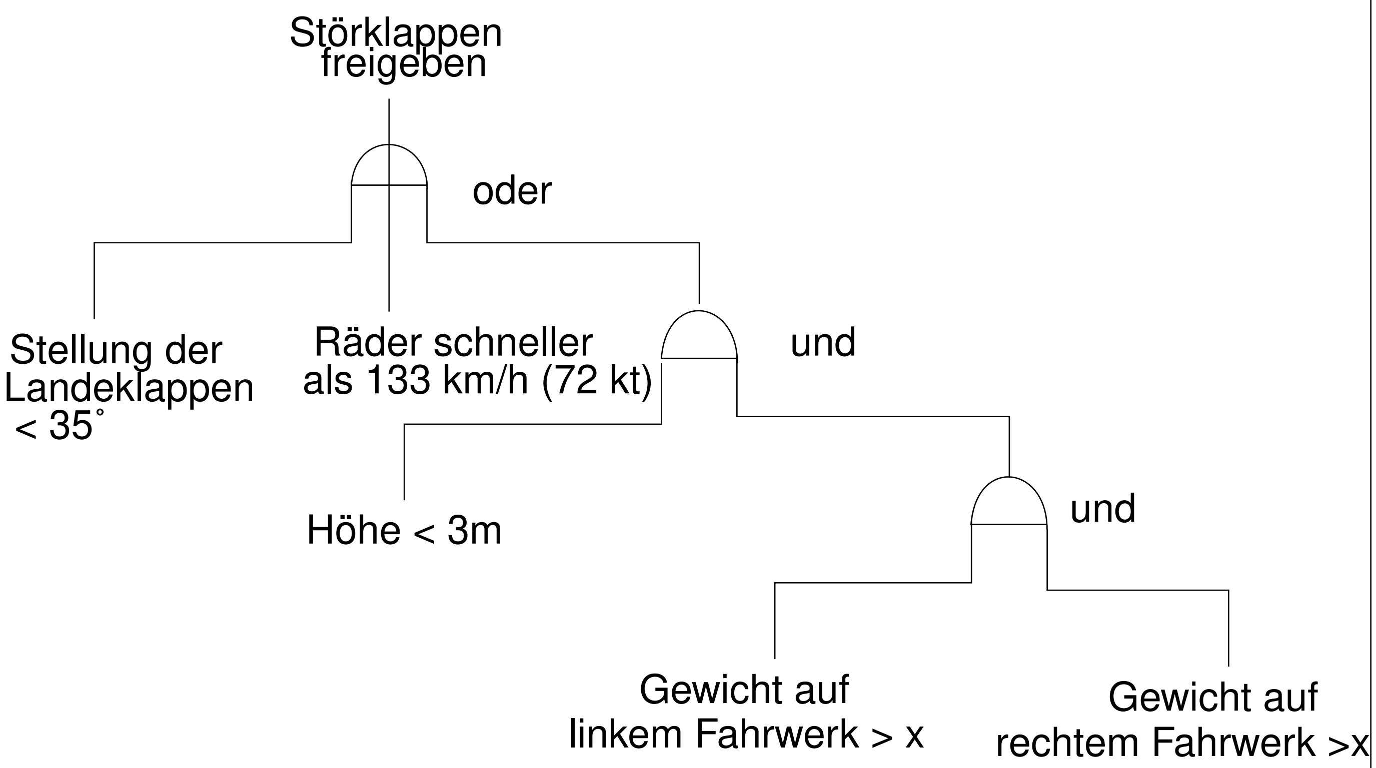
Die Störklappen dürfen benutzt werden

- im Reise- und Sinkflug (Bremswirkung)
- nach der Landung (Vernichtung des Auftriebes und Bremswirkung)

Sie dürfen nicht benutzt werden

- im Endanflug (gefährlicher Auftriebsverlust)

Tatsächliche Spezifikation der Störklappenfreigabe:



Aussagenlogische Formeln

Aussagenlogische Formeln sind korrekte Terme mit Variablen zur Signatur der boolschen Algebra:

O:		-> Bool	falsch
L:		-> Bool	wahr
$\wedge$:	Bool x Bool	-> Bool	**Konjunktion**
$\vee$:	Bool x Bool	-> Bool	**Disjunktion**
$\neg$:	Bool	-> Bool	**Negation**

Erweiterung:

$\rightarrow$:	Bool x Bool	-> Bool	**Implikation** $p \rightarrow q$ für $\neg p \vee q$
$\leftrightarrow$:	Bool x Bool	-> Bool	**Äquivalenz** $p \leftrightarrow q$ für $(p \rightarrow q) \wedge (q \rightarrow p)$

Operatoren (**Junktoren**) in **fallender Präzedenz**: $\neg \;\; \wedge \;\; \vee \;\; \rightarrow \;\; \leftrightarrow$

Variable, sowie O und L (Konstante) sind **atomare Aussagen**,
die übrigen Formeln sind **zusammengesetzt**.

Für **Variable** schreiben wir meist kleine Buchstaben p, q, ...
für **allgemeine Formeln** große Buchstaben F, G, H, ...

Die Definition der **Struktur** der Formeln heißt **Syntax der Aussagenlogik**.

Interpretation aussagenlogischer Formeln

Eine **passende Belegung** ordnet allen Variablen, die in einer Menge von Formeln $\mathcal{F}$ vorkommen, jeweils einen Wahrheitswert w oder f für wahr oder falsch zu.
Die Belegung kann als Substitution angegeben werden, z.B. σ = [p/w, q/f].

Eine **Interpretation** $\mathfrak{I}_\sigma$ einer aussagenlogischen Formel F bildet F auf einen Wahrheitswert ab:

- Für **Variable** ist die Interpretation $\mathfrak{I}_\sigma$ durch die **Belegung** σ definiert.
- Für **zusammengesetzte Formeln** wird sie durch folgende **Wahrheitstafeln** erweitert:

$\mathfrak{I}(O) = f$

$\mathfrak{I}(L) = w$

$\mathfrak{I}(F)$	$\mathfrak{I}(\neg F)$
w	f
f	w

$\mathfrak{I}(F)$	$\mathfrak{I}(G)$	$\mathfrak{I}(F \wedge G)$	$\mathfrak{I}(F \vee G)$	$\mathfrak{I}(F \rightarrow G)$	$\mathfrak{I}(F \leftrightarrow G)$
w	w	w	w	w	w
w	f	f	w	f	f
f	w	f	w	w	f
f	f	f	f	w	w

Eine Interpretation $\mathfrak{I}_\sigma$ mit einer Belegung σ für eine Formel F bestimmt einen **Wahrheitswert der Formel F:** $\mathfrak{I}_\sigma(F)$

Wenn $\mathfrak{I}_\sigma(F) = w$ gilt, heißt $\mathfrak{I}_\sigma$ auch ein **Modell der Formel F.**

Vorsicht beim Formalisieren umgangssprachlicher Aussagen

Vorsicht bei **Implikationen**; mit Belegungen prüfen, was gemeint ist:

1. **Wenn** es regnet, benutze ich den Schirm. $\quad$ regnet $\rightarrow$ Schirm
2. Ich benutze den Schirm, **wenn** es regnet. $\quad$ regnet $\rightarrow$ Schirm
3. Ich benutze den Schirm, **nur wenn** es regnet. $\quad$ Schirm $\rightarrow$ regnet

„Oder“ kann fast immer in das **nicht-ausschließende** $\vee$ übersetzt werden:

4. Hast Du ein Markstück oder zwei Fünfziger? $\quad$ mark $\vee$ zwei50er
5. Morgen fahre ich mit dem Zug oder mit dem Auto nach Berlin. $\quad$ zug $\vee$ auto
6. x ist kleiner y oder x ist gleich y. $\quad$ $x<y \vee x=y$
7. Der Händler gibt Rabatt oder ein kostenloses Autoradio. $\quad$ $\neg$ (rabatt $\leftrightarrow$ radio)

Aussagen sind häufig **kontext-abhängig**:

8. Weil ich die SWE-Klausur nicht bestanden habe,
   nehme ich am zweiten Termin teil. $\quad$ $\neg$ swe-k $\wedge$ swe-k2
9. Weil ich die Modellierungs-Klausur bestanden habe,
   nehme ich am zweiten Termin nicht teil. $\quad$ mod-k $\wedge$ $\neg$ mod-k2

Klammern sind meist nur aus dem Kontext erkennbar:

10. Sie wollten nicht verlieren oder unentschieden spielen. $\quad$ $\neg$ (verlieren $\vee$ unentschieden)

Erfüllbarkeit von Formeln

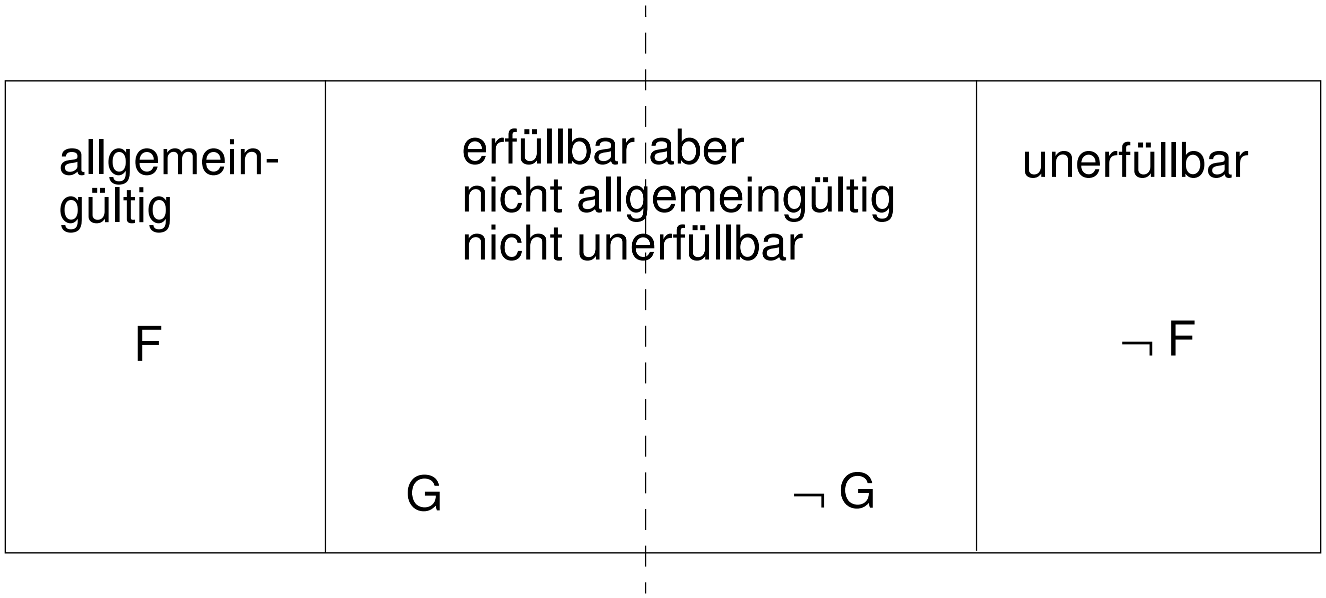
Eine Formel F heißt **erfüllbar**, wenn es
eine Interpretation $\mathfrak{I}_\sigma$ mit einer Belegung σ gibt, so dass gilt $\mathfrak{I}_\sigma(F) = w$,
sonst ist sie **unerfüllbar** (**widerspruchsvoll**).

z. B. $p \wedge q$ ist erfüllbar; $p \wedge \neg p$ ist unerfüllbar.

Eine Formel F heißt **allgemeingültig** oder **Tautologie**, wenn
für alle ihre Interpretationen $\mathfrak{I}_\sigma(F) = w$ gilt.

z. B. $p \vee \neg p$.

Eine Formel F ist genau dann allgemeingültig, wenn $\neg F$ unerfüllbar ist.



Gesetze der booleschen Algebra

Zwei Formeln F, G sind **logisch äquivalent**, $\mathbf{F \equiv G}$,
wenn sie für alle Interpretationen $\mathfrak{I}$ dasselbe Ergebnis haben: $\mathfrak{I}(F) = \mathfrak{I}(G)$

Für alle aussagenlogischen Fomeln X, Y, Z gelten folgende **logische Äquivalenzen**:

$(X \wedge Y) \wedge Z \equiv X \wedge (Y \wedge Z)$	$(X \vee Y) \vee Z \equiv X \vee (Y \vee Z)$	Assoziativität
$X \wedge Y \equiv Y \wedge X$	$X \vee Y \equiv Y \vee X$	Kommutativität
$X \wedge X \equiv X$	$X \vee X \equiv X$	Idempotenz
$X \vee (Y \wedge Z) \equiv (X \vee Y) \wedge (X \vee Z)$	$X \wedge (Y \vee Z) \equiv (X \wedge Y) \vee (X \wedge Z)$	Distributivität
$X \vee (X \wedge Y) \equiv X$	$X \wedge (X \vee Y) \equiv X$	Absorption
$X \wedge O \equiv O$ $X \wedge L \equiv X$	$X \vee O \equiv X$ $X \vee L \equiv L$	Neutrale Elemente
$X \wedge \neg X \equiv O$	$X \vee \neg X \equiv L$	Komplement
$\neg \neg X \equiv X$		Involution
$\neg (X \wedge Y) \equiv \neg X \vee \neg Y$	$\neg (X \vee Y) \equiv \neg X \wedge \neg Y$	De Morgan

Umformen mit Gesetzen der booleschen Algebra

Beispiel:

$(A \vee \underline{\neg (B \wedge A)}) \wedge (C \vee (D \vee C)) \equiv$	De Morgan
$(A \vee \underline{(\neg B \vee \neg A)}) \wedge (C \vee (D \vee C)) \equiv$	Kommutativität
$(\underline{A \vee (\neg A \vee \neg B})) \wedge (C \vee (D \vee C)) \equiv$	Assoziativität
$((\underline{A \vee \neg A}) \vee \neg B) \wedge (C \vee (D \vee C)) \equiv$	Komplement
$(\underline{L \vee \neg B}) \wedge (C \vee (D \vee C)) \equiv$	Kommutativität
$(\underline{\neg B \vee L}) \wedge (C \vee (D \vee C)) \equiv$	Neutrale Elemente
$\underline{L \wedge (C \vee (D \vee C))} \equiv$	Kommutativität
$\underline{(C \vee (D \vee C)) \wedge L} \equiv$	Neutrale Elemente
$(C \vee (\underline{D \vee C})) \equiv$	Kommutativität
$(\underline{C \vee (C \vee D})) \equiv$	Assoziativität
$((\underline{C \vee C}) \vee D) \equiv$	Idempotenz
$C \vee D$	

Logischer Schluss

Sei $\mathbb{A}$ eine Menge von Formeln und F eine Formel.
Wenn für **alle Interpretationen** $\mathfrak{I}$, die alle Formeln in $\mathbb{A}$ erfüllen, auch $\mathfrak{I}$ (F) gilt, dann sagen wir

„F folgt semantisch aus $\mathbb{A}$“ $\quad$ **$\mathbb{A}$ |= F**

$\mathbb{A}$ |= F heißt auch **logischer Schluss**,
$\mathbb{A}$ **Annahme** oder Antezedent, F **Folgerung** oder Konsequenz.

Die **Korrektheit eines logischen Schlusses** $\mathbb{A}$ |= F mit $\mathbb{A} = \{A_1, ..., A_n\}$ kann man prüfen:

a. durch Prüfen aller Interpretationen, die alle Formeln in A erfüllen

b. durch Wiederspruchsbeweis: $\quad \mathbf{A_1 \wedge ... \wedge A_n \wedge \neg F}$ muss **unerfüllbar** sein.

Beweise werden aus logischen Schlüssen aufgebaut.

Beispiel:

A: Wenn alle Menschen gleich sind, gibt es keine Privilegien.

B: Es gibt Privilegien.

C: Nicht alle Menschen sind gleich.

nachweisen: { A, B } **|=** C ist ein **korrekter logischer Schluss**.

3.2 Verifikation von Aussagen über Algorithmen

Hoaresche Logik: Kalkül zum Beweisen von **Aussagen über Algorithmen und Programme**, Programm-**Verifikation**, [C.A.R. Hoare, 1969].

Statische Aussagen über Zustände des Algorithmus, über Werte von Variablen nachweisen, ohne den Algorithmus auszuführen. Aussagen **gelten für alle Ausführungen** des Algorithmus. Im Gegensatz zum **dynamischen Testen**: Ausführen des Algorithmus für bestimmte Eingaben.

Durch logische Schlüsse über die Elemente von Algorithmen wird gezeigt, dass

- ein Algorithmus **aus jeder zulässigen Eingabe die geforderte Ausgabe** berechnet, z. B.

{ gegeben }	Algorithmus	{ gesucht }
{ a, b $\in \mathbb{N}$ }	Euklidischer Algorithmus	{ x ist größter gemeinsamer Teiler von a, b }

- eine Aussage an jeder Stelle des Algorithmus gilt, **Invariante**, z. B. Kellerinvariante in Mod-2.34
- an einer bestimmten **Stelle im Algorithmus eine bestimmte Aussage gilt** z. B. { $\neg$ empty (keller) } keller = pop (keller);
- **Schleifen terminieren.**

Ein **Algorithmus und die Aussagen dazu sollen zusammen konstruiert** werden.

Verifikation: Algorithmus berechnet ggT

Vorbedingung: $x, y \in \mathbb{N}$, d. h. $x > 0$, $y > 0$; sei G größter gemeinsame Teiler von x und y
Nachbedingung: $a = G$

Algorithmus mit { Aussagen über Variable }:

{ G ist ggT von x und y $\wedge$ x>0 $\wedge$ y>0 }
a := x; b := y;
{ INV: G ist ggT von a und b $\wedge$ a>0 $\wedge$ b>0 }
solange a ≠ b wiederhole
{ INV $\wedge$ a $\neq$ b }
falls a > b :
{ G ist ggT von a und b $\wedge$ a>0 $\wedge$ b>0 $\wedge$ a>b } $\rightarrow$
{ G ist ggT von a-b und b $\wedge$ a-b>0 $\wedge$ b>0 }
a := a - b
{ INV }
sonst
{ G ist ggT von a und b $\wedge$ a>0 $\wedge$ b>0 $\wedge$ b>a } $\rightarrow$
{ G ist ggT von a und b-a $\wedge$ a>0 $\wedge$ b-a>0 }
b := b - a
{ INV }
{ INV }
{ INV $\wedge$ a = b } $\rightarrow$
{ a = G }

Terminierung der Schleife:

- a+b fällt monoton
- a+b > 0 ist Invariante

Notation von Algorithmenelementen

Anweisungsform	**Notation**	**Beispiel**
Sequenz	Anweisung$_1$**;** Anweisung$_2$	`a := x;` `b := y`
Zuweisung	Variable **:=** Ausdruck	`a := x`

Alternative, zweiseitig	**falls** Bedingung **:**
	Anweisung$_1$
	sonst Anweisung$_2$
		`falls a > b :`
		`a := a - b`
		`sonst b := b - a`

bedingte Anweisung	**falls** Bedingung **:** Anweisung$_1$	`falls a < 0 :` `a := - a`
Aufruf eines Unteralgorithmus ua	ua	`berechneGgT`

Schleife	**solange** Bedingung **wiederhole**
	Anweisung
		`solange a≠b wiederhole`
		`falls a > b:`
		`...`

Notation von Aussagen zur Verifikation

Aussagen über den Algorithmenzustand, über Werte von Variablen werden in den Algorithmus eingefügt:

$$\{ P \} \quad A_1 \quad \{ Q \} \quad A_2 \quad \{ R \}$$

bedeutet: Q gilt immer zwischen der Ausführung von A_1 und der Ausführung von A_2

Beispiel: $\{ i + 1 \geq 0\}$ **`i := i + 1;`** $\{ i \geq 0 \}$ **`a [i] := k;`** {...}

Zur Verifikation eines Algorithmus muss für jede Anweisung S ein Nachweis geführt werden:

{ Vorbedingung P }   S   { Nachbedingung Q }

nachweisen: Wenn vor der Ausführung des Schrittes S die P gilt, dann gilt Q nach der Ausführung von S, falls S terminiert.

Beispiel: $\{ i + 1 \geq 0\}$ **`i := i + 1;`** $\{i \geq 0\}$ mit Zuweisungsregel nachweisen

Die Aussagen werden entsprechend der **Struktur von S verknüpft**.
Für jede Anweisungsform wird eine spezielle **Schlussregel** angewandt.

Eine **Spezifikation liefert Vorbedingung und Nachbedingung** des gesamten Algorithmus:

gegeben:	gesucht:
Aussagen über die Eingabe	Aussagen über Zusammenhang zwischen Ein- und Ausgabe

{ Vorbedingung }   Algorithmus   { Nachbedingung }

Schlussregeln für Sequenz

Hoarescher Kalkül definiert **für jede Anweisungsform** ein **Regelschema.**

Sequenzregel:

$$\frac{\{P\}\ S_1\ \{Q\} \qquad \{Q\}\ S_2\ \{R\}}{\{P\}\ S_1; S_2\ \{R\}}$$

Bedeutung:

Wenn $\{P\}\ S_1\ \{Q\}$ und $\{Q\}\ S_2\ \{R\}$ korrekte Schlüsse sind, dann ist auch $\{P\}\ S_1; S_2\ \{R\}$ ein korrekter Schluss

Beispiel:

$\{ x>0 \wedge y>0 \}$ **`a := x;`** $\{ a>0 \wedge y>0 \}$

$\{ a>0 \wedge y>0 \}$ **`b := y;`** $\{ a>0 \wedge b>0 \}$

$\{ x>0 \wedge y>0 \}$ **`a := x; b := y;`** $\{ a>0 \wedge b>0 \}$

im Algorithmus die Schritte

$\{ x>0 \wedge y>0 \}$
`a := x;`
$\{ a>0 \wedge y>0 \}$

und

$\{ a>0 \wedge y>0 \}$
`b := y;`
$\{ a>0 \wedge b>0 \}$

zusammensetzen:

$\{ x>0 \wedge y>0 \}$
`a := x;`
$\{ a>0 \wedge y>0 \}$
`b := y;`
$\{ a>0 \wedge b>0 \}$

Konsequenzregeln

Abschwächung der Nachbedingung

$$\frac{\{\,P\,\}\quad S\quad \{\,R\,\} \qquad \{\,R\,\}\ \rightarrow\ \{\,Q\,\}}{\{\,P\,\}\quad S\quad \{\,Q\,\}}$$

Verschärfung der Vorbedingung

$$\frac{\{\,P\,\}\ \rightarrow\ \{\,R\,\} \qquad \{\,R\,\}\quad S\quad \{\,Q\,\}}{\{\,P\,\}\quad S\quad \{\,Q\,\}}$$

Beispiel:

$$\frac{\{\,a+b > 0\,\}\quad x := a+b\quad \{\,x > 0\,\} \qquad \{\,x > 0\,\}\ \rightarrow\ \{\,x \geq 0\,\}}{\{\,a+b > 0\,\}\quad x := a+b\quad \{\,x \geq 0\,\}}$$

im Algorithmus können Implikationen in Ausführungsrichtung eingefügt werden:

$\{\,a+b > 0\,\}$
$x := a+b$
$\{\,x > 0\,\} \rightarrow \{\,2*x \geq 0\,\}$
$y := 2*x$
$\{\,y \geq 0\,\}$

Zuweisungsregel

Die Zuweisung x := e wertet den Ausdruck e aus und weist das Ergebnis der Variablen x zu.

$$\{ P^{x}_{e} \}\ x := e\ \{P\}$$

P^{x}_{e} steht für die **Substitution von x durch e in P**, d. h. P [x/e].

Beispiele:

$\{a > 0\}$	$x := a$	$\{x > 0\}$
$\{i + 1 > 0\}$	$i := i + 1$	$\{i > 0\}$

Wenn man zeigen will, dass **nach der Zuweisung eine Aussage P für x** gilt,
muss man zeigen, dass **vor der Zuweisung dieselbe Aussage P für e** gilt.

Beispiele im Algorithmus:

$\{ x>0 \wedge y>0 \}$
```
a := x;
```
$\{ a>0 \wedge y>0 \}$
```
b := y;
```
$\{ a>0 \wedge b>0 \}$

{ G ist ggT von a-b und b $\wedge$ a-b>0 $\wedge$ b>0 }
```
a := a - b
```
{ G ist ggT von a und b $\wedge$ a>0 $\wedge$ b>0 }

Beispiele für Zuweisungsregel

$\{ P^{x}_{e} \}$ in P x durch e ersetzt	$x := e$	$\{ P \}$	
1. $\{ a > 0 \}$	$x := a$	$\{ x > 0 \}$	
2. $\{ a > 0 \wedge a > 0 \}$	$x := a$	$\{ x > 0 \wedge a > 0 \}$	x durch a ersetzen - nicht umgekehrt
3. $\{ a > 0 \wedge x = 7 \}$	$x := a$	$\{ x > 0 \wedge x = 7 \}$	**falscher Schluss**! **alle** x durch a ersetzen!
4. $\{ a > 0 \wedge z > 0 \}$	$x := a$	$\{ x > 0 \wedge z > 0 \}$	$z > 0$ ist nicht betroffen
5. $\{ i + 1 > 0 \}$	$i := i + 1$	$\{ i > 0 \}$	
6. $\{ i \geq 0\} \equiv \{ i + 1 > 0 \}$	$i := i + 1$	$\{ i > 0 \}$	passend umformen
7. $\{ i = 2 \} \equiv \{ i + 1 = 3 \}$	$i := i + 1$	$\{ i = 3 \}$	passend umformen
8. $\{ \text{wahr} \} \equiv \{ 1 = 1 \}$	$x := 1$	$\{ x = 1 \}$	passend umformen
9. $\{ z = 5 \} \equiv$ $\{ z = 5 \wedge 1 = 1 \}$	$x := 1$	$\{ z = 5 \wedge x = 1 \}$	passend umformen

Aufrufregel

Aufruf eines Unteralgorithmus UA
mit der Spezifikation: Vorbedingung P, Nachbedingung Q:

$$\{ P \}\ UA\ \{ Q \}$$

Diese Regel gilt nur für Unteralgorithmen **ohne Parameter** und **ohne Funktionsergebnis**.
Sie lesen oder schreiben globale Variable. Sie können in P und Q vorkommen.

Regel für 2-seitige Alternative

$$\frac{\{ P \wedge B \}\ S_1\ \{ Q \} \qquad \{ P \wedge \neg B \}\ S_2\ \{ Q \}}{\{ P \}\ \text{falls } B\text{: } S_1 \text{ sonst } S_2\ \{ Q \}}$$

Aus der **gemeinsamen Vorbedingung P** führen beide Zweige auf **dieselbe Nachbedingung Q**

Beispiel:

$$\frac{\{ a > 0 \}\ b := a\ \{ b > 0 \} \rightarrow \{ b \geq 0 \} \qquad \{ \neg (a > 0) \} \rightarrow \{ -a \geq 0 \}\ b := -a\ \{ b \geq 0 \}}{\{ \text{true} \}\ \text{falls } a > 0\text{: } b := a \text{ sonst } b := -a\ \{ b \geq 0 \}}$$

im Algorithmus:

$\{ a>0 \wedge b>0 \wedge a \neq b \}$
`falls a > b :`
 $\{ a>0 \wedge b>0 \wedge a>b \} \rightarrow$
 $\{ a-b>0 \wedge b>0 \}$
 `a := a - b`
 $\{ \mathbf{a>0 \wedge b>0} \}$
`sonst`
 $\{ a>0 \wedge b>0 \wedge b>a \} \rightarrow$
 $\{ a>0 \wedge b-a>0 \}$
 `b := b - a`
 $\{ \mathbf{a>0 \wedge b>0} \}$
$\{ \mathbf{a>0 \wedge b>0} \}$

Regel für bedingte Anweisung

$$\frac{\{ P \wedge B \} \quad S \{ Q \} \qquad P \wedge \neg B \quad \rightarrow Q}{\{ P \} \text{ falls } B : S \{ Q \}}$$

Aus der **gemeinsamen Vorbedingung P** führen die Anweisung und die Implikation auf **dieselbe Nachbedingung Q**

Beispiel:

$$\frac{\{ P \wedge a < 0\} \ a := - a \quad \{ P \wedge a \geq 0 \} \qquad P \wedge \neg (a < 0) \ \rightarrow \ P \wedge a \geq 0}{\{ P \} \text{ falls } a < 0\text{: } a := -a \ \{ P \wedge a \geq 0 \}}$$

im Algorithmus:

$\{ P \}$
falls $a < 0$:
  $\{ P \wedge a < 0 \} \rightarrow \{ P \wedge -a > 0 \}$
  $a := - a;$
  $\{P \wedge a > 0 \} \rightarrow \{ P \wedge a \geq 0 \}$
leere Alternative:
  $\{ P \wedge \neg (a < 0) \ \rightarrow \ P \wedge a \geq 0 \}$
$\{ P \wedge a \geq 0 \}$

Schleifenregel

Wiederholung, Schleife:

$$\frac{\{\, P \wedge B \,\}\ \ S\ \ \{\, P \,\}}{\{\, P \,\}\ \text{solange } B \text{ wiederhole } S\ \{\, P \wedge \neg B \,\}}$$

Eine Aussage P heißt **Schleifeninvariante**, wenn man zeigen kann, dass sie an folgenden Stellen gilt: **vor der Schleife**,
vor und nach jeder Ausführung von S und
nach der Schleife.

Beispiel: Algorithmus zum Potenzieren

INV: $z \cdot a^b = x^y \wedge b \geq 0$

a := x; b := y; z := 1;
{ **INV** }
solange $b > 0$ wiederhole
  { **INV** $\wedge\ b > 0$ } $\equiv \{\ z \cdot a \cdot a^{b-1} = x^y \wedge (b-1) \geq 0\ \}$
  b := b - 1;
  $\{\ z \cdot a \cdot a^b = x^y \wedge b \geq 0\ \}$
  $z := z \cdot a$
  { **INV** }
{ **INV** $\wedge\ b \leq 0$ } $\rightarrow \{\ z \cdot a^b = x^y \wedge b = 0\ \} \rightarrow \{\ z = x^y\ \}$

Terminierung von Schleifen

Die **Terminierung einer Schleife muss separat nachgewiesen werden**.

solange B wiederhole S

1. Gib einen **ganzzahligen Ausdruck E** an über Variablen, die in der Schleife vorkommen, und zeige, dass E bei jeder Iteration durch S **verkleinert** wird.

2. Zeige, dass **E nach unten begrenzt** ist, z. B. dass $0 \leq E$ eine Invariante der Schleife ist.

Es kann auch eine andere Grenze als 0 gewählt werden.

E kann auch monoton **vergrößert werden und nach oben begrenzt** sein.

Nichtterminierung wird bewiesen, indem man zeigt,
dass $R \wedge B$ eine Invariante der Schleife ist und
dass es eine Eingabe gibt, so dass $R \wedge B$ vor der Schleife gilt.
R kann einen speziellen Zustand charakterisieren, in dem die Schleife nicht anhält.

Es gibt Schleifen, für die man **nicht entscheiden** kann,
ob sie für jede Vorbedingung **terminieren**.

Beispiele zur Terminierung

1. $\{ a > 0 \wedge b > 0\}$ — terminiert

```
             { a > 0 ∧ b > 0}
Schleife1    solange a ≠ b wiederhole
Schleife2            solange a > b wiederhole
                              a := a - b;
Schleife3            solange a < b wiederhole
                              b := b - a
```

2. $\{ a > 0 \wedge b > 0\}$ — terminiert nicht immer

```
             { a > 0 ∧ b > 0}
Schleife1    solange a ≠ b wiederhole
Schleife2            solange a ≥ b wiederhole
                              a := a - b;
Schleife3            solange a < b wiederhole
                              b := b - a
```

3. $\{ n \in \mathbb{N} \wedge n > 1 \}$ — Terminierung ist unbewiesen

```
             { n ∈ ℕ ∧ n > 1 }
             solange n > 1 wiederhole
                     falls n gerade:
                              n := n / 2
                     sonst    n := 3 * n + 1
```

Denksportaufgabe zu Invarianten

In einem Topf seien s schwarze und w weiße Kugeln, $s + w > 0$

solange mindestens 2 Kugeln im Topf sind

nimm 2 beliebige Kugeln heraus

falls sie gleiche Farbe haben:

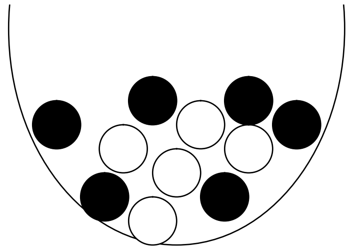
wirf beide weg und

lege eine neue schwarze Kugel in den Topf

falls sie verschiedene Farben haben:

lege die weiße Kugel zurück in den Topf und

wirf die schwarze Kugel weg



Welche Farbe hat die letzte Kugel?

Finden Sie Invarianten, die die Frage beantworten.

Schrittweise Konstruktion und Verifikation

Vorbedingung: $x \in \mathbb{R}$ und $n \in \mathbb{N}_0$

Nachbedingung: $q = x^n$

Algorithmus:

$\{ n \geq 0 \} \rightarrow \{ n = n \wedge n \geq 0 \wedge x = x \wedge 1 = 1 \}$

```
a := x; q := 1; i := n;
```

$\{ i = n \wedge i \geq 0 \wedge a = x \wedge q = 1 \} \rightarrow \{ INV \}$

```
solange i > 0 wiederhole
```

$\{ INV \wedge i > 0 \}$

falls i ungerade: $\{ INV \wedge i > 0 \wedge i \text{ ungerade} \} \rightarrow$

$\{ x^n = q * a * (a^2)^{i/2} \wedge i > 0 \}$ **q := q * a;** $\{ x^n = q * (a^2)^{i/2} \wedge i > 0 \}$

leere Alternative für i gerade:

$\{ INV \wedge i > 0 \wedge i \text{ gerade} \} \rightarrow \{ x^n = q * (a^2)^{i/2} \wedge i > 0 \}$

$\{ x^n = q * (a^2)^{i/2} \wedge i > 0 \}$

```
a := a * a;
```

$\{ x^n = q * a^{i/2} \wedge i > 0 \} \rightarrow \{ x^n = q * a^{i/2} \wedge i/2 \geq 0 \}$

```
i := i / 2
```

$\{ x^n = q * a^i \wedge i \geq 0 \} \equiv \{ INV \}$

$\{ INV \wedge i \leq 0 \} \rightarrow \{ q = x^n \}$

Terminierung der Schleife: i fällt monoton und $i \geq 0$ ist invariant.

Konstruktionsidee:

Invariante INV: $x^n = q * a^i \wedge i \geq 0$

Zielbedingung: $i \leq 0$

falls i gerade: $x^n = q * (a^2)^{i/2}$

falls i ungerade: $x^n = q * a * (a^2)^{i/2}$

Schritte:

1. Vor-, Nachbedingung
2. Schleifeninvariante
3. Schleife mit INV
4. Initialisierung
5. Idee für Schleifenrumpf
6. Alternative
7. Schleife komplett
8. Terminierung

3.3 Prädikatenlogik

Prädikatenlogik umfasst Aussagenlogik mit **atomaren Aussagen**, **Variablen**, **Junktoren.**
Zusätzliche Konzepte:

- **n-stellige Prädikate**
  z. B. „x ist eine Katze" bzw. als Formel: istKatze(x)
  teilt(a,b), größterGemeinsamerTeiler(a, b, g)
  Die Parameter werden aus einem Wertebereich gewählt.
  Prädikate haben einen Wahrheitswert.

- **Terme als Parameter von Prädikaten**:
  z. B. gleichwertig (**x + y**, **y + x**)

- **Quantoren „für alle x gilt P(x)**" und „**es gibt ein x, so dass P(x) gilt**"
  in Symbolen: $\forall$**xP(x)** bzw. $\exists$**xP(x)**
  Beispiel: $\forall$x (esIstNacht $\wedge$ istKatze(x) $\rightarrow$ istGrau(x));
  in Worten: „Nachts sind alle Katzen grau."

Schon **zur Modellierung** einfacher Aufgaben braucht man Konzepte **der Prädikatenlogik**:

z. B. größter gemeinsamer Teiler:

gegeben: $a \in \mathbb{N}$, $b \in \mathbb{N}$

gesucht: größter gemeinsamer Teiler g von a und b, d. h.
teilt(g, a) $\wedge$ teilt(g, b) $\wedge$ ($\forall$h (teilt(h, a) $\wedge$ teilt(h, b) $\rightarrow$ h $\leq$ g))

Vorschau auf Begriffe

Ähnliche Folge von Begriffen wie in der Aussagenlogik:

- **prädikatenlogische Formeln** als Sprache der Prädikatenlogik
  Syntax: Terme, Prädikate, logische Junktoren, Quantoren
- **gebundene** und **freie Variable**
- **Individuenbereich**: allgemeiner Wertebereich für Variable und Terme
- **Belegung** von Variablen mit Werten aus dem Individuenbereich
- **Interpretation**: Variablenbelegung und Definition der Funktionen und Prädikate
- **erfüllbar, allgemeingültig, unerfüllbar**: wie in Aussagenlogik definiert
- **logischer Schluss**: wie in Aussagenlogik definiert
- **Gesetze zum Umformen** von Formeln mit Quantoren

Prädikatenlogische Formeln

Prädikatenlogische Formeln werden induktiv wie folgt definiert:

1. **atomare Formeln** sind Anwendungen von Prädikaten in der Form **$P (t_1, ..., t_n)$**
   Dabei ist P ein n-stelliges Prädikatsymbol und
   die t_i sind Terme mit Variablen zu einer Signatur Σ
   (0-stellige Prädikatsymbole entsprechen den atomaren Aussagen der Aussagenlogik.)

2. **logische Junktoren** bilden prädikatenlogische Formeln:
   $\neg F \qquad F \wedge G \qquad F \vee G \quad F \rightarrow G \quad F \leftrightarrow G$
   aus prädikatenlogische Formeln F und G

3. der **Allquantor** $\forall$ und der **Existenzquantor** $\exists$ bilden prädikatenlogische Formeln:
   $\forall x\ F$ und $\exists x\ F$
   aus der prädikatenlogischen Formel F und definieren die Variable x

Nur nach (1. - 3.) gebildete Formeln sind **syntaktisch korrekte prädikatenlogische Formeln**.

Beispiel:

$$\text{teilt}(g, a) \wedge \text{teilt}(g, b) \wedge (\forall h\ (\text{teilt}(h, a) \wedge \text{teilt}(h, b) \rightarrow h \leq g))$$

Anmerkungen zu prädikatenlogischen Formeln

- **Prädikatsymbole und Operationssymbole** in Termen erhalten ihre Bedeutung erst durch die **Interpretation** der Formel (wie bei abstrakten Algebren), aber
- **Prädikate und Operationen werden häufig nicht explizit definiert**, sondern mit üblicher Bedeutung der Symbole angenommen.
- **Signatur Σ wird meist nicht explizit angegeben**, sondern aus den Operationen angenommen, die in den Termen verwendet werden.
- Hier: **Prädikatenlogik erster Stufe**: **Variable** sind nur als Operanden in Termen erlaubt, aber **nicht für Funktionen oder für Prädikate**.

Gebundene Variable

Ein Quantor $\forall x$ F oder $\exists x$ F **bindet alle Vorkommen des Variablennamens x** in der Formel F, außer den Vorkommen von x die durch einen weiteren Quantor innerhalb von F gebunden sind.

x ist der **Name der Variablen des Quantors**.

F ist der der **Wirkungsbereich des Quantors**.

Beispiel:

$$\underbrace{\forall x\ (P(x) \wedge Q(x))} \vee \underbrace{\exists y\ (P(y) \wedge \underbrace{\forall z\ R(y, z)})}$$

Quantoren mit ihrenWirkungsbereichen

Anmerkungen:

- Eine Variable hat einen Namen, z. B. x.
- Mehrere verschiedene Variable können denselben Namen haben.
- Ein Quantor führt eine Variable ein, z. B. $\forall x$ F.
  Ihr Name kann im Wirkungsbereich (auch mehrfach) vorkommen.
- Wirkungsbereiche von Quantoren können geschachtelt sein,
  sogar mit (verschiedenen) Variablen, die dieselben Namen haben.

Freie Variable

Ein Vorkommen eines Variablennamens y heißt **frei in der Formel F**, wenn es **nicht im Wirkungsbereich eines Quantors für y** liegt. y ist dann der Name einer **freien Variablen** der Formel F.

Beispiel: Formel F

$$\underset{\uparrow}{x} < 0 \wedge \exists y\, (P(y, \underset{\uparrow}{x}) \vee Q(y, \underset{\uparrow}{z}))$$

freie Vorkommen

F hat 2 freie Variable. Sie haben die Namen x und z.

Freie Variable werden **außerhalb der Formel gebunden** und mit Werten belegt.

Umbenennung von Variablen

In einer Formel können mehrere Vorkommen von Quantoren **verschiedene Variable mit gleichem Namen** einführen und in ihrem Wirkungsbereich binden:

Beispiele:

$$\forall y\ (\exists x\ R(x, y) \wedge \exists x\ Q(x, y)) \qquad \forall x\ \forall y\ (P(x, y) \wedge \exists x\ R(x, y))$$

Umbenennung: In einer Formel kann man **alle Vorkommen des Namens x einer Variablen durch einen neuen Namen z ersetzen**, der sonst nicht in der Formel vorkommt. Die Bedeutung der Formel ändert sich dadurch nicht.

Beispiele von oben:

$$\forall y\ (\exists x\ R(x, y) \wedge \exists z\ Q(z, y)) \qquad \forall x\ \forall y\ (P(x, y) \wedge \exists z\ R(z, y))$$

Damit kann man erreichen, dass **verschiedene Variable verschiedene Namen** haben.

Wir sagen dann: Die Variablen der Formel sind **konsistent umbenannt**.

Formeln, in denen alle Variablen verschiedene Namen haben sind meist besser lesbar.

Manche Definitionen sind einfacher für konsistent umbenannte Formeln.

Interpretation zu prädikatenlogischer Formel

Einer prädikatenlogischen Formel F wird durch eine **Interpretation** $\mathfrak{I}$ (F) **Bedeutung zugeordnet,** sodass man ihren Wahrheitswert (w oder f) berechnen kann.

Eine **Interpretation** $\mathfrak{I}$ wird bestimmt durch

- einen **Individuenbereich U,** der nicht leer ist (auch Universum genannt). Aus U stammen die Werte der Variablen und Terme.
- eine **Abbildung der Funktions- und Prädikatsymbole** einer Signatur Σ auf dazu passende konkrete Funktionen und Prädikate, notiert z. B. $\mathfrak{I}$(f), $\mathfrak{I}$(P)
- eine **Belegung der freien Variablen mit Werten aus U**, notiert z. B. $\mathfrak{I}$(x)

Bemerkungen:

- In der Prädikatenlogik enthält der **Individuenbereich U alle Individuen** - **auch verschiedenartige** - die für die Interpretation benötigt werden. Er ist **nicht in Wertebereiche gleichartiger Individuen** strukturiert (wie in Kapitel 2).
- **Jede Sorte aus** Σ wird deshalb **auf den ganzen Individuenbereich U** abgebildet.
- Eine **Interpretation** wird immer **passend zu einer Menge prädikatenlogischer Formeln** definiert. Nur darin vorkommende Funktionen, Prädikate und Variable interessieren.

Beispiel für eine passende Interpretation zu einer Formel

Zur Formel F = $\forall x\, P(x, f(x)) \wedge Q(g(a, z))$ ist folgendes $\mathfrak{I}$ eine passende Interpretation:

U := $\mathbb{N}_0$

$\mathfrak{I}(P)$:= { (m, n) | m, n $\in$ U und m < n }

$\mathfrak{I}(Q)$:= { n | n $\in$ U und n ist Primzahl }

$\mathfrak{I}(f)$ ist die Nachfolgerfunktion auf U, also $\mathfrak{I}(f)(n) = n + 1$

$\mathfrak{I}(g)$ ist die Additionsfunktion auf U also $\mathfrak{I}(g)(m, n) = m + n$

$\mathfrak{I}(a)$:= 2 (a ist eine Konstante)

$\mathfrak{I}(z)$:= 3 (z ist eine freie Variable)

Bemerkungen:

- Häufig wird die Interpretation von Funktions- und Prädikatssymbolen nicht explizit angegeben, sondern die „übliche Bedeutung der Symbole“ angenommen.
- Die Anwendung von $\mathfrak{I}$ zeigt, wie die Variablen der Quantoren Werte erhalten.

Das Beispiel stammt aus

U. Schöning: Logik für Informatiker, Spektrum Akademischer Verlag, 4. Aufl., 1995, S. 55

Wahrheitswerte prädikatenlogischer Formeln

Sei F eine prädikatenlogische Formel und $\Im$ eine dazu passende Interpretation, dann berechnet man den **Wahrheitswert $\Im$(F),** indem man $\Im$ **rekursiv anwendet** auf die Teile von F:

- die **Prädikatsymbole und deren Parameterterme**,
- die **Funktionssymbole und deren Operandenterme**,
- die **freien und gebundenen Variablen**,
- die **mit Junktoren verknüpften Teilformeln** und
- die **Quantor-Formeln**.

Die **Berechnung von Quantor-Formeln** ist definiert durch:

$\Im(\forall x\ G) = w$ genau dann, wenn **für jeden Wert d $\in$ U** gilt $\Im_{[x/d]}(G) = w$

$\Im(\exists x\ G) = w$ genau dann, wenn es **einen Wert d $\in$ U** gibt mit $\Im_{[x/d]}(G) = w$

Dabei ordnet $\Im_{[x/d]}$ (G) in G der Variablen x den Wert d zu und stimmt sonst mit der gerade angewandten Interpretetion $\Im$ überein.
Jeder Wert aus U wird also an die Variable des Quantors gebunden, damit die Quantor-Formel ausgerechnet und die Ergebnisse mit $\wedge$ bei $\forall$ und mit $\vee$ bei $\exists$ verknüpft.

Wir nehmen an, dass alle Variablen in F verschiedene Namen haben (konsistent umbenannt). Dann betrifft $\Im_{[x/d]}$ nur die Variable des gerade betrachteten Quantors und keine andere.

Beispiel für Interpretation einer Formel

Formel F:

$z \wedge \forall x\, \forall y\; P(x, y)$

Interpretation $\mathfrak{I}$:

$U = \{ 1, 2, 3 \}$

$\mathfrak{I}(P) = \{ (a, b) \mid a + b < 10 \}$

$\mathfrak{I}(z) = w$

Interpretation $\mathfrak{I}$ rekursiv angewandt:

$\mathfrak{I}(z \wedge \forall x\, \forall y\; P(x, y)) =$

$\mathfrak{I}(\mathfrak{I}(z) \wedge \mathfrak{I}(\forall x\, \forall y\; P(x, y))) =$

$\mathfrak{I}(z)$ und $\mathfrak{I}(\forall x\, \forall y\; \mathfrak{I}_{\mathbf{[x/d,\, y/e]}}(P)\; (\mathfrak{I}_{\mathbf{[x/d,\, y/e]}}(x),\, \mathfrak{I}_{\mathbf{[x/d,\, y/e]}}(y)))$

w und w (siehe Quantor-Formel)

w

Berechnung der Quantor-Formel: (aufzählen nur bei kleinem U praktikabel)

$\mathfrak{I}(\forall$ x	$\forall$ y	P (x, y))
1	1	w
1	2	w
1	3	w
2	1	w
2	2	w
2	3	w
3	1	w
3	2	w
3	3	w

also ist $\mathfrak{I}(\forall x\, \forall y\; P(x, y)) = w$

Beschränkung von Wertebereichen

In der **Modellierung** soll man die **Wertebereich**e der Variablen präzise angeben.
In der **Prädikatenlogik** können jedoch die **Variablen der Quantoren Werte** aus dem **gesamten Individuenbereich U** annehmen.
Deshalb müssen **Einschränkung explizit als Prädikat** formuliert werden.

Beschränkung des Wertebereiches bei Allquantoren durch Implikation $\rightarrow$:

„für alle $m \in M \subseteq U$ gilt $P(m)$“:

	ausführliche Notation:	abkürzende Notation:
allgemein:	$\forall m\ (m \in M \rightarrow P(m))$	$\forall m \in M: P(m)$
Beispiele:	$\forall i\ (i \in \{1, 2, 3, 4\} \rightarrow b_i = a_i^2)$	$\forall i \in \{1, 2, 3, 4\}: b_i = a_i^2$
	$\forall k\ (k \in \mathbb{N} \rightarrow a + k \geq a)$	$\forall k \in \mathbb{N}: a + k \geq a$

Beschränkung des Wertebereiches bei Existenzquantoren durch Konjunktion $\wedge$:

„es gibt ein $m \in M \subseteq U$, so dass $P(m)$ gilt“:

allgemein:	$\exists m\ (m \in M \wedge P(m))$	$\exists m \in M: P(m)$
Beispiele:	$\exists k\ (k \in \mathbb{N} \wedge a * k = b)$	$\exists k \in \mathbb{N}: a * k = b$
	$\exists i\ (i \in \{1, 2, 3, 4\} \wedge a_i = x)$	$\exists i \in \{1, 2, 3, 4\}: a_i = x$

Beispiel für prädikatenlogische Formeln (1)

Relationen $R \subseteq M \times M$ **ist reflexiv**, wenn für alle $x \in M$ gilt $(x, x) \in R$ (siehe Mod-2.10)

als Prädikat, abgekürzt: reflexiv(R): $R \subseteq M \times M \wedge \forall x \in M: (x, x) \in R$

als PL Formel, ausführlich: $R \subseteq M \times M \wedge \forall x\, (x \in M \rightarrow (x, x) \in R)$

Eine **ausführliche Interpretation**:

$\Im(\subseteq)$:= 2-stelliges Prädikat: „ist Teilmenge von“
$\Im(\in)$:= 2-stelliges Prädikat: „ist Element von“
$\Im((,))$:= 2-stellige Funktion: „bilde Paar aus“
U := { 1, 2, 3, 4 } *Individuenbereich*
$\Im(M)$:= { 1, 2, 3 } *Beispiel für M*
$\Im(R)$:= { (1, 1), (2, 2), (3, 3) } *Beispiel für R*

Wahrheitswert bestimmen: $R \subseteq M \times M \wedge \forall x\, (x \in M \rightarrow (x, x) \in R)$

$R \subseteq M \times M$	$\wedge$	x	$x \in M$	$\rightarrow$	$(x, x) \in R$
w		1	w	w	w
w		2	w	w	w
w		3	w	w	w
w		4	f	w	
w	$\wedge$			w	

Die Relation R ist reflexiv.

Beispiele für prädikatenlogische Formeln (2)

- Die Relation $R \subseteq M \times M$ **ist symmetrisch**, wenn für alle x, y $\in$ M gilt: aus x R y folgt y R x

  symmetrisch(R): $R \subseteq M \times M \wedge \forall x\ (x \in M \rightarrow \forall y\ (y \in M \rightarrow ((x, y) \in R \rightarrow (y, x) \in R)))$

  abgekürzt: $R \subseteq M \times M \wedge \forall x \in M{:}\ \forall y \in M{:}\ (x, y) \in R \rightarrow (y, x) \in R$

- Die Variablen in Axiomen abstrakter Algebren sind durch Allquantoren gebunden (Mod-2.27):

  K3: $pop\ (push\ (k, t)) \equiv k$

  gebunden: K3: $\forall k\ (k \in Keller \rightarrow \forall t\ (t \in Element \rightarrow pop\ (push\ (k, t)) \equiv k))$

  abgekürzt: K3: $\forall k \in Keller{:}\ \forall t \in Element{:}\ pop\ (push\ (k, t)) \equiv k$

- Prädikat „kleinstes Element einer Menge“:

  istKleinstes (k, M): $k \in M \wedge \forall i\ (i \in M \rightarrow k \leq i)$

Beispiele für prädikatenlogische Formeln (3)

- aus der Analysis:
  Eine Folge f(n) heißt konvergent mit dem Grenzwert a, wenn gilt

  $$\forall\, \varepsilon \in \mathbb{R}: \exists\, n_\varepsilon \in \mathbb{N}: \forall\, n \in \mathbb{N}: n > n_\varepsilon \rightarrow |\, f(n) - a\, | < \varepsilon$$

  dreifache Schachtelung der Quantoren; Reihenfolge ist wichtig!

- aus der Informatik:
  Eine Folge $a = (a_1, ..., a_n) \in \mathbb{N}^+$ heißt (nicht fallend) geordnet, wenn gilt

  $$\forall\, i \in \{1, ..., n\}: \forall\, j \in \{1, ..., n\}: i \leq j \rightarrow a_{\mathbf{i}} \leq a_{\mathbf{j}}$$

- Was bedeutet folgende Aufgabenstellung?

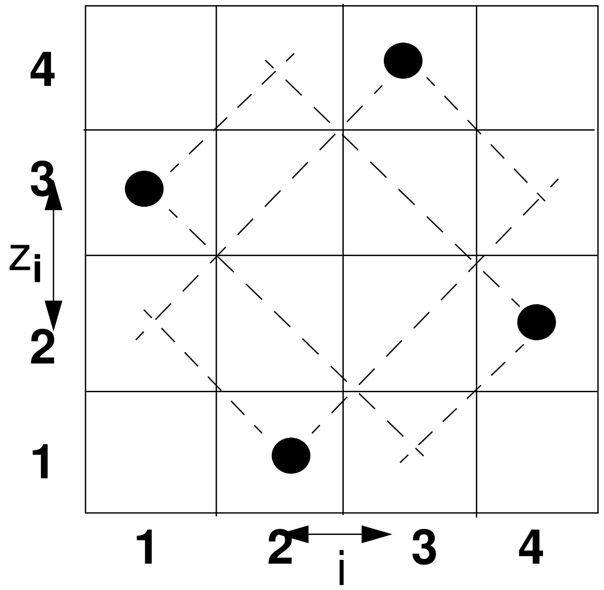
  gegeben: Eine Folge $a = (a_1, ..., a_n) \in \mathbb{N}^+$

  gesucht: $p \in Pos = \{1, ..., n\}$, so dass $\forall\, j \in Pos: a_{\mathbf{p}} \leq a_{\mathbf{j}} \wedge (a_{\mathbf{p}} = a_{\mathbf{j}} \rightarrow p \leq j)$

Beispiel: Spezifikation des n-Damen-Problems

gegeben:
Kantenlänge $n \in \mathbb{N}$ eines $n * n$ Schachbrettes

gesucht:
Menge P zulässiger Platzierungen von jeweils n Damen auf dem Schachbrett, so dass keine Dame eine andere nach Schachregeln schlägt:



Sei Index := {1, ..., n}

$P := \{ p \mid p = (z_1, ..., z_n) \in Index^n \wedge zulässig (p) \}$

z_i gibt die Zeilennummer der Dame in Spalte i an.

Dabei bedeutet
zulässig (p): $\forall\ i \in Index: \forall\ j \in Index: i \neq j \rightarrow z_i \neq z_j \wedge | z_i - z_j | \neq | i - j |$

Erfüllbarkeit und logischer Schluss

Die folgenden Begriffe sind in der Prädikatenlogik so **definiert wie in der Aussagenlogik**.
Aber: Interpretationen der Prädikatenlogik sind komplexe Strukturen.
Deshalb sind die Eigenschaften „**erfüllbar**" und „**allgemeingültig**" für prädikatenlogische Formeln **nicht allgemein entscheidbar**.

- Wenn für eine Interpretation $\mathfrak{I}$(F) = w gilt, heißt $\mathfrak{I}$ auch ein **Modell der Formel F.**
- Eine Formel F heißt **erfüllbar**, wenn es eine Interpretation $\mathfrak{I}$ gibt, so dass gilt $\mathfrak{I}$(F) = w, sonst ist sie **unerfüllbar** (**widerspruchsvoll**).
- Eine Formel F heißt **allgemeingültig** oder **Tautologie**, wenn für alle Interpretationen von F gilt $\mathfrak{I}$ (F) = w.
- Eine Formel F ist genau dann **allgemeingültig, wenn $\neg$ F unerfüllbar** ist.
- Zwei Formeln F und G sind **logisch äquivalent**, wenn sie für alle Interpretationen $\mathfrak{I}$ dasselbe Ergebnis haben: $\mathfrak{I}$(F) = $\mathfrak{I}$(G)
- Sei A eine Menge von Formeln und F eine Formel. Wenn für **alle Interpretationen** $\mathfrak{I}$, die alle Formeln in A erfüllen, auch $\mathfrak{I}$ (F) gilt, dann sagen wir **„F folgt semantisch aus A"** bzw. **A |= F**;
  A |= F heißt auch **logischer Schluss**.

Äquivalente Umformung prädikatenlogischer Formeln

Seien F und G beliebige prädikatenlogische Formel. Dann gelten folgende **Äquivalenzen**:

1. **Negation:**

   $\neg\forall x\ F \equiv \exists x\ \neg F$ $\qquad$ $\neg\exists x\ F \equiv \forall x\ \neg F$

2. **Wirkungsbereich der Quantoren verändern:**
   Falls x in G nicht frei vorkommt, gilt

   $(\forall x\ F \wedge G) \equiv \forall x\ (F \wedge G)$ $\qquad$ $(\forall x\ F \vee G) \equiv \forall x\ (F \vee G)$

   $(\exists x\ F \wedge G) \equiv \exists x\ (F \wedge G)$ $\qquad$ $(\exists x\ F \vee G) \equiv \exists x\ (F \vee G)$

   $G \equiv \exists x\ G$ $\qquad$ $G \equiv \forall x\ G$

3. **Quantoren zusammenfassen:**

   $(\forall x\ F \wedge \forall x\ G) \equiv \forall x\ (F \wedge G)$ $\qquad$ $(\exists x\ F \vee \exists x\ G) \equiv \exists x\ (F \vee G)$

   Folgende Formelpaare sind im allgemeinen **nicht äquivalent**:
   $(\forall x\ F \vee \forall x\ G) \not\equiv \forall x\ (F \vee G)$ $\qquad$ $(\exists x\ F \wedge \exists x\ G) \not\equiv \exists x\ (F \wedge G)$

4. **Quantoren vertauschen:**

   $\forall x\ \forall y\ F \equiv \forall y\ \forall x\ F$ $\qquad$ $\exists x\ \exists y\ F \equiv \exists y\ \exists x\ F$

Beispiele für Äquivalenzen

1. **Negation**: formal

	Alle haben den Schuss gehört.	$\forall x$ gehört(x)
negiert:	Es gibt einen, der den Schuss nicht gehört hat.	$\exists x \neg$ gehört(x)
falsch negiert:	Alle haben den Schuss nicht gehört.	$\forall x \neg$ gehört(x)

$\neg\forall i \in \text{Ind}: a_i < 10 \equiv$

$\neg\forall i\ (i \in \text{Ind} \to a_i < 10) \equiv$

$\exists i\ \neg(\neg\ i \in \text{Ind} \vee a_i < 10) \equiv$

$\exists i\ (\ i \in \text{Ind} \wedge \neg\ a_i < 10) \equiv$

$\exists i \in \text{Ind}: a_i \geq 10$

$(\exists\ x\ P(x)) \to P(y) \equiv$

$\neg(\exists\ x\ P(x)) \vee P(y) \equiv$

$(\forall\ x\ \neg P(x)) \vee P(y) \equiv$

$\forall\ x\ (\neg P(x) \vee P(y)) \equiv$

$\forall\ x\ (P(x) \to P(y))$

3. **Zusammenfassung von Quantoren**:

$(\forall i \in \text{Ind}: a_i < 10) \wedge (\forall i \in \text{Ind}: 0 < a_i) \equiv\ \forall i \in \text{Ind}: (a_i < 10 \wedge 0 < a_i)$

nicht äquivalent:

$(\forall i \in \text{Ind}: a_i < 10) \vee (\forall i \in \text{Ind}: 0 < a_i) \not\equiv\ \forall i \in \text{Ind}: (a_i < 10 \vee 0 < a_i)$

Beispiel für Umformungen

Die folgende prädikatenlogische Formel wird so umgeformt, dass alle Quantoren vorne (außen) stehen:

$$
\begin{array}{lll}
 & (\underline{\neg(\exists x\ P(x, y) \vee \forall z\ Q(z))} \wedge \exists w\ P(f(a, w))) & \text{DeMorgan} \\
\equiv & ((\underline{\neg\exists x\ P(x, y)} \wedge \underline{\neg\forall z\ Q(z)}) \wedge \exists w\ P(f(a, w))) & \text{Negation von Quantorformeln (1.)} \\
\equiv & ((\forall x\ \neg P(x, y) \wedge \exists z\ \neg Q(z)) \underline{\wedge} \exists w\ P(f(a, w))) & \text{Kommutativität} \\
\equiv & (\underline{\exists w\ P(f(a, w))} \wedge (\underline{\forall x\ \neg P(x, y)} \wedge \exists z\ \neg Q(z))) & \text{Wirkungsbereiche ausweiten (2.)} \\
\equiv & \exists w\ (P(f(a, w)) \underline{\wedge} \forall x\ (\neg P(x, y) \underline{\wedge} \exists z\ \neg Q(z))) & \text{Kommutativität} \\
\equiv & \exists w\ (\forall x\ (\underline{\exists z\ \neg Q(z)} \wedge \neg P(x, y)) \wedge P(f(a, w))) & \text{Wirkungsbereiche ausweiten (2.)} \\
\equiv & \exists w\ (\underline{\forall x\ \exists z\ (\neg Q(z) \wedge \neg P(x, y))} \wedge P(f(a, w))) & \text{Wirkungsbereiche ausweiten (2.)} \\
\equiv & \underline{\exists \mathbf{w}\ \forall \mathbf{x}\ \exists \mathbf{z}}\ (\neg Q(z) \wedge \neg P(x, y) \wedge P(f(a, w))) &
\end{array}
$$

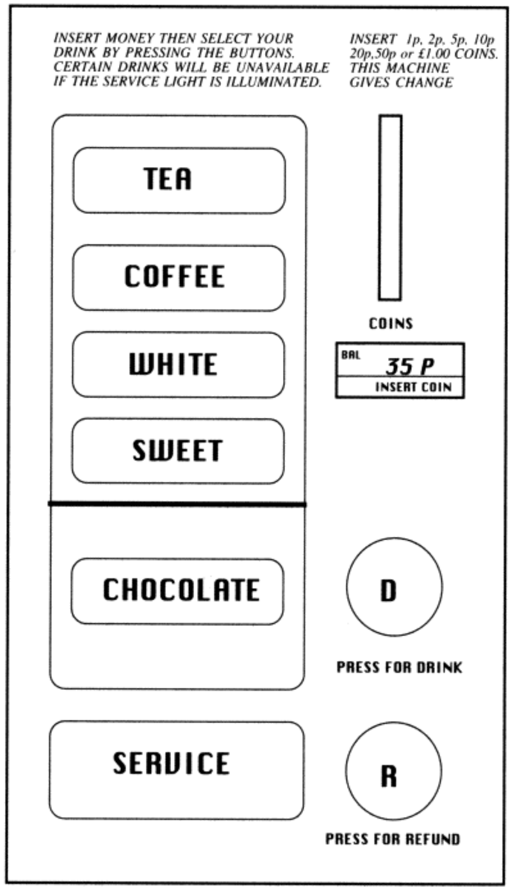
In diesem Beispiel hätten die Quantoren auch in anderer Reihenfolge enden können, wenn in anderer Reihenfolge umgeformt worden wäre. Das ist nicht allgemein so.

Jede prädikatenlogische Formel kann durch Anwenden der Äquivalenzen und Umbenennung von Variablen in die Form gebracht werden $Q_1x_1\ Q_2x_2\ ...\ Q_nx_n\ F$ (**Pränexform**) wobei Q_i Quantoren sind und F keine Quantoren enthält.

Ausschnitt aus einer Spezifikation in Z

Die **Spezifikationssprache Z** basiert auf typisierter Mengentheorie (Wertebereiche wie in Abschnitt 2) und verwendet Prädikatenlogik.

Ausschnitt aus der Fallstudie „A Drinks Dispensing Machine“ aus
Deri Sheppard: An Introduction to Formal Specification with Z and VDM, McGraw-Hill, 1994, S. 271ff



Get_Drink

$$
\begin{array}{l}
\Delta Abs\_State\_Machine \\
choice? : \mathbb{P} Selection\_buttons \\
d! : Drink \\
Change! : \mathrm{bag}\ British\_coin \\
\hline
choice? \in Drink \\
Value\ Balance \geq Prices\ choice? \\
\forall i : Recipe\ choice? \bullet count\ Stock\ i > 0 \\
Cups > 0 \\
\exists b : bag\ British\_coins \bullet (b \sqsubseteq Takings \land Value\ Balance = Value\ b + Prices\ choice?) \\
Balance' = [\![\ ]\!] \\
Stock' \uplus \{i : Recipe\ choice? \bullet i \mapsto 1\} = Stock \\
Cups' = Cups - 1 \\
Change! \sqsubseteq Takings \land Value\ Balance = Value\ Change! + Prices\ choice? \\
Takings' \uplus Change! = Takings \\
Prices' = Prices \\
Service\_light' = Service\_light \\
Report\_display' = insert\ coin \\
d! = choice?
\end{array}
$$

4. Modellierung mit Graphen

Modellierung beschreibt **Objekte und Beziehungen** zwischen ihnen.

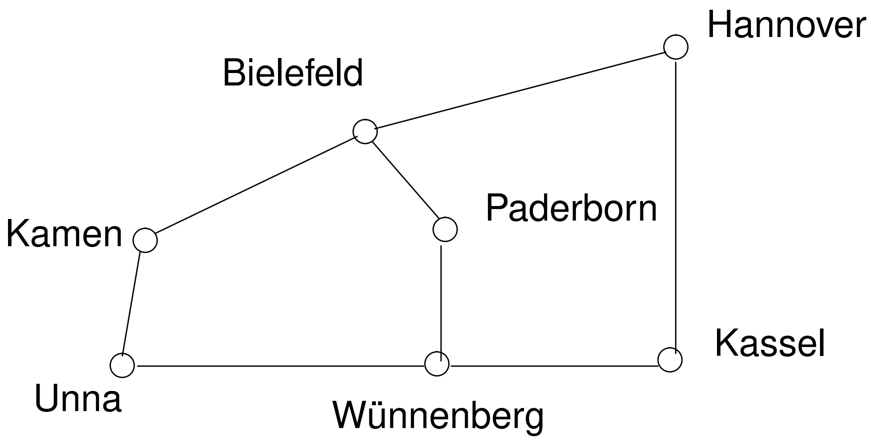
Graphen eignen sich zur Modellierung für ein **breites Aufgabenspektrum**.

Ein **Graph** ist eine Abstraktion aus Knoten und Kanten:

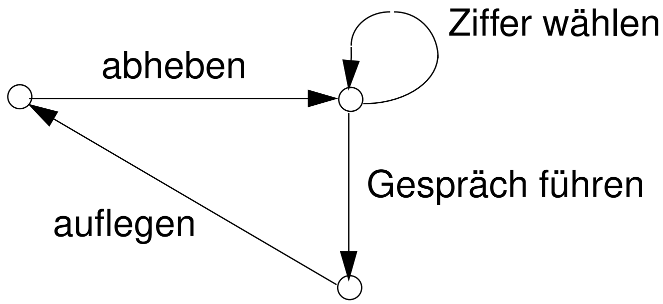
- **Knoten**: Eine Menge gleichartiger Objekte
- **Kanten**: Beziehung zwischen je zwei Objekten, 2-stellige Relation über Knoten

Je nach Aufgabenstellung werden **ungerichtete oder gerichtete** Graphen verwendet.

ungerichtet



gerichtet



Beschränkung auf **endliche Knotenmengen** und **2-stellige** Relation reicht hier aus.

Themenübersicht

4.1 Grundlegende Definitionen
gerichteter, ungerichteter Graph, Graphdarstellungen,
Teilgraphen, Grad, Markierungen

4.2 Wegeprobleme
Weg, Kreis, Rundwege, Zusammenhang

4.3 Verbindungsprobleme
Spannbaum

4.4 Modellierung mit Bäumen
gewurzelte Bäume, Entscheidungsbäume, Strukturbäume, Kantorowitsch-Bäume

4.5 Zuordnungsprobleme
konfliktfreie Markierung, bipartite Graphen

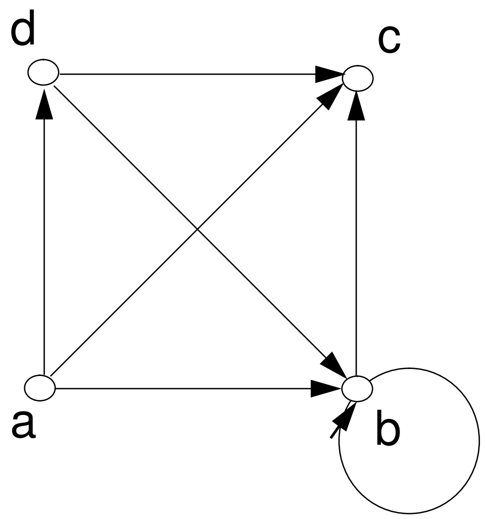
4.6 Abhängigkeitsprobleme
Anordnungen, Abfolgen

4.1 Grundlegende Definitionen Gerichteter Graph

Ein **gerichteter Graph** G = (V, E) hat eine endliche **Menge V von Knoten** und eine **Menge E gerichteter Kanten**, mit $E \subseteq V \times V$.

Die Kantenmenge E ist eine **2-stellige Relation** über V.

Beispiel:



V = {a, b, c, d}

E = {(a, b), (a, c), (a, d), (b, b), (b, c), (d, b), (d, c)}

Eine Kante wird als (v, u) oder v -> u notiert.

Eine Kante (v, v) heißt **Schleife** oder Schlinge.

Die Definition von Graphen schränkt ein auf

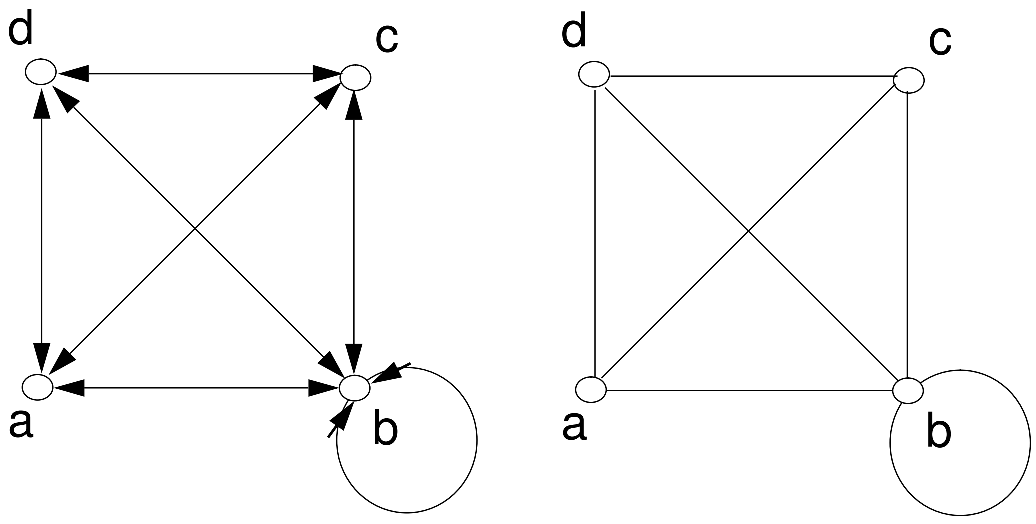
- endliche Graphen mit **endlichen Knotenmengen**,
- einfache Kanten: eine **Kante verbindet nicht mehr als 2 Knoten**, **zwischen zwei Knoten gibt es höchstens eine Kante**

Multigraph: Es kann mehr als eine Kante zwischen denselben Knoten geben (siehe Mod-4.7)

Ungerichteter Graph

Ist die **Kantenmenge** E eines gerichteten Graphen eine **symmetrische Relation**, so beschreibt er einen **ungerichteten Graphen**:
Zu jeder Kante x -> y aus E gibt es auch y -> x in E.

Wir fassen zwei Kanten x -> y, y -> x
zu einer **ungerichteten Kante** zusammen:
{x, y} die Menge der Knoten, die die Kante verbindet.



Ungerichtete Graphen werden auch direkt definiert:

Ein **ungerichteter Graph** G = (V, E) hat eine endliche **Menge V von Knoten** und eine **Menge E ungerichteter Kanten, mit E** $\subseteq$ **{ {x, y} | x, y** $\in$ **V }**

Der abgebildete Graph mit ungerichteten Kanten:
V = {a, b, c, d}   E = { {a, b}, {a, c}, {a, d}, { b }, {b, c}, {d, b}, {d, c} }

In dieser Notation ist eine Schleife eine 1-elementige Menge, z. B. { b }

Darstellung von Graphen

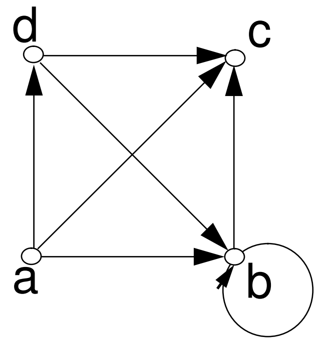
abstrakt:

Knotenmenge V = {a, b, c, d}

Kantenmenge E = {(a, b), (a, c), (a, d), (b, b), (b, c), (d, b), (d, c)}

anschaulich:

Graphik



Datenstrukturen für **algorithmische Berechnungen**:

Knotenmenge V als Indexmenge

lineare Ordnung der Knoten definieren

`a, b, c, d`

sei |V| = n

Adjazenzmatrix AM mit n * n Wahrheitswerten zur Darstellung der (gerichteten) Kanten:

AM(i, j) = (i, j) ∈ E

	a	b	c	d
a	f	w	w	w
b	f	w	w	f
c	f	f	f	f
d	f	w	w	f

Adjazenzlisten: zu jedem Knoten i eine Folge von Knoten, zu denen er eine Kante hat (i, j) ∈ E

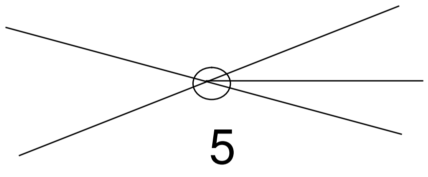
a	(b, c, d)
b	(b, c)
c	()
d	(b, c)

Ungerichtete Graphen als gerichtete Graphen mit symmetrischer Kantenmenge darstellen

Teilgraph, Knotengrad

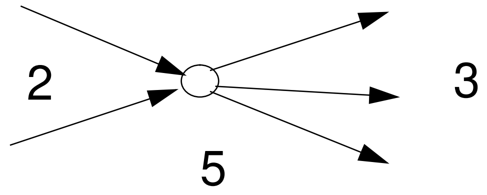
Der Graph G' = (V', E') ist ein **Teilgraph** des Graphen G = (V, E), wenn **V' $\subseteq$ V und E' $\subseteq$ E**.
(Gilt für **gerichtete** und **ungerichtete** Graphen.)

Zu einem Graphen G = (V, E) **induziert** eine Teilmenge der Knoten V' $\subseteq$ V den **Teilgraphen** G' = (V', E'), wobei E' alle Kanten aus E enthält, deren Enden in V' liegen.



Sei G = (V, E) ein **ungerichteter** Graph:

Der **Grad** eines Knotens v ist die Anzahl der Kanten {x, v}, die in v enden.



Sei G = (V, E) ein **gerichteter** Graph:

Der **Eingangsgrad** eines Knotens v ist die Anzahl der Kanten (x, v) $\in$ E, die in v münden.
Der **Ausgangsgrad** eines Knotens v ist die Anzahl der Kanten (v, x) $\in$ E, die von v ausgehen.
Der **Grad** eines Knotens v ist die Summe seines Eingangs- und Ausgangsgrades.

Der Grad eines gerichteten oder ungerichteten Graphen ist der maximale Grad seiner Knoten.

Markierte Graphen

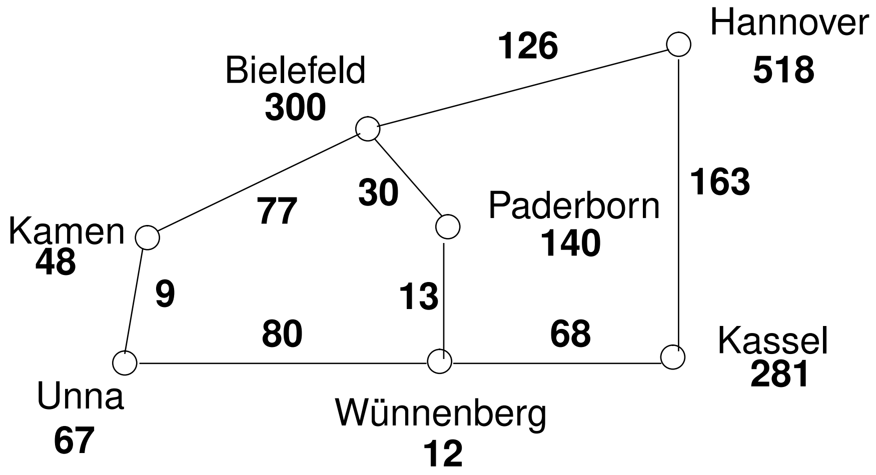
Ein Graph G = (V, E) modelliert eine Menge von Objekten V und die Existenz von Beziehungen zwischen ihnen.

Viele Aufgaben erfordern, dass den **Knoten und/oder den Kanten weitere Informationen** zugeordnet werden.

Dies leisten **Markierungsfunktionen**

Knotenmarkierung $MV : V \rightarrow WV$,
z.B. Einwohnerzahl: $V \rightarrow \mathbb{N}$

Kantenmarkierung $ME : E \rightarrow WE$,
z.B. Entfernung: $E \rightarrow \mathbb{N}$



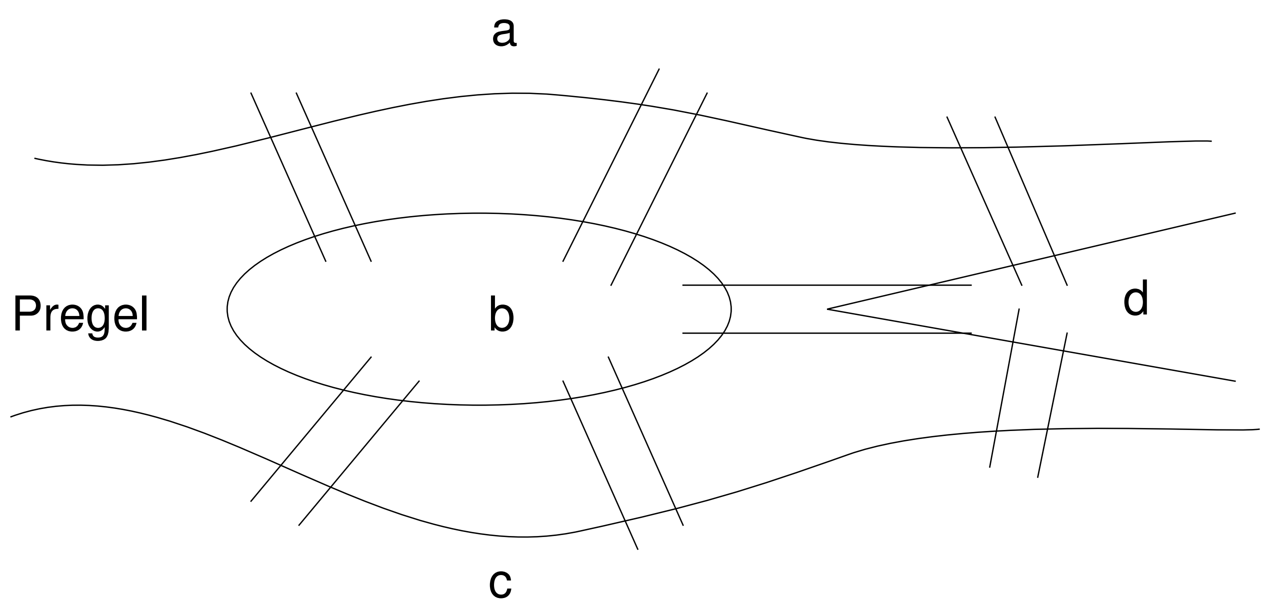
Spezielle Kantenmarkierungen:

Ordnung: $E \rightarrow \mathbb{N}$ legt die **Reihenfolge der Kanten** fest, die von einem Knoten ausgehen (z. B. im Kantorowitsch-Baum von links nach rechts, siehe Mod-2.19)

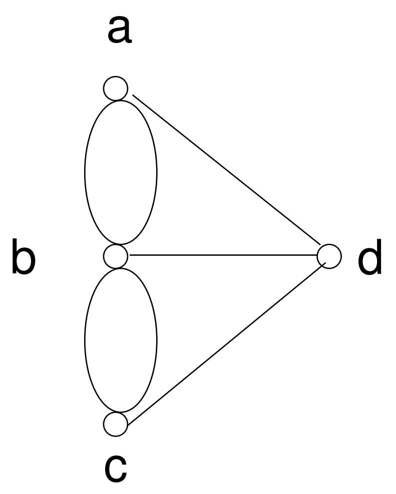
Anzahl: $E \rightarrow \mathbb{N}$ modelliert **mehrfache Verbindungen zwischen denselben Knoten**;
G ist dann ein **Mehrfachgraph (Multigraph)**.
In der graphischen Darstellung schreibt man die Anzahl an die Kante oder zeichnet mehrere Kanten.

4.2 Wegeprobleme

Beispiel: Königsberger Brückenproblem (Euler, 1736)



Skizze von Königsberg



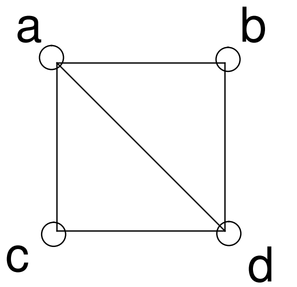
Multigraph dazu

a. Gibt es einen Weg, der jede der 7 Brücken genau einmal überquert und zum Ausgangspunkt zurückkehrt?

b. Gibt es einen Weg, der jede der 7 Brücken genau einmal überquert?

Wege und Kreise

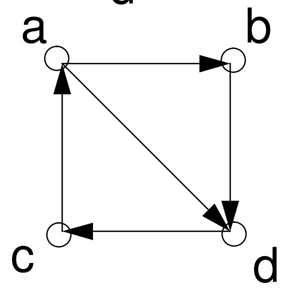
Sei G = (V, E) ein ungerichteter Graph.
Eine Folge von Knoten $(v_0, v_1, ..., v_n)$ mit $\{v_i, v_{i+1}\} \in E$ heißt ein **Weg von v_0 nach v_n**. Er hat die **Länge n** ≥ 0.



G_1

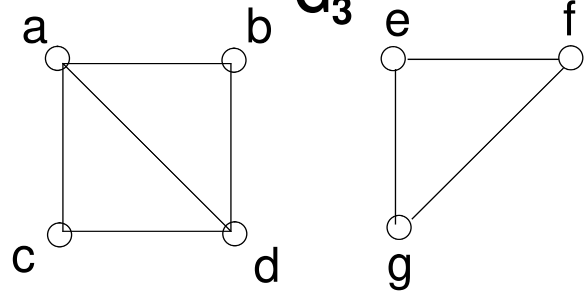
Entsprechend für gerichtete Graphen:
mit $(v_i, v_{i+1}) \in E$ für i = 1,..., n-1

G_2



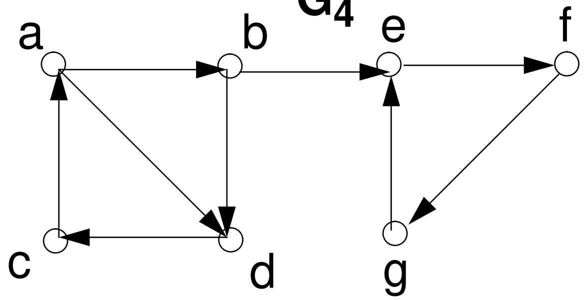
Ein Weg $(v_0, v_1, ..., v_n)$ einer Länge $n \geq 1$ mit $v_0 = v_n$ und paarweise verschiedenen Kanten $(v_0, v_1), ..., (v_{n-1}, v_n)$ heißt **Kreis im ungerichteten** Graphen und **Zyklus im gerichteten** Graphen.

G_3



Ein gerichteter Graph der keinen Zyklus enthält heißt **azyklischer Graph** (engl. **directed acyclic graph, DAG**).

G_4



Zusammenhang in Graphen

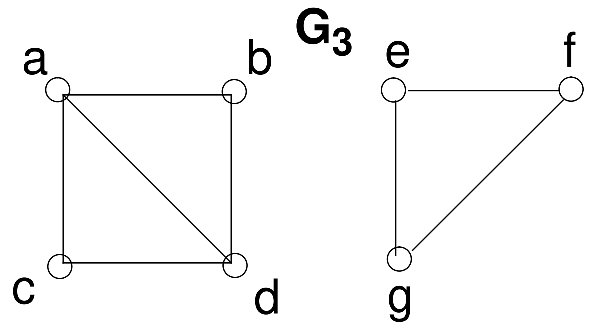
Ein Graph G = (V, E) heißt **zusammenhängend**, wenn es für beliebige Knoten v, w $\in$ V einen Weg von v nach w gibt.

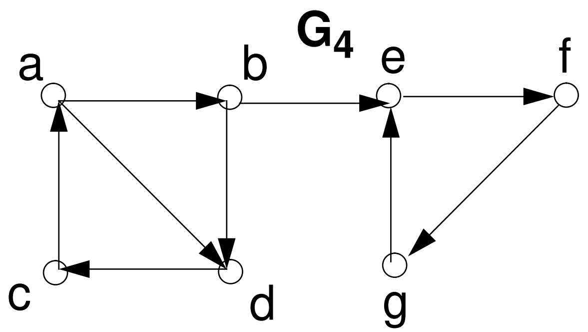
Ein gerichteter Graph heißt unter derselben Bedingung **stark zusammenhängend**.

Ein Teilgraph G' = (V', E') von G = (V, E) heißt **Zusammenhangskomponente**, wenn

- G' **zusammenhängend** ist und wenn
- G keinen anderen zusammenhängenden Teilgraphen G'' hat, der G' als Teilgraph enthält.

Zusammenhangskomponenten sind also
maximale Teilgraphen, die zusammenhängend sind.

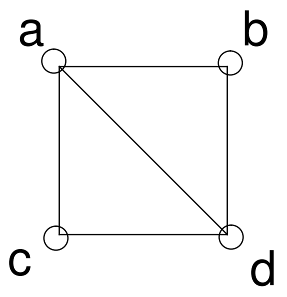




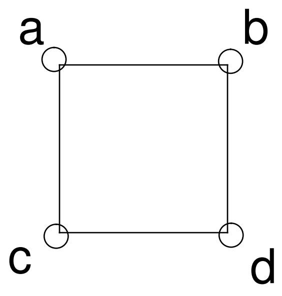
Spezielle Wege und Kreise

Sei G = (V, E) ein ungerichteter, zusammenhängender, schleifenfreier Graph.

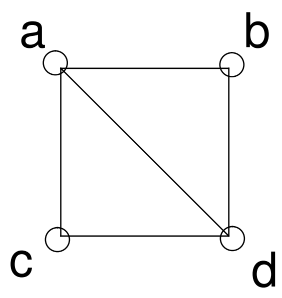
Ein **Euler-Weg** bzw. ein **Euler-Kreis** in G enthält **jede Kante aus E genau einmal**.



(a, b, d, a, c, d)
Euler-Weg



(a, b, d, c, a)
Euler-Kreis



(a, b, d, c, a)
Hamilton-Kreis

G hat einen **Euler-Kreis** genau dann, wenn **alle Knoten geraden Grad** haben.

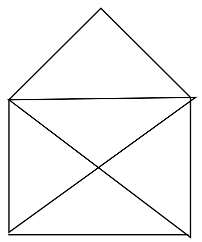
G hat einen **Euler-Weg**, der kein Kreis ist, genau dann,
wenn G genau **2 Knoten mit ungeradem Grad** hat.

Ein **Hamilton-Kreis** enthält **jeden Knoten aus V genau einmal**.

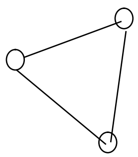
Wegeprobleme mit Euler-Wegen

1. Königsberger Brückenproblem (Mod-4.1):
   Euler-Weg, Euler-Kreis

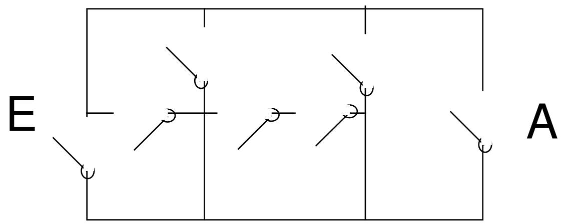
2. Kann man diese Figur in einem Zuge zeichnen?



3. Eine Inselgruppe mit n > 1 Inseln benötigt direkte Schiffsverbindungen zwischen allen Paaren von Inseln. Es gibt nur ein einziges Schiff. Kann es auf einer Tour alle Verbindungen genau einmal abfahren? Für welche n ist das möglich?

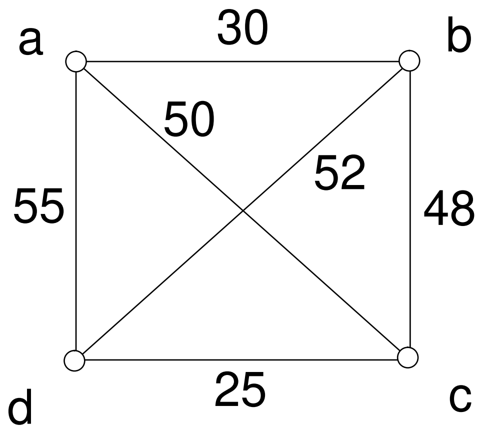


4. Planen Sie ein Gruselkabinett:
   Ein Haus mit n > 1 Räumen, 1 Eingangstür, eine Ausgangstür, beliebig vielen Innentüren. Jede Tür schließt nach Durchgehen endgültig. Die Besucher gehen einzeln durch das Haus. Es soll niemand eingesperrt werden.

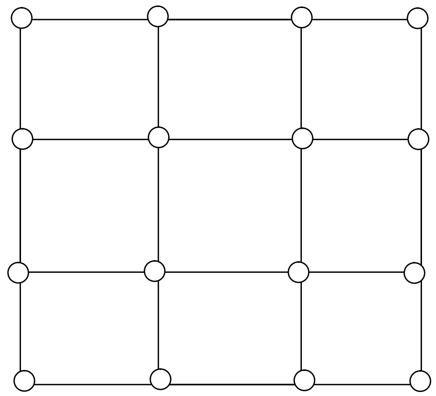


Wegeprobleme mit Hamilton-Kreisen

1. Traveling Salesman's Problem (Handlungsreisender): n Städte sind mit Straßen bestimmter Länge verbunden. Gesucht ist eine kürzeste Rundreise durch alle Städte.



2. In einem n * n Gitter von Prozessoren soll eine Botschaft sequentiell von Prozessor zu Prozessor weitergegeben werden. Sie soll jeden Prozessor erreichen und zum Initiator zurückkehren.
   Für welche n ist das möglich?



4.3 Verbindungsprobleme

Modellierung durch Graphen wie bei Wegeproblemen (4.2), aber hier interessiert
die **Existenz von Verbindungen** (Wegen) zwischen Knoten,
die **Erreichbarkeit** von Knoten,
nicht bestimmte Knotenfolgen.

Sei G = (V, E) ein
ungerichteter, zusammenhängender Graph
für alle folgenden Begriffe:

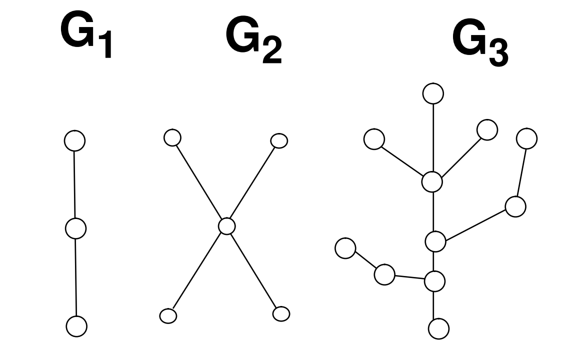
Wenn G **keine Kreise** enthält,
heißt er **(ungerichteter) Baum**.

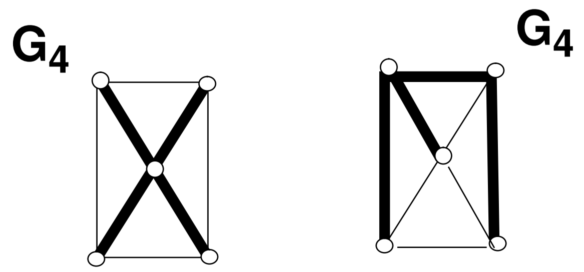
In Bäumen heißen **Knoten mit Grad 1 Blätter**.

Für jeden ungerichteten **Baum** G = (V, E) gilt
| E | = |V| - 1

Ein Teilgraph von G, der jeden Knoten aus V enthält und ein Baum ist, heißt **Spannbaum** zu G.

Bäume



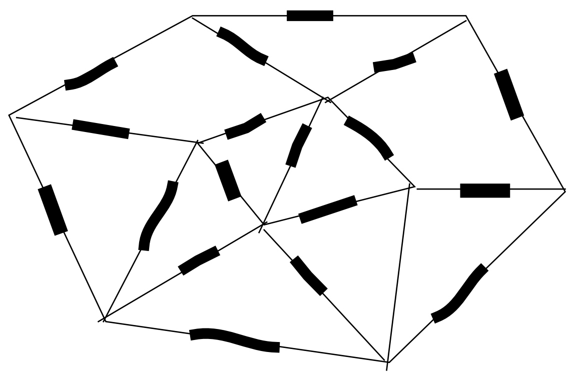


2 Spannbäume zu demselben Graphen G_4

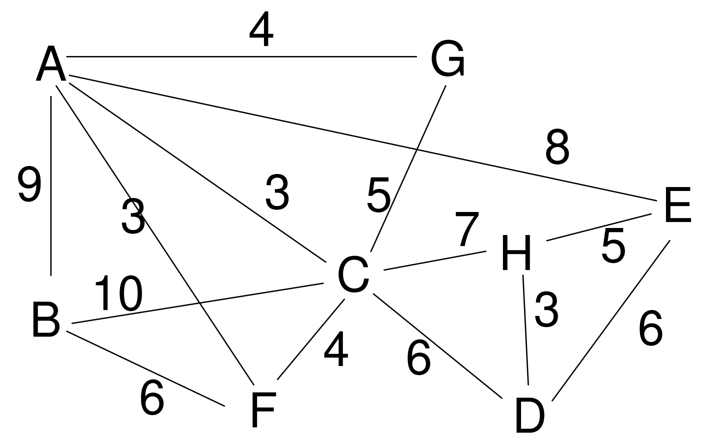
Modellierung mit Spannbäumen zu Graphen

Ein **Spannbaum** ist ein zusammenhängender Teilgraph mit der kleinsten Anzahl Kanten.
Er **modelliert kostengünstigen Zusammenhang**.

1. Aufständische Gefangene wollen eine minimale Anzahl von Gefängnistüren sprengen, so dass alle Gefangenen freikommen:



2. Alle Agenten A, ..., H sollen direkt oder indirekt miteinander kommunizieren. Die Risikofaktoren jeder paarweisen Verbindung sind:



Es soll ein Netz mit geringstem Risiko gefunden werden.

Verbindung und Zusammenhang

Sei G = (V, E) ein ungerichteter, zusammenhängender Graph.

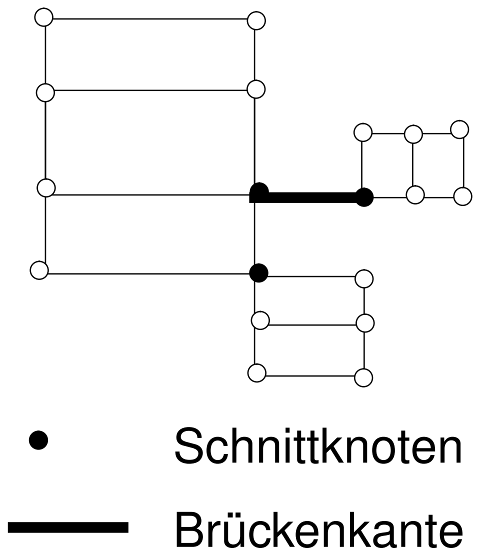
v ist ein **Schnittknoten** in G, wenn G **ohne v nicht mehr zusammenhängend** ist.

e ist eine **Brückenkante** in, wenn G **ohne e nicht mehr zusammenhängend** ist.

G heißt **orientierbar**, wenn man für **jede Kante eine Richtung** so festlegen kann, dass der entstehende **gerichtete Graph stark zusammenhängend** ist.

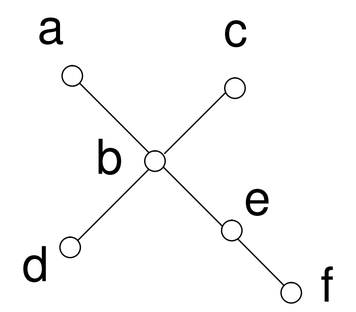
G ist genau dann **orientierbar**, wenn G **keine Brückenkante** hat.

1. In der Innenstadt sollen zur Hauptverkehrszeit alle Straßen zu Einbahnstraßen werden. Bleiben alle Plätze von überall erreichbar?
2. In einer Stadt sollen einzelne Straßen zur Reparatur gesperrt werden. Bleiben alle Plätze von überall erreichbar?

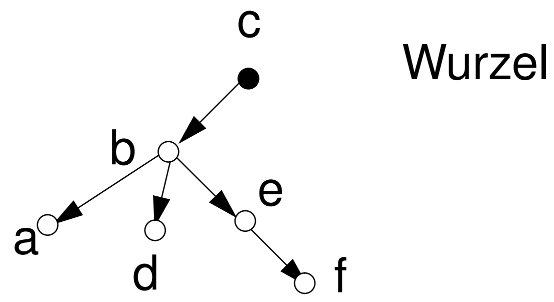


4.4 Modellierung mit Bäumen

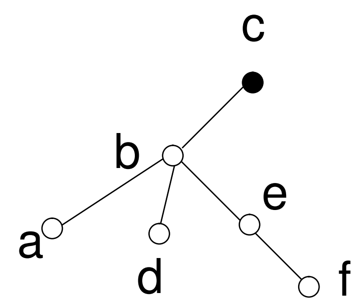
In einem **ungerichteten Baum** gibt es **zwischen zwei beliebigen Knoten genau einen Weg**.



Ein gerichteter, **azyklischer** Graph G ist ein **gerichteter Baum**, wenn alle Knoten einen **Eingangsgrad** ≤ **1** haben und es genau einen Knoten mit **Eingangsgrad 0 gibt, er ist die Wurzel** von G. G ist ein **gewurzelter Baum**.



Man kann aus einem **ungerichteten Baum** in eindeutiger Weise einen gerichteten machen, indem man **einen Knoten zur Wurzel bestimmt**.



Deshalb wird in gewurzelten Bäumen häufig die **Kantenrichtung nicht angegeben**.

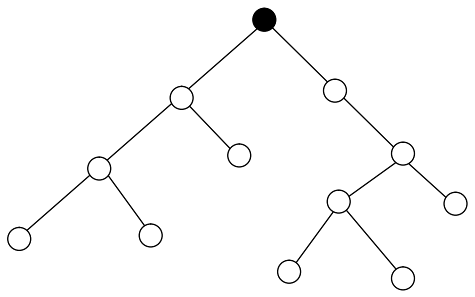
In einem gewurzelten Baum ist die **Höhe eines Knotens v** die größte Länge eines Weges von v zu einem Blatt. Die Höhe der Wurzel heißt **Höhe des Baumes**.

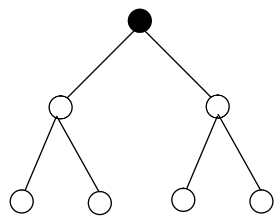
Knoten, die weder Wurzel noch Blatt sind heißen **innere Knoten**.

Binärbäume

Ein gewurzelter Baum heißt **Binärbaum**, wenn
seine Knoten einen **Ausgangsgrad von höchstens 2** haben.

Ein **Binärbaum heißt vollständig**, wenn jeder Knoten außer den Blättern den **Ausgangsgrad 2** hat und die **Wege zu allen Blättern gleich lang** sind.



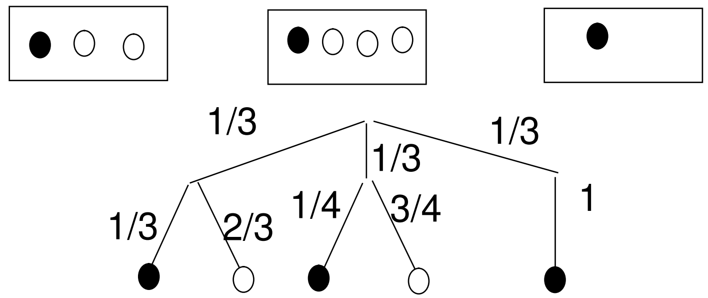


Ein vollständiger **Binärbaum der Höhe h** hat **2^h Blätter** und **2^{h+1}-1 Knoten**

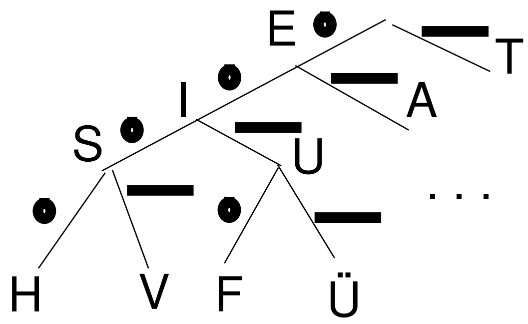
Modellierung von Entscheidungsbäumen

Knoten modelliert **Zwischenstand** einer
mehrstufigen Entscheidungsfolge
Kante modelliert eine der wählbaren **Alternativen**

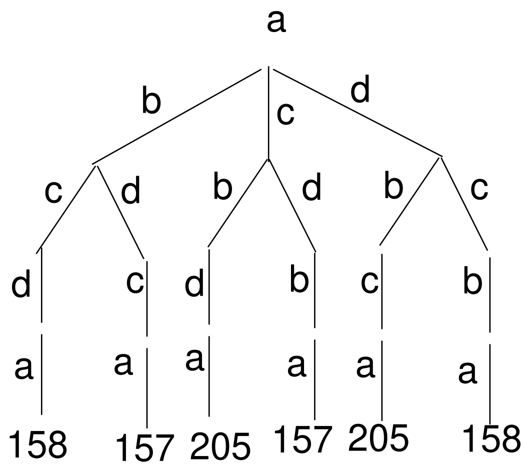
1. **Wahrscheinlichkeiten**,
   z. B. erst Schachtel, dann Kugel ziehen:



2. **Codierungen**,
   Z. B. Morse-Code



3. **Lösungsbaum** für kombinatorische Probleme,
   z. B. Traveling Salesman's Problem (Mod-4.12)
   Blätter repräsentieren einen Rundwege von a aus,
   Kanten sind mit Entscheidungen markiert



4. **Spielzüge**, z. B. Schach (ohne Bild)

Wird **derselbe Zwischenstand** durch verschiedene
Entscheidungsfolgen erreicht,
kann man **Knoten identifizieren**.
Es entsteht ein azyklischer oder zyklischer Graph.

Modellierung von Strukturen durch Bäume

Knoten modelliert ein **Objekt**.
Kante modelliert **Beziehung „besteht aus“**, „enthält“, „spezialisiert zu“, ...

Beispiele:

- **Typhierarchie**: Typ - Untertypen
- **Klassenhierarchie**: Oberklasse als Abstraktion ihrer Unterklassen (Mod-4.19a)
  Vererbungshierarchie: Unterklassen erben von ihrer Oberklasse
- **Objektbaum**: Objekt enthält (Referenzen auf) Teilobjekte
- **Kantorowitsch-Baum**: Operator mit seinen Operanden (Mod-4.19b)
- **Strukturbaum**: (Programm-)Struktur definiert durch eine kontextfreie Grammatik (Mod-4.19c)

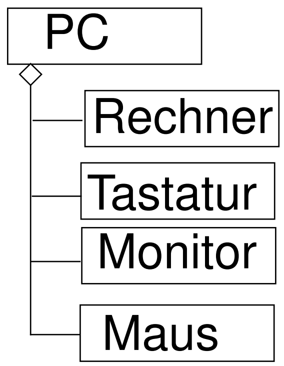
Identifikation gleicher Teilbäume führt zu azyklischen Graphen (DAGs).

Vorsicht:
Identifikation muss mit der **Bedeutung der Kanten verträglich** sein;
z. B. Ein Gegenstand kann nicht dasselbe Objekt mehrfach als Teil enthalten,
wohl aber mehrere Objekte derselben Art.

Klassen- und Objekthierarchien

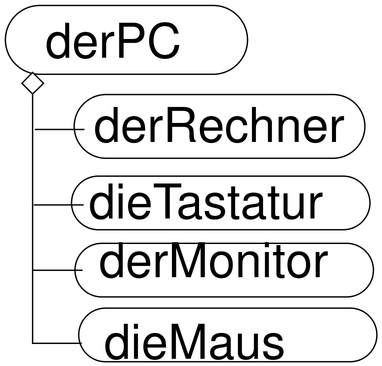
Kompositionsbeziehung im Klassendiagramm (UML Notation):
Knoten: **Klassen**
Kanten: definieren, **aus welcher Art von Objekten** ein Objekt besteht
z. B. ein Objekt der Klasse PC besteht aus
einem Rechner-Objekt, einem Tastatur-Objekt, ...

Diese Beziehung zwischen den Klassen könnte
auch ein allgemeiner Graph sein



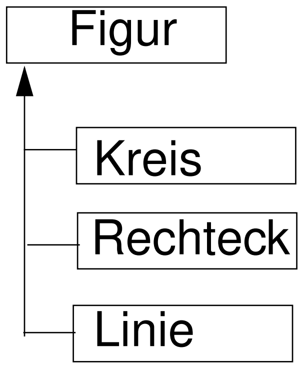
Objektbaum im Objektdiagramm (fast UML):
Knoten: **Objekte**
Kanten: definieren, **aus welchen Objekten** ein Objekt besteht
z. B. dieser PC besteht aus, diesem Rechner, ...

Diese Beziehung muss **konzeptionell ein Baum** sein.



Vererbungsbeziehung im Klassendiagramm (UML Notation):
Knoten: **Klassen**
Kanten: Unterklasse **erbt von** -> Oberklasse
Oberklasse **ist Abstraktion** <- ihrer Unterklassen
Kanten sind zur Wurzel hin gerichtet

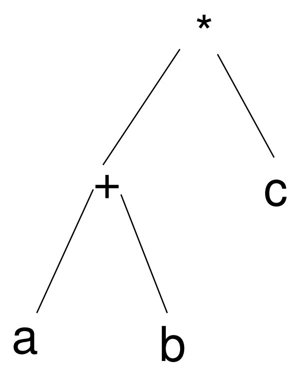
Baum bei Einfachvererbung (Java)
azyklischer Graph bei Mehrfachvererbung (C++)



Kantorowitsch-Bäume

Darstellung der Struktur von Termen, Formeln, Ausdrücken
(siehe Mod-2.19)

(a + b) * c



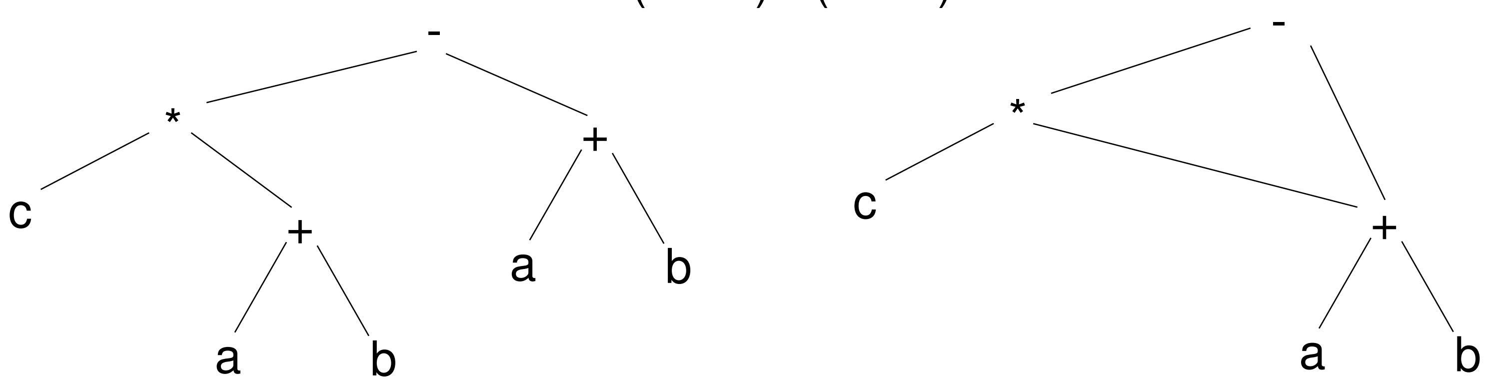
Knoten: Operator

Kanten: Verbindung zu den Operanden eines Operators
Die Kanten sind geordnet (Kantenmarkierung):
erster, zweiter, ... Operand

Identifikation gleicher Teilbäume führt zu azyklischen Graphen (DAGs):

Z. B. identifizieren Übersetzer gleiche Teilbäume, um Code zu erzeugen, der sie nur einmal auswertet:

c * (a + b) - (a + b)



Strukturbäume zu kontextfreien Grammatiken

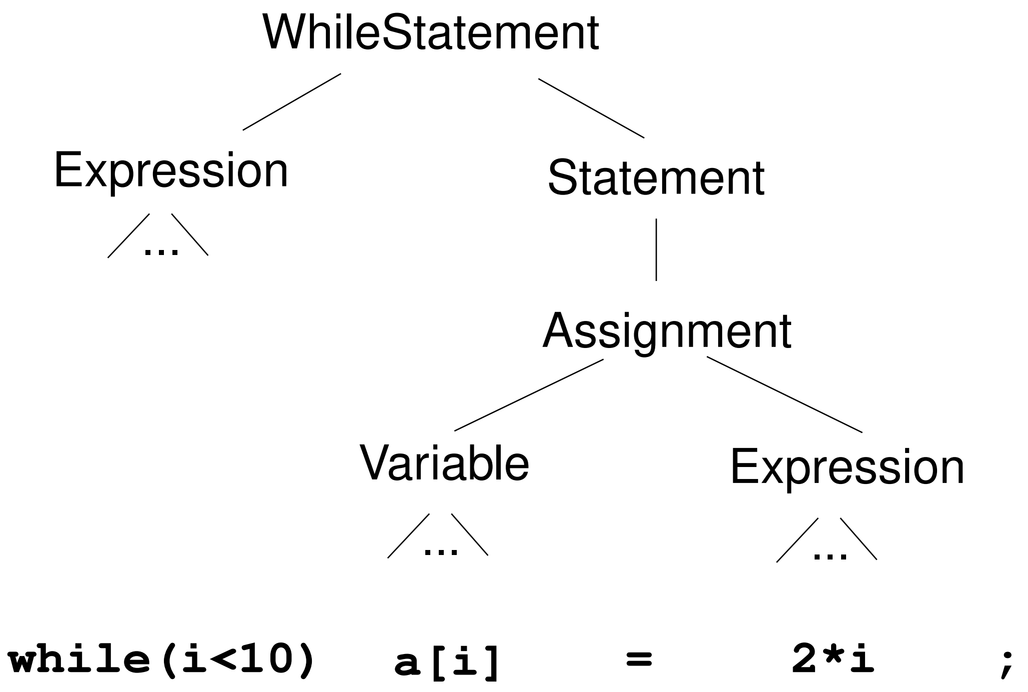
Kontextfreie Grammatiken definieren die Struktur von Programmen, Texten oder Daten.
Ein Programm, Text oder strukturierte Daten werden als Strukturbaum dargestellt.

Knoten: Programmkonstrukt (Nichtterminal der Grammatik)

Kante: Bezug zu Bestandteilen des Programmkonstruktes (Produktion der Grammatik)

Für die Repräsentation von Texten sind die **Kanten geordnet** (Kantenmarkierung)

Strukturbaum:



Produktionen aus der kontextfreien Grammatik:

Statement ::= Assigment

Statement ::= WhileStatement

...

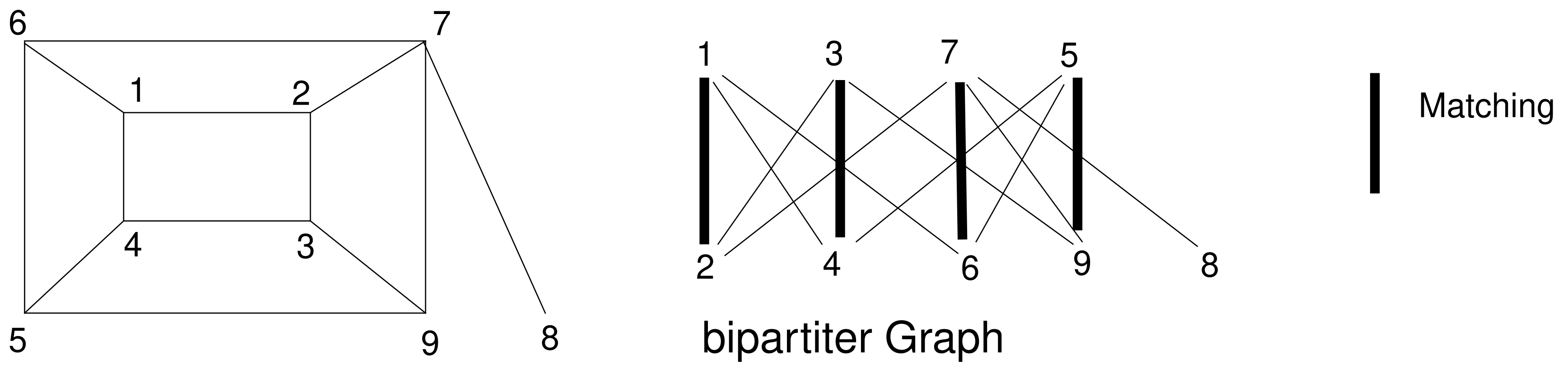
WhileStatement ::= Expression Statement

Assignment ::= Variable Expression

4.5 Zuordnungsprobleme

Aufgabenklasse **paarweise Zuordnung (Matching):**
Im ungerichteten Graphen G = (V, E) modelliert eine Kante {a, b} „a passt zu b“,
ggf. mit einer Kantenmarkierung als Abstufung

Gesucht ist eine **maximale Menge unabhängiger Kanten**,
das ist einTeilgraph M mit allen Knoten aus V und möglichst vielen Kanten aus E,
so dass der **Grad der Knoten höchstens 1** ist. M heißt ein **Matching** der Knoten von G.



bipartiter Graph

Graph G heißt **bipartit**, wenn V in **2 disjunkte Teilmengen $V = V_1 \cup V_2$** zerlegt werden kann,
so dass jede Kante zwei Knoten aus verschiedenen Teilmengen verbindet.

Häufig liefert die Aufgabenstellung schon bipartite Graphen, sogenannte **Heiratsprobleme**:

Mann - Frau Aufgabe - Bearbeiter Verbraucher - Produkte

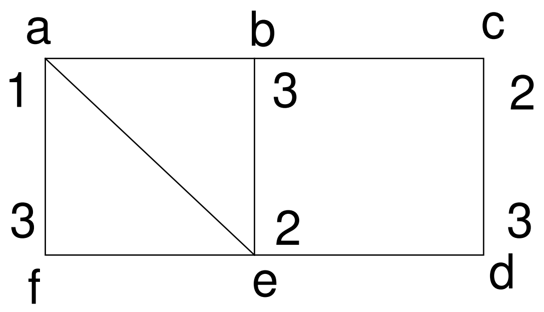
Konfliktfreie Knotenmarkierung (Färbung)

Aufgabenklasse **konfliktfreie Knotenmarkierung (Färbung):**

Im ungerichteten Graphen G = (V, E) modelliert eine Kante {a, b} „a ist unverträglich mit b“,

Gesucht ist eine Knotenmarkierung Färbung: V -> $\mathbb{N}$ („Farben“),
so dass durch eine Kante verbundene Knoten verschiedene Marken haben

Die chromatische Zahl eines Graphen G ist die minimale Zahl verschiedener „Farben“, die nötig ist, um G konfliktfrei zu markieren.



Es gilt: chromatische Zahl $\leq$ 1 + maximaler Knotengrad

Anwendungen:

Knoten:	**Kante:**	**Farbe / Marke:**
Staat auf Landkarte	gemeinsame Grenze	Farbe
Partygast	unverträglich	Tisch
Kurs	haben gemeinsame Teilnehmer	Termin
Prozess	benötigen gleiche Ressource	Ausführungszeitpunkt
Variable im Programm	gleichzeitig lebendig	Registerspeicher

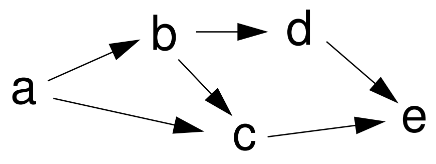
4.6 Abhängigkeitsprobleme

Graphen modellieren **Abhängigkeiten** zwischen Operationen und **Ausführungsreihenfolgen** von Operationen.

Abhängigkeitsgraph: gerichtet, azyklisch, voneinander abhängige Operationen.

Aufgaben dazu: sequentielle oder parallele **Anordnungen finden** (engl. **scheduling**).

Knoten: **Operation**, Ereignis; ggf. mit Dauer markiert
Kante: a -> b a ist **Vorbedingung** für b oder
b **benutzt** Ergebnis von a oder
a liest oder schreibt Ressource
bevor b sie überschreibt

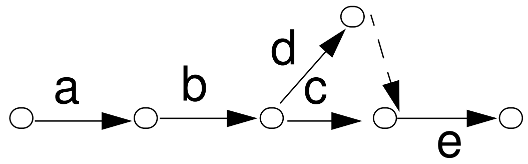


Anwendungen:

- **Projektplanung** mit abhängigen Teilaufgaben (PERT, CPM)
- abhängige **Transaktionen** mit einer Datenbank
- **Anordnung von Code** für die parallele Auswertung von Ausdrücken (Übersetzer)

Kritischer Pfad: längster Weg von einem Anfangsknoten zu einem Endknoten

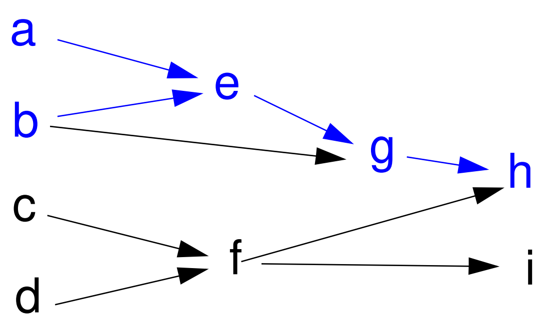
Duale Modellierung:
Knoten: Ereignis, Anfang und Ende einer Operation
Kante: Operation, ggf. mit Dauer markiert



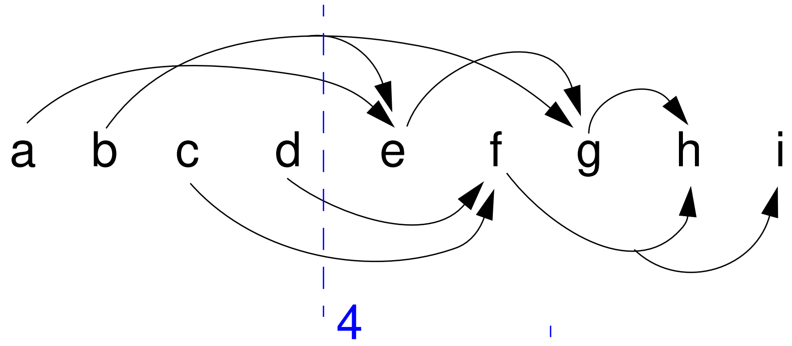
Anordnung von Abhängigkeitsgraphen

Anordnungsaufgaben:
gegebener Abhängigkeitsgraph

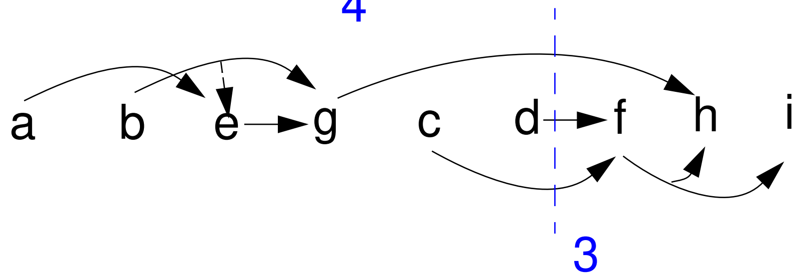
kritischer Pfad



sequentielle Anordnung der Knoten,
so dass **alle Kanten vorwärts** zeigen.

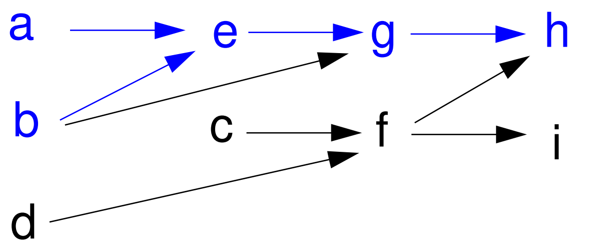


Meist sollen **Randbedingungen** erfüllt werden,
z. B. geringste Anzahl gleichzeitig benötigter Zwischenergebnisse im Speicher



parallele Anordnung mit
beschränkter Parallelität 3

Länge: 4 Schritte (Operationen)



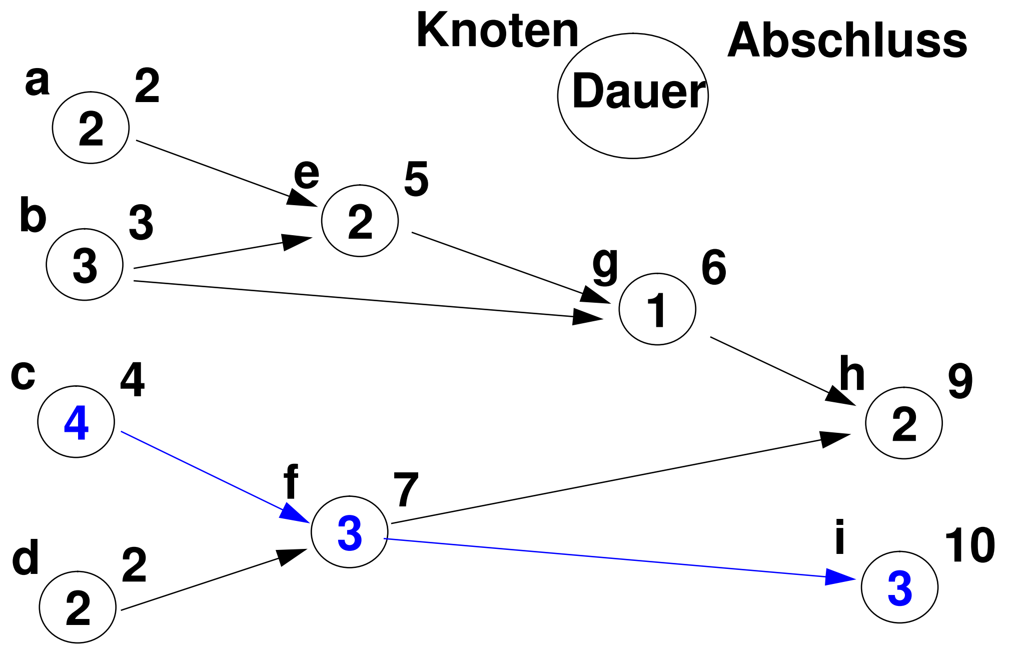
Operationen unterschiedlicher Dauer

Zwei **Knotenmarkierungen**:

Dauer der Operation und

frühester Abschlusstermin
= max. Abschluss der Vorgänger
+ Dauer des Knotens

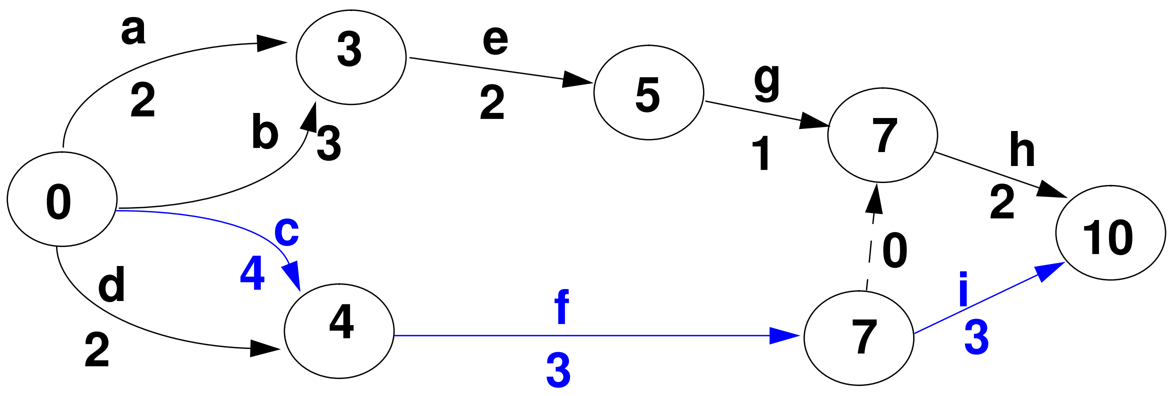
Kritischer Pfad gemäß maximaler
Summe der Dauer der Operationen



Duale Modellierung:

Kante: Operation
mit **Dauer** als Marke
Mehrfachkanten, Multigraph

Knoten: Ereignis
„vorangehende Operationen
sind abgeschlossen“
mit frühestem **Abschlusstermin**
als Marke



Ablaufgraphen

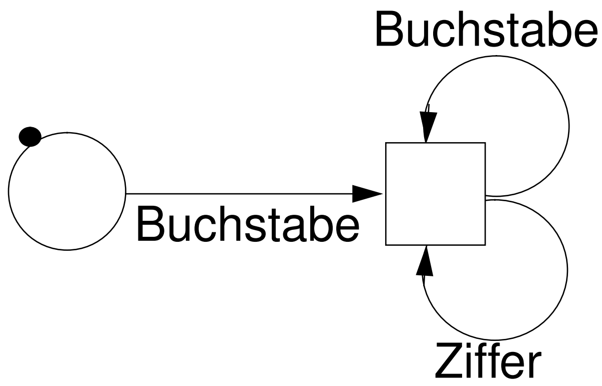
Gerichteter Graph (auch zyklisch) modelliert Abläufe.

Knoten: Verzweigungsstelle, Zustand
Kanten: Fortsetzungsmöglichkeit

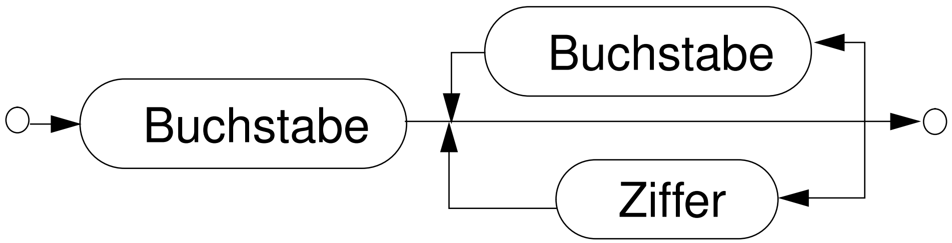
Jeder **Weg durch den Graphen** beschreibt einen **potenziellen Ablauf**
Die **Folge der Markierungen eines Weges** kann einen **Satz einer Sprache** modellieren.

Anwendungen:

- **Endlicher Automat** (siehe Kapitel 6)
  modelliert **Folgen von Zeichen**, Symbolen, ...
  Knoten:Zustand
  Kante: Übergang markiert mit Zeichen



- **Syntaxdiagramm**
  modelliert **Folgen von Zeichen**, Symbolen, ...
  Knoten: markiert mit Zeichen
  Kante: „kann folgen auf“
  dual zum endlichen Automaten



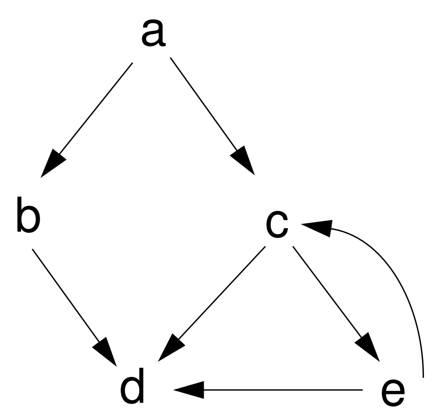
- **Aufrufgraphen** (siehe Mod-4.26)
- **Ablaufgraphen** (siehe Mod-4.27)

Aufrufgraphen

Gerichteter Aufrufgraph: **Aufrufbeziehung zwischen Funktionen** in einem Programm; wird benutzt in **Übersetzern** und in **Analysewerkzeugen** zur Software-Entwicklung.

Knoten: **Funktion** im Programm
Kante a -> b: Rumpf der Funktion a enthält einen Aufruf der Funktion b; **a könnte b aufrufen**

Zyklus im Aufrufgraph:
Funktionen, die sich **wechselweise rekursiv** aufrufen, z. B. (c, e, c)



Fragestellungen z. B.

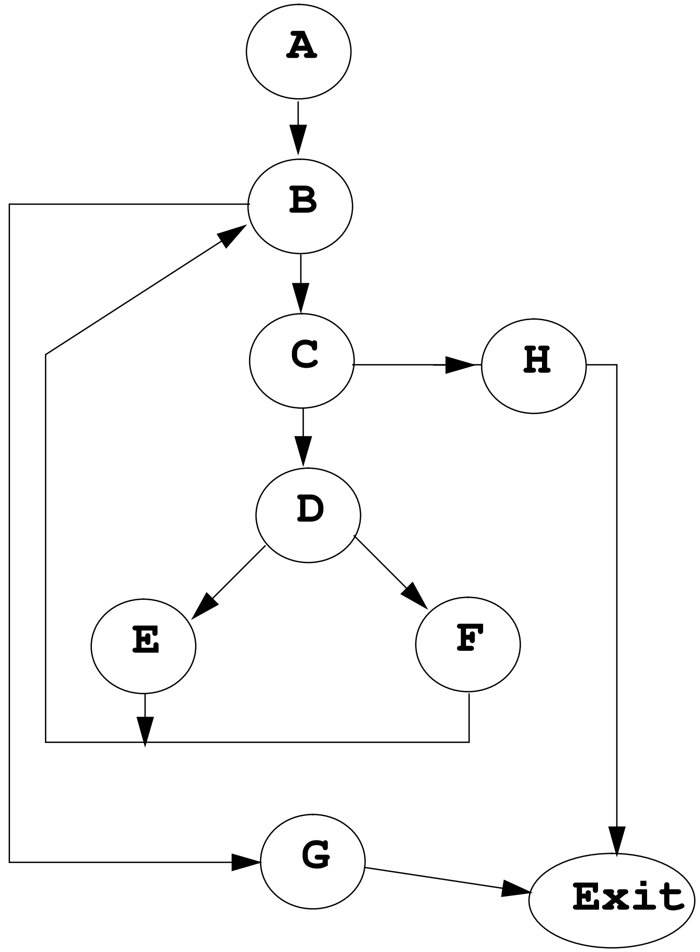
- Welche Funktionen sind nicht rekursiv?
- Welche Funktionen sind nicht (mehr) erreichbar?
- Indirekte Wirkung von Aufrufen,
  z. B. nur e verändere eine globale Variable x;
  welche Aufrufe lassen x garantiert unverändert? b, d

Programmablaufgraphen

Gerichteter Graph, modelliert **Abläufe durch ein verzweigtes Programm** (bzw. Funktion); wird benutzt in **Übersetzern** und in **Analysewerkzeugen** zur Software-Entwicklung.

Knoten: **unverzweigte Anweisungsfolge** (Grundblock), mit Verzweigung (Sprung) am Ende
Kante: **potenzieller Nachfolger im Ablauf**

```
ug = 0;                                 A
og = obereGrenze;
while (ug <= og)                        B
{   mitte = (ug + og) / 2;              C
    if (a[mitte] == x)
          return mitte;                 H
    else if (a[mitte] < x)              D
          ug = mitte + 1;               E
    else  og = mitte - 1;               F
}
return nichtGefunden;                   G
```



Fragestellungen, z. B.

- Menge von Wegen, die den **Graph überdecken**, **Software-Testen**
- Wege mit bestimmten Eigenschaften, **Datenflussanalyse**

Zusammenfassung zu Graphen

Problemklassen:

- Wegeprobleme
- Verbindungsprobleme
- Entscheidungsbäume
- hierarchische Strukturen
- Zuordnungsprobleme
- Abhängigkeitsprobleme
- Anordnungen in Folgen
- verzweigte Abläufe

Kanten- und Knotenbedeutung:

- verbunden, benachbart, ...
- Entscheidung, Alternative, Verzweigung
- Vorbedingung, Abhängigkeit
- (Un-)Verträglichkeit
- allgem. symmetrische Relation
- besteht aus, enthält, ist-ein
- (Halb-)Ordnungsrelation

Kanten-, Knotenmarkierungen:

- Entfernung, Kosten, Gewinn, ...
  bei Optimierungsproblemen
- „Färbung“, disjunkte Knotenmengen
  bei Zuordnungsproblemen
- Symbole einer Sprache

5. Modellierung von Strukturen
5.1 Kontextfreie Grammatiken

Kontextfreie Grammatik (KFG): formaler Kalkül, Ersetzungssystem; definiert

- **Sprache** als Menge von Sätzen; jeder **Satz** ist eine **Folge von Symbolen**
- **Menge von Bäumen**; jeder Baum repräsentiert die **Struktur eines Satzes** der Sprache

Anwendungen:

- Programme einer **Programmiersprache** und deren Struktur, z. B. Java, Pascal, C
- Sprachen als Schnittstellen zwischen Software-Werkzeugen, **Datenaustauschformate**, z. B. HTML, XML
- Bäume zur Repräsentation **strukturierter Daten**, z. B. in HTML
- Struktur von **Protokollen** beim Austausch von Nachrichten zwischen Geräten oder Prozessen

Beispiel zu HTML:

```
<table>
    <tr>
        <td>Mo</td>
        <td>11-13</td>
        <td>AM</td>
    </tr>
    <tr>
        <td>Fr</td>
        <td>9-11</td>
        <td>AM</td>
    </tr>
</table>
```

Kontextfreie Grammatik

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

T	**Menge der Terminalsymbole** (kurz: Terminale)
N	**Menge der Nichtterminalsymbole** (kurz: Nichtterminale) T und N sind disjunkte Mengen
$S \in N$	**Startsymbol** (auch Zielsymbol)
$P \subseteq N \times V^*$	**Menge der Produktionen**; $(A, x) \in P$, mit $A \in N$ und $x \in V^*$; statt (A, x) schreibt man A ::= x
$V = T \cup N$	heißt auch **Vokabular**, seine Elemente heißen **Symbole**

Man sagt „In der Produktion A ::= x steht A auf der **linken Seite** und x auf der **rechten Seite**.“
Man gibt Produktionen häufig **Namen**: p1: A ::= x
In Symbolfolgen aus V* werden die Elemente nur durch Zwischenraum getrennt: A ::= B C D

Beispiel:

Terminale **T** = { `(`, `)` }

Nichtterminale **N** = { `Klammern`, `Liste` }

Startsymbol **S** = `Klammern`

Name **Produktionsmenge P**

```
              N               V*
        {
p1:       Klammern ::= '(' Liste ')'
p2:       Liste    ::= Klammern Liste
p3:       Liste    ::=
        }
```

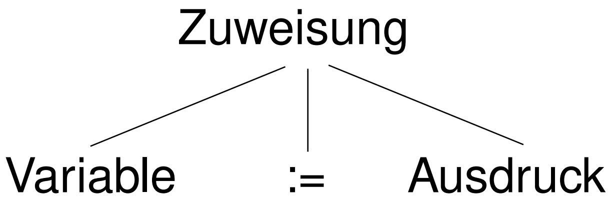
Bedeutung der Produktionen

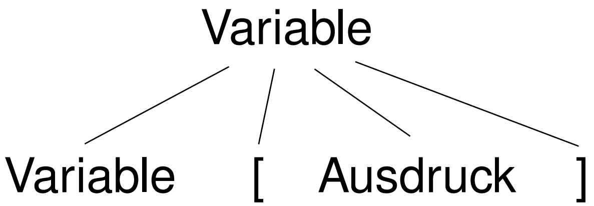
Eine Produktion A ::= x ist eine **Strukturregel**: A **besteht aus** x

Beispiele:

DeutscherSatz	::=	Subjekt Prädikat Objekt
Ein DeutscherSatz	*besteht aus (der Folge)*	Subjekt Prädikat Objekt
Klammern	::=	'(' Liste ')'
Zuweisung	::=	Variable ':=' Ausdruck
Variable	::=	Variable '[' Ausdruck ']'

Produktion graphisch als gewurzelter Baum mit geordneten Kanten und mit Symbolen als Knotenmarken:





Ableitungen

Produktionen sind **Ersetzungsregeln**: Ein Nichtterminal A in einer Symbolfolge u A v kann durch die rechte Seite x einer Produktion A ::= x ersetzt werden.
Das ist ein **Ableitungsschritt**; er wird notiert als u A v => u x v

z. B. **`Klammern Klammern Liste`** => **`Klammern ( Liste ) Liste`**
mit Produktion p1

Beliebig viele Ableitungsschritte nacheinander angewandt heißen **Ableitung**; notiert als u =>* v

Eine kontextfreie Grammatik **definiert eine Sprache**; das ist eine Menge von Sätzen.
Jeder Satz ist eine Folge von Terminalsymbolen, die aus dem Startsymbol ableitbar ist:

$L(G) = \{ w \mid w \in T^* \text{ und } S \Rightarrow^* w \}$

Grammatik auf Mod-5.2 definiert geschachtelte Folgen paariger Klammern als Sprachmenge:

{ `()`, `(())`, `(()()())`, `((()())(()))`, ... } $\subseteq$ L(G)

Ableitung des Satzes `(()())`:

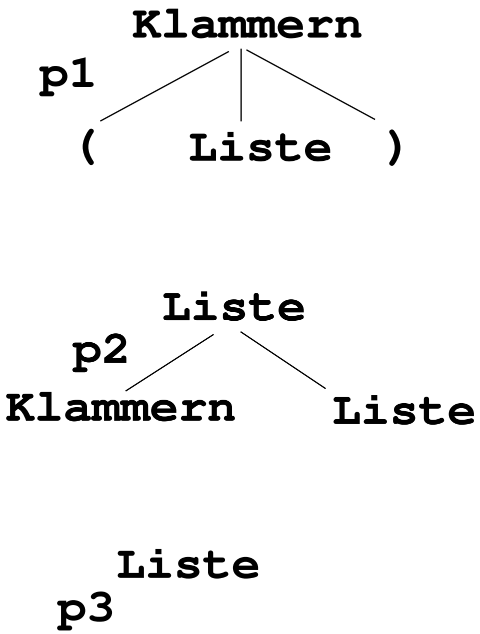
S **= Klammern**
=> (Liste)
=> (Klammern Liste)
=> (Klammern Klammern Liste)
=> (KLammern (Liste) Liste)
=> ((Liste) (Liste) Liste)
=> (( ) (Liste) Liste)
=> (( ) ( ) Liste)
=> (( ) ( ))

Ableitungsbäume

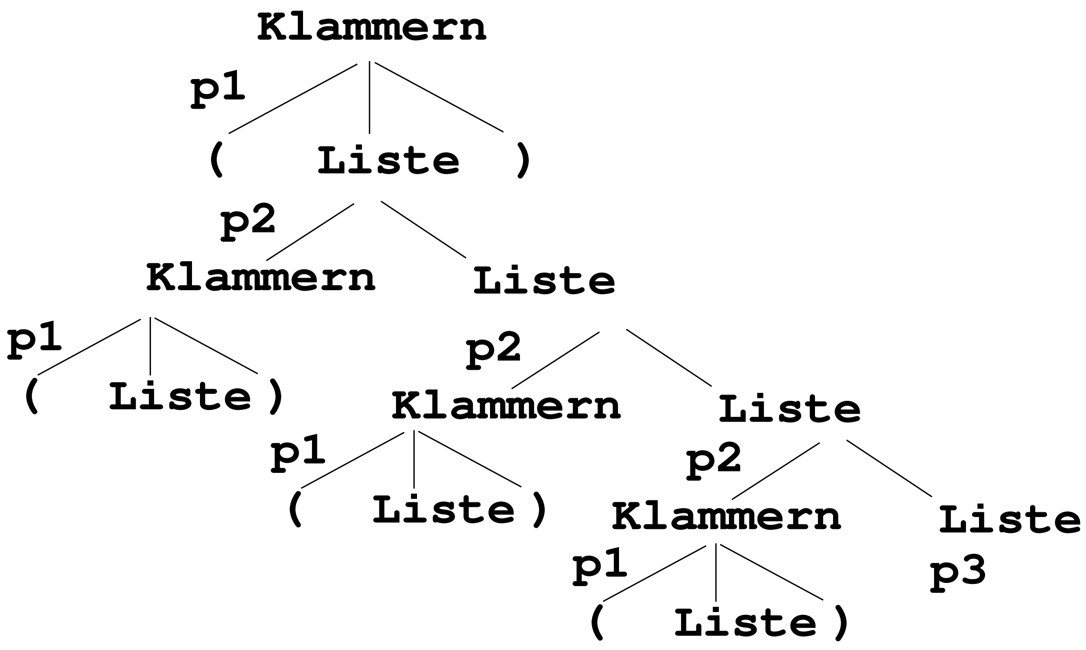
Jede Ableitung kann man als **gewurzelten Baum** darstellen:

- Die **Knoten** mit ihren Marken repräsentieren **Vorkommen von Symbolen**.
- Ein Knoten mit seinen direkten Nachbarn repräsentiert die **Anwendung einer Produktion**.
- Die **Wurzel** ist mit dem **Startsymbol** markiert.
- **Terminale** kommen nur an **Blättern** vor.

Produktionen:



ein Ableitungsbaum:



der Satz dazu: `(()()())`

Satz zum Baum: Terminale im links-abwärts Durchgang

Beispiel: Ausdrucksgrammatik

p1: Ausdruck ::= Ausdruck BinOpr Ausdruck

p2: Ausdruck ::= Zahl

p3: Ausdruck ::= Bezeichner

p4: Ausdruck ::= '(' Ausdruck ')'

p5: BinOpr ::= '+'

p6: BinOpr ::= '-'

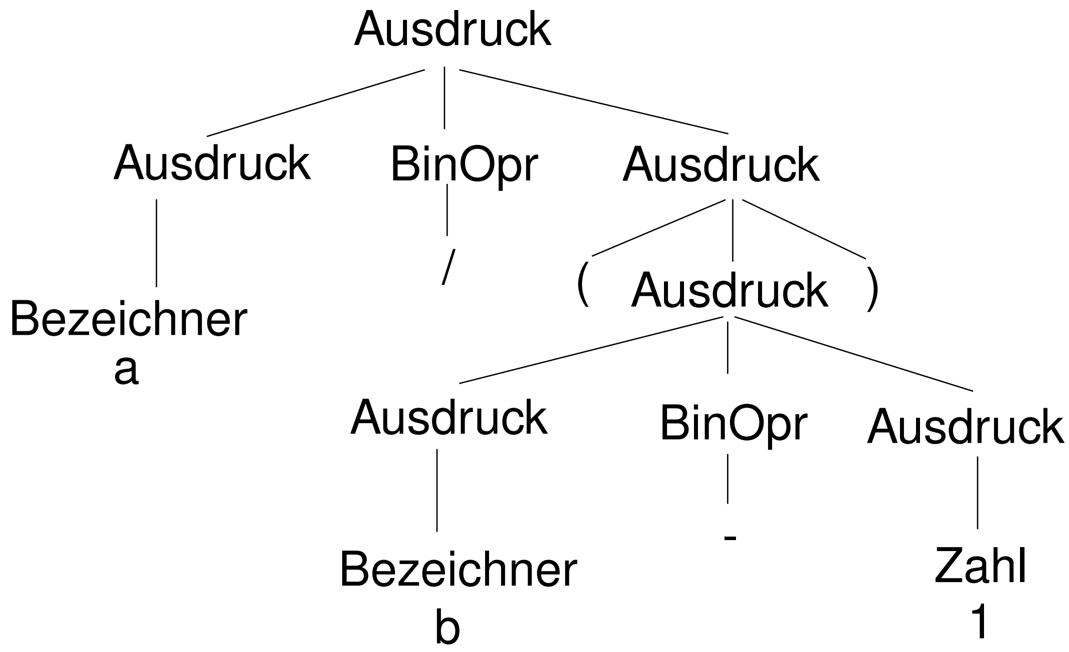
p7: BinOpr ::= '*'

p8: BinOpr ::= '/'

Startsymbol: Ausdruck

Terminale:
{ Zahl, Bezeichner, (,), +, -, *, / }

Ableitungsbaum zum Ausdruck
a / (b - 1)



Schreibweise der Terminale
Zahl und Bezeichner wird nicht in der KFG definiert.

Grammatik ist mehrdeutig: Es gibt **Sätze, die mehrere Ableitungsbäume** haben.

Beispiel: Tabellen in HTML

HTML: Hypertext Markup Language zur Darstellung von verzeigerten Dokumenten, insbesondere im WWW verwendet.
typisch: geklammerte Strukturen mit Klammern der Form **<x>...</x>**.
hier: vereinfachter Ausschnitt aus der Sprache zur Darstellung von Tabellen.

Produktionen der kontextfreien Grammatik:

```
Table   ::= '<table>' Rows '</table>'
Rows    ::= Row *
Row     ::= '<tr>' Cells '</tr>'
Cells   ::= Cell *
Cell    ::= '<td>' Text '</td>'
Cell    ::= '<td>' Table '</td>'
```

Beispieltext in HTML:

```
<table>
    <tr> <td>Tag</td>
         <td>Zeit</td>
         <td>Raum</td></tr>
    <tr> <td>Mo</td>
         <td>11:00-12.30</td>
         <td>AM</td></tr>
    <tr> <td>Fr</td>
         <td>9:15-10:45</td>
         <td>AM</td></tr>
</table>
```

Darstellung der Tabelle:

Tag	Zeit	Raum
Mo	11:00-12.30	AM
Fr	9:15-10:45	AM

Erweiterung der Notation vonKFGn:

X * auf der rechten Seite einer Produktion steht für eine **beliebig lange Folge von X**

(gleiche Bedeutung wie bei Wertebereichen)

5.2 Entity-Relationship-Modell

Entity-Relationship-Modell, ER-Modell (P. Chen 1976): Kalkül zur Modellierung von **Aufgabenbereichen mit ihren Objekten, Eigenschaften und Beziehungen**.

Weitergehende Zwecke:

- **Entwurf von Datenbanken**;
  Beschreibung der Daten, die die DB enthalten soll, „konzeptionelles Schema“
- **Entwurf von Software-Strukturen**
  Entwurfssprache UML basiert auf ER

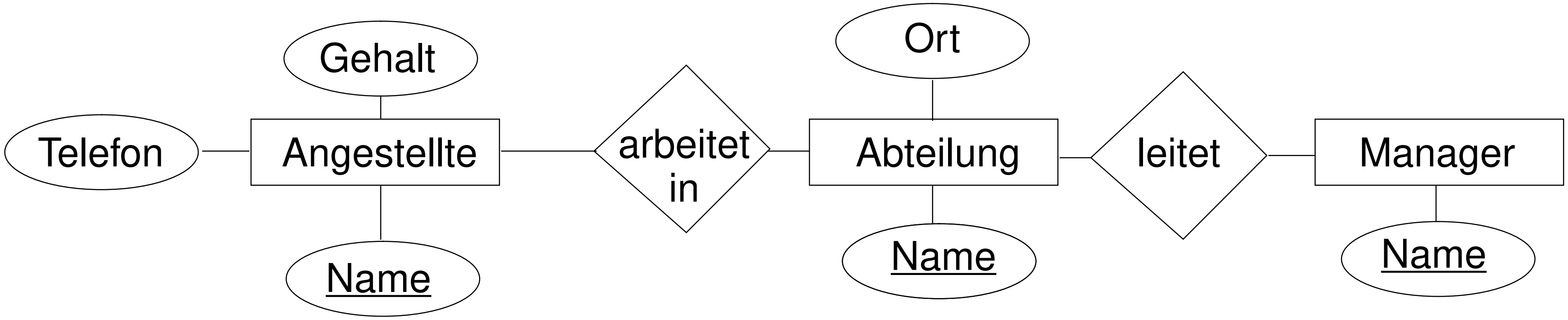
Grundbegriffe

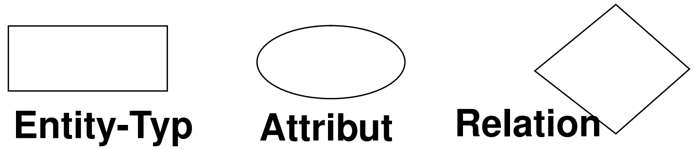
- **Entity** **Objekt** des Aufgabenbereiches
- **Relation** **Beziehung** zwischen Objekten
- **Attribut** Beschreibt ein **Eigenschaft** eines Objektes durch einen **Wert**

Graphische und textuelle **Notationen** für ER-Modellierungen; hier graphische

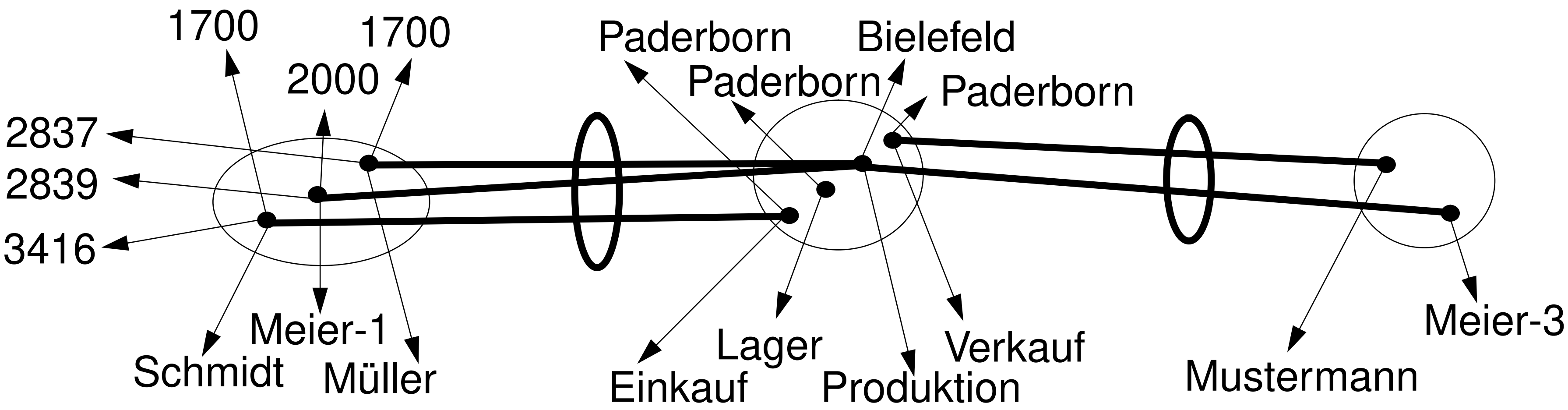
Einführendes Beispiel

Ausschnitt aus der Modellierung einer Firmenorganisation: [Beispiel nach J. D. Ullman: Principles ...]





Eine **konkrete Ausprägung** zu dem Modell:



Entities

Entity:

Objekt, Gegenstand aus dem zu modellierenden **Aufgabenbereich**
Jede Entity hat eine **eindeutige Identität**, verschieden von allen anderen

Entity-Menge (auch **Entity-Typ**):

Zusammenfassung von Objekten, die im Modell als **gleichartig** angesehen werden,
z. B. Angestellte, Abteilung, Manager

Im **Modell steht eine Entity-Menge** für die ggf. nicht-endliche Menge aller infrage kommenden Objekte dieser Art.

Eine **konkrete Ausprägung zu der Entity-Menge** ist eine endliche Teilmenge davon.

Abteilung

steht im Modell für die Menge aller in Unternehmen möglichen Abteilungen

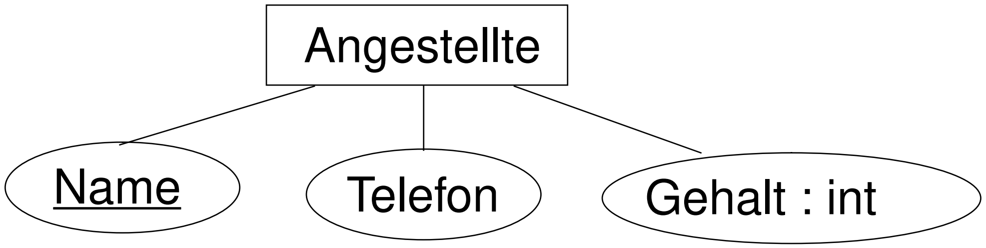
- Einkauf
- Verkauf
- Produktion
- Lager

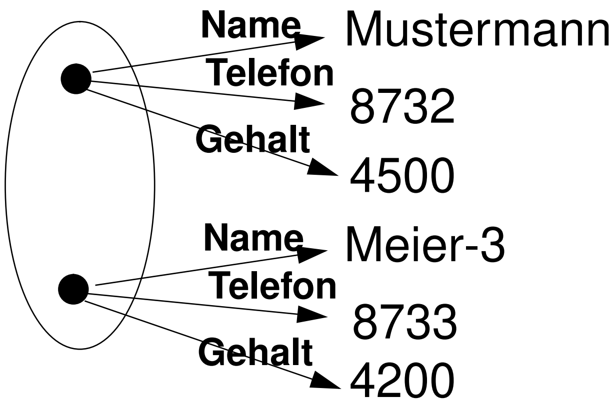
konkrete Ausprägung dazu:
die Menge der Abteilungen eines konkreten Unternehmens

Attribute

Attribute beschreiben Eigenschaften von Entities.
Einer Entity-Menge im Modell können Attribute zugeordnet werden, z. B.

eine konkrete Ausprägung:

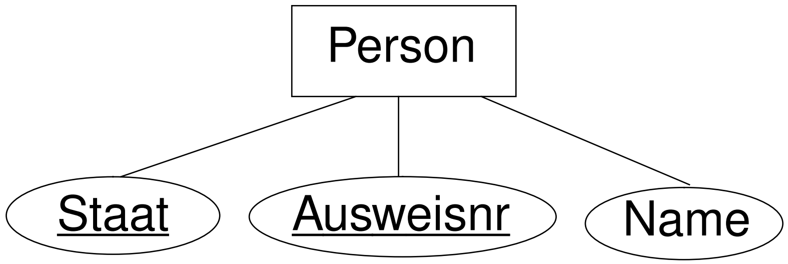




Ein Attribut ordnet jeder Entity aus der konkreten Entity-Menge einen Wert zu.
Der **Wertebereich eines Attributes** kann explizit angegeben sein, z. B. int für Gehalt, oder er wird passend angenommen.

Ein Attribut, dessen **Wert jede Entity eindeutig identifiziert**, heißt **Schlüsselattribut**.
Es wird im Modell unterstrichen.

Auch **mehrere Attribute zusammen** können den Schlüssel bilden:



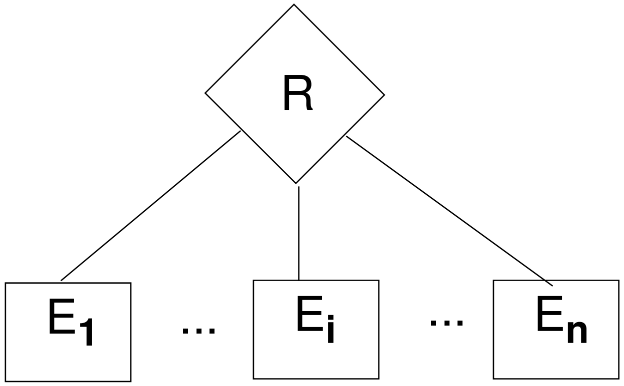
Relationen

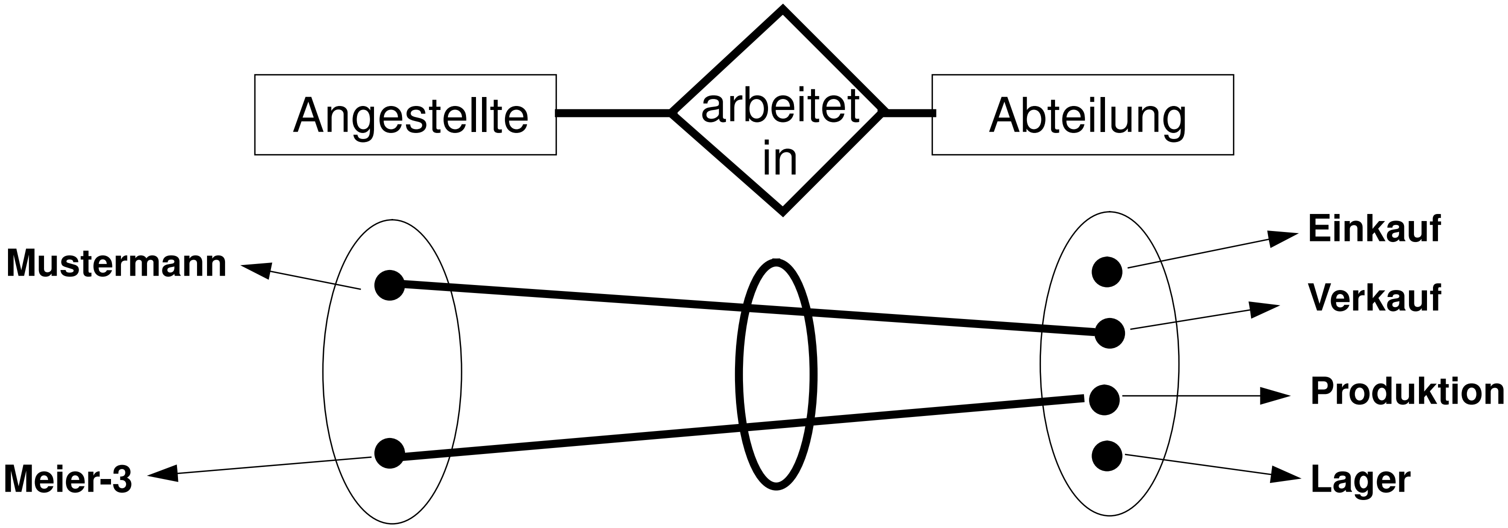
Relationen modellieren Beziehungen zwischen den Entities der Entity-Mengen.

n-stellige Relation R über n Entity-Mengen $E_1, ..., E_n$, mit $n \geq 2$:

Im Modell wird dadurch der **Typ der Relation** angegeben.

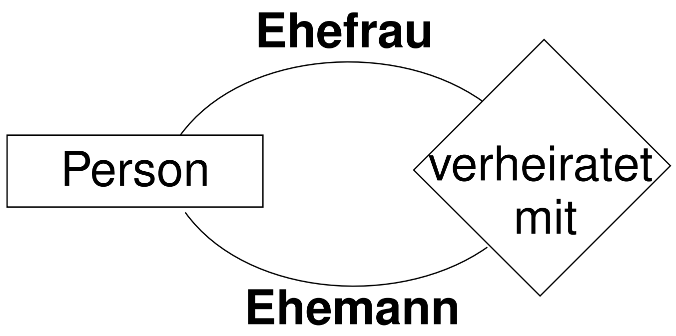
Eine **konkrete Ausprägung von R** ist eine **Menge von n-Tupeln (e_1, ... , e_n)**, wobei die e_i Entities aus den konkreten Ausprägungen der Entity-Mengen E_i sind.





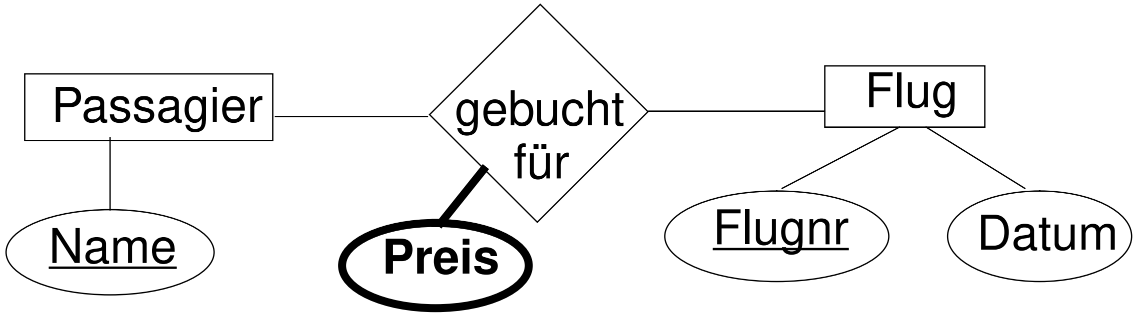
Rollen und Attribute in Relationen

Für manche Relationen wird aus ihrem Namen und der Graphik nicht klar, welche Bedeutung die Entity-Mengen in der Relation haben. Man kann das durch **Rollennamen an den Kanten** verdeutlichen.

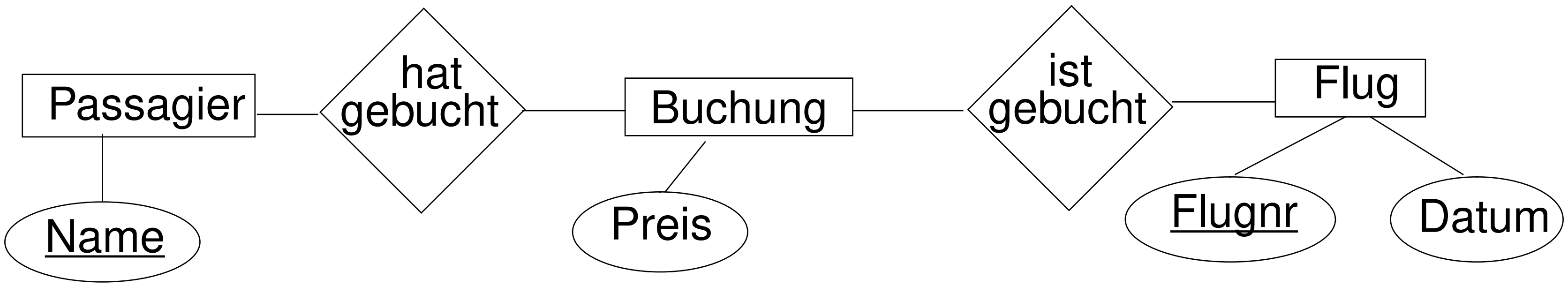


Auch **Relationen können Attribute haben**. Sie beschreiben **Eigenschaften zu jedem Tupel der Relation**.

Der Preis ist eine **Eigenschaft der Buchung** - nicht des Passagieres oder des Fluges.

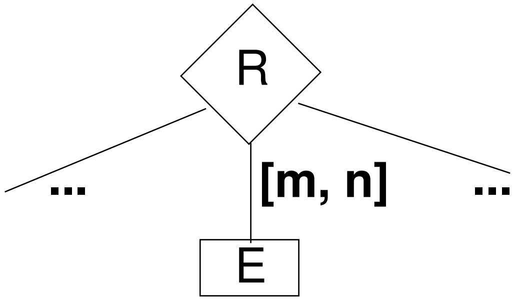


Man könnte natürlich auch **Buchungen als Entities** modellieren:



Kardinalität von Relationen

In Relationen wird duch Angaben zur **Kardinalität** bestimmt, wie oft eine Entity in den Tupeln der Relation vorkommen kann bzw. vorkommen muss:



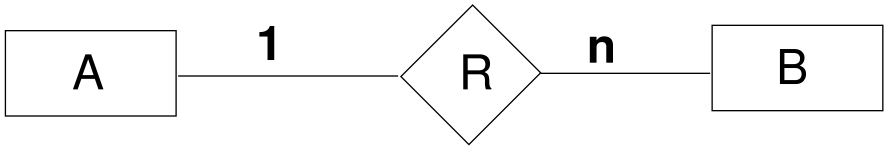
Für jede konkrete Ausprägung der Relation R muss gelten:
Jede Entity e aus der konkreten Entity-Menge zu E kommt **in mindestens m und höchstens n Tupeln** vor.

Spezielle Kardinalitäten:

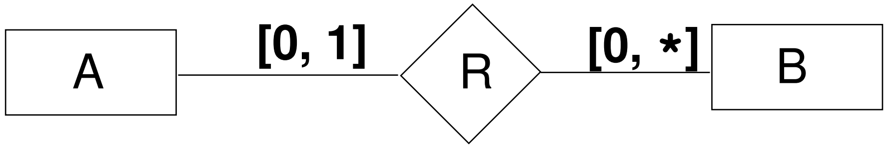
- [1, 1] in **genau einem** Tupel: totale Funktion von E auf die übrigen Rollen der Relation
- [0, 1] in **höchstens einem** Tupel: partielle Funktion von E auf die übrigen Rollen
- [0, *] in **beliebig vielen** Tupeln

Ohne Angabe wird [0, *] angenommen.

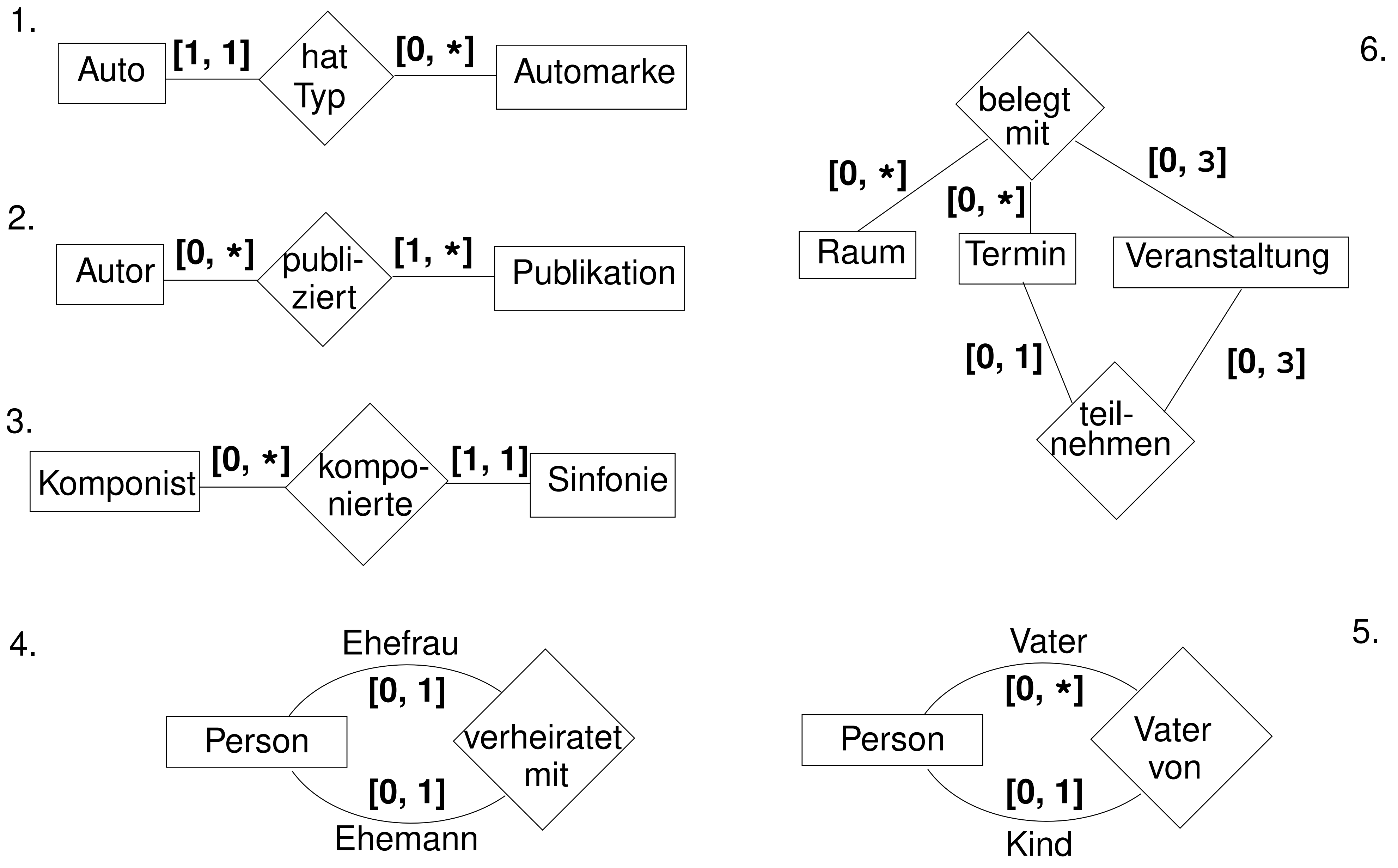
Kurznotation für 2-stellige Relationen:



bedeutet:

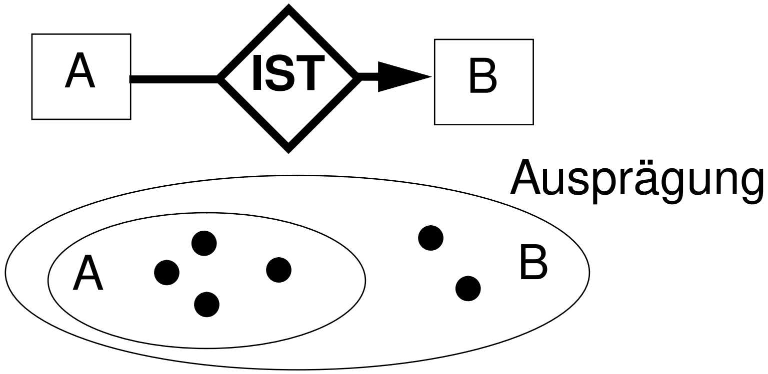


Beispiele zu Kardinalitäten in Relationen



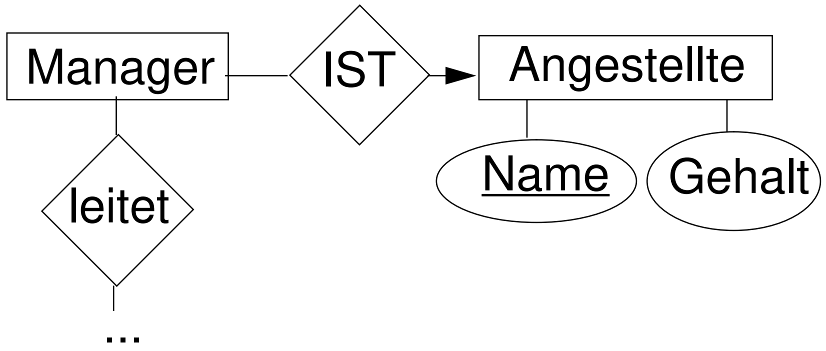
IST-Hierarchie

Die spezielle **Relation IST** (engl. is-a) definiert eine **Spezialisierungs-Hierarchie** für Entity-Mengen:



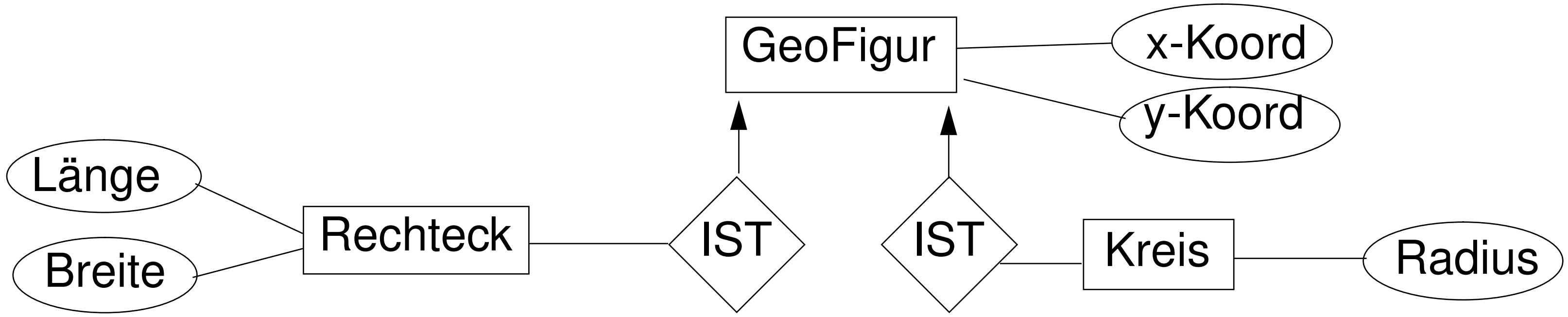
A IST B: Einige Entities der **allgemeineren Menge B** gehören auch der **spezielleren Menge A** an.

Jede konkrete Ausprägung zu A ist **Teilmenge** der konkreten Ausprägung zu B.
Es kann Entities in B geben, die nicht in A sind.

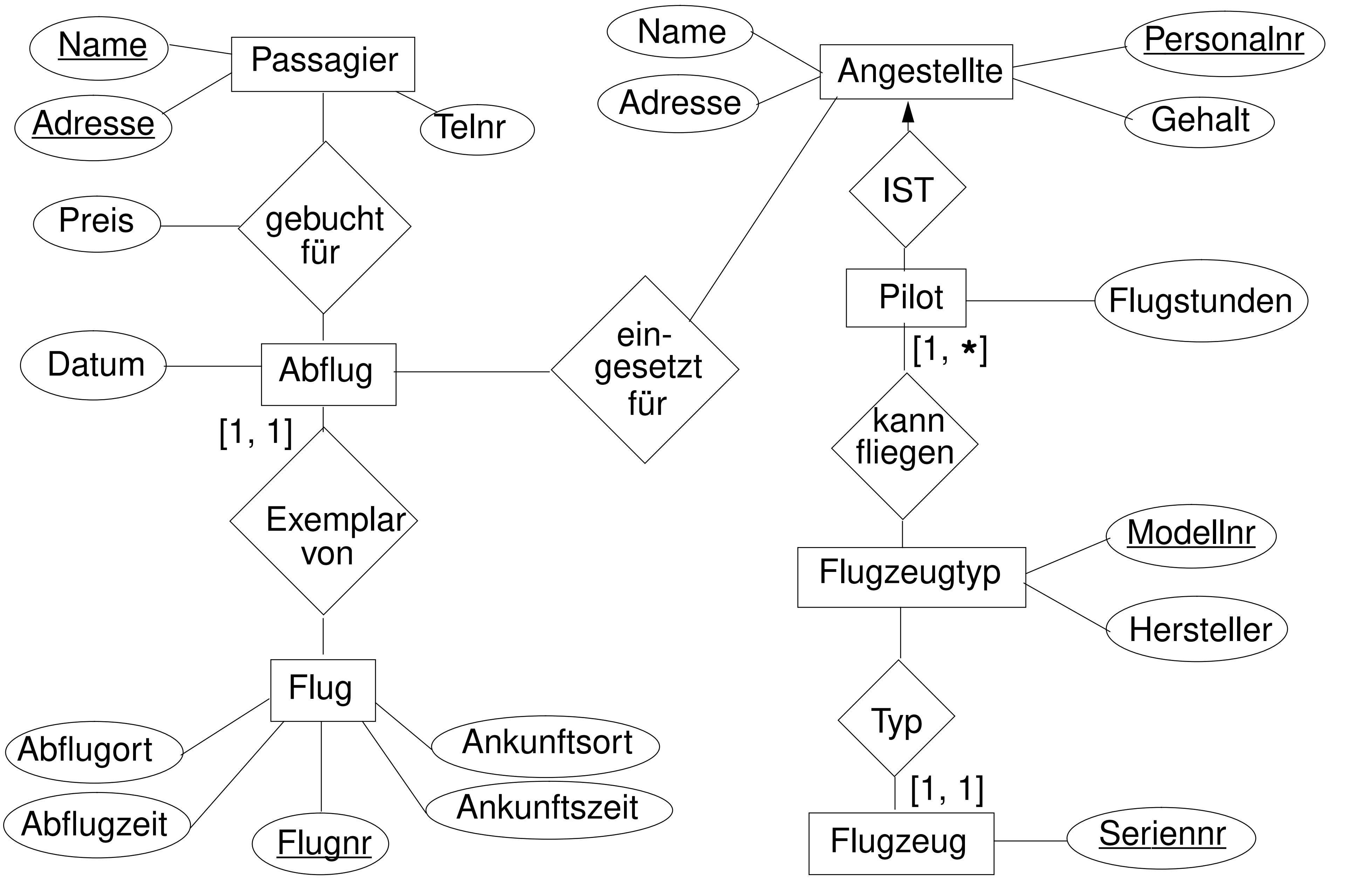


Die **Entities in A „erben" alle Attribute von B** und können noch weitere Attribute haben, die **spezielle A-Eigenschaften** beschreiben.

Auch **Schlüsselattribute** werden als solche **geerbt**.

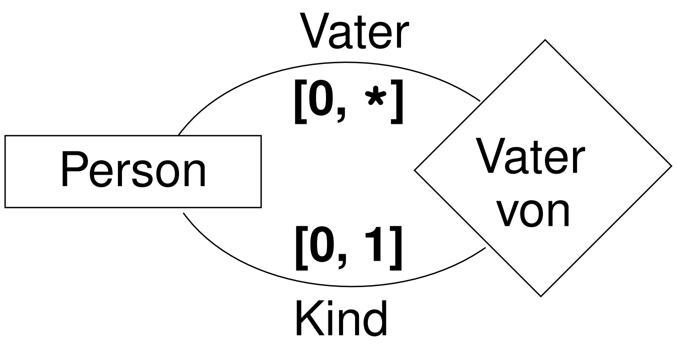


Beispiel: Fluggesellschaft

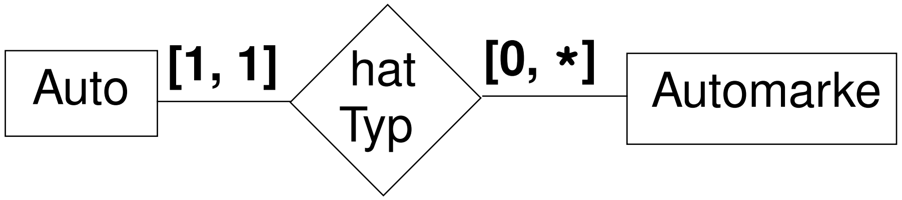


Hinweise zur Modellierung mit ER

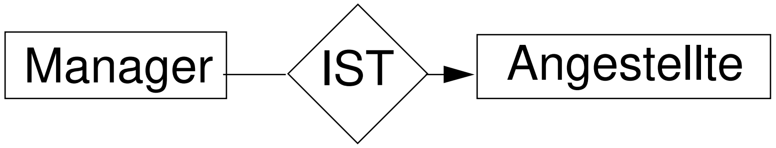
- In einem ER-Modell kommt **jede Entity-Menge nur einmal** vor.
- **Rollen** zu Relationen **angeben**, wo es nötig ist.
- Bedeutung der Kardinalitäten klarstellen.



- **Typ - Exemplar - Relationen** bewusst einsetzen.



- **Spezialisierung** sinnvoll einsetzen.



- Typ - Exemplar - Relation **nicht** mit Spezialisierung **verwechseln**

6. Modellierung von Abläufen
6.1 Endliche Automaten

Endlicher Automat:

Formaler Kalkül zur **Spezifikation von realen oder abstrakten Maschinen**. Sie

- reagieren auf **äußere Ereignisse**,
- ändern ihren **inneren Zustand**,
- produzieren ggf. **Ausgabe**.

Endliche Automaten werden **eingesetzt**, um

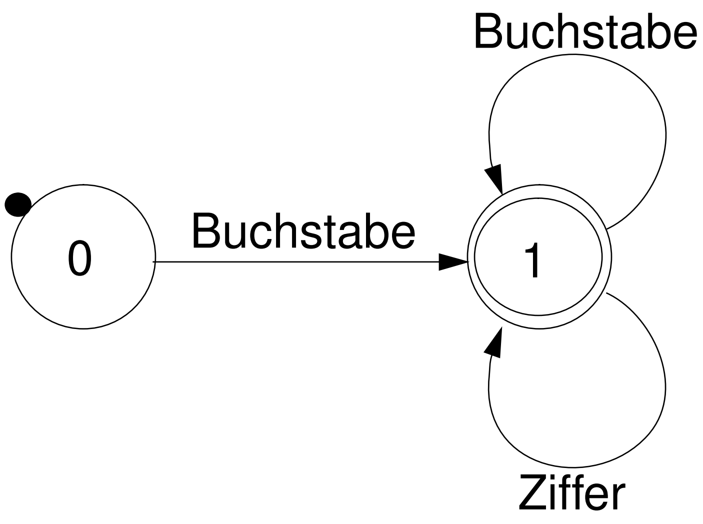
- das **Verhalten realer Maschinen** zu spezifizieren, z. B. Getränkeautomat,
- das **Verhalten von Software-Komponenten** zu spezifizieren, z. B. Reaktionen von Benutzungsoberflächen auf Bedienereignisse,
- **Sprachen zu spezifizieren**: Menge der Ereignis- oder Symbolfolgen, die der Automat akzeptiert, z. B. Schreibweise von Bezeichnern und Zahlwerten in Programmen

Zunächst definieren wir nur die **Eingabeverarbeitung** der Automaten;
das Erzeugen von **Ausgabe** fügen wir **später** hinzu.

Zwei einführende Beispiele

Endlicher Automat definiert eine **Sprache**, d. h. eine Menge von Wörtern. Ein Wort ist eine Folge von Zeichen.

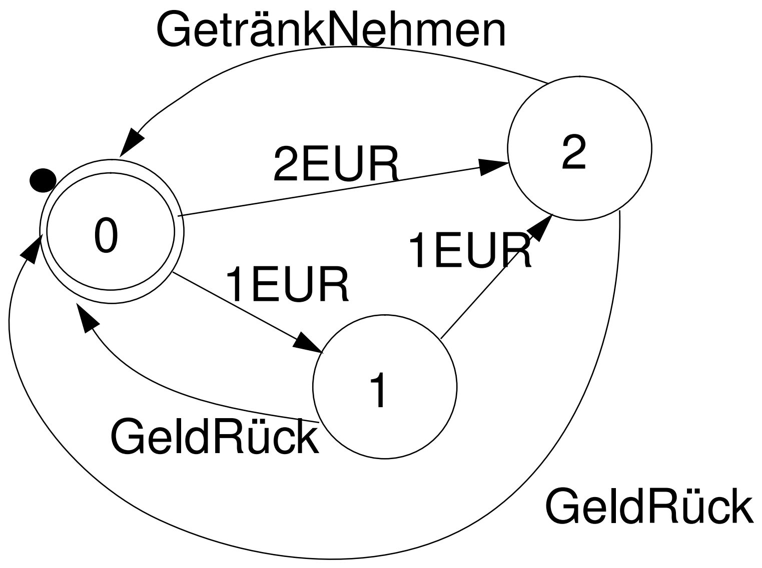
Hier: **Bezeichner** in Pascal-Programmen:



Akzeptiert Folgen von Buchstaben und Ziffern beginnend mit einem Buchstaben.

Endlicher Automat spezifiziert das **Verhalten einer Maschine**.

Hier: einfacher **Getränkeautomat**:



Akzeptiert Folgen von Ereignissen zur Bedienung eines Getränkeautomaten

Endliche Automaten können durch **gerichtete, markierte Graphen** dargestellt werden, **Ablaufgraphen**.

Alphabete

Alphabet: Eine **Menge von Zeichen** zur Bildung von Zeichenfolgen, häufig mit Σ bezeichnet.

Wir betrachten hier nur endliche Alphabete, z. B.

$\{0, 1\}$, $\{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$, $\{a, b, ..., z\}$

Ein **Wort über einem Alphabet** Σ ist eine **Zeichenfolge** aus Σ^*

statt $(a_1, a_2, ..., a_n) \in \Sigma^*$ schreiben wir $a_1\ a_2 \ldots a_n$, z. B. $10010 \in \{0, 1\}^*$

für die leere Folge schreiben wir ε (epsilon)

Reguläre Ausdrücke

Reguläre Ausdrücke beschreiben **Mengen von Worten**, die nach bestimmten Regeln aufgebaut sind. Seien F und G reguläre Ausdrücke, dann gilt

regulärer Ausdruck	**Menge von Worten**	**Erklärung**
a	$\{ a \}$	Zeichen a als Wort
ε	$\{ \varepsilon \}$	das leere Wort
F | G	$\{ f \mid f \in F \} \cup \{ g \mid g \in G \}$	Alternativen
F G	$\{ f\, g \mid f \in F, g \in G \}$	Zusammenfügen von Worten
$F^{\mathbf{n}}$	$\{ f_1\, f_2 \ldots f_n \mid \forall i \in \{1,..n\}: f_i \in F \}$	n Worte aus F
F^*	$\{ f_1\, f_2 \ldots f_n \mid n \geq 0 \text{ und } \forall i \in \{1,..n\}: f_i \in F \}$	Folgen von Worten aus F
F^+	$\{ f_1\, f_2 \ldots f_n \mid n \geq 1 \text{ und } \forall i \in \{1,..n\}: f_i \in F \}$	nicht-leere Folgen von Worten aus F
(F)	= F	Klammerung

Beispiele: $1^{\mathbf{3}}\ (1 \mid 0)^*\ 0^{\mathbf{3}}$

Bezeichner = B (B | D)*   mit B = a | b | ... | z   und D = 0 | 1 | ... | 9

Deterministischer endlicher Automat

Deterministischer endlicher Automat (engl.: deterministic finite automaton, DFA):

Quintupel $A = (\Sigma, Q, \delta, q_0, F)$ mit

Σ	endliches **Eingabealphabet**
Q	endliche **Menge von Zuständen**
δ	**Übergangsfunktion** aus $Q \times \Sigma$ -> Q
$q_0 \in Q$	**Anfangszustand**
$F \subseteq Q$	**Menge der Endzustände** (akzeptierend)

Wir nennen $r = \delta(q, a)$ **Nachfolgezustand von q unter a**.

A heißt **deterministisch, weil** es **zu jedem Paar (q, a)**, mit $q \in Q$, $a \in \Sigma$, **höchstens einen Nachfolgezustand** $\delta(q, a)$ gibt, d. h. δ ist eine **Funktion in Q.**

A heißt **vollständig, wenn** die **Übergangsfunktion** δ eine **totale** Funktion ist.

Gerichteter Graph zu endlichem Automaten

Knoten: **Zustände** des Automaten; Anfangszustand und Endzustände werden speziell markiert
Kanten: **Übergangsfunktion**, q -> r markiert mit a, genau dann wenn $\delta(q, a) = r$
Es gibt Kanten, die sich nur durch ihre Markierung unterscheiden, deshalb: **Multigraph**

Beispiele von Mod-6.2:

Σ := Menge der ASCII-Zeichen

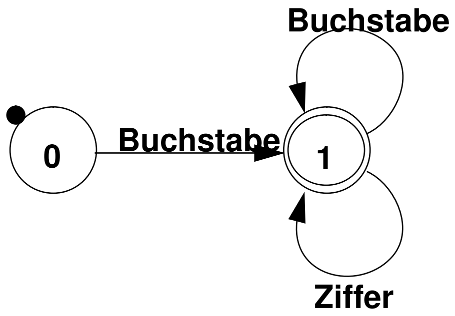
$Q := \{0, 1\}$

δ :=

	a...zA...Z	0...9	sonstige
0	1		
1	1	1	

$q_0 = 0$

$F = \{1\}$



Buchstabe, Ziffer
sind Namen reg. Ausdrücke

Σ := {**1EUR, 2EUR, GeldRück, GetränkNehmen**}

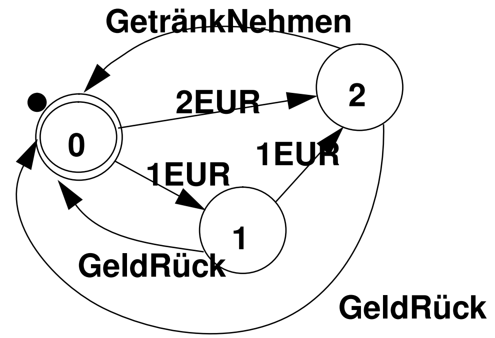
$Q := \{0, 1, 2\}$

δ :=

	1EUR	2EUR	GeldRück	GetränkNehmen
0	**1**	**2**		
1	**2**		**0**	
2			**0**	**0**

$q_0 = 0$

$F = \{0\}$



Akzeptierte Sprache

Die Zeichen einer Zeichenfolge bewirken nacheinander Zustandsübergänge in Automaten.
Zustandsübergangsfunktion erweitert für Zeichenfolgen:

Sei $\delta: Q \times \Sigma \to Q$ eine **Übergangsfunktion für Zeichen**,
dann ist $\hat{\delta}: Q \times \Sigma^* \to Q$ eine **Übergangsfunktion für Wörter**, rekursiv definiert:

- Übergang mit dem **leeren Wort**: $\hat{\delta}(q, \varepsilon) = q$ für alle $q \in Q$
- Übergang mit dem **Wort wa**: $\hat{\delta}(q, wa) = \delta(\hat{\delta}(q, w), a)$ für alle $q \in Q$, $w \in \Sigma^*$, $a \in \Sigma$

Statt $\hat{\delta}$ schreiben wir meist auch δ.

Sei $A = (\Sigma, Q, \delta, q_0, F)$ ein deterministischer endlicher Automat und $w \in \Sigma^*$.
A akzeptiert das Wort w genau dann, wenn $\delta(q_0, w) \in F$.
Die Menge $L(A) := \{ w \in \Sigma^* \mid \delta(q_0, w) \in F \}$ heißt die **von A akzeptierte Sprache**.

Beispiele für Sprachen, die von endlichen Automaten akzeptiert werden können:

$L_1 = \bigcup_{n, m \in \mathbb{N}} a^n b^m$ $\qquad L_2 = \Sigma^*$

Es gibt keinen endlichen Automaten, der $L_3 = \bigcup_{n \in \mathbb{N}} a^n b^n$ akzeptiert.

Nicht-deterministischer Automat

Nicht-deterministisch (allgemein) :
Es gibt mehrere Möglichkeiten der Entscheidung bzw. der Fortsetzung, es ist aber nicht festgelegt, welche gewählt wird.

Nicht-deterministischer endlicher Automat:
Die **Übergangsfunktion** δ kann einen Zustand q und ein Eingabezeichen a auf **mehrere Nachfolgezustände** abbilden $\delta : Q \times \Sigma$ -> Pow (Q).
Welcher gewählt wird, ist nicht festgelegt.
Σ, Q, q_0, F sind wie für deterministische endliche Automaten definiert.

Erweiterung von δ auf Zeichenfolgen:

Sei A = (Σ , Q, δ , q_0, F) ein nicht-deterministischer endlicher Automat; dann ist $\hat{\delta}$ definiert:

- Übergang mit dem **leeren Wort**: $\hat{\delta}(q, \varepsilon) = \{ q \}$ für alle $q \in Q$
- Übergang mit dem **Wort wa**: $\hat{\delta}(q, wa) = \{q' \in Q \mid \exists p \in \hat{\delta}(q, w): q' \in \delta(p, a)\}$ für alle $q \in Q$, $w \in \Sigma^*$, $a \in \Sigma$,
  d. h. **die Menge aller Zustände, die man von q mit wa erreichen kann**

Wir schreiben meist δ für $\hat{\delta}$

Ein nicht-deterministischer endlicher Automat A **akzeptiert** ein Wort w gdw. $\delta(q_0, w) \cap F \neq \emptyset$

$L(A) = \{w \in \Sigma^* \mid \delta(q_0, w) \cap F \neq \emptyset \}$ ist **die von A akzeptierte Sprache**.

Nicht-deterministische und deterministische Automaten

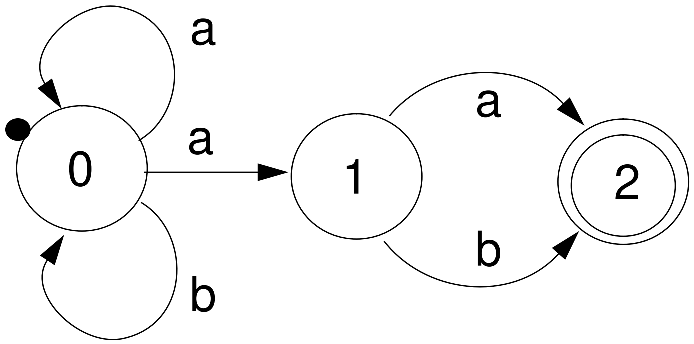
Satz: Sei L(A) die Sprache eines nicht-deterministischen Automaten.
Dann gibt es einen deterministischen Automaten, der L(A) akzeptiert.

Man kann **aus einem nicht-deterministischen Automaten A** = $(\Sigma , Q , \delta, q_0 , F)$
einen **deterministischen A'** = $(\Sigma, Q', \delta', q_0', F')$ systematisch **konstruieren**:

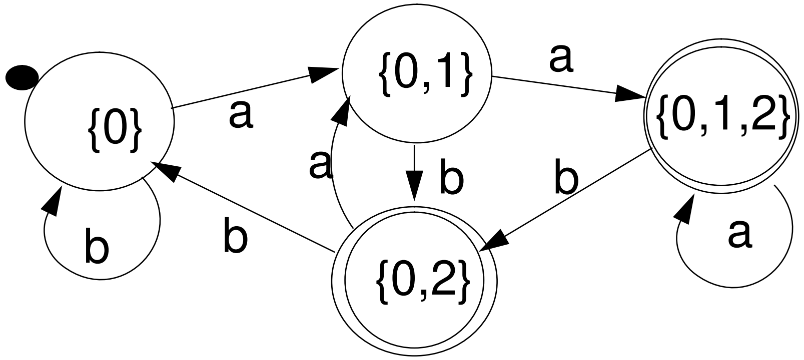
Jeder **Zustand aus Q' repräsentiert eine Menge von Zuständen aus Q, d. h. Q'** $\subseteq$ **Pow(Q)**

Beispiel:

nicht-deterministisch A



deterministisch A'



Die Zahl der Zustände kann sich dabei **exponentiell** vergrößern.

Konstruktion deterministischer Automaten

Sei A ein **nicht-deterministischer Automate A** = $(\Sigma , Q , \delta, q_0 , F)$ daraus wird ein **deterministischer Automat A'** = $(\Sigma, Q', \delta', q_0', F')$ systematisch **konstruiert**:

Jeder **Zustand aus Q' repräsentiert eine Menge von Zuständen aus Q, d. h. Q'** $\subseteq$ **Pow(Q)**

Konstruktionsschritte:

1. **Anfangszustand**: $q_0' = \{q_0\}$
2. Wähle einen schon konstruierten Zustand $q' \in Q'$
   wähle ein Zeichen $a \in \Sigma$

   berechne $r' = \delta'(q', a) = \bigcup_{\mathbf{q} \in \mathbf{q'}} \delta(q, a)$

   d. h. r' repräsentiert die Vereinigung aller Zustände, die in A von q unter a erreicht werden.
   r' wird **Zustand in Q'** und $\delta'(q', a) = r'$ wird **Übergang in** δ'.
3. **Wiederhole (2) bis keine neuen Zustände oder Übergänge** mehr konstruiert werden können.
4. **Endzustände**: $F' = \{q' \in Q' \mid q' \cap F \neq \emptyset\}$
   d. h. q' ist Endzustand, wenn seine Zustandsmenge einen Endzustand von A enthält.

Endliche Automaten mit Ausgabe

Man kann mit endlichen Automaten auch **Reaktionen der modellierten Maschine** spezifizieren: **Automaten mit Ausgabe**.

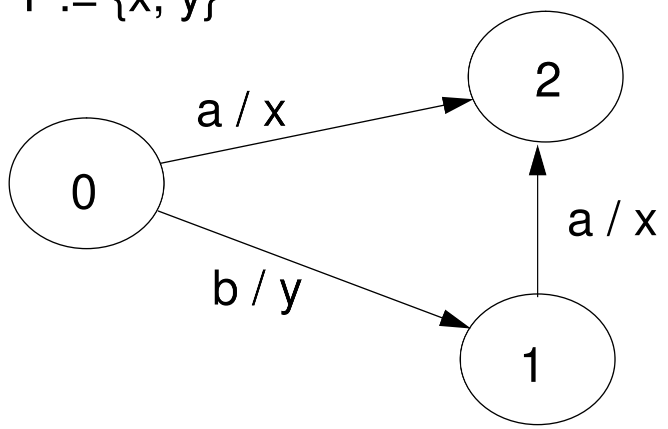
Wir erweitern den Automaten um ein **endliches Ausgabealphabet T** und um eine Ausgabefunktion. Es gibt 2 Varianten für die Ausgabefunktion:

Mealy-Automat:

Eine Ausgabefunktion $\lambda : Q \times \Sigma \rightarrow T^*$ ordnet den **Zustandsübergängen** jeweils ein **Wort über dem Ausgabealphabet** zu.

Graphische Notation:

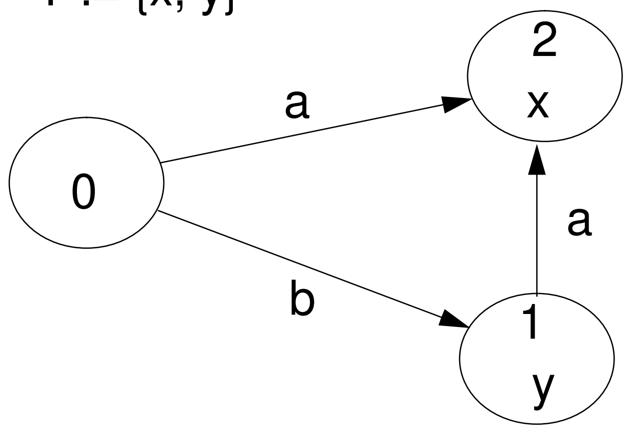
T := {x, y}



Moore-Automat:

Eine Ausgabefunktion $\mu : Q \rightarrow T^*$ ordnet den **Zuständen** jeweils ein **Wort über dem Ausgabealphabet** zu. Es wird bei Erreichen des Zustands ausgegeben.

T := {x, y}

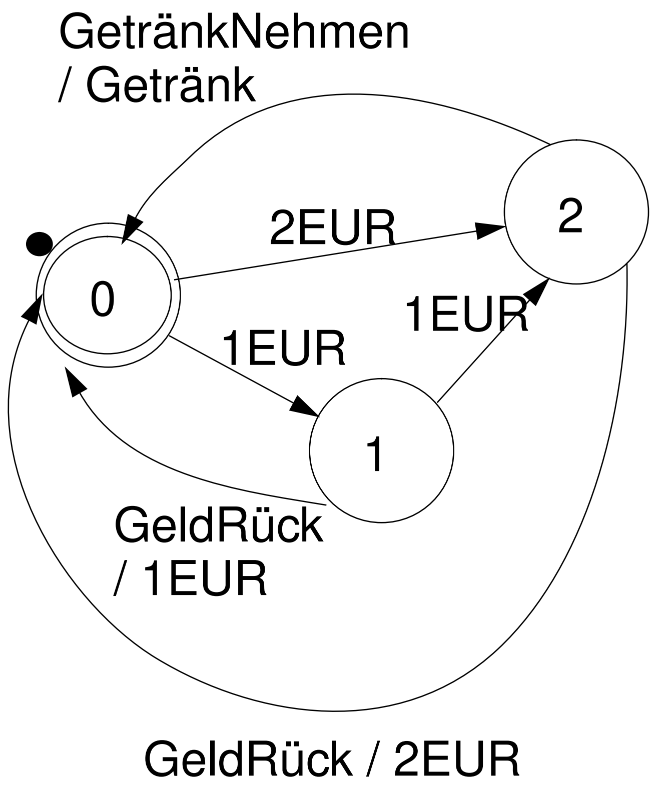


Ein **Mealy-Automat kann die Ausgabe feiner differenzieren** als ein Moore-Automat.

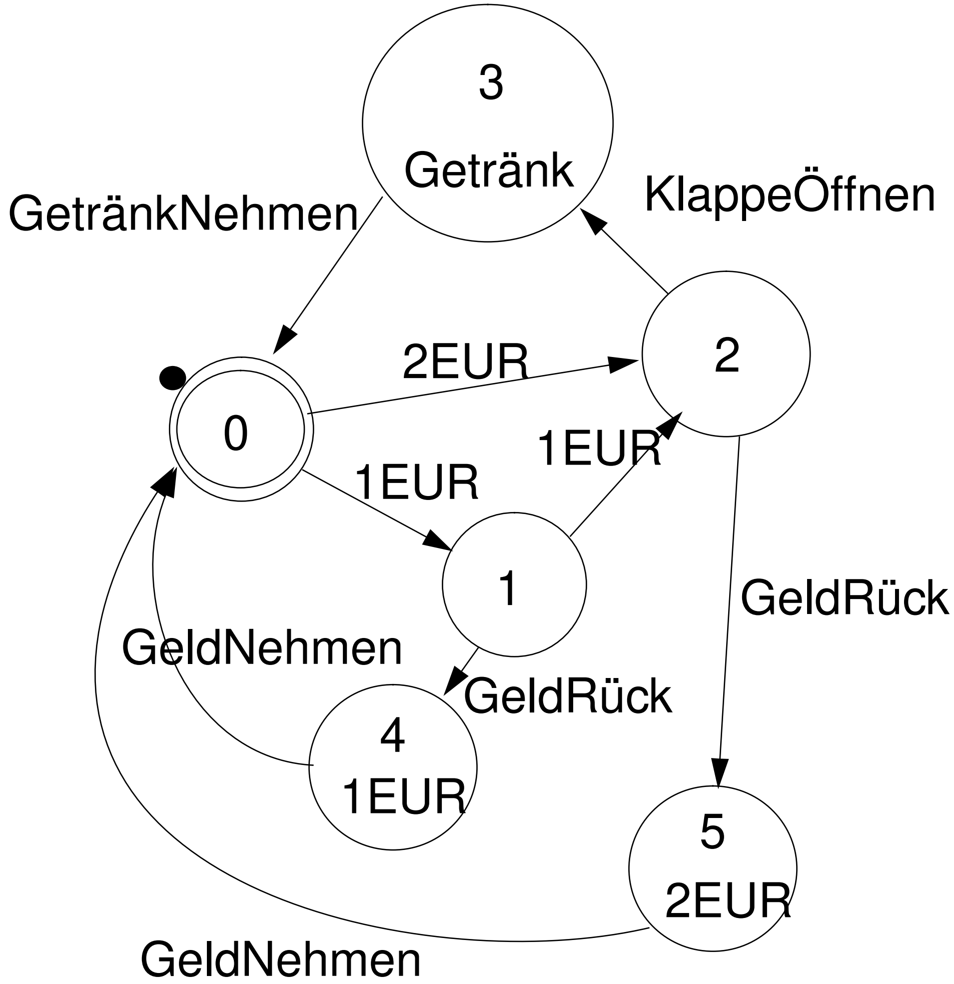
Beispiele für endliche Automaten mit Ausgabe

Die Spezifikation des Getränkeautomaten aus Mod-6.2 wird mit Ausgabe versehen:

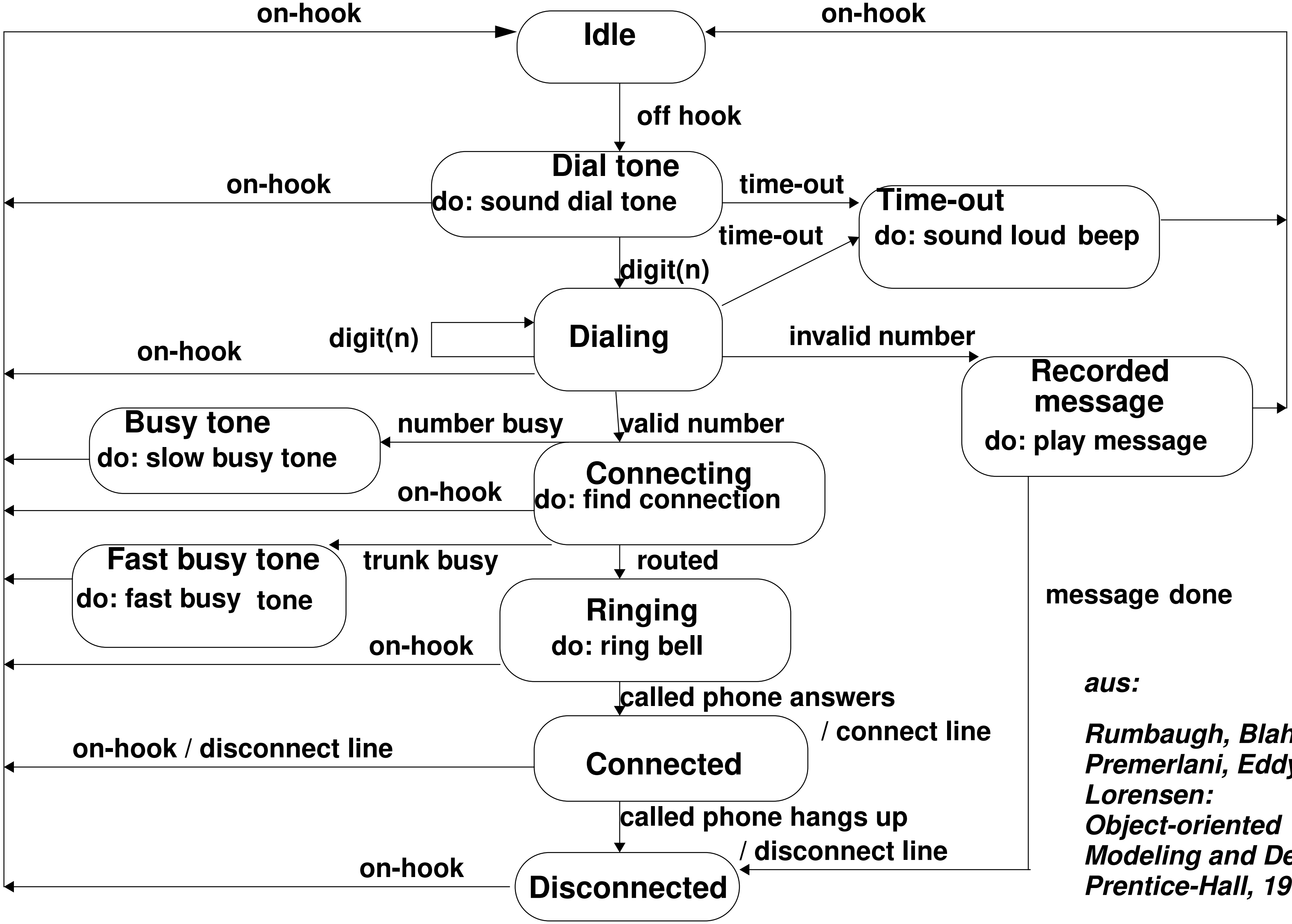
Mealy-Automat



Moore-Automat



Endlicher Automat zur Telefonbedienung



aus:

Rumbaugh, Blaha, Premerlani, Eddy, Lorensen: Object-oriented Modeling and Design, Prentice-Hall, 1991

6.2 Petri-Netze

Petri-Netz (auch Stellen-/Transitions-Netz):
Formaler Kalkül zur **Modellierung von Abläufen mit nebenläufigen Prozessen** und kausalen Beziehungen

Basiert auf **bipartiten gerichteten Graphen**:

- **Knoten** repräsentieren **Bedingungen**, Zustände bzw. **Aktivitäten**.
- **Kanten** verbinden **Aktivitäten** mit ihren **Vor- und Nachbedingungen**.
- **Knotenmarkierung** repräsentiert den veränderlichen **Zustand des Systems**.
- **graphische Notation**.

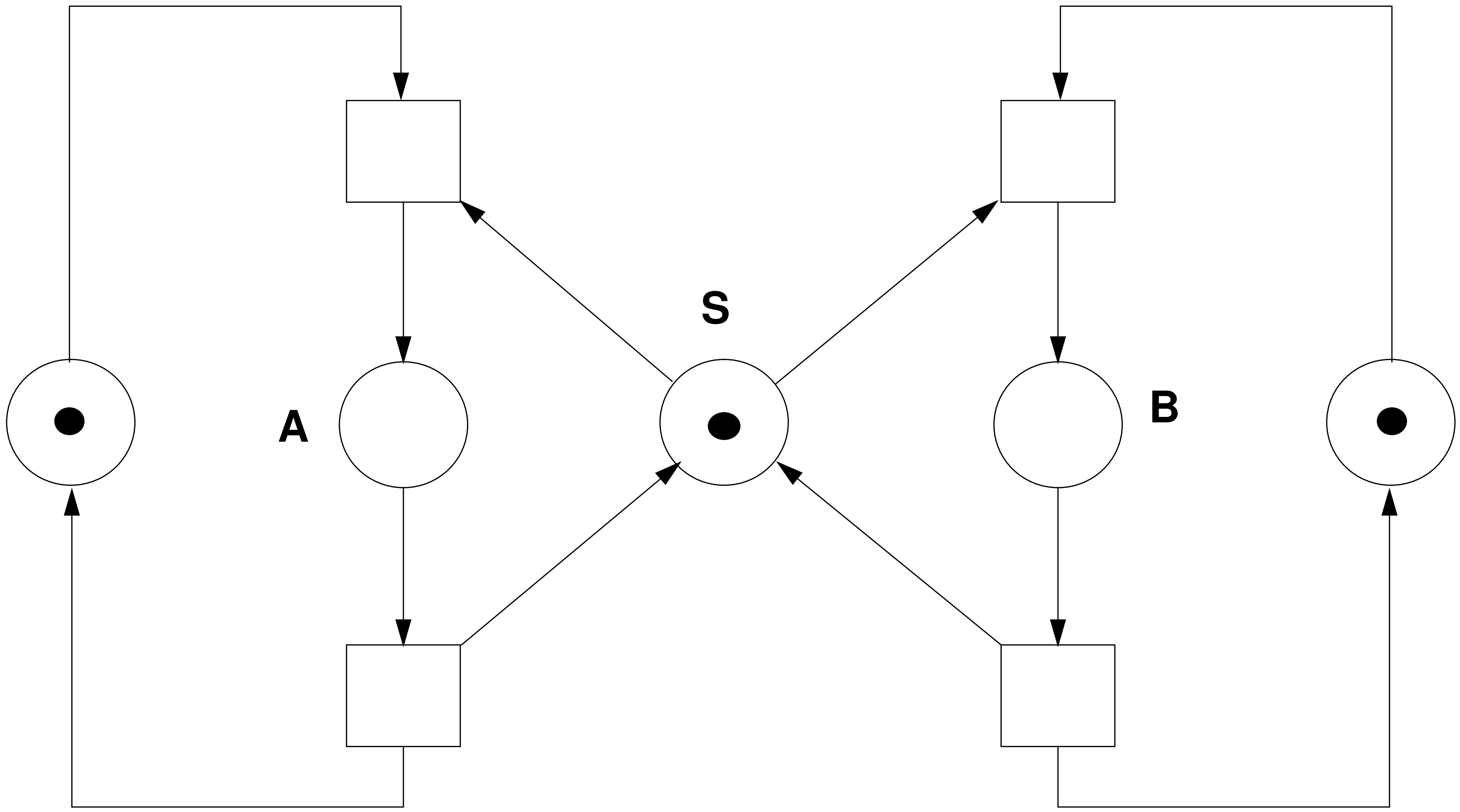
C. A. Petri hat sie 1962 eingeführt.
Es gibt zahlreiche Varianten und Verfeinerungen von Petri-Netzen. Hier nur die Grundform.

Anwendungen von Petri-Netzen zur Modellierung von

- realen oder abstrakten Automaten und Maschinen
- kommunizierenden Prozessen in der Realität oder in Rechnern
- Verhalten von Hardware-Komponenten
- Geschäftsabläufe
- Spielpläne

Einführendes Beispiel

Das Petri-Netz modelliert zwei **zyklisch ablaufende Prozesse**.
Die mittlere Stelle synchronisiert die beiden Prozesse,
so dass sie sich **nicht zugleich in den Zuständen A und B** befinden können.
Prinzip: **gegenseitiger Ausschluss** durch **Semaphor**



Definition von Petri-Netzen

Ein **Petri-Netz** ist ein Tripel P = (S, T, F) mit

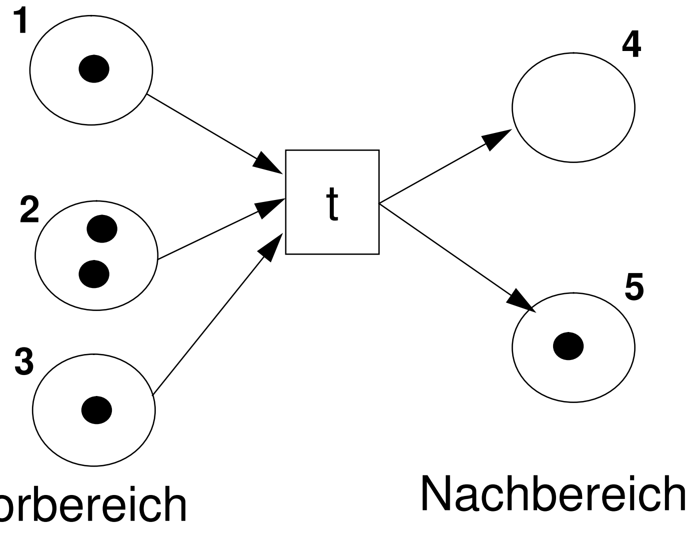
S **Menge von Stellen**,
repräsentieren Bedingungen, Zustände; graphisch Kreise

T **Menge von Transitionen** oder Übergänge,
repräsentieren Aktivitäten; graphisch Rechtecke

F **Relation** mit $F \subseteq S \times T \cup T \times S$
repräsentieren kausale oder zeitliche Vor-, Nachbedingungen von Aktivitäten aus T

P bildet einen **bipartiten, gerichteten Graphen** mit den Knoten $S \cup T$ und den Kanten F.

Zu einer **Transition t** in einem Petri-Netz P sind folgende Stellenmengen definiert
Vorbereich (t) $:= \{ s \mid (s, t) \in F\}$
Nachbereich (t) $:= \{ s \mid (t, s) \in F\}$



Der **Zustand des Petri-Netzes** wird durch eine **Markierungsfunktion** angegeben, die jeder Stelle eine **Anzahl von Marken** zuordnet:
$M_P: S \to \mathbb{N}_0$
Sind die stellen von 1 bis n nummeriert, so kann man M_P als Folge angeben, z. B. (1, 2, 1, 0, 1)

Schaltregel für Petri-Netze

Das **Schalten einer Transition** t überführt eine Markierung M in eine Markierung M‘.

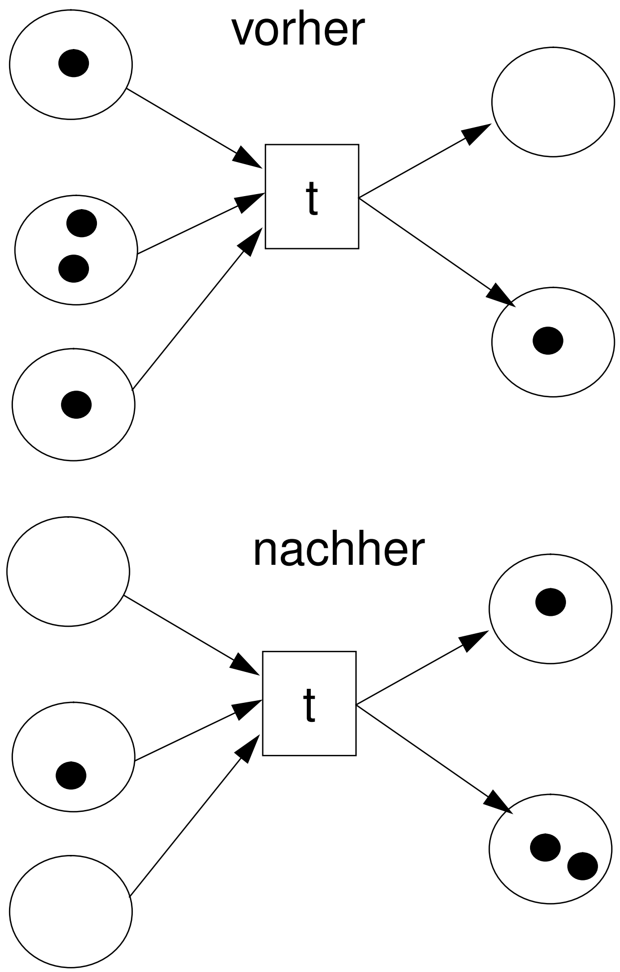
Eine **Transition t kann schalten**, wenn für alle Stellen $s \in$ Vorbereich (t) gilt $M(s) \geq 1$.

Wenn eine Transition t **schaltet,** gilt für die **Nachfolgemarkierung M‘**:

$M'(v) = M(v) - 1$ für alle $v \in$ Vorbereich(t) \ Nachbereich(t)

$M'(n) = M(n) + 1$ für alle $n \in$ Nachbereich(t) \ Vorbereich(t)

$M'(s) = M(s)$ sonst



Wenn in einem Schritt **mehrere Transitionen schalten können**, wird eine davon **nicht-deterministisch ausgewählt**.

In jedem Schritt schaltet genau eine Transition
- auch wenn das Petri-Netz parallele Abläufe modelliert!

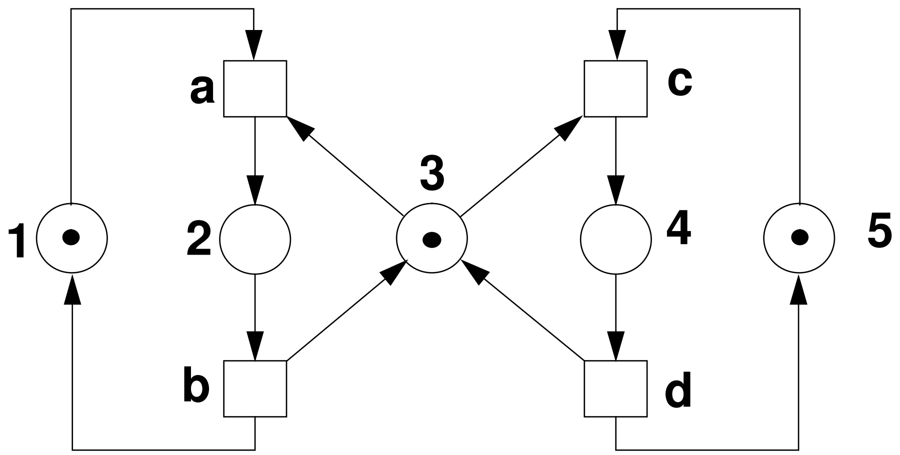
Zwei Transitionen mit gemeinsamen Stellen im Vorbereich können (bei passender Markierung) im **Konflikt** stehen:
Jede kann schalten, aber nicht beide nacheinander.

Markierungen

Zu jedem Petri-Netz wird eine **Anfangsmarkierung M_0** angeben.

z. B. $\mathbf{M_0} = (1, 0, 1, 0, 1)$

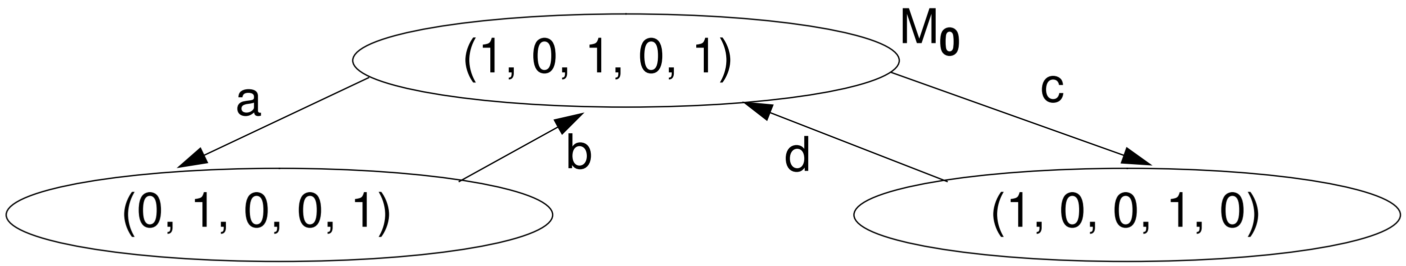
Wir sagen, eine **Markierung M_2 ist von einer Markierung M_1 aus erreichbar**, wenn es ausgehend von M_1 eine Folge von Transitionen gibt, die nacheinander schalten und M_1 in M_2 überführen können.



Die Markierungen eines Petri-Netzes kann man als gerichteten **Markierungsgraphen** darstellen:

- Knoten: erreichbare Markierung
- Kante a->b: Die Markierung a kann durch Schalten einer Transition in b übergehen.

Markierungsgraph



Schaltfolgen

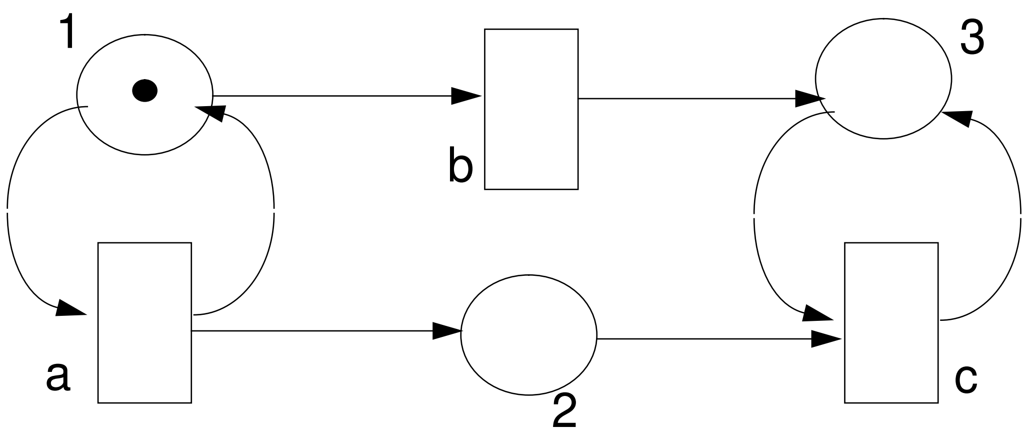
Schaltfolgen kann man angeben als

- Folge von Markierungen
- Folge der geschalteten Transitionen

Beispiel für eine **Schaltfolge** zum Petri-Netz auf Mod-6.15a:

(1, 0, 1, 0, 1)	a
(0, 1, 0, 0, 1)	b
(1, 0, 1, 0, 1)	c
(1, 0, 0, 1, 0)	d
(1, 0, 1, 0, 1)	

Schaltfolgen können als Wörter einer Sprache aufgefasst werden.



alle Schaltfolgen ohne Nachfolgemarkierung haben die Form:

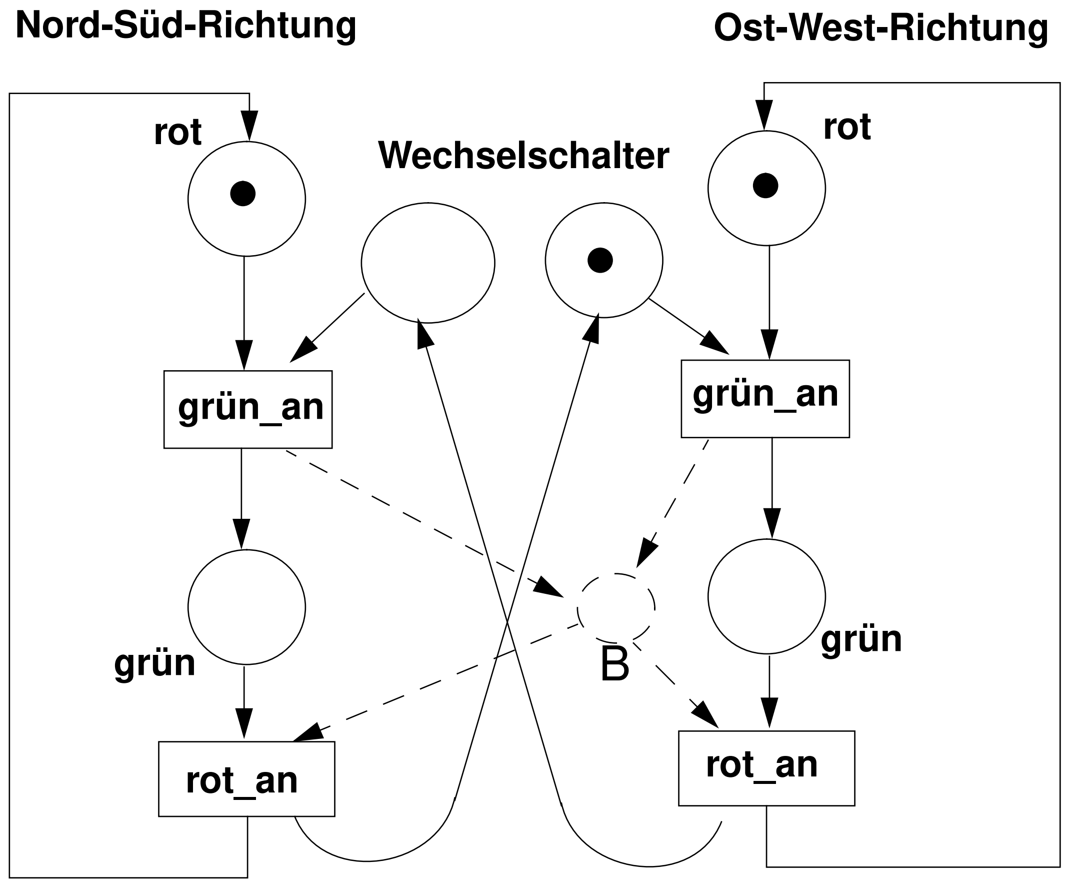
$a^{\mathbf{n}}\ b\ c^{\mathbf{n}}$

Petri-Netze können unbegrenzt zählen: Anzahl der Marken auf einer Stelle.

Modellierung alternierender zyklischer Prozesse

Beispiel: Einfache Modellierung einer Ampelkreuzung:

- 2 sich zyklisch wiederholende Prozesse
- Die beiden Stellen „Wechselschalter“ koppeln die Prozesse, sodass sie alternierend fortschreiten.
- Alle Stellen repräsentieren Bedingungen: 1 oder 0 Marken
- „Beobachtungsstelle“ B modelliert, wieviele Richtungen „grün“ haben

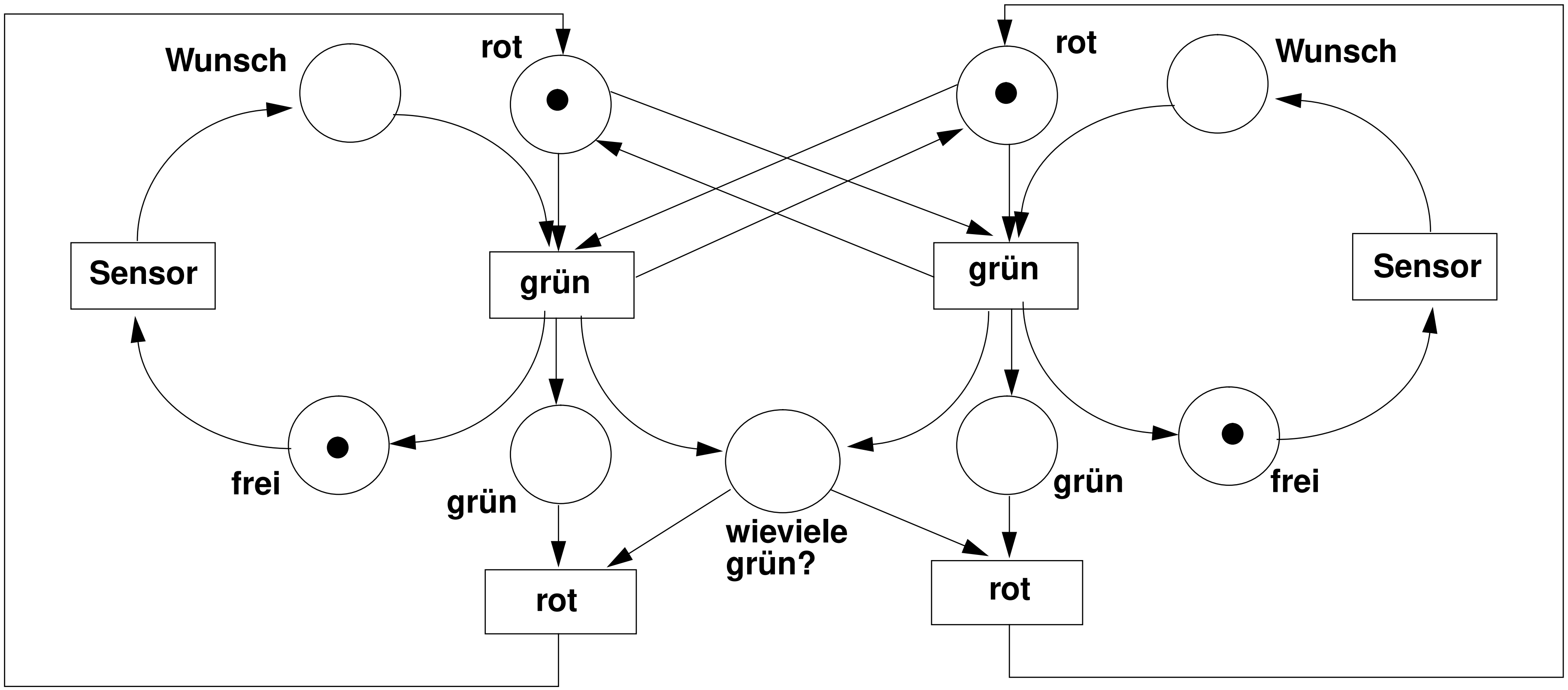


Beispiel für ein binäres Netz

Ein Petri-Netz heißt **binär (sicher)**, wenn für alle aus M_0 erreichbaren Markierungen M und für alle Stellen s gilt $M(s) \leq 1$.

Petri-Netze, deren **Stellen Bedingungen repräsentieren** müssen binär sein.

Beispiel: Modellierung einer Sensor-gesteuerten Ampelkreuzung:



aus: B. Baumgarten: Petri-Netze, Bibliographisches Institut & F. A. Brockhaus AG, 1990

Lebendige Petri-Netze

Petri-Netze modellieren häufig **Systeme, die nicht anhalten** sollen.

Ein Petri-Netz heißt **schwach lebendig**, wenn es zu jeder von M_0 erreichbaren Markierung eine Nachfolgemarkierung gibt.

Eine **Transition t heißt lebendig**, wenn es zu jeder von M_0 erreichbaren Markierung M' eine Markierung M" gibt, die von M' erreichbar ist, und in der t schalten kann.

Ein **Petri-Netz heißt lebendig**, wenn alle seine Transitionen lebendig sind.

Verklemmung: Ein System kann unerwünscht anhalten,
weil das **Schalten einiger Transitionen zyklisch voneinander abhängt**.

Sei: $\sigma \subseteq S$ eine Teilmenge der Stellen eines Petri-Netzes und

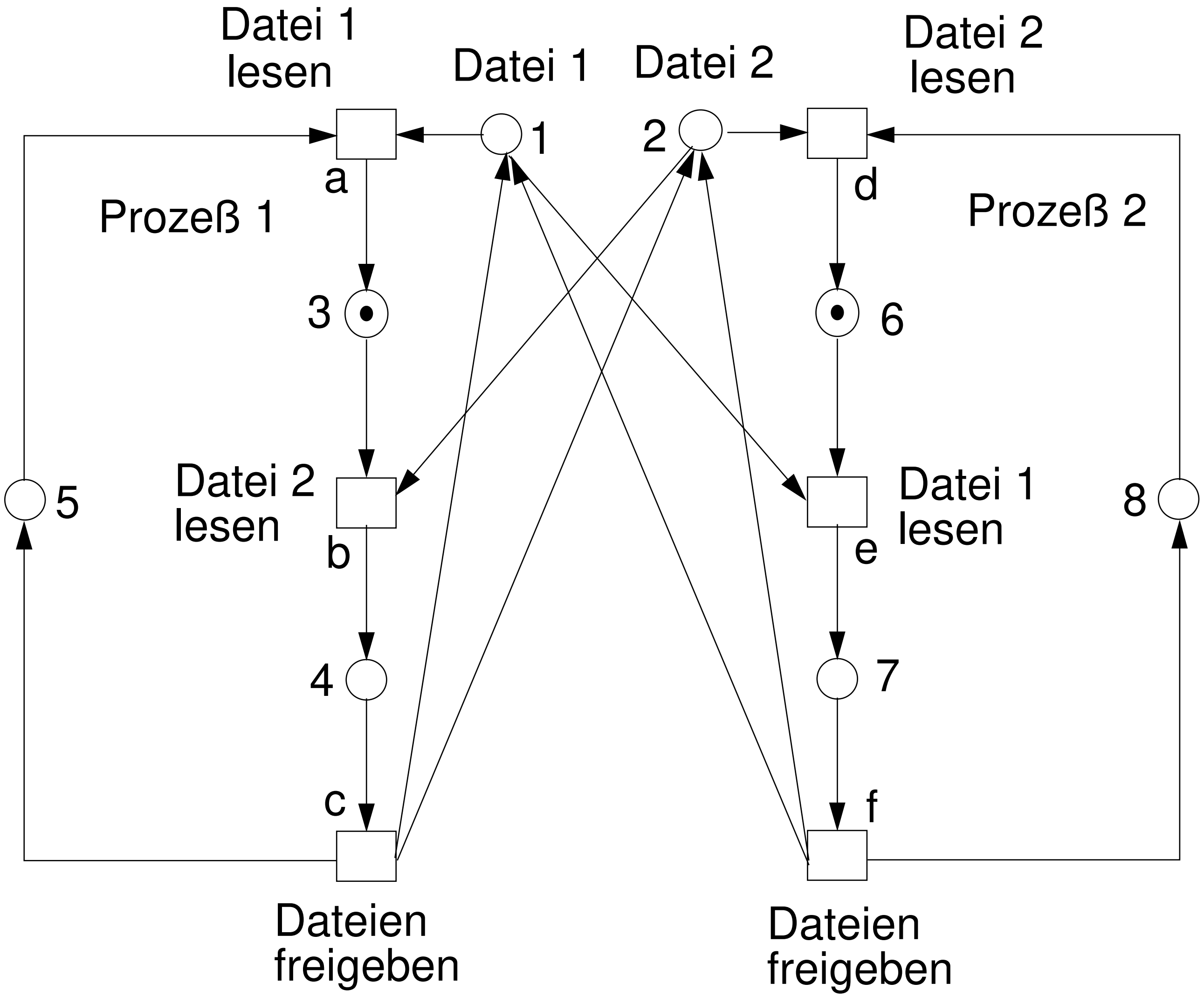
Vorbereich (σ) $:= \{t \mid \exists\, s \in \sigma : (t, s) \in F\}$,
d. h. die Transitionen, die auf Stellen in σ wirken

Nachbereich (σ) $:= \{t \mid \exists\, s \in \sigma : (s, t) \in F\}$,
d. h. die Transitionen, die Stellen in σ als Vorbedingung haben

Dann ist σ eine **Verklemmung, wenn Vorbereich (**σ) $\subseteq$ **Nachbereich** (σ).

Wenn **für alle s** $\in \sigma$ **gilt M (s) = 0**, dann kann es **keine Marken auf Stellen in** σ in einer Nachfolgemarkierung von M geben.

Verklemmung beim Lesen von Dateien



$\sigma = \{1, 2, 4, 5, 7, 8\}$

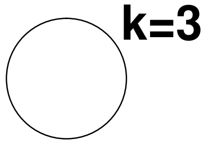
Vorbereich (σ)
$= \{b, c, e, f\}$

Nachbereich (σ)
$= \{a, b, c, d, e, f\}$

$M(\sigma) = 0$

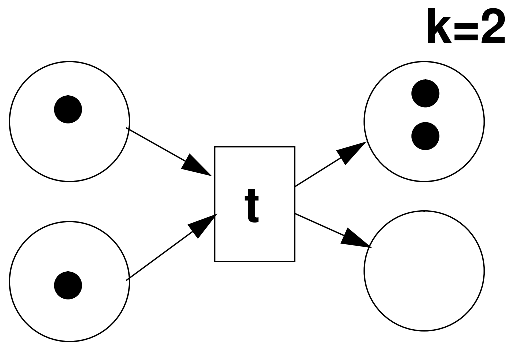
Kapazitäten und Gewichte

Man kann **Stellen eine begrenzte Kapazität** von $k \in \mathbb{N}$ Marken zuordnen.



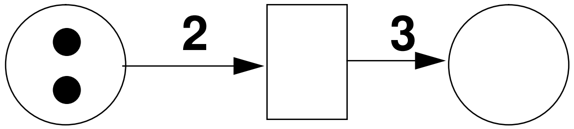
Die Bedingung, dass eine **Transition t schalten kann**, wird erweitert um:

> Die **Kapazität keiner der Stellen im Nachbereich von t darf überschritten** werden.

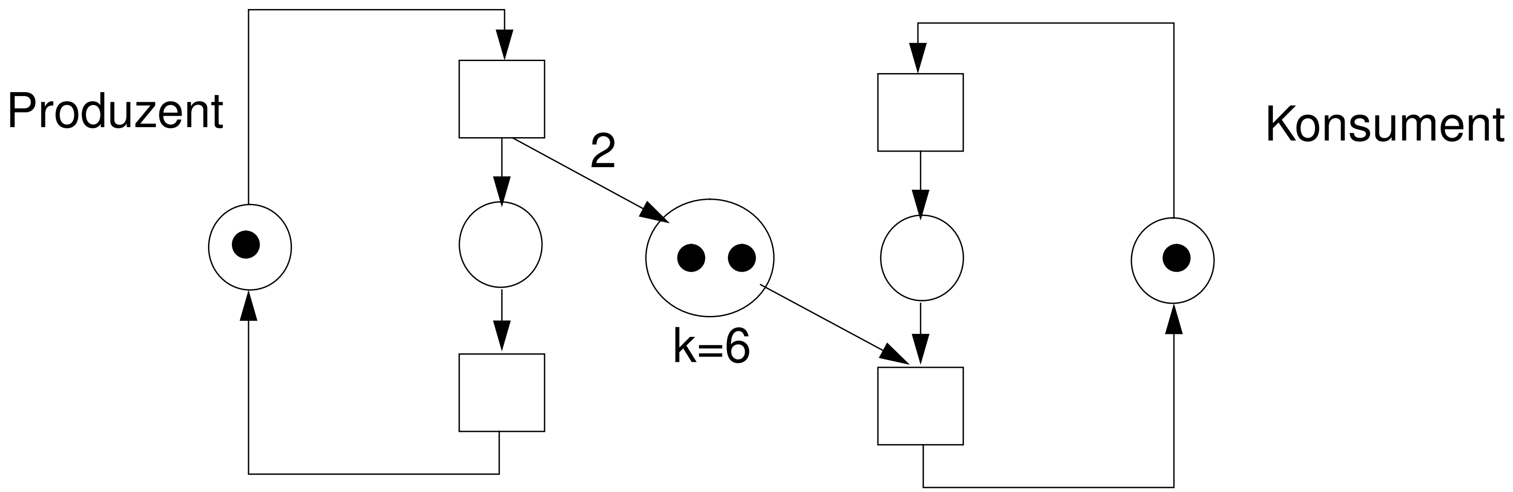


t kann nicht schalten

Kanten kann ein **Gewicht** $n \in \mathbb{N}$ zugeordnet werden: sie bewegen **beim Schalten n Marken**.



Beispiel: **Beschränkter Puffer**



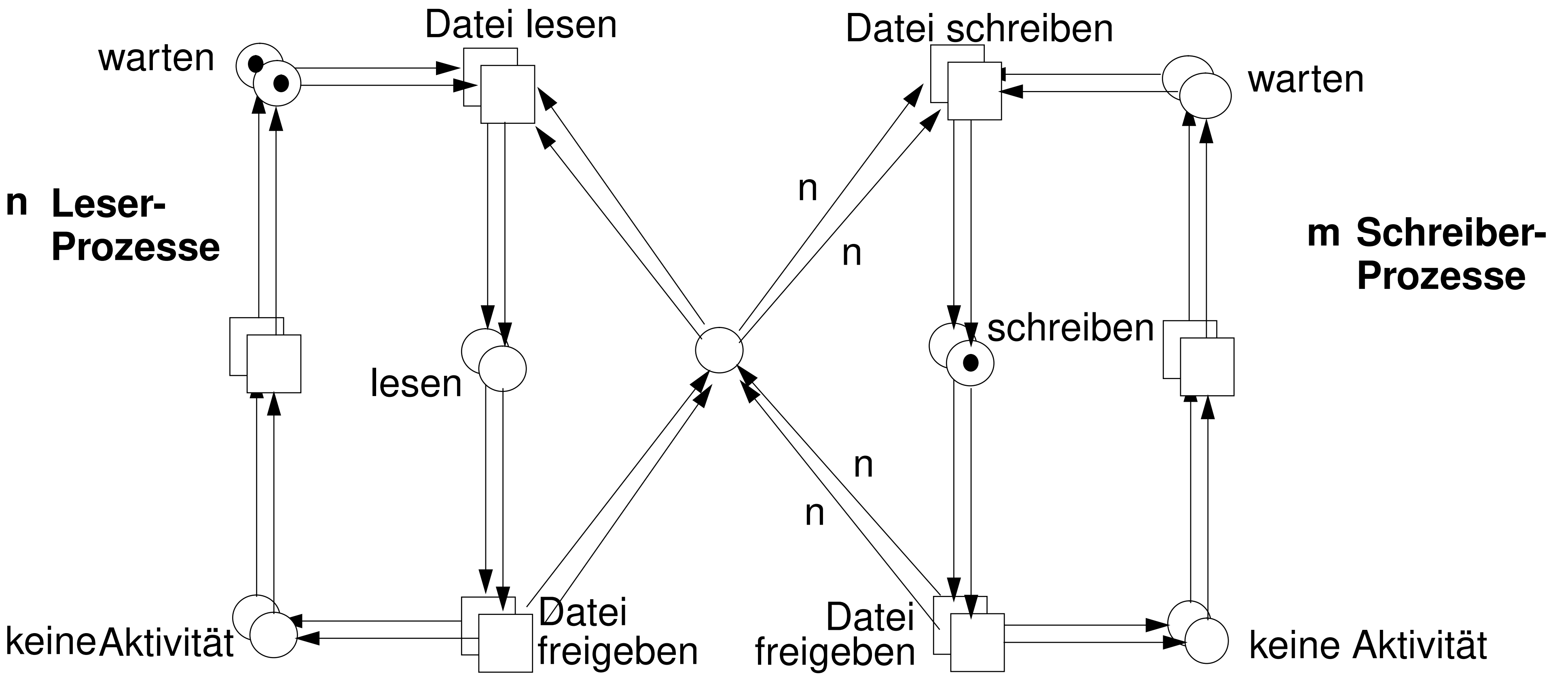
Beispiel: Leser-Schreiber-System

n Leser-Prozesse und **m Schreiber-Prozesse** operieren **auf derselben Datei**.

Mehrere **Leser können zugleich lesen**.

Ein **Schreiber** darf nur dann schreiben, wenn **kein anderer Leser oder Schreiber aktiv** ist.

Modellierung: ein **Schreiber entzieht der Synchronisationsstelle alle n Marken**.



7. Fallstudien

Jeweils **ein Gegenstandsbereich** steht im Vordergrund

Seine Strukturen, Eigenschaften, Zusammenhänge werden mit **verschiedenen Kalkülen** modelliert.

Verschiedene Kalküle werden eingesetzt, um

- **unterschiedliche Aspekte** zu beschreiben
- Beschreibungen derselben Aspekte zu **vergleichen**.

Fallstudie 1:	Autowerkstatt
Fallstudie 2:	Monopoly - Spiel
Fallstudie 3:	Getränkeautomat (siehe Übungen)

Fallstudie 1: Autowerkstatt

Wir modellieren die **Auftragsabwicklung in einer Autowerkstatt**.

Ziel: Datenbank entwerfen, Abläufe analysieren und verbessern

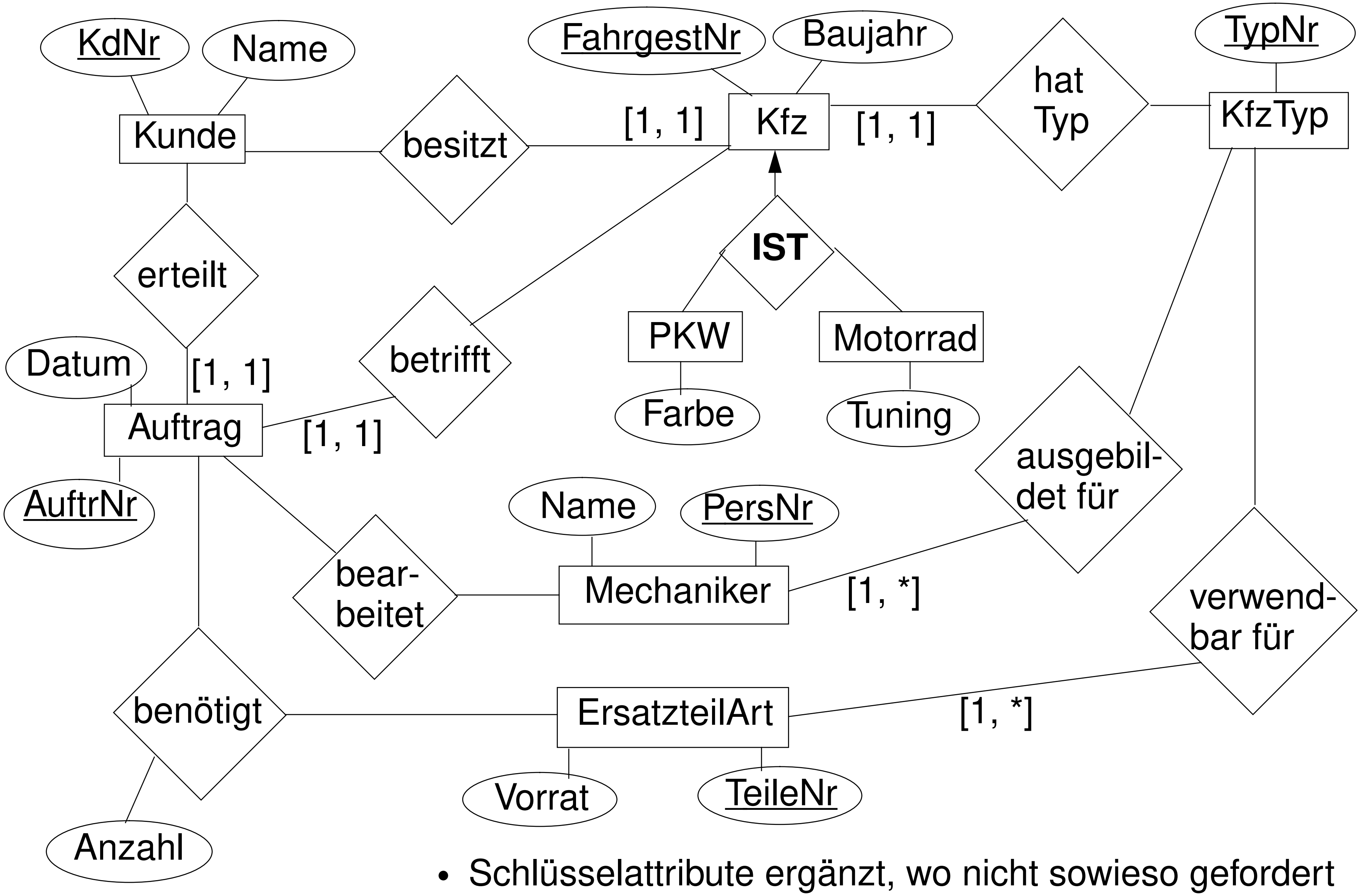
Teilaufgaben:

1. **Informationen und Zusammenhänge**
2. **Bedingungen und Regeln**
3. **Abläufe bei der Auftragsabwicklung**

Kurzbeschreibung der Informationsstruktur:

1. **Kunde**: hat einen Namen, besitzt Kraftfahrzeuge, erteilt Aufträge
2. **Auftrag**: hat Eingangsdatum, betrifft ein Kraftfahrzeug,
   wird von Mechanikern bearbeitet,
   benötigt Ersatzteile bestimmter Arten und Mengen
3. **Kraftfahrzeug**: hat Fahrgestellnummer und Baujahr,
   ist entweder ein PKW oder ein Motorrad;
   zu PKWs interessiert ihre Farbe, zu Motorrädern der Tuningsatz
4. **Typ**: Kraftfahrzeug hat einen Typ,
   Mechaniker ist für einige Typen ausgebildet,
   Ersatzteil ist für bestimmte Typen verwendbar

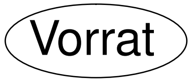
Informationsstruktur als ER-Modell



- Schlüsselattribute ergänzt, wo nicht sowieso gefordert
- Kardinalitäten: plausibel ohne unnötig einzuschränken

Vergleich ER-Modell und Wertebereiche

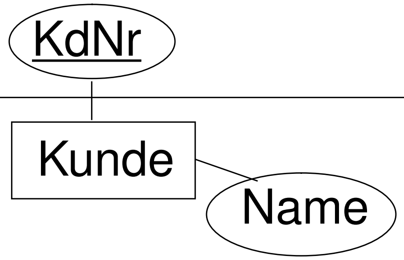
Attribut



Vorrat := $\mathbb{N}_0$

Wertemenge

Schlüsselattribut



KdNr := $\mathbb{N}_0$

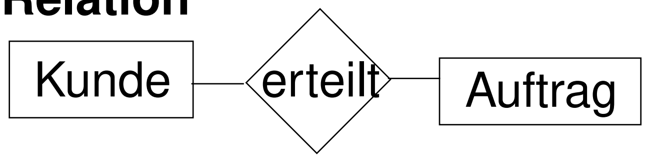
Indexmenge

Entity-Typ

Kunde := KdNr $\times$ Name

kartesisches Produkt
ohne Identität der Entities

Relation



erteilt := Pow (Kunde $\times$ Auftrag)

Relation

Kardinalität [1, 3]

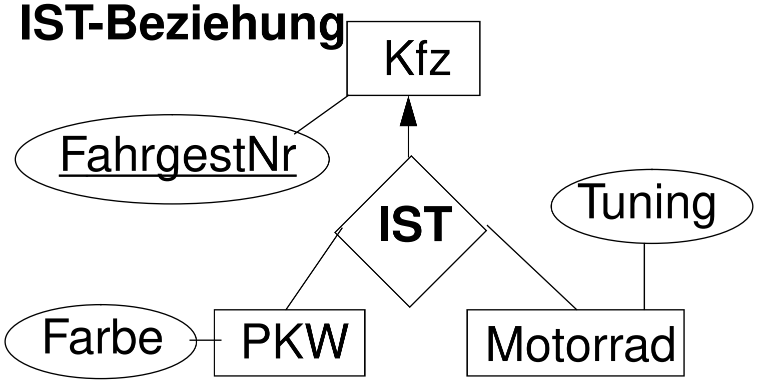
Prädikatenlogik:
$\forall$ a $\in$ Auftrag: 1 $\leq$ |{ (k, a) | (k, a) $\in$ erteilt }| $\leq$ 3

[1, 1]

erteilt: Auftrag -> Kunde

Funktion (hier: total)

IST-Beziehung



kartesisches Produkt und disjunkte Vereinigung:

Kfz := FahrgestNr $\times$ KfzVarianten

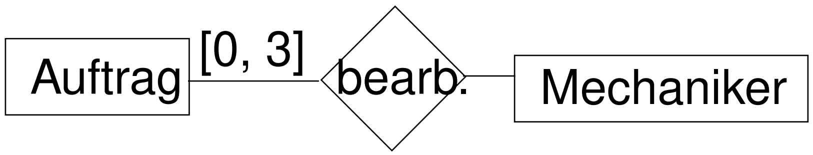
KfzArten := { istPKW, istMotorrad }

KfzVarianten := { (istPKW, p) | p $\in$ Farbe } $\cup$
{ (istMotorrad, m) | m $\in$ Tuning}

1.b Bedingungen

Ein Auftrag soll von höchstens 3 Mechanikern bearbeitet werden:

ER Kardinalität:



Prädikatenlogik: $\forall\ a \in \text{Auftrag}: 1 \leq |\{ (a, m) \mid (a, m) \in \text{bearb.} \}| \leq 3$

Ein Auftrag soll nur dann angenommen werden, wenn für den betreffenden KfzTyp auch Mechaniker ausgebildet sind.

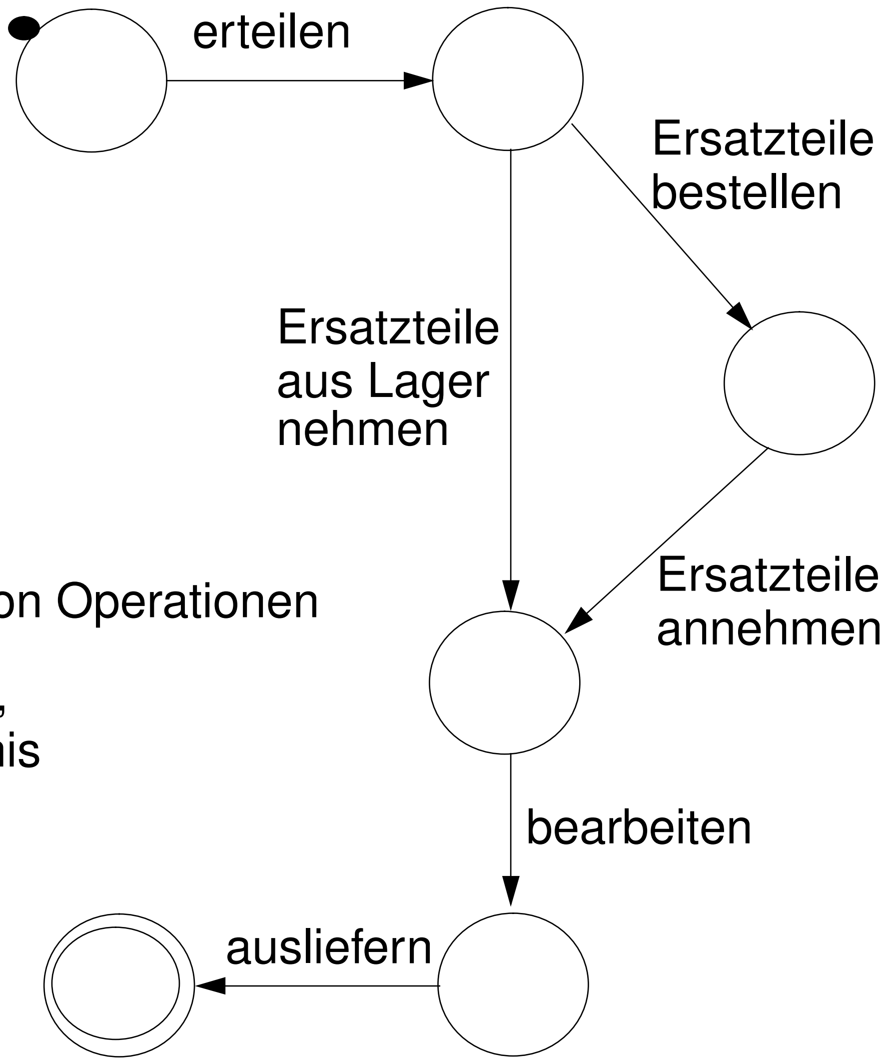
Prädikatenlogik: $\forall\ a \in \text{Auftrag}: \forall\ k \in \text{Kfz}: \forall\ t \in \text{KfzTyp}:$

$$((a, k) \in \text{betrifft} \wedge (k, t) \in \text{hatTyp}) \rightarrow$$

$$\exists\ m \in \text{Mechaniker}: (m, t) \in \text{ausgebildet}$$

1.c Ablauf der Auftragsbearbeitung (DEA)

- Auftrag wird **erteilt**,
- Verfügbarkeit der Ersatzteile **geprüft**,
- ggf. **bestellt**,
- von einem Mechaniker **bearbeitet**,
- Kraftfahrzeug wird dem Kunden **ausgeliefert**.

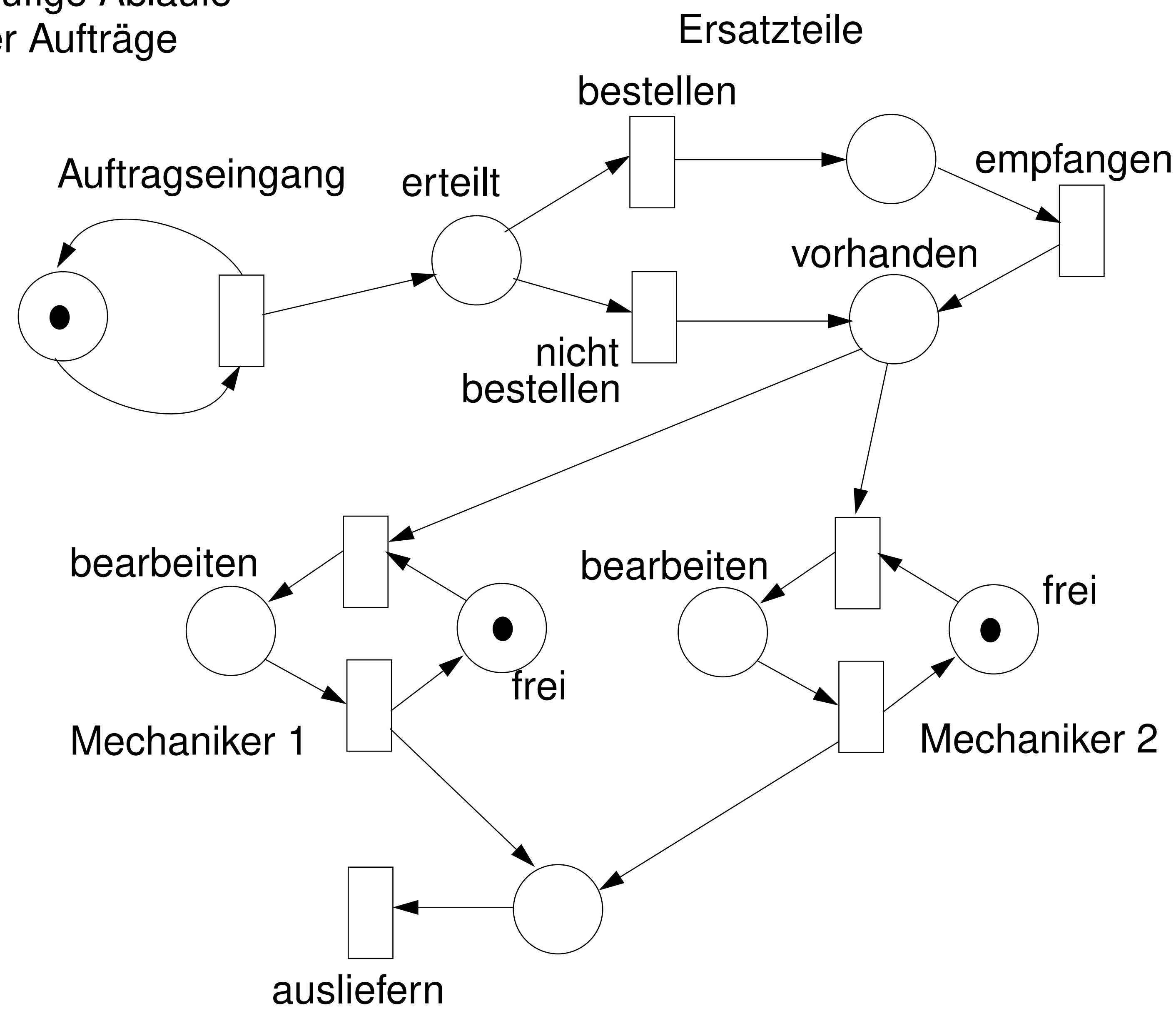


Deterministischer, endlicher Automat
beschreibt streng **sequentielle Abfolge** von Operationen

Auch als **Abhängigkeitsgraph** interpretierbar,
hier: Kante ist Operation, Knoten ist Ereignis

1.c Ablauf der Auftragsbearbeitung (Petri-Netz)

Petri-Netz
modelliert nebenläufige Abläufe
Durchlauf mehrerer Aufträge



Mechaniker
konkurrieren um
Aufträge:

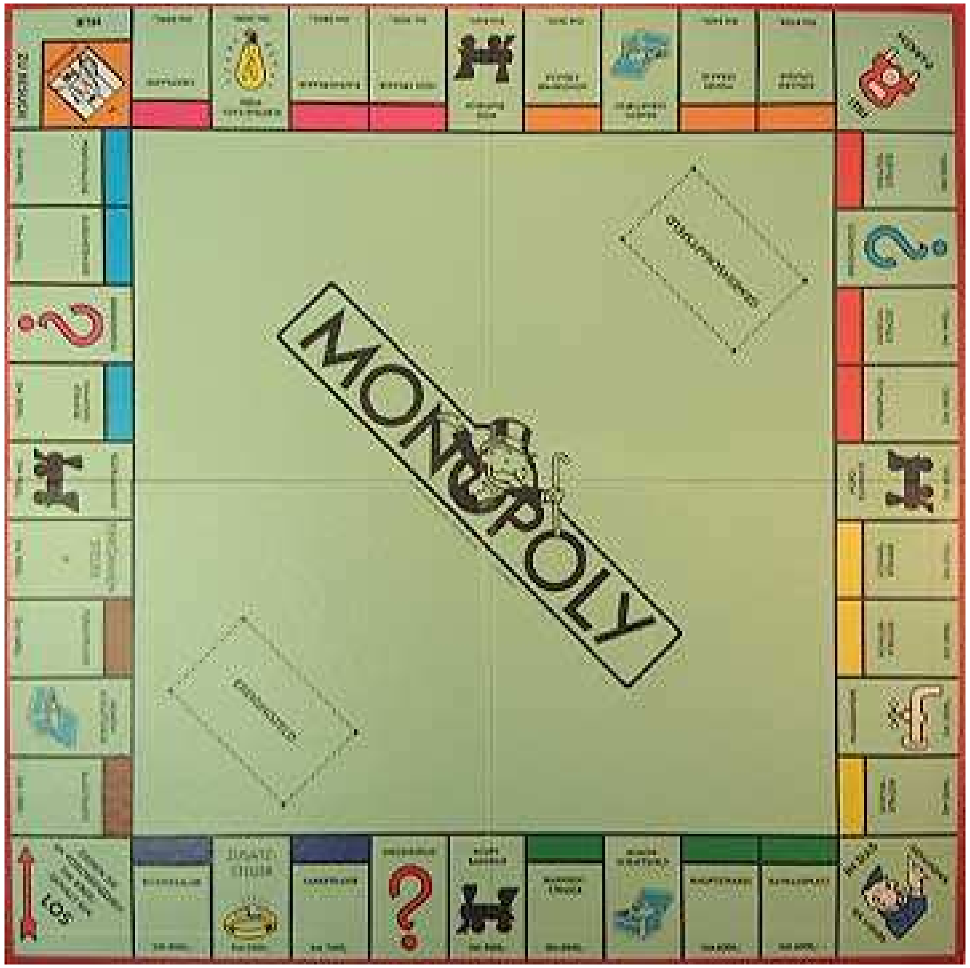
Fallstudie 2: Monopoly-Spiel

Wir modellieren Struktur und Ablauf des Monopoly-Spiels.

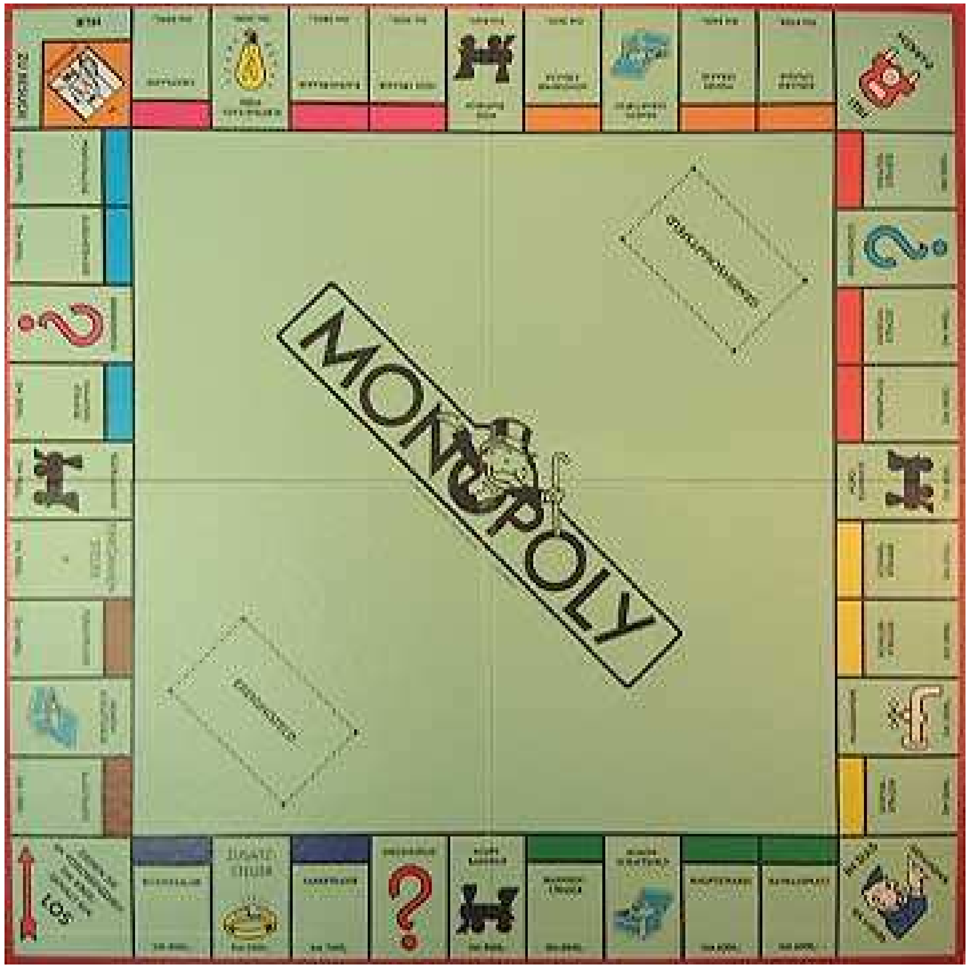
Ziel: Spielregeln präzisieren und formalisieren

Teilaufgaben:

1. **Informationen und Zusammenhänge**
2. **Bedingungen und Regeln**
3. **Spielabläufe**

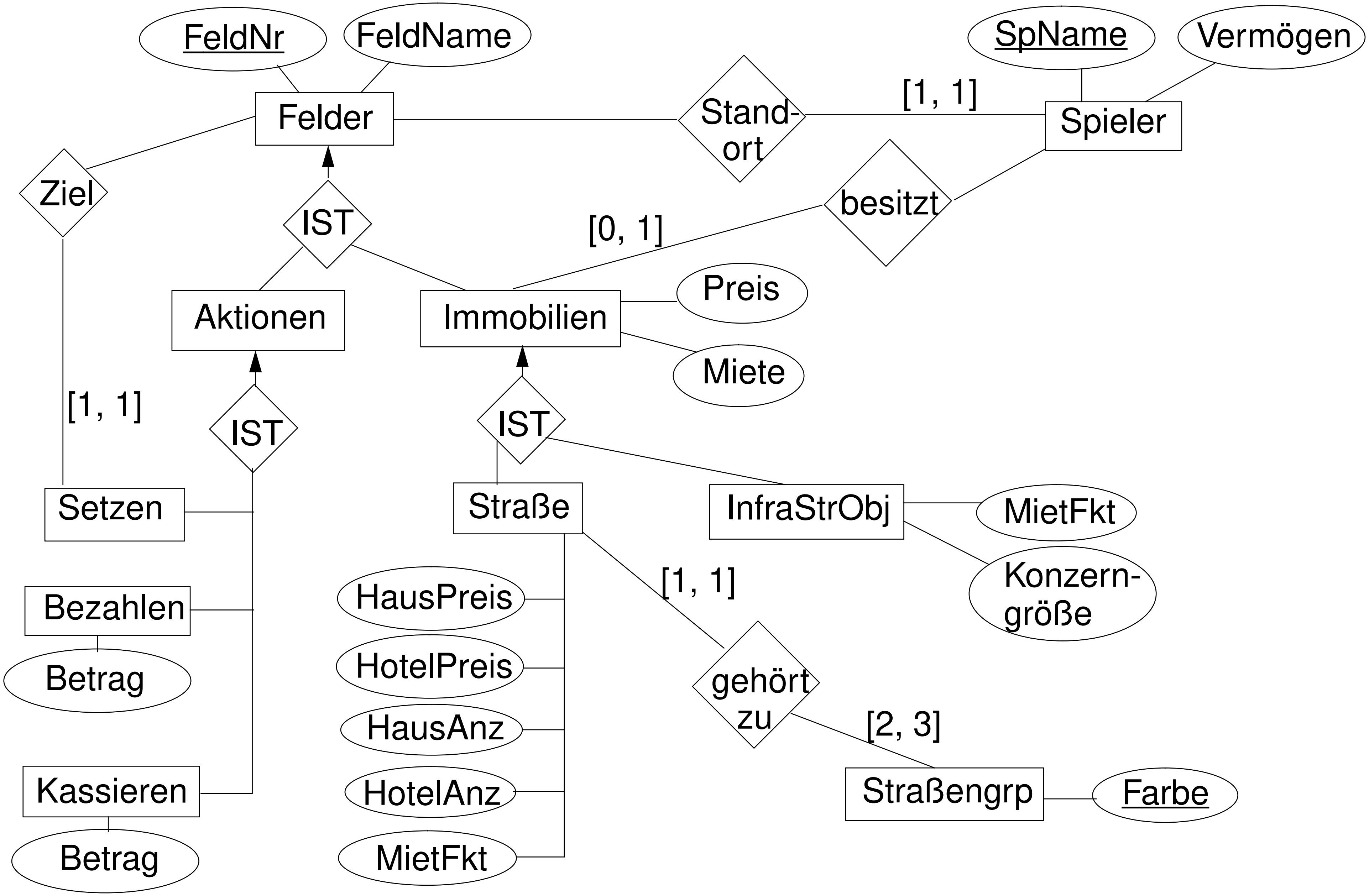


Kurzbeschreibung der Informationsstruktur



1. **Spieler**:
   hat einen Namen, steht auf einem Spielfeld, hat Vermögen, besitzt Immobilien
2. **Feld**:
   hat Nummer und Namen, ist entweder ein Aktionsfeld oder eine Immobilie
3. **Immobilie:**
   hat einen Preis und kostet Miete, ist entweder eine Straße oder ein Infrastrukturobjekt
4. **Straße**:
   hat Preise und Anzahl für Häuser und Hotels sowie Funktion zur Berechnung der Miete
5. **Infrastrukturobjekt**:
   hat Konzerngröße und eine Funktion zur Berechnung der Miete
6. **Aktionsfeld**:
   fordert auf zum Bezahlen oder Kassieren eines Betrages oder zum Setzen auf ein Feld
7. **Straßengruppe**:
   2 oder 3 Straßen werden zu einer Gruppe mit gleicher Farbe zusammengefasst

Informationsstruktur als ER-Modell



Einige Wertebereiche zur Informationsstruktur

FeldNr	:= { 1, 2, ..., 40 }
FeldArten	:= { istAktion, istImmobilie }
Felder	:= FeldNr $\times$ FeldName $\times$ FeldVarianten
FeldVarianten	:= { (istAktion, a) | a $\in$ Aktionen} $\cup$ { (istImmobilie, i) | i $\in$ Immobilien }
AktionsArten	:= { istSetzen, istBezahlen, istKassieren }
Aktionen	:= { (istSetzen) } $\cup$ { (istBezahlen, b) | b $\in$ Betrag } $\cup$ { (istKassieren, b) | b $\in$ Betrag }
Betrag	:= $\mathbb{N}_0$
ImmobilienArten	:= { istStraße, istInfraStrObj }
Immobilien	:= Preis $\times$ Miete $\times$ ImmobilienVarianten
ImmobilienVarianten	:= { (istStraße, s) | s $\in$ Straße } $\cup$ { (istInfraStrObj, i) | i $\in$ InfrastrObj }
Straße	:= HausPreis $\times$ HotelPreis $\times$ HausAnzahl $\times$ HotelAnzahl $\times$ MietFkt
besitzt	:= FeldNr –> SpName

Beispiele für Felder:

(1, Los, (istAktion, (istKassieren, 4000))) $\in$ Felder

(2, BadStraße, (istImmobilie, 1200, 40, (istStraße, 1000, 1000, 0, 0, MFkt2))) $\in$ Felder

(6, Südbahnhof, (istImmobilie, 4000, 1000, (istInfraStrObj, 2, MFktBhf))) $\in$ Felder

2.b Bedingungen

Die **Miete einer Straße steigt** je intensiver sie **bebaut** ist;
die **Miete eines Infrastrukturobjektes steigt**
je mehr **gleichartige Objekte** ein Spieler besitzt.

$\forall$ x $\in$ Immobilien: $\forall$ p $\forall$ m $\forall$ hap $\forall$ hop $\forall$ haanz $\forall$ hoanz $\forall$ n $\forall$ g

[x = (p, m, (istStraße, hap, hop, haanz, hoanz, f)) $\rightarrow$ m = f (haanz, hoanz)] $\wedge$

[x = (p, m, (istInfraStrObj, n, g)) $\rightarrow$ m = g (n)]

Eine Straße darf nur dann **bebaut** werden,
wenn der Besitzer **alle Straßen dieser Gruppe** besitzt.

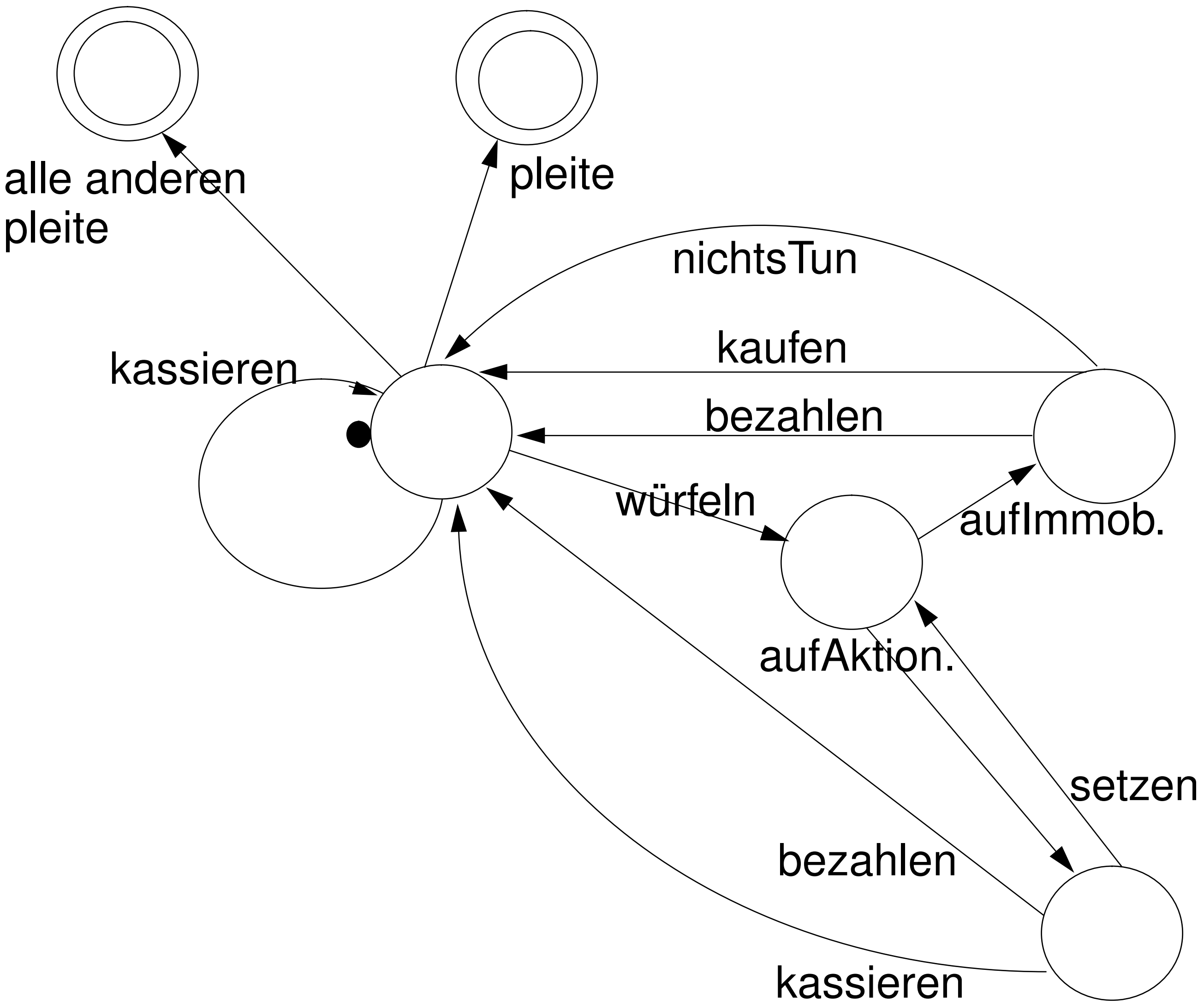
$\forall$ x $\in$ Felder: $\forall$ nr $\forall$ name $\forall$ p $\forall$ m $\forall$ hap $\forall$ hop $\forall$ haanz $\forall$ hoanz $\forall$ h

x = (nr, name, (istImmobilie, p, m, (istStraße, hap, hop, haanz, hoanz, h))) $\rightarrow$

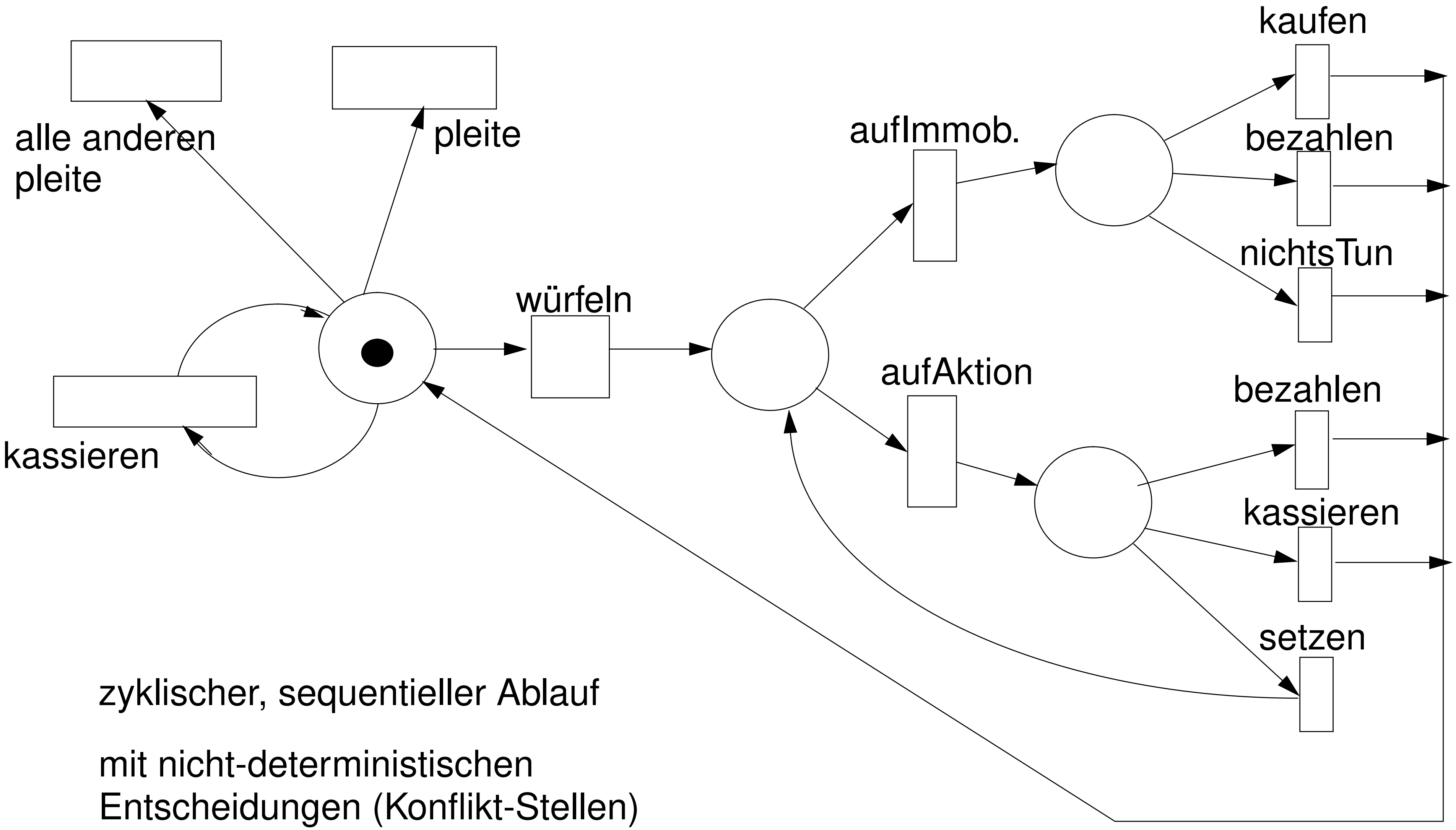
(haanz + hoanz > 0 $\wedge$ $\exists$ f $\in$ Farbe: (x, f) $\in$ gehörtZu $\wedge$ $\exists$ s $\in$ Spieler: (s, x) $\in$ besitzt

$\rightarrow$ $\forall$ g $\in$ Felder: (g, f) $\in$ gehörtZu $\rightarrow$ (s, g) $\in$ besitzt

2.c Aktionsfolgen eines Spielers (DEA)



2.c Aktionsfolgen eines Spielers (Petri-Netz)



zyklischer, sequentieller Ablauf

mit nicht-deterministischen Entscheidungen (Konflikt-Stellen)

8. Zusammenfassung
Zusammenfassung der Themen und Begriffe (1)

1. Modellbegriff

2.1 Wertebereiche beschrieben d. Mengen

Mengen, extensional, intensional, Operationen
Potenzmengen
Kartesisches Produkt
Indexmengen
Folgen
Relationen, Eigenschaften von Relationen
Funktionen, Eigenschaften,
spezielle Funktionen
disjunkte Vereinigung

2.2 Terme

Sorten, Signatur
korrekte Terme, Grundterme
Präfix-, Postfix-, Infix-Form, Funktionsform
Kantorowitsch-Bäume
Substitution
Umfassende Terme
Unifikation, allgemeinster Unifikator
Unifikationsverfahren
Abstrakte Algebra, Axiome
Konkrete Algebra
Datenstrukturen: Keller, Binärbaum
Konstruktor, Hilfskonstruktor, Projektion
Normalform

Zusammenfassung der Themen und Begriffe (2)

3.1 Aussagenlogik

AL Formeln, logische Junktoren
Belegung, Interpretation
Wahrheitstafeln
erfüllbar, unerfüllbar, allgemeingültig (Tautologie)
Gesetze der booleschen Algebra
aussagenlogischer Schluss, korrekter Schluss

3.2 Verifikation (Hoaresche Logik)

Schlussregeln für Sequenz,
Zuweisung,
2-seitige Alternative,
bedingte Anweisung,
Konsequenz, Aufruf,
Schleife, Schleifeninvariante,
Schleife aus Invariante konstruieren
Terminierung von Schleifen

3.3 Prädikatenlogik

PL Formeln,
gebundene und freie Variable
Wirkungsbereich von Quantoren
Umbenennung von Variablen
Interpretation von PL Formeln
Individuenbereich
Beschränkung von Wertebereichen
Umformungen
erfüllbar, unerfüllbar, allgemeingültig
PL Schluss

Zusammenfassung der Themen und Begriffe (3)

4. Graphen

4.1 Grundlegende Definitionen

Gerichtetet, ungerichteter Graph,
Multigraph, Teilgraph,
Grad, Eingangs-, Ausgangsgrad
Adjazenzmatrix, Adjazenzlisten

4.2 Wegeproblem

Weg, Kreis, Zyklus,
gerichteter azyklischer Graph,
zusammenhängend,
Zusammenhangskomponente,
Euler-Weg, Euler-Kreis, Hamilton-Kreis

4.3 Verbindungsprobleme

Baum, Spannbaum,
Schnittknoten, Brückenkante
orientierbarer Graph

4.4 Modellierung mit Bäumen

Gerichteter Baum, Wurzel, Höhe, Blätter
Binärbäume,
Entscheidungsbäume
Strukturbäume

4.5 Zuordnungsprobleme

Paarweise Zuordnung (Matching),
bipartit,
Färbung

4.6 Abhängigkeitsprobleme

Abhängigkeitsparagraph,
Anordnung (Scheduling),
Ablaufparagraph,
Aufrufgraph,
Programmablaufgraph

Zusammenfassung der Themen und Begriffe (4)

5. Modellierung von Strukturen

5.1 Kontextfreie Grammatiken

Terminale, Nichtterminale, Startsymbol
Produktionen,
Ableitung,
Sprache einer KFG,
Ableitungsbaum

5.2 Entity Relationship Modell

Entity-Menge, konkrete Ausprägung,
Attribut, Schlüsselattribut
Relation, Rollen, Kardinalität
IST-Spezialisierung

7. Fallstudien

Auftragsabwicklung in Autowerkstatt
Monopoly-Spiel
Getränkeautomat (Übungen)

6. Modellierung von Abläufen

6.1 Endliche Automaten

Alphabet,
reguläre Ausdrücke
deterministisch, nicht-deterministisch
Zustände, Übergangsfunktion
Anfangszustand, Endzustände
akzeptierte Sprache
NEA-DEA-Konstruktion,
Ausgabe, Mealy-Automat, Moore-Automat

6.2 Petri-Netze

Stellen, Transitionen, Vorbereich, Nachbereich,
Markierungsfunktion, Schaltregel,
Anfangsmarkierung, erreichbare Markierungen
zyklische Prozesse,
binäres Netz,
Verklemmung (deadlock),
Kapazitäten, Gewichte,
beschränkter Puffer, Leser-Schreiber-System

9. Wiederholung

Wunschkonzert – Modellierung WS 2001/2002

Nr.	Thema	# Stimmen	Platzierung
0	nichts	32	
1	Wertebereiche	**110**	**2. Platz**
2	Funktionen, Relationen	66	
3	Terme, Signaturen	64	
4	Substitution, Unifikation	50	
5	Algebren	45	
6	Aussagenlogische Formeln	14	
7	Aussagenlogische Schlüsse	27	
8	Verifikation	**182**	**1. Platz**
9	Prädikatenlogik: Interpretation	**75**	**3. Platz**
10	Prädikatenlogik: Umformungen	59	
11	Graphen: Wegeprobleme	8	
12	Graphen: Verbindungsprobleme	7	
13	Graphen: Bäume	10	
14	Graphen: Zuordnungsprobleme	7	
15	Graphen: Abhängigkeiten	8	
16	KFG: Bäume	5	
17	KFG: Ableitungen	16	
18	Entity−Relationship−Modell	16	
19	ER: Kardinalitäten	24	
20	Endliche Automaten	27	
21	Petri−Netze	21	
	Fehler:	0	
	Anzahl der Stimmen:	**873**	

9.1 Schlussregeln für Algorithmenelemente

Sequenzregel:

$$\frac{\{P\}\ S_1\ \{Q\} \qquad \{Q\}\ S_2;\ \{R\}}{\{P\}\ S_1;\ S_2\ \{R\}}$$

Zuweisungsregel:

$\{ P^{x}_{e} \}\ x := e;\ \{ P \}$

Wo in P x vorkommt, muss in P^{x}_{e} e stehen.

{P}
S_1;
{Q}
S_2
{R}

{ x = y*(1+q) + (r-y) }
r := r - y;
{ x = y*(1+q) + r }
q := 1 + q;
{ x = y*q + r }

Schlussregel für die Anweisung anwenden

Schlussregel für die Anweisung anwenden

Konsequenzregel:

$\{R\} \rightarrow \{Q\}$

in Ausführungsrichtung einfügen, wo nötig.

2-seitige Alternative

$$\frac{\{P \wedge B\}\ S_1\ \{Q\} \qquad \{P \wedge \neg B\}\ S_2\ \{Q\}}{\{P\}\ \text{falls B: } S_1 \text{ sonst } S_2\ \{Q\}}$$

{ P }

falls B:
    {P ∧ B}
        S_1
    { Q }

sonst
    {P ∧ ¬ B}
        S_2
    { Q }

{ Q }

{ true }

falls a > 0:
    { true ∧ a > 0 } ≡
        {a > 0} b := a; {b > 0} → { b ≥ 0 }
    { b ≥ 0 }

sonst
    {true ∧ ¬ (a > 0)}
        {¬ (a > 0) } ≡ { -a ≥ 0 } b := - a; { b ≥ 0 }
    { b ≥ 0 }

{ b ≥ 0 }

Bedingte Anweisung

$$\frac{\{P \wedge B\}\ S\ \{Q\} \qquad P \wedge \neg B \rightarrow Q}{\{P\}\ \text{falls}\ B\text{: } S\ \{Q\}}$$

(Bedingter Schritt)

$\{ P \}$

falls B:

  $\{P \wedge B\}$

  S_1

  $\{ Q \}$

  // leere Alternative:

  $\{P \wedge \neg B\} \rightarrow \{ Q \}$

$\{ Q \}$

$\{ \text{true} \}$

falls $a < 0$:

  $\{\text{true} \wedge a < 0\}$

    $\{a < 0\} \rightarrow \{ -a > 0 \}$

    $a := - a;$

    $\{a > 0\} \rightarrow \{ a \geq 0 \}$

  $\{ a \geq 0 \}$

  // leere Alternative:

  $\{\text{true} \wedge \neg (a < 0)\} \rightarrow \{ a \geq 0 \}$

$\{ a \geq 0 \}$

Schleifenregel

$$\frac{\{P \wedge B\}\ \ S\ \ \{P\}}{\{P\}\text{ solange B wiederhole S }\{P \wedge \neg B\}}$$

{P}
solange B wiederhole
  {P ∧ B}
  S
  {P}
{P ∧ ¬ B}

Terminierung separat zeigen.

Schleifeninvariante P

im Beispiel ist P:

$z * a^b = x^y \wedge b \geq 0$

a := x; b := y; z := 1;

$\{a = x \wedge b = y \wedge z = 1\} \rightarrow$

$\{z * a^b = x^y \wedge b \geq 0\}$

solange b > 0 wiederhole

  $\{z * a^b = x^y \wedge b \geq 0 \wedge b > 0\}$

  $\equiv \{z * a * a^{b-1} = x^y \wedge (b-1) \geq 0\}$

  b := b - 1;

  $\{z * a * a^b = x^y\ b \geq 0\}$

  z := z · a;

  $\{z * a^b = x^y \wedge b \geq 0\}$

$\{z * a^b = x^y \wedge b \geq 0 \wedge b \leq 0\}$

$\equiv \{z * a^b = x^y \wedge b = 0\} \rightarrow \{z = x^y\}$

9.2 Wertebereiche

Wertebereich: **Menge gleichartiger Werte** aus dem modellierten Gegenstandsbereich

1. Mengen:

Definition von Mengen durch
Aufzählen der Elemente (extensional): M := {1, 4, 9, 16, 25}
Angabe einer Bedingung (intensional): $M := \{a \mid a \in \mathbb{N}$, a ist Quadratzahl, $a \leq 30\}$

Indexmenge: Menge verwendet zur **eindeutigen Identifikation von Objekten**,
z. B. KartenSymbole := {7, 8, 9, 10, Bube, Dame, König, Ass}
vergleiche Schlüsselattribut im ER-Modell

2. Potenzmengen:

Potenzmenge (powerset) einer Grundmenge U ist die **Menge aller Teilmengen** von U,
geschrieben Pow (U) oder $\wp$ (U). $\text{Pow}(U) := \{ M \mid M \subseteq U \}$

Beispiel: Grundmenge U := {a, b} Potenzmenge Pow (U) = {∅, {a}, {b}, {a, b}}

Kardinalität: $|\text{Pow}(U)| = 2^n$ wenn $|U| = n$

Wertebereich n-stelliger **Relationen**: $\mathbf{Pow}\ (\mathbf{M_1} \times \mathbf{M_2} \times \ldots \times \mathbf{M_n})$

Wertebereich von **Funktionen**: $\mathbf{D \to B \subseteq Pow\ (D \times B)}$

Wertebereiche (2)

3. Kartesisches Produkt:

geordnete **n-Tupel**: $M_1 \times M_2 \times \ldots \times M_n := \{ (a_1, a_2, \ldots, a_n) \mid a_i \in M_i \text{ und } i \in I \}$
mit **Indexmenge** $I := \{1, \ldots, n\}$ und $n > 1$

Kardinalität: $| M_1 \times M_2 \times \ldots \times M_n | = \prod_{i \in I} |M_i|$ mit $I = \{1,\ldots, n\}$ mit $n > 1$

Beispiel: Daten := Tage $\times$ Monate $\times$ Jahre

bei **gleichen Mengen M_i** : $M \times M \times \ldots \times M = M^n$ mit $n > 1$

Beispiel: Wertebereich der Ergebnisse 3-maligen Würfelns: DreiWürfe := $\{1, 2, 3, 4, 5, 6\}^3$

4. Endliche Folgen

von Elementen aus A: n-Tupel über A von beliebiger Länge $n > 0$

$\mathbf{A^+} := \{ x \mid x \in A^i \text{ und } i \geq 1 \}$ d. h. $(a_1, \ldots, a_n) \in A^+$ für $n \geq 1$ und $a_i \in A$

$\mathbf{A^* := A^+} \cup \{ () \}$

Beispiel: $(1, 1, 2, 5, 5, 10, 20) \in \mathbb{N}^+$

Wertebereiche (3)

4. Relationen

n-stellige Relation: $R \subseteq M_1 \times M_2 \times \ldots \times M_n$

$R \in$ **Pow** $(\mathbf{M_1} \times \mathbf{M_2} \times \ldots \times \mathbf{M_n})$

Kardinalität ist $|$**Pow** $(\mathbf{M_1} \times \mathbf{M_2} \times \ldots \times \mathbf{M_n})| = 2^{\prod_{i \in I} |\mathbf{Mi}|}$, falls alle $\mathbf{M_i}$ endlich sind.

Beispiel: Oder := {(w, w), (w, f), (f, w)} $\subseteq$ $Bool^2$

5. Funktionen

Funktion f ist eine 2-stellige Relation f $\subseteq$ **D** $\times$ **B** mit folgender Eigenschaft:
Aus (x, y) $\in$ f und (x, z) $\in$ f folgt y = z, d. h. zu einem Urbild x gibt es nur ein Bild y.

Wertebereich D -> B ist die **Menge aller Funktionen, die von D auf B abbilden**.
Es gilt **D -> B** $\subseteq$ **Pow (D** $\times$ **B).**

Beispiel: Quadrat := { (a, b) | a $\in \mathbb{N}$ und b = a*a } $\in \mathbb{N} \to \mathbb{N}$

Kardinalität des Wertebereiches, aus dem Funktionen stammen $|D \to B| = (|B|+1)^{|\mathbf{D}|}$
Anzahl der totalen Funktionen in $|D \to B|$ ist $|B|^{|\mathbf{D}|}$

Wertebereiche (4)

Spezielle Funktionen

Identitätsfunktion $\mathbf{id_M}$: M -> M mit $id_{\mathbf{M}}$:= { (x, x) | x∈ M }

Charakteristische Funktion $\chi_{\mathbf{M}}$ **einer Menge** M ⊆ U, mit der Trägermenge U gibt an, welche Elemente der Trägermenge U in M enthalten sind.

Prädikate: Funktionen mit Bildbereich Bool, z. B. <= : $(\mathbb{N}_0 \times \mathbb{N}_0)$ -> Bool

Funktionen zur Modellierung von **mehrfachen Vorkommen** (**Multimengen, bags)**:
b: V -> $\mathbb{N}_0$ gibt für jeden Wert aus V an, wie oft er vorkommt

z. B. geldBeutel: brdMünzen -> $\mathbb{N}_0$
mit geldBeutel := {(1,3), (2, 0), (5,0), (10, 2), (50, 1), (100, 3), (200, 2), (500, 1)}

6. disjunkte Vereinigung

Wertebereiche $A_1, A_2, ..., A_n$ zu einem **vereinigten Wertebereich V** zusammengefasst .
Die Herkunft der Elemente aus $A_{\mathbf{i}}$ wird in den Paaren von V gekennzeichnet:

$\mathbf{V := \{ (i, a_i) \mid a_i \in A_i \}}$ wobei I := {1, .1, ..., n}

Die erste Komponente der Paare ist eine **Kennzeichenkomponente** (tag field).
Die A_i brauchen nicht paarweise disjunkt zu sein.

Kardinalität: $|V| = \sum_{i \in I} |A_i|$

9.3 Interpretation prädikatenlogischer Formeln

Prädikatenlogische Formel besteht aus:	**Beispiel:**	**Interpretation $\mathfrak{I}$ legt fest:**
	$\forall x\ P(x, f(x)) \wedge Q(g(a, z))$	Individuenbereich U := { ... }
atomaren Formeln	P(x, f(x)) Q(g(a, z))	
mit **Prädikatssymbolen**	P Q	Prädikate $\mathfrak{I}$(P) := { (a, b) | ...}
Termen als Parametern	x f(x) g(a, z)	
Funktionssymbolen	f g a	Funktionen $\mathfrak{I}$(f)(n) -> n+1
freien Variablen	z	Variablenwerte $\mathfrak{I}$(z) := 3
gebundenen Variablen	x	
Formeln mit **logischen Junktoren**	$P(x, f(x)) \wedge Q(g(a, z))$	Verknüpfung gemäß AL
Quantor-Formeln	$\forall x\ P(x, f(x)) \wedge Q(g(a, z))$	Bindung jedes Wertes aus U an Variable; Auswertung gemäß Bedeutung von $\forall$ und $\exists$

Auswertung von Quantor-Formeln

$\forall x\, F(x)$ jeden Wert aus U einsetzen und Ergebnisse mit $\wedge$ verknüpfen: $F(u_1) \wedge F(u_2) \wedge \ldots$

$\exists x\, F(x)$ jeden Wert aus U einsetzen und Ergebnisse mit $\vee$ verknüpfen: $F(u_1) \vee F(u_2) \vee \ldots$

Wenn man einen Wert findet, der das Ergebnis bestimmt,
$F(u_i) = f$ bei $\forall$ bzw. $F(u_j) = w$ bei $\exists$, braucht man nicht alle aufzuzählen.

Beispiel:

$U := \{ 1, 2, 3 \}$ $\mathfrak{I}(P) := \{ (a, b) \mid a + b < 5 \}$

$\exists x$	$\forall y$	$P(x, y)$				
1	1	w	$\wedge$			
1	2	w	$\wedge$			
1	3	w	=	**w**	$\vee$	← hier steht das Ergebnis fest
2	1	w	$\wedge$			
2	2	w	$\wedge$			
2	3	f	=	f	$\vee$	
3	1	w	$\wedge$			
3	2	f	$\wedge$			
3	3	f	=	f	= **w**	