Modellierung

Prof. Dr. Uwe Kastens

WS 2011 / 2012

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,
- die für die Methoden **typischen Techniken** erlernen und
- Kalküle an **typischen Beispielen** anwenden.

Insgesamt sollen sie lernen,

- Aufgaben **präzise** zu analysieren und zu beschreiben,
- **formale Kalküle als Arbeitsmittel** einzusetzen und
- den **praktischen Wert von präzisen Beschreibungen** erkennen.

siehe **Beschreibung des Moduls I.2.1 im Modulhandbuch**:
http://www.cs.uni-paderborn.de/studium/studiengaenge/pruefungswesen/modulhandbuch.html

Durchführung

Zu jedem **Modellierungskalkül** soll(en)

- mit einigen typischen kleinen **Beispielen motivierend** hingeführt 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	Kap. im Buch „Modellierung“
1. Einführung	1	1
2. Grundlegende Strukturen		
Wertebereiche	2	2
Beweistechniken	3	4.3
3. Terme, Algebren	4, 5	3
4. Logik		
Aussagenlogik	6	4.1
Verifikation von Algorithmen	7	-
Prädikatenlogik	8	4.2
5. Graphen	9, 10	5
Verbindung, Zuordnung, Anordnung		
6. Modellierung von Strukturen		
Kontextfreie Grammatiken,	11	6.1
XML		6.2
Entity-Relationship Modell	12	6.3
UML Klassendiagrame		6.4
7. Modellierung von Abläufen		
Endliche Automaten,	13	7.1
Petri-Netze	14	7.2
8. Projekte, Zusammenfassung	15	8

Literaturhinweise

Elektronisches Vorlesungsmaterial:

- **U. Kastens: Vorlesung Modellierung WS 2011 / 2012**
  http://ag-kastens.uni-paderborn.de/lehre/material/model

Das Buch zur Vorlesung:

- **Uwe Kastens, Hans Kleine Büning: Modellierung - Grundlagen und formale Methoden, 2. Auflage, Carl Hanser Verlag, 2008**

Weitere Bücher zum Nachlernen und Nachschlagen:

- **Gerhard Goos: Vorlesungen über Informatik, Band 1, 3. Auflage, Springer-Lehrbuch, 2000**
- **Thierry Scheurer: Foundations of Computing, System Development with Set Theory and Logic, Addison-Wesley, 1994**
- **Daniel J. Velleman: How To Prove It - A Structured Approach, 2nd ed., Cambridge University Press, 2006**

Bezüge zu anderen Vorlesungen

Modellierung

Problem verstehen bevor implementieren

Grundlagen der Programmierung

Eigenschaften und Strukturen von Sprachen

Grundlagen der Programmiersprachen

Spezifikation von Software-Aufgaben und -Lösungen

Softwareentwurf

Probleme und Aufgaben präzise beschreiben

Eigenschaften von D & A formal beschreiben

Datenstrukturen & Algorithmen

auf grundlegenden Kalkülen aufbauen

Berechenbarkeit & formale Sprachen Algorithmen & Komplexität

Elektronisches Skript: Startseite

Elektronisches Skript: Termine

Termine

Vorlesung

- `Mo, 11:15 - 12:45, Hörsaal L 2`
- `Fr, 11:15 - 12:45, Hörsaal L 1`

Beginn: 10. Okt 2011
Ende: 3. Feb 2012

Zentralübung

- `Mo, 13:00 - 13:45, Hörsaal L 2`

Beginn: 24. Okt 2011
Ende: 30. Jan. 2012

Übungen

vorläufige Liste, übernommen aus dem Vorlesungsverzeichnis:

- `Übung 01  Mo 14:00 N 3 206`

...

- `Übung 18  Fr 14:00 N 3 206`

Beginn: Mo 17. Okt. 2011
Ende: Fr 3. Feb. 2012

Klausurtermine

Es wird zwei Klausurtermine nach Ende der Vorlesungszeit geben. Ort, Beginn und die Anmeldezeit wird das ZPS festlegen

In der Klausur sind nur die folgenden Hilfsmittel erlaubt:

- Ein **beidseitig von Hand beschriebenes DIN A4 Blatt**. Das Blatt muss **persönlich von Hand** beschrieben sein. Es sind also insbesondere **keine Ausdrucke oder Kopien** erlaubt. Auf dem Blatt muss die **Matrikelnummer und der Name** stehen. Wer ein solches Blatt in der Klausur nutzt, muss es **mit der Klausur abgeben**. Bei der Klausureinsicht kann das Blatt wieder abgeholt werden.
- Studierende, deren Muttersprache nicht deutsch ist, dürfen außerdem in der Klausur ein **fremdsprachiges Wörterbuch ohne handschriftliche Eintragungen** benutzen.

Weitere Wiederholungen der Klausur findet erst nach dem nächsten Wintersemester statt und werden mit möglicher Weise anderen Modalitäten von einem anderen Dozenten durchgeführt. **Bonuspunkte werden dorthin NICHT übertragen.**

Regeln

Übungen:

Es werden 4-stündige Übungen angeboten. Darin werden Aufgaben zum Vorlesungsstoff **unter Anleitung gelöst**.

Hausübungen:

Es wird in jeder Woche ein **Hausübungszettel** ausgegeben (freitags). Abgabe der Lösungen am übernächsten Montag. Bearbeitung in **Gruppen** (2-4). Lösungen werden korrigiert, bewertet und zurückgegeben.

Kurztests:

Es werden voraussichtlich 4 Kurztests (ca. 20 min) während der Zentralübung geschrieben korrigiert, bewertet und zurückgegeben.

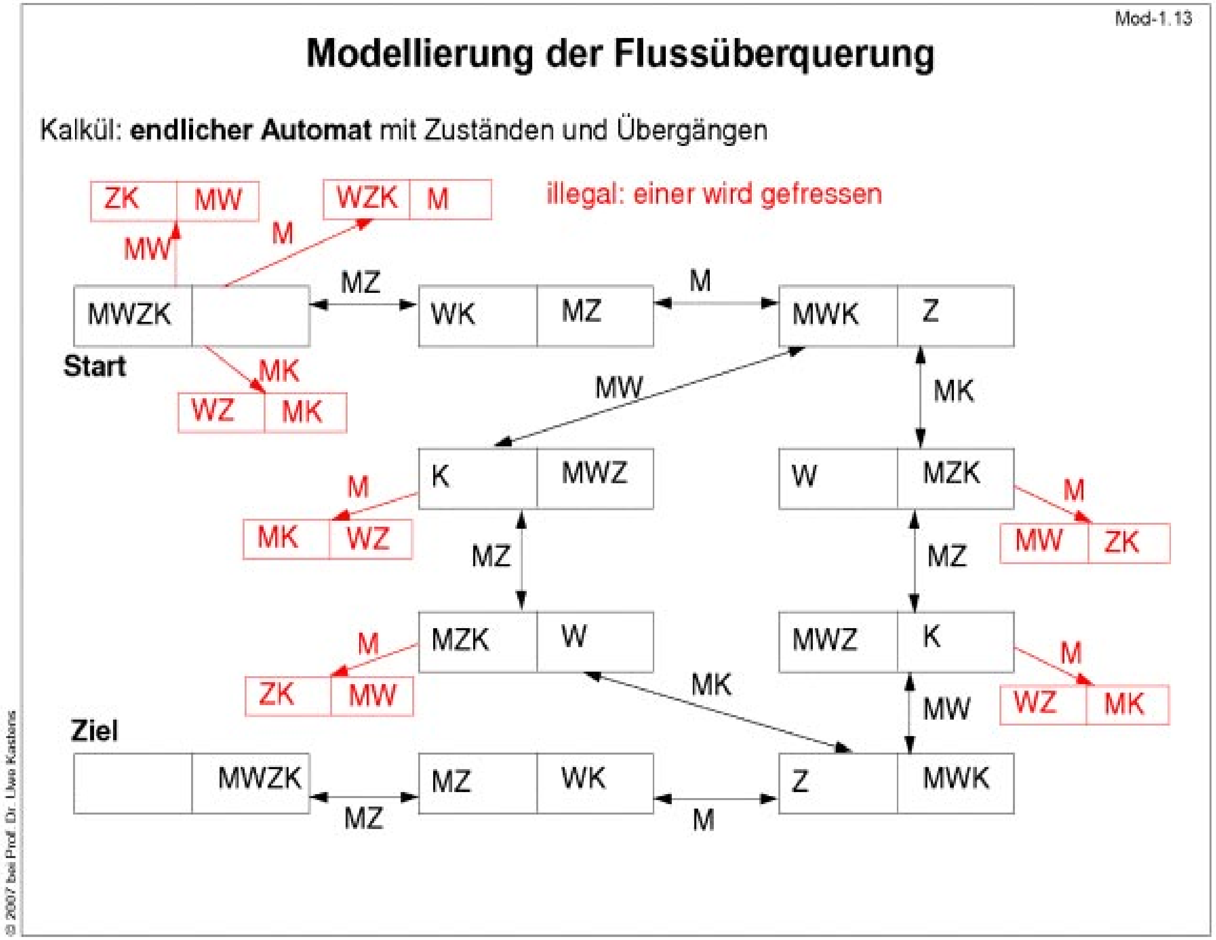
Bonus:

Durch **Vorrechnen** in den Übungen, Punkte aus den **Hausübungen** und den **Kurztests** kann ein Bonus erworben werden.
Damit kann die Note einer bestandenen Klausur um 1 oder 2 Notenschritte verbessert werden, z.B. von 2,3 auf 2,0 oder von 3,0 auf 2,3.

Details der Regeln findet man auf der **Organisationsseite**.

Elektronisches Skript: kommentierte Folien

Modellierung WS 2007/2008 - Folie 113



Ziele:
Prozess der Modellierung am Beispiel erkennen

in der Vorlesung:
Erläuterungen dazu (siehe auch nächste Folie):

- Bedeutung der Graphik und der Symbole,
- Zustände und Übergänge eines endlichen Automaten (siehe Kap. 8),
- Darstellung als Graph mit Knoten und Kanten (siehe Kap. 6)
- Wertebereiche der Information zu Zuständen (siehe Kap. 2)

Verständnisfragen:

- Prüfen Sie, ob das Modell die Aufgabe korrekt und vollständig beschreibt.

Autor: Prof. Dr. Uwe Kastens

Generiert mit Camelot | Probleme mit Camelot? | Geändert am: 01.10.2007

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

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?

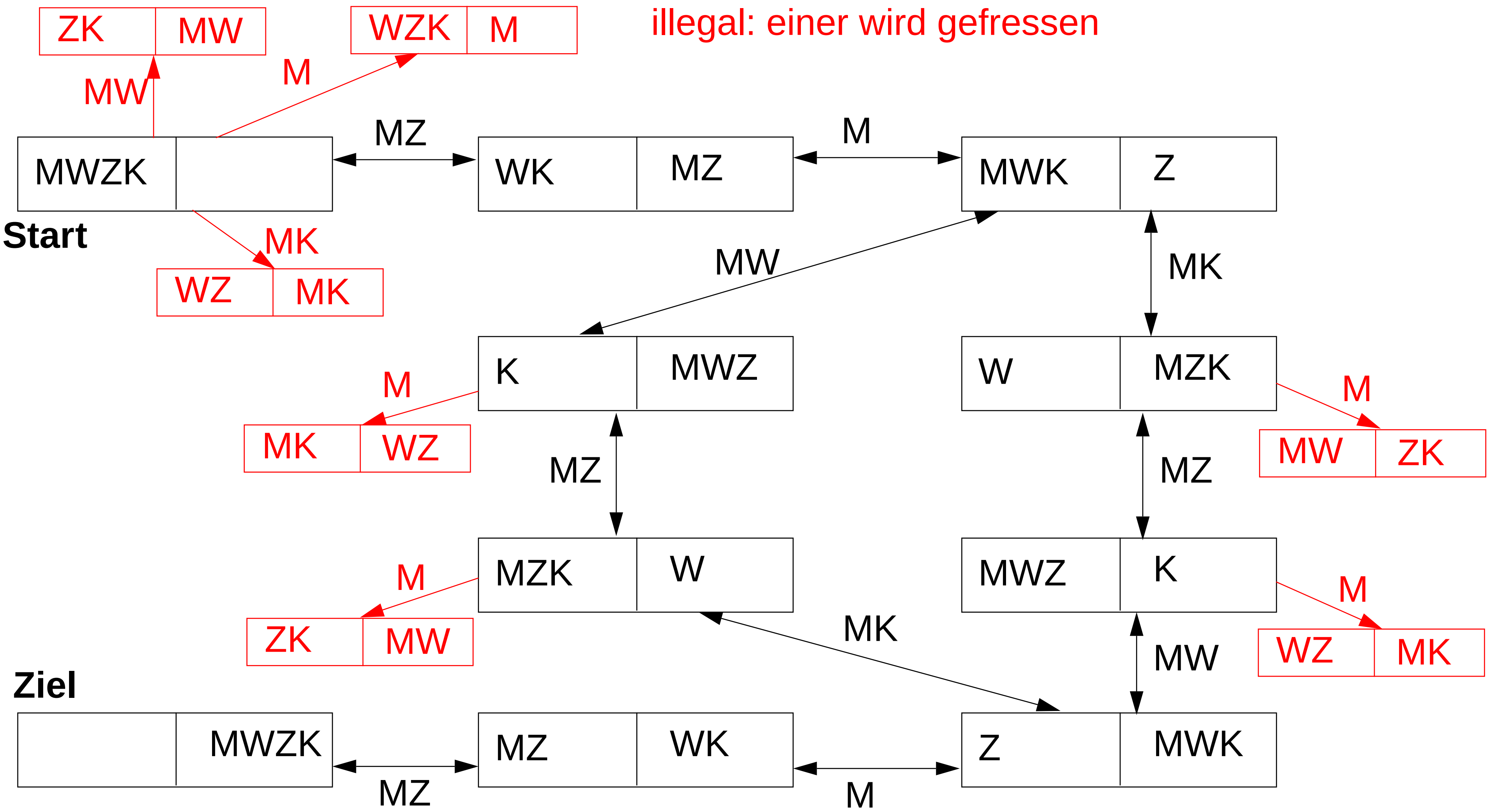
Quelle: Hopcroft, Ullman: Einführung in die Automatentheorie, formale Sprachen und Komplexitätstheorie, S. 14, 15

Erste Analyse: evtl. wichtige

- **Objekte**: Mann, Wolf, Ziege, Kohlkopf, Ufer (links u. rechts), Fluss, Boot
- **Eigenschaften, Beziehungen**: unbewacht an einem Ufer, Wolf frisst Ziege, Ziege frisst Kohl, Boot trägt Mann + 1 Objekt
- **Tätigkeiten**: Fluss überqueren, Objekt transportieren

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

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.

Kreative Leistung:

- Kalkül „endlicher Automat“ wählen, Bedeutung der Zustände und Übergänge festlegen

systematische Tätigkeit:

- speziellen Automat aufstellen, Lösungsweg finden

Meist kann man Lösungen am Modell entwickeln.

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)

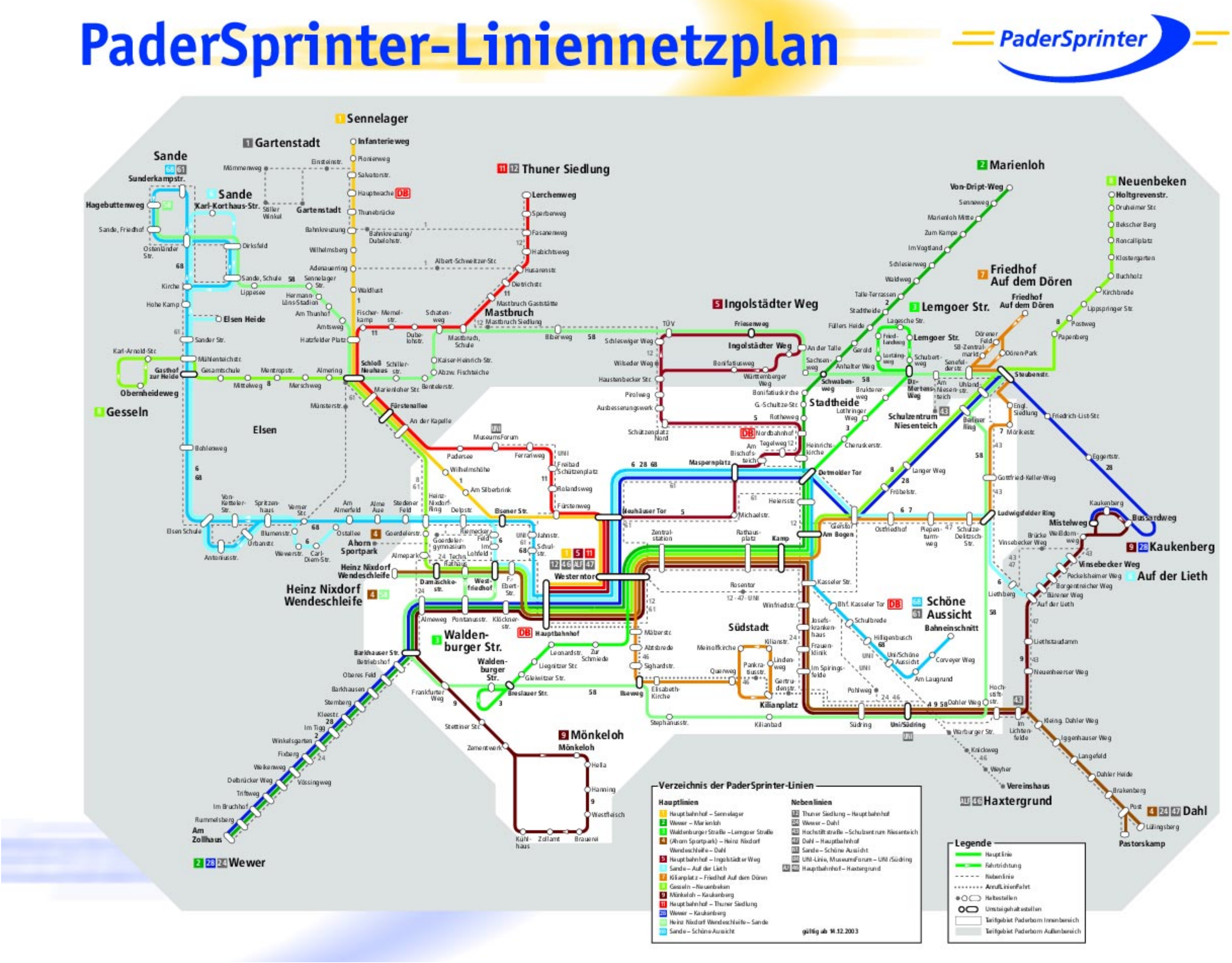
davon abweichende Bedeutungen:

- 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**

Modell: Buslinienplan



Modell: Busfahrplan

4 Dahl → Im Lichtenfelde → Universität/Südring → Husener Straße → Hauptbahnhof → Westfriedhof → HN Wendschleife

MONTAG BIS FREITAG

	5 Uhr 8	5 Uhr A 9	6 Uhr 9	6 Uhr 8	6 Uhr	6 Uhr 9	6 Uhr 8	6 Uhr 9	6 Uhr 8	6 Uhr
Pastoriskamp					11					50
Lüingsberg					12					51
Dahl Post					13					52
Brakenberg					14					53
Dahler Heide					15					54
Langefeld					16					55
Iggenhauser Weg					17					56
Kleingärten Dahler Weg					18					57
Im Lichtenfelde		19	01		19	25		45		58
Hochstiftstraße		20	02		20	26		46		59
Uni/Südring		21	03		21	27		47		00
Südring		22	04		22	28		48		01
Im Spiringsfelde		23	05		23	29		49		02
Frauenklinik		24	06		24	30		50		03
Josefskrankenhaus		25	07		25	31		51		04
Winfriedstraße		26	08		26	32		52		05
Kasseler Straße		27	09		28	33		53		07
Kamp	28	28	10	12	29	34	34	54	54	08
Rathausplatz	29	29	11	13	30	35	35	55	55	09
Zentralstation	30	30	12	14	31	36	36	56	56	10
❶ Westerntor	32	32	14	16	32	38	38	58	58	11
Hauptbahnhof Paderborn	34	34	16	18	34	40	40	00	00	13
Friedrich-Ebert-Straße	36			20	35		42		02	14
Westfriedhof	37			21	36		43		03	15
Technisches Rathaus	38			22	37		44		04	16
Damaschkestraße	39			23	38		45		05	17
HN Wendeschleife	40			24	39		46		06	18
Almepark				25			47		07	
Ahorn Sportpark										

	7 Uhr 9	7 Uhr 07	7 Uhr 8	7 Uhr	7 Uhr 9	8...-14... Uhr 8	8...-14... Uhr	8...-14... Uhr 9	8...-14... Uhr 8	8...-14... Uhr	8...-14... Uhr 9
Pastoriskamp		14		36			06			36	
Lüingsberg		15		37			07			37	
Dahl Post		16		39			09			39	
Brakenberg		17		40			10			40	
Dahler Heide		18		41			11			41	
Langefeld		20		42			12			42	
Iggenhauser Weg		21		43			13			43	
Kleingärten Dahler Weg		22		44			14			44	
Im Lichtenfelde	24	|		45	58		15	28		45	58
Hochstiftstraße	25	|		46	59		16	29		46	59
Uni/Südring	26	|		47	01		17	31		47	01
Südring	27	|		48	02		18	32		48	02
Im Spiringsfelde	28	|		49	03		19	33		49	03
Frauenklinik	29	|		50	04		20	34		50	04
Josefskrankenhaus	30	|		51	06		21	36		51	06
Winfriedstraße	31	|		52	07		22	37		52	07
Kasseler Straße	33	34		54	09		24	39		54	09
Kamp	34	|	36	55	10	10	25	40	40	55	10
Rathausplatz	35	|	37	56	11	11	26	41	41	56	11
Zentralstation	36	|	39	58	13	13	28	43	43	58	13
❶ Westerntor	38	39	41	00	15	15	30	45	45	00	15
Hauptbahnhof Paderborn	40	41	43	02	17	17	32	47	47	02	17
Friedrich-Ebert-Straße			44	03		18	33		48	03	
Westfriedhof			45	04		19	34		49	04	
Technisches Rathaus			46	05		20	35		50	05	
Damaschkestraße			47	06		21	36		51	06	
HN Wendeschleife			48	07		|	37		|	07	
Almepark			49			22			52		
Ahorn Sportpark											

	15 Uhr 8	15 Uhr	15 Uhr 9	15 Uhr 8	15 Uhr	15 Uhr 9	15 Uhr 8	16 Uhr	16 Uhr 9	16 Uhr 8	16 Uhr	16 Uhr 9
Pastoriskamp		06			36			06			36	
Lüingsberg		07			37			07			37	
Dahl Post		09			39			09			39	
Brakenberg		10			40			10			40	
Dahler Heide		11			41			11			41	
Langefeld		12			42			12			42	
Iggenhauser Weg		13			43			13			43	
Kleingärten Dahler Weg		14			44			14			44	
Im Lichtenfelde		15	28		45	58		15	28		45	58
Hochstiftstraße		16	29		46	59		16	29		46	59
Uni/Südring		17	31		47	01		17	31		47	01
Südring		18	32		48	02		18	32		48	02
Im Spiringsfelde		19	33		49	03		19	33		49	03
Frauenklinik		20	34		50	04		20	34		50	04
Josefskrankenhaus		21	36		51	06		21	36		51	06
Winfriedstraße		22	37		52	07		22	37		52	07
Kasseler Straße		24	39		54	09		24	39		54	09
Kamp	10	25	40	40	55	10	10	25	40	40	55	10
Rathausplatz	11	26	41	41	56	11	11	26	41	41	56	11
Zentralstation	13	28	43	43	58	13	13	28	43	43	58	13
❶ Westerntor	15	30	45	45	00	15	15	30	45	45	00	15
Hauptbahnhof Paderborn	17	32	47	47	02	17	17	32	47	47	02	17
Friedrich-Ebert-Straße	18	33		48	03		18	33		48	03	
Westfriedhof	19	34		49	04		19	34		49	04	
Technisches Rathaus	20	35		50	05		20	35		50	05	
Damaschkestraße	21	36		51	06		21	36		51	06	
HN Wendeschleife	|	37		52	07		|	37		|	07	
Almepark	22			53			22			52		
Ahorn Sportpark												

	17 Uhr 8	17 Uhr	17 Uhr 9	17 Uhr 8	17 Uhr	17 Uhr 9	18 Uhr 8	18 Uhr	18 Uhr 9	18 Uhr 8	18 Uhr	18 Uhr 9
Pastoriskamp		06			36			06			36	
Lüingsberg		07			37			07			37	
Dahl Post		09			39			09			39	
Brakenberg		10			40			10			40	
Dahler Heide		11			41			11			41	
Langefeld		12			42			12			42	
Iggenhauser Weg		13			43			13			43	
Kleingärten Dahler Weg		14			44			14			44	
Im Lichtenfelde		15	28		45	58		15	28		45	58
Hochstiftstraße		16	29		46	59		16	29		46	59
Uni/Südring		17	31		47	01		17	31		47	01
Südring		18	32		48	02		18	32		48	02
Im Spiringsfelde		19	33		49	03		19	33		49	03
Frauenklinik		20	34		50	04		20	34		50	04
Josefskrankenhaus		21	36		51	06		21	36		51	06
Winfriedstraße		22	37		52	07		22	37		52	07
Kasseler Straße		24	39		54	09		24	39		54	09
Kamp	10	25	40	40	55	10	10	25	40	40	55	10
Rathausplatz	11	26	41	41	56	11	11	26	41	41	56	11
Zentralstation	13	28	43	43	58	13	13	28	43	43	58	13
❶ Westerntor	15	30	45	45	00	15	15	30	45	45	00	15
Hauptbahnhof Paderborn	17	32	47	47	02	17	17	32	47	47	02	17
Friedrich-Ebert-Straße	18	33		48	03		18	33		48	03	
Westfriedhof	19	34		49	04		19	34		49	04	
Technisches Rathaus	20	35		50	05		20	35		50	05	
Damaschkestraße	21	36		51	06		21	36		51	06	
HN Wendeschleife	|	37		|	07		|	37		|	07	
Almepark	22			52	09		22			52	09	
Ahorn Sportpark					10						10	

	19 Uhr 8	19 Uhr	19 Uhr 9	19 Uhr	19 Uhr 9	20 Uhr 8	20 Uhr 9	20 Uhr 8	20 Uhr 8	20 Uhr
Pastoriskamp		06		36						35
Lüingsberg		07		37						|
Dahl Post		09		39						36
Brakenberg		10		40						37
Dahler Heide		11		41						38
Langefeld		12		42						39
Iggenhauser Weg		13		43						40
Kleingärten Dahler Weg		14		44						41
Im Lichtenfelde		15	28	45	58		25			42
Hochstiftstraße		16	29	46	59		26			43
Uni/Südring		17	31	47	01		27			44
Südring		18	32	48	02		28			45
Im Spiringsfelde		19	33	49	03		29			46
Frauenklinik		20	34	50	04		30			47
Josefskrankenhaus		21	36	51	06		31			48
Winfriedstraße		22	37	52	07		32			49
Kasseler Straße		24	39	54	09		34			50
Kamp	40	25	40	55	10	10	35		40	51
Rathausplatz	41	26	41	56	11	11	36		41	52
Zentralstation	43	28	43	58	13	13	37	38	43	53
❶ Westerntor	45	30	45	00	15	15	39	40	45	54
Hauptbahnhof Paderborn	47	32	47	02	17	17	41	42	47	56
Friedrich-Ebert-Straße	48	33		03		18		43		57
Westfriedhof	49	34		04		19		44		58
Technisches Rathaus	50	35		05		20		45		59
Damaschkestraße	51	36		06		21		46		00
HN Wendeschleife	52	37		07		|		|		01
Almepark				09		22		46		02
Ahorn Sportpark				10						04

	21 Uhr 8	21 Uhr 9	21 Uhr	22 Uhr 8	22 Uhr B 9	22 Uhr	23 Uhr B 9	23 Uhr	23 Uhr 8	0 Uhr 9	0 Uhr
Pastoriskamp			35			35		36			38
Lüingsberg			|			|		|			|
Dahl Post			35			36		37			39
Brakenberg			37			37		38			40
Dahler Heide			38			38		39			41
Langefeld			39			39		40			42
Iggenhauser Weg			40			40		41			43
Kleingärten Dahler Weg			41			41		42			44
Im Lichtenfelde		16	42		16	42	16	43		16	45
Hochstiftstraße		17	43		17	43	17	44		17	46
Uni/Südring		18	44		18	44	18	45		18	47
Südring		19	45		19	45	19	46		19	48
Im Spiringsfelde		20	46		20	46	20	47		20	49
Frauenklinik		21	47		21	47	21	48		21	50
Josefskrankenhaus		22	48		22	48	22	49		22	51
Winfriedstraße		23	49		23	49	23	50		23	52
Kasseler Straße		25	50		25	50	25	51		25	53
Kamp	24	26	51	27	26	51	26	52	55	26	54
Rathausplatz	25	27	52	28	27	52	27	53	56	27	55
Zentralstation	26	28	53	29	28	53	28	54	57	28	56
❶ Westerntor	27	30	54	30	30	54	30	55	58	30	57
Hauptbahnhof Paderborn	28	32	56	31		56		57	59	32	59
Friedrich-Ebert-Straße			57	33		57					
Westfriedhof			58	34		58					
Technisches Rathaus			59	35		59					
Damaschkestraße			00	36		00					
HN Wendeschleife			01	|		01					
Almepark				36							
Ahorn Sportpark											

A weiter bis HN-Wendeschleife
B weiter als Linie 7 Kilianplatz
8 Linie 8 (Gesseln)
9 Linie 9 (Mönkeloh)

Fahrplanerklärung

		12
	59	13
37	01	15
38	02	16

Stundenwechsel Der Bus wechselt hier in die nächste Stunde

S 51 11 **Parallelverkehr** Linien, die streckenweise auf derselben Route fahren, sind farbig gekennzeichnet.

8...-19... Uhr **Stundengruppen** In diesen Stunden fährt der Bus immer zu denselben Abfahrtsminuten.

Umsteigemöglichkeiten Mo-Fr (Anschlusszeit < 10 Min.) ❶ → 6 Sande, 6 Auf der Lieth, 28 Wewer, 28 Kaukenberg

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

▷ 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 → *Gegenstandsraum*

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 (→ *Klassifizierung*[2]) 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*

Modell in der mathematischen Logik

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.

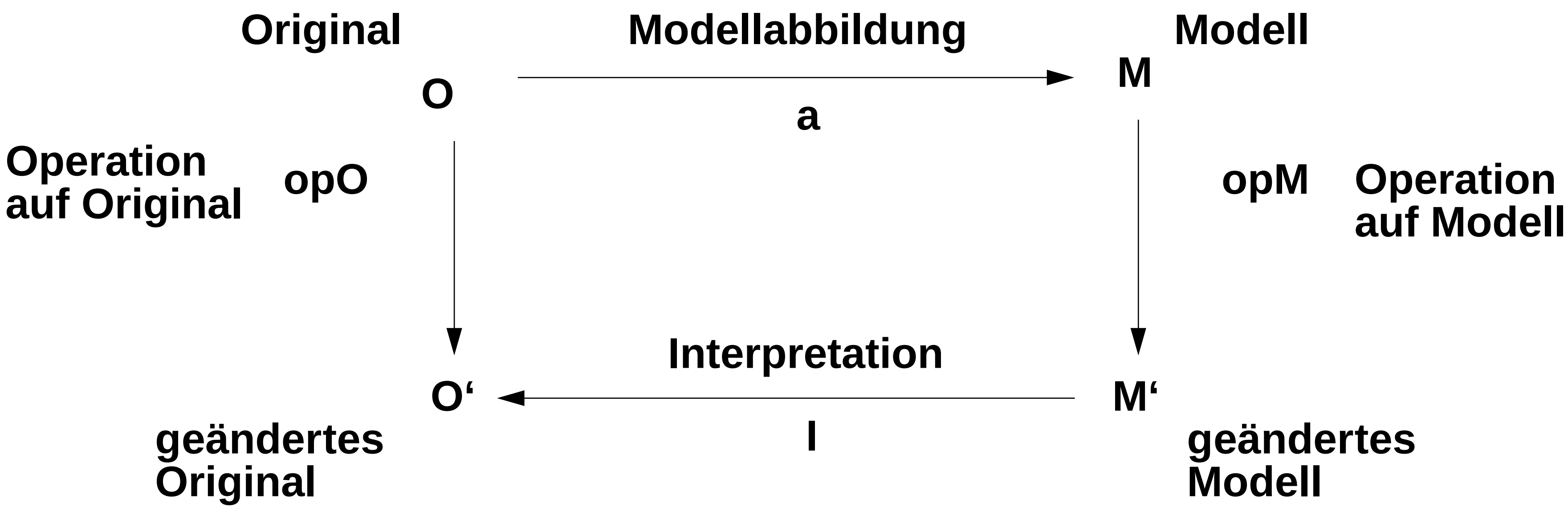
Für den Zweck die jeweils passende Modellierungsmethode (Kalkül) verwenden!

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



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:

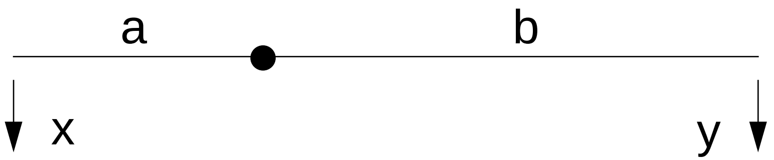
- **Struktur**: Wertebereiche, Entity-Relationship, KFG, Klassifikation, Typen
- **Eigenschaften**: Logik, Relationen
- **Beziehungen**: Graphen, Relationen, Logik, Entity-Relationship
- **Verhalten**: endliche Automaten, Petri-Netze, Algebren, Graphen

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:
Erst lege ich auf den Balken der Länge a ein Gewicht x; dann lege ich auf den Balken der Länge b ein Gewicht y = x * a / b; danach ist dieWaage wieder im Gleichgewicht.

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 Modellierung mit Wertebereichen

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; welche 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
- kartesische Produkte, Tupel
- Folgen
- Relationen
- Funktionen
- disjunkte Vereinigungen

Verwendung des Kalküls:

Modellierung von Strukturen und Zusammenhängen

Grundlage für alle anderen formalen Kalküle

abstrakte Grundlage für Typen in Programmiersprachen

Einführendes Beispiel

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]

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]

Wertebereiche für das Beispiel

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**“.

2.1 Mengen

Menge: Zusammenfassung von verschiedenen Objekten, den Elementen der Menge M.
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)**: $M := \{ a \mid a \in \mathbb{N}, \text{a ist Quadratzahl und } a \leq 30\}$
  allgemein: $M := \{ a \mid P(a) \}$
  wobei P (a) eine Aussage über a ist, die wahr oder falsch sein kann.

Mengen können **endlich** (z. B. {1, 4, 9, 16, 25}) oder
nicht-endlich sein (z. B. $\{ a \mid a \in \mathbb{N}, \text{a ist Quadratzahl} \}$)

Die **leere Menge** wird { } oder $\emptyset$ geschrieben.

Die **Anzahl der Elemente** einer Menge M heißt die **Kardinalität** von M,
geschrieben $|M|$ oder Card(M)

Eigenschaften von Mengen

Wichtige grundsätzliche Eigenschaften von Mengen:

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

$\{ 1, 2, 3, 4 \}$ $\{ i \mid i \in \mathbb{N}, 0 < i < 5 \}$ $\{ 1, 1, 2, 2, 3, 4 \}$ $\{ 2, 4, 1, 3 \}$

Mengen können aus **atomaren oder zusammengesetzten** Elementen gebildet werden,
z. B. nur atomare Elemente: { 1, 2, 3, 4 } { rot, gelb, blau } { Kreuz, Pik, Herz, Karo } { 1 }
Menge von Paaren: { (Pik, 10), (Herz, Dame) }
Menge von Mengen: { {rot, blau}, {blau}, ∅ } { ∅ }

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

Eine Menge kann auch **verschiedenartige Elemente** enthalten,
z. B. {1, (Pik, 10), rot, 9}
aber **nicht bei der Modellierung mit Wertebereichen:** hier sollen alle Elemente eines Wertebereiches gleichartig sein.

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, die 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 „$a \in \mathbb{N}$“.

Damit tatsächlich entschieden werden kann, **welche Elemente M enthält**, muss die Bedingung über a (hier „a ist Quadratzahl und $a \leq 30$“) **entscheidbar** sein.

Diese Einschränkungen schließen nicht aus, Mengen **rekursiv zu definieren**, z. B.

$$\text{Sonnensystem} := \{\text{Sonne}\} \cup \{ x \mid x \in \text{Himmelskörper}, x \text{ umkreist } y, y \in \text{Sonnensystem} \}$$

Mengenoperationen

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	$M \setminus N$	$:= \{x \mid x \in M \text{ und } x \notin N\}$

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

2.2 Potenzmengen

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

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

Beispiele:

Grundmenge U_1 := {a, b} Potenzmenge Pow (U_1) = {∅, {a}, {b}, {a, b}}

Grundmenge U_2 := {1, 2, 3} Pow (U_2) = { ∅, {1}, {2}, {3}, {1,2}, {1,3}, {2,3}, {1,2,3} }

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

Modellierung mit Potenzmengen

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};
```

2.3 Kartesische Produkte

Kartesisches Produkt der Mengen M und N:
Menge **aller geordneten Paare** mit erster Komponente aus M und zweiter Komponente aus N

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

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

Enthält **alle Kombinationen** von Werten aus M und N.
Falls $M = \emptyset$ oder $N = \emptyset$, ist $M \times N = \emptyset$.

z. B. Delegierte := Nation × Ind = { (A, 1), (A, 2),..., (B, 1), (B, 2), ... }

Verallgemeinert zu **n-Tupeln** (n>1, geordnet):

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

z. B. Daten := Tage × Monate × Jahre, $(24, 10, 2011) \in$ Daten

Folgende Wertebereiche sind verschieden. Ihre Elemente haben **unterschiedliche Struktur**:

$(a, b, c) \in A \times B \times C$ $\qquad$ $((a, b), c) \in (A \times B) \times C$

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

Beispiel:

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

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

2.4 Disjunkte Vereinigung

Die allgemeine **disjunkte Vereinigung** fasst n Wertebereiche (Mengen) A_1, A_2, ...A_i, ..., A_n zu einem **vereinigten Wertebereich V** zusammen, wobei $i \in I := \{1, ..., n\}$.
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** (engl. 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, er abstrahiert von den spezielleren A_i

Beispiele:

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

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

Zeichen = { (Buchstabe, b) | b ∈ Buchstaben} ∪ { (Ziffer, z) | z ∈ Ziffern}

2.5 Folgen

Endliche Folgen von Elementen aus A:

Sei $\mathbf{A^0} := \{ \varepsilon \}$ die Menge, die **nur die leere Folge** über A, ε **bzw. (),** enthält,

$\mathbf{A^1} := \{ (a) \mid a \in A\}$ die Menge **einelementiger Folgen** über A,

$\mathbf{A^n}$ mit $n > 1$ die Menge der **n-Tupel** über A,

dann ist $\mathbf{A^+} := \{ x \mid x \in A^{\mathbf{i}} \text{ und } i \geq 1 \}$
die Menge der **nicht-leeren Folgen** beliebiger Länge über A

und $\mathbf{A^*} := A^+ \cup A^0$ die Menge von Folgen über A,
die **auch die leere Folge** enthält.

Folgen notieren wir wie Tupel, d. h. $(a_1, ..., a_n) \in A^+$ für $n \geq 1$ und $a_i \in A$; $() \in A^*$

Beispiele:

$(1, 1, 2, 5, 5, 10, 20) \in \mathbb{N}^{\mathbf{+}}$
$(m, o, d, e, l, l) \in \text{Buchstaben}^{\mathbf{+}}$
neueAufträge := Auftrag$^{\mathbf{*}}$
gezogeneKarten := Karten*

2.6 Relationen

Relationen sind Teilmengen aus kartesischen Produkten.

n-stellige Relation: $R \subseteq M_1 \times M_2 \times \ldots \times M_n$ mit $n > 1$

R ist also eine Menge von n-Tupeln.

Wertebereich von R: $R \in Pow\ (M_1 \times M_2 \times \ldots \times M_n)$

Eine **1-stellige Relation** R über einer Menge M ist eine Teilmenge von M,
also $R \in Pow\ (M)$.

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

Beispiele:

Relation $\leq\ \subseteq \mathbb{N} \times \mathbb{N}$ ein Element daraus: $(27, 42) \in\ \leq$ also gilt $27 \leq 42$

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

Menüs22-10 := {(Lauchsuppe, Putenbraten, Eisbecher),
(Lauchsuppe, Kalbsteak, Ananas), (Salat, Omelett, Ananas)}

Menüs22-10 $\subseteq$ Vorspeisen $\times$ Hauptgerichte $\times$ Desserts mit

Vorspeisen := {Lauchsuppe, Salat, ...};
Hauptgerichte := {Putenbraten, Kalbsteak, Omelett, ...}
Desserts := {Eisbecher, Ananas, Schokoladenpudding, ...}

Kardinalität, Schreibweisen

Der **Wertebereich Pow ($M_1 \times M_2 \times \ldots \times M_n$)** hat die **Kardinalität**

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

d.h. es gibt $2^{\prod_{i \in I} |Mi|}$ verschiedene Relationen in dem Wertebereich.

Intensionale Definition einer Relation:

GültigeDaten $\subseteq$ Daten = Tage $\times$ Monate $\times$ Jahre

GültigeDaten := { (t, m, j) | t, m, j $\in \mathbb{N}$, m$\leq$12,
(m$\in$ {1,3,5,7,8,10,12} $\wedge$ t $\leq$ 31) $\vee$
(m$\in$ {4,6,9,11} $\wedge$ t $\leq$ 30) $\vee$
(m = 2 $\wedge$ t $\leq$ 29 $\wedge$ Schaltjahr (j)) $\vee$
(m = 2 $\wedge$ t $\leq$ 28 $\wedge \neg$ Schaltjahr (j))}

(24, 10, 2011), (29, 2, 2012) $\in$ GültigeDaten, (31, 4, 2010) $\notin$ GültigeDaten

alternative Schreibweisen für Elemente aus Relationen:
R(a) für a $\in$ R, z. B. GültigeDaten(24, 10, 2011)

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 $R \subseteq M \times M$ mit $M \neq \emptyset$ 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 $x\ R\ z$;
- **total**, wenn für alle $x, y \in M$ gilt: $x\ R\ y$ oder $y\ R\ x$;

Hinweise zum Anwenden der Definitionen (genauer in Kap. 4.1, 4.2):

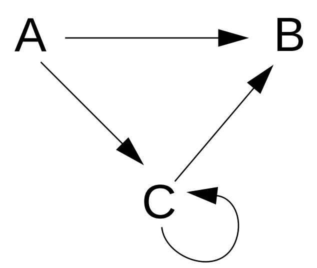
1. „**x R y**“ bedeutet „**(x, y)** $\in$ **R**“
2. **„**für alle $x \in M$ gilt ...“: der **gesamte Wertebereich M** muss geprüft werden
3. „für alle $x, y \in M$ gilt ...“: alle Paare von Werten aus M prüfen, auch solche mit $x = y$
4. „A **oder** B“ ist wahr, wenn **mindestens eins von beiden wahr**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

Beispiele für Eigenschaften 2-stelliger Relationen

Eigenschaft ist	sei M = { A, B, C } z.B. erfüllt von R = ...	z.B. nicht erfüllt von R = ...
reflexiv	{(A,A), (B,B), (C,C), (A,B)}	{(A,A), (B,C)}
irreflexiv	{(A,B)}	{(A,A)}
symmetrisch	{(A,B), (B,A), (C,C)}	{(A,B)}
antisymmetrisch	{(A,B), (C,C)}	{(A,B), (B,A)}
asymmetrisch	{(A,B), (C,A)}	{(A,B), (B,A)} oder {(C,C)}
transitiv	{(A,B), (B,C), (A,C)}	{(A,B), (B,C)}
total	{(A,A), (B,B), (C,C), (A,B), (B,C), (A,C), (C,B)}	{(A,A), (A,B), (A,C)}

{(A,B), (A,C), (C,B), (C,C)}

als gerichteter Graph:
(siehe Kap. 5)



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 totale Halbordnung ist,
  also total, (reflexiv,) antisymmetrisch und transitiv;
- **Äquivalenzrelation**, wenn R reflexiv, symmetrisch und transitiv ist.

Aussagen zu diesen Definitionen

1. Alle solche Ordnungsrelationen sind transitiv.
2. Ist R eine totale Ordnung, dann ist R auch eine Halbordnung und eine Quasiordnung.
3. Nur für totale Ordnungen wird gefordert, dass alle Elemente aus M „vergleichbar“ sind (total).
4. Enthält R „Zyklen über verschiedene Elemente“, z.B. $(a, b), (b, a) \in R$ mit $a \neq b$,
   dann ist R weder eine Halbordnung, strenge Halbordnung, noch eine totale Ordnung.

Beispiele für Ordnungsrelationen

sei $M = \{A, B, C\}$,
$eq_M := \{(A,B), (B,A), (A,A), (B,B), (C,C)\}$
$<_M := \{(A,B), (B,C), (A,C)\}$,
$\leq_M := \{(A,B), (B,C), (A,C), (A,A), (B,B), (C,C)\}$

$\leq \subseteq \mathbb{N} \times \mathbb{N}$, $\quad < \subseteq \mathbb{N} \times \mathbb{N}$

	$<_M$	$\leq_M$	eq_M	$<$	$\leq$
reflexiv	**-**	**+**	**+**	**-**	**+**
irreflexiv	**+**	**-**	**-**	**+**	**-**
symmetrisch	**-**	**-**	**+**	**-**	**-**
antisymmetrisch	**+**	**+**	**-**	**+**	**+**
asymmetrisch	**+**	**-**	**-**	**+**	**-**
transitiv	**+**	**+**	**+**	**+**	**+**
total	**-**	**+**	**-**	**-**	**+**
	strenge Ordnung	**totale**	**Äquivalenz**	**strenge**	**totale**

2.7 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 x $\in$ D gibt es höchstens 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.

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 aller Paare (x, y) $\in$ f heißt **Graph von f.**

Eine Funktion f $\in$ D -> B heißt
n-stellig, wenn der Definitionsbereich D ein Wertebereich von n-Tupeln ist, n > 1;
1-stellig, wenn D nicht als kartesisches Produkt strukturiert ist und nicht leer ist.

Man spricht auch von **0-stelligen Funktionen**, wenn D der **leere Wertebereich** ist;
0-stellige Funktionen sind **konstante Funktionen** für jeweils einen **festen Wert b = f()**;
man kann sie allerdings nicht als Menge von Paaren angeben.

Beispiele für Funktionen

Funktion	**aus dem Wertebereich**
not := {(w, f), (f, w)}	Bool -> Bool
id := {(w,w), (f, f)}	Bool -> Bool
oder := {((w,w),w), ((w,f),w), ((f,w),w), ((f,f),f)}	Bool $\times$ Bool -> Bool
Quadrat := {(a, b) | a, b $\in$ $\mathbb{N}$ 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 nicht verlangt wird, dass f für alle $x \in D$ definiert ist,
- **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 surjektiv und injektiv ist.

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

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

... falls D und B endlich sind.

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_M := { (x, x) | x $\in$ M }

Charakteristische Funktion χ_M einer Menge M $\subseteq$ U, mit der Trägermenge U

gibt für jedes Element der Trägermenge U an, ob es in M enthalten ist:

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

χ_M ist eine totale Funktion

Funktionen mit dem Bildbereich Bool heißen **Prädikate**.

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

Funktionen zur Modellierung von mehrfachen Vorkommen

In sogenannte **Multimengen (engl. bags)** können einige Werte mehrfach vorkommen.
Es ist relevant, wieoft jeder Wert vorkommt.

Das **mehrfache Vorkommen** von Werten in einer Multimenge modellieren wir mit einer Funktion:

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

geldBeutel $\in$ EUMünzen -> $\mathbb{N}_0$ mit

geldBeutel := {(1,3), (2, 0), (5,0), (10, 2), (20, 4), (50, 1), (100, 3), (200, 2)}

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:

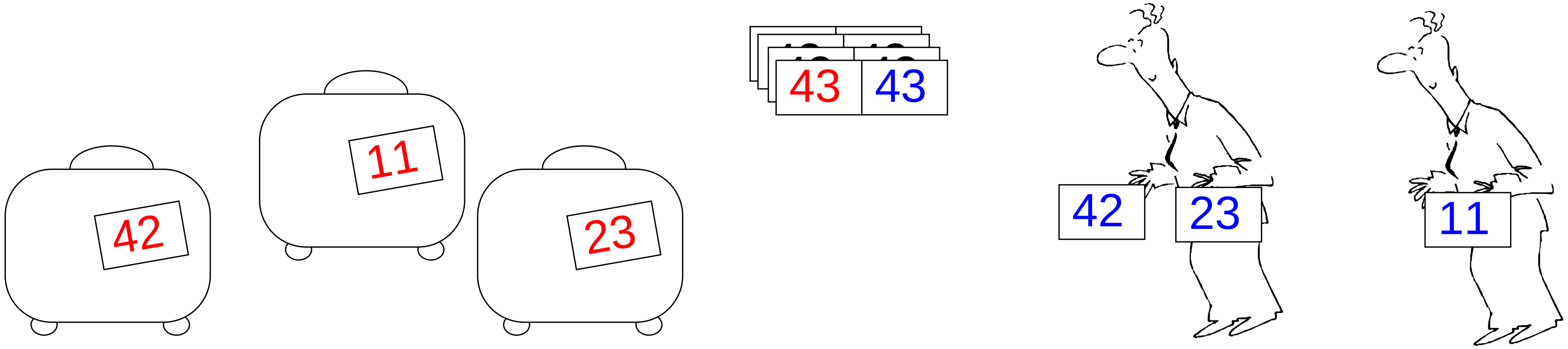
Beispiel:

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 ∈ FPositionen -> FWerte	

Zuordnungen zwischen Mengen:

z. B. Gepäckstücke ihren Eigentümern zuordnen durch ein Funktionenpaar

Marke1 ∈ Ind -> Gepäckstücke (injektiv) Marke2 ∈ Ind -> Eigentümer



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 ...

2x Beweise verstehen und konstruieren

Beweise werden in vielen Gebieten der Informatik benötigt

- innerhalb von **Informatik-Theorien**
  z.B. Komplexität von Aufgaben und Algorithmen
- **Eigenschaften von modellierten Aufgaben**
  z.B. Falls ein ungerichteter Graph zusammenhängend ist, gibt es mindestens einen Weg von Knoten a nach Knoten b.
- **Entwurf von Hardware und Software**
  z.B. Diese Synchronisation der „Dining Philosophers“ führt nie zur Verklemmung.
- **Eigenschaften implementierter Software oder Hardware**
  Verifikation von Programmeigenschaften

Dieses Thema wird im Buch „Modellierung“ im Abschnitt 4.3 behandelt.

Beispiel 1

Satz 2x.1:

Seien A und B zweistellige, antisymmetrische Relationen über der Menge M. Dann ist $C = A \cup B$ auch eine antisymmetrische Relation.

Beweis:

Wegen der Definition von antisymmetrisch gilt:
Für alle $x, y \in M$ gilt: Aus x A y und y A x folgt $x = y$.

Ebenso gilt:
Für alle $x, y \in M$ gilt: Aus x B y und y B x folgt $x = y$.

Wegen $C = A \cup B$ sind alle Elemente aus A oder B auch Elemente von C und es gilt:
Für alle $x, y \in M$ gilt: Aus x C y und y C x folgt $x = y$.

Also ist auch C antisymmetrisch. **qed.**

Gegenbeispiel

Der Satz 2x.1

Seien A und B zweistellige, antisymmetrische Relationen über der Menge M. Dann ist $C = A \cup B$ auch eine antisymmetrische Relation.

ist nicht korrekt. Man kann ihn durch ein **Gegenbeispiel widerlegen**:

z.B. A = {(a, a), (b, c)}, B = {(d, d), (c, b)}, C = {(a, a), (b, c), (d, d), (c, b)}

Der „Beweis" von Satz 2x.1 ist fehlerhaft.

Beispiel 2

Satz 2x.2:

Seien A und B zweistellige, symmetrische Relationen über der Menge M. Dann ist $C = A \cup B$ auch eine symmetrische Relation.

Beweis:

Sind A und B leer, dann ist auch C leer und ist gemäß Definition symmetrisch.

Ist C nicht leer, dann sei x C y für beliebige x und y.

Wegen $C = A \cup B$ gilt x A y oder x B y.

Falls x A y gilt, dann ist auch y A x, weil A symmetrisch ist.
Wegen $C = A \cup B$ ist auch y C x.

Falls x B y gilt, dann ist auch y B x, weil B symmetrisch ist.
Wegen $C = A \cup B$ ist auch y C x.

Also folgt aus x C y auch y C x. Deshalb ist auch C symmetrisch.

qed.

Eigenschaften von Beweisen

Beweise können

- korrekt oder fehlerhaft,
- verständlich oder unverständlich,
- elegant oder umständlich,
- wohl-strukturiert oder verschlungen

sein.

Zur Konstruktion von Beweisen gibt es

- **Regeln, Methoden, Strukturen, Strategien.**

Dazu wird in diesem Abschnitt eingeführt.
Erst Kapitel 4 liefert die notwendigen Grundlagen der Logik.
Das Buch [D. J. Velleman: How to prove it] enthält umfassendes Material zu diesem Thema.

Manche Beweise benötigen außerdem eine gute Beweisidee.

Form von Satz und Beweis

Ein **Satz (Theorem)** besteht aus
Voraussetzungen (Prämissen) und einer **Behauptung (Konklusion)**.

Voraussetzungen und Behauptung sind Aussagen.
Wenn alle Voraussetzungen wahr sind, dann muss auch die Behauptung wahr sein.

Satz 2x.2:
Seien A und B zweistellige, symmetrische Relationen über der Menge M.
Dann ist $C = A \cup B$ auch eine symmetrische Relation.

Der **Beweis** eines Satzes muss nachweisen, dass die Behauptung wahr ist und kann dabei verwenden

- die Voraussetzungen,
- Definitionen oder bekannte Tatsachen,
- im Beweis selbst oder anderweitig als wahr bewiesene Aussagen,
- Schlussregeln.

Beweisstruktur Fallunterscheidung

Beweise können in **Fallunterscheidungen** gegliedert sein. Typische Gründe dafür:

- **Sonderfall** abspalten (z.B. leer, nicht leer)
- ***oder* in der Voraussetzung** (z.B. $(x, y) \in C = A \cup B$ bedeutet $(x, y) \in A$ ***oder*** $(x, y) \in B$)
- ***und* in der Behauptung** (Beispiel später)

leer

nicht leer

$(x, y) \in A$

$(x, y) \in B$

Beweis 2x.2:

Sind A und B leer, dann ist auch C leer und ist gemäß Definition symmetrisch.

Ist C nicht leer, dann sei x C y für beliebige x und y.

Wegen $C = A \cup B$ gilt x A y oder x B y.

Falls x A y gilt, dann ist auch y A x, weil A symmetrisch ist.
Wegen $C = A \cup B$ ist auch y C x.

Falls x B y gilt, dann ist auch y B x, weil B symmetrisch ist.
Wegen $C = A \cup B$ ist auch y C x.

Also folgt aus x C y auch y C x.

Deshalb ist auch C symmetrisch. **qed.**

Implikation als Behauptung

Satz 2x.3:
Sei R eine zweistellige Relation über der Menge M.
Wenn **a R b und b R a mit a ≠ b**, dann ist R **weder eine Halbordnung (HO)**, **noch eine strenge Halbordnung (sHO)**, **noch eine totale Ordnung (tO)**.

Die **Behauptung des Satzes** hat die Form

P **impliziert** (Q1 und Q2 und Q3)
(a R b und b R a mit a ≠b) **impliziert** (**nicht** HO **und nicht** sHO **und nicht** tO)

Hier kann man zwei **Techniken zur Gliederung des Beweises** anwenden:

- **Behauptung P impliziert Q: füge P zu den Voraussetzungen und beweise Q.**
- **Behauptung Q_1 und Q_2 und ...: beweise jedes Q_i in einem einzelnen Fall.**

Damit bekommt der Beweis 2x.3 folgende Struktur:

Beweis 2x.3:
Wir nehmen an, es gelte P = (a R b und b R a mit a ≠b)
Beweis aus Voraussetzung und P folgt nicht HO
Beweis aus Voraussetzung und P folgt nicht sHO
Beweis aus Voraussetzung und P folgt nicht tO
also aus P folgt (nicht HO und nicht sHO und nicht tO)

Beweisstruktur ausfüllen

Beweis 2x.3:
Wir nehmen an, es gelte **a R b und b R a mit a ≠ b** für die zweistellige Relation R über der Menge M.

1. Dann verletzen a R b und b R a die Definition für Antisymmetrie. Also ist R **nicht eine Halbordnung**.
2. Da R gemäß (1) nicht antisymmetrisch ist, ist R auch **nicht eine totale Ordnung**.
3. Gemäß Satz 2x.4 (Mod-2.61) ist **R nicht eine strenge Halbordnung**.

Also folgt aus **a R b und b R a mit a ≠ b,** dass R **weder eine Halbordnung**, **noch eine strenge Halbordnung**, **noch eine totale Ordnung** ist. **qed.**

Konstruktionshilfen am Beispiel für Beweis 2x.3

gültige Aussagen:

$R \in Pow(M \times M)$

Behauptungen:

$a\, R\, b \wedge b\, R\, a \wedge a \neq b$
$\rightarrow (\neg HO \wedge \neg sHO \wedge \neg tO)$

Beweisstruktur:

Konstruktionshilfen am Beispiel für Beweis 2x.3

gültige Aussagen:

$R \in Pow(M \times M)$

$a\ R\ b \wedge b\ R\ a \wedge a \neq b$ (wg. Implik. in Beh.)

Behauptungen:

$\cancel{a\ R\ b \wedge b\ R\ a \wedge a \neq b \rightarrow (\neg HO \wedge \neg sHO \wedge \neg tO)}$

$\neg HO \wedge \neg sHO \wedge \neg tO$

Beweisstruktur:

Wir nehmen an, es gelte $Z = (a\ R\ b \wedge b\ R\ a \wedge a \neq b)$

Beweis aus Voraussetzung und Z folgt $\neg HO \wedge \neg sHO \wedge \neg tO$

also aus Z folgt (nicht HO und nicht sHO und nicht tO)

Konstruktionshilfen am Beispiel für Beweis 2x.3

gültige Aussagen:

$R \in Pow(M \times M)$

$a\ R\ b \wedge b\ R\ a \wedge a \neq b$ (wg. Implik. in Beh.)

Behauptungen:

$\cancel{a\ R\ b \wedge b\ R\ a \wedge a \neq b \rightarrow (\neg HO \wedge \neg sHO \wedge \neg tO)}$

$\cancel{\neg HO \wedge \neg sHO \wedge \neg tO}$

$\neg$ HO (3 Fälle wg. Konjunktion)

$\neg$ sHO

$\neg$ tO

Beweisstruktur:

Wir nehmen an, es gelte $Z = (a\ R\ b \wedge b\ R\ a \wedge a \neq b)$

~~Beweis aus Voraussetzung und Z folgt $\neg HO \wedge \neg sHO \wedge \neg tO$~~

Beweis aus Voraussetzung und Z folgt nicht HO

Beweis aus Voraussetzung und Z folgt nicht sHO

Beweis aus Voraussetzung und Z folgt nicht tO

also aus Z folgt (nicht HO und nicht sHO und nicht tO)

Konstruktionshilfen am Beispiel für Beweis 2x.3

gültige Aussagen:

$R \in Pow(M \times M)$

$a\ R\ b \wedge b\ R\ a \wedge a \neq b$ (wg. Implik. in Beh.)

R nicht antisymmetrisch (wg. Def. antisy.)

Behauptungen:

$\cancel{a\ R\ b \wedge b\ R\ a \wedge a \neq b \rightarrow (\neg HO \wedge \neg sHO \wedge \neg tO)}$

$\cancel{\neg HO \wedge \neg sHO \wedge \neg tO}$

$\neg HO$ (3 Fälle wg. Konjunktion)

$\neg sHO$

$\neg tO$

Beweisstruktur:

Wir nehmen an, es gelte $Z = (a\ R\ b \wedge b\ R\ a \wedge a \neq b)$

~~Beweis aus Voraussetzung und Z folgt $\neg HO \wedge \neg sHO \wedge \neg tO$~~

Beweis aus Voraussetzung und Z folgt nicht HO

Beweis aus Voraussetzung und Z folgt nicht sHO

Beweis aus Voraussetzung und Z folgt nicht tO

also aus Z folgt (nicht HO und nicht sHO und nicht tO)

Konstruktionshilfen am Beispiel für Beweis 2x.3

gültige Aussagen:

$R \in Pow(M \times M)$

$a R b \wedge b R a \wedge a \neq b$ (wg. Implik. in Beh.)

R nicht antisymmetrisch (wg. Def. antisy.)

R ist nicht Halbordnung (wg. Def. HO)

Behauptungen:

$\cancel{a R b \wedge b R a \wedge a \neq b \rightarrow (\neg HO \wedge \neg sHO \wedge \neg tO)}$

$\cancel{\neg HO \wedge \neg sHO \wedge \neg tO}$

~~$\neg$ HO (3 Fälle wg. Konjunktion)~~

$\neg$ sHO

$\neg$ tO

Beweisstruktur:

Wir nehmen an, es gelte $Z = (a R b \wedge b R a \wedge a \neq b)$

~~Beweis aus Voraussetzung und Z folgt $\neg HO \wedge \neg sHO \wedge \neg tO$~~

Beweis aus Voraussetzung und Z folgt nicht HO

Beweis aus Voraussetzung und Z folgt nicht sHO

Beweis aus Voraussetzung und Z folgt nicht tO

also aus Z folgt (nicht HO und nicht sHO und nicht tO)

Konstruktionshilfen am Beispiel für Beweis 2x.3

gültige Aussagen:

$R \in Pow(M \times M)$

$a\,R\,b \wedge b\,R\,a \wedge a \neq b$ (wg. Implik. in Beh.)

R nicht antisymmetrisch (wg. Def. antisy.)

R ist nicht Halbordnung (wg. Def. HO)

R ist nicht totale Ordnung (wg. Def. tO.)

Behauptungen:

~~$a\,R\,b \wedge b\,R\,a \wedge a \neq b \rightarrow (\neg HO \wedge \neg sHO \wedge \neg tO)$~~

~~$\neg HO \wedge \neg sHO \wedge \neg tO$~~

~~$\neg HO$~~ (3 Fälle wg. Konjunktion)

$\neg sHO$

~~$\neg tO$~~

Beweisstruktur:

Wir nehmen an, es gelte $Z = (a\,R\,b \wedge b\,R\,a \wedge a \neq b)$

~~Beweis aus Voraussetzung und Z folgt $\neg HO \wedge \neg sHO \wedge \neg tO$~~

Beweis aus Voraussetzung und Z folgt nicht HO

Beweis aus Voraussetzung und Z folgt nicht sHO

Beweis aus Voraussetzung und Z folgt nicht tO

also aus Z folgt (nicht HO und nicht sHO und nicht tO)

Konstruktionshilfen am Beispiel für Beweis 2x.3

gültige Aussagen:

$R \in Pow(M \times M)$

$a\,R\,b \wedge b\,R\,a \wedge a \neq b$ (wg. Implik. in Beh.)

R nicht antisymmetrisch (wg. Def. antisy.)

R ist nicht Halbordnung (wg. Def. HO)

R ist nicht totale Ordnung (wg. Def. tO.)

nicht sHO wird separat bewiesen (2x.4)

Behauptungen:

~~$a\,R\,b \wedge b\,R\,a \wedge a \neq b \rightarrow (\neg HO \wedge \neg sHO \wedge \neg tO)$~~

~~$\neg HO \wedge \neg sHO \wedge \neg tO$~~

~~$\neg HO$ (3 Fälle wg. Konjunktion)~~

~~$\neg sHO$~~

~~$\neg tO$~~

Beweisstruktur:

Wir nehmen an, es gelte $Z = (a\,R\,b \wedge b\,R\,a \wedge a \neq b)$

~~Beweis aus Voraussetzung und Z folgt $\neg HO \wedge \neg sHO \wedge \neg tO$~~

Beweis aus Voraussetzung und Z folgt nicht HO

Beweis aus Voraussetzung und Z folgt nicht sHO

Beweis aus Voraussetzung und Z folgt nicht tO

also aus Z folgt (nicht HO und nicht sHO und nicht tO)

Konstruktionshilfen am Beispiel für Beweis 2x.3

gültige Aussagen:

$R \in Pow(M \times M)$

$a\,R\,b \wedge b\,R\,a \wedge a \neq b$ (wg. Implik. in Beh.)

R nicht antisymmetrisch (wg. Def. antisy.)

R ist nicht Halbordnung (wg. Def. HO)

R ist nicht totale Ordnung (wg. Def. tO.)

nicht sHO wird separat bewiesen (2x.4)

Behauptungen:

$\cancel{a\,R\,b \wedge b\,R\,a \wedge a \neq b \rightarrow (\neg HO \wedge \neg sHO \wedge \neg tO)}$

$\cancel{\neg HO \wedge \neg sHO \wedge \neg tO}$

~~¬ HO (3 Fälle wg. Konjunktion)~~

$\cancel{\neg sHO}$

$\cancel{\neg tO}$

Beweisstruktur:

Wir nehmen an, es gelte $Z = (a\,R\,b \wedge b\,R\,a \wedge a \neq b)$

~~Beweis aus Voraussetzung und Z folgt ¬ HO ∧ ¬ sHO ∧ ¬ tO~~

Beweis aus Voraussetzung und Z folgt nicht HO

Beweis aus Voraussetzung und Z folgt nicht sHO

Beweis aus Voraussetzung und Z folgt nicht tO

also aus Z folgt (nicht HO und nicht sHO und nicht tO)

abschließend Beweistext zusammensetzen

Methode: Beweis durch Widerspruch

Ein Beweis durch Widerspruch führt häufig zum Ziel, wenn die Behauptung eine Negation ist:

Satz: Voraussetzung **V**. Behauptung **nicht P**.

Man nimmt dann die **nicht-negierte Behauptung mit als Voraussetzung** auf und leitet mit Schlussregeln daraus einen Widerspruch her, d.h. eine Aussage, die immer falsch ist, z. B. ($x \in M$ und $x \notin M$).

Beweis: Aus **V** und **P** folgt ein **Widerspruch**. Also war die Annahme **P** falsch.
Also gilt **nicht P**. **qed.**

Häufig ist **nicht P** ein geeignetes Ziel für den Widerspruchsbeweis:

Beweis: Aus **V** und **P** folgt **nicht P**. Also gilt **(P und nicht P)**.
Also war die Annahme **P** falsch, also gilt **nicht P**. **qed.**

Beispiel für Beweis durch Widerspruch

Satz 2x.4:

Sei R eine zweistellige Relationen über der Menge M.
Wenn **a R b und b R a mit a $\neq$ b**, dann ist **R nicht eine strenge Halbordnung**.

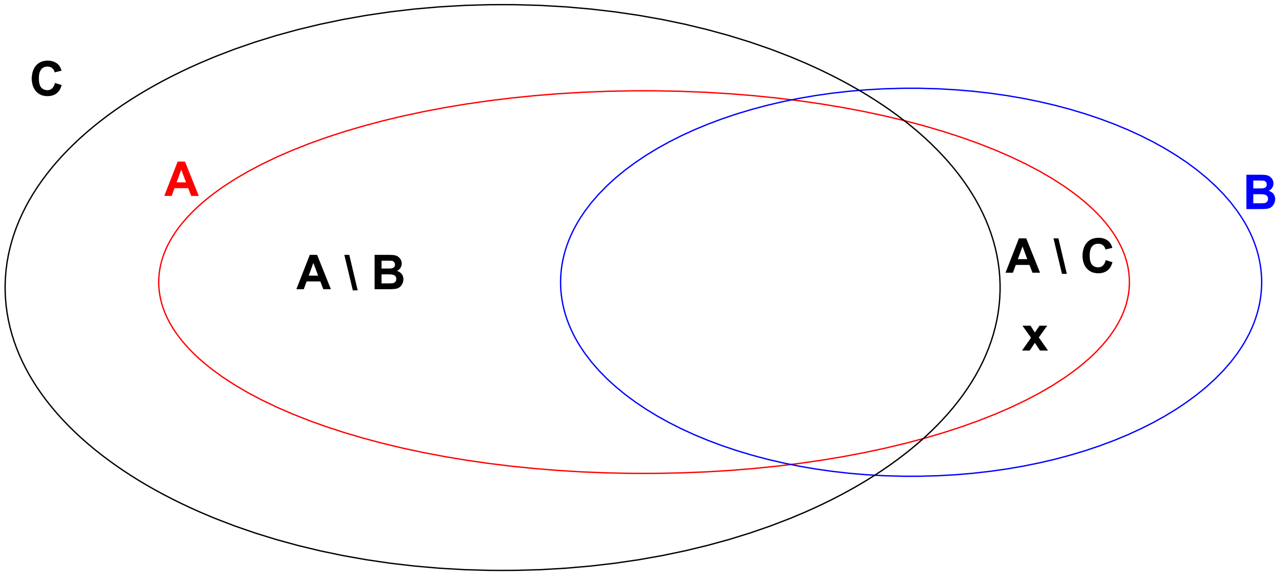
Beweis durch Widerspruch:

Sei **a R b und b R a mit a $\neq$ b**.
Wir nehmen an, dass **R eine strenge Halbordnung** ist.
Dann muss R irreflexiv und transitiv sein.
Wegen der Transitivität folgt aus a R b und b R a auch **a R a und b R b**.
a R a verletzt jedoch die Definition von **Irreflexivität**.
Also ist die Annahme, dass R eine **strenge Halbordnung ist, falsch**.
Also ist **R nicht eine strenge Halbordnung**. **qed.**

Satz 2x.5 zur Konstruktion eines Widerspruchsbeweises

Satz 2x.5:

A, B, C seien Mengen mit $A \setminus B \subseteq C$. **Dann gilt:**
Aus $x \in A \setminus C$ folgt $x \in B$.



Konstruktion eines Widerspruchsbeweises 2x.5

gültige Aussagen:

A, B, C Mengen

$A \setminus B \subseteq C$

Behauptungen:

$x \in A \setminus C \rightarrow x \in B$

Beweisstruktur:

Für die Mengen A, B, C gilt $A \setminus B \subseteq C$.

Konstruktion eines Widerspruchsbeweises 2x.5

gültige Aussagen:

A, B, C Mengen

$A \setminus B \subseteq C$

(es gibt ein $x \in A \setminus C$)

Behauptungen:

$x \in A \setminus C \rightarrow x \in B$ Implikation

$x \in B$

Beweisstruktur:

Für die Mengen A, B, C gilt $A \setminus B \subseteq C$.

Es gibt ein $x \in A \setminus C$.

Beweise $x \in B$.

Konstruktion eines Widerspruchsbeweises 2x.5

gültige Aussagen:

A, B, C Mengen

$A \setminus B \subseteq C$

(es gibt ein $x \in A \setminus C$)

$x \notin B$

Behauptungen:

$x \in A \setminus C \rightarrow x \in B$ — Implikation

$x \in B$ — zeige Widerspruch welchen?

Beweisstruktur:

Für die Mengen A, B, C gilt $A \setminus B \subseteq C$.

Es gibt ein $x \in A \setminus C$.

Beweise $x \in B$.

Wir nehmen an $x \notin B$ und zeigen einen Widerspruch:

Konstruktion eines Widerspruchsbeweises 2x.5

gültige Aussagen:

A, B, C Mengen

$A \setminus B \subseteq C$

(es gibt ein $x \in A \setminus C$)

$x \notin B$

$x \in A$

$x \notin C$

Def. \

Behauptungen:

$x \in A \setminus C \rightarrow x \in B$

$x \in B$

Implikation

zeige Widerspruch

welchen?

Beweisstruktur:

Für die Mengen A, B, C gilt $A \setminus B \subseteq C$.

Es gibt ein $x \in A \setminus C$.

Beweise $x \in B$.

Wir nehmen an $x \notin B$ und zeigen einen Widerspruch:

Wegen $x \in A \setminus C$ gilt $x \in A$ und $x \notin C$.

Konstruktion eines Widerspruchsbeweises 2x.5

gültige Aussagen:

A, B, C Mengen

$A \setminus B \subseteq C$

(es gibt ein $x \in A \setminus C$)

$x \notin B$

$x \in A$

$x \notin C$

$x \in C$ Widerspruch!

Def. \

Behauptungen:

$x \in A \setminus C \rightarrow x \in B$ — Implikation

$x \in B$ — zeige Widerspruch welchen?

Beweisstruktur:

Für die Mengen A, B, C gilt $A \setminus B \subseteq C$.

Es gibt ein $x \in A \setminus C$.

Beweise $x \in B$.

Wir nehmen an $x \notin B$ und zeigen einen Widerspruch:

Wegen $x \in A \setminus C$ gilt $x \in A$ und $x \notin C$.

Wegen $A \setminus B \subseteq C$ und $x \notin B$ und $x \in A$ gilt $x \in C$.

Das ist ein Widerspruch.

Konstruktion eines Widerspruchsbeweises 2x.5

gültige Aussagen:

A, B, C Mengen

$A \setminus B \subseteq C$

(es gibt ein $x \in A \setminus C$)

$x \notin B$

$x \in A$

$x \notin C$

$x \in C$ Widerspruch!

Def. \

Behauptungen:

$x \in A \setminus C \rightarrow x \in B$ — Implikation

$x \in B$ — zeige Widerspruch welchen?

Beweisstruktur:

Für die Mengen A, B, C gilt $A \setminus B \subseteq C$.

Es gibt ein $x \in A \setminus C$.

Beweise $x \in B$.

Wir nehmen an $x \notin B$ und zeigen einen Widerspruch:

Wegen $x \in A \setminus C$ gilt $x \in A$ und $x \notin C$.

Wegen $A \setminus B \subseteq C$ und $x \notin B$ und $x \in A$ gilt $x \in C$.

Das ist ein Widerspruch.

Also ist die Annahme $x \notin B$ falsch; es gilt $x \in B$.

Also, für Mengen A, B, C mit $A \setminus B \subseteq C$ gilt: Aus $x \in A \setminus C$ folgt $x \in B$. **q.e.d.**

Unendlich viele Primzahlen

Satz 2x.6: Es gibt **unendlich viele Primzahlen**.

Beweis durch Widerspruch (nach Euclid) 2x.6:
Wir nehmen an, dass es **endlich viele Primzahlen** gibt, nämlich $\mathbf{p_1, p_2, ..., p_n}$.
Sei $\mathbf{m = p_1 p_2 ... p_n + 1}$.
m ist nicht durch p_1 teilbar, denn m dividiert durch p_1 ergibt $p_2...p_n$ mit Rest 1. Aus demselben Grund ist m nicht durch $p_2, ..., p_n$ teilbar.
Wir verwenden nun die Tatsache, dass jede natürliche Zahl, die größer als 1 ist, entweder **eine Primzahl** ist oder als **Produkt von Primzahlen** geschrieben werden kann. m ist größer als 1, also ist m entweder eine Primzahl oder m ist ein Produkt von Primzahlen.

Nehmen wir an, **m ist eine Primzahl**. m ist größer als jede Zahl $p_1, p_2, ..., p_n$. Also haben wir eine weitere Primzahl gefunden. Das **widerspricht** der Annahme, dass $p_1, p_2, ..., p_n$ **alle** Primzahlen sind.

Nehmen wir nun an, dass **m ein Produkt von Primzahlen** ist. Sei q eine dieser Primzahlen. Dann ist q ein Teiler von m. Da $p_1, p_2, ..., p_n$ nicht Teiler von m sind, haben wir eine weitere Primzahl gefunden. Das ist wie oben ein **Widerspruch**.

Die Annahme, dass es endlich viele Primzahlen gibt, hat zum **Widerspruch** geführt. Also gibt es unendlich viele Primzahlen. **qed.**

Methode: Beweis durch Induktion

Beweise durch Induktion sind geeignet für Aussagen der Form

Für alle n $\in \mathbb{N}_0$ **gilt P (n)** .

Ein Beweis durch Induktion hat folgende Struktur:

Induktionsanfang: Beweis von **P (0)** .

Induktionsschritt: Sei n $\in \mathbb{N}_0$ beliebig aber fest.
Beweis von **Aus P (n) folgt P (n+1)**. **qed.**

Manchmal reicht im Beweis des Induktionsschrittes P (n) als Vorbedingung nicht aus.
Dann kann man in der folgenden Variante P (0), P (1), ..., P (n) verwenden:

Variante des Induktionsbeweises:

Induktionsanfang: Beweis von **P (0)** .

Induktionsschritt: Sei n $\in \mathbb{N}_0$ beliebig aber fest.
Beweis von **Aus [P (0), P (1), ..., P (n)] folgt P (n+1)**. **qed.**

Zum Beweis von Aussagen der Form **Für alle n** $\in \mathbb{N}_0$ **, n** $\geq$ **k gilt P (n)** beginnt man im Induktionsanfang mit **P (k) statt P (0)**.

Statt *Beweis durch Induktion* sagt man auch *Beweis durch vollständige Induktion*.

Beispiel für Beweis durch Induktion

Satz 2x.7:

Für alle n $\in \mathbb{N}_0$ **gilt** $\mathbf{2^0 + 2^1 + ... + 2^n = 2^{n+1} - 1}$.

Beweis durch Induktion:

Induktionsanfang:

Für $n = 0$ gilt $2^0 = 1 = 2^1 - 1$.

Induktionsschritt:

Sei $n \in \mathbb{N}_0$ beliebig aber fest und

sei $\mathbf{2^0 + 2^1 + ... + 2^n = 2^{n+1} - 1}$. Dann ist

$$
\begin{aligned}
2^0 + 2^1 + ... + 2^n + 2^{n+1} &= (\mathbf{2^0 + 2^1 + ... + 2^n}) + 2^{n+1} \\
&= (2^{n+1} - 1) + 2^{n+1} \\
&= 2 * 2^{n+1} - 1 \\
&= \mathbf{2^{n+2} - 1}
\end{aligned}
$$

qed.

Zusammenfassung

Satzform: Voraussetzungen V. Behauptung B.

Beweismethoden:

Direkter Beweis:
Aus **V** und bewiesenen Tatsachen mit Schlussregeln **B** nachweisen.

Widerspruchsbeweis:
Nicht **B** annehmen. Aus **V** *und nicht* **B** einen Widerspruch ableiten. Also gilt **B**.

Induktionsbeweis von Behauptung B = **Für alle n** $\in \mathbb{N}_0$ **gilt P (n):**
Induktionsanfang: Beweis von P (0),
Induktionsschritt: Beweis von Aus P (n) folgt P (n+1)

Techniken:

Fallunterscheidung bei **Sonderfällen**, V_1 ***oder*** V_2, B_1 ***und*** B_2

Wenn B = P ***impliziert*** Q, dann aus V *und* P die Behauptung Q folgern.

Viele weitere Strategien, Techniken und Beispiele im Buch von Velleman, z.B.

Wenn B = P ***impliziert*** Q, dann aus V *und nicht* Q die Behauptung *nicht* P folgern.

3 Terme und Algebren
3.1 Terme

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 Signaturen

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**.

Eine **Sorte s** $\in$ **S** ist ein **Name für eine Menge von Termen**, z. B. ARITH, BOOL;
verschiedene Namen benennen disjunkte Mengen

Eine **Operation f** $\in$ **F** ist ein **Operatorsymbol**, beschrieben durch
Anzahl der Operanden (**Stelligkeit)**,
Sorten der Operanden und **Sorte des Ergebnisses**

0-stellige Operatoren sind Konstante, z. B. true, 1

Beispiele:

einzelne Operatoren:

Name	**Operandensorten**	**Ergebnissorte**
+:	ARITH x ARITH	-> ARITH
<:	ARITH x ARITH	-> BOOL
$\wedge$:	BOOL x BOOL	-> BOOL
true:		-> BOOL
1:		-> ARITH

Signatur Σ_{BOOL} := (S_{BOOL}, F_{BOOL})

S_{BOOL} := { BOOL },
F_{BOOL} :=
{ true: -> BOOL,
false: -> 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.

Induktive Definition der **Menge τ der korrekten Terme der Sorte s zur Signatur Σ = (S, F)**:
Sei die Signatur $\Sigma = (S, F)$. Dann ist t ein **korrekter Term der Sorte $s \in S$**, wenn gilt

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

f ($t_1, \ldots, t_n$) 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}:

false $\quad \neg$ true $\quad$ true $\wedge$ 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 $f\ (t_1, t_2, ..., t_n)$	 $\wedge (< (0, a), \neg (< (a, 10)))$
Präfixform:	Operator **vor** seinen Operanden $f\ \ t_1\ \ t_2 ...\ t_n$	 $\wedge\ <\ 0\ \ a\ \ \neg <\ a\ \ 10$
Postfixform:	Operator **nach** seinen Operanden $t_1\ \ t_2 ...\ t_n\ \ f$	 $0\ a\ <\ a\ 10\ <\neg\ \wedge$
Infixform	2-stelliger Operator **zwischen** seinen (beiden) Operanden $t_1\ f\ \ t_2$	 $0 < a\ \wedge\ \neg\ a < 10$

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

Präzedenzen und Klammern für Infixform

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 jeder seiner Unterterme geklammert ist:

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:

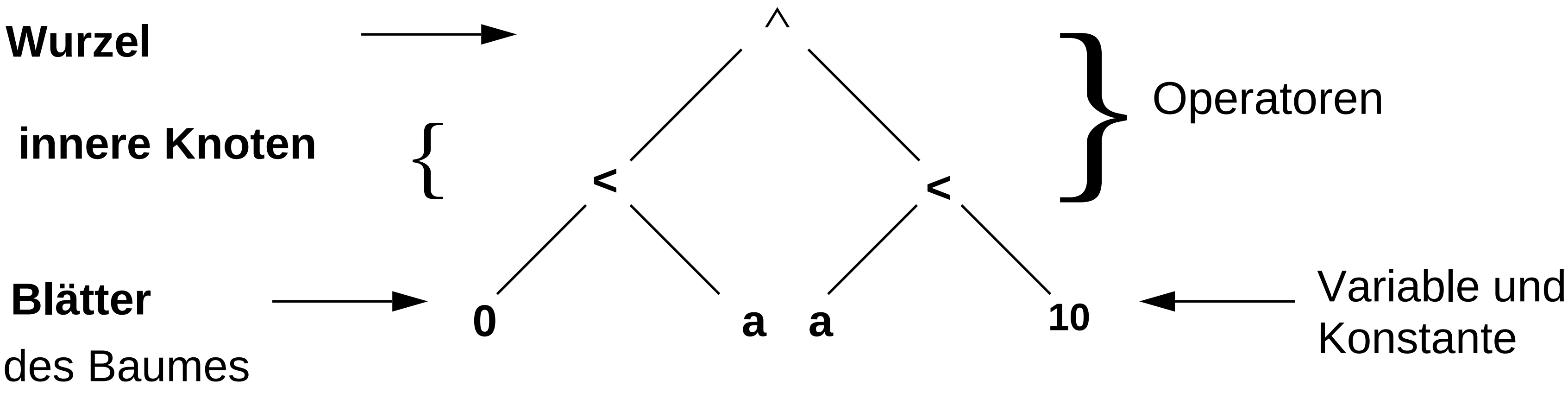
links-assoziativ:	x **+** 3 **+** y	steht für (x **+** 3) **+** y
rechts-assoziativ:	x ****** 3 ****** y	steht für x ****** (3 ****** y)

Funktionsform, Präfixform, Postfixform benötigen weder
Regeln für Präzedenz oder Assoziativität noch zusätzliche Klammern!

Terme als Bäume

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

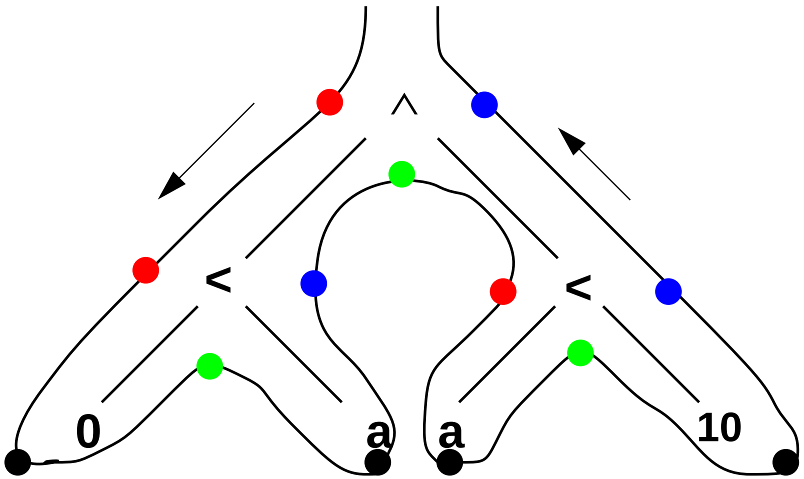
Term: $\mathbf{0} < \mathbf{a} \wedge \mathbf{a} < \mathbf{10}$



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 und Unifikation

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

Eine **einfache Substitution** σ **= [v / t]** ist ein Paar aus einer Variablen v und einem Term t zur Signatur Σ. 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 einfachen Substitution u** σ mit σ **= [v / t]**, ist **definiert** durch

- **u [v / t] = t** , falls u die zu ersetzende Variable v ist,
- **u [v / t] = u** , falls u $\neq$ v und u eine Konstante oder eine andere Variable ist,
- **u [v / t] = f (u_1 [v / t], u_2 [v / t], ..., u_n [v / t])** , 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

In einer **mehrfachen Substitution** $\sigma = [\, \mathbf{v_1 / t_1, ..., v_n / t_n}]$ müssen alle Variablen $\mathbf{v_i}$ paarweise verschieden sein. In jedem $\mathbf{v_i / t_i}$ müssen $\mathbf{v_i}$ und $\mathbf{t_i}$ jeweils derselben Sorte $\mathbf{s_i}$ angehören.
σ wird dann auf einen Term u wie folgt angewandt:

- $\mathbf{u\, \sigma = t_i}$, falls u = v_i für ein i $\in$ {1, ..., n},
- $\mathbf{u\, \sigma = u}$, falls u eine Konstante ist oder eine Variable, die nicht unter v_i für ein i $\in$ {1, ..., n} vorkommt,
- $\mathbf{u\, \sigma = f\, (u_1\, \sigma, u_2\, \sigma, ..., u_n\, \sigma)}$, falls u = f (u_1, u_2, ..., u_n)

D. h. σ **ist die gleichzeitige Substitution aller Vorkommen jeder Variablen** $\mathbf{v_i}$ jeweils durch den Term t_i.

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

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

Die **leere Substitution** wird **[ ]** notiert. Für alle Terme t gilt **t [ ] = t**.
Außerdem gilt **[v / v] = [ ]** für jede Variable v.

Hintereinanderausführung von Substitutionen

Auf einen Term können **mehrere Substitutionen hintereinander** ausgeführt werden,

z. B. $u\, \sigma_1\, \sigma_2\, \sigma_3 = ((u\, \sigma_1)\, \sigma_2)\, \sigma_3$

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

Mehrere **Substitutionenen hintereinander** können als **eine Substitution** angesehen werden:

z. B. $u\, \sigma_1\, \sigma_2\, \sigma_3 = u\ (\sigma_1\, \sigma_2\, \sigma_3) = u\, \sigma$

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 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 [ ]

Eine **Halbordnung ist umfasst nicht**, weil

nicht antisymmetrisch: Terme, die sich nur in den Variablennamen unterscheiden, kann man ineinander umformen, z. B.
$2*x\,[\,x\,/\,y\,] = 2*y$ und $2*y\,[\,y\,/\,x\,] = 2*x$

Deshalb gilt zwar der allgemeinere Term $a * (b + c)$ umfasst den spezielleren $2 * (3 + 4*x)$, aber nicht immer ist ein Term s allgemeiner als ein Term t, wenn s umfasst t: $2*x$ und $2*y$

Unifikation

Die **Unifikation substituiert zwei Terme, sodass 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: σ_1 = [x / 2, y / z] σ_2 = [x / 2, z / y],
σ_3 = [x / 2, y / 1, z / 1] σ_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.

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. σ_1 [z / 1] $= \sigma_3$

Es kann **mehrere allgemeinste Unifikatoren** geben. Sie können durch **Umbenennen von Variablen** ineinander überführt werden, z. B.

σ_1 [**z / y**] = [x / 2, y / z] [z / y]= [x / 2, y / y, z / y] = [x / 2, 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:

$s = +(*\ (2, x), 3)$
$t = +(z, \qquad x)$ $\qquad \sigma = [\,]$

Schritt ↓ Abweichungspaar

1 $\quad s\sigma = +(*\ (2, x), 3)$ $\quad$ Fall 2b: $\quad \sigma = [\,]\,[\, z / *\ (2, x)\,]$
$t\sigma = +(z, \qquad x)$

↓ Abweichungspaar

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

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

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

3.2 Algebren

Eine **Algebra** ist eine **formale Struktur,** definiert durch
eine **Trägermenge**, **Operationen** darauf 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 der Signatur

Gesetze erlauben, Vorkommen von Termen durch andere Terme zu ersetzen
z. B. ¬ false –> true pop (push (k, t)) –> k

Eine **konkrete Algebra** zu einer abstrakten Algebra

definiert **konkrete Funktionen** zu den Operationen der Signatur,
so dass die Gesetze in **Gleichungen zwischen den Funktionstermen** übergehen.

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 eine **Menge von Axiomen (Gesetzen) Q**.

Axiome haben die Form $t_1 \rightarrow t_2$, wobei t_1, t_2, **korrekte Terme gleicher Sorte** sind, die **Variablen** enthalten können. Die Algebra definiert, wie man Terme **mit den Axiomen in andere Terme umformen** kann.

Mit Axiomen umformen heißt: Unter Anwenden eines Axioms $t_1 \rightarrow t_2$ kann man einen Term s_1 in einen Term s_2 umformen. Wir schreiben $s_1 \rightarrow s_2$, 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 $t_2 \sigma = r_2$

Terme	s_1 =	.... r_1	->	.... r_2	=	s_2
		=		=		
		$t_1 \sigma$		$t_2 \sigma$		
Axiom		t_1	->	t_2		

s ist in t umformbar, wenn es eine endliche Folge von Termen $s = s_0, s_1, ..., s_n = t$ mit $s_{i-1} \rightarrow s_i$ gibt; wir schreiben dann $s \rightarrow t$.

„->" ist **transitiv**. Wenn es auch **irreflexiv** ist (so sollten die Axiome gewählt werden), ist es eine **strenge Halbordnung**.

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

Die Axiome sind geeignet, alle korrekten Terme ohne Variablen in in einen der beiden Terme **true** oder **false** umzuformen.

true und **false** heißen **Normalformen** (siehe Folie 3.20).

Konkrete Algebra

Zu einer abstrakten Algebra $\mathcal{A}_a = (\tau, (S, F), Q)$, kann man

konkrete Algebren wie $\mathbf{\mathcal{A}_k = (W_k, F_k, Q)}$

angeben, wobei

W_k eine **Menge von Wertebereichen** ist, je einer **für jede Sorte** aus S,

F_k eine **Menge von Funktionen** ist, je eine **für jede Operation** aus F.

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

Den **Axiomen Q** müssen **Gleichungen zwischen den Funktionstermen** in den Wertebereichen entsprechen.

Es können in der konkreten Algebra noch weitere Gleichungen 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_k:	$\{\emptyset, \{1\}\}$	Sorte BOOL
F_k:	$\{1\}$	true
	$\emptyset$	false
	Mengendurchschnitt $\cap$	$\wedge$
	Mengenvereinigung $\cup$	$\vee$
	Mengenkomplement bezüglich $\{1\}$	$\neg$

Axiome Q:
Man kann zeigen, dass die Axiome Gleichungen zwischen den Termen in W_k entsprechen:
z. B. $\emptyset \cap x = \emptyset$ entspricht false $\wedge$ x -> false

Die boolesche Algebra mit den üblichen logischen Funktionen ist natürlich auch eine konkrete Algebra zur abstrakten Algebra Bool.

Beispiel 2.2: Datenstruktur Keller

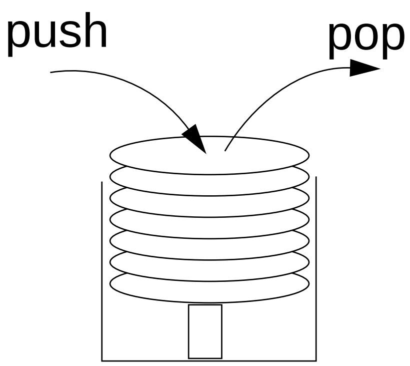
Die Eigenschaften einer **Datenstruktur Keller** beschreiben wir zunächst informell.
Folgende **Operationen** kann man mit einem Keller ausführen:

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.

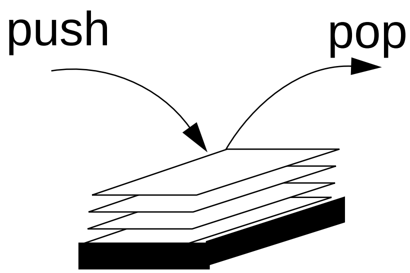
Die Eigenschaften der Datenstruktur Keller sollen präzise durch eine abstrakte Algebra spezifiziert werden.

Beispiele

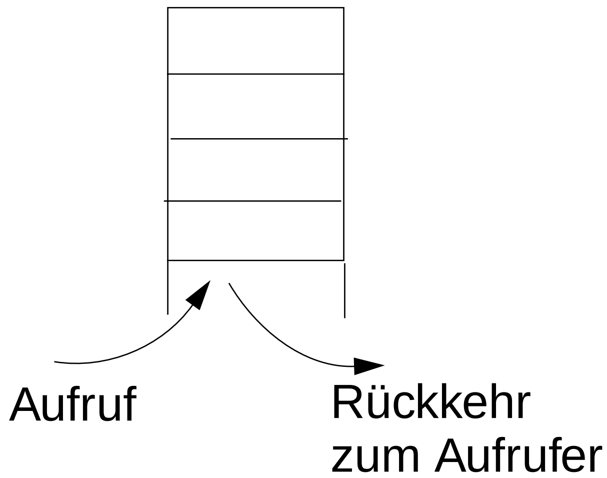
Tellerstapel



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)	-> true
K2:	empty (push (k, t))	-> false
K3:	pop (push (k, t))	-> k
K4:	top (push (k, t))	-> 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 werden in 3 disjunkte Mengen eingeteilt:

Konstruktoren: Ergebnissorte ist die definierte Sorte

Hilfskonstruktoren: Ergebnissorte ist die definierte Sorte und sie können durch Axiome aus Termen entfernt werden

Projektionen: andere Ergebnissorte

z. B. in der Keller-Algebra: definierte Sorte ist Keller

createStack:		-> Keller	**Konstruktor**
push:	Keller x Element	-> Keller	**Konstruktor**
pop:	Keller	-> Keller	**Hilfskonstruktor** (K3 entfernt ihn)
top:	Keller	-> Element	**Projektion**
empty:	Keller	-> BOOL	**Projektion**

Normalform

Terme ohne Variable der definierten Sorte sind in **Normalform**, wenn sie nur **Konstruktoren** enthalten **kein Axiom anwendbar** ist.

Normalform-Terme der Algebra Bool sind: true false

Normalform-Terme der Keller-Algebra haben die Form:

push (.... push (createStack, n_1) , ...), n_m), mit $m \geq 0$

Die **Terme in Normalform** sind die minimalen Elemente bzgl. der strengen Halbordnung **->**.

Terme s, t, die in **dieselbe Normalform** umformbar sind, heißen **gleichbedeutend**, $s \equiv t$.

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)

Für manche **Projektionen** gibt es nicht zu jedem Term in Normalform ein anwendbares Axiom; 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

   Zählt man in einem Term von innen nach außen die push-Operationen positiv und die pop-Operationen negativ, und ist der Wert immer nicht-negativ, so ergibt sich die **Anzahl der Elemente im Keller**, andernfalls ist der Term undefiniert.
   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 -> NAT
2. Ergebnis-Sorte **NAT zu den Sorten zufügen**:
   S = {Keller, Element, BOOL, NAT}
3. **Axiome zufügen**, so dass size für jeden Keller-Wert definiert ist:
   K7: size (createStack) -> null
   K8: size (push (k, t)) -> succ (size (k))
4. Weil in der **Normalform** nur createStack und push vorkommen, braucht size nur für solche Terme definiert zu werden.

Dabei wird vorausgesetzt, dass folgende Algebra bekannt ist:
Sorten: S = {NAT}
Operationen: null: -> NAT, succ: NAT -> NAT
(succ (n) modelliert den Nachfolger von n, also n + 1.)

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 = $\mathbb{N}^*$	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	-> $\mathbb{N}$	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 **Infixform in Postfixform** umwandeln

gegeben: Term t in Infixform, mit 2-stelligen Operatoren unterschiedlicher Präzedenz; (zunächst) ohne Klammern

gesucht: Term t in Postfixform

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 Infixform aufeinander folgende **Operatoren echt steigender Präzedenz** stehen in der Postfixform **in umgekehrter Reihenfolge**; also kellern.
4. **Operatorkeller enthält Operatoren echt steigender Präzedenz**. Es gilt die **Kellerinvariante KI**:

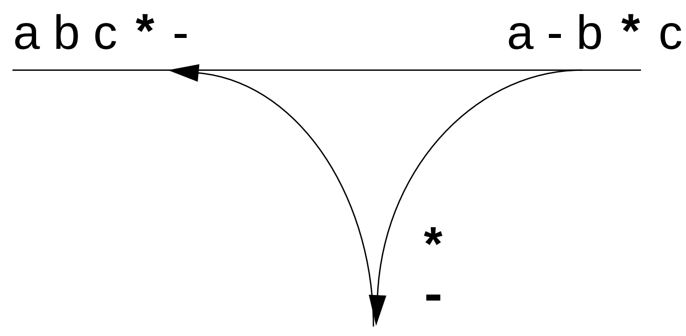
   Sei push (... push (CreateStack, opr_1), opr_2), ...) dann gilt

   Präzedenz ($opr_{\mathbf{i}}$) < Präzedenz ($opr_{\mathbf{i+1}}$)

Algorithmus: Infix- in Postfixform wandeln

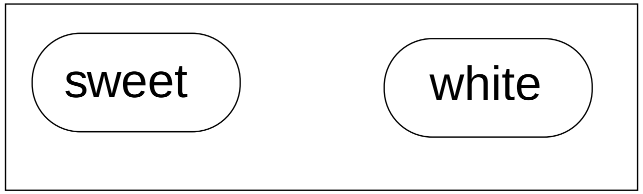
Die Eingabe enthält einen Term in Infixform;
die Ausgabe soll den Term in Postfixform enthalten

```
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.

Abstrakte Algebra für Teilaspekt des Getränkeautomaten



Knöpfe des Getränkeautomaten
zur Auswahl von Zutaten

Signatur $\Sigma = (S, F)$;

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 der Sorte Add und
für alle c der Sorte Choice gilt:

Q_1: press (a, press (a, c)) -> c

Q_2: press (sweet, press (white, c)) ->
press (white, press (sweet, c))

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

4 Logik
4.1 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 booleschen Algebra
- **Belegung von Variablen** mit Wahrheitswerten
- **Interpretation** aussagenlogischer Formeln
- **Gesetze der booleschen 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:

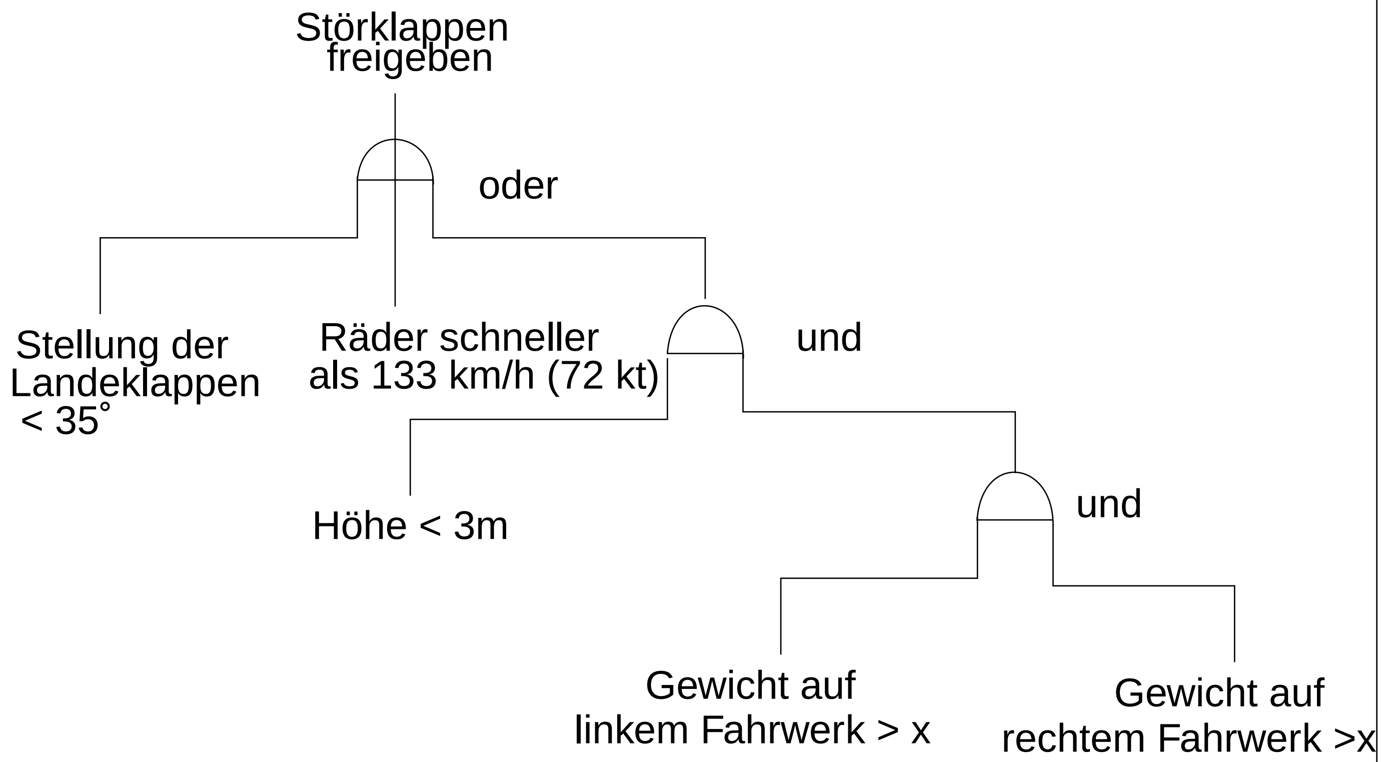
Die Störklappen dürfen benutzt werden

- 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 booleschen Algebra:

false:		-> Bool	falsch, f
true:		-> Bool	wahr, w
$\wedge$:	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 false und true (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 F vorkommen, jeweils einen Wahrheitswert w oder f (für wahr oder falsch) zu.
Die Belegung kann als Substitution angegeben werden, z.B. $\sigma = [\, p / w, q / f \,]$.

Eine **Interpretation** $\Im_\sigma$ einer aussagenlogischen Formel F bildet F auf einen Wahrheitswert ab:

- Für **Variable** ist die Interpretation $\Im_\sigma$ durch die **Belegung** σ definiert.
- Für **zusammengesetzte Formeln** wird sie durch folgende **Wahrheitstafeln** erweitert:

$\Im$(false)=f
$\Im$(true)=w

$\Im$(F)	$\Im(\neg$F)
w	f
f	w

$\Im$(F)	$\Im$(G)	$\Im(F \wedge G)$	$\Im(F \vee G)$	$\Im(F \rightarrow G)$	$\Im(F \leftrightarrow G)$
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 $\Im_\sigma$ mit einer Belegung σ für eine Formel F bestimmt einen **Wahrheitswert der Formel F:** $\Im_\sigma(F)$

Wenn $\Im_\sigma(F) = w$ gilt, heißt $\Im_\sigma$ 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 einen Euro oder zwei Fünfziger? $\quad$ euro $\vee$ 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 GP-Klausur nicht bestanden habe,
   nehme ich am zweiten Termin teil. $\quad$ $\neg$ gp-k1 $\wedge$ gp-k2
9. Weil ich die Modellierungsklausur bestanden habe,
   nehme ich am zweiten Termin nicht teil. $\quad$ mod-k1 $\wedge$ $\neg$ 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 $\Im_\sigma$ mit einer Belegung σ gibt, so dass gilt $\Im_\sigma$ (F) = w,
sonst ist sie **widerspruchsvoll** (**unerfüllbar**), d.h.
für alle Interpretationen $\Im_\sigma$ mit einer Belegung σ gilt $\Im_\sigma$ (F) = f.

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

Eine Formel F heißt **allgemeingültig** oder **Tautologie**, wenn
für alle ihre Interpretationen $\Im_\sigma$ (F) = w gilt.

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

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

allgemein-gültig F	erfüllbar aber nicht allgemeingültig	widerspruchs-voll $\neg$ F

Gesetze der booleschen Algebra

Zwei Formeln F, G sind **logisch äquivalent**, **F ≡ 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
$X \wedge \text{false} \equiv \text{false}$ $X \wedge \text{true} \equiv X$	$X \vee \text{false} \equiv X$ $X \vee \text{true} \equiv \text{true}$	Neutrale Elemente
$X \wedge \neg X \equiv \text{false}$	$X \vee \neg X \equiv \text{true}$	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:

$(A \lor \underline{\neg (B \land A)}) \land (C \lor (D \lor C)) \equiv$	De Morgan
$(A \lor \underline{(\neg B \lor \neg A)}) \land (C \lor (D \lor C)) \equiv$	Kommutativität
$(\underline{A \lor (\neg A \lor \neg B)}) \land (C \lor (D \lor C)) \equiv$	Assoziativität
$((\underline{A \lor \neg A}) \lor \neg B) \land (C \lor (D \lor C)) \equiv$	Komplement
$(\underline{true \lor \neg B}) \land (C \lor (D \lor C)) \equiv$	Kommutativität
$(\underline{\neg B \lor true}) \land (C \lor (D \lor C)) \equiv$	Neutrale Elemente
$\underline{true \land (C \lor (D \lor C))} \equiv$	Kommutativität
$\underline{(C \lor (D \lor C)) \land true} \equiv$	Neutrale Elemente
$(C \lor (\underline{D \lor C})) \equiv$	Kommutativität
$(\underline{C \lor (C \lor D)}) \equiv$	Assoziativität
$((\underline{C \lor C}) \lor D) \equiv$	Idempotenz
$C \lor D$	

Logischer Schluss

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

„F folgt semantisch aus $\mathcal{A}$" **$\mathcal{A}$ |= F**

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

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

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

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

Beweise werden aus logischen Schlüssen aufgebaut.

Beispiel:

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

V: Es gibt Privilegien.

W: Nicht alle Menschen sind gleich.

nachweisen: **{ U, V } |= W** ist ein **korrekter logischer Schluss**.

4.2 Prädikatenlogik

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

- A = (τ, Σ) ist die so genannte **Termalgebra** (mit Variablen, ohne Axiome) mit Signatur
  Σ = ({T}, F), wobei T die Sorte „Term“ ist und alle Operationen $f \in F$ von der Form f: $T^n \to T$ sind. **Terme** sind die korrekten Terme bzgl. dieser Termalgebra.

- **n-stellige Prädikate** sind Operationen P: $T^n \to$ BOOL. In einer Konkretisierung entsprechen ihnen n-stellige Relationen,
  z. B. „x ist eine Katze“ bzw. als Formel: istKatze(x)
  teilt(a,b),   größterGemeinsamerTeiler(a, b, g)

- **Quantoren „für alle x gilt** α“ und „**es gibt ein x, so dass** α **gilt**“
  in Symbolen: $\forall \mathbf{x} \alpha$ bzw. $\exists \mathbf{x} \alpha$
  Beispiel: $\forall$x (esIstNacht $\wedge$ istKatze(x) $\to$ 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) $\to$ 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, widerspruchsvoll**:
  wie in der Aussagenlogik definiert
- **logischer Schluss**: wie in der Aussagenlogik definiert
- **Gesetze zum Umformen** von Formeln mit Quantoren

Prädikatenlogische Formeln

Prädikatenlogische Formeln (PL-Formeln) werden induktiv wie folgt definiert:

1. **Primformeln** sind Anwendungen von Prädikaten in der Form **$P\ (t_1, ..., t_n)$ oder** Gleichungen in der Form **$t_1 = t_2$.**
   Dabei ist **P** ein n-stelliges Prädikatsymbol und die **t_i** sind Terme der Termalgebra.
   0-stellige Prädikatsymbole entsprechen den atomaren **Aussagen der Aussagenlogik.**

2. **logische Junktoren** bilden prädikatenlogische Formeln:
   $\neg\alpha \qquad \alpha \wedge \beta \qquad \alpha \vee \beta$
   sowie $\alpha \rightarrow \beta \quad \alpha \leftrightarrow \beta$ **als Abkürzungen**
   mit prädikatenlogischen Formeln α und β

3. der **Allquantor** $\forall$ und der **Existenzquantor** $\exists$ bilden prädikatenlogische Formeln:
   $\forall\mathbf{x}\ \alpha$ und $\exists\mathbf{x}\ \alpha$
   mit der prädikatenlogischen Formel α; sie definieren die Variable x

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

Quantoren haben die gleiche **Präzedenz** wie $\neg$, also höhere als $\wedge$.

Beispiele:

$\text{teilt}(g, a) \wedge \text{teilt}(g, b) \wedge (\forall h\ (\text{teilt}(h, a) \wedge \text{teilt}(h, b) \rightarrow \leq (h, g))$ (siehe Folie 4.21)

$\forall x\ \forall y\ \forall z\ ((R(x, y) \wedge R(x,z)) \rightarrow y = z)$ „R ist eine Funktion“

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**. Nur solche Variablen dürfen quantifiziert werden.

Vorkommen von Variablen

Wir sagen: (Eine Variable mit Namen) **x kommt in einer PL-Formel α vor**, wenn sie in einer Primformel und dort in einem Term vorkommt.

Für eine PL-Formel der Form $\forall x\ \alpha$ oder $\exists x\ \alpha$ ist α der **Wirkungsbereich (für x) des Quantors**. x ist der **Name der Variablen des Quantors**.

Beispiel:

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

Quantoren mit ihrenWirkungsbereichen

Anmerkungen:

- Eine Variable hat einen Namen; **mehrere Variable können den gleichen Namen haben**.
- Ein Quantor definiert eine Variable, z. B. $\forall x\ \alpha$ definiert (eine Variable mit Namen) x.
  Ihr **Name kann im Wirkungsbereich (auch mehrfach) vorkommen**.
- **Wirkungsbereiche** von Quantoren können **geschachtelt** sein,
  sogar mit (verschiedenen) Variablen, die **dieselben Namen** haben.

Freie und gebundene Variable

(Ein Vorkommen von) **x** in einer Formel α **heißt frei**, wenn es nicht im Wirkungsbereich für x eines Quantors liegt.

Ein **Quantor** $\forall$**x** α **bzw.** $\exists$**x** α **bindet** alle (Vorkommen von) x, die frei sind in α. (Das Vorkommen von) x heißt dann **gebunden**.

Beispiel: Formel α

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

freie Vorkommen

gebundene Vorkommen

In α gibt es 3 freie Variable; sie haben die Namen y, x, z.

2 Variable haben den Namen y;
eine kommt frei vor in R(y), die andere kommt 2 mal gebunden in α vor.

Umbenennung von Variablen

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

Beispiele:

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

Umbenennung: In einer Formel kann man **alle (gebundenen) Vorkommen des Namens x der Variablen eines Quantors in dessen Wirkungsbereich durch einen neuen Namen z ersetzen**, der sonst nicht in der Formel vorkommt. Die Bedeutung der Formel, (genauer: semantische Aussagen über sie), ändert sich dadurch nicht.

Beispiele von oben:

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

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 α wird durch eine **Interpretation** $\mathfrak{I}$ (α)
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** auf dazu passende konkrete Funktionen und Relationen, notiert als z. B. $\mathfrak{I}$(h), $\mathfrak{I}$(P)
- eine **Belegung der freien Variablen mit Werten aus U**, notiert z. B. $\mathfrak{I}$(x).
- die Interpretation der Junktoren und Quantoren (definiert auf Folie 4.31)

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).
- **Der Sorte T** wird deshalb **der ganze Individuenbereich U** zugeordnet.
- 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 $\alpha = (\forall x\ P(x, h(x))) \wedge Q(g(a, z))$ ist folgendes $\mathfrak{I}$ eine passende Interpretation:

$U := \mathbb{N}$

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

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

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

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

$\mathfrak{I}(a) := 2$ (a ist eine Konstante, d.h. eine 0-stellige Funktion, $2 \in U$)

$\mathfrak{I}(z) := n$ (z ist eine freie Variable, $n \in U$)

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 (Folie 4.31).

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

Wahrheitswerte prädikatenlogischer Formeln

Sei α eine prädikatenlogische Formel und $\mathfrak{I}$ eine dazu passende Interpretation, dann berechnet man den **Wahrheitswert $\mathfrak{I}(\alpha)$,** indem man $\mathfrak{I}$ **rekursiv anwendet** auf die Teile von α:

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

Interpretation von PL-Formeln (vollständige Definition)

Die Interpretation der Symbole wird auf prädikatenlogische Formeln, deren Variablen konsistent umbenannt sind, erweitert:

Für jeden Term $\mathbf{h(t_1, ..., t_n)}$ wird definiert: $\Im(h\ (t_1, ..., t_n)) = \Im(h)(\Im(t_1), ..., \Im(t_n))$.

Für Formeln gilt (Definition durch Induktion über den Aufbau der prädikatenlogischen Formeln):

1. $\Im(P\ (t_1, ..., t_n))$ = w genau dann, wenn $(\Im(t_1), ..., \Im(t_n)) \in \Im(P)$
2. $\Im(t_1 = t_2)$ = w genau dann, wenn $\Im(t_1) = \Im(t_2)$
3. $\Im(\neg\alpha)$ = w genau dann, wenn $\Im(\alpha)$ = f
4. $\Im(\alpha \wedge \beta)$ = w genau dann, wenn $\Im(\alpha)$ = w **und** $\Im(\beta)$ = w
5. $\Im(\alpha \vee \beta)$ = w genau dann, wenn $\Im(\alpha)$ = w **oder** $\Im(\beta)$ = w
6. $\Im(\forall x\alpha)$ = w genau dann, wenn **für jeden Wert d** $\in$ **U** gilt $\Im_{[x/d]}(\alpha)$ = w
7. $\Im(\exists x\alpha)$ = w genau dann, wenn es **einen Wert d** $\in$ **U gibt** mit $\Im_{[x/d]}(\alpha)$ = w

Dabei ordnet $\Im_{[x/d]}(\alpha)$ in α der Variablen **x** den Wert **d** zu und stimmt sonst mit der gerade angewandten Interpretation $\Im$ überein.

Beispiel für Interpretation einer Formel

Formel α:

$R \wedge \forall x\ \forall y\ P(x, y)$

Interpretation $\Im$:

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

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

$\Im(R) = w$

Interpretation $\Im$ rekursiv gemäß Mod-4.31 angewandt:

Nr.:		$\Im(R \wedge \forall x\ \forall y\ P(x, y))$
4	=	$\Im(R)$ und $\Im(\forall x\ \forall y\ P(x, y))$
$\Im$,6,6	=	w und für jedes d, e $\in$ U gilt $\Im_{[x/d,\ y/e]}(P(x, y))$
1	=	w und für jedes d, e $\in$ U gilt $(\Im_{[x/d,\ y/e]}(x), \Im_{[x/d,\ y/e]}(y)) \in \Im_{[x/d,\ y/e]}(P)$
$\Im$	=	w und für jedes d, e $\in \{ 1, 2, 3 \}$ gilt $(d, e) \in \{ (a, b) \mid a + b < 10 \}$
	=	w und w
	=	w

Elementare Interpretationen

Wir betrachten für die Beispiele A bis G eine Interpretation $\Im$ mit Individuenbereich U = $\mathbb{N}$.

a. freie Variable: $\Im$ (u) = 1 $\in$ U, $\Im$ (v) = 2 $\in$ U (bestimmte Elemente von U)

b. 0-stellige Prädikate: $\Im$ (A) = w oder $\Im$ (A) = f (boolesche Variable)

c. 1-stellige Prädikate: $\Im$ (P) = M $:= \{1, 2, 3\} \subseteq$ U (Teilmenge von U)

d. 2-stellige Prädikate: $\Im$ (Q) = R := {(1, 2), (2, 2)} $\subseteq$ U $\times$ U (Relation auf U)

A. $\Im$ (P(u)) = w gdw $\Im$ (u) $\in$ $\Im$ (P), d. h. 1 $\in$ M

B. $\Im$ (Q(u, v)) = w gdw ($\Im$ (u), $\Im$ (v)) $\in$ $\Im$ (Q), d. h. (1, 2) $\in$ R

C. $\Im$ ($\forall$x P(x)) = w gdw (Für alle d $\in$ U gilt: d $\in$ M) = f, d. h. M $\neq$ U

D. $\Im$ ($\exists$x P(x)) = w gdw Es existiert d $\in$ U mit d $\in$ M, M $\neq \emptyset$

E. $\Im$ ($\forall$x Q(x, x)) = w gdw (Für alle d $\in$ U gilt: (d, d) $\in$ R) = f, d. h. R ist nicht reflexiv

F. $\Im$ ($\exists$x Q(x, x)) = w gdw Es gibt ein d $\in$ U mit (d, d) $\in$ R, d. h. R ist nicht irreflexiv

G. $\Im$ ($\forall$x $\forall$y (Q(x,y) $\wedge$ Q(y, x) $\rightarrow$ x = y)) = w
gdw Für alle d, e $\in$ U gilt: aus (d, e) $\in$ R und (e, d) $\in$ R folgt d = e,
d. h. R ist antisymmetrisch

Beschränkung von Wertebereichen

In der **Prädikatenlogik** kann die **Interpretation von Variablen** Werte aus dem **gesamten Individuenbereich U** annehmen (im Unterschied zu einem **Wertebereich**).
Deshalb muss eine **Einschränkung explizit als Relation** formuliert werden.

Beschränkung des Wertebereiches bei Allquantoren durch Implikation $\rightarrow$:
„Für alle $m \in U$ gilt: **aus** $m \in M$ **folgt** $Q(m, n)$“ oder abgekürzt „$\forall m \in M: Q(m, n)$“

als PL-Formel: $\forall x\ (P(x) \rightarrow Q(x, y))$

	ausführliche Notation:	abkürzende Notation:
Beispiele:	Für alle $i \in U$ gilt: aus $i \in \{1, 2, 3, 4\}$ folgt $b_i = a_i^2$	$\forall i \in \{1, 2, 3, 4\}: b_i = a_i^2$
	Für alle $k \in U$ gilt: aus $k \in \mathbb{N}$ folgt $a + k \geq a$	$\forall k \in \mathbb{N}: a + k \geq a$

Beschränkung des Wertebereiches bei Existenzquantoren durch Konjunktion $\wedge$:
„Es gibt ein $m \in U$, sodass $m \in M$ **und** $Q(m, n)$ “ oder abgekürzt „$\exists m \in M: Q(m, n)$“

PL-Formel: $\exists x\ (P(x) \wedge Q(x, y))$

Beispiele:	Es gibt ein $k \in U$, sodass $k \in \mathbb{N}$ **und** $a * k = b$	$\exists k \in \mathbb{N}: a * k = b$
	Es gibt ein $i \in U$, sodass $i \in \{1, 2, 3, 4\}$ **und** $a_i = x$	$\exists i \in \{1, 2, 3, 4\}: a_i = x$

Beispiele für PL-Formeln und deren Interpretation (1)

Die Variablen in Gleichungen konkreter Algebren sind durch Allquantoren gebunden:

Axiom K3: pop (push (k, x)) –>k (in der abstrakten Keller-Algebra)

Gleichung: $\forall a \in \mathbb{N}^* : \forall n \in \mathbb{N}$: remove (append (a, n)) = a (konkrete Algebra)

PL-Formel: $\forall k\ \forall x\ (P(k) \wedge S(x) \rightarrow h\ (\ g\ (k, x)) = k)$

Interpretation: $U = \mathbb{N}^* \cup \mathbb{N}$, $\Im(S) = \mathbb{N}$, $\Im(P) = \mathbb{N}^*$

$\Im(h)$ = remove: $\mathbb{N}^* \rightarrow \mathbb{N}^*$,

$\Im(g)$ = append: $\mathbb{N}^* \times \mathbb{N} \rightarrow \mathbb{N}^*$

Es gilt: $\Im(\forall k\ \forall x\ (P(k) \wedge S(x) \rightarrow h(\ g\ (k, x)) = k))) = w$

gdw $\forall a \in \mathbb{N}^* \cup \mathbb{N} : \forall n \in \mathbb{N}^* \cup \mathbb{N} : \Im_{[k/a,\ x/n]} (P(k) \wedge S(x) \rightarrow h\ (\ g\ (k, x)) = k) = w$

gdw $\forall a \in \mathbb{N}^* \cup \mathbb{N} : \forall n \in \mathbb{N}^* \cup \mathbb{N}$: Aus $a \in \mathbb{N}^*$ und $n \in \mathbb{N}$ folgt: remove (append (a, n)) = a

gdw $\forall a \in \mathbb{N}^* \cup \mathbb{N} : \forall n \in \mathbb{N}$: remove (append (a, n))= a

Beispiele für PL-Formeln und deren Interpretation (2)

Aus der Analysis:

Eine Funktion a : $\mathbb{N} \rightarrow \mathbb{R}$, a(n) = a_n, heißt Nullfolge, wenn gilt

$\forall\ \varepsilon \in \mathbb{R}^+ : \exists\ n_0 \in \mathbb{N} : \forall\ n \in \mathbb{N}$ mit $n_0 < n : |\ a_n\ | < \varepsilon$

Dreifache Schachtelung der Quantoren; Reihenfolge ist wichtig!

PL-Formel α: $\forall x(P_1(x) \rightarrow \exists y(P_2(y) \wedge \forall z(P_2(z) \wedge Q(y, z)) \rightarrow Q(h(z), x))))$

Interpretation: $U = \mathbb{R}$, $\Im(P_1) = \mathbb{R}^+$, $\Im(P_2) = \mathbb{N}$,

$\Im(Q) = \{\ (r, s) \mid (r, s) \in \mathbb{R}^+ x \mathbb{R}^+$ und $r < s\ \}$,

$\Im(h) : \mathbb{N} \rightarrow \mathbb{R}$, $\Im(h)\ (i) = |\ a_i\ |$

Es gilt: $\Im(\alpha) = w$ gdw a_n ist eine Nullfolge

Beispiele für PL-Formeln und deren Interpretation (3)

Aus der Informatik:

Eine Folge $a = (a_1, ..., a_k) \in \mathbb{N}^k$ heißt monoton wachsend, wenn gilt

$\forall\, i \in \{1, ..., k\}: \forall\, j \in \{1, ..., k\}$ mit $i \leq j$ gilt $a_i \leq a_j$

PL-Formel β : $\forall x(P(x) \rightarrow \forall y((P(y) \wedge Q(x, y)) \rightarrow Q(h(x), h(y))))$

Interpretation: $U = \mathbb{N}^k \cup \{1, ..., k\}$, $\Im(P) = \{1, ..., k\}$,

$\Im(Q) = \{ (m, n) \in \mathbb{N} \times \mathbb{N} \mid m \leq n\}$

$\Im(h) : \{1, ..., k\} \rightarrow \mathbb{N}$, $\Im(h)(i) = a_i$

Es gilt: $\Im(\beta) = w$ gdw a_n ist monoton wachsend

Was bedeutet $\Im(P(x) \wedge \forall y(P(y) \rightarrow (Q(h(x), h(y)) \wedge (h(x) = h(y) \rightarrow Q(x, y))))) = w$

mit $\Im(x) = i$, $i \in \{1, ..., k\}$, bei sonst unveränderter Interpretation?

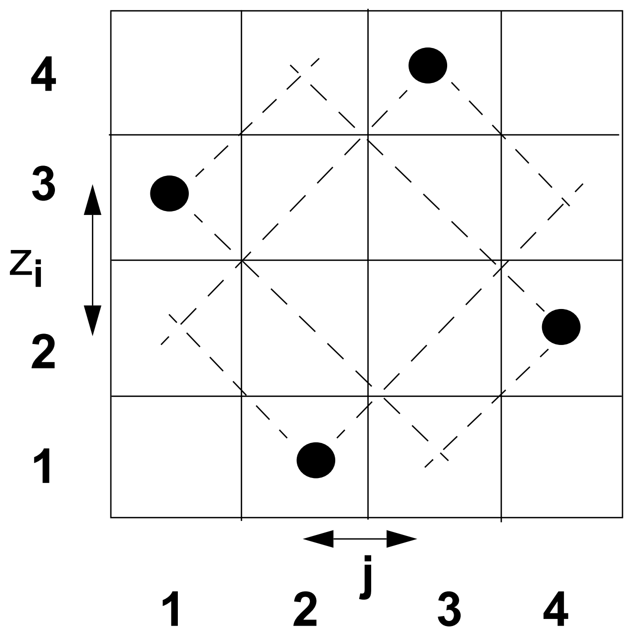
Beispiel: Spezifikation des n-Damen-Problems

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}

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

z_i gibt die Zeilennummer der Dame in Spalte i an.

Dabei bedeutet

zulässig (p): $\forall\, i \in \text{Index}: \forall\, j \in \text{Index}: i \neq j \rightarrow z_i \neq z_j \wedge |\, z_i - z_j \,| \neq |\, i - j \,|$

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

Äquivalente Umformung prädikatenlogischer Formeln

Seien α und β beliebige prädikatenlogische Formel. Dann gelten folgende **Äquivalenzen**:

1. **Negation:**

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

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

   $(\forall x\ \alpha) \wedge \beta \equiv \forall x\ (\alpha \wedge \beta)$ $\qquad$ $(\forall x\ \alpha) \vee \beta \equiv \forall x\ (\alpha \vee \beta)$

   $(\exists x\ \alpha) \wedge \beta \equiv \exists x\ (\alpha \wedge \beta)$ $\qquad$ $(\exists x\ \alpha) \vee \beta \equiv \exists x\ (\alpha \vee \beta)$

   $\beta \equiv \exists x\ \beta$ $\qquad$ $\beta \equiv \forall x\ \beta$

3. **Quantoren zusammenfassen:**

   $(\forall x\ \alpha \wedge \forall x\ \beta) \equiv \forall x\ (\alpha \wedge \beta)$ $\qquad$ $(\exists x\ \alpha \vee \exists x\ \beta) \equiv \exists x\ (\alpha \vee \beta)$

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

4. **Quantoren vertauschen:**

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

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)

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

gdw $\neg\forall i\ (i \in \text{Ind} \rightarrow a_i < 10)$

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

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

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

$(\exists\ x\ P(x)) \rightarrow P(y)$

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

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

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

$\equiv \forall\ x\ (P(x) \rightarrow P(y))$

2. Zusammenfassung von Quantoren:

Äquivalent:

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

Nicht äquivalent, vielmehr gilt nur:

Aus $(\forall i \in \text{Ind}: a_i < 10) \vee (\forall i \in \text{Ind}: 0 < a_i)$ **folgt** $\forall i \in \text{Ind}: (a_i < 10 \vee 0 < a_i)$

Beispiel für Umformungen

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

	$\underline{\neg(\exists x\ P(x, y) \vee \forall z\ Q(z))} \wedge \exists u\ f(a, u) = a$	DeMorgan
$\equiv$	$(\underline{\neg\exists x\ P(x, y)} \wedge \underline{\neg\forall z\ Q(z)}) \wedge \exists u\ f(a, u) = a$	Negation von Quantorformeln (x, z)
$\equiv$	$(\forall x\ \neg P(x, y) \wedge \exists z\ \neg Q(z)) \underline{\wedge} \exists u\ f(a, u) = a$	Kommutativität
$\equiv$	$\underline{\exists u\ f(a, u) = a} \wedge (\underline{\forall x\ \neg P(x, y)} \wedge \exists z\ \neg Q(z))$	Wirkungsbereiche ausweiten (u, x)
$\equiv$	$\exists u\ (f(a, u) = a \underline{\wedge} \forall x\ (\neg P(x, y) \underline{\wedge} \exists z\ \neg Q(z)))$	Kommutativität (2 mal)
$\equiv$	$\exists u\ (\forall x\ (\underline{\exists z\ \neg Q(z)} \wedge \neg P(x, y)) \wedge f(a, u) = a)$	Wirkungsbereich ausweiten (z)
$\equiv$	$\exists u\ (\underline{\forall x\ \exists z\ (\neg Q(z) \wedge \neg P(x, y))} \wedge f(a, u) = a)$	Wirkungsbereiche ausweiten (x, z)
$\equiv$	$\underline{\exists \mathbf{u}\ \forall \mathbf{x}\ \exists \mathbf{z}}\ (\neg Q(z) \wedge \neg P(x, y) \wedge f(a, u) = a)$	

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

Normalformen

- **Definition:** Eine PL-Formel α ist in **Negationsnormalform (NNF)** genau dann, wenn jedes Negationszeichen in α unmittelbar vor einer Primformel steht und α die Junktoren $\rightarrow$ und $\leftrightarrow$ nicht enthält.

- **Definition:** Eine PL-Formel α ist in **pränexer Normalform (PNF)** genau dann, wenn sie von der Form $Q_1x_1\ Q_2x_2\ ...\ Q_nx_n\ \beta$ ist, wobei Q_i Quantoren sind und β keine Quantoren enthält.

- **Satz:** Zu jeder PL-Formel gibt es **logisch äquivalente Formeln** in Negationsnormalform bzw. in pränexer Normalform.

Erzeugung der PNF

Die Erzeugung der pränexen Normalform geschieht in zwei Schritten:

1. Konsistente **Umbenennung** der Variablen (siehe Folie 4.27)
2. Quantoren nach links mit Hilfe der folgenden **Ersetzungsregeln** (Äquivalenzen):
   a. Ersetze $(\forall x\alpha) \wedge \beta$ durch $\forall x(\alpha \wedge \beta)$ (wegen (1) kommt x nicht frei in β vor)
   b. Ersetze $(\exists x\alpha) \wedge \beta$ durch $\exists x(\alpha \wedge \beta)$
   c. Ersetze $(\forall x\alpha) \vee \beta$ durch $\forall x(\alpha \vee \beta)$
   d. Ersetze $(\exists x\alpha) \vee \beta$ durch $\exists x(\alpha \vee \beta)$
   e. Ersetze $\neg\forall x\alpha$ durch $\exists x\neg\alpha$
   f. Ersetze $\neg\exists x\alpha$ durch $\forall x\neg\alpha$

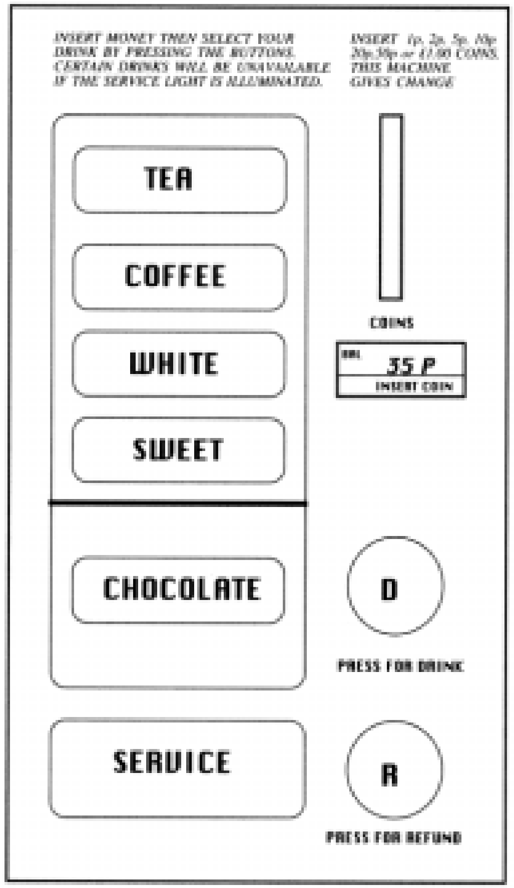
Komplexität der Prädikatenlogik erster Stufe

- Es gibt für die Prädikatenlogik erster Stufe einen **vollständigen, korrekten Kalkül** zur Herleitung allgemeingültiger Formeln.
- Die Prädikatenlogik ist **unentscheidbar**, d. h. es gibt kein Verfahren, das für eine beliebige PL-Formel feststellen kann, ob sie allgemeingültig ist.
- Die Prädikatenlogik ist **rekursiv aufzählbar**, d. h. es gibt ein Verfahren, das für eine beliebige PL-Formel feststellen kann, ob sie allgemeingültig ist, das aber im negativen Fall nicht notwendig terminiert.
- Die **natürlichen Zahlen** lassen sich in der Prädikatenlogik erster Stufe **nicht** modellieren.

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

$\Delta Abs\_State\_Machine$
$choice? : \mathbb{P} Selection\_buttons$
$d! : Drink$
$Change! : \mathrm{bag}\ British\_coin$

$choice? \in Drink$
$Value\ Balance \geq Prices\ choice?$
$\forall i : Recipe\ choice? \bullet count\ Stock\ i > 0$
$Cups > 0$
$\exists b : bag\ British\_coins \bullet (b \sqsubseteq Takings \land Value\ Balance = Value\ b + Prices\ choice?)$
$Balance' = [\![\ ]\!]$
$Stock' \uplus \{i : Recipe\ choice? \bullet i \mapsto 1\} = Stock$
$Cups' = Cups - 1$
$Change! \sqsubseteq Takings \land Value\ Balance = Value\ Change! + Prices\ choice?$
$Takings' \uplus Change! = Takings$
$Prices' = Prices$
$Service\_light' = Service\_light$
$Report\_display' = insert\ coin$
$d! = choice?$

4.3 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 (Werte von Variablen), die der Algorithmus (das Programm) an bestimmten Stellen annehmen kann, z. B.
... {pegel < max} pegel := pegel + 1; ... {0<i $\wedge$ i<10} a[i] := 42; ... {x = GGT};

Aussagen müssen beweisbar **für alle Ausführungen** des Algorithmus gelten. Im Gegensatz zum **dynamischen Testen**: Ausführen des Algorithmus für bestimmte Eingaben.

Schlussregeln für Anweisungformen erlauben logische Schlüsse über Anweisungen hinweg:
{pegel+1 $\leq$ max} pegel := pegel + 1; {pegel $\leq$ max} wegen Schlussregel für Zuweisungen

Verifikation beweist, dass

- an einer bestimmten Programmstelle eine Aussage über Zustände gilt,
- vor und nach Ausführung eines Programmstückes eine **Invariante** gilt,
- ein Algorithmus **aus jeder zulässigen Eingabe die geforderte Ausgabe** berechnet, z. B.
  { a, b $\in \mathbb{N}$ }   Euklidischer Algorithmus   { x ist GGT von a, b }
- eine **Schleife terminiert.**

Ein **Algorithmus und dieAussagen dazu sollen zusammen konstruiert** werden.

Vorschau auf Konzepte

Aussagen charakterisieren Zustände der Ausführung

Algorithmen in informeller Notation

Schlussregeln für Anweisungsformen anwenden

Invariante von Schleifen (und anderen Konstrukten)

Schlussketten über Anweisungen hinweg verifizieren Aussagen

Nachweis der **Terminierung von Schleifen**

Beispiel zur Vorschau: Verifikation des Algorithmus 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 ∧ x>0 ∧ y>0 }
a := x; b := y;
{ INV: G ist ggT von a und b ∧ a>0 ∧ b>0 }
solange a ≠ b wiederhole
    { INV ∧ a ≠ b }
    falls a > b :
            { G ist ggT von a und b ∧ a>0 ∧ b>0 ∧ a>b } →
            { G ist ggT von a-b und b ∧ a-b>0 ∧ b>0 }
            a := a - b
            { INV }
    sonst
            { G ist ggT von a und b ∧ a>0 ∧ b>0 ∧ b>a } →
            { G ist ggT von a und b-a ∧ a>0 ∧ b-a>0 }
            b := b - a
            { INV }
    { INV }
{ INV ∧ a = b } →
{ a = G }
```

Terminierung der Schleife:

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

Aussage charakterisiert Programmzustände

Eine **Aussage P** an einer **Stelle in einem Algorithmus** (Programm) vor oder nach einer Anweisung ... S_1 {P} S_2 ...

charakterisiert alle Zustände, die das Programm an dieser Stelle **bei irgendeiner Ausführung** annehmen kann. P wird über **Variable des Algorithmus** formuliert.

Z. B. ... $\{ 0 \leq i \wedge i < 10 \}$ a[i] := 42; ...
Bei jeder Ausführung liegt der Wert von i im angegebenen Intervall.
Eine Aussage über andere Variablen wird hier nicht gemacht.

Nur die **gerade interessierende Eigenschaften der Zustände** werden beschrieben.

Aussagen können unterschiedlich scharf formuliert werden:

$\{ f \}$	kein Zustand erfüllt P, Stelle nicht erreichbar
$\{ 0 \leq i \wedge i < 2 \wedge a[i] > 0 \}$	schärfer; evtl. weniger Zustände; schwieriger zu verifizieren
$\{ 0 \leq i \wedge i < 2 \}$	
$\{ 0 \leq i \wedge i < 10 \}$	
$\{ 0 \leq i \}$	schwächer; evtl. mehr Zustände; leichter zu verifizieren
$\{ w \}$	beliebige Zustände erfüllen P

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 :`** ...

Vor- und Nachbedingung von Anweisungen

Aussage Q charakterisiert die Zustände, die eine Ausführung zwischen den Anweisungen A_1 und A_2 annehmen kann:

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

Q ist **Nachbedingung** von A_1 und **Vorbedingung** 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 der Anweisung S die Aussage 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 }

Zuweisungsregel

Hoare'scher Kalkül definiert **für jede Anweisungsform** eine **Schlussregel.**

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

$$\boxed{\{ P_{[x/e]} \}\ x := e\ \{ P \}}$$

Wenn vor der Ausführung $P_{[x/e]}$ gilt (P wobei x durch e substituiert ist),
gilt nach der Ausführung der Zuweisung P.

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

$\{ P_{[x/e]} \}$	$x := e$	$\{ P \}$	
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. $\{ i \geq 0\} \leftrightarrow \{ i + 1 > 0 \}$	$i := i + 1$	$\{ i > 0 \}$	passend umformen
7. $\{ i = 2 \} \leftrightarrow \{ i + 1 = 3 \}$	$i := i + 1$	$\{ i = 3 \}$	passend umformen
8. $\{ \text{wahr} \} \leftrightarrow \{ 1 = 1 \}$	$x := 1$	$\{ x = 1 \}$	passend umformen
9. $\{ z = 5 \} \leftrightarrow$ $\{ z = 5 \wedge 1 = 1 \}$	$x := 1$	$\{ z = 5 \wedge x = 1 \}$	passend umformen

Schlussregeln für Sequenz

Sequenzregel:

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

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:

$$\frac{\{ x>0 \wedge y>0 \}\quad \texttt{a := x;}\quad \{ a>0 \wedge y>0 \} \qquad \{ a>0 \wedge y>0 \}\quad \texttt{b := y;}\quad \{ a>0 \wedge b>0 \}}{\{ x>0 \wedge y>0 \}\quad \texttt{a := x; b := y;}\quad \{ a>0 \wedge b>0 \}}$$

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

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

Verschärfung der Vorbedingung

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

Beispiel:

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

im Algorithmus können Implikationen in Ausführungsrichtung eingefügt werden:

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

Regel für 2-seitige Alternative

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

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

Beispiel:

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

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

{ $a>0 \wedge b>0$ }

Regel für bedingte Anweisung

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

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

Beispiel:

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

im Algorithmus:

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

Aufrufregel

Der **Unteralgorithmus** UA habe **keine Parameter** und liefere **kein Ergebnis**.
Seine **Wirkung auf globale Variable** sei spezifiziert durch
die **Vorbedingung P und die Nachbedingung Q**.

Dann gilt für einen **Aufruf** von UA die Schlussregel

$$\{ P \}\ UA()\ \{ Q \}$$

(Ohne Parameter und Ergebnis ist diese Regel nur von sehr begrenztem Nutzen.)

Schleifenregel

Wiederholung, Schleife:

$$\frac{\{\ INV \wedge B\ \}\ \ S\ \ \{\ INV\ \}}{\{\ \mathbf{INV}\ \}\ \textbf{solange B wiederhole S}\ \{\ \mathbf{INV} \wedge \neg\, \mathbf{B}\ \}}$$

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

$\{\ \mathbf{INV} \wedge b > 0\ \} \leftrightarrow \{\ z \cdot a \cdot a^{b-1} = x^y \wedge (b\text{-}1) \geq 0\ \}$
b := b - 1;
$\{\ z \cdot a \cdot a^b = x^y \wedge b \geq 0\ \}$
z := z · a
{ **INV** }

$\{\ \mathbf{INV} \wedge b \leq 0\ \} \leftrightarrow \{\ z \cdot a^b = x^y \wedge b = 0\ \} \rightarrow \{\ z = x^y\ \}$

Terminierung von Schleifen

Die **Terminierung einer Schleife** solange B wiederhole S
muss separat nachgewiesen werden:

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)

1.

	$\{ a > 0 \wedge b > 0\}$
Schleife1	solange $a \neq b$ wiederhole
Schleife2	solange $a > b$ wiederhole
	a := a - b;
Schleife3	solange $a < b$ wiederhole
	b := b - a

terminiert weil:

a. INV = $a > 0 \wedge b > 0$ ist Invariante für jede der 3 Schleifen, denn

	{INV}
Schleife1	solange $a \neq b$ wiederhole $\{INV \wedge a \neq b\}$
Schleife2	solange $a > b$ wiederhole $\{INV \wedge a > b\}\rightarrow$
	$\{a-b > 0 \wedge b > 0\}$ a := a - b; {INV}
	{INV}
Schleife3	solange $a < b$ wiederhole $\{INV \wedge a < b\}\rightarrow$
	$\{a > 0 \wedge b-a > 0\}$ b := b - a {INV}
	{INV}
	{INV}

b. *Schleife2:* a fällt monoton, weil $b > 0$; a ist begrenzt, weil $a > 0$.
Schleife3: b fällt monoton, weil $a > 0$; b ist begrenzt, weil $b > 0$.
Schleife1: a+b fällt monoton, weil wg. $a \neq b$ Schl. 2 o. 3 mind. 1x iteriert wird;
a+b begrenzt, wg. INV.

Beispiele zur Terminierung (2)

2.

	$\{ a > 0 \wedge b > 0\}$
Schleife1	solange $a \neq b$ wiederhole
Schleife2	solange $a \geq b$ wiederhole
	$a := a - b;$
Schleife3	solange $a < b$ wiederhole
	$b := b - a$

terminiert nicht immer:

$a > 0$ ist nicht invariant in den Schleifen.

Die Nachbedingung von Schleife 2 ist $a < b \wedge a \geq 0$.

Schleife 3 kann erreicht werden im Zustand R: $a = 0$, z.B. wenn initial $a = 2*b$ gilt.

$a = 0 \wedge a < b$ ist invariant in Schleife 3 und $a < b$ ist die Schleifenbedingung.

$$\{a = 0 \wedge a < b\} \rightarrow \{a = 0 \wedge a < b\text{-}a\}\ b := b - a\ \{a = 0 \wedge a < b\}$$

Beispiele zur Terminierung (3)

3.

$\{ n \in \mathbb{N} \wedge n > 1 \}$
solange n > 1 wiederhole
    falls n gerade:
        n := n / 2
    sonst n := 3 * n + 1

Terminierung / Nichtterminierung ist unbewiesen;
einige Ausführungen mit Anfangswerten n:

n	
2	1
3	10 5 16 8 4 2 1
4	2 1
5	16 8 4 2 1
6	3 10 5 16 8 4 2 1
7	22 11 34 17 52 26 13 50 25 76 38 19 ...

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:

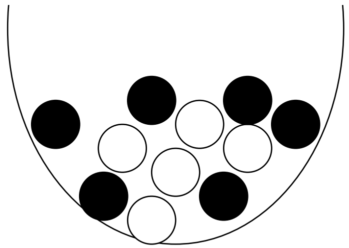
wirf beide weg und

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

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

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

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

4. Modellierung mit Graphen

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

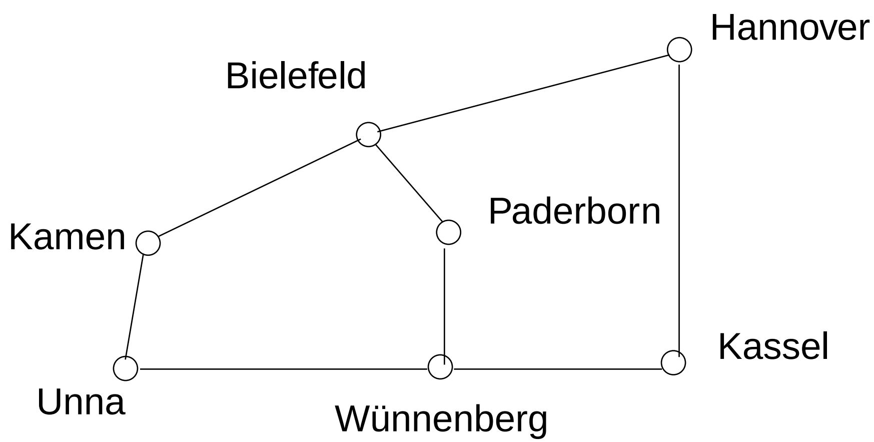
Graphen eignen sich zur Modellierung für ein **breites Aufgabenspektrum**.

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

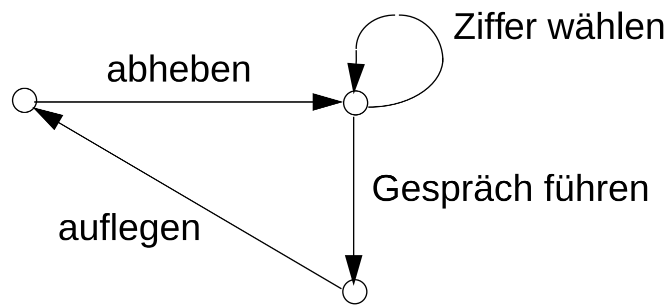
- **Knoten**: Eine Menge gleichartiger Objekte
- **Kanten**: Beziehung zwischen je zwei Objekten, 2-stellige Relation über Knoten

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

5.1 Grundlegende Definitionen
Gerichteter Graph

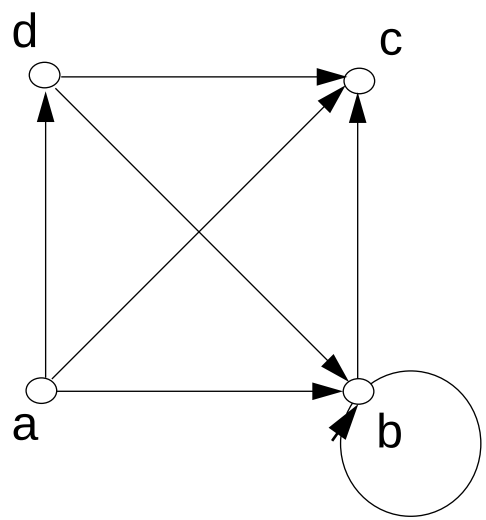
Ein **gerichteter Graph** G = (V, E) hat eine endliche **Menge V von Knoten** und eine **Menge E gerichteter Kanten**, mit $E \subseteq V \times V$.

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

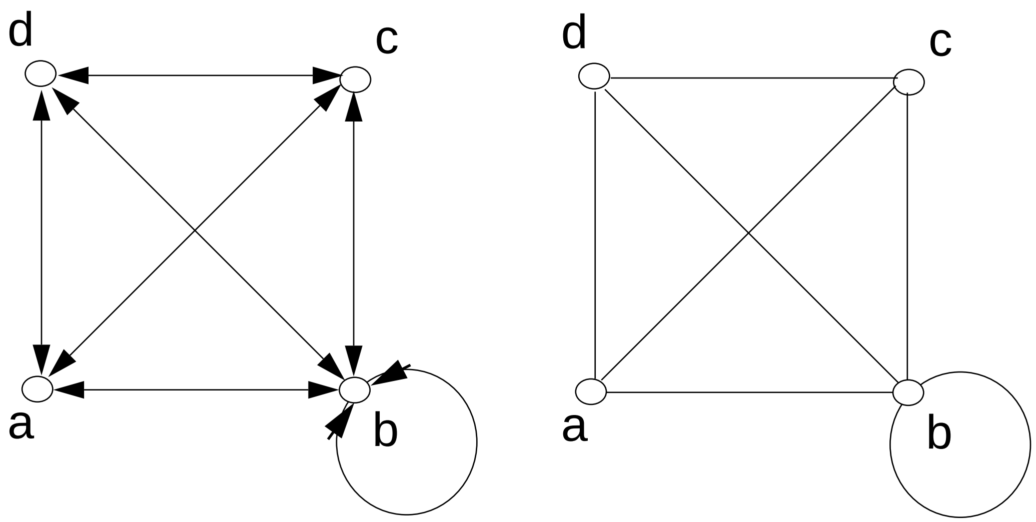
- endliche Graphen mit **endlichen Knotenmengen**,
- einfache Kanten:
  - eine **Kante verbindet nicht mehr als zwei Knoten**,
  - von Knoten x nach Knoten y gibt es höchstens eine Kante

Multigraph: Es kann mehr als eine Kante von Knoten x nach Knoten y geben (siehe Mod-5.7)

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

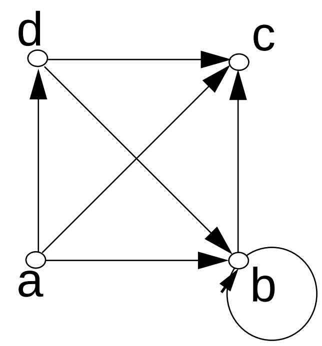
abstrakt:

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:

$$AM(i, j) = (i, j) \in E$$

	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

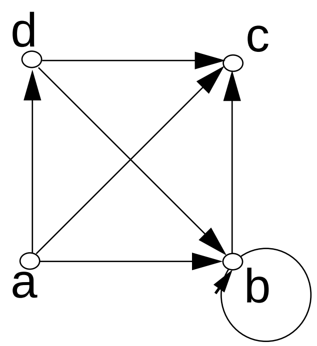
a	**(b, c, d)**
b	**(b, c)**
c	**()**
d	**(b, c)**

Ungerichtete Graphen als gerichtete Graphen mit symmetrischer Kantenmenge darstellen

Teilgraph

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.)

Graph G = (V, E):



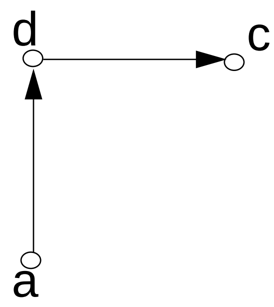
V = {a, b, c, d}

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

Teilgraph G' = (V', E') zu G

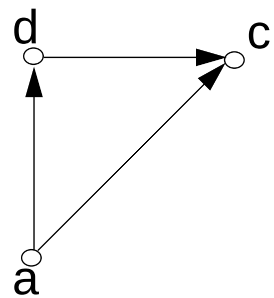
V' = {a, c, d}

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



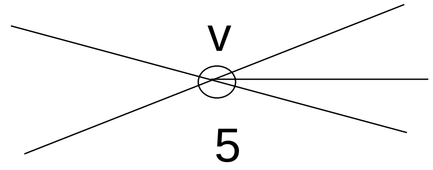
Zu einem Graphen G = (V, E) **induziert** eine Teilmenge der Knoten V' $\subseteq$ V den **Teilgraphen** G' = (V', E'), wobei E' alle Kanten aus E enthält, deren Enden in V' liegen.

Teilgraph G" zu G
durch V" = {a, c, d} **induziert**

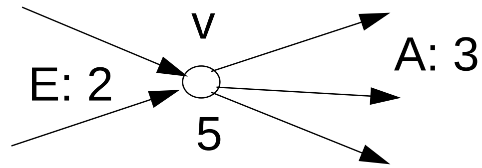


Knotengrad

Sei G = (V, E) ein **ungerichteter** Graph:
Der **Grad** eines Knotens v ist die
Anzahl der Kanten {x, v}, die in v enden.



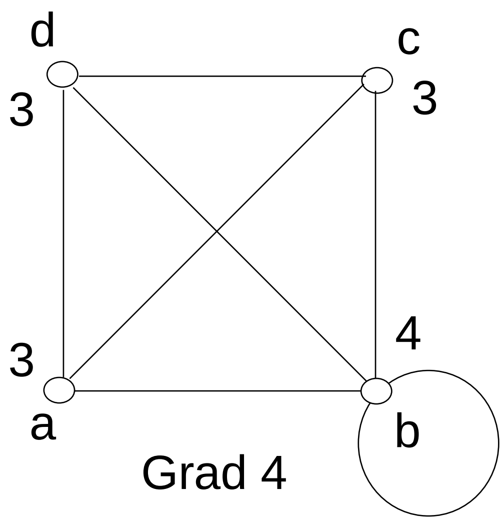
Sei G = (V, E) ein **gerichteter** Graph:
Der **Eingangsgrad** eines Knotens v ist die
Anzahl der Kanten (x, v) $\in$ E, die in v münden.

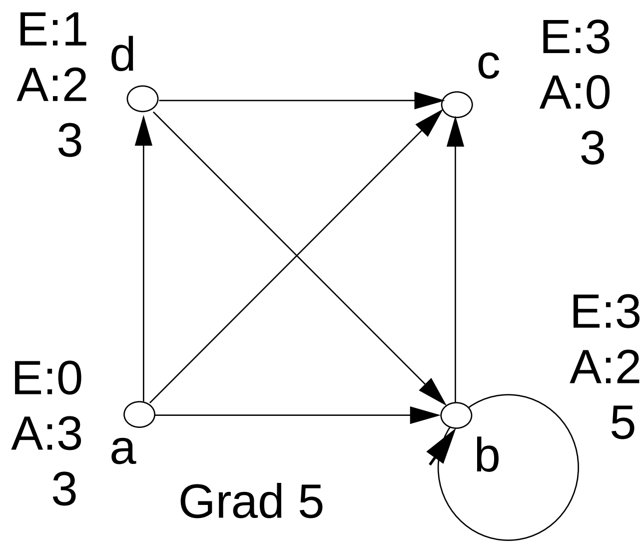


Der **Ausgangsgrad** eines Knotens v ist die
Anzahl der Kanten (v, x) $\in$ E, die von v ausgehen.

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

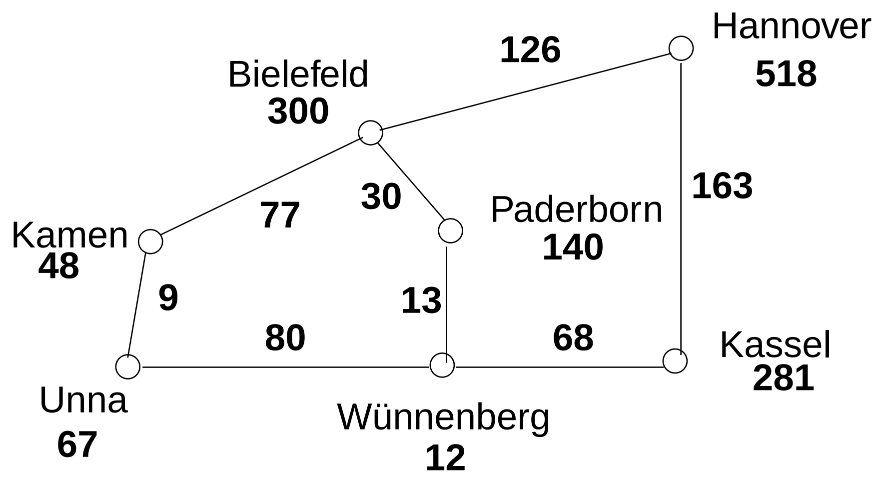
MV : $\mathbf{V} \to$ WV,

z.B. EinwohnerzahlTsnd: $\mathbf{V} \to \mathbb{N}$

Kantenmarkierung

ME :$\mathbf{E} \to$ WE,

z.B. EntfernungKm: $\mathbf{E} \to \mathbb{N}$

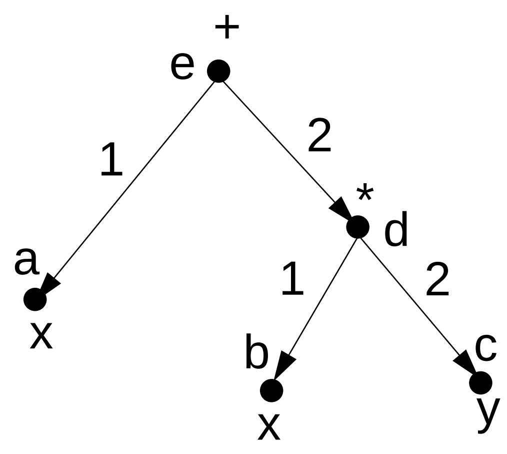


Spezielle Kantenmarkierungen

Ordnung von Kanten:

$E \to \mathbb{N}$

legt die **Reihenfolge der Kanten** fest, die von einem Knoten ausgehen, z. B. im Kantorowitsch-Baum von links nach rechts.

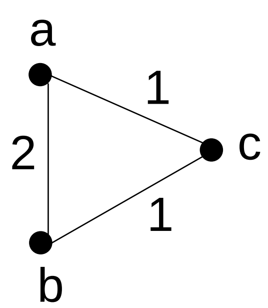


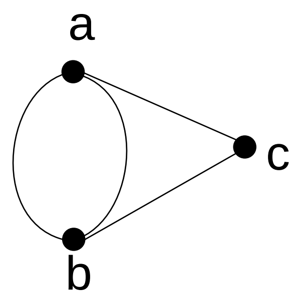
V := {a, b, c, d, e}
MV := {(a, x), (b, x), (c, y), (d,*), (e, +)}
ME := {((e,a), 1), ((e,d), 2), ((d,b), 1), ((d,c),2)}

Anzahl von Kanten:

$E \to \mathbb{N}$

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.

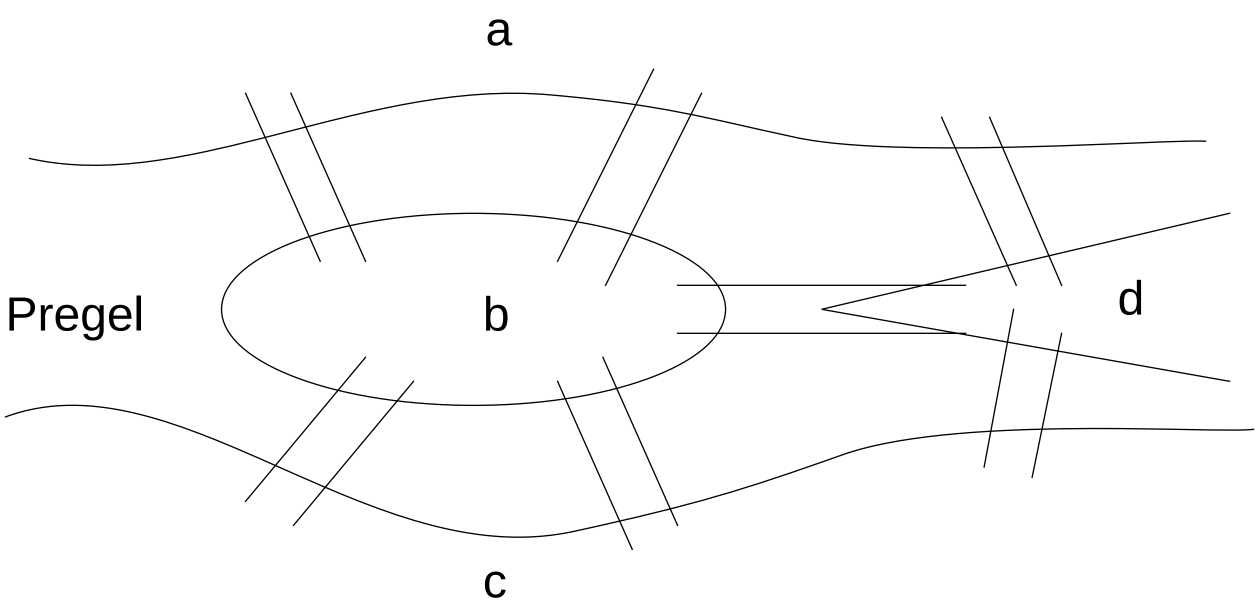




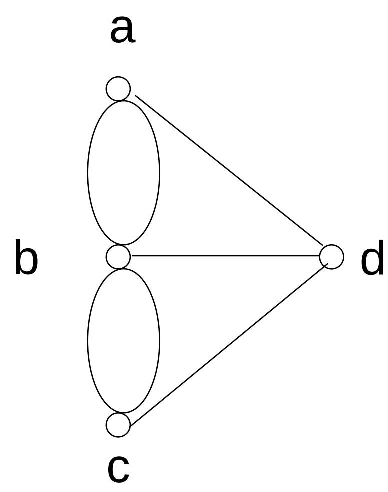
ME := {({a, b}, 2), ({a, c}, 1), ({b, c}, 1)}

5.2 Wegeprobleme

Beispiel: **Königsberger Brückenproblem** (Euler, 1736)



Skizze von Königsberg



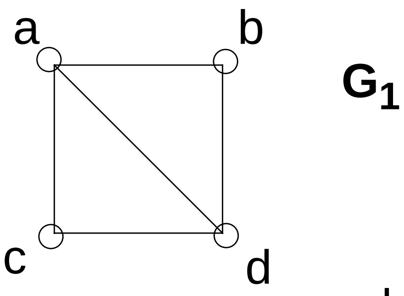
Multigraph dazu

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

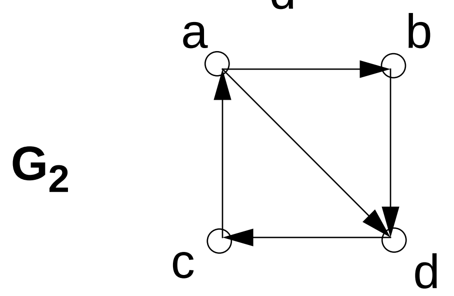
b. Gibt es einen Weg, der jede der 7 Brücken genau einmal überquert?

Wege und Kreise

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



Entsprechend für gerichtete Graphen:
mit $(v_i, v_{i+1}) \in E$ für i = 0,..., n-1



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.

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

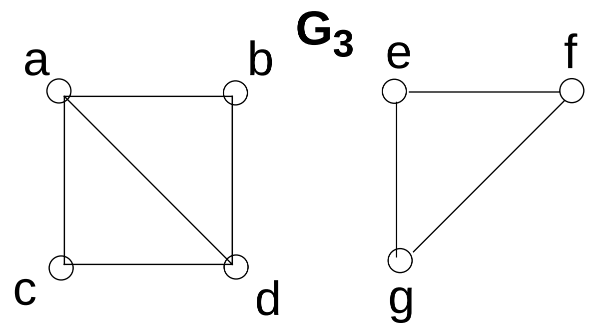
Zusammenhang in Graphen

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

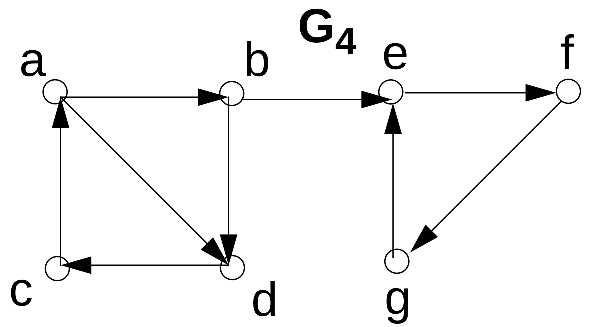
Ein gerichteter Graph heißt unter derselben Bedingung **stark zusammenhängend**.

Ein Teilgraph G' = (V', E') eines ungerichteten (gerichteten) Graphen G = (V, E) heißt **(starke) Zusammenhangskomponente**, wenn

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



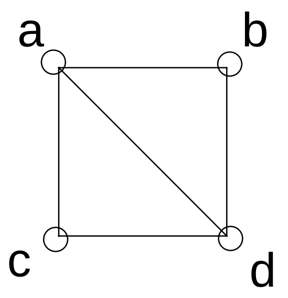
Zusammenhangskomponenten sind also **maximale Teilgraphen, die zusammenhängend** sind.



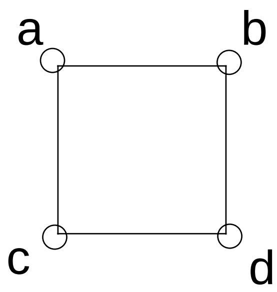
Spezielle Wege und Kreise

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

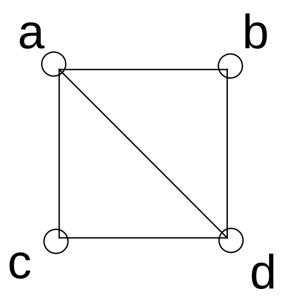
Ein **Euler-Weg** bzw. ein **Euler-Kreis** in G ist ein Weg, der **jede Kante aus E genau einmal** enthält.



(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.

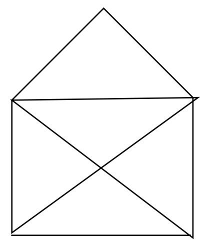
G hat einen **Euler-Weg**, der kein Kreis ist, genau dann,
wenn G genau **2 Knoten mit ungeradem Grad** hat.

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

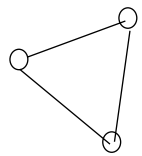
Wegeprobleme mit Euler-Wegen

1. Königsberger Brückenproblem (Mod-5.8):
   Euler-Weg, Euler-Kreis

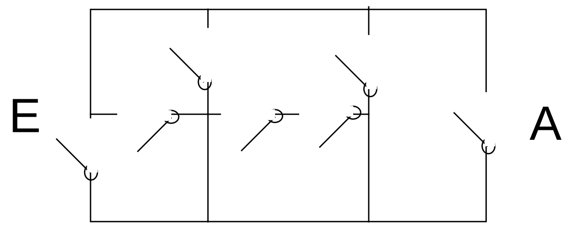
2. Kann man diese Figur in einem Zuge zeichnen?



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?

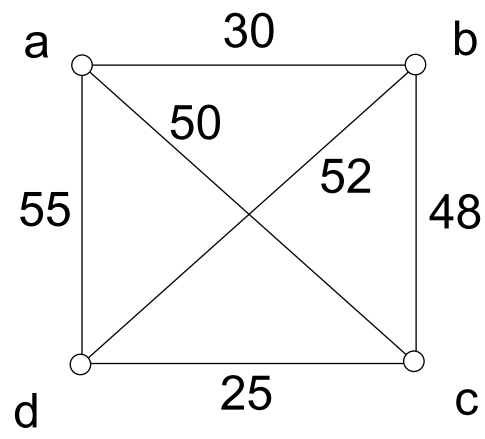


4. Planen Sie ein Gruselkabinett:
   Ein Haus mit n > 1 Räumen, 1 Eingangstür, eine Ausgangstür, beliebig vielen Innentüren. Jede Tür schließt nach Durchgehen endgültig. Die Besucher gehen einzeln durch das Haus. Es soll niemand eingesperrt werden.

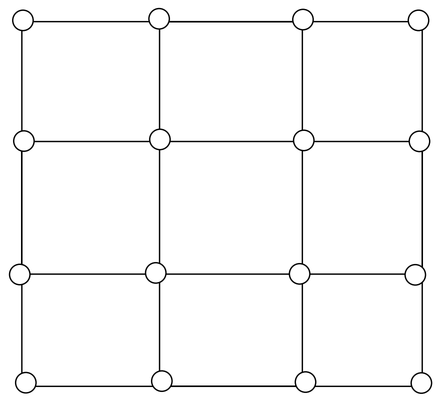


Wegeprobleme mit Hamilton-Kreisen

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?



5.3 Verbindungsprobleme

Modellierung durch Graphen wie bei Wegeproblemen (Abschnitt 5.2), 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**.

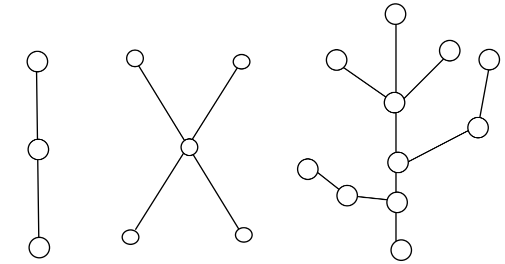
In Bäumen heißen **Knoten mit Grad 1 Blätter**.

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

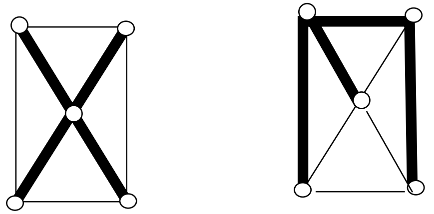
Ein zusammenhängender Teilgraph von G,
der jeden Knoten aus V enthält und ein Baum ist,
heißt **Spannbaum** zu G.

Bäume

G_1 **G_2** **G_3**



G_4 **G_4**

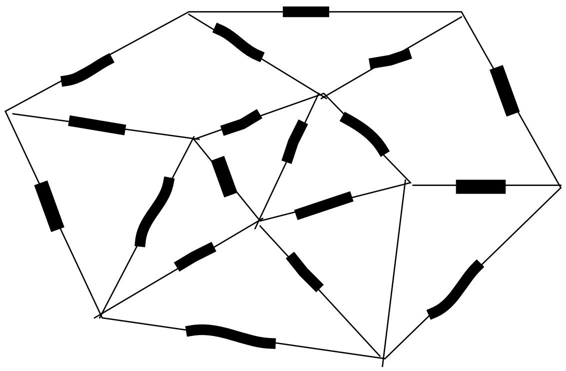


2 Spannbäume zu
demselben Graphen G_4

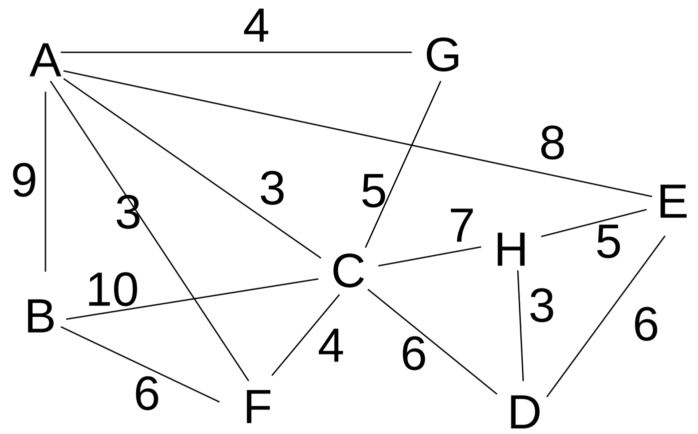
Modellierung mit Spannbäumen zu Graphen

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.

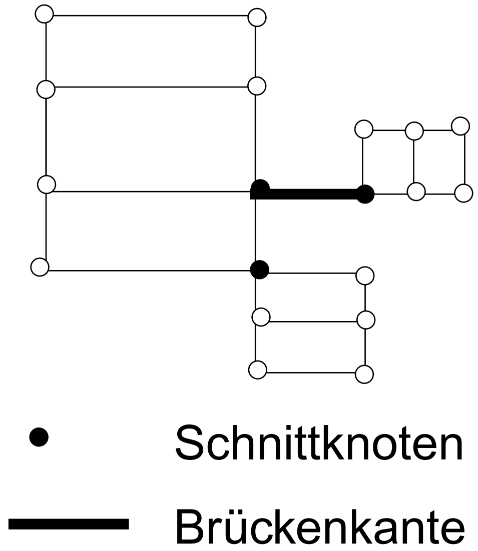
v ist ein **Schnittknoten** in G, wenn G **ohne v nicht mehr zusammenhängend** ist.

e ist eine **Brückenkante** in G, 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.

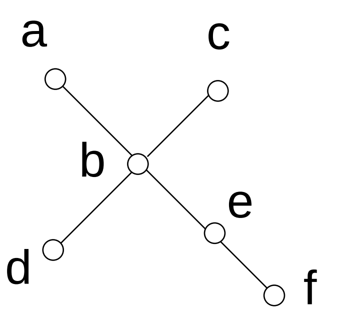
G ist genau dann **orientierbar**, wenn G **keine Brückenkante** hat.

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?

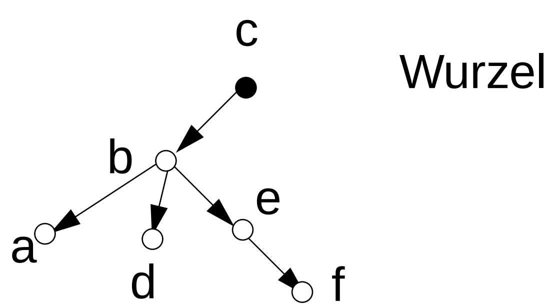


5.4 Modellierung mit Bäumen

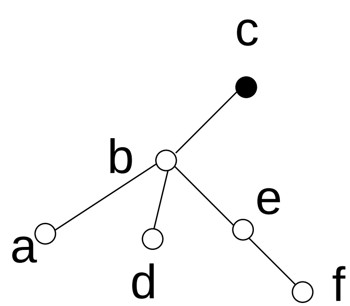
In einem **ungerichteten Baum** gibt es
zwischen zwei beliebigen Knoten genau einen Weg.



Ein gerichteter, **azyklischer** Graph G ist ein **gerichteter Baum**,
wenn alle Knoten einen **Eingangsgrad** $\leq$ **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**.

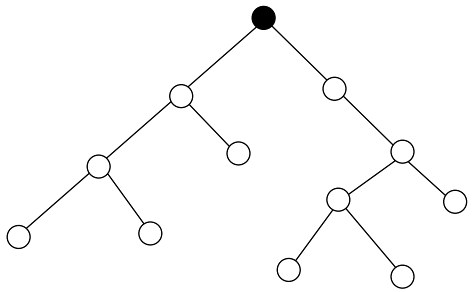
In einem gewurzelten Baum ist die **Höhe eines Knotens v**
die größte Länge eines Weges von v zu einem Blatt.
Die Höhe der Wurzel heißt **Höhe des Baumes**.

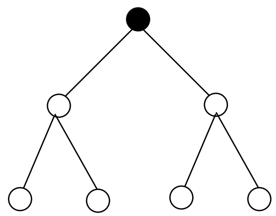
Knoten, die weder Wurzel noch Blatt sind heißen
innere Knoten.

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.





Höhe 2

Knoten: 7

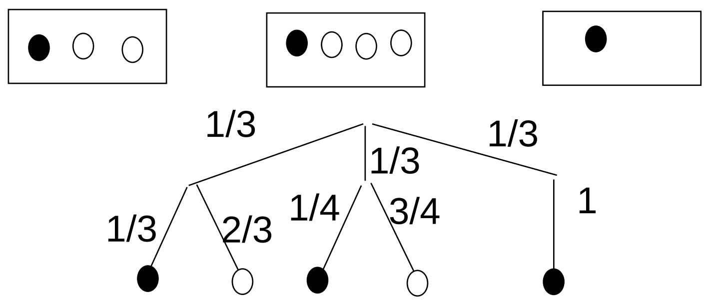
Blätter: 4

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

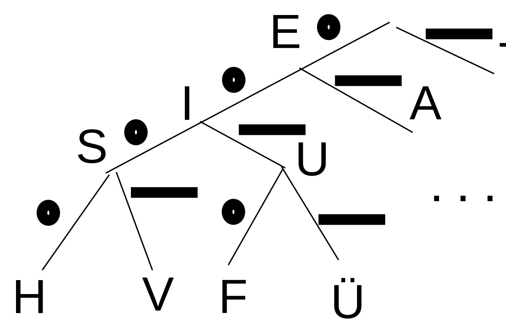
Modellierung von Entscheidungsbäumen

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

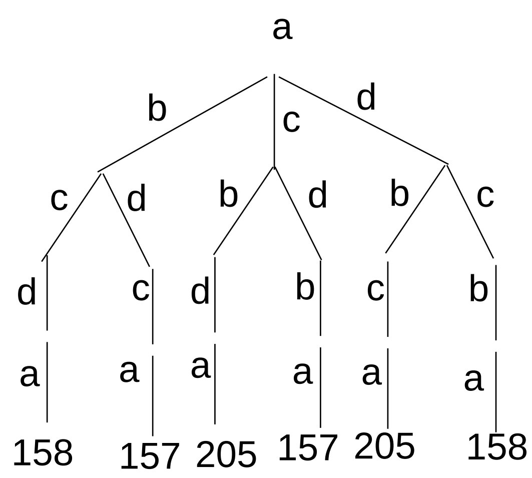
1. **Wahrscheinlichkeiten**,
   z. B. erst Schachtel, dann Kugel ziehen:



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



3. **Lösungsbaum** für kombinatorische Probleme,
   z. B. Traveling Salesman's Problem (Mod-5.13)
   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-5.21)
  Vererbungshierarchie: Unterklassen erben von ihrer Oberklasse
- **Objektbaum**: Objekt enthält (Referenzen auf) Teilobjekte
- **Kantorowitsch-Baum**: Operator mit seinen Operanden (Mod-5.22)
- **Strukturbaum**: (Programm-)Struktur definiert durch eine kontextfreie Grammatik (Mod-5.23)

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, Folie 6.19ff):

Knoten: **Klassen**

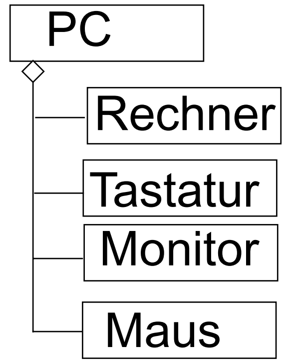
Kanten: definieren, **aus welcher Art von Objekten** ein Objekt **besteht**

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



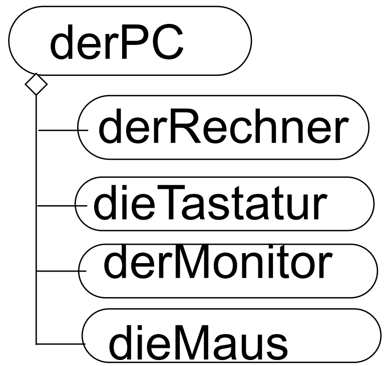
Objektbaum im Objektdiagramm (fast UML):

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**

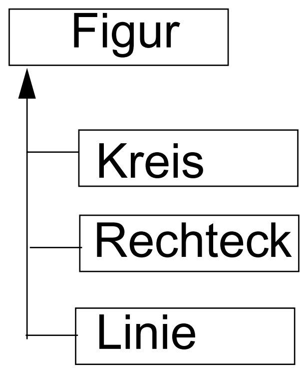
Kanten: Unterklasse **erbt von** -> Oberklasse

Oberklasse **ist Abstraktion** <- ihrer Unterklassen

Kanten sind zur Wurzel hin gerichtet

Baum bei Einfachvererbung (Java)

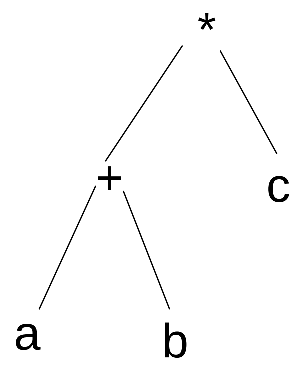
azyklischer Graph bei Mehrfachvererbung (C++)



Kantorowitsch-Bäume

Darstellung der Struktur von Termen, Formeln, Ausdrücken
(siehe Mod-3.6)

(a + b) * c



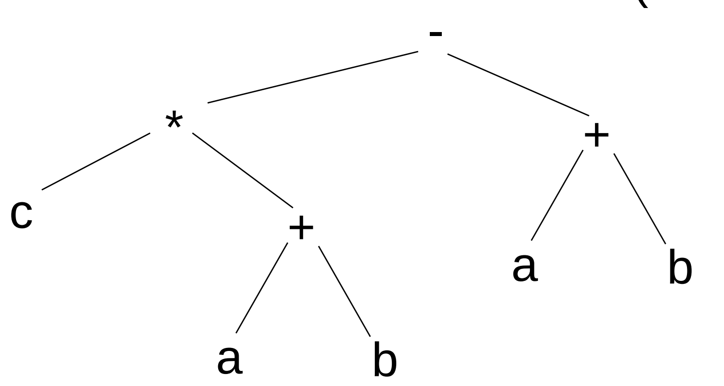
Knoten: Operator, Blattoperand

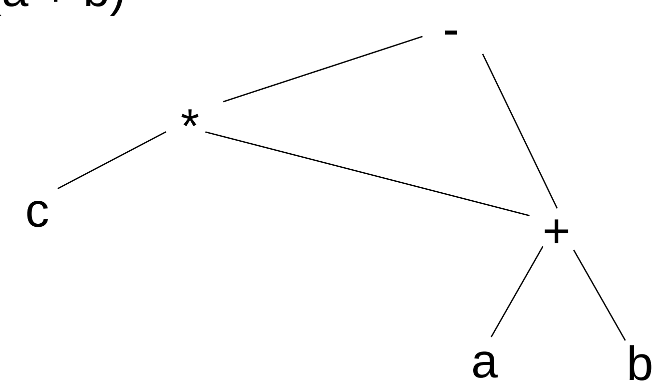
Kanten: Verbindung zu den Operanden eines Operators
Die Kanten sind geordnet (Kantenmarkierung):
erster, zweiter, ... Operand

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

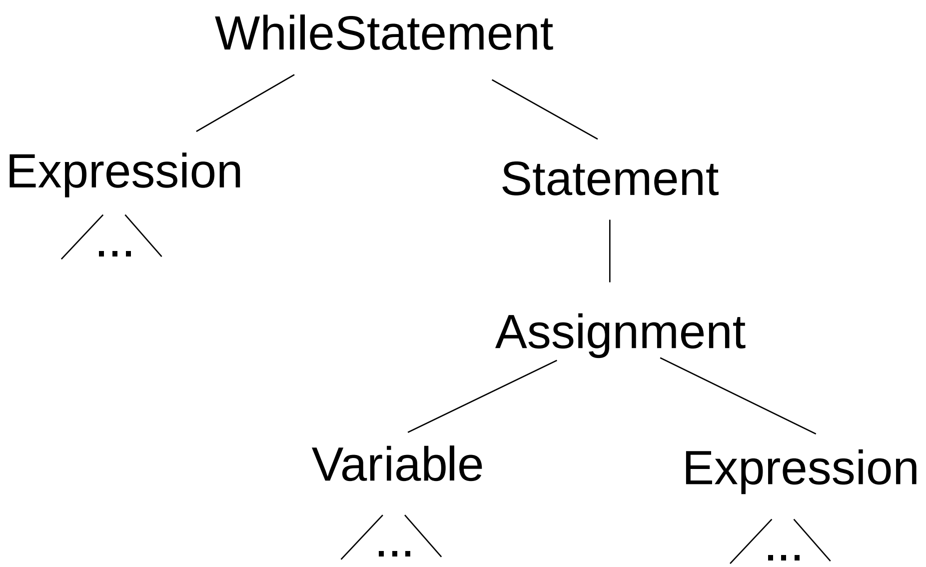
Kontextfreie Grammatiken definieren die Struktur von Programmen, Texten oder Daten.
Ein Programm, Text oder strukturierte Daten werden als Strukturbaum dargestellt.

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:




```
while(i<10)  a[i]  =  2*i  ;
```

Produktionen aus der kontextfreien Grammatik:

Statement ::= Assigment

Statement ::= WhileStatement

...

WhileStatement ::= Expression Statement

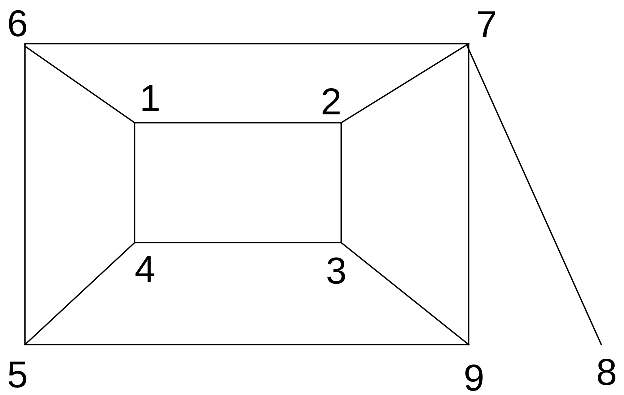
Assignment ::= Variable Expression

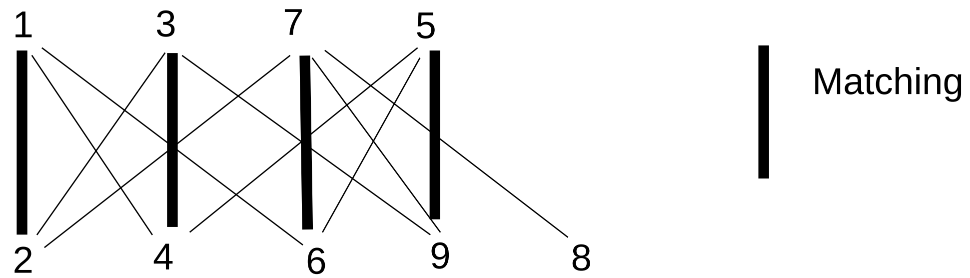
5.5 Zuordnungsprobleme

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** $\mathbf{V = V_1} \cup \mathbf{V_2}$ zerlegt werden kann, so dass jede Kante zwei Knoten aus verschiedenen Teilmengen verbindet.

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

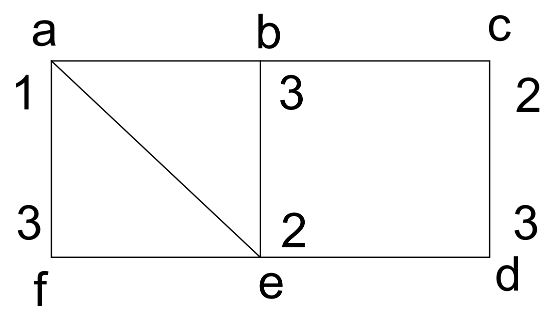
Mann - Frau   Aufgabe - Bearbeiter   Verbraucher - Produkte

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 ≤ 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

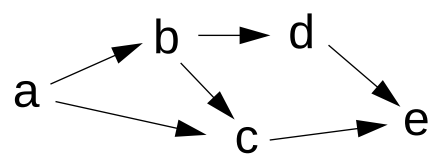
5.6 Abhängigkeitsprobleme

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

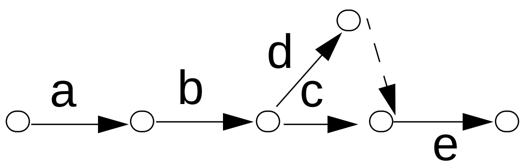


Anwendungen:

- **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

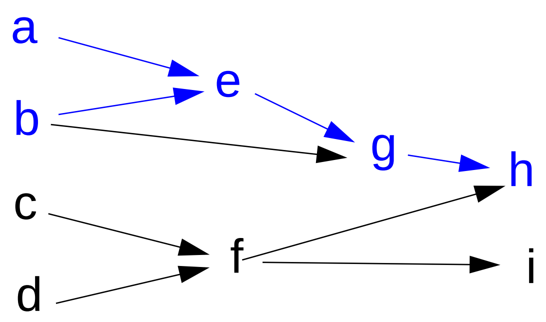
Duale Modellierung:
Knoten: Ereignis, Anfang und Ende einer Operation
Kante: Operation, ggf. mit Dauer markiert



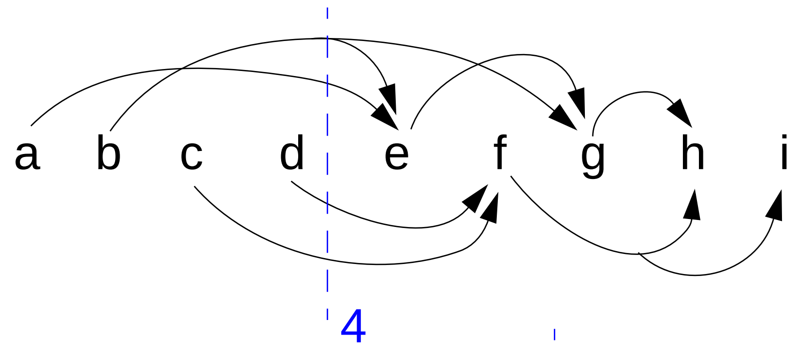
Anordnung von Abhängigkeitsgraphen

Anordnungsaufgaben:
gegebener Abhängigkeitsgraph

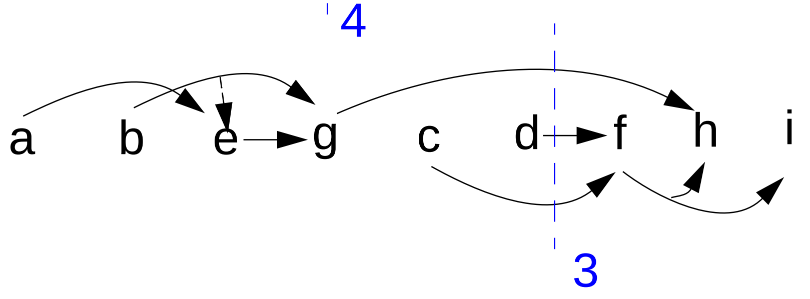
kritischer Pfad



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

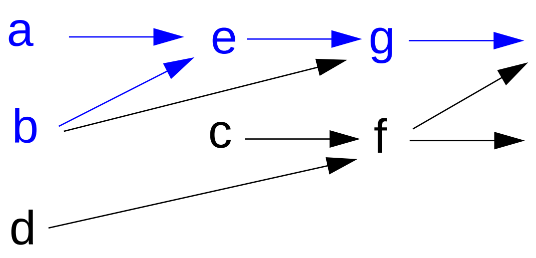


Meist sollen **Randbedingungen** erfüllt werden,
z. B. geringste Anzahl gleichzeitig benötigter Zwischenergebnisse im Speicher



parallele Anordnung mit
beschränkter Parallelität 3

Länge: 4 Schritte (Operationen)



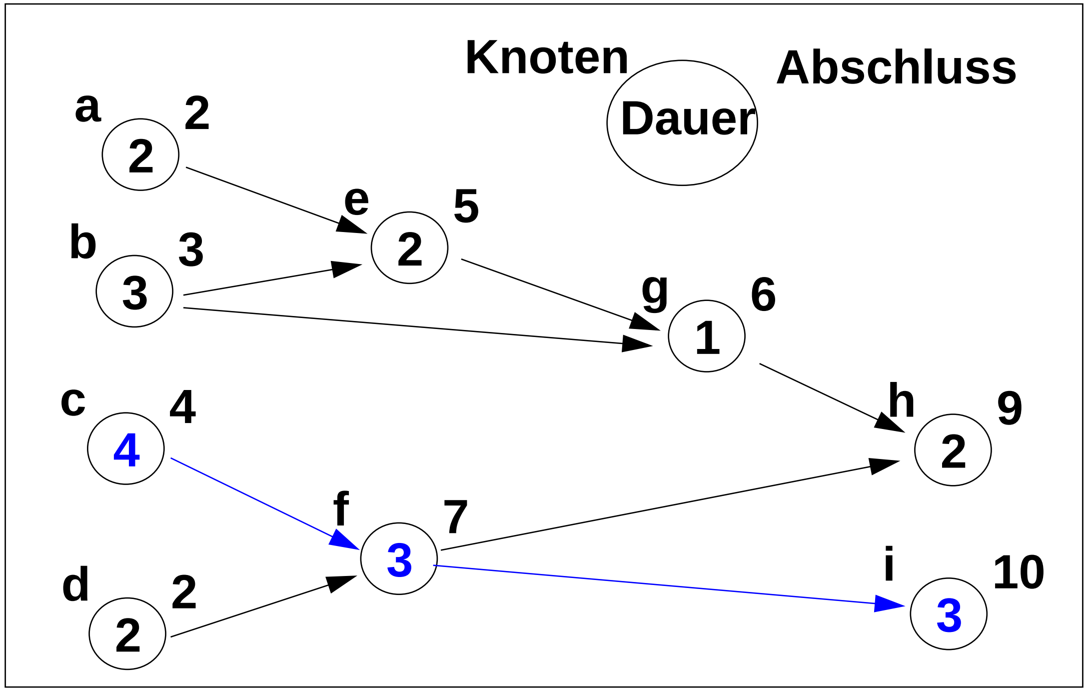
Operationen unterschiedlicher Dauer

Zwei **Knotenmarkierungen**:

Dauer der Operation und

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

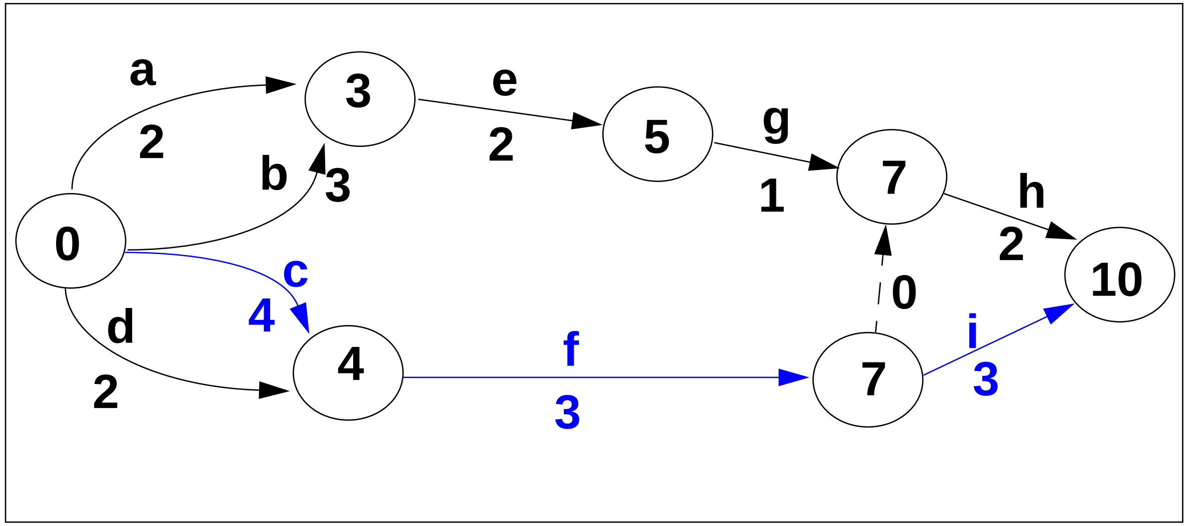
Kritischer Pfad gemäß maximaler
Summe der Dauer der Operationen



Duale Modellierung:

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

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



Ablaufgraphen

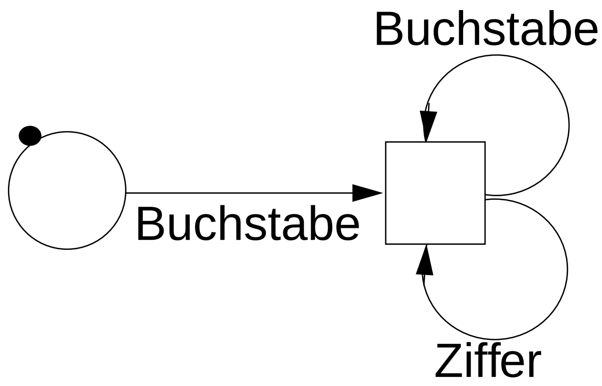
Gerichteter Graph (auch zyklisch) modelliert Abläufe.

Knoten: Verzweigungsstelle, Zustand
Kanten: Fortsetzungsmöglichkeit

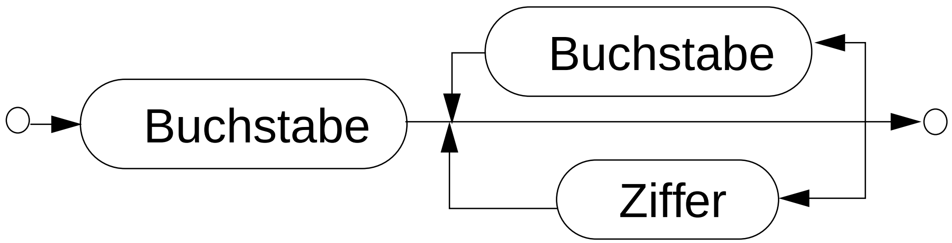
Jeder **Weg durch den Graphen** beschreibt einen **potenziellen Ablauf**
Die **Folge der Markierungen eines Weges** kann einen **Satz einer Sprache** modellieren.

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 a->b:„auf a kann b folgen“
  dual zum endlichen Automaten



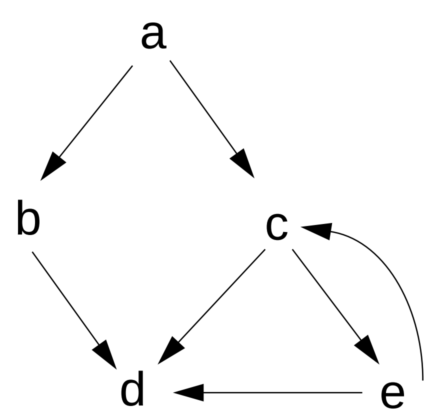
- **Aufrufgraphen** (siehe Mod-5.30)
- **Ablaufgraphen** (siehe Mod-5.31)

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.

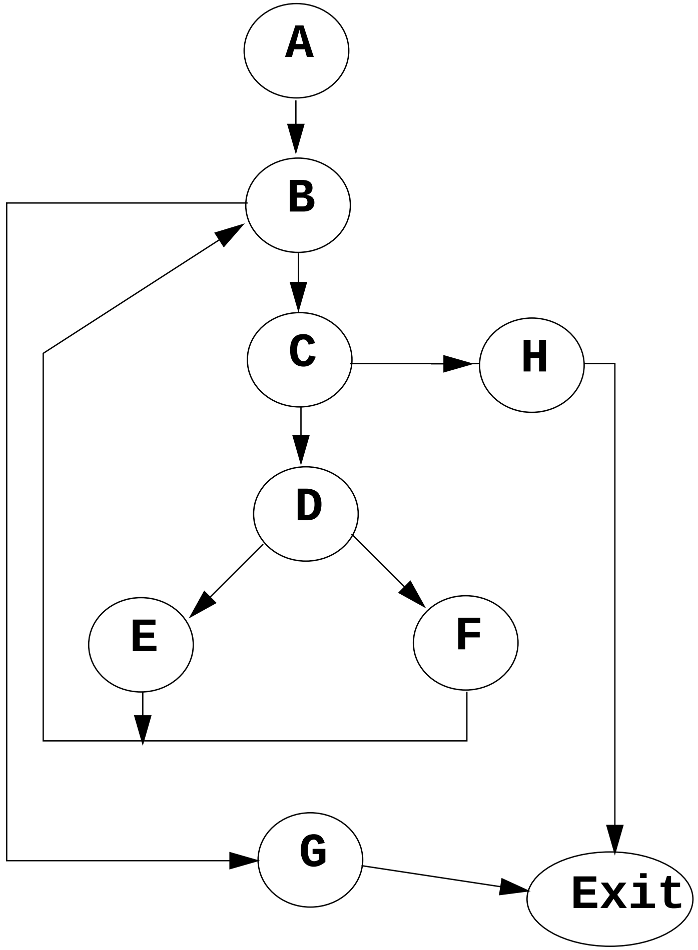
- Welche Funktionen sind nicht rekursiv?
- Welche Funktionen sind nicht (mehr) erreichbar?
- Indirekte Wirkung von Aufrufen,
  z. B. nur e verändere eine globale Variable x;
  welche Aufrufe lassen x garantiert unverändert? b, d

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

6 Modellierung von Strukturen
6.1 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`

Produktionsmenge P =

Name *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 | w $\in$ T* und S =>* w }

Grammatik auf Mod-6.2 **definiert** geschachtelte Folgen paariger Klammern als **Sprache**:

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

Ableitung des Satzes (( ) ( )) :

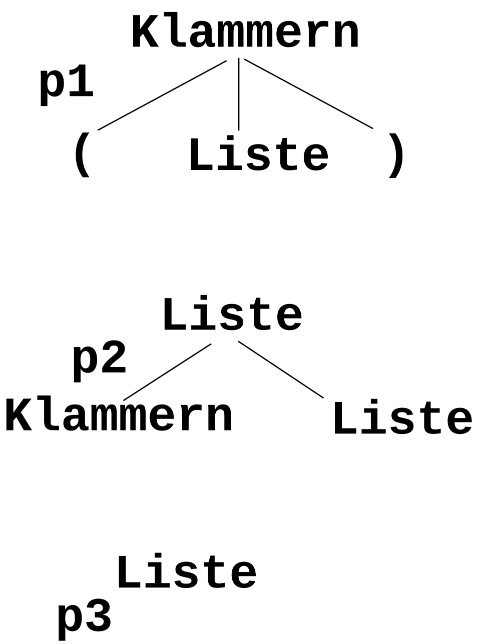
S = Klammern
=> (Liste)
=> (Klammern Liste)
=> (Klammern Klammern Liste)
=> (Klammern (Liste) Liste)
=> ((Liste) (Liste) Liste)
=> (( ) (Liste) Liste)
=> (( ) ( ) Liste)
=> (( ) ( ))

Ableitungsbäume

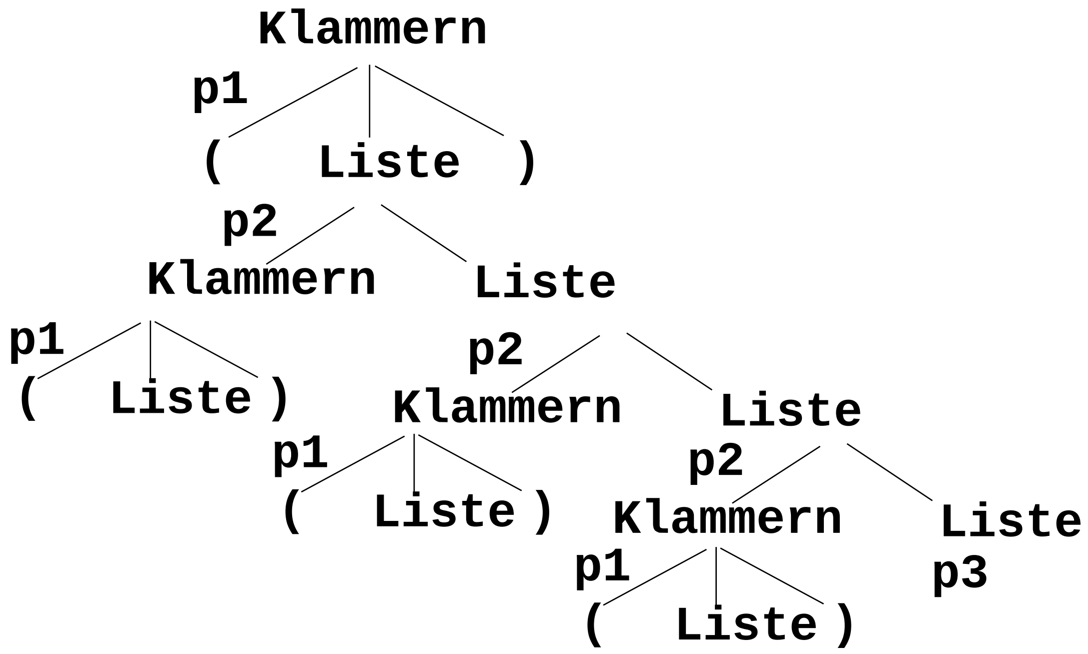
Jede Ableitung kann man als **gewurzelten Baum** darstellen:

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 ::= ’-’

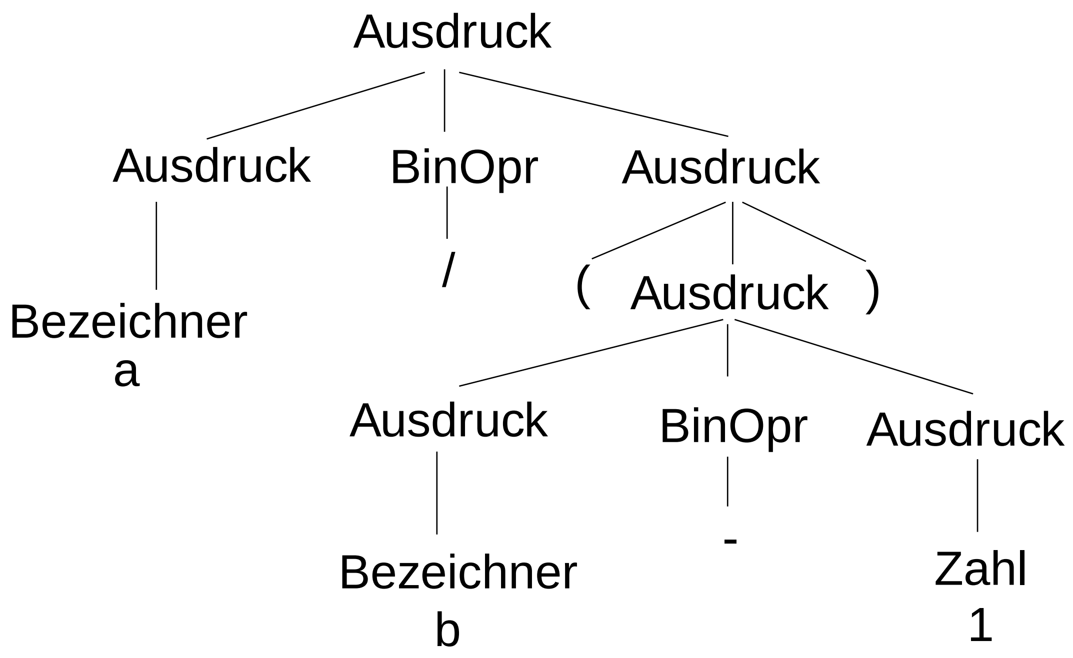
p7: BinOpr ::= ’*’

p8: BinOpr ::= ’/’

Startsymbol: Ausdruck

Terminale:
T = { 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 von KFGn:

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

(gleiche Bedeutung wie bei Wertebereichen)

6.2 Baumstrukturen in XML Übersicht

XML (Extensible Markup Language, dt.: Erweiterbare Auszeichnungssprache)

- seit 1996 vom W3C definiert, in Anlehnung an SGML
- Zweck: Beschreibungen **allgemeiner Strukturen** (nicht nur Web-Dokumente)
- **Meta-Sprache** ("erweiterbar"):
  Die Notation ist festgelegt (Tags und Attribute, wie in HTML),
  Für beliebige Zwecke kann **jeweils eine spezielle syntaktische Struktur** definiert werden (DTD)
  Außerdem gibt es Regeln (XML-Namensräume), um XML-Sprachen in andere **XML-Sprachen zu importieren**
- X**HTML** ist so als XML-Sprache definiert
- Viele **Sprachen sind aus XML abgeleitet,** z.B. SVG, MathML, SMIL, RDF, WML
- **individuelle XML-Sprachen** werden definiert, um strukturierte Daten zu speichern, die von **Software-Werkzeugen geschrieben und gelesen** werden
- XML-Darstellung von strukturierten Daten kann mit verschiedenen Techniken **in HTML transformiert** werden, um sie **formatiert anzuzeigen:**
  XML+CSS, XML+XSL, SAX-Parser, DOM-Parser

Dieser Abschnitt orientiert sich eng an **SELFHTML** (Stefan Münz), http://de.selfhtml.org

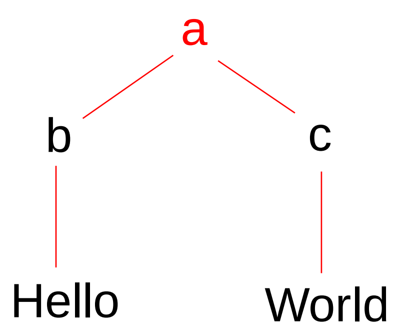
3 elementare Prinzipien

Die XML-Notation basiert auf 3 elementaren Prinzipien:

A: Vollständige Klammerung durch Tags

```
<a>
    <b>Hello</b>
    <c>World</c>
</a>
```

B: Klammerstruktur ist äquivalent zu gewurzeltem Baum



C: Kontextfreie Grammatik definiert Bäume;
eine DTD ist eine KFG

```
a ::= b c
b ::= PCDATA
c ::= PCDATA
```

Notation und erste Beispiele

Ein Satz in einer XML-Sprache ist ein Text, der durch Tags strukturiert wird.

Tags werden **immer** in **Paaren von Anfangs- und End-Tag** verwendet:

```
<ort>Paderborn</ort>
```

Anfangs-Tags können Attribut-Wert-Paare enthalten:

```
<telefon typ="dienst">05251606686</telefon>
```

Die **Namen von Tags und Attributen** können für die XML-Sprache **frei gewählt** werden.

Mit Tags gekennzeichnete Texte können geschachtelt werden.

```
<adressBuch>
<adresse>
   <name>
      <nachname>Mustermann</nachname>
      <vorname>Max</vorname>
   </name>
   <anschrift>
      <strasse>Hauptstr 42</strasse>
      <ort>Paderborn</ort>
      <plz>33098</plz>
   </anschrift>
</adresse>
</adressBuch>
```

$(a+b)^2$ in MathML:

```
<msup>
   <mfenced>
      <mrow>
         <mi>a</mi>
         <mo>+</mo>
         <mi>b</mi>
      </mrow>
   </mfenced>
   <mn>2</mn>
</msup>
```

Ein vollständiges Beispiel

Kennzeichnung des Dokumentes als XML-Datei

Datei mit der Definition der Syntaktischen Struktur dieser XML-Sprache (DTD)

Datei mit Angaben zur Transformation in HTML

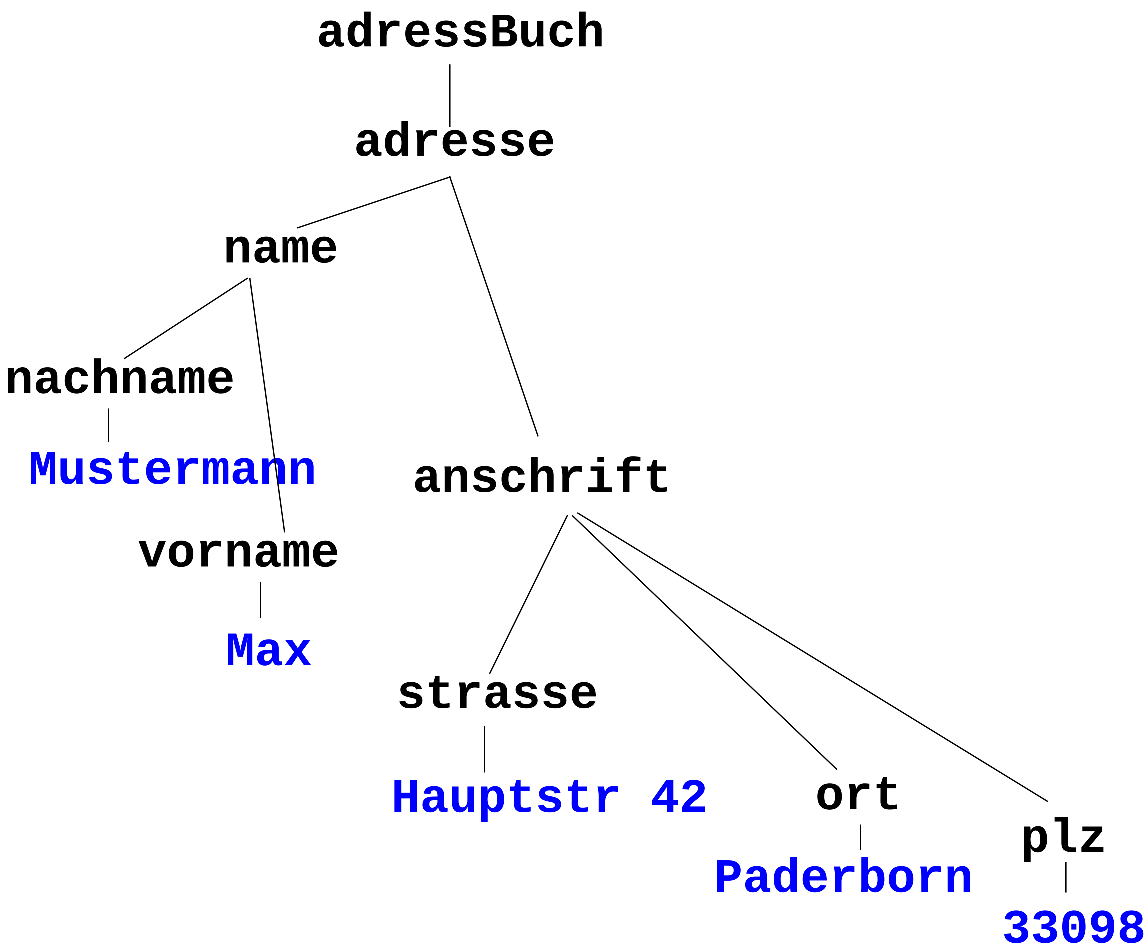
```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE adressBuch SYSTEM "adressBuch.dtd">
<?xml-stylesheet type="text/xsl" href="adressBuch.xsl" ?>
<adressBuch>
<adresse>
   <name>
      <nachname>Mustermann</nachname>
      <vorname>Max</vorname>
   </name>
   <anschrift>
      <strasse>Hauptstr 42</strasse>
      <ort>Paderborn</ort>
      <plz>33098</plz>
   </anschrift>
</adresse>
</adressBuch>
```

Baumdarstellung von XML-Texten

Jeder XML-Text ist durch Tag-Paare **vollständig geklammert** (wenn er *wohlgeformt* ist).

Deshalb kann er eindeutig **als Baum dargestellt** werden. (Attribute betrachten wir hier nicht)
Wir markieren die inneren Knoten mit den Tag-Namen; die Blätter sind die elementaren Texte:

```
<adressBuch>
<adresse>
   <name>
      <nachname>Mustermann
      </nachname>
      <vorname>Max
      </vorname>
   </name>
   <anschrift>
      <strasse>Hauptstr 42
      </strasse>
      <ort>Paderborn</ort>
      <plz>33098</plz>
   </anschrift>
</adresse>
</adressBuch>
```



XML-Werkzeuge können die Baumstruktur eines XML-Textes
ohne weiteres ermitteln und ggf. anzeigen.

Wohlgeformte XML-Texte

XML-Texte sind **wohlgeformt** (well-formed), wenn sie folgende Regeln erfüllen:

1. Ein Element beginnt mit einem Anfangs-Tag und endet mit einem gleichnamigen End-Tag. Dazwischen steht eine evtl. leere Folge von Elementen und elementaren Texten.
2. Elementare Texte können beliebige Zeichen, aber keine Tags enthalten.
3. ein XML-Text ist ein Element.

wohlgeformt

```
<a>
  <b>
    <c>1</c>
    <d>2</d>
  </b>
  <e>3</e>
</a>
```

wohlgeformt

```
<a>
  1
  <b>
    2
    <c>3</c>
    4
    <d>5</d>
  </b>
  <e>6</e>
</a>
```

nicht wohlgeformt

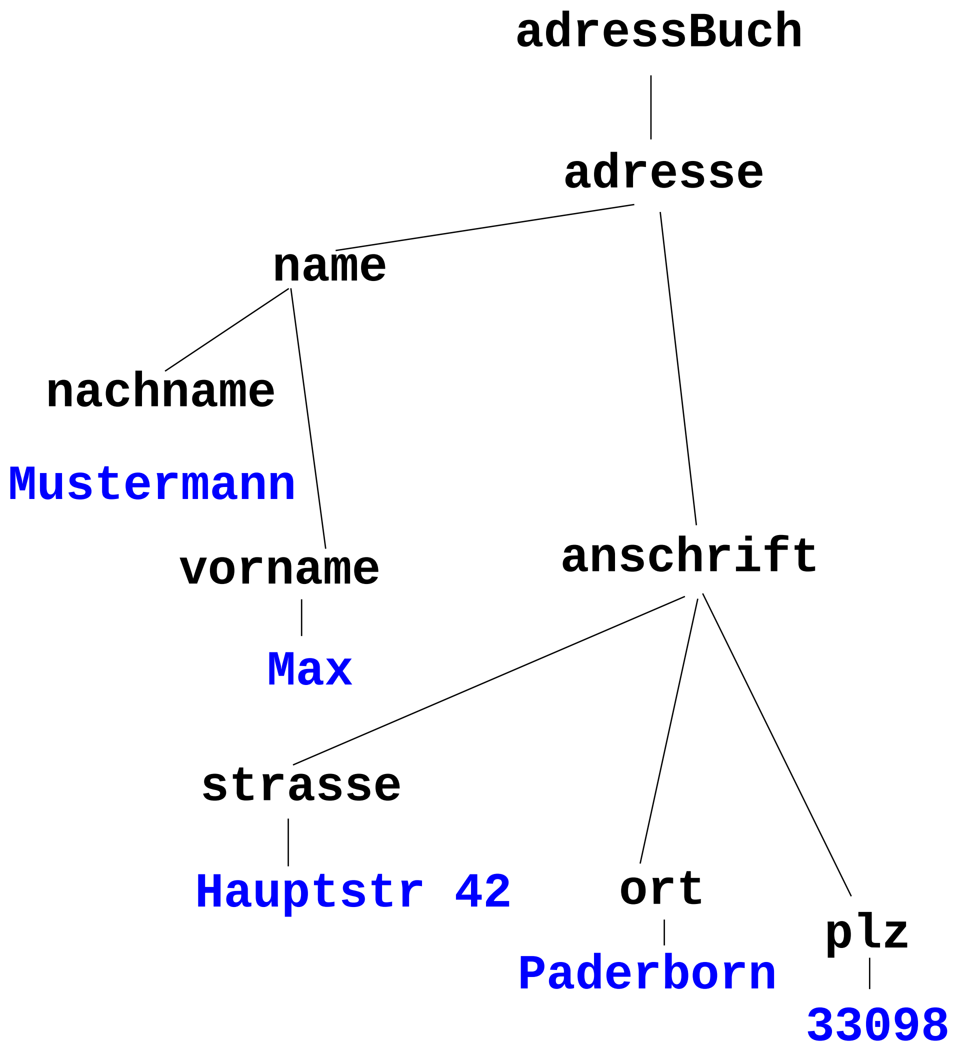
```
<a>
  <b>
    <c>1</b>
  </c>
</a>
```

Grammatik definiert die Struktur der XML-Bäume

Mit **kontextfreien Grammatiken (KFG)** kann man **Bäume definieren**.

Folgende KFG definiert korrekt strukturierte Bäume für das Beispiel Adressbuch:

```
adressBuch ::= adresse*
adresse    ::= name anschrift
name       ::= nachname vorname
Anschrift  ::= strasse ort plz
nachname   ::= PCDATA
vorname    ::= PCDATA
strasse    ::= PCDATA
ort        ::= PCDATA
plz        ::= PCDATA
```



Document Type Definition (DTD) statt KFG

Die Struktur von XML-Bäumen und -Texten wird in der **DTD-Notation** definiert. Ihre Konzepte entsprechen denen von KFGn:

KFG

```
adressBuch ::= adresse*
adresse    ::= name anschrift
name       ::= nachname vorname
Anschrift  ::= strasse ort plz
nachname   ::= PCDATA
vorname    ::= PCDATA
strasse    ::= PCDATA
ort        ::= PCDATA
plz        ::= PCDATA
```

DTD

```
<!ELEMENT adressBuch(adresse)*            >
<!ELEMENT adresse   (name, anschrift)     >
<!ELEMENT name      (nachname, vorname)>
<!ELEMENT anschrift (strasse, ort, plz)>
<!ELEMENT nachname  (#PCDATA)             >
<!ELEMENT vorname   (#PCDATA)             >
<!ELEMENT strasse   (#PCDATA)             >
<!ELEMENT ort       (#PCDATA)             >
<!ELEMENT plz       (#PCDATA)             >
```

weitere Formen von DTD-Produktionen:

`(Y)+`	nicht-leere Folge
`(A | B)`	Alternative
`(A)?`	Option
`EMPTY`	leeres Element

6.3 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

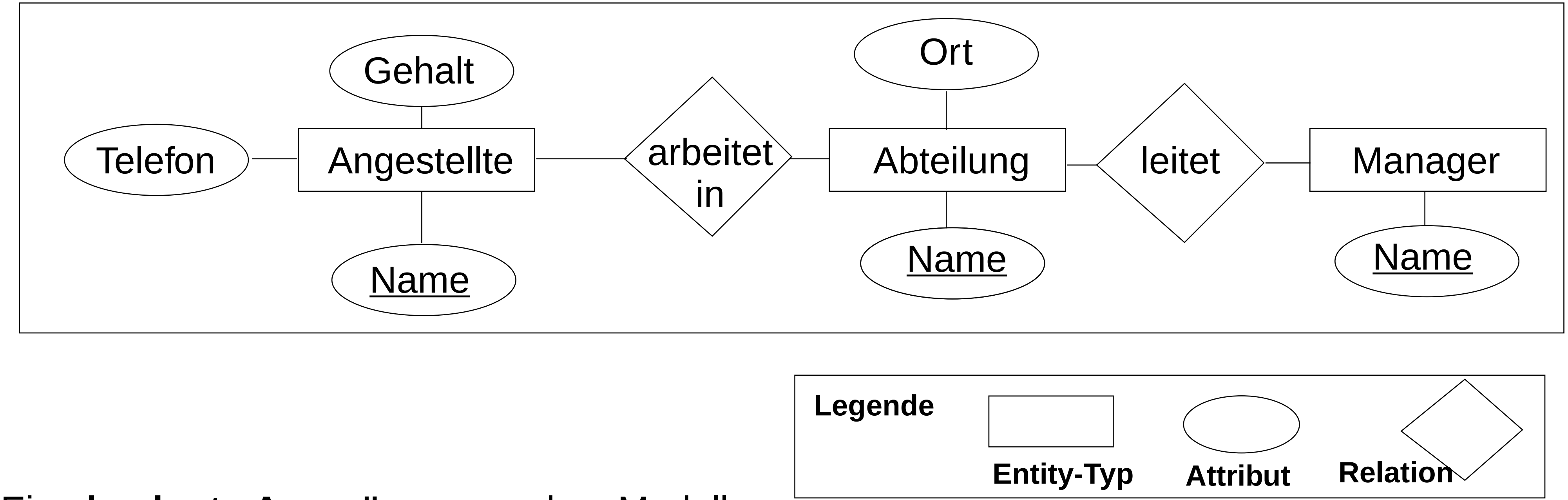
Grundbegriffe

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

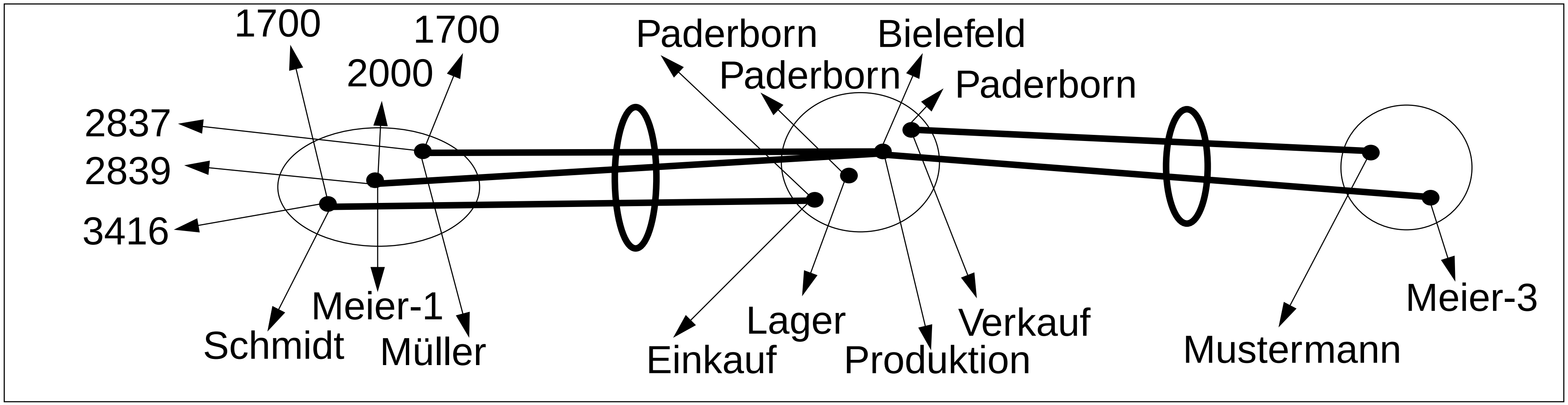
Graphische und textuelle **Notationen** für ER-Modellierungen; hier graphische

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

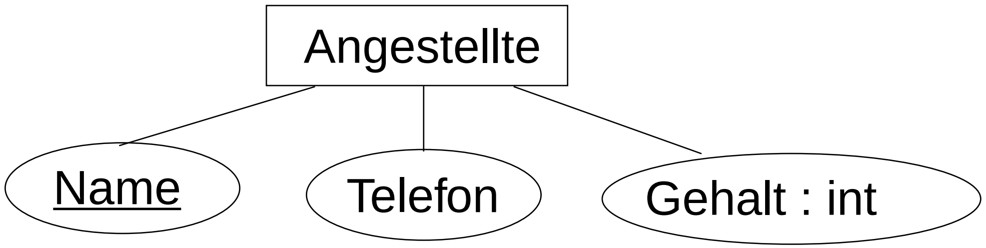
steht im Modell für die **Menge aller** in Unternehmen **möglichen Abteilungen**

- Einkauf
- Verkauf
- Produktion
- Lager

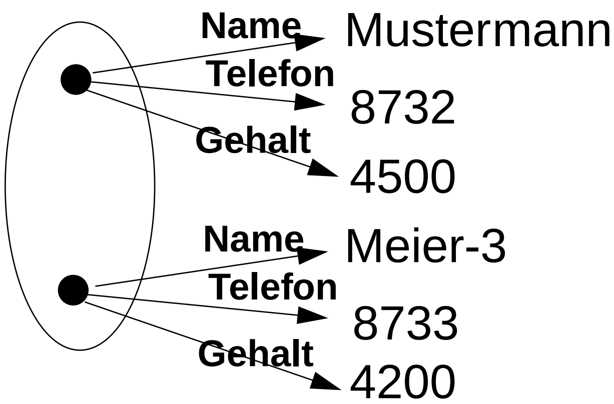
konkrete Ausprägung dazu:
die **Menge der Abteilungen** eines konkreten Unternehmens

Attribute

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



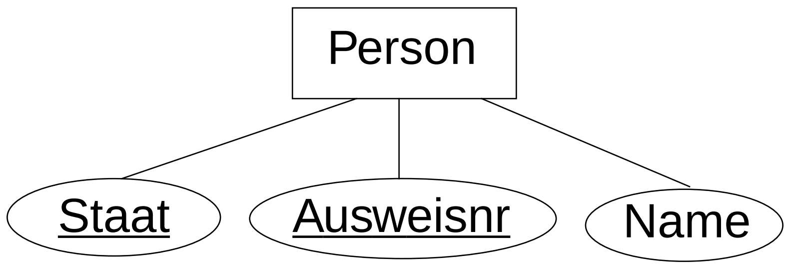
eine konkrete Ausprägung:



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.

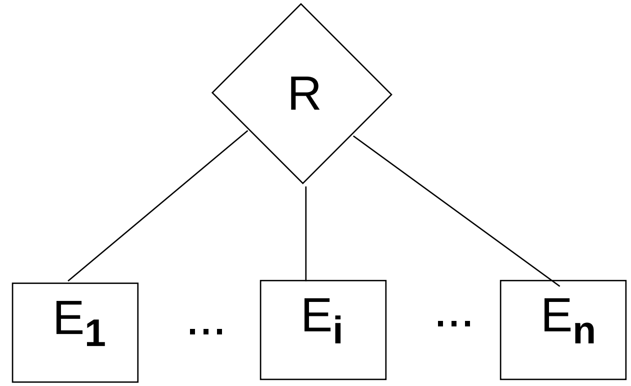
Auch **mehrere Attribute zusammen** können den Schlüssel bilden:



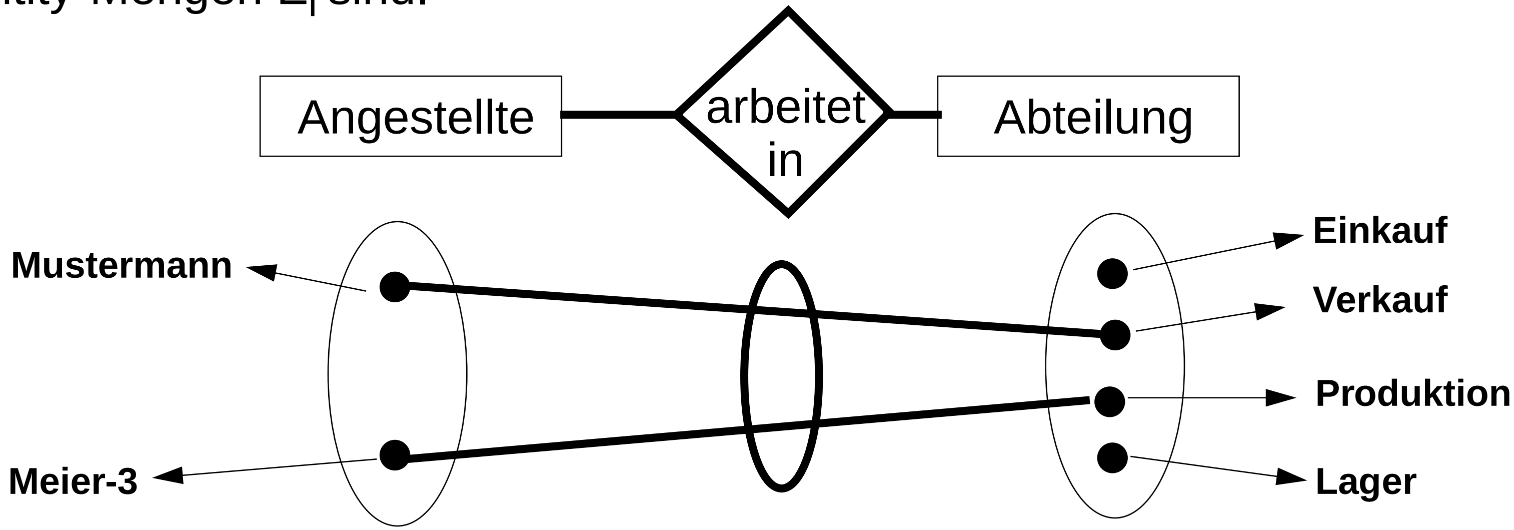
Relationen

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.

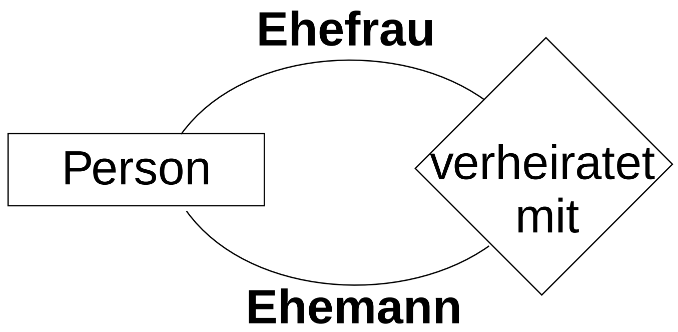


Eine **konkrete Ausprägung von R** ist eine **Menge von n-Tupeln $(e_1, ... , e_n)$**, wobei die e_i Entities aus den konkreten Ausprägungen der Entity-Mengen E_i sind.



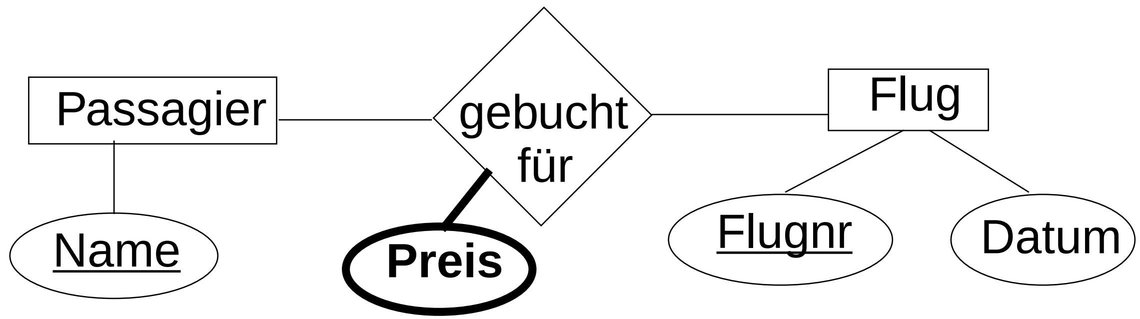
Rollen und Attribute in Relationen

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.

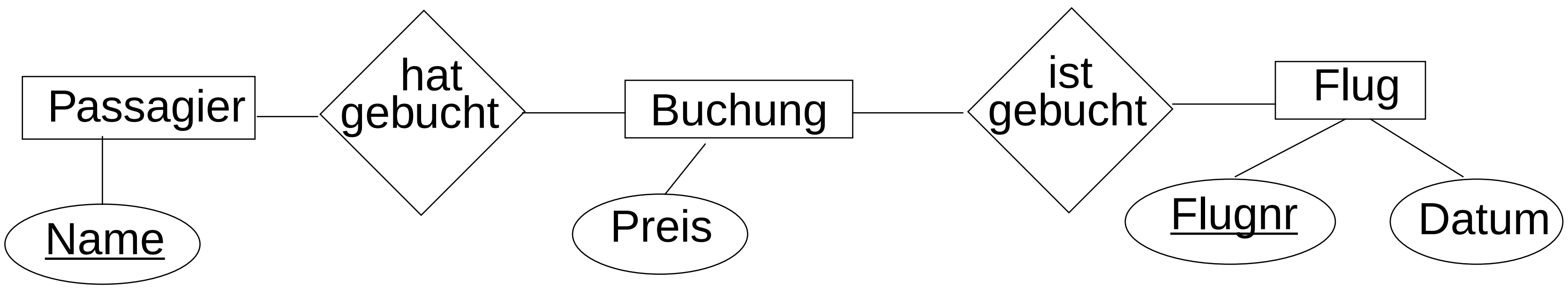


Auch **Relationen können Attribute haben**. Sie beschreiben **Eigenschaften zu jedem Tupel der Relation**.

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

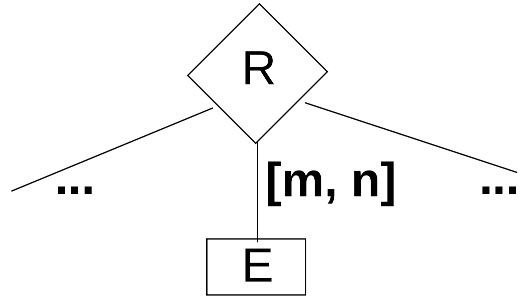


Man könnte natürlich auch **Buchungen als Entities** modellieren:



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:



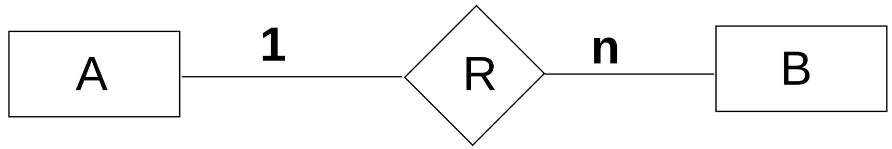
Für jede konkrete Ausprägung der Relation R muss gelten:
Jede Entity e aus der konkreten Entity-Menge zu E kommt
in mindestens m und höchstens n Tupeln vor.

Spezielle Kardinalitäten:

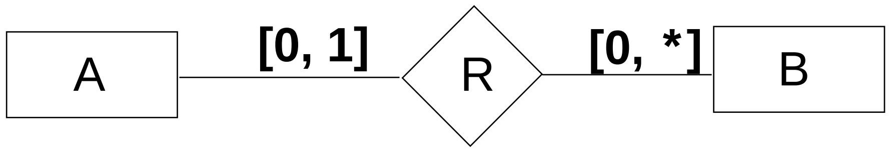
- [1, 1] in **genau einem** Tupel: totale Funktion von E auf die übrigen Rollen der Relation
- [0, 1] in **höchstens einem** Tupel: partielle Funktion von E auf die übrigen Rollen
- [0, *****] in **beliebig vielen** Tupeln

Ohne Angabe wird [0, *] angenommen.

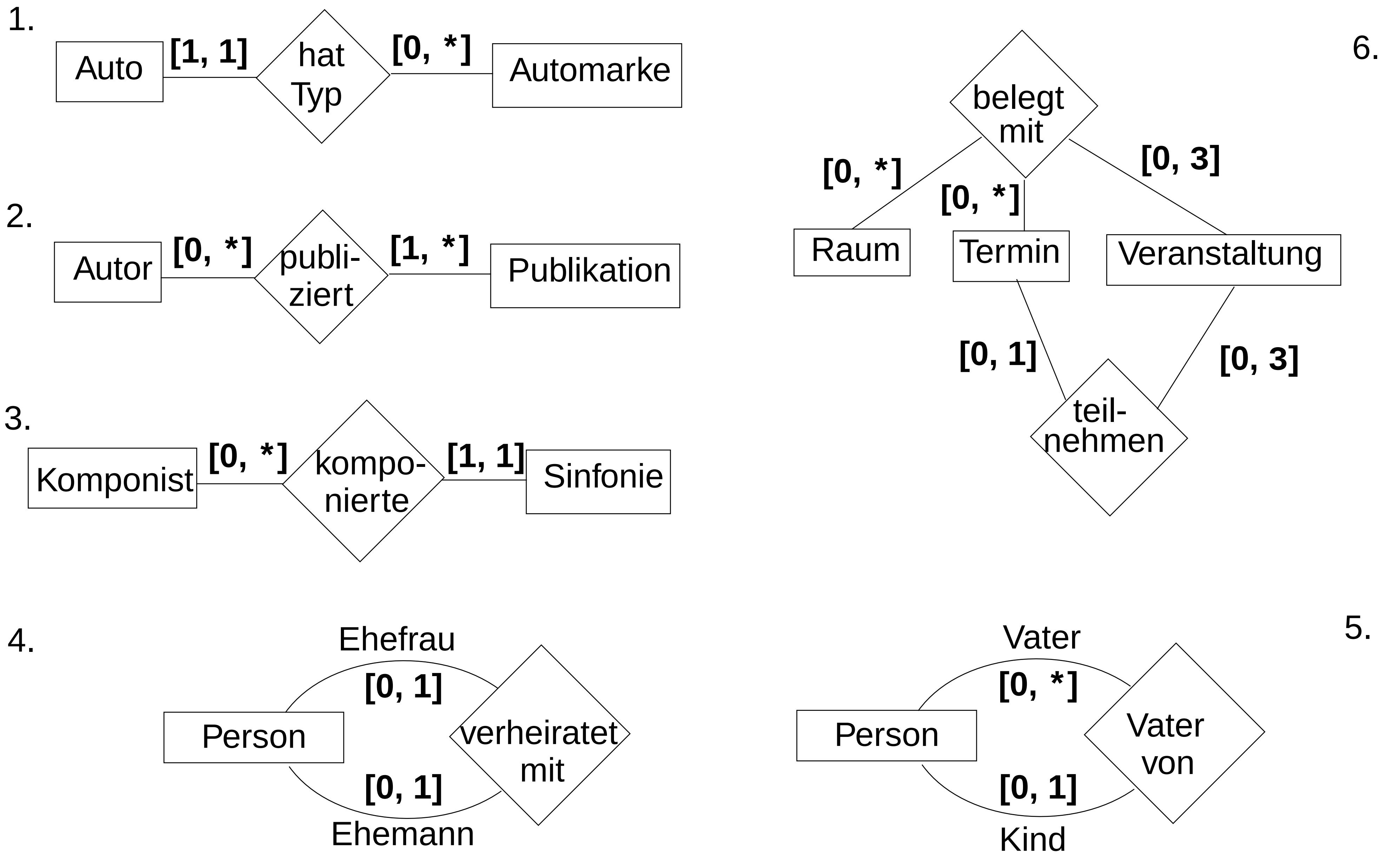
Kurznotation für 2-stellige Relationen:



bedeutet:

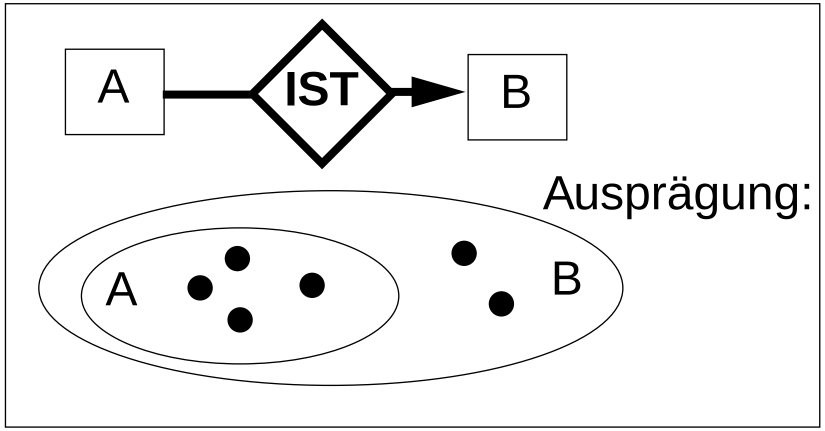


Beispiele zu Kardinalitäten in Relationen



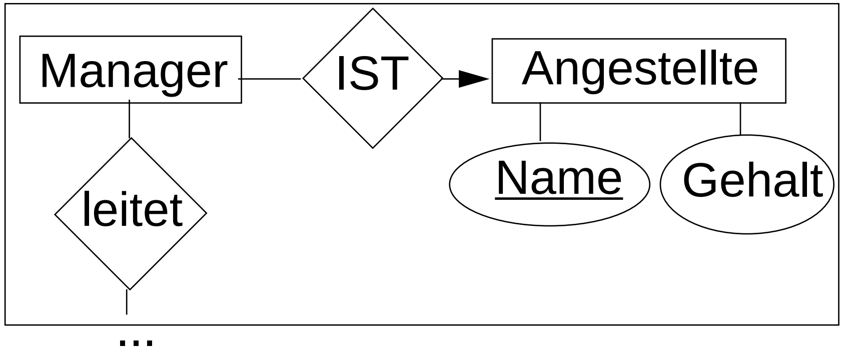
IST-Hierarchie

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



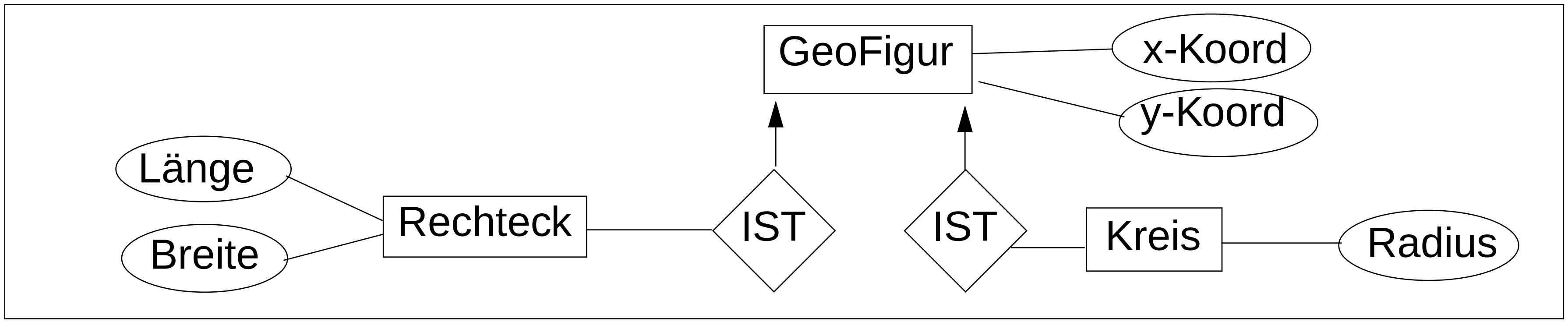
A IST B: Einige Entities der **allgemeineren Menge B** gehören auch der **spezielleren Menge A** an.

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

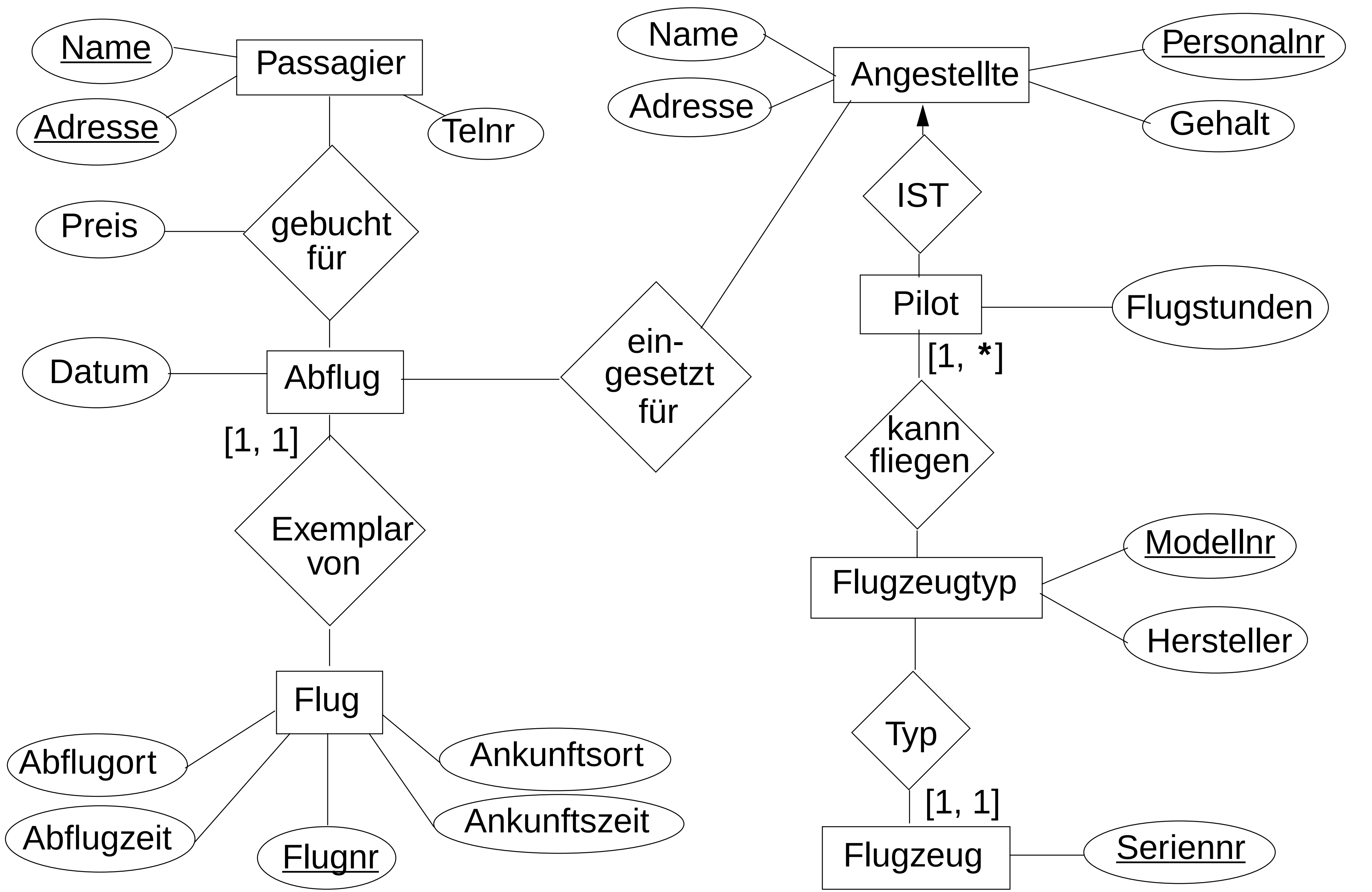


Die **Entities in A „erben" alle Attribute von B** und können noch weitere Attribute haben, die **spezielle A-Eigenschaften** beschreiben.

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

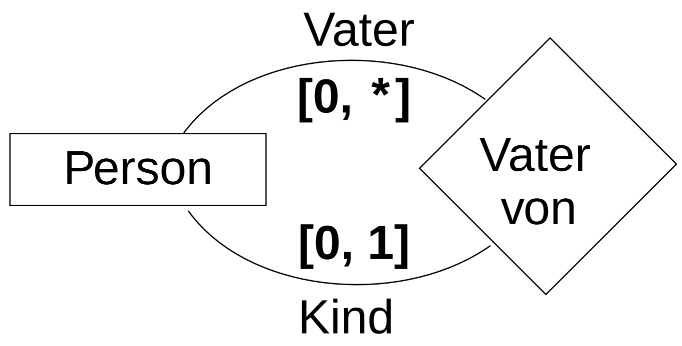


Beispiel: Fluggesellschaft

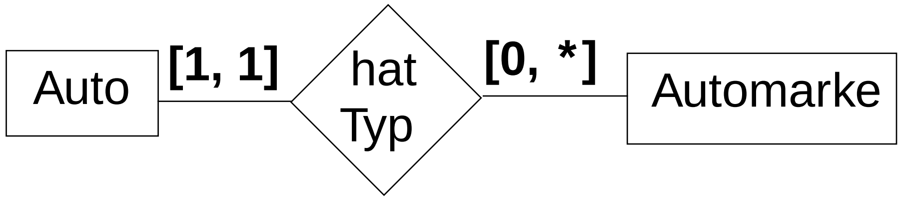


Hinweise zur Modellierung mit ER

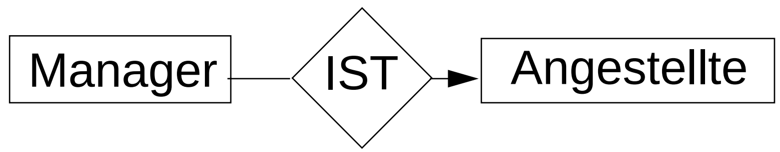
- In einem ER-Modell kommt **jede Entity-Menge nur einmal** vor.
- **Rollen** zu Relationen **angeben**, wo es nötig ist.
- Bedeutung der Kardinalitäten klarstellen.



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



- **Spezialisierung** sinnvoll einsetzen.



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

6.4 Klassendiagramme in UML
Übersicht

1. **UML (Unified Modelling Language)**:
   die derzeit wichtigste Sprache zur **Modellierung von Systemen**
2. Als **Zusammenfassung mehrerer Modellierungssprachen**
   1997 in der Version 1.1 definiert;
   Version 2.0 von 2005 ist Grundlage aktueller UML-Versionen.
3. ***Object Management Group*** macht aktuelle Dokumente zu UML verfügbar:
   Object Management Group: UML Resource Page. www.uml.org (2010)
4. UML umfasst **13 Teilsprachen (*Diagrammtypen*)**, um unterschiedliche Aspekte von Systemen zu beschreiben, z. B.
   Klassendiagramme für Systemstruktur, statische Eigenschaften und Beziehungen,
   Statecharts für Abläufe von Operationen.
5. Für den Gebrauch durch Menschen hat UML graphische Notationen (visuelle Sprachen);
   Software-Werkzeuge verwendendie XML Sprache **XMI (XML Metadata Interchange)**
6. **Einführendes Buch**:
   Chris Rupp, Stefan Queins, Barbara Zengler:
   UML 2 glasklar. 3. Auflage; Carl Hanser Verlag (2007)

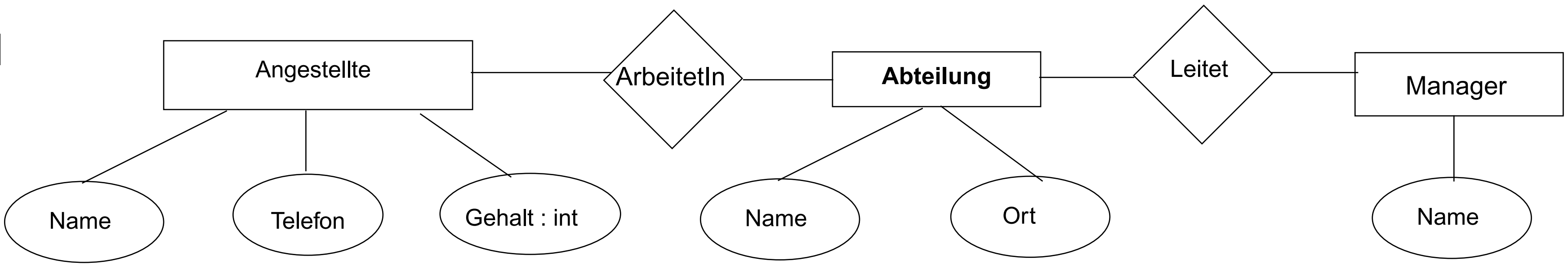
Bezug zum ER-Modell

Klassendiagramme dienen zur Modellierung von
Systemstruktur, **statischen Eigenschaften** und **Beziehungen**.

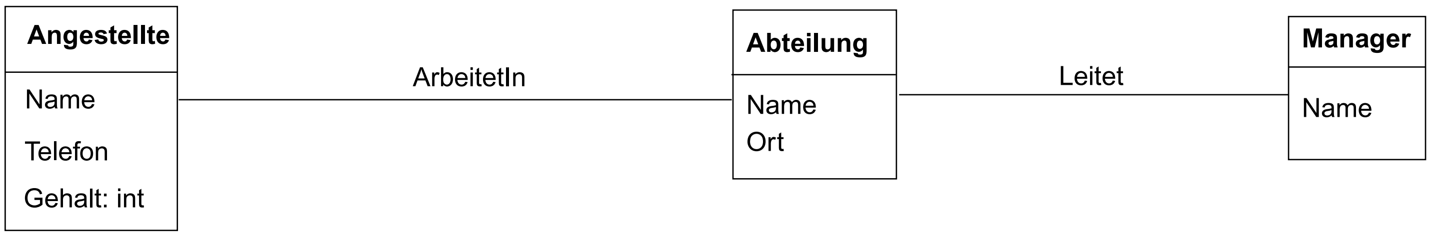
Sie basieren auf den gleichen Grundkonzepten wir das Entity-Relationship-Modell:

ER-Modell	**UML Klassendiagramm**
Entity-Menge	Klasse
Attribut	Attribut
Relation	Assoziation

ER-Modell



UML Klassendiagramm



Klasse mit Attributen

Klasse: repräsentiert eine Menge gleichartiger Objekte (wie im ER-Modell);

Attribute (und ggf. Operationen) werden im Rechteck der Klasse angegeben.

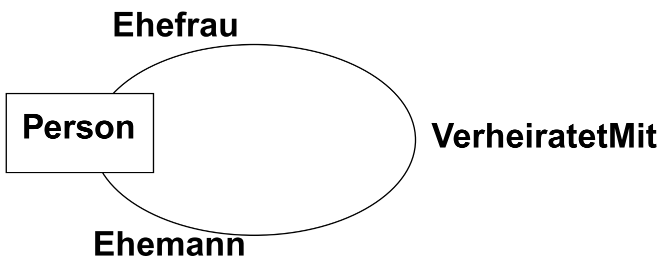
Angestellte
Name Telefon Gehalt: int

Objekte einer Klasse werden so dargestellt:

<u>**H-Schmidt:Angestellte**</u>
Name: String = “Schmidt“ Telefon: Integer = 3416 Gehalt: Integer = 1700

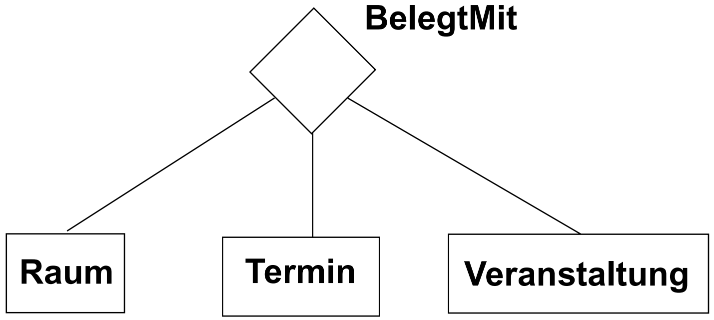
Assoziationen

zweistellig
◀ gibt die Leserichtung an

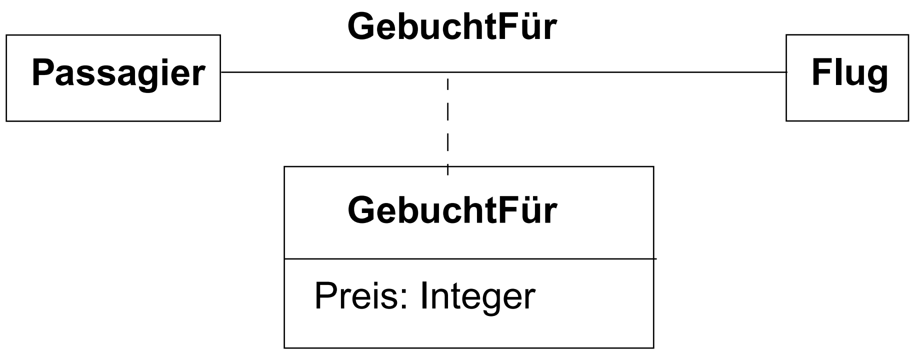


zweistellig

mit Angabe der Rollen



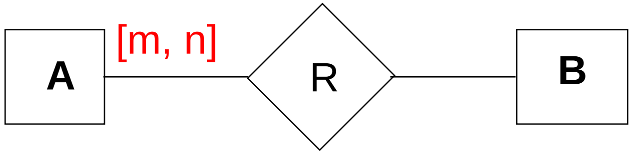
mehrstellig

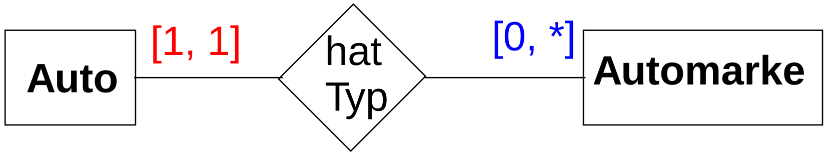


Assoziation mit Attributen

Kardinalität von 2-stelligen Assoziationen

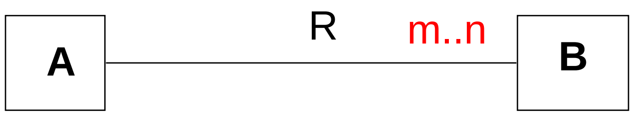
ER:

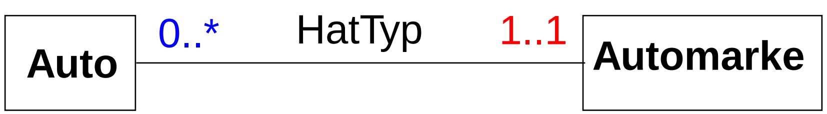




Jedes Objekt aus A kommt in den Tupeln der Relation R mindestens m und höchstens n mal vor.

UML:





Jedem Objekt aus A ordnet die Relation R mindestens m und höchstens n verschiedene Objekte aus B zu.

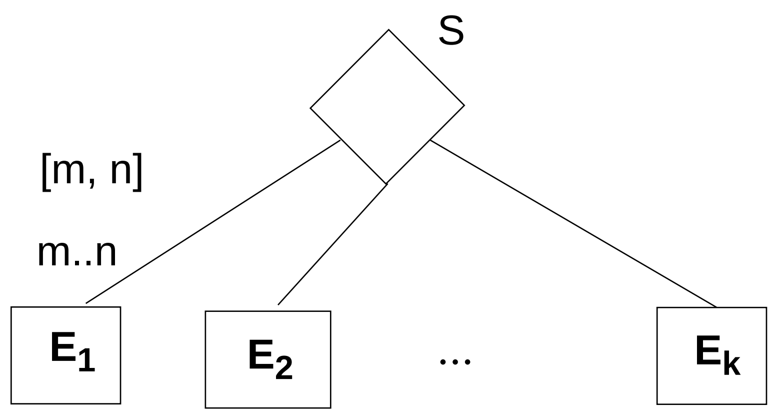
Kardinalität von k-stelligen Assoziationen

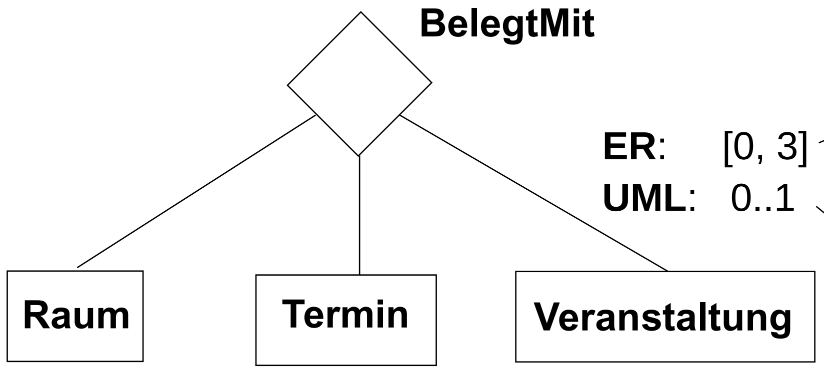
Jedes Objekt aus E1 kommt in den Tupeln der Relation S mindestens m und höchstens n mal vor.

ER: [m, n]

UML: m..n

Jeder Kombination von Objekten aus E2, ..., En ordnet die Relation S mindestens m und höchstens n Objekte aus E1 zu.





ER: [0, 3]

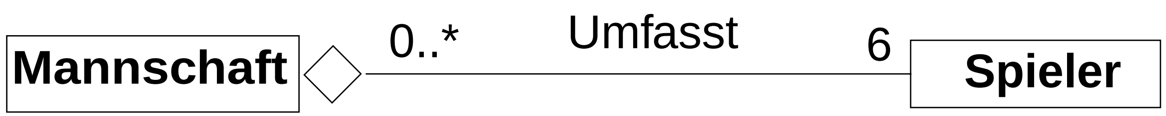
UML: 0..1

Für jede Veranstaltung sind zwischen 0 und 3 Raum-Termin-Kombinationen vorgesehen. (nicht in UML formulierbar)

Für jede Raum-Termin-Kombination ist höchstens eine Veranstaltung vorgesehen. (nicht in ER formulierbar)

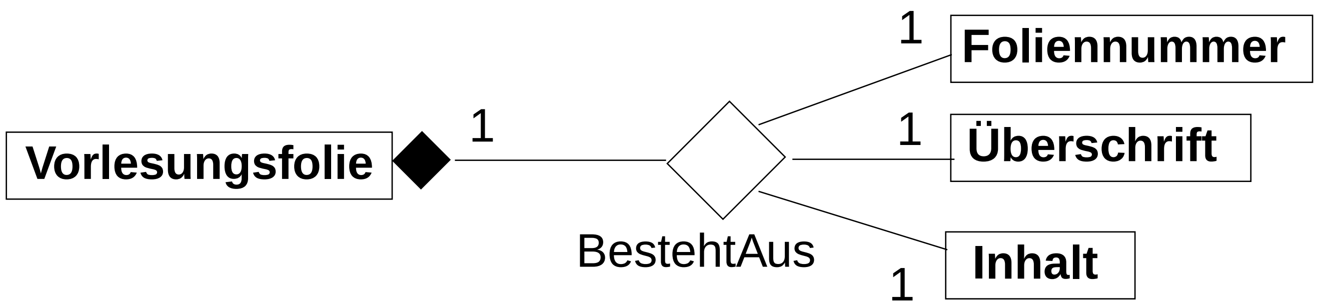
Aggregation und Komposition

Aggregation: Objekte werden zu einem größeren Objekt zusammengefasst.
sie können prinzipiell auch allein existieren.



- Eine Mannschaft umfasst immer 6 Spieler
- Ein Spieler kann einer, mehreren oder auch keiner Mannschaft angehören

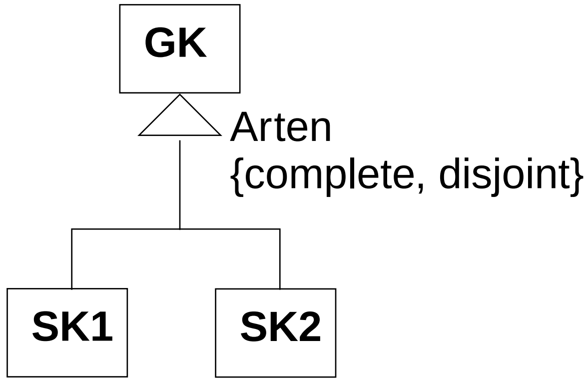
Komposition: Jedes Teilobjekt gehört unverzichtbar zu genau einem ganzen Objekt.



Eine Vorlesungsfolie besteht immer aus einer Foliennummer, einer Überschrift und dem Folieninhalt.

Generalisierung, Spezialisierung

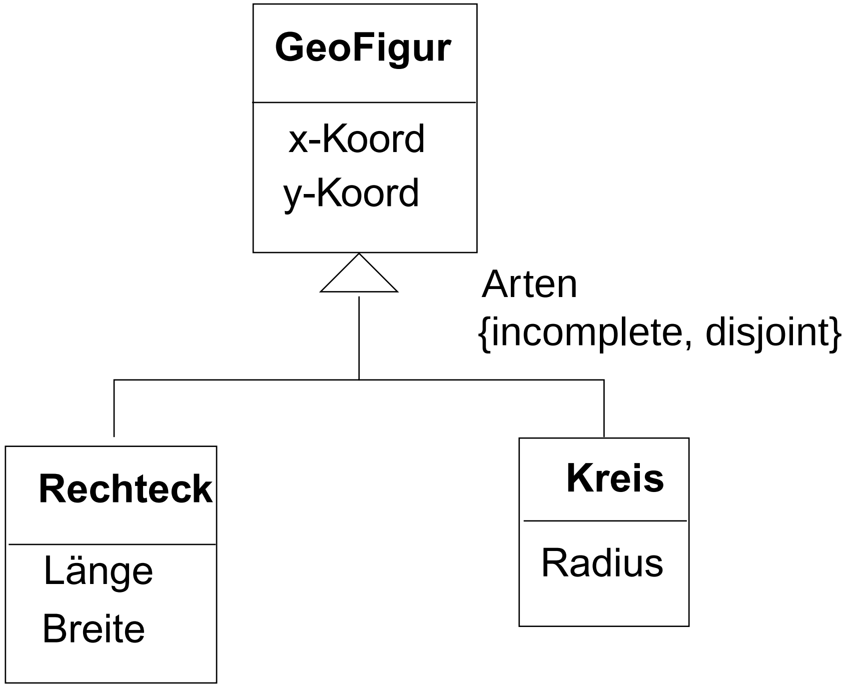
Die Generalisierung (Spezialisierung) dient zur Modellierung von **Abstraktionshierarchien** (wie die **IST**-Relation in ER):



SK1 und SK2 sind **speziellere** Arten der **allgemeineren** GK.

GK heißt auch **Oberklasse** der **Unterklassen** SK1 und SK2.

Die Assoziation kann **benannt** werden, hier *Arten*.



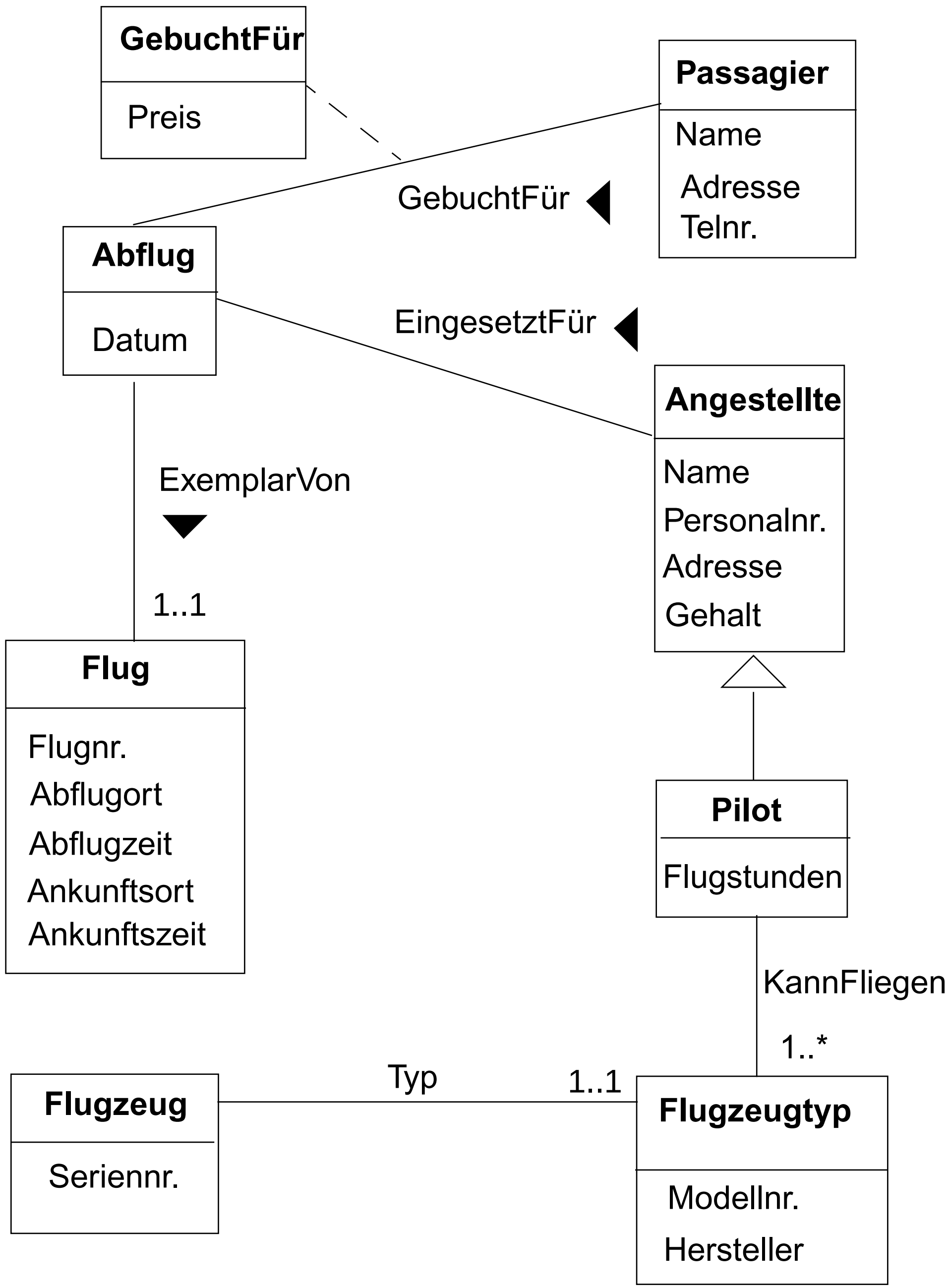
Hinsichtlich der Objekte gilt: SK1 und SK2 sind **Teilmengen** von GK.

Das Verhältnis der Unterklassen zueinanderkann weiter charakterisiert werden:

- ***disjoint***: Die Teilmengen sind paarweise disjunkt.
- ***complete***: Es gibt in dem Modell **keine weiteren Unterklassen** von GK

Modell einer Fluggesellschaft

vergl. Folie 6.17



7 Modellierung von Abläufen
7.1 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

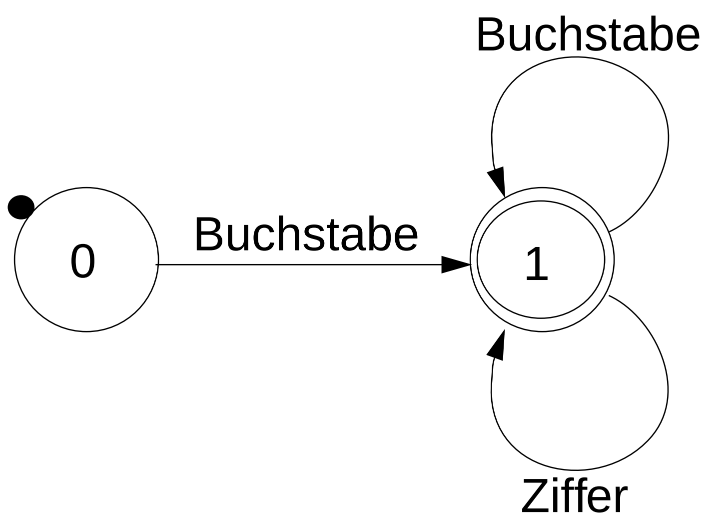
- das **Verhalten realer Maschinen** zu spezifizieren, z. B. Getränkeautomat,
- das **Verhalten von Software-Komponenten** zu spezifizieren, z. B. Reaktionen von Benutzungsoberflächen auf Bedienereignisse,
- **Sprachen zu spezifizieren**: Menge der Ereignis- oder Symbolfolgen, die der Automat akzeptiert,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

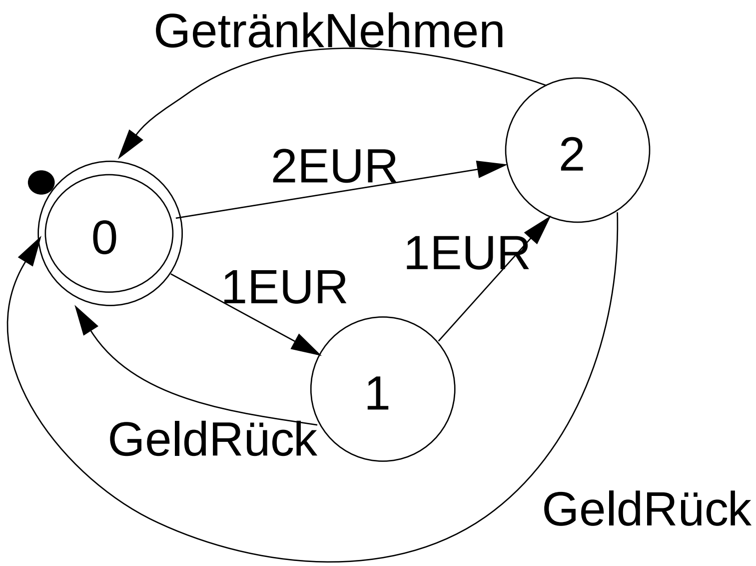
Hier: **Bezeichner** in Pascal-Programmen:



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 $a_1\ a_2\ ...\ a_n$,
z. B. $10010 \in \{0, 1\}^*$

für die leere Folge schreiben wir auch 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^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: $1^3\, (1 \mid 0)^*\, 0^3$

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 = (Σ, Q, δ, 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-7.2:

Σ := Menge der ASCII-Zeichen

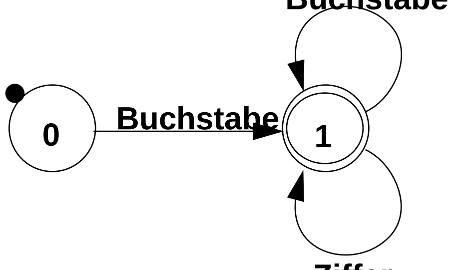
Q := {0, 1}

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**}

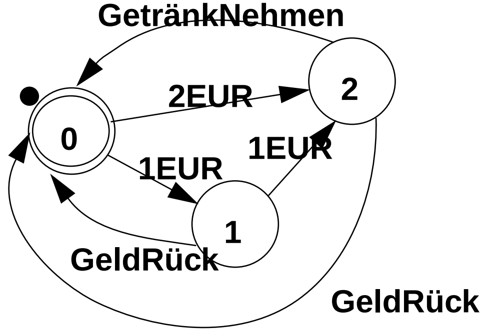
Q := {0, 1, 2}

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.

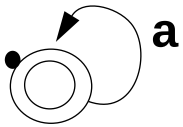
Sei $A = (\Sigma, Q, \delta, q_0, F)$ ein deterministischer endlicher Automat und $w \in \Sigma^*$.
A akzeptiert das Wort w genau dann, wenn $\delta(q_0, w) \in F$.
Die Menge $L(A) := \{ w \in \Sigma^* \mid \delta(q_0, w) \in F \}$ heißt die **von A akzeptierte Sprache**.

Beispiele für Sprachen, die von endlichen Automaten akzeptiert werden können:

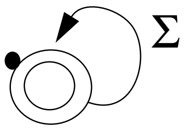
$L_1 = a^+ b^+ = \bigcup_{n, m \in \mathbb{N}} a^n b^m$ $\quad L_2 = \Sigma^*$

Es gibt keinen endlichen Automaten, der $L_3 = \bigcup_{n \in \mathbb{N}} a^n b^n$ akzeptiert.

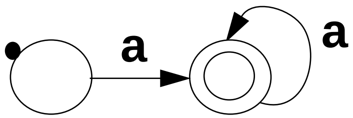
Beispiele: Endliche Automaten und ihre Sprachen

1. 

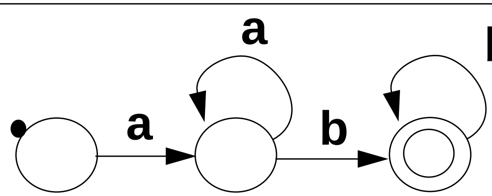
a^*

2. 

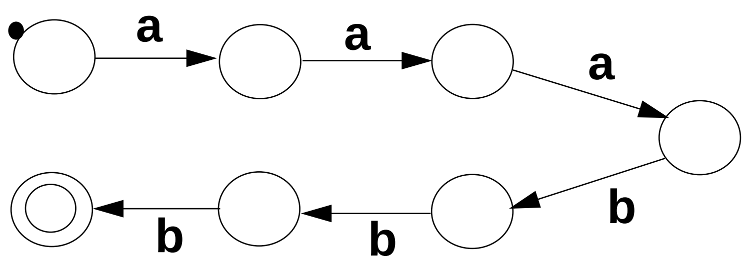
Σ^*

3. 

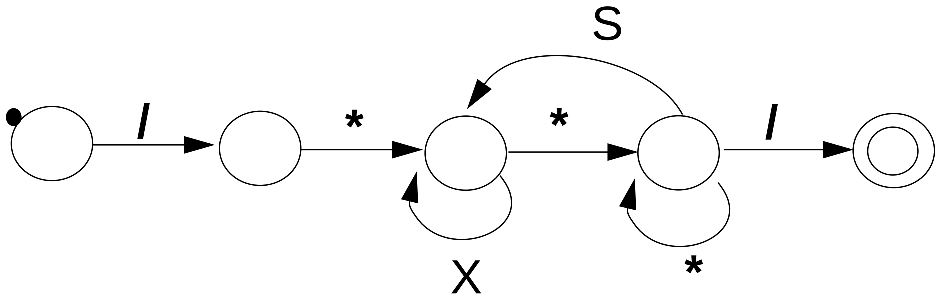
a^+

4. 

a^+b^+

5. 

a^3b^3

6. 

$X = \Sigma \setminus \{ * \}, \quad S = \Sigma \setminus \{ /, * \}$

Sprache: /* Σ* */

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 $\delta : Q \times \Sigma \to Pow(Q)$.
Welcher gewählt wird, ist nicht festgelegt.
Σ, Q, q_0, F sind wie für deterministische endliche Automaten definiert.

Erweiterung von δ auf Zeichenfolgen:

Sei $A = (\Sigma, Q, \delta, q_0, F)$ 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**.

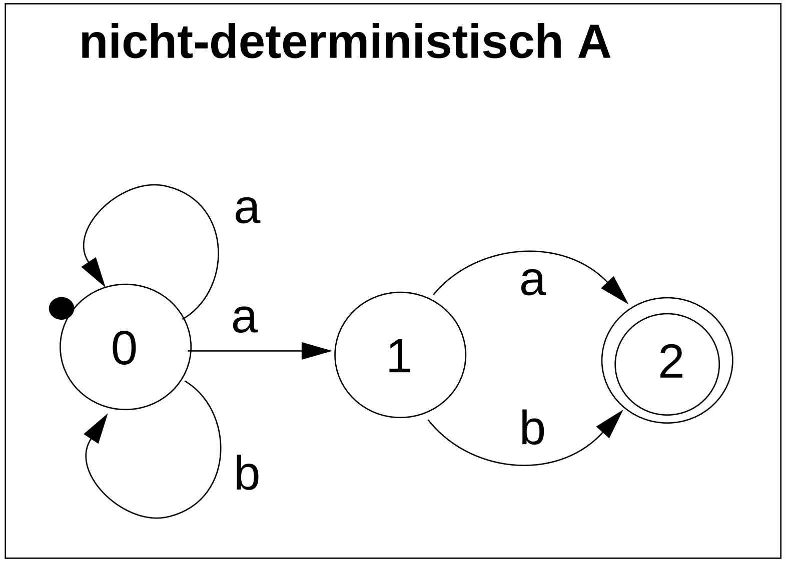
Nicht-deterministische und deterministische Automaten

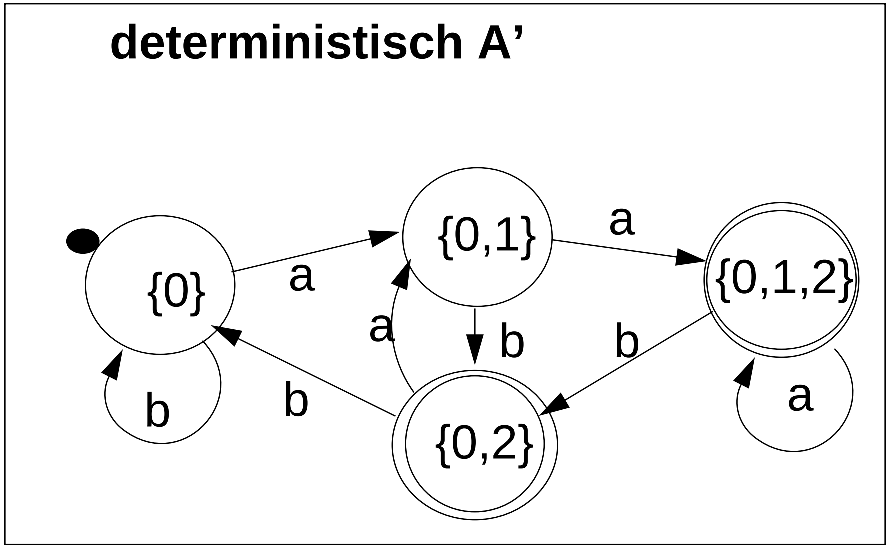
Satz: Sei L(A) die Sprache eines nicht-deterministischen Automaten.
Dann gibt es einen deterministischen Automaten, der L(A) akzeptiert.

Man kann **aus einem nicht-deterministischen Automaten A** = (Σ , Q , δ, q_0 , F)
einen **deterministischen A'** = (Σ, Q', δ', q_0', F') systematisch **konstruieren**:

Jeder **Zustand aus Q' repräsentiert eine Menge von Zuständen aus Q, d. h. Q'** $\subseteq$ **Pow(Q)**

Beispiel:





Die Zahl der Zustände kann sich dabei **exponentiell** vergrößern.

Konstruktion deterministischer Automaten

Sei A ein **nicht-deterministischer Automate A** = (Σ , Q , δ, q_0 , F) daraus wird ein **deterministischer Automat A'** = (Σ, Q', δ', q_0', F') systematisch **konstruiert**:

Jeder **Zustand aus Q' repräsentiert eine Menge von Zuständen aus Q, d. h. Q' $\subseteq$ Pow(Q)**

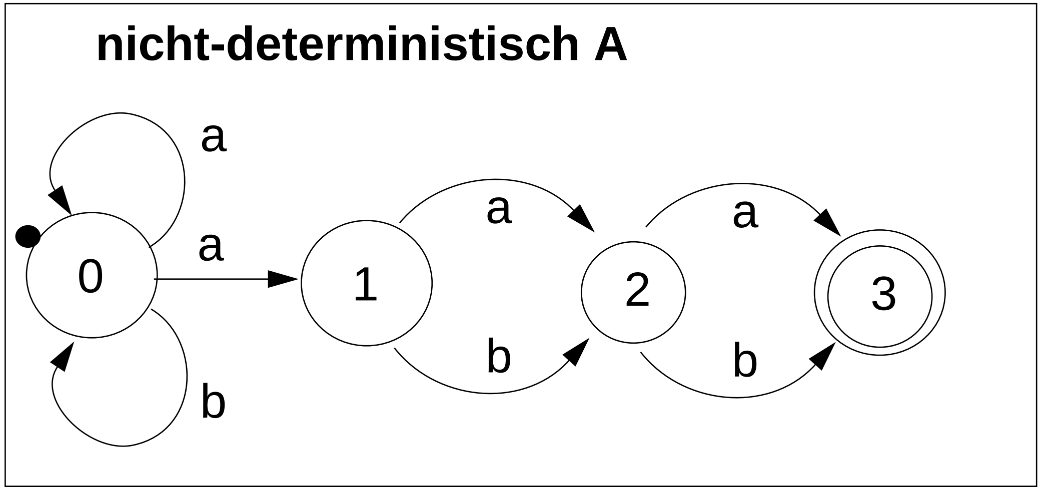
Konstruktionsschritte:

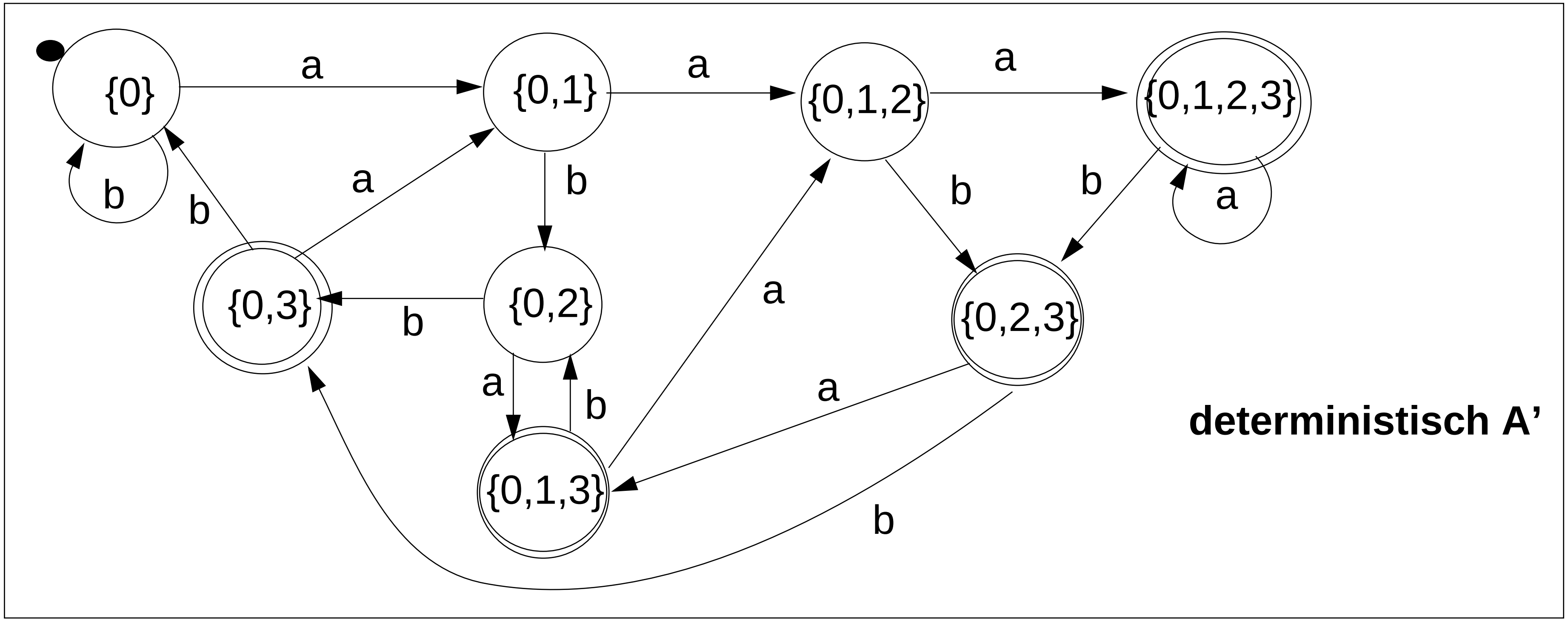
1. **Anfangszustand**: $q_0' = \{q_0\}$
2. Wähle einen schon konstruierten Zustand $q' \in Q'$
   wähle ein Zeichen $a \in \Sigma$
   berechne $r' = \delta'(q', a) = \bigcup_{\mathbf{q} \in \mathbf{q'}} \delta(q, a)$
   d. h. r' repräsentiert die Vereinigung aller Zustände, die in A von q unter a erreicht 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.

Beispiel zur Konstruktion NDEA -> DEA

Sprache: (a | b)* a (a | b)^2

Worte w über {a, b} mit |w| > 2 und drittletztes Zeichen ist ein a





Endliche Automaten mit Ausgabe

Man kann mit endlichen Automaten auch **Reaktionen der modellierten Maschine** spezifizieren: **Automaten mit Ausgabe**.

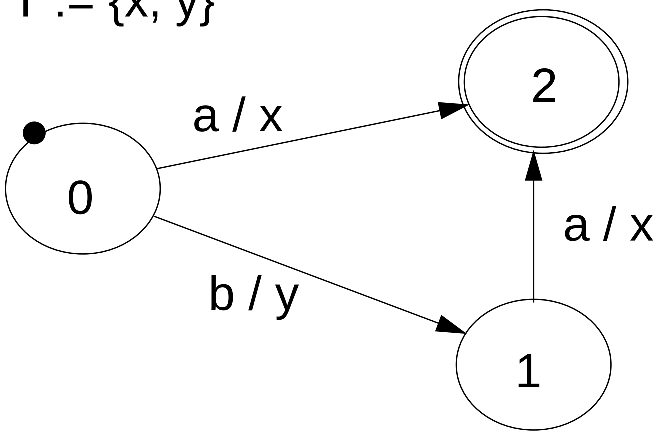
Wir erweitern den Automaten um ein **endliches Ausgabealphabet T** und um eine Ausgabefunktion. Es gibt 2 Varianten für die Ausgabefunktion:

Mealy-Automat:

Eine Ausgabefunktion $\lambda : Q \times \Sigma \to T^*$ ordnet den **Zustandsübergängen** jeweils ein **Wort über dem Ausgabealphabet** zu.

Graphische Notation:

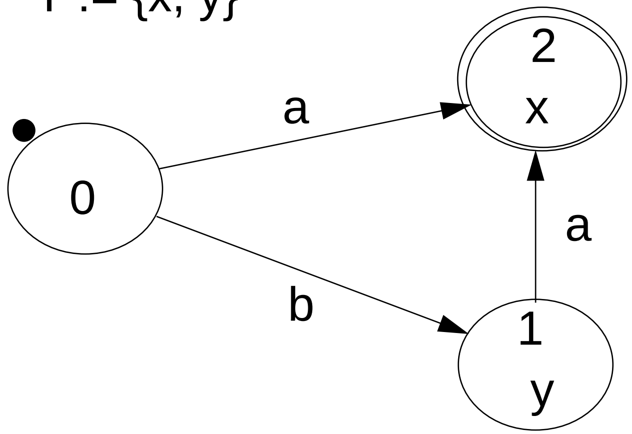
T := {x, y}



Moore-Automat:

Eine Ausgabefunktion $\mu : Q \to T^*$ 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.

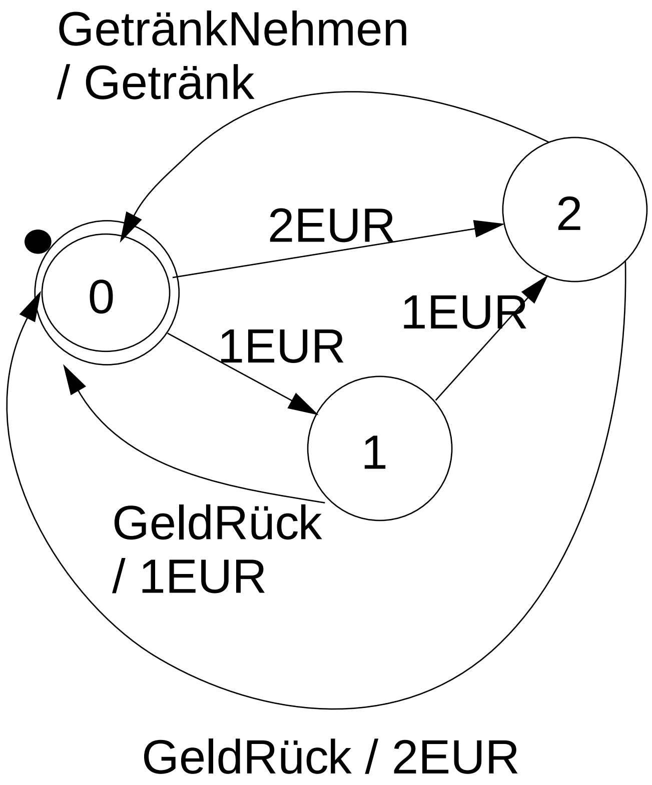
Beispiele für endliche Automaten mit Ausgabe

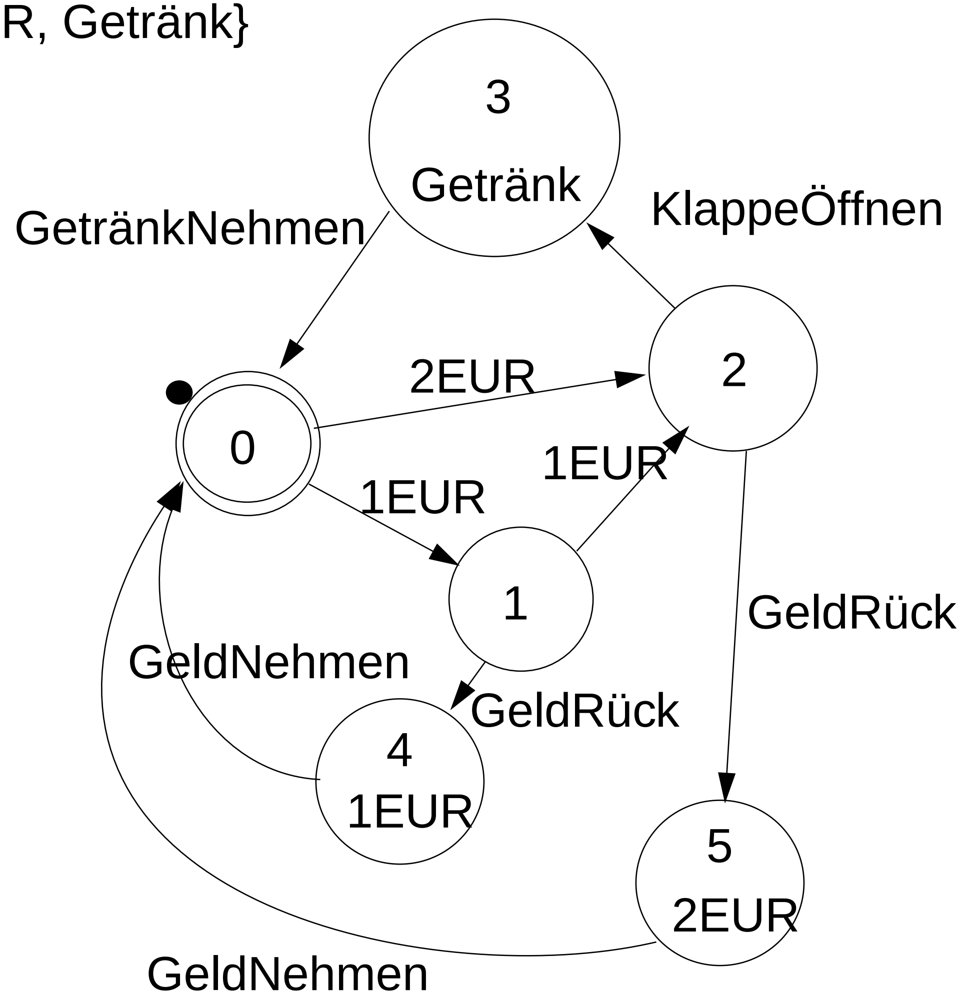
Die Spezifikation des Getränkeautomaten aus Mod-7.2 wird mit Ausgabe versehen:

Mealy-Automat

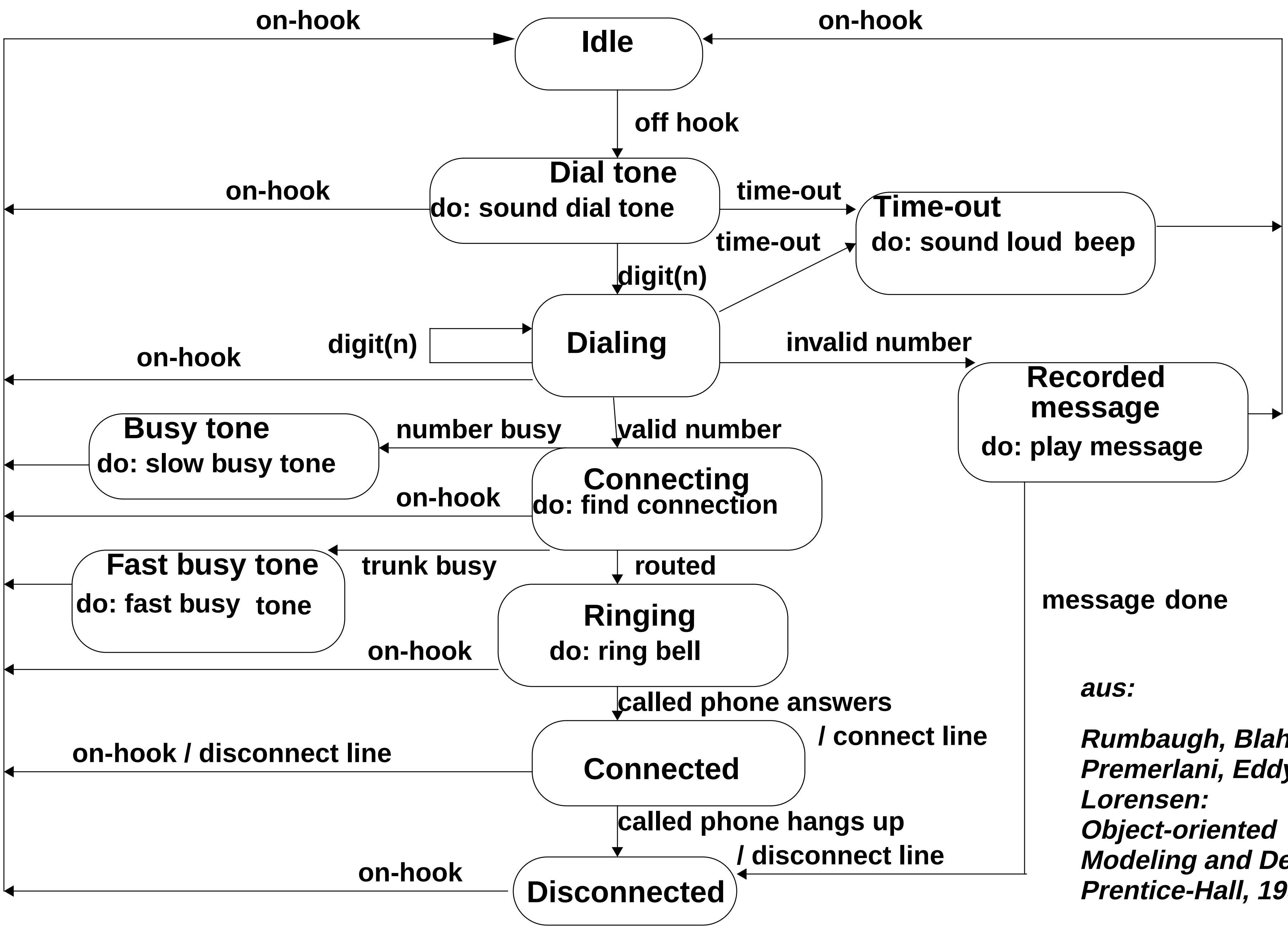
Moore-Automat

T = {1EUR, 2EUR, Getränk}





Endlicher Automat zur Telefonbedienung



aus:

Rumbaugh, Blaha, Premerlani, Eddy, Lorensen: Object-oriented Modeling and Design, Prentice-Hall, 1991

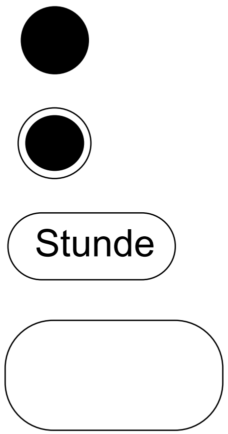
Endliche Automaten in UML: Modell einer Uhr

UML Diagrammtyp Statecharts: Modellierung von Abläufen

Konzeptuelle Grundlage:
Endliche Automaten

Zustände können **hierarchisch zu Teilautomaten verfeinert** werden.

Mehrere Teilautomaten können „quasi-gleichzeitig“ Übergänge ausführen - zur **Modellierung von Nebenläufigkeit**.



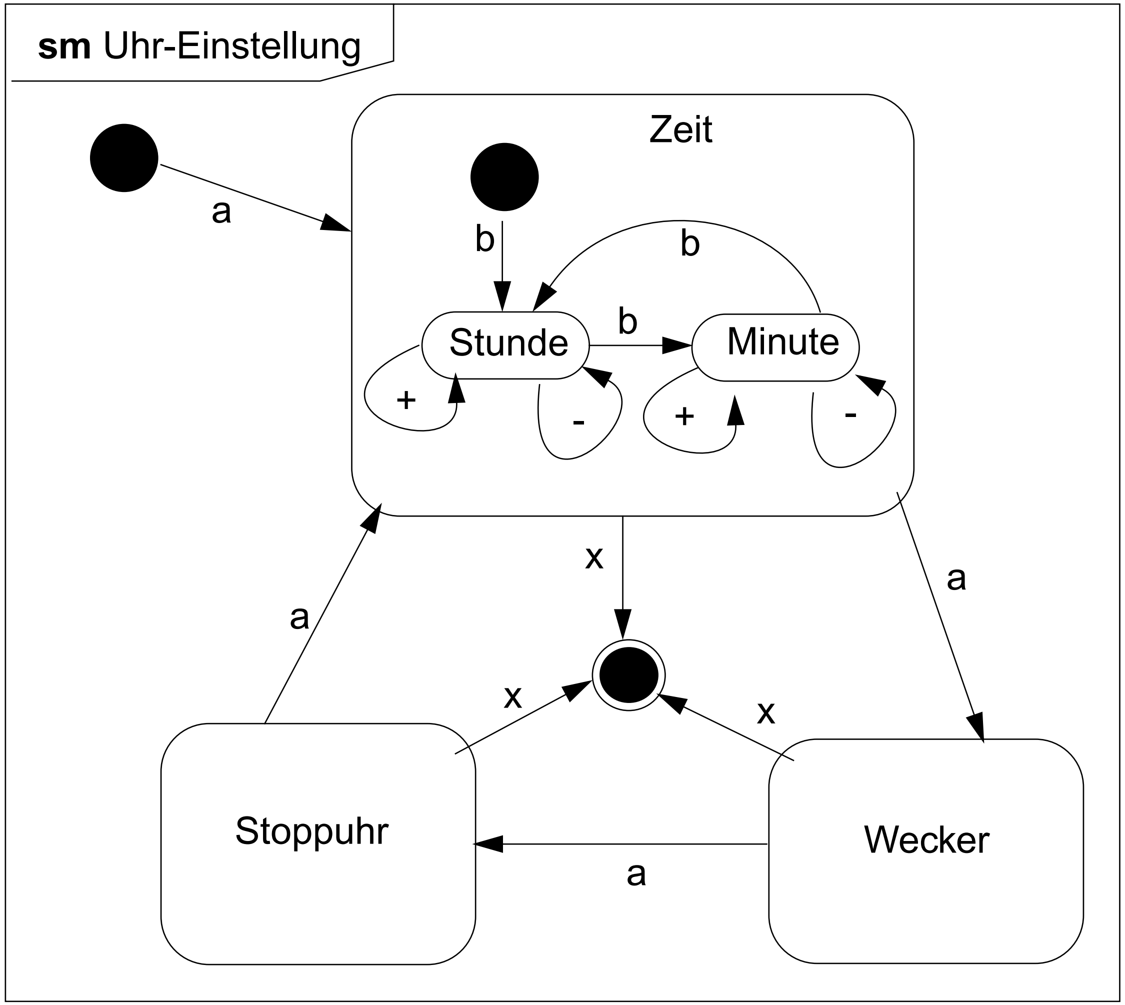
Anfangszustand

Endzustand

elementarer Zustand

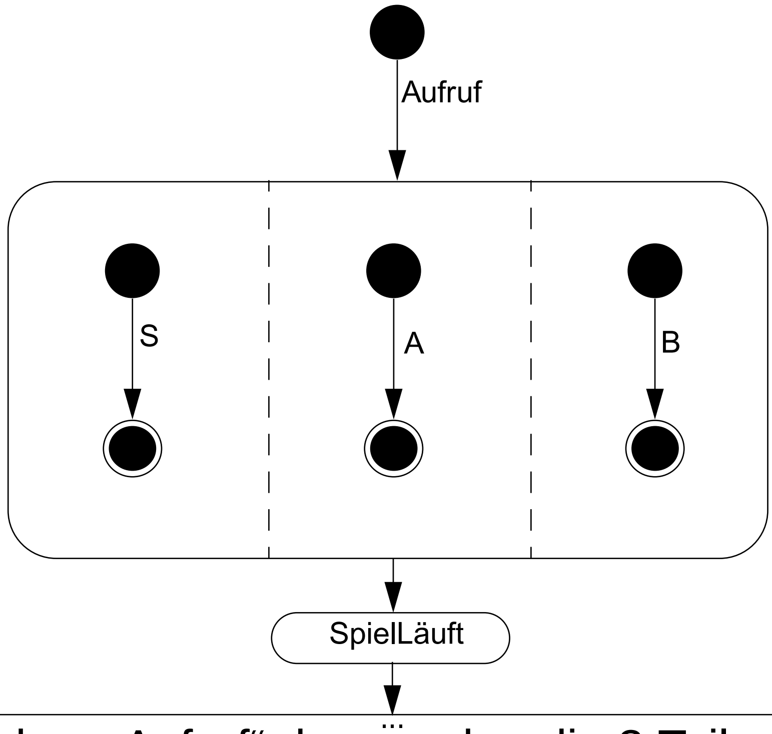
Teilautomat

Bedienung einer Uhr Einstellen von Zeit, Wecker, Stoppuhr

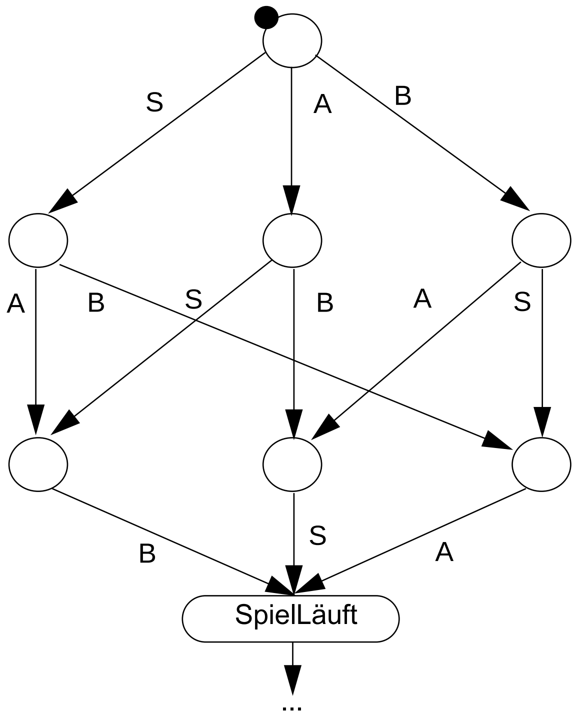


Modellierung von Nebenläufigkeit: Beginn eines Tennisspieles

UML Statechart



Det. endlicher Automat



Mit dem „Aufruf“ des werden die 3 Teilautomaten des mittleren Zustandes „gleichzeitig“ aktiviert.

Sie führen jeweils einen Übergang aus (Ankunft von Schiedsrichter, Spieler A, Spieler B).

Wenn sie ihre Endzustände erreicht haben, wird der zusammengesetzte Zustand verlassen.

Der gleichbedeutende **endliche Automat** modelliert **alle Reihenfolgen der Übergänge** S, A, B.

Das **Statechart abstrahiert** davon.

7.2 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**.

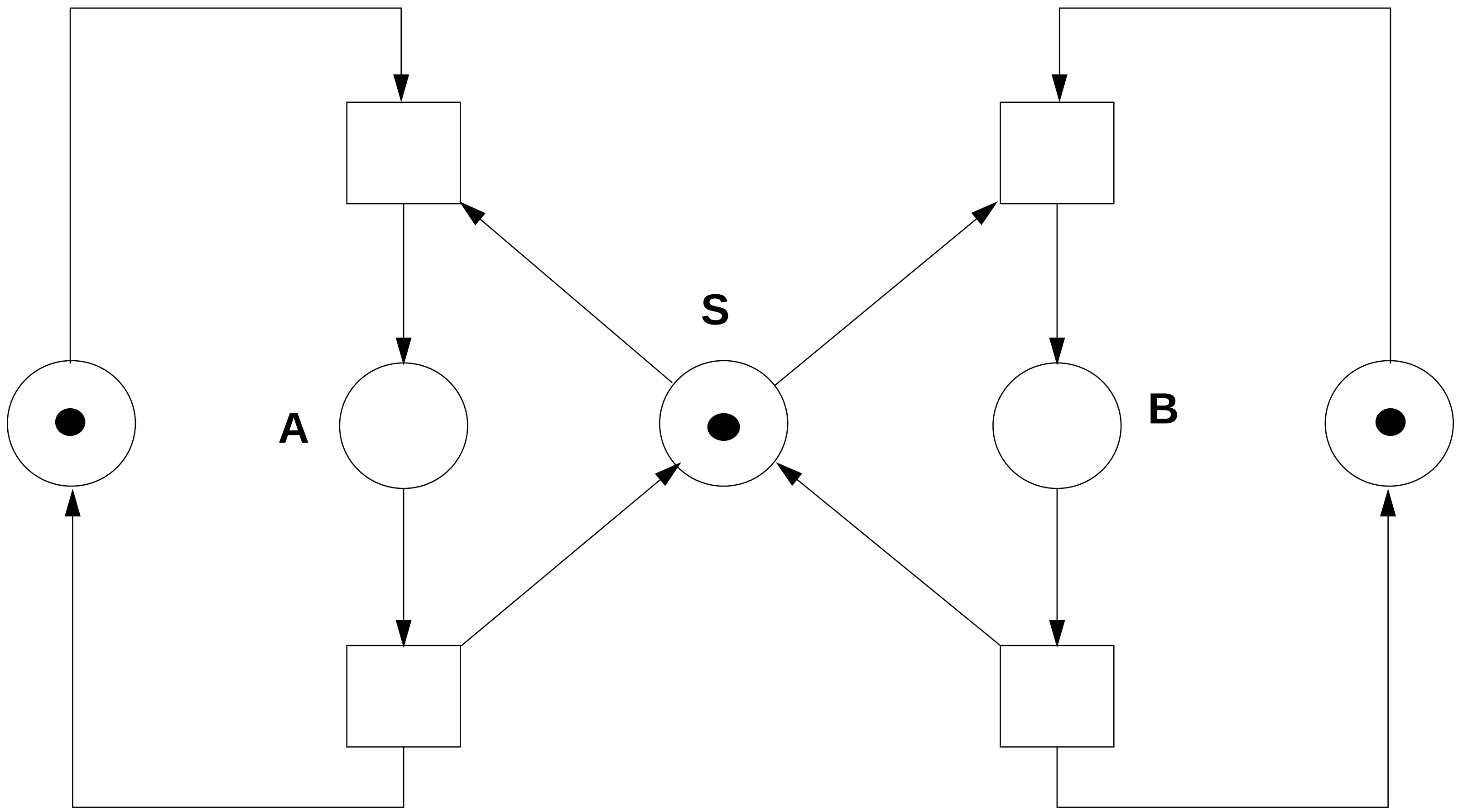
C. A. Petri hat sie 1962 eingeführt.
Es gibt zahlreiche Varianten und Verfeinerungen von Petri-Netzen. Hier nur die Grundform.

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

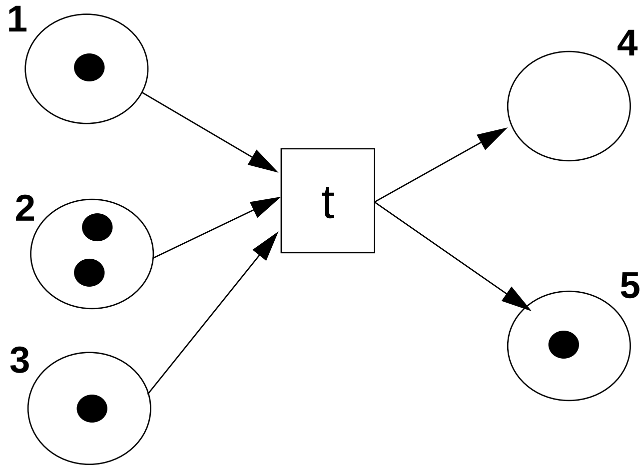
T **Menge von Transitionen** oder Übergänge,
repräsentieren Aktivitäten; graphisch Rechtecke

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‘.

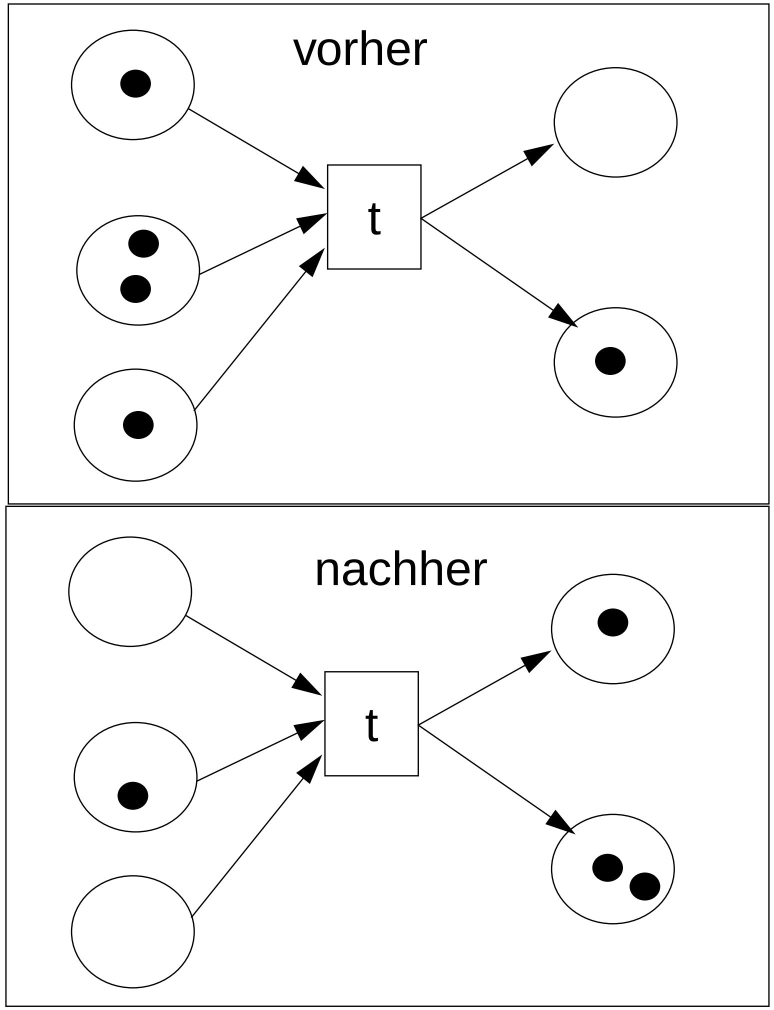
Eine **Transition t kann schalten**, wenn für alle Stellen s $\in$ Vorbereich (t) gilt M(s) $\geq$ 1.

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!

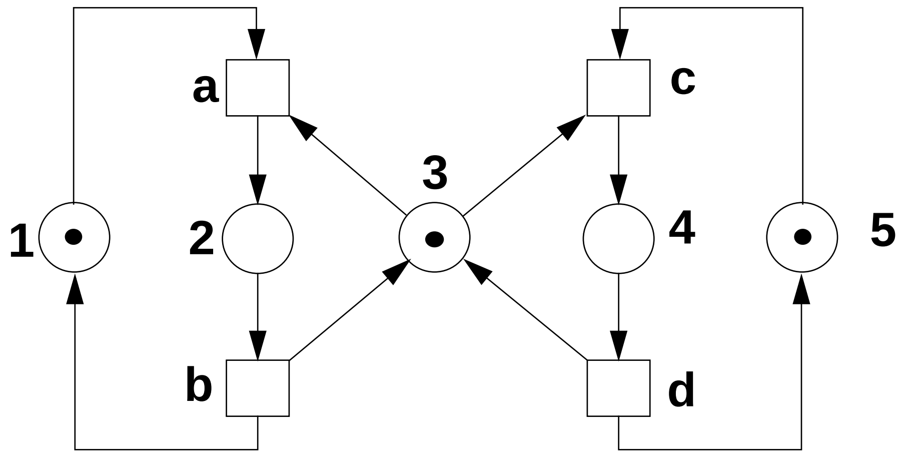
Zwei Transitionen mit gemeinsamen Stellen im Vorbereich können (bei passender Markierung) im **Konflikt** stehen:
Jede kann schalten, aber nicht beide nacheinander.

Markierungen

Zu jedem Petri-Netz wird eine
Anfangsmarkierung M_0 angeben.

z. B. $\mathbf{M_0} = (1, 0, 1, 0, 1)$

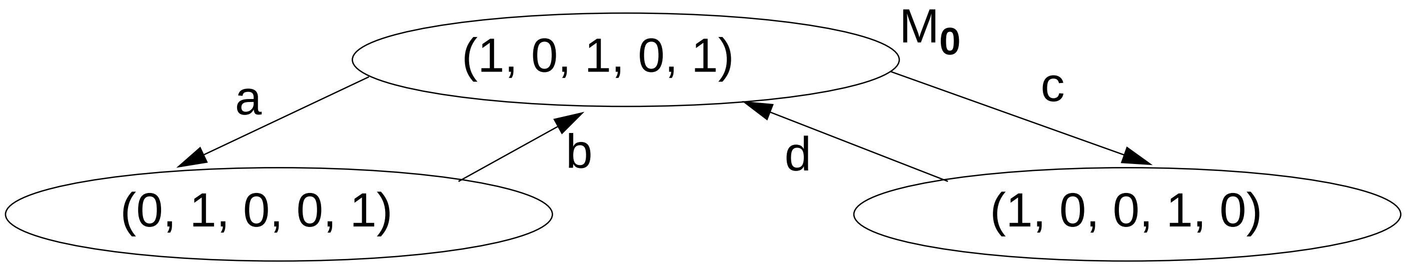
Wir sagen, eine **Markierung M_2 ist von einer Markierung M_1 aus erreichbar**, wenn es ausgehend von M_1 eine Folge von Transitionen gibt, die nacheinander schalten und M_1 in M_2 überführen können.



Die Markierungen eines Petri-Netzes kann man als gerichteten **Markierungsgraphen** darstellen:

- Knoten: erreichbare Markierung
- Kante x->y: Die Markierung x kann durch Schalten einer Transition in y übergehen.

Markierungsgraph



Schaltfolgen

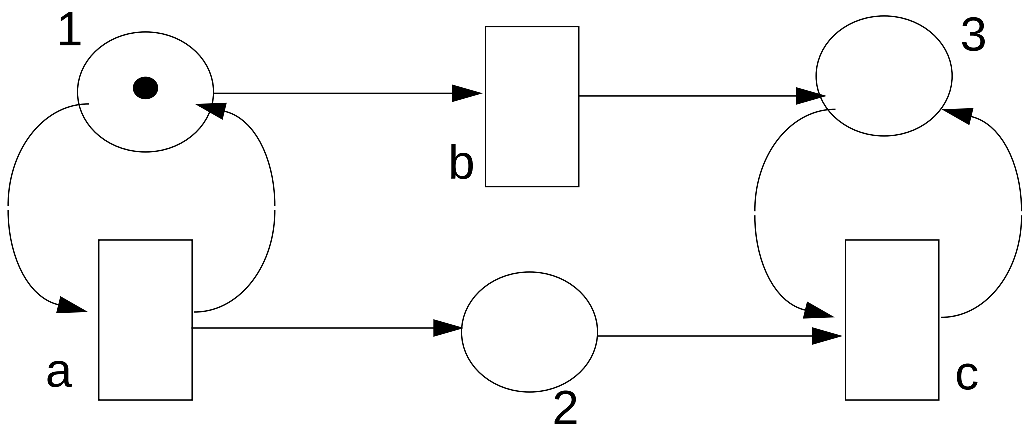
Schaltfolgen kann man angeben als

- Folge von Markierungen
- Folge der geschalteten Transitionen

Beispiel für eine **Schaltfolge** zum Petri-Netz auf Mod-7.19:

(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:

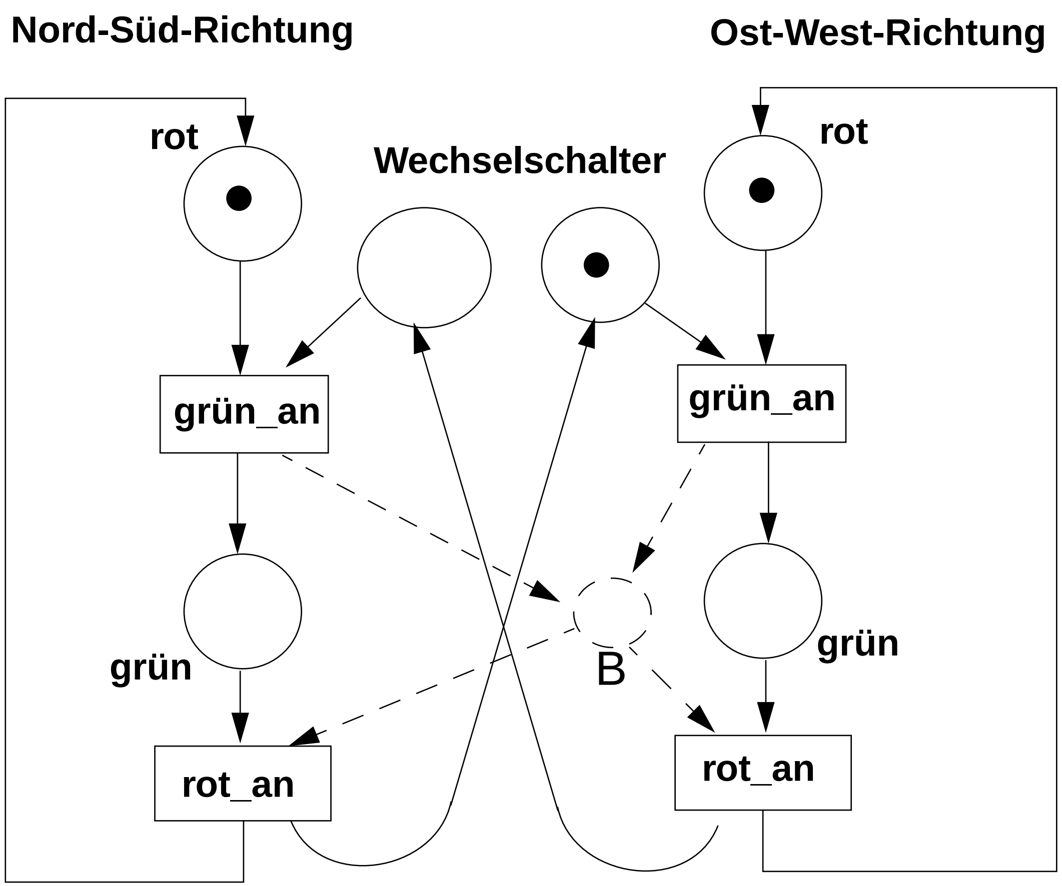
$a^{\mathbf{n}}\ b\ c^{\mathbf{n}}$

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

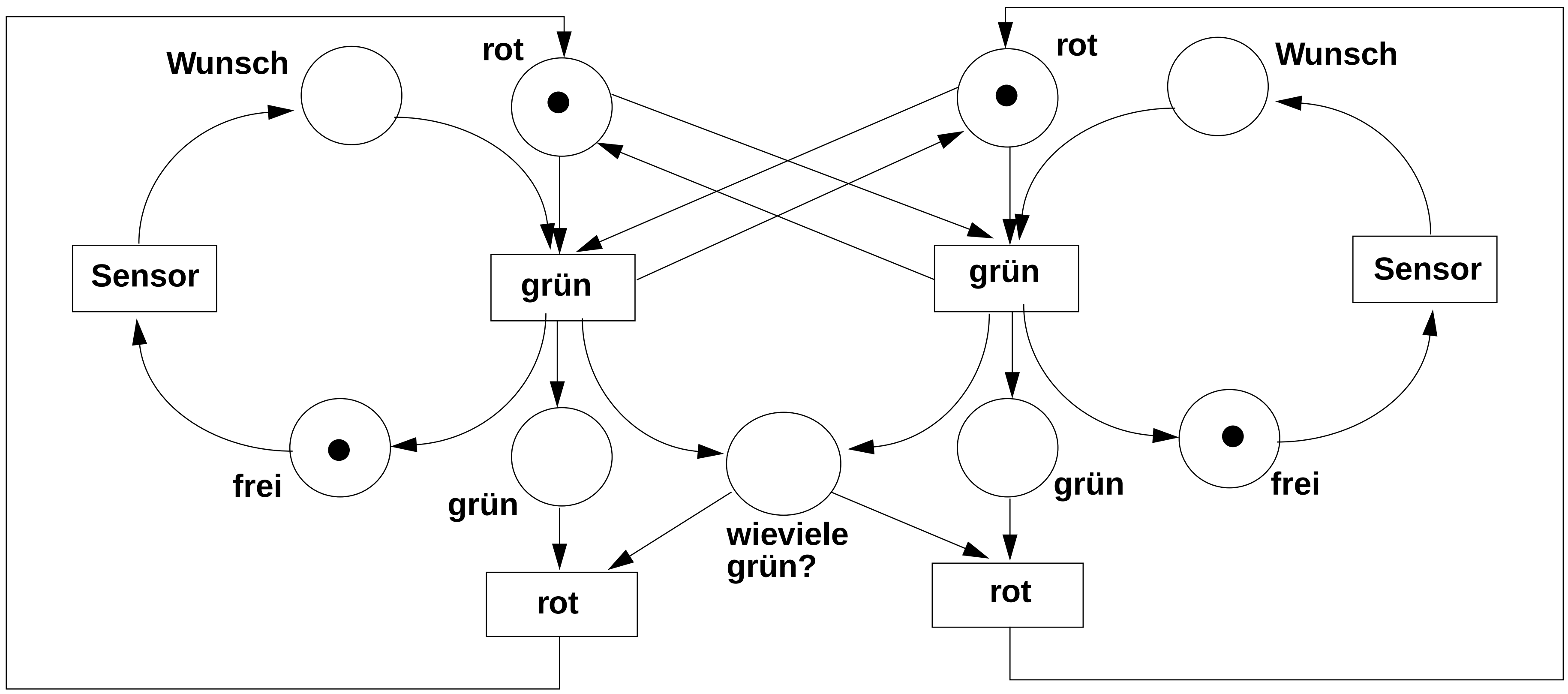


Beispiel für ein binäres Netz

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

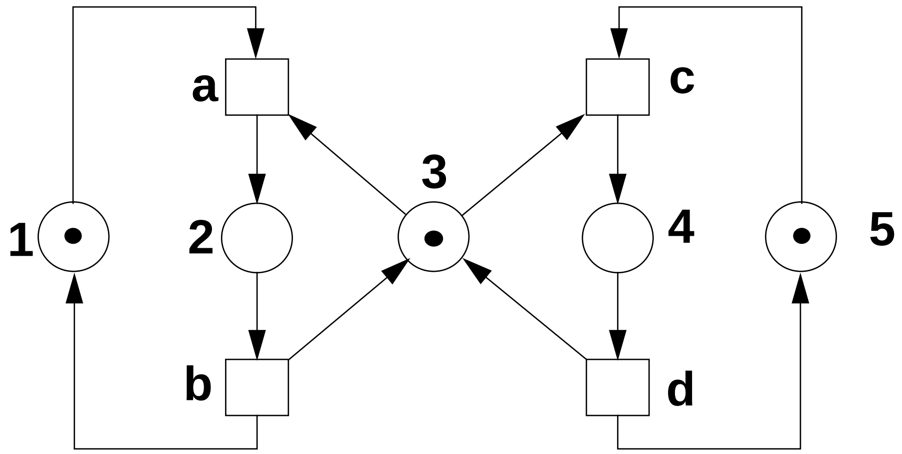
Petri-Netze modellieren häufig **Systeme, die nicht anhalten** sollen.

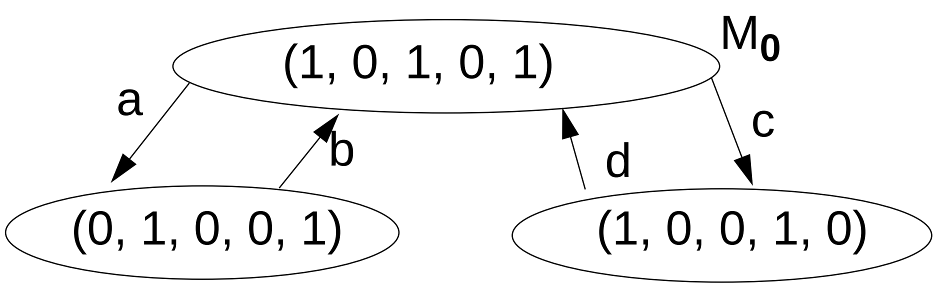
Ein Petri-Netz heißt **schwach lebendig**, wenn es zu jeder von M_0 erreichbaren Markierung eine Nachfolgemarkierung gibt.

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.

Beispiel für ein **lebendiges Petri-Netz** (Mod-7.19):





Verklemmungen

Verklemmung: Ein System kann unerwünscht anhalten,
weil das **Schalten einiger Transitionen zyklisch voneinander abhängt**.

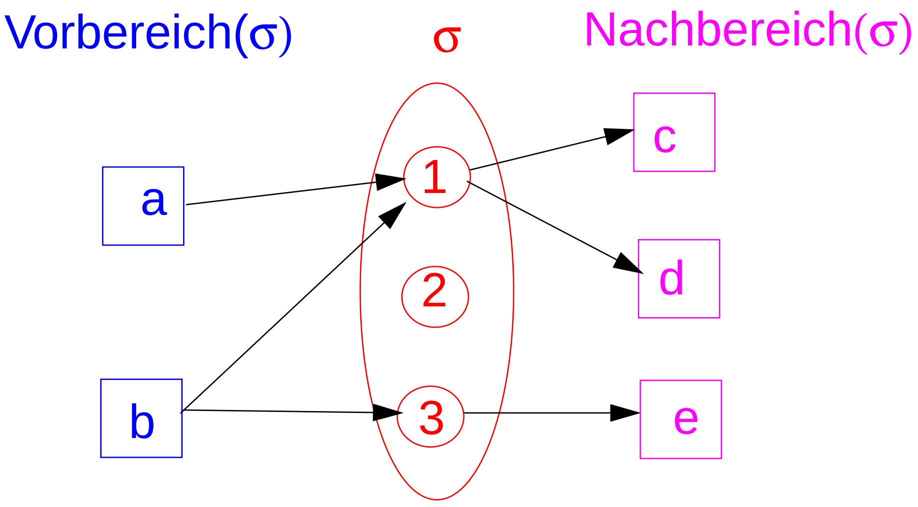
Sei: $\sigma \subseteq S$ eine Teilmenge der Stellen eines Petri-Netzes und

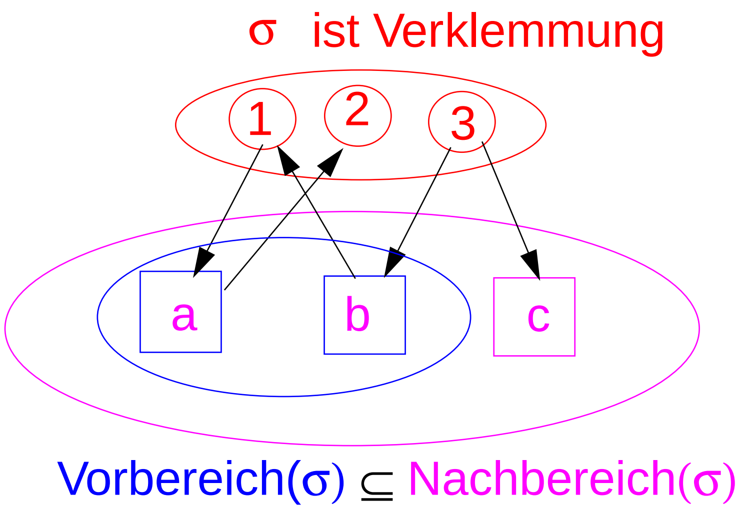
Vorbereich (σ) $:= \{t \mid \exists\, s \in \sigma : (t, s) \in F\}$,
d. h. die Transitionen, die auf Stellen in σ wirken

Nachbereich (σ) $:= \{t \mid \exists\, s \in \sigma : (s, t) \in F\}$,
d. h. die Transitionen, die Stellen in σ als Vorbedingung haben

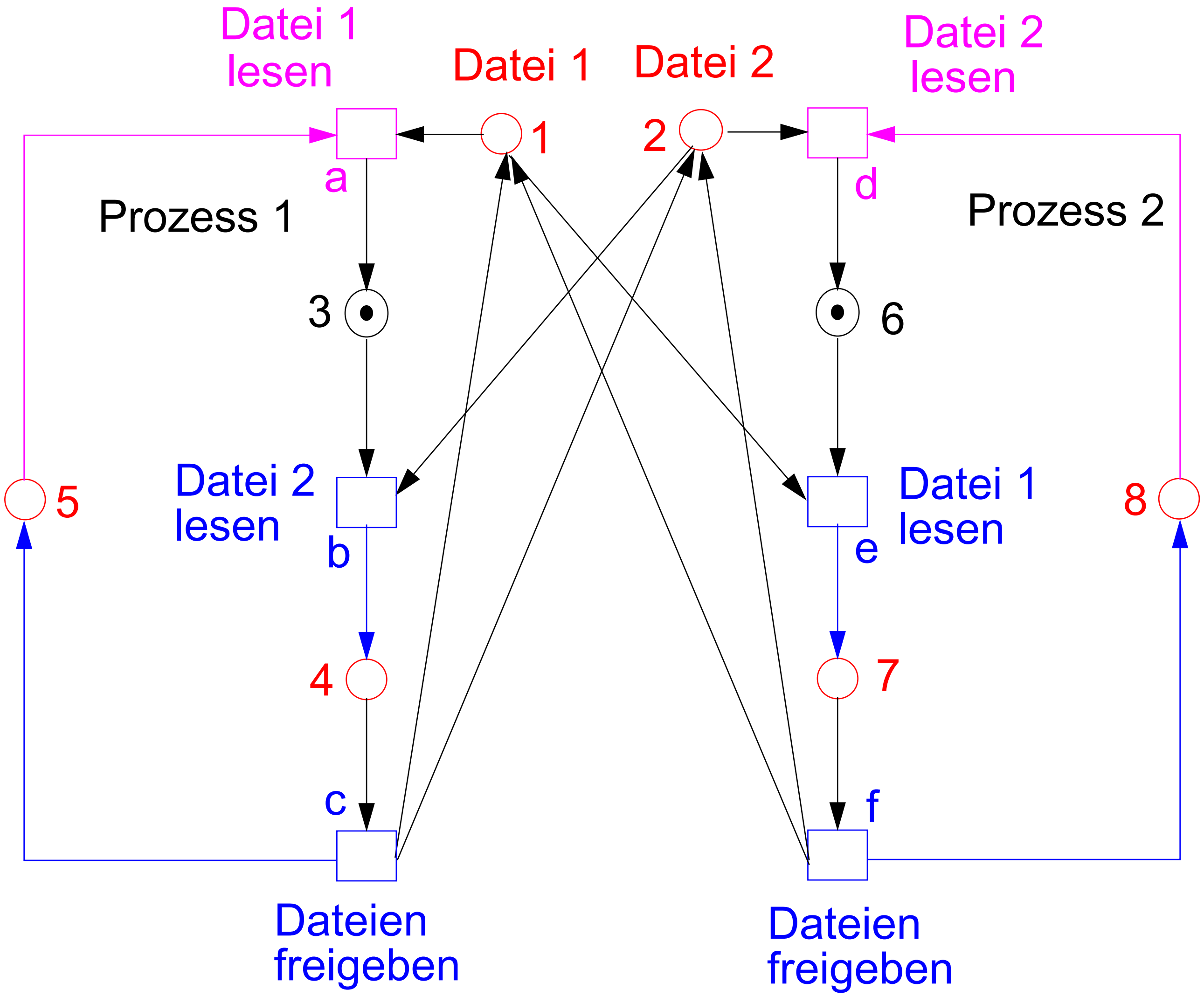
Dann ist σ eine **Verklemmung, wenn Vorbereich** (σ) $\subseteq$ **Nachbereich** (σ).

Wenn **für alle s** $\in$ σ **gilt M (s) = 0**, dann kann es **keine Marken auf Stellen in** σ in einer Nachfolgemarkierung von M geben.





Verklemmung beim Lesen von Dateien



s = {1, 2, 4, 5, 7, 8}

Vorbereich (s)
= {b, c, e, f}

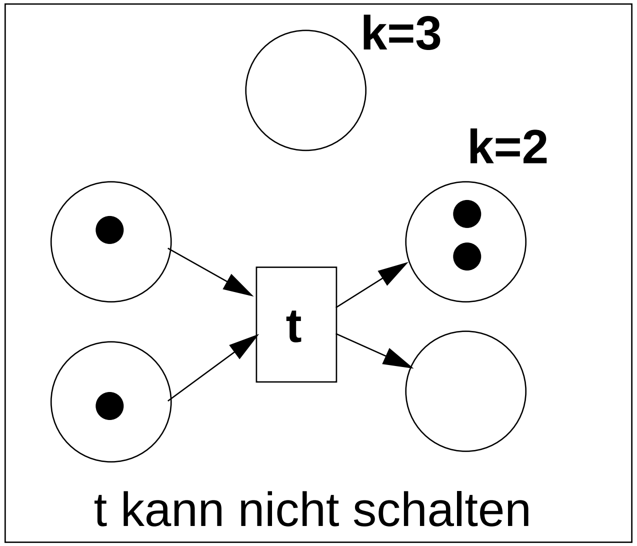
Nachbereich (s)
= {a, b, c, d, e, f}

M (s) = 0

Anfangsmarkierung:
(1, 1, 0, 0, 1, 0, 0, 1)

Kapazitäten und Gewichte

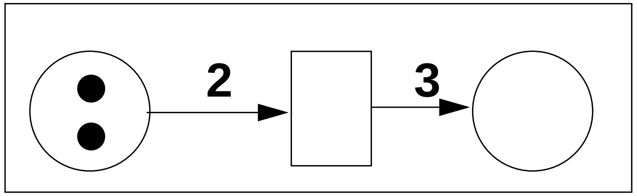
Man kann **Stellen eine begrenzte Kapazität** von $k \in \mathbb{N}$ Marken zuordnen.



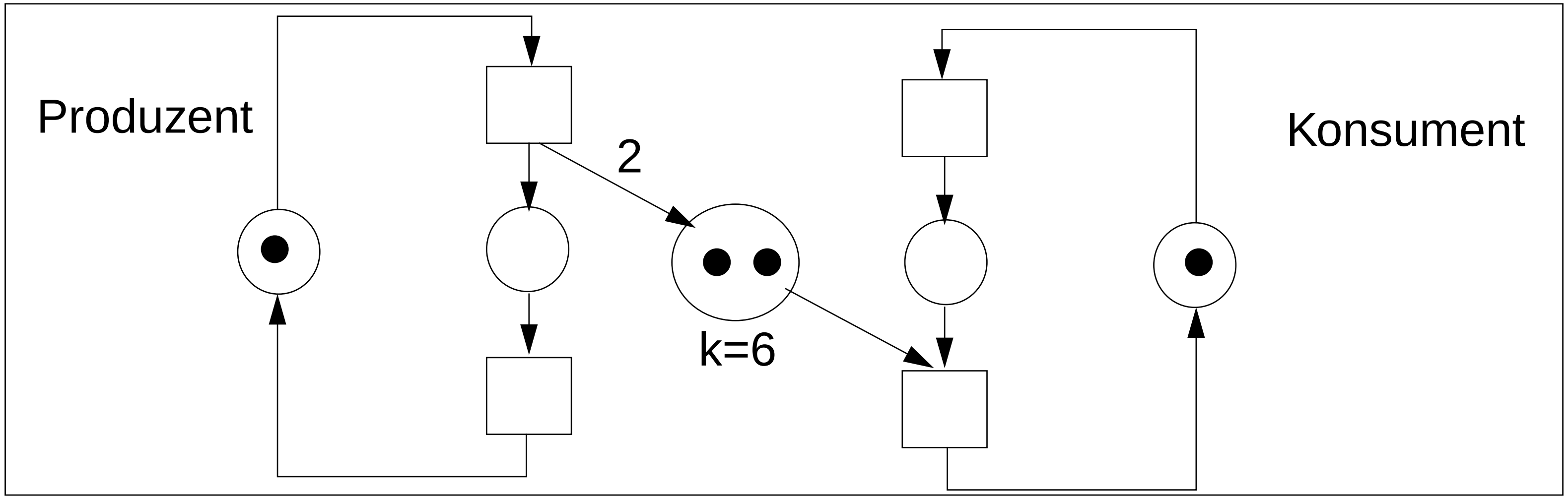
Die Bedingung, dass eine **Transition t schalten kann**, wird erweitert um:

Die **Kapazität keiner der Stellen im Nachbereich von t darf überschritten** werden.

Kanten kann ein **Gewicht** $n \in \mathbb{N}$ zugeordnet werden: sie bewegen **beim Schalten n Marken**.



Beispiel: **Beschränkter Puffer**



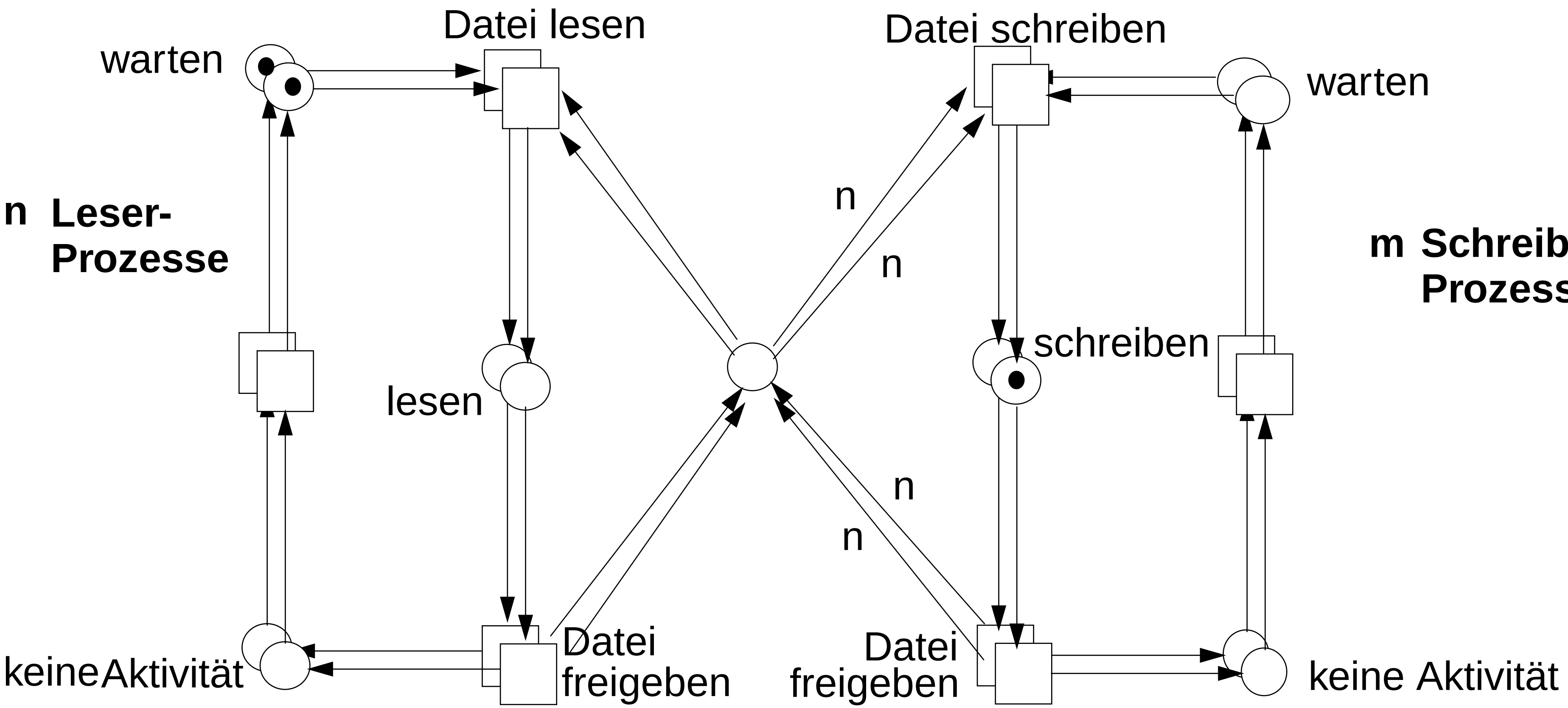
Beispiel: Leser-Schreiber-System

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**.



8 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

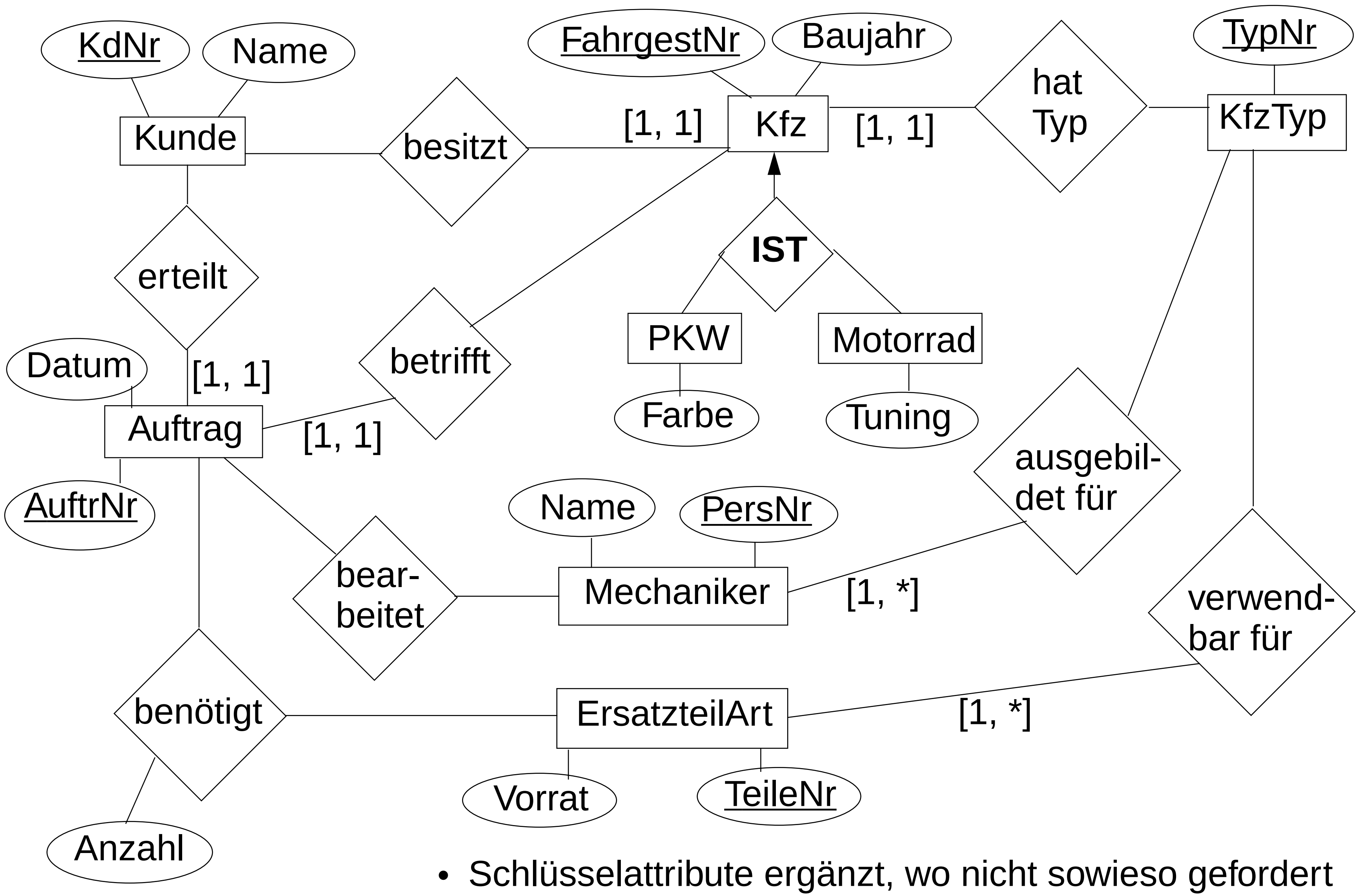
Teilaufgaben:

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

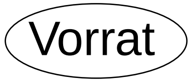
8.1.a Informationsstruktur als ER-Modell



- Schlüsselattribute ergänzt, wo nicht sowieso gefordert
- Kardinalitäten: plausibel ohne unnötig einzuschränken

Vergleich ER-Modell und Wertebereiche

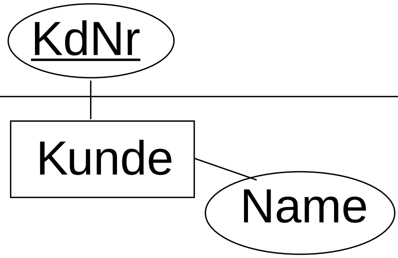
Attribut



Vorrat := $\mathbb{N}_0$

Wertemenge

Schlüsselattribut



KdNr := $\mathbb{N}_0$

Indexmenge

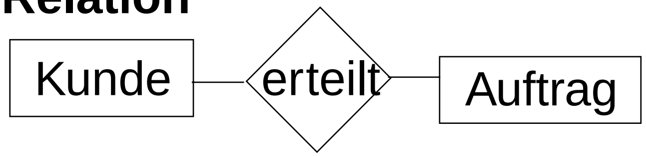
Entity-Typ

Kunde := KdNr $\times$ Name

kartesisches Produkt

ohne Identität der Entities

Relation



erteilt := Pow (Kunde $\times$ Auftrag)

Relation

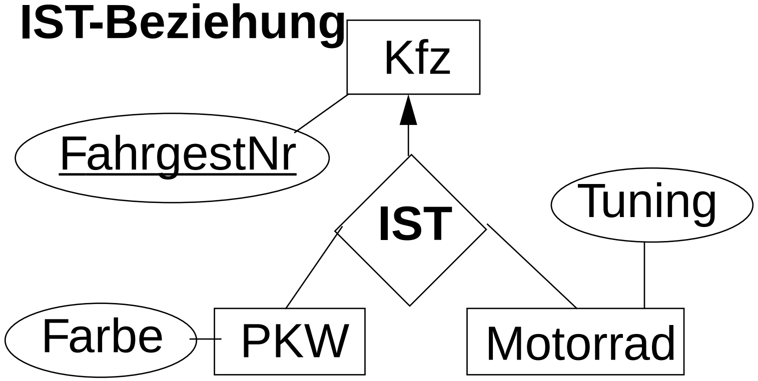
Kardinalität [1, 3]

Prädikatenlogik:
$\forall$ a $\in$ Auftrag: $1 \leq |\{ (x,y) \mid (x,y) \in \text{erteilt} \wedge y=a\}| \leq 3$

[1, 1]

erteilt: Auftrag -> Kunde **Funktion** (hier: total)

IST-Beziehung



kartesisches Produkt und disjunkte Vereinigung:

Kfz := FahrgestNr $\times$ KfzVarianten

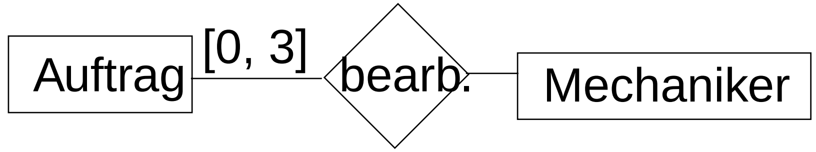
KfzArten := { istPKW, istMotorrad }

KfzVarianten := { (istPKW, p) | p $\in$ Farbe } $\cup$
{ (istMotorrad, m) | m $\in$ Tuning}

8.1.b Bedingungen

Ein Auftrag soll von höchstens 3 Mechanikern bearbeitet werden:

ER Kardinalität:



Prädikatenlogik: $\forall\, a \in \text{Auftrag}: 0 \leq |\{ (x,y) \mid (x,y) \in \text{bearb.} \wedge x=a\}| \leq 3$

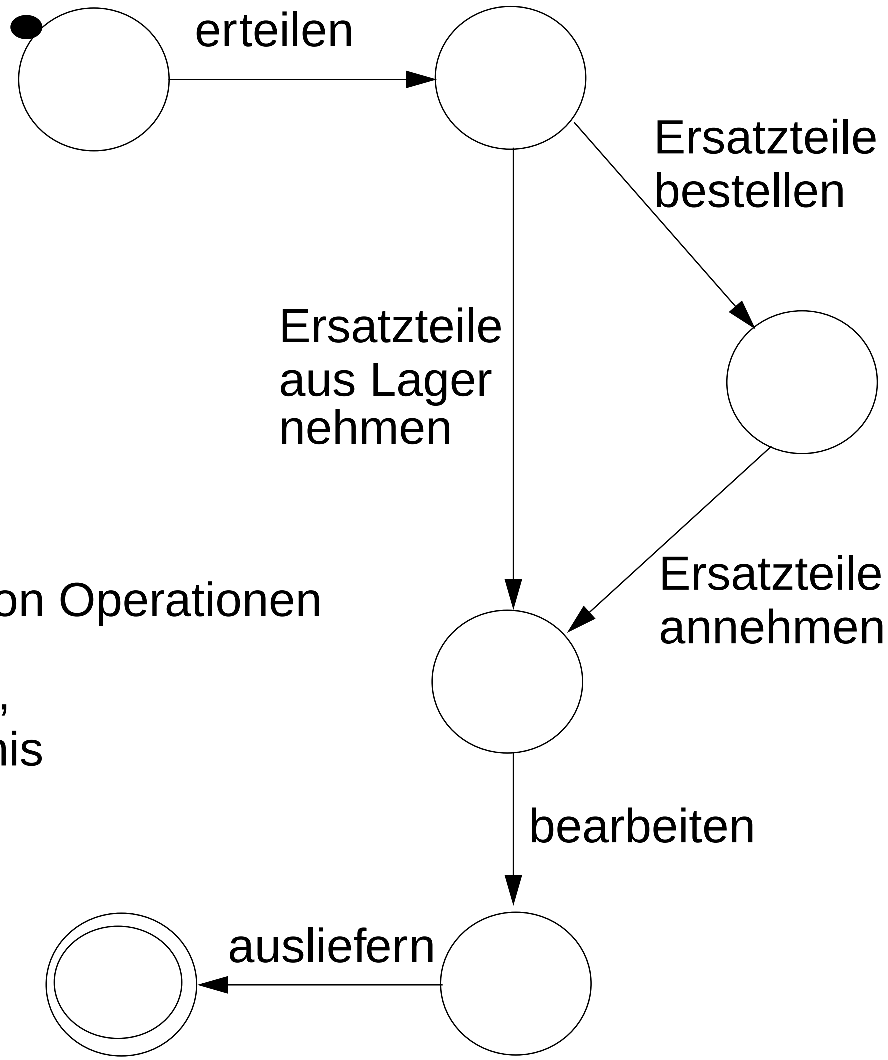
Ein Auftrag soll nur dann angenommen werden, wenn für den betreffenden KfzTyp auch Mechaniker ausgebildet sind.

Prädikatenlogik: $\forall\, a \in \text{Auftrag}: \forall\, k \in \text{Kfz}: \forall\, t \in \text{KfzTyp}:$

$$((a, k) \in \text{betrifft} \wedge (k, t) \in \text{hatTyp}) \rightarrow$$
$$\exists\, m \in \text{Mechaniker}: (m, t) \in \text{ausgebildet}$$

8.1.c Ablauf der Auftragsbearbeitung (DEA)

- Auftrag wird **erteilt**,
- Verfügbarkeit der Ersatzteile **geprüft**,
- ggf. **bestellt**,
- von einem Mechaniker **bearbeitet**,
- Kraftfahrzeug wird dem Kunden **ausgeliefert**.

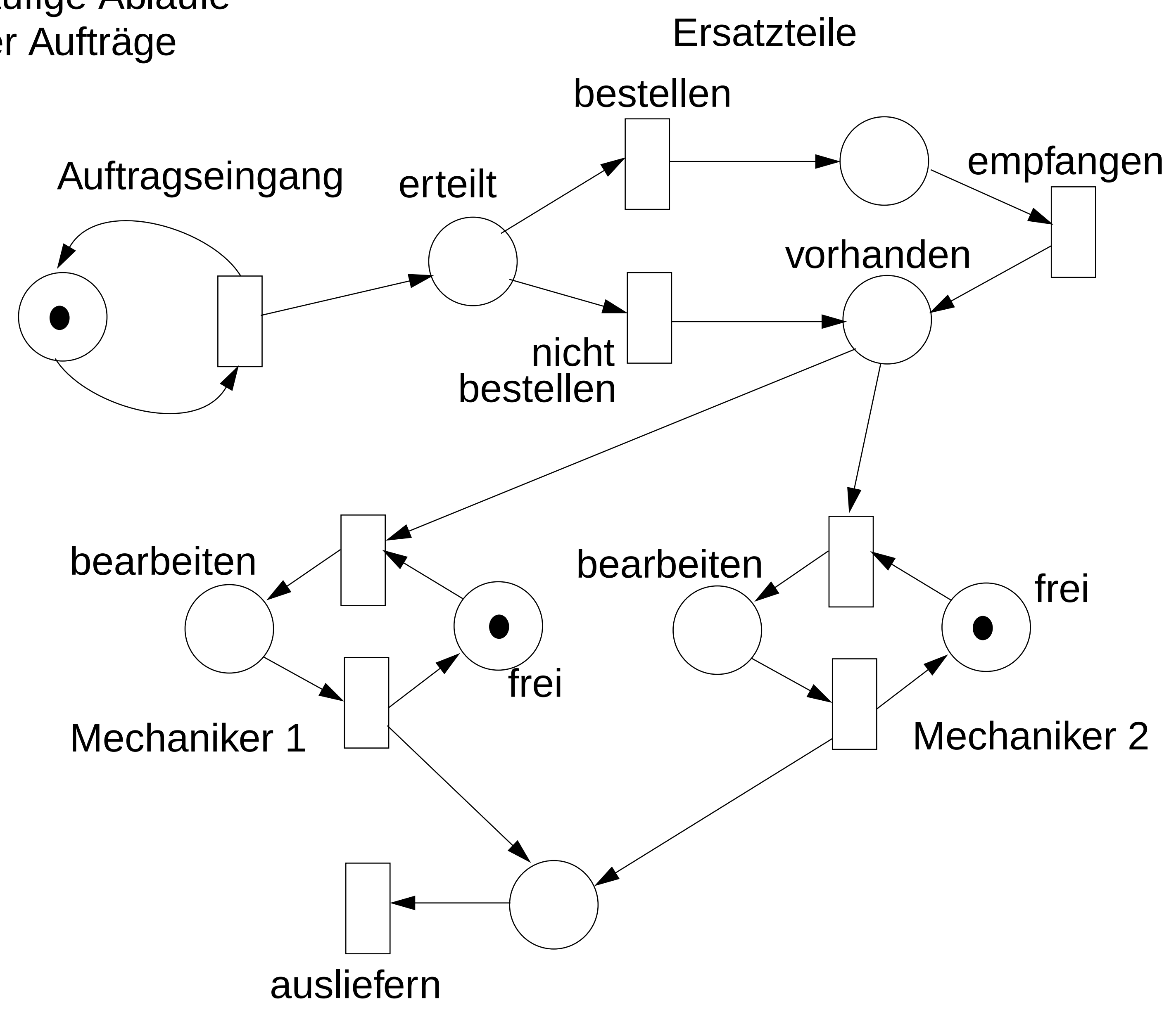


Deterministischer, endlicher Automat
beschreibt streng **sequentielle Abfolge** von Operationen

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:

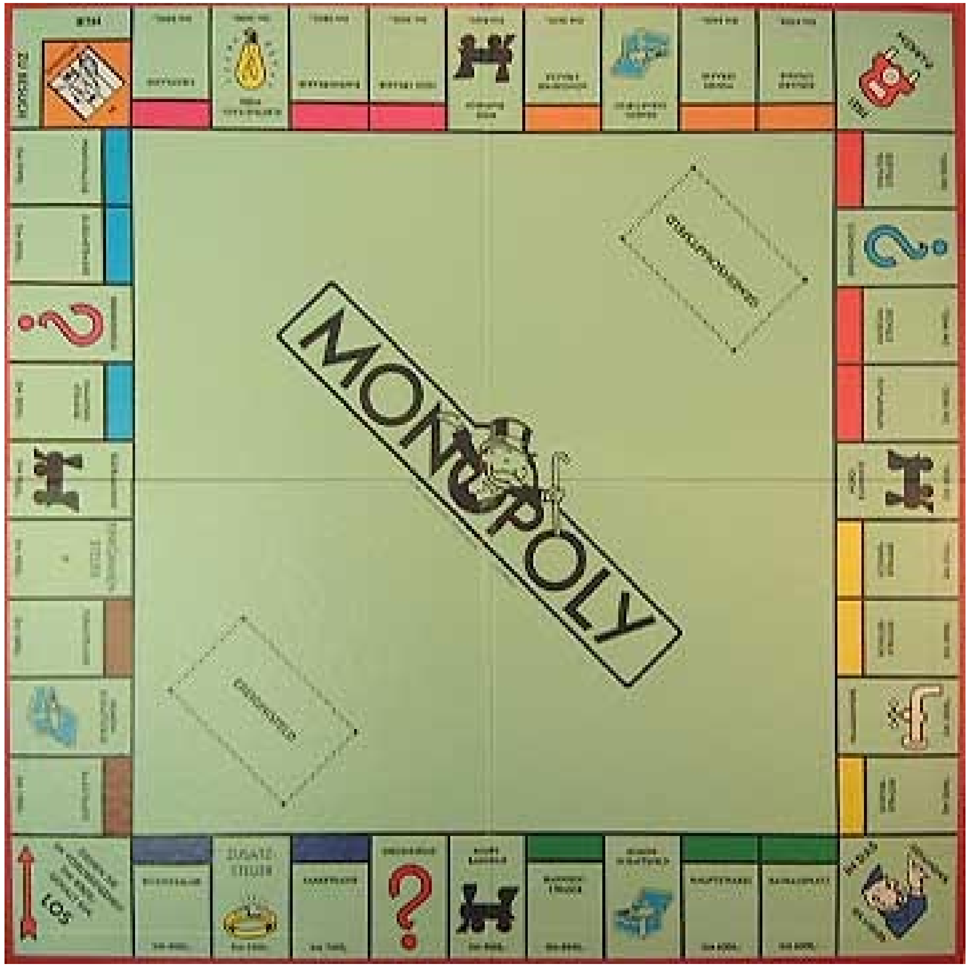
Fallstudie 2: Monopoly-Spiel

Wir modellieren Struktur und Ablauf des Monopoly-Spiels.

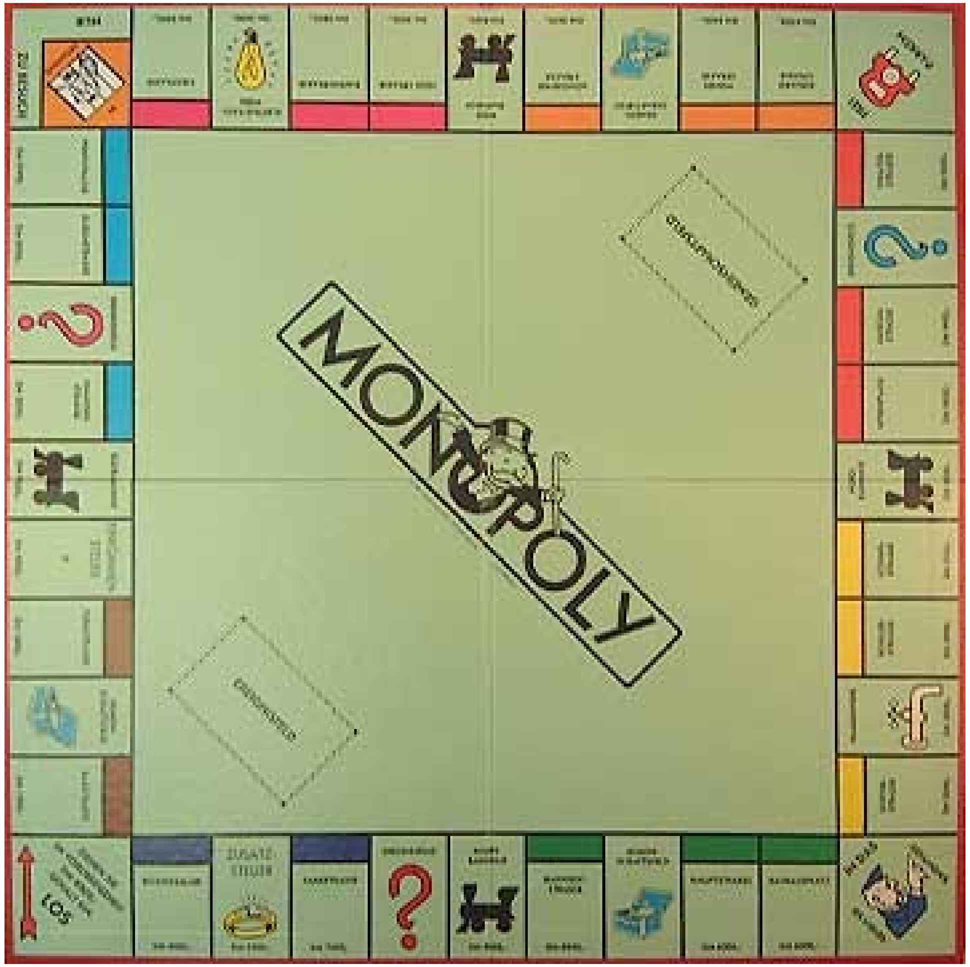
Ziel: Spielregeln präzisieren und formalisieren

Teilaufgaben:

1. **Informationen und Zusammenhänge**
2. **Bedingungen und Regeln**
3. **Spielabläufe**

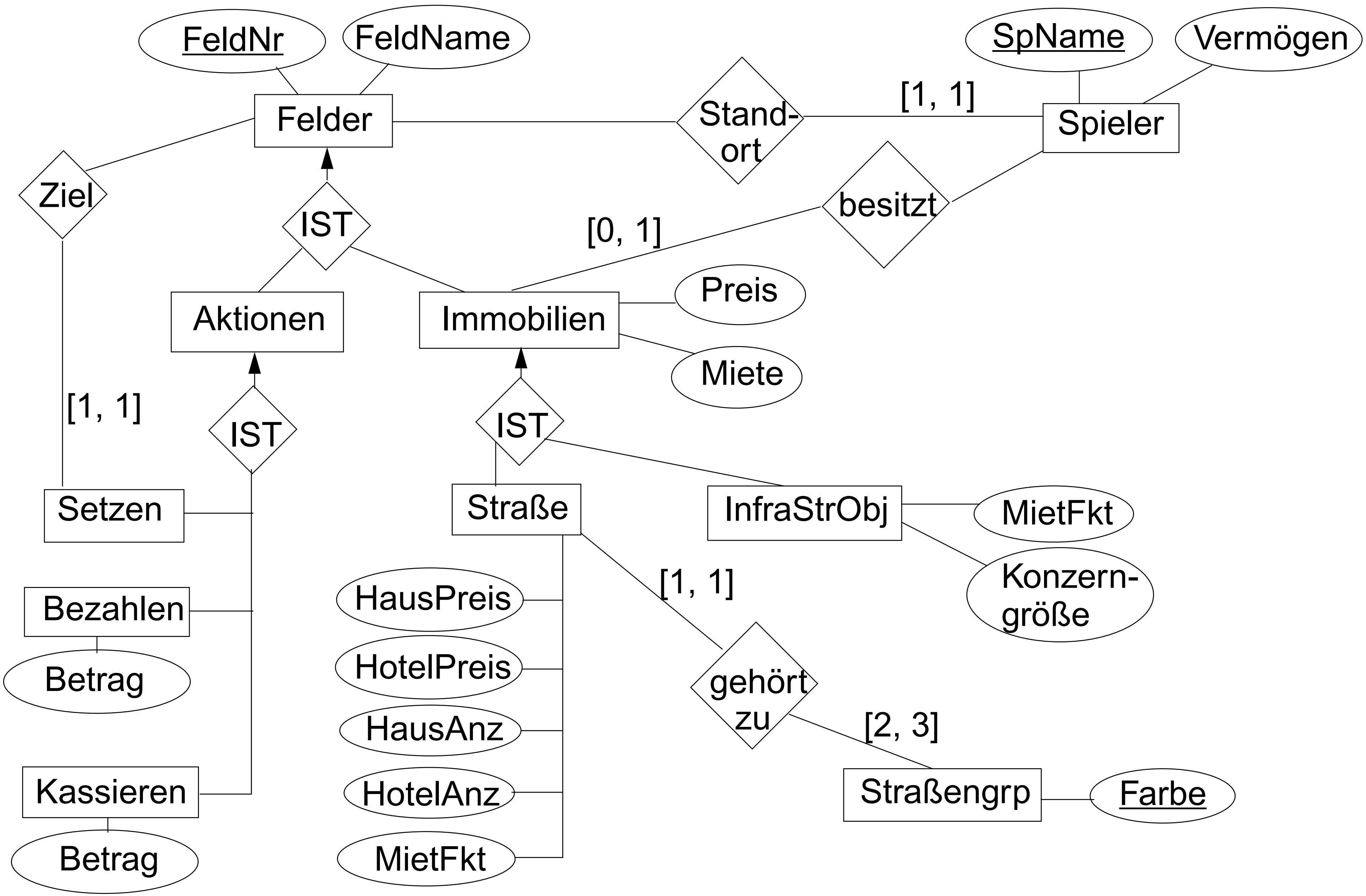


Kurzbeschreibung der Informationsstruktur



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 entwe-
   der 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

8.2.a 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

8.2.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.

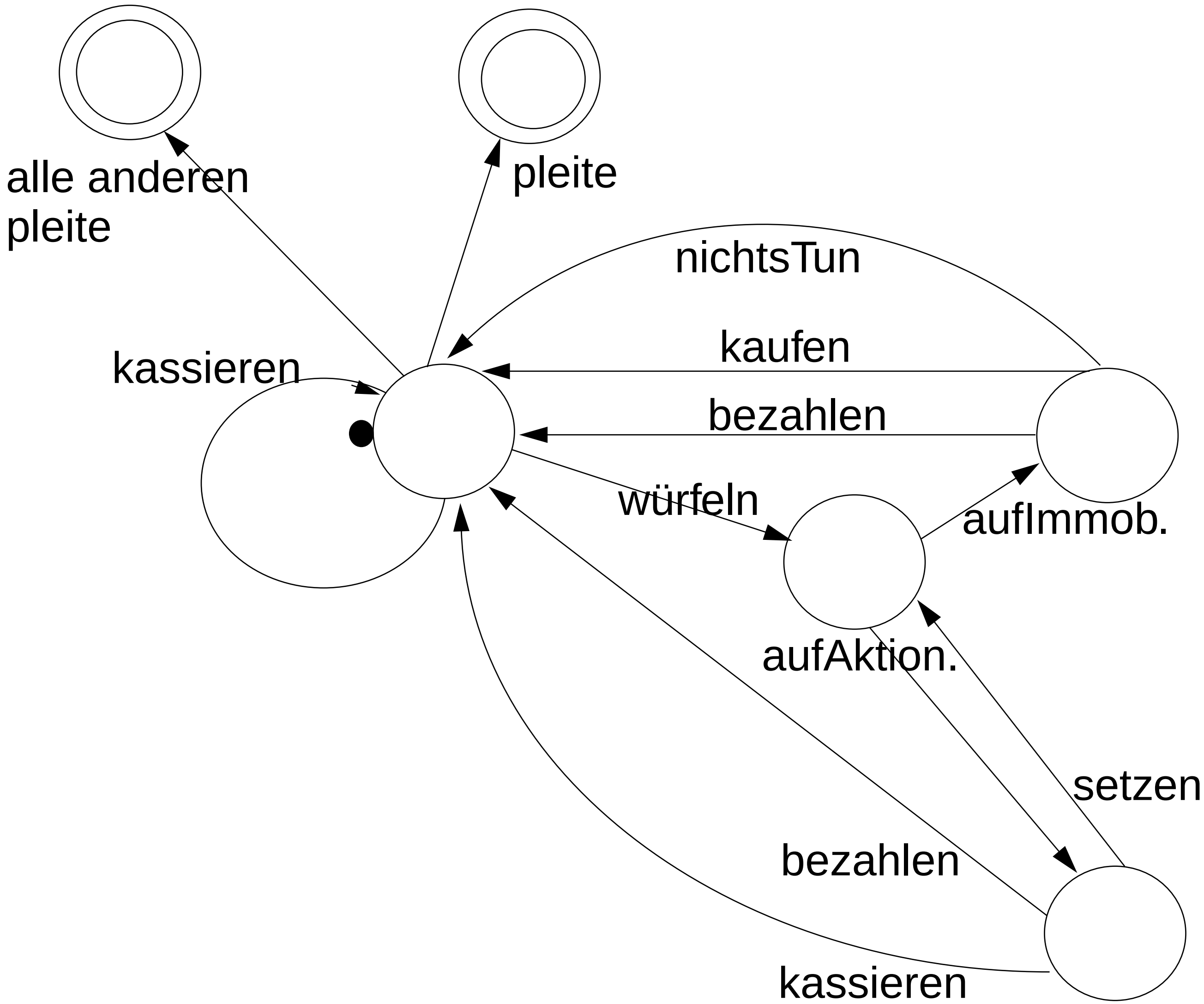
$\forall$ x $\in$ Felder: $\forall$ nr $\forall$ name $\forall$ p $\forall$ m $\forall$ hap $\forall$ hop $\forall$ haanz $\forall$ hoanz $\forall$ h

x = (nr, name, (istImmobilie, p, m, (istStraße, hap, hop, haanz, hoanz, h))) $\rightarrow$

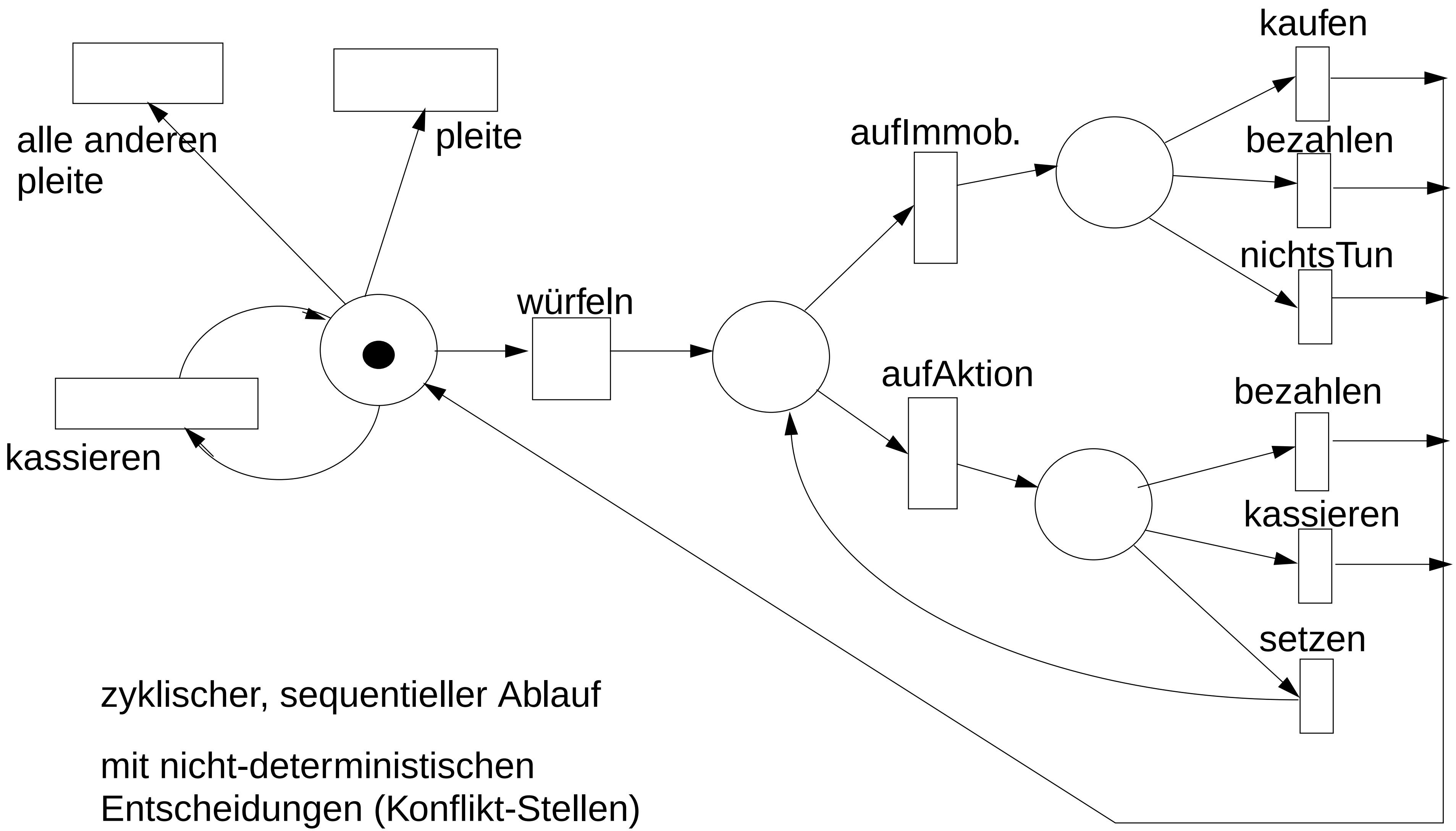
(haanz + hoanz > 0 $\wedge$ $\exists$ f $\in$ Farbe: (x, f) $\in$ gehörtZu $\wedge$ $\exists$ s $\in$ Spieler: (s, x) $\in$ besitzt

$\rightarrow$ $\forall$ g $\in$ Felder: (g, f) $\in$ gehörtZu $\rightarrow$ (s, g) $\in$ besitzt

2.c Aktionsfolgen eines Spielers (DEA)



8.2.c Aktionsfolgen eines Spielers (Petri-Netz)



zyklischer, sequentieller Ablauf

mit nicht-deterministischen
Entscheidungen (Konflikt-Stellen)

9 Zusammenfassung
Zusammenfassung der Themen und Begriffe (1)

1 Modellbegriff

2 Wertebereiche beschrieben d. Mengen

Mengen, extensional, intensional, Operationen
Potenzmengen
Kartesisches Produkt
Indexmengen
Folgen
Relationen, Eigenschaften von Relationen
Ordnungsrelationen
Funktionen, Eigenschaften,
spezielle Funktionen
disjunkte Vereinigung

2x Beweise verstehen und konstruieren

Satz, Voraussetzung, Behauptung, Beweis
Widerspruchsbeweis, Induktionsbeweis

3.1 Terme

Sorten, Signatur
korrekte Terme, Grundterme
Präfix-, Postfix-, Infix-Form, Funktionsform
Kantorowitsch-Bäume
Substitution
Umfassende Terme
Unifikation, allgemeinster Unifikator
Unifikationsverfahren

3.2 Algebren

Abstrakte Algebra, Axiome
Konkrete Algebra
Datenstrukturen: Keller, Binärbaum
Konstruktor, Hilfskonstruktor, Projektion
Normalform

Zusammenfassung der Themen und Begriffe (2)

4.1 Aussagenlogik

AL Formeln, logische Junktoren
Belegung, Interpretation
Wahrheitstafeln
erfüllbar, unerfüllbar, allgemeingültig (Tautologie)
Gesetze der booleschen Algebra
aussagenlogischer Schluss

4.2 Prädikatenlogik

PL Formeln,
gebundene und freie Variable
Wirkungsbereich von Quantoren
Umbenennung von Variablen
Interpretation von PL Formeln
Individuenbereich
Beschränkung von Wertebereichen
Umformungen, Normalformen
erfüllbar, unerfüllbar, allgemeingültig
PL Schluss

4.3 Verifikation (Hoaresche Logik)

Aussage charakterisiert Programmzustände
Zuweisungsregel
Konsequenzregeln, Sequenzregel,
2-seitige Alternative, bedingte Anweisung,
Schleife, Schleifeninvariante,
Schleife aus Invariante konstruieren
Terminierung von Schleifen

Zusammenfassung der Themen und Begriffe (3)

5 Graphen

5.1 Grundlegende Definitionen

Gerichtetet, ungerichteter Graph,
Multigraph, Teilgraph,
Grad, Eingangs-, Ausgangsgrad
Adjazenzmatrix, Adjazenzlisten

5.2 Wegeproblem

Weg, Kreis, Zyklus,
gerichteter azyklischer Graph,
zusammenhängend,
Zusammenhangskomponente,
Euler-Weg, Euler-Kreis, Hamilton-Kreis

5.3 Verbindungsprobleme

Baum, Spannbaum,
Schnittknoten, Brückenkante
orientierbarer Graph

5.4 Modellierung mit Bäumen

Gerichteter Baum, Wurzel, Höhe, Blätter
Binärbäume,
Entscheidungsbäume
Strukturbäume

5.5 Zuordnungsprobleme

Paarweise Zuordnung (Matching),
bipartit,
Färbung

5.6 Abhängigkeitsprobleme

Abhängigkeitsparagraph,
Anordnung (Scheduling),
Ablaufparagraph,
Aufrufgraph,
Programmablaufgraph

Zusammenfassung der Themen und Begriffe (4)

6. Modellierung von Strukturen

6.1 Kontextfreie Grammatiken

Terminale, Nichtterminale, Startsymbol
Produktionen,
Ableitung, Sprache einer KFG,
Ableitungsbaum

6.2 Baumstrukturen in XML

XML-Sprachen, Tag-Klammern,
KFG definiert Bäume (entspr. DTD)

6.3 Entity Relationship Modell

Entity-Menge, konkrete Ausprägung,
Attribut, Schlüsselattribut
Relation, Rollen, Kardinalität
IST-Spezialisierung

6.4 Klassendiagramme in UML

Vergleich mit ERM

7. Modellierung von Abläufen

7.1 Endliche Automaten

Alphabet, reguläre Ausdrücke
deterministisch, nicht-deterministisch
Zustände, Übergangsfunktion
akzeptierte Sprache
NEA-DEA-Konstruktion,
Ausgabe, Mealy-Automat, Moore-Automat,
UML Statecharts

7.2 Petri-Netze

Stellen, Transitionen, Markierungsfunktion,
Schaltregel, Markierungsgraph,
zyklische Prozesse, binäres Netz,
Lebendigkeit, Verklemmung (deadlock),
Kapazitäten, Gewichte, beschränkter Puffer,
Leser-Schreiber-System

8. Fallstudien

Auftragsabwicklung in Autowerkstatt
Monopoly-Spiel
Getränkeautomat (Übungen)