Formale Grundlagen
2014S

V. Pillwein

29. August 2014

Inhaltsverzeichnis

1 Grundbegriffe Mengen

Eine Menge ist eine Zusammenfassung von Objekten, wobei klar sein muss, ob ein Objekt zur Menge gehört oder nicht. Beispiele für Menge sind

- $M = \{1, 3, 7, 12, 47\}$
- $M = \{\text{rot}, \text{schwarz}, \text{blau}\}$
- $M = \{\Delta, \bigcirc, \spadesuit, \square, \diamondsuit\}$

Diesen Beispielen ist gemeinsam, dass die Mengen aufzählend angegeben sind und die Mengen *endlich* sind. Die Reihenfolge der Objekte spielt in einer Menge keine Rolle, d.h., $\{a, b, c\} = \{c, b, a\}$. Jedes Element der Menge wird nur einmal aufgeführt, d.h. wir betrachten keine Mengen der Form $\{a, a, b, b, b\}$ (keine sogenannten Multisets).

Definition 1.1. *Die Objekte einer Menge M heissen* Elemente *von M. Wir schreiben $m \in M$ für "m ist ein Element der Menge M" und $m \notin M$ für "m ist* kein *Element der Menge M".*

Einige Mengenbezeichnungen, die in der Vorlesung verwendet werden:

- leere Menge: $\emptyset$, oder $\{\ \}$
- natürliche Zahlen: $\mathbb{N} = \{0, 1, 2, 3, \dots\}$; die positiven natürlichen Zahlen werden mit $\mathbb{N}^* = \mathbb{N}\backslash\{0\}$ bezeichnet
- ganze Zahlen: $\mathbb{Z} = \{\dots, -4, -3, -2, -1, 0, 1, 2, 3, 4, \dots\}$
- rationale Zahlen: $\mathbb{Q} = \{\frac{p}{q} \mid p \in \mathbb{Z},\ q \in \mathbb{N}^*\}$
- reelle Zahlen: $\mathbb{R}$, wobei

$$\begin{aligned} \mathbb{R}^+ &= \{x \in \mathbb{R} \mid x > 0\}, \quad \text{und} \quad \mathbb{R}_0^+ = \mathbb{R}^+ \cup \{0\}, \quad \text{bzw.} \\ \mathbb{R}^- &= \{x \in \mathbb{R} \mid x < 0\}, \quad \text{und} \quad \mathbb{R}_0^- = \mathbb{R}^- \cup \{0\} \end{aligned}$$

Zu einer gegebenen Menge X definieren wir die *Potenzmenge* $P(X)$ als die Menge aller Teilmengen von X, d.h.,

$$P(X) = \{Y \mid Y \subseteq X\}.$$

Damit gilt insbesondere $\emptyset \in P(X)$ und $X \in P(X)$.

Beispiel 1.2. *Sei $A = \{1, 2, 3\}$, dann ist*

$$P(A) = \{\emptyset, \{1\}, \{2\}, \{3\}, \{1, 2\}, \{1, 3\}, \{2, 3\}, \{1, 2, 3\}\}.$$

Definition 1.3. *Die* Kardinalität *oder* Mächtigkeit *einer Menge X ist die Anzahl der Elemente in X und wird mit $|X|$ bezeichnet, d.h., sie ist definiert als*

$$|X| = \begin{cases} n, & \textit{falls } X \textit{ genau } n \textit{ Elemente besitzt} \\ \infty, & \textit{falls } X \textit{ eine unendliche Menge ist.} \end{cases}$$

Beispiel 1.4.

$$|\{1,3,7,12\}| = 4, \quad |\emptyset| = 0, \quad |\mathbb{N}| = \infty.$$

Definition 1.5. *Seien X, Y nichtleere Mengen. Die* Produktmenge *(oder das* kartesische Produkt*) $X \times Y$ von X und Y ist definiert als*

$$X \times Y = \{(x, y) \mid x \in X \text{ und } y \in Y\}.$$

Die Elemente (x, y) in $X \times Y$ heissen Tupel *oder* geordnetes Paar*. Zwei Tupel sind gleich, wenn sie in allen Komponenten übereinstimmen, d.h., $(a, b) = (c, d)$ genau dann, wenn $a = c$ und $b = d$.*

Falls $X = Y$ ist, dann schreibt man für die Produktmenge $X \times X = X^2$.

Man beachte, dass bei Tupeln die Reihenfolge der Komponenten eine Rolle spielt (z.B. gilt $(1, 2) \neq (2, 1)$).

Beispiel 1.6. *Sei $A = \{a, b, c\}$ und $B = \{1, 2\}$. Dann ist*

$$A \times B = \{(a, 1), (b, 1), (c, 1), (a, 2), (b, 2), (c, 2)\}.$$

Definition 1.7. *Produktmengen können auch aus mehr als zwei Mengen gebildet werden und analog gilt*

$$X_1 \times X_2 \times \cdots \times X_n = \{(x_1, x_2, \ldots, x_n) \mid x_1 \in X_1, x_2 \in X_2, \ldots, x_n \in X_n\}.$$

Die Elemente dieser Produktemenge werden n-Tupel (oder nur Tupel) genannt. Zwei Tupel $(a_1, a_2, \ldots, a_n)$ und $(b_1, b_2, \ldots, b_n)$ sind gleich, wenn sie in allen Komponenten übereinstimmen, d.h., wenn für alle Indizes $k = 1, \ldots, n$ gilt, dass $a_k = b_k$.

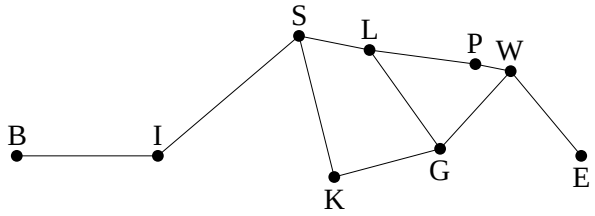
Wenn $X_1 = X_2 = \cdots = X_n = X$ gilt, dann wird die Produktmenge wieder kurz mit X^n bezeichnet.

2 Graphen und Bäume

Ein Graph ist eine Menge von Knoten und Kanten. Zunächst betrachten wir *ungerichtete* Graphen, d.h., Graphen in denen jede Kante eine Verbindung darstellt, die in beide Richtungen "befahren" werden kann.

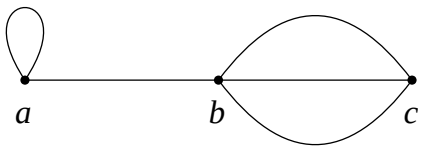
Beispiel 2.1. *(einfacher ungerichteter Graph - Bahnnetz) Knotenmenge sind die Landeshauptstädte, Kanten sind Zugverbindungen*

$$V = \{B, I, S, L, P, W, E, K, G\}$$
$$E = \{\{B, I\}, \{I, S\}, \{S, K\}, \{S, L\}, \{L, G\}, \{L, P\}, \{P, W\}, \{W, E\}, \{W, G\}, \{G, K\}\}$$



Definition 2.2. *Ein* einfacher ungerichteter Graph $G = (V, E)$ *ist ein Paar einer nichtleeren Menge V von* Knoten *(engl.* vertices*) und einer Menge E von* Kanten *(engl.* edges*), die aus zweielementigen Teilmengen $\{u, v\}$ von V besteht mit $u \neq v$. Die Kante $\{u, v\}$ wird auch als uv (oder vu) angeschrieben. Wenn $e = \{u, v\} \in E$, dann heissen die Knoten u, v* Endknoten *der Kante e.*

Im Gegensatz zu einfachen Graphen, gibt es auch *ungerichtete Multigraphen* in denen *Schlingen* und *Mehrfachkanten* vorkommen. Im Bild rechts gibt es eine Schlinge beim Knoten a und Mehrfachkanten zwischen b und c. Im folgenden werden wir nur einfache Graphen betrachten.



Definition 2.3. *Sei $G = (V, E)$ ein einfacher ungerichteter Graph. Zwei Knoten $u, v \in V$ heissen* benachbart *oder* adjazent*, wenn sie durch eine Kante verbunden sind, d.h., wenn $\{u, v\} \in E$. Zwei Kanten e_1 und e_2 des Graphen heissen* inzident *(aneinander angrenzend), wenn sie einen gemeinsamen Endknoten haben.*

Beispiel 2.4. *In Beispiel 2.1 sind zum Beispiel die Knoten B und I benachbart (adjazent), oder auch die Knoten I und S, nicht aber die Knoten B und S. Die Kanten $\{B, I\} = BI$ und $\{I, S\} = IS$ sind inzident.*

Definition 2.5. *Sei $G = (V, E)$ ein ungerichteter einfacher Graph und $v \in V$. Dann bezeichnen wir mit $E(v)$ die Menge aller benachbarten (adjazenten) Knoten, d.h.,*

$$E(v) = \{w \in V \mid \{v, w\} \in E\}.$$

Der Grad *$g(v)$ eines Knotens v ist die Anzahl aller benachbarter Knoten, d.h., $g(v) = |E(v)|$. Ein Knoten heisst* isoliert*, wenn $g(v) = 0$.*

Beispiel 2.6. *Für die Knoten aus Beispiel 2.1 gilt:*

$$\begin{aligned} E(B) &= \{I\}, & E(I) &= \{B, S\}, & E(S) &= \{I, K, L\}, \\ E(L) &= \{S, G, P\}, & E(P) &= \{L, W\}, & E(W) &= \{P, G, E\}, \\ E(E) &= \{W\}, & E(G) &= \{K, L, W\}, & E(K) &= \{S, G\}, \end{aligned}$$

und folglich,

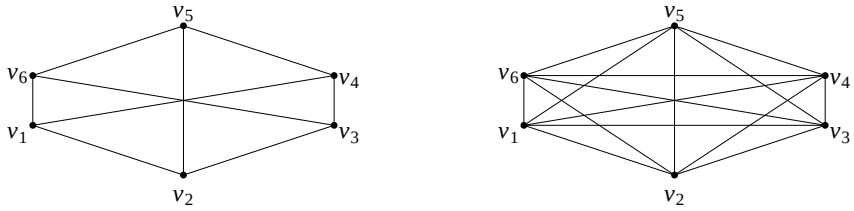
$$\begin{aligned} g(B) &= 1, & g(I) &= 2, & g(S) &= 3, \\ g(L) &= 3, & g(P) &= 2, & g(W) &= 3, \\ g(E) &= 1, & g(G) &= 3, & g(K) &= 2. \end{aligned}$$

In diesem Graphen gibt es keinen isolierten Knoten.

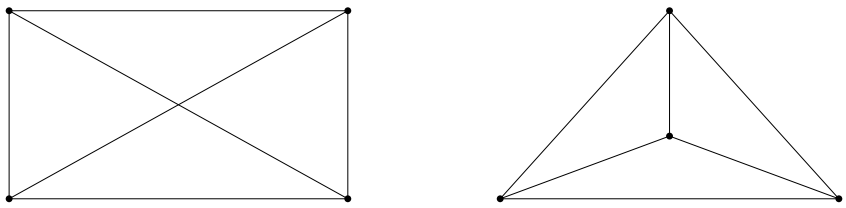
Beispiel 2.7. *Gegeben ist der ungerichtete Graph $G = (V, E)$ mit $V = \{a, b, c, d, e\}$ und $E = \{\{b, c\}, \{b, d\}, \{c, d\}, \{d, e\}\}$. Skizzieren Sie den Graphen und bestimmen die Grade aller Knoten.*

Definition 2.8. *Sei $G = (V, E)$ ein einfacher, ungerichteter Graph, dann heisst G* regulär*, wenn alle Knoten denselben Grad d haben (auch d-regulär). Sei $|V| = n$, dann heisst G* vollständiger Graph*, wenn jeder Knoten mit allen anderen Knoten durch eine Kante verbunden ist.*

Beispiel 2.9. *Sei $V = \{v_1, v_2, \ldots, v_6\}$. Links sehen wir einen 3-regulären Graphen über dieser Knotenmenge und rechts den vollständigen Graphen über V. In diesen Graphen sieht man, dass sich Kanten auch kreuzen können. Kreuzungspunkte von Kanten sind aber nicht automatisch Knoten! Zum Beispiel im linken Graphen ist der Kreuzungspunkt in der Mitte* kein *Knoten.*



Graphen, die kreuzungsfrei gezeichnet werden können heissen planare Graphen*. Der vollständige Graph vierter Ordnung kann zum Beispiel mit überkreuzten Kanten, aber auch planar (kreuzungsfrei) gezeichnet werden:*



Lemma 2.10. *(Handschlaglemma) Sei $G = (V, E)$ ein einfacher ungerichteter Graph. Dann ist die Summe aller Grade $\sum_{v \in V} g(v)$ gerade.*

Beweis. In einem ungerichteten Graphen, trägt jede Kante genau zweimal zur Summe bei. Damit ist die Summe der Grade gerade und es gilt $\sum_{v \in V} g(v) = 2|E|$. □

Lemma 2.11. *Sei $G = (V, E)$ ein einfacher ungerichteter Graph. Die Anzahl der Knoten in G mit ungeradem Grad ist gerade.*

Beweis. Sei $U \subseteq V$ die Menge der Knoten mit ungeradem Grad. Dann gilt mit dem Handschlaglemma:

$$\sum_{v \in U} g(v) + \sum_{v \in V \setminus U} g(v) = 2|E|.$$

Für $v \in V \setminus U$ ist $g(v)$ gerade, damit ist auch die Summe über $v \in V \setminus U$ gerade. Bezeichnen wir diese Summe mit $2M$. Für $v \in U$ gilt, dass $g(v) = 2k_v + 1$ für ein $k_v \in \mathbb{N}$. Wenn wir das oben einsetzen erhalten wir

$$\sum_{v \in U} (2k_v + 1) = 2(|E| - M).$$

Die Summe auf der linken Seite lässt sich aufspalten in

$$\sum_{v \in U} (2k_v + 1) = 2 \sum_{v \in U} k_v + \sum_{v \in U} 1 = 2 \sum_{v \in U} k_v + |U|.$$

Zusammengefasst liefert das

$$|U| = 2 \left(|E| - M - \sum_{v \in U} k_v \right),$$

das heisst die Anzahl der Knoten mit ungeradem Grad ist gerade. □

Beispiel 2.12. *Ist es möglich in einer Gruppe von 13 Personen, dass jede mit (a) genau 6 anderen (b) genau 5 anderen (c) genau 1 anderen (d) allen anderen mit je genau einer Ausnahme bekannt ist?*

Satz 2.13. *Sei $G = (V, E)$ ein einfacher ungerichteter Graph. Dann gibt es (mindestens) zwei Knoten vom gleichen Grad.*

Beweis. Sei $|V| = n$. Für alle $v \in V$ gilt $0 \leq g(v) \leq n-1$. Falls es Knoten $\tilde{v}$ mit $g(\tilde{v}) = 0$ gibt, dann kann es keinen Knoten vom Grad $n-1$ geben. Umgekehrt, falls es einen Knoten vom Grad $n-1$ gibt, dann existiert kein Knoten vom Grad 0. D.h., es gilt sogar $0 \leq g(v) \leq n-2$ oder $1 \leq g(v) \leq n-1$ für alle $v \in V$. Damit gibt es nur $n-1$ mögliche Grade für n Knoten, also müssen mindestens zwei Knoten den gleichen Grad haben. □

Definition 2.14. *Sei $G = (V, E)$ ein Graph und seien $V' \subseteq V$ und $E' \subseteq E$ so dass alle Endknoten von Kanten in E' in V' liegen. Dann ist $G' = (V', E')$ ein* Teilgraph *von G und wir schreiben auch $G' \subseteq G$. Der Teilgraph*

$$G[V'] = (V', \{\{u, v\} \mid \{u, v\} \in E \wedge u, v \in V'\})$$

heisst der von V' knoteninduzierte *Teilgraph. Der Teilgraph*

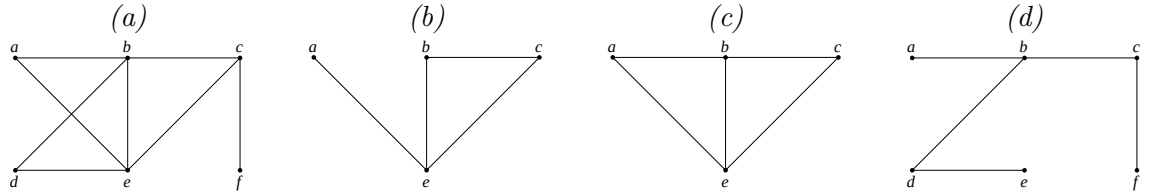
$$G[E'] = (\{v \in V \mid v \text{ ist Endknoten einer Kante in } E'\}, E')$$

heisst der von E' kanteninduzierte *Teilgraph. Ein Teilgraph heisst* aufspannend*, falls er alle Knoten von G enthält.*

Beispiel 2.15. *Sei $G = (V, E)$ mit Knoten $V = \{a, b, c, d, e, f\}$ und Kanten*

$$E = \{\{a, b\}, \{a, e\}, \{b, c\}, \{b, d\}, \{b, e\}, \{c, e\}, \{c, f\}, \{d, e\}\},$$

wie unter (a) abgebildet, unter (b) sieht man einen Teilgraphen von G, unter (c) den von $V' = \{a, b, c, e\}$ knoteninduzierten Teilgraphen und unter (d) einen aufspannenden Teilgraphen.



Definition 2.16. *Sei $G = (V, E)$ ein einfacher ungerichteter Graph und $v_1, v_n \in V$. Eine Folge $W(v_1, v_n) = (v_1, v_2, \ldots, v_{n-1}, v_n)$ mit $\{v_j, v_{j+1}\} \in E$ für $j = 1, \ldots, n-1$ heisst ein* Weg *von v_1 nach v_n in G. Einen Weg, bei dem alle vorkommenden Knoten verschieden sind nennen wir einen* einfachen Weg.

Die Schreibweise, die wir für Wege verwenden unterstreicht, dass die Reihenfolge der Knoten wesentlich ist.

Bemerkung 2.17. *Achtung: Die Verwendung des Begriffs* Weg *ist in der Literatur nicht einheitlich. Es werden u.a. auch die Begriffe* Kantenzug *oder* Spaziergang *verwendet für verschiedene Einschränkungen (unterschiedliche Knoten, unterschiedliche Kanten,...).*

Beispiel 2.18. *Sei $G = (V, E)$ der Graph aus Beispiel 2.15. Dann ist (a, b, d, e, b, c) ein Weg von a nach c in G, aber kein einfacher Weg, da der Knoten b zweimal besucht wird. Die Wege (a, b, c) oder (a, e, c) sind zwei einfache Wege von a nach c in G.*

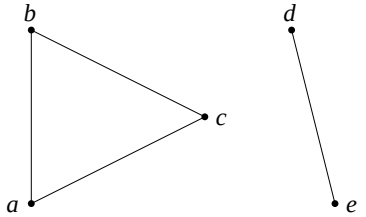
Definition 2.19. *Sei $G = (V, E)$ ein einfacher, ungerichteter Graph. Zwei Knoten $v_1, v_2 \in V$ heissen* verbindbar *in G, wenn es einen Weg $W(v_1, v_2)$ in G gibt.*

G heisst zusammenhängend*, wenn je zwei unterschiedliche Knoten stets durch einen Weg verbunden werden können.*

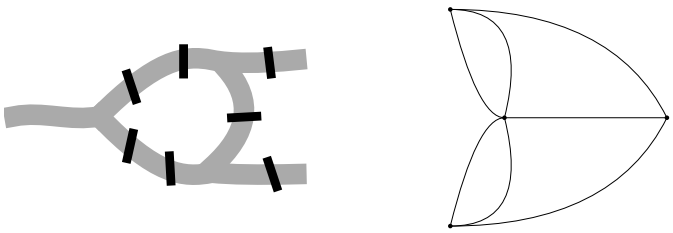
Ein geschlossener Weg in G, d.h., ein Weg bei dem der Ausgangs- und Endknoten gleich sind, heisst Kreis *(oder* Zyklus*). Wir bezeichnen einen Kreis als* einfach*, wenn alle vorkommenden Kanten nur einmal verwendet werden.*

Beispiel 2.20. *Der Graph G aus Beispiel 2.15 ist zusammenhängend. Der Weg (a, b, e, a) in G ist ein Kreis, ebenso der Weg (a, e, c, b, a).*

Beispiel 2.21. *Seien $V = \{a, b, c, d, e\}$ und $E = \{\{a, b\}, \{a, c\}, \{b, c\}, \{d, e\}\}$. Dieser Graph ist nicht zusammenhängend (es führt z.B. kein Weg von a nach d), aber er enthält einen Kreise (z.B. (a, b, c, a) oder (e, d, e), wobei letzterer kein einfacher Kreis ist).*



Beispiel 2.22. *(Königsberger Brückenproblem) Euler klärte 1736 die Frage, ob es in der Stadt Königsberg einen Weg gibt, der jede der sieben Brücken, die über die Pregel führten* genau einmal *quert und wieder zum Ausgangspunkt zurückkehrt. Euler stellte die verschiedenen Ufer durch Knoten und die Brücken durch Kanten dar und bestimmte Kriterien für die Existenz eines "Eulerschen Kreises".*



Definition 2.23. *Sei $G = (V, E)$ ein einfacher, ungerichteter, zusammenhängender Graph. Ein Weg in G, der jede Kante in E* genau einmal *durchläuft, heisst* Eulerscher Weg. *Ist dieser Weg ausserdem ein Kreis, so heisst er* Eulerscher Kreis. *Ein Graph, der einen Eulerschen Kreis enthält, wird* Eulersch *genannt.*

Satz 2.24. *Ein ungerichteter, zusammenhängender Graph ist genau dann Eulersch, wenn alle Knoten geraden Grad haben.*

Beweis. Angenommen, der gegebene Graph enthält einen Eulerschen Kreis. Da der Graph zusammenhängend ist, wird jeder Knoten mindestens einmal durchlaufen. Da jede Kante nur einmal verwendet wird, wird bei jedem Durchlauf eines Knoten der Kantenzähler um zwei erhöht, damit ist der Grad jedes Knoten gerade.

Sei umgekehrt ein zusammenhängender Graph gegeben, in dem jeder Knoten geraden Grad hat. Man wähle einen beliebigen Knoten aus und wähle eine Kante. Am benachbarten Knoten muss wieder eine noch unbenutzte Kante existieren, da alle Knoten geraden Grad haben. Der Prozess wird fortgesetzt, bis man wieder den ersten Knoten erreicht (der noch mindestens eine unbenutzte Kante besitzen muss, wegen der Grad-Bedingung und erreichbar sein muss da der Graph zusammenhängend ist). Wenn man zu diesem Zeitpunkt bereits einen Eulerschen Kreis gefunden hat, ist man fertig. Sonst, wähle einen Knoten, der noch Teil unbenutzter Kanten ist und starte den Vorgang erneut von diesem Knoten aus, bis ein weiterer Kreis entstanden ist. Nach einer endlichen Zahl von Wiederholungen erhält man so eine endliche Menge von Kreisen. Da der Graph zusammenhängend ist, müssen diese Kreise zu einem Kreis zusammengefügt werden können, in dem jede Kante nur einmal durchlaufen wird. □

Mit diesem Satz folgt, dass der Graph aus Beispiel 2.22 nicht Eulersch ist. Der Beweis gibt einen Algorithmus zur Bestimmung eines Eulerschen Kreises, sofern einer existiert, vor.

Beispiel 2.25. *Sei $G = (V, E)$ der ungerichtete Graph mit $V = \{A, B, C, D, E\}$ und Kanten $E = \{(A, B), (A, C), (B, C), (B, D), (B, E), (C, D), (C, E), (D, E)\}$. Gibt es einen Eulerschen Weg im Graphen? Einen Eulerschen Kreis?*

Beispiel 2.26. *Sei $G = (V, E)$ der ungerichtete Graph mit $V = \{A, B, C, D, E, F, G, H, I\}$ und Kanten*

$$E = \{(A, B), (A, C), (B, D), (C, D), (D, E), (D, I), (E, F), (G, H), (H, I), (G, I)\}.$$

Gibt es einen Eulerschen Weg im Graphen? Einen Eulerschen Kreis?

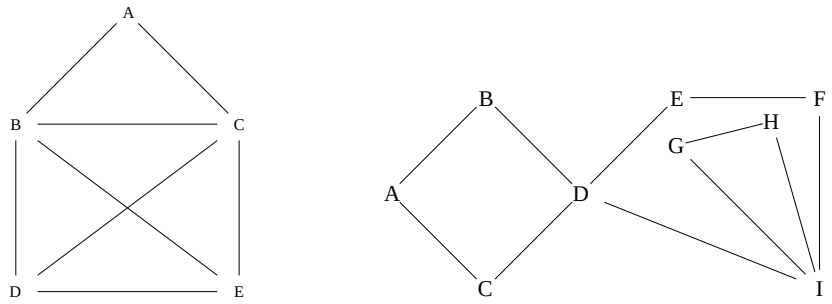


Abbildung 2.1: Graphen zu Beispiel 2.25 (links) und zu Beispiel 2.26 (rechts)

Mit Hilfe von Satz 2.24 lässt sich feststellen, ob die Graphen in Beispiel 2.25 und 2.26 einen Eulerschen Kreis beinhalten. Folgt man dem Beweis von Satz 2.24 dann kann man eine Methode ableiten um den Eulerschen Kreis im Graphen von Beispiel 2.24 (hier besitzen alle Knoten geraden Grad) zu bestimmen:

Wir wählen eine beliebigen Startknoten, z.B. A und folgen immer einer noch unbenutzten Kante solange bis wir wieder den Knoten A erreichen, zum Beispiel durch den Weg

$$W_1(A, A) = (A, B, D, C, A).$$

Dann wählen wir einen Knoten, der noch (mindestens) eine unbenutzte Kante hat, z.B. den Knoten D und folgen wieder immer einer noch unbenutzten Kante, solange bis wieder der Knoten D erreicht ist, z.B. durch den Weg

$$W_2(D, D) = (D, E, F, I, D).$$

Es gibt immer noch unbenutzte Kanten: wir wählen z.B. den Knoten G und erhalten zum Beispiel

$$W_3 = (G, H, I, G).$$

Nachdem alle Kanten verwendet wurden werden die drei gefunden Kreise an den Schnittpunkten zusammengehängt zu einem Eulerschen Kreise, zum Beispiel:

$$(A, B, D, E, F, I, G, H, I, D, C, A).$$

Der Beweis von Satz 2.24 kann variiert werden um ein Kriterium für die Existenz eines Eulerschen Wegs zu erhalten.

Lemma 2.27. *Ein ungerichteter, zusammenhängender Graph enthält genau dann einen Eulerschen Weg, wenn es genau zwei Knoten mit ungeradem Grad gibt und alle anderen Knoten geraden Grad haben. Die beiden Knoten mit ungeradem Grad sind dann der Ausgangs- und der Endknoten des Eulerschen Wegs.*

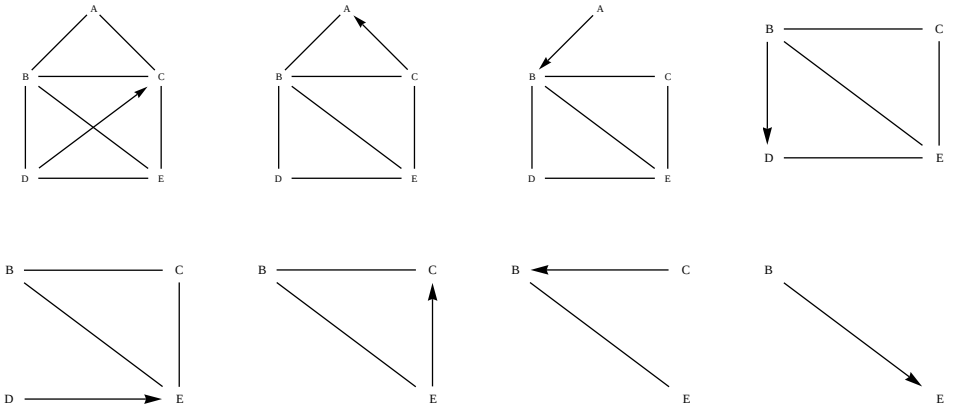
Mit diesem Kriterium folgt, dass der Graph aus Beispiel 2.25 einen Eulerschen Weg enthält. Der Algorithmus von *Fleury* gibt eine Methode um einen Eulerschen Weg zu bestimmen:

1. Startpunkt ist ein Knoten mit ungeradem Grad
2. Gehe entlang einer Kante und entferne diese Kante; falls der passierte Knoten durch die Kantenentfernung isoliert ist, wird der Knoten entfernt.

Schritt 2 wird so oft wie möglich wiederholt, wobei dieser Schritt so auszuführen ist, dass der verbleibende Teilgraph *zusammenhängend* bleibt. Bei dem Graphen aus Beispiel 2.25 kann ausgehend vom Startknoten D zum Beispiel wie im Bild der Eulersche Weg

$$W(D,E) = (D, C, A, B, D, E, C, B, E)$$

mit dieser Methode gefunden werden. Die Pfeile zeigen dabei jeweils die aktuell "befahrene" Kante an.



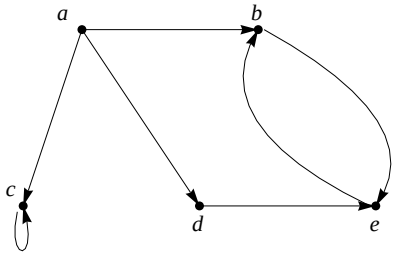
Definition 2.28. *Sei G ein endlicher, zusammenhängender Graph. Ein Weg in G heisst* Hamiltonsch*, falls jeder Knoten in G* genau einmal *durchlaufen wird. Ist dieser Weg ausserdem ein Kreis, so nennt man ihn einen* Hamiltonschen Kreis. *Einen Graphen nennt man* Hamiltonsch*, falls er einen Hamiltonschen Kreis enthält.*

Ein *gerichteter* Graph besteht aus einer Menge von Knoten und einer Menge von Kanten zwischen den gegebenen Knoten, wobei die Kanten eine *ausgezeichnete* Richtung besitzen. Die Kanten werden in diesem Fall nicht als Mengen sondern als *Tupel* (geordnete Paare) angegeben.

Definition 2.29. *Ein* gerichteter Graph $G = (V, E)$ *ist ein Paar einer nichtleeren Menge V von* Knoten *(engl.* vertices*) und einer Menge $E \subseteq V \times V$ von* Kanten *(engl.* edges*).*

Beispiel 2.30. *(gerichteter Graph)*

$V = \{a, b, c, d, e\}$
$E = \{(a,b), (a,c), (a,d), (b,e), (d,e), (e,b), (c,c)\}$



Dieser Graph $G = (V, E)$ enthält eine Schlinge, nämlich die Kante (c, c).

Definition 2.31. *Sei $G = (V, E)$ ein gerichteter Graph und sei $v \in V$, dann bezeichnen wir mit $E(v)$ die Menge der* Ausgangsknoten *von v und mit $E^{-1}(v)$ die Menge der* Eingangsknoten *von v. Das heisst*

$$E(v) = \{w \in V \mid (v,w) \in E\} \quad \textit{und} \quad E^{-1}(v) = \{u \in V \mid (u,v) \in E\}.$$

Ist $E(v)$ eine endliche Menge, dann ist $|E(v)|$ der Ausgangsgrad *von v; ist $E^{-1}(v)$ eine endliche Menge, dann ist $|E^{-1}(v)|$ der* Eingangsgrad *von v. Die Summe aus Eingangsgrad und*

Ausgangsgrad ist der Grad $g(v)$ *des Knoten* v*, d.h.* $g(v) = |E(v)| + |E^{-1}(v)|$*. Ein Knoten* v *heisst* isoliert*, wenn* $g(v) = 0$ *ist. Ein Knoten* v *mit* $|E(v)| > 0$ *und* $|E^{-1}(v)| = 0$ *heisst* Quelle*. Ein Knoten* v *mit* $|E(v)| = 0$ *und* $|E^{-1}(v)| > 0$ *heisst* Senke*.*

Beispiel 2.32. *Für den Graphen* $G = (V, E)$ *aus Beispiel 2.30 gilt:*

$$\begin{array}{lllll} E(a) = \{b, c, d\}, & E(b) = \{e\}, & E(c) = \{c\}, & E(d) = \{e\}, & E(e) = \{b\}, \\ E^{-1}(a) = \{\,\}, & E^{-1}(b) = \{a, e\}, & E^{-1}(c) = \{a, c\}, & E^{-1}(d) = \{a\}, & E^{-1}(e) = \{b, d\}, \\ g(a) = 3, & g(b) = 3, & g(c) = 3, & g(d) = 2, & g(e) = 3. \end{array}$$

Der Knoten a *ist eine Quelle.*

Lemma 2.33. *(Handschlaglemma) Sei* $G = (V, E)$ *ein gerichteter Graph. Dann ist die Summe aller Grade* $\sum_{v \in V} g(v)$ *gerade.*

Beweis. In einem gerichteten Graphen gilt

$$\sum_{v \in V} g(v) = \sum_{v \in V} \big(|E(v)| + |E^{-1}(v)|\big).$$

d.h. auch hier trägt jede Kante zweimal zur Summe bei. Auch im Fall eines ungerichteten Graphen gilt damit dass $\sum_{v \in V} g(v) = 2|E|$. □

Der Beweis des nächsten Satzes kann direkt von den entsprechenden Aussagen für ungerichtete Graphen übernommen werden.

Lemma 2.34. *Sei* $G = (V, E)$ *ein gerichteter, endlicher Graph. Die Anzahl der Knoten in* G *mit ungeradem Grad ist gerade. Ausserdem gilt, dass es (mindestens) zwei Knoten vom gleichen Grad gibt.*

Definition 2.35. *Sei* $G = (V, E)$ *ein gerichteter Graph:*

- *Zwei Kanten* $e_1, e_2 \in E$ *heissen* parallel *(oder* Mehrfachkanten*), wenn* $e_1 = (v_1, v_2) = e_2$*.*
- *Eine Kante* (v, v)*, mit* $v \in V$*, heisst* Schlinge*.*
- G *heisst* einfach *oder* schlicht*, wenn er keinen Schlingen und keine Parallelen enthält.*
- G *heisst* regulär*, wenn* G *einfach ist und alle Knoten den selben Eingangs- und Ausgangsgrad haben.*
- *Sei* $|V| = n$*. Dann heisst* G vollständiger Graph der Ordnung n*, wenn* G *ein einfacher Graph ist, der alle möglichen Kanten ohne Schlingen enthält, d.h.,* $E = V \times V \backslash \{(v, v) \mid v \in V\}$*.*

Definition 2.36. *Sei* $G = (V, E)$ *ein gerichteter Graph und seien* $V' \subseteq V$ *und* $E' \subseteq E \cap \big(V' \times V'\big)$*. Dann ist* $G' = (V', E')$ *ein* Teilgraph *von* G *und wir schreiben auch* $G' \subseteq G$*. Der Teilgraph*

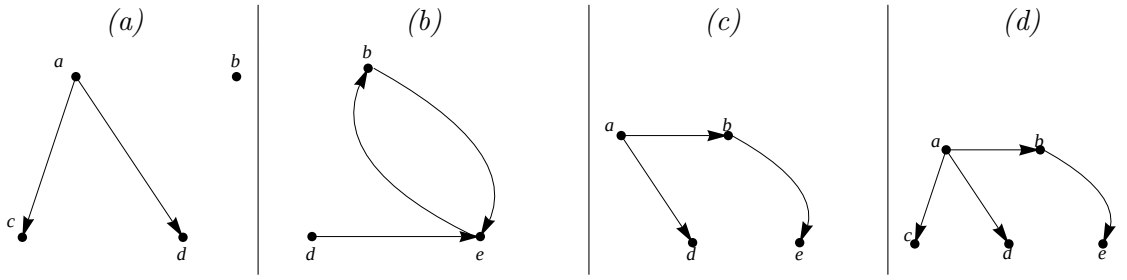
$$G[V'] = (V', E \cap V' \times V')$$

heisst der von V' knoteninduzierte *Teilgraph. Der Teilgraph*

$$G[E'] = (\{v \in V \mid \exists\, w \in V :\ (v, w) \in E' \vee (w, v) \in E'\}, E')$$

heisst der von E' kanteninduzierte *Teilgraph. Ein Teilgraph heisst* aufspannend*, falls er alle Knoten von* G *enthält.*

Beispiel 2.37. *Wir betrachten wieder den Graphen aus Beispiel 2.30. Unter (a) sehen wir einen Teilgraphen mit* $V_1 = \{a, b, c, d\}$ *und* $E_1 = \{(a, c), (a, d)\}$*, unter (b) den von* $V_2 = \{b, d, e\}$ *knoteninduzierten Teilgraphen* $G[V_2]$ *mit der Kantenmenge* $E_2 = \{(d, e), (b, e), (e, b)\}$*, unter (c) den von* $E_3 = \{(a, b), (a, d), (b, e)\}$ *kanteninduzierten Teilgraphen* $G[E_3]$ *mit der Knotenmenge* $V_3 = \{a, b, d, e\}$ *und unter (d) einen aufspannenden Teilgraphen.*



Definition 2.38. *Sei* $G = (V, E)$ *ein gerichteter Graph.*

- *Eine Folge* $W(v_1, v_n) = (v_1, v_2, \ldots, v_n)$ *mit* $(v_j, v_{j+1}) \in E$ *für* $j = 1, \ldots, n-1$ *heisst* (gerichteter) Weg *von* v_1 *nach* v_n *in* G*. Ein (gerichteter) Weg heisst* einfach*, wenn alle vorkommenden Knoten verschieden sind.*
- *Zwei Knoten* $v_1, v_2 \in V$ *heissen* (gerichtet) verbindbar *in* G*, wenn es einen Weg* $W(v_1, v_2)$ *in* G *gibt.*
- *Ein Weg* $W(v_1, v_n)$ *heisst* Zyklus *oder* Kreis*, wenn* $n > 1$ *und* $v_1 = v_n$*. Ein Kreis heisst* einfach*, wenn alle vorkommenden Kanten nur einmal verwendet werden.*

Beispiel 2.39. *Sei* G *der Graph aus Beispiel 2.30.* $W_1(a, b) = (a, b)$ *und* $W_2(a, b) = (a, d, e, b)$ *sind zwei (gerichtete) Wege in* G *von* a *nach* b*. Beide Wege sind einfache Wege. Die Knoten* a *und* b *sind damit (gerichtet) verbindbar. Der Weg* $W(b, e) = (b, e, b)$ *ist ein (einfacher) Kreis.*

Definition 2.40. *Sei* $G = (V, E)$ *ein gerichteter Graph.* G *heisst* stark zusammenhängend von $v \in V$ aus*, wenn es zu jedem anderen Knoten* $w \in V$ *(d.h.,* $w \neq v$*) einen (gerichteten) Weg* $W(v, w)$ *im Graphen gibt.*

Der Graph heisst stark zusammenhängend*, wenn er von jedem Knoten* $v \in V$ *aus stark zusammenhängend ist.*

Der Graph heisst (schwach) zusammenhängend*, wenn der ungerichtete Graph, der entsteht, wenn jede gerichtete Kante durch eine ungerichtete Kante ersetzt wird, zusammenhängend ist.*

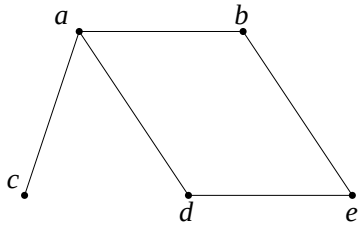
Beispiel 2.41. *Sei* G *der Graph aus Beispiel 2.30. Vom Knoten* a *aus sind alle anderen Knoten in* G *durch Wege erreichbar, zum Beispiel durch folgende Wege:*

$$W(a, b) = (a, b), \quad W(a, c) = (a, c), \quad W(a, d) = (a, d), \quad W(a, e) = (a, d, e).$$

Der Graph ist damit stark zusammenhängend von a *aus. Der Graph ist aber nicht stark zusammenhängend als solcher, da zum Beispiel vom Knoten* d *aus nur die Knoten* b *und* e*, nicht aber die Knoten* a *und* c *erreichbar sind. Der Graph ist schwach zusammenhängend, da der ungerichtete Graph* $G_1 = (V_1, E_1)$ *mit*

$$V_1 = \{a, b, c, d, e\} \quad \textit{und} \quad E_1 = \{\{a, b\}, \{a, c\}, \{a, d\}, \{b, e\}, \{d, e\}\}$$

zusammenhängend ist (die gerichteten Kanten (b, e) *und* (e, b) *fallen zusammen, die Schlinge bei* c *können wir vernachlässigen).*



Definition 2.42. *Sei $G = (V, E)$ ein einfacher, ungerichteter Graph und $V' \subseteq V$. Der knoteninduzierte Graph $G[V']$ heisst* Zusammenhangskomponente *von G, falls $G[V']$ zusammenhängend ist und keine Teilmenge $W \subseteq V$ existiert mit $V' \subsetneq W$ und $G[W]$ ist zusammenhängend.*

Ein isolierter Knoten $v \in V$ bildet eine eigene Zusammenhangskomponente mit leerer Kantenmenge.

Sei $v \in V'$ ein beliebiger Knoten aus der Knotenmenge der Zusammenhangskomponente, dann werden wir die Zusammenhangskomponente, die v enthält auch mit $[v] = G[V']$ anschreiben und v als Repräsentanten *der Zusammenhangskomponente bezeichnen.*

Zusammenhangskomponenten sind also die grösstmöglichen zusammenhängenden Teilgraphen. Die Wahl des Repräsentanten der Zusammenhangskomponente ist beliebig.

Definition 2.43. *Sei $G = (V, E)$ ein gerichteter Graph und $V' \subseteq V$. Der knoteninduzierte Graph $G[V']$ heisst* (starke) Zusammenhangskomponente *von G, falls $G[V']$ stark zusammenhängend ist und keine Teilmenge $W \subseteq V$ existiert mit $V' \subsetneq W$ und $G[W]$ ist stark zusammenhängend.*

Isolierte Knoten, Quellen sowie Senken bilden jeweils eigene Zusammenhangskomponenten mit leeren Kantenmengen.

Sei $v \in V'$ ein beliebiger Knoten aus der Knotenmenge der starken Zusammenhangskomponente, dann werden wir die Zusammenhangskomponente, die v enthält auch mit $[v] = G[V']$ anschreiben und v als Repräsentanten *der Zusammenhangskomponente bezeichnen.*

Zusammenhangskomponenten sind auch bei gerichteten Graphen die grösstmöglichen zusammenhängenden Teilgraphen und auch hier ist die Wahl des Repräsentanten der Zusammenhangskomponente beliebig.

Beispiel 2.44. *Sei $G = (V, E)$ der gerichtete Graph mit Knotenmenge $V = \{v_1, v_2, v_3, v_4, v_5\}$ und Kantenmenge*

$$E = \{(v_1, v_3), (v_1, v_4), (v_2, v_1), (v_2, v_3), (v_3, v_4), (v_3, v_5), (v_4, v_5), (v_5, v_2)\}.$$

Dieser Graph enthält (zum Beispiel) den Kreis $W(v_1, v_1) = (v_1, v_3, v_4, v_5, v_2, v_1)$. Durch Teilwege ist damit jeder Knoten mit jedem Knoten verbunden und der Graph ist stark zusammenhängend. Er enthält eine Zusammenhangskomponente $[v_1] = G$.

Beispiel 2.45. *Sei $G = (V, E)$ der gerichtete Graph mit Knotenmenge $V = \{a, b, x, y, z\}$ und Kantenmenge $E = \{xy, yz, zx, za, ab, ba\}$. Dieser Graph ist stark zusammenhängend von den Knoten x, y, z aus. Der Graph ist schwach zusammenhängend und hat zwei Zusammenhangskomponenten, die von den Knotenmengen $V_1 = \{x, y, z\}$ und $V_2 = \{a, b\}$ induziert werden.*

Definition 2.46. *Eine* Partition *einer Menge M ist eine Familie von Mengen $\{P_i \mid i \in \mathcal{I}\}$ ($\mathcal{I}$ bezeichnet eine Indexmenge) sodass*

1. *Keine der Mengen P_i ist leer.*

2. *Jedes Element der Menge M liegt in* genau einer *Menge der Partition.*

Die Vereinigung aller Mengen der Partition ergibt also die Menge M. Die Knotenmengen der Zusammenhangskomponenten eines (gerichteten oder ungerichteten) Graphen bilden eine Partition der Knotenmenge des Graphen.

Sei $G = (V, E)$ ein ungerichteter Graph mit Knotenmenge $V = \{v_1, v_2, \ldots, v_n\}$ und Zusammenhangskomponenten $[v_1], \ldots, [v_k]$ (mit $1 \leq k \leq n$).

Jeder Knoten muss in einer Zusammenhangskomponente liegen (entweder bildet er eine eigene Zusammenhangskomponente, wenn er isoliert ist, oder er ist mit (mindestens) einem anderen Knoten verbunden und liegt in der entsprechenden Zusammenhangskomponente).

Angenommen ein Knoten $w \in V$ liegt in zwei unterschiedlichen Zusammenhangskomponenten $[u_1] = (U_1, E_1)$ und $[u_2] = (U_2, E_2)$. Nach Definition von Zusammenhang gibt es im Graphen G Wege von w zu jedem Knoten in U_1 und Wege von w zu jedem Knoten in U_2. Damit können die beiden Komponenten $[u_1]$ und $[u_2]$ über den Knoten w verbunden werden, d.h., keine der beiden ist maximal, was ein Widerspruch zur Definition von Zusammenhangskomponente ist. Die Wahl des Repräsentanten ist beliebig, da von jedem Knoten in der Zusammenhangskomponente aus jeder andere Knoten erreichbar ist. Analog verhält es sich bei gerichteten Graphen.

Definition 2.47. *Seien X, Y Mengen. Eine* Funktion $f\colon X \to Y$ *ist eine Vorschrift, die* jedem $x \in X$ genau ein $y \in Y$ *zuordnet. Das zu $x \in X$ eindeutig bestimmte $y \in Y$ wird mit $y = f(x)$ bezeichnet. Wir schreiben auch $f\colon X \to Y, x \mapsto f(x)$ oder betrachten f als eine Teilmenge der Produktmenge $X \times Y$, d.h., $f \subseteq X \times Y$.*

Beispiel 2.48. *Sei $X = \{1, 2, 3, 4, 5, 6\}$ und $Y = \{a, b, c, d, e, f\}$.*

- *$g_1 = \{(1, a), (2, f), (3, c), (4, d), (5, b), (6, e)\} \subseteq X \times Y$ ist eine Funktion, da jedem Element $x \in X$ genau ein Element $y \in Y$ zugewiesen wird. In anderen Schreibweisen kann die Funktion $g_1\colon X \to Y$ auch so angegeben werden:*

  $$g_1(1) = a,\ g_1(2) = f,\ g_1(3) = c,\ g_1(4) = d,\ g_1(5) = b,\ g_1(6) = e,$$

  oder

  $$1 \mapsto a,\ 2 \mapsto f,\ 3 \mapsto c,\ 4 \mapsto d,\ 5 \mapsto b,\ 6 \mapsto e.$$

- *$g_2 = \{(1, a), (2, f), (3, d), (4, a), (5, c), (6, c)\} \subseteq X \times Y$ ist ebenfalls eine Funktion, da jedem Element $x \in X$ genau ein Element $y \in Y$ zugewiesen wird.*

- *$g_3 = \{(1, a), (2, f), (3, d), (4, c), (5, e)\} \subseteq X \times Y$ ist keine Funktion, da der Zahl 6 kein Wert zugewiesen wird.*

- *$g_4 = \{(1, a), (2, b), (3, c), (4, d), (5, e), (6, f), (1, f)\} \subseteq X \times Y$ ist keine Funktion, da der Zahl 1 zwei* unterschiedliche *Werte zugewiesen werden.*

Beispiel 2.49. *Sei $X = \mathbb{N}$ und $Y = \mathbb{N}$. Die Abbildung $g_5\colon \mathbb{N} \to \mathbb{N}, x \mapsto 2x + 1$ ist eine Funktion. Jeder natürlichen Zahl $x \in \mathbb{N}$ wird genau eine natürliche Zahl zugewiesen. Eine andere Art die Funktion anzuschreiben ist als Teilmenge der Produktmenge:*

$$g_5 = \{(x, 2x + 1) \mid x \in \mathbb{N}\} \subseteq X \times Y = \mathbb{N} \times \mathbb{N} = \mathbb{N}^2.$$

Definition 2.50. *Sei $f \subseteq X \times Y$ eine Funktion. Die Menge X heisst der* Definitionsbereich *(engl.:* domain*) von f. Die Menge aller $y \in Y$, für die ein $x \in X$ existiert sodass $y = f(x)$, d.h.,*

$$f(X) = \{y \in Y \mid \exists\, x \in X \colon f(x) = y\} \subseteq Y$$

heisst das Bild *oder der* Bildbereich *oder der* Wertebereich *(engl.:* range *oder* image*) von f unter X. Ist $y \in f(X)$, dann wird ein $x \in X$ mit $f(x) = y$ als* Urbild *bezeichnet.*

Beispiel 2.51. *Für die Funktionen g_1 und g_2 aus Beispiel 2.48 gilt $g_1(X) = \{a, b, c, d, e, f\} = Y$ und $g_2(X) = \{a, c, d, f\} \subsetneq Y$.*

Seien $X_1 = \{1, 2, 3, 4\}$, $Y_1 = \{a, b, c\}$ und $f_1 = \{(1, a), (2, b), (3, c), (4, b)\}$. Diese Abbildung ist eine Funktion (jedem $x \in X_1$ wird genau ein $y \in Y_1$ zugeordnet) und der Wertebereich ist $f_1(X_1) = \{a, b, c\} = Y_1$. Jeder Wert in Y_1 wird mindestens einmal durch die Abbildung f_1 erreicht. Das Urbild von a unter f_1 ist 1. Urbilder von b unter f_1 sind 1 und 4. Das Urbild von c unter f_1 ist 3.

Seien $X_2 = \{1, 2, 3\}$, $Y_2 = \{a, b, c, d\}$ und $f_2 = \{(1, a), (2, c), (3, d)\}$. Diese Abbildung ist ebenfalls eine Funktion und es gilt $f_2(X_2) = \{a, c, d\} \subsetneq Y_2$. Jeder Wert in Y_2 wird höchstens einmal von der Abbildung f_2 erreicht. Das Urbild von a unter f_2 ist 1, das Urbild von c unter f_2 ist 2, das Urbild von d unter f_2 ist 3 und b besitzt kein Urbild unter f_2.

Die Funktion g_5 aus Beispiel 2.49 bildet die Menge der natürlichen Zahlen auf die Menge der ungeraden natürlichen Zahlen ab, d.h., nicht jedes $y \in Y = \mathbb{N}$ wird von der Funktion erreicht.

Definition 2.52. *Sei $f \subseteq X \times Y$ eine Funktion.*

(1) f heisst surjektiv*, wenn jedes $y \in Y$ im Bild von f liegt (d.h., wenn $Y = f(X)$).*

(2) f heisst injektiv*, wenn jedes Element im Bild von f* genau ein *Urbild in X besitzt.*

(3) f heisst bijektiv*, wenn f sowohl injektiv als auch surjektiv ist.*

Beispiel 2.53. *Die Funktion g_1 aus Beispiel 2.48 ist surjektiv, da $g_1(X) = Y$. Die Funktion ist ausserdem injektiv, weil jedem $x \in X$ höchstens ein $y \in Y$ zugewiesen wird. Nach Defintion ist die Funktion damit bijektiv (jedem $x \in X$ wird* genau ein *$y \in Y$ zugewiesen).*

Die Funktion g_2 aus Beispiel 2.48 ist weder injektiv noch surjektiv: es gilt $g_2(X) \subsetneq Y$, damit ist sie nicht surjektiv. Da zum Beispiel $g_2(1) = a$ und $g_2(4) = a$, aber $1 \neq 4$ gilt, ist sie nicht injektiv (das Urbild von a ist nicht eindeutig bestimmt).

Die Funktion f_1 aus Beispiel 2.51 ist surjektiv, da $f_1(X_1) = Y_1$, sie ist aber nicht injektiv, da das Urbild von b nicht eindeutig bestimmt ist.

Die Funktion f_2 aus Beispiel 2.51 ist nicht surjektiv, da $f_2(X_2) \subsetneq Y_2$, aber sie ist injektiv, da jedes Element im Bild $f_2(X_2)$ genau ein Urbild besitzt. Weder f_1 noch f_2 sind bijektiv.

Definition 2.54. *Seien $G_1 = (V_1, E_1)$ und $G_2 = (V_2, E_2)$ zwei einfache Graphen. Ein* Graphen-Homomorphismus *von G_1 nach G_2 ist eine Funktion $h : V_1 \to V_2$ für die gilt:*

(a) h ist surjektiv.

(b) Für alle $v, w \in V_1$ gilt für gerichtete Graphen

$$(v, w) \in E_1 \quad \Longrightarrow \quad (h(v), h(w)) \in E_2 \ \vee \ h(v) = h(w),$$

bzw. für ungerichtete Graphen

$$\{v, w\} \in E_1 \quad \Longrightarrow \quad \{h(v), h(w)\} \in E_2 \ \vee \ h(v) = h(w).$$

Wir definieren das Bild $h(G_1)$ zu $G_1 = (V_1, E_1)$ als $h(G_1) = (h(V_1), h(E_1))$, wobei

- $h(V_1) = \{h(v) \mid v \in V_1\}$, *und*
- *im gerichteten Fall* $h(E_1) = \{(h(v), h(w)) \mid \forall\ (v, w) \in E_1 :\ h(v) \neq h(w)\}$ *und im ungerichteten Fall, und* $h(E_1) = \{\{h(v), h(w)\} \mid \forall\ (v, w) \in E_1 :\ h(v) \neq h(w)\}$.

Mit dieser Notation verlangen wir von h ausserdem, dass

(c) $h(E_1) = E_2$ *gilt (also insgesamt, dass* $h(G_1) = h(G_2)$*).*

Die Bedingung (b) in Definition 2.54 sagt aus, dass Kanten zwischen zwei Knoten entweder auf Kanten zwischen den Bildern dieser Knoten abgebildet werden, oder verschiedene Knoten auf einen Knoten abgebildet werden und die Kanten dann kollabieren (verschwinden).

Beispiel 2.55. *Sei $G_1 = (V_1, E_1)$ der ungerichtete Graph aus Beispiel 2.25 mit Knotenmenge $V_1 = \{A, B, C, D, E\}$ und Kantenmenge*

$$E_1 = \{\{A, B\}, \{A, C\}, \{B, C\}, \{B, D\}, \{B, E\}, \{C, D\}, \{C, E\}, \{D, E\}\}.$$

Weiters sei $G_2 = (V_2, E_2)$ mit $V_2 = \{x, y, z\}$ und $E_2 = \{\{x, y\}, \{y, z\}\}$. Wir definieren die Abbildung

$$h = \{(A, x), (B, y), (C, y), (D, z), (E, z)\} \subseteq V_1 \times V_2.$$

Diese Abbildung ist surjektiv (es gilt $h(V_1) = V_2$). Wir überprüfen, ob Kanten gemäß der Definition erhalten bleiben:

$$\begin{aligned}
&\{A, B\} \mapsto \{h(A), h(B)\} = \{x, y\} \in E_2, && \{A, C\} \mapsto \{h(A), h(C)\} = \{x, y\} \in E_2,\\
&\{B, C\}\colon h(B) = h(C) = y, && \{B, D\} \mapsto \{h(B), h(D)\} = \{y, z\} \in E_2,\\
&\{B, E\} \mapsto \{h(B), h(E)\} = \{y, z\} \in E_2, && \{C, D\} \mapsto \{h(C), h(D)\} = \{y, z\} \in E_2,\\
&\{C, E\} \mapsto \{h(C), h(E)\} = \{y, z\} \in E_2, && \{D, E\}\colon h(D) = h(E) = z,
\end{aligned}$$

d.h., für alle Kanten gilt, dass sie entweder (strukturell) erhalten bleiben oder verschwinden und damit $h(G_1) = G_2$.

Aus der Angabe von h als Funktion zwischen zwei Knotenmengen und Bedingung (b) aus Definition 2.54 folgt bereits wie man die Kantenmenge des Bilds $h(G)$ konstruieren kann.

Beispiel 2.56. *Sei $G_1 = (V_1, E_1)$ der ungerichtete Graph aus Beispiel 2.1 (Bahnnetz) mit*

$$\begin{aligned}
V_1 &= \{B, I, S, L, P, W, E, K, G\},\\
E_1 &= \{\{B, I\}, \{I, S\}, \{S, K\}, \{S, L\}, \{L, G\}, \{L, P\}, \{P, W\}, \{W, E\}, \{W, G\}, \{G, K\}\}.
\end{aligned}$$

Seien $V_2 = \{n, o, s, w\}$ und $E_2 = \{\{w, n\}, \{n, s\}, \{n, o\}, \{o, s\}\}$, und wir definieren die Abbildung

$$h = \{(B, w), (I, w), (S, n), (L, n), (P, n), (W, o), (E, o), (K, s), (G, s)\} \subseteq V_1 \times V_2.$$

Diese Abbildung ist ein Graphen-Homomorphismus zwischen (V_1, E_1) und (V_2, E_2), der die Landeshauptstädte in die Kategorien West, Nord, Ost, Süd einteilt.

Ein Graphen-Homomorphismus ist strukturerhaltend, z.B., wenn G_1 zusammenhängend ist und h ein Graphen-Homomorphismus von G_1 nach G_2 ist, dann ist auch das Bild von h in G_2 zusammenhängend.

Beispiel 2.57. *Sei $G_1 = (V_1, E_1)$ der ungerichtete Graph mit $V_1 = \{a, b, c, d, e, f\}$ und $E_1 = \{ab, bc, ca, de, ef, fd\}$ und sei $G_2 = (V_2, E_2)$ der ungerichtete Graph mit $V_2 = \{A, B\}$ und $E_2 = \emptyset$. Die Abbildung*

$$h = \{(a, A), (b, A), (c, A), (d, B), (e, B), (f, B)\} \subseteq V_1 \times V_2$$

ist ein Graphen-Homomorphismus von G_1 nach G_2, die Zusammenhangskomponenten auf einen Knoten abbildet.

Ein zusammenhängender, ungerichteter Graph kann auf einen einzelnen Knoten durch einen Homomorphismus abgebildet werden.

Beispiel 2.58. *Sei $G = (V, E)$ der gerichtete Graph aus Beispiel 2.45 mit Knotenmenge $V = \{a, b, x, y, z\}$ und Kantenmenge $E = \{xy, yz, zx, za, ab, ba\}$. Seien $W = \{u, v\}$, $F = \{(u, v)\}$ und die Abbildung $h = \{(x, u), (y, u), (z, u), (a, v), (b, v)\} \subseteq V \times W$. Dann ist h ein Graphen-Homomorphismus von (V, E) nach (W, F), der (starke) Zusammenhangskomponenten auf Knoten abbildet.*

Ein stark zusammenhängender, gerichteter Graph kann auf einen einzelnen Knoten durch einen Homomorphismus abgebildet werden.

Definition 2.59. *Seien $G_1 = (V_1, E_1)$ und $G_2 = (V_2, E_2)$ zwei einfache Graphen. Ein* Graphen-Isomorphismus *von G_1 nach G_2 ist eine Funktion $\phi : V_1 \to V_2$ für die gilt:*

(a) ϕ ist bijektiv.

(b) Für alle $v, w \in V_1$ gilt für gerichtete Graphen:

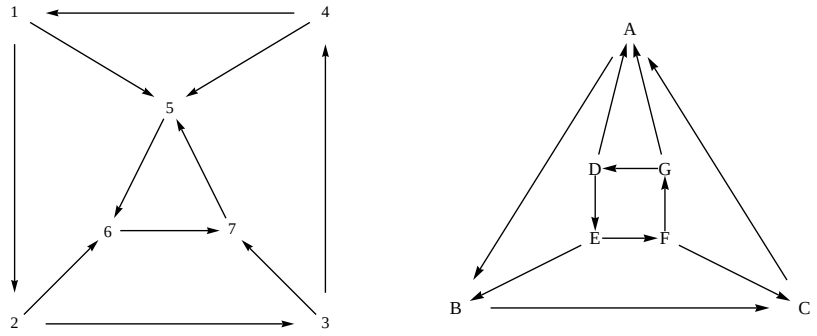
$$(v, w) \in E_1 \quad \Longleftrightarrow \quad (\phi(v), \phi(w)) \in E_2,$$

bzw. für ungerichtete Graphen

$$\{v, w\} \in E_1 \quad \Longleftrightarrow \quad \{\phi(v), \phi(w)\} \in E_2.$$

Falls ein Graphen-Isomorphismus zwischen G_1 und G_2 existiert, heissen G_1 und G_2 isomorph *und wir schreiben $G_1 \simeq G_2$. Ist $G_1 = G_2$, dann heisst ϕ* Automorphismus. *Das Bild eines Graphen-Isomorphismus ist analog zum Graphen-Homomorphismus definiert.*

Beispiel 2.60. *Seien $G_1 = (V_1, E_1), G_2 = (V_2, E_2)$ zwei Graphen mit Knotenmengen $V_1 = \{1, 2, 3, 4, 5, 6, 7\}$ und $V_2 = \{A, B, C, D, E, F, G\}$ und Kantenmengen wie im Bild dargestellt. Sind die beiden Graphen isomorph?*



Wir vergleichen charakteristische Daten der beiden Graphen:

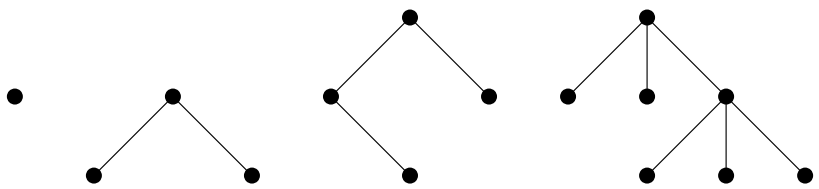
- *Gilt* $|V_1| = |V_2|$*?*
- *Gilt* $|E_1| = |E_2|$*?*
- *Wie sehen die Eingangs- und Ausgangsgrade in den beiden Graphen aus?*

G_1			G_2		
v	$\lvert E^{-1}(v)\rvert$	$\lvert E(v)\rvert$	w	$\lvert E^{-1}(w)\rvert$	$\lvert E(w)\rvert$
1	1	2	A	3	1
2	1	2	B	2	1
3	1	2	C	2	1
4	1	2	D	1	2
5	3	1	E	1	2
6	2	1	F	1	2
7	2	1	G	1	2

Lösung: die Graphen sind isomorph und ein Isomorphismus ist gegeben durch:

$$h(5) = A, \quad h(6) = B, \quad h(7) = C, \quad h(1) = D, \quad h(2) = E, \quad h(3) = F, \quad h(4) = G.$$

Definition 2.61. *Sei* $G = (V, E)$ *ein einfacher, ungerichteter Graph. Wenn* G *keine einfachen Kreise enthält, dann heisst* G Wald. *Ist* G *ausserdem* zusammenhängend, *so nennt man* G *einen* Baum. *Knoten mit Grad* 1 *heissen* Blätter *(engl.* leaves*), alle anderen Knoten heissen* innere Knoten.



Die Abbildung zeigt einen (ungerichteten) Wald von 4 Bäumen. Die Zusammenhangskomponenten in einem Wald sind Bäume. In einem (ungerichteten) Baum sind je zwei Knoten durch einen eindeutig bestimmten einfachen Weg verbunden.

Definition 2.62. *Ein einfacher, gerichteter Graph $G = (V, E)$ ist ein (gerichteter)* Baum, *genau dann wenn die folgenden Bedingungen erfüllt sind:*

(a) G ist zusammenhängend und kreisfrei.

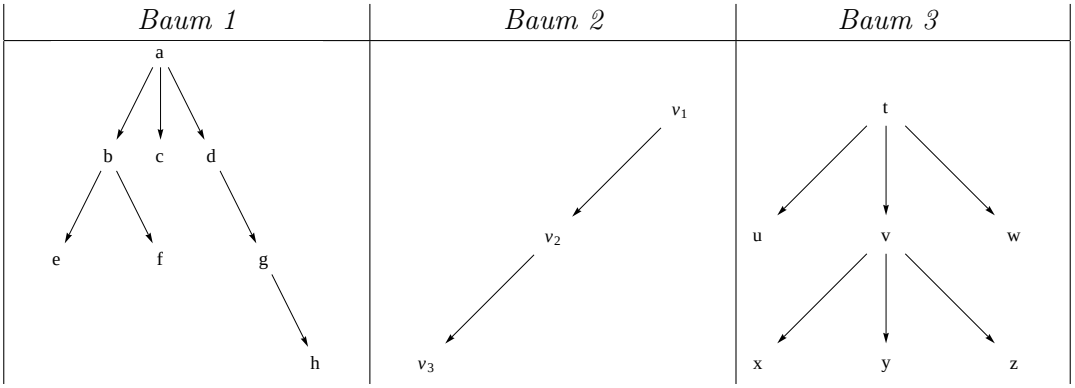
(b) G enthält genau einen Knoten mit Eingangsgrad 0*. Dieser Knoten wird als* Wurzel *(engl.* root*) des Baums bezeichnet.*

(c) Alle Knoten ausser der Wurzel haben den Eingangsgrad 1*.*

Ein Graph, dessen sämtliche Zusammenhangskomponenten Bäume sind, heisst Wald*. Knoten mit Ausgangsgrad* 0 *heissen* Blätter *(engl. leaves). Die* inneren Knoten *eines Baums sind alle Knoten, die nicht Blätter sind.*

In einem gerichteten Baum gibt es zwischen zwei Knoten entweder genau einen Weg oder keinen Weg.

Beispiel 2.63. *Bestimmen Sie zu den abgebildeten Bäumen jeweils die Knoten- und Kantenmenge, alle Blätter und alle inneren Knoten.*



Ein (gerichteter oder ungerichteter) Baum mit n Knoten hat genau $n - 1$ Kanten: Ein Baum mit einem Knoten ($n = 1$) hat keine ($n - 1 = 0$) Kanten (nur eine Wurzel). In einem Baum mit mehr als einem Knoten hat jeder Knoten, ausser der Wurzel, genau eine eingehende Kante.

Definition 2.64. *Sei $T = (V, E)$ ein gerichteter Baum mit Wurzel $r \in V$. Die* Tiefe *(engl. depth) $d_T(v)$* eines Knoten $v \in V$ *ist die* Länge des Wegs $W(r, v)$*, d.h., die Anzahl der besuchten Kanten des Wegs.*

Die Tiefe des Baums T *ist die grösste Distanz von der Wurzel zu einem Blatt, d.h. $d(T) = \max_{v \in V} d_T(v)$.*

Beispiel 2.65. *Für die Bäume aus Beispiel 2.63 gilt: Baum 1 hat Tiefe 3, der Knoten a hat die Tiefe* 0*, die Knoten b, c, d haben Tiefe* 1*, die Knoten e, f, g haben Tiefe 2 und der Knoten h hat Tiefe* 3*. Baum 2 und Baum 3 haben jeweils Tiefe 2.*

Ein Baum, in dem jeder Knoten höchstens zwei Nachfolger hat heisst *binärer Baum*. Ein binärer Baum kann höchstens 2^d Knoten mit Tiefe d haben. Ein binärer Baum der Tiefe t kann damit höchstens

$$2^0 + 2^1 + \cdots + 2^t = 2^{t+1} - 1$$

Knoten haben.

Definition 2.66. *Sei $G = (V, E)$ ein endlicher, einfacher Graph. Ein Teilgraph $T = (V, E_T)$ mit $E_T \subseteq E$, der ein Baum ist, heisst* aufspannender Baum *(engl. spanning tree) oder* Spannbaum.

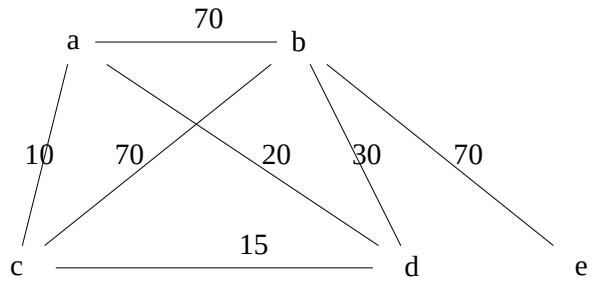
In einem *gewichteten Graphen* sind die Kanten zusätzlich mit einer Gewichtsfunktion, die z.B. als Kosten verstanden werden können, versehen. Ein Spannbaum in einem gewichteten Graphen, der die Summe der Gewichte (Kosten) minimiert wird *minimaler Spannbaum* (engl. minimal spanning tree) genannt. Ein Algorithmus zur Bestimmung eines *minimal spanning trees* ist der *Algorithmus von Kruskal*:

Input Ungerichteter, einfacher Graph, zusammenhängender $G = (V, E)$, $|E| = n$, mit Gewichtsfunktion $w\colon E \to \mathbb{R}$.

Output Kantenmenge E_T eines minimal spanning tree von G.

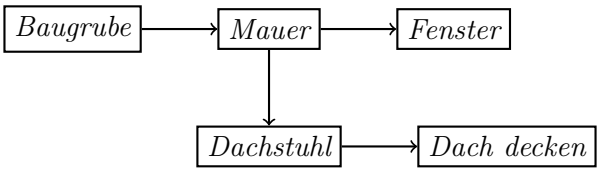
1. Sortiere die Kanten von G aufsteigend nach ihrem Gewicht: $E_S = \{e_1, e_2, \ldots, e_n\}$ mit $w(e_j) \leq w(e_{j+1})$
2. Initialisiere $E_T = \emptyset$
3. Für alle $j = 1, \ldots, n$: falls $E_T \cup \{e_j\}$ keinen einfachen Kreis enthält, setze $E_T = E_T \cup \{e_j\}$
4. Return E_T

Beispiel 2.67. *Was ist das Ergebnis von Kruskal's Algorithmus mit dem abgebildeten Graphen als Input?*



Topologisches Sortieren (TopSort) Gegeben einen gerichteten azyklischen Graph $G = (V, E)$ (engl. *directed acyclic graph = DAG*), ist eine lineare Anordnung der Knoten gesucht, sodass für jede Kante $(v, w) \in E$, v in der Ordnung vor w kommt. D.h., wenn die Knoten zum Beispiel Aufgaben (tasks) entsprechen und die (gerichteten) Kanten die Abhängigkeit angeben, welche Aufgaben abgeschlossen werden müssen, damit die nächste Aufgabe in Angriff genommen werden kann, dann liefert TopSort eine gültige Abfolge der Aufgaben.

Beispiel 2.68. *Auf einer Baustelle muss zunächst das Fundament ausgehoben werden, bevor eine Mauer errichtet werden kann. Dann kann man mit entweder Fenster oder Dachstuhl beginnen (oder parallel mit beiden), aber das Dach kann erst gedeckt werden, wenn der Dachstuhl fertig ist, also:*



Mögliche topologische Sortierungen sind z.B. (Baugrube, Mauer, Fenster, Dachstuhl, Dach decken), oder (Baugrube, Mauer, Dachstuhl, Fenster, Dach decken).

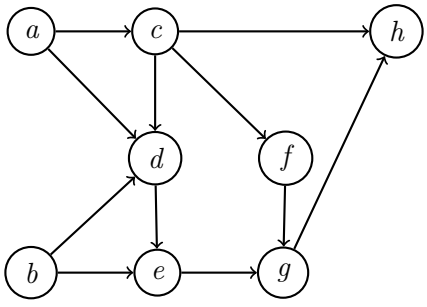
Wenn der gegebene Graph keine Zyklen enthält, dann gibt es mindestens einen Knoten mit Eingangsgrad 0 (*Quelle*), mindestens einen Knoten mit Ausgangsgrad 0 (*Senke*), und es existiert *mindestens* eine topologische Sortierung, die in linearer Zeit berechenbar ist (Kahn, 1962).

Input Gerichteter azyklischer Graph $G = (V, E)$

Output Topologische Sortierung T

1. Initialisiere T als die leere Liste; $T = (\,)$
2. Setze L gleich die Menge aller Quellen und isolierter Knoten in G; $L = \{v \in V \mid |E^{-1}(v)| = 0\}$
3. While $L \neq \emptyset$ do
   (a) wähle einen Knoten v aus L; $L = L \backslash \{v\}$
   (b) füge v zu T hinzu
   (c) Für jede Kante $e \in E$:
      (c1) If $e = (v, w)$ then:
      - $E = E \backslash \{e\}$
      - If $|E^{-1}(w)| = 0$ then füge w zu L hinzu
4. If $E \neq \emptyset$ then return "Fehler" (der Graph hat einen Zyklus)
5. return T

Beispiel 2.69. *Wir wenden TopSort auf den Graphen $G = (V, E)$ mit $V = \{a, b, c, d, e, f, g, h\}$ und $E = \{(a, c), (a, d), (b, d), (b, e), (c, d), (c, f), (c, h), (d, e), (e, g), (f, g), (g, h)\}$ an (siehe Bild).*



Eine topologische Sortierung dieses Graphen ist z.B. (a, c, f, b, d, e, g, h).

Petri-Netze wurden 1962 im Rahmen seiner Dissertation von Carl Adam Petri eingeführt und zur Modellierung, Analyse und Simulation von diskreten, dynamischen (vorwiegend verteilten) Systemen entwickelt. Wir können sie als gerichtete Graphen mit zwei Arten von Knoten betrachten: *Stellen* (auch *Plätze, Zustände*) und *Transitionen* (auch *Hürden, Zustandsübergänge*). Kanten dürfen jeweils nur von einer Art Knoten zur anderen Art von Knoten führen.

Stellen, von denen Kanten zu einer Transition t führen heissen *Eingabestellen* von t, oder auch *Vorbereich* von t. Stellen zu denen (von einer Transition t aus) Kanten führen, heissen *Ausgabestellen* von t, oder auch *Nachbereich* von t.

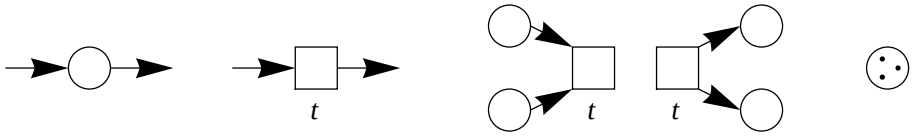


Abbildung 2.2: Stelle, Transition, Eingabestellen von t, Ausgabestellen von t, Stelle mit Marken

Um dynamische Vorgänge beschreiben zu können, werden die Stellen mit *Marken* (auch *tokens*) belegt. Bewegungsabläufe der Marken im Netz werden durch die folgende *Schaltregel* festgelegt:

(a) Eine Transition t kann *schalten* (oder *feuern*), wenn jede Eingabestelle von t mindestens eine Marke enthält.

(b) Schaltet eine Transition, dann wird von *jeder* Eingabestelle eine Marke entfernt und zu *jeder* Ausgabestelle eine Marke hinzugefügt.

Graphisch werden Petri-Netze so dargestellt, dass Stellen als Kreise gezeichnet werden, Transitionen als Rechtecke (oder Balken) und Marken als schwarze Punkte in den Kreisen der entsprechenden Stellen, siehe Figur 2.2.

Im allgemeinen kann eine Stelle auch mehr als eine Marke enthalten und eine Transition mehr als eine Marke pro Eingabestelle als Input benötigen, bzw., mehr als eine Marke an die Ausgabestellen abgeben. In dem Fall werden die Stellen zusätzlich mit einem Label versehen, das die Kapazität angibt und die Kanten mit einem Gewicht versehen. Als Defaultwert (d.h. falls kein Gewicht angegeben ist) für Kanten gilt das Gewicht 1.

Ausserdem können auch verschiedene Marken verwendet werden (die verschiedenen Eingabegrössen entsprechen). Vom Typ *Boolean* sind jene Netze, bei denen jede Stelle entweder eine oder keine Marke hat. In dem Fall erweitert sich die Schaltregel (a) zu

(a') Eine Transition t kann *schalten* (oder *feuern*), wenn jede Eingabestelle von t eine Marke enthält und jede Ausgabestelle leer ist.

Beispiel 2.70. *Wir betrachten ein boolesches Petri-Netz zur Zubereitung einer Wurstsemmel:*

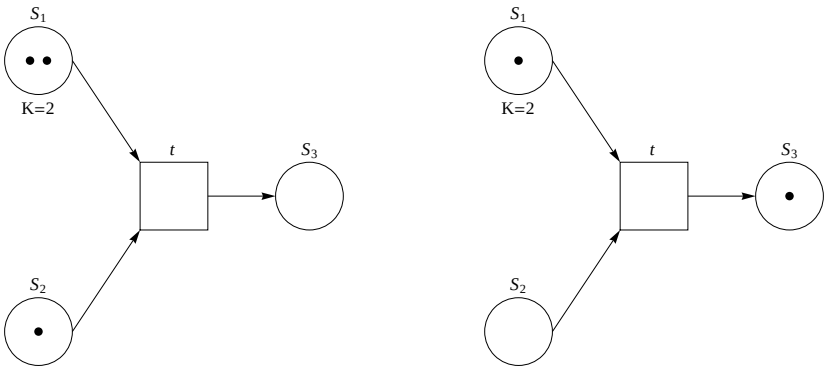
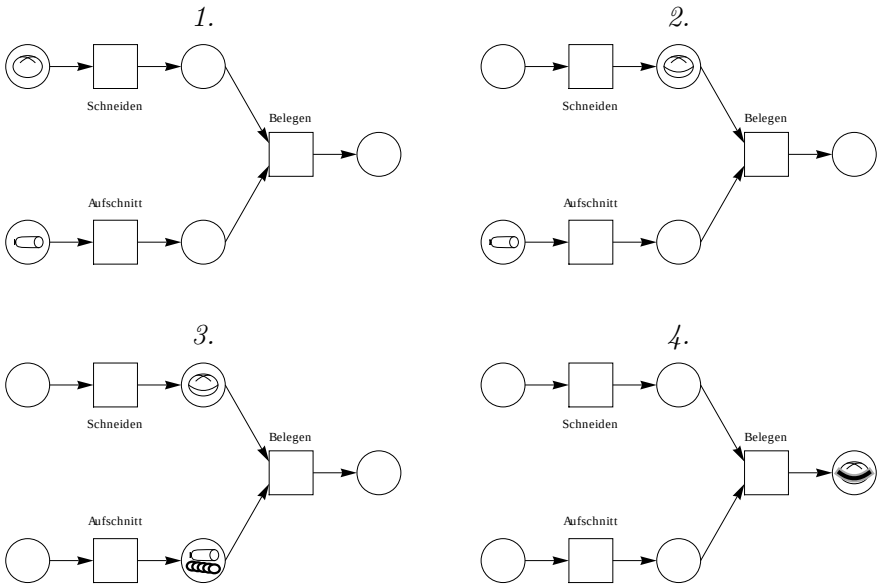


Abbildung 2.3: Links: Die Transition t kann feuern; rechts: Zustand nach Schaltung von t, von jeder der beiden Eingabestellen s_1, s_2 wurde eine Marke entfernt und *eine* Marke wurde bei der Ausgabestelle hinzugefügt. (Dieses Netz ist *nicht* boolean.)



1. *Im ersten Schritt sind die Eingabestellen der Transitionen "Schneiden" und "Aufschnitt" belegt und die jeweiligen Ausgabestellen frei, d.h. jede der beiden Transitionen könnte feuern. Wir schneiden zuerst die Semmel.*

2. *Im zweiten Schritt gibt es keine Auswahlmöglichkeit mehr - nur die Transition "Aufschnitt" kann aktiviert werden, "Belegen" wartet noch auf Input.*

3. *Im dritten Schritt sind beide Eingabestellen von "Belegen" besetzt und die Transition kann feuern (keine der beiden anderen Transitionen könnte im Moment aktiviert werden).*

4. *Wir haben erfolgreich eine Wurstsemmel zubereitet.*

Weitere Beispiele findet man im Online-Tutorial von Wil van der Aalst (TU Eindhoven)

http://www.informatik.uni-hamburg.de/TGI/PetriNets/introductions/aalst/

Von den dort angeführten Beispielen betrachten wir jetzt die Ampelschaltungen und die "Dining philosophers".

Beispiel 2.71. *(http://.../trafficlight1.swf) Modell einer Ampel, die von rot zu grün zu gelb zu rot schaltet.*

Beispiel 2.72. *(http://.../trafficlight2.swf) Die Erweiterung des vorherigen Modells zu zwei Ampeln, die nicht gleichzeitig grün sein dürfen.*

Beispiel 2.73. *(http://.../philosopher4.swf) (Dining philosophers) Zunächst betrachten wir einen Philosophen: ein Philosoph kann zwischen den beiden Zuständen "Denken" und "Essen" wechseln. Nun werden n (im Link $n = 4$) Philosophen an einen Tisch gesetzt, auf dem eine (sich nie leerende) Schüssel Spaghetthi steht und zwischen je zwei Philosophen liegt eine Gabel. Im Anfangszustand denken alle Philosophen. Ein Philosoph kann zu essen beginnen, falls ihm zwei Gabeln zur Verfügung stehen. Durch das Aufnehmen von zwei Gabeln geht der Philosoph vom Zustand des Denkens zum Zustand des Essens über. Durch Ablegen der Gabeln wird das Essen beendet und Denken wieder aufgenommen.*

3 Sprachen und Automaten

3.1 Sprache und Grammatik

Definition 3.1. *Ein* Alphabet Σ *ist eine* endliche *Menge. Jedes Element* $\sigma \in \Sigma$ *ist ein* Zeichen *des Alphabets. Die Elemente von* Σ^n *für* $n \in \mathbb{N}$ *werden als* Wörter *der Länge* n *bezeichnet. Das Wort der Länge* 0 *heisst das* leere Wort *und wird mit* ε *bezeichnet.*

Die Verkettung von zwei Wörtern $x = (x_1, \ldots, x_n) \in \Sigma^n$ und $y = (y_1, \ldots, y_m) \in \Sigma^m$ über dem Alphabet Σ wird als $xy = (x_1, \ldots, x_n, y_1, \ldots, y_m) \in \Sigma^{n+m}$ definiert. Für Wörter verwenden wir statt der Tupelschreibweise der Einfachheit halber die Notation $(x_1, x_2, \ldots, x_n) = x_1 x_2 \cdots x_n$ (was auch mehr nach Wörtern aussieht). Das leere Wort ist das Wort ohne Zeichen und es gilt $\varepsilon x = x\varepsilon = x$. Mit dieser Notation gilt

$$\Sigma^0 = \{\varepsilon\}, \quad \Sigma^1 = \Sigma, \quad \Sigma^{n+1} = \{xy \mid x \in \Sigma, y \in \Sigma^n\}.$$

Ausserdem definieren wir

$$\Sigma^+ = \bigcup_{i=1}^{\infty} \Sigma^i \quad \text{und} \quad \Sigma^* = \bigcup_{i=0}^{\infty} \Sigma^i.$$

Die Menge Σ^* wird als *Kleenesche Hülle* von Σ bezeichnet.

Definition 3.2. *Jede Teilmenge* $L \subseteq \Sigma^*$ *ist eine* formale Sprache *über* Σ.

Beispiel 3.3. *Für das Alphabet* $\Sigma = \{a, b\}$ *gilt*

$$\Sigma^0 = \{\varepsilon\}, \quad \Sigma^1 = \{a, b\}, \quad \Sigma^2 = \{aa, ab, ba, bb\}, \ldots,$$

und, zum Beispiel,

$$\Sigma^* = \{\varepsilon, a, b, aa, ab, ba, bb, aaa, aab, aba, \ldots\}.$$

Sprachen über Σ *sind dann zum Beispiel,*

$$L_1 = \{ab, ba\},\ L_2 = L_1^2 = \{abab, abba, baab, baba\},\ L_3 = \{a, b, aa, ab, ba, aaa, aab, aba, baa\},$$
$$L_4 = \{ab, aab, aaab, aaaab, \ldots\}.$$

Letzteres ist die Sprache aller Wörter mit beliebig vielen a (mindestens einem) gefolgt von genau einem b. Diese Sprache ist eine unendliche Sprache.

Beispiel 3.4. *(Dyck-Sprache* D_2*) Zum Alphabet* $\Sigma = \{(,), [,]\}$*, sei die Sprache* L *über* Σ *definiert als die Menge der korrekt geklammerten Ausdrücke. Dann sind die folgenden Wörter*

$$(), \quad ([]), \quad ()[()]()$$

Elemente der Sprache, nicht aber

$$), \quad [(]), \quad (x).$$

Beispiel 3.5. *(Palindromsprache) Zum Alphabet* $\Sigma = \{a, b, c, d, \ldots, x, y, z\}$ *sei* L *die Sprache über* Σ *definiert als die Menge aller spiegelbildlich angeordneten Zeichenketten. Dann sind*

$$aaa, aba, rotor, radar, reliefpfeiler,$$

Wörter dieser Sprache, aber nicht

$$bba, cba, fluegel, auto, brueckenpfeiler.$$

Im Bereich der formalen Sprachen sind verschiedene Fragestellungen von Interesse, zum Beispiel:

- Gegeben eine Sprache L über einem Alphabet Σ und ein Wort $\omega \in \Sigma^*$, stelle fest, ob ω ein Element der Sprache L ist. (Wortproblem)
- Gegeben eine Sprache L über einem Alphabet Σ, stelle fest, ob die Sprache nichtleer ist. (Leerheitsproblem)
- Gegeben eine Sprache L über einem Alphabet Σ, stelle fest ob die Sprache endlich ist. (Endlichkeitsproblem)
- Gegeben zwei Sprachen L_1 und L_2 über einem Alphabet Σ, stelle fest, ob die beiden Sprachen äquivalent sind, also $L_1 = L_2$ gilt. (Äquivalenzproblem)
- Gegeben eine Sprache L über einem Alphabet Σ, gibt es eine Beschreibung, aus der sich alle Wörter ableiten lassen. (Spracherzeugung)

Wir beginnen mit dem letztgenannten Problem und zu diesem Zweck führen wir den Begriff einer *Grammatik* ein.

Definition 3.6. $G = (V, \Sigma, S, \Pi)$ *heisst* Grammatik*, wenn*

(a) V *und* Σ *Alphabete sind mit* $V \cap \Sigma = \emptyset$*;* V *heisst die Menge der* Nichtterminale *(Variablen, Nonterminale, Nichtterminalzeichen) und* Σ *die Menge der* Terminale *(Endzeichen, Terminalzeichen),*

(b) $S \in V$ *ist (*S *heisst* Startvariable *(Anfangssymbol)),*

(c) Π *ist eine* endliche *Menge von Paaren* (P, Q) *ist mit* $P \in (V \cup \Sigma)^* V (V \cup \Sigma)^*$ *und* $Q \in (V \cup \Sigma)^*$*. Ein Paar* $(P, Q) \in \Pi$ *heisst* Ersetzungsregel *oder* Produktion.

Nichtterminale werden üblicherweise mit Grossbuchstaben, Terminale mit Kleinbuchstaben gekennzeichnet. Nichtterminale (wenn sie zum Beispiel aus Wörtern im herkömmlichen Sinn bestehen), werden auch durch spitze Klammern kenntlich gemacht, z.B.,

$$\langle \text{Hauptwort} \rangle, \quad \langle \text{Zeitwort} \rangle, \quad \langle \text{Eigenschaftswort} \rangle.$$

Beispiel 3.7. *Eine Grammatik zur Erzeugung der Dyck-Sprache* D_2 *ist gegeben durch* $G = (\{S\}, \{(,), [,]\}, S, \Pi)$ *mit*

$$\Pi = \{(S, \varepsilon), (S, SS), (S, [S]), (S, (S))\},$$

d.h., $D_2 = L(G)$. Man beachte, dass es sich hier um keine *Funktion handelt - die Zuweisung ist* nichtdeterministisch. *Für die Produktionsmenge* Π *werden wir auch folgende Notation verwenden:*

$$S \to \varepsilon \quad (a)$$
$$S \to SS \quad (b)$$
$$S \to [S] \quad (c)$$
$$S \to (S) \quad (d)$$

oder noch kürzer die folgende Notation verwenden:

$$S \to \varepsilon \mid SS \mid [S] \mid (S).$$

Definition 3.8. *(Fortsetzung von Definition 3.6) Jede Grammatik erzeugt eine Sprache $L(G)$, die alle Wörter über dem Alphabet Σ enthält, die sich aus dem Startsymbol ableiten lassen, d.h.,*

$$L(G) = \{\omega \in \Sigma^* \mid S \to^* \omega\},$$

wobei "$\to^*$" für eine Folgen von Anwendungen von verschiedenen Ersetzungsregeln aus Π *steht. Wir nennen $L(G)$, die von G* erzeugte Sprache, *bzw. G auch die* erzeugende Grammatik.

Beispiel 3.9. *(Fortsetzung von Beispiel 3.7) Durch Anwenden der Ersetzungsregeln (Produktionen) können Wörter der erzeugten Sprache abgleitet werden. Im folgenden führen wir eine* Linksableitung *durch, d.h., es wird immer das Nichtterminal ersetzt, das am weitesten* links *steht:*

$$S \xrightarrow{(b)} SS \xrightarrow{(d)} (S)S \xrightarrow{(a)} ()S \xrightarrow{(b)} ()SS \xrightarrow{(c)} ()[S]S \xrightarrow{(d)} ()[(S)]S \xrightarrow{(a)} ()[()]S$$
$$\xrightarrow{(d)} ()[()](S) \xrightarrow{(a)} ()[()]().$$

Wir leiten das gleiche Wort durch eine Rechtsableitung *ab, d.h., es wird immer das Nichtterminal ersetzt, das am weitesten* rechts *steht:*

$$S \xrightarrow{(b)} SS \xrightarrow{(d)} S(S) \xrightarrow{(a)} S() \xrightarrow{(b)} SS() \xrightarrow{(c)} S[S]() \xrightarrow{(d)} S[(S)]() \xrightarrow{(a)} S[()]()$$
$$\xrightarrow{(d)} (S)[()]() \xrightarrow{(a)} ()[()]().$$

ACHTUNG: Im allgemeinen ist die Ableitungssequenz nicht eindeutig bestimmt! (Beispiel: das Wort $()[()]() \in D_2$)

Beispiel 3.10. *Wir definieren die Grammatik $G = (\{E, T\}, \{x, y, \oplus, \odot, (,)\}, E, \Pi\}$ ($E = \langle expression \rangle$, $T = \langle term \rangle$), mit Produktionen*

$$E \to E \oplus E$$
$$E \to T$$
$$T \to T \odot T$$
$$T \to (E)$$
$$T \to x$$
$$T \to y$$

bzw. in Kurzschreibweise:

$$E \to E \oplus E \mid T$$
$$T \to T \odot T \mid (E) \mid x \mid y$$

Definition 3.11. *Sei $G = (V, \Sigma, S, \Pi)$ eine Grammatik und $(P, Q) \in \Pi$ eine Produktion. Dann heisst (P, Q)*

- linkslinear*, falls $P \in V$ und $Q \in (V\Sigma^*) \cup \Sigma^*$ (d.h., auf der linken Seite kann nur ein Nichtterminalzeichen stehen und falls auf der rechten Seite ein Nichtterminalzeichen steht, dann nur ganz links).*
- rechtslinear*, falls $P \in V$ und $Q \in (\Sigma^* V) \cup \Sigma^*$ (d.h., auf der linken Seite kann nur ein Nichtterminalzeichen stehen und falls auf der rechten Seite ein Nichtterminalzeichen vorkommt, dann nur ganz rechts).*
- kontextfrei*, falls $P \in V$ (d.h., auf der linken Seite steht nur ein Nichtterminalzeichen).*
- kontextsensitiv*, falls $P = \omega_1 X \omega_2$ und $Q = \omega_1 \omega \omega_2$ mit $X \in V$ und $\omega \in (V \cup \Sigma)^+$ (d.h., das Nichtterminalzeichen X wird im Kontext $\omega_1 .. \omega_2$ durch das* nichtleere *Wort ω ersetzt).*

Die Grammatik G heisst linkslinear, wenn alle ihre Produktionen linkslinear sind (und analog für rechtslinear, kontextfrei und kontextsensitiv).

Beispiel 3.12. *Zu welchen Klassen gehören die folgenden Produktionen, wobei S, X Nichtterminalzeichen und a, b Terminalzeichen sind:*

$$(a) \quad S \to \varepsilon \quad (b) \quad S \to a \quad (c) \quad S \to aX \quad (d) \quad S \to Sb$$
$$(e) \quad S \to aSb \quad (f) \quad S \to aSb \quad (g) \quad aXb \to bSXa$$

Bemerkung 3.13.

- *Eine Grammatik, die* rechtslinear *oder* linkslinear *ist, wird auch* regulär *genannt.*
- *Jede reguläre Grammatik ist auch kontextfrei (auf der linken Seite steht nur ein Nichtterminalzeichen, die Einschränkung für eine reguläre Grammatik betrifft nur die rechte Seite).*
- *Jede kontextfreie Produktion außer der ε-Produktion ($P \to \varepsilon$) ist auch kontextsensitiv (mit $\omega_1 = \omega_2 = \varepsilon$).*
- *In einer kontextsensitiven Grammatik gibt es* keine Wortverkürzung.
- *Eine Produktion der Form $P \to \varepsilon$ ist in einer kontextsensitiven Grammatik G ausgeschlossen. Das führt dazu, dass das leere Wort nie in der von G erzeugten Sprache liegen kann. Oft möchte man das leere Wort auch in einer von einer kontxtsensitiven Grammatik erzeugten Sprache haben und daher wird oft die Ausnahme zugelassen, dass als einzige ε-Produktion die Ersetzungsregel $S \to \varepsilon$ (also vom Startsymbol auf das leere Wort) zugelassen wird.*

Noam Chomsky hat 1957 die vier Klassen von Grammatiken aus Definition 3.11 eingeführt (rechtslineare und linkslineare Grammatiken fallen in der Klasse der regulären Grammatiken zusammen). Diese Klassen führen zu einer Einteilung von (formalen) Sprachen, die nach der Struktur der Produktionen der erzeugenden Grammatik definiert ist.

Definition 3.14. *(Chomsky-Sprachklassen) Sei Σ ein Alphabet. Eine Sprache $L \subseteq \Sigma^*$ heisst*

- regulär *oder* Chomsky-3 *oder* Typ 3-Sprache*, wenn es eine reguläre (d.h., eine rechtslineare oder eine linkslineare) Grammatik G mit $L = L(G)$ gibt;*
- kontextfrei *oder* Chomsky-2 *oder* Typ 2-Sprache*, wenn es eine kontextfreie Grammatik G mit $L = L(G)$ gibt;*
- kontextsensitiv *oder* Chomsky-1 *oder* Typ 1-Sprache*, wenn es eine kontextsensitive Grammatik G mit $L = L(G)$ gibt;*
- Chomsky-0 *oder* Typ 0-Sprache*, wenn es eine (Phrasenstruktur-)Grammatik (d.h., eine Grammatik, die keinen weiteren Einschränkungen unterliegt) G mit $L = L(G)$ gibt.*

Wenn eine Sprache durch eine rechtslineare Grammatik erzeugt werden kann, dann existiert eine linkslineare Grammatik, die die selbe Sprache erzeugt und umgekehrt. Eine Sprache ist also genau dann rechtslinear, wenn sie linkslinear ist und wir verwenden (wie in der Definition) den Ausdruck reguläre Sprache.

Eine Sprache L wird als Typ n-Sprache bezeichnet, wenn eine Typ n-Grammatik existiert, die L erzeugt. Die Menge aller Typ n-Sprachen bezeichnen wir mit $\mathcal{L}_n$ und es gilt (siehe auch Bemerkung 3.13)

$$\mathcal{L}_3 \subsetneq \mathcal{L}_2 \subsetneq \mathcal{L}_1 \subsetneq \mathcal{L}_0.$$

Beispiel 3.15. *Für jede dieser Sprachklassen gibt es Beispiele, die in $\mathcal{L}_n$, aber nicht in der nächstkleineren Klasse $\mathcal{L}_{n+1}$ liegt (die Inklusion oben ist also strikt):*

- $L_3 = \{(ab)^n \mid n \in \mathbb{N}\}$ *ist eine Typ 3-Sprache; (Wörter sind Wiederholungen der Zeichenkette ab beliebig oft (auch 0 mal), also z.B.: $\varepsilon, ab, abab, ababab, \dots$)*
- $L_2 = \{a^n b^n \mid n \in \mathbb{N}\}$ *ist eine Typ 2-Sprache, aber keine Typ 3-Sprache; (Wörter, die eine beliebige Anzahl von a enthalten (auch 0 mal) und anschliessend die gleiche Anzahl von b, also z.B.: $\varepsilon, ab, aabb, aaabbb, \dots$)*
- $L_1 = \{a^n b^n c^n \mid n \in \mathbb{N}\}$ *ist eine Typ 1-Sprache, aber keine Typ 2-Sprache; (Wörter, die eine beliebige Anzahl von a enthalten (auch 0 mal), gefolgt von der gleichen Anzahl von b, gefolgt von der gleichen Anzahl von c, also z.B.: $\varepsilon, abc, aabbcc, aaabbbccc, \dots$)*
- $L_0 = \{a^{2^n} \mid n \in \mathbb{N}\}$ *ist eine Typ 0-Sprache, aber keine Typ 1-Sprache. (Wörter, die aus einer Zeichenkette von 2^n a's bestehen, also z.B.: $a, aa, aaaa, aaaaaaaa, \dots$)*

3.2 Endliche Automaten und reguläre Sprachen

Lemma 3.16. *(Normalform) Sei $L \subseteq \Sigma^*$ eine rechtslineare Sprache. Dann gibt es eine Grammatik $G = (V, \Sigma, S, \Pi)$ mit $L = L(G)$ sodass alle Produktionen in Π von einer der folgenden drei Formen sind:*

- $X \to aY$ *(mit $X, Y \in V$ und $a \in \Sigma$)*
- $X \to a$ *(mit $X \in V$ und $a \in \Sigma$)*
- $S \to \varepsilon$

Beweisidee. Umformen und eventuell Einführen neuer Variablen. □

Beispiel 3.17. *Wir betrachten die Menge von Produktionen*

$$\Pi = \{S \to abS, S \to aY, Y \to Z, Z \to \varepsilon\}.$$

Das erste Element kann durch Einführen einer neuen Variable ersetzt werden:

$$\{S \to aR, R \to bS, S \to aY, Y \to Z, Z \to \varepsilon\}.$$

Die vorletzte Produktion ist eine Kettenproduktion *und kann wie folgt ersetzt werden:*

$$\{S \to aR, R \to bS, S \to aY, Y \to \varepsilon, Z \to \varepsilon\}.$$

Jede ε-Produktion $X \to \varepsilon$ (mit $X \neq S$) kann redundant gemacht und dann gelöscht werden:

$$\{S \to aR, R \to bS, S \to a\}.$$

Definition 3.18. *Ein* deterministischer endlicher Automat *über einem Alphabet Σ, kurz* DEA *(oder* DFA *für* deterministic finite automaton*) ist ein Tupel $A = (Q, \Sigma, \delta, q_0, F)$ mit folgenden Komponenten:*

- *Q ist eine endliche Menge von* Zuständen *(engl. states).*
- *$\delta\colon M \subseteq Q \times \Sigma \to Q$ ist eine* Funktion *genannt die* Überleitungsfunktion *oder* Transition*.*
- *$q_0 \in Q$ ist der* Anfangszustand*.*
- *$F \subseteq Q$ ist eine Menge von* Endzuständen*.*

Bemerkung 3.19.

- *Die Übergangsfunktion ist auf einer Teilmenge von $Q \times \Sigma$ definiert, da nicht jeder Zustand durch jedes Zeichen in einen neuen Zustand übergeführt werden muss.*
- *Um endliche Automaten zu skizzieren werden Zustände mit Kreisen gezeichnet und Transitionen durch Pfeile angegeben. Das (oder die) Zeichen, das zur Transition führt wird beim Pfeil angegeben. Der Startzustand ist durch einen eingehenden Pfeil gekennzeichnet. Endzustände durch einen Kreis mit doppelter Kreislinie. (siehe folgendes Beispiel)*

Beispiel 3.20. *$A_1 = (\{q_0, q_1\}, \{0, 1\}, \delta_1, q_0, \{q_1\})$, wobei die Transition δ_1 gegeben ist durch*

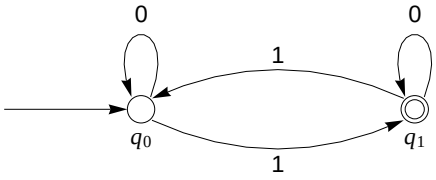
$$q_0 \xrightarrow{0} q_0, \quad q_1 \xrightarrow{0} q_1, \quad q_0 \xrightarrow{1} q_1, \quad q_1 \xrightarrow{1} q_0.$$

Die Transition kann auch durch Tupel angegeben werden, in diesem Fall durch:

$$\delta = \{(q_0, 0, q_0), (q_1, 0, q_1), (q_0, 1, q_1), (q_1, 1, q_0)\},$$

oder in Funktionsschreibweise durch

$$\delta(q_0, 0) = q_0, \quad \delta(q_1, 0) = q_1, \quad \delta(q_0, 1) = q_1, \quad \delta(q_1, 1) = q_0.$$



Gegeben einen endlichen Automaten $A = Q, \Sigma, \delta, q_0, F)$ definieren wir die von A akzeptierte Sprache $L(A)$, als die Sprache jener Wörter $w = x_1 x_2 \dots x_n \in \Sigma^*$, für die es ausgehend vom Startzustand q_0 eine Folge von Zuständen $q_1, \dots, q_{n-1} \in Q$ gibt und einen Endzustand $q \in F$ sodass

$$q_0 \xrightarrow{x_1} q_1 \xrightarrow{x_2} q_2 \xrightarrow{x_3} \dots \xrightarrow{x_n} q.$$

Das heisst, wenn die gegebene Zeichenkette vom Automaten verarbeitet werden kann und am Wortende in einem Endzustand hält. Für die obige Ableitungsfolge werden wir auch die Kurznotation

$$q_0 \xrightarrow{w} q$$

verwenden (d.h. es existieren Zustände sodass ausgehend vom Startzustand beim Durchlauf des Automaten durch Transitionen ein Endzustand erreicht werden kann).

Definition 3.21. *Die von einem DEA* $A = (Q, \Sigma, \delta, q_0, F)$ akzeptierte Sprache *ist definiert als*

$$L(A) = \{w \in \Sigma^* \mid \exists\, q \in F \colon q_0 \xrightarrow{w} q\}.$$

Zwei Automaten A_1, A_2 *heissen* (sprach-)äquivalent*, wenn* $L(A_1) = L(A_2)$.

Wenn A ein deterministisch endlicher Automat ist, heisst jede von A akzeptierte Sprache L auch *deterministisch endlich akzeptierbar.*

Beispiel 3.22. *Die von dem in Beispiel 3.20 definierten Automat* A_1 *akzeptierte Sprache ist*

$$L(A_1) = \{w \in \{0,1\}^* \mid w \text{ enthält ungerade viele Zeichen } 1\}.$$

Wir betrachten die beiden Wörter $w_1 = 01011000$ *und* $w_2 = 011011$ *und wollen jeweils feststellen, ob diese Wörter in* $L(A)$ *liegen. Für* w_1 *erhalten wir die Sequenz:*

$$q_0 \xrightarrow{0} q_0 \xrightarrow{1} q_1 \xrightarrow{0} q_1 \xrightarrow{1} q_0 \xrightarrow{1} q_1 \xrightarrow{0} q_1 \xrightarrow{0} q_1 \xrightarrow{0} q_1 \xrightarrow{\varepsilon} q_1.$$

Das Wortende ist durch ein leeres Wort gekennzeichnet und das leere Wort führt auf den aktuellen Zustand zurück (entspricht der identischen Abbildung). Die Folge zeigt also, dass das Wort vom gegebenen Automaten akzeptiert wird, da

$$q_0 \xrightarrow{w_1} q_1 \in F.$$

Für w_2 *erhalten wir die Sequenz*

$$q_0 \xrightarrow{0} q_0 \xrightarrow{1} q_1 \xrightarrow{1} q_0 \xrightarrow{0} q_0 \xrightarrow{1} q_1 \xrightarrow{1} q_0 \xrightarrow{\varepsilon} q_0 \notin F.$$

Hier hält der Automat nicht in einem Endzustand, das Wort wird also nicht vom Automaten akzeptiert und liegt damit nicht in $L(A)$.

Definition 3.23. *Ein* nichtdeterministischer endlicher Automat *über einem Alphabet* Σ*, kurz* NEA *(oder* NFA *für* nondeterministic finite automaton*) ist ein Tupel* $A = (Q, \Sigma, \delta, Q_0, F)$ *mit folgenden Komponenten:*

- Q *ist eine endliche Menge von* Zuständen *(engl. states).*
- $\delta\colon Q \times \Sigma \to Q$ *ist die* Überleitungsrelation *(d.h.,* δ *ist eine Abbildung von* $Q \times \Sigma$ *nach* Q*, die* keine Funktion sein muss*).*
- $Q_0 \subseteq Q$ *ist eine Menge von* Anfangszuständen.
- $F \subseteq Q$ *ist eine Menge von* Endzuständen.

D.h., ein NEA $A = (Q, \Sigma, \delta, Q_0, F)$ ist ein DEA, wenn Q_0 nur aus einem Element besteht und δ eine Funktion ist. Eine Sprache heisst *endlich akzeptierbar*, wenn sie von einem NEA akzeptiert wird.

Beispiel 3.24. *Zu einem gegebenen Alphabet* Σ *und einem Wort* $v \in \Sigma^*$ *definieren wir die Sprache*

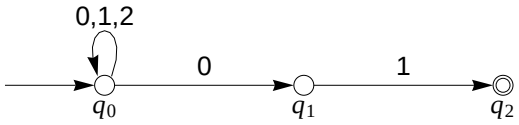
$$L_v = \{wv \mid w \in \Sigma^*\},$$

d.h., die Sprache der Wörter mit der Endung v*, und suchen einen Automaten, der diese Sprache erkennt (akzeptiert). Sei speziell* $\Sigma = \{0, 1, 2\}$ *und* $v = 01$ *und wir definieren den nichtdeterministischen endlichen Automaten* A_{01} *wie folgt:*

$$A_{01} = (\{q_0, q_1, q_2\}, \Sigma, \delta, \{q_0\}, \{q_2\}),$$

mit der Transition δ *gegeben durch*

$$\delta = \{(q_0, 0, q_0), (q_0, 1, q_0), (q_0, 2, q_0), (q_0, 0, q_1), (q_1, 1, q_2)\}.$$



Tatsächlich gilt der folgende Satz, der drei nichttriviale Sätze zusammenfasst.

Satz 3.25. *Es gilt, dass*

(A1) jede Typ 3-Sprache (d.h. jede rechtslineare/linkslineare/reguläre Sprache) endlich akzeptierbar ist;

(A2) jede endlich akzeptierbare Sprache deterministisch endlich akzeptierbar ist;

(A3) jede deterministisch endlich akzeptierbare Sprache eine Typ 3-Sprache ist.

Damit sind regulär, endlich akzeptierbar und deterministisch endlich akzeptierbar äquivalente *Begriffe.*

Wir werden diese Aussagen nicht beweisen, aber die einzelnen Punkte kurz diskutieren und an Beispielen illustrieren.

(A1) Sei $L \subseteq \Sigma^*$ eine rechtslineare Sprache, dann ist L endlich akzeptierbar.

Angenommen, dass L eine rechtslineare Sprache ist, dann existiert eine rechtslineare Grammatik $G = \{V, \Sigma, S, \Pi\}$, die L erzeugt, d.h., $L = L(G)$. Nach Lemma 3.16 können wir annehmen, dass die Produktionen in Π von der Form

$$A \to aB, \quad A \to a \quad \text{oder} \quad S \to \varepsilon$$

sind (mit $A, B \in V,\ a \in \Sigma$). Zu zeigen ist, dass dann ein nichtdeterministischer endlicher Automat existiert, der ein Wort genau dann akzpetiert, wenn es in L liegt. Ausgehend von G definieren wir den nichtdeterministischen endlichen Automaten $A(G) = (Q, \Sigma, \delta, Q_0, F)$ über dem gleichen Alphabet Σ durch:

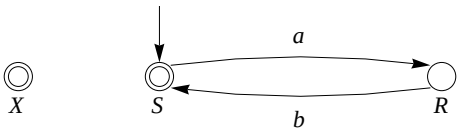
$$\begin{aligned} Q &= V \cup \{X\}, \quad \text{mit } X \notin V \\ \delta &= \{(A, a, B) \mid (A \to aB) \in \Pi\} \cup \{(A, a, X) \mid (A \to a) \in \Pi\} \\ Q_0 &= \{S\} \\ F &= \begin{cases} \{S, X\} & \text{falls } (S \to \varepsilon) \in \Pi \\ \{X\} & \text{sonst} \end{cases} \end{aligned}$$

Beispiel 3.26. *Sei $\Sigma = \{a, b\}$ und die Grammatik $G_1 = (\{S, R\}, \Sigma, S, \Pi_1\}$ durch die Erzeugungsregeln*

$$S \to \varepsilon \mid aR, \quad R \to bS,$$

gegeben. Dann ist der entsprechende NEA gegeben durch $Q = \{S, R, X\}$, $F = \{S, X\}$ mit der Transition δ_1 definiert als

$$X \to X, \quad S \xrightarrow{a} R, \quad R \xrightarrow{b} S.$$



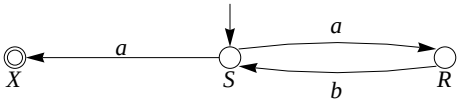
Die hier erzeugte Sprache ist $L(G_1) = \{(ab)^n \mid n \in \mathbb{N}\}$. Der Zustand X ist in diesem Automaten redundant und kann gelöscht werden (gemeinsam mit der entsprechenden Transition).

Beispiel 3.27. *Sei $\Sigma = \{a, b\}$ und die Grammatik $G_2 = (\{S, R\}, \Sigma, S, \Pi_2\}$ durch die Erzeugungsregeln*

$$S \to a \mid aR, \quad R \to bS,$$

gegeben. Dann ist der entsprechende NEA gegeben durch $Q = \{S, R, X\}$, $F = \{S, X\}$ mit der Transition δ_s definiert als

$$S \xrightarrow{a} X, \quad S \xrightarrow{a} R, \quad R \xrightarrow{b} S.$$



Die hier erzeugte Sprache ist $L(G_2) = \{(ab)^n a \mid n \in \mathbb{N}\}$.

(A2) (Rabin und Scott, 1959) Sei $L \subseteq \Sigma^*$ endlich akzeptierbar, dann ist L deterministisch endlich akzeptierbar.

Gegeben einen nichtdeterministischen endlichen Automaten $A = (Q, \Sigma, \delta, Q_0, F)$ mit $L = L(A)$ muss ein deterministischer endlicher Automat $B = (Q_B, \Sigma, \delta_B, q_{0B}, F_B)$ konstruiert werden, der die gleiche Sprache erzeugt, d.h., $L(A) = L(B)$.

Dazu wird die *Potenzmengenkonstruktion* verwendet: die Menge der Zustände von B ist die Potenzmenge der Zustände von A, d.h., $Q_B = P(Q)$. Für die Überleitungsrelation $\delta_B \colon Q_B \times \Sigma \to Q_B$ gilt: ein Tupel $(S, a) \in Q_B \times \Sigma$ wird einer Menge $R \in Q_B$ zugeordnet (d.h., $\delta_B(S, a) = R$), genau dann wenn

$$R = \{q' \in Q \mid \exists\, q \in S \colon \delta(q, a) = q'\}.$$

Der Anfangszustand $q_{0B} = Q_0$ und die Finalzustände sind alle Mengen, die Finalzustände von A enthalten, d.h.,

$$F_B = \{S \subseteq Q \mid S \cap F \neq \emptyset\}.$$

Damit ist der gesuchte DEA vollständig bestimmt. Die Zustände des neuen Automaten sind damit Mengen, die durch neue Bezeichnungen ersetzt werden können.

Beispiel 3.28. *(Fortsetzung von Suffix Beispiel 3.24) Zur Suffix-Erkennung hatten wir den NEA* $A_{01} = (Q = \{q_0, q_1, q_2\}, \Sigma = \{0, 1, 2\}, \delta, Q_0 = \{q_0\}, F = \{q_2\})$ *definiert mit*

$$\delta = \{(q_0, 0, q_0), (q_0, 1, q_0), (q_0, 2, q_0), (q_0, 0, q_1), (q_1, 1, q_2)\},$$

aus dem wir jetzt einen DEA $B = (Q_B, \Sigma, \delta_B, q_{0B}, F_B)$ *konstruieren. Zunächst bestimmen wir die Menge der Zustände, den Anfgangszustand und die Finalzustände:*

$$\begin{aligned} &Q_B = P(Q) = \{\emptyset, \{q_0\}, \{q_1\}, \{q_2\}, \{q_0, q_1\}, \{q_0, q_2\}, \{q_1, q_2\}, \{q_0, q_1, q_2\}\}, \\ &q_{0B} = \{q_0\}, \quad F_B = \{\{q_2\}, \{q_0, q_2\}, \{q_1, q_2\}, \{q_0, q_1, q_2\}\}. \end{aligned}$$

Zur Konstruktion von δ_B *betrachten wir ein paar einzelne Fälle, die volle Information findet sich dann unten im Graphen. Wir beginnen mit dem Bild von* $(\{q_0\}, 0)$*:*

$$\begin{aligned} \delta_B \colon \{q_0\} \xrightarrow{0} &\{q' \in Q \mid \exists\, q \in \{q_0\} \colon \delta(q, 0) = q'\} \\ &= \{q' \in Q \mid \delta(q_0, 0) = q'\} = \{q_0, q_1\}, \end{aligned}$$

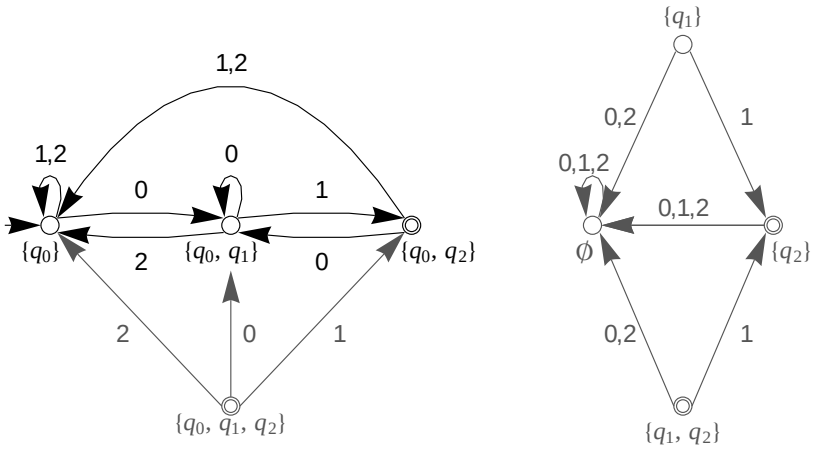
d.h., $\delta_B(\{q_0\}, 0) = \{q_0, q_1\}$*. Die leere Menge wird durch jedes* $a \in \Sigma$ *auf die leere Menge abgebildet. Wir betrachten noch zwei weitere Fälle:*

$$\delta_B \colon \{q_0, q_1\} \xrightarrow{1} \{q' \in Q \mid \exists\, q \in \{q_0, q_1\} \colon \delta(q, 1) = q'\} = \{q_0, q_2\},$$

da $\delta(q_0, 1) = q_0$ *und* $\delta(q_1, 1) = q_2$*.*

$$\delta_B \colon \{q_0, q_2\} \xrightarrow{0} \{q' \in Q \mid \exists\, q \in \{q_0, q_2\} \colon \delta(q, 0) = q'\} = \{q_0, q_1\},$$

hier kommen alle Beiträge von q_0*.*



Die im Graphen grau gezeichneten Teile des Automaten werden vom Anfangszustand nie erreicht und sind damit überflüssig. Der schwarze Teil ist der sprachäquivalente, minimale Teil der den gesuchten DEA angibt mit der Spezifikation

$$B_{min} = (\{\{q_0\}, \{q_0, q_1\}, \{q_0, q_2\}\}, \{0, 1, 2\}, \delta_{min}, \{q_0\}, \{q_0, q_2\}),$$

mit δ_{min} wie im Bild angegeben. Zur einfacheren Darstellung können zum Beispiel die folgenden Bezeichnungen für die Zustände eingeführt werden:

$$z_0 = \{q_0\}, \quad z_1 = \{q_0, q_1\}, \quad z_2 = \{q_0, q_2\},$$

und damit lässt sich der DEA schreiben als: $B_{min} = (\{z_0, z_1, z_2\}, \{0, 1, 2\}, \delta_{min}, z_0, z_2)$ mit δ_{min} gegeben durch

$$z_0 \xrightarrow{0} z_1, \quad z_0 \xrightarrow{1,2} z_0, \quad z_1 \xrightarrow{0} z_1, \quad z_1 \xrightarrow{1} z_2, \quad z_1 \xrightarrow{2} z_0, \quad z_2 \xrightarrow{0} z_1, \quad z_2 \xrightarrow{1,2} z_0.$$

(A3) Sei $L \subseteq \Sigma^*$ endlich akzeptierbar, dann ist L eine Typ 3-Sprache.

Dazu sei ein DEA $A = (Q, \Sigma, \delta, q_0, F)$ gegeben, mit $L = L(A)$. Aus diesem Automaten konstruieren wir die Grammatik $G = (V, \Sigma, S, \Pi)$ wie folgt:

$$\begin{aligned} V &= Q, \\ \Pi &= \{q \to aq' \mid (q, a, q') \in \delta\} \cup \{q \to \varepsilon \mid q \in F\}, \\ S &= q_0. \end{aligned}$$

Beispiel 3.29. *(Ungerade Anzahl von Einsern, Fortsetzung von Beispiel 3.20) In Beispiel 3.20 hatten wir die Sprache $L(A_1)$ durch den Automaten $A_1 = (\{q_0, q_1\}, \{0, 1\}, \delta_1, q_0, \{q_1\})$ mit Transition δ_1 gegeben durch*

$$q_0 \xrightarrow{0} q_0, \quad q_1 \xrightarrow{0} q_1, \quad q_0 \xrightarrow{1} q_1, \quad q_1 \xrightarrow{1} q_0,$$

definiert. Gemäss der obigen Konstruktion erhalten wir daraus die Grammatik $G_1 = (V, \Sigma, S, \Pi)$ mit $V = \{q_0, q_1\}$, $\Sigma = \{0, 1\}$, $S = q_0$ und der Erzeugungsvorschrift Π gegeben durch

$$\begin{aligned} q_0 &\longrightarrow 0q_0 \mid 1q_1 \\ q_1 &\longrightarrow 0q_1 \mid 1q_0 \mid \varepsilon \end{aligned}$$

Wir haben bis jetzt zwei verschiedene Arten kennengelernt, um eine Typ 3-Sprache anzugeben - über die erzeugende Grammatik oder den akzeptierenden Automaten. Eine dritte Art, ist den *regulären Ausdruck* (engl.: regular expression) für die Sprache anzugeben (daher auch der Name "reguläre Sprache"). Der reguläre Ausdruck gibt auch die Bildungsvorschrift an, verwendet aber nur *Terminalzeichen*, *oder* (als Zeichen dafür "|"), *beliebige Wiederholung* inklusive *null-mal* (als Zeichen dafür "*"), *beliebige Wiederholung* aber *mindestens einmal* (als Zeichen dafür $^+$) und *Klammerung*. Wir geben zwei Beispiele. Für die Sprache $L(A_{01})$ (Wörter mit Endung 01) lautet ein regulärer Ausdruck:

$$(0|1|2)^*01$$

Für die Sprache $L(A_1)$ (Wörter mit einer ungeraden Anzahl von 1ern):

$$0^*1(0^*10^*10^*)^*0^*.$$

Die Sprache, die eine ungerade Anzahl von 1 oder 2 erlaubt (d.h., die Sprache, die z.B. 0212, 02202, 111 akzeptiert), ist durch den regulären Ausdruck

$$0^*(1 \mid 2)\big(0^*(1 \mid 2)0^*(1 \mid 2)0^*\big)^*0^*$$

definiert.

3.3 Kellerautomaten und kontextfreie Sprachen

Endliche Automaten sind einfach handzuhaben und auch wenn reguläre Sprachen in vielen Anwendungen, wie zum Beispiel der Teilworterkennung vorkommen, so können sie unter anderem für Syntaxbeschreibungen nicht verwendet werden. Ein Grund dafür ist, dass Klammerstrukturen in dieser Anwendung beliebige Schachtelungstiefen vorkommen, wie z.B. für

- arithmetische Ausdrücke: $17 \cdot (x + 4 \cdot (x - 1))$
- Schleifen: While ... Do While ... Do ⟨body⟩ EndWhile EndWhile

Beispiel 3.30. *Ein Beispiel für kontextfreie Sprachen ist $L_2 = \{a^n b^n \mid n \in \mathbb{N}\}$, die durch die Grammatik $G_2 = (\{S\}, \{a, b\}, S, \Pi)$ mit Produktionen*

$$S \longrightarrow \varepsilon \mid aSb$$

erzeugt werden kann. Diese Grammatik ist weder rechts- noch linkslinear, aber kontextfrei.

Zur Erinnerung, eine Grammatik ist kontextfrei, falls für alle Produktionen (P, Q) gilt, dass $P \in V$ (ein Nichtterminalzeichen) und $Q \in (V \cup \Sigma)^*$.

Definition 3.31. *Eine Grammatik $G = (V, \Sigma, S, \Pi)$ heisst* eingeschränkt kontextfrei *(oder ε-frei), falls*

- *alle Produktionen kontextfrei sind, und*
- *es entweder keine ε-Produktion gibt, oder die einzige ε-Produktion ist $S \to \varepsilon$ und dann kommt S auf* keiner *rechten Seite einer Produktion vor. Im ersten Fall gilt $\varepsilon \notin L(G)$, im zweiten $\varepsilon \in L(G)$.*

Jede eingeschränkt kontextfreie Sprache (d.h. jede Sprache, die von einer eingeschränkt kontextfreien Grammatik erzeugt wird), ist natürlich auch kontextfrei. Tatsächlich gilt auch die Umkehr im folgenden Sinn.

Satz 3.32. *Sei $G = (V, \Sigma, S, \Pi)$ eine kontextfreie Grammatik. Dann kann aus G eine eingeschränkt kontextfreie Grammatik G_1 erzeugt werden mit $L(G) = L(G_1)$.*

Beweis. Wir beginnen damit die Nichtterminalzeichen V in zwei Mengen V_1 und $V_2 = V \backslash V_1$ zu zerlegen, wobei V_1 alle Variablen enthält, die zu ε führen. Dazu setzen wir zuerst $V_1 = \{X \in V \mid (X \to \varepsilon) \in \Pi\}$.

Solange noch X gibt mit $(X \to X_1 \cdots X_m) \in \Pi$ mit $X \notin V_1$, aber für alle $1 \leq j \leq m$ gilt $X_j \in V_1$, dann fügen wir X zu V_1 hinzu, d.h. $V_1 = V_1 \cup \{X\}$. Diese Konstruktion muss irgendwann abbrechen, da es nur eine endliche Anzahl von Nichtterminalzeichen und Produktionen gibt.

Die Variablen, die in V_2 liegen können das leere Wort nicht produzieren, weil jede Produktion entweder ein Terminal- oder ein Nichtterminalzeichen hinzufügt. Im zweiten Schritt müssen wir jetzt die Regeln verändern, um wieder die gleiche Sprache zu erzeugen:

1. Entferne alle Produktionen der Form $X \to \varepsilon$
2. Solange es noch Produktionen der Form $X \to uYv$ gibt mit $Y \in V_1$ und $uv \in (V \cup \Sigma)^+$ und es die Produktion $X \to uv$ noch nicht gibt, füge $(X \to uv)$ zu Π hinzu.

Die neuen Regeln sind kontextfrei und es gibt keine ε-Produktion. Dieser Konstruktionsschritt muss abbrechen, da die Länge der Regeln verkleinert werden. Falls $S \notin V_1$ gilt, dann ist die Konstruktion abgeschlossen. Sonst wird S zu einem normalen Nichtterminalzeichen und es wird ein neues Startsymbol S_0 eingeführt und die beiden Regeln $S_0 \to \varepsilon$ und $S_0 \to S$. □

Beispiel 3.33. *Weiter mit $L_2 = \{a^n b^n \mid n \in \mathbb{N}\}$ mit der Grammatik G_2 aus Beispiel 3.30 mit $\Pi = \{S \to \varepsilon, S \to aSb\}$. Diese Grammatik ist kontextfrei, aber nicht eingeschränkt kontextfrei. Wenn wir dem obigen Beweis folgen erhalten wir die Mengen*

$$V_1 = \{S\}, \quad V_2 = \emptyset, \quad \textit{und} \quad V = V_1 \cup V_2.$$

Bei der Regelproduktion wird $S \to \varepsilon$ entfernt und $S \to ab$ hinzugefügt. In diesem Fall muss ein neues Startsymbol S_0 eingeführt werden und insgesamt erhalten wir so die neue Grammatik

$$G_2^{(1)} = (\{S_0, S\}, \{a, b\}, S_0, \Pi^{(1)} = \{S_0 \to \varepsilon, S_0 \to S, S \to ab, S \to aSb\}).$$

Es gibt verschiedene Normalformen für kontextfreie Grammatiken, eine übliche ist wie folgt definiert.

Definition 3.34. *Eine kontextfreie Grammatik $G = (V, \Sigma, S, \Pi)$ ist in* Chomsky-Normalform *genau dann wenn*

(a) G ist eingeschränkt kontextfrei.

(b) Jede Produktion in Π ausser $S \to \varepsilon$ ist entweder von der Form $A \to a$ oder $A \to BC$ mit $a \in \Sigma$ und $A, B, C \in V$.

Jede kontextfreie Grammatik lässt sich in Chomsky-Normalform überführen. Wir wissen bereits, dass sich aus jeder kontextfreien Grammatik eine eingeschränkt kontextfreie Grammatik konstruieren lässt. Ausgehend davon können *Kettenregeln* $A \to B$ wie bei der Normalform für rechtslineare Grammatiken durch Einsetzen eliminiert werden.

Dann müssen Terminalzeichen getrennt werden. Dazu wird für jedes Terminalzeichen $\sigma \in \Sigma$ ein neues Nichtterminalzeichen V_σ eingeführt und die zusätzliche Regel $V_\sigma \to \sigma$.

Im letzten Schritt werden mehrelemenitge Nichtterminalketten schrittweise eliminiert durch Einführen neuer Nichtterminalzeichen, z.B. für $m \geq 2$:

$$\begin{aligned} & A \to A_1 A_2 A_3 \cdots A_m \quad \Rightarrow \quad A \to A_1 D_1,\ D_1 \to A_2 A_3 \cdots A_m \\ \Rightarrow \quad & A \to A_1 D_1,\ D_1 \to A_2 D_2, \ldots, D_{m-2} \to A_{m-1} A_m. \end{aligned}$$

Beispiel 3.35. *Die Grammatik $G_2^{(1)}$ aus Beispiel 3.33 können durch Einführen der neuen Nichtterminalzeichen A, B, T mit den folgenden Produktionen die Terminalzeichen ersetzt und Nichtterminalketten eliminiert werden:*

$$\begin{aligned} S_0 &\longrightarrow \varepsilon \mid S \\ S &\longrightarrow AB \mid AT \\ T &\longrightarrow SB \\ A &\longrightarrow a \\ B &\longrightarrow b \end{aligned}$$

Im letzten Schritt muss noch die Kettenproduktion $S_0 \to S$ durch Einsetzen eliminert werden und man erhält so die Chomsky-Normalform:

$$\begin{aligned} S_0 &\longrightarrow \varepsilon \mid AB \mid AT \\ S &\longrightarrow AB \mid AT \\ T &\longrightarrow SB \\ A &\longrightarrow a \\ B &\longrightarrow b \end{aligned}$$

Weiter mit L_2: Es existiert kein endlicher Automat, der L_2 akzeptiert. Der Grund dafür ist, dass es bei einem endlichen Automaten keine Möglichkeit gibt, die Anzahl der bereits gelesenen a's zu speichern - ausser durch die Zahl der Zustände. Da die Länge der Wörter aber nach oben unbeschränkt ist, kann so keine endlicher Automat definiert werden.

Definition 3.36. *Ein* nichtdeterministischer Kellerautomat *oder* Pushdown Automat *(kurz:* PDA*) ist ein Tupel $K = (Q, \Sigma, \Gamma, Z_0, \delta, q_0, F)$ mit*

- Q *einer* endlichen *Menge von Zuständen*
- Σ *dem Eingabealphabet der Maschine*
- Γ *dem Kelleralphabet*
- $Z_0 \in \Gamma$ *dem Startsymbol des Kellers*
- $\delta \subseteq (Q \times \Gamma \times (\Sigma \cup \{\square\})) \times (Q \times \Gamma^*)$ *der* Übergangsrelation *oder* Transitionsrelation

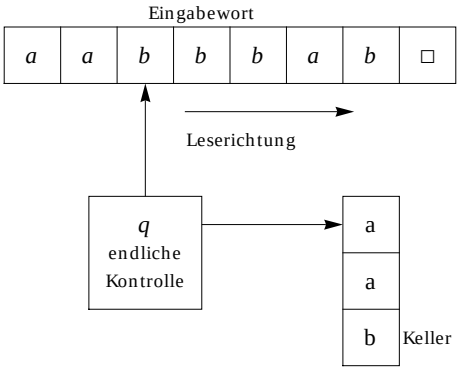


Abbildung 3.1: Skizze eines Kellerautomaten

- $q_0 \in Q$ *dem Anfangszustand*
- $F \subseteq Q$ *der Menge der Endzustände*

Für Transitionen verwenden wir ähnlich zu früher die Notation

$$(q_1, Z) \xrightarrow{\sigma} (q_2, B_1 \dots B_k)$$

wobei q_1 den aktuellen Zustand bezeichnet und Z das aktuell *oberste* Kellersymbol, das durch den Übergang *gelöscht* wird; $\sigma \in \Sigma$ ist das Zeichen, das beim Übergang gelesen wird, wobei $\sigma = \square$ das Ende des Wortes anzeigt, d.h., virtuell fügen wir jedem Wort am Ende das Zeichen $\square$ hinzu, um das Wortende anzuzeigen; q_1 ist der neue Zustand nach der Transition und $B_1 \dots B_k$ ist das *Wort*, das nach Löschen von Z in den Keller geschrieben wird, wobei B_1 das neue *oberste* Zeichen im Keller ist (falls $k \geq 1$).

Ein Wort wird von einem Kellerautomaten (mit leerem Keller) *akzeptiert*, wenn nach Abarbeitung des Wortes ein *Finalzustand erreicht* wird und der *Keller leer* ist.

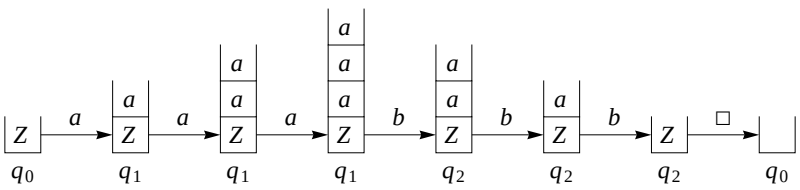
Beispiel 3.37. *(Kellerautomat zur Erkennung von L_2) Sei*

$$K_2 = (\{q_0, q_1, q_2\}, \{a, b\}, \{a, Z\}, Z, \delta, q_0, \{q_0\})$$

mit der Transitionsrelation δ:

$$(q_0, Z) \xrightarrow{a} (q_1, aZ)$$
$$(q_1, a) \xrightarrow{a} (q_1, aa)$$
$$(q_1, a) \xrightarrow{b} (q_2, \varepsilon)$$
$$(q_2, a) \xrightarrow{b} (q_2, \varepsilon)$$
$$(q_2, Z) \xrightarrow{\square} (q_0, \varepsilon)$$

Zum Beispiel wird das Wort aaabbb folgendermassen akzeptiert:



Beispiel 3.38. *Wir betrachten die (eingeschränkte) Palindromsprache erzeugt von der Grammatik*

$$G = (\{S\}, \{0,1\}, S, \Pi) \quad \textit{mit} \quad \Pi\colon S \longrightarrow 0 \mid 1 \mid 0S0 \mid 1S1.$$

(In der Sprache $L(G)$ liegen nur Palindrome von ungerader Wortlänge!) Diese Sprache wird von dem Kellerautomaten $K = (\{q_0, q_1\}, \{0,1\}, \{Z, 0, 1\}, \delta, q_0, \{q_0\})$ mit der folgenden Übergangsrelation akzeptiert:

$$\begin{aligned} (q_0, Z) &\xrightarrow{0} (q_0, 0Z) & (q_0, Z) &\xrightarrow{0} (q_1, Z) \\ (q_0, Z) &\xrightarrow{1} (q_0, 1Z) & (q_0, Z) &\xrightarrow{1} (q_1, Z) \end{aligned}$$

$$\begin{aligned} (q_0, 0) &\xrightarrow{0} (q_0, 00) & (q_0, 1) &\xrightarrow{0} (q_0, 01) \\ (q_0, 0) &\xrightarrow{1} (q_0, 10) & (q_0, 1) &\xrightarrow{1} (q_0, 11) \\ (q_0, 0) &\xrightarrow{0} (q_1, 0) & (q_0, 1) &\xrightarrow{1} (q_1, 1) \\ (q_0, 0) &\xrightarrow{1} (q_1, 0) & (q_0, 1) &\xrightarrow{0} (q_1, 1) \end{aligned}$$

$$\begin{aligned} (q_1, 0) &\xrightarrow{0} (q_1, \varepsilon) \\ (q_1, 1) &\xrightarrow{1} (q_1, \varepsilon) \end{aligned}$$

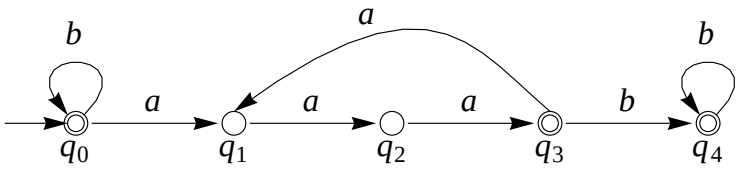
$$(q_1, Z) \xrightarrow{\square} (q_0, \varepsilon)$$

Man beachte, dass diese Relation tatsächlich nichtdeterministisch *ist. Im Gegensatz zu regulären Sprachen gilt für kontextfreie Sprache* nicht *die Äquivalenz von deterministisch und nichtdeterministisch.*

Ohne Beweis notieren wir die Äquivalenz von Kellerautomaten und kontextfreien Grammatiken.

Satz 3.39. *Zu jeder kontextfreien Grammatik G kann man einen* nichtdeterministischen *Kellerautomaten K generieren, der Wörter genau dann akzeptiert, wenn sie in $L(G)$ liegen. Umgekehrt, kann man zu jedem Kellerautomaten K eine kontextfreie Grammatik G konstruieren, sodass $L(G)$ genau jene Wörter enthält, die von K akzeptiert werden.*

Beispiel 3.40. *Sei $L_1 = \{b^m(aaa)^n b^k \mid m, n, k \in \mathbb{N}\}$. Diese Sprache ist regulär und wird von dem endlichen Automaten*



Beispiel 3.41. *Sei $L_2 = \{b^m(aaa)^n b^{2m} \mid m, n \in \mathbb{N}\}$. Diese Sprache ist kontextfrei, aber nicht regulär. Sie wird von zum Beispiel von dem Kellerautomaten mit folgender Übergangsrelation akzeptiert: In der ersten Spalte stehen die Übergänge für zulässige Wörter, bei denen alle drei*

Blöcke vorkommen (ausser dem leeren Wort in der obersten Zeile), in der zweiten Spalte stehen die Übergänge für zulässige Wörter, die nur aus a bestehen und in der dritten Spalte jene für die zulässigen Wörter, die nur aus b bestehen:

$$
\begin{array}{lll}
(q_0, Z) \xrightarrow{\square} (q_f, \varepsilon) & & \\
(q_0, Z) \xrightarrow{b} (q_1, bbZ), & (q_0, Z) \xrightarrow{a} (q_{a1}, Z) & (q_0, Z) \xrightarrow{b} (q_{b1}, Z) \\
(q_1, b) \xrightarrow{b} (q_1, bbb), & (q_{a1}, Z) \xrightarrow{a} (q_{a2}, Z) & (q_{b1}, Z) \xrightarrow{b} (q_{b2}, Z) \\
(q_1, b) \xrightarrow{a} (q_{a1}, b) & (q_{a2}, Z) \xrightarrow{a} (q_{a3}, Z) & (q_{b2}, Z) \xrightarrow{b} (q_{b3}, Z) \\
(q_{a1}, b) \xrightarrow{a} (q_{a2}, b) & (q_{a3}, Z) \xrightarrow{a} (q_{a1}, Z) & (q_{b3}, Z) \xrightarrow{b} (q_{b1}, Z) \\
(q_{a2}, b) \xrightarrow{a} (q_{a3}, b) & (q_{a3}, Z) \xrightarrow{\square} (q_f, \varepsilon) & (q_{b3}, Z) \xrightarrow{\square} (q_f, \varepsilon) \\
(q_{a3}, b) \xrightarrow{a} (q_{a1}, b) & & \\
(q_{a3}, b) \xrightarrow{b} (q_2, \varepsilon) & & \\
(q_2, b) \xrightarrow{b} (q_2, \varepsilon) & & \\
(q_2, Z) \xrightarrow{\square} (q_f, \varepsilon) & &
\end{array}
$$

Die Zustände können so gelesen werden: q_0 ist der Anfangszustand, q_1 ist der Zustand in dem der erste b-Block gelesen wird, q_{a1} bedeutet es wurde ein a gelesen, q_{a2} bedeutet es wurden zwei a gelesen (analog für b), ..., q_2 ist der Zustand, in dem die b aus dem Keller gelöscht werden und q_f der Finalzustand.

Alternativ kann statt im ersten Block für jedes gelesene b zwei Zeichen im Keller zu markieren (durch das Hinausschreiben von bb) auch nur eines abgespeichert werden. In dem Fall darf man beim hinteren Block nur bei jedem zweiten gelesenen b ein Zeichen aus dem Keller löschen.

Die Zeichen im Keller müssen nicht die gleichen sein, die im Grundalphabet liegen (es gibt ein eigenes Kelleralphabet). Man könnte also statt sich "b" zu merken auch andere Symbole in den Keller schreiben.

Für Grammatiken in *Chomsky-Normalform* existiert ein einfacher Algorithmus um zu entscheiden ob ein gegebenes Wort in der von der Grammatik erzeugten Sprache liegt, der als *CYK-Algorithms* bekannt ist nach *Cocke, Younger* und *Kasami*.

Wenn eine Grammatik $G = (V, \Sigma, S, \Pi)$ in Chomsky-Normalform ist, dann kann das leere Wort nur in $L(G)$ liegen, wenn es eine Erzeugung $S \to \varepsilon$ gibt. Betrachten wir also ab jetzt nur nichtleere Wörter $w = x_1 x_2 \ldots x_n$. Die können nur aus Regeln der Form $A \to a$ (Abbildung auf ein Terminalzeichen) oder $A \to BC$ (Abbildung auf zwei Teilwörter) gebildet werden. Im CYK-Algorithmus werden alle möglichen Kombinationen für diese Teilwortbildung ausgehend vom gegebenen Wort w in einer Tabelle nach den Teilwortlängen aufgelistet. Wenn an der Stelle des Worts mit voller Länge n als mögliche Ausgangsposition das Startsymbol S vorkommt, dann kann das Wort in der gegebenen Grammatik erzeugt werden, sonst nicht.

CYK-Algoritmus Input: kontextfreie Grammatik $G = (V, \Sigma, S, \Pi)$ in Chomsky-Normalform und ein Wort $w = x_1 x_2 \ldots x_n \in \Sigma^+$; Output: $w \in L(G)$

1. Initialisiere den $n \times n$-Array cyk mit Einträgen $\emptyset$

2. Erste Zeile:
   For $i = 1, i \leq n, i++$ Do $\text{cyk}[i, 1] = \{A \mid (A \to x_i) \in \Pi\}$

3. Restliche Zeilen:
   For $j = 2, j \leq n, j++$ Do
   For $i = 1, i \leq n + 1 - j, i++$ Do
   For $k = 1, k \leq j - 1, k++$ Do
   $\text{cyk}[i, j] = \text{cyk}[i, j] \cup \{A \mid \exists\, B, C \colon (A \to BC) \in \Pi \wedge B \in \text{cyk}[i, k] \wedge C \in \text{cyk}[i + k, j - k]\}$

4. Return $S \in \text{cyk}[1, n]$

Zu einem Eingabewort der Länge n benötigt der Algorithmus die Größenordnung von n^3 Schritten.

Beispiel 3.42. *Wir betrachten die Grammatik $G_2^{(1)}$ zur Erzeugung von L_2 in Chomsky-Normalform (siehe Beispiel 3.35). Wir beschränken uns auf nichtleere Wörter, d.h., wir betrachten nur die Produktionen*

$$S \to AB \mid AT, \quad T \to SB, \quad A \to a, \quad B \to b.$$

Zu $w = aaabbb$ resultiert eine Anwendung des CYK-Algorithmus in der folgenden Tabelle:

	$i \longrightarrow$						
w	a	a	a	b	b	b
$j = 1$	A	A	A	B	B	B
$j = 2$			S			
$j = 3$			T			
$j = 4$		S				
$j = 5$		T				
$j = 6$	S					

Das heisst das Wort liegt in der Sprache. Jetzt ein Durchlauf für $x = aabba$:

	$i \longrightarrow$				
x	a	a	b	b	a
$j = 1$	A	A	B	B	A
$j = 2$		S			
$j = 3$		T			
$j = 4$	S				
$j = 5$					

Dieses Wort ist nicht Element von L_2.

3.4 Turingmaschinen und kontextsensitive Sprachen

Turingmaschinen wurden ca. 1936 von Alan M. Turing entwickelt um *Berechenbarkeit* zur charakterisieren. Diesen Aspekt von Turingmaschinen werden wir im nächsten Abschnitt betrachten. Zuerst verwenden wir sie als *Sprachakzeptoren*, d.h., um festzustellen ob ein gegebenes

Wort Teil einer vorgegebenen Sprache ist. Mit Turingmaschinen können Sprachen charakterisiert werden, die über reguläre und kontextfreie Sprachen hinausgehen.

In Beispiel 3.15 wurde behauptet, dass

$$L_1 = \{a^n b^n c^n \mid n \in \mathbb{N}^*\}$$

eine kontextsensitive Sprache ist (der Einfachheit halber entfernen wir das leere Wort aus der Sprache). Das wollen wir jetzt näher untersuchen.

Beispiel 3.43. *Zunächst definieren wir L_1 durch die Grammatik G mit $V = \{B, S\}$, $\Sigma = \{a, b, c\}$, Startsymbol S und Produktionen Π gegeben durch:*

$$S \longrightarrow aSBc \mid abc, \quad cB \longrightarrow Bc, \quad bB \longrightarrow bb.$$

Diese Produktionsregeln sind zwar intuitiv das was wir suchen, aber die mittlere dieser drei Ableitungsregeln ist nicht kontextsensitiv. Das Vertauschen der Reihenfolge von zwei Nichtterminalzeichen *kann durch eine Hintereinanderausführung von kontextsensitiven Produktionen ausgedrückt werden. Zum Beispiel kann die Produktion $CB \longrightarrow BC$ erreicht werden durch zusätzliche Variablen X, Y und*

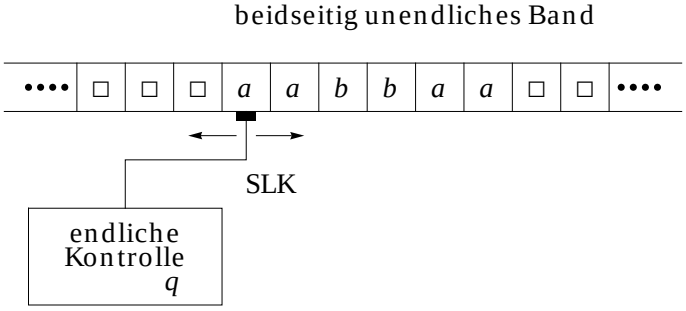
$$\begin{aligned} CB &\longrightarrow XB, \\ XB &\longrightarrow XY, \\ XY &\longrightarrow BY, \\ BY &\longrightarrow BC. \end{aligned}$$

Indem wir neue Nichtterminalzeichen C, X, Y einführen erhalten wir eine kontextsensitive *Grammatik, die L_1 erzeugt, also $L_1 = L(G_1)$ mit $G_1 = (\{B, C, X, Y, S\}, \{a, b, c\}, S, \Pi_1)$ mit Π_1 gegeben durch*

$$\begin{aligned} S &\longrightarrow aSBC \mid abC, \\ CB &\longrightarrow XB, \\ XB &\longrightarrow XY, \\ XY &\longrightarrow BY, \\ BY &\longrightarrow BC, \\ bB &\longrightarrow bb, \\ bC &\longrightarrow bc, \\ cC &\longrightarrow cc. \end{aligned}$$

Ersetzungsregeln wie in diesem Beispiel beschrieben, die die Länge eines Wortes zumindest nicht verkürzen, werden auch *monoton* genannt. Allgemein gilt, dass jede monotone Produktion in eine Reihe von kontextsensitiven Produktionen überführt werden kann.

Eine Turingmaschine besteht aus einem *unendlichen Band*, d.h., einem Band mit unendlich vielen Feldern. Auf diesen Feldern steht je ein Zeichen, wobei nur auf *endlichen* vielen Feldern nicht das *Bandzeichen* $\square$ steht (dieses Zeichen steht für ein unbeschriftetes Feld). Ausserdem gibt es einen *Schreib- und Lesekopf* (kurz *SLK*), der immer auf einem bestimmten Feld des Bands (der Turingmaschine) steht. Die *Steuereinheit* (*endliche Kontrolle*) ist immer in einem bestimmten Zustand.



Ein Schritt der Maschine besteht aus vier Teilen:

- der SLK liest das Zeichen im aktuellen Feld
- der SLK schreibt ein Zeichen auf das aktuelle Feld
- der SLK bewegt sich nach links (L), rechts (R) oder bleibt stehen (N)
- die Kontrolle (Steuereinheit) geht in den nächsten Zustand über (der auch der gleiche wie der aktuelle sein kann)

Definition 3.44. *Eine Turingmaschine ist ein Tupel* $M = (Q, \Sigma, \Gamma, \square, \delta, q_0, q_f)$ *mit*

- Q *einer* endlichen *Menge von Zuständen;*
- Σ *dem Alphabet der Maschine ("die nach aussen sichtbaren Zeichen");*
- Γ *dem* Bandalphabet *der Maschine (Zeichen, die als Feldbeschriftungen verwendet werden dürfen);* $\square$ *ist das* Bandzeichen *und es gilt, dass* $\Gamma \supseteq \Sigma \cup \{\square\}$*; die Zeichen in* $\Gamma \backslash (\Sigma \cup \{\square\})$ *werden auch* Hilfszeichen *oder* Sonderzeichen *genannt;*
- $\delta \subseteq (Q \times \Gamma) \times (Q \times \Gamma \times \{L, N, R\})$ *ist die Übergangsrelation;*
- q_0 *ist der* Anfangszustand *und* q_f *der* Endzustand*;*

Falls die Übergangsrelation δ *eine (totale) Funktion ist, dann ist* M *eine* deterministische Turingmaschine *(kurz* DTM*), andernfalls eine* nichtdeterministische Turingmaschine *(kurz* NTM*).*

Ein Berechnungszustand (*Konfiguration*) einer Turingmaschine kann man beschreiben durch ein Wort $\alpha q \beta$ mit $\alpha, \beta \in \Gamma^+, q \in Q$ wobei

- q ist der momentane Zustand
- α ist die Beschriftung des Bandes links vom aktuellen Feld (dabei werden unendlich viele Bandzeichen weggelassen)
- β ist die Beschriftung des Bandes rechts vom aktuellen Feld (dabei werden unendlich viele Bandzeichen weggelassen)

Die aktuelle Konfiguration im obigen Bild einer Turingmaschine ist also $\square qabbaa\square$ oder auch $\square qabbaa$ oder $\square qabbaa\square\square$.

q_0	a	q_b	x	R
q_b	a	q_b	a	R
q_b	y	q_b	y	R
q_b	b	q_c	y	R
q_c	b	q_c	b	R
q_c	z	q_c	z	R
q_c	c	q_E	z	R
q_c	c	q_Z	z	L
q_Z	z	q_Z	z	L
q_Z	b	q_Z	b	L
q_Z	y	q_Z	y	L
q_Z	a	q_Z	a	L
q_Z	x	q_0	x	R
q_E	$\square$	q_K	$\square$	L
q_K	z	q_K	z	L
q_K	y	q_K	y	L
q_K	x	q_K	x	L
q_K	$\square$	q_f	$\square$	N

Tabelle 3.1: Turingtafel zu Beispiel 3.46

Definition 3.45. *Ein Wort $w \in \Sigma^*$ liegt in der von einer gegebenen Turingmaschine M akzeptierten Sprache $L(M)$, wenn ausgehend von der Anfangskonfiguration $\square q_0 w \square$ durch die Übergangsrelationen der Endzustand q_f erreicht werden kann. Zwei Turingmaschinen M_1 und M_2 heissen* sprachäquivalent*, wenn $L(M_1) = L(M_2)$.*

In dieser Definition muss nicht zwischen einer DTM und einer NTM unterschieden werden, da zu jeder nichtderministischen Turingmaschine eine *sprachäquivalente* deterministische Turingmaschine gefunden werden kann.

Beispiel 3.46. *Die Sprache $L_1 = \{a^n b^n c^n \mid n \in \mathbb{N}^*\}$ ist Turing-akzeptierber. Die akzeptierende Turingmaschine geht wie folgt vor:*

- *das erste a wird durch ein x ersetzt, dann wird das erste b gesucht und durch ein y ersetzt und dann das erste c durch ein z*
- *man geht zurück zum zweiten a, ersetzt es durch ein x, läuft zum zweiten b, ersetzt es durch ein y, läuft zum zweiten c und ersetzt es durch ein z*
- *diese Schritte werden wiederholt solange, bis nach dem erzeugten z ein $\square$ steht*
- *dann wird überprüft, ob es nur noch einen z-Block, gefolgt von einem y-Block, gefolgt von einem x-Block gibt (abgeschlossen mit einem Bandzeichen); in dem Fall wird das Wort von der Maschine akzeptiert*
- *die Maschine* blockiert*, wenn das nicht eintritt, oder schon vorher, falls das erwartete Zeichen (a, b, c) nicht gefunden wird*

Die akzeptierende Turingmaschine $M_1 = (Q, \Sigma, \Gamma, \square, \delta, q_0, q_f)$ *ist gegeben durch:* $\Sigma = \{a, b, c\}$, $\Gamma = \{a, b, c, x, y, z, \square\}$, $Q = (q_0, q_b, q_c, q_Z, q_K, q_E, q_f)$. *Hier ist*

- q_0*: Anfangszustand und immer der Zustand in dem ein a durch ein x überschrieben wird*
- q_b*: suche b*
- q_c*: suche c*
- q_E*: c gefunden, Ende des Bands erreicht?*
- q_Z*: gehe zurück*
- q_K*: kontrolliere ob alle Blöcke korrekt vorhanden*
- q_f*: Finalzustand*

Die Übergangsrelation $\delta \subseteq (Q \times \Gamma) \times (Q \times \Gamma \times \{L, N, R\})$ geben wir mittels einer *Turingtafel* an. Dabei werden die Komponenten $(q_{akt}, a, q_{neu}, b, X)$ für jedes Element in δ eingetragen.

Ohne Beweis schliessen wir die Klassifizierung der formalen Sprachen ab.

Satz 3.47. *Sei* Σ *ein Alphabet und* $L \subseteq \Sigma^*$. *Dann ist L vom Typ Chomsky-0, genau dann wenn L Turing-akzeptierbar ist.*

Zur Charakterisierung von Typ 1-Sprachen benötigt man eine Unterklasse von NTM, bei der sich die Maschine nie über die linke oder rechte Grenze des Eingabeworts hinausbewegt. Diese speziellen NTMs werden *linear beschränkte Automaten* genannt und es gilt:

Satz 3.48. *Sei* Σ *ein Alphabet und* $L \subseteq \Sigma^*$. *Dann ist L vom Typ Chomsky-1, genau dann wenn L von einem linear beschränkten Automaten akzeptiert wird.*

Es gibt verschiedene Varianten von Turingmaschinen, die alle äquivalent zueinander sind. Ein Beispiel dafür sind Turingmaschinen mit mehreren Bändern und Schreib-Leseköpfen.

Definition 3.49. *Eine* k-Band-NTM *hat die Form* $M_k = (Q, \Sigma, \Gamma, \square, \delta, q_0, q_f)$ *mit*

- $Q, \Sigma, \Gamma, \square, q_0, q_f$ *wie in Definition 3.44*
- $\delta \subseteq (Q \times \Gamma^k) \times (\Gamma^k \times Q \times \{L, N, R\}^k)$

Hier bedeutet $(q, (a_1, a_2, \ldots, a_k), (b_1, b_2, \ldots, b_k), \bar{q}, (d_1, d_2, \ldots, d_k)) \in \delta$:

- die Steuereinheit (endliche Kontrolle) ist aktuell im Zustand q
- auf den k Bändern zeigt der SLK aktuell auf die Felder mit Beschriftung $a_1, a_2, \ldots, a_k$

und von diesem Zustand aus kann M_k

- das Zeichen a_j auf dem j-ten Band durch b_j ersetzen;
- die Steuereinheit geht in den Zustand $\bar{q}$ über;
- die SLKs der einzelnen Bänder bewegen sich gemäss d_j.

Üblicherweise wird das erste Band als Ein- und Ausgabeband verwendet. Man beachte, dass sich die SLKs verschiedener Bänder in verschiedene Richtungen bewegen können. Wären die Köpfe gekoppelt, dann hätte man ein Band mit mehreren Spuren. Das kann einfach durch eine 1-Band NTM über dem Bandalphabet Γ^k beschrieben werden.

Aber auch für den allgemeinen Fall gilt, dass zu einer k-Band NTM eine sprachäquivalente 1-Band NTM konstruiert werden kann.

4 Berechenbarkeit

Churchsche These: Die Klasse der intuitiv berechenbaren Funktionen ist genau die Klasse der Turing-berechenbaren Funktionen

In diesem Abschnitt wollen wir die Begriffe *Berechenbarkeit* und *Entscheidbarkeit* formalisieren. Um für eine gegebene Abbildung

$$f\colon \mathbb{N}^k \to \mathbb{N} \quad (\text{bzw. } f\colon (\Sigma^*)^k \to \Sigma^*)$$

festzustellen, ob sie berechenbar ist genügt (intuitiv) ein Berechnungsverfahren (einen *Algorithmus*) anzugeben. Dabei werden wir neben Funktionen wie in Definition 2.47 eingeführt auch einen schwächeren Funktionsbegriff benützen, bei dem nicht jeder Eingabe eine Ausgabe zugeordnet wird (falls eine Ausgabe zugeordnet wird ist sie allerdings immer noch *eindeutig bestimmt*):

Definition 4.1. *Seien X, Y Mengen. Eine* (totale) Funktion *$f \subseteq X \times Y$ ist eine Vorschrift, die* jedem *$x \in X$* genau ein *$y \in Y$ zuordnet. Das zu $x \in X$ eindeutig bestimmte $y \in Y$ wird mit $y = f(x)$ bezeichnet.*

Eine partielle Funktion *$f \subseteq X \times Y$ ist eine Vorschrift mit der Eigenschaft, dass für alle $x \in X$ mit $f(x) = y$ und $f(x) = z$ (d.h., $(x, y) \in f$ und $(x, z) \in f$) gilt: $y = z$.*

Sei $f \subseteq X \times Y$ eine partielle Funktion. Dann ist der Vorbereich *(engl.* domain*) von f definiert als*

$$\mathrm{dom}(f) = \{x \in X \mid \exists\, y \in Y : f(x) = y\}.$$

Jede totale Funktion ist damit auch eine partielle Funktion, aber nicht umgekehrt.

Beachte, dass Nichtberechenbarkeit im Gegensatz zur Berechenbarkeit schwieriger zu verifizieren ist, da man zeigen muss, dass *kein Algorithmus* zur Berechenbarkeit *existiert*. Im folgenden werden wir uns bei Turingmaschinen auf *deterministische* TM beschränken. Wenn wir eine TM als Berechner einer (partiellen) Funktion f betrachten, dann

- ist die Bandbeschriftung zu Beginn $(x_1, \ldots, x_k)$ (die Argumente von f)
- falls die Funktion f auf $(x_1, \ldots, x_k)$ definiert ist, dann gibt es genau eine terminierende Berechnung und der Bandinhalt im Endzustand enthält $f(x_1, \ldots, x_k)$
- falls die Funktion f auf $(x_1, \ldots, x_k)$ nicht definiert ist, so kreist die TM

Definition 4.2. *Die (partielle) Funktion $f\colon (\Sigma^*)^k \to \Sigma^*$ (mit $k \in \mathbb{N}$) heisst* Turing-berechenbar*, wenn es eine DTM $M = (Q, \Sigma, \Gamma, \Box, \delta, q_0, q_f)$ gibt, die genau dann den Endzustand q_f erreicht mit Bandbeschriftung $y = f(x_1, \ldots, x_k)$, wenn $(x_1, \ldots, x_k) \in \mathrm{dom}(f)$. In der Endkonfiguration steht der SLK auf dem ersten Feld des Ergebnisses und von dort bis zum ersten Bandzeichen steht nur die Ausgabe der Funktion.*

Um einfache Beispiele zu betrachten werden wir das *binäre Zahlensystem* einführen. Das Dezimalsystem (nur zur Erinnerung) stellt Zahlen zur Basis 10 dar mit Ziffern $0, 1, 2, 3, 4, 5, 6, 7, 8, 9$ in der Form

$$15679 = 1 \cdot 10^4 + 5 \cdot 10^3 + 6 \cdot 10^2 + 7 \cdot 10^1 + 9 \cdot 10^0 = (15679)_{10}.$$

Die Bezeichnung $(15679)_{10}$ wird hier nur verwendet um klarzustellen in welcher Basis wir arbeiten, üblicherweise werden wir das weglassen.

Die Ziffern der Darstellung einer Zahl im Dezimalsystem können durch wiederholte Division durch 10 mit Rest ermittelt werden, d.h., zu $x \in \mathbb{N}$ werden der Quotient $q \in \mathbb{N}$ und der Rest $r \in \mathbb{N}$ mit $x = 10q + r$ und $0 \leq r \leq 9$ berechnet, so lange bis $q = 0$ ist, zum Beispiel:

$$427 = 10 \cdot 42 + 7, \quad 42 = 10 \cdot 4 + 2, \quad 4 = 10 \cdot 0 + 4 \quad \text{also} \quad 427 = 4 \cdot 10^2 + 2 \cdot 10^1 + 7 \cdot 10^0.$$

Im Binärsystem werden Zahlen zur Basis 2 dargestellt unter Verwendung der Ziffern $0, 1$, zum Beispiel

$$\begin{aligned}(15)_{10} &= 1 \cdot 2^3 + 1 \cdot 2^2 + 1 \cdot 2^1 + 1 \cdot 2^0 = 1111 = (1111)_2\\ (20)_{10} &= 1 \cdot 2^4 + 0 \cdot 2^3 + 0 \cdot 2^1 + 1 \cdot 2^0 = 1001 = (1001)_2\\ (15679)_{10} &= (11111100101111)_2\end{aligned}$$

Die Ziffern der Darstellung einer Zahl im Binärsystem werden analog durch wiederholte Division mit Rest durch die Basis 2 gewonnen:

$$14 = 2 \cdot 7 + 0, \quad 7 = 2 \cdot 3 + 1, \quad 3 = 2 \cdot 1 + 1, \quad 1 = 2 \cdot 0 + 1,$$

also

$$14 = 1 \cdot 2^3 + 1 \cdot 2^2 + 1 \cdot 2^1 + 0 \cdot 2^0 = (1110)_2.$$

Jede natürliche Zahl $b > 1$ kann als Basis verwendet werden mit Ziffern $0, 1, \ldots, b-1$ und die Darstellung zur Basis b wird analog wie oben bestimmt.

Beispiel 4.3. *Wir bestimmen eine DTM, die die Nachfolgefunktion (s für* successor*)*

$$s \colon \mathbb{N} \to \mathbb{N}, \quad x \mapsto x + 1$$

berechnet, wobei wir die natürlichen Zahlen im Binärsystem über dem Alphabet $\{0, 1\}$ *codieren.*

1. *lese bis ans Ende der gegebenen Zahl* x *(Zustand* q_0*: es wird nur gelesen (von links nach rechts) und wieder zurück gestellt)*
2. *addiere* 1 *zur letzten Ziffer*
3. *falls ein Übertrag entsteht wird der zur nächsten Ziffer weiter getragen, usw. (Zustand* q_1*: es wird* 1 *addiert und ein Übertrag nach links mitgenommen; Zustand* q_2*: der Übertrag ist abgearbeitet, es wird nur mehr von rechts nach links gelesen und die Zeichen werden wieder zurückgestellt)*
4. *in der Endposition zeigt der SLK auf die erste Ziffer von* $x + 1$

Die Turingtafel, die diese Übergangsrelation darstellt ist in Tabelle 4.1 angegeben. Sei $x = 1111$*, dann sehen die Konfigurationsübergänge eines Durchlaufs zur Berechnung von* $y = x + 1$ *wie in Tabelle 4.2 aus.*

q_0	0	q_0	0	R
q_0	1	q_0	1	R
q_0	$\square$	q_1	$\square$	L
q_1	0	q_2	1	L
q_1	1	q_1	0	L
q_1	$\square$	q_f	1	N
q_2	0	q_2	0	L
q_2	1	q_2	1	L
q_2	$\square$	q_f	$\square$	R

Tabelle 4.1: Turingtafel zu Beispiel 4.3

	x=	1	1	1	1		
	$\square$	q_0	1	1	1		
		1	q_0	1	1		
		1	1	q_0	1		
		1	1	1	q_0	$\square$	
		1	1	1	1	q_0	$\square$
		1	1	1	q_1	$\square$	
		1	1	q_1	0		
		1	q_1	0	0		
	$\square$	q_1	0	0	0		
$\square$	q_1	0	0	0	0		
$\square$	q_f	0	0	0	0		
$y =$	1	0	0	0	0		

Tabelle 4.2: Konfigurationsübergänge zu Beispiel 4.3

Neben Turing-Berechenbarkeit kann man auch andere Arten der Berechenbarkeit einführen, wie zum Beispiel die WHILE-Berechenbarkeit: dazu wird eine Teilklasse von Programmen definiert mit starken syntaktischen Einschränkungen wie etwa

- es werden nur Variablen, die Werte aus den natürlichen Zahlen annehmen zugelassen;
- es werden nur einfache Zuweisungen auf eine Konstante $c \in \mathbb{N}$, eine Variable x oder den Wert $x + 1$ zugelassen;
- an Booleschen Ausdrücken werden nur **true**, **false**, $x = c$, $x = y$, $x \neq c$, $x \neq y$ zugelassen;
- nur das WHILE-Kommando wird als Schleifenkommando zugelassen.

Eine Funktion wird dann analog zu Turing-Berechnbarkeit als WHILE-berechenbar bezeichnet, wenn ein WHILE Programm existiert, das die Funktion im Sinn von Definition 4.2 berechnet. Die Begriffe der Turing-Berechenbarkeit und WHILE-Berechenbarkeit sind *äquivalent*. Ein WHILE-Programm kann durch eine Mehrband-Turingmaschine simuliert werden, indem das Band i der Variable x_i entspricht (z.B. in Binärdarstellung beschrieben). Jede Mehrband-Turingmaschine kann wiederum durch eine Einband-Turingmaschine simuliert werden, damit ist jede WHILE-berechenbare Funktion auch Turing-berechenbar.

Umgekehrt, kann zu einer gegebenen Turing-Maschine ein WHILE-Programm konstruiert werden, das die Maschine simuliert. Dazu müssen die Konfigurationen binär (oder in einem anderen Zahlensystem) kodiert werden. Zustandsübergänge und Bewegungen des SLK können dann durch arithmetische Operationen ausgedrückt werden, Lese- und Schreibvorgänge durch Variablenzuweisungen. Zusammenfassend gilt folgender Satz:

Satz 4.4. *Eine Funktion ist genau dann Turing-berechenbar, wenn sie WHILE-berechenbar ist.*

Aus diesem Grund ist es keine Einschränkung wenn wir im folgenden Berechenbarkeit mit Turing-Berechenbarkeit gleich setzen. Als nächstes betrachten wir *Entscheidbarkeit*. Entscheidbarkeit kann als spezielle Form von Berechenbarkeit gesehen werden, wobei wir wieder n Eingabeparameter haben und einen Ausgabeparameter, der vom Typ Boolean ist, d.h., der Ausgabewert ist entweder 0 oder 1 (bzw. true oder false). In diesem Zusammenhang kehren wir auch wieder zu formalen Sprachen zurück und betrachten das *Wortproblem*, d.h., gegeben eine Sprache $L \subseteq \Sigma^*$ über einem Alphabet Σ und gegeben ein Wort $w \in \Sigma^*$ ist zu *entscheiden* ob $w \in L$ gilt. Im folgenden bezeichnet Σ immer ein gegebenes Alphabet.

Definition 4.5. *Eine Sprache $L \subseteq \Sigma^*$ heisst* semi-entscheidbar *genau dann wenn ein Algorithmus existiert, der für jedes Wort $w \in \Sigma^*$ anhält, falls $w \in L$ und andernfalls kreist.*

Da wir Berechenbarkeit über Turingmaschinen definiert haben und bereits gezeigt haben, dass WHILE-Berechenbarkeit und Turing-Berechenbarkeit äquivalent sind, kann die Definition auch so gelesen werden, dass eine Sprache $L \subseteq \Sigma^*$ semi-entscheidbar ist genau dann wenn eine Turing-Maschine existiert, die für jedes Wort $w \in \Sigma^*$ mit Ausgabewert 1 (oder true) terminiert, falls $w \in L$ und andernfalls kreist. Mit anderen Worten, eine Sprache ist semi-entscheidbar, genau dann wenn sie *Turing-akzeptierbar* ist.

Definition 4.6. *Eine Sprache $L \subseteq \Sigma^*$ heisst* entscheidbar *genau dann wenn ein Algorithmus (eine Turingmaschine) existiert, der (die) mit dem Ergebnis* 1 *anhält, falls $w \in L$ und sonst mit dem Ergebnis* 0 *anhält.*

Um den Zusammenhang zu Berechenbarkeit noch deutlicher zu machen definieren wir die (partielle) charakteristische Funktion einer Sprache. Sei $L \subseteq \Sigma^*$ eine Sprache, dann ist die *partielle charakteristische Funktion* von L, $\hat{\chi}_L \colon \Sigma^* \to \{1\}$ definiert durch

$$w \mapsto \begin{cases} 1 & w \in L \\ \text{undefiniert} & \text{sonst} \end{cases}$$

Die *charakteristische Funktion* von L, $\chi_L \colon \Sigma^* \to \{0, 1\}$ ist definiert durch

$$w \mapsto \begin{cases} 1 & w \in L \\ 0 & w \notin L \end{cases}$$

Mit diesen Definitionen gilt, dass

- eine Sprache L semi-entscheidbar ist, wenn die partielle charakteristische Funktion $\hat{\chi}_L$ Turing-berechenbar ist.
- eine Sprache L ist entscheidbar, wenn die charakteristische Funktion χ_L Turing-berechenbar ist.

Sei M eine gegebene Turingmaschine. Mit $M(w)$ bezeichnen wir einen Durchlauf der Turingmaschine zur Eingabe w und mit $M(w)\downarrow$, dass die Maschine zur Eingabe w *terminiert*. Es gilt der folgende Zusammenhang zwischen Semi-Entscheidbarkeit und Entscheidbarkeit.

Satz 4.7. *Eine Sprache $L \subseteq \Sigma^*$ ist entscheidbar genau dann, wenn sowohl L als auch $\bar{L} = \Sigma^* \backslash L$ semi-entscheidbar sind.*

Beweis. " $\Rightarrow$: " Ein Entscheidungsalgorithmus kann so umgebaut werden, dass er in eine unendliche Schleife geht, wenn $w \notin L$ (bzw. $w \in L$) ist. Damit ist ein Semi-Entscheidungsalgorithmus für L (bzw. $\bar{L}$) gegeben.

" $\Leftarrow$: " Sei M eine DTM, die den Semi-Entscheidungsalgorithmus für L berechnet und $\bar{M}$ eine DTM, die den Semi-Entscheidungsalgorithmus für $\bar{L}$ berechnet. Dann ist ein Entscheidungsalgorithmus für L gegeben durch:

IN: $w \in \Sigma^*$
for $i = 0, 1, \ldots$ do
    if $M(w)\downarrow$ nach i Schritten then return 1
    if $\bar{M}(w)\downarrow$ nach i Schritten then return 0
end for

□

Wie wir später sehen werden, existieren Sprachen, die semi-entscheidbar, aber nicht entscheidbar sind. Vorher betrachten wir noch eine äquivalente Charakterisierung von Semi-Entscheidbarkeit.

Definition 4.8. *Eine Sprache $L \subseteq \Sigma^*$ heisst* rekursiv aufzählbar *(engl.* recursively enumerable*) genau dann, wenn sie entweder leer ist, oder wenn eine totale, Turing-berechenbare Funktion $f\colon \mathbb{N} \to \Sigma^*$ existiert mit der Eigenschaft*

$$L = \{f(0), f(1), f(2), \ldots\}.$$

Die Äquivalenz von Semi-Entscheibarkeit und rekursiver Aufzählbarkeit notieren wir ohne Beweis:

Satz 4.9. *Ein Sprache L ist genau dann semi-entscheidbar, wenn sie rekursiv aufzählbar ist.*

Zusammenfassend gilt damit, dass die folgenden Aussagen äquivalent sind:

- Die Sprache L ist Turing-akzeptierbar.
- Die Sprache L wird von einem WHILE-Programm akzeptiert.
- Die Sprache L wird von einer Grammatik erzeugt (d.h. ist vom Typ Chomsky-0).
- Die Sprache L ist semi-entscheidbar.
- Die Sprache L ist rekursiv aufzählbar.

Es gibt Sprachen, die *nicht* Turing-akzeptierbar sind und analog existieren auch partielle Funktionen ($f\colon \mathbb{N} \to \mathbb{N}$), die nicht Turing-berechenbar sind.

Ein Begriff, der nicht mit Aufzählbarkeit zu verwechseln ist, ist die *Abzählbarkeit*. Dazu ein kurzer Einschub.

Zwei Mengen X, Y heissen *gleichmächtig*, genau dann wenn eine Bijektion $f\colon X \to Y$ existiert. In Zeichen schreiben wir $X \sim Y$. Falls eine Teilmenge $Z \subseteq Y$ existiert sodass eine Bijektion $f\colon X \to Z$ exisitiert, dann heisst X *höchstens so mächtig* wie Y, in Zeichen $X \preceq Y$ (bzw. $X \prec Y$, falls Z eine echte Teilmenge von Y ist).

Eine Menge X ist *höchstens abzählbar*, falls $X \preceq \mathbb{N}$ und *überabzählbar*, falls $\mathbb{N} \prec X$. X ist endlich, falls $X \prec \mathbb{N}$ und unendlich, falls $\mathbb{N} \preceq X$. Insbesondere ist jede Menge, die gleichmächtig zur Menge der natürlichen Zahlen ist, abzählbar unendlich.

Beispiel 4.10. *Die Menge $X = \mathbb{N} \times \mathbb{N}$ ist abzählbar unendlich. Wir notieren die Tupel in X folgendermassen:*

$$\begin{array}{ccccc} \vdots & \vdots & \vdots & \vdots & \dots \\ (0,3) & (1,3) & (2,3) & (3,3) & \dots \\ (0,2) & (1,2) & (2,2) & (3,2) & \dots \\ (0,1) & (1,1) & (2,1) & (3,1) & \dots \\ (0,0) & (1,0) & (2,0) & (3,0) & \dots \end{array}$$

Die Elemente von X werden jetzt "diagonal" gezählt, d.h., wir listen sie in der Reihenfolge $(0,0), (0,1), (1,0), (0,2), (1,1), (2,0), \dots$. Die dazugehörige bijektive Funktion kann explizit angegeben werden:

$$f\colon \mathbb{N} \times \mathbb{N} \to \mathbb{N}, \quad (x,y) \mapsto \frac{x(x+3)}{2} + xy + \frac{y(y+1)}{2}.$$

Beispiel 4.11. *Die Potenzmenge $P(\mathbb{N})$ ist überabzählbar unendlich. Um das zu zeigen nehmen wir an sie wäre abzählbar unendlich. Dann müsste eine bijektive Abbildung $f\colon \mathbb{N} \to P(\mathbb{N})$ existieren. Definieren wir die Menge $M = \{x \in \mathbb{N} \mid x \notin f(x)\}$. Diese Menge ist nicht leer, da zumindest der Index der leeren Menge in ihr liegen muss. Außerdem gilt, da f bijektiv ist, dass es ein $m \in \mathbb{N}$ geben muss mit $M = f(m)$. Dann gilt, dass dieser Index m in der Menge M liegt, genau dann wenn $m \notin f(m)$ (nach Definition der Menge M), d.h., als $m \in M$ genau dann, wenn $m \notin f(m) = M$. Das ist ein Widerspruch. Also kann keine solche Abbildung existieren und damit ist $P(\mathbb{N})$ überabzählbar unendlich.*

Sei Σ ein Alphabet, dann ist $\Sigma^* = \bigcup_{i \geq 0} \Sigma^i$ abzählbar unendlich (also gleichmächtig mit $\mathbb{N}$). Die Menge aller Sprachen über Σ ist die Menge aller Teilmengen von Σ^* und damit gleichmächtig zur Potenzmenge $P(\mathbb{N})$. Nach dem obigen Beispiel bedeutet das, dass es überabzählbar unendlich viele Sprachen gibt.

Auf der anderen Seite gilt, dass es nur abzählbar unendlich viele Turingmaschinen gibt. Damit gibt es auch nur abzählbar unendlich viele Turing-akzeptierbare Sprachen. Foglich gilt (informal argumentiert), dass es Sprachen geben muss, die nicht Turing-akzeptierbar (oder äquivalent, nicht semi-entscheidbar) sind.

Wir betrachten jetzt ein klassisches Beispiel für eine Sprache, die semi-entscheidbar, aber nicht entscheidbar ist: Jeder Turingmaschine kann ein *Index* zugewiesen werden. (Eine Turingmaschine ist durch ihre Zustandsmenge, das Bandalphabet (das eine Obermenge des Alphabets ist) sowie durch die Übergangsrelation bestimmt. Die Übergangsrelation kann binär codiert werden und diese Codierungen können gereiht werden, sodass sich ein eindeutig bestimmter

Index ergibt, der als eine natürliche Zahl oder in binärer Darstellung gegeben sein kann.) Sei $\alpha \in \{0,1\}^*$ ein (binärer) Index und wir bezeichnen die dazugehörige Turingmaschine mit M^α.

Eine Sprache, die semi-entscheidbar, aber nicht entscheidbar ist, ist durch das *spezielle Halteproblem* gegeben:

$$sHP = \{w \in \{0,1\}^* \mid M^w(w) \downarrow\},$$

d.h., die Sprache, die alle Wörter enthält, für die die Turingmaschine angesetzt auf ihren eigenen Index terminiert. Dieses Problem ist ein Selbstanwendungsproblem für Turingmaschinen.

Semi-Entscheidbarkeit ist einfach zu sehen: Zu gegebenem $w \in \{0,1\}^*$ sei M die Maschine, die durch den Index w gegeben ist. Wir "decodieren" die Maschine und führen sie mit dem Input w aus. Falls die Turingmaschine terminiert, dann hält der Algorithmus und akzeptiert die Eingabe.

Um die *Unentscheidbarkeit* zu zeigen führen wir wieder einen Widerspruchsbeweis durch. Angenommen es existiert ein Entscheidungsalgorithmus M_{sHP}, der die Sprache sHP entscheidet. Ausgehend von dieser Maschine konstruieren wir eine Turingmaschine M_1 wie folgt:

IN: $w \in \Sigma^*$
simuliere $M_{sHP}(w)$
if $M_{sHP}(w) \downarrow$ mit $w \in sHP$ then kreise
if $M_{sHP}(w) \downarrow$ mit $w \notin sHP$ then stoppe

Sei α der Index von M_1, d.h., $M_1 = M^\alpha$. Nach Konstruktion von M_1 gilt $M_1(\alpha) \downarrow$ genau dann, wenn $\alpha \notin sHP$.

Nach Definition der Sprache sHP gilt, $\alpha \notin sHP$ genau dann wenn $M^\alpha(\alpha)$ *nicht* terminiert, d.h., wenn $M_1(\alpha)$ nicht terminiert. Also gilt: $M_1(\alpha)$ terminiert genau dann wenn $M_1(\alpha)$ nicht terminiert. Das ist ein Widerspruch, daher war die Annahme der Existenz eines Entscheidungsalgorithmus falsch.

Satz 4.12. *Das spezielle Halteproblem ist semi-entscheidbar, aber nicht entscheidbar.*

Entscheidbarkeit und das Wortproblem formaler Sprachen Zur Erinnerung, das Wortproblem ist das Problem: gegeben eine Sprache L über einem Alphabet Σ und ein Wort $\omega \in \Sigma^*$, stelle fest, ob ω ein Element der Sprache L ist. Das Wortproblem ist für reguläre, kontextfreie und kontextsensitive Sprachen entscheidbar, nicht aber für Typ-0 Sprachen.

- L reguläre Sprache: in diesem Fall existiert ein deterministisch endlicher Automat A mit $L = L(A)$. Um festzustellen, ob ein gegebenes Wort w in der Sprache L liegt muss das Wort Zeichen für Zeichen vom Automaten abgearbeitet werden, d.h. man beobachtet die Zustandsübergänge im Automaten, die durch die Eingabe von w hervorgerufen werden. Falls ein Endzustand erreicht wird, dann liegt w in der Sprache.

- L kontextfreie Sprache: dann existiert eine kontextfreie Grammatik G mit $L = L(G)$, die in Chomsky Normalform gebracht werden kann. Um festzustellen, ob ein gegebenes Wort $w \in L$ liegt, kann der CYK-Algorithmus angewandt werden, der in endlicher Zeit terminiert.

Sei L eine *kontextsensitive Sprache*, dann ist das Wortproblem ebenfalls entscheidbar. Sei $G = (V, \Sigma, S, \Pi)$ eine kontextsensitive Grammatik mit $L = L(G)$. Wir definieren zu natürlichen

Zahlen $m, n \in \mathbb{N}$ die Mengen

$$T_m^n = \{w \in (V \cup \Sigma)^* \mid |w| \leq n \text{ und } w \text{ lässt sich aus } S \text{ in höchstens } m \text{ Schritten ableiten}\}.$$

Wenn n fixiert wird, dann muss es ein $m \in \mathbb{N}$ geben, ab dem sich T_m^n nicht mehr ändert, d.h.,

$$T_m^n = T_{m+1}^n = T_{m+2}^n = \ldots,$$

da in einer kontextsensitiven Grammatik keine wortverkürzenden Produktionen vorkommen. Damit ist auch die Vereinigung über alle Mengen T_m^n bei fixem n, d.h. die Menge

$$T_n = \bigcup_{m \geq 0} T_m^n,$$

endlich. Man beachte, dass kontextsensitive Sprachen nicht wortverkürzend sein können (im Gegensatz zu Typ-0 Sprachen). Damit ergibt sich ein Algorithmus wie folgt: gegeben ein Wort w der Länge n, wenn $w \in T_n$ liegt, dann gilt $w \in L$, sonst nicht.

Beispiel 4.13. *Sei $L_1 = \{a^n b^n c^n \mid n \in \mathbb{N}^*\}$ die Sprache aus Beispiel 3.43 mit Grammatik G mit $V = \{B, S\}$, $\Sigma = \{a, b, c\}$, Startsymbol S und Produktionen Π gegeben durch:*

$$S \longrightarrow aSBc \mid abc, \quad cB \longrightarrow Bc, \quad bB \longrightarrow bb.$$

Dann gilt:

$$\begin{aligned}
&T_m^0 = \emptyset, \quad m \geq 0 \\
&T_m^1 = T_m^2 = \{S\}, \quad m \geq 0 \\
&T_0^3 = \{S\}, \quad T_1^3 = \{S, abc\} = T_m^3, \quad m \geq 1 \\
&T_0^4 = \{S\}, \quad T_1^4 = \{S, aSBc, abc\} = T_m^4, \quad m \geq 1 \\
&T_m^5 = \{S, abc, aSBc\}, \quad m \geq 1 \\
&T_m^6 = \{S, abc, aSbc, aabcBc, aabBcc, aabbcc\}, \quad m \geq 4.
\end{aligned}$$

5 Komplexität

Im letzten Kapitel haben wir uns damit beschäftigt, ob etwas berechenbar oder entscheidbar ist. Jetzt geht es darum festzustellen, ob etwas *effizient* berechenbar, bzw. entscheidbar, ist. Effizienz bezieht sich auf den Bedarf an Ressourcen wie Rechenzeit oder Speicherplatz, die in Abhängigkeit der Grösse der Eingabeparameter angegeben werden.

Dabei geht es einerseits darum für ein konkretes Problem ein effizientes Verfahren anzugeben und dann den Rechenaufwand zu analysieren. Dadurch erhält man sowohl einen Algorithmus zur Lösung des Problems als auch eine obere Schranke für die Komplexität des Problems. Es kann aber auch ein Berechnungsverfahren geben, das schneller als der gefundene Algorithmus ist. Damit beschäftigen wir uns im ersten Teil dieses Abschnitts.

Man kann auch umgekehrt versuchen eine untere Schranke für die Komplexität eines Problems zu finden. Eine triviale untere Schranke der Zeitkomplexität für ein Problem mit Eingabe der Grösse n ist n, da zumindest die Eingabe gelesen werden muss. Im allgemeinen ist es schwieriger untere Schranken (genommen über allen möglichen Verfahren) für die Komplexität zu bestimmen.

Weitere Fragen untersuchen die Stärke verschiedener Maschinenmodelle, zum Beispiel ob und inwieweit das nichtdeterministische Berechnungsmodell dem deterministischen äquivalent ist. Damit beschäftigen wir uns im zweiten Teil dieses Kapitels.

5.1 Komplexitätsabschätzungen

Definition 5.1. *Es seien $f, g\colon \mathbb{N} \to \mathbb{R}^+$ Funktionen.*

(a) Es gilt $f(n) = O(g(n))$, wenn eine reelle Zahl $C > 0$ und $n_0 \in \mathbb{N}$ existieren, sodass für alle natürlichen Zahlen $n \geq n_0$ gilt

$$f(n) \leq Cg(n).$$

(b) Es gilt $f(n) = \Omega(g(n))$, wenn eine reelle Zahl $c > 0$ und $n_0 \in \mathbb{N}$ existieren, sodass für alle natürlichen Zahlen $n \geq n_0$ gilt

$$f(n) \geq cg(n).$$

Die Notationen $O(g(n))$ und $\Omega(g(n))$ vernachlässigen die Konstanten und sind so zu verstehen, dass für grosses n das Verhalten der Funktion $g(n)$ entspricht.

Ein Polynom ist eine Funktion der Form

$$p(x) = a_0 + a_1x + a_2x^2 + \cdots + a_dx^d.$$

Falls $a_d \neq 0$, dann ist d der Grad des Polynoms. Wenn der führende Term x^d herausgehoben wird, erhalten wir

$$p(x) = \left(\frac{a_0}{x^d} + \frac{a_1}{x^{d-1}} + \cdots + \frac{a_{d-1}}{x} + a_d\right) x^d.$$

Für grosse Werte von x sind alle Terme bis auf den führenden vernachlässigbar und es gilt für ein Polynom p vom Grad d, dass $p(n) = O(n^d)$. Für den Logarithmus (egal zu welcher

n	10^6	$300\log(n)$	$100n$	$10n^2$	2^n
1	10^6	0.	100	10	2.
10	10^6	690.776	1000	1000	1024.
100	10^6	1381.55	10000	100000	1.268×10^{30}
1000	10^6	2072.33	100000	10^7	1.072×10^{301}
100000	10^6	3453.88	10^7	10^{11}	9.99×10^{30102}
10^6	10^6	4144.65	10^8	10^{13}	9.9×10^{301029}

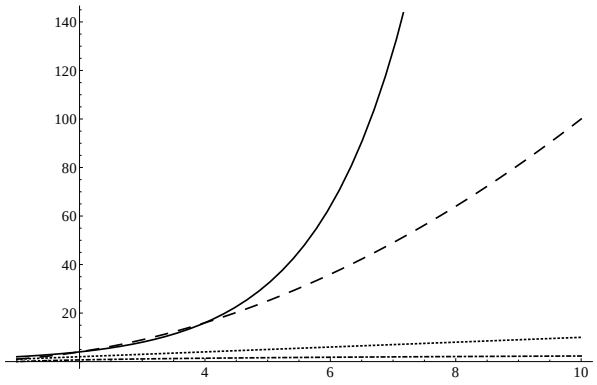


Abbildung 5.1: Die Funktionen $\log(x)$ (Strich-Punkt), x (Punkte), x^2 (gestrichelt), und 2^x (schwarze Linie) im Vergleich

Basis) gilt $\log(n) = O(n)$. Da ein Input der Länge n zumindest n Schritte zum Lesen der Eingabe erfordert, kann es in dem Sinn bestenfalls eine Komplexität von $O(n\log(n))$ geben. Da $n\log(n) = O(n^2)$ ist auch diese Komplexität polynomiell. Exponentielle Funktionen c^n (mit $c > 1$) können nicht durch Polynome abgeschätzt werden.

Beispiel 5.2. *Wir untersuchen die Komplexität von Cocke Younger Kasami:*

CYK-Algoritmus *Input: kontextfreie Grammatik* $G = (V, \Sigma, S, \Pi)$ *in Chomsky-Normalform und ein Wort* $w = x_1x_2\ldots x_n \in \Sigma^+$*; Output:* $w \in L(G)$

1. *Initialisiere den* $n \times n$*-Array* cyk *mit Einträgen* $\emptyset$

2. *Erste Zeile:*
   For $i = 1, i \leq n, i{+}{+}$ *Do* $\text{cyk}[i,1] = \{A \mid (A \to x_i) \in \Pi\}$

3. *Restliche Zeilen:*
   For $j = 1, j \leq n, j{+}{+}$ *Do*
   For $i = 1, i \leq n+1-j, i{+}{+}$ *Do*
   For $k = 1, k \leq j-1, k{+}{+}$ *Do*
   $\text{cyk}[i,j] = \text{cyk}[i,j] \cup \{A \mid \exists\, B, C\colon (A \to BC) \in \Pi \wedge B \in \text{cyk}[i,k] \wedge C \in \text{cyk}[i+k, j-k]\}$

4. *Return* $S \in \text{cyk}[1,n]$

Zur Bestimmung der Komplexität schätzen wir die Anzahl der Zuweisungen. *Die Initialisierung (Schritt 1) verlangt* n^2 *Zuweisungen. Die Berechnung der ersten Zeile (Schritt 2) verlangt* n *Zuweisungen. In der Berechnung der restlichen Zeilen (Schritt 3) läuft* j *von* 1 *bis* n *und* i, k höchstens *von* 1 *bis* n. *Damit kann die Anzahl der Zuweisungen in diesem Schritt nach oben durch* n^3 *abgeschätzt werden.*

Insgesamt erhalten wir das die Anzahl von Zuweisungen im CYK-Algorithmus von der Grössenordnung $O(n + n^2 + n^3) = O(n^3)$ *ist.*

In der Untersuchung von CYK oben haben wir die Anzahl der Zuweisungen gezählt. Wenn Komplexität als Maß für die Rechenzeit genommen wird, muss der Zeitaufwand für eine Wertzuweisung so gross angesetzt werden wie die Anzahl der Bits, die bei dieser Aktion übertragen werden. In unserer Analyse ist jeder Schritt gleich (also mit 1) gewichtet worden. Diese beiden unterschiedlichen Zählweisen führen zu *Bit-Komplexität*, bzw. zu *arithmetischer Komplexität* (wie oben). Wenn die gespeicherten Zahlenwerte eine Konstante nicht überschreiten, dann unterscheiden sich die beiden Abschätzungen nur um einen konstanten Faktor, sind also äquivalent.

Das Wortproblem für reguläre Sprachen ist (wenn ein akzeptierender DEA gegeben ist), in linearer Zeit entscheidbar, da nur die Wortlänge abgearbeitet werden muss.

Betrachten wir für das Wortproblem für kontextsensitive Sprachen das Verfahren, das am Ende des letzten Abschnitts besprochen wurde: zu einem gegebenen Wort der Länge n hängt die Komplexität im wesentlichen von der Konstruktion der Menge T_n ab. Die Mächtigkeit von T_n ist durch die Anzahl der Wörter von Länge höchstens n in $(V \cup \Sigma)^*$ beschränkt. Wenn $|V \cup \Sigma| = c$ (wobei $c \in \mathbb{N}$ eine fixe natürliche Zahl ist), dann ist die Anzahl der Wörter von Länge $\leq n$ gegeben durch

$$\sum_{k=0}^{n} c^k = \frac{c^{n+1} - 1}{c - 1} = O(c^n).$$

Damit erhalten wir als worst-case Abschätzung eine exponentielle Komplexität.

Durch das Zulassen von Mehrbandmaschinen können realistischere Komplexitätsfunktionen $f(n)$ angegeben werden, als z.B. nur durch Verwendung von Einbandmaschinen. Allgemein gilt, dass eine $f(n)$-rechenzeitbeschränkte Mehrbandmaschine durch eine $O(f(n)^2)$-rechenzeitbeschränkte Einbandmaschine simuliert werden kann. Wenn $f(n)$ ein Polynom ist, dann ist auch $f(n)^2$ ein Polynom. Damit ändert sich zwar die Komplexitätsabschätzung, aber die Klasse *polynomielle Komplexität* bleibt erhalten. Analog zum Zählen von Zuweisungen bei WHILE-Programmen (oder allgemein Programmiersprachen), werden bei Turingmaschinen Zustandsübergänge gezählt.

5.2 Komplexitätsklassen

Ein zentrales Thema der Komplexitätstheorie ist die Stärke verschiedener Maschinenmodelle. Zum Beispiel die Frage, inwiefern das nichtdeterministische Berechnungsmodell dem deterministischen äquivalent ist. Wir unterscheiden im wesentlichen die zwei Klassen P und NP:

$P =$ Klasse aller Probleme, die durch deterministische Algorithmen in polynomieller Zeit lösbar sind

und

$$NP = \text{Klasse aller Probleme, die durch nichtdeterministische Algorithmen in polynomieller Zeit lösbar sind}$$

Jeder deterministische Algorithmus ist auch nichtdeterministisch, daher gilt $P \subseteq NP$. Ein nichtdeterministischer Algorithmus für ein Problem aus NP kann mit Hilfe eines deterministischen Algorithmus mit exponentiellem Zeitbedarf simuliert werden.

Wir definieren jetzt Reduzierbarkeit im Kontext von polynomiell zeitbeschränkten Berechnungen.

Definition 5.3. *Seien $L_1 \subseteq \Sigma_1^*$ und $L_2 \subseteq \Sigma_2^*$ Sprachen. Dann heisst L_1 auf L_2* polynomiell reduzierbar*, falls es eine totale, mit polynomieller Komplexität berechenbare Funktion $f\colon \Sigma_1^* \to \Sigma_2^*$ gibt sodass für alle $x \in \Sigma_1^*$ gilt*

$$x \in L_1 \qquad \Longleftrightarrow \qquad f(x) \in L_2,$$

in Zeichen $L_1 \leq_p L_2$.

Falls L_1 auf L_2 polynomiell reduzierbar ist, dann liegt L_1 (zumindest) in der gleichen Klasse wie L_2.

Lemma 5.4. *Falls $L_1 \leq_p L_2$ und $L_2 \in P$ (bzw. $L_2 \in NP$), dann ist auch $L_1 \in P$ (bzw. $L_1 \in NP$).*

Beweis. Sei $L_1 \leq_p L_2$ durch eine Funktion f, die durch eine Turingmaschine M_f in $O(p(n))$ berechnet werden kann für ein Polynom p. Angenommen $L_2 \in P$ und kann durch eine Turingmaschine M in $O(q(n))$ berechnet werden für ein Polynom q.

Dann ist die Hintereinanderschaltung $M_f; M$ der beiden Maschinen ein Algorithmus von polynomieller Komplexität: Zu gegebenem $x \in L_1$, sei $|x|$ die Grösse des Inputs. Dann kann die Rechenzeit von $M_f; M$ abgeschätzt werden durch:

$$p(|x|) + q(|f(x)|) \leq p(|x|) + q(p(|x|)) = r(|x|).$$

Wenn p, q Polynome sind, ist auch r ein Polynom. Analog geht der Fall $L_2 \in NP$ mit einer nichtdeterministischen Maschine M. □

Wir haben bereits erwähnt, dass offensichtlich $P \subseteq NP$ gilt. Bis jetzt ist nicht bekannt, ob $P = NP$ gilt. Diese Frage wird das $P - NP$-Problem genannt. Für viele Probleme ist bekannt, dass sie in NP liegen, aber man kennt keine (deterministisch) polynomiellen Algorithmen zur Lösung dieser Probleme. Für einige der Probleme, die bekanntermaßen in NP liegen und für die kein polynomieller Algorithmus bekannt ist, wurde gezeigt, dass sie in der folgenden Weise verbunden sind: entweder existieren für alle Probleme polynomielle Algorithmen oder für keines der Probleme. Im ersten Fall gilt $P = NP$, im zweiten $P \neq NP$. Diese Probleme werden NP-vollständig genannt. Da es sehr viele wichtige und gründlich untersuchte NP-vollständige Probleme gibt, gilt es als eher unwahrscheinlich, dass Gleichheit gilt.

Definition 5.5. *Eine Sprache A heisst NP-*hart*, falls für alle Sprachen $L \in NP$ gilt: $L \leq_p A$. Eine Sprache A heisst NP-*vollständig*, falls A NP-hart ist und $A \in NP$ gilt.*

Satz 5.6. *Sei A NP-vollständig, dann gilt: $A \in P \;\Leftrightarrow\; P = NP$.*

Beweis. Sei $A \in P$ und L eine beliebige Sprache in NP. Da A NP-hart ist, gilt $L \leq_p A$ und mit Lemma 5.4 folgt, dass $L \in P$. Da L beliebig aus NP gewählt war folgt $P = NP$. Andererseits, falls $P = NP$ und da $A \in NP$, gilt $A \in P$. □

Lemma 5.7. *Die Relation $\leq_p$ ist eine* transitive Relation *auf Sprachen, das heisst, wenn L_1, L_2, L_3 Sprachen sind und sowohl $L_1 \leq_p L_2$ als auch $L_2 \leq_p L_3$ gilt, dann gilt $L_1 \leq_p L_3$.*

Beweis. Seien $L_1 \subseteq \Sigma_1^*$, $L_2 \subseteq \Sigma_2^*$ und $L_3 \subseteq \Sigma_3^*$ Sprachen jeweils über den Alphabeten Σ_1, Σ_2, und Σ_3. Angenommen es gilt $L_1 \leq_p L_2$ und $L_2 \leq_p L_3$. Dann ist zu zeigen, dass $L_1 \leq_p L_3$. Aus $L_1 \leq_p L_2$ folgt nach Definition die Existenz einer totalen, in polynomieller Zeit berechenbaren Funktion $f_1 \colon \Sigma_1^* \to \Sigma_2^*$ sodass für alle $x \in \Sigma_1^*$ gilt, $x \in L_1 \Leftrightarrow f_1(x) \in L_2$. Analog existiert eine totale, in polynomieller Zeit berechenbare Funktion $f_2 \colon \Sigma_2^* \to \Sigma_3^*$ mit $y \in L_2 \Leftrightarrow f_2(y) \in L_3$. Dann gilt, dass die Hintereinanderausführung $f_3 = f_2 \circ f_1 \colon \Sigma_1^* \to \Sigma_3^*$ eine totale Funktion ist, die in polynomieller Zeit berechenbar ist. Ausserdem gilt für jedes $x \in L_1$:

$$x \in L_1 \quad \Leftrightarrow \quad y = f_1(x) \in L_2 \quad \Leftrightarrow f_2(y) = f_2(f_1(x)) = f_3(x) \in L_3,$$

d.h., $L_1 \leq_p L_3$. □

Polynomielle Reduzierbarkeit wird verwendet um ausgehend von bekanntermassen NP-vollständigen Problemen zu zeigen, dass andere Probleme NP-vollständig sind mittels folgender Aussage.

Korollar 5.8. *Sei $L_1 \leq_p L_2$. Falls L_1 NP-vollständig ist und $L_2 \in NP$ gilt, dann ist auch L_2 NP-vollständig.*

Beweis. Aus der Transitivität von $\leq_p$ folgt, dass wenn L_1 NP-vollständig ist und $L_1 \leq_p L_2$, dass für jedes $L \in NP$ gilt $L \leq_p L_1 \leq_p L_2$, d.h., $L \leq_p L_2$. Damit ist L_2 NP-hart. Da ausserdem $L_2 \in NP$, ist L_2 somit NP-vollständig. □

Ein nichtdeterministischer Algorithmus hält für jeden Fall, der die Antwort "ja" liefert (im Entscheidungsfall), in polynomieller Zeit an. Der Zeitbedarf richtet sich nach der kürzest möglichen Rechnung, die zu dieser Lösung führt. In anderen Fällen muss der Algorithmus nicht halten. Für jeden "ja"-Fall kann eine Überprüfung in polynomieller Zeit durchgeführt werden.

Als Beispiel betrachten wir die die Probleme *Hamiltonscher Kreis* und *Travelling Salesman*.

Sei G ein endlicher, zusammenhängender Graph. Ein Weg in G heisst *Hamiltonsch*, falls jeder Knoten in G *genau einmal* durchlaufen wird. Ist dieser Weg ausserdem ein Kreis, so nennt man ihn einen *Hamiltonschen Kreis*. Einen Graphen nennt man *Hamiltonsch*, falls er einen Hamiltonschen Kreis enthält. Das Problem *Hamiltonscher Kreis* besteht darin festzustellen, ob ein gegebener Graph Hamiltonsch ist. In Abbildung 5.2 ist links ein *maximaler Nicht-Hamiltonscher Graph* abgebildet. Maximal Nicht-Hamiltonsch bedeutet, dass wenn eine Kante zwischen zwei beliebigen, nicht angrenzenden (adjazenten) Knoten hinzugefügt wird dann wird der Graph Hamiltonsch. Im mittleren Graphen ist eine weitere Kante hinzugefügt worden und damit ist der Graph Hamiltonsch: die Kanten des Graphen sind grau gezeichnet, der Hamiltonsche Kreis ist strichliert.

Das *Travelling Salesman Problem* (TSP) geht von einem ungerichteten, gewichteten Graphen G aus. Das Problem ist einen Rundweg zu finden, der jeden Knoten genau einmal besucht und dessen Gesamtkosten (die Summe über die Gewichte der Kanten) minimal sind.

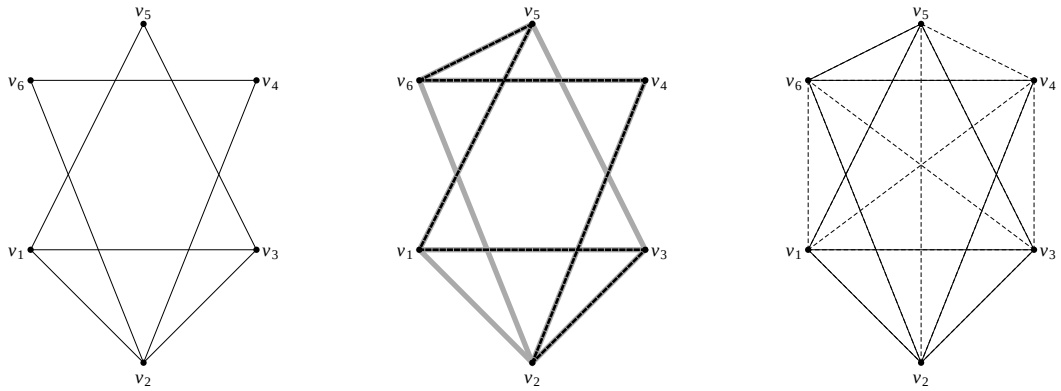


Abbildung 5.2: Links: maximaler Nicht-Hamiltonscher Graph, Mitte: Hamiltonscher Graph (Kreis gestrichelt), Rechts: Graph mit Hamiltonscher Kreis auf TSP reduziert

Satz 5.9. Hamiltonscher Kreis *ist NP-vollständig.*

Ohne Beweis. □

Korollar 5.10. TSP *ist NP-hart.*

Beweis. Wir zeigen, dass das Problem Hamiltonscher Kreis polynomiell auf TSP reduziert werden kann. Gegeben sei ein ungerichteter Graph $G = (V, E)$ mit n Knoten. Wir erweitern G zu einem vollständigen Graphen $\overline{G} = (V, \overline{E})$ mit der folgenden Gewichtsfunktion:

$$w(u, v) = \begin{cases} 1 & (u, v) \in E \\ 2 & (u, v) \notin E \end{cases} \qquad \forall\, u, v \in V.$$

In Abbildung 5.2 rechts ist der vollständige Graph eingezeichnet wobei die durchgezogenen Linien jene mit Gewicht 1 sind (also die aus dem ursprünglichen Graphen) und die strichlierten jene mit Gewicht 2 sind (also die zur Vervollständigung dazu genommen wurden).

Dann ist der Graph G Hamiltonsch *genau dann, wenn* die Lösung von TSP in $\overline{G}$ einen Rundweg der Länge n liefert. Die Reduktion von Hamiltonscher Kreis zu TSP kann offensichtlich in polynomieller Zeit durchgeführt werden. Damit gilt:

$$\text{Hamiltonscher Kreis} \quad \leq_p \quad \text{TSP}.$$

Wegen Satz 5.9 gilt (nach Definition 5.5), dass für alle Sprachen L in NP: $L \leq_p$ Hamiltonscher Kreis. Wegen der Transitivität der polynomiellen Reduzierbarkeit folgt mit obigem Ergebnis damit:

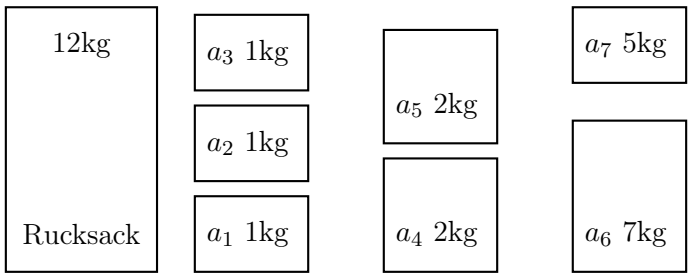
$$\text{Für alle } L \in NP\text{:} \quad L \leq_p \text{TSP},$$

und damit ist TSP NP-hart. □

Um zu zeigen, dass TSP NP-vollständig ist, müßte man noch beweisen, dass TSP $\in NP$ gilt (siehe Definition 5.5). Das ist allerdings *nicht* der Fall. Das ähnlich lautende Entscheidungsproblem: *gegeben ein ungerichteter gewichteter Graph G und eine Zahl K, entscheide, ob es in G einen Rundweg gibt, dessen Kosten K nicht überschreiten* ist NP-vollständig. Zwei weitere Beispiele für NP-vollständige Probleme sind

- *Rucksack Problem* (Knapsack problem) Gegeben sind natürliche Zahlen $a_1, a_2, \ldots, a_k \in \mathbb{N}$ und eine Schranke $b \in \mathbb{N}$. Gefragt ist, ob eine Teilmenge $I \subseteq \{1, 2, \ldots, k\}$ existiert sodass $\sum_{i \in I} a_i = b$.
- *Färbbarkeit* (Graph coloring) Gegeben ist ein ungerichteter Graph $G = (V, E)$ und eine natürliche Zahl k. Gefragt ist, ob eine Färbung der Knoten mit k Farben existiert, so, dass keine zwei benachbarten Knoten in G dieselbe Farbe haben.

Knapsack problem. Gegeben sind natürliche Zahlen $a_1, a_2, \ldots, a_k \in \mathbb{N}$ und eine Schranke $b \in \mathbb{N}$. Gefragt ist, ob eine Teilmenge $I \subseteq \{1, 2, \ldots, k\}$ existiert sodass $\sum_{i \in I} a_i = b$. Die Zahl b entspricht hier dem maximalen Gewicht, das (zum Beispiel) in einen Rucksack gefüllt werden kann (im Bild unten sind maximal 12kg erlaubt). Die Zahlen a_i entsprechen Gewichten von Gegenständen, die in den Rucksack gepackt werden sollen (d.h. im Bild unten haben wir 3 Gegenstände vom Gewicht 1kg, zwei Gegenstände vom Gewicht 2kg und je einen Gegenstand vom Gewicht 5kg und 7kg). Gesucht ist jetzt eine Auswahl von Gegenständen mit denen der Rucksack optimal angefüllt werden kann. Dieses Problem kann eine, mehrere oder gar keine Lösung besitzen.



In dem speziellen Beispiel gibt es verschiedene Varianten, zum Beispiel den Rucksack mit den Elementen $\{a_7, a_6\}$, oder $\{a_1, a_2, a_3, a_4, a_5, a_7\}$, oder $\{a_6, a_5, a_4, a_3\}$, oder ..., zu packen. Alle möglichen Zusammenstellungen der Gegenstände $\{a_1, a_2, \ldots, a_7\}$ sind durch die Potenzmenge $P(\{a_1, a_2, \ldots, a_7\})$ gegeben. Wenn n Gegenstände vorhanden sind, dann hat die Potenzmenge die Kardinalität 2^n. Die Komplexität des Problems ist also im schlimmsten Fall exponentiell.

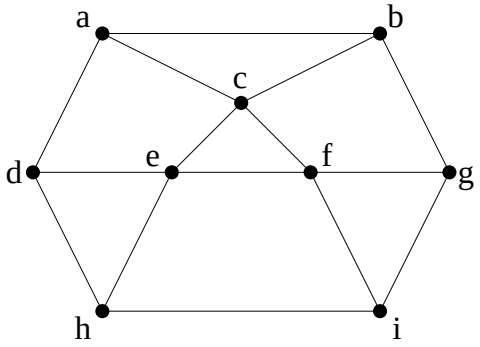
Es gibt verschiedene Varianten des Knapsack-Problems, eine übliche ist das *0-1 Knapsack Problem*: hier wird die Anzahl der Stücke a_i mit Gewicht w_i (w für englisch weight) und Wert v_i (v für englisch value), die gepackt werden dürfen mit 1 beschränkt (d.h. von jedem Gewicht kann eins oder kein Stück gepackt werden) und die maximale Füllmenge b muss nicht genau erreicht werden, sondern ist eine obere Schranke. Mathematisch formuliert sieht das so aus:

$$\text{Maximiere} \quad \sum_{i=0}^{n} x_i v_i \quad \text{unter der Bedingung} \quad \sum_{i=0}^{n} x_i w_i \leq b,$$

wobei x_i Werte aus $\{0, 1\}$ annehmen kann. Es soll also eine Füllung von maximalen Wert erreicht werden, die die Gesamtfüllmenge nicht übersteigt. Beim unbeschränkten Knapsack-Problem ist die Stückzahl nach oben nicht limitiert (nach unten durch 0).

Graph coloring Gegeben ist ein ungerichteter Graph $G = (V, E)$ und eine natürliche Zahl k. Gefragt ist, ob eine Färbung der Knoten mit k Farben existiert, so, dass keine zwei benachbarten Knoten in G dieselbe Farbe haben. Wenn die Anzahl der Knoten $|V| = n$ ist und

die Anzahl der verfügbaren Farben k, dann gibt es insgesamt k^n Möglichkeiten den Graphen zu färben. Auch hier gilt, dass das Problem für einen gegebenen Graphen keine oder mehrere Lösungen haben kann.



Der abgebildete Graph besitzt keine Färbung der Knoten, bei der zwei benachbarte immer unterschiedliche Farben haben, wenn mit *zwei* Farben gearbeitet wird (das Beispiel ist klein genug, dass mit einem brute-force Zugang alle Kombinationen ausgerechnet werden und überprüft werden können). Wenn *drei* Farben zugelassen werden gibt es verschiedene Lösungen, zum Beispiel können die folgenden drei Farbgruppen verwendet werden (jede Menge von Knoten enthält Knoten der gleichen Farbe):

$$\begin{array}{rlll} & \{a,e,i\}, & \{b,d,f\}, & \{c,g,h\} \\ \text{oder} & \{a,f,h\}, & \{b,e,i\}, & \{c,d,g\}. \end{array}$$

Das Färbbarkeitsproblem tritt in Anwendungen zum Beispiel auf, wenn verschiedenen Tasks gewisse time slots zugewiesen werden müssen, wobei es Konflikte zwischen den Tasks geben kann, d.h. zum Beispiel verschiedene Aufgaben müssten gleichzeitig auf die selbe Ressource zugreifen. Die Knoten im Graphen entsprechen dann den Tasks, die Kanten den Konflikten zwischen den Tasks und die Farben den jeweiligen time slots. Auch Sudokus können als Färbbarkeitsproblem interpretiert werden.

Strategien für *NP*-vollständige Probleme Alle derzeit bekannten Algorithmen zur Lösung *NP*-vollständiger Probleme haben höhere Komplexität als polynomiell und es ist unbekannt, ob sie in polynomieller Zeit lösbar sind (siehe P vs. *NP*). Es gibt verschiedene Techniken, um diese Schwierigkeit zu umgehen, ein paar sind hier angeführt:

- *Näherungsweise Lösung*: bei manchen Fragestellungen kann es ausreichend sein eine Lösung zu finden, die der optimalen Lösung "nahe genug" kommt.
- Einsatz von *heuristischen Algorithmen*: Verwendung von Methoden, die in der Praxis "gut genug" funktionieren, für die es aber keinen Beweis gibt, dass sie immer eine Lösung finden oder immer die optimale Lösung (vgl. auch Punkt 1), oder für die es keine bewiesenen Komplexitätsresultate gibt.
- Einsatz von *probabilistischen Algorithmen*: Zulassen von einem gewissen Grad an Zufälligkeit zum Beispiel in der Auswahl von Zwischenschritten (z.B. statt alle Möglichkeiten in fest vorgegebener Ordnung durchzuprobieren), die im Mittel ein gutes Ergebnis liefern. Das Endergebnis kann dabei mit gewisser Wahrscheinlichkeit auch falsch sein (beachte:

die Überprüfung, ob ein Output tatsächlich die Lösung eines *NP*-vollständigen Problems ist, ist in polynomieller Zeit möglich). *Genetische Algorithmen* fallen unter diese Klasse von Algorithmen.