

### 3 Komplexitätstheorie

#### 3.1 Komplexitätsklassen

Im Kapitel 2 haben wir verschiedene Berechenbarkeitsbegriffe für Funktionen definiert und ihre Äquivalenz festgestellt. Leider haben wir auch gelernt, dass es Funktionen gibt, die *nicht* berechenbar sind.

In diesem Kapitel nun beschäftigen wir uns nur mit den *berechenbaren* Funktionen und interessieren uns nun dafür, wie *schwierig* es ist, sie zu berechnen, also welche *Ressourcen* zu ihrer Berechnung benötigt werden. Die wichtigsten Ressourcen sind sicherlich *Rechenzeit und Speicherplatz*. Das findet seinen Niederschlag in der Betrachtung von *Zeit- und Raumkomplexität* von Algorithmen und Problemen.

Bevor wir uns jedoch mit diesen Komplexitäten beschäftigen, möchten wir hier die LANDAUSCHEN Symbole angeben, um Funktionen vergleichen zu können:

**Definition 3.1** Seien  $f$  und  $g$  zwei Funktionen  $f, g: \mathbb{N} \rightarrow \mathbb{R}$ .

- (i) Wir sagen  $f = O(g)$  genau dann, wenn  $\lim_{n \rightarrow \infty} \frac{f(n)}{g(n)} = c$  mit einem  $c \geq 0$  gilt.
- (ii) Wir sagen  $f = \Theta(g)$  genau dann, wenn  $f = O(g)$  und  $g = O(f)$  gelten.

Eine dazu äquivalente Definition, wie man zeigen kann, ist

**Definition 3.2** Seien  $f$  und  $g$  zwei Funktionen  $f, g: \mathbb{N} \rightarrow \mathbb{R}$ .

- (i) Wir sagen  $f = O(g)$  genau dann, wenn es positive Konstanten  $c_1$  und  $c_2$  gibt, so dass

$$f(n) \leq c_1 \cdot g(n) + c_2$$

- (ii) Wir sagen  $f = \Theta(g)$  genau dann, wenn  $f = O(g)$  und  $g = O(f)$  gelten.

Gilt also  $f = O(g)$ , so wächst  $f$  asymptotisch nicht schneller als  $g$ ; gilt  $f = \Theta(g)$ , so haben  $f$  und  $g$  asymptotisch gleiches Wachstum.

Welchen Einfluss die Anzahl der Rechenschritte in Abhängigkeit von der Länge der Eingabe auf die Laufzeit eines Programms hat, wird in der folgenden Tabelle verdeutlicht. Annahme: ein Programm wird berechnet, wobei die Funktion  $f$  die Anzahl der auszuführenden Operationen angibt. In Tabelle 3.1 ist die Zeit für die Ausführung angegeben, falls ein Computer eine Million Operationen pro Sekunde ausführt. (Wenn wir eine andere Geschwindigkeit des Computers annehmen, z.B.  $10^9$  Operationen je Sekunde, so ändern sich die Tabellenwerte nur um einen konstanten Faktor. Wir merken aber an, daß aus physikalischen Gründen eine Schranke für die Geschwindigkeit existiert.)

$f \setminus n$	5	10	50	100	200
$n^2$	0,000 025 s	0,0001 s	0,0025 s	0,01 s	0,04 s
$n^5$	0,003 125 s	0,1 s	312,5 s	3 h	89 h
$2^n$	0,000 032 s	0,001 024 s	36 a	$10^{17}$ a	$10^{47}$ a
$n^n$	0,003 125 s	3 h	$10^{71}$ a	$10^{185}$ a	$10^{447}$ a

Tabelle 3.1: Rechenzeiten eines Computers

**Definition 3.3** Es sei  $M$  eine Mehrband-Turingmaschine mit dem Eingabealphabet  $\Sigma$ . Dann ist  $\text{time}_M$  die Funktion  $\text{time}_M: \Sigma^* \rightarrow \mathbb{N}$ , wobei  $\text{time}_M(x)$  die Anzahl der Rechenschritte von  $M$  bei der Abarbeitung der Eingabe  $x$  ist.

**Beispiel 3.4** Ist  $M$  die Turingmaschine zur Berechnung der Nachfolgerfunktion aus Beispiel 2.8, so gilt  $|x| + 2 \leq \text{time}_M(x) \leq 2|x| + 2$  für alle  $x \in \{0, 1\}^+$ .

**Definition 3.5** Es seien  $\varphi : \Sigma^* \rightarrow \Sigma^*$  und  $f : \mathbb{N} \rightarrow \mathbb{N}$  totale Funktionen. Wir sagen, dass  $\varphi$  mit einem Aufwand von  $f$  berechnet werden kann, wenn eine Mehrband-Turingmaschine  $M$  existiert, die  $\varphi$  berechnet und die  $\text{time}_M(x) \leq f(|x|)$  für alle  $x \in \Sigma^*$  erfüllt.

**Beispiel 3.6** Die Nachfolgerfunktion kann mit einem Aufwand von  $f(n) = 2n + 2$  berechnet werden. Dabei ist  $n$  die Länge der Binärdarstellung des Eingabewertes  $x$ , es gilt also  $n \approx \log_2 x$ .

Bei der Untersuchung von Komplexitäten wollen wir uns wieder auf Entscheidungsprobleme, d.h. auf charakteristische Funktionen von Mengen, konzentrieren.

**Definition 3.7** Sei  $f : \mathbb{N} \rightarrow \mathbb{N}$  eine Funktion. Die Klasse  $\text{TIME}(f(n))$  besteht aus allen Mengen, die von einer Mehrband-Turingmaschine  $M$  entschieden werden können mit  $\text{time}_M(x) \leq f(|x|)$  für alle  $x \in \Sigma$ .

**Definition 3.8** Ein Polynom ist eine Funktion  $p : \mathbb{N} \rightarrow \mathbb{N}$  der Form

$$p(n) = a_k n^k + a_{k-1} n^{k-1} + \dots + a_2 n^2 + a_1 n + a_0$$

wobei  $k \in \mathbb{N}$  und  $a_i \in \mathbb{N}$  für  $i = 0, 1, 2, \dots, k$  gilt.

**Definition 3.9** Die Komplexitätsklasse  $\mathbb{P}$  ist wie folgt definiert:

$$\mathbb{P} = \bigcup_{p \text{ Polynom}} \text{TIME}(p(n))$$

Wir bemerken, dass bei der Simulation von Mehrband-TM durch (Einband-)TM eine Berechnung einer Mehrband-Turingmaschine mit  $t$  Schritten durch eine Berechnung der Einband-Turingmaschine mit  $O(t^2)$  Schritten simuliert wird. Eine Menge  $A$  liegt also genau dann in  $\mathbb{P}$ , wenn  $A$  durch eine (Einband-)TM in Polynomialzeit entschieden wird.

Die Churchsche These kann wie folgt erweitert werden: Ein Problem kann in polynomial beschränkter Zeit genau dann gelöst werden, wenn es in  $\mathbb{P}$  liegt. Um zu zeigen, dass eine Menge in  $\mathbb{P}$  liegt, reicht es zu zeigen dass es eine TM (einen Algorithmus) gibt, die (der) das Problem in einer Zeitkomplexität  $O(n^k)$  für ein  $k \in \mathbb{N}$  entscheidet.

In  $\mathbb{P}$  liegen zum Beispiel

- das Palindromproblem,
- das Problem, ob zwei Zahlen teilerfremd sind,
- das Problem, ob ein gegebener Graph planar ist,
- das Problem, ob ein gegebener Graph einen Eulerkreis enthält,

usw.

## 3.2 Das Domino-Problem

Wir haben eine endliche Anzahl von Dominosteintypen gegeben, wobei es von jedem Typ beliebig viele Exemplare gibt. Jeder Dominostein ist in vier Dreiecke aufgeteilt und in jedem dieser Dreiecke steht ein Symbol, das aus einer endlichen Menge  $\Sigma$  kommt. Ein Dominostein hat also die folgende Form:



Außerdem ist ein Rahmen vorgegeben, dessen Rand in Zellen aufgeteilt ist. Jede Zelle ist genau so groß wie ein Dominostein und ist mit einem Symbol versehen.

Die Frage ist, ob dieses Domino-Spiel eine Lösung hat, d.h. ob wir den Rahmen mit den Dominosteinen ausfüllen können, so daß

- für jedes Paar von benachbarten Dominosteinen  $s$  und  $s'$ , die zwei Dreiecke, die die gemeinsame Kante von  $s$  und  $s'$  enthalten, das gleiche Symbol haben, und
- jedes Dreieck, das den Rand des Rahmens berührt, das gleiche Symbol hat, wie die Zelle des Randes, die berührt wird.

Bei dem Ausfüllen des Rahmens dürfen die Dominosteine nicht gedreht werden.

Bei der Beschäftigung mit dem Domino-Problem fällt uns folgendes auf:

1. Man kann ein gegebenes Problem durch systematisches Probieren lösen. Eine bessere Lösung ist im allgemeinen Fall nicht zu sehen. Es ist auch tatsächlich unbekannt, ob das Domino-Problem in Polynomialzeit gelöst werden kann.
2. Hat man dagegen einen mit Domino-Steinen ausgefüllten Rahmen, so kann man in polynomieller Zeit bezüglich der Rahmengröße entscheiden, ob die Ausfüllung korrekt ist.

Diese beiden Eigenschaften (keine bessere Lösung als systematisches Probieren in Sicht, Verifizierung einer möglichen Lösung in Polynomialzeit) besitzen auch viele andere Probleme. Im folgenden Abschnitt soll diese Klasse von Problemen formal definiert werden.

### 3.3 Nichtdeterministische Turingmaschinen und die Klasse NP

**Definition 3.10** Eine nichtdeterministische Turingmaschine ist gegeben durch ein 7-Tupel  $M = (Z, \Sigma, \Gamma, \delta, z_0, \square, E)$ . Hierbei sind

- $Z$  eine endliche Menge (Zustandsmenge),
- $\Sigma$  ein Alphabet (Eingabealphabet),
- $\Gamma$  ein Alphabet (Bandalphabet) mit  $\Sigma \subseteq \Gamma$ ,
- $\delta: Z \times \Gamma \rightarrow 2^{Z \times \Gamma \times \{L, R, N\}}$  eine Funktion (Überföhrungsfunktion),
- $z_0 \in Z$  (Anfangszustand),
- $\square \in \Gamma \setminus \Sigma$  (Leerzeichen, Blank),
- $E \subseteq Z$  (Menge der Endzustände).

Die NTM unterscheidet sich von der TM lediglich durch die Überföhrungsfunktion, die für einen gegebenen Zustand und ein gegebenes Bandsymbol mehrere Aktionen erlaubt. Konfigurationen sind für NTM genauso definiert wie für TM. Formal definieren wir die Arbeitsweise einer NTM, d.h. die Nachfolgerelation  $\vdash$ , wie folgt:

**Definition 3.11** Wir definieren in der Menge der Konfigurationen einer nichtdeterministischen Turingmaschine  $M = (Z, \Sigma, \Gamma, \delta, z_0, \square, E)$  die binäre Relation „ $\vdash$ “ wie folgt. Es gilt

$$a_1 \dots a_m z b_1 \dots b_n \vdash \begin{cases} a_1 \dots a_m z' c b_2 \dots b_n & \text{falls } (z', c, N) \in \delta(z, b_1), m \geq 0, n \geq 1, \\ a_1 \dots a_m c z' b_2 \dots b_n & \text{falls } (z', c, R) \in \delta(z, b_1), m \geq 0, n \geq 2, \\ a_1 \dots a_{m-1} z' a_m c b_2 \dots b_n & \text{falls } (z', c, L) \in \delta(z, b_1), m \geq 1, n \geq 1, \end{cases}$$

$$a_1 \dots a_m z b_1 \vdash a_1 \dots a_m c z \square \text{ falls } (z', c, R) \in \delta(z, b_1), m \geq 0,$$

$$z b_1 \dots b_n \vdash z \square c b_2 \dots b_n \text{ falls } (z', c, L) \in \delta(z, b_1), n \geq 1.$$

**Beispiel 3.12** Es sei  $\delta(z, a) = \{(z_1, a, R), (z_2, b, L), (z_3, c, N)\}$ . Dann gelten für die Konfiguration  $ac zab b$  folgende Nachfolgebeziehungen:

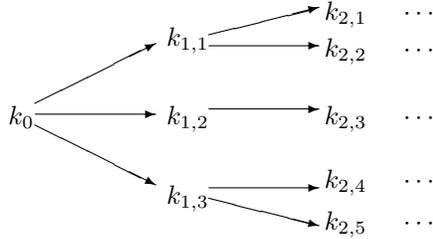
$$ac zab b \vdash ac az_1 b b, \quad ac zab b \vdash az_2 c b b b, \quad ac zab b \vdash ac z_3 c b b.$$

Die Arbeitsweisen von TM und NTM lassen sich wie folgt veranschaulichen:

**TM:** Es gibt für jede Eingabe *genau einen* Berechnungspfad:

$$k_0 \vdash k_1 \vdash k_2 \vdash \dots$$

**NTM:** Es gibt für jede Eingabe einen *Baum* der möglichen Berechnungspfade:



Da es zu einer Eingabe einer NTM in der Regel mehrere mögliche Berechnungen gibt, definieren wir für die NTM keine berechnete Funktion, sondern die *akzeptierte Menge*.

**Definition 3.13** Sei  $M = (Z, \Sigma, \Gamma, \delta, z_0, \square, E)$  eine nichtdeterministische Turingmaschine. Die von  $M$  akzeptierte Menge  $T(M)$  ist definiert durch

$$T(M) = \{x \in \Sigma^* \mid z_0 x \vdash^* wzy \text{ für } z \in E, w, y \in \Gamma^*\}.$$

Eine Eingabe  $x$  wird also genau dann von der NTM akzeptiert, wenn *eine der möglichen* Berechnungen zu einem Endzustand führt, d.h. wenn es im zugehörigen Berechnungsbaum eine Endkonfiguration gibt. Die bisher betrachteten (deterministischen) Turingmaschinen sind ein Spezialfall der nichtdeterministischen Turingmaschinen. Bezüglich der akzeptierten Sprachen sind nichtdeterministische und deterministische Turingmaschinen äquivalent.

**Satz 3.14** Für jede NTM  $M$  existiert eine TM  $M'$  mit  $T(M') = T(M)$ .

*Beweis.* Die Beweisidee (*Dovetailing*) ist wie folgt:

- Bestimme für wachsendes  $n$  alle Konfigurationen, die in  $n$  Schritten erreichbar sind.
- Wird eine Endkonfiguration erreicht, so akzeptiere.

Etwas genauer erfolgt die Simulation der NTM  $M$  durch eine 2-Band-TM mit folgender Arbeitsweise:

- Band 1 enthält alle nach  $n$  Schritten von  $M$  erreichbaren Konfigurationen (zuerst also die Startkonfiguration).
- Ist eine der Konfigurationen auf Band 1 eine Endkonfiguration von  $M$ , so akzeptiere.
- Anderenfalls schreibe für jede Konfiguration auf Band 1 alle Nachfolgekonfigurationen auf Band 2. Lösche dabei Band 1.
- Verschiebe den Inhalt von Band 2 nach Band 1. Band 1 enthält nun alle nach  $n+1$  Schritten erreichbaren Konfigurationen. Beginne nun wieder von vorn.

□

**Folgerung 3.15** Eine Menge ist genau dann semi-entscheidbar, wenn sie durch eine NTM akzeptiert wird.

*Beweis.* Man kann aus einer TM, die eine Menge  $A$  akzeptiert, sehr leicht eine TM konstruieren, die die "halbe" charakteristische Funktion  $\chi'_A$  berechnet, und umgekehrt. Die Behauptung folgt dann aus der Äquivalenz von TM und NTM. □

Als nächstes sollen nichtdeterministische Komplexitätsklassen definiert werden.

**Definition 3.16** Sei  $M = (Z, \Sigma, \Gamma, \delta, z_0, \square, E)$  eine nichtdeterministische Turingmaschine; die Funktion  $time_M : \Sigma^* \rightarrow \mathbb{N}$  ist definiert durch

$$time_M(x) = \begin{cases} \text{Minimum der Längen aller akzeptierenden Rechnungen von } M \text{ auf } x & \text{falls } x \in T(M), \\ 0 & \text{falls } x \notin T(M). \end{cases}$$

Für eine von  $M$  akzeptierte Eingabe  $x$  ist  $time_M(x)$  also die *kleinste Tiefe* einer Endkonfiguration im zugehörigen Berechnungsbaum.

**Definition 3.17** Sei  $f: \mathbb{N} \rightarrow \mathbb{N}$  eine totale Funktion. Die Klasse  $NTIME(f(n))$  besteht aus allen Mengen  $A$ , die von einer nichtdeterministischen Mehrband-Turingmaschine  $M$ <sup>9</sup> akzeptiert werden können mit  $time_M(x) \leq f(|x|)$  für alle Eingaben  $x$ .

**Definition 3.18** Die Komplexitätsklasse  $\mathbb{NP}$  ist wie folgt definiert:

$$\mathbb{NP} = \bigcup_{p \text{ Polynom}} NTIME(p(n))$$

Erneut gehört eine Menge  $A$  genau dann zu  $\mathbb{NP}$ , wenn  $A$  durch eine nichtdeterministische (Einband-)TM in Polynomialzeit akzeptiert wird.

Abbildung 3.1 gibt einen Überblick über die mengentheoretischen Beziehungen der bisher definierten Klassen. Dabei heißt eine Menge “LOOP-berechenbar”, wenn ihre charakteristische Funktion LOOP-berechenbar ist. Es ist bisher unbekannt, ob die Inklusion  $\mathbb{P} \subseteq \mathbb{NP}$  echt ist. Diese als “ $\mathbb{P} = \mathbb{NP}$  - Problematik” bekannte Frage ist eines der wichtigsten offenen Probleme der heutigen Mathematik. Alle anderen Inklusionen sind echt.

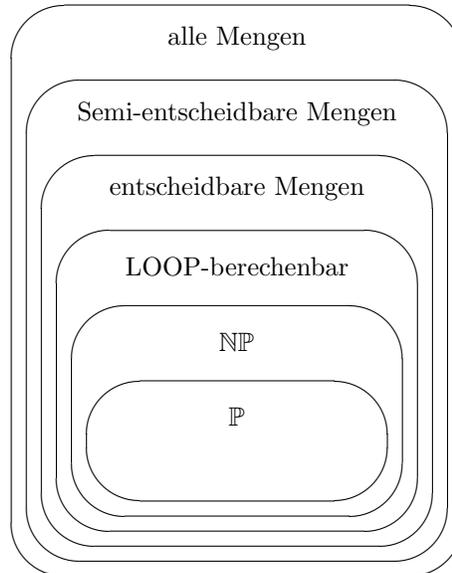


Abbildung 3.1: Komplexitätsklassen

<sup>9</sup>Nichtdeterministische Mehrband-Turingmaschinen lassen sich analog zu deterministischen Mehrband-Turingmaschinen definieren. Wir verzichten auf eine formale Definition, da dieser Begriff nur in dieser Definition benötigt wird.

### 3.4 NP-Vollständigkeit

Ob es ein Problem aus  $\text{NP}$  gibt, das nicht in Polynomialzeit lösbar ist, weiß man also zur Zeit nicht. Wir wollen in diesem Unterabschnitt die Klasse der  $\text{NP}$ -vollständigen Probleme kennen lernen. Diese können im folgenden Sinne als die “schwierigsten Probleme” dieser Klasse  $\text{NP}$  angesehen werden: Kann ein  $\text{NP}$ -vollständiges Problem in Polynomialzeit lösen, so kann man alle Probleme aus  $\text{NP}$  in Polynomialzeit lösen. Wir werden insbesondere sehen, dass das Domino-Problem  $\text{NP}$ -vollständig ist.

**Definition 3.19** Seien  $A, B \subseteq \Sigma^*$  Mengen. Dann heißt  $A$  auf  $B$  polynomial reduzierbar (in Zeichen  $A \leq_p B$ ) falls es eine totale und in polynomialer Zeit berechenbare Funktion  $f: \Sigma^* \rightarrow \Sigma^*$  gibt, so dass für alle  $x \in \Sigma^*$  gilt:

$$x \in A \implies f(x) \in B.$$

Wir wollen hier nur eine einfache Reduktion vorführen. Wir definieren die folgenden Mengen:

$$\mathbf{SOS} := \{(a_1, \dots, a_m, b) : m, a_1, \dots, a_m, b \in \mathbb{N} \text{ und es gibt } c_1, \dots, c_m \in \{0, 1\}, \text{ so daß } \sum_{i=1}^m c_i a_i = b\}$$

und

$$\begin{aligned} \mathbf{RS} &:= \{(g_1, \dots, g_m, k_1, \dots, k_m, G, K) : \\ & m, g_1, \dots, g_m, k_1, \dots, k_m, G, K \in \mathbb{N} \\ & \text{und es gibt } c_1, \dots, c_m \in \{0, 1\}, \\ & \text{so daß } \sum_{i=1}^m c_i g_i \leq G \text{ und } \sum_{i=1}^m c_i k_i \geq K\}. \end{aligned}$$

$\mathbf{SOS}$  ist als Teilmengensummenproblem (*Sum of Subsets*) bekannt;  $\mathbf{RS}$  steht für *Rucksackproblem*: Wir haben  $m$  Päckchen mit Lebensmitteln. Das  $i$ -te Päckchen hat Gewicht  $g_i$  und enthält  $k_i$  Kalorien. Wir wollen entscheiden, ob wir einen Rucksack mit einer Teilmenge der Päckchen packen können, so daß das Gesamtgewicht höchstens  $G$  und die Kalorienzahl mindestens  $K$  ist.

**Satz 3.20**  $\mathbf{SOS} \leq_p \mathbf{RS}$ .

*Beweis.* Nach Definition suchen wir eine in Polynomialzeit berechenbare Funktion  $f$ , die Eingaben für  $\mathbf{SOS}$  auf Eingaben für  $\mathbf{RS}$  abbildet, so daß

$$(a_1, \dots, a_m, b) \in \mathbf{SOS} \iff f(a_1, \dots, a_m, b) \in \mathbf{RS}.$$

Damit  $f(a_1, \dots, a_m, b)$  eine Eingabe für  $\mathbf{RS}$  ist, muß dieser Funktionswert also die Form

$$f(a_1, \dots, a_m, b) = (g_1, \dots, g_m, k_1, \dots, k_m, G, K)$$

haben. Wir definieren

$$f(a_1, \dots, a_m, b) := (a_1, \dots, a_m, a_1, \dots, a_m, b, b).$$

Es ist klar, daß  $f$  in polynomieller Zeit berechenbar ist. Außerdem gilt

$$\begin{aligned} (a_1, \dots, a_m, b) \in \mathbf{SOS} & \iff \text{es gibt } c_1, \dots, c_m \in \{0, 1\} \text{ mit } \sum_{i=1}^m c_i a_i = b \\ & \iff \text{es gibt } c_1, \dots, c_m \in \{0, 1\} \text{ mit } \sum_{i=1}^m c_i a_i \leq b \text{ und } \sum_{i=1}^m c_i a_i \geq b \\ & \iff (a_1, \dots, a_m, a_1, \dots, a_m, b, b) \in \mathbf{RS} \\ & \iff f(a_1, \dots, a_m, b) \in \mathbf{RS}. \end{aligned}$$

□

**Lemma 3.21** (i) Gilt  $A \leq_p B$  und  $B \in \text{NP}$ , so ist auch  $A \in \text{NP}$ .

(ii) Gilt  $A \leq_p B$  und  $B \in \mathbb{P}$ , so ist auch  $A \in \mathbb{P}$ .

*Beweis.* Wir zeigen nur die erste Behauptung; der Beweis der zweiten Behauptung erfolgt analog. Es gelte  $A, B \subseteq \Sigma^*$ ,  $A \leq_p B$  und  $f : \Sigma^* \rightarrow \Sigma^*$  sei die nach Definition verlangte Funktion mit  $x \in A \iff f(x) \in B$ . Es sei  $M_1$  eine TM, die  $f$  berechnet und  $\text{time}_{M_1}(x) \leq p(|x|)$  für ein Polynom  $p$  und alle  $x \in \Sigma^*$  erfüllt. Da  $B \in \mathbb{NP}$  gilt, existiert eine NTM  $M_2$ , die  $B$  akzeptiert und  $\text{time}_{M_2}(y) \leq q(|y|)$  für ein Polynom  $q$  und alle  $y \in \Sigma^*$  erfüllt.

Wir konstruieren eine NTM  $M$  mit  $T(M) = A$ , indem auf eine Eingabe  $x$  zunächst die TM  $M_1$  angesetzt wird und anschließend die NTM  $M_2$  mit dem Ergebnis  $f(x)$  weiter arbeitet. Da  $M_1$  zur Bearbeitung von  $x$  höchstens  $p(|x|)$  Schritte ausführt, gilt  $|f(x)| \leq p(|x|)$ , wobei wir o.B.d.A.  $p(n) \geq n$  für alle  $n \in \mathbb{N}$  voraussetzen. Wir erhalten  $\text{time}_M(|x|) \leq p(|x|) + q(p(|x|))$ , d.h.  $\text{time}_M$  ist polynomiell beschränkt.  $\square$

**Definition 3.22** Eine Menge  $A$  heißt  $\mathbb{NP}$ -vollständig genau dann, wenn

- (i)  $A \in \mathbb{NP}$  und
- (ii) für alle Mengen  $A' \in \mathbb{NP}$   $A' \leq_p A$  gilt.

Mengen, die Bedingung (ii) erfüllen, werden auch  $\mathbb{NP}$ -hart genannt.

Dann gilt offensichtlich folgender Satz.

**Satz 3.23** Sei  $A$   $\mathbb{NP}$ -vollständig, dann gilt

$$A \in \mathbb{P} \iff \mathbb{P} = \mathbb{NP}.$$

Wir wollen als nächstes zeigen, dass das Domino-Problem  $\mathbb{NP}$ -vollständig ist. Dazu müssen wir nach Definition beweisen, dass man jede Menge  $A \in \mathbb{NP}$  in Polynomialzeit auf das Domino-Problem reduzieren kann.

**Satz 3.24 Domino** ist  $\mathbb{NP}$ -vollständig.

*Beweis.* Es ist klar, daß **Domino**  $\in \mathbb{NP}$ : Eine Lösung besteht aus einer Auslegung des Rahmens mit Dominosteinen. Die Anzahl der Bits, die wir brauchen um so eine Auslegung darzustellen ist polynomiell in der Länge der Eingabe. Außerdem können wir in Polynomialzeit überprüfen, ob eine vorgegebene Lösung korrekt ist.

Wir müssen also noch beweisen, daß

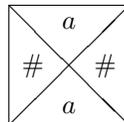
$$A \leq_p \mathbf{Domino} \text{ für alle Mengen } A \in \mathbf{NP}.$$

Sei  $A$  eine Menge aus  $\mathbb{NP}$ . Dann gibt es ein Polynom  $p$  und eine nichtdeterministische Turing-Maschine  $M = (Z, \Sigma, \Gamma, \delta, z_0, \square, E)$ , die die Menge  $A$  in der Zeit  $p$  akzeptiert. Wir dürfen o.B.d.A. annehmen, dass am Ende einer erfolgreichen Berechnung von  $M$  das Band leer ist.

Wir konstruieren jetzt für  $M$  ein Dominospiel  $\Pi_M$ . Ein Eingabewort  $w$  wird dann derart auf einen Rahmen  $R_w$  transformiert, dass eine akzeptierende Berechnung von  $M$  einer Auslegung des Rahmens  $R_w$  mit Steinen aus  $\Pi_M$  entspricht.

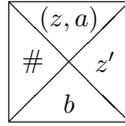
Das Dominospiel hat die folgenden Typen von Dominosteinen:

1. Für jedes Bandsymbol  $a$  der Turing-Maschine  $M$ :



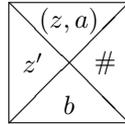
Bedeutung: Vor und nach dem Befehl steht der Kopf nicht in diesem Feld.

2. Für jeden Befehl  $(z, a) \rightarrow (z', b, R)$  der Turing-Maschine  $M$ :



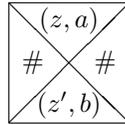
Bedeutung: Vor dem Befehl steht der Kopf in diesem Feld; er macht einen Schritt nach rechts.

3. Für jeden Befehl  $(z, a) \rightarrow (z', b, L)$  der Turing-Maschine  $M$ :



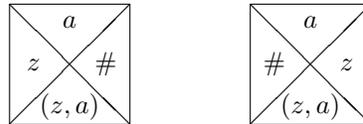
Bedeutung: Vor dem Befehl steht der Kopf in diesem Feld; er macht einen Schritt nach links.

4. Für jeden Befehl  $(z, a) \rightarrow (z', b, N)$  der Turing-Maschine  $M$ :



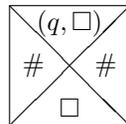
Bedeutung: Vor und nach dem Befehl steht der Kopf in diesem Feld.

5. Für jeden Zustand  $z$  und jedes Alphabet-Symbol  $a$  der Turing-Maschine  $M$  gibt es zwei Dominosteintypen:



Bedeutung: Der linke Dominosteintyp gibt an, daß der Kopf von links in dieses Feld kommt; der rechte Dominosteintyp gibt an, daß der Kopf von rechts in dieses Feld kommt.

6. Für jeden Endzustand  $q$  der Turing-Maschine  $M$  gibt es den Dominosteintyp:



Bedeutung: Nach einer akzeptierenden Rechnung werden die Information über den Endzustand und die Position des Lese/Schreibkopfes gelöscht.

Damit liegen die Dominosteintypen fest. Der Rahmen  $R_w$  für ein Eingabewort  $w = a_1 a_2 \dots a_n$  ist in Abbildung 3.2 dargestellt. Der obere Rand dieses Rahmens entspricht dem Anfang einer Berechnung, der untere Rand entspricht dem Ende einer akzeptierenden Berechnung. Der linke und rechte Rand entsprechen Grenzen für den Kopf. Offenbar kann  $R_w$  in Polynomialzeit bezüglich  $|w|$  bestimmt werden.

Für eine Lösung des Dominospiels gelten folgende Feststellungen:

- Der obere Teil der ersten Zeile enthält die Startkonfiguration von  $M$  für die Eingabe  $w$ . Der untere Teil der letzten Zeile enthält nur Blanks.
- Enthält der obere Teil einer Zeile eine Konfiguration  $k$  der NTM  $M$ , so enthält der untere Teil dieser Zeile eine Nachfolgekongfiguration von  $k$ , falls  $k$  keine Endkonfiguration ist, bzw. nur Blanks, falls  $k$  eine Endkonfiguration ist.
- Der obere Teil einer Zeile hat den gleichen Inhalt wie der untere Teil der Vorgängerzeile.

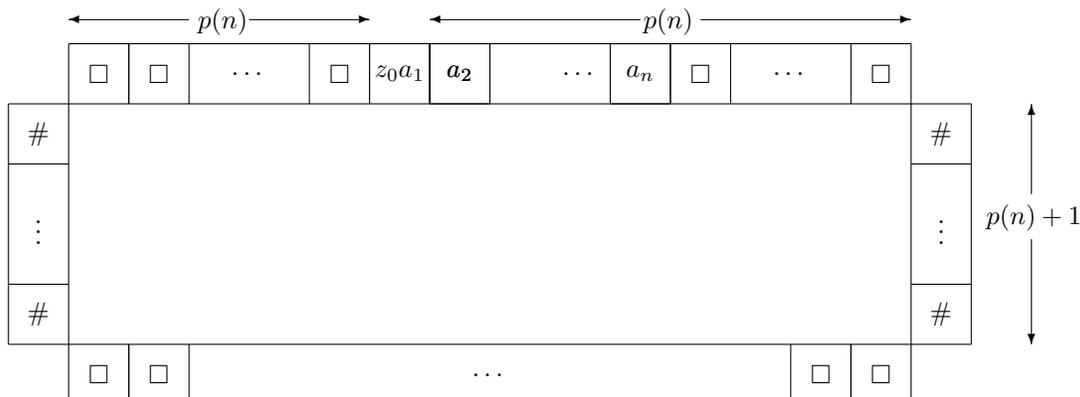


Abbildung 3.2: Rahmen des Dominospiels.

Somit entspricht eine Lösung des Domino-Problems für den Rahmen  $R_w$  einer akzeptierenden Rechnung von  $M$  für die Eingabe  $w$  mit höchstens  $p(|w|)$  Schritten. Umgekehrt kann man aus jeder akzeptierenden Rechnung mit höchstens  $p(|w|)$  Schritten eine Lösung des Domino-Problems konstruieren. Es gilt also:

$$\begin{aligned}
 w \in A &\iff \text{es gibt mindestens eine akzeptierende Berechnung von } M, \\
 &\quad \text{die höchstens } p(n) \text{ Schritte macht} \\
 &\iff \text{das Dominoproblem } (\Pi_M, R_w) \text{ ist lösbar.}
 \end{aligned}$$

Damit haben wir nachgewiesen, dass  $A \leq_p \mathbf{Domino}$  für jede beliebige Menge  $A \in \mathbf{NP}$  gilt. Also ist die Menge **Domino** **NP**-vollständig.  $\square$

**Satz 3.25** Sei  $A$  **NP**-vollständig,  $B \in \mathbf{NP}$  und gelte  $A \leq_p B$ . Dann ist auch  $B$  **NP**-vollständig.

Der letzte Satz hat große Bedeutung bei der Untersuchung von Problemen. Will man nämlich zeigen, dass ein Problem  $B$  **NP**-vollständig ist, so braucht man nicht auf die Definition zurück zu gehen, sondern es reicht die Reduktion eines **NP**-vollständigen Problems  $A$ .

Ohne Beweis führen wir hier an, dass folgende Probleme **NP**-vollständig sind (für die genauen Definitionen siehe Literatur).

- **SAT**, das Erfüllbarkeitsproblem der Aussagenlogik:

**Eingabe:** Boolesche Formel  $\varphi$  in konjunktiver Normalform

**Frage:** Ist die Formel  $\varphi$  erfüllbar?

Für dieses Problem wird die **NP**-Vollständigkeit in den meisten Lehrbüchern direkt gezeigt. Man kann aber auch relativ einfach zeigen, dass **Domino**  $\leq_p$  **SAT** gilt.

- **3SAT** (wie **SAT**, jedoch höchstens 3 Literale pro Alternative).

- **Clique**, das Cliquesproblem:

**Eingabe:** ungerichteter Graph  $G$ ,  $k \in \mathbb{N}$

**Frage:** Enthält  $G$  einen vollständigen Teilgraphen mit  $k$  Knoten?

- **Set Partition**, das Partitionierungsproblem:

**Eingabe:** Menge  $U$ , Gewichtsfunktion  $g : U \rightarrow \mathbb{N}$

**Frage:** Gibt es eine Zerlegung  $U = V \cup W$ ,  $V \cap W = \emptyset$ ,

$$\text{so dass } \sum_{v \in V} g(v) = \sum_{w \in W} g(w)?$$

- **Bin Packing**:

- Eingabe:**  $m$  Gegenstände mit Volumen  $a_1, a_2, \dots, a_m$ ,  
 $k$  Behälter mit Volumen jeweils  $\ell$ .
- Frage:** Kann man die Gegenstände auf die  $k$  Behälter verteilen?
- Hamiltonkreis-Problem:  
**Eingabe:** gerichteter Graph  $G$   
**Frage:** Gibt es in  $G$  einen Kreis, der jeden Knoten genau einmal passiert?
  - TSP, das Rundreiseproblem:  
**Eingabe:** Schranke  $M \in \mathbb{N}$ ,  $n$  Städte,  $n \times n$ -Kostenmatrix  $C$  mit  
 $C_{i,j}$  = Kosten einer Fahrt von Stadt  $i$  nach Stadt  $j$   
**Frage:** Gibt es eine Rundreise durch alle Städte  
mit Kosten von höchstens  $M$ ?

Eine umfangreiche Sammlung NP-vollständiger Probleme findet man im Buch von GAREY und JOHNSON [6].