2

Reguläre Sprachen und endliche Automaten

Sei $\Sigma = \{a, b, \ldots\}$ ein endliches *Alphabet*. Ein *endliches Wort* über Σ ist eine Folge $w = a_0 \ldots a_{n-1}$, wobei $a_i \in \Sigma$ für $i = 0, \ldots, n-1$. Wir schreiben $|w|$ für die *Länge* von w, also n in diesem Fall, und $w(i)$ für das i-te Symbol a_i von w. Das *leere Wort*, also die Folge der Länge 0, wird mit ϵ bezeichnet. Σ^* bezeichnet die Menge aller Wörter über Σ und Σ^+ die Menge aller nichtleeren Wörter über Σ. Die *Konkatenation* zweier Wörter w und v entsteht durch Anhängen der Folge v an die Folge w und wird mit wv bezeichnet.

Eine *Sprache* ist eine Menge von Wörtern, also eine Teilmenge $L \subseteq \Sigma^*$. Somit sind alle üblichen, mengentheoretischen Operationen wie Vereinigung, Schnitt, Komplement, Differenz, etc. auch Operationen auf Sprachen. Das *Komplement* einer Sprache L bezeichnen wir üblicherweise mit $\overline{L}$, und es ist definiert als $\Sigma^* \setminus L$.

Andere wichtige Operatioen auf Sprachen sind Konkatenation und Kleene-Abschluss (Kleene-Stern). Sind L_1 und L_2 zwei Sprachen, so ist deren *Konkatenation* die Sprache $L_1 L_2 := \{w_1 w_2 \mid w_1 \in L_1, w_2 \in L_2\}$, die also durch Konkatenation beliebiger Wörter aus L_1 mit beliebigen Wörtern aus L_2 entsteht.

Ist L eine Sprache, dann ist ihr Kleene-Abschluss die Sprache $L^* := \{w_1 \ldots w_n \mid n \in \mathbb{N}, w_i \in L \text{ für alle } i = 1, \ldots, n\}$. Diese entsteht also durch beliebig, aber nur endlich, oft wiederholte Konkatenation der Sprache mit sich selbst. Beachte, dass $\epsilon \in L^*$ für jedes $L \subseteq \Sigma^*$ gilt, denn auch $n = 0$ ist in der Definition des Kleene-Abschlusses zugelassen. Anders gesagt gilt für alle $L \subseteq \Sigma^*$:

$$L^0 := \{\epsilon\} \qquad L^{i+1} := LL^i \qquad L^* := \bigcup_{i \in \mathbb{N}} L^i$$

Eine Klasse von Sprachen, für die wir uns hier insbesondere interessieren, ist die der *regulären Sprachen* über dem Alphabet Σ, bezeichnet mit REG_Σ. Kann Σ aus dem Kontext erkannt werden, so schreiben wir auch einfach nur REG. Sie ist definiert als die kleinste Klasse von Sprachen, für die gilt:

1. $\emptyset \in \mathrm{REG}_\Sigma$, $\{\epsilon\} \in \mathrm{REG}_\Sigma$ und $\{a\} \in \mathrm{REG}_\Sigma$ für jedes $a \in \Sigma$,
2. wenn $L_1, L_2 \in \mathrm{REG}$, dann $L_1 \cup L_2 \in \mathrm{REG}$, $L_1 L_2 \in \mathrm{REG}$ und $L_1^* \in \mathrm{REG}$.

Hier bedeutet *kleinste Klasse*, dass keine Sprache, die nicht durch (1) oder (2) abgedeckt ist, zu REG gehört. Dies ist gleichbedeutend damit, dass eine Sprache nur dann zu REG gehört, wenn sie entweder von einer in (1) genannten Form ist, oder aber sich laut (2) aus einer oder zweien Sprachen zusammensetzt, von denen bereits bekannt ist, dass sie zu REG gehören. Noch anders gesagt: Jede reguläre Sprache lässt sich durch Anwenden von nur *endlich vielen* Operationen, die in (2) genannt sind, aus den Sprachen, die in (1) genannt sind, erzeugen.

Man sieht z.B. leicht, dass jede endliche Menge von Wörtern eine reguläre Sprache bildet, da sich jede Sprache, die nur ein Wort enthält durch endlich viele Konkatenationen von Sprachen der Form $\{a\}$ bilden lässt. Eine Sprache mit endlich vielen Wörtern lässt sich dann leicht als endliche Vereinigung beschreiben. Beachte, dass die Klasse der regulären Sprachen nicht unter unendlichen Vereinigungen abgeschlossen ist, denn dann wäre ja jede Sprache regulär, da sich jede Sprache als unendliche Vereinigung der Einermengen ihrer Wörter schreiben lässt.

Der Punkt (2) in der Definition von REG bedeutet, dass REG *abgeschlossen* ist unter den Operationen Vereinigung, Konkatenation und Kleene-Iteration. REG ist darüberhinaus noch abgeschlossen unter einer ganzen Menge anderer Operationen, u.a. unter all den üblichen mengentheoretischen Operationen. Dies ist jedoch für den Schnitt und das Komplement nicht unbedingt offensichtlich aus der Definition. Um solche Abschlusseigenschaften zu beweisen, braucht man in der Regel andere Charakterisierungen der regulären Sprachen. Diese sind auch nötig, um reguläre Sprachen repräsentieren und verarbeiten zu können, da es sich bei Sprachen im Allgemeinen um unendlich große Mengen handelt. Wir behandeln im folgenden drei solcher Charakterisierungen: reguläre Ausdrücke, nicht-deterministische endliche Automaten, deterministische endliche Automaten.

Definition 1. *Die Syntax von* regulären Ausdrücken *über dem Alphabet Σ ist die Folgende.*

$$\alpha \ ::= \ \emptyset \mid \epsilon \mid a \mid \alpha\alpha \mid \alpha \cup \alpha \mid \alpha^*$$

wobei $a \in \Sigma$. In anderen Worten: Es gibt die atomaren, regulären Ausdrücke $\emptyset$, ϵ und a für jedes $a \in \Sigma$, und reguläre Ausdrücke lassen sich aus denen durch Verwendung der zweistelligen Symbole für Vereinigung und Konkatenation (hier lediglich durch Hintereinanderschreiben bezeichnet) sowie des einstelligen Stern-Symbols zusammensetzen.

Die Größe $|\alpha|$ eines regulären Ausdrucks α ist definiert, als die Zahl der in ihm enthaltenen Symbole ohne Klammern, also z.B.: $|a(b \cup c)^*| = 5$. Jedem regulären Ausdruck α wird induktiv über seinen Aufbau eine Sprache $L(\alpha)$ über Σ zugeordnet.

$$\begin{array}{ll} L(\emptyset) := \emptyset & L(\alpha\beta) := L(\alpha)L(\beta) \\ L(a) := \{a\} & L(\alpha \cup \beta) := L(\alpha) \cup L(\beta) \\ L(\epsilon) := \{\epsilon\} & L(\alpha^*) := (L(\alpha))^* \end{array}$$

Wir benutzen auch die Abkürzung $\alpha^+ := \alpha\alpha^*$.

Theorem 1. *Für alle Sprachen L gilt: L is regulär gdw. es einen regulären Ausdruck* α *gibt mit* $L = L(\alpha)$.

Man beachte, dass verschiedene reguläre Ausdrücke durchaus die gleiche Sprache definieren können. Zum Beispiel ist $L(\alpha_1) = L(\alpha_2)$ für $alpha_1 := (ab)^*(ba)^*$ und $\alpha_2 := \epsilon \cup a(\epsilon \cup (ba)^*bb(ab)^*)b$ über dem Alphabet $\Sigma = \{a, b\}$.

Die Frage, ob zwei reguläre Ausdrücke dieselbe Sprache definieren lässt sich anhand der syntaktischen Struktur der Ausdrücke nur mühsam erkennen.

Charakterisierungen der regulären Sprachen mit Automaten, die wir nunmehr einführen, erlauben die algorithmische Beantwortung solcher Fragen auf recht direkte Weise.

Definition 2. *Ein* nicht-deterministischer Automat *(NFA) ist ein Tupel* $\mathcal{A} = (Q, \Sigma, q_I, \delta, F)$ *mit*

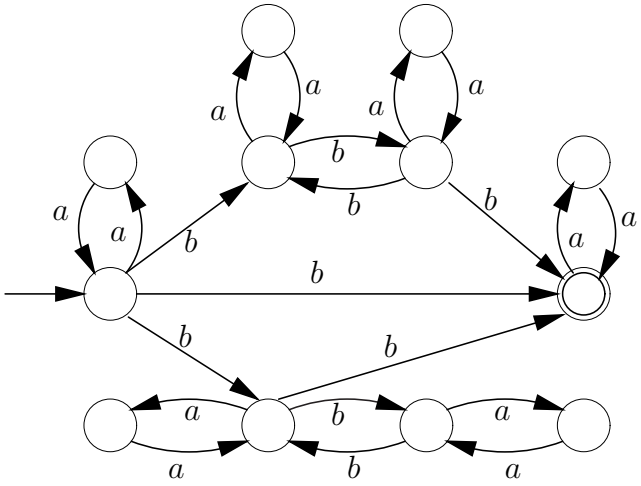
- *endlicher* Zustandsmenge Q,
- Eingabealphabet Σ,
- Startzustand $q_I \in Q$,
- Transitionsfunktion $\delta : Q \times \Sigma \to 2^Q$,
- *Endzustandsmenge* $F \subseteq Q$.

Ein Lauf *eines NFA* $\mathcal{A}$ *auf einem Wort* $w = a_0 \dots a_{n-1}$ *ist eine Folge* $q_0 \dots q_n$, *so dass* $q_0 = q_I$ *und für alle* $i = 0, \dots, n-1$ *gilt:* $q_{i+1} \in \delta(q_i, a_i)$. *Der Lauf heißt* akzeptierend, *falls* $q_n \in F$. *Sei* $L(\mathcal{A}) := \{w \in \Sigma^* \mid$ *es gibt einen akzeptierenden Lauf von* $\mathcal{A}$ *auf* $w\}$ *die von* $\mathcal{A}$ erkannte Sprache.

Ein NFA $\mathcal{A}$ *heißt* deterministisch *(DFA), falls für alle* $q \in Q$ *und alle* $a \in \Sigma$ *gilt:* $|\delta(q, a)| = 1$. *In diesem Fall schreiben wir auch einfach* $\delta(q, a) = p$ *statt* $\delta(q, a) = \{p\}$.

NFAs lassen sich am einfachsten als kanten-beschrifteter Graph darstellen. Die Zustände bilden die Knoten, die Zustandsübergangsfunktion δ wird durch Kanten in dem Graph repräsentiert. Ist $q' \in \delta(q, q)$ für einen Zustand q und ein Alphabetsymbol a, so wird eine Kante von q nach q' mit Beschriftung a gezogen. Normalerweise wird der Anfangszustand durch eine von nirgendwoher eingehende und unbeschriftete Kante gekennzeichnet und Endzustände durch einen Doppelkreis ausgezeichnet.

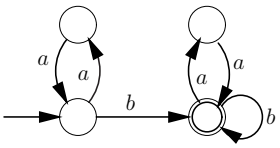
Beispiel 1. Sei $\mathcal{A}$ der folgende NFA.



Dann gilt

$$L(\mathcal{A}) \;=\; (aa)^*\Big(\; b \;\cup\; \big(b(aa)^*b(aa)^*\big)^+ b \;\cup\; b\big((aa)^*b(aa)^*b\big)^*b\Big)(aa)^*$$

Mit anderen Worten: L besteht aus der Menge aller Wörter, in denen der Buchstabe a nur in Blöcken gerader Länge vorkommt und die entweder ein b, oder eine ungerade Anzahl ≥ 3 oder eine gerade Anzahl ≥ 2 des Buchstaben b enthalten. Diese Sprache lässt sich auch einfacher beschreiben: $L(\mathcal{A}) = (aa)^*b\big((aa)^* \cup b\big)^*$. Außerdem gilt $L(\mathcal{A}) = L(\mathcal{A}')$ für den folgenden NFA $\mathcal{A}'$.



Viele der Operationen, unter denen die Klasse der regulären Sprachen abgeschlossen ist, lassen sich ebenfalls auf NFAs ausführen, welches die folgenden Lemmata besagen. Wir zeigen exemplarisch die Konstruktion für den Kleene-Stern, die anderen sind Übungen.

Lemma 1. *Seien $\mathcal{A}_i$ zwei NFAs für $i = \{1, 2\}$. Dann existieren NFAs $\mathcal{A}^\cup$ und $\mathcal{A}^;$, sodass $L(\mathcal{A}^\cup) = L(\mathcal{A}_1) \cup L(\mathcal{A}_2)$ und $L(\mathcal{A}^;) = L(\mathcal{A}_1)L(\mathcal{A}_2)$.*

Beweis. Übung.

Lemma 2. *Sei $\mathcal{A}$ ein NFA. Dann existiert ein NFA $\mathcal{A}^*$, sodass $L(\mathcal{A}^*) = L(\mathcal{A})^*$.*

Beweis. Sei $\mathcal{A} = (Q, \Sigma, q_I, \delta, F)$ und q_I' ein neuer Zustand, der nicht in Q enthalten ist. Wir geben zuerst die Konstruktion von $\mathcal{A}^*$ an und beweisen dann, dass dieser NFA die gewünschte Sprache erkennt.

$$\mathcal{A}^* \;:=\; (Q \cup \{q_I'\}, \Sigma, q_I', \delta^*, F \cup \{q_I'\})$$

wobei für alle $a \in \Sigma$ und alle $q \in Q \cup \{q_I'\}$:

$$\delta^*(q,a) \;:=\; \begin{cases} \delta(q_I, a) & , \text{ falls } q = q_I' \\ \delta(q,a) \cup \delta(q_I, a) & , \text{ falls } q \in F \\ \delta(q,a) & , \text{ falls } q \in Q \setminus F \end{cases}$$

Wir beobachten, dass ein Zustand in $\mathcal{A}^*$ also dieselben Transitionen wie derselbe Zustand in $\mathcal{A}$ hat mit dem Unterschied, dass Endzustände zusätzlich noch die Transitionen des ursprünglichen Anfangszustand haben. Der neue Anfangszustand ist eine Kopie des alten Anfangszustands, der aber von keinem anderen Zustand aus erreichbar ist.

Es bleibt zu zeigen, dass $L(\mathcal{A}^*) = L(\mathcal{A})^*$ gilt. Für den $\subseteq$-Teil nehmen wir an, dass $w \in L(\mathcal{A}^*)$ für ein $w = a_1 \dots a_n \in \Sigma^*$ gilt. Also existiert ein akzeptierender Lauf $q_I', q_1, \dots, q_n$ auf w. Aus obiger Beobachtung folgt sofort, dass es $i_0, i_1, \dots, i_m$ gibt, sodass

- $i_0 = 0$,
- $i_0 < i_1 \dots < i_m$,
- $i_m = n$,
- für alle $j = 1, \dots, m$ ist $q_{i_{j-1}}, \dots, q_{i_j}$ ein akzeptierender Lauf von $\mathcal{A}$ auf $a_{i_{j-1}+1} \dots a_{i_j}$.

Somit ist $w \in L(\mathcal{A})^*$.

Für den $\supseteq$-Teil sei $w \in L(\mathcal{A})^*$. Falls $w = \epsilon$, so gilt $w \in L(\mathcal{A}^*)$, da der Anfangszustand von $\mathcal{A}^*$ auch Endzustand ist. Sei $w \neq \epsilon$. Also existiert eine Zerlegung $w = v_0 \dots v_k$, sodass $v_i \neq \epsilon$ und $v_i \in L(\mathcal{A})$ für alle $i = 0, \dots, k$.

Wir zeigen nun durch Induktion über k, dass $v_0 \dots v_k \in L(\mathcal{A}^*)$ gilt. Im Induktionsanfang ist $k = 0$, d.h. $w = v_0$. Da $v_0 \in L(\mathcal{A})$ gibt es einen akzeptierenden Lauf $q_0, q_1, \dots, q_m$ von $\mathcal{A}$ auf v_0 mit $q_m \in F$. Da $v_0 \neq \epsilon$ gilt $m \geq 1$. Jetzt ist aber $q_I', q_1, \dots, q_m$ auch ein akzeptierender Lauf von $\mathcal{A}^*$ auf v_0, da der neue Anfangszustand q_0' dieselben Transitionen wie der alte q_0 hat und q_m auch weiterhin Endzustand in $\mathcal{A}^*$ ist.

Sei nun $k > 0$ und $v_0 \dots v_{k-1} \in L(\mathcal{A}^*)$, d.h. es gibt einen akzeptierenden Lauf $q_I', q_1, \dots, q_m$ von $\mathcal{A}^*$ auf $v_0 \dots v_{k-1}$. Da $v_k \in L(\mathcal{A})$ gibt es auch einen akzeptierenden Lauf von $p_0, \dots, p_l$ von $\mathcal{A}$ auf v_k. Insbesondere gilt $p_0 = q_I$, $p_l \in F$ und $l \geq 1$, da $v_k \neq \epsilon$. Dann ist aber $q_I', q_1, \dots, q_m, p_1, \dots, p_l$ ein akzeptierender Lauf von $\mathcal{A}^*$ auf $v_0 \dots v_k$, denn es gilt $p_1 \in \delta^*(q_m, v_k(0))$, da $p_1 \in \delta(q_I, v_k(0))$.

Theorem 2. *Wird L von einem regulären Ausdruck α beschrieben, so wird L auch von einem NFA $\mathcal{A}$ erkannt, sodass $|\mathcal{A}| = \mathcal{O}(|\alpha|)$ gilt.*

Wir überlassen das genaue Führen dieses Beweises als Übungsaufgabe. Die Konstruktion eines NFAs kann induktiv über den Aufbau regulärer Ausdrücke erfolgen, wobei der Induktionsschritt die obigen Lemmas benutzt. Man beachte jedoch, dass Aussage über die Größe des NFAs zu schwach ist, um durch Induktion bewiesen zu werden: Es gilt zwar z.B. $\mathcal{O}(n) + \mathcal{O}(n) = \mathcal{O}(n)$; wenn man aber eine Anzahl von Termen der Form $\mathcal{O}(n)$ addiert, die wiederum selbst von n abhängt, so ist die Summe im allgemeinen nicht mehr linear in n.

Die Umkehrung des obigen Satzes gilt ebenfalls, allerdings braucht man dazu z.B. das *Arden'sche Lemma*.

Lemma 3. *Seien $U, V \subseteq \Sigma^*$ Sprachen, sodass $\epsilon \notin U$, und $L \subseteq \Sigma^*$ eine Sprache, sodass $L = UL \cup V$. Dann gilt $L = U^*V$.*

Den Beweis stellen wir wieder als Übungsaufgabe. Dieses Lemma ist das Werkzeug mit dem sich für die von einem NFA beschriebene Sprache ein regulärer Ausdruck finden lässt. Der Trick dabei ist, einen NFA mit n Zuständen als rekursives Gleichungssystem mit n Gleichungen zu betrachten, welches man sukzessive durch Anwendung des Arden'schen Lemmas lösen kann.

Theorem 3. *Wird L von einem NFA mit n Zuständen erkannt, so gibt es auch einen regulären Ausdruck α, der L beschreibt, sodass $|\alpha| = \mathcal{O}(|\mathcal{A}|)$ gilt.*

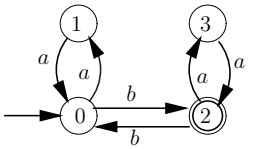
Beweis. Sei $L = L(\mathcal{A})$ für einen NFA $\mathcal{A} = (Q, \Sigma, q_I, \delta, F)$. O.B.d.A. nehmen wir an, dass $Q = \{0, \ldots, n-1\}$ und $q_I = 0$ gilt.

Für jedes $i = 0, \ldots, n-1$ definieren wir nun die Sprache X_i bestehend aus all den Wörtern, für die es einen in i beginnenden akzeptierenden Lauf in $\mathcal{A}$ gibt. Formell also $X_i = L(Q, \Sigma, i, \delta, F)$. Beachte, dass $L = X_0$. Diese Sprachen genügen nun dem folgenden Gleichungssystem:

$$X_i \;=\; \Big(\bigcup_{a\in\Sigma}\bigcup_{j\in\delta(i,a)} \{a\}X_j\Big) \;\cup\; \begin{cases}\{\epsilon\} & \text{, falls } i \in F\\ \emptyset & \text{, sonst}\end{cases}$$

Mithilfe von offensichtlichen Transformationen und sukzessiver Anwendung des Arden'schen Lemmas können nun reguläre Ausdrücke für die X_i und somit auch für L selbst bestimmt werden.

Als Beispiel betrachten wir:



Das Gleichungssystem lautet (wir lassen Mengeklammern bei Einermengen weg):

$$\begin{aligned} X_0 &= aX_1 \cup bX_2 && \text{(I)} \\ X_1 &= aX_0 && \text{(II)} \\ X_2 &= aX_3 \cup bX_0 \cup \{\epsilon\} && \text{(III)} \\ X_3 &= aX_2 && \text{(IV)} \end{aligned}$$

Einsetzen von IV in III liefert

$$X_2 = aaX_2 \cup bX_0 \cup \epsilon \text{ (V)}$$

also

$$X_2 = (aa)^*(bX_0 \cup \epsilon) \text{ (VI)}$$

Einsetzen von VI und II in I liefert

$$\begin{aligned} X_0 &= aaX_0 \cup b(aa)^*(bX_0 \cup \epsilon) \\ X_0 &= (aa \cup b(aa)^*b)X_0 \cup b(aa)^* \\ X_0 &= (aa \cup b(aa)^*b)^*b(aa)^* \end{aligned}$$

wobei sich die zweite Gleichung durch Vereinfachung und die dritte Gleichung wiederum mit dem Arden'schen Lemma ergibt. Diese beinhaltet den gesuchten regulären Ausdruck.

Theorem 4. *Wird eine Sprache L von einem NFA mit n Zuständen erkannt, so wird L auch von einem DFA mit höchstens 2^n vielen Zuständen erkannt.*

Beweis. Sei $L = L(\mathcal{A})$ für einen NFA $\mathcal{A} = (Q, \Sigma, q_0, \delta, F)$. Dann sei $\mathcal{P} := (2^Q, \Sigma, \{q_0\}, \Delta, F')$ wobei für jedes $S \subseteq Q$ und jedes $a \in \Sigma$.

$$\Delta(S, a) \;:=\; \bigcup_{q \in S} \delta(q, a)$$

und $F' := \{S \subseteq Q \mid F \cap S \neq \emptyset\}$.

Man beachte, dass $\mathcal{P}$ in der Tat ein DFA ist: Vom Zustand S erreicht man unter Lesen von a genau einen Zustand, auch wenn die Transitionsfunktion hier als Vereinigungsmenge geschrieben ist. Zustände sind allerdings Mengen von Zuständen aus $\mathcal{A}$, weswegen solch eine Vereinigungsmenge einen Zustand darstellt.

Es bleibt zu zeigen, dass $L(\mathcal{P}) = L(\mathcal{A})$ gilt. Für die $\supseteq$-Richtung existiere ein akzeptierender Lauf $q_0, \ldots, q_n$ von $\mathcal{A}$ auf einem Wort $w = a_0, \ldots, a_{n-1}$. Da $\mathcal{P}$ deterministisch ist, existiert ein eindeutiger Lauf $S_0, \ldots, S_n$ von $\mathcal{P}$ auf w. Es gilt offensichtlich $q_0 \in S_0$, da $S_0 = \{q_0\}$. Darüberhinaus wird die folgende Invariante für alle $i = 0, \ldots, n-1$ bewahrt: Ist $q_i \in S_i$, so ist $q_{i+1} \in S_{i+1}$. Dies ist der Fall, weil $q_{i+1} \in \delta(q_i, a_i)$ für alle diese i gilt. Somit gilt letztendlich auch $q_n \in S_n$, und da $q_n \in F$ auch $F \cap S_n \neq \emptyset$, weswegen $S_0, \ldots, S_n$ ein akzeptierender Lauf von $\mathcal{P}$ auf w ist.

Für die $\subseteq$-Richtung sei $S_0, \ldots, S_n$ ein akzeptierenden Lauf von $\mathcal{P}$ auf einem Wort $w = a_0 \ldots a_{n-1}$. Falls $n = 0$, also $w = \epsilon$, so muss $S_0 \cap F \neq \emptyset$ gelten, was aber $q_0 \in F$ bedeutet, weswegen $\mathcal{A}$ auch w akzeptiert.

Sei also $n > 0$. Beachte, dass jeder $\mathcal{A}$-Zustand q in einem S_i in gewissem Sinne mit einem $q' \in S_{i-1}$ verbunden ist: Für jedes $i = 1, \ldots, n$ und jedes $q \in S_i$ existiert ein $q' \in S_{i-1}$, sodass $q \in \delta(q', a_{i-1})$. Dies ist eine sofortige Konsequenz aus der Definition von Δ. Dies bedeutet aber auch, dass sich für jedes $i = 1, \ldots, n$ und jedes $q \in S_i$ ein Lauf von $\mathcal{A}$ auf dem Wort $a_0 \ldots a_{i-1}$ konstruieren lässt, der in q endet. Da $S_n \cap F \neq \emptyset$, gibt es ein $q \in S_n$ mit $q \in F$. Die Anwendung dieser Beobachtung auf dieses q und $i = n$ liefert dann einen akzeptierenden Lauf von $\mathcal{A}$ auf w.

Da jeder DFA auch ein NFA per Definition ist, folgt also, dass NFAs, DFAs und reguläre Ausdrücke alle genau die regulären Sprachen erkennen bzw. beschreiben.

Korollar 1 (Kleene). *Die folgenden Aussagen sind für eine beliebige Sprache L alle äquivalent.*

- $L \in \mathrm{REG}$
- *Es gibt einen regulären Ausdruck α mit $L = L(\alpha)$.*
- *Es gibt einen NFA $\mathcal{A}$ mit $L = L(\mathcal{A})$.*
- *Es gibt einen DFA $\mathcal{A}$ mit $L = L(\mathcal{A})$.*

Eine wichtige Konsequenz aus der Tatsache, dass DFAs ausreichen, um reguläre Sprachen zu beschreiben, ist der Komplementabschluss der Klasse der regulären Sprachen. Beachte, dass zu einem gegebenen regulären Ausdruck α nicht ohne weiteres ein regulärer Ausdruck $\bar{\alpha}$ gefunden werden kann, sodass $L(\bar{\alpha}) = \overline{L(\alpha)} := \Sigma^* \setminus L(\alpha)$. Dasselbe gilt für NFAs, da das Komplementieren aus der existenziellen Quantifizierung über Läufe eine universelle Quantifizierung machen würde. Diese kann aber i.A. nicht wieder als existentielle Quantifizierung beschrieben und somit mit einem NFA erkannt werden. Da DFAs aber auf einem gegebenen Wort jeweils einen eindeutigen Lauf hat, sind bei diesem Automatenmodell die Aussagen "es gibt einen Lauf auf w" und "für alle Läufe auf w" äquivalent.

Theorem 5. *Für alle $L \subseteq \Sigma^*$ gilt: $L \in \mathrm{REG} \Rightarrow \overline{L} \in \mathrm{REG}$.*

Beweis. Sei $L \in \mathrm{REG}$. Laut Satz 1 existiert dann ein regulärer Ausdruck α mit $L = L(\alpha)$. Laut Satz 2 lässt sich daraus ein NFA $\mathcal{A}'$ konstruieren, sodass ebenfalls $L(\mathcal{A}') = L(\alpha) = L$ gilt. Nach Satz 4 lässt sich dieser per Potenzmengenkonstruktion in einen äquivalenten DFA $\mathcal{A}$ umwandeln. Es gilt also auch $L(\mathcal{A}) = L$.

Sei nun $\mathcal{A} = (Q, \Sigma, q_0, \delta, F)$. Definiere $\overline{\mathcal{A}} := (Q, \Sigma, q_0, \delta, Q \setminus F)$. Es bleibt zu zeigen, dass $L(\overline{\mathcal{A}}) = \overline{L(\mathcal{A})}$ gilt.

Für die $\subseteq$-Richtung nehmen wir an, dass $w \in L(\overline{\mathcal{A}})$ gilt für ein beliebiges $w \in \Sigma^*$. Da $\mathcal{A}$ deterministisch ist, gibt es einen eindeutigen Lauf $q_0, \ldots, q_n$ von $\mathcal{A}$ auf w, sodass $q_n \in Q \setminus F$. Beachte, dass $Q \setminus F$ die Endzustände von $\overline{\mathcal{A}}$ sind. Da $\mathcal{A}$ und $\overline{\mathcal{A}}$ aber dieselben Anfangszustände und Transitionsfunktion haben,

ist $q_0, \ldots, q_n$ auch ein Lauf von $\mathcal{A}$ auf w. Dieser endet aber in einem Nicht-Endzustand bzgl. $\mathcal{A}$. Da ein DFA höchstens einen Lauf auf einem gegebenen Wort haben kann, gibt es keinen akzeptierenden Lauf von $\mathcal{A}$ auf w, und somit gilt $w \notin L(\mathcal{A})$ bzw. $w \in \overline{L(\mathcal{A})}$.

Für die $\supseteq$-Richtung benutzen wir die $\subseteq$-Richtung. Beachte, dass $\overline{\mathcal{A}}$ wiederum ein DFA ist, für den dann auch die $\subseteq$-Richtung, also $L(\overline{\overline{\mathcal{A}}}) \subseteq \overline{L(\overline{\mathcal{A}})}$ gelten muss. Da $F \subseteq Q$ gilt auch $Q \setminus (Q \setminus F) = F$. Das bedeutet einfach, dass zweimal Komplementieren wieder den alten Automaten herstellt, also $\overline{\overline{\mathcal{A}}} = \mathcal{A}$, woraus $L(\mathcal{A}) \subseteq \overline{L(\overline{\mathcal{A}})}$ folgt. Dies ist aber äquivalent dazu, dass $\overline{L(\mathcal{A})} \supseteq \overline{\overline{L(\overline{\mathcal{A}})}} = L(\overline{\mathcal{A}})$ gilt, was noch zu beweisen war.

Somit ist $\overline{\mathcal{A}}$ auch ein NFA und nach den Sätzen 3 und 1 ist $L(\overline{\mathcal{A}}) = \overline{L}$ auch eine reguläre Sprache.

Sei L also eine reguläre Sprache L, die von einem NFA mit n Zuständen erkannt wird. Dann lässt sich zwar prinzipiell ein DFA für diese Sprache und damit auch für $\overline{L}$ konstruieren. Die Anzahl seiner Zustände ist im allgemeinen aber exponentiell in n, d.h. sie kann nur durch 2^n nach oben abgeschätzt werden. Man kann sogar zeigen, dass dies optimal ist, d.h., dass es Sprachen $L_n, n \in \mathbb{N}$ gibt, die von einem NFA mit $n+1$ Zuständen, aber nicht von einem DFA mit weniger als 2^n Zuständen erkannt werden.

Beachte, dass das Verfahren im obigen Beweis so präsentiert ist, dass der resultierende DFA immer 2^n Zustände hat. Dies lässt sich jedoch verbessern, indem man nur den Teil des DFA konstruiert, der vom Anfangszustand $\{q_0\}$ aus erreichbar ist. In den meisten Fällen erhält man so einen DFA mit wesentlich weniger als 2^n Zuständen.

Darüberhinaus kann der entstandene DFA noch durch Zusammenfassung *ununterscheidbarer* Zustände minimiert werden. Hierbei sind stets Endzustände von Nicht-Endzuständen unterscheidbar und außerdem sind — rekursiv — q_1, q_2 unterscheidbar, wenn es ein Symbol a gibt, sodass $\delta(q_1, a)$ und $\delta(q_2, a)$ unterscheidbar sind. Für Details zu dieser Prozedur verweisen wir auf die Literatur, z.B. [**?**].

Zum Abschluss dieses Kapitels betrachten wir noch die algorithmische Handhabbarkeit der regulären Sprachen. Dazu definieren wir zwei Entscheidungsprobleme.

- Das *Leerheitsproblem* für reguläre Sprachen ist das folgende. Gegeben ist eine reguläre Sprache L in der Form eines regulären Ausdrucks, NFAs oder DFAs, entscheide, ob $L = \emptyset$ ist.
- Das *Wortproblem* für reguläre Sprachen ist: Gegeben eine reguläre Sprache L wie zuvor sowie ein Wort $w \in \Sigma^*$, gilt $w \in L$?

Ersteres ist im Grunde das wichtigste Entscheidungsproblem für reguläre Sprachen, da sich viele andere Fragen – so auch z.B. das Wortproblem – auf dieses zurückführen lassen. Bevor wir zeigen, dass diese beiden Probleme entscheidbar sind, machen wir noch zwei Bemerkungen. Erstens kann die Komplexität solcher (oder ähnlicher) Entscheidungsprobleme natürlich von

der Art und Weise, wie L repräsentiert ist, abhängen. Zweitens interessiert man sich oft für komplementäre Probleme. So will man z.B. wissen, ob ein gegebener NFA eine *nicht-leere* Sprache beschreibt. Genau genommen handelt es sich dabei um das *Nicht-Leerheitsproblem*. Im folgenden werden wir diese beiden aber nicht weiter unterscheiden. Der Grund dafür ist, dass wir bei solchen Entscheidungsverfahren meistens an deterministischen Verfahren interessiert sind, und bei solchen macht es prinzipiell keinen Unterschied, ob man das gegebene Problem oder sein Komplement löst.

Theorem 6. *Das Leerheitsproblem für NFAs mit n Zuständen lässt sich in Zeit $\mathcal{O}(n)$ lösen.*

Beweis. Sei $\mathcal{A} = (Q, \Sigma, q_0, \delta, F)$ mit $|Q| = n$. Man kann $\mathcal{A}$ als gerichteten und kantenbeschrifteten Graphen mit Knotenmenge Q und Kantenrelation $q \xrightarrow{a} q'$ gdw. $q' \in \delta(q, a)$, auffassen. Mithilfe einer Breiten- oder Tiefensuche, die alle von q_0 aus erreichbaren Zustände markiert, lässt sich in Zeit $\mathcal{O}(n)$ feststellen, ob es einen Zustand $q \in F$ gibt, welcher von q_0 aus erreichbar ist. Ist dies der Fall, so ist $L(\mathcal{A}) \neq \emptyset$, denn der Pfad von q_0 nach q beschreibt einen akzeptierenden Lauf von $\mathcal{A}$ auf dem Wort, welches durch Konkantenation der einzelnen Kantenbeschriftungen entlang dieses Pfades entsteht. Wird in der Suche kein Endzustand markiert, so muss $L(\mathcal{A}) = \emptyset$ sein, dann kein Lauf von $\mathcal{A}$ auf irgendeinem Wort kann in einem Endzustand enden.

Ein Entscheidungsverfahren für reguläre Sprachen, die durch reguläre Ausdrücke gegeben sind, erhält man dann z.B. durch Vorschalten der Konstruktion aus Satz 2.

Das Wortproblem ist ebenfalls entscheidbar. Man kann im Prinzip ein ähnliches Verfahren verwenden, in dem der NFA $\mathcal{A}$ wieder als gerichteter Graph aufgefasst wird. Allerdings wird dieser so modifiziert, dass nur noch Pfade vom Anfangszustand aus existieren, deren Beschriftung genau das Eingabewort w ergibt. Dies ist jedoch nicht ganz trivial, da solche Pfade Schleifen bilden können, wobei man aufpassen muss, dass der entstehende Graph nicht einen NFA darstellt, welcher aufgrund der Schleifen mehr erkennt als nur das Wort w. Allgemein konstruiert man zuerst einen NFA $\mathcal{A}_w$, dessen Sprache genau $\{w\}$ ist. Dies ist leicht mit $|w| + 1$ vielen Zuständen möglich. Dann benutzt man den Abschluss der regulären Sprachen unter Durchschnitt (siehe Aufgabe 7 unten), um einen NFA zu erhalten, der die Sprache $L(\mathcal{A}) \cap \{w\}$ erkennt, welchen man dann mit obigen Verfahren aus Leerheit testet. Insgesamt ist dies in Zeit $\mathcal{O}(n \cdot |w|)$ möglich, falls $\mathcal{A}$ genau n Zustände hat.

Theorem 7. *Die folgenden Probleme sind für reguläre Sprachen L, L' und Wörter $w \in \Sigma^*$ entscheidbar.*

1. *Das* Wortproblem*: Gegeben w und L, gilt $w \in L$?*
2. *Das* Leerheitsproblem*: Gegeben L, ist $L = \emptyset$?*
3. *Das* Universalitätsproblem*: Gegeben L, ist $L = \Sigma^*$?*
4. *Das* Schnittproblem*: Gegeben L, L', ist $L \cap L' = \emptyset$?*

5. Das Äquivalenzproblem*: Gegeben L, L', ist $L = L'$?*
6. Das Inklusionsproblem*: Gegeben L, L', ist $L \subseteq L'$?*

Beweis als Übung.

3

Spiele

Spiele sind ein wesentliches Hilfsmittel bei der Betrachtung logischer und automatentheoretischer Probleme. Solche lassen sich oft spieltheoretisch charakterisieren. Dabei handelt es sich um nichts anderes als eine Reduktion. Ein Entscheidungs- oder Berechnungsproblem L_1 wird in ein anderes L_2 eingebettet, so dass ein Algorithmus zum Entscheiden oder Lösen von L_2 auch für L_1 verwendet werden kann. Das allgemeinere Problem L_2 ist in vielen Fällen dann das Problem, ein gegebenes Spiel zu lösen. In diesem Kapitel erklären wir zunächst, wie man Spiele formalisieren kann, um sie dann als solch ein generisches Hilfsmittel zum Lösen anderer Probleme zu benutzen, und was es überhaupt heißt, ein Spiel zu lösen.

Aus spieltheoretischer Sicht betrachten wir hier nur sehr einfache Spiele – sogenannte *2-Personen-Nullsummen-Spiele mit perfekter Information*. Der Name besagt, dass sich in dem Spiel zwei Spieler mit entgegengesetzen Zielen gegenüberstehen. Der eine Spieler gewinnt genau dann, wenn der andere verliert. Zusätzlich sind zu jedem Zeitpunkt beide Spieler immer voll informiert über den momentanen Zustand des Spiels, d.h. es gibt keine versteckte Information. Dies schließt z.B. die meisten Kartenspiele aus, bei denen ein Spieler typischerweise nicht weiß, welche Karten sein Gegenüber besitzt. Auch Spiele wie Tic-Tac-Toe fallen aus diesem Rahmen. Dies ist zwar ein 2-Personen-Nullsummen-Spiel mit perfekter Information, aber es ist nicht der Fall, dass der eine Spieler gewinnt genau dann, wenn der andere verliert, da in diesem Spiel Unentschieden möglich ist. Beispiele der von uns betrachteten Spiele sind Chomp oder Nim.

Alle diese Spiele haben die Eigenschaft, dass sie nur von endlicher Dauer sind und dass der Gewinner nur anhand der am Ende erreichten Konfiguration ermittelt wird. Bei Nim z.B. geht es nur darum, wer das letzte Streichholz aufnimmt, die zuvor genommenen oder deren Reihenfolge etc. haben keinen Einfluss auf den Gewinn. Solche Spiele nennt man *Erreichbarkeitsspiele*. Später werden wir auch noch Spiele betrachten, die zwar weiterhin 2-Personen-Nullsummen-Spiele mit perfekter Information, jedoch von unendlicher Dauer sind. Diese können natürlich nicht wirklich gespielt werden, sondern entstehen

als Abstraktionen oder auch Veranschaulichungen bestimmter automatentheoretischer Aufgabenstellungen und Definitionen.

3.1 Spiele als Graphen

Wir definieren zunächst eine Arena als einen paritionierten, gerichteten Graphen. Intuitiv gesehen beginnt ein Spiel in einem Knoten einer Arena, und die Partitionierung bestimmt, welcher Spieler den nächsten Knoten auswählt. Dieser muss ein Nachfolger des aktuellen Knoten in dem Graphen sein.

Definition 3. *Eine* Arena *für ein Spiel zwischen zwei Spielern P_0 und P_1 ist ein gerichteter Graph $\mathcal{G} = (V, V_0, V_1, E)$ mit Knotenmenge V und Kantenmenge E, so dass $V = V_0 \cup V_1$ und $V_0 \cap V_1 = \emptyset$.*

Eine Partie *ist eine (möglicherweise unendliche und) maximale Sequenz von Knoten $v_0, v_1, \ldots$, so dass für alle i gilt: $(v_i, v_{i+1}) \in E$. Maximalität bedeutet, dass sie entweder unendlich lang ist oder in einem v_n endet, so dass es kein $w \in V$ gibt mit $(v_n, w) \in E$. Wir sagen, dass diese Partie in Knoten v_0* beginnt*.*

Ein Spiel *ist ein Paar $(\mathcal{G}, \mathcal{W})$, wobei $\mathcal{W}$ eine Menge von Partien ist. Diese gibt an, welche Partien von Spieler P_0 gewonnen werden. Spieler P_1 gewinnt alle anderen Partien.*

Eine Strategie für Spieler P_i in Knoten v_0 ist eine Funktion σ, die jeder endlichen Sequenz $v_0, \ldots, v_n$ von Knoten, für die

- *$(v_j, v_{j+1}) \in E$ für alle $j = 0, \ldots, n-1$ und*
- *$v_n \in V_i$*

gilt, einen Knoten w zuordnet, so dass $(v_n, w) \in E$.

Eine Partie $v_0, v_1, \ldots$ ist konform *mit einer Strategie σ für Spieler P_i, falls für alle j gilt: wenn $v_j \in V_i$ dann $v_{j+1} = \sigma(v_0, \ldots, v_j)$.*

Eine Strategie σ für Spieler P_i ist eine Gewinnstrategie*, falls P_i jede Partie gewinnt, die konform mit σ ist.*

Das Problem, ein Spiel zu lösen, ist dann, zu einem gegebenen Spiel und einem gegebenen Knoten v in dessen Arena zu entscheiden, ob Spieler P_0 eine Gewinnstrategie in v hat. Gegebenenfalls wird noch verlangt, diese zu berechnen.

Man beachte, dass es üblicherweise auf endlichen Arenen bereits unendlich viele Partien geben kann. Deswegen betrachtet man normalerweise symbolische Repräsentierungen der Menge $\mathcal{W}$ von Partien, die von dem einen Spieler gewonnen werden. Diese können z.B. von der Art sein, dass er jede Partie gewinnt, in der ein gewisser Knoten vorkommt oder unendlich oft auftritt, etc. In solchen Fällen erlauben wir uns, von der strikten Notation $(\mathcal{G}, \mathcal{W})$ abzurücken und solche Spiele auf entsprechende Art und Weise zu notieren. Auch werden wir je nach Bedarf Spielernamen wählen, die aussagekräftiger für die jeweilige Anwendung sind als die hier verwendeten P_0 und P_1.

Beispiel 2.

Die oben angeführte Definition des Problems, ein Spiel zu lösen, kann man auch als *lokales Lösen* bezeichnen, da nur verlangt wird, eine Strategie zu finden, die Gewinnstrategie in einem einzigen Knoten des Spiels ist. Das *globale Problem* würde dementsprechend verlangen, für jeden Knoten des Spiels zu entscheiden, ob Spieler P_0 eine Gewinnstrategie in diesem Knoten hat. Es sollte klar sein, dass diese beiden Probleme bis auf einen linearen Faktor im Zeitverbrauch äquivalent sind. Außerdem ist es häufig so, dass *Uniformität* gilt: Hat ein Spieler Gewinnstrategien für zwei verschiedenen Knoten, so gibt es auch eine Strategie, welche Gewinnstrategie in beiden Knoten ist. Dies muss aber dann im Einzelfall überprüft werden.

Spiele sind besonders interessant, weil sich mit ihrer Hilfe viele Resultate dualisieren lassen. Dies ist der Fall, wenn Spiele determiniert sind.

Definition 4. *Ein Spiel $(\mathcal{G}, \mathcal{W})$ zwischen Spielern P_0 und P_1 mit $\mathcal{G} = (V, V_0, V_1, E)$ ist* determiniert*, wenn für alle Knoten $v \in V$ gilt und alle $i \in \{0, 1\}$: Spieler P_i hat eine Gewinnstrategie in v gdw. Spieler P_{1-i} keine Gewinnstrategie in Knoten v hat.*

Der Teil "$\Longrightarrow$" dieser Äquivalenz gilt immer (Übung). Wenn ein Spieler eine Gewinnstrategie in einem Knoten hat, so kann der andere keine in diesem Knoten haben. Die Schwierigkeit Determiniertheit eines Spiels zu zeigen besteht also darin, aus der Nicht-Existenz einer Gewinnstrategie für den einen Spieler die Existenz einer solchen für den anderen zu folgern.

Wir betrachten noch spezielle Strategien, die algorithmisch einfacher zu handhaben sind als allgemeine.

Definition 5. *Sei $\mathcal{G} = (V, V_0, V_1, E)$ eine Arena. Eine Strategie für Spieler P_i in Knoten $v_0 \in V$ heißt* positional*, wenn für alle Knoten $v_0, v_1, v'_1, \ldots, v_{n-1}, v'_{n-1}, v_n$ gilt:*

$$\sigma(v_0, v_1, \ldots, v_{n-1}, v_n) = \sigma(v_0, v'_1, \ldots, v'_{n-1}, v_n)$$

Ein Spiel $(\mathcal{G}, \mathcal{W})$ heißt positional determiniert*, wenn es determiniert ist und darüberhinaus für beide Spieler P_i und alle Knoten $v \in V$ gilt: Spieler P_i hat eine Gewinnstrategie in v gdw. Spieler P_i eine positionale Gewinnstrategie in v hat.*

In anderen Worten gesprochen hängt bei einer positionalen Strategie die Wahl eines Nachfolgerknotens in der Arena nicht von dem Weg ab, auf dem der aktuelle Knoten erreicht wurde, sondern nur von diesem selbst. Aus diesem Grund werden solche Strategien manchmal auch *gedächtnislos* genannt.

3.2 Erreichbarkeitsspiele

Zum Abschluss betrachten wir noch eine spezielle Klasse von Spielen für die sich Determiniertheit und die Existenz von gedächtnislosen Gewinnstrategien relativ einfach zeigen lassen.

Definition 6. *Sei $(\mathcal{G}, \mathcal{W})$ ein Spiel zwischen Spielern P_0 und P_1 mit $\mathcal{G} = (V, V_0, V_1, E)$. Dies ist ein* Erreichbarkeitsspiel*, wenn*

- *jede Partie in der Arena $\mathcal{G}$ endlich ist und*
- *für alle Partien $\pi = v_0, \ldots, v_n$ und $\pi' = w_0, \ldots, w_m$ gilt: falls $v_n = w_m$, dann ist $\pi \in \mathcal{W}$ gdw. $\pi' \in \mathcal{W}$.*

In einem Erreichbarkeitsspiel ist der Gewinner einer Partie also eindeutig durch den letzten Knoten in dieser Partie bestimmt.

Beachte, dass ein Erreichbarkeitsspiel nicht unbedingt selbst endlich sein muss. In der Erweiterung von Nim, wo ein Spieler zuerst eine beliebige Anzahl von Streichhölzern auf den Tisch legt, worauf dann Nim mit dieser Anzahl in üblicher Weise gespielt wird, ist jede Partie nur endlich lang, aber es gibt unendlich viele verschiedene Spielpositionen, nämlich ungefähr eine für jede Anzahl an Streichhölzern und jeden Spieler, der als nächstes ziehen muss.

Erreichbarkeitsspiele haben die Eigenschaft, dass sich die Knotenmenge linear wohl-ordnen lässt, so dass bei jedem Zug ein Abstieg in der Ordnung erfolgt. Ist das Spiel darüber hinaus noch endlich, so kann man die Knotenmenge dann mit einem Anfangsstück der natürlichen Zahlen indizieren, so dass der Abstieg in den Indizes sichtbar wird.

Definition 7. *Eine* wohlfundierte Ordnung *auf einer Menge M ist eine binäre Relation $\preceq \subseteq M \times M$, für die folgendes gilt.*

- *(Transitivität) Für alle $x, y, z \in M$ mit $x \preceq y$ und $y \preceq z$ gilt $x \preceq z$.*
- *(Reflexivität) Für alle $x \in M$ gilt $x \preceq x$.*
- *(Anti-Symmetrie) Für alle $x, y \in M$ gilt: Falls $x \preceq y$ und $y \preceq x$, dann ist $x = y$.*
- *(Wohlfundiertheit) Es gibt keine unendliche Kette $x_0, x_1, x_2, \ldots$, so dass für alle $i \in \mathbb{N}$ gilt: $x_{i+1} \preceq x_i$ und $x_{i+1} \neq x_i$.*

Lemma 4. *Die transitive Hülle der Kantenrelation eines Erreichbarkeitsspiels ist eine wohlfundierte Ordnung auf dessen Knotenmenge.*

Beweis. Sei $(\mathcal{G}, \mathcal{W})$ ein Erreichbarkeitsspiel. Für die Aussage des Lemmas ist $\mathcal{W}$ uninteressant. Sei $\mathcal{G} = (V, V_0, V_1, E)$ und E^+ die transitive Hülle der Kantenrelation E. D.h. $(v, w) \in E^+$ gdw. es ein $n \in \mathbb{N}$ und $v_0, \ldots, v_n$ gibt, so dass $v_0 = v$, $v_n = w$ und $(v_i, v_{i+1}) \in E$ für alle $i = 0, \ldots, n-1$.

Offensichtlich ist E^+ transitiv. Es bleibt lediglich zu zeigen, dass $\mathcal{G}$ nicht Arena eines Erreichbarkeitsspiels sein kann, wenn E^+ nicht reflexiv, nicht anti-symmetrisch oder nicht wohl-fundiert wäre. In allen drei Fällen ließen sich sofort unendliche Partien konstruieren.

Wir zeigen nun in einem, dass Erreichbarkeitsspiele determiniert sind und die Spieler jeweils positionale Gewinnstrategien besitzen.

Theorem 8. *Erreichbarkeitsspiele sind positional determiniert.*

Beweis. Sei $(\mathcal{G}, \mathcal{W})$ ein Erreichbarkeitsspiel mit $\mathcal{G} = (V, V_0, V_1, E)$. Laut Lemma 4 ist die transitive Hülle E^+ der Kantenrelation E eine wohlfundierte Ordnung $\preceq$ auf V. Wir zeigen nun durch wohlfundierte Induktion, dass für alle $v \in V$ gilt: Spieler P_i hat eine positionale Gewinnstrategie für v, falls Spieler P_{i-1} keine Gewinnstrategie für v hat. Es sollte klar sein, dass daraus Determiniertheit folgt, da die Umkehrung immer gilt.

Sei $v \in V$. Betrachte die Menge $U := \{w \in V \mid (v, w) \in E\}$. Offensichtlich gilt $U \subseteq \{w \in V \mid (v, w) \in E^+\}$, und somit können wir die Aussage für alle $w \in U$ als bereits bewiesen annehmen. Das bedeutet, dass es für jedes $w \in U$ ein $j_w \in \{0, 1\}$ und eine positionale Gewinnstrategie σ_w für Spieler P_{j_w} in Knoten w gibt. Jetzt unterscheiden wir zwei Fälle.

Fall 1, $v \in V_i$ und es gibt $w \in U$ mit $p_w = i$. D.h. es gibt einen Nachfolgerknoten w, so dass der Besitzer des aktuellen Knotens v aus das Spiel beginnend in diesem Nachfolger gewinnt. Dann gewinnt er auch von v aus mit der Strategie $\sigma_v := \sigma_w[v \mapsto w]$. Diese erweitert die positionale Strategie σ_w um den Zug $v \mapsto w$, was weiterhin positional ist. Es ist außerdem eine Gewinnstrategie für Spieler P_i, da jede Partie, die mit σ_v konform ist, eine Partie, die wiederum mit σ_w konform ist, als echtes Suffix hat, und somit P_i auch jede solche Partie gewinnen muss, denn der Gewinner ist eindeutig durch den letzten Knoten in einer Partie bestimmt.

Fall 2, $v \in V_i$ und für alle $w \in U$ gilt $p_w = 1 - i$. D.h. Spieler $1 - i$ gewinnt von allen Nachfolgern von v aus. Dann gewinnt Spieler $1 - i$ auch von v aus, denn Spieler i muss von dort zu einem dieser Nachfolger ziehen. Eine positionale Gewinnstrategie für Spieler $1 - i$ ergibt sich durch Vereinigung der positionalen Strategien σ_w für alle $w \in U$. Diese können jedoch nicht-disjunkte Domains haben. Deswegen sei $U = \{w_1, \ldots, w_k\}$ eine Aufzählung aller Nachfolger von v. Definiere für beliebiges $u \in V$:

$$\sigma_v(u) \;:=\; \begin{cases} \sigma_{w_j}(u) & \text{, falls } u \in dom(\sigma_{w_j}) \text{ und } u \notin dom(\sigma_{w_h}) \text{ für alle } h < j \\ \bot & \text{, sonst} \end{cases}$$

Das bedeutet, dass σ_v so ziehen lässt wie das jeweilige σ_w für das kleinste w nach der festgelegten Aufzählung. Somit hat jede Partie, die konform zu σ_v ist, ein Suffix, welches konform zu einem σ_w ist. Da diese alle von Spieler $1 - i$ gewonnen werden, gewinnt Spieler $1 - i$ auch alle Partien, die konform zu σ_v sind. Außerdem ist σ_v offensichtlich positional. Damit ist gezeigt, dass Spieler $1 - i$ eine positionale Gewinnstrategie hat, falls Spieler i keine hat. □

Übungsaufgaben

Übung 1. Beweise Satz 1. *Hinweis:* Beide Richtungen der Äquivalenz können leicht durch Induktion (über den Aufbau regulärer Sprachen bzw. den Aufbau regulärer Ausdrücke) gezeigt werden.

Übung 2. Warum ist es nicht nötig, in der Definition der regulären Sprachen auch $\{\epsilon\}$ miteinzuschließen?

Übung 3. Beweise Lemma 1: Zu zwei gegebenen NFAs kann man jeweils einen NFA konstruieren, der genau die Vereinigung bzw. die Konkatenation der von den beiden gegebenen NFAs erkannten Sprachen erkennt.

Übung 4. Beweise Satz 2: Reguläre Ausdrücke lassen sich induktiv in NFAs übersetzen. Der resultierenden NFA ist von derselben Größenordung wie der reguläre Ausdruck.

Übung 5. Beweise das Arden'sche Lemma (Lemma 3). *Hinweis:* Benutze dabei, dass $L^* = \bigcup_{i \in \mathbb{N}} L^i$ gilt.

Übung 6. Sei $\alpha_n := (a \cup b)^* a (a \cup b)^{n-1}$ für beliebiges $n \geq 2$, wobei $\beta^0 := \epsilon$ und $\beta^{i+1} := \beta\beta^i$. Zeige, dass

a) $L(\alpha_n)$ von einem NFA mit $n+1$ Zuständen erkannt wird.
b) $L(\alpha_n)$ nicht von einem DFA mit weniger als 2^n Zuständen erkannt wird.

Übung 7. Zeige, dass die Klasse REG neben den zentralen Operationen Vereinigung, Konkatenation und Kleene-Stern noch unten den folgenden Operationen abgeschlossen ist.

1. *Durchschnitt*: $(L_1, L_2 \in \text{REG} \Rightarrow L_1 \cap L_2 \in \text{REG})$,
2. *Differenz*: $(L_1, L_2 \in \text{REG} \Rightarrow L_1 \setminus L_2 \in \text{REG})$,
3. *Spiegelung*: $(L \in \text{REG} \Rightarrow \{a_n \dots a_1 \mid a_1 \dots a_n \in L\} \in \text{REG}$,
4. *Homomorphismen*: Seien Σ, Δ Alphabete und $h : \Sigma \to \Delta^*$ eine Abbildung. Diese kann homomorph zu einer Abbildung $\hat{h} : \Sigma^* \to \Delta^*$ erweitert werden: $\hat{h}(\epsilon) = \epsilon$ und $\hat{h}(av) = h(a)\hat{h}(v)$ für jedes $a \in \Sigma$. Zeige nun, dass gilt: $L \in \text{REG} \Rightarrow \{\hat{h}(w) \mid w \in L\} \in \text{REG}$.

5. *Shuffle-Produkt*: Wir definieren eine Abbildung $\bowtie$, die zwei Wörter auf die Menge aller ihrer Verzahnungen abbildet. Für alle $a \in \Sigma$, alle $w, v \in \Sigma^*$ sei

$$\begin{aligned} \epsilon \bowtie v &:= \{v\} \\ aw \bowtie v &:= \{uau' \mid uu' \in w \bowtie v\} \end{aligned}$$

Beachte, dass in der letzten Klausel u und u' auch jeweils das leere Wort sein können.
Diese Verzahnung lässt sich dann in natürlicher Weise auf Sprachen fortsetzen:

$$L_1 \bowtie L_2 := \bigcup_{w \in L_1} \bigcup_{v \in L_2} w \bowtie v$$

Zeige nun, dass gilt: $L_1, L_2 \in \mathrm{REG} \Rightarrow L_1 \bowtie L_2 \in \mathrm{REG}$.

Übung 8. Sei $(\mathcal{G}, \mathcal{W})$ ein Spiel zwischen Spielern P_0 und P_1, so dass $\mathcal{G} = (V, V_0, V_1, E)$. Sei $v \in V$. Angenommen, Spieler P_i hat eine Gewinnstrategie in v. Zeigen Sie, dass Spieler P_{1-i} keine Gewinnstrategie in v haben kann. *Hinweis:* Nehmen Sie an, dass dies der Fall wäre, und führen Sie dies zu einem Widerspruch, indem Sie eine bestimmte Partie konstruieren.

Teil II

Endliche Wörter

4

Die schwache monadische Logik zweiter Stufe

In diesem Kapitel lernen wir die erste Anwendung von Automaten auf Entscheidungsprobleme in der Logik kennen. Wir definieren die *schwache monadische Logik zweiter Stufe* (engl.: *weak monadic second order logic*, wMSO) und zeigen, dass die Allgemeingültigkeit und Erfüllbarkeit von Formeln in dieser Logik mithilfe von Automaten entschieden werden kann.

4.1 Syntax und Semantik

Definition 8. *Seien zwei abzählbar unendliche Mengen von* erststufigen Variablen $V_1 = \{x, y, \ldots\}$ *und* zweitstufigen Variablen $V_2 = \{X, Y, \ldots\}$ *gegeben. Formeln der* schwachen monadischen Logik zweiter Stufe *über* V_1, V_2 *sind gegeben durch folgende Grammatik.*

$$\varphi \quad ::= \quad x < y \mid X(x) \mid \varphi_1 \vee \varphi_2 \mid \neg\varphi \mid \exists x.\varphi \mid \exists X.\varphi$$

wobei $x, y \in V_1$ *und* $X \in V_2$.

Das bedeutet also, dass für je zwei erststufige Variablen x, y der Ausdruck $x < y$ eine Formel ist. Außerdem ist $X(x)$ eine Formel, wenn X zweitstufige und x erststufige Variable ist. Sind φ_1, φ_2 Formeln, so auch $\varphi_1 \vee \varphi_2$. Ist φ Formel, so auch $\neg\varphi$ und $\exists x.\varphi$ und $\exists X.\varphi$, wobei wiederum x erst- und X zweitstufig ist.

In der Regel verwendet man Kleinbuchstaben für erststufige Variablen und Großbuchstaben für zweitstufige Variablen. Man kann aber auch andere Objekte, z.B.: Bezeichner, für die Variablen verwenden; es muss nur immer klar sein, was eine Variable ist und welche Stufe sie hat.

Intuitiv rangieren die erststufigen Variablen über natürliche Zahlen und die zweitstufigen über endliche Mengen von natürlichen Zahlen. Die anderen Symbole haben die übliche Bedeutung, wir definieren sie später noch formal. So bedeutet etwa die Formel $\neg\exists y.y < x$, dass x gleich Null ist.

Wir benutzen die üblichen Abkürzungen $\varphi \wedge \psi := \neg(\neg\varphi \vee \neg\psi)$, $\varphi \to \psi := \neg\varphi \vee \psi$, $\forall x.\varphi := \neg\exists x.\neg\varphi$, $\forall X.\varphi := \neg\exists X.\neg\varphi$, $x \leq y := \neg(y < x)$, $x = y := x \leq y \wedge \neg(x < y)$, $x{=}0 := \neg\exists y.y < x$, etc.

Die Formel

$$\exists X.(\forall x.x{=}0 \to X(x)) \wedge (\forall x.X(x) \to \exists y.x < y \wedge X(y))$$

besagt dann, dass eine endliche Menge X existiert, die die Null enthält, also insbesondere nicht leer ist, und die zu jedem Element noch ein größeres enthält. So eine Menge gibt es nicht, also ist diese Formel falsch genauso, wie etwa die Formel $\exists x.x < x$.

Ein Vorkommen einer Variablen x (oder X) in einer Formel heißt *gebunden*, wenn im Syntaxbaum über ihm der Operator $\exists x$, bzw. $\exists X$ vorkommt. Ansonsten heißt dieses Vorkommen *frei*. Wir schreiben auch $\varphi(X_1, \ldots, X_n, x_1, \ldots, x_m)$ um anzudeuten, dass die freien Variablen in φ zu der Menge $\{X_1, \ldots, X_n, x_1, \ldots, x_m\}$ gehören. Eine Formel ohne freie Variablen wird auch *Satz* genannt.

In der Formel $\neg\exists y.y < x$ ist also x frei und y gebunden. In der Formel $\forall x.(\neg\exists y.y < x) \to X(x)$ ist X frei und x, y sind beide gebunden. Die Bedeutung einer Formel hängt von den Werten ihrer freien Variablen ab; je nach deren Belegung kann sie wahr oder falsch sein. Ein *Satz* ist also per se entweder wahr oder falsch. Z.B.: ist die o.a. Formel $\exists x.x < x$ ein falscher Satz, während $\exists X.\forall x.X(x) \to \exists y.x < y \wedge X(y)$ ein wahrer Satz ist; man kann nämlich für X die leere Menge nehmen.

Definition 9. *Wir definieren die Semantik einer wMSO-Formel als eine Relation $\models$ zwischen Belegungen und Formeln; erstere sind definiert als Abbildungen I, die erststufigen Variablen natürliche Zahlen und zweitstufigen Variablen endliche Mengen von natürlichen Zahlen zuordnen.*

$$\begin{aligned}
I \models x = y \quad & \textit{gdw.} \quad I(x) = I(y) \\
I \models x < y \quad & \textit{gdw.} \quad I(x) < I(y) \\
I \models X(x) \quad & \textit{gdw.} \quad I(x) \in I(X) \\
I \models \varphi \vee \psi \quad & \textit{gdw.} \quad I \models \varphi \textit{ oder } I \models \psi \\
I \models \neg\varphi \quad & \textit{gdw.} \quad I \not\models \varphi \\
I \models \exists x.\varphi \quad & \textit{gdw.} \quad \textit{es gibt ein } i \in \mathbb{N}, \textit{sodass } I[x \mapsto i] \models \varphi \\
I \models \exists X.\varphi \quad & \textit{gdw.} \quad \textit{es gibt eine endliche Teilmenge } M \subseteq \mathbb{N}, \textit{sodass } I[X \mapsto M] \models \varphi
\end{aligned}$$

wobei $I[x \mapsto i]$ diejenige Abbildung ist, die x auf i abbildet und sich wie I auf allen anderen Argumenten verhält.

Zwei Formeln sind äquivalent, *$\varphi \equiv \psi$, falls für alle I gilt: $I \models \varphi$ gdw. $I \models \psi$.*

Ist φ ein Satz, so gilt $I \models \varphi \iff I' \models \varphi$ für alle Belegungen I, I'; die Bedeutung eines Satzes hängt also gar nicht von der Belegung ab. Man schreibt in diesem Falle daher $\models \varphi$, falls $I \models \varphi$ für ein und damit für alle I gilt.

Gilt $I \models \varphi$ für mindestens ein I, so ist φ *erfüllbar*. Eine Interpretation I mit $I \models \phi$ heißt *Modell* von φ. Die kleinste Zahl n, sodass $I(x) < n$ und $I(X) \subseteq \{0, \ldots, n-1\}$ heißt *Größe des Modells* I. Eine Formel, die kein Modell hat, heißt *unerfüllbar*. Eine Formel φ, derart dass $I \models \varphi$ für alle I gilt, heißt *allgemeingültig*. Eine Formel φ ist allgemeingültig, genau dann, wenn $\neg\varphi$ unerfüllbar ist.

4.1.1 Endliche Belegungen

Allgemeiner gilt, dass $I \models \varphi \iff I' \models \varphi$, falls I und I' auf den freien Variablen von φ übereinstimmen; die Bedeutung einer Formel φ hängt also nur von der Einschränkung der jeweiligen Belegung auf die freien Variablen ab. Daher definiert man die Notation $I \models \varphi$ auch für endliche, partielle Belegungen I solange diese auf den freien Variablen von φ definiert sind.

Dadurch kann man insbesondere auch sinnvoll die Frage stellen, ob es zu gegebenem I und φ entscheidbar ist, ob $I \models \varphi$ gilt. Wir werden diese Frage weiter unten bejahen.

4.2 Verbindung zur Theorie formaler Sprachen

Sei Σ ein Alphabet; für jedes Symbol $a \in \Sigma$ führen wir eine zweitstufige Variable P_a ein.

Ein Wort w über Σ definiert dann eine Belegung I_w dieser Variablen durch $I_w(P_a) = \{i < |w| \mid w_i = a\}$. Es bezeichnet also $I_w(P_a)$ all diejenigen Positionen, an denen in w ein a steht. Diese Positionen werden immer ab 0 gezählt, also ist z.B.: $(abab)_1 = b$ und $I_w(P_b) = \{1, 3\}$. Beachte, dass die Menge $I_w(P_a)$ stets endlich ist.

Um I_w auch formal zu einer Belegung zu machen, legt man I_w für die anderen Variablen willkürlich fest oder beruft sich auf die in 4.1.1 getroffene Konvention über endliche Belegungen.

Umfassen die freien Variablen einer Formel φ höchstens die Variablen P_a für $a \in \Sigma$, also keine erststufigen Variablen und auch keine anderen zweitstufigen Variablen als die P_a, so macht es Sinn, ihren Wahrheitswert unter einer Belegung I_w zu betrachten, denn dieser hängt dann nicht von den willkürlichen Setzungen ab. Mit der weiter oben diskutierten Erweiterung der Semantik auf partielle Belegungen, kann man dann auch auf die willkürlichen Setzungen verzichten.

Betrachten wir als Beispiel $\Sigma = \{a, b\}$ und die Formel $\varphi := \forall x.\neg(P_a(x) \wedge P_b(x))$. Für jedes Wort $w \in \Sigma^*$ gilt hier $I_w \models \varphi$, denn an keiner Position kann sowohl a, als auch b, stehen. Die Formel $\psi := \forall x.\forall y.(P_a(x) \wedge P_b(y)) \rightarrow x < y$ gilt nicht für alle Wörter: es ist $I_{aaabbbb} \models \psi$, aber $I_{aabab} \not\models \psi$. Dies deshalb, weil hier $P_b(2)$ (formal $2 \in I_{aabab}(P_b)$) und $P_a(3)$ gilt, aber natürlich nicht $3 < 2$.

Auf diese Weise definieren wMSO-Formeln Mengen von Wörtern, also Sprachen:

Definition 10. *Sei Σ ein Alphabet und φ eine wMSO-Formel deren freie Variablen ausschließlich zweitstufig und in $\{P_a \mid a \in \Sigma\}$ enthalten sind. Die Sprache $L(\varphi) \subseteq \Sigma^*$ ist dann definiert durch*

$$L(\varphi) := \{w \in \Sigma^* \mid I_w \models \varphi\}$$

wobei $I_w(P_w) = \{i < |w| \mid w_i = a\}$.

Definition 11. *Eine Sprache $L \subseteq \Sigma^*$ heißt* wMSO-definierbar*, falls es eine wMSO-Formel φ gibt mit $L = L(\varphi)$.*

Die Formel $\psi := \forall x.\forall y.(P_a(x) \wedge P_b(y)) \rightarrow x < y$ definiert also die Sprache a^*b^*. Wir geben jetzt weitere Beispiele:

Beispiel 3. 1. $L(\Sigma^* ab \Sigma^*)$ ist wMSO-definierbar durch die Formel

$$\exists x.\exists y.P_a(x) \wedge P_b(y) \wedge x < y \wedge \neg\exists z.x < z \wedge z < y$$

2. $L = \{w \subseteq \{a,b\}^* \mid |w|$ ist ungerade $\}$ ist wMSO-definierbar durch:

$$\forall max.(\neg(P_a(max) \vee P_b(max))) \wedge (\forall y.\neg(P_a(y) \vee P_b(y)) \rightarrow max \leq y) \rightarrow \\ \exists X.X(0) \wedge \big(\forall x.\forall y.succ(x,y) \wedge y \leq max \rightarrow (X(x) \leftrightarrow \neg X(y))\big) \wedge X(max)$$

   wobei $\psi(0) := \exists z.z{=}0 \wedge \psi(z)$ und $succ(x,y) := x < y \wedge \neg\exists z.x < z \wedge z < y$. Die Variable *max* bezeichnet hier immer die Wortlänge; sie ist ja festgelegt auf die kleinste Position an der kein Buchstabe mehr steht. Die Menge X umfasst alle geraden Zahlen kleiner oder gleich der Wortlänge, da sie 0 enthält und "immer abwechselt".
3. $L = \{w \in \{a,b,c\}^* \mid$ in w folgen auf jedes a nur a's bis irgendwann ein b auftritt $\}$:

$$\forall x.P_a(x) \rightarrow \exists y.P_b(y) \wedge x < y \wedge \forall z.x < z \wedge z < y \rightarrow P_a(z)$$

Die hier (in Bsp. 2) definierte Formel $succ(x,y)$ besagt, dass y der Nachfolger von x ist.

Man kann wMSO auch mit $succ(x,y)$ anstelle von $x < y$ als primitiver zweistelliger Formel einführen: Die Formel $x < y$ lässt sich dann nämlich wie folgt definieren:

$$x \leq y := \forall X.(\forall u.X(u) \rightarrow \forall v.succ(u,v) \rightarrow succ(y,v) \vee X(v)) \rightarrow X(x) \rightarrow X(y)$$
$$x{=}y \;\; := x \leq y \wedge y \leq x$$
$$x < y := x \leq y \wedge \neg(x = y)$$

Mit anderen Worten: $x \leq y$, genau dann, wenn jede endliche Menge, die bis einschließlich y unter Nachfolger abgeschlossen ist, mit x auch y enthält.

4.2.1 Von Automaten zu Formeln

Theorem 9. *Zu jeder regulären Sprache L über einem Alphabet Σ lässt sich eine wMSO-Formel φ_L angeben, sodass $L(\varphi_L) = L$.*

Beweis. Es sei ein NFA $\mathcal{A} = (Q, \Sigma, q_I, \delta, F)$ für L vorgelegt. Wir nehmen an, dass $Q = \{0, 1, \ldots, n\}$ mit $q_I = 0$. Wir führen nun zweitstufige Variablen $X_0, \ldots, X_n$ ein, die diesen Zuständen entsprechen. Die Idee ist, φ_L in der Form $\exists X_0 \ldots \exists X_n \ldots$ zu konstruieren, wobei die existentiellen Zeugen für die X_i dann einen erfolgreichen Lauf des Automaten repräsentieren sollen. Insbesondere soll $X_q(i)$ bedeuten, dass der Automat nach Abarbeiten des i-ten Symbols im Zustand q ist.

Hierzu definieren wir folgende Hilfsformeln:

$$
\begin{aligned}
\mathit{uni}(\mathit{max}) &:= \forall x.x < \mathit{max} \rightarrow (\textstyle\bigvee_q X_q(x)) \wedge (\bigwedge_{q \neq q'} \neg(X_q(x) \wedge X_{q'}(x))) \\
\mathit{init}(\mathit{max}) &:= \mathit{max} = 0 \vee \forall x.x{=}0 \rightarrow \textstyle\bigvee_{(q',a):q' \in \delta(0,a)} P_a(x) \wedge X_{q'}(x)) \\
\mathit{lauf}(\mathit{max}) &:= \forall y.y < \mathit{max} \rightarrow \forall x.\mathit{succ}(x,y) \rightarrow \textstyle\bigvee_{(q,q',a):q' \in \delta(q,a)} P_a(x) \wedge X_q(x) \wedge X_{q'}(y) \\
\mathit{akz}(\mathit{max}) &:= [\mathit{max}{=}0 \vee] \exists x.\mathit{succ}(x, \mathit{max}) \wedge \textstyle\bigvee_{q \in F} X_q(x)
\end{aligned}
$$

Der eckig eingeklammerte Teil in der letzten Formel wird nur dann hinzugenommen, wenn $q_I \in F$, ansonsten entfällt er.

Ein "großes oder", wie in $\bigvee_q \ldots \varphi_q$ bezeichnet eine Disjunktion ($\vee$) mit $n = |Q|$ Summanden. Man beachte, dass n fest ist, sodass man solch eine Disjunktion mit der vorhandenen Syntax ausdrücken kann. Eine alternative Notation wäre $\varphi_0 \vee \cdots \vee \varphi_n$. Analog bezeichnet $\bigwedge_{q \neq q'} \ldots$ eine Konjunktion mit $n^2 - n$ Faktoren, die den Zustandspaaren (q, q') mit $q \neq q'$ entsprechen.

Jetzt setzen wir

$$
\begin{aligned}
\varphi_L := \exists X_0 \ldots \exists X_n.\forall \mathit{max}.(\neg(P_a(\mathit{max}) \vee P_b(\mathit{max}))) \wedge \\
(\forall y.\neg(P_a(y) \vee P_b(y) \rightarrow \mathit{max} \leq y) \rightarrow \\
\mathit{uni}(\mathit{max}) \wedge \mathit{init}(\mathit{max}) \wedge \mathit{lauf}(\mathit{max}) \wedge \mathit{akz}(\mathit{max}))
\end{aligned}
$$

Wie im Beispiel weiter oben bezeichnet *max* gerade die Wortlänge.

Die Klausel *uni*(*max*) stellt sicher, dass an jeder Position $x < |w|$ genau eine der $X(x)$ gesetzt ist; *init*(*max*) stellt sicher, dass der Zustand an Position 0 durch Lesen des ersten Symbols vom Startzustand aus entsteht, es sei denn, $\mathit{max} = 0$. Die Klausel *lauf*(*max*) besagt, dass an allen weiteren Positionen der Folgezustand aus dem vorherigen mithilfe von δ entsteht und *akz*(*max*) schließlich besagt, dass der letzte Zustand ein Endzustand ist es sei denn $\mathit{max} = 0$ und $q_I \in F$.

4.2.2 Von Formeln zu Automaten

Nunmehr wollen wir eine Art Umkehrung des eben bewiesenen Satzes formulieren, dahingehend, dass die Semantik beliebiger wMSO-Formeln durch NFA beschrieben werden kann. Hierzu repräsentieren wir Belegungen I einer

festen endlichen Teilmenge $\mathcal{X}$ der Variablen als Wörter w_I über einem geeigneten Alphabet $\Sigma_\mathcal{X}$ und ordnen dann jeder Formel φ, die nur Variablen aus $\mathcal{X}$ enthält (frei oder gebunden) einen NFA A_φ zu (und zwar durch ein konstruktives Verfahren), derart dass $I \models \varphi \iff w_I \in L(A_\varphi)$. Man kann dann insbesondere feststellen, ob ein Satz φ wahr ist, indem man prüft, ob $L(A_\varphi) \neq \emptyset$ ist.

Sei also jetzt eine endliche Variablenmenge $\mathcal{X}$ fixiert, die sowohl erst- als auch zweitstufige Variablen beinhaltet. Wir wählen als Alphabet $\Sigma_\mathcal{X}$ Bewertungen dieser Variablen mit Wahrheitswerten, also

$$\Sigma_\mathcal{X} = 2^\mathcal{X}$$

Ein Buchstabe in Σ ist also eine Funktion von $\mathcal{X}$ nach $\{0, 1\}$.

Einer Belegung I der Variablen in $\mathcal{X}$ ordnen wir jetzt ein Wort $w_I \in \Sigma_\mathcal{X}^*$ zu. Die Länge von w_I ist gerade so groß, dass alle Zahlen, die in I eine Rolle spielen, Positionen in w_I sind, d.h. $|w_I|$ wird so klein wie möglich gewählt unter der Bedingung, dass für alle $x \in \mathcal{X}$ gilt $I(x) < |w_I|$ und für alle $X \in \mathcal{X}$und $i \in I(X)$ gilt $i < |w_I|$.

Für $i < |w_I|$ definieren wir dann den i-ten Buchstaben $(w_I)_i$ durch

$$\begin{array}{l}(w_I)_i(x) = 0, \text{ falls } i \neq I(x)\\ (w_I)_i(x) = 1, \text{ falls } i = I(x)\\ (w_I)_i(X) = 0, \text{ falls } i \notin I(X)\\ (w_I)_i(X) = 1, \text{ falls } i \in I(X)\end{array}$$

Man kann sich das Wort w_I als ein mehrspuriges $0-1$-Wort vorstellen, mit einer Spur für jede Variable. Ist zum Beispiel $\mathcal{X} = \{X, Y, x\}$ und $I(X) = \{0, 4\}$ und $I(Y) = \{1, 3\}$ und $I(x) = 2$, so hat w_I die Länge 5 und kann wie folgt veranschaulicht werden:

$$\begin{array}{c|l} X & 1\ 0\ 0\ 0\ 1 \\ Y & 0\ 1\ 0\ 1\ 0 \\ x & 0\ 0\ 1\ 0\ 0 \end{array}$$

Der Buchstabe $(w_I)_3$ ist also die Funktion, die X auf 0 und Y auf 1 und x auf 0 abbildet, oder zusammengefasst die Spalte $\begin{pmatrix}0\\1\\0\end{pmatrix}$.

Die Spuren, die erststufigen Variablen entsprechen, enthalten immer genau eine 1 und sonst nur 0en, während die Spuren, die den zweitstufigen Variablen entsprechen, beliebige Bitmuster aufweisen.

Durch Induktion über den Formelaufbau definieren wir nun den gesuchten Automaten A_φ. Der Automat erwartet jeweils, dass das Eingabewort tatsächlich von der Form w_I ist.

Der Automat für $X(x)$ muss prüfen, ob an der Stelle, an der in der x-Spur eine 1 steht, auch in der X-Spur eine 1 steht.

Der Automat für $x < y$ prüft, ob das Vorkommen der Eins auf der x-Spur vor dem auf der y-Spur erscheint.

Den Automaten für $\varphi \vee \psi$ erhält man aus den Automaten für φ und ψ wie in der Automatenkonstruktion für die Vereinigungsmenge; intuitiv rät man nichtdeterministisch, ob man das angebotene Wort entweder gemäß A_φ oder gemäß A_ψ verarbeiten möchte.

Den Automaten für $\neg\varphi$ erhält man aus A_φ durch Komplementierung, also Determinisierung mit der Potenzmengenkonstruktion gefolgt von der Ersetzung von F durch $Q \setminus F$.

Den Automaten für $\exists X.\varphi$ erhält man aus A_φ, indem man den Inhalt der zu X gehörigen Spur nichtdeterministisch rät. Die Zustände von $A_{\exists X.\varphi}$ sind dieselben wie die von A_φ, auch Start- und Endzustände sind unverändert. Die Folgezustände von (q, a) umfassen alle Folgezustände von q in A_φ, die einem Symbol a' entsprechen, das mit a an allen Spuren bis auf der zu X gehörigen übereinstimmt. Darüberhinaus ist noch der Möglichkeit Rechnung zu tragen, dass X Elemente enthält, die größer oder gleich der Länge des Eingabewortes sind. Alle Zustände, von denen aus ein Pfad zu einem Endzustand existiert, dessen Beschriftungen allenfalls in der X-Spur eine Eins enthalten, werden daher zu Endzuständen gemacht.

Den Automaten für $\exists x.\varphi$ erhält man in ähnlicher Weise, wobei man zusätzlich dafür Sorge tragen muss, dass genau einmal eine 1 geraten wird und sonst immer nur 0er. Die detaillierte Konstruktion in diesem Falle verbleibt als Übung.

Aus der geschilderten Konstruktion ergeben sich nun wichtige Folgerungen:

Theorem 10. *Sei φ eine Formel und I eine Belegung. Es ist entscheidbar, ob $I \models \varphi$.*

Theorem 11 (Büchi-Elgot, '60). *Eine Sprache $L \subseteq \Sigma^*$ ist regulär gdw. sie MSO-definierbar ist.*

Zuletzt analysieren wir noch die worst-case-Komplexität der Übersetzung einer MSO-Formel in einen NFA. Seien $\mathcal{A}_1$ und $\mathcal{A}_2$ zwei NFAs mit jeweils höchstens n Zuständen. Dann hat ein NFA

- für die Vereinigung höchstens $2n + 1$,
- für das Komplement höchstens 2^n,
- für die existenzielle Quantifizierung höchstens n

Zustände. Damit kann eine Formel φ mit k logischen Operatoren zu einem NFA $\mathcal{A}$ mit $2^c_{O(k)}$ vielen Zuständen für ein $c \in \mathbb{N}$ führen, wobei

$$2^c_0 := c \qquad\qquad 2^c_{k+1} := 2^{2^c_k}$$

4.3 MONA: eine Implementierung der monadischen Logik

In diesem Abschnitt lernen wir die praktisch einsetzbare Implementierung *MONA* (www. brics.dk/mona) der schwachen monadischen Logik kennen.

Das Werkzeug MONA

- liest eine wMSO Formel,
- konstruiert den dazugehörigen Automaten,
- entscheidet mit dessen Hilfe, ob die Formel gültig, unerfüllbar, oder keines von beiden ist.
  - Falls erfüllbar, so wird eine erfüllende Interpretation minimaler Größse angegeben.
  - Falls nicht gültig, so wird eine falsifizierende Interpretation minimaler Größe angegeben ("Gegenbeispiel").
- Falls gewünscht, wird auch der Automat als Zustandstabelle angegeben.

Wir erläutern die Syntax und Verwendung von MONA hier anhand von Beispielen; für die formale Definition verweisen wir auf die Dokumentation.

4.3.1 Einfaches Beispiel

Sei `even.mona` eine Datei des folgenden Inhalts:

```
var2 A;
var1 maxi;
maxi = 9;
0 notin A &
all1 i: i < maxi => (i+1 in A <=> i in A)
```

Die ersten beiden Zeilen deklarieren zwei freie Variablen: `A` (zweitstufig) und `maxi` (erststufig). Die dritte Zeile entspricht der Formel

$$\exists x_0.\exists x_2 \ldots \exists x_9. \bigwedge_{i<9} succ(x_i, x_{i+1}) \wedge x_0 = 0 \wedge x_9 = maxi$$

und stellt sicher, dass *maxi* den Wert neun erhält.
Mit dem Befehl `mona even.mona` erhält man folgende erfüllende Interpretation:

$$A \hat{=} 0101010101$$
$$A = \{1, 3, 5, 7, 9\}$$
$$maxi = 9$$

Mit der Option `-w` erhält man auch den Automaten.

Die MONA-Datei kann, wie auch in diesem Beispiel, mehrere Formeln enthalten; diese verstehen sich als Konjunktion, sind also simultan zu erfüllen.

4.3.2 Binärzähler

Als nächstes wollen wir mit vier Mengenvariablen bis 16 zählen:

```
var2 A, B, C, D;
var1 maxi;
```

```
0 notin A & 0 notin B & 0 notin C & 0 notin D  &
maxi in A & maxi in B & maxi in C & maxi in D  &
all1 i: i < maxi =>
    (i+1 in A <=> i notin A)
  & ((i+1 in B <=> i notin B) <=> (i+1 notin A & i in A))
  & ((i+1 in C <=> i notin C) <=> (i+1 notin B & i in B))
  & ((i+1 in D <=> i notin D) <=> (i+1 notin C & i in C)))
;
```

An der Stelle 0 ist $A = B = C = D = 0$ (formal also $\neg A(0)$, etc.), an der Stelle *maxi* ist $A = B = C = D = 1$. Dazwischen schreiten A, B, C, D wie ein Binärzähler fort. Somit muss *maxi* mindestens 15 sein.

Wir können also mit k Mengenvariablen eine Mindestgröße des Modells von 2^k erreichen. Durch Verfeinerung dieser Methode kann man noch viel größere Modelle erzwingen, aber dazu später mehr.

4.3.3 Spezifikation der Addition von Binärzahlen beliebiger Größe

Im folgenden Beispiel verwenden wir endliche Mengen von natürlichen Zahlen, also die Werte zweitstufige Variablen, um Bitmuster, also z.B. Binärzahlen zu repräsentieren. So repräsentiert man z.B. das Bitmuster 0110101 entsprechend der Binärzahl $(1010110)_2 = 86$ als $\{1, 2, 4, 6\}$ und es ist $86 = 2^6 + 2^4 + 2^2 + 2^1$.

Auf diese Weise repräsentiert also eine zweitstufige Variable ein Bitmuster beliebiger Länge; man kann das verwenden, um "Leitungen" in Familien logischer Schaltkreise, die mit einer Wortlänge parametrisiert sind, darzustellen. Wir tun dies hier für eine Familie von Addierwerken.

Wir geben uns eine Wortlänge $ vor:

```
var1 $;
```

Hier ist $ ein ganz normaler Bezeichner.

Die folgenden beiden Direktiven relativieren freie und gebundene Variablen auf den Bereich 0..$:

```
defaultwhere1(p) = p <= $;
defaultwhere2(P) = P sub {0,...,$};
```

Ab jetzt bedeutet also zum Beispiel `all1 i: ...` in Wirklichkeit `all1 i: i <= $ => ...`. Wir definieren folgende Hilfsprädikate:

```
pred at_least_two(var0 A,var0 B,var0 C) =
       (A & B) | (A & C) | (B & C);
pred mod_two(var0 A,var0 B,var0 C,var0 @d) = (A <=> B <=> C <=> @d);
```

Das erste drückt aus, dass mindestens zwei von drei Boole'schen Variablen wahr sind. Die zweite drückt aus, dass *@d* die Summe modulo 2 von A, B, C ist.

Solche Hilfsprädikate können als Abkürzungen ähnlich wie Makros in der Programmiersprache C verstanden werden. Für die "nulltstufigen" Boole'schen Variablen können beliebige Formeln eingesetzt werden. MONA erlaubt Boole'sche ("nulltstufige") Variablen auch in quantifizierter Position.

Diese verstehen sich als Konjunktion ($\forall$), bzw. Disjunktion ($\exists$), über die entsprechenden Wahrheitswerte. Z.B. bedeutet $\forall A.\phi(A)$ dann $\phi(0) \wedge \phi(1)$.

Das nun folgende Prädikat drückt aus, dass *Result* die Summe von X und Y ist. Dabei werden die zweitstufigen Variablen *Result*, X, Y jeweils als Binärzahlen aufgefasst. *Cin* ist ein initialer Übertrag; *Cout* ist der resultierende Übertrag:

```
pred add(var2 X, var2 Y, var2 Result, var0 Cin, var0 Cout) =
ex2 C: (0 in C <=> Cin)
      & (all1 p:
           mod_two(p in X, p in Y, p in C, p in Result)
      & (p < $ =>  ((p+1 in C)
                 <=> at_least_two(p in X, p in Y, p in C))))
      & (Cout <=> at_least_two($ in X, $ in Y, $ in C));
```

Die existentiell quantifizierte Variable C bezeichnet die Übertragsbits. Generell bietet sich existentielle Quantifizierung für die Modellierung interner Leitungen an.

Die wMSO erweitert die klassische Aussagenlogik um eine weitere "unendliche" Dimension: eine zweitstufige Variable bezeichnet gleich einen ganzen Vektor Boole'scher Variablen beliebiger Länge. Im Beispiel nutzen wir diese zusätzliche Dimension, um parametrisierte Schaltkreise zu modellieren (Addieren von $-vielen Bits statt nur 3 oder 5 Bits). Alternativ oder manchmal sogar gleichzeitig lässt sich die unendliche Dimension zur Modellierung der Zeit einsetzen.

4.3.4 Synchronaddierer

Als nächstes definieren wir einen "effizienteren" Addierer, in dem der Übertrag separat vorab berechnet wird. Dadurch muss nicht auf das Ergebnis des jeweils vorherigen Ergebnisbits gewartet werden.

```
pred sync_add(var2 X, var2 Y, var2 Result, var0 Cin, Cout) =
   ex2 C:
      (all1 i:i in C <=> i=0 & Cin |
           ex1 j: j<i & (all1 k:j<k&k<i=> k in X | k in Y) &
                at_least_two(j in X, j in Y, j=0&Cin)) &
      (all1 p: mod_two(p in X, p in Y, p in C, p in Result))      &
      (Cout <=> at_least_two($ in X, $ in Y, $ in C));
```

Das Übertragsbit ist an Position i gesetzt, wenn entweder $i = 0$ und *Cin* gesetzt ist, oder aber an einer früheren Position ein Übertrag aufgetreten ist und zwischen i und j nicht wieder verschluckt wurde.

Wir können nun formulieren, dass beide Versionen äquivalent sind:

```
all2 X,Y,Z: all0 Cin, Cout:
  add(X,Y,Z,Cin,Cout) <=> sync_add(X,Y,Z,Cin,Cout);
```

Das ist in der Tat der Fall, wie die folgende Mona-Ausgabe zeigt.

```
Automaton has 3 states and 3 BDD-nodes
ANALYSIS
Formula is valid
```

Als Gegenprobe ändern wir $j < i$ zu $j \leq i$ um: die Formel wird unerfüllbar. Wir können auch konkret Ergebnisse ausrechnen:

```
var2 X, Y, Z;
var0 Cout;
$ = 5;
X = {2,3}; # X=12
Y = {0,2,3,5}; # Y=35
sync_add(X,Y,Z,false,Cout);
```

Ein (das) erfüllende(s) Modell setzt Z auf $\{0, 3, 4, 5\}$ und $Cout = \mathit{false}$.

4.4 Komplexität des Entscheidungsproblems für wMSO

Sei 2_n definiert durch $2_0 = 1$ und $2_{n+1} = 2^{2_n}$. Wir haben bereits einen Algorithmus kennengelernt, der zu einer gegebenen wMSO Formel der Länge n einen Automaten der Größe $2_{O(n)}$ bildet und daraus abliest, ob die Formel erfüllbar ist. Wir wollen jetzt zeigen, dass es prinzipiell nicht effizienter gehen kann in folgendem Sinne.

Theorem 12. *Sei T eine Turingmaschine und p ein Polynom, so dass T bei Eingabe eines Wortes w nach spätestens $2_{p(|w|)}$ Schritten hält. Es existiert eine in polynomialer Zeit berechenbare Funktion f, die jedem Wort w eine wMSO-Formel $f(w)$ zuordnet, derart, dass $f(w)$ wahr ist, genau dann, wenn T das Wort w akzeptiert.*

Somit ist das Entscheidungsproblem für wMSO nicht leichter, als irgendein beliebiges anderes Problem der Zeitkomplexität $2_{\mathrm{poly}(n)}$. Aus dem Zeithierarchiesatz der Komplexitätstheorie, siehe etwa [**?**], folgt dann, dass die angegebene obere Schranke für das Entscheidungsproblem nicht essentiell verbessert werden kann.

Zum Beweis des Satzes geben wir uns also eine solche Maschine T und Polynom p vor. Die Maschine T akzeptiert w, genau dann wenn es ein Wort

$$\tilde{w} = w_0 \# w_1 \# w_2 \# w_3 \# w_4 \# w_5 \# w_6 \# w_7 \# \ldots \# w_N$$

gibt, wobei $N = 2_{p(|w|)}$, die Raute (#) ein besonderes (Trenn-)Symbol ist, und die w_i, $i = 0, \ldots, N$ globale Konfigurationen (Zustand, Kopfposition und Bandinhalt) kodieren, in einer Weise, dass:

- w_0 die initiale Konfiguration von T mit Eingabe w ist,
- w_{i+1} die Folgekonfiguration von w_i gemäß der Übergangstafel von T ist (i.Z. manchmal $w_i \vdash_T w_{i+1}$)
- w_N eine akzeptierende Konfiguration ist.

Durch Wahl einer geeigneten Kodierung können wir annehmen, dass alle w_i die Länge N haben.

Die genannten Eigenschaften an solch ein Wort $\tilde{w}$ lassen sich sehr einfach in wMSO spezifizieren, sofern es nur gelingt, ein Prädikat $dist(x, y)$ zu definieren, derart dass $dist(x, y) \iff y - x = N$. Indem wir die w_i ggf. mit Leerzeichen künstlich verlängern, sehen wir, dass schon $dist(x, y) \iff y - x = N'$ wobei $N' \geq N$ genügen würde.

Ein solches wollen wir jetzt definieren. Natürlich muss die Größe der Formel $dist(x, y)$ polynomiell in n sein, sonst entstünde keine polynomielle Reduktion; die oben skizzierte Übersetzungsfunktion f wäre also nicht in polynomieller Zeit berechenbar.

Wir dürfen also insbesondere nicht einfach schreiben

$$dist(x, y) \iff y = x + 1 + 1 + 1 + 1 + 1 + \cdots + 1 \ (N + 1 \text{ Summanden})$$

Sei die Funktion F definiert durch $F(0) = 1, F(n + 1) = F(n)2^{F(n)}$.

Proposition 1. *Zu gegebenem $n \in \mathbb{N}$ lässt sich in polynomieller Zeit eine Formel $dist_n(x, y)$ berechnen, derart dass $dist_n(x, y) \iff y - x = F(n)$. Insbesondere ist die Länge von $dist_n(x, y)$ polynomiell in n.*

Beweis. Man definiert die Formeln rekursiv über n wie folgt.

Wir setzen $dist_0(x, y) : \iff y = x + 1$. Wegen $F(0) = 1$ leistet dies das Verlangte.

Sei jetzt $dist_n(x, y)$ bereits konstruiert. Um $dist_{n+1}(x, y)$ zu definieren, verlangen wir die Existenz eines Wortes, das zwischen x und y alle Binärzahlen der Länge $dist_n(x, y)$ der Reihe nach hintereinandergeschrieben enthält. Gelingt es, das zu erzwingen, so gilt gerade $y - x = F(n + 1)$ wie verlangt.

Hier ist die Lösung in MONA-Notation. Man beachte, dass man *dist* nicht uniform in n definieren kann, es gibt also keine wMSO-Formel $\phi(n, x, y)$ in drei Variablen derart, dass $\phi(n, x, y) \iff dist_n(x, y)$ wäre, aber das wird ja zum Glück auch nicht benötigt.

```
pred distnpluseins(var1 x,y) =
   ex2 B, C:
   # Abstand mindestens 1
        x+1<y &
   # a,b durchlaufen alle Paare mit Abstand distn und  x<=a<y
     all1 a: x <= a & a < y => ex1 b: distn(a,b) &
   # y-x ist mindestens distn
      (a=x => b<=y) &
   # Initialisierung des ersten Blocks: B=10...0, C=00...0
     (a=x => a in B & a notin C &
           all1 c:a<c&c<b => c notin B & c notin C) &
   # B-Besetzung in alle Bloecke kopieren.
     (a in B <=> b in B) &
   # Verlangen, dass bei y wieder ein Block anfaengt, also (b-a) | (y-x)
     (b=y => b in B) &
```

```
# Besetzung des letzten Blocks zu C=11....1
  (b=y => all1 c: (a<=c & c<b) => c in C) &
# Das erste C-bit jeden Blocks wechselt von Block zu Block
  (b in B  => (a in C <=> b notin C)) &
# Die folgenden C-bits jeden Blocks wechseln, wenn das
#  vorhergehende Bit von Eins auf Null geht.
  (a+1 notin B  =>
    ((b+1 notin C <=> a+1 in C) <=> (b notin C & a in C)));
```

Zu Testzwecken kann man definieren:

```
pred distn(var1 x, y) = y=x+5;
var1 x, y;
distnpluseins(x,y);
```

und erhält als Antwort: $x = 0, y = 160$.

Indem man B, C zu freien Variablen macht, kann man sich deren Wertverlauf mit MONA ansehen. Man kann auch einzelne Konjunkte weglassen, um das Funktionsprinzip von *distnpluseins* zu studieren.

Dieses Ergebnis geht auf Stockmeyers Dissertation zurück, in der sich außerdem die folgende sehr konkrete Version findet.

Proposition 2 (Stockmeyer). *Jeder Boole'sche Schaltkreis, der die Erfüllbarkeit von wMSO-Formeln der Länge 613 entscheidet, hat mindestens* 10^{128} *Verbindungsleitungen.*

In der Praxis funktioniert MONA auch mit längeren Formeln ganz gut. Das liegt daran, dass viele Formeln, die aus der Praxis kommen, relativ einfach sind. Manchmal aber, speziell wenn man sich vertippt, kann MONA auch in eine de facto Endlosschleife geraten.

4.5 Presburger Arithmetik

Unter der Presburger Arithmetik versteht man die erststufige Logik über den natürlichen Zahlen und der Signatur $0, +, =, <$.

Hier sind drei Formeln der Presburger Arithmetik, die letzte verwendet Abkürzungen.

$\exists x.y = x + x$	"y ist gerade"
$\exists xr.r < 5 \wedge y = x + x + x + x + x + r$	"$y = r \bmod 5$"
$\exists z.\forall x > z.x = 0 \bmod 3 \Rightarrow \exists uv.x = 15u + 27v$	

Quantifikation über Mengen von natürlichen Zahlen so wie in der wMSO gibt es in der Presburger Arithmetik nicht.

Es gibt in der Presburger Arithmetik auch keine Multiplikation und man kann diese auch nicht irgendwie definieren; insbesondere gibt es keine Formel $\phi(x, y)$, derart dass $\phi(x, y) \iff y = x^2$ wäre.

Die Allgemeingültigkeit und die Erfüllbarkeit von Formeln der Presburger Arithmetik ist durch Übersetzung in wMSO entscheidbar:

Variablen der Presburger Arithmetik werden immer in Mengenvariablen übersetzt: die "Bedeutung" einer solchen Mengenvariablen ist durch die Binärkodierung gegeben. So wird 42 etwa durch $\{5, 3, 1\}$ kodiert, da ja $42 = 2^5 + 2^3 + 2^1$

Die Relation $x = y+z$ lässt sich nun durch eine wMSO Formel beschreiben, wobei man im Prinzip so vorgeht wie bei dem Addierwerk aus Abschnitt 4.3.3. Die Relation $x < y$ ersetzt man durch $\exists z.y = x+z+1$. Somit wird es möglich, zu jeder Presburger Formel ϕ eine wMSO Formel $\hat{\phi}$ anzugeben, derart dass ϕ erfüllbar ist, gdw. $\hat{\phi}$ erfüllbar ist. Folglich ist die Presburger Arithmetik entscheidbar.

Es gibt andere Entscheidungsverfahren für die Presburger Arithmetik, die auf Quantorenelimination (Verallgemeinerung von Gauss-Elimination) beruhen. Diese liefern insbesondere ein Verfahren mit Zeitkomplexität $DTIME(2^{2^{2^{cn}}})$.

Dagegen hat das angegebene Verfahren anscheinend die schlechtere Komplexität $DTIME(2_{cn})$ (vgl. Satz **??**). In der Praxis zeigt sich aber, dass Automaten für wMSO Formeln, die aus der Übersetzung von Presburger Formeln entstehen, relativ klein bleiben und das resultierende Entscheidungsverfahren mit den auf Quantorenelimination basierenden konkurrieren kann. Kürzlich [**?**] wurde eine theoretische Begründung für diese empirische Beobachtung geliefert:

Proposition 3. *Der minimale deterministische Automat für eine gegebene wMSO-Formel $\hat{\phi}$, wobei ϕ eine Presburger-Formel ist, hat Grösse $2^{2^{O(n)}}$, wobei n die Länge von ϕ ist.*