Die Logiken von Boole und Büchi-Elgot-Trakhtenbrot in der Beschreibung diskreter Systeme

Wolfgang Thomas
Lehrstuhl für Informatik VII, RWTH Aachen, 52056 Aachen

Zusammenfassung

Die von Büchi, Elgot und Trakhtenbrot um 1960 eingeführte Logik zur Beschreibung des Verhaltens endlicher Automaten wird in einer intuitiven Darstellung vorgestellt, und zwar als Erweiterung der Booleschen Logik um einen Parameter für diskrete Zeit. Ausgehend von der Charakterisierung der Booleschen Logik durch OBDD's ("ordered binary decision diagrams") wird die Äquivalenz dieser "Büchi-Elgot-Trakhtenbrot-Logik" mit dem Berechnungsmodell des endlichen Automaten erläutert. Abschließend werden Anwendungen und didaktische Aspekte der Theorie diskreter Systeme diskutiert.

1 Einleitung

Vor etwa 150 Jahren veröffentlichte George Boole sein Werk *An Investigation of the Laws of Thought* [Bo1854], in dem die "arithmetischen" Gesetze der Aussagenlogik, der *Booleschen Logik*, in einer bis heute gültigen Form dargestellt sind. Boole verstand seine Theorie als Beitrag zur Grundlagenforschung der Logik. Es dauerte ein Jahrhundert, bis sie Grundlage der Schaltkreistheorie wurde und damit auch zu einem Fundament der Informatik. In diesem Kontext sprechen wir heute statt von Wahrheitswerten eher von "Bits". Die Verschmelzung der Booleschen Logik mit der Welt der Informatik-Systeme geht heute so weit, dass Ingenieure unter "Logik" nun eher eine Schaltung verstehen als eine logische Formel. Die Boolesche Logik ist also nicht mehr nur ein Kapitel der mathematischen Logik, sondern ebenso ein elementarer Bestandteil der Informatik und der Ingenieurwissenschaften im allgemeinen.

Vor 40 Jahren veröffentlichten Büchi und Elgot ([BüEl58], [Büc60], [Elg61]) sowie unabhängig davon Trakhtenbrot ([Tra58], [Tra62]) erste Arbeiten zu einer Logik, die als Erweiterung der Booleschen Logik durch einen Zeitparamater (für diskrete Zeit) verstanden werden kann. Eine Motivation für die Konzeption dieses Logik-Formalismus war der Wunsch, eine für die Verhaltensbeschreibung von Schaltungen adäquate Sprache zu entwickeln. Doch standen für Büchi, Elgot und Trakhtenbrot zunächst ebenfalls Anwendungen im Bereich der mathematischen Logik im Vordergrund. Insbesondere sollte die Entscheidbarkeit gewisser arithmetischer Theorien gezeigt werden; es ging u.a. um die "monadische (bzw. schwache monadische) Theorie zweiter Stufe der Nachfolgerfunktion mit einstelligen Prädikaten". Diese abschreckende und abseitig klingende Bezeichnung täuscht leicht darüber hinweg, dass es sich um einen ebenso fundamentalen Formalismus wie den der Booleschen Logik handelt, mit derselben universellen (und damit auch: praktischen) Bedeutung. Wir wollen im folgenden etwas kürzer von der *Büchi-Elgot-Trakhtenbrot-Logik* sprechen.

Gemeinsam ist beiden Logik-Systemen, dass ihre Formeln algorithmisch in die Form von Schaltungen, Diagrammen bzw. Automaten transformiert werden können, d.h. in eine "operationale" Darstellungsform. Hinzu kommt, dass die zentralen Fragen über diese Logiken algorithmisch lösbar sind. So gibt es Tests für die Erfüllbarkeit und für die Äquivalenz von Formeln (die auch effizient durchführbar sind, wenn man von der operationalen Darstellung ausgeht). Diese Sachverhalte sind die Basis für viele Verfahren der automatisierten Konstruktion bzw. Verifikation von Systemen (Schaltkreisen, Programmen). Zwar werden in der heutigen Praxis der Programmverifikation für zustandsbasierte Systeme vorwiegend verwandte Logiken eingesetzt (insbesondere temporale Logiken), doch lassen sich diese in die Büchi-Elgot-Trakhtenbrot-Logik einbetten bzw. als Varianten davon verstehen.

Im vorliegenden Essay soll versucht werden, die Boolesche und die Büchi-Elgot-Trakhtenbrot-Logik auf intuitiver Ebene und in integrierter Form darzustellen. Insbesondere werden die Bezüge zur Automatentheorie in einer neuen (vielleicht ungewöhnlichen) Form erläutert; Analogien und Querbezüge treten dann klarer hervor. Ein wichtiges Detail ist die Einbindung der durch Bryant [Bry92] konzipierten binären Entscheidungsdiagramme (*ordered binary decision diagrams*, kurz OBDD's), die zu einem großen Fortschritt bei der algorithmischen Behandlung Boolescher Funktionen geführt haben. Scharnier zwischen Boolescher Logik und Büchi-Elgot-Trakhtenbrot-Logik wird der Begriff des (Booleschen) Quantors sein; wir gehen also vom Formalismus der quantifizierten Booleschen Formeln aus.

Zum Schluss diskutieren wir noch die Plazierung dieser Theorie und verwandter Gebiete im Curriculum technischer Fächer.

Da hier aus Platzgründen eine detaillierte Entwicklung unterbleiben muss, unterstellen wir beim Leser eine grobe Vertrautheit mit den elementaren Begriffsbildungen aus Logik und Automatentheorie.

2 Boolesche Logik und Büchi-Elgot-Trakhtenbrot-Logik

2.1 Boolesche Logik

Wir wollen diskrete Systeme betrachten, deren Zustände durch n-Tupel von Bits (Booleschen Werten) gegeben sind. In diesem Fall lässt sich der Zustandsraum durch eine Bedingung an solche n-Tupel $(b_1, \ldots, b_n)$ beschreiben, d.h. durch eine Boolesche Formel $\varphi(x_1, \ldots, x_n)$ in den Booleschen Variablen $x_1, \ldots, x_n$. Solche Formeln werden aus den Variablen $x_1, \ldots, x_n$ und den Konstanten $0, 1$ für die beiden Booleschen Werte wie üblich aufgebaut; wir wollen hier den einstelligen

Junktor $\neg$ ("nicht") sowie die zweistelligen Junktoren $\wedge$ ("und"), $\vee$ ("oder"), $\rightarrow$ ("wenn, so") und $\equiv$ ("genau dann, wenn") zulassen. Ein *Boolesches Modell* für Formeln $\varphi(x_1, \ldots, x_n)$ ist ein Bitvektor $\overline{b} = (b_1, \ldots, b_n)$. Wir sagen, dass φ in $\overline{b}$ *gilt*, wenn die (wie üblich definierte) Auswertung von φ bei der Belegung von x_i durch b_i den Wert 1 ergibt. Betrachten wir z.B. $\varphi_1 = (x_1 \wedge x_2) \vee (x_1 \wedge \neg x_2)$ und $\varphi_2 = x_1 \wedge (x_3 \rightarrow x_2)$ sowie das Modell $\overline{b} = (1, 0, 1)$. Dann gilt φ_1 in $\overline{b}$, jedoch gilt φ_2 nicht in $\overline{b}$.

Für manche Zwecke ist es nützlich, den Rahmen der Booleschen Formeln durch *Boolesche Quantoren* $\exists, \forall$ zu ergänzen, die über den Bereich $\{0, 1\}$ der beiden Booleschen Werte rangieren; wir sprechen dann von *quantifizierten Booleschen Formeln*. Die Anwendung von Quantoren führt dazu, dass Variablen gebunden werden; die nicht im Wirkungsbereich eines Quantors stehenden Variablen heißen freie Variablen. Zum Beispiel kommen in $\exists x_3 (x_1 \wedge x_3 \;\rightarrow\; x_2)$ die Variablen x_1, x_2 frei vor. Die Gültigkeitsbeziehung über Modellen $(b_1, \ldots, b_n)$ wird nun für Formeln $\varphi(x_1, \ldots, x_n)$ in den *freien* Variablen $x_1, \ldots, x_n$ erklärt. Die Bedeutung der Quantoren ergibt sich aus der Festlegung, dass in jedem beliebigen Modell $(b_1, \ldots, b_n)$ die Äquivalenz

$$\exists x_{n+1} \psi(x_1, \ldots, x_n, x_{n+1}) \;\equiv\; (\psi(x_1, \ldots, x_n, 0) \vee \psi(x_1, \ldots, x_n, 1))$$

gelten soll; analog wird beim Allquantor $\forall$ die Konjunktion $\wedge$ an Stelle der Disjunktion $\vee$ benutzt. Diese Erläuterung macht zugleich klar, dass durch Boolesche Quantoren die Ausdrucksstärke der Booleschen Logik nicht vergrößert wird; Boolesche Quantoren erlauben aber die Verkürzung Boolescher Formeln.

In diesem Sinne dienen Boolesche Quantoren der prägnanten und intuitiv leicht verständlichen Formulierung von Bedingungen. Man denke etwa an einen Schaltkreis mit den Eingabevariablen $x_1, \ldots, x_n, y_1, \ldots, y_m$, dessen Ausgabebit genau dann 1 sein soll, wenn der Vektor $(b_1, \ldots b_n, c_1, \ldots, c_m)$ der Eingabebits die Formel $\varphi(x_1, \ldots, x_n, y_1, \ldots, y_m)$ erfüllt. Will man nun ausdrücken, dass der Schaltkreis in $x_1, \ldots, x_n$ korrekt funktioniert, sofern nur die Komponenten $(c_1, \ldots, c_m)$ eine Bedingung ψ erfüllen, schreibt man einfach

$$\forall y_1 \ldots \forall y_m (\psi(y_1, \ldots, y_m) \rightarrow \varphi(x_1, \ldots x_n, y_1, \ldots, y_m)).$$

Wendet man Quantoren auf alle Variablen einer Booleschen Formel an, so kommen keine Variablen mehr frei vor, und die Auswertung der Formel ist möglich ohne Bezug auf ein Boolesches Modell. Man sagt dann, die Formel habe den Wert 0 bzw. 1. Damit lassen sich dann die Begriffe der Erfüllbarkeit und Allgemeingültigkeit Boolescher Formeln einfach fassen: Eine Boolesche Formel $\varphi(x_1, \ldots, x_n)$ ist erfüllbar (bzw. allgemeingültig) genau dann, wenn die quantifizierte Formel $\exists x_1 \ldots \exists x_n \varphi(x_1, \ldots, x_n)$ (bzw. $\forall x_1 \ldots \forall x_n \varphi(x_1, \ldots, x_n)$) den Wert 1 hat.

Das Erfüllbarkeitsproblem für Boolesche Ausdrücke $\varphi(x_1, \ldots, x_n)$ (ohne Boolesche Quantoren) führt auf das prominenteste offene Problem der Komplexitätstheorie. Das naive Verfahren zum Test, ob $\exists x_1 \ldots \exists x_n \varphi(x_1, \ldots, x_n)$ den Wert 1 hat (also φ erfüllbar ist), besteht im Ausprobieren aller 2^n Bitvektoren der Länge n (mit jeweiliger Auswertung der Formel φ). Dies bedeutet dass der Zeitaufwand in n exponentiell wächst, und man weiß bis heute nicht, ob auch ein Verfahren mit nur polynomial vielen Rechenschritten existiert. Eine Verallgemeinerung dieses Problems betrifft Formeln der Gestalt

$$\exists \overline{y_1}\ \forall \overline{y_2}\ \exists \overline{y_3}\ \ldots\ \exists/\forall \overline{y_k}\ \varphi(\overline{y_1}, \overline{y_2}, \ldots, \overline{y_k})$$

wobei die $\overline{y_i}$ Variablentupel der Länge n seien und φ wieder keine Booleschen Quantoren enthalte. Mit Bezug auf die Anzahl der Quantorenblöcke spricht man von *Σ_k-Formeln.* Der naive Ansatz zum Test auf Gültigkeit führt hier zu einem k-fach exponentiellen Rechenaufwand; und wieder ist offen, ob mit wachsendem k ein echt wachsender Aufwand tatsächlich nötig ist. Für $k = 1, 2, \ldots$ bestimmt das Gültigkeitsproblem für Σ_k-Formeln die k-te Stufe der sog. *Polynomzeithierarchie* – eine Redeweise, die ein echtes Anwachsen der Komplexität vielleicht unzulässigerweise unterstellt.

2.2 Büchi-Elgot-Trakhtenbrot-Logik

Das dynamische Verhalten diskreter Systeme lässt sich in der Booleschen Logik nur rudimentär beschreiben, nämlich unter Berücksichtigung nur beschränkt vieler Zeitpunkte. Betrachtet man zum Beispiel nur zwei benachbarte Zeitpunkte, kann man wie folgt verfahren: Man führt für jede Systemvariable x_i ($i = 1, \ldots, n$) eine Kopie x'_i ein (für den "Wert von x_i zum nächsten Zeitpunkt") und formuliert eine Boolesche Bedingung an $x_1, \ldots, x_n, x'_1, \ldots, x'_n$. Will man das Systemverhalten aber über eine in der Länge nicht festgelegte Folge von Zeitpunkten verfolgen, so ist eine Erweiterung der Logik (und der betrachteten Modelle) erforderlich.

Auf der Seite der Modelle betrachtet man dann *Folgen* $(\overline{b}(0), \ldots, \overline{b}(t_{\max}))$ von Bitvektoren. Ist man an der Untersuchung nichtterminierender Systeme interessiert, lässt man die unendliche Folge der Zeitpunkte $t = 0, 1, \ldots$ zu, und es ergeben sich Modelle der Form $(\overline{b}(0), \overline{b}(1), \ldots)$. Wir beschränken uns im folgenden auf den ersten Fall, da er in natürlicher Weise auf die Standardtheorie endlicher Automaten führt. Im zweiten Fall ist hat man es mit der Theorie der ω-Automaten zu tun, mit ähnlichen Resultaten, jedoch komplexeren Begriffsbildungen und Beweisen, die wir hier überschlagen wollen (vgl. etwa [Tho90]).

In einer Folge von Bitvektoren der Form $(\overline{b}(0), \ldots, \overline{b}(t_{\max}))$ denken wir uns die $\overline{b}(i)$ als Spaltenvektoren geschrieben, zum Beispiel so:

$$\binom{0}{0}\binom{0}{0}\binom{1}{0}\binom{0}{1}\binom{0}{1}\binom{1}{1}\binom{0}{0}\binom{1}{0}\binom{1}{1}\binom{0}{0}\binom{0}{0}\binom{0}{0}$$

Die i-te Komponente des Modells ist die Folge der Einzelbits $b_i(0), \ldots, b_i(t_{\max})$, die sich als i-te Zeile aus dem Array der Folge der Spaltenvektoren ergibt. In den Formeln der Büchi-Elgot-Trakhtenbrot-Logik rangiert die Variable x_i nun über solche Bitfolgen (nicht a priori festgelegter Länge) und nicht mehr über die Einzelbits 0 und 1. In diesem Sinne könnte man die Boolesche Logik auch als *Bit-Logik* und die Büchi-Elgot-Trakhtenbrot-Logik auch als *Bitfolgen-Logik* bezeichnen. [1]

Damit wird die Syntax der Büchi-Elgot-Trakhtenbrot-Logik motiviert: An Stelle der atomaren Formeln x_i der Booleschen Logik treten atomare Formeln der Gestalt $x_i(\tau)$ mit der Bedeutung "zum Zeitpunkt τ hat x_i den Wert 1"; hierbei ist τ ein *Zeitterm*. Als Zeitterme lassen wir die Konstante 0 (für den Initialzeitpunkt), $t_{\max}$ (für den Endzeitpunkt des betrachteten Modells) und Zeitvariable $s, t, t_1, \ldots$ zu, sowie Terme, die sich hieraus durch Anwendung der Nachfolgerfunktion (geschrieben $'$) ergeben. Zum Beispiel bezeichnet also $0'''$ den Zeitpunkt 3 und t' den nächsten Zeitpunkt nach t. Aus den atomaren Formeln $x_i(\tau)$ entstehen die Formeln der Büchi-Elgot-Trakhtenbrot-Logik nun wiederum durch Anwendung der Junktoren $\neg$, $\wedge$, $\vee$, $\rightarrow$, $\equiv$ sowie der Quantoren $\exists$, $\forall$. Ein Quantor darf auf eine Zeitvariable t oder auf eine Bitfolgen-Variable x_i angewendet werden. Im ersten Fall rangiert er über die Zeitpunkte von 0 bis $t_{\max}$, im zweiten Fall über Funktionen $b : \{0, \ldots, t_{\max}\} \rightarrow \{0, 1\}$.

Wie üblich ist nun festgelegt, wann in einem Modell $(\overline{b}(0), \ldots, \overline{b}(t_{\max}))$ eine Formel $\varphi(x_1, \ldots, x_n)$ *gilt*. (Ein kritischer Punkt in der Definition ist die Festlegung, was der "Nachfolgerzeitpunkt von $t_{\max}$" sein soll; hier vereinbaren wir, dass dies wieder $t_{\max}$ sein soll.) Die durch $\varphi(x_1, \ldots, x_n)$ *definierte Menge* enthalte genau diejenigen Bitvektor-Folgen $(\overline{b}(0), \ldots, \overline{b}(t_{\max}))$, in denen $\varphi(x_1, \ldots, x_n)$ gilt.

Wir verwenden im folgenden auch die $<$-Beziehung zwischen Zeittermen. Dies erweitert die Ausdruckskraft der Logik nicht, da man in der Büchi-Elgot-Trakhtenbrot-Logik eine entsprechende Definition angeben kann: Es gilt nämlich $t_1 < t_2$ genau dann, wenn eine Bitfolge existiert, die mit 0 beginnt, schwach monoton aufsteigend ist, bei t_1 noch den Wert 0 hat, bei t_2 jedoch den Wert 1. Kurz: $t_1 < t_2$ wird durch folgende Formel ausgedrückt:

$$\exists x_1(\neg x_1(0) \ \wedge \ \forall t(x_1(t) \rightarrow x_1(t')) \ \wedge \ \neg x_1(t_1) \ \wedge \ x_1(t_2))$$

Betrachten wir ein Beispiel, unter das auch die oben angegebene Beispielfolge aus Bitpaaren fällt: Die zweite Komponente beginnt mit dem Eintrag 0, behält ihren Wert jeweils bei, wenn die erste Komponente zuvor 0 war, wechselt aber nach einer 1 in der ersten Komponente jeweils in der zweiten Komponente von 0 auf 1 bzw. zurück von 1 auf 0. Eine definierende Formel der Büchi-Elgot-Trakhtenbrot-Logik für die Menge solcher Bitpaar-Folgen ist

$$\begin{aligned} \varphi_0(x_1, x_2) \ :=& \ \ \neg x_2(0) \ \wedge \ \forall t(t < t_{\max} \ \rightarrow \\ & ((\neg x_1(t) \ \rightarrow \ x_2(t') \equiv x_2(t)) \ \wedge \ (x_1(t) \ \rightarrow \ x_2(t') \equiv \neg x_2(t)))). \end{aligned}$$

[1] In der Standardterminologie der Logik wird der Akzent etwas anders gesetzt: Hier spricht man von der *schwachen monadischen Theorie zweiter Stufe der Nachfolgerfunktion*. Dadurch wird hervorgehoben, dass auf dem Grundbereich $\{0, \ldots, t_{\max}\}$ eines Modells die Nachfolgerfunktion $'$ zur Verfügung steht und neben Quantoren $\exists t, \forall t$ über Elemente dieses Bereichs auch Quantoren $\exists x, \forall x$ über Teilmengen des Bereichs bereitstehen. (Man identifiziert eine Bitfolge mit der Menge der Stellen, wo der Wert 1 vorliegt, also mit einer Teilmenge von $\{0, \ldots, t_{\max}\}$.) Da Relationen als "Objekte zweiter Stufe" zählen und Mengen als unäre Relationen betrachtet werden, laufen die Quantoren $\exists x, \forall x$ also über "monadische Objekte zweiter Stufe". Die Theorie heißt "schwach monadisch", weil diese Mengen hier immer endlich sind.

Mit dem "Umschalten" der zweiten Komponente nach Einträgen 1 in der ersten Komponente werden die Vorkommen dieser 1-Werte modulo 2 gezählt: In der ersten Komponente kommt eine gerade Anzahl von Einträgen 1 vor, wenn der letzte Eintrag in der zweiten Komponente, also $b_2(t_{\max})$, gleich 0 ist (abgesehen von dem Fall, dass auch $b_1(t_{\max}) = 1$ ist; dann muss auch $b_2(t_{\max}) = 1$ sein). Kurz: Die erste Komponente weist geradzahlig viele Bits 1 auf genau dann, wenn die letzten Bits der beiden Komponenten übereinstimmen. Man kann also die Menge der Bitfolgen $(b_1(0), \ldots, b_1(t_{\max}))$ mit geradzahlig vielen Werten 1 definieren durch die folgende Formel:

$$\varphi_1(x_1) \quad := \quad \exists x_2 \ (\varphi_0(x_1, x_2) \ \wedge \ x_1(t_{\max}) \equiv x_2(t_{\max}))$$

Es sei bemerkt, dass diese Menge der Bitfolgen mit geradzahlig vielen Einträgen 1 nur mit Zuhilfenahme eines Bitfolgen-Quantors definiert werden kann. (Der Beweis erfordert eine tiefere Analyse der Logik, die sich nach Verzicht auf die Quantoren $\exists x_i$, $\forall x_i$ ergibt; er muss hier übergangen werden.) Die Situation ist also anders als bei der Booleschen Logik, wo die Quantoren $\exists x_i$, $\forall x_i$ ohne Verlust an Ausdrucksstärke eliminierbar sind.

Allgemein lassen sich viele Systemeigenschaften, die die Zeitabfolge von Zuständen betreffen, in der Büchi-Elgot-Trakhtenbrot-Logik ausdrücken. Häufig ist man nur an Formeln $\varphi(x_1, \ldots, x_n)$ interessiert, in denen zwar das $<$-Zeichen zugelassen ist, aber keine Quantoren über Bitfolgen auftreten (so dass nur noch Quantoren über Zeitpunkte verbleiben). Dieser Teilformalismus ist ausdrucksäquivalent zur *präpositionalen temporalen Logik*, die in vielen Anwendungen benutzt wird. Im Vergleich zur Büchi-Elgot-Trakhtenbrot-Logik ist für die temporale Logik eine Notation kennzeichnend, in der Quantoren für Zeitpunkte nur verdeckt (in Form sog. temporaler Operatoren) auftreten. Hierdurch entfallen die Variablen $s, t, t_1, \ldots$ in der Syntax, und es werden sehr kompakte Formulierungen möglich.

Ein Beispiel (wieder für Folgen von Bitpaaren) mag dies verdeutlichen. Wir betrachten die Bedingung "immer wenn die erste Komponente den Wert 1 hat, hat vom nächsten Zeitpunkt an die zweite Komponente den Wert 1, bis schließlich wieder die erste Komponente den Wert 1 hat." Eine entsprechende temporale Formel benutzt x_1, x_2 als Aussagevariablen (!) sowie die Operatoren G ("globally", "immer"), X ("next", "zum nächsten Zeitpunkt") und U ("until", "bis"):

$$\mathrm{G}(x_1 \rightarrow \mathrm{X}(x_2 \mathrm{U} x_1)).$$

Die Übertragung in die Büchi-Elgot-Trakhtenbrot-Logik lautet:

$$\forall t_1(x_1(t_1) \rightarrow \exists t_2(t_1 < t_2 \wedge x_1(t_2) \wedge \forall s(t_1 < s \wedge s < t_2 \rightarrow x_2(s))))$$

Wie für die Boolesche Logik wollen wir auch für die Formeln der Büchi-Elgot-Trakhtenbrot-Logik die grundlegenden Fragen nach deren Erfüllbarkeit, Allgemeingültigkeit oder deren Äquivalenz untersuchen. Lassen sich diese Probleme hier ebenfalls algorithmisch lösen? Es gibt eine prinzipielle Schwierigkeit dadurch, dass z.B. beim Erfüllbarkeitstest die Anzahl der in Frage kommenden Modelle nicht mehr endlich ist. Während bei Booleschen Formeln mit n

freien Variablen nur 2^n mögliche Modelle existieren, ist für die Büchi-Elgot-Trakhtenbrot-Formeln die Länge der Modelle (d.h. der Bitvektor-Folgen) nicht durch eine feste Zahl beschränkt. Damit scheidet ein "Test durch Ausprobieren" aus.

In den folgenden Abschnitten skizzieren wir eine Lösung dieses Problems, wieder beginnend mit einer Diskussion der Booleschen Logik. Wir entwickeln eine "operationale Darstellung" von Formeln, d.h. eine algorithmische Form durch Diagramme. Diese Diagramme beschreiben, wie man in beliebigen Modellen den zugehörigen Wahrheitswert (0 oder 1) der Formel ausrechnen kann, und sind insofern mit den Formeln "äquivalent". Es stellt sich zugleich heraus, dass man diese algorithmische Form der Darstellung in eine "bestmögliche Form" (eine "minimale Darstellung") bringen kann – ein Unterfangen, das auf der Ebene der Formeln in beiden betrachteten Logiken ein tiefliegendes, noch völlig unverstandenes Problem ist.

3 Diagramme, Automaten und ihre Minimierung

Wir stellen zwei Verfahren vor, mit denen man prüfen kann, ob eine vorgegebene Formel in einem Modell (passenden Formats) gilt. Im einen Fall handelt es sich um Boolesche Formeln und Bitvektor-Modelle, im anderen um Formeln der Büchi-Elgot-Trakhtenbrot-Logik und Bitvektorfolgen-Modelle. Die Grundidee ist in beiden Fällen dieselbe: Man stellt den Auswertungsprozess für eine Formel als sequentielles Verfahren dar, welches ein Modell schrittweise verarbeitet: Boolesche Modelle (Boolsche Vektoren) werden Bit für Bit und Bitvektorfolgen werden Vektor für Vektor abgearbeitet.

3.1 Binäre Entscheidungsdiagramme

Binäre Entscheidungsdiagramme sind eine Weiterentwicklung der Idee des Entscheidungsbaumes. Betrachten wir eine Boolesche Funktion f in den Variablen $x_1, \ldots, x_n$, so entsprechen die möglichen Argumente $(b_1, \ldots, b_n) \in \{0,1\}^n$ den 2^n möglichen Pfaden durch die ersten n Stufen des vollständigen binären Baumes. Der Vektor $(b_1, \ldots, b_n)$ bestimmt Bit für Bit den Verlauf des Pfades: Beim i-ten Knoten (von der Wurzel an gerechnet) biegt man nach links im Fall $b_i = 0$ über eine sog. 0-Kante, nach rechts im Fall $b_i = 1$ über eine sog. 1-Kante ab. Zur besseren Orientierung beschriftet man die Knoten auf Level i des Baumes mit x_i (beginnend mit x_1 an der Wurzel). Der letzte (d.h. $(n+1)$-te) Knoten des Pfades, der nach Abarbeitung des Vektors $(b_1, \ldots, b_n)$ erreicht wird, heißt Endknoten; man markiert ihn mit 1, falls $f(b_1, \ldots, b_n) = 1$, sonst mit 0. Ist f durch eine Formel $\varphi(x_1, \ldots, x_n)$ gegeben, hat man bei einem Endknoten $(b_1, \ldots, b_n)$ die Formel auszuwerten und als Marke den sich ergebenden Wahrheitswert einzutragen. Alle Information, die die Formel über Modelle $(b_1, \ldots, b_n)$ enthält, ist offenbar auch im zugehörigen Entscheidungsbaum kodiert; insofern sind Formel und Entscheidungsbaum äquivalent.

Zwei Reduktionsprozesse erlauben nun die Verkleinerung dieser Entscheidungsstruktur. Dabei wird die Baumgestalt i.a. aufgelöst, und ein azyklischer

Graph entsteht:

1. *Verschmelzung:* Haben zwei Knoten k, k' mit Beschriftung x_i isomorphe "Untergraphen" (bestehend aus den Knoten, die von k bzw. k' aus erreichbar sind), kann man diese Untergraphen identifizieren und k, k' zusammenziehen.

2. *Kantenkontraktion:* Führen von einem mit x_i beschrifteten Knoten k sowohl die ausgehende 0-Kante als auch die ausgehende 1-Kante zum gleichen Knoten k' (etwa mit x_j für $j > i$ beschriftet), so zieht man diese Kanten zusammen, streicht k und setzt k' mit der Beschriftung x_j unmittelbar an dessen Stelle.

In einem ersten Reduktionsschritt stellt man fest, dass alle Endknoten mit Markierung 0 trivialerweise die gleichen Untergraphen haben (sie bestehen ja nur aus eben diesem Knoten), analog für die Endknoten 1. Man kann also die 2^n Endknoten durch nur zwei Knoten ersetzen, je einen für die Marken 0 und 1. Tritt überhaupt nur einer der Werte $0, 1$ auf, so genügt natürlich nur ein (entsprechend markierter) Knoten. Nun fährt man mit weiteren Reduktionsschritten so lange wie möglich fort. Man nennt einen Graphen *reduziert*, wenn keine der beiden Regeln mehr anwendbar ist und spricht dann von einem "OBDD" ("ordered binary decision diagram"). Da bei den Reduktionsschritten das Auswertungsergebnis für die gegebene Funktion f bzw. die Formel φ erhalten bleibt, ergibt sich eine (zumeist) kompaktere Darstellung als durch den Entscheidungsbaum. Bryant [Bry92] hat nun folgende bemerkenswerte Eigenschaften der OBDD's gezeigt:

1. Ein OBDD ist (bei fester Ordnung der Variablen $x_1, \ldots, x_n$) durch die zugrundeliegende n-stellige Funktion f *eindeutig bestimmt*; es kommt also nicht auf die spezifische Reihenfolge der Reduktionsschritte bei der Herstellung an.

2. Die Äquivalenz Boolescher Formeln ist somit bestimmbar durch den (effizient durchführbaren) Test auf Isomorphie der induzierten OBDD's.

3. Die Erfüllbarkeit einer Booleschen Formel ist bestimmbar durch den (effizient durchführbaren) Test auf Existenz eines mit 1 markierten Knoten im zugehörigen OBDD.

4. Die Standardoperationen für Boolesche Funktionen, nämlich Konjunktion, Disjunktion und Negation, sind als Operationen auf den zugehörigen OBDD's effizient durchführbar (sogar so, dass sie unmittelbar wieder OBDD's liefern).

Diese einfachen, aber lange unentdeckt gebliebenen Sachverhalte sind der Schlüssel für die effiziente Lösung vieler algorithmischer Fragen zur Booleschen Logik. Sie besagen nämlich, dass man für die Erfüllbarkeit und die Äquivalenz Boolescher Formeln und ebenso für grundlegende Operationen effiziente Verfahren in der Hand hat, *wenn man erst einmal die Ebene der OBDD's erreicht hat.*

Die Grundprobleme der Komplexitätstheorie (insbesondere zum Zeitaufwand für den Erfüllbarkeitstest bei Booleschen Formeln) sind hierdurch natürlich nicht aus der Welt geschafft. Es gibt Beispiele Boolescher Funktionen (oder Formeln), bei denen der Reduktionsprozess nicht entscheidend greift, also ein Diagramm exponentieller Grösse (wie es der Entscheidungsbaum ist) bestehen bleibt. Häufig ergeben sich dann durch eine Umordnung der Variablen $x_1, \dots, x_n$ wesentlich kleinere OBDD's. Diese Effekte sind noch nicht völlig verstanden, und Heuristiken zum Auffinden geeigneter Umordnungen der Variablen Boolescher Funktionen sind ein Gegenstand aktueller Forschung.

3.2 Endliche Automaten

Die Idee einer sequentiellen Auswertung kann auch bei den Formeln der Büchi-Elgot-Trakhtenbrot-Logik angewandt werden. Diese Auswertung muss für eine Formel $\varphi(x_1, \dots, x_n)$ folgendes leisten: Wird ein Modell $(\overline{b}(0), \dots, \overline{b}(t_{\max}))$ Vektor für Vektor eingegeben, muss schließlich als Antwort geliefert werden, ob φ in diesem Modell gilt oder nicht.

Die fundamentale Einsicht von Büchi, Elgot und Trakhtenbrot in ihren Arbeiten [Büc60], [Elg61] bzw. [Tra62] besteht darin, dass man diese sequentielle Auswertung endlichen Automaten übertragen kann. Die Auswertung benötigt also neben den Eingabesymbolen (den Bitvektoren) nur einen endlichen Speicher beschränkter Größe. Diese Eigenschaft der Büchi-Elgot-Trakhtenbrot-Logik ist nicht offensichtlich und erfordert den Rückgriff auf interessante automatentheoretische Konstruktionen. Wir deuten die wichtigsten Ideen an, zunächst durch eine Vorstellung der endlichen Automaten in Analogie zu den OBDD's.

Ein endlicher Automat wird hier im Sinne eines endlichen deterministischen Akzeptors verstanden: Er hat eine endliche Menge Q interner Zustände mit ausgezeichnetem Anfangszustand q_0. Eine Folge von Eingabesymbolen $a_0, \dots, a_l$ (mit Buchstaben a_i aus einem Eingabealphabet A, in unseren Beispielen also mit Bitvektoren a_i aus $\{0,1\}^n$) bringt den Automaten sukzessiv in neue Zustände, die wir mit $q_1, \dots, q_{l+1}$ bezeichnen wollen; hierbei ergibt sich q_{i+1} jeweils aus dem vorangehenden Zustand q_i und der Eingabe a_i durch Anwendung der Automaten-Transitionsfunktion (von $Q \times A$ nach Q).

Endliche Automaten können damit als eine verallgemeinerte Art von Entscheidungsdiagrammen verstanden werden: An die Stelle der mit Variablen beschrifteten Knoten treten die Zustände aus Q, und die Verzweigungen über die 0-Kanten und die 1-Kanten werden hier zu Übergängen auf neue Zustände, jetzt unterschieden nach den möglichen Eingaben aus A. Das Auswertungsergebnis wird nicht durch gesonderte mit 0 bzw. 1 markierte Knoten, sondern durch die Unterscheidung der Zustände nach "akzeptierenden" und "nicht-akzeptierenden" Zuständen produziert. Wenn ein Automat $\mathcal{A}$ durch eine Eingabefolge von seinem Anfangszustand in einen seiner akzeptierenden Zustände geführt wird, sagt man, er *akzeptiere* die Eingabefolge. Die Menge aller akzeptierten Eingabefolgen ist die *durch $\mathcal{A}$ erkannte Sprache*. Der wesentliche Unterschied zu Entscheidungsdiagrammen liegt darin, dass die Eingaben nicht mehr (vom ersten bis zum n-ten Bit) fest numeriert sind: Während diese Numerierung in den Entscheidungsdiagrammen zur Beschriftung der Knoten mit den Variablen x_i führt, muss aufgrund der variablen Länge der Eingabefolgen in einem Automaten ein- und derselbe Zustand über verschiedene, i.a. über unendlich viele Eingabefolgen vom Anfangszustand aus erreichbar sein: der Transitionsgraph ist also nicht mehr azyklisch.

Trotz dieser Verallgemeinerung der Transitionsstruktur lassen sich fast alle Sachverhalte, die oben für die binären Entscheidungsdiagramme genannt wurden, in angemessener Neuformulierung retten. Eine Ausnahme bildet nur die Regel der Kantenkontraktion beim Reduktionsprozess – diese Sonderrolle aber ist klar, da die Kantenkontraktionsregel auf die in binären Entscheidungsdiagrammen feste Indizierung der Eingabesymbole zurückgreift, welche für endliche Automaten nicht mehr zur Verfügung steht. Die andere Reduktionsregel ist auf endliche Automaten übertragbar: Zwei Knoten eines binären Entscheidungsdiagramms waren ja zu verschmelzen, wenn für sie "die Zukunft gleich" ist (d.h. wenn die entsprechenden Untergraphen isomorph sind). In Automaten definiert man für zwei Zustände die Zukunft als "gleich", wenn für jede beliebige Eingabefolge jeweils von beiden Zuständen aus akzeptierende Zustände bzw. von beiden Zuständen aus nicht-akzeptierende Zustände erreicht werden. Der Bezug auf beliebige Eingabefolgen macht es ein wenig aufwendig, dies für Zustandspaare algorithmisch zu entscheiden; hier muss die zyklische Struktur des Automaten benutzt werden. Doch ist auch diese Art von Reduktion effizient durchführbar: Aus einem Automaten mit m Zuständen kann man einen entsprechenden *reduzierten Automaten* (in dem keine Verschmelzung mehr möglich ist) mit einem Zeitaufwand von nur $O(m \cdot \log(m))$ Schritten berechnen (vgl. etwa [AHU74, Sect. 4.13]).

Diese reduzierten Automaten haben die kleinste Zustandszahl unter allen Automaten, welche dieselbe Sprache erkennen. Darüberhinaus gelten alle Analoga der oben formulierten Ergebnisse über OBDD's:

1. Ein reduzierter Automat ist durch die erkannte Sprache bis auf Isomorphie *eindeutig bestimmt*; es kommt also nicht auf die spezifische Reihenfolge der Reduktionsschritte bei der Herstellung an.

2. Die Gleichheit der durch zwei Automaten erkannten Sprachen ist somit einfach testbar durch den (effizient durchführbaren) Test auf Isomorphie der entsprechenden reduzierten Automaten.

3. Das "Erfüllbarkeitsproblem" für Automaten, d.h. die Frage, ob es eine akzeptierte Eingabefolge gibt, ist entscheidbar durch den (effizient durchführbaren) Test, ob es im Transitionsgraphen einen Pfad vom Anfangszustand in einen akzeptierenden Zustand gibt.

4. Die Booleschen Standardoperationen für Sprachen (Mengen von Eingabefolgen), nämlich Durchschnitt, Vereinigung und Komplementbildung, sind als Operationen auf den zugehörigen reduzierten Automaten effizient durchführbar.

Die Punkte 1. und 2. sind Standardresultate der Automatentheorie (für eine integrierte Darstellung vergleiche etwa [AHU74, Sect. 4.8]). Für die Behauptung 3. sei angemerkt, dass man die Existenz eines Weges vom Anfangszustand in einen akzeptierenden Zustand etwa mit dem Verfahren der Breitensuche durch den Transitionsgraphen (in Linearzeit) entscheiden kann. Auf Punkt 4. gehen wir im nachfolgenden Abschnitt im Zusammenhang mit Büchi-Elgot-Trakhtenbrot-Logik näher ein.

3.3 Schaltkreise

Sowohl die Bitvektor-Modelle der Booleschen Logik als auch die Bitvektorfolgen-Modelle der Büchi-Elgot-Trakhtenbrot-Logik erlauben auch Auswertungen mit (teilweise) parallelen Berechnungen. Hierzu benutzt man die *Booleschen Schaltkreise* (üblicherweise aufgebaut aus "nicht"-, "und"- sowie "oder"-Gattern), auf die wir hier nur ganz am Rande eingehen. Die Verbindungen zwischen den Gattern sorgen für eine teils sequentiell, teils parallel erfolgende Transformation der Eingabebits. Auch wird durch die Möglichkeit, das Ergebnisbit eines Gatters an mehrere weitere Gatter zu leiten, erreicht, dass Mehrfachauswertungen der gleichen Teilformel vermieden werden können. Dennoch sind die Booleschen Schaltkreise ein den Formeln noch sehr ähnlicher Formalismus, bei dem allerdings die streng induktive Formelkonstruktion aufgebrochen ist. Ähnlich wie bei den Formeln greifen algorithmische Verfahren aber nicht so gut wie bei den sequentiellen Auswertungsmodellen der Entscheidungsdiagramme und Automaten: So verbirgt sich zum Beispiel in der Ermittlung "bestmöglicher" Schaltkreise (etwa im Minimierungsverfahren von Quine-McCluskey) ein NP-vollständiges Problem, und auch die Eindeutigkeit solcher minimaler Lösungen ist nicht mehr gegeben.

Auch für die Bitvektorfolgen-Modelle ist eine Schaltkreistheorie entwickelt worden. Die Modelle variabler Länge werden dort durch *Schaltkreisfamilien* definiert, wobei für eine gegebene Länge des Inputs jeweils ein eigener Schaltkreis zuständig ist. Für die durch Büchi-Elgot-Trakhtenbrot-Formeln (oder endliche Automaten) definierten Mengen genügen spezielle Familien von Schaltkreisen, bei denen die Anzahl der Gatter (jeweils mit nur ein oder zwei Eingängen) polynomial und die Schaltkreistiefe logarithmisch in der Länge des Inputs wächst. Für eine detaillierte Übersicht über die einschlägigen Ergebnisse der Schaltkreis-Komplexitätstheorie im Kontext der Automatentheorie sei auf die Monographie von Straubing [Str94] verwiesen.

4 Der Äquivalenzsatz von Büchi-Elgot-Trakhtenbrot

Man nennt eine Formel $\varphi(x_1, \ldots, x_n)$ der Büchi-Elgot-Trakhtenbrot-Logik äquivalent zu einem endlichen Automaten $\mathcal{A}$ über dem Eingabealphabet $\{0,1\}^n$, wenn die Modelle, in denen φ gilt, mit den Bitvektor-Folgen übereinstimmen, die $\mathcal{A}$ akzeptiert. Mit diesem Begriff können wir nun den fundamentalen *Satz von Büchi, Elgot und Trakhtenbrot* formulieren:

Zu jeder Formel $\varphi(x_1, \ldots, x_n)$ der Büchi-Elgot-Trakhtenbrot-Logik kann man einen äquivalenten endlichen Automaten $\mathcal{A}$ über dem Eingabealphabet $\{0,1\}^n$ konstruieren und umgekehrt auch zu jedem endlichen Automaten eine äquivalente Formel.

Zunächst betrachten wir die erste Teilbehauptung. Sie wird induktiv über den Aufbau der Formeln der Büchi-Elgot-Trakhtenbrot-Logik bewiesen. Wir gehen nur auf den Induktionsschritt ein, da er die wichtigen Konstruktionen enthält.

Hier muss gezeigt werden, dass alle Junktoren und Quantoren der Büchi-Elgot-Trakhtenbrot-Logik durch Automatenkonstruktionen "abgefangen" werden können. Hinsichtlich der Booleschen Junktoren (es genügen $\wedge$, $\vee$, $\neg$) leisten das die oben in Punkt 3. genannten "Standardoperationen". Man muss (für $\wedge$ und $\vee$) nur beachten, dass zu zwei gegebenen Automaten $\mathcal{A}_1$ und $\mathcal{A}_2$ neue Automaten für den Durchschnitt und die Vereinigung der erkannten Sprachen gebildet werden können. Dies leistet der *Produktautomat* aus $\mathcal{A}_1$ und $\mathcal{A}_2$ mit jeweils angepasster Definition der akzeptierenden Zustände. (Zustandsmenge des Produktautomaten ist das kartesische Produkt der beiden gegebenen Zustandsmengen.) Die Komplementbildung, die der Anwendung der Negation entspricht, erfolgt in einem gegebenen Automaten einfach durch Vertauschung der akzeptierenden mit den nicht-akzeptierenden Zuständen.

Es bleibt die Behandlung der Quantoren. Hier ist eine neue Idee vonnöten. Während, wie oben erläutert, in der Booleschen Logik die Quantoren eliminierbar sind, tragen sie in der Büchi-Elgot-Trakhtenbrot-Logik wesentlich zu deren Ausdruckskraft bei. Man kann sich aber überlegen, dass alle erlaubten Quantoren $\exists t$, $\forall t$, $\exists x$, $\forall x$ unter Zuhilfenahme der Negation $\neg$ auf die Behandlung des Existenzquantors $\exists x$ zurückgeführt werden können.

Betrachten wir hierzu eine Formel, die mit einem Existenzquantor beginnt:

$$\varphi_1(x_1, \ldots, x_n) \quad := \quad \exists x_{n+1}\ \varphi_0(x_1, , \ldots, x_n, x_{n+1})$$

Gemäß Induktionsvoraussetzung nehmen wir an, dass für die im Kern auftretende Formel $\varphi_0(x_1, , \ldots, x_n, x_{n+1})$ bereits ein geeigneter Automat $\mathcal{A}_0$ über dem Eingabealphabet $\{0,1\}^{n+1}$ vorliegt (der also die Menge derjenigen Folgen erkennt, in denen $\varphi_0(x_1, \ldots, x_{n+1})$ gilt). Im Induktionsschritt muss nun ein solcher Automat $\mathcal{A}_1$ für $\varphi_1(x_1, \ldots, x_n)$ konstruiert werden.

An diesem Punkt benutzen wir den Begriff des *nichtdeterministischen Automaten*. Wir konzipieren $\mathcal{A}_1$ in dieser Form: $\mathcal{A}_1$ liest eine Folge $(\bar{b}(0), \ldots, \bar{b}(t_{\max}))$ aus Bitvektoren der Länge n ein und "rät" dabei nichtdeterministisch eine Bitfolge $(b_{n+1}(0), \ldots, b_{n+1}(t_{\max}))$; auf der Kombination der eingegebenen und der geratenen Bitfolge arbeitet er nun wie $\mathcal{A}_0$. Es fehlt dann nur noch der Übergang zurück zum ursprünglichen Modell des deterministischen Automaten. Dieser wird durch die auf Myhill [Myh57] und Rabin/Scott [RS60] entwickelte *Potenzmengenkonstruktion* bewerkstelligt, mit einem Automaten, dessen Zustände als Zustands*mengen* des vorliegenden nichtdeterministischen Automaten gebildet werden.

Insgesamt sind damit die wesentlichen Schritte zusammengetragen, die für die Transformation von Büchi-Elgot-Trakhtenbrot-Formeln in endliche Automaten benötigt werden.

Es ist nützlich, sich diese Transformation von Formeln in Automaten noch einmal für den Fall der Formeln

$$\exists\overline{y_1}\ \forall\overline{y_2}\ \exists\overline{y_3}\ \ldots\ \forall\overline{y_k}\ \varphi(\overline{y_1},\overline{y_2},\ldots,\overline{y_k})$$

mit k Blöcken aus Bitfolgen-Quantoren zu vergegenwärtigen, wobei wir annehmen, dass in φ keine solchen Quantoren mehr auftreten. Analog zu den Σ_k-Formeln der Booleschen Logik werden sie Σ_k-Formeln der Büchi-Elgot-Trakhtenbrot-Logik genannt. Man kann diese Formeln auch schreiben in der Form

$$\exists\overline{y_1}\neg\exists\overline{y_2}\neg\exists\overline{y_3}\ \ldots\ \neg\exists\overline{y_k}\neg\varphi(\overline{y_1},\overline{y_2},\ldots,\overline{y_k}).$$

Sie erfordern nach dem oben skizzierten Verfahren k-mal die Anwendung der Potenzmengenkonstruktion. Damit vergrößert sich die Zustandszahl (ausgehend von der Zustandszahl n des Automaten für die Formel φ) k-fach exponentiell. Hiermit meinen wir ein Wachstum der Form

$$g_k(n) \quad := \quad \left. 2^{2^{\cdots^{2^n}}} \right\} k$$

mit einem Turm der Höhe k von Zweierpotenzen. Wie Meyer und Stockmeyer gezeigt haben (vgl. [Mey75], [Sto74], [AHU74, Sect. 11.4]), ist dieses extreme Aufwandswachstum unvermeidbar. Ihre Untersuchungen zeigen, daß *jedes* Verfahren zur Transformation von Büchi-Elgot-Trakhtenbrot-Formeln in Automaten im schlechtesten Fall einen Zeitaufwand benötigt, der (in der Formellänge n gemessen) über jede der genannten Funktionen $g_k(n)$ hinausgeht. Die Situation ist also auch in Bezug auf den Rechenaufwand schwieriger als im Fall der quantifizierten Booleschen Logik: Dort ergibt sich bei der Transformation von Σ_k-Formeln in OBDD's nur ein *vermutetes* Anwachsen des Aufwands, gemäß den Stufen der "Polynomzeithierarchie".

Diese Komplexitätsbetrachtungen lassen es zweifelhaft erscheinen, ob die Übersetzung von Büchi-Elgot-Trakhtenbrot-Formeln in Automaten überhaupt im praktischen Sinne interessant ist. Wir gehen darauf im nächsten Abschnitt ein.

Abschließend sei noch die umgekehrte Richtung des Satzes von Büchi, Elgot und Trakhtenbrot betrachtet: Sie besagt, dass man aus einem vorliegenden endlichen Automaten $\mathcal{A}$ (über dem Eingabealphabet $\{0,1\}^n$) eine äquivalente Formel $\varphi(x_1,\ldots,x_n)$ gewinnen kann. Eine solche Formel lässt sich leicht angeben. Sie muss in einem Modell $(\overline{b}(0),\ldots,\overline{b}(t_{\max}))$ ausdrücken, dass für diese Eingabe eine Zustandsfolge existiert, die den Automaten $\mathcal{A}$ zum Akzeptieren führt. Hat der Automat etwa 2^m Zustände, wird die Zustandsfolge durch eine Folge von Bitvektoren $(\overline{c}(0)\ldots\overline{c}(t_{\max}))$ über $\{0,1\}^m$ dargestellt. Die Tatsache des Akzeptierens ist somit formalisierbar mit einer Formel der Gestalt

$$\varphi(x_1,\ldots,x_n) := \exists y_1\ldots\exists y_m\psi(x_1,\ldots,x_n,y_1,\ldots,y_m),$$

bei der ψ folgendes ausdrückt: Die durch $y_1,\ldots,y_m$ kodierte Zustandsfolge beginnt mit dem Anfangszustand des Automaten $\mathcal{A}$, für jeden Zeitpunkt t wird die

Transitionsfunktion von $\mathcal{A}$ respektiert (an dieser Stelle benutzt man die Nachfolgerfunktion $'$), und vom letzten Zustand aus wird mit dem letzten Eingabe-Vektor ein akzeptierender Zustand erreicht. Es stellt sich heraus, dass man in ψ nur noch einen einzigen Quantor (über Zeitpunkte t) benötigt; die Formel φ ist also eine Σ_1-Formel.

Eine interessante Konsequenz ergibt sich hieraus mit der Verknüpfung beider Teilaussagen des Satzes: Verwandelt man eine beliebige Formel, zum Beispiel eine Σ_k-Formel, zunächst in einen Automaten und dann wieder zurück in eine Formel, sieht man, dass sich eine äquivalente Σ_1-Formel ergibt, dass sich also in diesem Sinne die Quantorenstruktur stark vereinfachen lässt. Formeln der Büchi-Elgot-Trakhtenbrot-Logik lassen sich also immer auch als Σ_1-Formeln schreiben.

5 Anwendungen auf algorithmische Probleme

Die Transformation von Formeln der Büchi-Elgot-Trakhtenbrot-Logik in endliche Automaten eröffnet zunächst, wie oben ausgeführt, eine Möglichkeit, algorithmisch zu entscheiden, ob eine gegebene Formel erfüllbar ist: Man führt die Transformation in einen äquivalenten endlichen Automaten durch und testet, ob dieser eine Eingabefolge akzeptiert (d.h., ob es einen Pfad vom Anfangszustand in einen akzeptierenden Zustand gibt).

Eine Erweiterung dieses Verfahrens wird für Anwendungen in der Programm- (oder System-) Verifikation genutzt, auch unter dem Begriff "Model-Checking" bekannt. Hier stehen Programme im Vordergrund, bei denen der Kontrollfluss (und weniger der Datenfluss) die entscheidende Rolle spielt, etwa Steuerungsprogramme oder Kommunikationsprotokolle. Man repräsentiert ein System durch einen (eventuell nichtdeterministischen) endlichen Automaten $\mathcal{A}$ und verwendet als Spezifikation eine logische Formel, die Anforderungen an die Aktionen- oder Zustandsfolgen des Systems ausdrückt. Die Korrektheit des Systems ist dann gegeben, wenn alle möglichen Systemläufe die durch die Logik-Formel beschriebenen Bedingungen erfüllen. Im Kontext unserer Überlegungen kann der Korrektheitstest algorithmisch durchgeführt werden, wenn die betrachteten Systemläufe als endlich unterstellt werden und die Spezifikation durch eine Formel φ der Büchi-Elgot-Trakhtenbrot-Logik erfolgt. Der Korrektheitstest hat dann nur zu entscheiden, ob jede Eingabefolge, die der Automat $\mathcal{A}$ akzeptiert, auch ein Modell von φ ist. Anders ausgedrückt: Es darf keine Folge geben, die $\mathcal{A}$ akzeptiert und zugleich die Formel $\neg\varphi$ erfüllt. Dazu geht man in drei Schritten vor:

1. Stelle zur Formel $\neg\varphi$ einen äquivalenten endlichen Automaten $\mathcal{B}$ her.

2. Konstruiere den Produktautomaten $\mathcal{C}$ aus dem Systemautomaten $\mathcal{A}$ und dem Automaten $\mathcal{B}$ (so dass $\mathcal{C}$ den Durchschnitt der durch $\mathcal{A}$ und $\mathcal{B}$ erkannten Sprachen definiert).

3. Prüfe, ob dieser Produktautomat $\mathcal{C}$ keine Eingabefolge akzeptiert.

Geht Punkt 3. negativ aus, ist das System nicht korrekt, und eine durch $\mathcal{C}$ akzeptierte Folge (die der Test mitliefert) benennt ein "Fehlerszenarium" des Systems $\mathcal{A}$. Diese Eigenschaft des Korrektheitstests macht das Verfahren zu einem "Debugger". In dieser Funktion werden heute viele Model-Checker industriell genutzt.

Zwei Punkte in dem skizzierten Verfahren sind besonders kritisch. Der erste betrifft die Zustandszahl des Systemautomaten $\mathcal{A}$. Setzt sich das System aus mehreren nebenläufigen Komponenten zusammen, so wächst die Anzahl der globalen Zustände exponentiell in der Anzahl der Systemkomponenten, was einem kombinatorisch-algorithmischen Zugang Grenzen setzt. Zweitens wurde ja schon im vorangehenden Abschnitt erläutert, dass für komplexe Formeln die Herstellung äquivalenter Automaten einen über jede konkrete Vorstellung hinausgehenden Rechenaufwand erfordern kann. Man hat dies zum Anlass genommen, spezielle Logiken zu betrachten, bei denen dieser Effekt hinsichtlich der Übersetzung in Transitionssysteme abgemildert ist (insbesondere temporale Logiken).

Ein anderer Zugang besteht darin, mit der Standard-Übersetzungsprozedur Erfahrungen zu sammeln, um ihr Verhalten "in realistischen Beispielen" zu erkennen. Einen Durchbruch erzielten Basin und Klarlund mit dem durch Klarlund und Mitarbeiter an der Universität Aarhus entwickelten Tool MONA ([BK95], [Kla98]). In MONA ist eine Fülle von Optimierungen für die algorithmische Manipulation von Automaten implementiert, und es wird die Tatsache ausgenutzt, dass die endlichen Automaten, die im Übersetzungsprozess als Zwischenergebnisse entstehen, effizient reduzierbar sind (vgl. Abschnitt 3.2). Vor allem diese Automatenreduktionen, die wenn immer möglich angewandt werden, führen zu erstaunlichen Erfolgen: Es können Formeln in Automaten übersetzt werden, die als Text mehrere Seiten in Anspruch nehmen und im Rahmen der Komplexitätsanalyse im "worst case" als unerreichbar gelten müssten. Ein tieferes Verständnis für diese positiven experimentellen Befunde muss noch entwickelt werden; vielleicht ist der mäßige Nichtdeterminismus in den aus praktischen Beispielen hervorgehenden Automaten ein Grund. Jedenfalls erscheinen die von Meyer und Stockmeyer konstruierten Beispiele (vgl. [Mey75, Sto74]) als nicht typisch für Anwendungen in der Programmverifikation.

Ein weiterer Aspekt kommt hinzu. Die Standardformulierungen der automatentheoretischen Algorithmen benutzen eine abstrakte Darstellungsform; insbesondere werden hier die Eingabesymbole und die internen Zustände von Automaten als Elemente abstrakter Mengen A und Q gesehen. In diesem Rahmen sind die genannten Automatenkonstruktionen (z.B. Bildung des Produktautomaten, Potenzmengenkonstruktion) besonders transparent durchführbar. Lehrbücher der Automatentheorie wie [HU79] beziehen sich in der Regel auf diese abstrakte Darstellung. Dabei geht aber der Bezug zu wichtigen Merkmalen der konkreten Anwendung verloren. In Anwendungen wird nämlich ein "Systemautomat" $\mathcal{A}$ normalerweise nicht abstrakt, sondern durch Boolesche Formeln beschrieben, die seinen Zustandsraum und seine Transitionsfunktion definieren. Auch in Originalarbeiten zur Transformation von Formeln in Automaten (etwa in [Büc60]) findet sich diese "Boolesche Sicht", wie sie ja auch im vorliegenden

Artikel betont wird. An Stelle eines großen Automaten (dessen Zustandsraum zuweilen nur durch eine nichtssagende riesige Anzahl von Zuständen charakterisierbar ist) tritt in dieser Sicht ein i.a. viel kleineres Objekt, nämlich eine die Zustandsmenge und die Transitionen definierende Boolesche Formel oder, noch kleiner, das entsprechende OBDD. Diese Verankerung in der Booleschen Logik wurde auf dem Feld der Programmverifikation erst in den neunziger Jahren wiederentdeckt, vor allem im Zuge der Einführung der OBDD's. Für die Programmverifikation mit temporalen Logiken war es K. McMillan, der in seiner Dissertation [McM93] diese Technik des "Symbolic Model-Checking" auf der Basis von OBDDs entwickelte und damit für Anwendungen neue Größenordnungen (Systeme mit 2^{100} Zuständen und mehr) erschloss. Auch das Werkzeug MONA basiert auf geschickten (Datenstrukturen für) OBDD-Darstellungen von Automaten.

Die mathematischen Analysen zur Büchi-Elgot-Trakhtenbrot-Logik haben also schon nach einigen Jahrzehnten ihren praktischen Nutzen gezeitigt, viel schneller als im Falle der Booleschen Logik. Wichtiger als diese Anwendungs-Funktion ist aber wohl der begriffliche Fortschritt, der mit diesen Forschungen erzielt wurde. Er manifestiert sich insbesondere in der gelungenen Verknüpfung der Welt der Logik-Formeln mit der Welt der Transitionssysteme und Automaten.

Wir müssen hier darauf verzichten, interessante Fortentwicklungen der Theorie, die in der aktuellen Foschung vorangetrieben werden, zu diskutieren, etwa zur automatischen Synthese von Controllern oder zur Einbeziehung der kontinuierlichen Zeit in die Logik-Systeme und die Automatenmodelle. Eine direkter zugängliche Frage im Zusammenhang mit Verallgemeinerungen der Büchi-Elgot-Trakhtenbrot-Logik wird jedoch noch im nachfolgenden Abschnitt behandelt.

6 Zweidimensionale Modelle

Die Modelle der Büchi-Elgot-Trakhtenbrot-Logik sind zweidimensional in dem Sinne, dass sie als Bit-Matrizen aufgefasst werden können, und zwar mit einer festen Anzahl von Zeilen (der Länge der verwendeten Bitvektoren entsprechend) und mit einer endlichen, aber a priori nicht beschränkten Anzahl von Spalten (indiziert von 0 bis $t_{\max}$). Diese asymmetrische Situation legt die Frage nahe, ob nicht auch für die Anzahl der Zeilen die feste Schranke aufgehoben werden kann.

Eine solche Liberalisierung würde neue Anwendungen eröffnen, zum Beispiel für Systeme, deren Größe nicht fest, sondern durch einen freien Parameter n beschränkt ist. Eine Hardware-Beschreibung könnte zum Beispiel die Anzahl der relevanten Speicherzellen als solch einen Größenparameter aufweisen. Die Verifikation sollte dann uniform für alle Parameterwerte n erfolgen. Als Modelle der Büchi-Elgot-Trakhtenbrot-Logik wären dann, selbst in Bezug auf eine einzige Formel, Folgen aus Bitvektoren einer beliebigen Länge n zuzulassen.

Im Rahmen der ursprünglichen Büchi-Elgot-Trakhtenbrot-Logik sind diese Situationen eingeschränkt modellierbar, nämlich dadurch, dass man die Rolle der beiden Dimensionen (Bitvektorlänge, Zeit) vertauscht. Dann nimmt man die (in der Länge nicht fest beschränkten) Zeilenvektoren für die Darstellung der Systemzustände, bezahlt aber dadurch, dass nun für die Zeitachse nur eine feste Anzahl von Werten auftritt. Für manche Anwendungen, z.B. in der Verifikation parametrisierter Hardware, ist dieser Ansatz erfolgreich eingesetzt worden (vgl. z.B. [BK95]).

Wie steht es mit der Erweiterung der Büchi-Elgot-Trakhtenbrot-Logik auf Modelle, die endliche zweidimensionale Bitmuster sind, jedoch in *beiden* Dimensionen von nicht fest beschränkter Größe? Es handelt sich dann um Matrizen

$$p : \{0, \ldots, s_{\max}\} \times \{0, \ldots, t_{\max}\} \to \{0, 1\},$$

auch *Bilder* (*"pictures"*) genannt. Da die Zeilen nicht mehr durch einen festen Vorrat von Namen indizierbar sind, muss auch hierfür eine Bezeichnung durch Terme eingeführt werden, so wie zuvor die Zeitterme für die Elemente des Zeitbereichs $\{0, \ldots, t_{\max}\}$ benutzt wurden. Man benötigt also die Konstanten 0, $s_{\max}$, $t_{\max}$ und die Nachfolgerfunktion $'$ für jede der beiden Dimensionen einzeln. Statt der Formeln $x_i(t)$ verwendet man atomare Formeln der Form $x(s, t)$ ("das Bit an der Stelle (s, t) ist 1"). Wir verzichten hier auf eine ausführliche Definition der so entstehenden Logik, geben aber eine Beispielformel an. Diese soll die Menge der quadratischen Bilder beschreiben. Hierzu drückt man aus, dass jedes Muster aus Bitwerten 1, das im Modell die Position $(0, 0)$ umfasst und sich von beliebigen Punkten aus diagonal nach "rechts unten" fortsetzt, schließlich den Punkt $(s_{\max}, t_{\max})$ treffen muss. (Dann gilt das auch für das kleinste solche Bitmuster, d.h. gerade für die mit 1 markierte bei $(0, 0)$ beginnende Diagonale.) Die entsprechende Formel lautet folgendermaßen:

$$\forall x(x(0,0) \ \land \ \forall s \forall t(x(s,t) \to x(s',t')) \ \to \ x(s_{\max}, t_{\max}))$$

Man kann zeigen, dass diese erweiterte Logik nicht mehr algorithmisch handhabbar ist; die allgemeine Einbeziehung der zweiten Dimension markiert also einen Qualitätssprung. Insbesondere wird das Erfüllbarkeitsproblem unentscheidbar. Der intuitive Grund liegt darin, dass man durch zweidimensionale Bit-Arrays die Berechnungen darstellen kann, welche Turingmaschinen durchführen. Die beiden Dimensionen erfassen die beiden Parameter Platz und Zeit (Anzahl der benutzten Felder des Rechenbandes, Anzahl der Rechenschritte). Aus einer Turingmaschine M kann man eine Formel φ_M der erweiterten Büchi-Elgot-Trakhtenbrot-Logik herstellen, die eine terminierende M-Berechnung, zum Beispiel ausgehend von leeren Eingabeband aus, beschreibt (in Gestalt eines zweidimensionalen endlichen Bitmusters). Der Test, ob es ein solches Bitmuster gibt (also ob φ_M erfüllbar ist), beantwortet folglich die Frage nach Termination der Turingmaschine M angesetzt auf das leere Band. Ein allgemeiner Erfüllbarkeitstest für Formeln der erweiterten Logik würde somit das Halteproblem für Turingmaschinen lösen. Dieses aber ist nicht entscheidbar.

Vor kurzem sind weitere Resultate erzielt worden, die zeigen, dass über zweidimensionalen Bitmustern auch andere Merkmale der Büchi-Elgot-Trakhtenbrot-Theorie verlorengehen. So wurde am Ende von Abschnitt 4 erwähnt, dass jede Formel dieser Logik zu einer Σ_1-Formel äquivalent ist. Wie in [MST98] bewiesen wird, erlauben die Σ_k-Formeln der erweiterten Logik über zweidimensionalen Bildern für wachsendes k die Definition immer komplexerer Eigenschaften von Bildern, lassen also keine Reduktion auf Σ_1-Formeln zu. Es ist eine interessante offene Frage, für welche (praktisch nutzbaren) Fragmente der Logik über zweidimensionalen Modellen dennoch Resultate erzielt werden können, die in Analogie zur klassischen Theorie von Büchi, Elgot und Trakhtenbrot stehen.

7 Zur Rolle einer allgemeinen Theorie diskreter Systeme

Boolesche Logik und Büchi-Elgot-Trakhtenbrot-Logik sind Bestandteile einer weitergefassten allgemeinen Theorie diskreter Systeme. Wie die vorangehenden Überlegungen beispielhaft gezeigt haben, sind die (anwendungsorientierte) Logik, die Automatentheorie und die Algorithmik Kerngebiete dieser Theorie. Im weiteren Sinne gehören auch die Berechenbarkeitstheorie, die Komplexitätstheorie, die Theorie der Grammatiken, Term- und Graphersetzungssysteme, die Netztheorie sowie Teilgebiete der Theorie der Programmiersprachen dazu.

Unter den mathematisch geprägten Wissenschaftsgebieten ist dieses (vorwiegend in der theoretischen Informatik beheimatete) Theoriefeld recht neu. Ausgehend von unseren Betrachtungen zu den Logiken von Boole und Büchi, Elgot und Trakhtenbrot wollen wir versuchen, die Ziele und Besonderheiten dieser Systemtheorie etwas genauer zu umreißen, auch im Kontext der Ausgestaltung von Studienfächern.

Die mathematischen Objekte, die in dieser Theorie betrachtet werden, sind vor allem verschiedene Formen von Transitionsstrukturen, einschließlich hierarchischer und verteilter Modelle, sowie Verknüpfungen davon. Während in der klassischen Mathematik, auch in der Graphentheorie, die "strukturellen" oder "algebraischen" Eigenschaften, die auf dem Isomorphie- und Homomorphiebegriff basieren, im Vordergrund stehen, interessiert sich die diskrete Systemtheorie primär für unterschiedliche Arten von "Verhalten" dieser Strukturen und für die Zusammenhänge zwischen Strukturen und ihrem Verhalten. (Beispielsweise haben wir das Verhalten eines endlichen Automaten durch die von ihm erkannte Sprache definiert und eine Logik zur alternativen Beschreibung dieses Verhaltens betrachtet. Es gibt daneben aber auch eine Vielzahl anderer Optionen für den Verhaltensbegriff.) Allgemein besteht das Ziel darin, logische, algebraische und algorithmische Verfahren zur Beschreibung und Analyse des Verhaltens von Transitionsstrukturen zu entwickeln und diese Verfahren für Anwendungen in der Systemkonstruktion auszugestalten. (Als Beispiel aus der vorliegenden Arbeit sei an Abschnitt 5 erinnert, wo die Fortentwicklung eines Erfüllbarkeitstests zu einem Model-Checking-Verfahren skizziert wurde.)

Die fundamentalen Begriffe und Resultate der Theorie diskreter Systeme bilden für wichtige Bereiche der heutigen Technik, insbesondere für die Systeme der Informationstechnik, eine mathematische Grundlage, die hier in ihrer Bedeutung vergleichbar ist mit derjenigen der klassischen Gebiete der Analysis und Algebra. Die meisten Systeme, die informatische oder informationstechnische Komponenten haben, enthalten heute in ihrem gedanklichen Aufbau hochkomplexe Verknüpfungen von Transitionsstrukturen. Bei der Systementwicklung wird mit diesen Strukturen und ihren Spezifikationen "gerechnet" wie in der Analysis mit Reihenentwicklungen oder in der Algebra mit Gleichungssystemen. So ist es nur natürlich, dass in vielen technischen Disziplinen (auch über den Bereich der Informatik hinaus) Fachleute gesucht sind, die – jenseits der Vertrautheit mit gängigen Programmiersprachen – mit Aufbau und Funktion diskreter Systeme argumentativ umgehen können und deren Gesetzmäßigkeiten kennen, sei es nun auf der Ebene der Bits, der Datenstrukturen, oder des globalen Systementwurfs.

Trotz seiner übergreifenden Bedeutung erscheint dieses Theoriefeld jedoch nur in einem einzigen Studiengang, nämlich dem der Informatik, als ernstzunehmender Anteil im Curriculum (und zwar hauptsächlich im Teilgebiet "Theoretische Informatik"), und vielleicht ist diese bisher konkurrenzlose Prägung ihrer Grundlagen – neben manchen anderen Aspekten natürlich – sogar ein tieferer Grund für den Erfolg der Informatik als Studienfach.

Zwar hat man auch in den der Informatik benachbarten Ingenieurfächern an vielem, was aus der Informatik kommt, Interesse (wie zum Beispiel Informatik-Bezeichnungen für eigene Studiengänge belegen), doch gilt als mathematisch geprägtes Fundament der Ingenieurausbildung nach wie vor allein der Kanon "Höhere Mathematik I - IV". Die abstrakten Strukturen der allgemeinen Theorie diskreter Systeme, auf denen die informatischen Konstruktionen beruhen, werden dort aber nicht erfasst. Anknüpfend an die einleitenden Bemerkungen dieses Artikels kann man vielleicht etwas verkürzt sagen: Für die heutigen Systementwürfe der Informationstechnik reicht die wohlbekannte Boolesche Logik als "Logik" eben nicht mehr aus.

In den fachlichen Gedankenaustausch, der zwischen der Informatik und ihren Nachbarfächern ohnehin besteht, sollten auch diese Fragen aufgenommen werden. Besonders die theoretische Informatik selbst, vielleicht allzu emsig beschäftigt mit der Ausgestaltung vieler neuer Theorien, muss sich mehr als bisher um dieses Gespräch kümmern. Es bedarf noch intensiver Anstrengungen, um die angesprochene Systemtheorie in eine wirklich geschlossene, für die Anwender direkter zugängliche Form zu bringen. (Die vorangehenden Abschnitte dieser Arbeit sollen ein ganz kleiner Beitrag zu diesen Bemühungen sein.)

Die hier angedeuteten Einschätzungen zur Funktion der Logik, Automatentheorie und Algorithmik sind nicht neu. Bereits 1962, als es die "Informatik" noch gar nicht gab, führte J.R. Büchi in einem eingeladenen Vortrag vor der Gesellschaft für Angewandte Mathematik und Mechanik ([Büc62]) folgendes aus:

> Die Automatentheorie [...] ist ein Zweig (und zwar ein ganz junger und bescheidener) einer hochentwickelten mathematischen Disziplin, die als Ganzes für den Computerspezialisten sehr wohl sein könnte, was für den Physiker die Theorie der Differentialgleichungen und was für den Statistiker die Maßtheorie ist. Gemeint natürlich ist die formale Logik; allerdings nicht mit Akzent auf Grundlagenforschung, sondern als eine Art von Mathematik gewisser endlicher Strukturen. Der angehende Computermann sollte (vielleicht sogar auf Kosten

der Analysis) lernen, was der Logiker über exakte Sprachen weiß und wie man mit solchen umgeht. Vor allem sollte er sich ein tieferes Verständnis aneignen des Begriffes der Berechenbarkeit [...].

8 Danksagung

Der Autor dankt Professor B.A. Trakhtenbrot für aufschlussreiche Gespräche während der Tagung MFCS'98 in Brünn über Fragen im Umkreis dieser Arbeit.

Literatur

[AHU74] A.V. Aho, J.E. Hopcroft, J.D. Ullman, *The Design and Analysis of Computer Algorithms*, Addison-Wesley, Reading, Mass. 1974.

[BK95] D. Basin, N. Klarlund, Hardware-verification using monadic second-order logic, in: Computer Aided Verification (P. Wolper, Ed.), Lecture Notes in Computer Science **939**, Springer-Verlag, Berlin 1995, pp. 31-41.

[Bo1854] G. Boole, *An Investigation of the Laws of Thought*, Walton and Marberly, London 1854.

[Bry92] R. Bryant, Symbolic Boolean manipulation with ordered binary decision diagrams, *ACM Computing Surveys* **24** (1992), 293-318.

[Büc60] J.R. Büchi, Weak second-order arithmetic and finite automata, *Z. Math. Logik Grundl. Math.* **6** (1960), 66-92.

[Büc62] J.R. Büchi, Mathematische Theorie des Verhaltens endlicher Automaten, *Zeitschrift für Angew. Math. und Mechanik* **42** (1962), T9-16.

[BüEl58] J.R. Büchi, C.C. Elgot, Decision problems of weak second order arithmetics and finite automata, Part I, *Notices Amer. Math. Soc.* **5** (1958), 834.

[Elg61] C.C. Elgot, Decision problems of finite automata design and related arithmetics, *Trans. Amer. Math. Soc.* **98** (1961), 21-51.

[HU79] J.E. Hopcroft, J.D. Ullman, *Introduction to Automata Theory, Languages, and Computation*, Addison-Wesley, Reading, Mass. 1979.

[Kla98] N. Klarlund, Mona & Fido: The logic-automaton connection in practice, in: *Computer Science Logic* (M. Nielsen, W. Thomas, Eds.), Lecture Notes in Computer Science **1414**, Springer-Verlag, Berlin 1998, pp. 311-326.

[McM93] K. L. McMillan, *Symbolic Model Checking*, Kluwer Acad. Publ., Boston 1993.

[Mey75] A.R. Meyer, Weak monadic second-order theory of successor is not elementary recursive, in: *Logic Colloquium, Proc. Symp. on Logic, Boston, 1972* (R. Parikh, Ed.), Lecture Notes in Computer Science **453**, Springer-Verlag, Berlin 1975, pp. 132-154.

[MST98] O. Matz, N. Schweikardt, W. Thomas, The monadic quantifier alternation hierarchy over grids and graphs, *Information and Computation* (erscheint).

[Myh57] J. Myhill, Finite automata and representation of events, WADC Rep. TR 57-624, Fundamental Concepts in the Theory of Systems, 1957, pp. 112-137.

[RS60] M.O. Rabin, D. Scott, Finite automata and their decision problems, *IBM J. Research and Development* **3** (1959), 114-125.

[Sto74] L.H. Stockmeyer, *The Complexity of Decision Problems in Automata Theory and Logic*, PhD Thesis, Dep. of Electr. Eng., MIT, Cambridge, Mass. 1974.

[Str94] H. Straubing, *Finite Automata, Formal Logic, and Circuit Complexity*, Birkhäuser, Boston 1994.

[Tho90] W. Thomas, Automata on infinite objects, in: *Handbook of Theoretical Computer Science*, Vol. b (J.v. Leeuwen, Ed.), Elsevier Sci. Publ., Amsterdam 1990, pp. 135-191.

[Tra58] B.A. Trakhtenbrot, The synthesis of logical nets whose operators are described in terms of one-place predicate calculus, *Dokl. Akad. Nauk SSSR* **118** (1958), 646-649 (in Russisch).

[Tra62] B.A. Trakhtenbrot, Finite automata and the logic of one-place predicates, *Sib. Math. J.* **3** (1962), 103-131 in Russisch). Engl. Transl.: *AMS Transl.* **59** (1966), 23-55.