

Monadische Logik zweiter Stufe

Sven Linker

24. September 2007

Ausarbeitung für das Seminar „Beyond First-Order Logic“
bei Dr. A. Schäfer

1 Einleitung

Die Komplexität moderner Software- und Hardwaresysteme stellt an Systemdesigner die Aufgabe, die Anwesenheit gewünschter und die Abwesenheit ungewünschter Eigenschaften eines Systems nachzuweisen. Dazu werden immer häufiger formale Methoden wie zum Beispiel *model checking* eingesetzt. Bei diesem Verfahren wird das System durch Automaten dargestellt und deren Abläufe bzw. die bei einem Ablauf akzeptierten Eingaben (sogenannte *Worte*) analysiert. Die Implementierung einer Spezifikation mit Automaten ist jedoch im Allgemeinen nicht einfach und fehleranfällig. Um die Beschreibungen zu verkürzen und für Menschen lesbarer zu halten, werden Systeme häufig in einer Logik definiert, die dann in äquivalente Automaten übersetzt und geprüft werden können. Ein hierfür häufig benutztes Werkzeug ist MONA [HJJ⁺95], das Logiken zweiter Stufe als Eingabe nutzt.

In der vorliegenden Ausarbeitung werden *endliche Automaten*, die endliche oder unendlich lange Worte akzeptieren, sowie die *monadische Logik zweiter Stufe über Wörtern* (MSO) vorgestellt. Anschließend wird bewiesen, dass die Klasse der Sprachen, die durch solche Automaten akzeptiert werden, gleich der Klasse der durch monadische Logik zweiter Stufe definierten Sprachen ist. Innerhalb des Beweises wird eine effektive Übersetzung von Automaten nach MSO und umgekehrt genutzt.

Durch diese Ergebnisse ist es möglich, durch MSO beschriebene Systeme und gewünschte Eigenschaften in Automaten \mathcal{S} bzw. \mathcal{E} zu übersetzen. Für diese Automaten lässt sich überprüfen, ob alle von \mathcal{E} akzeptierten Worte auch von \mathcal{S} akzeptiert werden.

2 Monadische Logik zweiter Stufe (MSO)

2.1 Prädikatenlogik erster Stufe über Wörter

Prädikatenlogik erster Stufe kann genutzt werden, um Aussagen über Wörter der Form $w = a_0 \dots a_{n-1}$ einer Sprache $\mathcal{L} \subseteq A^*$ zu treffen. Hierzu werden Variablen erster Stufe x, y, \dots mit natürlichen Zahlen kleiner n belegt, die einer Position innerhalb des Wortes

entsprechen. Weiterhin werden die Symbole $=$, $<$ und S , sowie $Q_a(x)$ mit $a \in A$ eingeführt. Das heißt, die Syntax solcher Formeln ist wie folgt definiert:

$$\varphi ::= x = y \mid S(x, y) \mid x < y \mid Q_a(x) \mid \neg\varphi' \mid \varphi' \wedge \varphi'' \mid \exists\varphi'$$

Hierbei bezeichnen x und y Variablen erster Stufe, φ' und φ'' Formeln und a einen Buchstaben aus A . Die weiteren booleschen Verknüpfungen können wie üblich als Abkürzungen aufgefasst werden.

Um eine solche Formel zu interpretieren, werden Modelle benötigt, die auf den Worten der entsprechenden Sprache aufbauen.

Definition 2.1 (Wortmodell) Sei $w = a_0 \dots a_{n-1}$ ein Wort der Sprache \mathcal{L} , dann ist

$$\underline{w} = (\text{dom}(w), S^w, <^w, (Q_a^w)_{a \in A})$$

das w entsprechende Wortmodell, wobei $\text{dom}(w) = \{0, \dots, n-1\}$ den Positionen innerhalb des Wortes, S^w der Nachfolgerrelation auf $\text{dom}(w)$, definiert durch $(i, i+1) \in S^w$ für $0 \leq i < n-1$ und $<^w$ der natürlichen Ordnung auf $\text{dom}(w)$ entspricht. Die Elemente der Familie $(Q_a^w)_{a \in A}$ stellen unäre Prädikate $Q_a^w(x)$ dar, die genau dann wahr sind, wenn sich in dem Wort w an der Stelle x der Buchstabe a befindet.

Diese Definition lässt sich leicht auf unendliche Wörter, sogenannte ω -Wörter, erweitern. ω -Wortmodelle werden mit $\underline{\omega}$ bezeichnet und unterscheiden sich von den obigen Modellen dadurch, dass die Domäne $\text{dom}(w)$ immer durch die Menge der natürlichen Zahlen \mathbb{N} gegeben ist. Eine Sprache, die aus ω -Wörtern besteht, wird als ω -Sprache bezeichnet.

Um darzustellen, dass in einer Formel φ nur die Variablen x_1, \dots, x_n frei, d.h. nicht von einem Quantor gebunden, vorkommen, wird die Notation $\varphi(x_1, \dots, x_n)$ genutzt. Eine Formel, in der keine Variable frei vorkommt, wird als *Satz* bezeichnet.

Die Semantik einer Formel $\varphi(x_1, \dots, x_n)$ über einem Modell \underline{w} wird bestimmt, indem für jedes x_i ($0 \leq i \leq n$) eine Wortposition $p_i \in \text{dom}(w)$ gewählt und die Symbole $=, <, S, Q_a$ durch Gleichheit, $<^w$, S^w und Q_a^w interpretiert werden. Die booleschen Verknüpfungen und die Quantoren werden wie üblich in der Prädikatenlogik interpretiert. Ist die Formel $\varphi(x_1, \dots, x_n)$ erfüllt, schreiben wir $(\underline{w}, p_1, \dots, p_n) \models \varphi(x_1, \dots, x_n)$. Die Sprache, die durch einen Satz φ gegeben ist, ist definiert durch $\mathcal{L}(\varphi) = \{w \in A^* \mid \underline{w} \models \varphi\}$. Wenn zu einer Sprache \mathcal{L} ein Satz φ existiert, so dass $\mathcal{L} = \mathcal{L}(\varphi)$, bezeichnen wir \mathcal{L} als *FO-definierbar*¹. Aus Gründen der Übersichtlichkeit werden noch die Abkürzungen $\text{last}(x) \equiv \neg\exists y S(x, y)$ und $\text{first}(x) \equiv \neg\exists y S(y, x)$ eingeführt.

2.2 Monadische Logik zweiter Stufe

Monadische Logik zweiter Stufe (MSO) ist eine Erweiterung oben beschriebenen Logik erster Stufe um sogenannte Variablen zweiter Stufe X_1, X_2, \dots, X_n ². Diese Variablen

¹Die Bezeichnung FO-definierbar ergibt sich aus der englischen Übersetzung nach *first-order-definable*

²Solche Variablen zweiter Stufe werden im Folgenden mit großen Buchstaben, die vorher dargestellten Variablen erster Stufe mit kleinen Buchstaben bezeichnet.

stellen nun keine einzelnen Werte, sondern Teilmengen der Domäne dar, über die auch quantifiziert werden kann. Da diese Mengen einstellige atomare Formeln $X_i(x)$ induzieren, die die Beziehung $x \in X$ ausdrücken, wird eine solche Logik *monadisch* genannt. Um die Definition der Erfüllbarkeit zu vereinfachen, wird eine modifizierte Logik mit gleicher Aussagekraft genutzt, MSO_0 . Hierbei werden alle Variablen erster Stufe eliminiert, indem sie durch Mengen mit jeweils einem Element (sogenannte *singletons*) ersetzt werden. Dadurch lassen sich die Formeln $X(x)$ durch Teilmengenbeziehungen $\{x\} \subseteq X$ ausdrücken. Die atomaren Formeln dieser Logik sind

$$X \subseteq Y, \quad Sing(X), \quad Suc(X, Y), \quad X \subseteq Q_a \quad (\text{für } a \in A),$$

die X ist Teilmenge von Y , X ist ein singleton, X und Y sind singletons $\{x\}$ bzw. $\{y\}$ mit $S(x, y)$ und an allen Positionen aus X befindet sich Buchstabe a ausdrücken.

Eine MSO_0 -Formel $\varphi(X_1, \dots, X_n)$ mit den freien Variablen X_1, \dots, X_n wird durch ein Tupel $(\underline{w}, P_1, \dots, P_n)$, d.h. ein Wortmodell und n Teilmengen $P_1, \dots, P_n \subseteq \text{dom}(w)$ interpretiert. Ein solches Tupel kann als Wort über dem Alphabet $A' = A \times \{0, 1\}^n$ aufgefasst werden, wobei ein Buchstabe (a, c_1, \dots, c_n) an der Position p bedeutet, dass in dem Wort w an dieser Stelle der Buchstabe a vorhanden ist, und dass diese Position p zur Teilmenge P_j gehört, gdw. $c_j = 1$. Für ω -Wortmodelle lässt sich diese Übersetzung analog durchführen. Ein solches Wortmodell wäre zum Beispiel

$$\begin{array}{c|cccccc} w & a & b & b & a & b & a & \dots \\ P_1 & 0 & 1 & 0 & 1 & 0 & 1 & \dots \\ P_2 & 1 & 0 & 1 & 0 & 1 & 0 & \dots \end{array}$$

Hierbei geben P_1 und P_2 die Mengen der ungeraden, bzw. der geraden Zahlen an.

3 Endliche und ω -Automaten

Ein nichtdeterministischer, endlicher Automat (NEA) wird durch ein Fünf-Tupel $\mathcal{A} = (Q, A, q_0, \Delta, F)$ beschrieben. Hierbei bezeichnet Q eine endliche Menge an Zuständen, A das Eingabealphabet, q_0 den Startzustand, $\Delta \subseteq Q \times A \times Q$ die Übergangsrelation und $F \subseteq Q$ die Menge der Endzustände. Ein NEA \mathcal{A} akzeptiert ein Wort $w = a_0 \dots a_{n-1} \in A^*$, wenn es einen erfolgreichen *Lauf* von \mathcal{A} auf w gibt, d.h. eine Sequenz von Zuständen $\rho = \rho(0) \dots \rho(n)$ mit $\rho(0) = q_0$, $(\rho(i), a_i, \rho(i+1)) \in \Delta$ für $i < n$ und $\rho(n) \in F$. Die von \mathcal{A} akzeptierte Sprache $\mathcal{L}(\mathcal{A})$ besteht aus allen von \mathcal{A} akzeptierten Wörtern.

Ein ω -Automat unterscheidet sich von dieser Definition nur durch die Akzeptanzbedingung. Da ω -Automaten unendliche Worte akzeptieren sollen, reicht eine Menge an Endzuständen, bei deren Erreichen der Automat terminieren kann, nicht aus, weshalb verschiedene Akzeptanzbedingungen entwickelt wurden, von denen im Folgenden die Büchi-Bedingung [Büc62] und die Muller-Bedingung [Mul63] beschrieben werden. Beiden Bedingungen ist gemeinsam, dass bestimmte Zustände *unendlich oft* durchlaufen werden müssen, damit der Automat ein Wort akzeptiert. Die Menge an Zuständen, die unendlich oft erreicht werden, wird mit $\text{In}(\rho)$ bezeichnet und durch $\text{In}(\rho) = \{q \in Q \mid \exists^\omega i: \rho(i) = q\}$ definiert, wobei die Aussage „es gibt unendlich viele“ durch den Quantor \exists^ω ausgedrückt wird.

Für die Büchi-Bedingung wird nun eine Menge $F \subseteq Q$ an ausgezeichneten Zuständen gewählt. Damit der Automat ein ω -Wort akzeptiert, muss für einen zugehörigen Lauf ρ des Automaten $\text{In}(\rho) \cap F \neq \emptyset$ gelten, d.h. dass mindestens ein Zustand der Menge F unendlich oft durchlaufen wird.

Die Muller-Akzeptanzbedingung benötigt hingegen eine Familie $\mathcal{F} \subseteq \mathcal{P}(Q)$ an Endzustandsmengen. Für die Akzeptanz eines ω -Wortes mit Lauf ρ muss dann $\bigvee_{F \in \mathcal{F}} \text{In}(\rho) = F$ gelten, d.h. dass die Menge aller unendlich oft durchlaufenen Zustände eine Menge der Familie \mathcal{F} ist.

Ein Automat wird dann je nach Akzeptanzbedingung *Büchi-Automat* oder *Muller-Automat* genannt, die von ihnen erkannten Sprachen *Büchi-*, bzw. *Muller-akzeptierbar*.

Lemma 3.1 *Nichtdeterministische Büchi- und Muller-Automaten (deterministische sowie nichtdeterministische) akzeptieren dieselbe Klasse an Sprachen.*

Auf einen Beweis wird hier verzichtet, er ist zum Beispiel in [Tho97] dargestellt.

4 MSO und Automaten

Wie in den vorherigen Abschnitten dargestellt, können sowohl durch MSO-Formeln, als auch durch Automaten Sprachen definiert werden. Büchi und Elgot haben festgestellt [Büc60, Elg61], dass zwischen diesen Sprachen ein starker Zusammenhang besteht.

Theorem 4.1 *Eine Sprache mit endlichen Worten ist von einem endlichen Automaten akzeptierbar gdw. sie MSO-definierbar ist. Die Übersetzung von endlichen Automaten nach MSO ist effektiv durchführbar.*

Beweis:

„ \Rightarrow “ : Sei $\mathcal{A} = (Q, A, q_0, \Delta, F)$ ein endlicher Automat, wobei ohne Beschränkung der Allgemeinheit $Q = \{0, \dots, k\}$ und $q_0 = 0$ angenommen werden kann. Um nun zu zeigen, dass $\mathcal{L}(\mathcal{A})$ MSO-definierbar ist, müssen wir einen Satz φ finden, so dass $\mathcal{L}(\mathcal{A}) = \mathcal{L}(\varphi)$.

Unser Ansatz hierfür ist folgender: Betrachten wir ein Wort $w = a_0 \dots a_{n-1}$, dann soll unser gesuchter Satz die Existenz eines erfolgreichen Laufes $p_0 \dots p_n$ behaupten. Ist $w \in \mathcal{L}(\mathcal{A})$, dann soll $\underline{w} \models \varphi$ gelten, ansonsten nicht. Die Zustände werden zunächst als paarweise disjunkte Mengen X_0, \dots, X_k aufgefasst, wobei jedes X_i die Positionen von w enthalten soll, an denen der Automat in Zustand i ist. Die Kodierung des Laufes in eine MSO-Formel muss nun verschiedene Eigenschaften erfüllen:

- Die Mengen sollen paarweise disjunkt sein, da an jeder Wortposition nur ein Zustand gelten kann: Diese Forderung kann durch die Formel $\bigwedge_{i \neq j} \forall x \neg (X_i(x) \wedge X_j(x))$ dargestellt werden.
- Den Anfang des Laufs bildet der Startzustand: $\forall x (\text{first}(x) \rightarrow X_0(x))$
- Nach Zustand q können nur durch die Übergangsrelation definierte Zustände erreicht werden: $\forall x \forall y (S(x, y) \rightarrow \bigvee_{(i,a,j) \in \Delta} (X_i(x) \wedge Q_a(x) \wedge X_j(y)))$

- Nach dem letzten gelesenen Buchstaben befindet sich der Automat im Endzustand: $\forall x(\text{last}(x) \rightarrow \bigvee_{\exists j \in F: (i,a,j) \in \Delta} (X_i(x) \wedge Q_a(x)))$

Die Konjunktion dieser Teilformeln ergibt den gewünschten Satz:

$$\begin{aligned} \underline{\omega} \models \exists X_0 \dots \exists X_k & \left(\bigwedge_{i \neq j} \forall x \neg (X_i(x) \wedge X_j(x)) \right) \\ & \wedge \forall x (\text{first}(x) \rightarrow X_0(x)) \\ & \wedge \forall x \forall y (S(x, y) \rightarrow \bigvee_{(i,a,j) \in \Delta} (X_i(x) \wedge Q_a(x) \wedge X_j(y))) \\ & \wedge \forall x (\text{last}(x) \rightarrow \bigvee_{\exists j \in F: (i,a,j) \in \Delta} (X_i(x) \wedge Q_a(x))) \end{aligned}$$

Besondere Aufmerksamkeit benötigt jedoch das leere Wort ϵ . Dieses erfüllt den oben gegebenen Satz. Akzeptiert \mathcal{A} das leere Wort jedoch nicht, muss ein weiteres Konjunktionsglied wie $\exists x(x = x)$ in den Satz eingefügt werden.

„ \Leftarrow “: Für die Rückrichtung wird die Tatsache genutzt, dass MSO_0 genauso mächtig wie MSO ist. Der Beweis wird durch Induktion geführt, indem für die atomaren Formeln von MSO_0 Automaten definiert werden, und diese dann entsprechend der betrachteten Formel $\varphi(X_1, \dots, X_n)$ verknüpft werden. Die Definition der Automaten für den Induktionsanker, also für die atomaren Formeln $X_j \subseteq X_k$, $\text{Sing}(X_j)$, $\text{Suc}(X_j, X_k)$ und $X_j \subseteq Q_a$ ist relativ leicht und wird hier nur für den Fall $\text{Sing}(X_j)$ dargestellt .

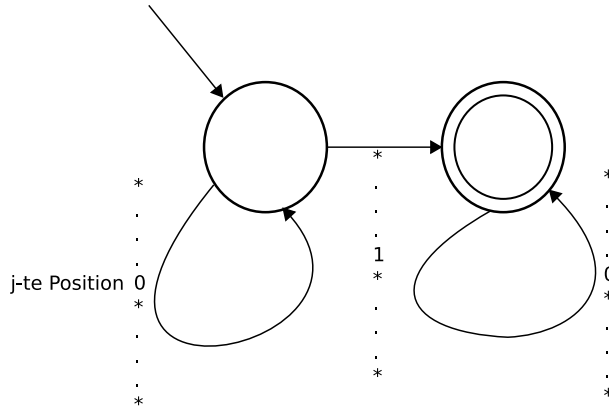


Abbildung 1: Ein endlicher Automat, der $\mathcal{L}(\text{Sing}(X_j))$ akzeptiert. Die Symbole 0 und 1 befinden sich an jter Stelle des Buchstabens.

Für den Induktionsschritt genügt es, die logisch vollständige Menge an booleschen Operatoren $\{\neg, \vee\}$ und dem Existenzquantor zu betrachten, da sich die anderen Operatoren und Quantoren mit dieser Menge ausdrücken lassen. Die Operationen auf Sprachen, die diesen Verknüpfungen entsprechen, sind Komplementierung, Vereinigung und Projektion. Reguläre Sprachen, d.h. Sprachen, die von endlichen Automaten erkannt werden können, sind gegenüber den ersten beiden Operationen abgeschlossen [Ull01]. Für die Projektion nehmen wir an, die Sprache über $A \times \{0, 1\}^n$, die durch die Formel $\psi(X_1, \dots, X_n)$ definiert wird, werde durch den Automaten \mathcal{A} akzeptiert. Dann müssen wir einen Automaten für eine Formel $\phi(X_0, \dots, X_{n-1}) = \exists X_n \psi(X_0, \dots, X_n)$ angeben können, der auf Worten über $A \times \{0, 1\}^{n-1}$ definiert ist. Der benötigte Automat \mathcal{A}'

besitzt dieselbe Zustandsmenge, denselben Startzustand und dieselben Endzustände wie \mathcal{A} . Die Transitionsrelation Δ' hingegen ergibt sich aus der Transitionsrelation Δ von \mathcal{A} nach der folgenden Vorschrift:

$$\Delta' = \{(q_1, (a, c_1, \dots, c_{n-1}), q_2) \mid \exists c_n \in \{0, 1\}: (q_1, (a, c_1, \dots, c_n), q_2) \in \Delta\}.$$

Damit kann \mathcal{A}' nichtdeterministisch die entsprechenden Transitionen von \mathcal{A} ausführen. \square

In [Büc62] hat Büchi festgestellt, dass ein solches Ergebnis auch auf unendliche Worte übertragbar ist. Hierfür war es jedoch notwendig zu beweisen, dass die Sprache der Büchi-akzeptierbaren Sprachen unter Komplementierung abgeschlossen ist. Der ursprüngliche Beweis Büchis basierte auf der Form der Sprache, es ist jedoch auch möglich, diesen Beweis durch Betrachtung deterministischer Automaten zu vereinfachen. Leider ist es nicht möglich, nichtdeterministische Büchi-Automaten in deterministische zu transformieren, da die Klasse der von deterministischen Büchi-Automaten Sprachen echt kleiner ist als die der nichtdeterministischen unendlichen Automaten. Nach Lemma 3.1 gibt es jedoch deterministische Automaten mit Muller-Akzeptanzbedingung, die dieselbe Sprache wie nichtdeterministische Büchi-Automaten akzeptieren. Die Klasse der von deterministischen Muller-Automaten akzeptierten Sprachen ist offensichtlich abgeschlossen bezüglich des Komplements, da nur die Familie \mathcal{F} durch ihr Komplement $\mathcal{P}(Q) \setminus \mathcal{F}$ ersetzt werden muss.

Damit bleibt zu zeigen, dass sich nichtdeterministische Büchi-Automaten effektiv in deterministische Muller-Automaten transformieren lassen. Dies wurde z.B. von McNaughton in [McN66] oder von Safra in [Saf88] gezeigt.

Theorem 4.2 (McNaughtons Theorem [McN66]) *Ein Büchi-Automat kann effektiv in einen äquivalenten deterministischen Muller-Automaten übersetzt werden.*

Der Beweis dieses Theorems ist recht aufwändig und wird aus Platzgründen ausgelassen.

Mit diesem Ergebnis lässt sich Büchis Theorem beweisen.

Theorem 4.3 (Büchis Theorem [Büc62]) *Eine ω -Sprache ist Büchi-akzeptierbar gdw. sie MSO-definierbar ist. Die Übersetzung von Büchi-Automaten in MSO-Formeln und umgekehrt ist effektiv.*

Beweis:

Der Beweis verläuft größtenteils analog zu 4.1. Sei ein Büchi-Automat \mathcal{A} gegeben, dann ist die entsprechende MSO-Formel wie in 4.1 zu konstruieren, nur das Konjunktionsglied, dass die Akzeptanz beschreibt, muss auf die Büchi-Bedingung angepasst werden. Eine mögliche Formulierung wäre (mit F als Menge der Endzustände):

$$\bigvee_{i \in F} \exists x (X_i(x)) \wedge \forall x (X_i(x) \rightarrow \exists y (x < y \wedge X_i(y)))$$

Die Rückrichtung ist ebenso analog zu 4.1 möglich, nur für die Konstruktion der Negation bzw. den Beweis, dass die Sprache bezüglich des Komplements abgeschlossen ist, wird die Übersetzung in Muller-Automaten benötigt. \square

5 Zusammenfassung

In der vorliegenden Ausarbeitungen wurde nach [Büc62] dargestellt, dass eine Sprache, die durch monadische Logik zweiter Stufe definiert wurde, durch einen ω -Automaten akzeptiert werden kann. Weiterhin wurde erläutert, wie sich aus einer MSO-Formel effektiv ein Automat konstruieren lässt, der genau die Sprache erkennt, deren Worte die Formel erfüllen. Damit ist es möglich, ein System und die Anforderungen an ein solches durch MSO zu beschreiben und in äquivalente Automaten zu übersetzen. Diese Automaten können dann wie gewünscht durch einen *model checker* wie z.B. [HJJ⁺95] analysiert werden.

Die hier vorgestellten Techniken sind inzwischen technisch ausgereift und gut untersucht worden. Moderne Systeme betrachten jedoch auch zeitliche Eigenschaften, die zum Beispiel durch Logiken wie den Duration Calculus (DC) [ZHR91] ausgedrückt werden können. Für eine Teilmenge des DC wurden ähnliche Äquivalenzresultate wie in der vorliegenden Arbeit gefunden, wodurch sich die Möglichkeit ergibt, DC-Formeln durch entsprechende Werkzeuge wie DCValid [Pan01] zu überprüfen.

Literatur

- [Büc60] J.R. Büchi. Weak second-order arithmetic and finite automata. *Z. Math. Logik Grundl. Math.*, 6:66–92, 1960.
- [Büc62] J. R. Büchi. On a decision method in restricted second-order arithmetic. In *Proc. of the International Congress on logic, Math, and Philosophy of Science (1960)*. Stanford University Press, 1962.
- [Elg61] C.C. Elgot. Decision problems of finite automata design and related arithmetics. *Trans. Amer. Math. Soc.*, 98:21–52, 1961.
- [HJJ⁺95] J.G. Henriksen, J. Jensen., M. Jørgensen, N. Klarlund, B. Paige, T. Rauhe, and A. Sandholm. Mona: Monadic second-order logic in practice. In *Tools and Algorithms for the Construction and Analysis of Systems, First International Workshop, TACAS '95, LNCS 1019*, 1995.
- [McN66] R. McNaughton. Testing and generating infinite sequences by a finite automaton. *Inform. Contr.*, 9:521–530, 1966.
- [Mul63] David E. Muller. Infinite sequences and finite machines. In *Proc. of the 4th Symp. on Switching Circuit Theory and Logical Design*, pages 3–16. IEEE, 1963.
- [Pan01] P.K. Pandya. Specifying and deciding quantified discrete-time duration calculus formulae using DCVALID. In *Proc. Real-Time Tools, RTTOOLS'2001*, 2001.
- [Saf88] S. Safra. On the complexity of ω -automata. In *Proc. 29th Symp. on Foundations of Computer Science*, pages 319–327. IEEE, 1988.

- [Tho97] W. Thomas. Languages, automata, and logic. In G. Rozenberg and A. Salomaa, editors, *Handbook of Formal Languages*, volume 3, chapter 7, pages 389–455. Springer Verlag, 1997.
- [Ull01] J. E. Hopcroft R. Motwani J. D. Ullman. *Introduction to automata theory, languages, and computation*. Addison-Wesley, 2. edition, 2001.
- [ZHR91] Zhou C., C.A.R. Hoare, and A.P. Ravn. A calculus of durations. *Inf. Process. Lett.*, 40(5):269–276, 1991.