Seminar Automatentheorie
Alternierende Automaten und LTL

Marc Wolter Mat.Nr. 219921

30.04.2001

1 Einleitung

Endliche Automaten auf unendlichen Wörtern sind ein nützliches Werkzeug zur Spezifikation und Verifikation von nichtterminierenden Programmen.
Die grundlegende Idee ist einleuchtend: jedem Zustand eines Programmes wird eine endliche Menge P von Aussagen (engl. „propositions"), die in diesem Zustand gelten, zugeordnet. Jede Berechnung des Programmes impliziert dann auch ein unendliches Wort über dem Alphabet 2^P. Aus allen korrekten Berechnungen ergibt sich nun eine Sprache aus unendlichen Wörtern über 2^P , die von einem Automaten erkannt werden kann. Probleme über Programmen und deren Verifikation können einfach auf automatentheoretische Probleme reduziert werden. So wird zum Beispiel das Problem der Korrektheit eines Programmes auf das Leerheitsproblem bei Automaten reduziert.
Da Berechnungen von Programmen im Allgemeinen nicht endlich sind, müssen wir uns daher mit Automaten auf unendlichen Wörtern befassen. Anders als bei Automaten auf endlichen Wörtern reicht das Erreichen eines Endzustandes als Akzeptanzbedingung nicht aus, weswegen man auf erweiterte Akzeptanzen zurückgreift, die auf dem unendlichen Vorkommen von Zuständen gründen. Die bekannteste Akzeptanz ist die Büchi-Bedingung, in Kapitel 2 wird aber Akzeptanz anhand von Färbung der Zustände vorgestellt, mit der im folgenden auch gearbeitet wird.
Weiterhin betrachten wir eine andere Art von Automaten: Alternierende Automaten. In den (üblichen) nichtdeterministischen Automaten hat man in jedem Schritt eine existentielle Wahl: man macht einen Schritt aus einer Auswahl von mehreren und nur noch der neue Zustand ist entscheidend. Es gibt jedoch auch die Möglichkeit einer universellen Wahl: man muss alle Schritte gleichzeitig machen und läßt auf jedem neuen Zustand eine Kopie des Automaten laufen.
Alternierende Automaten bieten beides, existentielle und universelle Wahlmöglichkeiten und sind somit sehr geeignet für Verifikationen, da ihre Struktur der von boolscher Logik ähnelt.

Zur Spezifikation von Anforderungen an fortlaufende Programme eignen sich temporäre Logiken, da sie eine Zeitkomponente beinhalten, die bei bestimmten Anforderungen eine Rolle spielt („fünf Sekunden nach Drücken des Knopfes soll etwas geschehen“). Kapitel 3 beschäftigt sich mit der temporären Logik LTL, die in Kapitel 4 in einen schwachen alternierenden Parity-Automaten übersetzt wird. Anhand dieses Automaten kann man nun das Erfüllbarkeitsproblem für LTL lösen, wobei sich jedoch herausstellt, das die Übersetzung von LTL in alternierende Automaten zwar einfach, jedoch das Nicht- Leerheitsproblem auf alternierenden Automaten dafür um so schwerer ist.
Zum Schluß wird noch die Komplexität der Operationen betrachtet, woran deutlich wird, wie schwer Probleme dieser Art sind, dass nämlich die Erfüllbarkeit von LTL in PSPACE liegt, man jedoch durch bestimmte Restriktionen diese Probleme stark vereinfachen kann.

2 Alternierende Automaten

Die Funktion von (nicht)deterministischen Automaten sollte wohl bekannt sein und wird hier nicht nochmal näher erläutert. Die Arbeit mit nichtdeterministischen Automaten führt aber in Kapitel 4 bei der Übersetzung einer LTL-Formel zu einem exponentiellen Aufwand, weswegen wir hier alternierende Automaten einführen, die diesen Aufwand umgehen. Die Besonderheit von alternierenden Automaten ist, daß die Übergangsfunktion keine Menge von Nachfolgezuständen angibt, sondern eine Formel, die die Nachfolgezustände erfüllen müssen.

Definition 2.1 Alternierende Automaten [LT2000] Sei X eine endliche Menge. Die Menge $B^+(X)$ der positiven boolschen Formeln über X enthält TRUE, FALSE, alle Elemente aus X sowie alle boolschen Kombinationen der Elemente aus X mit $\wedge, \vee$. Eine Teilmenge S von X ist ein Modell von $\theta \in B^+(X)$ gdw die Interpretation, die den Elementen von S TRUE und allen Elementen aus $X \setminus S$ FALSE zuweist erfüllt θ. Ein alternierender endlicher Automat (AFA) A hat die Form $A = (Q, \Sigma, q_0, \delta, AK)$, wobei

- Q ist eine endliche Menge von Zuständen
- Σ ist ein endliches Alphabet
- $q_0 \in Q$ ist der Anfangszustand
- $\delta : Q \times \Sigma \rightarrow B^+(Q)$ ist die Übergangsfunktion. Diese beschreibt die Anforderungen, die die Menge von Nachfolgezustände erfüllen muss. Näheres dazu unter „Run DAGs statt run trees“.
- AK ist die Akzeptanzkomponente

Bemerkung 1 Betrachtet man den nichtdeterministischen endlichen Automaten (NEA) $A = (Q', \Sigma, q_0, \delta', F)$, in dem δ ' einem Zustand s und einem Buchstaben a eine Menge von Zuständen zuordnet, so läßt sich diese nichtdeterministische Wahl $\delta'(s,a) = (s_1, s_2, \ldots, s_n)$ in einem alternierenden Automaten als $\delta(s,a) = s_1 \vee s_2 \vee \ldots \vee s_n$ ausdrücken. Man kann leicht erkennen, das deterministische, nichtdeterministische und universelle Automaten Spezialfälle von alternierenden Automaten sind.

Bemerkung 2 Anstatt eines Anfangszustandes q_0 kann ein alternierenden Automat durch eine Anfangsformel θ definiert sein.

Bisher wurde die Akzeptanzkomponente noch nicht genauer betrachtet. Wie in der Einleitung erwähnt benötigen Automaten auf unendlichen Wörtern erweiterte Akzeptanzen, die sich mit dem unendlich Auftreten von Zuständen beschäftigen.
Wie bei allen Automaten akzeptiert ein alternierender Automat A ein Wort $w \in \Sigma^*$ gdw es einen Lauf von A gibt, so dass jeder unendliche Pfad dieses Laufs die Akzeptanzbedingung erfüllt.
Hierzu ist zu bemerken, das FALSE keine Modelle hat und jeder Lauf, der in FALSE endet, nicht akzeptiert. Analog dazu erfüllen alle $S \subseteq X$ TRUE und ein Lauf kann in Blättern mit der Beschriftung TRUE enden (muß aber nicht zwingend akzeptieren, denn es können ja auch nichtakzeptierende Pfade existieren). Deswegen müssen nur die unendlichen Pfade die Akzeptanzbedingung erfüllen, da endliche Pfade in TRUE enden.
Die Sprache L(A) eines altern. Automaten A enthält alle Wörter $w \in \Sigma^*$, die A akzeptiert. Einen unendlichen Pfad π drückt man durch die Sequenz der Zustände aus, die dieser Pfad durchläuft. Die Menge $In(\pi)$ enthält alle Zustände, die unendlich oft in π vorkommen, die Menge $Occ(\pi)$ enthält alle Zustände, die mindestens einmal in π vorkommen.
Ein alternierender Büchi-Automat A hat die Form $A = (Q, \Sigma, q_0, \delta, F)$, wobei

- die Akzeptanzkomponente $F \subseteq Q$ die Menge der Endzustände ist und
- A akzeptiert den Pfad π gdw $F \cap In(\pi) \neq \emptyset$

Co-Büchi Automaten akzeptieren gdw die Endzustände nicht unendlich oft besucht werden.
Ein alternierender Parity-Automat A hat die Form $A = (Q, \Sigma, q_0, \delta, c)$, wobei

- die Akzeptanzkomponente $c : Q \to 0, \ldots, k$ eine Färbung der Zustände von A ist und
- A akzeptiert den Pfad π gdw $min(In(c(\pi)))$ gerade ist

Durch eine leichte Veränderung der Akzeptanzbedingung erhält man eine schwache Version der beiden obigen Akzeptanzen. Eine Akzeptanzbedingung wird schwach genannt, wenn statt der Menge $In(\pi)$ die Menge $Occ(\pi)$

verwendet wird:
Ein unendlicher Pfad π erfüllt

- eine schwache Büchi Akzeptanzbedingung gdw $F \cap Occ(\pi) \neq \emptyset$
- eine schwache co-Büchi Akzeptanzbedingung gdw $F \cap Occ(\pi) = \emptyset$
- eine schwache Parity-Akzeptanzbedingung gdw $min(Occ(c(\pi)))$ gerade ist

In [MSS86] wurden zum ersten Mal schwache alternierende Automaten eingeführt, die noch stärkere Einschränkungen unterliegen. Diese Einschränkungen sind jedoch bei Automaten auf Wörtern nicht relevant, sondern beschränken erst bei Baum-Automaten die Mächtigkeit der schwachen Automaten.

Definition 2.3 (Schwache alternierende Automaten)
Ein alternierender Automat $A = (Q, \Sigma, q_0, \delta, AK)$ heißt schwach (WAA) gdw es existiert eine Partition $Q_1, ..., Q_m$ von Q, so dass für jedes $q \in Q_i, q' \in Q_j$ mit einer Kante von q nach q' gilt: $j \geq i$ und weiterhin gilt $Q_i \subseteq AK$ oder $Q_i \cap AK = \emptyset$ für alle $i \in 1, ..., m$.

Beispiel mit schwacher Parity-Bedingung:
$A = (q_0, ..., q_3, a, b, c, q_0, \delta, c)$ wobei δ und c wie folgt:

δ	a	b	c	c(q)
q_0	q_0	q_0	$q_0 \wedge q_1$	3
q_1	$q_1 \wedge q_2$	q_1	TRUE	2
q_2	FALSE	q_3	FALSE	1
q_3	q_3	q_3	TRUE	1

Dabei wird Q wie folgt partitioniert: $Q = Q_1, Q_2, Q_3$ mit $Q_i = \{q | c(q) = i\}$. Der Automat A akzeptiert ein Wort w gdw w das Infix cvc beinhaltet, wobei $v \in a, b^*$ und in v folgt auf jedes a unmittelbar ein b.

Im weiteren werde ich mich mit schwachen alternierenden Parity-Automaten mit schwacher Akzeptanz beschäftigen. Dies entspricht der Vorgehensweise aus [LT2000] , obwohl viele Beweise an [KV2000] orientiert sind. Der Umgang mit verschiedenen alternierenden Automaten gleicht sich aber sehr, so daß Transformation von einem Automaten in einen anderen oft sehr einfach sind.
Zu bemerken sei noch, das die Definition von schwachen alternierenden Automaten oft auch auf der schwachen Akzeptanzbedingung gegründet ist, ich jedoch hier Def. 2.3. verwende und die schwache Parity-Bedingung als Zusatz sehe.

Viele Quellen (z.B. [KV2000], [MSS86] arbeiten mit schwachen Büchi - Automaten, weswegen hier noch kurz gezeigt wird, das beide Automaten gleich mächtig sind:

Satz 2.4
Schwache alternierende Büchi-Automaten und schwache alternierende Parity-Automaten sind gleich mächtig.

Beweis [LT2000] :
„$\Rightarrow$“: Gegeben ein schwacher altern. Büchi-Automat A. Dieser akzeptiert ein Wort ω gdw in jedem unendliche Pfad π existiert ein $i \in 1, ..., m$ mit $In(\pi) \subseteq Q_i \subseteq F$. Diese Bedingung kann auch durch eine Parity-Bedingung ausgedrückt werden, da sie genau dann erfüllt ist, wenn der Pfad das Q_i betritt und nie wieder verlässt. Deshalb definiert man die Färbung $c : Q \to \mathbb{N}$ wie folgt:

$$c(q) = \begin{cases} 2i & Q_i \subseteq F \\ 2i+1 & Q_i \cap F = \emptyset \end{cases}$$

Es ist leicht einzusehen, daß der durch diese Konstruktion entstandene schwache Parity-Automat äquivalent zu A ist.
„$\Leftarrow$“: Gegeben sei ein schwacher Parity-Automat $A_P = (Q, \Sigma, q_0, \delta, c)$, wobei N_e die Menge der geraden Zahlen in $\mathbb{N}$ ist. Dann konstruiert man den Büchi-Automaten $A_B = (Q', \Sigma, q_0', \delta', F')$ mit

- $Q' = Q \times C$
- $q_0' = (q_0, c(q_0))$
- $F' = Q \times N_e$
- $\delta'((q, i), a) = \phi(\delta(q, a), i)$

wobei die Hilfsfunktion $\phi : B^+(Q) \times N \to B^+(Q')$ wie folgt definiert ist: $\phi(\psi, i) = \psi[q \setminus (q, min\{i, c(q)\}])$, d.h. jedes $q \in Q$ in ψ wird durch c zusammen mit Minimum der bisher betrachteten Färbungen ersetzt. Dieser Wert kann also entlang eines Pfades nicht steigen. Nun braucht man nur noch die Mengen Q_i anhand der Färbung der zweiten Komponente zu wählen und auch hier sieht man leicht, daß der entstandene Automat A_B äquivalent zu A_P ist.

2.1 Run DAG statt run trees

Durch die besondere Übergangsfunktion von alternierenden Automaten ergeben sich auch Besonderheiten bei den möglichen Läufen („runs“) auf diesen. Die Definition eines Baumes sollte hier klar sein und wird nicht erläutert.
Da in alternierende Automaten universelle Verzweigung auftritt entspricht ein Lauf r auf einem unendlichen Wort $\omega = \omega_0\omega_1\omega_2\ldots$ keiner unendlichen

Sequenz von Zuständen sondern einem unendlichen Q-beschrifteten Baum T_ω = (T,r), wobei

- die Wurzel von T_ω ist mit q_0 beschriftet
- Sei $x \in T_\omega$ ein Knoten mit r(x)=q und $\delta(q, \omega_{|x|}) = \theta$. Dann existiert eine (möglicherweise leeres) Modell $S = q_1, q_2, \ldots, q_k$ für θ und es gilt $xc \in T_\omega$ sowie $r(xc) = q_c$ für alle $1 \leq c \leq k$, d.h. die Nachfolger von x sind mit den Elementen eines Modells von θ beschriftet.

T_ω kann auch bei unendlichen Wörtern Blätter haben, da für $\theta = TRUE$ x keine Nachfolger braucht. Ebenso existiert kein Lauf der einen Übergang mit $\theta = FALSE$ hat.
Diese Laufbäume können jedoch auch vereinfacht werden:

Definition 2.6
Zwei Knoten x_1 und x_2 eines Baumes sind ähnlich gdw $|x_1| = |x_2|$ und $r(x_1) = r(x_2)$. Ein Lauf (T,r) heißt gedächtnislos gdw für alle ähnlichen Knoten x_1 und x_2 und für alle $y \in N^*$ gilt: $x_1y \in T$ gdw $x_2y \in T$ und $r(x_1y) = r(x_2y)$. D.h. sobald man an einem ähnlichen Knoten x_1 angekommen ist ist der Pfad zu diesem Knoten für den weiteren Verlauf nicht mehr relevant, man hätte ebenso den Pfad zu x_2 wählen können, da x_1 und x_2 sich von nun an gleich verhalten.

Satz 2.7 [EJ91] Für alternierende Parity-Automaten existieren gedächtnislose akzeptierende Läufe.

Sei |Q| = n, so ist es einleuchtend, daß jede Menge von mehr als n Knoten auf der selben Tiefe mindestens zwei ähnliche Knoten enthält. Aber auch bei weniger Knoten können mehrere ähnliche Knoten auf der selben Tiefe existieren. Daraus ergibt sich, daß man einen Lauf nicht mehr als Baum betrachtet, sondern ähnliche Knoten zusammenfaßt und so einen gerichteten, azyklischen Graphen (DAG = directed acyclic graph) G = (V,E) mit den folgenden Eigenschaften erhält:

- $V \subseteq Q \times \mathbb{N}$ mit $(q_0, 0) \in V$
- $E \subseteq \bigcup l > 0 (Q \times \{l\}) \times (Q \times \{l+1\})$
- $\forall (q,l) \in V \setminus \{(q_0,0)\} \exists q' \in Q : ((q', l-1), (q,l)) \in E$.

Beispiel:
Sei A ein schwacher alternierender Automat wie im obigen Beispiel. Auf dem Suffix u=cacbabc des ω-Wortes w existieren z.B. folgende Laufbäume:

$$
\begin{array}{ccccccccccccccc}
 & c & & a & & c & & b & & a & & b & & c & \\
\hline
q_0 & \rightarrow & q_1 & \rightarrow & q_1 & \rightarrow & \mathrm{T} & & & & & & & & \\
 & & \searrow & q_2 & \rightarrow & \mathrm{F} & & & & & & & & & \\
 & & & & & & & & & & & & & & \\
q_0 & \rightarrow & q_0 & \rightarrow & q_0 & \rightarrow & q_1 & \rightarrow & q_1 & \rightarrow & q_1 & \rightarrow & q_1 & \rightarrow & \mathrm{T} \\
 & & & & & & & & & & \searrow & q_2 & \rightarrow & q_3 & \rightarrow \ \mathrm{T} \\
 & & & & & & & & & & & & & & \\
q_0 & \rightarrow & q_0 & \rightarrow & q_0 & \rightarrow & q_0 & \rightarrow & q_0 & \rightarrow & q_0 & \rightarrow & q_0 & \rightarrow & \ldots \\
\hline
\end{array}
$$

Der zweite Laufbaum akzeptiert, die anderen beiden nicht. Der erste Baum akzeptiert nicht, da er in einem FALSE-Zustand endet, der letzte Baum hat $q_0 = 3$ als kleinste vorkommende Färbung.

3 Linear Time Logic (LTL)

Wie schon erwähnt beinhalten temporäre Logiken eine Zeitkomponente. LTL ist eine einfache temporäre Logik, die stark an die boolsche Logik erinnert, zusätzlich jedoch einige Zeitoperatoren mitbringt. LTL arbeitet auf unendlichen Sequenzen α von Mengen von Bedingungen, denen zu jedem Zeitpunkt ein Wahrheitswert zugeordnet wird. Diese Zeitwerte sind natürlich abstrakt und diskret, besitzen keine Einheit, sondern entsprechen dem Zahlenstrahl der natürlichen Zahlen.

3.1 Ein kurzer Überblick [Tho2000]

Gegeben sei eine Menge $p_1, \ldots, p_n (n > 0)$ von Bedingungen („propositions"). Eine LTL-Formel $\phi(p_1, \ldots, p_n)$ ist wie folgt induktiv aufgebaut:

- jedes p_i ist eine LTL-Formel
- wenn ψ, ϕ LTL-Formeln sind, dann auch $\neg\psi, \psi\wedge\phi, \psi\vee\phi, \psi \rightarrow \phi, \psi \leftrightarrow \phi$
- wenn ψ, ϕ LTL-Formeln sind, dann auch
  - Xψ (im nächsten Schritt gilt ψ),
  - Fψ (ψ gilt irgendwann mal in der Zukunft),
  - Gψ (ψ gilt von nun an immer),
  - $\psi U \phi$ (ϕ gilt irgendwann und bis dahin gilt ψ)

Bemerkung zu Sequenzen $\alpha \in (2^{\{p_1,\ldots,p_n\}})^\omega$
$\alpha = \alpha(0)\alpha(1)\alpha(2)\ldots$ mit $\alpha(i) \in (2^{\{p_1,\ldots,p_n\}})^\omega$
$\alpha^i = \alpha(i)\alpha(i+1)\alpha(i+2)\ldots [\alpha^i(0) = \alpha(i)]$
Hierbei steht jedes i für einen Zeitpunkt, $\alpha(0)$ ist der Beginn der Zeitmessung.

Die Erfüllbarkeitsrelation $\alpha \models \phi$ ist induktiv über ϕ definiert:

1. $\alpha \models p_i :\Leftrightarrow p_i \in \alpha(0)$
2. $\neg, \vee, \wedge$ haben die übliche Bedeutung
3. $\alpha \models X\phi :\Leftrightarrow \alpha^1 \models \phi$
4. $\alpha \models F\phi :\Leftrightarrow \exists i \geq 0 : \alpha^i \models \phi$
5. $\alpha \models G\phi :\Leftrightarrow \forall i \geq 0 : \alpha^i \models \phi$
6. $\alpha \models \psi U \phi :\Leftrightarrow \exists j \geq 0 : \alpha^j \models \phi \wedge \forall 0 \leq i < j : \alpha^i \models \psi$

Im folgenden werden wir nur die Punkte 1), 2), 3) und 6) benutzen, da sie ausreichen, die anderen Punkte darzustellen:
$F\phi \equiv TRUE\ U\phi$
$G\phi \equiv \neg F \neg \phi$
Weiterhin lassen sich darstellen:
$TRUE \equiv p \vee \neg p$
$FALSE \equiv p \wedge \neg p$

Bemerkung 1
$\phi U \psi \equiv \psi \vee (\phi \wedge X(\phi U \psi))$
$\neg G\psi \equiv F \neg \psi$
$GG\psi \equiv G\psi$
$FF\psi \equiv F\psi$
$(\phi_1 \wedge \phi_2) U \psi \equiv (\phi_1 U \psi) \wedge (\phi_2 U \psi)$
$\phi U(\psi_1 \vee \psi_2)) \equiv (\phi U \psi_1) \vee (\phi U \psi_2)$

Bemerkung 2
Jede FO-Formel ist äquivalent zu einer LTL-Formel.

Beispiel einer LTL-Formel:
$\phi = XX(p_1 \to Fp_2)$
Diese Formel über zwei Bedingungen p_1, p_2 bedeutet ausgeschrieben: wenn im übernächsten Schritt die Bedingung p_1 gilt, dann gilt auch von diesem Zeitpunkt an (oder in diesem Zeitpunkt) irgendwann mindestens einmal die Bedingung p_2.
Nun betrachten wir eine Sequenz α über $\{p_1, p_2\}$, die ϕ nicht erfüllt: $\alpha = \{p_1\}\emptyset\{p_1\}\emptyset\{p_1\}\ldots$
Am einfachsten betrachtet man die Auswertung systematisch über die Unterformeln von ϕ:

α	$\{p_1\}$	$\emptyset$	$\{p_1\}$	$\emptyset$	$\{p_1\}$	$\ldots$
p_1	+	-	+	-	+	$\ldots$
p_2	-	-	-	-	-	$\ldots$
Fp_2	-	-	-	-	-	$\ldots$
$p_1 \to Fp_2$	-	+	-	+	-	$\ldots$
$X(p_1 \to Fp_2)$	+	-	+	-	+	$\ldots$
$XX(p_1 \to Fp_2)$	-	+	-	+	-	$\ldots$

(hierbei bedeutet +=die linke Seite ist erfüllt und -=die linke Seite ist nicht erfüllt) Man sieht, daß $\{p_1\} = \alpha(0) \quad \phi$ nicht erfüllt.

4 Konstruktion eines alternierenden Automaten

Man kann eine LTL-Formel ϕ in einen Büchi-Automaten übersetzen, jedoch bedeutet diese Übersetzung exponentiellen Aufwand, der bei der folgenden Übersetzung in alternierende Automaten vermieden werden kann. Der so entstehende Automat ist sogar ein linearer Automat, d.h. es treten in dem Graphen der Übergangsfunktion keine Zykel der Länge > 1 auf.

Satz 4.1
Zu einer gegebenen LTL-Formel ϕ über $P = \{p_1, \ldots, p_n\}$ kann man einen schwachen alternierenden Parity- Automaten $A_\phi = (Q, \Sigma, \phi, \delta, c)$ konstruieren, wobei $\Sigma = 2^P$ und $|Q| \in O(|\phi|)$, so dass $L(A_\phi)$ genau die Menge der Berechnungen ist, die ϕ erfüllen.

Die Menge Q der Zustände enthält alle Unterformeln von ϕ sowie deren Negation. Der Anfangszustand ist ϕ selbst. Die Partition in disjunkte $Q_1, \ldots, Q_k$ ist durch $Q_i = \{q | c(q) = i\}$ definiert.

Die Färbung c färbt die Zustände so dass:

- wenn ϕ eine echte Unterformel von ψ ist, dann gilt $c(\psi) > c(\phi)$
- Formeln der Form $\neg(\phi U \psi)$ haben eine gerade Färbung, alle anderen Zustände erhalten eine ungerade Färbung.

Es fehlt noch die Konstruktion der Übergangsfunktion δ, wozu ich erst die Notation der dualen Formel einführe:
Das duale $\overline{\phi}$ einer Formel ϕ erhält man durch Auswechseln von $\vee$ und $\wedge$ sowie TRUE und FALSE sowie durch Negierung von Unterformeln, z.B. $\overline{\neg p \wedge (q \vee Xq)}$ wird zu $p \vee (\neg q \wedge \neg Xq)$. Formal:

- $\overline{\psi} = \neg\psi$ für alle $\psi \in Q$
- $\overline{TRUE} = \text{FALSE}$

- $\overline{FALSE} = \text{TRUE}$
- $\overline{(\alpha \vee \beta)} = (\overline{\alpha} \wedge \overline{\beta})$
- $\overline{(\alpha \wedge \beta)} = (\overline{\alpha} \vee \overline{\beta})$

Diese Notation wird nun benutzt, um δ zu definieren:

1. $\delta(p, a) = TRUE$,falls $p \in a$
2. $\delta(p, a) = FALSE$,falls $p \notin a$
3. $\delta(\psi \wedge \phi, a) = \delta(\psi, a) \wedge \delta(\phi, a)$,$\vee$ analog
4. $\delta(\neg\psi, a) = \overline{\delta(\psi, a)}$
5. $\delta(X\psi, a) = \psi$
6. $\delta(\phi U \psi, a) = \delta(\psi, a) \vee (\delta(\phi, a) \wedge \phi U \psi)$

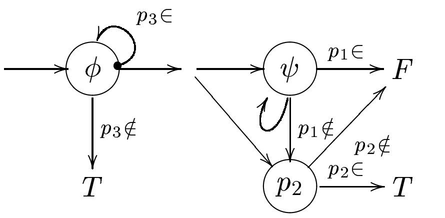
Beweis:
Der Beweis folgt eigentlich aus der Konstruktion.
A_ϕ akzeptiert eine Eingabe α, wenn für jeden unendlichen Pfad π des Laufes von α auf A_ϕ die Bedingung „min $(Occ(c(\pi)))$ ist gerade“ erfüllt ist.
Aus der Konstruktion von δ kann man zwei Fälle lesen:
In den Punkten 1)-5) erhält man als neue Formel immer Unterformeln bzw. Verknüpfungen von Unterformeln der ursprünglichen Formel. Dadurch, daß man immer weniger komplexe Formeln erhält, gelangt man irgendwann zu einer TRUE/FALSE Aussage.
Die einzige Möglichkeit zu einem unendlichen Pfad zu gelangen ist durch Punkt 6). Jeder unendliche Pfad ist also irgendwann durch $\phi U \psi$ oder $\neg\phi U \psi$ beschriftet. Nun kann aber $\phi U \psi$ nur erfüllt werden, wenn irgendwann einmal ψ gilt, aber dann ist der Pfad nicht mehr unendlich. Ein erfüllbarer, unendlicher Pfad kann also nur mit $\neg\phi U \psi$ beschriftet sein, da dies bedeutet, daß ψ nie gilt. Aus diesem Grund werden alle Zustände der Form $\neg\phi U \psi$ mit einer geraden Zahl gefärbt, da unendliche Pfade durch Zustände der Form $\phi U \psi$ nicht erfüllbar sind.

Beispiel:
$\phi = (\neg(X\neg p_2)Up_1)U\neg p_3$
Der daraus konstruierte Automat sieht folgendermaßen aus:
$A_\phi = (Q, 2^{\{p_1, p_2, p_3\}}, \phi, \delta, c)$, wobei die Zustandsmenge, die Übergangsfunktion und die Färbung anhand folgender Tabelle festgelegt sind ($\psi = \neg(X\neg p_2)Up_1$):

$q \in Q$	c	$\delta(q,\emptyset)$	$\delta(q,\{p_1\})$	$\delta(q,\{p_2\})$	$\delta(q,\{p_3\})$	$\delta(q,\{p_1,p_2\})$
ϕ	7	T	T	T	$(p_2 \vee \psi) \wedge \phi$	T
$\neg\phi$	6	F	F	F	$(\neg p_2 \wedge \neg\psi) \vee \neg\phi$	F
$(X\neg p_2)Up_1$	5	$\neg p_2 \wedge \neg\psi$	T	$\neg p_2 \wedge \neg\psi$	$\neg p_2 \wedge \neg\psi$	T
ψ	4	$p_2 \vee \psi$	F	$p_2 \vee \psi$	$p_2 \vee \psi$	F
$X\neg p_2$	3	$\neg p_2$	$\neg p_2$	$\neg p_2$	$\neg p_2$	$\neg p_2$
$\neg X\neg p_2$	3	p_2	p_2	p_2	p_2	p_2
$\neg p_3$	1	T	T	T	F	T
$\neg p_2$	1	T	T	F	T	F
$\neg p_1$	1	T	F	T	T	F
p_3	1	F	F	F	T	F
p_2	1	F	F	T	F	T
p_1	1	F	T	F	F	T

$q \in Q$	c	$\delta(q,\{p_1,p_3\})$	$\delta(q,\{p_2,p_3\})$	$\delta(q,\{p_1,p_2,p_3\})$
ϕ	7	$(p_2 \vee \psi) \wedge \phi$	$(p_2 \vee \psi) \wedge \phi$	$(p_2 \vee \psi) \wedge \phi$
$\neg\phi$	6	$(\neg p_2 \wedge \neg\psi) \vee \neg\phi$	$(\neg p_2 \wedge \neg\psi) \vee \neg\phi$	$(\neg p_2 \wedge \neg\psi) \vee \neg\phi$
$(X\neg p_2)Up_1$	5	T	$\neg p_2 \wedge \neg\psi$	T
ψ	4	F	$p_2 \vee \psi$	F
$X\neg p_2$	3	$\neg p_2$	$\neg p_2$	$\neg p_2$
$\neg X\neg p_2$	3	p_2	p_2	p_2
$\neg p_3$	1	F	F	F
$\neg p_2$	1	T	F	F
$\neg p_1$	1	F	T	F
p_3	1	T	T	T
p_2	1	F	T	T
p_1	1	T	F	T

Der Automat in graphischer Form:



Dabei ist eine vereinfachte Notation benutzt. Übergänge zu TRUE oder FALSE sind keineswegs Übergänge zu Zuständen, da weder TRUE noch FALSE Zustände von A_ϕ sind, sie sollen jedoch die Akzeptanz bzw. die Verwerfung des Automaten darstellen. Eine weitere Vereinfachung der Notation ist, daß statt jedem einzelnen $a \in \Sigma$ eine Bedingung $p_i[\in,\notin]a$ an dieses a gestellt wird. Nichtdeterministische Übergänge sind wie gewohnt dargestellt, zur Darstellung von $\wedge$ - Übergängen wird der Pfeil mit einem Punkt $\cdot$ markiert. Die Klammerung von Formeln wird hier intuitiv durch

die Verzweigung von Pfeilen dargestellt.
An der Darstellung des Automaten kann man akzeptierende Wörter leichter erkennen als an der Formel: zum Beispiel akzeptiert der Automat jedes Wort der Form $\{p_2p_3\}\{p_2p_3\}\ldots\{p_2\}$, da durch p_3 die ϕ-Schleife erfüllt wird und durch das p_2 nichtdeterministisch die ψ - Schleife durchlaufen wird, bis im vorletzten Schritt in den Zustand p_2 übergegangen wird und durch das letzte $\{p_2\}$ sowohl die ϕ - Schleife als auch der Zustand p_2 in TRUE übergeht.
Der Automat akzeptiert niemals durch einen unendlichen Pfad, da durch die und-Verknüpfung die ϕ-Schleife entsteht, die nur durch einen Buchstaben, der nicht p_3 enthält, beendet werden kann. Dies sieht man leicht an der Formel, da $\neg q_3$ irgendwann erfüllt sein muß, damit $(\neg(X\neg p_2)Up_1)U\neg p_3$ erfüllt wird.

5 Leerheitstest auf alternierenden Automaten

Der Test auf Leerheit bei (schwachen) alternierenden Automaten ist keineswegs so trivial wie der Test auf nichtdeterministischen endlichen oder Büchi Automaten. Durch das Auftreten von universellen und existenziellen Übergängen reicht eine Erreichbarkeit von Endzuständen nicht mehr als einfache Bedingung aus. Es ist jedoch möglich, die (schwachen) alternierenden Automaten in Büchi-Automaten zu übersetzen, um so den einfacheren Test durchzuführen. Diese Übersetzung selbst bedeutet aber einen erheblichen Aufwand und benötigt exponentiell mehr Platz, womit ich mich aber genauer im nächsten Kapitel beschäftigen werde.

5.1 Nicht-Leerheitsproblem bei Büchi-Automaten

Sei A ein nichtdeterministischer endlicher Büchi-Automat. L(A) ist nicht leer gdw es einen Endzustand $q \in F$ gibt, der vom Anfangszustand q_0 aus erreichbar ist und es einen nichtleeren Pfad π von q nach q gibt. Die Konsequenz ist klar, und zwar daß jede nichtleere Büchi-erkennbare ω- Sprache ein periodisches ω-Wort enthält.

Definition 5.1
Sei G=(V,E) ein Graph. Eine starke Zusammenhangskomponente (SCC) ist eine maximale Teilmenge C von V, so daß es für alle $u, v \in C$ einen Pfad von u nach v gibt.

Aus der Graphtheorie ist bekannt, das jeder gerichtete Graph eindeutig in seine starken Zusammenhangskomponenten unterteilt ist und ein Algorithmus in Zeit O(|V|+|E|) diese herausfindet. Daraus läßt sich ein effektiver und einfacher Algorithmus zum Lösen des Leerheitsproblems für Büchi- Automaten bauen: man sucht die erreichbaren SCCs. Enthält eine erreichbare SCCs einen Endzustand, so ist L(A) nicht leer.

5.2 Schwache alternierende Parity-Automaten zu Büchi-Automaten

Das Arbeiten mit alternierenden Automaten hat in manchen Gesichtpunkten einige Vorteile, z.B. das einfache Komplementieren. Miyano und Hayashi schlugen in [MH84] eine Übersetzung von alternierenden Büchi Automaten in Büchi Automaten vor, dieser Ansatz wurde in [KV2000] etwas vereinfacht, aber meiner Meinung nach anschaulich vorgestellt. An diesem Vorschlag orientiert sich auch folgende Übersetzung eines schwachen alternierenden Parity- Automaten in einen Büchi - Automaten.

Satz 5.2
Sei $A = (Q, \Sigma, q_0, \delta, c)$ ein schwacher alternierender Parity-Automat. Dann existiert ein nichtdeterministischer endlicher Büchi Automat $A' = (2^Q \times 2^Q, \Sigma, (\{q_0\}, \emptyset), \delta', 2^Q \times \emptyset)$, so daß L(A)=L(A').

Beweis: Der Automat A' errät einen Lauf auf dem Automaten A. Dabei behält er zu jedem Zeitpunkt eine ganze Ebene des Laufbaumes von A und mit jedem gelesenen Buchstaben wird die nächste Ebene des Laufbaumes geraten. Die Zustände $(S, O) \in 2^Q \times 2^Q$ speichern dabei die besuchten Zustände (in einer Ebene) und davon die besuchten Zustände, die ungerade gefärbt sind. Daher akzeptiert A' nur Pfade, die unendlich oft einen Zustand der Form $2^Q \times \emptyset$ besucht haben, d.h. einen Zustand ohne ungerade gefärbte Zustände aus A. Dies bedeutet, das A in einer gerade gefärbten Teilmenge $Q_i \subseteq Q$ unendlich lange verweilt und dadurch, daß in einem Pfad die Färbung der Zustände nie steigen kann, ist somit die schwache Parity-Akzeptanzbedingung erfüllt.

Nun muß nur noch die Übergangsfunktion δ' definiert werden. Hierbei werden zwei Fälle unterschieden: Für $a \in \Sigma$ sei δ' wie folgt definiert:

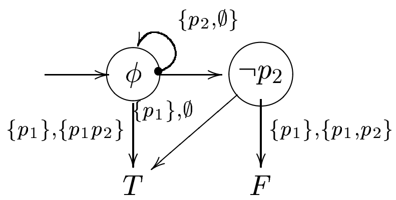
1. $O \neq \emptyset$: $\delta'((S, O), a) = \{(S', O' \setminus \{q | q \in O'$und c(q) gerade$\})|$ S' erfüllt $\bigwedge_{q \in S} \delta(q, a), O' \subseteq S'$, O' erfüllt $\bigwedge_{q \in O} \delta(q, a)\}$
2. $O = \emptyset$: $\delta'((S, O), a) = \{(S', S' \setminus \{q | q \in S'$und c(q) gerade$\})|$ S' erfüllt $\bigwedge_{q \in S} \delta(q, a)\}$

Zusätzlich gilt folgendes: Für alle $S \in 2^Q$ gilt: falls $\bigcap_{q \in S} \delta(q, a) = TRUE$ dann $S' = \emptyset$. Somit gelangt man in den Zustand $(\emptyset, \emptyset)$, falls man von einer Ebene nur noch TRUE-Übergänge hat und somit auf jeden Fall akzeptiert. Der Zustand $(\emptyset, \emptyset)$ ist eine akzeptierende Senke, da $(\emptyset, \emptyset) \in F$ und $\delta((\emptyset, \emptyset), a) = (\emptyset, \emptyset)$ für alle $a \in \Sigma$.

Beispiel: Da das Beispiel aus Kapitel 4 hier zu umfangreich wäre, betrachten ich die Teilformel $\phi = (X \neg p_2) U p_1$ des obigen Beispiels. Die Konstruktion mit dem vollständigen Beispiel aus Kapitel 4 befindet sich im Anhang. Der zugehörige Automat $A_\phi = (Q, 2^{\{p_1,p_2\}}, \phi, \delta, c)$ sieht wie folgt aus (Tabellennotation):

	c	$\delta(q,\emptyset)$	$\delta(q,\{p_1\})$	$\delta(q,\{p_2\})$	$\delta(q,\{p_1,p_2\})$
$(X\neg p_2)Up_1$	5	$\neg p_2 \wedge \phi$	T	$\neg p_2 \wedge \phi$	T
$(\neg X\neg p_2)Up_1$	4	$p_2 \vee \neg\phi$	F	$p_2 \vee \neg\phi$	F
$(X\neg p_2)$	3	$\neg p_2$	$\neg p_2$	$\neg p_2$	$\neg p_2$
$(\neg X\neg p_2)$	3	p_2	p_2	p_2	p_2
(p_2)	1	F	F	T	T
$(\neg p_2)$	1	T	T	F	F
(p_1)	1	F	T	F	T
$(\neg p_1)$	1	T	F	T	F

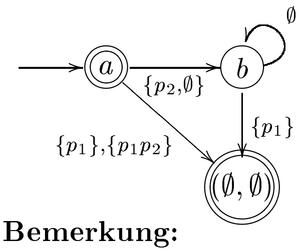
In graphischer Form (unter Auslassung von nicht erreichbaren Zuständen):



Der Automat akzeptiert ein Wort w gdw in einem Buchstaben von w p_1 vorkommt. Tritt ab dem zweiten Buchstaben jedoch p_2 auf, verwirft er sofort.
Die Konstruktion aus Satz 5.2 sieht wie folgt aus:

	$\emptyset$	$\{p_1\}$	$\{p_2\}$	$\{p_1,p_2\}$
$a = (\{\phi\},\emptyset)$	$(\{\phi,\neg p_2\},\{\phi,\neg p_2\})$	$(\emptyset,\emptyset)$	$(\{\phi,\neg p_2\},\{\phi,\neg p_2\})$	$(\emptyset,\emptyset)$
$b = (\{\phi,\neg p_2\},\{\phi,\neg p_2\})$	$(\{\phi,\neg p_2\},\{\phi,\neg p_2\})$	$(\emptyset,\emptyset)$	FALSE	FALSE
$(\emptyset,\emptyset)$	$(\emptyset,\emptyset)$	$(\emptyset,\emptyset)$	$(\emptyset,\emptyset)$	$(\emptyset,\emptyset)$

Graphisch:



Bemerkung:
Da diese Umformungen einen großen Aufwand beinhalten, verwendet man oft Sonderfälle von alternierenden Automaten, bei denen der Leerheitstest einfacher ist. So werden beim Model Checking alternierende Automaten mit $|\Sigma| = 1$ verwendet, wobei dann die folgenden beiden Sätze gelten:

Satz 5.3[BVW94] Das Nicht-Leerheitsproblem für alternierende Büchi - Automaten über einem 1-Buchstaben - Alphabet ist in quadratischer Zeit entscheidbar.

Satz 5.4 [BVW94] Das Nicht-Leerheitsproblem für schwache alternierende Büchi - Automaten über einem 1-Buchstaben - Alphabet ist in linearer Zeit entscheidbar.

6 Komplexität

Im folgenden werden Betrachtungen bezüglich der Komplexität der Konstruktionen aus Kapitel 5 aufgezeigt.
Satz 6.1
a) [EL85] Das Nicht-Leerheitsproblem für nichtdeterministische Büchi - Automaten ist in linearer Zeit entscheidbar.
b) [VW94] Das Nicht-Leerheitsproblem für nichtdeterministische Büchi - Automaten der Größe n ist mit Speicherplatz in $O(log^2 n)$ entscheidbar.

Der Beweis zu b) ist einfach und soll deswegen hier kurz genauer erläutert werden: Um zu entscheiden, ob der alternierende Büchi Automat A irgendein Wort akzeptiert, muß man testen, ob es einen erreichbaren Endzustand q gibt, der auch von sich selbst erreichbar ist. Der einfachste Weg ist, nichtdeterministisch einen Endzustand q zu raten, dann nichtdeterministisch einen Pfad von q_0 zu q und von q zu q zu suchen. Um einen Pfad von x nach y zu suchen, benötigt man:

1. Sei x der aktuelle Zustand
2. wähle einen Übergang von x aus und ersetze x durch den neuen Zustand
3. ist x = y, so ist der Pfad gefunden, ansonsten weiter bei (2)

Man braucht maximal nur 3 Zustände zu behalten, daher benötigt der nichtdeterministische Algorithmus nur logarithmischen Platz.
Das Nicht-Leerheitsproblem für alternierende Büchi-Automaten ist sogar NLOGSPACE-vollständig [VW94].

Satz 6.2
Das Nicht-Leerheitsproblem für schwache alternierende Automaten ist PSPACE - vollständig.

Beweis: Nicht-Leerheit von Büchi-Automaten kann nach Satz 6.1 in nichtdeterministisch logarithmischer Zeit getestet werden und da A_ϕ exponentielle Größe hat, erhält man einen Algorithmus, der polynomiellen Platz benötigt. Dieses Argument benötigt etwas Vorsicht: Da A_ϕ exponentiell groß ist, können wir diesen Automaten nicht einfach berechnen und auf Leerheit testen. Jedoch kann man A_ϕ „on-the-fly" konstruieren, d.h. immer wenn der Algorithmus für den Leerheitstest von einem Zustand q_1 in den Zustand q_2 innerhalb von A_ϕ gehen möchte, rät der Algorithmus q_2 und überprüft, ob

q_2 direkt mit q_1 verbunden ist. Sobald dies bestätigt ist, kann der Algorithmus q_1 verwerfen. So benötigt der Algorithmus in jedem Schritt nur zwei Zustände von A_ϕ, es ist unnötig den ganzen Automat in einem Schritt zu berechnen.
Somit bleibt zu zeigen, dass das Nicht-Leerheitsproblem für schwache alternierende Automaten PSPACE- schwer ist. Dazu geht man einen Umweg über das Universalitätsproblem bei nichtdeterministischen endlichen Automaten über endlichen Wörtern (NEA), das in PSPACE liegt [Var96]:
Wenn das Universalitätsproblem über NEA PSPACE-schwer ist, dann trivialerweise auch das Universalitätsproblem über schwachen alternierenden Automaten. Da aber WAA sehr einfach zu komplementieren sind (durch eine duale Übergangsfunktion) ist damit auch das Nicht-Leerheitsproblem über WAA PSPACE-schwer. Es bleibt also zu zeigen:

Satz 6.3
Das Universalitätsproblem für NEA ist PSPACE-vollständig.

Beweis: Der Beweis soll hier vom Prinzip her aufgezeigt werden und verzichtet daher auf einige Formalitäten, die aber klar sein sollten.
Ob $L(A) = \Sigma^*$ für einen NEA $A = (Q, \Sigma, q_0, \delta, F)$ gilt, ist in polynomiellen Speicherplatz entscheidbar. Dies ist einfach, da man den Automaten A komplementieren kann und dann auf Leerheit testet. Der Leerheitstest ist in NLOGSPACE möglich, das Komplementierung im schlimmsten Fall leider exponentiell, wobei man sich dann aber wieder auf die „on-the-fly“ Konstruktion beruft und auch in diesem Fall den Test in PSPACE durchführen kann.
Um die Schwere zu zeigen, muß man zeigen, dass für alle Sprachen $U \in PSPACE \quad U \leq_P L(A)$ gilt, also U polynomiell auf L(A) reduzierbar ist. Die Sprache U drücke ich durch eine (polynomiell)speicherplatzbeschränkte deterministische Turingmaschine (DTM) M aus, was möglich ist, da $U \in PSPACE$.
Zu einer solchen DTM M und einem Eingabewort x mit |x|=n konstruiert man einen NEA A_x, wobei gilt $x \in L(M) \equiv L(A_x) = \Sigma^*$. Dabei wird A_x so aufgebaut, dass $L(A_x) =$ { u | u kodiert keine akzeptierende Berechnung der DTM M auf dem Wort x } , wenn also keine akzeptierende Berechnung von M auf x existiert, so ist A_x universell.

Anstatt mit dem NEA A zu arbeiten, wird im folgenden ein regulärer Ausdruck R verwendet, da die Konstruktion damit klarer wirkt. Da man zu einem regulären Ausdruck einen höchstens polynomiell größeren NEA konstruieren kann, beeinflusst es den Beweis nicht.

Behauptung [AHU]:
Sei $M = (Q, T, I, \delta, \flat, q_0, q_f)$ eine Einband-DTM mit p(n)-Speicherplatzbeschränkung, wobei p ein Polynom ist. Dann existiert ein weiteres Polynom

p', so dass wenn $x \in I^*$ und |x|=n dann kann man einen polynomiell größeren regulären Ausdruck R_x konstruieren, so dass das Komplement von R_x die leere Menge bezeichnet gdw M x nicht akzeptiert.

Beweis: Der Ausdruck R_x beschreibt genau die Berechnungen von M, die nicht in einer Akzeptanz von x enden. R_x verwendet das Alphabet $\Delta \cup \{\sharp\}$, wobei $\Delta = T \cup \{[qX[|q \in Q$ und $X \in T\}$ und der Buchstabe $[qX]$ genau die Stelle auf dem Eingabeband von M wiederspiegelt, auf der der Lesekopf steht, der gerade X gelesen hat, dazu befindet sich die Turingmaschine in Zustand q. Jede Konfiguration (ID) benötigt höchstens p(|x|) Stellen auf dem Eingabeband, also kann jede ID durch ein Wort w_i mit $|w_i|$=p(|x|) dargestellt werden, das genau einen Buchstaben nicht aus T enthält, und zwar ein $[qX]$. Eine Berechnung kann nun als $\sharp w_1 \sharp w_2 \sharp \dots \sharp w_k \sharp \in \Delta^*$ für ein $k \geq 1$ dargestellt werden, wobei kürzere w_i mit Leerzeichen auf die Länge p(n) gebracht werden.

Ein Wort $y \in \Delta^*$ stellt keine akzeptierende Berechnung von M dar, wenn

1. y hat nicht die Form $\sharp w_1 \sharp w_2 \sharp \dots \sharp w_k \sharp$ für ein $k \leq 1$, wobei für alle $1 \leq i \leq k : |w_i| = n$ und alle Buchstaben aus y bis auf einen sind aus T, dieser eine ist aus $Q \times T$.

2. y beginnt nicht mit $\sharp[q_0 y_1]y_2 \dots y_n \flat\flat\flat\sharp$, wobei $y = y_1 \dots y_n$ und die Anzahl der Blanks entspricht p(n)-n.

3. In y erscheint nie ein Buchstabe der Form $[q_f X]$, es wird also nie ein Endzustand erreicht.

4. Zwei aufeinanderfolgende ID's kodieren keinen korrekten Schritt von M.

Zu jeder Bedingung i) erstellt man einen regulären Ausdruck R_i (i=1,2,3,4) und somit erhält man $R_x = R_1 + R_2 + R_3 + R_4$, wobei $x = a_1 a_2 \dots a_n$.

$$\begin{aligned}
R_1 = \; & \Delta^* + \Delta^*\sharp\Delta^* + \Delta(\Delta + \sharp)^* + (\Delta + \sharp)\Delta^* + \\
& (\Delta + \sharp)^*\sharp T^*\sharp(\Delta + \sharp)^* + \\
& (\Delta + \sharp)^*\sharp\Delta^*(Q \times T)\Delta^*(Q \times T)\Delta^*\sharp(\Delta + \sharp)^* + \\
& (\Delta + \sharp)^*\sharp\Delta^{p(n)+1}\Delta^*\sharp(\Delta + \sharp)^* + \\
& A_0 + \dots + A_{p(n)-1}
\end{aligned}$$

wobei $A_i = (\Delta + \sharp)^*\sharp\Delta^i\sharp(\Delta + \sharp)^*$

$R_2 = B_1 + \ldots + B_{p(n)}$ wobei

$$\begin{array}{rcl} B_1 & = & \sharp(\Delta - [q_0 a_1])\Delta^{p(n)-1}\sharp(\Delta + \sharp)^* \\ B_i & = & \sharp\Delta^{i-1}(T - [a_i])\Delta^{p(n)-i}\sharp(\Delta + \sharp)^* \text{ für } 1 < i \leq n \\ B_i & = & \sharp\Delta^{i-1}(T - \{\flat\})\Delta^{p(n)-i}\sharp(\Delta + \sharp)^* \text{ für } n < i \leq p(n) \end{array}$$

$R_3 = ((\Delta + \sharp) - (\{q_f\} \times T))^*$

$R_4 =$ Summe der Terme $R_{c_1c_2c_3}$ für alle $c_1, c_2, c_3 \in \Delta \cup \{\sharp\}$, wobei

$R_{c_1c_2c_3} = (\Delta + \sharp)^* c_1c_2c_3(\Delta + \sharp)^{p(n)-1} f'(c_1c_2c_3)(\Delta + \sharp)^*$
wobei $f'(c_1c_2c_3) = (\Delta \cup \{\sharp\}) - f(c_1c_2c_3)$, wenn die Funktion $f(c_1c_2c_3)$ den Buchstaben p(n)+1 Buchstaben rechts von c_2 (die gleiche Stelle in der nächsten ID) anhand der Übergangsfunktion δ der NTM definiert, z.B. $f(c_1c_2c_3) = c_2$ falls $c_1, c_2, c_3 \in T$.

Anhand der Konstruktion kann man erkennen, das R_x in $O(p^2(n))$ liegt, also offensichtlich eine polynomielle Konstruktion. □
Somit ist Satz 6.3 und mit ihm auch Satz 6.2 gezeigt.

Satz 6.4
Das Erfüllbarkeitsproblem für LTL ist PSPACE-vollständig.

Beweis: Mit Satz 6.2 und der Konstruktion aus Kapitel 4 folgt dies unmittelbar.

7 Zusammenfassung

In Kapitel 2 bis 4 wurde der enge Zusammenhang von (schwachen) alternierenden und Logiken (im besonderen LTL) gezeigt. Dabei wurde deutlich, das die Alternation diese Automaten der Logik ähnlicher macht als „übliche“ nichtdeterministische Automaten.
In Kapitel 5 und 6 wurde jedoch trotzdem die Schwierigkeit von bestimmten Problemen auf Automaten und somit auch die Schwierigkeit von logischen Problemen deutlich, die auch über die Automaten nicht vereinfacht werden können. Obwohl die Konstruktion in Kapitel 5 exponentiell ist, so tritt dieser schlechteste Fall in der Anwendung nicht immer auf, wodurch das Verfahren wieder attraktiver wird.

8 Referenzen

Literatur

[AHU] A.V. Aho, J.E. Hopcroft, J.D. Ullman. The Design and Analysis of Computer Algorithms, Kapitel 10.6, Addison-Wesley.

[BVW94] O. Bernholtz, M.Y. Vardi, P. Wolper. An automata-theoretic approach to branching time model-checking, LNCS 818, S. 142-155

[EJ91] E.A. Emerson, C. Jutla. Tree automata, μ-calculus and determinacy. Proc. 32nd IEEE Symposium on Foundations of Computer Science, S. 368-377

[EL85] E.A. Emerson, C.-L. Lei. Modlaities for model checking: Branching time logic strikes back. In Proceedings of the Twelfth ACM Symposium on Principles of Programming Languages, S. 84-96, New Orleans

[KV2000] O. Kupferman, M.Y. Vardi. Weak Alternating Automata are not that weak, http://www.cs.rice.edu/ vardi/papers/

[LT2000] C. Löding, W. Thomas. Alternating Automata and Logics over Infinite Words. LNCS 1872 S. 521-535

[MH84] S. Miyano, T. Hayashi. Alternating finite automata on ω - words. Theoretical Computer Science 32, S. 321-330, 1984

[Mo84] A. Mostowski. Regular expressions for infinite trees and standard form for automata. LNCS 208, S. 157-168

[MSS86] D.E. Muller, A. Saoudi, P.E. Schupp. Alternating automata, the weak monadic theory of the tree and its complexity, LNCS 226 S. 275-283, 1987

[Tho2000] W. Thomas. Vorlesung Automata and Modelchecking, WS2000/01 RWTH-Aachen

[Var96] Moshe Y. Vardi. An Automata Theoretic Approach to Linear Time Logic. LNCS 1043 S. 238-266

[Var94] Moshe Y. Vardi. Alternating Automata: Checking Truth and Validity for Temporal Logics, http://www.cs.rice.edu/ vardi/papers/

[VW94] M.Y. Vardi, P. Wolper. Reasoning about infinte computations. Information and Computation 115, S. 1 -37, 1994

9 Anhang

Die vollständige Konstruktion mit dem Beispiels aus Kapitel 4.

	$\delta(q,\emptyset)$	$\delta(q,\{p_1\})$	$\delta(q,\{p_2\})$	$\delta(q,\{p_3\})$	$\delta(q,\{p_1,p_2\})$
$(\phi,\emptyset)$	$(\emptyset,\emptyset)$	$(\emptyset,\emptyset)$	$(\emptyset,\emptyset)$	$(\{\phi,\psi,p_2\},\{\phi,p_2\})$, $(\{\phi,\psi\},\{\phi\})$, $(\{\phi,p_2\},\{\phi,p_2\})$	$(\emptyset,\emptyset)$
$(\{\phi,\psi,p_2\},$ $\{\phi,p_2\})$	FALSE	FALSE	$(\{\psi,p_2\},\emptyset)$, $(\{\psi\},\emptyset)$, $(\{p_2\},\emptyset)$	FALSE	FALSE
$(\{\phi,\psi\},\{\phi\})$	$(\{\psi,p_2\},\emptyset)$	FALSE	$(\{\psi,p_2\},\emptyset)$	$(\{\phi,\psi,p_2\},\{\phi,p_2\})$, $(\{\phi,\psi\},\{\phi\})$, $(\{\phi,p_2\},\{\phi,p_2\})$	FALSE
$(\{\phi,p_2\},\{\phi,p_2\})$	FALSE	FALSE	$(\emptyset,\emptyset)$	FALSE	$(\emptyset,\emptyset)$
$(\{\psi,p_2\},\emptyset)$	FALSE	FALSE	$(\{\psi,p_2\},\{p_2\})$, $(\{\psi\},\emptyset)$, $(\{p_2\},\{p_2\})$	FALSE	FALSE
$(\{\psi\},\emptyset)$	$(\{\psi,p_2\},\{p_2\})$, $(\{\psi\},\emptyset)$, $(\{p_2\},\{p_2\})$	FALSE	$(\{\psi,p_2\},\{p_2\})$, $(\{\psi\},\emptyset)$, $(\{p_2\},\{p_2\})$	$(\{\psi,p_2\},\{p_2\})$, $(\{\psi\},\emptyset)$, $(\{p_2\},\{p_2\})$	FALSE
$(\{p_2\},\emptyset)$	FALSE	FALSE	$(\emptyset,\emptyset)$	FALSE	$(\emptyset,\emptyset)$
$(\{\psi,p_2\},\{p_2\})$	FALSE	FALSE	$(\{\psi,p_2\},\emptyset)$, $(\{\psi\},\emptyset)$, $(\{p_2\},\emptyset)$	FALSE	FALSE
$(\{p_2\},\{p_2\})$	FALSE	FALSE	$(\emptyset,\emptyset)$	FALSE	$(\emptyset,\emptyset)$

	$\delta(q,\{p_1,p_3\})$	$\delta(q,\{p_2,p_3\})$	$\delta(q,\{p_1,p_2,p_3\})$
$(\phi,\emptyset)$	$(\{\phi,\psi,p_2\},\{\phi,p_2\})$, $(\{\phi,\psi\},\{\phi\})$, $(\{\phi,p_2\},\{\phi,p_2\})$	$(\{\phi,\psi,p_2\},\{\phi,p_2\})$, $(\{\phi,\psi\},\{\phi\})$, $(\{\phi,p_2\},\{\phi,p_2\})$	$(\{\phi,\psi,p_2\},\{\phi,p_2\})$, $(\{\phi,\psi\},\{\phi\})$, $(\{\phi,p_2\},\{\phi,p_2\})$
$(\{\phi,\psi,p_2\},$ $\{\phi,p_2\})$	FALSE	$(\{\phi,\psi,p_2\},\{\phi,p_2\})$, $(\{\phi,\psi\},\{\phi\})$, $(\{\phi,p_2\},\{\phi,p_2\})$	FALSE
$(\{\phi,\psi\},\{\phi\})$	FALSE	$(\{\phi,\psi,p_2\},\{\phi,p_2\})$, $(\{\phi,\psi\},\{\phi\})$, $(\{\phi,p_2\},\{\phi,p_2\})$	FALSE
$(\{\phi,p_2\},$ $\{\phi,p_2\})$	FALSE	$(\{\phi,\psi,p_2\},\{\phi,p_2\})$, $(\{\phi,\psi\},\{\phi\})$, $(\{\phi,p_2\},\{\phi,p_2\})$	$(\{\phi,\psi,p_2\},\{\phi,p_2\})$, $(\{\phi,\psi\},\{\phi\})$, $(\{\phi,p_2\},\{\phi,p_2\})$
$(\{\psi,p_2\},\emptyset)$	FALSE	$(\{\psi,p_2\},\{p_2\}),(\{\psi\},\emptyset)$, $(\{p_2\},\{p_2\})$	FALSE
$(\{\psi\},\emptyset)$	FALSE	$(\{\psi,p_2\},\{p_2\}),(\{\psi\},\emptyset)$, $(\{p_2\},\{p_2\})$	FALSE
$(\{p_2\},\emptyset)$	FALSE	$(\emptyset,\emptyset)$	$(\emptyset,\emptyset)$
$(\{\psi,p_2\},\{p_2\})$	FALSE	$(\{\psi,p_2\},\emptyset),(\{\psi\},\emptyset)$, $(\{p_2\},\emptyset)$	FALSE
$(\{p_2\},\{p_2\})$	FALSE	$(\emptyset,\emptyset)$	$(\emptyset,\emptyset)$