Bachelorarbeit

Ausgegeben und betreut von

Prof. Dr. Martin Leucker

Institut für Softwaretechnik und Programmiersprachen
Universität zu Lübeck

UNIVERSITÄT ZU LÜBECK
INSTITUTE OF SOFTWARE ENGINEERING
AND PROGRAMMING LANGUAGES

Transformation von Paritätsautomaten in Büchi-Automaten

Torben Scheffel

Lübeck, den 15. November 2011

Erklärung

Hiermit versichere ich, dass ich die vorliegende Arbeit
selbstständig und unter ausschließlicher Verwendung
der angegebenen Literatur angefertigt habe.

_____

(Torben Scheffel)
Bad Oldesloe, 15.11.2011

Abstract In dieser Arbeit geht es um die Umwandlung von alternierenden Paritätsautomaten mit maximal 3 Paritäten zu nicht-deterministischen Büchi-Automaten. Hierfür werden mehrere mögliche Umwandlungen vorgestellt, sowie die davon ausgewählte Transformation genauer beschrieben. Insgesamt wird im Rahmen dieser Arbeit ein Java-Programm erstellt und dessen Funktion genau beschrieben, welches diese Umwandlung automatisiert vornimmt.

Abstract This thesis is about the transformation of alternating parity automata with a maximum of three parities into non-deterministic Büchi-automata. For this purpose, several possible transformations are presented and the selected transformation is described more accurately. A program is written in Java and described, to perform this transformation automatically.

Inhaltsverzeichnis

1 Einleitung

Eines der größten Probleme in der Software- und Hardwareentwicklung ist, eine 100%ige Verlässlichkeit des Systems zu garantieren, daher also den Beweis zu erbringen, dass das System zuverlässig ist. Eine Möglichkeit zum Beweisen der Fehlerfreiheit eines Systems wird auch „Model-Checking“ genannt. Ein Aussetzer eines solchen Systems kann enorme Folgen haben, wie zum Beispiel bei Systemen, die in der Raumfahrt genutzt werden. Bei einem Fehler können hohe Kosten für das Unternehmen entstehen oder auch Tote zu beklagen sein. Das bekannteste Beispiel dafür ist wohl der Softwarefehler bei der Ariane V[1], welcher durch einen Überlauf einer Variable des Lenksystems entstand.

Bis heute werden solche Verlässlichkeitstests oft per Hand gemacht, also Testfälle geschrieben. Hierbei gibt es aber mehrere Probleme. Zum Einen sind diejenigen, welche die Tests schreiben, oft auch an der Entwicklung des Systems beteiligt und betrachten Stellen, an denen sie Probleme vermuten, besonders und andere, bei denen sie Probleme eher weniger vermuten, werden oft weniger beim Testen beachtet. Dadurch werden Fehler übersehen, weil eher da viel getestet wird, wo der Tester auch bei der Entwicklung gemerkt hat, dass es Schwierigkeiten gab. Selbst wenn es eine sehr gute Organisation im Unternehmen gibt, was das Testen angeht, ist es nicht gewährleistet, dass das System nach dem Testen wirklich fehlerfrei ist. Daher will man dazu übergehen, ein Modell eines Systems in einer Spezifikationssprache zu erstellen, welches dann von Rechnern getestet werden kann, um die Fehlerfreiheit des Systems zu beweisen.

Ein genau so wichtiger Bereich ist auch die Laufzeitverifikation von Programmen, bei der Programme während der Laufzeit überwacht werden. Dadurch sollen Fehler möglichst früh erkannt werden, damit noch rechtzeitig reagiert werden kann, bevor diese Fehler zu noch größeren Problemen führen. Dabei werden, zum Beispiel, Variablen überprüft, ob diese bestimmte Werte nicht über- oder unterschritten haben.

Das gewünschte Verhalten eines Programmes wird sowohl für das Model-Checking als auch für die Laufzeitverifikation mit Hilfe von Spezifikationssprachen wie die Linear Temporal Logic (LTL), die Computation Tree Logic (CTL), μ-calculus oder die Regular Linear Temporal Logic (RLTL) spezifiziert [leu02]. Alle zugelassenen Variablenbelegungen sind dann ein Modell für die jeweilige Formel und alle anderen sind kein Modell. Das heißt, ist eine Variablenbelegung kein Modell, so ist ein nicht zulässiger Zustand des Systems erreicht und es können Maßnahmen ergriffen werden.

Hierbei ist die RLTL bzw. die Regular Linear Temporal Logic with past (pRLTL) eine Erweiterung von der LTL und wird in [sale07] und in [sale10] vorgestellt.

[1] `http://de.wikipedia.org/wiki/Ariane_V88`

Diese, in einer Spezifikationssprache verfassten, Formeln, können nun in Automaten überführt werden, um die Überprüfung für bestimmte Anwendungsbereiche zu vereinfachen. Die Überführung von pRLTL zu alternierenden Paritätsautomaten wird zum Beispiel in [san11] beschrieben. Je nachdem, für welchen Zweck man diese Spezifizierung nutzen will, sind Formeln oder Automaten besser geeignet. Die theoretische Umwandlung sowie ein Programm, dass die Umwandlung von pRLTL zu alternierenden Paritätsautomaten automatisiert vornimmt, wird in [sch11] vorgestellt.

Wie gerade erwähnt, können pRLTL-Formeln in alternierende Paritätsautomaten umgewandelt werden, welche auf unendlichen Wörtern arbeiten. Da die Akzeptanzbedingung von alternierenden Paritätsautomaten für einen Computer nur schwer zu überprüfen ist, wird der alternierende Paritätsautomat weiter in einen nicht-deterministischen Büchi-Automaten umgewandelt, der die selbe Sprache akzeptiert und auch auf unendlichen Wörtern arbeitet. Diese Umwandlung wird das Hauptthema dieser Arbeit sein. Sie ist allerdings aufgrund vieler Faktoren nicht einfach umsetzbar. Zu diesen gehören unter anderem:

- Die stark unterschiedlichen Akzeptanzbedingungen, welche für eine direkte Umwandlung zueinander ungeeignet sind und
- die Eigenschaft des Alternierends, welche eine exponentielle Vergrößerung der Menge der Zustände erzeugt, wenn sie zu nicht-Determinismus umgewandelt wird. Besondere Probleme bei der Umwandlung von einem alternierenden zu einem nicht-deterministischen Automaten gibt es bei Automatentypen, die mehr als nur 2 verschiedene Arten von Zuständen haben.

Ein weiteres Problem ist es, dass diese beiden Probleme zusammentreffen, denn dadurch entstehen noch einige Spezialfälle, die eine direkte Umwandlung weiter erschweren. Vor allem aber entsteht eine Erhöhung der Zustände von n auf $2^{O(n^2)}$ [kla08].

Insgesamt ergibt sich dann, mit dem vorher erwähnten Programm aus [sch11], eine Programmkette, mit deren Hilfe pRLTL-Formeln in nicht-deterministische Büchi-Automaten automatisiert umgewandelt werden können. Diese könnte man weiter transformieren, um sie dann in einem Model-Checker wie, zum Beispiel, SPIN[2] zu verwenden.

[2] `http://spinroot.com/spin/whatispin.html`

2 Akzeptanzbedingungen und Eigenschaften von ω-Automaten

In diesem Kapitel werden alle Automatentypen definiert, die im weiteren Verlauf benötigt werden. Dies sind ausschließlich endliche Automaten die auf unendlichen Worten arbeiten, auch ω-Automaten genannt. Darunter fallen Büchi-Automaten und Paritätsautomaten sowie Streett- und Rabin-Automaten, aber auch der Determinismus, der nicht-Determinismus und die Eigenschaft des Alternierends eines Automaten werden für die weiteren Kapitel der Arbeit wichtig sein. Weiter wird noch eine graphische Darstellungsmöglichkeit für alternierende Automaten gegeben, welche in späteren Teilen dieser Arbeit noch verwendet wird.

2.1 Notationen und Definitionen

2.1.1 ω-Automat

ω-Automaten sind endliche Automaten, die auf unendlich Worten arbeiten. Intuitiv heißt das, sie halten niemals an. Da sie nur endlich viele Zustände haben, gelangen sie irgendwann mal in einen Zyklus, den sie unendlich oft durchlaufen.

Definition 1 (ω-Automat)**.** *Ein ω-Automat, im weiteren Verlauf nur Automat genannt, A ist ein Tupel $A = (\Sigma, Q, I, F, \delta)$, wobei*

- Σ *das* Eingabealphabet*,*
- Q *eine endliche Menge der* Zustände*,*
- I *die* Anfangskonfiguration *der Zustände,*
- F *die* Akzeptanzbedingung *und*
- δ *die* Transitionsfunktion *ist.*

Je nachdem, welche Eigenschaft der Automat hat, werden I und δ unterschiedlich sein und F ändert sich mit der Akzeptanzbedingung. Σ und Q werden bei allen betrachteten Automatentypen gleich definiert sein.

2.1.2 Positive boolsche Formeln und Modelle

Die Menge der positiven boolschen Formeln über eine Menge Q von Zuständen, wird $\mathbb{B}^+(Q)$ genannt. In einer solchen Formel gibt es keine Negationen sondern nur Zustände, die als boolsche Variablen fungieren und die durch $\wedge$ oder $\vee$ verknüpft sind.

Definition 2 (Menge der positiven boolschen Formeln)**.** *Sei Q eine Menge aus boolschen Variablen. Die Menge der positiven boolschen Formeln $\mathbb{B}^+(Q)$ ist dann die kleinste Menge, für die gilt:*

- $Q \subseteq \mathbb{B}^+(Q)$,
- $x, y \in \mathbb{B}^+(Q) \Rightarrow x \vee y, x \wedge y \in \mathbb{B}^+(Q)$.

Sei $x \in \mathbb{B}^+(Q)$ und M eine Menge von Zuständen. M ist ein Modell für x, also $M \models x$, wenn x erfüllt ist wenn man alle Zustände in M als wahr und alle Zustände, die nicht in M sind, als falsch auswertet. M ist ein minimales Modell für x, also $M \models\!\!\!\!\cdot\, x$, wenn M ein Modell für x ist und es keine Menge N gibt, die auch ein Modell für x ist und für die gilt $N \subset M$.

2.1.3 Lauf-Bäume

Ein Lauf eines ω-Automaten kann als ein Directed Acyclic Graph (DAG) dargestellt werden, der dann Lauf-Baum heißt. Dieser enthält dann in seinen Knoten jeweils den Zustand, in dem sich der Automat gerade befindet, und die Entfernung zu seiner Wurzel, auch Tiefe genannt. Befindet sich ein Automat in mehreren Zuständen gleichzeitig, so gibt es auch in dem Lauf-Baum mehrere Zustände mit gleicher Tiefe.

Definition 3 (Lauf-Baum)**.** *Sei $A = (\Sigma, Q, I, F, \delta)$ ein ω-Automat und $w = w_0 w_1 w_2 w_3 \ldots \in \Sigma^\omega$ ein unendliches Wort. Ein Lauf ρ von w auf A ist ein Σ-beschrifteter, unendlicher DAG, auch Lauf-Baum genannt, für den gilt:*

- *Die Wurzeln sind alle Knoten des Baumes $\{(q_0, 0) \mid q_0 \in M_I\}$, wobei M_I ein minimales Modell für die Anfangskonfiguration I ist,*
- *alle weiteren Knoten sind mit (p, x) beschriftet, wobei $p \in Q$ und $x \in \mathbb{N}$ die Tiefe ist,*
- *sei P die Menge Zustände der Kinder von einem Knoten (p, x), es gilt also für jedes Kind $(q, x+1)$ von (p, x) $q \in P$, so gilt $P \models\!\!\!\!\cdot\, \delta(p, w_x)$.*

In Abbildung 2.1 sieht man ein Beispiel für einen Lauf-Baum. Man sieht, dass der Lauf-Baum genau den Pfad widerspiegelt, den der Automat bei Eingabe des Wortes $w = (ab)^\omega$ nimmt. Dies gilt auch allgemein, dass ein Lauf-Baum immer genau den Lauf eines Automaten für ein Eingabewort widerspiegelt. Damit sind Lauf-Bäume nichts anderes, als eine Darstellung für einen Lauf, in der nicht nur die Reihenfolge der Zustände gezeigt wird, durch die der Automat läuft, sondern in der, durch die zweite Komponente eines Knoten, auch Tiefe genannt, in dem Lauf-Baum auch die zeitliche Komponente darstellt wird. Dies ist später für Automaten, die gleichzeitig in mehreren

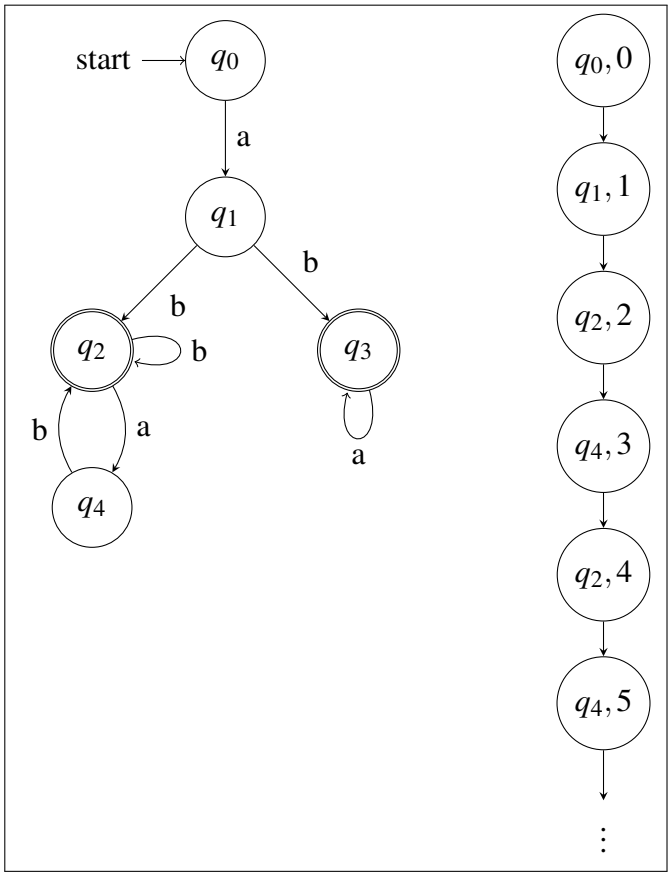

Abbildung 2.1: Beispiel für einen nicht-deterministischen Büchi-Automaten (links) und dessen Lauf-Baum (rechts) für einen Lauf eines unendlichen Wortes $w = (ab)^{\omega}$ auf dem Automaten.

Zuständen sein können, sehr wichtig, da man dadurch bei einem Lauf erkennen kann, in welchen Zuständen der Automat gleichzeitig ist.

Da Läufe eines ω-Automaten unendlich lange dauern wird auch dieser Lauf-Baum unendlich groß und er wird in seinen Knoten in der ersten Komponente immer weiter abwechselnd q_2 und q_4 haben. Damit sind Lauf-Bäume auch eher ein theoretisches Konstrukt, welches für die spätere Umwandlung als Darstellung benötigt wird, und sind nur, wie in Abbildung 2.1, in Ausschnitten darstellbar.

Wird in einem Lauf-Baum durch die Zustände in den Knoten die Akzeptanzbedingung des Automaten erfüllt, so ist auch der Lauf, für den der Baum erstellt wurde, akzeptierend.

2.2 Eigenschaften von Automaten

In diesem Unterkapitel werden der Determinismus, Nicht-Determinismus, das Alternieren eines Automaten und eine grafische Darstellung für alternierende Automaten beschrieben. Die Akzeptanzbedingungen werden im nächsten Unterkapitel behandelt.

2.2.1 Determinismus

Ein Automat $A = (\Sigma, Q, I, F, \delta)$ heißt deterministischer Automat, wenn seine Transitionsfunktion eine partielle Abbildung $\delta : Q \times \Sigma \to Q$ ist. Das heißt, dass δ von dem aktuellen Zustand für jede Eingabe entweder eindeutig auf einen neuen Zustand abbildet oder auf keinen Zustand abbildet. Bildet δ für eine Eingabe von einem Zustand auf keinen Folgezustand ab, so wird das Wort verworfen. Weiterhin darf I nur aus einem Zustand bestehen, der dann Startzustand heißt.

Ein Lauf eines Wortes auf einem deterministischen Automaten besteht immer nur aus einem Pfad, der fest vorbestimmt ist. Ist dieser Pfad akzeptierend, so ist auch der Lauf akzeptierend. Dieser Lauf ist also einfach nur eine Kette von Zuständen, die nacheinander besucht werden.

2.2.2 Nicht-Determinismus

Ein Automat $A = (\Sigma, Q, I, F, \delta)$ heißt nicht-deterministischer Automat, wenn seine Transitionsfunktion eine partielle Abbildung $\delta : Q \times \Sigma \to 2^Q$ ist, wobei 2^Q die Potenzmenge von Q sein soll. Das heißt, dass δ entweder von dem aktuellen Zustand für jede Eingabe auf eine beliebige Teilmenge von Q abbilden kann oder auf die leere Menge. Bildet δ von einem Zustand bei einer Eingabe auf die leere Menge ab, so wird das Wort verworfen. Von der Menge der Folgezustände kann dann ein Zustand nicht-deterministisch als Folgezustand ausgewählt werden. Auch I darf dann eine beliebige Teilmenge von Q sein und als Startzustand wird dann ein Zustand aus I nicht-deterministisch ausgewählt.

Ein Lauf eines Wortes auf einem nicht-deterministischen Automaten besteht aus einem Pfad, dessen Verlauf allerdings nicht-deterministisch von dem Automaten bestimmt wird. Ist dieser Pfad akzeptierend, so ist auch der Lauf akzeptierend. Dieser Lauf ist also wie bei einem deterministischen Automaten einfach nur eine Kette von Zuständen, die nacheinander besucht werden.

2.2.3 Alternierende Automaten

Bei einem alternierenden Automaten ist die Transitionsfunktion δ eine partielle Abbildung $\delta : Q \times \Sigma \to \mathbb{B}^+(Q)$. Das heißt, es könnte zum Beispiel $\delta(q,x) = q_1 \vee q_2$ oder $\delta(q,x) = q_1 \wedge q_2$ gelten. Dabei würde das $\vee$ dafür stehen, dass nicht-deterministisch entweder nach q_1 oder nach q_2 gegangen wird, genau wie bei einem nicht-deterministischen Automaten. Die große Besonderheit ist das $\wedge$, welches heißt, dass sowohl nach q_1 als auch nach q_2 gegangen wird. Nun muss man sich dies so vorstellen, dass es danach 2 „Parallelautomaten“ gibt, daher, der selbe Automat nun 2 mal existiert und sich einmal in q_1 und einmal in q_2 befindet. Weiter ist es auch möglich, dass δ für eine Eingabe von einem Zustand aus auf keinen Folgezustand abbildet. Wird diese Eingabe gelesen, so ist der Pfad, auf dem dies passiert ist, verwerfend.

Ein alternierender Automat akzeptiert ein Wort w, wenn alle Pfade (also alle Parallelautomaten) das Wort akzeptieren, das heißt, wenn es in jedem Pfad einen Zyklus gibt, der die Akzeptanzbedingung erfüllt.

Ein alternierender Automat bewegt sich von einem Zustand p bei Eingabe x in alle Folgezustände, die in einer nicht-deterministisch ausgewählten Menge M sind, für die gilt $M \models \delta(p,x)$. Der Automat befindet sich dann nach dem Lesen von x in allen Zuständen aus M gleichzeitig. Es macht keinen Sinn, eine Menge zu wählen, die kein minimales Modell für $\delta(p,x)$ ist, denn wenn der Automat das Wort mit keiner minimal gewählten Anzahl von Pfaden akzeptiert, so wird er es auch bei mehr Pfaden nicht akzeptieren.

I ist die Anfangskonfiguration der aktuellen Zustände und bei einem alternierenden Automaten gilt $I \in \mathbb{B}^+(Q)$. Das heißt, ein alternierender Automat startet einen Lauf auch in einem minimalen Modell von I.

Ein Lauf eines Wortes auf einem alternierenden Automaten besteht aus mehreren Pfaden, die parallel durchlaufen werden. Jedes mal, wenn bei einer Transition die vorher erwähnte Menge M mehr als einen Zustand enthält, entsteht für jeden Zustand ein Pfad, denn man befindet sich in allen Zuständen gleichzeitig und von jedem dieser Zustände aus werden weitere Buchstaben gelesen. Damit ist ein Lauf eines alternierenden Automaten ein Lauf-Baum, welcher für jeden aus einer Transition entstanden Pfad eine Verzweigung enthält. Sind alle diese Pfade akzeptierend, so ist auch der Lauf akzeptierend.

Definition 4. *Ein Automat $A = (\Sigma, Q, I, F, \delta)$ heißt alternierender Automat, wenn die Transitionsfunktion δ eine Abbildung $\delta : Q \times \Sigma \to \mathbb{B}^+(Q)$ ist und $I \in \mathbb{B}^+(Q)$. Es gilt*

- *Sei p der aktuelle Zustand und x der gelesene Eingabebuchstabe. So befindet sich der Automat danach in allen Zuständen $q \in M$, wobei $M \models \delta(p,x)$*

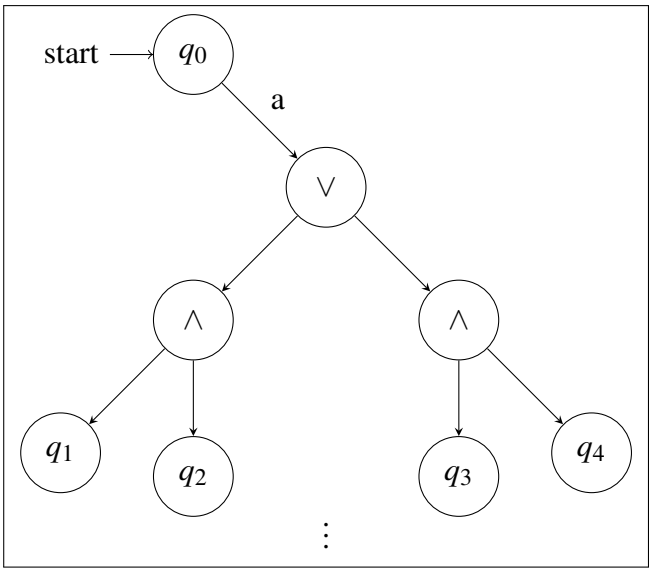

Abbildung 2.2: Eine Abbildung von einem Zustand und einem Eingabebuchstaben auf eine positive boolsche Formel in einem alternierenden Automaten als graphische Darstellung.

- *Die Startzustände sind alle Zustände $q \in M$, wobei $M \models I$.*

Läufe von alternierenden Automaten werden als ein Lauf-Baum dargestellt. Man kann auch Läufe von deterministischen oder nicht-deterministischen Automaten als Lauf-Baum darstellen, allerdings sind diese trivial, denn sie sind nur eine Liste aus Knoten ohne Verzweigungen. Dadurch, dass ein alternierender Automat in mehreren Zuständen gleichzeitig sein kann, können sich Lauf-Bäume von Läufen auf alternierenden Automaten auch verzweigen.

Da man das Verhalten, dass mehrere Pfade von einem Zustand aus akzeptierend sein sollen, bei alternierenden Automaten sehr leicht darstellen kann, sind diese in diesen Fällen deutlich kompakter als deterministische oder nicht-deterministische Automaten, allerdings sind alternierende Automaten auch nicht mächtiger als deterministische oder nicht-deterministische.

2.2.4 Grafische Darstellung von alternierenden Automaten

In diesem Abschnitt werde ich eine Möglichkeit geben, alternierende Automaten auf gut überschaubare Weise darzustellen. Diese wurde auch schon in [leu02] benutzt. Eine andere Möglichkeit ist, zum Beispiel, in [meh06] gegeben.

Um eine Übersichtliche Darstellung für alternierende Automaten zu schaffen werde ich zu den vorhandenen Zuständen noch „Logikzustände“ einfügen. Mit Hilfe dieser wird angezeigt, zu welcher positiven boolschen Formel von Zuständen der Automat bei einer gelesenen Eingabe von einem Zustand aus wechselt.

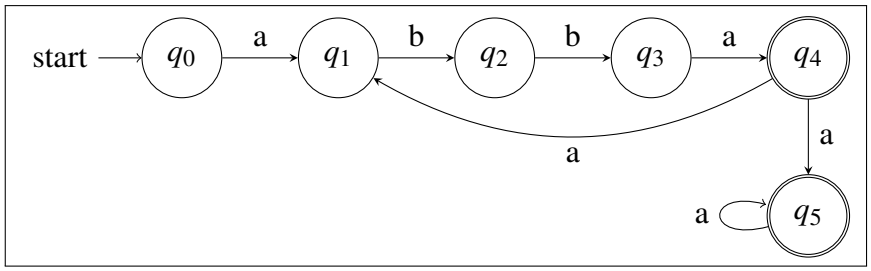

Abbildung 2.3: Ein Beispiel für einen nicht-deterministischen Büchiautomaten über dem Alphabet $\Sigma = \{a,b\}$, der alle Worte $(abba)^\omega$ oder $(abba)^x a^\omega$ mit $x \in \mathbb{N}, x > 0$ akzeptiert.

In Abbildung 2.2 sieht man $\delta(q_0,a) = (q_1 \wedge q_2) \vee (q_3 \wedge q_4)$ grafisch dargestellt. Es gibt von q_0 beim Einlesen von einem a den Übergang zu dem $\vee$, einem „Logikzustand“. Diese Logikzustände sind nur in der grafischen Darstellung vorhanden, sie gehören nicht zum Automaten und sollen nur die Transitionsfunktion besser darstellen. Von diesem $\vee$ aus bewegt man sich dann ohne weitere Eingabe weiter, bis man zu einem Zustand kommt, der kein Logikzustand ist. Man läuft dann von dem $\vee$ nicht-deterministisch zu einem der beiden $\wedge$. Von dem gewählten $\wedge$ geht man dann in 2 Zustände. Insgesamt befindet man sich, nachdem man in q_0 ein a gelesen hat, entweder in q_1 und in q_2 oder in q_3 und in q_4. Durch diese Art der Darstellung kann man relativ leicht nachverfolgen, wo sich der Automat nach einer gelesenen Eingabe befindet.

2.3 Akzeptanzbedingungen

2.3.1 Büchi-Automaten

Ein Büchi-Automat ist ein ω-Automat, bei dem F die Menge der akzeptierenden Zustände ist. Sei $Inf(\rho)$ die Menge der unendlich oft betrachteten Zustände in einem Lauf ρ auf einem Büchi-Automaten. Ein Pfad, der in dem Lauf ρ gelaufen wird, heißt genau dann akzeptierend, wenn $F \cap Inf(\rho) \neq \emptyset$, es in den unendlich oft betrachteten Zuständen also mindestens einen Zustand gibt, der akzeptierend ist.

Ein Beispiel für einen nicht-deterministischen Büchi-Automaten (NBA) ist in Abbildung 2.3 zu sehen. Dieser akzeptiert Worte der Form $(abba)^\omega$ oder $(abba)^x a^\omega$ mit $x \in \mathbb{N}, x > 0$, denn wenn er unendlich oft *abba* liest, so durchläuft er auch unendlich oft q_4 und wenn er nur endlich oft *abba* liest, mindestens aber 1 mal, und danach unendlich viele a's, so wechselt er nicht-deterministisch von q_4 nach q_5, der dann unendlich oft gesehen wird. Da sowohl q_4 als auch q_5 akzeptierend sind wird also in beiden Fällen ein akzeptierender Zustand unendlich oft gesehen.

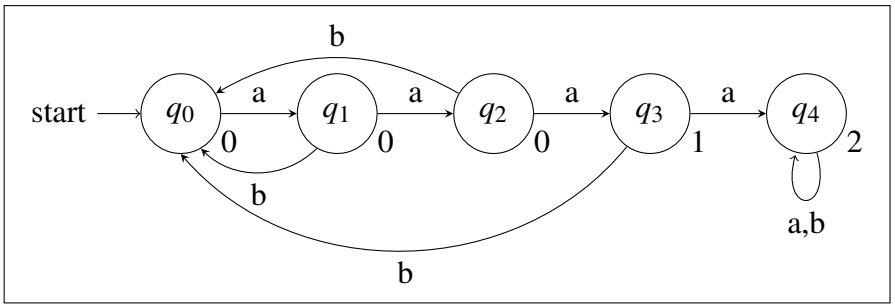

Abbildung 2.4: Ein Beispiel für einen deterministischen Paritätsautomaten über dem Alphabet $\Sigma = \{a,b\}$. Die Paritäten stehen jeweils an den Zuständen. Der Automat akzeptiert alle Worte, die entweder irgendwann mindestens ein Mal vier a's hintereinander haben oder die nicht unendlich oft immer mal wieder eine Folge aus drei a's besitzen.

2.3.2 Paritätsautomaten

Ein Paritätsautomat ist ein ω-Automat, bei dem F eine Funktion $F : Q \to \mathbb{N}$ ist. Das heißt, F bildet jeden Zustand auf eine Zahl ab. Diese Zahl wird auch Parität, Priorität oder Farbe genannt. In dieser Arbeit wird diese Zahl Parität heißen.

Sei ρ wie zuvor ein Lauf über ein unendlich langes Eingabewort w und sei weiter $Inf(\rho)$ die Menge der Zustände, die unendlich oft in dem Lauf ρ vorkommen. Ein Pfad der in dem Lauf ρ gelaufen wird heißt genau dann akzeptierend, wenn $\max\{F(q) \mid q \in Inf(\rho)\}$, also die größte Parität der unendlich oft betrachteten Zustände, gerade ist. Ist dies nicht der Fall, so wird w verworfen.

Ein n-Paritätsautomat ist ein Paritätsautomat mit nur n Paritäten. In dieser Arbeit werden nur 3-Paritätsautomaten benutzt. Diese haben dann entweder die Paritäten 0, 1 und 2 oder 1, 2 und 3. Man kann natürlich auch andere Werte benutzen, wichtig ist nur, dass die mittlere Parität gerade ist und die anderen beiden ungerade oder umgekehrt, denn sonst wäre eine Parität überflüssig.

Ein Beispiel für einen deterministischen Paritätsautomaten (DPA) ist in Abbildung 2.4 zu sehen. Dieser akzeptiert Worte, die entweder mindestens ein Mal vier a's hintereinander haben oder die nicht unendlich oft immer mal wieder eine Folge aus drei a's besitzen. Dies liegt daran, dass dieser Automat ein Wort nur verwirft, wenn er unendlich oft den Zustand q_3 durchläuft und nicht nach q_4 wechselt, denn dann ist die größte, unendlich oft gesehene, Parität 1, also ungerade. Also verwirft er nur, wenn das Wort nie vier a's hintereinander hat, aber unendlich oft immer mal wieder eine Folge aus drei a's besitzt.

2.3.3 Streett-Automaten

Ein Streett-Automat ist ein ω-Automat, bei dem F eine Menge von 2-Tupeln (B_i, G_i) ist, wobei B_i und G_i wiederum Mengen aus Zuständen sind.

Sei ρ wie zuvor ein Lauf über ein unendlich langes Eingabewort w und sei weiter $Inf(\rho)$ die Menge der Zustände, die unendlich oft in dem Lauf ρ vorkommen. Ein Pfad, der in dem Lauf ρ gelaufen wird, heißt genau dann akzeptierend, wenn für alle i gilt:

$$\forall i : inf(\rho) \cap B_i \neq \emptyset \Rightarrow inf(\rho) \cap G_i \neq \emptyset.$$

Man muss also, wenn man unendlich oft einen Zustand aus B_i sieht auch einen aus G_i unendlich oft sehen. Sieht man keinen aus B_i unendlich oft, so ist für das i-te Tupel die Bedingung erfüllt.

Aus der oben genannten Bedingung hat sich für die Zustände aus den Mengen B_i auch „böse Zustände“ und für die Zustände aus den Mengen G_i auch „gute Zustände“ als Name ergeben. Alle Zustände, die in keiner Menge sind heißen einfach neutrale Zustände.

Da für die Umwandlung nur 1-Streett-Automaten, also Streett-Automaten, deren Menge F nur ein Tupel (B,G) enthält, benötigt werden, wird im Folgenden statt B_i und G_i nur noch B und G benutzt.

2.3.4 Rabin-Automaten

Ein Rabin-Automat ist ein ω-Automat, bei dem F wie bei der Streett-Akzeptanzbedingung eine Menge von 2-Tupeln (B_i, G_i) ist, wobei B_i und G_i wiederum Mengen aus Zuständen sind.

Sei ρ wie zuvor ein Lauf über ein unendlich langes Eingabewort w und sei weiter $Inf(\rho)$ die Menge der Zustände, die unendlich oft in dem Lauf ρ vorkommen. Ein Pfad, der in dem Lauf ρ gelaufen wird, heißt bei einem Rabin-Automaten genau dann akzeptierend, wenn:

$$\exists i : inf(\rho) \cap B_i \neq \emptyset \wedge inf(\rho) \cap G_i = \emptyset.$$

Damit ist die Rabin-Akzeptanzbedingung genau die Negation der Streett-Akzeptanzbedingung.

Beweis. Sei ρ wie zuvor ein Lauf über ein unendlich langes Eingabewort w und sei weiter $Inf(\rho)$ die Menge der Zustände, die unendlich oft in dem Lauf ρ vorkommen. Ein Streett-Automat akzeptiert, wenn

$$\forall i : inf(\rho) \cap B_i \neq \emptyset \Rightarrow inf(\rho) \cap G_i \neq \emptyset.$$

Für eine Implikation gilt

$$A \Rightarrow B \equiv \neg A \vee B$$

Damit folgt:

$$\forall i : inf(\rho) \cap B_i \neq \emptyset \Rightarrow inf(\rho) \cap G_i \neq \emptyset \equiv \forall i : \neg inf(\rho) \cap B_i \neq \emptyset \vee inf(\rho) \cap G_i \neq \emptyset$$

Da die Rabin-Akzeptanzbedingung genau die Negation der Streett-Bedingung sein soll, folgt für die Rabin-Bedingung:

$$\begin{aligned}
&\neg(\forall i : \neg inf(\rho) \cap B_i \neq \emptyset \vee inf(\rho) \cap G_i \neq \emptyset) \\
\equiv &\exists i : \neg(\neg inf(\rho) \cap B_i \neq \emptyset \vee inf(\rho) \cap G_i \neq \emptyset) \\
\equiv &\exists i : \neg\neg inf(\rho) \cap B_i \neq \emptyset \wedge \neg inf(\rho) \cap G_i \neq \emptyset \\
\equiv &\exists i : inf(\rho) \cap B_i \neq \emptyset \wedge \neg inf(\rho) \cap G_i \neq \emptyset \\
\equiv &\exists i : inf(\rho) \cap B_i \neq \emptyset \wedge inf(\rho) \cap G_i = \emptyset
\end{aligned}$$

□

Da es aus Gründen der Benennung mehr Sinn macht, wird die Rabin-Akzeptanzbedingung mit vertauschtem G_i und B_i benutzt, also

$$\exists i : inf(\rho) \cap G_i \neq \emptyset \wedge inf(\rho) \cap B_i = \emptyset.$$

Weiter werden für die Umwandlung nur 1-Rabin-Automaten, also Rabin-Automaten, deren Menge F nur ein Tupel (B,G) enthält, benötigt und daher wird im Folgenden statt B_i und G_i nur noch B und G geschrieben.

Der Unterschied zwischen 1-Rabin-Automaten und 1-Strett-Automaten liegt also nur darin, dass 1-Rabin-Automaten akzeptieren, wenn unendlich oft ein Zustand aus G, aber nur endlich oft ein Zustand aus B durchlaufen wird und 1-Streett-Automaten akzeptieren, wenn nur endlich oft ein Zustand aus B oder unendlich oft ein Zustand aus G gesehen wird.

3 Die Transformation in der Theorie

In diesem Kapitel wird das Kernthema dieser Arbeit behandelt. Es werden zuerst verschiedene Umwandlungen vorgestellt sowie deren Vor- und Nachteile aufgezeigt. Danach wird die, in dieser Arbeit benutzte, Transformation genauer beschrieben sowie ein Beispielautomat transformiert. Nach dem theoretischen Teil folgt dann die Beschreibung der in dem Programm genutzten Algorithmen und die Vorstellung des Java-Programms im Allgemeinen.

3.1 Verschiedene Transformationen

In diesem Bereich werden einige Umwandlungen und deren Vor- und Nachteile vorgestellt. Bei vielen von diesen wird versucht, nicht nur einen alternierenden 3-Paritätsautomaten (3-APA) umzuwandeln, sondern allgemein einen alternierenden Paritätsautomaten (APA). Dadurch erhöht sich die Komplexität erheblich und aus diesem Grund kamen auch viele Umwandlungen für die Implementierung nicht in Frage.

Einer der Ansätze für die Umwandlung von APAs zu nicht-deterministischen Büchi-Automaten (NBA) stammt von Christian Dax und Felix Klaedtke, er wird in [kla08] vorgestellt. Es wird versucht, diese Transformation mit Einführung eines zusätzlichen Eingabealphabetes und Komplementierung zu lösen. Das heißt, es wird zuerst aus dem APA $A = (Q, \Sigma, q_0, \delta_A, F)$ ein co-APA $B = (Q, \Sigma \times \Gamma, q_0, \delta_B, Q^\omega \backslash F)$ gemacht. Dieser soll dann durch Komplementierung direkt in einen NBA C umgewandelt werden, in welchem das Alphabet $\Sigma \times \Gamma$ wieder auf Σ abgebildet werden muss, um den gesuchten NBA zu erhalten. Das Problem ist allerdings, dass die Komplementierung von einem co-APA direkt zu einem NBA extrem schwierig ist, denn für die Komplementierung wird die „Safra construction“ benutzt, beschrieben in [saf88], bei deren Implementierung laut Kupfermann und Vardi in [kv05] schon einige renomierte Leute gescheitert sind.

Auch Kupfermann und Vardi selber haben eine Transformation beschrieben, die wieder auf eine andere Weise arbeitet. Sie versuchen durch wiederholtes aufrufen des selben Algorithmus Schritt für Schritt die Anzahl Paritäten des Paritätsautomaten auf Eins zu senken, um den entstandenen Automaten dann direkt in einen alternierenden Büchi-Automaten zu transformieren. Dabei wird dem APA dann eine zweite Menge von Zuständen hinzugefügt, dieser Automatentyp wird dann schwacher-Paritäts alternierender Automat genannt, die am Anfang leer ist und in jedem Durchlauf des Algorithmus weitere Zustände erhält, welche keine Parität haben. Das heißt, wenn als einzige Parität nur noch 1 vorhanden ist, so hat der Automat eine Menge mit Zuständen ohne Parität und eine Menge mit Zuständen mit Parität 1. Den alternierenden Büchi-Automaten (ABA) kann man dann direkt daraus ablesen.

Diese Idee wird in [kv98] beschrieben. Sie hat den Vorteil, dass sie getrennt die Transformation der Akzeptanzbedingung vornimmt und man danach einen Büchi-Automaten hat, bei dem man nur noch das Alternierende zum nicht-Deterministischen umformen muss. Dies sollte, aufgrund der getrennten Vorgehensweise, deutlich besser in einer Programmiersprache zu implementieren sein.

Der große Nachteil dieses Verfahrens ist aber, dass die Menge der Zustände des ABA, der sich aus dem APA ergibt, pro Parität in dem gegebenen APA exponentiell steigt. Bei der Umwandlung von dem ABA zu dem NBA steigt diese dann nochmals exponentiell. So erhält man schon bei einem Paritätsautomaten mit nur 3 Zuständen und 3 Paritäten erst einen ABA mit etwa 300 Zuständen und dann einen NBA mit, im schlechtesten Fall, etwa 2^{300} Zuständen. Dies würde aber den Speicher jedes Computers übersteigen, weshalb die Umwandlung für praktische Zwecke unbrauchbar ist.

Ein weiterer Ansatz stammt von César Sánchez, den er in [sf11] vorstellt. Diese Umwandlung ist speziell für die Transformation von 3-APAs zu einem NBA entworfen. César Sánchez versucht hierbei zuerst den alternierenden Paritätsautomaten in einen alternierenden Streett- oder Rabin-Automaten umzuwandeln, abhängig von der höchsten Parität in dem 3-APA, um von diesem dann direkt zu einem nicht-deterministischen Büchi-Automaten zu gelangen.

Mit Hilfe von César Sánchez's Ansatz wird genau diese Umwandlung und ihre Problematik in dieser Arbeit behandelt. Am Ende ist ein Programm entstanden, welches diese Transformation automatisiert vornimmt.

3.2 Die Transformation

Der in dieser Arbeit benutzte Ansatz für die Transformation stammt von César Sánchez aus dem Artikel [sf11] . Diese Transformation ist extra für die Umwandlung von 3-APAs zu NBAs kreiert worden.

3.2.1 Die grundsätzliche Idee

Die grundsätzliche Idee für diese Umwandlung liegt darin, den gegebenen 3-APA, je nachdem ob seine Zustände die Paritäten 0,1 und 2 oder 1,2 und 3 besitzen, in einen alternierenden Streett-Automaten (ASA) oder einen alternierenden Rabin-Automaten (ARA) umzuwandeln, die nur ein Tupel (B,G) besitzen. Die Unterscheidung der beiden Fälle wird deshalb gemacht, weil sich ein 3-APA mit den Paritäten 0,1 und 2 zu einem ASA mit nur einem Tupel (B,G) umwandeln lässt, nicht aber zu einem ARA mit nur einem solchen Tupel und gleicher Anzahl an Zuständen und Transitionen. Genau umgekehrt ist es mit dem Fall, bei dem der 3-APA die Paritäten 1,2 und 3 besitzt, denn dieser lässt sich nicht in einen ASA mit nur einem Tupel umwandeln, der auch die gleiche Anzahl an Zuständen und Transitionen hat, aber in einen ARA mit nur einem solchen Tupel.

Durch die Unterscheidung in die Fälle 0,1 und 2 und 1,2 und 3 als Paritäten sind alle möglichen 3-APAs abgedeckt, denn wären die beiden kleinsten oder größten Paritäten gerade oder ungerade so könnte man eine entfernen, da es dann keinen Unterschied machen würde, ob es diese gibt oder nicht.

Von dem entstandenen ASA oder ARA aus wird dann ein NBA erstellt, der die selbe Sprache akzeptiert. Dies wird über zwei verschiedene Wege gemacht, je nachdem, was für einen Ausgangsautomaten es gab.

3.2.2 APA zu ASA

Bei dieser Umwandlung wird ein APA mit den Paritäten 0,1 und 2 in einen ASA umgewandelt. Die Umwandlung von einem APA zu einem ASA ist sehr einfach. Dazu sehen wir uns zuerst die beiden Akzeptanzbedingungen an:

Ein APA akzeptiert ein Wort, wenn das Maximum der Paritäten der unendlich oft betrachteten Zustände gerade ist. Bei einem 3-APA mit den Paritäten 0,1 und 2 sieht man, dass er entweder akzeptiert, wenn er nur endlich oft Zustände mit der Parität 1 sieht oder wenn er unendlich oft Zustände mit der Parität 2 sieht. Ein ASA mit nur einem Tupel (B,G) hingegen akzeptiert ein Wort, wenn er entweder nur endlich oft Zustände aus der Menge B sieht oder unendlich oft Zustände aus der Menge G.

Es ist also leicht zu sehen, dass diese Umwandlung direkt geschehen kann, indem alle Zustände des 3-APA mit Parität 1 in die Menge B des ASA kommen und alle mit Parität 2 in die Menge G. Dieser Zwischenschritt ist aufgrund der Direktheit dieser Umwandlung nicht nötig, da man die gesamte Umwandlung auch von einem Paritätsautomaten aus durchführen könnte, allerdings ist es mit einem ASA deutlich leichter darzustellen.

Bei dieser Umwandlung entsteht noch keine Erhöhung der Anzahl der Zustände oder der Transitionen.

3.2.3 ASA zu NBA

In diesem Teil der Arbeit wird der erste Teil der Umwandlung beschrieben. Nachdem wir den 3-APA mit den Paritäten 0,1 und 2 in einen ASA umgewandelt haben, wollen wir diesen ASA nun in den gesuchten NBA umwandeln, der die selbe Sprache akzeptiert wie der 3-APA.

Sei $A = (Q_A, \Sigma_A, I_A, \delta_A, (B,G))$ ein ASA und n die Menge der Zustände in A, also $n = |Q_A|$. Es soll aus A nun ein NBA $N = (Q_N, \Sigma_N, I_N, \delta_N, F)$ konstruiert werden.

Es wird damit begonnen alle möglichen Zustände, also die Menge Q_N, des Büchiautomaten zu erstellen. Diese Zustände werden 4-Tupel (S,O,f,ok) sein. In den einzelnen Komponenten dieser Tupel werden sich alle nötigen Informationen gemerkt. Darunter fallen zum Einen die Möglichkeit des alternierenden Automaten, sich in mehreren Zuständen gleichzeitig zu befinden und zum Anderen die verschiedenen Paritäten des Paritätsautomaten.

In der Menge $S \subseteq Q_A$ eines Zustandes aus Q_N sind alle Zustände, in denen sich der APA gerade an der selben Stelle des Laufes, also auf der gleichen Höhe in einem Lauf-Baum des Laufes, befinden würde. $O \subseteq S$ ist dann eine Menge von Zuständen, in der alle Zustände aus S sind, für die noch keine zufrieden stellende Folge von Zuständen gefunden wurde. f ist eine Abbildung $f : S \to [2n]$, die jeden Zustand aus S auf eine Zahl zwischen 0 und $2n$ abbildet. Die Funktion f ist dafür da, um nicht-deterministisch Vorhersagen über den weiteren Verlauf des Pfades zu treffen. *ok* ist ein boolean, das angibt, ob der Zustand akzeptierend ist oder nicht.
Weiter gelten noch zwei Bedingungen:

- $SQ1$:
  Ist $q \in B$, so ist $f(q)$ gerade.
- $SQ2$:
  Wenn $O \neq \emptyset$ dann ist $ok = false$.

Die erste Bedingung ist dafür da, damit f bei einem Nachfolgezustand nie auf einen anderen Wert abbilden kann als bei dem Vorgänger, solange beide in B des ASA sind. Ist der Nachfolger nun nicht mehr böse, so kann, wie später erklärt wird, nicht-deterministisch für den Nachfolger entschieden werden, ob f auf den selben Wert oder auf einen kleineren Wert abbildet. Die zweite Bedingung sagt, dass es nur akzeptierende Zustände geben kann, bei denen O leer ist, bei denen also für jeden Zustand in S eine zufrieden stellende Folge von Zuständen gefunden wurde. Wichtig ist, dass diese Bedingung nicht sagt, dass jeder Zustand, in dem O leer ist, auch ein akzeptierender Zustand ist.

Die Menge der akzeptierenden Zustände ist nun, wie weiter oben beschrieben, die Menge aller Zustände $(S, O, f, true) \in Q_N$. Das heißt, es ist die Menge aller Zustände, in denen O leer und *ok* nicht-deterministisch *true* ist.

Es gelte $M \models I_A$. Die Startzustände sind alle Zustände (S, O, f, ok) für die gilt:

- $S = M$,
- $O = \{q \in M | q \notin G \text{ und } f(q) \text{ ist gerade}\}$,
- f bildet jeden Zustand beliebig auf $[2n]$ ab,
- $ok = false$.

Da I_A die positive boolsche Formel an Startzuständen in dem ASA ist, muss es in dem NBA möglich sein, nicht-deterministisch eine Menge an Zuständen des ASA auszuwählen, in denen der NBA gleichzeitig startet. O enthält dann alle $q \in S$, die nicht in G sind, aber für die $f(q)$ gerade ist, denn dann sind sie potentiell böse. f kann beliebig gewählt werden, damit kann dann nicht-deterministisch ausgewählt werden, auf welcher Ebene des NBA gestartet wird. *ok* ist zu Beginn immer $false$. Dies gibt aber keine Probleme mit der Akzeptanz, denn sollten alle Zustände in dem Startzustand eine gerade Parität haben und man kann mit einer Transition von ihnen zu sich selbst gehen, so würde man in dem NBA nicht-deterministisch einen akzeptierenden Zustand als Folgezustand auswählen können.

Die Transitionsfunktion δ_N ist eine Abbildung $\delta_N : Q_N \times \Sigma \to 2^{Q_N}$. Ein Zustand (S', O', f', ok') ist in $\delta_N((S, O, f, ok), a)$, wenn es für jedes $q \in S$ eine Menge $M_q \models \delta_A(q, a)$ gibt, so dass gilt:

- $SD1$:
$$S' = \cup_{q \in S} M_q.$$
S' besteht also aus einer Vereinigung von beliebigen M_q, wobei für jedes $q \in S$ genau ein M_q ausgewählt wird. Da es für ein q mehrere M_q geben kann, wird eines nicht-deterministisch ausgewählt, um den Folgezustand (S', O', f', ok') eines Zustandes zu bestimmen.

- $SD2$:
Sei $pred(p)$ definiert als
$$pred(p) = \{q \in S | p \in M_q\}.$$
Das heißt, $pred(p)$ enthält alle Vorgänger $q \in S$ von $p \in S'$ für die eine Nachfolgermenge M_q gewählt wurde, die p enthält.
Die Bedingung $SD2$ lautet dann:
$$f'(p) \leq \min\{f(q) | q \in pred(p) \backslash G\}.$$
$SD2$ sagt nun aus, dass nicht-deterministisch entschieden wird, ob die Funktion f' einen Zustand $p \in S'$ auf den gleichen oder einen kleineren Wert als das Minimum der Werte seiner Vorgänger abbildet. Wichtig ist, dass dieser Wert nie größer wird. Dadurch wird sichergestellt, dass der Automat nur endlich viele Zustände enthält und sich irgendwann ein Zyklus bildet. Gute Zustände aus der Menge G des ASA werden als Vorgänger ausgeklammert, da ihr Wert nicht wichtig ist, denn für einen guten Zustand muss nichts mehr entschieden werden. Sind alle Vorgänger aus G, so bildet $f'(p)$ wieder auf $2n$ ab.

- $SD3$:
$SD3$ beschreibt die Menge O' in dem Folgezustand (S', O', f', ok'). Sei hierzu im Folgenden $p \in S' \backslash G$. Dann muss für O' gelten:
  - Ist $ok = true$ und $f'(p)$ gerade, so ist $p \in O'$.
  - Ist $ok = false$ und für irgendein $q \in (pred(p) \cap O)$ gilt $f(q) = f'(p)$, dann ist $p \in O'$.

  Zunächst werden nur Zustände $p \in S' \backslash G$ genommen, denn ein Zustand in G des ASA kann nie in O' sein, da O' nur die Zustände enthalten soll, für die noch kein zufriedenstellender Weg gefunden wurde. War nun im vorherigen Zustand (S, O, f, ok) $ok = true$, so war dieser akzeptierend und $O = \emptyset$. Da Zustände $p \in S'$ nur in B des ASA gewesen sein können, wenn $f'(p)$ gerade ist (siehe $SQ1$), werden vorsichtshalber erstmal alle potentiell bösen Zustände in das neue O' getan. War $ok = false$ und es gab einen Vorgänger $q \in S$, der auch in O war, von $p \in S'$, bei dem $f(q) = f'(p)$ gilt, so ist auch $p \in O'$, denn der Automat hat geraten, dass noch kein zufriedenstellender Weg zu p gefunden wurde.

Um nun die Idee dieser Umwandlung noch etwas zu veranschaulichen, stelle man sich den Lauf des 3-APA als Lauf-Baum vor. Dieser verzweigt sich irgendwann mal und hat dann bei Tiefe x i Verzweigungen. Das heißt, der Automat durchläuft gleichzeitig i Pfade. Dieses Verhalten muss

nun irgendwie in dem NBA dargestellt werden, da dieser nicht alternierend ist und sich daher nicht in mehreren Zuständen gleichzeitig befinden kann. Außerdem hat der NBA nur zwei Arten von Zuständen, nicht wie ein Paritätsautomat drei. Haben wir nun in dem Lauf-Baum auf Tiefe x i Pfade. Der NBA merkt sich nun die i Zustände, in denen der ASA gerade gleichzeitig in diesem Lauf ist, in der Menge S. Die Menge O ist dann dafür da, um sich für jeden dieser i Pfade zu merken, ob dieser noch Betrachtungen der Folgezustände braucht, um sicher zu gehen, ob der Pfad bisher akzeptierend ist oder nicht. Ist O leer, so sind alle Pfade ok und damit kann es einen akzeptierenden Zustand geben. Danach wird dieser Check wieder zurückgesetzt und beginnt dann von neuem, bis wieder alle Pfade ok sind. Da die Pfade aber unendlich lang sind, kann man nie sagen, ob in diesem Pfad irgendwann immer wieder eine Folge von Zuständen gesehen wird, so dass alle Pfade ok sind. Dafür ist die Funktion f da. Durch sie wird nicht-deterministisch ein Zeitpunkt gewählt, ab dem der Automat für einen Pfad entscheidet, dass sich in diesem nichts mehr ändert. Da, wie in $SQ1$ beschrieben, f von einem bösen Zustand nur den Wert wechseln kann, wenn ein nicht-böser Zustand als Folgezustand gesehen wird, kann bei jedem danach gesehenen nicht-bösem Folgezustand geraten werden, ob der Pfad nun ok sein wird oder nicht.

Die Erhöhung der Zustände in dieser Umwandlung ist in $2^{O(n \log n)}$. Dies ergibt sich zum Einen durch das Entfernen des Alternierends, dessen Erhöhung der Zustände in 2^n liegt und dem Umwandeln der Akzeptanzbedingung. Die Erhöhung ist aber nicht doppelt-exponentiell, denn sie ist extra auf 3-APAs zugeschnitten und nutzt es aus, dass es nur maximal drei Paritäten gibt.

3.2.4 APA zu ARA

Ein APA mit den Paritäten 1,2 und 3 wird zunächst in einen ARA transformiert. Ähnlich wie die Transformation von einem 3-APA in einen ASA, ist auch die Umwandlung von einem 3-APA in einen ARA sehr direkt. Schauen wir uns hierzu wieder die beiden Akzeptanzbedingungen an:

Ein APA akzeptiert ein Wort, wenn das Maximum der Paritäten der unendlich oft betrachteten Zustände gerade ist. Bei einem 3-APA mit den Paritäten 1,2 und 3 sieht man, dass er nur dann akzeptiert, wenn er unendlich oft Zustände mit Parität 2 sieht aber nur endlich oft welche mit Parität 3. Ein ARA hingegen akzeptiert ein Wort, wenn er nur endlich oft Zustände aus der Menge B aber unendlich oft Zustände aus der Menge G sieht.

Es ist also leicht zu sehen, dass diese Umwandlung direkt geschehen kann, indem alle Zustände des 3-APA mit Parität 3 in die Menge B des ARA kommen und alle mit Parität 2 in die Menge G. Dieser Zwischenschritt ist aufgrund der Direktheit dieser Umwandlung nicht nötig, da man die gesamte Umwandlung auch von einem Paritätsautomaten aus durchführen könnte, allerdings ist es mit einem ARA deutlich leichter darzustellen.

Bei dieser Umwandlung entsteht, wie bei der Umwandlung von einem 3-APA zu einem ASA, noch keine Erhöhung der Anzahl der Zustände oder der Transitionen.

3.2.5 ARA zu NBA

In diesem Unterkapitel wird die Transformation von einem ARA, der vorher aus dem 3-APA konstruiert wurde, zu einem NBA beschrieben, der dann die selbe Sprache wie der 3-APA akzeptiert.

Sei $A = (Q_A, \Sigma_A, I_A, \delta_A, (B,G))$ ein ARA und n die Menge der Zustände in A, also $n = |Q_A|$. Es soll aus A nun ein NBA $N = (Q_N, \Sigma_N, I_N, \delta_N, F)$ konstruiert werden.

Es wird damit begonnen alle möglichen Zustände, also die Menge Q_N, des Büchi-Automaten zu erstellen. Diese Zustände werden 4-Tupel $(S, O, f, \mathit{phase})$ sein. In den einzelnen Komponenten dieser Tupel werden sich alle nötigen Informationen gemerkt. Darunter fallen zum Einen die Möglichkeit des alternierenden Automaten, sich in mehreren Zuständen gleichzeitig zu befinden und zum Anderen die verschiedenen Paritäten des Paritätsautomaten.

In der Menge $S \subseteq Q_A$ eines Zustandes aus Q_N sind immer alle Zustände, in denen sich der APA gerade an der selben Stelle des Laufes, also auf der gleichen Höhe in einem Lauf-Baum des Laufes, befinden würde. $O \subseteq S$ ist dann eine Menge von Zuständen, in der immer alle Zustände aus S sind, für die noch keine zufrieden stellende Folge von Zuständen gefunden wurde. f ist eine Abbildung $f : S \to [2n]$, die jeden Zustand aus S auf eine Zahl zwischen 0 und $2n$ abbildet. Die Funktion f ist dafür da, um nicht-deterministisch Vorhersagen über den weiteren Verlauf der Pfad zu treffen. *phase* ist eine Zahl aus $\{0,1,2\}$. Diese gibt wieder an, ob der Zustand akzeptierend sein kann oder ob dies noch nicht klar ist, da der Automat noch nicht weiß, ob wirklich nicht unendlich oft Zustände aus B aber unendlich oft Zustände aus G gesehen werden.
Weiter gelten noch 2 Bedingungen:

- $RQ1$:
  Ist $q \in B$, so ist $f(q)$ gerade.
- $RQ2$:
  Wenn $O \neq \emptyset$ dann ist $\mathit{phase} \neq 2$.

Die erste Bedingung ist dafür da, damit f bei einem Nachfolgezustand nie auf einen anderen Wert abbilden kann als bei dem Vorgänger, solange beide in B des ARA sind. Ist der Nachfolger nun nicht mehr böse, so kann, wie später erklärt wird, nicht-deterministisch für den Nachfolger entschieden werden, ob f auf den selben Wert oder auf einen kleineren Wert abbildet. Die zweite Bedingung sagt, dass es nur akzeptierende Zustände geben kann, bei denen O leer ist, bei denen also für jeden Zustand in S eine zufrieden stellende Folge von Zuständen gefunden wurde. Wichtig ist, dass diese Bedingung nicht sagt, dass jeder Zustand, in dem O leer ist, auch ein akzeptierender Zustand ist.

Die Menge der akzeptierenden Zustände ist nun, wie weiter oben beschrieben, die Menge aller Zustände $(S, O, f, 2) \in Q_N$. Das heißt, es ist die Menge aller Zustände, in denen O leer ist und *phase* nicht-deterministisch 2 ist.

Es gelte $M \models I_A$. Die Startzustände sind alle Zustände (S, O, f, ok) für die gilt:

- $S = M$,

- $O = M \backslash G$,
- f bildet jeden Zustand beliebig auf $[2n]$ ab,
- $phase = 0$.

Da I_A die positive boolsche Formel an Startzuständen in dem ARA ist, muss es in dem NBA möglich sein, nicht-deterministisch eine Menge an Zuständen des ARA auszuwählen, in denen der NBA gleichzeitig startet. O enthält dann alle $q \in S$, die nicht in G sind. Der Grund dafür ist, dass bei dem 3-APA mit den Paritäten 1,2 und 3 nur die mittlere Parität gut ist und damit jeder Zustand, der nicht in G ist, potentiell böse ist. f kann beliebig gewählt werden, damit kann dann nicht-deterministisch ausgewählt werden, auf welcher Ebene des NBA gestartet wird. *phase* ist zu Beginn 0. Daher muss die Phase bis 2 erstmal aufgebaut werden, damit es einen akzeptierenden Zustand gibt. Dadurch entstehen aber keine Probleme mit der Akzeptanz, denn sollte der Startzustand eine gerade Parität haben und man kann mit einer Transition von ihm zu sich selbst gehen, so würde man in dem NBA nicht-deterministisch über einen Zustand mit $phase = 1$ zu einem akzeptierenden kommen können.

Die Transitionsfunktion δ_N ist eine Abbildung $\delta_N : Q_N \times \Sigma \to 2^{Q_N}$. Ein Zustand $(S', O', f', phase')$ ist in $\delta_N((S, O, f, phase), a)$, wenn es für jedes $q \in S$ eine Menge $M_q \models \delta_A(q, a)$ gibt, so dass gilt:

- $RD1$:
  $$S' = \cup_{q \in S} M_q.$$
  S' besteht also aus einer Vereinigung von beliebigen M_q, wobei für jedes $q \in S$ genau ein M_q ausgewählt wird. Da es für ein q mehrere M_q geben kann, wird eines nicht-deterministisch ausgewählt, um den Folgezustand $(S', O', f', phase')$ eines Zustandes zu bestimmen.
- $RD2$:
  Sei $pred(p)$ definiert als
  $$pred(p) = \{q \in S | p \in M_q\}.$$
  Das heißt, $pred(p)$ enthält alle Vorgänger $q \in S$ von $p \in S'$ für die eine Nachfolgermenge M_q gewählt wurde, die p enthält.
  Die Bedingung $RD2$ lautet dann:
  $$f'(p) \leq \min\{f(q) | q \in pred(p)\}.$$
  $RD2$ sagt nun aus, dass nicht-deterministisch entschieden wird, ob die Funktion f' einen Zustand $p \in S'$ auf den gleichen oder einen kleineren Wert als das Minimum der Werte seiner Vorgänger abbildet. Wichtig ist, dass dieser Wert nie größer wird. Dadurch wird sichergestellt, dass der Automat nur endlich viele Zustände enthält und irgendwann sich ein Zyklus bildet. Gute Zustände aus der Menge G des ARA werden hier als Vorgänger nicht ausgeklammert, denn die guten Zustände haben bei einem ARA keine höhere Parität als die bösen und damit ist das Sehen eines guten Zustandes kein automatisches Kriterium für Akzeptanz.
- $RD3$:
  Diese Bedingung beschreibt die Entstehung von $phase'$. Es gilt:

- Wenn $O = \emptyset$ und $phase = 0$, dann ist $phase' \in \{0,1\}$.
- Wenn $O = \emptyset$ und $phase = 1$, dann ist $phase' \in \{1,2\}$.
- Wenn $O \neq \emptyset$, dann ist $phase' = phase$.
- Wenn $phase = 2$, dann ist $phase' = 0$.

*RD*3 stellt zum Einen sicher, dass die Phase nicht gewechselt werden kann, wenn noch nicht für jeden Zustand ein zufriedenstellender Weg gefunden wurde. Erst wenn dies passiert ist, kann nicht-deterministisch die Phase erhöht werden, bis der Nachfolgezustand bei $phase' = 2$ akzeptierend ist. War ein Zustand akzeptierend, so beginnt die nächste Phase wieder bei 0. Dies stellt einen akzeptierenden Zyklus dar.

- *RD*4 :
  *RD*4 beschreibt die Menge O' in dem Folgezustand (S', O', f', ok'). Sei hierzu im Folgenden $p \in S'$. Dann muss für O' gelten:
  - Ist $phase' = 0$, $phase = 2$ und $p \notin G$, dann ist $p \in O'$.
  - Ist $phase' = 0$, $phase = 0$, $p \notin G$ und $pred(p) \cap O \neq \emptyset$, dann ist $p \in O'$.
  - Ist $phase' = 1$, $phase = 0$ und $f(p)$ gerade, dann ist $p \in O'$.
  - Ist $phase' = 1$, $phase = 1$ und es gibt ein $q \in pred(p) \cap O$, so dass gilt $f'(p) = f(q)$, dann ist $p \in O'$.
  - Trifft keine dieser Bedingungen zu, ist $O' = \emptyset$

  War $phase = 2$, so ist jedes $p \in S'$ auch in O', wenn es nicht in G ist, denn jeder dieser Zustände ist dann verdächtig, keinen zufriedenstellenden Weg darzustellen. Sind sowohl die vorherige als auch die folgende Phase 0, so bleibt p verdächtig, also $p \in O'$, wenn p nicht in G ist und es einen Vorgänger von p gab, der auch verdächtig war. Ist die nachfolgende Phase hingegen 1 und die vorherige 0, so ist p nur in O', wenn $f'(p)$ gerade ist. Denn dann könnte p in B des ARA gewesen sein. Sind beide Phasen 1, so ist p nur in O', wenn es ein $q \in S$ gibt, das Vorgänger von p ist und in O war und für das sich der abgebildete Wert nicht verändert hat. Bei dem letzten Fall, nämlich dass die Phase vorher 1 und nachher 2 ist, muss O' aufgrund von *RQ*2 leer sein, da sonst die Phase nicht 2 sein kann. Da dieser Folgezustand dann akzeptierend ist macht das auch Sinn, denn es gibt keinen verdächtigen Zustand mehr in O'.

Um die Idee dieser Umwandlung noch etwas zu veranschaulichen, stelle man sich, genau wie bei dem ASA, den Lauf des 3-APA als Lauf-Baum vor. Dieser verzweigt sich irgendwann mal und hat dann bei Tiefe x i Verzweigungen. Das heißt, der Automat durchläuft gleichzeitig i Pfade. Dieses Verhalten muss nun irgendwie in dem NBA dargestellt werden, da dieser nicht alternierend ist und sich daher nicht in mehreren Zuständen gleichzeitig befinden kann. Außerdem hat der NBA nur zwei Arten von Zuständen, nicht wie ein Paritätsautomat drei. Haben wir nun in dem Lauf-Baum auf Tiefe x i Pfade. Der NBA merkt sich nun die i Zustände, in denen der ARA gerade gleichzeitig in diesem Lauf ist, in der Menge S. Die Menge O ist dann dafür da, um sich für jeden dieser i Pfade zu merken, ob dieser noch Betrachtungen der Folgezustände braucht, um sicher zu gehen,

dass der Pfad bisher akzeptierend ist oder nicht. Ist O leer, so sind alle Pfade ok und damit kann es einen akzeptierenden Zustand geben. Danach wird dieser Check wieder zurückgesetzt und beginnt dann von neuem, bis wieder alle Pfade ok sind. Anders als bei der Umwandlung über einen ASA, sind bei der Transformation über einen ARA sowohl die Zustände mit Parität eins, als auch die mit Parität 3 nicht ok. Daher gibt es bei der Umwandlung über einen ARA statt dem boolean *ok*, welches nur zwei Werte annehmen kann, die Zahl *phase*, die 3 verschiedene Werte annehmen kann. Diese stellt sicher, dass ein Zustand in dem NBA auch wirklich nur dann akzeptierend ist, wenn in dem Pfad des Lauf-Baums des 3-APAs nur noch unendlich viele Zustände mit Parität zwei und nicht unendlich viele mit Parität 3 vermutet werden. Da die Pfade aber unendlich lang sind, kann man nie sagen, ob in diesem Pfad irgendwann immer wieder eine Folge von Zuständen gesehen wird, so dass alle Pfade ok sind. Dafür ist die Funktion f da. Durch sie wird nicht-deterministisch ein Zeitpunkt gewählt, ab dem der Automat für einen Pfad entscheidet, dass sich in diesem nichts mehr ändert. Da, wie in $RQ1$ beschrieben, f von einem bösen Zustand nur den Wert wechseln kann, wenn ein nicht-böser Zustand als Folgezustand gesehen wird, kann bei jedem danach gesehenen nicht-bösem Folgezustand geraten werden, ob der Pfad nun ok sein wird oder nicht.

Die Erhöhung der Zustände in dieser Umwandlung ist, genau wie bei der Umwandlung über einen ASA, in $2^{O(n \log n)}$. Dies ergibt sich zum Einen durch das Entfernen des Alternierends, dessen Erhöhung der Zustände in 2^n liegt und dem Umwandeln der Akzeptanzbedingung. Die Erhöhung ist aber nicht doppelt-exponentiell, denn sie ist extra auf 3-APAs zugeschnitten und nutzt es aus, dass es nur maximal drei Paritäten gibt.

3.3 Beispiele

Jetzt werden einige Ausschnitte von Umwandlungen als Beispiele gegeben. Diese beziehen sich einmal auf die Umwandlung über einen ASA und einmal über einen ARA. Dabei sollen die wichtigsten Teile dieser Umwandlungen verdeutlicht werden.

3.3.1 Beispiel: APA zu NBA über ASA

Zuerst wird nun in Abbildung 3.1 die Umwandlung von einem APA in einen ASA noch einmal dargestellt. Wie zuvor beschrieben sind alle Zustände mit Parität 1 in der Menge B des ASA und alle Zustände mit Parität zwei in der Menge G.

Die Transformation von dem ASA zu dem NBA wird anhand von Ausschnitten erklärt, da es aufgrund der Anzahl der Zustände des NBAs nicht möglich ist, in dieser Arbeit ein komplettes Beispiel einer Umwandlung zu geben.

Starte der ASA mit 3 Zuständen, also $n = 3$ in nur einem Zustand q_0 das heißt, q_0 ist das einzige minimale Modell. Der Startzustand des NBA würde nun nicht-deterministisch aus allen Zuständen $(\{q_0\}, O, f, false)$ ausgewählt werden, wobei f beliebig auf einen Wert zwischen 0 und 6 abbildet. O hängt dann von f ab. Bildet f von q_0 auf einen geraden Wert ab, so ist q_0 auch in O, sonst nicht.

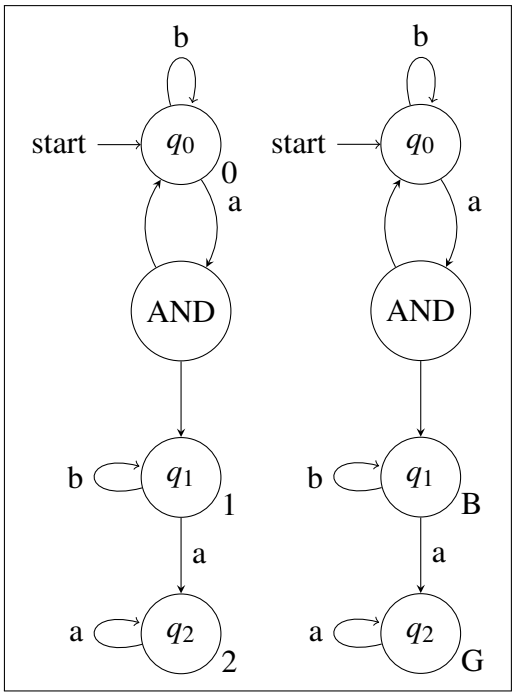

Abbildung 3.1: Ein Beispiel für eine Umwandlung von einem APA (links) zu einem ASA (rechts). Die Paritäten bzw. die Mengen von guten und bösen Zuständen des ASA, in denen die Zustände sind, stehen an den Zuständen

Um nun einen Folgezustand eines Zustandes des NBAs zu bestimmen, sucht man für jeden Zustand in der Menge S ein minimales Modell der Folgezustände in dem ASA. Dabei kann man noch nicht-deterministisch für jeden Folgezustand entscheiden, ob f' für einen Folgezustand auf eins weniger abbildet als f für den Vorgänger oder nicht. Hierbei muss man allerdings beachten, dass f' nicht unter 0 sinken kann und wenn der Folgezustand in B des ASA ist, so muss f' für diesen Zustand auf einen geraden Wert abbilden. Je nachdem, wie das vorherige O war und wie f' gewählt wird, ergibt sich O'. Ist O' leer, so kann nicht-deterministisch entschieden werden, dass $ok' = true$ gilt. Wird dies gemacht, so ist der Folgezustand akzeptierend.

Nehmen wir hierzu den ASA aus 3.2 als einen Ausschnitt des ASA, der umgewandelt werden soll. Sei dieser gerade gleichzeitig in den Zuständen q_0, q_3, q_5 und q_7. Gehen wir davon aus, der aktuelle Zustand des NBA ist

$$(\{q_0,q_3,q_5,q_7\},\{q_3,q_5\},\{(q_0,3),(q_3,4),(q_5,2),(q_7,0)\},false).$$

Liest man nun das nächste Zeichen, so könnte der NBA in folgende Zustände wechseln:

1. $(\{q_1,q_2,q_4,q_6,q_8\},\{q_4,q_6\},\{(q_1,3),(q_2,2),(q_4,4),(q_6,2),(q_8,0)\},false)$
2. $(\{q_1,q_2,q_4,q_6,q_8\},\{q_4,q_6\},\{(q_1,2),(q_2,2),(q_4,4),(q_6,2),(q_8,0)\},false)$
3. $(\{q_1,q_2,q_4,q_6,q_8\},\{q_6\},\{(q_1,3),(q_2,2),(q_4,3),(q_6,2),(q_8,0)\},false)$

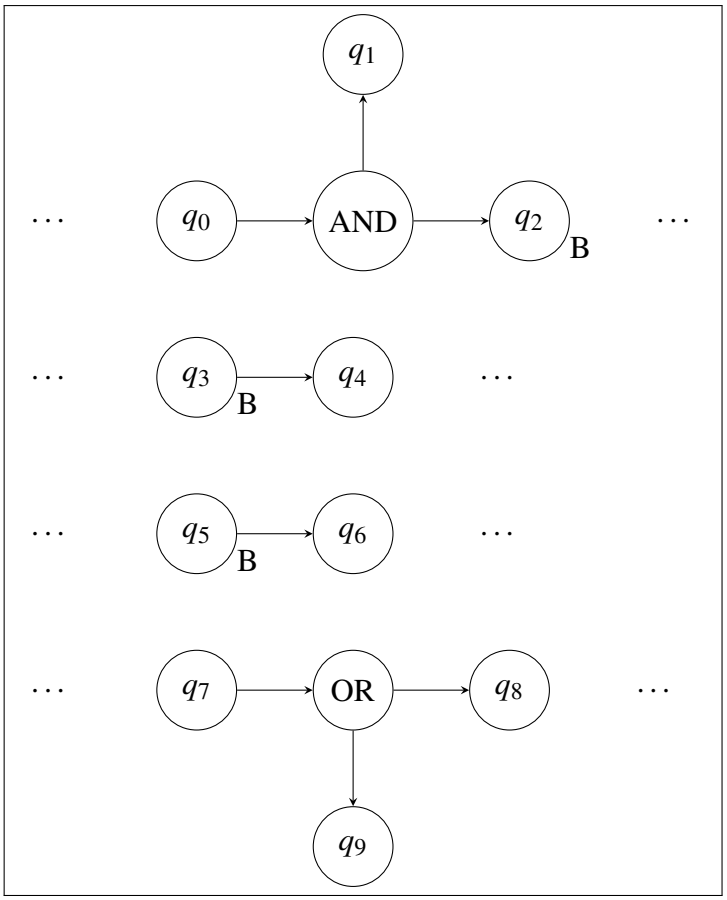

Abbildung 3.2: Ein Beispiel für einen ASA, an dem die Umwandlung zu einem NBA erklärt wird. Die Mengen, in denen die Zustände sind, stehen an den Zuständen

4. $(\{q_1,q_2,q_4,q_6,q_8\},\{q_4\},\{(q_1,3),(q_2,2),(q_4,4),(q_6,1),(q_8,0)\},false)$
5. $(\{q_1,q_2,q_4,q_6,q_8\},\{q_6\},\{(q_1,2),(q_2,2),(q_4,3),(q_6,2),(q_8,0)\},false)$
6. $(\{q_1,q_2,q_4,q_6,q_8\},\{q_4\},\{(q_1,2),(q_2,2),(q_4,4),(q_6,1),(q_8,0)\},false)$
7. $(\{q_1,q_2,q_4,q_6,q_8\},\emptyset,\{(q_1,3),(q_2,2),(q_4,3),(q_6,1),(q_8,0)\},false)$
8. $(\{q_1,q_2,q_4,q_6,q_8\},\emptyset,\{(q_1,2),(q_2,2),(q_4,3),(q_6,1),(q_9,0)\},false)$
9. $(\{q_1,q_2,q_4,q_6,q_8\},\emptyset,\{(q_1,3),(q_2,2),(q_4,3),(q_6,1),(q_8,0)\},true)$
10. $(\{q_1,q_2,q_4,q_6,q_8\},\emptyset,\{(q_1,2),(q_2,2),(q_4,3),(q_6,1),(q_9,0)\},true)$
11. $(\{q_1,q_2,q_4,q_6,q_9\},\{q_4,q_6\},\{(q_1,3),(q_2,2),(q_4,4),(q_6,2),(q_9,0)\},false)$
12. $(\{q_1,q_2,q_4,q_6,q_9\},\{q_4,q_6\},\{(q_1,2),(q_2,2),(q_4,4),(q_6,2),(q_9,0)\},false)$
13. $(\{q_1,q_2,q_4,q_6,q_9\},\{q_6\},\{(q_1,3),(q_2,2),(q_4,3),(q_6,2),(q_9,0)\},false)$
14. $(\{q_1,q_2,q_4,q_6,q_9\},\{q_4\},\{(q_1,3),(q_2,2),(q_4,4),(q_6,1),(q_9,0)\},false)$
15. $(\{q_1,q_2,q_4,q_6,q_9\},\{q_6\},\{(q_1,2),(q_2,2),(q_4,3),(q_6,2),(q_9,0)\},false)$
16. $(\{q_1,q_2,q_4,q_6,q_9\},\{q_4\},\{(q_1,2),(q_2,2),(q_4,4),(q_6,1),(q_9,0)\},false)$
17. $(\{q_1,q_2,q_4,q_6,q_9\},\emptyset,\{(q_1,3),(q_2,2),(q_4,3),(q_6,1),(q_9,0)\},false)$
18. $(\{q_1,q_2,q_4,q_6,q_9\},\emptyset,\{(q_1,2),(q_2,2),(q_4,3),(q_6,1),(q_9,0)\},false)$
19. $(\{q_1,q_2,q_4,q_6,q_9\},\emptyset,\{(q_1,3),(q_2,2),(q_4,3),(q_6,1),(q_9,0)\},true)$
20. $(\{q_1,q_2,q_4,q_6,q_9\},\emptyset,\{(q_1,2),(q_2,2),(q_4,3),(q_6,1),(q_9,0)\},true)$

Man sieht, dass es sehr viele nicht-deterministische Entscheidungen des Automaten gibt. Hierbei ist der Zeitpunkt, wann f' für einen Zustand auf einen geringeren Wert abbildet als f für die Vorgänger des Zustandes sehr wichtig, denn, zum Beispiel, wenn f auf 0 abbildet und der Zustand noch in O ist, so muss irgendwann ein Zustand aus G des ASA gesehen werden. Wie man bei 1., 2., 10. und 11. sieht, können die Zustände q_4 und q_6, wie ihre Vorgänger, böse bleiben oder nicht, wie in 3. bis 10. oder 13. bis 20.. Bleiben sie nicht böse, so hat der Automat nicht-deterministisch geraten, dass erstmal keine weiteren bösen Zustände gesehen werden. Dies kann er bei fast jedem Zustandsübergang machen. Ist O' leer, kann dann entschieden werden, ob $ok = true$ oder $ok = false$.

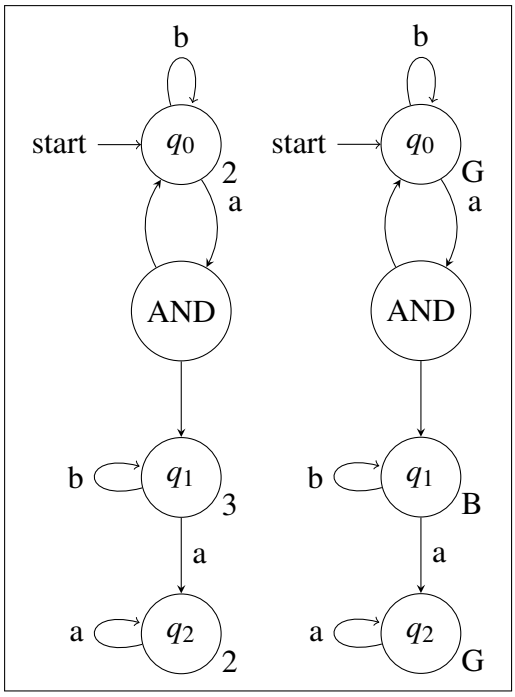

Abbildung 3.3: Ein Beispiel für eine Umwandlung von einem APA (links) zu einem ARA (rechts). Die Paritäten bzw. die Mengen von guten und bösen Zuständen des ARA, in denen die Zustände sind, stehen an den Zuständen

3.3.2 Beispiel: APA zu NBA über ARA

Zuerst wird nun in Abbildung 3.3 die Umwandlung von einem APA in einen ARA noch einmal dargestellt. Wie zuvor beschrieben sind alle Zustände mit Parität drei in der Menge B des ARA und alle Zustände mit Parität zwei in der Menge G.

Die Transformation von dem ARA zu dem NBA wird anhand von Ausschnitten erklärt, da es aufgrund der Anzahl der Zustände des NBAs nicht möglich ist, in dieser Arbeit ein komplettes Beispiel einer Umwandlung zu geben.

Starte der ARA mit 3 Zuständen, also $n = 3$ in nur einem Zustand q_0 das heißt, q_0 ist das einzige minimale Modell. Der Startzustand des NBA würde nun nicht-deterministisch aus allen Zuständen $(\{q_0\}, O, f, 0)$ ausgewählt werden, wobei f beliebig auf einen Wert zwischen 0 und 6 abbildet. O hängt dann von f ab. Bildet f von q_0 auf einen geraden Wert ab, so ist q_0 auch in O, sonst nicht.

Um nun einen Folgezustand eines Zustandes des NBAs zu bestimmen, sucht man für jeden Zustand in der Menge S ein minimales Modell der Folgezustände in dem ARA. Dabei kann man noch nicht-deterministisch für jeden Folgezustand entscheiden, ob f' für den Folgezustand auf einen geringeren Wert abbildet als f für den Vorgänger oder nicht. Hierbei muss man allerdings beachten, dass f' nicht unter 0 sinken kann und wenn der Folgezustand in B des ARA ist, so muss f' gerade sein. Je nachdem, wie das vorherige O war und wie f' gewählt wird, ergibt sich O'. Ist O' leer, so

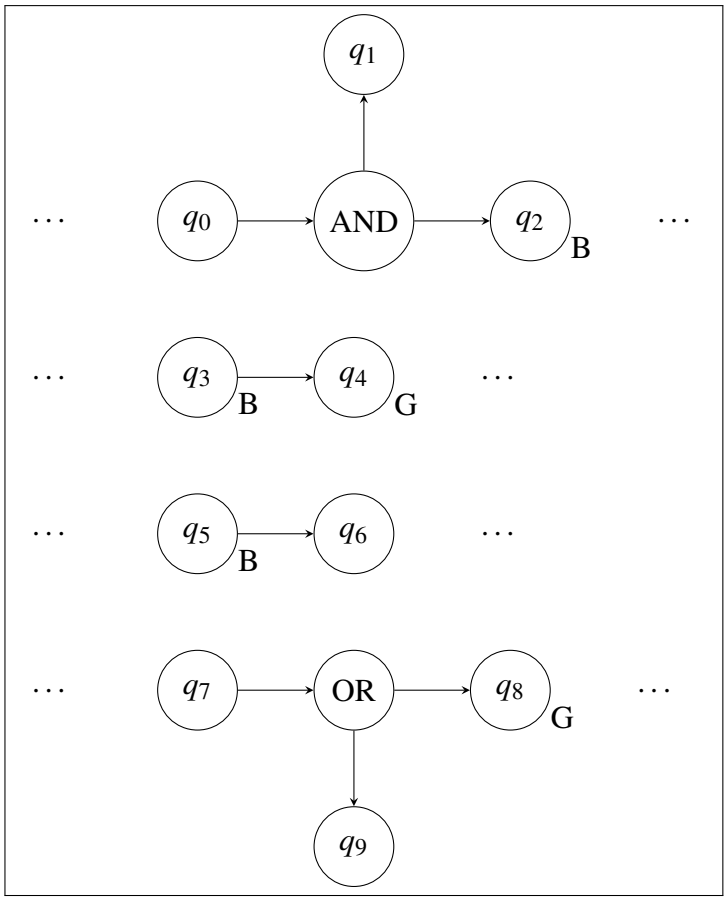

Abbildung 3.4: Ein Beispiel für einen ARA, an dem die Umwandlung zu einem NBA erklärt wird. Die Mengen, in denen die Zustände sind, stehen an den Zuständen

kann nicht-deterministisch entschieden werden, dass *phase* inkrementiert wird. Wird dies gemacht, so ist der Folgezustand akzeptierend.

Nehmen wir hierzu den ARA aus Abbildung 3.4 als einen Ausschnitt des ARA, der umgewandelt werden soll. Sei dieser gerade gleichzeitig in den Zuständen q_0, q_3, q_5 und q_7. Gehen wir davon aus, der aktuelle Zustand des NBA ist

$$(\{q_0, q_3, q_5, q_7\}, \{q_3, q_5\}, \{(q_0, 3), (q_3, 4), (q_5, 2), (q_7, 0)\}, 1).$$

Liest man nun das nächste Zeichen, so könnte der NBA in folgende Zustände wechseln:

1. $(\{q_1, q_2, q_4, q_6, q_8\}, \{q_4, q_6\}, \{(q_1, 3), (q_2, 2), (q_4, 4), (q_6, 2), (q_8, 0)\}, 1)$
2. $(\{q_1, q_2, q_4, q_6, q_8\}, \{q_4, q_6\}, \{(q_1, 2), (q_2, 2), (q_4, 4), (q_6, 2), (q_8, 0)\}, 1)$
3. $(\{q_1, q_2, q_4, q_6, q_8\}, \{q_6\}, \{(q_1, 3), (q_2, 2), (q_4, 3), (q_6, 2), (q_8, 0)\}, 1)$
4. $(\{q_1, q_2, q_4, q_6, q_8\}, \{q_4\}, \{(q_1, 3), (q_2, 2), (q_4, 4), (q_6, 1), (q_8, 0)\}, 1)$
5. $(\{q_1, q_2, q_4, q_6, q_8\}, \{q_6\}, \{(q_1, 2), (q_2, 2), (q_4, 3), (q_6, 2), (q_8, 0)\}, 1)$

6. $(\{q_1,q_2,q_4,q_6,q_8\},\{q_4\},\{(q_1,2),(q_2,2),(q_4,4),(q_6,1),(q_8,0)\},1)$
7. $(\{q_1,q_2,q_4,q_6,q_8\},\emptyset,\{(q_1,3),(q_2,2),(q_4,3),(q_6,1),(q_8,0)\},1)$
8. $(\{q_1,q_2,q_4,q_6,q_8\},\emptyset,\{(q_1,2),(q_2,2),(q_4,3),(q_6,1),(q_9,0)\},1)$
9. $(\{q_1,q_2,q_4,q_6,q_8\},\emptyset,\{(q_1,3),(q_2,2),(q_4,3),(q_6,1),(q_8,0)\},2)$
10. $(\{q_1,q_2,q_4,q_6,q_8\},\emptyset,\{(q_1,2),(q_2,2),(q_4,3),(q_6,1),(q_9,0)\},2)$
11. $(\{q_1,q_2,q_4,q_6,q_9\},\{q_4,q_6\},\{(q_1,3),(q_2,2),(q_4,4),(q_6,2),(q_9,0)\},1)$
12. $(\{q_1,q_2,q_4,q_6,q_9\},\{q_4,q_6\},\{(q_1,2),(q_2,2),(q_4,4),(q_6,2),(q_9,0)\},1)$
13. $(\{q_1,q_2,q_4,q_6,q_9\},\{q_6\},\{(q_1,3),(q_2,2),(q_4,3),(q_6,2),(q_9,0)\},1)$
14. $(\{q_1,q_2,q_4,q_6,q_9\},\{q_4\},\{(q_1,3),(q_2,2),(q_4,4),(q_6,1),(q_9,0)\},1)$
15. $(\{q_1,q_2,q_4,q_6,q_9\},\{q_6\},\{(q_1,2),(q_2,2),(q_4,3),(q_6,2),(q_9,0)\},1)$
16. $(\{q_1,q_2,q_4,q_6,q_9\},\{q_4\},\{(q_1,2),(q_2,2),(q_4,4),(q_6,1),(q_9,0)\},1)$
17. $(\{q_1,q_2,q_4,q_6,q_9\},\emptyset,\{(q_1,3),(q_2,2),(q_4,3),(q_6,1),(q_9,0)\},1)$
18. $(\{q_1,q_2,q_4,q_6,q_9\},\emptyset,\{(q_1,2),(q_2,2),(q_4,3),(q_6,1),(q_9,0)\},1)$
19. $(\{q_1,q_2,q_4,q_6,q_9\},\emptyset,\{(q_1,3),(q_2,2),(q_4,3),(q_6,1),(q_9,0)\},2)$
20. $(\{q_1,q_2,q_4,q_6,q_9\},\emptyset,\{(q_1,2),(q_2,2),(q_4,3),(q_6,1),(q_9,0)\},2)$

Man sieht, dass es sehr viele nicht-deterministische Entscheidungen des Automaten gibt. Hierbei ist der Zeitpunkt, wann f' für einen Zustand auf einen geringeren Wert abbildet als f für die Vorgänger des Zustandes sehr wichtig, denn, zum Beispiel, wenn f auf 0 abbildet und der Zustand noch in O ist, so kann nicht mehr akzeptiert werden, da dieser dann nicht mehr aus O verschwinden kann. Wie man bei 1., 2., 10. und 11. sieht, können die Zustände q_4 und q_6, wie ihre Vorgänger, böse bleiben oder nicht, wie in 3. bis 10. oder 13. bis 20.. Bleiben sie nicht böse, so hat der Automat nicht-deterministisch geraten, dass erstmal keine weiteren bösen Zustände gesehen werden. Dies kann er bei fast jedem Zustandsübergang machen. Ist O' leer, kann dann entschieden werden, ob $phase = 2$ oder $phase = 1$ ist.

4 Die Transformation als Javaprogramm

In diesem Teil der Arbeit wird das Javaprogramm zur automatisierten Umwandlung von einem APA zu einem NBA vorgestellt. Dabei wird genauer auf das Ein- und Ausgabeformat eingegangen, die genutzten Algorithmen und Datenstrukturen erklärt und die Minimierungsverfahren für den NBA vorgestellt.

4.1 Einleitung

Das Programm, welches während dieser Arbeit entstand, wurde in Java geschrieben. Die Wahl des Programmiersprache fiel wegen der Plattformunabhängigkeit und der großen Verbreitung auf Java Version 1.6.0. Es wurden keine zusätzlichen Tools genutzt und auch keine Bibliotheken, wodurch das Programm keine Abhängigkeiten besitzt. Das Programm ist ein Eclipse-Projekt und kann daher unter anderem mit Eclipse erstellt werden.

Weiterhin besteht es aus 22 Klassen mit insgesamt 95 Methoden und etwa 2500 Zeilen Code.

4.2 Ein- und Ausgabeformat

Die Eingabe geschieht in einem Format, dass mit Malte Schmitz zusammen erstellt wurde, um die Schnittstelle unserer Programme zu bilden. Siehe [sch11]. Die Ausgabe ist im Dotty[1]-Format, damit der NBA direkt dargestellt werden kann.

Das Eingabeformat wird an dem folgendem Beispiel erklärt:

```
apw {
alphabet = ["a", "b", "c"]
states = [q0: 0, q1: 0, q2: 1, q3: 2]
start = q1 and q2
delta(q0, "a") = (q1 and q2) or q0
delta(q1, "c") = q3
delta(q2, "c") = q3
delta(q3, "b") = q2
}
```

[1] http://www.graphviz.org/doc/info/lang.html

Ein APA beginnt mit dem Schlüsselwort apw (alternating parity-automaton on words). Die Parität der Zustände wird jeweils hinter den Zuständen angegeben. Diese muss bei jedem Zustand vorhanden sein. Die Zustände werden immer von $q0$ an durchnummeriert, daher, bei drei Zuständen gibt es $q0$, $q1$ und $q2$. „start“ und „delta“ bilden auf positive boolsche Formeln ab, um die Startzustände bzw. die Endzustände einer Transition zu beschreiben. Diese bestehen aus Zuständen, die durch die Schlüsselwörter „and“ oder „or“ zu positiven boolschen Formeln zusammengesetzt werden können. Dabei hat „and“ eine engere Bindung als „or“ und es sind Klammern erlaubt, allerdings darf die ganze Formel nicht umklammert werden.

Die Elemente des Alphabetes können aus einem oder mehreren Zeichen bestehen. Außerdem ist ein „?“ erlaubt, welches angibt, dass eine Transition bei allen Eingaben genommen wird. ε-Transitionen sind nicht erlaubt, da die benutzte Umwandlung für diese nicht funktioniert.

Bei der Ausgabe wird im Dotty-Format ein NBA dargestellt. An dem folgenden Beispiel wird das Ausgabeformat erklärt:

```
digraph finite_state_machine {
rankdir=LR;
size="200"
node [shape = doublecircle]; 0 2 4 8 10 12;
node [shape = circle]; 1 3 5 6 7 9 11 13 14 15;
node [shape = none, label = ""]; F0;
F0 -> 9 [ label = "start" ];
node [shape = none, label = ""]; F1;
F1 -> 14 [ label = "start" ];
node [shape = none, label = ""]; F2;
F2 -> 13 [ label = "start" ];
node [shape = none, label = ""]; F3;
F3 -> 15 [ label = "start" ];
0 -> 8 [ label = "b" ];
(...)
2 -> 8 [ label = "b" ];
2 -> 9 [ label = "b" ];
2 -> 14 [ label = "b" ];
(...)
9 -> 1 [ label = "a" ];
9 -> 0 [ label = "a" ];
10 -> 1 [ label = "a" ];
10 -> 6 [ label = "a" ];
(...)
15 -> 7 [ label = "a" ];
15 -> 4 [ label = "a" ];
}
```

Die ersten drei Zeilen bezeichnen nur, dass es ein Automat wird, dass dieser von links nach rechts gezeichnet wird und die Größe des Bildes. In der vierten Zeile werden die akzeptierenden Zustände benannt und in der fünften Zeile die nicht akzeptierenden. Die Zustände sind immer von 0

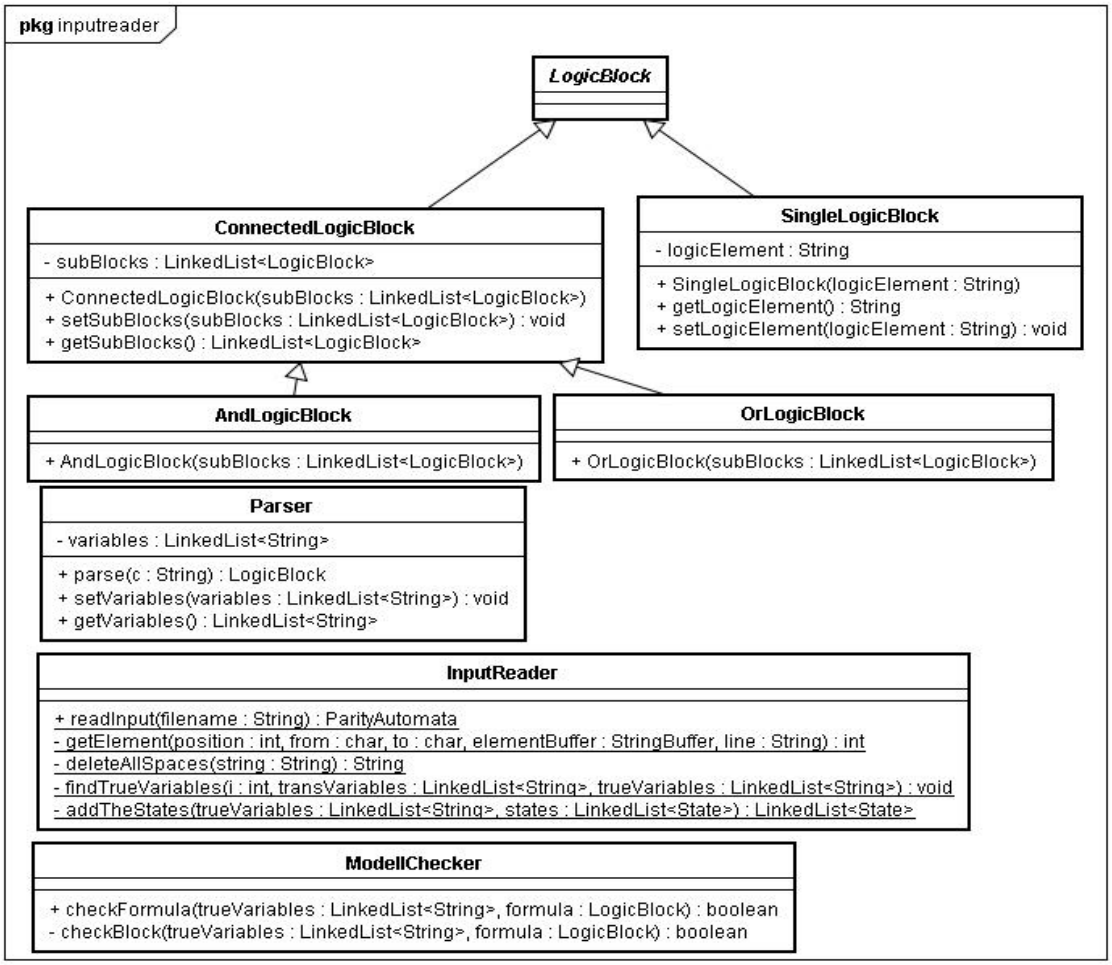

Abbildung 4.1: Das Paket Inputreader als Klassendiagramm.

an durchnummeriert. Danach folgen die Startpfeile zu den Startzuständen jeweils zweizeilig. Als nächstes sind noch die Transitionen mit ihrer Eingabe beschrieben.

4.3 Aufbau, Algorithmen und Datenstrukturen

Zuerst einmal muss die Eingabe geparst werden. Dies geschieht in der Klasse InputReader. Dabei sind die Zustände sowie ihre Paritäten leicht ablesbar. Genauso ist es mit dem Alphabet und der höchsten Parität der Zustände. Die Schwierigkeit liegt im Parsen der Transitionen und der Startzustände.

Um die Startzustände oder eine Transition zu parsen, wird zuerst der Teil der Zeile gelesen, der die positive boolsche Formel enthält. Dieser String wird dann dem Parser übergeben, der für diese Transition die positive boolsche Formel als Baum zurückgibt. Dieser Baum besteht dann aus

```
boolean checkBlock(List trueVariables, LogicBlock formula){
   if (formula ist ein SingleLogicBlock){
      if (Der Wert von formula ist in trueVariables){
         return true;
      }
      return false;
   } else if (formula ist ein OrLogicBlock){
      for (alle subBlocks block von formula){
         if(checkBlock(trueVariables, block)){
            // Ein "Oder" ist wahr, wenn ein Unterblock wahr ist.
            return true;
         }
      }
      return false;
   } else {
      for (alle subBlocks block von formula){
         if(!checkBlock(trueVariables, block)){
            // Ein "Und" ist nicht wahr, falls
            // ein Unterblock nicht wahr ist.
            return false;
         }
      }
      return true;
   }
}
```

Listing 4.1: Der ModellChecker in Pseudocode.

Objekten der Klassen, die von „Logic Block“ erben (siehe Abbildung 4.1). Hierbei stellen Or- und AndLogicBlock „or“ und „and“ dar und daher haben sie nur eine Liste aus Kindern. SingleLogicBlocks sind dann die Variablen, die dort in dem LogicElement als String stehen. Weiter liest und merkt sich der Parser alle vorkommenden Zustände. Nun wird die Potenzmenge der vorkommenden Zustände bestimmt und mit jedem Element der Potenzmenge getestet, ob dieses ein Modell für die positive boolsche Formel ist. Dies geschieht mit Hilfe des ModellCheckers. Dieser bekommt die Wurzel des Formel-Baumes und alle Variablen übergeben, die wahr sind. Dann wertet er rekursiv den Formel-Baum aus, bis feststeht, welchen Wert die Wurzel hat, und dieser Wert wird dann zurückgegeben. Dies wird in dem Listing 4.1 in Pseudocode dargestellt.

Schaut man sich die vorher beschriebene Umwandlung genau an, so sieht man, dass von den Endzuständen der Transitionen bzw. der Formel der Startzustände des APA immer nur die minimalen Modelle gebraucht werden, von denen eines nicht-deterministisch als Folgezustände bzw. Startzustand ausgewählt wird. Daher wird, wenn der ModellChecker „true“ zurückgibt, eine neue Transition mit den Zuständen als Folgezustände erstellt bzw. ein neuer Startzustand zu der Liste der Startzustände des NBAs hinzugefügt. Das heißt, es wird nicht die positive boolsche Formel als Folgezustände in der Transition gespeichert, sondern diese wird gleich ausgewertet und es wird für

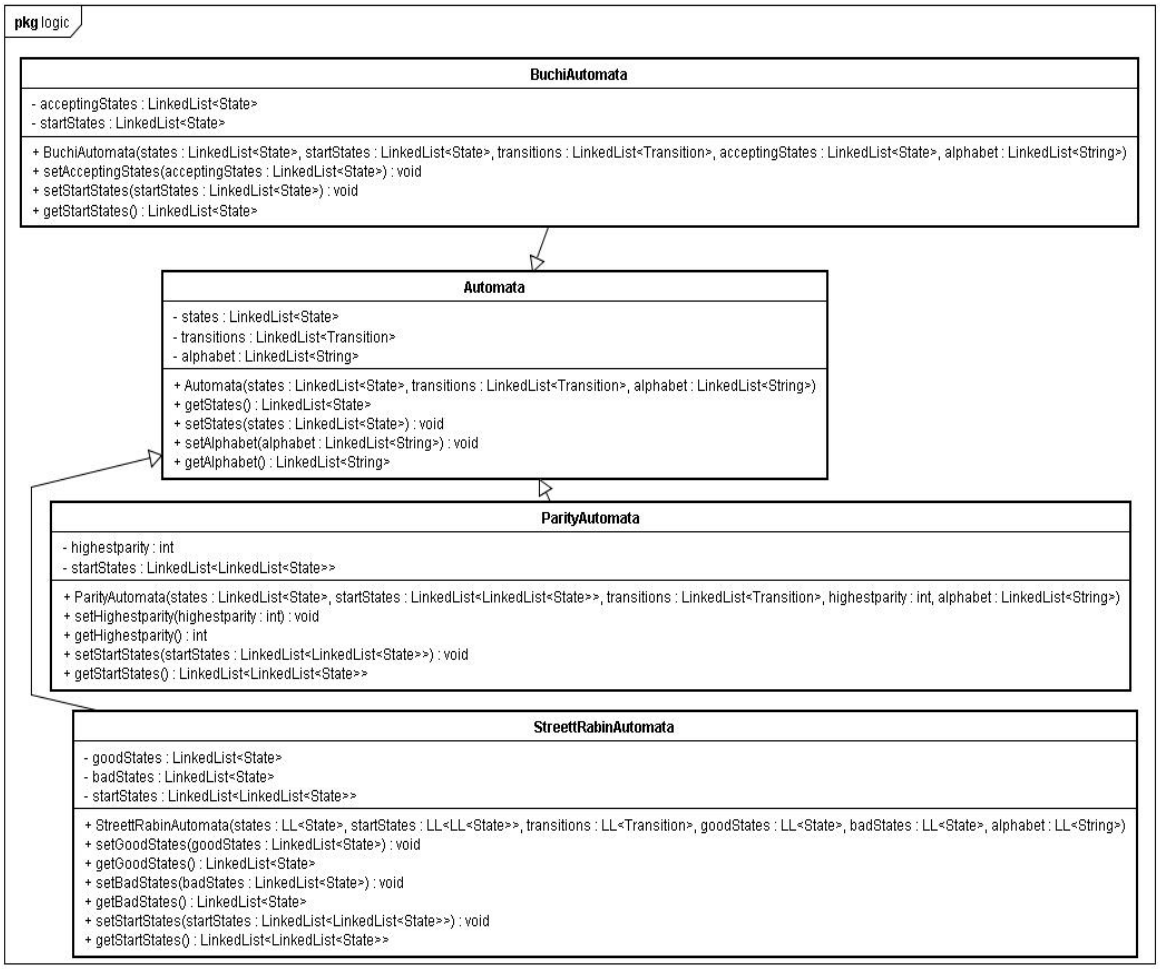

Abbildung 4.2: Die Automatenklassen als Klassendiagramm. In dem Konstruktor „StreetRabin-Automata“ wurde „LinkedList“ durch „LL“ abgekürzt.

jedes Modell eine neue Transition mit gleichem Anfangszustand und Eingabesymbol erstellt.

Danach werden noch alle Transitionen und möglichen Mengen von Startzuständen entfernt, deren Folgezustände bzw. die selber kein minimales Modell der positiven boolschen Formel sind. Die Rückgabe der Methode ist dann ein alternierender Paritätsautomat mit einer Liste von Zuständen, einer Liste von Listen von Startzuständen, einer höchsten Parität, einem Alphabet und einer Liste von Transitionen.

Die Zustände des Paritätsautomaten sind ParityStates, die eine ID und eine Parität haben. Die Liste der Listen der Startzustände ist eine Liste mit Mengen von Zuständen des ASA bzw. ARA, in denen der NBA gleichzeitig starten kann. In der Liste von Transitionen sind Transitions, die einen aktuellen Zustand, eine Eingabe und eine Liste von Folgezuständen, in die gleichzeitig gegangen wird, haben.

Danach wird in der Klasse Transformation die Methode transform aufgerufen und der Paritätsau-

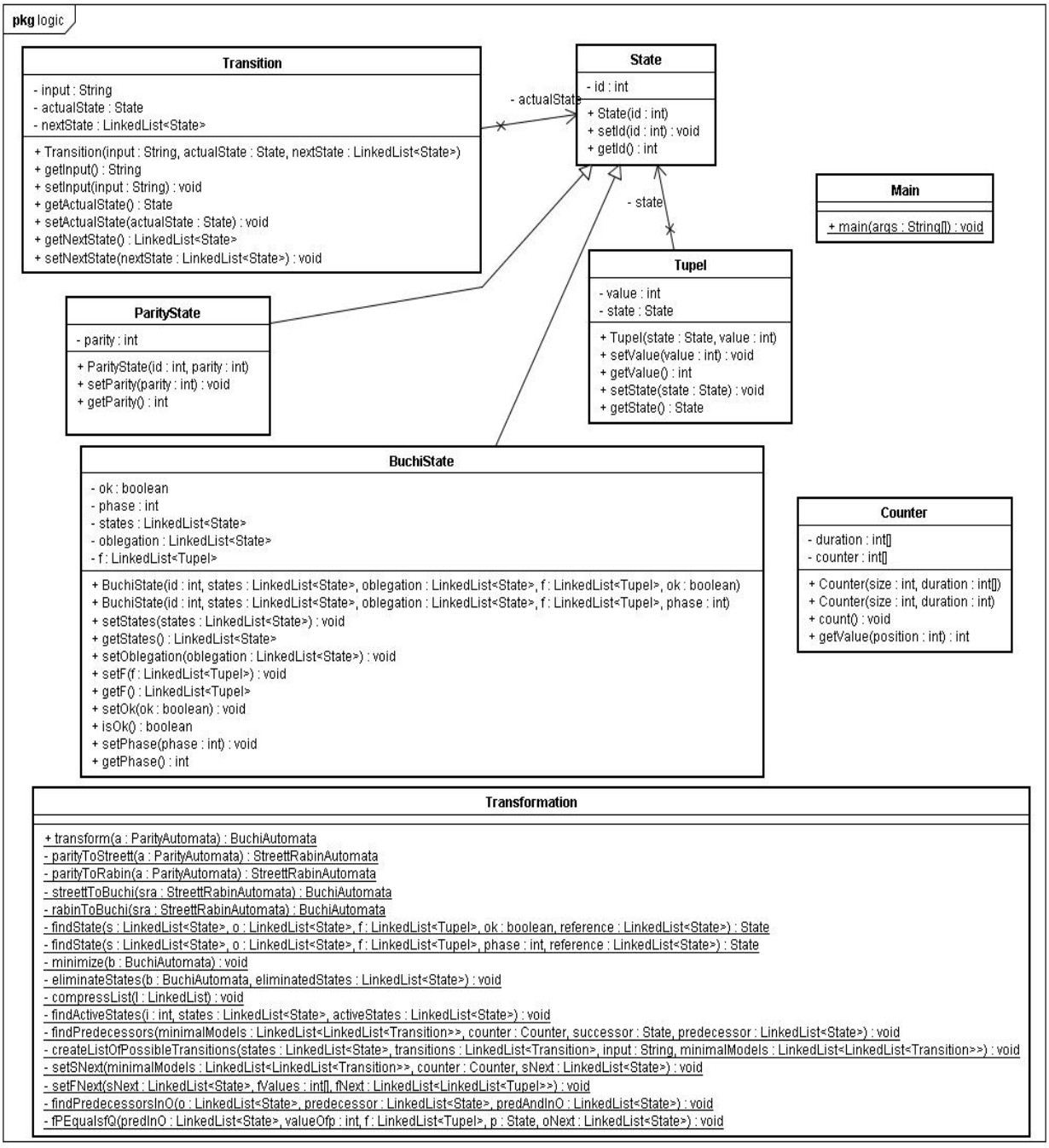

Abbildung 4.3: Das Paket Logic ohne die Automatenklassen als Klassendiagramm.

```
StreettRabinAutomata parityToStreett(ParityAutomata a){
   for(alle Zustände von a){
      if (Parität = 1){
         füge den Zustand zu B des ASA hinzu;
      } else if (Parität = 2){
         füge den Zustand zu G des ASA hinzu;
      }
      füge den Zustand zu den Zuständen des ASA hinzu;
   }
   return den ASA;
}

StreettRabinAutomata parityToRabin(ParityAutomata a){
   for(alle Zustände von a){
      if (Parität = 3){
         füge den Zustand zu B des ARA hinzu;
      } else if (Parität = 2){
         füge den Zustand zu G des ARA hinzu;
      }
      füge den Zustand zu den Zuständen des ARA hinzu;
   }
   return den ARA;
}
```

Listing 4.2: Umwandlung von einem APA zu einem ASA / ARA in Pseudocode.

tomat übergeben. Alle Klassen, die Automaten darstellen, sind in dem Klassendiagramm in Abbildung 4.2 und alle anderen benutzten Klassen sind in dem Klassendiagramm in Abbildung 4.3 dargestellt. Es wird geprüft, ob die höchste Parität des Paritätsautomaten kleiner als 3 ist oder nicht und dann die Umwandlung über einen ASA oder über einen ARA aufgerufen. Egal, welche Methode aufgerufen wird, es wird zuerst der Paritätsautomat in einen StreettRabinAutomata umgewandelt, der dann den ASA oder den ARA darstellt. Dabei werden, je nachdem welche Parität sie haben, die Zustände des Paritätsautomaten zum Einen in die generelle Menge der Zustände des StreettRabinAutomata getan und zum Anderen eventuell zusätzlich in die Menge der guten oder bösen Zustände, siehe Listing 4.2, in dem diese Umwandlung in Pseudocode dargestellt wird.

Mit dem entstandenen Automaten wird dann die eigentliche Transformation zu einem NBA durchgeführt.

Die Transformation läuft in einer etwas anderen Reihenfolge ab, als sie im theoretischen Teil beschrieben ist. Dadurch entsteht zum Einen weniger Code und zum Anderen wird weniger Speicherplatz verbraucht, da nur die Zustände erstellt werden, die auch von einem Startzustand aus erreichbar sind.

Die Startzustände werden erstellt, indem in einer Schleife über alle minimalen Modelle der positiven boolschen Formel der Startzustände gelaufen wird (Listing 4.4). Das minimale Modell ist dann

```
class Counter{

    // Verschiedene Maximalwerte
    Counter(size, duration[]){
       counter := new int[size];
       this.duration := duration;
    }

    // Gleiche Maximalwerte
    Counter(size, duration){
       counter := new int[size];
       this.duration := new int[size];
       for (i := 0 to duration.length - 1){
          this.duration[i] := duration;
       }
    }

    void count(){
       counter[0]++;
       for (i := 0 to counter.length - 2){
          if(counter[i] = duration[i]){
             counter[i] := 0;
             counter[i + 1]++;
          }
       }
    }
}
```

Listing 4.3: Die Klasse Counter in Pseudocode.

die Menge S eines Startzustandes. Dann wird ein Counter erstellt, der immer bis $2n$ läuft. Dieser besteht aus einem Array aus Integer, dessen Einträge als einzelne Zähler genutzt werden, und einem Maximalwert für jedes Element des Integer-Arrays. Wird der Counter erhöht, so wird das kleinste Feld des Arrays inkrementiert. Erreicht dieses den Maximalwert, so wird es auf 0 gesetzt und das nächste Feld inkrementiert, bis dieses den Maximalwert erreicht. Dies wird so fortgesetzt. Damit ist gewährleistet, dass es jede Kombination von Zahlen der Felder des Arrays nur genau ein Mal gibt, bis das letztes Element des Arrays seinen Maximalwert erreicht (siehe Listing 4.3). Solange der letzte Zähler $2n$ nicht überschritten hat, wird f erstellt, indem jedem Zustand in S der Wert seines Zählers zugeordnet wird. Ist dieser Wert ungerade und der Zustand aus S in B, so wird kein Tupel zu f hinzugefügt. Bildet f nun auf einen gerade Wert ab und ist das Element aus S nicht in G, so ist es auch in O bzw. bei einem ARA ist es in O, wenn f auf einen gerade Wert abbildet. Gibt es nun so viele Elemente in f wie in S, daher wurde kein fehlerhaftes Tupel festgestellt, so wird der Zustand zu der Menge der nicht-deterministisch auswählbaren Startzustände hinzugefügt. Außerdem wird der Zustand auch den Zuständen des NBA hinzugefügt, sofern es ihn nicht schon gibt.

```
for (alle minimalen Modelle der Startzustandsformel) {
   füge die Elemente des minimalen Modells zu S hinzu;
   counter = new Counter(Anzahl der minimalen Modelle, 2n + 1);
   while (der letzte counter von counter ist nicht am Ende) {
      for (alle x in S) {
         if (!(x ist in B && counter(x) % 2 = 1)) {
            fuege das Tupel (x, counter(x)) zu f hinzu;
            if (counter(x) % 2 = 0 && x ist nicht in G) {
               fuege x zu O hinzu;
            }
         }
      }
      if (f.size() = S.size()) {
         if (Zustand (S,O,f,false) existiert) {
            fuege (S,O,f,false) zu den Startzuständen hinzu;
         } else {
         erstelle Zustand (S,O,f,false);
         fuege (S,O,f,false) zu den Startzuständen hinzu;
      }
      counter.count();
   }
}
```

Listing 4.4: Die Erstellung der Startzustände ausgegangen von einem ASA in Pseudocode.

Das Komplexeste beim Implementieren der Umwandlung war das Erstellen der Transitionen. Zuerst wird ein Stack erstellt, der zu Beginn alle Startzustände enthält. Die Transitionen werden dann in einer Schleife erstellt, die so lange läuft, bis der Stack leer ist. Zu Beginn der Schleife wird der oberste Zustand auf dem Stack heruntergenommen und für jede mögliche Eingabe a der Algorithmus aus der vorher vorgestellten Umwandlung ausgeführt (siehe Listing 4.5). Dadurch entstehen dann alle Transitionen, die von diesem Zustand ausgehen.

```
erstelle einen Stack stack mit allen Startzuständen;
while (stack ist nicht leer) {
   state = stack.pop();
   for (alle möglichen Eingaben a des Alphabetes) {
      suche für jedes Element in S von state alle Transitionen mit
      dem Element als Anfangszustand und der Eingabe a;
      speichere die gefundenen Transitionen in einer Liste l an die
      Position, die auch das Element in S hat;
      if (es gibt eine solche Transition für jedes Element aus S) {
         erstelle einen neuen Counter counter, der jeweils für jedes
         Element  von l so weit läuft, wie es dort Elemente gibt;
         while (der letzte counter von counter ist nicht am Ende) {
            for (i = 0 to Elemente in S - 1) {
               füge alle Endzustände der Transition
               l(i).get(counter.counter(i)) zu S' hinzu;
            }
            for (alle Elemente in S') {
               suche alle Vorgänger, die nicht in G sind;
               value = kleinster Wert auf den f bei einem
               der Vorgänger abbildet, sonst 2n;
               speichere value in fValues an der Position von
               dem Element in S';
            }
            // f'list enthält alle möglichen f' des Folgezustandes
            erstelle eine Liste f'list mit Listen aus Tupeln;
            // Für jedes Element aus S' kann entschieden werden,
            // ob f' auf einen geringeren oder den selben Wert
            // abbildet, wie f für die Vorgänger.
            tue jede mögliche Kombination aus Tupeln als
            Liste in f'list;
            for (alle Listen f' in f'list) {
               if (ok) {
                  for (alle Tupel t in f') {
                     if (t.value % 2 = 0) {
```

```
                        O'.add(t.getState);
                    }
                }
            } else {
                for (alle Elemente p in S'\G) {
                    finde alle Vorgänger, die auch in O sind;
                    bildet für einen dieser f auf den selben Wert
                    ab wie f' für p, füge p zu O' hinzu;
                }
            }
            if (es gibt kein nicht erlaubtes Tupel in f') {
                if (O' = ∅) {
                    if ((S',O',f',true) existiert) {
                    erstelle eine neue Transition für den NBA
                    von state mit a nach (S',O',f',true);
                    } else {
                        erstelle (S',O',f',true);
                        stack.push((S',O',f',true));
                        erstelle eine neue Transition für den NBA
                        von state mit a nach (S',O',f',true);
                    }
                }
                if ((S',O',f',false) existiert) {
                erstelle eine neue Transition für den NBA
                von state mit a nach (S',O',f',false);
                } else {
                    erstelle (S',O',f',false);
                    stack.push((S',O',f',false));
                    erstelle eine neue Transition für den NBA
                    von state mit a nach (S',O',f',false);
                }
            }
        }
        counter.count();
    }
  }
}
```

Listing 4.5: Die Erstellung der Transitionen und der Zustände ausgegangen von einem ASA in Pseudocode.

Zuerst wird eine Liste l aus Listen von Transitionen erstellt. Dann werden für jedes Element x der Menge S des betrachteten Zustandes alle Transitionen in dem ASA bzw. ARA gesucht, die für a von diesem Zustand ausgehen. Die gefundenen Transitionen werden dann in l an die Position geschrieben, die x auch in S hat. Die Endzustände dieser Transitionen in der Liste sind dann alle möglichen Folgezustände des Elementes aus S. Nun werden also, um S' eines Folgezustandes zu

bilden, immer für jedes Element aus S die Endzustände einer Transition aus der dazugehörigen Liste genommen. Welche Transitionen genommen werden, wird nicht-deterministisch entschieden. Damit jede Kombination der möglichen Folgezustände für ein Element aus S einmal vorkommt, wird wieder ein „Counter“ genutzt.

Dies ist auch die Bedingung *SD*1 bzw. *RD*1. Um als nächstes f' zu bestimmen wird für jedes Element x' aus S' sein Vorgänger in S gesucht (siehe *SD*2 bzw. *RD*2). Da durch den Counter noch gemerkt wurde, welche Transitionen in den Listen in l betrachtet wurden, kann nun einfach überprüft werden, in welchen dieser Transitionen x' als Endzustand vorhanden ist. Der Anfangszustand dieser Transitionen ist dann ein Vorgänger von x'. Wird von einem ASA ausgegangen, müssen davon noch alle Zustände aus G entfernt werden. Nun wird das Minimum gesucht, auf das f für diese Vorgänger abbildet. Sollte die Menge der Vorgänger leer sein, weil alle Zustände in G waren, so bildet f wieder auf $2n$ ab. Dieser Wert wird sich gemerkt. Danach muss für jeden Zustand in S' nicht-deterministisch entschieden werden, ob f' auf diesen Wert abbildet oder auf einen Wert, der eins kleiner ist, wenn erlaubt. Das heißt, es gibt einen Folgezustand in dem NBA für jede mögliche Abbildung f'. Diese wird wieder mit Hilfe eines Counters erstellt. Der letzte Teil unterscheidet sich bei den beiden Umwandlungen und wird mit der Variante über einen ASA beginnen. Als nächstes wird dann O' erstellt, von dessen Leere auch ok abhängt. War $ok = true$, so werden einfach alle Elemente aus S' hinzugefügt, für die f' auf eine gerade Zahl abbildet. Ist es nicht der Fall, so müssen zuerst wieder alle Vorgänger gefunden werden. Für jeden dieser Vorgänger wird dann geprüft, ob er auch in O war. Dann wird über alle diese Vorgänger gelaufen und deren Wert, auf den f abbildet, mit dem Wert verglichen, auf den f' für das Element aus S' abbildet. Ist dieser gleich, so wird das Element zu O' hinzugefügt (siehe *SD*3). War der Ausgangsautomat nun ein ARA, so wird $phase'$ als auch O' gleichzeitig bestimmt. Dazu wird in einer Schleife über i von 0 bis 2 gelaufen, da $phase'$ genau diese Werte annehmen kann, und sofern nach *RD*3 der Wert i als $phase'$ erlaubt ist, was durch eine einfache Abfrage der dort angegebenen Bedingungen getestet wird, wird zu *RD*4 übergegangen. Es wird überprüft, welcher der vier Fälle eingetreten ist, falls keiner eingetreten ist bleibt O' leer, und dann werden die entsprechenden Aktionen ausgeführt. Die Fälle $phase = 2$ und $i = 0$ sowie $phase = 0$ und $i = 1$ sind durch eine Schleife ausführbar. Der Fall $phase = 1$ und $i = 1$ ist genau der selbe Fall, der vorher auch schon in der Umwandlung über den ASA in dem Fall $ok = false$ beschrieben wurde. Bei dem letzten Fall $phase = 0$ und $i = 0$ werden ähnlich wie bei dem vorherigen Fall die Schnittmenge aus den Vorgängern und den Elementen aus O gebildet und wenn diese nicht leer ist und das Element aus S' nicht in G ist, wird es O' hinzugefügt.

Danach wird eine neue Transition erstellt, mit dem gefundenen Nachfolger (S', O', f', ok') bzw. $(S', O', f', phase')$ als Endzustand. Gab es den Endzustand der Transition noch nicht in der Menge der Zustände des NBAs, so wird dieser sowohl dort hinzugefügt als auch auf den Stack getan, da für diesen noch keine Transitionen erstellt wurden. Hierbei ist das Bestimmen der Phase bzw. von O' die innerste Aktion, die Bestimmung von f' etwas weiter außen und die Bestimmung der nächsten Zustände die äußerste. Das heißt, es werden zuerst für eine Menge S' und eine Funktion f' alle Folgezustände mit verschiedenen Phasen und Mengen O' bestimmt. Dann für eine Menge S' das nächste f' gebildet und wenn alle verschiedenen f' für eine Menge S' durch sind, die nächste Menge S' erstellt, bis alle Folgezustände für einen Zustand des NBA gefunden wurden.

```
for (alle Zustände des NBA) {
   if (ok) {
      füge den Zustand zu den Akzeptierenden hinzu;
   }
}
```

Listing 4.6: Die Erstellung der akzeptierenden Zustände ausgegangen von einem ASA in Pseudocode.

Dann wird mit dem nächsten Zustand auf dem Stack fortgefahren und für diesen alle Transitionen erstellt. Dies geht so lange, bis der Stack leer ist, daher für jeden, von den Startzuständen aus erreichbaren, Zustand alle Transitionen erstellt wurden.

Zur Erstellung der Liste der akzeptierenden Zustände wird über alle Zustände des NBAs gelaufen (siehe Listing 4.6). Ist, im Falle eines ASA als Ausgangsautomat, $ok = true$, so wird der Zustand zur Liste der akzeptierenden Zustände hinzugefügt. Im Falle eines ARA als Ausgangsautomat wird geprüft, ob $phase = 2$ gilt und dann der Zustand zu der Liste der akzeptierenden Zuständen hinzugefügt.

Am Ende wird ein Büchi-Automat zurückgegeben, für den als nächstes Minimierungen durchgeführt werden, welche in der nächsten Sektion beschrieben sind. Nach der Minimierung wird der Automat dann durch den DotPrinter im Dot-Format in eine Text-Datei geschrieben.

4.4 Minimierungsverfahren

Nachdem der NBA erstellt wurde enthält dieser meistens noch viele Zustände, welche die akzeptierte Sprache des Automaten nicht beeinflussen. Diese werden nach der Transformation durch verschiedene Minimierungsverfahren entfernt. Insgesamt wird der Automat dadurch deutlich kleiner.

Es werden zwei verschiedene Minimierungen wiederholt durchlaufen. Dies geschieht in einer Schleife, die erst verlassen wird, wenn der Automat nach einem Durchlauf genau so viele Zustände und Transitionen hat, wie vor dem Durchlauf.

Als erste Minimierung wird nacheinander von jedem Zustand des NBA aus eine Breitensuche durchgeführt. Der Zustand ist der erste auf dem Stack. Dann wird eine Schleife durchlaufen, die so lange läuft, bis der Stack leer ist. Am Anfang der Schleife werden alle Zustände auf dem Stack zu einer Menge von besuchten Zuständen hinzugefügt. Dann werden alle Transitionen gesucht, die einen der Zustände auf dem Stack als Startzustände haben und die Endzustände der Transitionen werden dann auf den Stack gelegt. Vor Ende der Schleife werden dann alle Zustände vom Stack entfernt, die schon in der Liste der besuchten Zustände sind. Die Schleife endet dann, wenn alle, von dem aktuell betrachteten Zustand aus besuchbaren, Zustände besucht wurden. Ist unter diesen besuchten Zuständen kein akzeptierender, so können alle Zustände, die besucht wurden, entfernt

werden, da dieser Zyklus dann ein totes Ende ist und wenn der Automat jemals bei einem Lauf in diesen Zyklus kommen würde, so kann er nie akzeptieren.

Die zweite Minimierung sucht für jeden Zustand nacheinander alle Transitionen durch, ob es eine ausgehende Transition von diesem Zustand gibt. Gibt es keine, so ist der Zustand ein totes Ende, da die Eingabeworte bei dem NBA unendlich lang sind und es dann bei dem nächsten gelesenen Buchstaben keine Transition gäbe. Damit kann der Zustand, sowie alle zu ihm führenden Transitionen, dann entfernt werden.

Diese Minimierungsverfahren haben zusammen den großen Vorteil, dass sie einen NBA, dessen akzeptierte Sprache leer ist, so weit minimieren, dass er keinen Zustand mehr enthält. Auch im Durchschnitt wird die Menge der Zustände des NBAs durch die Aneinanderreihung dieser Minimierungen um 30-90% verringert.

4.5 Experimentelle Resultate

Es werden nun noch einige beispielhafte Werte für die Erhöhung der Zustände in Abhängigkeit von der Anzahl der Zustände des Ausgangsautomaten gegeben, um eine Basis für die Vergleichbarkeit zu bisherigen aber auch zu späteren Umwandlungen zu schaffen. Es wurden dazu 194 APAs aus den LTL-Formeln aus [dwy98] durch das Programm aus [sch11] generiert, die dann von dem, in dieser Arbeit entstandenen, Programm in NBAs transformiert wurden. Da nur der schlechteste Fall wirklich interessant ist, wird auch nur dieser an diesem Punkt betrachtet.

Nach der Umwandlung der 3-APAs in NBAs ergaben sich folgende Maximalwerte:

Anzahl der Zustände des APA	Anzahl der Zustände des NBA
1	5
2	10
3	29
4	69
5	111
6	144
7	620
8	1229
9	1489

Wie man sieht steigt die Anzahl der Zustände des NBA sehr schnell an, wenn die Menge der Zustände des APA sich erhöht. Der Anstieg passt auch etwa zu der in dem Paper [sf11] erwähnten Erhöhung von $2^{O(n \log n)}$ für einen 3-APA mit n Zuständen.

5 Zusammenfassung

Man hat in dieser Arbeit gesehen, dass es schon viele Ansätze für die Umwandlung von einem APA zu einem NBA gibt. Nur wenige sind allerdings für die Praxis relevant. Dies liegt vor allem an der Erhöhung der Anzahl der Zustände, die oft so extrem ist, dass dies von einem normalen Rechner nicht mehr zu bewältigen ist.

Die ausgewählte Transformation hat dabei die geringste Erhöhung der Zustände und ist damit die beste Wahl für ein Programm um 3-APAs in NBAs umzuwandeln, da sie keinerlei schwierige Konstruktionen benutzt und die einzige Umwandlung ist, die extra auf 3-APAs zugeschnitten wurde. Dadurch, dass die Umwandlung extra für 3-APAs entworfen wurde, kann das Problem der doppelt-exponentiellen Erhöhung der Zustände, welche durch das Entfernen des Alternierends und durch das Umwandeln der Akzeptanzbedingung entsteht, umgangen werden. Daher entsteht nur eine Erhöhung von n Zuständen des APAs auf $2^{O(n \log n)}$ bei dem NBA statt $2^{2^{O(n)}}$. Damit ist es möglich, dieses Programm auch auf weit verbreiteten Rechnern zu nutzen.

Wurde die RLTL-Formel mit Hilfe des Programms aus [sch11] in einen APA umgewandelt, so kann diese mit Hilfe des, während dieser Arbeit entstandenen, Programms weiter in einen NBA transformiert werden. Durch einen Leerheitstest kann dann das Model-Checking durchgeführt und damit die Korrektheit eines Programms geprüft werden. Dazu kann ein Model-Checker, wie zum Beispiel SPIN, verwendet werden.

Damit würde man auf das Testen eines Programmes von Hand verzichten können. Die Verifikation eines Programmes wird dann von einem Computer durchgeführt, wodurch menschliches Versagen beim Durchführen der Tests ausgeschlossen wird. Dadurch können schwerwiegende Fehler vermieden werden, wie sie am Anfang dieser Arbeit erwähnt wurden, welche viel Geld und auch Leben gekostet haben.

Zum Verifizieren spezifiziert man das Verhalten des Programmes mit einer Spezifikationssprache. In diesem Fall wird das Verhalten als RLTL-Formel spezifiziert und kann dann mit Hilfe des Programmes aus [sch11] und dem in dieser Arbeit entstandenen in einen NBA umgewandelt werden. An diesem NBA sind dann noch einige Veränderungen bezüglich des Formates nötig, je nachdem, welches Programm man für das Model-Checking benutzt, damit der NBA als Eingabe funktioniert. Danach kann der Model-Checker das Programm, das verifiziert werden soll, verifizieren und damit entweder die Korrektheit beweisen oder widerlegen.

Abkürzungsverzeichnis

CTL Computation Tree Logic

LTL Linear Temporal Logic

RLTL Regular Linear Temporal Logic

pRLTL Regular Linear Temporal Logic with past

DAG Directed Acyclic Graph

DPA Deterministischer Paritätsautomat

NBA Nicht-deterministischer Büchi-Automat

ABA Alternierender Büchi-Automat

APA Alternierender Paritätsautomat

3-APA Alternierender Paritätsautomat mit maximal 3 Paritäten

ASA Alternierender Streett-Automat

ARA Alternierender Rabin-Automat

Abbildungsverzeichnis

Listings

Literaturverzeichnis

[dwy98] Matthew B. Dwyer, George S. Avrunin und James C. Corbett, *Property Specification Patterns for Finite-State Verification*, , 1998

[kla08] Christian Dax und Felix Klaedtke, *Alternation Elimination by Complementation*, ETH Zürich, Computer Science Department, 2008

[kv05] Orna Kupferman und Moshe Y. Vardi, *Complementation Constructions for Nondeterministic Automata on Infinite Words*, In: Proceedings of the 11th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS) vol. 3440 of Lect. Notes Comput. Sci., UC Berkeley und Rice University, 2005

[kv98] Orna Kupferman und Moshe Y. Vardi, *Weak Alternating Automata and Tree Automata Emptiness*, UC Berkeley und Rice University, 1998

[leu02] Martin Leucker, *Logics for Mazurkiewicz Traces*, Dissertation an der RWTH Aachen, 2002

[meh06] Karl Mehltretter, *Übersetzung zwischen dem Linearzeit-μ-Kalkül und schwachen alternierenden Paritätsautomaten*, LMU München, 2006

[saf88] Shmuel Safra, *On the complexity of ω-automata*, In: Proceedings of the 29th FOCS, 1988

[sale07] César Sánchez und Martin Leucker, *Regular Linear Temporal Logic*, Spanish Council for Scientific Research (CSIC), 2007

[sale10] César Sánchez und Martin Leucker, *Regular Linear Temporal Logic with Past*, Spanish Council for Scientific Research (CSIC), 2010

[san11] César Sánchez, *The Power of Dualization in Regular Linear Temporal Logic*, Spanish Council for Scientific Research (CSIC), 2011

[sch11] Malte Schmitz, *Transformation von Regular Linear Temporal Logic zu Paritätsautomaten*, Bachelorarbeit an der Universität zu Lübeck, 2011

[sf11] César Sánchez und Julián Samborski-Forlese, *Efficient Regular Linear Temporal Logic using Dualization and Stratification*, Spanish Council for Scientific Research (CSIC) und IMDEA Software Institute, 2007 (unveröffentlicht)