Konstruktion und Klassifizierung von Strategien für unendliche Spiele auf endlichen Graphen

Diplomarbeit
vorgelegt von
Olaf Junge

Betreuer
Prof. Dr. Wolfgang Thomas

Mai 1994
Institut für Informatik und Praktische Mathematik
der Christian-Albrechts-Universität zu Kiel

Inhaltsverzeichnis

Kapitel 1

Einleitung

1.1 Einführung

Unendliche Spiele, oder besser Spiele von unendlicher Dauer, spielen innerhalb der Informatik eine zunehmend bedeutende Rolle. Sie sind geeignet, Prozesse zu beschreiben, die nicht terminieren, und sich somit nicht mit den üblichen Methoden der Automaten- und Sprachentheorie untersuchen lassen. Nichtterminierende Prozesse mit definiertem Ein-/Ausgabe-Verhalten werden auch *reaktive Systeme* genannt. Ein Beispiel für ein reaktives System ist das Betriebssystem eines Computers. Angefangen von elementaren Ein- und Ausgabefunktionen, über die Pflege des Dateisystems, bishin zur Verwaltung von mehreren Benutzern und Verteilung der Rechenzeit auf viele gleichzeitig ablaufende Prozesse, ist ein Betriebssystem fundamentaler Bestandteil eines Computers. Die Interaktion des Betriebssystems mit seinen Benutzern, kann man sich als ein Spiel unendlicher Dauer vorstellen, denn im Normalfall soll ja ein Betriebssystem nicht terminieren, sondern die Rechenleistung des Computers ständig bereithalten. Einen unerwünschten Zustand oder einen unvorhergesehenen Abbruch des Betriebssystems könnte man dabei als einen Verlust des Spiels betrachten. Eine Analyse des Spiels könnte dann zum Beispiel zur Untersuchung der Korrektheit des Betriebssystems beitragen.

Ein weiteres Beispiel für ein reaktives System ist die Führung eines Wirtschaftsunternehmens. Hier spielt der Chef des Unternehmens nach den Gesetzen der freien Marktwirtschaft gegen seine Konkurrenten und sonstige wirtschaftliche Faktoren. Auch in diesem Fall ist klar, daß eine endliche Analyse des Systems keinen Aufschluß über einen dauerhaften Erfolg des Unternehmens liefern kann. Eine Betrachtung als unendliches Spiel könnte auch hier neue Erkenntnisse bringen.

Der Gegenstand der Untersuchungen in dieser Diplomarbeit ist eine bestimmte Klasse von Spielen unendlicher Dauer, die auf endlichen Graphen gespielt werden. Es handelt sich dabei um *Zweipersonen-Nullsummen-Spiele mit vollständiger Information* und einem Wert von $+1$ oder -1. Anschaulich sind dies Spiele mit zwei Spielern, in denen immer genau einer der beiden Spieler gewinnt (Nullsummen-Spiel mit Wert $+1$ oder -1), ohne verdeckte Züge und Zufallsereignisse (vollständige Information).

Endliche Zweipersonen-Nullsummen-Spiele mit vollständiger Information sind zum Beispiel Brettspiele wie Schach, Mühle und Dame. Da diese Spiele von realen Menschen gespielt werden, und diese nicht unendlich lange auf den Ausgang einer Partie warten können, wurden die Regeln dieser Spiele so gestaltet, daß die Endlichkeit des Spiels gewährleistet ist, indem die Länge der Partie begrenzt wird. Beim Schach gibt es daher beispielsweise Regeln, die das Spiel in ein Remis enden lassen, wenn dreimal dieselbe Stellung auf dem Brett war, oder 50 Züge lang kein irreversibler Zug gemacht wurde. Der Ausgang eines solchen Spiels kann, da es nun stets endlich ist, leicht aus seiner Schlußstellung ermittelt werden.

Endliche Spiele wurden bereits im Jahr 1944 von John von Neumann und Oskar Morgenstern in ihrem Buch „Theory of Games and Economic Behavior“ [NeuMor44] formalisiert und eingehend untersucht. Unter anderem ist aus der Spieltheorie bekannt, daß die endlichen Zweipersonen-Nullsummen-Spiele mit vollständiger Information, sie werden dort auch als offene Matrix-Spiele bezeichnet, eine *reine* optimale Strategie für beide Spieler besitzen. Eine Konsequenz aus dieser Erkenntnis ist, daß sich für jedes Spiel dieser Art im voraus bestimmen läßt, welcher Spieler eine Gewinnstrategie hat und somit gegen jede denkbare Strategie des Gegners das Spiel gewinnen kann. Man spricht in diesem Zusammenhang auch von der „Determiniertheit“ dieses Spiels. Die optimalen Strategien lassen sich theoretisch aus der Auszahlungsmatrix des Spiels bestimmen, da es nur endlich viele reine Strategien gibt. Unglücklicherweise sind die meisten dieser Spiele aber so komplex, daß diese Aufgabe sich auch von den rechenstärksten Computern nicht bewältigen läßt.

Bei den Spielen unendlicher Dauer ist die Bestimmung optimaler Strategien weitaus schwieriger. 1953 beschäftigten sich D. Gale und F. M. Stewart in [GalSte53] mit unendlichen Spielen folgender Form: Die beiden Spieler nennen abwechselnd eine Ziffer aus $\{0,1\}$. Die dabei entstehende unendliche Folge von Nullen und Einsen bildet ein ω-Wort aus $\{0,1\}^\omega$. Zur Bestimmung des Gewinners wird eine Teilmenge von $\{0,1\}^\omega$, die sogenannte *Gewinnmenge* des Spiels, herangezogen. Ist das in der Partie enstandene ω-Wort in der Gewinnmenge enthalten, so hat der anziehende Spieler gewonnen. Gale und Stewart stellten dabei fest, daß es Spiele dieser Art gibt, die nicht determiniert sind, also für keinen der Spieler eine Gewinnstrategie besitzen.

Beschränkt man sich jedoch auf Gewinnmengen, die gewisse topologische Eigenschaften besitzen, so zeigt sich daß die dazugehörigen Spiele sehr wohl determiniert sind. Gale und Stewart zeigten dies für Spiele mit *offenen*, bzw. *abgeschlossenen* Gewinnmengen. Die Begriffe *offen* und *abgeschlossen* definieren sich dabei über die sogenannte *Cantor-Topologie* auf ω-Wörtern aus $\{0,1\}^\omega$. Die offenen und abgeschlossenen Mengen bilden hierbei gerade die unterste Stufe der sogenannten *Borel-Hierarchie*.

Im Jahre 1955 gelang es Ph. Wolfe zu zeigen, daß auch Spiele mit Gewinnmengen in der zweiten Stufe der Borel-Hierarchie determiniert sind. M. Davis erweiterte das Resultat 1964 in [Dav64] auf die dritte Stufe der Borel-Hierarchie. Eine umfassende Aussage wurde dann 1975 von D. A. Martin bewiesen. Martin zeigte daß alle Spiele mit Gewinnmengen in einer beliebigen Stufe der Borel-Hierarchie die Eigenschaft der Determiniertheit besitzen.

Eine interessante Klasse von unendlichen Spielen ergibt sich, wenn man sich auf *reguläre* Gewinnmengen beschränkt. Die Regularität der Gewinnmenge erlaubt es dann, den Gewinner einer Partie durch einen endlichen Automaten zu bestimmen. Da die regulären Mengen in der dritten Stufe der Borel-Hierarchie enthalten sind ist es klar, daß die dazugehörigen Spiele determiniert sind. Von Interesse ist jetzt also nicht mehr die Frage, ob immer einer der Spieler eine Gewinnstrategie hat, sondern vielmehr welcher es ist und wie dessen Gewinnstrategie aussieht.

J. R. Büchi und L. H. Landweber haben 1969 in [BücLan69] gezeigt, daß Gewinner und Gewinnstrategien sich effektiv bestimmen und die Gewinnstrategien sich sogar durch einen endlichen Automaten darstellen lassen.

Die Möglichkeit, die regulären Gewinnmengen durch endliche Automaten definieren zu können, läßt sich nutzen, um ein etwas spezielleres Spielmodell, als das von Gale und Stewart, zu entwickeln. Robert McNaughton schlägt in [McN93] folgende Modellierung vor: Auf einem Spielbrett befinden sich eine Zeichnung eines endlichen Graphen sowie ein Spielstein, der von beiden Spielern abwechselnd über die Kanten des Graphen von Knoten zu Knoten geschoben werden muß. Der Graph ist dabei so beschaffen, daß stets mindestens ein Zug möglich ist. Das Spiel endet somit nie, und der Gewinner wird aus der Menge der im Laufe des Spiels unendlich oft besuchten Knoten bestimmt. Der endliche Graph ist dabei in gewisser Weise ein endlicher Automat, der den Gewinner des Spiels ermittelt. McNaughton zeigt dann unter anderem die folgende Aussage: Es hat immer einer der beiden Spieler eine *LVR-Gewinnstrategie*, welche sich auch effektiv bestimmen läßt. Eine LVR-Strategie ist eine Strategie, die sich nur auf das sogenante „Last Visitation Record“ einer Stellung bezieht, das anschaulich alle Knoten in der Reihenfolge des jeweils letzten Besuches des Spielsteins enthält.

Man kann die LVR-Gewinnstrategie eines Spiels, bedingt durch die Endlichkeit der Menge aller LVR, als einen endlichen Automaten darstellen, dessen Zustandsanzahl allerdings, bedingt durch die Anzahl möglicher „Last Visitation Records“, von der Größenordnung $O(n!)$ ist, wobei n die Anzahl der *relevanten* Knoten des Spiels ist. *Relevant* ist ein Knoten des Graphen, wenn er zu einer Menge speziell gekennzeichneter Knoten gehört, die zur Definition der Gewinnbedingung benutzt werden.

McNaughton charakterisiert in seinem Artikel auch eine Unterklasse der regulären Spiele, die sogenannten *No-Memory-Spiele*, welche sich durch besonders einfache Gewinnstrategien auszeichnen. Er gibt weiterhin eine notwendige Bedingung an, die alle No-Memory-Spiele erfüllen. Wir werden in dieser Arbeit neben den *No-Memory-Spielen* noch die Klasse der *VS-Spiele* einführen und für beide Spielklassen hinreichende Bedingungen aufzeigen. Außerdem werden wir Verfahren vorstellen, die No-Memory- bzw. VS-Strategien solcher Spiele zu bestimmen.

1.2 Überblick

Im zweiten Kapitel werden wir uns den Grundlagen der unendlichen Spiele auf endlichen Graphen und deren Strategien widmen. Wir werden einen kleinen Einblick in die Theorie der ω-Sprachen und der Automatentheorie auf unendlichen Wörtern nehmen. In jedem dieser Gebiete, werden wir Sprechweisen und Definitionen einführen, die für die formale Behandlung der unendlichen Spiele in den folgenden Kapiteln nötig sind.

Der Ansatz von McNaughton wird dann im dritten Kapitel vorgestellt. Wir haben versucht die spieltheoretischen Begriffe, die bei McNaughton eher umgangssprachlich eingeführt werden, weitestgehend zu formalisieren. Im Abschnitt 3.2 geben wir eine Übersicht über die in dieser Arbeit untersuchten Strategieklassen.

Eine wesentliche Idee in dieser Arbeit ist es, die allgemeinen Spiele nach dem Ansatz von McNaughton nach bestimmten Kriterien in Klassen einzuteilen, so daß sich „einfachere“ Gewinnstrategien berechnen lassen, als dies mit dem Verfahren von McNaughton möglich ist, das ja die sogenannten *LVR-Strategien* liefert. *Einfacher* bedeutet in diesem Zusammenhang entweder, daß die endlichen Automaten, die die Gewinnstrategien darstellen, kleiner werden (bezüglich der Zustandsmenge), oder deren Konstruktion weniger zeitaufwendig ist. Mit dem Ziel so einer Klasseneinteilung werden wir in Abschnitt 3.3 die endlichen Graphen eines Spiels nach McNaughton mit einem geeignetem Alphabet beschriften, und so in sogenannte Muller-Automaten überführen. Muller-Automaten sind endliche

Automaten, die unendlich lange Wörter akzeptieren. Die von diesen Automaten erkannten sogenannten ω-Sprachen lassen sich dann in die Borel-Hierarchie einordnen. Wie wir dann zeigen werden, ist die Borel-Klasse, in die ein Spiel so eingeordnet wird, von der Art der Beschriftung völlig unabhängig.

Im Kapitel 4 werden wir dann den verschiedenen Spielklassen Strategieklassen zuordnen. Es wird sich dabei herausstellen, daß es im wesentlichen drei Klassen von Spielen gibt, deren Gewinnstrategien sich in ihrer Komplexität unterscheiden.

In der ersten Stufe der Borel-Hierarchie lassen sich alle Spiele mit einer sogenannten No-Memory-Strategie gewinnen. Bei diesen Strategien werden keinerlei Kenntnisse über den Partieverlauf benutzt, sodaß an einem Knoten des Spielgraphen zu jedem Zeitpunkt der gleiche Zug gemacht wird. Der dazugehörige Strategie-Automat hat folglich die gleiche Größe wie der Spielgraph selbst, er entsteht aus diesem im Wesentlichen durch Streichung von Kanten. Die No-Memory-Strategien stellen sich somit als einfachste Strategien für Spiele auf endlichen Graphen dar.

Liegt ein Spiel in der zweiten Stufe der Borel-Hierarchie, so werden wir Gewinnstrategien ermitteln, die sich auf ein sogenanntes *Visitation Set* VS stützen. Das VS ist stets eine Teilmenge der Knotenmenge des Spielgraphen und die Menge aller möglichen VS hat somit eine maximale Mächtigkeit von 2^n. Die Strategie-Automaten, die wir dann konstruieren werden, haben folglich eine Größenordnung von $O(2^n)$.

Die regulären Spiele schließlich, die innerhalb der dritten Stufe der Borel-Hierarchie liegen, lassen sich mit den von McNaughton bestimmten LVR-Strategien gewinnen. Die Strategie-Automaten haben dann allerdings eine Größe von $O(n!)$.

Die Bestimmung der Gewinnstragien für die „einfacheren“ Spiele, sprich Spiele aus der ersten und zweiten Stufe der Borel-Hierarchie, werden wir vornehmen, indem wir die Spiele aus den einfacheren Spielklassen in Spiele auf speziellen Automaten überführen, die entsprechend vereinfachte Akzeptanzbedingungen aufweisen. Für diese Spiele, die im allgemeinen eine größere Knotenmenge haben, lassen sich jedoch sehr viel einfacher Gewinnstrategien berechnen. Wir sind sogar in der Lage, für diese Spiele No-Memory-Gewinnstrategien zu konstruieren. Auf diese Weise erhalten wir auch auf direktem Weg Gewinnstrategie-Automaten für die ursprünglichen Spiele.

Im fünften Kapitel werden wir dann für jedes dieser speziellen Spiele, es handelt sich dabei um die sogenannten *Spiele mit 1-Akzeptanz*, die *deterministischen Büchi-Spiele* und die *Spiele mit Occurrence Check*, die Gewinnstrategien bestim-

men und einen Algorithmus zur Konstruktion dieser Strategien angeben. Eine Komplexitätsbetrachtung zu den Konstruktionsalgorithmen schließt sich jeweils an. Wir werden zeigen, daß die Konstruktion der Gewinnstrategien bei Spielen mit 1-Akzeptanz eine Zeitkomplexität von $O(n^3)$, und bei deterministischen Büchi-Spielen eine Zeitkomplexität von $O(n^4)$ hat.

Eine knappe Zusammenfassung dieser Arbeit, und Ausblicke auf offene Fragen findet man dann in Kapitel 6.

Im Anhang befinden sich zum Abschluß der Arbeit Sourcecodelistings von konkreten Implementationen einiger der vorgestellten Algorithmen in der Programmiersprache C. Entwickelt wurde dieses Programm auf einem PC mit Unix-Betriebssystem.

Kapitel 2

Grundlagen

2.1 ω-Sprachen

Eine ω-Sprache ist eine Menge von unendlichen Wörtern über einem endlichen Alphabet. Sei Σ so ein endliches Alphabet, dann ist

$$\Sigma^\omega = \{a_0 a_1 a_2 \ldots | a_i \in \Sigma \text{ für alle } i \in I\!N\}$$

die Menge aller ω-Wörter über Σ.

Auf den ω-Wörtern läßt sich eine Metrik definieren:

$$d: \quad \Sigma^\omega \times \Sigma^\omega \to I\!R$$
$$(a_0 a_1 \ldots, b_0 b_1 \ldots) \mapsto \begin{cases} 0 & \text{falls } \forall i \in I\!N : a_i = b_i \\ (\frac{1}{2})^n \text{ mit } n = min(i | a_i \neq b_i) & \text{sonst} \end{cases}$$

Zusammen mit dieser Metrik wird die Menge Σ^ω zu einem *metrischen Raum* (Σ^ω, d). Um eine Topologie auf den ω-Wörtern definieren zu können, brauchen wir noch die Begriffe der *Umgebung* und der *offenen Menge*.

Definition 2.1 Sei (X, d) ein metrischer Raum, $w \in X$ und $r > 0$. Dann heißt

$$\mathrm{B}(w, r) := \{x \in X | d(w, x) < r\}$$

die *offene Kugel* mit Mittelpunkt w und Radius r bezüglich der Metrik d.

Eine Teilmenge $U \subseteq X$ heißt *Umgebung* von w, falls ein $\varepsilon > 0$ existiert, mit

$$\mathrm{B}(w, \varepsilon) \subseteq U.$$

Der Begriff der Umgebung erlaubt uns dann eine Definition der *offenen Menge*.

Definition 2.2 Eine Teilmenge U eines metrischen Raumes (X, d) heißt *offen*, wenn sie Umgebung jedes ihrer Punkte ist, d.h. wenn zu jedem $w \in U$ ein $\varepsilon > 0$ existiert, so daß

$$\mathrm{B}(w, \varepsilon) \subseteq U.$$

Eine Teimenge U eines metrischen Raumes (X, d) heißt *abgeschlossen*, wenn das Komplement $X \setminus U$ offen ist.

Die oben definierte Metrik induziert somit eine Topologie auf Σ^ω, die sogenannte *Cantor-Topologie*.

Für den metrischen Raum (Σ, d) bezeichnen wir die Menge der *offenen* Sprachen mit G und die Menge der *abgeschlossenen* Sprachen mit F. Weiterhin bezeichnen wir die Menge, der als Ergebnis abzählbar vieler Durchschnitte von offenen Sprachen zu erhaltenden Sprachen, mit G_δ, und die Menge, der als Ergebnis abzählbar vieler Vereinigungen von abgeschlossenen Sprachen zu erhaltenden Sprachen, mit F_σ. Setzt man die Kette der abzählbaren Vereinigungen bzw. Durchschnitte fort, so erhält man die *Borel-Hierarchie* der ω-Sprachen.

Definition 2.3 Sei Σ ein endliches Alphabet und d die oben definierte Metrik auf Σ^ω. Wir definieren dann

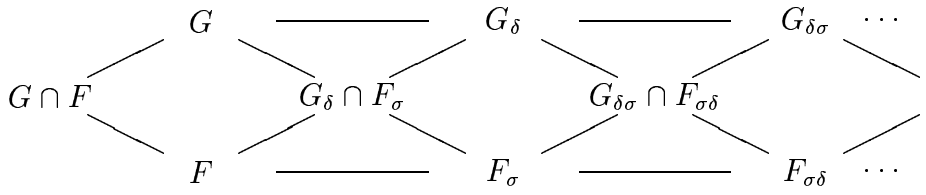
$$\begin{aligned}
G &:= \{L \subseteq \Sigma^\omega | L \text{ ist offene Menge in } (\Sigma^\omega, d)\} \\
F &:= \{L \subseteq \Sigma^\omega | L \text{ ist abgeschlossene Menge in } (\Sigma^\omega, d)\} \\
G_\delta &:= \{L \subseteq \Sigma^\omega | L = \bigcap_{i \in M} L_i \text{ mit } M \text{ abzählbar und } \forall i \in M : L_i \in G\} \\
F_\sigma &:= \{L \subseteq \Sigma^\omega | L = \bigcup_{i \in M} L_i \text{ mit } M \text{ abzählbar und } \forall i \in M : L_i \in F\} \\
G_{\delta\sigma} &:= \{L \subseteq \Sigma^\omega | L = \bigcup_{i \in M} L_i \text{ mit } M \text{ abzählbar und } \forall i \in M : L_i \in G_\delta\} \\
F_{\sigma\delta} &:= \{L \subseteq \Sigma^\omega | L = \bigcap_{i \in M} L_i \text{ mit } M \text{ abzählbar und } \forall i \in M : L_i \in F_\sigma\} \\
&\vdots
\end{aligned}$$

Diese Mengenfolge nennen wir die Borel-Hierarchie der ω-Sprachen.

Bemerkung 2.1 Für die Mengen der Borel-Hierarchie gelten folgende zusammenhänge:

$$\begin{array}{rclcrcl} G & \subset & G_\delta & , & G & \subset & F_\sigma \\ F & \subset & F_\sigma & , & F & \subset & G_\delta \\ G_\delta & \subset & G_{\delta\sigma} & , & G_\delta & \subset & F_{\sigma\delta} \\ F_\sigma & \subset & F_{\sigma\delta} & , & F_\sigma & \subset & G_{\delta\sigma} \\ & \vdots & & & & \vdots & \end{array}$$

Man kann diese Beziehungen in einem Mengendiagramm verdeutlichen:



Jede der Linien in diesem Diagramm stellt eine Mengeninklusion dar, dabei ist jede Inklusion echt (vgl. z.B. [Mos80]).

2.2 Automaten als endliche Graphen

Zunächst möchten wir ein paar nützliche Definitionen angeben:

Definition 2.4 Sei Q eine endliche Zustandsmenge, dann definieren wir

$$\begin{array}{rrcl} \text{In}: & Q^\omega & \to & 2^Q \\ & s_0 s_1 \ldots & \mapsto & \{q \in Q | \exists^\omega i \in I\!N : s_i = q\}. \end{array}$$

$\text{In}(\sigma)$ gibt uns die Menge von Zuständen an, die in der Zustandsfolge σ unendlich oft vorkommen.

$$\begin{array}{rrcl} \text{Occ}: & Q^\omega & \to & 2^Q \\ & s_0 s_1 \ldots & \mapsto & \{q \in Q | \exists i \in I\!N : s_i = q\}. \end{array}$$

$\text{Occ}(\sigma)$ gibt uns die Menge von Zuständen an, die in der Zustandsfolge σ überhaupt vorkommen.

In der Theorie der ω-Sprachen gibt es, genau wie in der Theorie der endlichen Sprachen, Automatenmodelle, die uns helfen die Sprachen in Klassen einzuteilen.

Als erstes wollen wir das Automatenmodell von Büchi erwähnen, das nichtdeterministisch ist.

Definition 2.5 (Büchi-Automat) Ein Büchi-Automat wird repräsentiert durch ein Quintupel $\mathcal{A} = (Q, \Sigma, \Delta, q_0, F)$ mit

Q eine endliche Zustandsmenge,

Σ ein endliches Alphabet,

$\Delta \subseteq Q \times \Sigma \times Q$ eine Transitionsrelation,

$q_0 \in Q$ ein Anfangszustand und

$F \subseteq Q$ eine Menge von Endzuständen.

Sei $\alpha = a_0 a_1 \ldots \in \Sigma^\omega$ ein ω-Wort über dem Alphabet Σ. Ein *Lauf* von $\mathcal{A}$ auf α ist eine Folge $\sigma = s_0 s_1 \ldots$ mit $s_0 = q_0$ und $(s_i, a_i, s_{i+1}) \in \Delta$ für alle $i \geq 0$. Der Lauf heißt *akzeptierend*, wenn $\mathrm{In}(\sigma) \cap F \neq \emptyset$ gilt, also wenn ein Zustand aus F unendlich oft besucht wird.

$\mathcal{A}$ akzeptiert ein Wort $\alpha \in \Sigma^\omega$ genau dann, wenn es einen akzeptierenden Lauf von $\mathcal{A}$ auf α gibt.

Der Nichtdeterminismus in der Definition des Büchi-Automaten ist von fundamentaler Wichtigkeit. Beschränkt man sich auf eine deterministische Variante, den sogenannten deterministischen Büchi-Automaten, so erhält man eine echt kleinere Klasse von erkennbaren Sprachen.

Ein weiteres Automatenmodell, daß genauso mächtig ist, wie das des Büchi-Automaten, aber nicht den Nachteil des Nichtdeterminismus beinhaltet, ist der Muller-Automat.

Definition 2.6 (Muller-Automat) Ein Muller-Automat wird repräsentiert durch ein Quintupel $\mathcal{A} = (Q, \Sigma, \delta, q_0, \mathcal{F})$ mit

Q eine endliche Zustandsmenge,

Σ ein endliches Alphabet,

$\delta : Q \times \Sigma \to Q$ eine Transitionsrelation,

$q_0 \in Q$ ein Anfangszustand und

$\mathcal{F} \subseteq 2^Q$ eine Sammlung von Endzustandsmengen.

Sei $\alpha = a_0 a_1 \ldots \in \Sigma^\omega$ ein ω-Wort über dem Alphabet Σ. Der eindeutig bestimmte *Lauf* von $\mathcal{A}$ auf α ist eine Folge $\sigma = s_0 s_1 \ldots$ mit $s_0 = q_0$ und

$s_{i+1} = \delta(s_i, a_i)$ für alle $i \geq 0$. Der Lauf heißt *akzeptierend*, wenn $\mathrm{In}(\sigma) \in \mathcal{F}$ gilt, also wenn die unendlich oft besuchten Zustände genau einer Endzustandsmenge aus $\mathcal{F}$ entsprechen.

$\mathcal{A}$ akzeptiert ein Wort $\alpha \in \Sigma^\omega$ genau dann, wenn der Lauf von $\mathcal{A}$ auf α ein akzeptierender Lauf ist.

Robert McNaughton zeigte 1966, daß diese beiden Automaten-Modelle äquivalent sind.

Satz 2.1 (McNaughton) Eine ω-Sprache ist genau dann regulär, d.h. Büchi-erkennbar, wenn sie Muller-erkennbar ist.

□

Landweber gab 1969 eine Charakterisierung von regulären ω-Sprachen an, die sich mit verschiedenen Akzeptanzbedingungen eines deterministischen endlichen Automaten beschäftigt.

Definition 2.7 (1-Akzeptanz, 2-Akzeptanz) Sei $\mathcal{A} = (Q, \Sigma, \delta, q_0, F)$ ein deterministischer endlicher Automat über dem Alphabet Σ und $\alpha \in \Sigma^\omega$. Sei $\sigma = s_0 s_1 \ldots$ der eindeutig bestimmte Lauf von $\mathcal{A}$ auf α.

$$\begin{aligned} \mathcal{A} \text{ 1-akzeptiert } \alpha &\Leftrightarrow \exists i \in I\!N : s_i \in F \\ \mathcal{A} \text{ 2-akzeptiert } \alpha &\Leftrightarrow \exists^\omega i \in I\!N : s_i \in F \end{aligned}$$

Eine Sprache $L \subseteq \Sigma^\omega$ heißt 1-erkennbar, bzw. 2-erkennbar, wenn ein deterministischer endlicher Automat existiert der L 1-akzeptiert, bzw. 2-akzeptiert.

Die Sprachklassen der Borel-Hierarchie korrespondieren mit den Automaten mit 1-Akzeptanz bzw. 2-Akzeptanz wie folgt:

Satz 2.2 (Landweber)

1. Eine reguläre ω-Sprache ist genau dann in G, wenn sie 1-erkennbar ist.
2. Eine reguläre ω-Sprache ist genau dann in G_δ, wenn sie 2-erkennbar ist.
3. Es ist entscheidbar, ob eine reguläre ω-Sprache (z.B. gegeben durch einen Muller-Automaten) 1-erkennbar bzw. 2-erkennbar ist.

□

Wie bereits vorher erwähnt, bestätigt sich hier die Tatsache, daß deterministische Büchi-Automaten, die nichts anderes als deterministische endliche Automaten mit 2-Akzeptanz sind, weniger mächtig sind, als nichtdeterministische Büchi-Automaten.

Bemerkung 2.2 Die Sprachklassen lassen sich auch ohne die Zuhilfenahme von Automaten charakterisieren. Setzen wir

$$\lim W := \{a_0 a_1 \ldots \in \Sigma^\omega | \exists^\omega n : a_0 \ldots a_n \in W\},$$

dann können wir feststellen:

1. Eine ω-Sprache $L \subseteq \Sigma^\omega$ wird genau dann von einem deterministischen Büchi-Automaten erkannt, wenn $L = \lim W$ für eine reguläre Menge $W \subseteq \Sigma^\star$.
2. Eine ω-Sprache $L \subseteq \Sigma^\omega$ wird genau dann von einem deterministischen endlichen Automaten 1-erkannt, wenn $L = W\Sigma^\omega$ für eine reguläre Menge $W \subseteq \Sigma^\star$.

□

Kapitel 3

Unendliche Spiele auf Muller-Automaten

Spiele mit unendlicher Dauer wurden zum erstenmal von Gale und Stewart [GalSte53] behandelt. Büchi und Landweber beschäftigten sich in [BücLan69] mit Spielen unendlicher Dauer, deren Gewinnbedingungen mit Hilfe des „sequential calculus“ S1S formuliert werden. Einen ganz anderen Weg ging Robert McNaughton in [McN93], der einen endlichen Graphen als „Spielfeld“ betrachtet, und durch eine den Muller-Automaten entlehnte Akzeptierbedingung den Gewinner feststellt.

3.1 Der Ansatz von McNaughton

Definition 3.1 (Unendliches Spiel) Ein Spiel $\mathcal{G}$ wird repräsentiert durch ein Sextupel $(Q, Q_1, Q_2, K, W, \Omega)$ mit

$Q_1 \cup Q_2 = Q \neq \emptyset$,
$Q_1 \cap Q_2 = \emptyset$,
$\forall k \in K : \exists q_1 \in Q_1, q_2 \in Q_2 : k = (q_1, q_2) \vee k = (q_2, q_1)$,
$\forall q \in Q : \exists q' \in Q : (q, q') \in K$,
$W \subseteq Q$ und
$\Omega \subseteq 2^W$.

Dabei bezeichnet Q die Knotenmenge eines bipartiten gerichteten Graphen, $\{Q_1, Q_2\}$ eine Partitionierung von Q, K die Kantenmenge des Graphen, W

die Menge der *relevanten* Knoten von $\mathcal{G}$ und Ω die *Akzeptanzmenge* des Spiels. Die Spieler ziehen abwechselnd eine imaginäre Markierung entlang der Kanten aus K, wobei Spieler i am Zug ist, wenn die Markierung auf einem Knoten aus Q_i liegt. Spieler 1 gewinnt das Spiel genau dann, wenn die Menge der unendlich oft besuchten Knoten aus W Element von Ω ist, andernfalls gewinnt Spieler 2.

Um diese umgangssprachliche Gewinnbedingung zu präzisieren, sind noch ein paar Definitionen notwendig.

Definition 3.2 (Partie,Gewinnpartie) Sei $\mathcal{G} = (Q, Q_1, Q_2, K, W, \Omega)$ ein Spiel.

$$P_{\mathcal{G}} := \{p = q_0 q_1 \ldots \in Q^\omega | \forall i \in I\!N : (q_i, q_{i+1}) \in K\}$$

bezeichnet die Menge der *Partien* von $\mathcal{G}$,

$$P_{\mathcal{G},q} = \{q_0 q_1 \ldots \in P_{\mathcal{G}} | q_0 = q\}$$

die Menge der Partien von $\mathcal{G}$, die mit dem Knoten q beginnen.

Die *Gewinnpartien* von Spieler 1 liegen in der Menge

$$G_1 := \{p \in P_{\mathcal{G}} | \mathrm{In}(p) \cap W \in \Omega\},$$

die von Spieler 2 in der Menge

$$G_2 := P_{\mathcal{G}} \setminus G_1.$$

Ein Ablauf des Spiels $\mathcal{G}$ sieht also folgendermaßen aus: Eine Markierung wird auf einen Anfangsknoten $q_0 \in Q$ gelegt. Die beiden Spieler ziehen nun abwechselnd die Markierung von einem Knoten zu einem seiner Nachbarknoten. So entsteht eine unendliche Folge von Knoten, eine *Partie* aus $P_{\mathcal{G},q_0}$. Spieler i hat die Partie genau dann gewonnen, wenn die Partie in G_i liegt.

Betrachtet man ein endliches Anfangsstück einer Partie, so führt dies auf den Begriff der *Stellung*.

Definition 3.3 (Stellung) Sei $\mathcal{G} = (Q, Q_1, Q_2, K, W, \Omega)$ ein Spiel.

$$S_{\mathcal{G}} := \{s_0 \ldots s_n \in Q^+ | \forall j \in \{1, \ldots, n\} : (s_{j-1}, s_j) \in K\}$$

bezeichnet die Menge aller *Stellungen* von $\mathcal{G}$. Gilt für ein $s = s_0 \ldots s_n \in S_{\mathcal{G}}$, daß $s_n \in Q_i$, so ist in dieser Stellung Spieler i *am Zug*. Wir bezeichnen die Menge aller Stellungen in denen Spieler i am Zug ist mit $S_{\mathcal{G},i}$.

In einer Stellung, in der ein Spieler am Zug ist, kann er die Folgestellung bestimmen und damit den weiteren Verlauf der Partie beeinflußen. Die Art und Weise in der er seinen Zug wählt, bezeichnet man als *Strategie*.

Definition 3.4 (Strategie) Sei $\mathcal{G} = (Q, Q_1, Q_2, K, W, \Omega)$ ein Spiel. Eine *Strategie* für Spieler i im Spiel $\mathcal{G}$ ist eine Funktion

$$\tau : S_{\mathcal{G},i} \to Q_{3-i}, \qquad s_0 s_1 \dots s_n \mapsto s_{n+1}$$

mit $(s_n, s_{n+1}) \in K$.

$$T_{\mathcal{G},i} = \{\tau | \tau \text{ ist Strategie für Spieler } i \text{ im Spiel } \mathcal{G}\}$$

für $i \in \{1, 2\}$ ist somit die Menge aller Strategien für Spieler i im Spiel $\mathcal{G}$.

Ist ein Spieler in der Stellung $s = s_0 s_1 \dots s_n$ am Zug und spielt gemäß der Strategie τ, so ergibt sich die Folgestellung $s' = s_0 s_1 \dots s_n \tau(s)$.

Führt die Strategie eines Spielers gegen alle möglichen Strategien des Gegners zu einer Gewinnpartie, so bezeichnet man diese Strategie als *Gewinnstrategie*.

Definition 3.5 (Gewinnstrategie) Sind $\tau_1 \in T_{\mathcal{G},1}$ und $q_0 \in Q$ und gilt

$$\forall \tau_2 \in T_{\mathcal{G},2} : q_0 q_1 q_2 \dots \in G_1,$$

$$\text{mit } q_i = \begin{cases} \tau_1(q_0 \dots q_{i-1}) & \text{falls } q_{i-1} \in Q_1 \\ \tau_2(q_0 \dots q_{i-1}) & \text{falls } q_{i-1} \in Q_2 \end{cases} \text{ für alle } i \in I\!N_{>0},$$

so ist τ_1 eine *Gewinnstrategie* für Spieler 1 in $\mathcal{G}$, beim Spielen vom Knoten q_0 aus. Analog definieren wir die *Gewinnstrategien* für Spieler 2. Die Menge der Knoten, von denen aus Spieler i eine Gewinnstrategie hat, nennen wir $\mathcal{W}_{\mathcal{G},i}$. Da $\mathcal{G}$ determiniert ist, gilt

$$\mathcal{W}_{\mathcal{G},1} \cup \mathcal{W}_{\mathcal{G},2} = Q$$

und

$$\mathcal{W}_{\mathcal{G},1} \cap \mathcal{W}_{\mathcal{G},2} = \emptyset.$$

3.2 Strategien

Es existieren verschiedene Typen von Strategien, die sich darin unterscheiden, in welchem Ausmaß sie vom bisherigen Spielverlauf abhängen. Die einfachsten Strategien sind die *No-Memory-Strategien*, die sich nur auf den zuletzt besuchten Knoten beziehen.

Definition 3.6 (No-Memory-Strategie) Eine Strategie $\tau \in T_{\mathcal{G},i}$ heißt *No-Memory-Strategie*, wenn für alle Stellungen $s_1 = q_0 \dots q_j, s_2 = q'_0 \dots q'_k \in \mathrm{Def}(\tau)$ mit $q_j = q'_k$ gilt: $\tau(s_1) = \tau(s_2)$. Die No-Memory-Strategien sind somit genau die Strategien, die nicht vom gesamten bisherigen Spielverlauf, sondern nur vom aktuellen Zustand abhängen.

Ein Spiel $\mathcal{G}$ heißt *No-Memory-Spiel*, wenn von jedem Knoten aus einer der beiden Spieler eine No-Memory-Gewinnstrategie hat.

In manchen Spielen ist es für eine Gewinnstrategie wichtig, im Verlaufe des Spiels immer wieder eine der Endzustandsmengen aus Ω vollständig zu besuchen. Wir möchten für Strategien dieser Art die Bezeichnung *VS-Strategien* verwenden. Das „Visitation Set“ VS einer Stellung soll diejenigen Knoten aus W enthalten, die seit der letzten Rücksetzung des VS besucht wurden. Sobald eine Menge aus Ω in VS enthalten ist, wird das VS auf die leere Menge zurückgesetzt.

Definition 3.7 (VS-Strategie) Sei $\mathcal{G} = (Q, Q_1, Q_2, K, W, \Omega)$ ein Spiel. Das „Visitation Set“ *VS* einer Stellung definieren wir folgendermaßen:

$$\mathrm{VS} : S_{\mathcal{G}} \to 2^W,$$
$$\mathrm{VS}(q) = \emptyset \text{ für alle } q \in Q$$

Sei nun $s \in S_{\mathcal{G}}$ und $q \in Q$, dann ist

$$\mathrm{VS}(sq) = \begin{cases} \emptyset & \text{falls } q \in W \wedge \exists F \in \Omega : F \subseteq \mathrm{VS}(s) \cup \{q\} \\ \mathrm{VS}(s) \cup \{q\} & \text{falls } q \in W \wedge \forall F \in \Omega : F \not\subseteq \mathrm{VS}(s) \cup \{q\} \\ \mathrm{VS}(s) & \text{falls } q \notin W \end{cases}$$

Eine Strategie $\tau \in T_{\mathcal{G},i}$ heißt *VS-Strategie*, wenn für alle Stellungen

$$s_1 = q_0 \dots q_j, s_2 = q'_0 \dots q'_k \in \mathrm{Def}(\tau)$$

mit $\mathrm{VS}(s_1) = \mathrm{VS}(s_2)$ und $q_j = q'_k$ gilt: $\tau(s_1) = \tau(s_2)$.

Ein weiterer Typ von Strategien ist der der *LVR-Strategien*. Diese Strategien zeichnen sich dadurch aus, daß sie nur vom sogenannten „*Last Visitation Record*“ LVR einer Stellung abhängen. Das LVR ist im Wesentlichen ein Element aus der Menge der Permutationen von W, an dem sich ablesen läßt, in welcher Reihenfolge die Knoten aus W zuletzt besucht wurden.

Definition 3.8 (LVR-Strategie) Sei $\mathcal{G} = (Q, Q_1, Q_2, K, W, \Omega)$ ein Spiel. Das „Last Visitation Record“ *LVR* einer Stellung kann man wie folgt induktiv definieren:

$$\mathrm{LVR} : S_{\mathcal{G}} \to \{w_1 \dots w_n \in W^{\leq |W|} | \forall i \neq j \in \{1, \dots, n\} : w_i \neq w_j\},$$

$$\mathrm{LVR}(\varepsilon) = \varepsilon$$

Sei nun $s \in S_{\mathcal{G}}$, $\mathrm{LVR}(s) = w_0 \dots w_n$ und $q \in Q$, dann ist

$$\mathrm{LVR}(sq) = \begin{cases} w_0 \dots w_n & \text{falls } q \notin W \\ w_0 \dots w_n q & \text{falls } q \in W \wedge q \notin \{w_0, \dots, w_n\} \\ w_0 \dots w_{i-1} w_{i+1} \dots q & \text{falls } q \in W \wedge q = w_i \end{cases}$$

Eine Strategie $\tau \in T_{\mathcal{G},i}$ heißt *LVR-Strategie*, wenn für alle Stellungen

$$s_1 = q_0 \dots q_j, s_2 = q'_0 \dots q'_k \in \mathrm{Def}(\tau)$$

mit $\mathrm{LVR}(s_1) = \mathrm{LVR}(s_2)$ und $q_j = q'_k$ gilt: $\tau(s_1) = \tau(s_2)$.

3.3 Beschriftung von Spielgraphen

Ein Spiel $\mathcal{G}$ läßt sich mit Hilfe einer Beschriftungsfunktion β in einen *verallgemeinerten* Muller-Automaten $\mathcal{A}_{\mathcal{G},\Sigma,\beta}$ über einem passenden Alphabet Σ überführen. Mit *verallgemeinert* ist in diesem Zusammenhang, das Fehlen eines ausgezeichneten Anfangszustandes gemeint. Ein Alphabet ist *passend*, wenn die maximale Anzahl von Kanten, die von einen Knoten in $\mathcal{G}$ ausgehen, kleiner oder gleich dessen Mächtigkeit ist. Es werden dann alle Kanten des Graphen derart beschriftet, daß sich ein deterministischer Automat ergibt.

Definition 3.9 (Beschriftung) Sei $\mathcal{G} = (Q, Q_1, Q_2, K, W, \Omega)$ ein Spiel, Σ ein Alphabet mit

$$|\Sigma| \geq \max_{q \in Q}(d_+(q))$$

und

$$\beta : K \to 2^{\Sigma} \text{ mit } \forall q \in Q : \dot{\bigcup_{(q,q') \in K}} \beta((q, q')) = \Sigma$$

dann definieren wir $\mathcal{A}_{\mathcal{G},\Sigma,\beta} := (Q, \Sigma, \delta, \mathcal{F})$ wobei $\delta : Q \times \Sigma \to Q$, $(q, a) \mapsto q'$ falls ein $k = (q, q') \in K$ mit $a \in \beta(k)$ existiert und $\mathcal{F} = \{F \in 2^Q | F \cap W \in \Omega\}$.

Wählt man nun noch einen Anfangszustand $q_0 \in Q$, so läßt sich die von $\mathcal{A}_{\mathcal{G},\Sigma,\beta} = (Q, \Sigma, \delta, \mathcal{F})$ erkannte Sprache definieren als $\mathrm{L}_{q_0}(\mathcal{A}_{\mathcal{G},\Sigma,\beta}) = \mathrm{L}(\mathcal{A}')$. $\mathcal{A}' = (Q, \Sigma, \delta, q_0, \mathcal{F})$ ist dabei ein Muller-Automat nach üblicher Definition. Mit $\mathrm{L}(\mathcal{A}_{\mathcal{G},\Sigma,\beta})$ bezeichnen wir $\{\mathrm{L}_q(\mathcal{A}_{\mathcal{G},\Sigma,\beta}) | q \in Q\}$, die Menge aller von diesem Automaten erkennbaren Sprachen, bei entsprechender Wahl des Anfangszustandes.

Im Folgenden wollen wir uns mit der topologischen Zugehörigkeit der Sprachen aus $\mathrm{L}(\mathcal{A}_{\mathcal{G},\Sigma,\beta})$ beschäftigen. Landweber stellte in [Lan69] eine Charakterisierung von Automaten bezüglich der 1-Akzeptanz und 2-Akzeptanz vor. Eine übersichtliche Darstellung findet man in [Tho90]:

Lemma 3.1 Es gibt effektive Verfahren einen Muller-Automaten $\mathcal{A}$ in einen Muller-Automaten $\mathcal{A}_1$ bzw. $\mathcal{A}_2$ zu überführen, so daß $\mathrm{L}(\mathcal{A}_1)$ 1-erkennbar ist, $\mathrm{L}(\mathcal{A}_2)$ 2-erkennbar ist und

$$\mathrm{L}(\mathcal{A}) \in G \Longleftrightarrow \mathcal{A} = \mathcal{A}_1$$

sowie

$$\mathrm{L}(\mathcal{A}) \in G_\delta \Longleftrightarrow \mathcal{A} = \mathcal{A}_2.$$

□

Wir möchten im folgenden ein ähnliches Lemma beweisen, daß sich mit verallgemeinerten Muller-Automaten beschäftigt. Der Beweis ist in großen Teilen identisch mit dem aus [Tho90]. Die konstruktiven Elemente des Beweises werden später bei der Bestimmung von Strategien wichtig sein.

Lemma 3.2 Es gibt effektive Verfahren einen verallgemeinerten Muller-Automaten $\mathcal{A}$ in verallgemeinerte Muller-Automaten $\mathcal{A}_1$ bzw. $\mathcal{A}_2$ zu überführen, so daß $\mathrm{L}(\mathcal{A}_1) \subseteq G$, $\mathrm{L}(\mathcal{A}_2) \subseteq G_\delta$ und

$$\mathrm{L}(\mathcal{A}) \subseteq G \Longleftrightarrow \mathcal{A} = \mathcal{A}_1$$

sowie

$$\mathrm{L}(\mathcal{A}) \subseteq G_\delta \Longleftrightarrow \mathcal{A} = \mathcal{A}_2.$$

Beweis Sei $\mathcal{A} = (Q, \Sigma, \delta, \mathcal{F})$ ein verallgemeinerter Muller-Automat. Als „Schleife von $\mathcal{A}$“ bezeichnen wir eine stark zusammenhängende Teilmenge von Q, wobei wir $\mathcal{A}$ als einen gerichteten Graphen betrachten.

(1) Wir nennen einen Zustand q von $\mathcal{A}$ genau dann einen „$\mathcal{F}$-Zustand“, wenn q in einer Schleife von $\mathcal{A}$ liegt, die eine Menge aus $\mathcal{F}$ bildet. Sei

nun $\mathcal{F}_1$ die Menge, die wir erhalten, wenn wir $\mathcal{F}$ um die Schleifen von $\mathcal{A}$ erweitern, die von einem $\mathcal{F}$-Zustand aus erreichbar sind. $\mathcal{A}_1 = (Q, \Sigma, \delta, \mathcal{F}_1)$ ist dann der gesuchte verallgemeinerte Muller-Automat.

Setzen wir $F_1 := \bigcup_{F \in \mathcal{F}_1} F$, so ist $\mathcal{A}_1$ äquivalent zu $\mathcal{A}_1' = (Q, \Sigma, \delta, F_1)$ mit 1-Akzeptanz, denn für jedes $q_0 \in Q$ und jedes $\alpha \in \Sigma^\omega$ gilt: Ein Lauf von $\mathcal{A}_1'$ auf α von q_0 aus erreicht genau dann einen Zustand aus F_1, wenn ein Lauf von $\mathcal{A}_1$ auf α von q_0 aus schließlich nur noch Zustände in einer Schleife aus $\mathcal{F}_1$ liefert.

Ist nun $\mathcal{A} = \mathcal{A}_1$, so gilt auch $\mathrm{L}(\mathcal{A}) \subseteq G$. Zu zeigen bleibt also noch, daß $\mathcal{A} = \mathcal{A}_1$ auch aus $\mathrm{L}(\mathcal{A}) \subseteq G$ folgt. Hierzu müssen wir zeigen, daß $\mathcal{F}_1 \subseteq \mathcal{F}$ gilt.

Sei $F \in \mathcal{F}_1$, dann existiert ein $\mathcal{F}$-Zustand q von $\mathcal{A}$, von dem aus die Schleife F erreicht werden kann. Da q ein $\mathcal{F}$-Zustand ist, gibt es ein ω-Wort $\alpha \in \mathrm{L}_q(\mathcal{A})$, auf dem $\mathcal{A}$ schließlich in eine Schleife aus $\mathcal{F}$ grät, die q enthält. Da $\mathrm{L}_q(\mathcal{A}) \in G$ ist, existiert ein reguläres $W \subseteq \Sigma^\omega$ mit $\mathrm{L}_q(\mathcal{A}) = W\Sigma^\omega$. Somit gibt es ein $w \in W$, das Präfix von α ist. Weiterhin gilt für alle $\beta \in \Sigma^\omega$: $w\beta \in \mathrm{L}_q(\mathcal{A})$. Jede Schleife, die von q aus erreichbar ist, und somit auch F, wird folglich von einem ω-Wort aus $\mathrm{L}_q(\mathcal{A})$ induziert. Es folgt $F \in \mathcal{F}$.

(2) $\mathcal{A}_2$ wird wie folgt aus $\mathcal{A}$ durch Erweiterung von $\mathcal{F}$ zu $\mathcal{F}_2$ konstruiert:

Für jedes q in einer Schleife $F \in \mathcal{F}$ und jede Schleife E, die q enthält, füge $F \cup E$ zu $\mathcal{F}_2$ hinzu.

$\mathrm{L}_{q_0}(\mathcal{A}_2)$ ist 2-erkennbar für alle $q_0 \in Q$: Wir konstruieren hierzu einen deterministischen Büchi-Automaten $\mathcal{A}_2'$, der wie $\mathcal{A}$ arbeitet und sich zusätzlich in einem „Zustandsspeicher“ S die bereits besuchten Zustände merkt, und diesen stets auf $\emptyset$ zurücksetzt,wenn S eine $\mathcal{F}$-Menge enthält. Die Zustände von $\mathcal{A}_2'$ haben die Form $(q, S) \in Q \times 2^Q$ und der Automat gelangt beim Lesen des Buchstabens $a \in \Sigma$ vom Zustand (q, S) in den Zustand $(\delta(q, a), S \cup \{\delta(q, a)\})$ falls $S \cup \{\delta(q, a)\}$ keine $\mathcal{F}$-Menge enthält, und in den Zustand $(\delta(q, a), \emptyset)$ andernfalls. Der Anfangszustand ist $(q_0, \emptyset)$ und $\{(q, \emptyset) | q \in Q\}$ ist die Endzustandsmenge.

2-Akzeptanz für $\mathcal{A}_2'$ bedeutet, daß $\mathcal{A}_2'$ auf einem $\alpha \in \Sigma^\omega$ schließlich nur noch Zustände aus einer Schleife F' annimmt, die die Erweiterung einer Menge aus $\mathcal{F}$ ist. Es gilt $\alpha \in \mathrm{L}(\mathcal{A}_2') \iff \alpha \in \mathrm{L}_{q_0}(\mathcal{A}_2)$, und damit $\mathrm{L}_{q_0}(\mathcal{A}_2) \in G_\delta$ für alle $q_0 \in Q$, also $\mathrm{L}(\mathcal{A}_2) \subseteq G_\delta$. Gilt $\mathcal{A} = \mathcal{A}_2$, so gilt auch $\mathrm{L}(\mathcal{A}) \subseteq G_\delta$.

Gelte nun $\mathrm{L}(\mathcal{A}) \subseteq G_\delta$ und seien $F \in \mathcal{F}$ und $E \subseteq Q$ zwei Schleifen mit gemeinsamen Zustand q. Wir zeigen $F \cup E \in \mathcal{F}$ (und somit $\mathcal{F} = \mathcal{F}_2$).

Nach Voraussetzung ist $\mathrm{L}_q(\mathcal{A}) \in G_\delta$, also etwa $\mathrm{L}_q(\mathcal{A}) = \lim W$ für ein reguläres $W \subseteq \Sigma^\star$. Über die Schleife F können wir ein ω-Wort $\alpha \in \Sigma^\omega$ konstruieren, das ein Präfix $u_1 \in W$ hat. Von $\delta^\star(q, u_1)$ laufen wir die Schleife F weiter, bis wir wieder bei q angelangt sind (sagen wir mit dem Wort v_1), und

weiter über die Schleife E zurück nach q (sagen wir mit dem Wort w_1). Wiederholtes Anwenden dieses Verfahrens liefert ein ω-Wort $u_1v_1w_1u_2v_2w_2\ldots$, das in $\lim W$ enthalten ist und $\mathcal{A}$ schließlich genau die Zustände aus $F \cup E$ annehmen läßt. Somit gilt $F \cup E \in \mathcal{F}$.

□

Wir haben mit dem letzten Lemma gezeigt, daß bei einem beschrifteten Spielgraphen die Zugehörigkeit der von ihm erkannten Sprachen zu einer Klasse der Borel-Hierarchie effektiv getestet werden kann. Es drängt sich jetzt natürlich die Frage auf, ob ein Spiel durch verschiedene Beschriftungen auch Sprachen aus verschiedenen Borel-Klassen erkennt. Genauere Betrachtung des Beweises von Lemma 3.2 läßt die interessante Schlußfolgerung zu, daß sich durch verschiedene Beschriftungen keineswegs andere Borel-Klassen erreichen lassen.

Folgerung 3.1 Ist $\mathcal{G} = (Q, Q_1, Q_2, K, W, \Omega)$ ein Spiel und sind β,β' Beschriftungen über Σ, so gilt

$$\mathrm{L}(\mathcal{A}_{\mathcal{G},\Sigma,\beta}) \subseteq G \Longleftrightarrow \mathrm{L}(\mathcal{A}_{\mathcal{G},\Sigma,\beta'}) \subseteq G$$

sowie

$$\mathrm{L}(\mathcal{A}_{\mathcal{G},\Sigma,\beta}) \subseteq G_\delta \Longleftrightarrow \mathrm{L}(\mathcal{A}_{\mathcal{G},\Sigma,\beta'}) \subseteq G_\delta.$$

Die Gültigkeit dieser Behauptung läßt sich aus dem Lemma 3.2 ableiten, da in dessen Beweis nur vergrößerte Endzustandsmengen konstruiert werden, ohne auf konkrete Beschriftungen von Transitionen Bezug zu nehmen.

□

Da nun die Sprachen, die von einem beschrifteten Spiel erkannt werden, in der Borel-Klassifizierung nicht von der Beschriftung abhängig sind, bietet es sich an, diese Klassifizierung auf Spiele auszuweiten. Wir werden also im Folgenden auch Spiele in die Borel-Hierarchie einordnen.

Definition 3.10 Sei $\mathcal{G}$ ein Spiel und β eine beliebige Beschriftung über einem passenden Alphabet Σ. Wir definieren nun

$$\mathcal{G} \text{ ist ein Spiel aus } H \Longleftrightarrow \mathrm{L}(\mathcal{A}_{\mathcal{G},\Sigma,\beta}) \subseteq H$$

für alle Sprachklassen $H \subseteq 2^{\Sigma^\omega}$.

Wir interessieren uns hauptsächlich für Sprachklassen aus der Borel-Hierarchie, deshalb ist für uns wichtig:

$$\begin{aligned}
\mathcal{G} \text{ ist ein Spiel aus } G &\iff \mathrm{L}(\mathcal{A}_{\mathcal{G},\Sigma,\beta}) \subseteq G \\
\mathcal{G} \text{ ist ein Spiel aus } G_\delta &\iff \mathrm{L}(\mathcal{A}_{\mathcal{G},\Sigma,\beta}) \subseteq G_\delta. \\
\mathcal{G} \text{ ist ein Spiel aus } F &\iff \mathrm{L}(\mathcal{A}_{\mathcal{G},\Sigma,\beta}) \subseteq F \\
\mathcal{G} \text{ ist ein Spiel aus } F_\sigma &\iff \mathrm{L}(\mathcal{A}_{\mathcal{G},\Sigma,\beta}) \subseteq F_\sigma.
\end{aligned}$$

Eine explizite Beschriftung eines Spiels zur Klassifizierung ist nicht unbedingt erforderlich, wie die folgende Bemerkung zeigt.

Bemerkung 3.1 Man kann bei der Klassifizierung eines Spiels auch auf eine Beschriftung verzichten, indem man die Partien aus Q^ω betrachtet und die Borel-Hierarchie analog durch eine Definition der Cantor-Topologie auf Q^ω charakterisiert.

Diese Vorgehensweise kann man auch als eine spezielle Beschriftung verstehen, also etwa $\beta((q,q')) = q'$. Diese Beschriftung ist zwar nicht vollständig, aber die Zugehörigkeit des Spiels zu einer Klasse der Borel-Hierarchie wird hierdurch nicht verfälscht.

Die Einordnung in die Borel-Klassen F und F_σ läßt sich nicht direkt über die Konstruktion aus Lemma 3.2 erreichen. Wenn man jedoch von den komplementären Automaten der beschrifteten Spiele, das heißt den Automaten mit $2^Q \setminus \mathcal{F}$ anstelle von $\mathcal{F}$, ausgeht, liefert die Konstruktion genau die gewünschte Eigenschaft.

3.4 Komplemente

Der Begriff des Komplements läßt sich nicht ohne weiteres von den Sprachen und Automaten auf Spiele übertragen. Schuld daran ist eine latente Asymmetrie der Spieldefinition, in der die Gewinnmenge immer in Bezug auf den ersten Spieler angegeben wird.

Definition 3.11 (Komplement) Es gibt zwei Möglichkeiten das Komplement eines Spiels $\mathcal{G} = (Q, Q_1, Q_2, K, W, \Omega)$ zu definieren:

$$\begin{aligned}
\overline{\mathcal{G}}^Q &= (Q, Q_2, Q_1, K, W, \Omega) \\
\overline{\mathcal{G}}^L &= (Q, Q_1, Q_2, K, W, 2^W \setminus \Omega)
\end{aligned}$$

Im ersten Fall tauschen Spieler 1 und Spieler 2 die Rollen, im zweiten Fall wird die Gewinnbedingung umgekehrt. Wenden wir auf ein Spiel $\mathcal{G}$ beide Komplementfuntionen an, so erhalten wir $\overline{\overline{\mathcal{G}}^L}^Q = \overline{\overline{\mathcal{G}}^Q}^L$ und stellen fest, daß es sich dabei um dasselbe Spiel wie $\mathcal{G}$ handelt, nur mit vertauschten Seiten. Insbesondere hat dieses Spiel die gleichen Gewinnstrategien wie $\mathcal{G}$, nur jeweils für den anderen Spieler. Wir wollen daher $\overline{\overline{\mathcal{G}}^L}^Q$ zu $\mathcal{G}$ *strategieäquivalent* nennen.

Zur Verdeutlichung der Wirkung der verschiedenen Komplemente auf ein Spiel, wollen wir an dieser Stelle das Schachspiel heranziehen, obwohl es natürlich weder ein klassisches unendliches Spiel ist, noch einen Wert aus $\{-1, +1\}$ hat, aber diese beiden Aspekte sind für die Veranschaulichung der beiden Komplemente nicht wesentlich. Sei also $\mathcal{G}$ im Folgenden das Schachspiel:

- $\overline{\mathcal{G}}^L$ ist ein Schachspiel, bei dem Weiß gewinnt, wenn der weiße König Schachmatt gesetzt wird und vice versa.
- $\overline{\mathcal{G}}^Q$ ist ein Schachspiel, bei dem jeder Spieler die gegnerischen Figuren zieht.
- $\overline{\overline{\mathcal{G}}^L}^Q$ ist ein Schachspiel mit vertauschten Seiten, $\overline{\overline{\mathcal{G}}^L}^Q$ ist somit strategieäquivalent zu $\mathcal{G}$.

Wie man leicht sieht, beschreiben auch $\overline{\mathcal{G}}^L$ und $\overline{\mathcal{G}}^Q$ ein und dasselbe Spiel, nämlich eine Art „Räuberschach“ ohne Schlagzwang, mit dem einzigen Unterschied, daß die Seiten vertauscht sind. Diese Strategieäquivalenz von $\overline{\mathcal{G}}^L$ und $\overline{\mathcal{G}}^Q$ ergibt sich auch ganz formal, denn

$$\overline{\mathcal{G}}^L = \overline{\overline{\overline{\mathcal{G}}^Q}^L} \text{ ist strategieäquivalent zu } \overline{\mathcal{G}}^Q.$$

Doch jetzt wieder zurück zu unseren unendlichen Graph-Spielen. Betrachtet man die Sprachen der durch eine Beschriftung der Komplement-Spiele hervorgehenden Muller-Automaten, so sieht man leicht, daß diese in verschiedenen, komplementären topologischen Klassen liegen.

Lemma 3.3 Es gilt für eine beliebige Beschriftung β über passendem Alphabet Σ und einen beliebigen Anfangszustand q_0:

1.
$$\mathrm{L}_{q_0}(\mathcal{A}_{\overline{\mathcal{G}}^L,\Sigma,\beta}) = \Sigma^\omega \setminus \mathrm{L}_{q_0}(\mathcal{A}_{\mathcal{G},\Sigma,\beta})$$
und somit auch
$$\begin{aligned} \mathcal{G} \text{ ist ein Spiel aus } G &\iff \overline{\mathcal{G}}^L \text{ ist ein Spiel aus } F \\ \mathcal{G} \text{ ist ein Spiel aus } G_\delta &\iff \overline{\mathcal{G}}^L \text{ ist ein Spiel aus } F_\sigma. \end{aligned}$$

2.
$$\mathrm{L}_{q_0}(\mathcal{A}_{\overline{\mathcal{G}}^Q,\Sigma,\beta}) = \mathrm{L}_{q_0}(\mathcal{A}_{\mathcal{G},\Sigma,\beta})$$

Beweis Sei $\mathcal{A}_{\mathcal{G},\Sigma,\beta} = (Q, \Sigma, \delta, \mathcal{F})$, dann gilt $\mathcal{A}_{\overline{\mathcal{G}}^L,\Sigma,\beta} = (Q, \Sigma, \delta, 2^Q \setminus \mathcal{F})$ und $\mathcal{A}_{\mathcal{G},\Sigma,\beta} = \mathcal{A}_{\overline{\mathcal{G}}^Q,\Sigma,\beta}$. Seien $\alpha \in \Sigma^\omega$ und $q_0 \in Q$, dann sei $\sigma = q_0 q_1 \ldots \in Q^\omega$ der eindeutig bestimmte Lauf von $\mathcal{A}_{\mathcal{G},\Sigma,\beta}$ auf α. Ist $\mathrm{In}(\sigma) \in \mathcal{F}$, so akzeptieren $\mathcal{A}_{\mathcal{G},\Sigma,\beta}$ und $\mathcal{A}_{\overline{\mathcal{G}}^Q,\Sigma,\beta}$ aber $\mathcal{A}_{\overline{\mathcal{G}}^L,\Sigma,\beta}$ nicht. Gilt aber $\mathrm{In}(\sigma) \notin \mathcal{F}$, dann ist $\mathrm{In}(\sigma) \in 2^Q \setminus \mathcal{F}$ und $\mathcal{A}_{\overline{\mathcal{G}}^L,\Sigma,\beta}$ akzeptiert α, aber weder $\mathcal{A}_{\overline{\mathcal{G}}^Q,\Sigma,\beta}$ noch $\mathcal{A}_{\mathcal{G},\Sigma,\beta}$ akzeptieren α. Somit gilt

$$\alpha \in \mathrm{L}_{q_0}(\mathcal{A}_{\mathcal{G},\Sigma,\beta}) \Leftrightarrow \alpha \in \mathrm{L}_{q_0}(\mathcal{A}_{\overline{\mathcal{G}}^Q,\Sigma,\beta})$$

und

$$\alpha \in \mathrm{L}_{q_0}(\mathcal{A}_{\mathcal{G},\Sigma,\beta}) \Leftrightarrow \alpha \notin \mathrm{L}_{q_0}(\mathcal{A}_{\overline{\mathcal{G}}^L,\Sigma,\beta}).$$

□

An dieser Stelle wollen wir uns die Frage stellen, ob sich aus der im vorigen Abschnitt eingeführte Einordnung der Spiele in die Borel-Hierarchie auch eine Klassifizierung der Spiele bezüglich deren Gewinnstrategien ergibt. Die Stufen der Borel-Hierarchie $G \cup F$, $G_\delta \cup F_\sigma$ und $G_{\delta\sigma} \cap F_{\sigma\delta}$ sind abgeschlossen bezüglich des Sprachen-Komplements. Man könnte jetzt vermuten, daß die Strategie-Klassen der in dieser Hierarchie liegenden Spiele direkt mit den Sprach-Klassen korrespondieren. An einem einfachen Beispiel können wir jedoch zeigen, daß dies nicht der Fall sein kann.

Satz 3.1 Die Klasse der No-Memory Spiele ist *nicht* abgeschlossen gegen Komplementbildung.

Beweis Man betrachte das Spiel $\mathcal{G}$ aus Abbildung 3.1, wobei $Q_1 = \{1, 3, 5\}$, $Q_2 = \{2, 4, 6\}$, $W = \{1, 2\}$ und $\Omega = \{\{1\}, \{2\}\}$ ist. Wie man leicht sieht, hat Spieler 2 eine No-Memory Gewinnstrategie beim Spiel von beliebigem Startknoten, indem er immer von 2 nach 1, von 4 nach 3, und von 6 nach 5 zieht. Er erreicht damit, daß entweder Knoten 1 und 2, oder keiner von beiden unendlich oft besucht wird. $\mathcal{G}$ ist somit ein No-Memory Spiel.

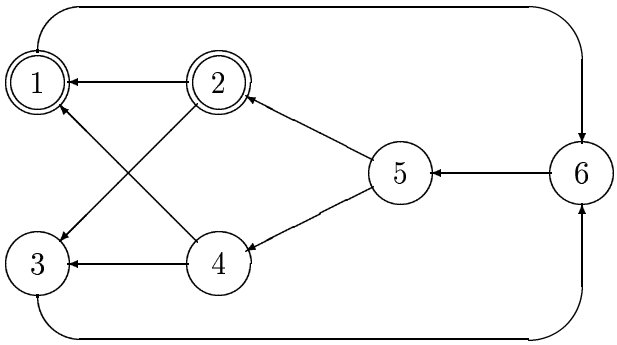


Abbildung 3.1: No-Memory-Spiel dessen Komplement kein No-Memory-Spiel ist

Betrachtet man nun das Spiel $\overline{\mathcal{G}}^L = (Q, Q_1, Q_2, K, W, 2^W \setminus \Omega)$, so zeigt sich, daß nun Spieler 1 eine LVR-Gewinnstrategie beim Spiel von beliebigem Startknoten hat, indem er, falls Knoten 1 im LVR als letzte Komponente steht, von 5 nach 2 zieht, und ansonsten nach 4. Eine No-Memory-Gewinnstrategie hat Spieler 1 in diesem Spiel jedoch nicht, denn zieht Spieler 1 von 5 immer nach 2, so zieht Spieler 2 daraufhin immer nach 1 und gewinnt, zieht Spieler 1 von 5 immer nach 4, so zieht Spieler 2 daraufhin immer nach 3 und gewinnt dann ebenfalls. $\overline{\mathcal{G}}^L$ ist somit *kein* No-Memory Spiel.

□

Die fehlende Abgeschlossenheit der No-Memory-Spiele gegen Komplementbildung bedeutet für uns, daß wir nicht hoffen dürfen, die Strategieklassen der unendlichen Spiele mit den Borel-Klassen zur Deckung bringen zu können.

Aus dem Begriff der Strategieäquivalenz läßt sich jedoch eine interessante Eigenschaft der Borel-Klassen entwickeln.

Lemma 3.4 Ist H eine Sprach-Klasse und gilt für alle $\mathcal{G}$ mit $\mathrm{L}(\mathcal{A}_{\mathcal{G},\Sigma,\beta}) \subseteq H$: $\mathcal{G}$ ist ein Spiel der Strategieklasse S, so gilt dies auch für alle $\mathcal{G}$ mit $\mathrm{L}(\mathcal{A}_{\mathcal{G},\Sigma,\beta}) \subseteq Co\text{-}H$, wobei $Co\text{-}H = \{L \in \Sigma^\omega | \Sigma^\omega \setminus L \in H\}$.

Beweis Sei $\mathcal{G}$ ein Spiel mit $\mathrm{L}(\mathcal{A}_{\mathcal{G},\Sigma,\beta}) \subseteq Co\text{-}H$ und $\mathcal{G}' = \overline{\overline{\mathcal{G}}^L}^Q$, dann gilt $\mathrm{L}(\mathcal{A}_{\mathcal{G}',\Sigma,\beta}) \subseteq H$, und somit ist $\mathcal{G}'$ ein Spiel der Strategieklasse S. Da $\mathcal{G}'$ und $\mathcal{G}$ strategieäquivalent sind, ist auch $\mathcal{G}$ ein Spiel der Strategieklasse S.

□

Dieses Lemma hat eine sehr schöne Konsequenz für die Untersuchung der Strate-

gien von Spielen aus einer Klasse der Borel-Hierarchie. Haben nämlich alle Spiele einer bestimmten Borel-Klasse Gewinnstrategien aus der gleichen Strategieklasse, so haben auch alle Spiele aus der komplementären Borel-Klasse Gewinnstrategien aus dieser Strategieklasse.

Wir werden dies anwenden, indem wir zuerst zeigen, daß Spiele aus G No-Memory-Gewinnstrategien besitzen, um dann zu folgern, daß auch die Spiele aus F No-Memory-Gewinnstrategien haben. Der gleiche Schluß ergibt sich für den Fall G_δ und F_σ, nur daß es dort um VS-Gewinnstrategien geht.

Kapitel 4

Strategien und Borel-Klassen

Im vorigen Kapitel haben wir einige Zusammenhänge von Spiel-Klassen bezüglich der Borel-Hierarchie und der Klassifizierung derer Gewinnstrategien gezeigt. Wir konnten zwar schon erkennen, daß die Borel-Spiel-Klassen nicht exakt mit den Strategie-Klassen korrespondieren, denn nach Lemma 3.1 ist die Klasse der No-Memory-Spiele nicht gegen Komplement abgeschlossen, aber wir sind jetzt in der Lage zu zeigen, daß die Zugehörigkeit eines Spiels zu einer Borel-Klasse eine hinreichende Bedingung für die Existenz einer Gewinnstrategie aus einer dazugehörigen Strategie-Klasse ist. Wir werden im Folgenden zeigen:

1. Spiele aus G und F erlauben No-Memory-Gewinnstrategien.
2. Spiele aus G_δ und F_σ erlauben VS-Gewinnstrategien.
3. Reguläre Spiele erlauben LVR-Gewinnstrategien.

Das Ergebnis in 3 ist schon länger bekannt und wird unter anderem von Robert McNaughton in [McN93] konstruktiv bewiesen. Bei der Untersuchung der Spiele aus G und F bzw. aus G_δ und aus F_σ brauchen wir uns nur auf Spiele aus G bzw. G_δ zu konzentrieren, denn Lemma 3.4 liefert uns sofort die Aussage für die komplementären Spielklassen.

Wir werden in diesem Kapitel schon auf Definitionen und Sätze aus Kapitel 5 vorgreifen. Die dort beschriebenen Spiele und deren Gewinnstrategien basieren auf speziellen Automaten und sollen deshalb erst anschließend an die Betrachtung der Spiele auf Muller-Automaten behandelt werden.

4.1 Spiele in G und F

Sei $\mathcal{G}$ ein Spiel aus G oder F. Wenn das Spiel aus F ist, so bilden wir das strategieäquivalente Spiel $\overline{\overline{\mathcal{G}}^L}^Q$, welches aus G ist. Wir gehen daher ohne Beschränkung der Allgemeinheit im Folgenden von einem Spiel in G aus. Um eine Gewinnstrategie zu bestimmen, konstruieren wir in drei Schritten aus $\mathcal{G} = (Q, Q_1, Q_2, K, W, \Omega)$ ein Spiel $\mathcal{G}'$ mit 1-Akzeptanz:

Zunächst überführen wir $\mathcal{G}$ mit einer Beschriftung β über einem passenden Alphabet Σ in einen verallgemeinerten Muller-Automaten $\mathcal{A}_{\mathcal{G},\Sigma,\beta} = (Q, \Sigma, \delta, \mathcal{F})$, wobei $\mathcal{F} = \{M \in 2^Q | M \cap W \in \Omega\}$. Dann führen wir die in Lemma 3.2 beschriebene Konstruktion von F durch, indem wir alle Schleifen aus $\mathcal{F}$ vereinigen. Das gesuchte Spiel mit 1-Akzeptanz sieht dann folgendermaßen aus:

$\mathcal{G}' = (Q, Q_1, Q_2, K, F)$ mit $F = \{M \in 2^Q | M \cap W \in \Omega$ und M ist Schleife von $\mathcal{G}\}$

$\mathcal{G}'$ steht in engem Zusammenhang mit dem Spiel $\mathcal{G}$. Wir stellen folgende Einzelheiten fest:

Lemma 4.1 Sei $\mathcal{G} = (Q, Q_1, Q_2, K, W, \Omega)$ ein Spiel aus G und $\mathcal{G}' = (Q, Q_1, Q_2, K, F)$ wie oben konstruiert, dann gilt:

1. Jede Partie p von $\mathcal{G}$ ist auch eine Partie von $\mathcal{G}'$ und umgekehrt.
2. Jede Gewinnpartie von $\mathcal{G}$ ist auch Gewinnpartie von $\mathcal{G}'$ und umgekehrt.

Beweis Die Aussage 1 ergibt sich zwangsläufig, da $\mathcal{G}$ und $\mathcal{G}'$ den gleichen Spielgraphen haben, und daher auch die gleichen Partien erlauben. Die Aussage 2 ergibt sich aus dem Beweis von Lemma 3.2. Dort haben wir gesehen, wie sich aus einem verallgemeinerten Muller-Automaten, der nur Sprachen aus G erkennt, ein 1-akzeptierender deterministischer endlicher Automat konstruieren läßt, der zu diesem äquivalent ist. Die Konstruktion von $\mathcal{G}'$ ist nur eine Erweiterung dieser Konstruktion auf Spiele.

Sei β eine Beschriftung von $\mathcal{G}$ mit passendem Alphabet Σ, dann gilt $\mathrm{L}(\mathcal{A}_{\mathcal{G},\Sigma,\beta}) \subseteq G$, da $\mathcal{G}$ ein Spiel aus G ist. Haben wir eine Partie p von $\mathcal{G}$, so ist diese ein akzeptierender Lauf auf $\mathcal{A}_{\mathcal{G},\Sigma,\beta}$. Der 1-akzeptierende Automat $\mathcal{A}'_1$ aus dem Beweis von Lemma 3.2 ist äquivalent zu $\mathcal{A}_{\mathcal{G},\Sigma,\beta}$. Das Spiel $\mathcal{G}'$ enspricht dem Automaten $\mathcal{A}'_1$, und ein akzeptierender Lauf auf $\mathcal{A}'_1$ ist eine Gewinnpartie von $\mathcal{G}'$. Somit ist p genau dann Gewinnpartie von $\mathcal{G}$, wenn p Gewinnpartie von $\mathcal{G}'$ ist.

□

Die Gewinnstrategien für ein Spiel $\mathcal{G}$ aus G läßt sich jetzt folgendermaßen bestimmen: Wir konstruieren ein äquivalentes Spiel $\mathcal{G}'$ mit 1-Akzeptanz, bestimmen für dieses dessen No-Memory-Gewinnstrategien und übertragen die Strategien wieder zurück auf $\mathcal{G}$.

Satz 4.1 Ist $\mathcal{G} = (Q, Q_1, Q_2, K, W, \Omega)$ ein Spiel aus G oder aus F, so ist $\mathcal{G}$ ein No-Memory-Spiel.

Beweis Ist $\mathcal{G}$ ein Spiel aus F so betrachten wir statt $\mathcal{G}$ das strategieäquivalente Spiel $\overline{\overline{\mathcal{G}}^L}^Q$, das aus G ist. Sei also ohne Beschränkung der Allgemeinheit $\mathcal{G}$ aus G. Wir konstruieren dann $\mathcal{G}' = (Q, Q_1, Q_2, K, F)$ mit

$$F = \{M \in 2^Q | M \cap W \in \Omega \text{ und } M \text{ ist Schleife von } \mathcal{G}\}.$$

$\mathcal{G}'$ ist ein Spiel mit 1-Akzeptanz und hat nach Satz 5.1 No-Memory-Gewinnstrategien τ_1 und τ_2 auf den Gewinnknotenmengen $\mathcal{W}_{\mathcal{G},1}$ bzw. $\mathcal{W}_{\mathcal{G},2}$ für Spieler 1 bzw. Spieler 2. Nach Lemma 4.1 ist jede Gewinnpartie von $\mathcal{G}'$ auch Gewinnpartie von $\mathcal{G}$. Somit sind τ_1 und τ_2 auch Gewinnstrategien für $\mathcal{G}$.

□

4.2 Spiele in G_δ und F_σ

Sei im $\mathcal{G}$ ein Spiel aus G_δ oder F_σ. Wenn das Spiel aus F_σ ist, so bilden wir das strategieäquivalente Spiel $\overline{\overline{\mathcal{G}}^L}^Q$, welches aus G_δ ist. Wir gehen daher im Folgenden ohne Beschränkung der Allgemeinheit von einem Spiel in G_δ aus. Um eine Gewinnstrategie zu bestimmen, konstruieren wir aus $\mathcal{G} = (Q, Q_1, Q_2, K, W, \Omega)$ ein deterministisches Büchi-Spiel $\mathcal{G}' = (Q', Q_1', Q_2', K', F)$, mit

$$\begin{aligned}
Q' &= Q \times 2^W, \\
Q_1' &= Q_1 \times 2^W, \\
Q_2' &= Q_2 \times 2^W, \\
K' &= \{((q,S),(q',S')) \in Q' \times Q' | (q,q') \in K \wedge S' = \rho(S,q')\}
\end{aligned}$$

$$\text{mit } \rho(S,q) = \begin{cases} S \cup \{q\} & \text{falls } S \cup \{q\} \notin \Omega \wedge q \in W \\ \emptyset & \text{falls } S \cup \{q\} \in \Omega \wedge q \in W \\ S & \text{falls } q \notin W \end{cases} \quad \text{und}$$

$$F = \{(q, \emptyset) | q \in Q\}.$$

Weiterhin definieren wir eine Abbildung π, die Zustände aus $\mathcal{G}'$ auf die Zustände aus $\mathcal{G}$ abbildet:

$$\pi : Q' \to Q, (q_0, S_0)(q_1, S_1) \ldots \mapsto q_0 q_1 \ldots$$

Das deterministische Büchi-Spiel $\mathcal{G}'$ steht mit $\mathcal{G}$ in folgender Beziehung:

Lemma 4.2 Sei $\mathcal{G} = (Q, Q_1, Q_2, K, W, \Omega)$ ein Spiel aus G_δ und $\mathcal{G}' = (Q', Q'_1, Q'_2, K', F)$ wie oben konstruiert, dann gilt:

1. Für jede Partie $p = p_0 p_1 \ldots$ von $\mathcal{G}$ existiert genau eine Partie $p' = (p'_0, \emptyset)(p'_1, S_1) \ldots$ von $\mathcal{G}'$ mit $\pi(p') = p$. Es gilt dann $\forall i \geq 0 : S_i = \mathrm{VS}(p_0, \ldots, p_i)$.
2. $p' = (q_0, \emptyset)(q_1, S_1) \ldots$ ist genau dann eine Gewinnpartie von $\mathcal{G}'$ für Spieler i, wenn $\pi(p')$ eine Gewinnpartie von $\mathcal{G}$ für Spieler i ist.

Beweis Zu 1: Sei $p = q_0 q_1 \ldots$ eine Partie von $\mathcal{G}$, dann ist $p' = (q_0, \emptyset)(q_1, \rho(\emptyset, q_1))(q_2, \rho(\rho(\emptyset, q_1), q_2)) \ldots$ die einzige Partie von $\mathcal{G}'$ mit $\pi(p') = p$ die mit $(q_0, \emptyset)$ beginnt, da zu jedem Zustand $(q, S) \in Q'$ und jedem Knoten $q' \in Q$ mit $(q, q') \in K$ genau ein Zustand $(q', S') \in Q'$ mit $((q, S), (q', S')) \in K'$ existiert, nämlich $(q', \rho(S, q))$. Für jede Stellung $s = (q_0, \emptyset)(q_1, S_1) \ldots (q_n, S_n) \in S_{\mathcal{G}'}$ gilt somit $S_n = \mathrm{VS}(q_0 \ldots q_n)$.

Zu 2: Die Aussage ergibt sich aus dem Beweis von Lemma 3.2. Dort haben wir gesehen, wie sich aus einem verallgemeinerten Muller-Automaten, der nur Sprachen aus G_δ erkennt, ein deterministischer Büchi-Automat konstruieren läßt, der zu diesem äquivalent ist. Die Konstruktion von $\mathcal{G}'$ ist nur eine Erweiterung dieser Konstruktion auf Spiele. Sei β eine Beschriftung von $\mathcal{G}$ mit passendem Alphabet Σ, dann gilt $\mathrm{L}(\mathcal{A}_{\mathcal{G},\Sigma,\beta}) \subseteq G_\delta$, da $\mathcal{G}$ ein Spiel aus G_δ ist. Haben wir eine Partie p von $\mathcal{G}$, so ist diese ein akzeptierender Lauf auf $\mathcal{A}_{\mathcal{G},\Sigma,\beta}$. Der deterministische Büchi-Automat $\mathcal{A}'_2$ aus dem Beweis von Lemma 3.2 ist äquivalent zu $\mathcal{A}_{\mathcal{G},\Sigma,\beta}$. Das Spiel $\mathcal{G}'$ enspricht dem Automaten $\mathcal{A}'_2$, und ein akzeptierender Lauf auf $\mathcal{A}'_2$ ist eine Gewinnpartie von $\mathcal{G}'$. Somit ist $p = \pi(p')$ genau dann Gewinnpartie von $\mathcal{G}$, wenn $p' = (q_0, \emptyset)(q_1, S_1) \ldots$ Gewinnpartie von $\mathcal{G}'$ ist.

□

Die Gewinnstrategien für ein Spiel $\mathcal{G}$ aus G_δ läßt sich jetzt folgendermaßen bestimmen: Wir konstruieren ein äquivalentes Büchi-Spiel $\mathcal{G}'$, bestimmen für dieses dessen No-Memory-Gewinnstrategien und übertragen die Strategien wieder zurück auf $\mathcal{G}$, wodurch sich VS-Strategien für $\mathcal{G}$ ergeben.

Satz 4.2 Ist $\mathcal{G} = (Q, Q_1, Q_2, K, W, \Omega)$ ein Spiel aus G_δ oder aus F_σ, so hat $\mathcal{G}$ VS-Gewinnstrategien.

Beweis Ist $\mathcal{G}$ ein Spiel aus F_σ so betrachten wir statt $\mathcal{G}$ das strategieäquivalente Spiel $\overline{\overline{\mathcal{G}}^L}^Q$, das aus G_δ ist. Sei also ohne Beschränkung der Allgemeinheit $\mathcal{G}$ aus G_δ. Wir konstruieren dann nach der obigen Vorschrift das deterministische Büchi-Spiel $\mathcal{G}'$. Nach Satz 5.2 hat $\mathcal{G}'$ No-Memory-Gewinnstrategien τ_1' für Spieler 1 auf $\mathcal{W}_{\mathcal{G}',1}$ und τ_2' für Spieler 2 auf $\mathcal{W}_{\mathcal{G}',2}$. Sei nun $s = q_0 \dots q_n \in S_\mathcal{G}$ eine Stellung von $\mathcal{G}$. Nach Lemma 4.2 existiert genau eine Stellung $s' = (q_0, \emptyset)(q_1, S_1) \dots (q_n, S_n) \in S_{\mathcal{G}'}$ von $\mathcal{G}'$, wobei $S_n = \mathrm{VS}(q_0 \dots q_n)$. Sei $q_n \in Q_i$, dann setze

$$\tau_i(q_0 \dots q_n) = \tau_i'((q_0, \emptyset)(q_1, S_1) \dots (q_n, S_n)).$$

Da τ_i' eine No-Memory-Strategie ist, gibt es eine Funktion f mit $\tau_i'((q_0, \emptyset)(q_1, S_1) \dots (q_n, S_n)) = \mathrm{f}(q_n, \mathrm{VS}(q_0 \dots q_n))$. Somit ist $\tau_i(q_0 \dots q_n) = \mathrm{f}(q_n, \mathrm{VS}(q_0 \dots q_n))$ eine VS-Strategie für Spieler i im Spiel $\mathcal{G}$. Da nach Lemma 4.2 mit jeder Gewinnpartie p' von $\mathcal{G}'$ auch $\pi(p')$ Gewinnpartie von $\mathcal{G}$ ist, folgt, daß τ_i auch Gewinnstrategie für Spieler i in $\mathcal{G}$ ist.

□

4.3 Reguläre Spiele

Allgemein reguläre Spiele haben, wie Robert McNaughton in [McN93] konstruktiv beweist, stets LVR-Gewinnstrategien. Wir möchten an dieser Stelle einen Algorithmus vorstellen, der dem Beweisschema von McNaughton folgt.

4.3.1 Algorithmus

function LVR-Strategie(player, $Q, Q_1, Q_2, K, W, \Omega$)
if $|Q| \leq 2$
  then begin
    if $W \in \Omega$

then begin $U_1 := Q$; $U_2 := \emptyset$; **end**
else begin $U_1 := \emptyset$; $U_2 := Q$; **end**;
return U_{player} ;
end
else begin
for $w \in W$ **do**
begin
$Q_w := \text{force}(1, Q, , Q_1, Q_2, K, W, \{w\}, \text{false}, W)$;
Sei $(Q', Q'_1, Q'_2, K', W', \Omega')$ das Sub-Spiel von $\mathcal{G}$ auf $Q \setminus Q_w$
$U^2_w :=$ LVR-Strategie$(2, Q', Q'_1, Q'_2, K', W', \Omega')$;
$Q_w := \text{force}(2, Q, Q_1, Q_2, K, W, \{w\}, \text{false}, W)$;
Sei $(Q', Q'_1, Q'_2, K', W', \Omega')$ das Sub-Spiel von $\mathcal{G}$ auf $Q \setminus Q_w$
$U^1_w :=$ LVR-Strategie$(2, Q', Q'_1, Q'_2, K', W', \Omega')$;
end;
$N_1 := \text{force}(1, Q, Q_1, Q_2, K, W, \bigcup_{w \in W} U^1_w, \text{player} = 1, W)$;
$N_2 := \text{force}(2, Q, Q_1, Q_2, K, W, \bigcup_{w \in W} U^2_w, \text{player} = 2, W)$;
if $N_1 \cup N_2 = Q$ **then return** N_{player};
elseif $N_1 \cup N_2 \neq \emptyset$
then begin
Sei $(Q', Q'_1, Q'_2, K', W', \Omega')$ das Sub-Spiel auf $Q \setminus (N_1 \cup N_2)$
$U_{\text{player}} :=$LVR-Strategie$(\text{player}, Q', Q'_1, Q'_2, K', W', \Omega')$;
return $U_{\text{player}} \cup N_{\text{player}}$;
end;
elseif $W \in \Omega$ **then** winner:= 1 **else** winner:= 2;
for $w \in W$ **do**
begin
$Q_w :=$force(winner, $Q, K, W, \{w\}$,true, $\{w\}$);
for $q \in Q_{\text{winner}} \setminus Q_w$ **do**
begin
wähle q' mit $(q, q') \in K$;
lösche alle $((q, \text{lvr}), (q'', \text{lvr}')) \in K_S$
mit $q'' \neq q'$ und w erstes Element aus W in lvr;
end;
end;
if winner=player **then return** Q **else return** $\emptyset$;
end;

Die Unterfunktion „force“ bestimmt die Menge von Knoten von denen aus Spieler „player“ das Erreichen der Knotenmenge M erzwingen kann. Die Strategie dazu wird im globalen Strategiegraphen festgelegt, indem die nicht zur Gewinnstrategie gehörenden Kanten aus der Kantenmenge K_S gelöscht werden. Dabei werden nur

die Teile des Strategiegraphen betrachtet, deren auf W reduziertes Last Visitation Record, an erster Stelle ein Element aus V hat.

function force(player, Q, Q_1, Q_2, K, W, M, make-strategie, V)
todo:=true;
while todo **do**
  begin
  todo:=false;
  for $q \in Q$ **do**
    begin
    if $q \in Q_{\text{player}} \wedge \exists q' \in M : (q, q') \in K$
      then begin
        $M := M \cup \{q\}$;
        todo:=true;
        if make-strategie=true **then**
          lösche alle $((q, lvr), (q'', lvr'')) \in K_S$ mit $q'' \neq q'$
            und erstes Element in lvr beschränkt auf W ist aus V;
        end
    elseif $q \notin Q_{\text{player}} \wedge \forall q' \in Q : (q, q') \in K \rightarrow q' \in M$
      then begin
        $M := M \cup \{q\}$;
        todo:=true;
        end;
    end;
  end;
return M;

Satz 4.3 ([McN93]) Der Algorithmus „LVR-Strategie“ ist korrekt und liefert für jedes Spiel $\mathcal{G}$ dessen LVR-Gewinnstrategien.

□

4.4 Synthese

Wir sind jetzt in der Lage einen Synthese-Algorithmus zu definieren, der die drei Gewinnstrategie-Algorithmen für unendliche Spiele zusammenfaßt. Der Vorteil liegt auf der Hand: Nach dem Algorithmus von McNaughton können wir für jedes reguläre Spiel LVR-Gewinnstrategien bestimmen. Wir haben jedoch gesehen, daß

es unter den regulären Spielen „einfachere“ Spiele gibt, nämlich die Spiele aus G und F, bzw. aus G_δ und F_σ, die einfachere Gewinnstrategien, nämlich No-Memory- bzw. VS-Gewinnstrategien erlauben.

Die Vorgehensweise ist einfach: Ist ein Spiel $\mathcal{G}$ gegeben, so wählen wir eine beliebige Beschriftung β über einem passenden Alphabet Σ. Nach Lemma 3.2 sind wir jetzt in der Lage zu entscheiden ob $\mathrm{L}(\mathcal{A}_{\mathcal{G},\Sigma,\beta})$ in G oder F, bzw. in G_δ oder F_σ liegt. Liegt dann das Spiel $\mathcal{G}$ in G oder F, so können wir nach Satz 4.1 No-Memory-Gewinnstrategien konstruieren. Liegt das Spiel $\mathcal{G}$ in G_δ oder F_σ, so bestimmen wir nach Satz 4.2 VS-Gewinnstrategien. Ansonsten wenden wir die Konstruktion der LVR-Gewinnstrategien nach Satz 4.3 an.

Wir erhalten somit folgenden Synthese-Algorithmus:

function synthese($\mathcal{G}$)
begin
    if $\mathcal{G}$ in G oder F
      then Löse $\mathcal{G}$ gemäß Satz 4.1
      else
          if $\mathcal{G}$ in G_δ oder F_σ
            then Löse $\mathcal{G}$ gemäß Satz 4.2
            else Löse $\mathcal{G}$ gemäß Satz 4.3
end

Kapitel 5

Unendliche Spiele auf speziellen Automaten

5.1 Spiele mit 1-Akzeptanz

Als Spiele mit 1-Akzeptanz bezeichnen wir unendliche Spiele, die auf endlichen Graphen gespielt werden, und deren Gewinner durch die von den deterministischen ω-Automaten entliehene Bedingung der sogenannten 1-Akzeptanz bestimmt wird.

Diese Definition unterscheidet sich von Definition 3.1 nur durch das Fehlen von W und darin, daß es nur eine Menge F von Endzuständen gibt.

Definition 5.1 (Spiel mit 1-Akzeptanz) Ein Spiel $\mathcal{G}$ wird repräsentiert durch ein Quintupel (Q, Q_1, Q_2, K, F) mit

$$
\begin{aligned}
&Q_1 \cup Q_2 = Q \neq \emptyset,\\
&Q_1 \cap Q_2 = \emptyset,\\
&\forall k \in K : \exists q_1 \in Q_1, q_2 \in Q_2 : k = (q_1, q_2) \vee k = (q_2, q_1),\\
&\forall q \in Q : \exists q' \in Q : (q, q') \in K,\\
&F \subseteq Q.
\end{aligned}
$$

Spieler 1 gewinnt dieses Spiel genau dann, wenn einer der Zustände aus F im Laufe des Spiels irgendwann einmal besucht wird.

Die Definitionen von *Partie* und *Stellung* sind analog zu den Definitionen 3.2 und 3.3. Lediglich eine *Gewinnpartie* muß neu definiert werden. Zur Erinnerung: Die Funktion Occ liefert die Menge der während der gesamten (unendlichen) Partie besuchten Knoten.

Definition 5.2 (Gewinnpartie eines Spiels mit 1-Akzeptanz) Die *Gewinnpartien* von Spieler 1 liegen in der Menge

$$G_1 := \{p \in P_\mathcal{G} | \mathrm{Occ}(p) \cap F \neq \emptyset\}.$$

Die Gewinnpartien von Spieler 2 liegen dann in der Menge

$$G_2 := P_\mathcal{G} \setminus G_1.$$

Die von determistischen Automaten mit 1-Akzeptanz erkannten Sprachen liegen in der Borel-Klasse G (Vergleiche Abschnitt 2.2). Wir werden zeigen, daß Spiele mit 1-Akzeptanz No-Memory-Gewinnstrategien erlauben.

5.1.1 Gewinnstrategien

Wir beginnen die Bestimmung der Gewinnstrategie für Spieler 1 mit einer Mengenkonstruktion. Sei $\mathcal{G} = (Q, Q_1, Q_2, K, F)$ ein Spiel mit 1-Akzeptanz, dann definieren wir:

$$\begin{aligned}
W_0 &= F \\
W_{i+1} &= W_i \cup z_1(W_i) \\
&\quad \text{mit } z_1 : 2^Q \rightarrow 2^Q \\
&\qquad\qquad W \mapsto \{q \in Q_1 | \exists w \in W : (q,w) \in K\} \\
&\qquad\qquad\quad \cup \{q \in Q_2 | \forall q' \in Q : (q,q') \in K \Rightarrow q' \in W\} \\
N &= \text{kleinster Index mit } W_N = W_{N+1}
\end{aligned}$$

Ausgehend von der Endzustandsmenge F werden sukzessive die Mengen W_i konstruiert, in denen die Zustände enthalten sind von denen aus Spieler 1 die Menge F in höchstens i Zügen erreichen kann.

Die Funktion $z_1(W)$ ermittelt dabei genau die Zustände, von denen entweder Spieler 1 in einem Zug nach W ziehen kann, oder von denen Spieler 2 in jedem Fall nach W ziehen muß.

Bei der Konstruktion der Folge $W_0, W_1, \ldots$ gilt stets $W_i \subset W_{i+1}$ oder $W_i = W_{i+1}$. Im letzteren Fall gilt dann sogar $W_i = W_{i'}$ für alle $i' > i$. Da stets $W_i \subseteq Q$ gibt es einen kleinsten Index $N \leq |Q|$ mit $W_N = W_{N+1}$. W_N ist somit die Menge, aus der Spieler 1 das Erreichen von F in Null oder mehr Zügen erzwingen kann. Dies ist geanu die Menge von Knoten, von denen aus Spieler 1 eine No-Memory-Gewinnstrategie hat.

Lemma 5.1 Sei $\mathcal{G} = (Q, Q_1, Q_2, K, F)$ ein Spiel mit 1-Akzeptanz, dann ist

$$\begin{array}{rcll} \tau_1: & S_{\mathcal{G},1} & \to & Q_2 \\ & sq & \mapsto & \begin{cases} \text{ein } q' \text{ mit } (q,q') \in K \wedge q' \in W_{j_q - 1} & \text{falls } q \in W_N \wedge j_q > 0 \\ \text{ein } q' \text{ mit } (q,q') \in K & \text{sonst} \end{cases} \end{array}$$

wobei j_q den kleinsten Index mit $q \in W_{j_q}$ bezeichnet, eine No-Memory-Gewinnstrategie für Spieler 1 auf W_N.

Beweis Sei $p = p_0 p_1 \ldots \in P_{\mathcal{G}}$ mit $p_0 \in W_N$ und $\tau_1(p_0 \ldots p_n) = p_{n+1}$ für alle $p_n \in Q_1$. Zu zeigen ist jetzt, daß p in G_1 liegt, also $\exists i \in I\!N : p_i \in F$. Hierzu beweisen wir: Ist k kleinster Index mit $p_i \in W_k$, dann existiert ein $j \leq k$ mit $p_{i+j} \in F$. Wir führen dazu eine vollständige Induktion über k.

$k = 0$**:** Es gilt $p_i \in W_0 = F$.

$k > 0$**:** Es gibt zwei Fälle zu unterscheiden:

$p_i \in Q_1$**:** $p_{i+1} = \tau_1(p_0 \ldots p_i) \in W_{k-1}$.

$p_i \in Q_2$**:** $p_{i+1} \in W_{k'}$ mit $k' < k$.

In beiden Fällen ergibt sich, daß $p_{i+1} \in W_{k'}$ mit $k' < k$, somit existiert nach Induktionsvoraussetzung ein $j' \leq k' < k$ mit $p_{i+1+j'} \in F$, also ist $j = 1 + j' \leq k$ mit $p_{i+j} \in F$.

Ist k also kleinster Index mit $p_0 \in W_k$, dann existiert ein $j \leq k$ mit $p_j \in F$ und p ist somit aus G_1.

□

Aus der gleichen Mengenkonstruktion läßt sich auch eine Gewinnstrategie für Spieler 2 konstruieren. Die Idee diese Gewinnstrategie ist es, genau die Gewinnmenge W_N von Spieler 1 zu vermeiden.

Lemma 5.2 Sei $\mathcal{G} = (Q, Q_1, Q_2, K, F)$ ein Spiel mit 1-Akzeptanz, dann ist

$$\begin{array}{rccl} \tau_2: & S_{\mathcal{G},2} & \to & Q_1 \\ & sq & \mapsto & \begin{cases} \text{ein } q' \text{ mit } (q,q') \in K \wedge q' \notin W_N & \text{falls } q \notin W_N \\ \text{ein } q' \text{ mit } (q,q') \in K & \text{falls } q \in W_N \end{cases} \end{array}$$

eine No-Memory-Gewinnstrategie für Spieler 2 auf $Q \setminus W_N$.

Beweis Sei $p = p_0 p_1 \ldots \in P_\mathcal{G}$ mit $p_0 \in Q \setminus W_N$ und gelte $\tau_2(p_0 \ldots p_n) = p_{n+1}$ für alle $p_n \in Q_2$. Zu zeigen ist, daß p in G_2 liegt, also $\forall i \in I\!N : p_i \notin F$. Wir beweisen dazu durch vollständige Induktion über i, $\forall i \in I\!N : p_i \notin W_N$.

$i = 0$**:** Es gilt nach Voraussetzung $p_0 \in Q \setminus W_N$.

$i \to i+1$**:** Gelte also $p_i \notin W_N$. Es gibt zwei Fälle zu unterscheiden:

- $p_i \in Q_1$**:** Es existiert kein $q \in W_N$ mit $(p_i, q) \in K$, da ansonsten Spieler 1 einen Zug nach W_N hätte, und damit $p_i \in W_N$ gälte. Somit muß $p_{i+1} \in Q \setminus W_N$ gelten.
- $p_i \in Q_2$**:** Da Spieler 2 die Strategie τ_2 spielt, gilt $p_{i+1} = \tau_2(p_0 \ldots p_i) \notin W_N$.

Es gilt also $\forall i \in I\!N : p_i \notin W_N$, und da $F \subseteq W_N$ folgt daraus auch $\forall i \in I\!N : p_i \notin F$, und somit ist p eine Gewinnpartie für Spieler 2.

□

Satz 5.1 Ein Spiel mit 1-Akzeptanz ist ein *No-Memory-Spiel*.

Beweis Sei $\mathcal{G}$ ein Spiel mit 1-Akzeptanz, dann vollziehen wir die oben definierte Mengenkonstruktion. Nach Lemma 5.1 hat Spieler 1 eine No-Memory-Gewinnstrategie auf W_N und nach Lemma 5.2 hat Spieler 2 eine No-Memory-Gewinnstrategie auf $Q \setminus W_N$. Somit hat von jedem Knoten von $\mathcal{G}$ aus einer der beiden Spieler eine No-Memory-Gewinnstrategie, und $\mathcal{G}$ ist damit ein No-Memory-Spiel.

□

5.1.2 Algorithmus

Aus den konstruktiven Beweisen der Lemmata 5.1 und 5.2 leiten wir folgendermaßen einen Algortihmus her: Gegeben ist ein Spiel $\mathcal{G} = (Q, Q_1, Q_2, K, F)$ mit 1-Akzeptanz. Simultan zur iterativen Berechnung der Mengen W_i in der Variablen

W wird die Gewinnstrategie von Spieler 1 berechnet, und die nicht zur Gewinnstrategie gehörigen Kanten aus dem Strategiegraphen mit der Kantenmenge K_S entfernt. Abschließend wird die Gewinnstrategie für Spieler 2 auf $Q \setminus W$ bestimmt, indem zu jedem Knoten aus $Q \setminus W$ eine Kante mit Zielknoten in der selben Menge gesucht wird, und alle anderen Kanten, die nicht zur Gewinnstrategie gehören, aus K_S gelöscht werden.

```
solve1(Q, Q_1, Q_2, K, F)
begin
K_S := K;
W := F;
repeat
        done:=true;
        for q ∈ Q \ W do
            begin
            if q ∈ Q_1
              then begin
                    if ∃q' ∈ W mit (q, q') ∈ K
                      then begin
                            W := W ∪ {q};
                            done:=false;
                            lösche alle (q, p) ∈ K_S mit p ≠ q';
                            end
                    end
              else begin
                   if ∀q' ∈ Q : (q, q') ∈ K ⇒ q' ∈ W
                     then begin
                           W := W ∪ {q};
                           done:=false;
                           end
                   end
            end
until done
for q ∈ Q_2 \ W do
    begin
    finde q' ∈ Q \ W mit (q, q') ∈ K;
    lösche alle (q, p) ∈ K_S mit p ≠ q';
    end
end
```

Der Algorithmus nimmt als Eingabe $\mathcal{G} = (Q, Q_1, Q_2, K, F)$ ein Spiel mit 1-Akzeptanz und liefert als Ergebnis W, die Menge der Gewinnknoten von Spieler 1, sowie in K_S die Kanten des *Strategiegraphen* (Q, K_S). Aus dem Strategiegraphen lassen sich wie folgt die Gewinnstrategien ablesen:

$$\begin{aligned} \tau_i : S_{\mathcal{G},i} &\to Q_{3-i} \\ sq &\mapsto \text{ ein } q' \text{ mit } (q, q') \in K_S. \end{aligned}$$

Von den Knoten $Q_1 \cap W$, bzw. $Q_2 \cap (Q \setminus W)$ existiert jeweils nur eine Kante in K_S, die dann die Gewinnstrategie vorgibt.

5.1.3 Komplexität

Aus der Schleifen-Struktur des Algorithmus' erkennt man, daß er eine Zeitkomplexität von $O(n^3)$ hat, wobei $n = |Q|$. Der Platzbedarf ist $O(n)$, da es nur eine konstante Anzahl von Mengenvariablen gibt, deren Mächtigkeit höchstens n ist.

5.2 Deterministische Büchi-Spiele

Als deterministische Büchi-Spiele bezeichnen wir unendliche Spiele, die auf endlichen Graphen gespielt werden, und deren Gewinner durch die von den deterministischen Büchi-Automaten entliehene Bedingung, der sogenannten 2-Akzeptanz, bestimmt wird.

Diese Definition unterscheidet sich von Definition 5.1 nur durch die veränderte Akzeptanzbedingung.

Definition 5.3 (Det. Büchi-Spiel) Ein Spiel $\mathcal{G}$ wird repräsentiert durch ein Quintupel (Q, Q_1, Q_2, K, F) mit

$$\begin{aligned} &Q_1 \cup Q_2 = Q \neq \emptyset, \\ &Q_1 \cap Q_2 = \emptyset, \\ &\forall k \in K : \exists q_1 \in Q_1, q_2 \in Q_2 : k = (q_1, q_2) \vee k = (q_2, q_1), \\ &\forall q \in Q : \exists q' \in Q : (q, q') \in K, \end{aligned}$$

$F \subseteq Q$.

Spieler 1 gewinnt dieses Spiel genau dann, wenn einer der Zustände aus F im Laufe des Spiels unendlich oft besucht wird.

Die Definitionen von *Partie* und *Stellung* sind analog zu den Definitionen 3.2 und 3.3. Lediglich eine *Gewinnpartie* muß neu definiert werden.

Definition 5.4 (Gewinnpartie eines det. Büchi-Spiels) Die *Gewinnpartien* von Spieler 1 liegen in der Menge

$$G_1 := \{p \in P_{\mathcal{G}} | \mathrm{In}(p) \cap F \neq \emptyset\},$$

die von Spieler 2 in der Menge

$$G_2 := P_{\mathcal{G}} \setminus G_1.$$

Die von deterministischen Büchi-Automaten erkannten Sprachen liegen in der Borel-Klasse G_δ (Vergleiche Abschnitt 2.2). Wenn wir, wie in Satz 4.2, ein Spiel auf einem Muller-Automaten, der eine Sprache aus G_δ erkennt, in einen deterministisches Büchi-Spiel überführen, vergrößert sich die Zahl der Knoten um den Faktor $2^{|W|}$, wobei W die Menge der *relevanten* Knoten des Spiels auf dem Muller-Automaten ist. Es ist also nicht sehr verwunderlich, wenn wir jetzt feststellen werden, daß ein deterministisches Büchi-Spiel stets No-Memory-Gewinnstrategien besitzt.

5.2.1 Gewinnstrategien

Die Gewinnstrategien für ein deterministisches Büchi-Spiel wollen wir nach folgender Idee entwickeln: Wir bestimmen iterativ die Mengen F_i, deren Erreichen Spieler 1 $(i+1)$-mal erzwingen kann. Daraus resultiert eine Menge F_M, deren Erreichen Spieler 1 unendlich oft erzwingen kann. Die Menge $\mathcal{W}_{\mathcal{G},1}$, von der aus Spieler 1 das Spiel nach F_M zwingen kann, ist dann genau die Gewinnknotenmenge von Spieler 1. Die Gewinnstrategie von Spieler 2 auf $Q \setminus \mathcal{W}_{\mathcal{G},1}$ wird jetzt so bestimmt, daß nur noch endlich oft ein Knoten aus $F \setminus F_M$ besucht werden kann.

Wir beginnen die Bestimmung der Gewinnstrategie für Spieler 1 mit ein paar Mengenkonstruktionen. Sei $\mathcal{G} = (Q, Q_1, Q_2, K, F)$ ein deterministisches Büchi-Spiel, dann definieren wir:

$$
\begin{aligned}
F_0 &= F \\
W_i^0 &= F_i \\
W_i^{j+1} &= W_i^j \cup z_1(W_i^j) \\
&\quad \text{mit } z_1 : 2^Q \to 2^Q \\
&\qquad W \mapsto \{q \in Q_1 | \exists w \in W : (q,w) \in K\} \\
&\qquad\qquad \cup \{q \in Q_2 | \forall q' \in Q : (q,q') \in K \Rightarrow q' \in W\} \\
N_i &= \text{ kleinster Index mit } W_i^{N_i} = W_i^{N_i+1} \\
L_i &= \{q \in F_i \cap Q_1 | \neg \exists q' \in W_i^{N_i} : (q,q') \in K\} \\
&\quad \cup \{q \in F_i \cap Q_2 | \forall q' \in Q : (q,q') \in K \Rightarrow q' \notin W_i^{N_i}\} \\
F_{i+1} &= F_i \setminus L_i \\
M &= \text{ kleinster Index mit } F_M = F_{M+1}
\end{aligned}
$$

Ausgehend von der gesamten Endzustandsmenge F werden sukzessive verkleinerte Mengen F_i bestimmt, indem Zustände eliminiert werden, von denen aus Spieler 1 ein erneutes Erreichen von F_i nicht erzwingen kann.

Zu diesem Zweck werden sukzessive die Mengen W_i^j konstruiert, in denen die Zustände enthalten sind von denen aus Spieler 1 die Menge F_i in höchstens j Zügen erreichen kann.

Die Funktion $z_1(W)$ ermittelt dabei genau die Zustände, von denen entweder Spieler 1 in einem Zug nach W ziehen kann, oder von denen Spieler 2 in jedem Fall nach W ziehen muß.

Bei der Konstruktion der Folge $W_i^0, W_i^1, \ldots$ gilt stets $W_i^j \subset W_i^{j+1}$ oder $W_i^j = W_i^{j+1}$. Im letzteren Fall gilt dann sogar $W_i^j = W_i^{j'}$ für alle $j' > j$. Da stets $W_i^j \subseteq Q$ gibt es einen kleinsten Index $N_i \leq |Q|$ mit $W_i^{N_i} = W_i^{N_i+1}$. $W_i^{N_i}$ ist somit die Menge, aus der Spieler 1 das Erreichen von F_i in Null oder mehr Zügen erzwingen kann.

F_{i+1} ergibt sich dann aus F_i durch Streichung aller Zustände, von denen aus Spieler 1 nicht in einem Zug in $W_i^{N_i}$ landet, und somit das erneute Erreichen von F_i nicht sichergestellt ist. Auch hier ist klar, daß die Folge der F_i bei einem Index M mit $F_M = F_{M+1}$ abbricht, da stets $F_i \supset F_{i+1}$ oder $F_i = F_{i+1}$, wobei im letzteren Fall wiederum $F_i = F_{i'}$ für alle $i' > i$ gilt.

Die Menge $W_i^{N_i}$ gibt somit die Menge aller Knoten an, von denen aus Spieler 1 $(i+1)$-mal das Erreichen eines Knotens aus F erzwingen kann.

Das Ergebnis dieser Konstruktion ist $\mathcal{W}_{\mathcal{G},1} = W_M^{N_M}$, die Menge der Zustände, von denen aus Spieler 1 eine No-Memory-Gewinnstrategie hat.

Lemma 5.3 Sei $\mathcal{G} = (Q, Q_1, Q_2, K, F)$ ein deterministisches Büchi-Spiel, dann ist

$$\tau_1 : \; S_{\mathcal{G},1} \;\to\; Q_2$$

$$sq \;\mapsto\; \begin{cases} \text{ein } q' \text{ mit } (q,q') \in K \wedge q' \in W_M^{j_q - 1} & \text{falls } q \in \mathcal{W}_{\mathcal{G},1} \wedge j_q > 0 \\ \text{ein } q' \text{ mit } (q,q') \in K \wedge q' \in \mathcal{W}_{\mathcal{G},1} & \text{falls } q \in \mathcal{W}_{\mathcal{G},1} \wedge j_q = 0 \\ \text{ein } q' \text{ mit } (q,q') \in K & \text{falls } q \notin \mathcal{W}_{\mathcal{G},1} \end{cases}$$

wobei j_q den kleinsten Index mit $q \in W_M^{j_q}$ bezeichnet, eine No-Memory-Gewinnstrategie für Spieler 1 auf $\mathcal{W}_{\mathcal{G},1}$.

Beweis Sei $p = p_0 p_1 \ldots \in P_{\mathcal{G}}$ mit $p_0 \in \mathcal{W}_{\mathcal{G},1}$ und $\tau_1(p_0 \ldots p_n) = p_{n+1}$ für alle $p_n \in Q_1$. Zu zeigen ist jetzt, daß p in G_1 liegt, also $\exists^\omega i : p_i \in F$. Wir zeigen zu diesem Zweck, daß zu jedem Zeitpunkt der Partie, in einer endlichen Anzahl von Zügen ein Zustand aus F erreicht wird, also

$$\forall i \in I\!N : \exists 0 \leq j \leq k : p_{i+j} \in F, \text{ wobei } k \text{ kleinster Index mit } p_i \in W_M^k.$$

Wir führen hierzu eine vollständige Induktion über k.

$k = 0$**:** Es gilt $p_i \in W_M^0 = F_M \subseteq F$.

$k > 0$**:** Es gibt zwei Fälle zu unterscheiden:

$p_i \in Q_1$**:** $p_{i+1} = \tau_1(p_0 \ldots p_i) \in W_M^{k-1}$.

$p_i \in Q_2$**:** $p_{i+1} \in W_M^{k'}$ mit $k' < k$.

In beiden Fällen sieht man daß $p_{i+1} \in W_M^{k'}$ mit $k' < k$, somit existiert nach Induktionsvoraussetzung ein $j' \leq k' < k$ mit $p_{i+1+j'} \in F$, also ist $j = 1 + j' \leq k$ mit $p_{i+j} \in F$.

□

Die Bestimmung der No-Memory-Gewinstrategie für Spieler 2 richtet sich nach folgender Überlegung. Spieler 2 gewinnt genau dann, wenn höchstens endlich oft ein Zustand von F erreicht wird. Spieler 1 kann nur auf $\mathcal{W}_{\mathcal{G},1} = W_M^{N_M}$ das unendlich häufige Erreichen eines F-Knotens erzwingen, von den Knoten aus $W_i^{N_i}$ mit $i < M$ gelingt ihm das höchstens $(i+1)$-mal, denn immer wenn ein F-Knoten aus $W_i^{N_i}$ erreicht wird, kann Spieler 2 das Spiel nach $W_{i-1}^{N_{i-1}}$, bzw. nach

$Q \setminus W_0^{N_0}$, falls $i = 0$, zwingen. Ist das Spiel erst einmal nach $Q \setminus W_0^{N_0}$ geraten, so kann Spieler 1 keinen F-Knoten mehr erzwingen, und Spieler 2 gewinnt ganz einfach indem er $W_0^{N_0}$ vermeidet. Spieler 2 kann also auf zwei verschiedene Arten gewinnen:

1. Falls Spieler 1 immer wieder versucht einen F-Knoten zu besuchen, kann Spieler 2 das Spiel schließlich nach $Q \setminus W_0^{N_0}$ zwingen, wo er das Spiel endlos halten kann.
2. Falls Spieler 1 das Erreichen eines F-Knotens vermeidet, ist das Spiel trivialerweise für Spieler 2 gewonnen.

Lemma 5.4 Sei $\mathcal{G} = (Q, Q_1, Q_2, K, F)$ ein deterministisches Büchi-Spiel, dann ist

$$\tau_2 : S_{\mathcal{G},2} \to Q_1$$

$$sq \mapsto q' \text{ mit } \begin{cases} (q,q') \in K \wedge q' \in Q \setminus W_0^{N_0} & \text{falls } q \in Q \setminus W_0^{N_0} \\ (q,q') \in K \wedge q' \in Q \setminus W_{j_q+1}^{N_{j_q+1}} & \text{falls } q \in (W_{j_q}^{N_{j_q}} \setminus F_{j_q}) \setminus W_{j_q+1}^{N_{j_q+1}} \\ (q,q') \in K \wedge q' \in Q \setminus W_{j_q}^{N_{j_q}} & \text{falls } q \in F_{j_q} \setminus W_{j_q+1}^{N_{j_q+1}} \\ (q,q') \in K & \text{falls } q \in \mathcal{W}_{\mathcal{G},1} \end{cases}$$

wobei $j_q < M$ den größten Index mit $q \in W_{j_q}^{N_{j_q}}$ bezeichnet, eine No-Memory-Gewinnstrategie für Spieler 2 auf $\mathcal{W}_{\mathcal{G},2} = Q \setminus W_M^{N_M}$.

Beweis Sei $p = p_0 p_1 \ldots \in P_{\mathcal{G}}$ mit $p_0 \in \mathcal{W}_{\mathcal{G},2}$ und $\tau_2(p_0 \ldots p_n) = p_{n+1}$ für alle $p_n \in Q_2$. Zu zeigen ist jetzt $p \in G_2$, das heißt $\exists n_0 : \forall i > n_0 : p_i \notin F$.

Wir zeigen hierzu: Gilt $p_i \in Q \setminus W_j^{N_j}$ so wird in $p_i p_{i+1} p_{i+2} \ldots$ höchstens j-mal ein F-Knoten besucht.

Wir zeigen dies durch vollständige Induktion über j.

$j = 0$**:** $p_i \in Q \setminus W_0^{N_0}$. Für alle $k > i$ gilt $p_k \in Q \setminus W_0^{N_0}$, denn hätte Spieler 1 einen Zug von $p \in Q \setminus W_0^{N_0}$ nach $W_0^{N_0}$, so gälte bereits $p \in W_0^{N_0}$. Spieler 2 hat auch von jedem Knoten $p \in Q_2 \setminus W_0^{N_0}$ einen Zug nach $p' = \tau_2(sp) \in Q \setminus W_0^{N_0}$, da ansonsten ebenfalls schon $p \in W_0^{N_0}$ gälte. Somit gilt $\forall k > i : p_k \notin W_0^{N_0} \supseteq F$.

$j \to j+1$**:** Sei $p_i \in Q \setminus W_{j+1}^{N_{j+1}}$. Gilt $p_i \notin W_j^{N_j} \setminus W_{j+1}^{N_{j+1}}$ so gilt bereits $p_i \in Q \setminus W_j^{N_j}$ und somit wird nach Induktionsvoraussetzung in $p_i p_{i+1} p_{i+2} \ldots$ höchstens j-mal ein F-Knoten besucht. Sei also $p_i \in W_j^{N_j} \setminus W_{j+1}^{N_{j+1}}$. Es gilt $\forall k > i : p_k \in Q \setminus W_{j+1}^{N_{j+1}}$, da Spieler 1 keinen Zug von $p \in Q \setminus W_{j+1}^{N_{j+1}}$

nach $W_{j+1}^{N_{j+1}}$ hat, denn sonst wäre ja schon $p \in W_{j+1}^{N_{j+1}}$, und Spieler 2 für jeden Knoten $p \in Q_2 \setminus W_{j+1}^{N_{j+1}}$ einen Zug nach $\tau_2(sp) \in Q \setminus W_{j+1}^{N_{j+1}}$ hat.

Wir unterscheiden jetzt drei Fälle:

1. Gilt $p_k \in Q \setminus W_j^{N_j}$ für ein $k \geq i$ und $p_{k'} \in W_j^{N_j} \setminus F_j$ für alle k' mit $i \leq k' < k$, so wird nach Induktionsvoraussetzung in $p_k p_{k+1} \ldots$ höchstens j-mal ein F-Zustand besucht. Da für alle k' mit $i \leq k' < k$ gilt, daß $p_{k'} \notin F_j$, folgt die Behauptung.
2. Gilt $p_k \in F_j$ für ein kleinstes $k \geq i$, dann gilt $p_{k+1} \in Q \setminus W_j^{N_j}$ und nach Induktionsvoraussetzung wird in $p_k p_{k+1} p_{k+2} \ldots$ höchstens j-mal ein F-Knoten besucht. Also wird in $p_i p_{i+1} p_{i+2} \ldots$ höchstens $(j+1)$-mal ein F-Knoten besucht.
3. Gilt $p_k \in W_j^{N_j} \setminus F_j$ für alle $k \geq i$, so wird nie wieder ein F-Knoten besucht, und die Behauptung gilt trivialerweise.

Somit ist $p \in G_2$ und τ_2 eine Gewinnstrategie von Spieler 2.

□

Satz 5.2 Ein deterministisches Büchi-Spiel ist ein *No-Memory-Spiel*.

Beweis Sei $\mathcal{G}$ ein deterministisches Büchi-Spiel, dann vollziehen wir die oben definierte Mengenkonstruktion. Nach Lemma 5.3 hat Spieler 1 eine No-Memory-Gewinnstrategie auf $W_M^{N_M}$ und nach Lemma 5.4 hat Spieler 2 eine No-Memory-Gewinnstrategie auf $Q \setminus W_M^{N_M}$. Somit hat von jedem Knoten von $\mathcal{G}$ aus einer der beiden Spieler eine No-Memory-Gewinnstrategie, und $\mathcal{G}$ ist damit ein No-Memory-Spiel.

□

5.2.2 Algorithmus

Aus den konstruktiven Beweisen der Lemmata 5.3 und 5.4 leiten wir folgendermaßen einen Algorithmus her: Gegeben ist ein deterministisches Büchi-Spiel $\mathcal{G} = (Q, Q_1, Q_2, K, F)$. Simultan zu der iterativen Berechnung der Mengen F_i und $W_i^{N_i}$ in den Variablen F_1 bzw. W wird die Gewinnstrategie von Spieler 2 berechnet und die nicht zur Gewinnstrategie gehörigen Kanten aus dem Strategiegraphen mit der Kantenmenge K_S entfernt. Die Variable W_2 enthält dabei in jeder Iteration die Menge $Q \setminus W_i^{N_i}$. Ist die Berechnung von $W_M^{N_M}$ abgeschlossen, erkennbar daran, daß $L_M = \emptyset$, so wird abschließend die Gewinnstrategie von Spieler 1 auf $W_M^{N_M}$ bestimmt, indem die Berechnung der $W_M^0 \ldots W_M^{N_M}$ noch einmal wiederholt wird, diesmal jedoch mit simultaner Bestimmung der Gewinnzüge von

Spieler 1. Hierzu werden wiederum die nicht zur Gewinnstrategie von Spieler 1 gehörenden Kanten aus K_S gelöscht.

```
solve2(Q, Q_1, Q_2, K, F)
begin
F_1 := F;
W_2 := ∅;
repeat
        W := F_1;
        repeat
                done:=true;
                for q ∈ Q \ W_2 \ W do
                    begin
                    if q ∈ Q_1
                      then begin
                             if ∃q' ∈ W mit (q, q') ∈ K
                               then begin
                                      W := W ∪ {q};
                                      done:=false;
                                      end
                             end
                      else begin
                             if ∀q' ∈ Q : (q, q') ∈ K ⇒ q' ∈ W
                               then begin
                                      W := W ∪ {q};
                                      done:=false;
                                      end;
                             end;
                    end;
        until done;
        for q ∈ Q_2 \ W \ W_2 do
            begin
            finde q' ∈ Q \ W mit (q, q') ∈ K;
            lösche alle (q, p) ∈ K_S mit p ≠ q';
            end;
        L := ∅;
        for q ∈ F_1 do
            begin
            if q ∈ Q_1
              then begin
                     if ∀q' ∈ Q : (q, q') ∈ K ⇒ q' ∈ Q \ W
                       then L := L ∪ {q};
```

```
                end
            else if ∃q' ∈ Q \ W mit (q, q') ∈ K
                then begin
                    L := L ∪ {q};
                    lösche alle (q, p) ∈ K_S mit p ≠ q';
                    end;
        end;
    F_1 := F_1 \ L;
    W_2 := (Q \ W) ∪ L;
until L = ∅;
W := F_1;
repeat
    done:=true;
    for q ∈ Q \ W_2 \ W do
        begin
        if q ∈ Q_1
          then begin
              if ∃q' ∈ W mit (q, q') ∈ K
                then begin
                    W := W ∪ {q};
                    done:=false;
                    lösche alle (q, p) ∈ K_S mit p ≠ q';
                    end;
              end
          else begin
              if ∀q' ∈ Q : (q, q') ∈ K ⇒ q' ∈ W
                then begin
                    W := W ∪ {q};
                    done:=false;
                    end;
              end;
        end;
until done;
for q ∈ F_1 ∩ Q_1 do
    begin
    finde q' ∈ W mit (q, q') ∈ K;
    lösche alle (q, p) ∈ K_S mit p ≠ q';
    end;
end
```

Der Algorithmus nimmt als Eingabe $\mathcal{G} = (Q, Q_1, Q_2, K, F)$ ein deterministisches Büchi-Spiel und liefert als Ergebnis W, die Menge der Gewinnknoten von Spieler 1, sowie in K_S die Kanten des *Strategiegraphen* (Q, K_S). Der Strategiegraph ist wie im Abschnitt 5.1.2 zu interpretieren.

5.2.3 Komplexität

Aus der Schleifen-Struktur des Algorithmus' erkennt man, daß er eine Zeitkomplexität von $O(n^4)$ hat, wobei $n = |Q|$. Der Platzbedarf liegt wie beim Algorithmus „solve1“ bei $O(n)$, da wiederum nur eine konstante Anzahl von Mengenvariablen gebraucht wird, deren Mächtigkeit höchstens n ist.

5.3 Spiele mit Occurrence-Check

Als dritten Spezialfall betrachten wir nun die unendlichen Spiele die auf einem Muller-Automaten mit Occurrence-Check gespielt werden. Diese Spiele unterscheiden sich nur durch eine andere Definition der Gewinnpartie. Zur Erinnerung: Die Funktion Occ liefert die Menge der während der gesamten (unendlichen) Partie besuchten Knoten.

Definition 5.5 (Gewinnpartie) Die Gewinnpartien von Spieler 1 in einem Spiel $\mathcal{G} = (Q, Q_1, Q_2, K, W, \Omega)$ mit Occurrence-Check liegen in der Menge

$$G_1 := \{p \in P_{\mathcal{G}} | \mathrm{Occ}(p) \cap W \in \Omega\}.$$

Die Gewinnpartien von Spieler 2 liegen dann entsprechend in der Menge

$$G_2 := P_{\mathcal{G}} \setminus G_1.$$

5.3.1 Gewinnstrategien

Die von Muller-Automaten mit Occurrence-Check erkannten Sprachen liegen in $F_\sigma \cap G_\delta$, wie Staiger und Wagner in [StaWag74] zeigen. Es liegt daher nahe anzunehmen, daß die Spiele auf diesen Automaten recht einfache Gewinnstrategien

haben. Wie wir leicht zeigen können, reichen No-Memory-Strategien dafür jedoch nicht aus.

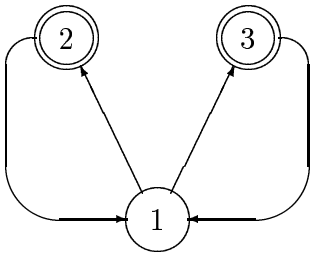


Abbildung 5.1: Spiel mit Occurrence-Check ohne No-Memory-Strategie

Wir betrachten das Spiel $\mathcal{G} = (Q, Q_1, Q_2, K, W, \Omega)$ auf dem Graphen in Abbildung 5.1 mit $Q_1 = \{1\}, Q_2 = \{2, 3\}, W = \{2, 3\}$ und $\Omega = \{\{2, 3\}\}$. Spieler 1 hat in diesem Spiel gewonnen, sobald es ihm gelungen ist sowohl Knoten 2 als auch Knoten 3 zu besuchen. Offensichtlich ist dies aber nicht mit einer No-Memory-Strategie zu bewerkstelligen, da Spieler 1 sich dann für einen der beiden Knoten entscheiden muß, und daher nie beide besuchen kann.

Im vorigen Kapitel haben wir Spiele $\mathcal{G}$ auf Muller-Automaten mit $\mathrm{L}(\mathcal{A}_{\mathcal{G},\Sigma,\beta}) \subseteq G_\delta$ in deterministischen Büchi-Spiele überführt, die dann eine No-Memory-Gewinnstrategie haben. Somit ergab sich eine VS-Strategie für das ursprüngliche Spiel. Die Spiele mit Occurrence-Check lassen sich in ähnlicher Weise behandeln.

Ist $\mathcal{G} = (Q, Q_1, Q_2, K, W, \Omega)$ ein Spiel mit Occurrence-Check, dann konstruieren wir ein deterministisches Büchi-Spiel $\mathcal{G}' = (Q', {Q_1}', {Q_2}', K', F)$, wobei

$$\begin{aligned} Q' &= Q \times 2^W, \\ {Q_1}' &= Q_1 \times 2^W, \\ {Q_2}' &= Q_2 \times 2^W, \\ K' &= \{((p, M), (q, N)) \in Q' \times Q' | (p, q) \in K \wedge N = M \cup (\{q\} \cap W)\} \\ F &= Q \times \Omega \end{aligned}$$

Wie wir wissen hat $\mathcal{G}'$ eine No-Memory-Gewinnstrategie. Diese Strategie läßt sich auf $\mathcal{G}$ übertragen, man erhält dann eine VS-Strategie für das Spiel mit Occurrence-Check.

Es stellt sich nun die Frage, ob es noch einfachere Gewinnstrategien für Spiele mit Occurrence-Check gibt. Denkbar sind Strategien, die einen Zähler

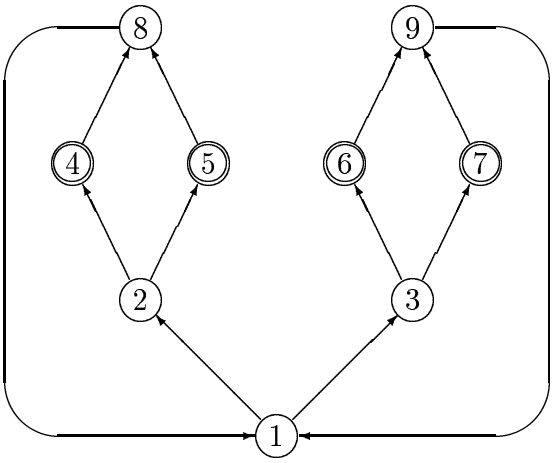


Abbildung 5.2: Spiel mit Occurrence-Check

für jede *unabhängige* Akzeptanzmenge $M \subseteq \Omega$ benutzen, um sich zu merken, wieviele Zustände aus M schon besucht wurden. Betrachtet man zum Beispiel das Spiel $\mathcal{G} = (Q, Q_1, Q_2, K, W, \Omega)$ auf dem Graphen aus Abbildung 5.2 mit $Q_1 = \{2, 3, 8, 9\}$, $Q_2 = \{1, 4, 5, 6, 7\}$, $W = \{4, 5, 6, 7\}$ und $\Omega = \{\{4, 5\}, \{4, 5, 6\}, \{4, 5, 7\}, \{6, 7\}, \{4, 6, 7\}, \{5, 6, 7\}, \{4, 5, 6, 7\}\}$, so sieht man, daß Spieler 1 gewinnt, wenn er entweder die Knoten 4 und 5, oder die Knoten 6 und 7 besucht hat. Spieler 2 kann ihn an diesem Vorhaben nicht hindern, er kann lediglich durch seine Strategie beeinflussen, welche der beiden Mengen $\{4, 5\}$ und $\{6, 7\}$ vollständig besucht wird.

Für die Gewinnstrategie von Spieler 1 reicht es also aus, sich für die beiden Mengen $\{4, 5\}$ und $\{6,7\}$, je einen Zähler zu merken. Nachdem man nun eine Reihenfolge in jeder Menge festgelegt hat, in der man deren Knoten zu besuchen gedenkt, kann man die Gewinnstrategie für dieses Spiel allein über diese Zähler festlegen, so daß zum Beispiel beim ersten Erreichen von Knoten 2 (der zu $\{4, 5\}$ gehörige Zähler steht dann noch auf Null) nach Knoten 4, beim jedem weiteren Erreichen von Knoten 2 (der zu $\{4, 5\}$ gehörige Zähler steht auf Eins) nach Knoten 5 gezogen wird. Entsprechend definiert man die Strategie im Knoten 3.

Kapitel 6

Zusammenfassung

Im Folgenden möchten wir die Ergebnisse dieser Arbeit kurz zusammenfassen. Wir haben uns mit unendlichen Spielen und deren Gewinnstrategien beschäftigt. Bei Robert McNaughton war zu lesen, daß Spiele nach seiner Definition — bei uns beschrieben als „Spiele auf Muller-Automaten“ — LVR-Gewinnstrategien haben, die sich effektiv konstruieren lassen. Diese Gewinnstrategien haben, dargestellt als Strategiegraphen, eine Größenordnung von $O(n!)$. Wir haben jetzt die McNaughton'schen Spiele, mittels einer Beschriftung und einigen Sätzen aus der Theorie der ω-Sprachen, topologisch in die Borel-Hierarchie der ω-Sprachen eingeordnet. Die Zugehörigkeit eines Spiels zu einer Stufe der Borel-Hierarchie unterhalb von $G_{\delta\sigma} \cap F_{\sigma\delta}$ erweist sich als hinreichendes Kriterium dafür, daß weniger komplexe Gewinnstrategien existieren. Liegt ein Spiel in G_δ bzw. F_σ, so haben wir gezeigt, daß dieses Spiel VS-Gewinnstratgien besitzt. Diese Gewinnstrategien haben, dargestellt als Strategiegraphen, eine Größenordnung von $O(2^n)$. Liegt ein Spiel sogar in G oder F, so reichen zum Gewinn des Spiels No-Memory-Strategien, die von der Größenordnung $O(n)$ sind.

Um die Gewinnstrategien für die Spiele der unteren Stufen der Borel-Hierarchie zu berechnen, haben wir Spiele mit vereinfachten Gewinnbedingungen, nämlich Spiele auf deterministischen Büchi-Automaten und Spiele auf deterministischen endlichen Automaten mit 1-Akzeptanz, konstruiert, deren (Gewinn-)Partien sich auf Teilmengen bijektiv abbilden lassen. Auf diese Weise konnten wir die Gewinnstrategien der einfacheren Spiele auf die McNaughton'schen Spiele übertragen.

In der folgenden Tabelle ist der Zusammenhang der topologischen Spielklassen, der zur Definition verwendeten Automatenmodelle und der daraus resultierenden Strategieklassen dargestellt.

	Automatenmodell			
top. Klasse	1-Akz.	2-Akz.	Occ.-Check	Muller
G, F	No-Memory			No-Memory
G_δ, F_σ	—	No-Memory		VS
$G_\delta \cap F_\sigma$	—	No-Memory	VS	VS
$G_{\delta\sigma} \cap F_{\sigma\delta}$	—	—	—	LVR

Hier sieht man deutlich den Zusammenhang zwischen der Mächtigkeit des der Spieldefinition zugrundeliegenden Automatenmodells und der Komplexität der Gewinnstrategien. No-Memory-Gewinnstrategien von Spielen aus G bzw. F lassen sich eins zu eins auf Spiele mit 1-Akzeptanz übertragen. Bei Spielen aus G_δ bzw. F_σ ergibt sich ein „Blow up“ von $O(2^n)$. Die allgemein regulären Spiele lassen sich zwar auf kein einfacheres herkömmliches Automatenmodell reduzieren, aber die LVR-Gewinnstrategien, die sie besitzen, kann man als No-Memory-Strategien auf einem Spielgraphen, der um alle möglichen LVR erweitert ist, betrachten. Dies ergibt dann einen „Blow up“ von $O(n!)$.

Angsichts der Tatsache, daß die allgemeinen Muller-Automaten sehr viel mächtiger als die Muller-Automaten mit Occurrence-Check sind, ist zu vermuten, daß es noch eine Strategieklasse zwischen VS und No-Memory gibt, die zum Gewinnen von Spielen mit Occurrence-Check geeignet sind.

Anhang A

Implementation

Die Implementation einiger in dieser Diplomarbeit vorgestellten Algorithmen fand auf einem i486-PC mit Unix-Betriebssystem (Linux) unter Verwendung eines Standard-C-Compilers (GNU-C) statt.

Im Folgenden ein Source-Code-Listing der wichtigsten Programmteile:

```
/*******************/
/*  Hauptprogramm  */
/*******************/

#include <stdio.h>
#include <strings.h>
#include "mengen.h"    /* einfache Mengenoperationen */
#include "games.h"     /* Datenstrukturen fuer Spiele */

extern void solve1();  /* Loesung von Spielen mit 1-Akzeptanz */
extern void solve2();  /* Loesung von det. Buechi-Spielen */
extern void solve_oc();/* Loesung von Spielen mit Occurrence-Check */
extern void solve3();  /* Loesung von regulaeren Spielen */

void solve_error(text)
char *text;
{
   printf("Error in solve: %s\n",text);
   exit(-1);
}
```

```
/*************************************************************/
/* Das Hauptprogramm liest einen Spielspezifikations-Datenfile */
/* und verzweigt je nach Spiel-Typ in die entsprechenden       */
/* Loesungsfunktionen.                                         */
/*************************************************************/

main(argc,argv)
int argc;
char *argv[];
{
   game *g;

   if(argc<2) solve_error("No input file");
   g=game_read(argv[1]);
   switch(g->type)
   {
      case TYPE_DEA:
        solve1(g);
        break;
      case TYPE_BUECHI:
        solve2(g);
        break;
      case TYPE_OCCURRENCE:
        solve_oc(g);
        break;
      case TYPE_MULLER:
        solve3(g);
        break;
   }
   game_free(g);
}
```

A.1 Spiele mit 1-Akzeptanz

```
/*********************************************/
/* Die Funktion solve1 ist eine Implementation */
/* von Algorithmus "solve1" aus Abschnitt 5.1  */
/*                                             */
/* Eingabeparameter: g (Spiel mit 1-Akzeptanz) */
```

```
/* Ausgabe: Protokoll der Mengenkonstruktion   */
/*          und Strategiegraph als Adjazenz-   */
/*          Matrix.                            */
/*********************************************/

#include <stdio.h>
#include <strings.h>
#include "mengen.h"
#include "games.h"

void solve1(g)
game *g;
{
   game *k_s;       /* Strategiegraph */
   simple_set *w;   /* Mengenvariable fuer die W_i */
   int done;
   int q,q1,p;
   int won;

   game_print(g);
   k_s=game_copy(g);    /* Strategiegraph mit Spielgraph
                           initialisieren  */
   w=set_create(g->n);

   /*************************************/
   /* Gewinnmenge W und Gewinnstrategie  */
   /* fuer Spieler 1 auf W ermitteln     */
   /*************************************/

   set_copy(g->final,w);
   do
   {
      done=1;
      for(q=0;q<g->n;q++)
      {
         if(!set_element(q,w))
         {
            if(set_element(q,g->player1))
            {
               for(q1=0;q1<g->n;q1++)
               {
                  if(*(g->matrix+q*g->n+q1)==1 &&
                     set_element(q1,w))
```

```
                {
                   for(p=0;p<g->n;p++)
                   {
                      if(p!=q1 && *(g->matrix+q*g->n+p)==1)
                      {
                         *(k_s->matrix+q*k_s->n+p)=0;
                      }
                   }
                   set_add_element(q,w);
                   done=0;
                   break;
                }
             }
          }
          else
          {
             won=1;
             for(q1=0;q1<g->n;q1++)
             {
                if(*(g->matrix+q*g->n+q1)==1 &&
                   !set_element(q1,w))
                {
                   won=0;
                   break;
                }
             }
             if(won)
             {
                set_add_element(q,w);
                done=0;
             }
          }
       }
    }
 }while(!done);

 /************************************/
 /* Strategie fuer Spieler 2 ermitteln */
 /************************************/

 for(q=0;q<g->n;q++)
 {
    if(!set_element(q,w) &&
```

```
         !set_element(q,g->player1))
      {
         for(q1=0;q1<g->n;q1++)
         {
            if(*(g->matrix+q*g->n+q1)==1 &&
               !set_element(q1,w))
            {
               for(p=0;p<g->n;p++)
               {
                  if(p!=q1 && *(g->matrix+q*g->n+p)==1)
                  {
                     *(k_s->matrix+q*g->n+p)=0;
                  }
               }
               break;
            }
         }
      }
   }
   printf("Strategiegraph:\n"); /* Ergebnis: Strategiegraph */
   game_print(k_s);
   game_free(k_s);
   set_free(w);
}
```

A.2 Deterministische Büchi-Spiele

```
/*********************************************/
/* Die Funktion solve2 ist eine Implementation */
/* von Algorithmus "solve2" aus Abschnitt 5.2  */
/*                                             */
/* Eingabeparameter: g (det. Buechi-Spiel)     */
/* Ausgabe: Protokoll der Mengenkonstruktion   */
/*          und Strategiegraph als Adjazenz-   */
/*          Matrix.                            */
/*********************************************/

#include <stdio.h>
#include <strings.h>
#include "mengen.h"
```

```
#include "games.h"

void solve2(g)
game *g;
{
   simple_set *w;    /* Mengenvariable fuer die W_i^j */
   simple_set *f1;   /* Mengenvariable fuer die F_i */
   simple_set *w2;   /* Mengenvariable fuer Q\W_i^N */
   simple_set *l;    /* Mengenvariable fuer L_i */
   simple_set *w_neu,*w_compl,*f_neu; /* Hilfsvariablen */
   int q,q1,p;    /* Schleifenvariablen */
   int done,won;
   game *k_s;     /* Strategiegraph */

   game_print(g);
   k_s=game_copy(g);   /* Strategiegraph mit Spielgraph
                          initialisieren */
   w=set_create(g->n);
   f1=set_create(g->n);
   set_copy(g->final,f1);
   w2=set_create(g->n);
   l=set_create(g->n);

   /*****************************************************/
   /* Konstruktion der Menge W_i^N mit gleichzeitiger  */
   /* Bestimmung der Gewinnstrategie von Spieler 2 auf */
   /* Q\W_i^N                                          */
   /*****************************************************/

   do
   {
      set_copy(f1,w);
      do
      {
         done=1;
         for(q=0;q<g->n;q++)
         {
            if(!set_element(q,w2) && !set_element(q,w))
            {
               if(set_element(q,g->player1))
               {
                  for(q1=0;q1<g->n;q1++)
                  {
```

```
                if(set_element(q1,w) &&
                   *(g->matrix+q*g->n+q1)==1)
                {
                   set_add_element(q,w);
                   done=0;
                   break;
                }
             }
          }
          else
          {
             won=1;
             for(q1=0;q1<g->n;q1++)
             {
                if(*(g->matrix+q*g->n+q1)==1 &&
                   !set_element(q1,w))
                {
                   won=0;
                   break;
                }
             }
             if(won)
             {
                set_add_element(q,w);
                done=0;
             }
          }
       }
    }
 }while(!done);

 /*********************************/
 /* Bestimmung der Gewinnstrategie */
 /* von Spieler 2 auf Q\W\W2       */
 /*********************************/

 for(q=0;q<g->n;q++)
 {
    if(!set_element(q,g->player1) &&
       !set_element(q,w) &&
       !set_element(q,w2))
    {
       for(q1=0;q1<g->n;q1++)
```

```
      {
         if(!set_element(q1,w) &&
            *(g->matrix+q*g->n+q1)==1)
         {
            for(p=0;p<g->n;p++)
            {
               if(*(k_s->matrix+q*g->n+p)==1 && p!=q1)
               {
                  *(k_s->matrix+q*g->n+p)=0;
               }
            }
            break;
         }
      }
   }
}

printf("F_i: ");     /* Zwischenergebnis F_i */
set_print(f1);
printf("W_i: ");     /* Zwischenergebnis W_i^N */
set_print(w);

/**********************/
/* Bestimmung von L_i */
/**********************/

set_empty(l);
for(q=0;q<g->n;q++)
{
   if(set_element(q,f1))
   {
      if(set_element(q,g->player1))
      {
         won=1;
         for(q1=0;q1<g->n;q1++)
         {
            if(*(g->matrix+q*g->n+q1)==1 &&
               set_element(q1,w))
            {
               won=0;
               break;
            }
         }
```

```
            if(won)
            {
               set_add_element(q,l);
            }
         }
         else
         {
            for(q1=0;q1<g->n;q1++)
            {
               if(*(g->matrix+q*g->n+q1)==1 &&
                     !set_element(q1,w))
               {
                  set_add_element(q,l);
                  for(p=0;p<g->n;p++)
                  {
                     if(*(k_s->matrix+q*g->n+p)==1 && p!=q1)
                     {
                        *(k_s->matrix+q*g->n+p)=0;
                     }
                  }
                  break;
               }
            }
         }
      }
   }
   printf("L_i: ");    /* Zwischenergebnis L_i */
   set_print(l);
   f_neu=set_difference(f1,l);
   set_copy(f_neu,f1);
   set_free(f_neu);
   w_compl=set_complement(w);
   w_neu=set_union(w_compl,l);
   set_copy(w_neu,w2);
   printf("W2: ");
   set_print(w2);
   set_free(w_neu);
   set_free(w_compl);
}while(set_size(l)!=0);

printf("Strategie fuer Spieler 2:\n");
game_print(k_s);  /* Zwischenergebnis Strategiegraph mit
                     Gewinnstrategie fuer Spieler 2  */
```

```
/************************************/
/* Bestimmung der Gewinnstrategie von */
/* Spieler 1 auf Q/W2                */
/************************************/

set_copy(f1,w);
do
{
   done=1;
   for(q=0;q<g->n;q++)
   {
      if(!set_element(q,w2) && !set_element(q,w))
      {
         if(set_element(q,g->player1))
         {
            for(q1=0;q1<g->n;q1++)
            {
               if(*(g->matrix+q*g->n+q1)==1 &&
                  set_element(q1,w))
               {
                  set_add_element(q,w);
                  done=0;
                  for(p=0;p<g->n;p++)
                  {
                     if(*(k_s->matrix+q*g->n+p)==1 && p!=q1)
                     {
                        *(k_s->matrix+q*g->n+p)=0;
                     }
                  }
                  break;
               }
            }
         }
         else
         {
            won=1;
            for(q1=0;q1<g->n;q1++)
            {
               if(*(g->matrix+q*g->n+q1)==1 &&
                  !set_element(q1,w))
               {
                  won=0;
```

```
                    break;
                }
            }
            if(won)
            {
                set_add_element(q,w);
                done=0;
            }
        }
    }
}while(!done);

/*************************************/
/* Bestimmung der Gewinnstrategie von */
/* Spieler 1 auf F_M                  */
/*************************************/

for(q=0;q<g->n;q++)
{
   if(set_element(q,g->player1) && set_element(q,f1))
   {
      for(q1=0;q1<g->n;q1++)
      {
         if(*(g->matrix+q*g->n+q1)==1 &&
            set_element(q1,w))
         {
            for(p=0;p<g->n;p++)
            {
               if(*(k_s->matrix+q*g->n+p)==1 && p!=q1)
               {
                  *(k_s->matrix+q*g->n+p)=0;
               }
            }
            break;
         }
      }
   }
}
printf("Strategiegraph:\n");  /* Ergebnis: Strategiegraph */
game_print(k_s);
game_free(k_s);
set_free(w);
```

```
   set_free(w2);
   set_free(l);
   set_free(f1);
}
```

A.3 Spiele mit Occurrence-Check

```
/************************************************/
/* Die Funktion solve_oc ist eine Implementation */
/* der Konstruktion aus Abschnitt 5.3           */
/*                                              */
/* Eingabeparameter: g (Spiel mit Occurrence-   */
/*                   Check)                     */
/* Ausgabe: Protokoll der Mengenkonstruktion    */
/*          und Strategiegraph als Adjazenz-    */
/*          Matrix.                             */
/************************************************/


#include <stdio.h>
#include <strings.h>
#include "mengen.h"
#include "games.h"

extern void solve2(); /* Es wird auf die Loesung von
                         det. Buechi-Spielen zurueckgegriffen */


/****************************************************/
/* Die Funktion OCM_to_DB konstruiert aus einem     */
/* Spiel mit Occurrence-Check ein det. Buechi-Spiel */
/****************************************************/

game *OCM_to_DB(g)
game *g;
{
   game *g2;
   int i,j,k,l,m,n_k;

   g2=(game*)malloc(sizeof(game));
```

```
g2->type=TYPE_BUECHI;
m=set_size(g->final);
g2->n=g->n*(1<<m);
g2->player1=set_create(g2->n);
for(i=0;i<g->n;i++)
{
   if(set_element(i,g->player1))
   {
      for(j=0;j<(1<<m);j++)
      {
         set_add_element(i*(1<<m)+j,g2->player1);
      }
   }
}
g2->matrix=(char*)malloc(g2->n*g2->n);
for(i=0;i<g->n;i++)
{
   for(j=0;j<(1<<m);j++)
   {
      n_k=0;
      for(k=0;k<g->n;k++)
      {
         for(l=0;l<(1<<m);l++)
         {
            if(*(g->matrix+i*g->n+k)==1 &&
               ((!set_element(k,g->final) && j==l) ||
                (set_element(k,g->final) && ((l&(1<<n_k))!=0) &&
                  (l&(~(1<<n_k)))==(j&(~(1<<n_k)))
                     )))
            {
               *(g2->matrix+((i*(1<<m)+j)*g2->n)+(k*(1<<m)+l))=1;
            }
            else
            {
               *(g2->matrix+((i*(1<<m)+j)*g2->n)+(k*(1<<m)+l))=0;
            }
         }
         if(set_element(k,g->final))
         {
            n_k++;
         }
      }
   }
```

```
   }
   g2->final=set_create(g2->n);
   for(i=0;i<g->n_omega;i++)
   {
      k=0;
      for(j=0;j<m;j++)
      {
         k<<=1;
         if(set_element(set_nth_element(j,g->final),g->omega[i]))
         {
            k|=1;
         }
      }
      for(l=0;l<g->n;l++)
      {
         set_add_element(l*(1<<m)+k,g2->final);
      }
   }
   return(g2);
}

/*************************************************/
/* Die Funktion solve_oc wandelt ein Spiel       */
/* mit Occurrence-Check in ein det. Buechi-Spiel  */
/* um, und loest dieses mit der Funktion "solve2" */
/*************************************************/

void solve_oc(g)
game *g;
{
   game *g2;

   game_print(g);
   g2=OCM_to_DB(g);
   solve2(g2);
   game_free(g2);
}
```

Literaturverzeichnis

[BücLan69] J. R. Büchi und L. H. Landweber, „Solving sequential conditions by finite-state strategies“ *Trans. Am. Math. Soc.* **138**, 295–311, 1969.

[Dav64] Morton Davis, „Infinite Games of perfect information“ *Ann. Math. Studies* **52** *(Advances in game theory)*, 85–101, Princeton Univ. Press, 1964.

[GalSte53] D. Gale und F. M. Stewart, „Infinite games with perfect information“ *Ann. Math. Studies* **28** *(Contribution to the theory of games)*, 245–266, Princeton Univ. Press, 1953.

[Lan69] L. H. Landweber, „Decision Problems for ω-automata“, *Math. Systems Theory* **3** (1969), 376–384.

[McN93] Robert McNaughton, „Infinite games played on finite graphs“, *Annals of Pure and Applied Logic* **65**, 149–184, Elsevier Science Publishers B. V, 1993.

[Mos80] Y. N. Moschovakis, *Descriptive Set Theory*, North-Holland, Amsterdam 1980.

[NeuMor44] John von Neumann und Oskar Morgenstern, „Theory of Games and Economic Behaviour“, Princeton University Press, Princeton 1953.

[StaWag74] L. Staiger und K. Wagner, „Automatentheoretische und automatenfreie Charakterisierungen topologischer Klassen regulärer Folgenmengen“, *Elektron. Informationsverarb. Kybernet. EIK* **10** (1974), 379–329.

[Tho90] W. Thomas, „Automata on infinite objects“ *Handbook of theoretical computer science, Vol. B*, Elsevier Science Publishers B. V., und MIT Press, 1990.

Hiermit erkläre ich, daß ich diese Diplomarbeit selbständig verfaßt und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet habe.

Kiel, 4. Mai 1994