Technische Universität Braunschweig

Bachelorarbeit

Good-for-Games Automaten: Ein Überblick

Kai Harz

14. Oktober 2019

Institute of Theoretical Computer Science
Prof. Dr. Roland Meyer

Betreuer:
Sebastian Muskalla, M. Sc.

Eidesstattliche Erklärung

Ich erkläre hiermit an Eides statt, dass ich die vorliegende Bachelorarbeit selbstständig verfasst sowie alle benutzten Quellen und Hilfsmittel vollständig angegeben habe und dass die Arbeit nicht bereits als Prüfungsarbeit vorgelegen hat.

Braunschweig, 14. Oktober 2019

_____

Abstract

In der folgenden Arbeit werden sich Paritätsspiele angeschaut. Infolge dessen wird das 2018 von Lehtinen vorgestellte Verfahren betrachtet, das es ermöglicht, Paritätsspiele in quasipolynomieller Zeit zu lösen. Bei diesem Verfahren handelt es sich um die Registerspiele. Im Laufe der Arbeit wird ihre Funktionsweise erläutert und gezeigt, wie sie sich auf das Konzept der good-for-Games Automaten übertragen lassen. Außerdem wird über die Struktur der universellen Bäume gezeigt, dass für auf diesen Automaten basierenden Algorithmen eine quasipolynomielle untere Schranke existiert.

Aufgabenstellung der Bachelorarbeit

Paritätsspiele sind Zwei-Personen-Spiele mit perfekter Information, bei denen der Gewinner einer unendlich langen Partie nicht bereits an einem endlichen Präfix erkennbar ist. Sie eignen sich zum Modellieren von sogenannten Liveness-Verifikationsproblemen. Insbesondere finden sie Verwendung in Verfahren für Entscheidungsprobleme für die Spezifikationslogiken S2S (monadische Prädikatenlogik zweiter Stufe über unendlichen Binärbäumen) und dem modalen μ-Kalkül, einer Fixpunktlogik. Beispielsweise verwendet der moderne Beweis von Rabins Baum-Theorem, welches zeigt, dass die Erfüllbarkeit der Logik S2S entscheidbar ist, Paritätsspiele und deren Eigenschaften.

Die Komplexität von Paritätsspielen, also dem Problem, zu entscheiden, welcher Spieler den Sieg erzwingen kann, ist seit langem ein offenes Problem. Zielonkas Algorithmus wird in der Praxis häufig verwendet, sein Zeitverbrauch ist allerdings exponentiell sowohl in der Knotenanzahl als auch in der höchsten verwendeten Priorität. Neuere Verfahren liefern subexponentielle, aber immer noch superpolynomielle Laufzeiten. Im Jahr 2017 wurde mit dem Algorithmus von Calude et al. ein Durchbruch erzielt; Dieser löst Paritätsspiele in quasipolynomieller Zeit, wobei nur die Prioritäten exponentiell eingehen. In den folgenden Jahren wurden andere Algorithmen mit ähnlichen Zeitschranken vorgestellt, unter anderem von Jurdzinski und Lazic sowie von Lehtinen. Später wurde von Czerwinski et al. gezeigt, dass diesen drei Algorithmen das selbe Prinzip zugrunde liegt. Man kann sie so auffassen, dass sie einen sogenannten good-for-small-Games Automaten konstruieren. Das Produkt dieses Automaten mit dem gegebenen Paritätsspiel ist ein größeres Spiel, jedoch mit einfacherer Gewinnbedingung (bei Calude et al. sowie bei Jurdzinski und Lazic: Safety- statt Paritätsbedingung; bei Lehtinen: Paritätsspiel mit nur noch logarithmisch vielen Prioritäten). Die Konstruktion ist insbesondere im Fall von Lehtinens Verfahren interessant, da der konstruierte Automat hier nicht-deterministisch ist. Das Resultat von Czerwinski et al. beweist auch, dass die spezielle Art von good-for-small-Games Automaten, die aus den oben genannten Konstruktionen resultiert, in ihrer Größe durch eine quasipolynomielle Zustandsanzahl nach unten beschränkt ist.

Ziel der Arbeit ist es, den Begriff der good-for-small-Games Automaten zu erläutern und darzulegen, wie diese zum Lösen von Paritätsspielen verwendet werden können. Insbesondere sollen das Verfahren von Lehtinen, der Beweis seiner Korrektheit und die aus dem Verfahren resultierende Automatenkonstruktion dargestellt werden.
Lehtinen hat zwischenzeitlich eine vereinfachte Form ihres Verfahrens veröffentlicht, die Automatenkonstruktion von Czerwinski et al. bezieht sich jedoch noch auf die ursprüngliche Veröffentlichung. Als weiteres Ziel der Arbeit sollen die Konstruktionen aneinander angepasst werden, so dass der Zusammenhang zwischen beiden besser erkennbar wird.

Inhaltsverzeichnis

1 Einleitung

Ein Konzept aus der Theoretischen Informatik, das genutzt wird um Problemstellungen abzubilden, sind die sogenannten Spiele. In diesen nehmen mindestens zwei verschiedene Entitäten, Spieler genant, Einfluss auf den Systemablauf. Dabei kann die Art, wie die einzelnen Spieler Einfluss nehmen, unterschiedlich sein. Damit können zum Beispiel Situationen modelliert werden, in denen eine Entität nicht die Kontrolle über alle Parameter hat. Einer der Spieler kann dabei ein Programm darstellen, das eine bestimmt Aufgabe besitzt, und der andere Spieler stellt Einflüsse von außerhalb dar, auf die das Programm reagieren muss. Ein andere Bereich in dem Spiele Anwendung finden, ist die Verifikation. Auf den ersten Blick ist dabei nicht ersichtlich, an welcher Stelle die verschiedenen Spieler auftauchen. Wenn man sich aber zum Beispiel ein System anschaut, das über eine bestimmte Sorte Nichtdeterminismus verfügt und das Model der Eigenschaft, die zu verifizieren ist, über eine andere Art Nichtdeterminismus verfügt, dann kann der Nichtdeterminismus über die Spieler dargestellt werden. Ein Beispiel dafür wären nichtdeterministische Automaten auf Bäumen, wobei die Verzweigungsstruktur der Bäume als eine Art des Nichtdeterminismus aufgefasst wird.

Nun besitzen Spiele verschiedene Gewinnbedingungen, um verschiedene Sachverhalte darzustellen. Die simpelste Bedingung ist Safety. Dabei gibt es eine Menge an ungewollten Zuständen und einer der Spieler versucht diese Zustände zu erreichen und der andere unendlich lange diese zu vermeiden. Das Besondere dabei ist, dass, wenn der eine Spieler diese Zustände erreichen kann, dies immer in endlicher Zeit möglich ist. Dadurch ist diese Bedingung für bestimmte Modellierungen nicht ausreichend. Eines davon ist Liveness, zum Beispiel wenn ein Programm eine unendliche Anzahl an Anfragen erhält, soll es auch eine unendliche Anzahl an Antworten abschicken. Hierfür können die Büchi und CoBüchi Bedingungen genutzt werden. Auch hier existiert eine Menge an schlechten Zuständen, aber diesmal ist es akzeptabel, wenn diese erreicht werden, solange die guten Zustände nur unendlich oft erreicht werden. Eine Erweiterung davon ist Parität. Dort gibt es eine Hierarchie der guten und schlechten Zustände ausgedrückt durch Prioritäten. Dabei erhalten die schlechten Zustände ungerade Prioritäten und die guten Zustände gerade Prioritäten. Man kann sich das so vorstellen, dass eine ungerade Priorität für einen Fehler steht, der von den kleineren Prioritäten nicht abgefangen werden kann. Eine gerade Priorität hingegen steht dafür, dass alle Fehler, die in kleineren Prioritäten auftreten behoben werden. Die Akzeptanzbedingung für Parität lautet nun, dass die höchste Priorität, die unendlich oft auftaucht gerade ist. Das heißt, dass jedes Mal, wenn ein Fehler auftritt, dieser durch einen Zustand mit höherer Priorität behoben wird.

Eine der am häufigsten gestellten Fragen ist dabei, ob einer der Spieler sicherstellen

kann, dass er das Spiel gewinnt, unabhängig vom Verhalten des anderen Spielers. Eine daraus folgende Frage ist, wie man einen solchen Spieler bestimmen kann und wie lange ein Algorithmus dauert, der diesen bestimmt. Man kann nämlich Paritätsspiele für S2S-Logik (monadische Prädikatenlogik zweiter Stufe über unendlichen Binärbäumen) und dem modalen μ-Kalkül nutzen. Beispielsweise verwendet der moderne Beweis für Rabins Baum-Theorem, welches die Erfüllbarkeit der S2S-Erfüllbarkeit zeigt, Paritätsspiele und deren positionelle Determiniertheit.

Bisher wurde zum Lösen von Paritätsspielen häufig Zielonkas Algorithmus [1] verwendet, dieser hat aber superpolynomiellen Zeitbedarf. Im Kontrast dazu stehen Safety und Büchi-Spiele, für die es polynomielle Algorithmen gibt. Gleichzeitig ist bereits bekannt, dass das Ermitteln des Gewinners eines Paritätsspiels sowohl in NP als auch in co-NP liegt. Entsprechend besteht die Hoffnung, dass es einen Algorithmus mit polynomiellen Zeitbedarf gibt, der das Problem lösen kann.

Seit 2017 wurden nun 3 Verfahren von Calude et al. [2], Jurdzinski und Lazik [3] sowie Lehtinen [4] gefunden, die zumindest einen quasipolynomiellen Algorithmus erzeugen. Aus diesem ergeben sich zwei Ansätze, um einen polynomiellen Algorithmus zu erhalten. Zum einen sind die Algorithmen *fixed-parameter tractable*, das heißt die Größe des Graphen geht nur polynomiell in die Laufzeit ein und die größte Priorität ist der exponentielle Faktor. Außerdem hat die quasipolynomiell Laufzeit hier die Form $2^{(\lg n)^k}$, wobei n die Größe des Spiele und k eine Konstante ist. Die Hoffnung ist nun, dass diese Konstante auf 1 reduziert werden kann oder die Priorität um eine passenden Faktor verkleinert werden kann. Der andere Ansatz besteht im schon oft erfolgreich genutzten Prinzip, ein Problem auf ein einfacher zu lösendes Problem zu reduzieren. In diesem Fall wird versucht Paritätsspiele auf Safety-Spiele zu reduzieren, da diese nur polynomiellen Zeitbedarf haben. Dabei ist wichtig, das die Reduktion selber auch in polynomieller Zeit durchführbar ist. Mit den drei neuen Verfahren ist zumindest eine quasipolynomielle Reduktion möglich. Genauer liefern die Verfahren einen *good-for-small-Games* Automaten.

Good-for-Games Automaten haben die besondere Eigenschaft, dass sie mit einem Spiel kombiniert werden können, um ein neues Spiel zu erzeugen, das die Gewinnbedingung des Automaten besitzt, gleichzeitig aber den gleichen Gewinner wie das alte Spiel aufweist. *Good-for-small-Games* Automaten funktionieren genauso, nur kann hier die Übereinstimmung der Gewinner nur für Spiele bis zu einer bestimmten Größe garantiert werden. Die Hoffnung war, das mit gewissen Anpassungen der Verfahren eine polynomielle Reduktion möglich ist. Leider haben Czerwinski et al. [5] gezeigt, das es eine quasipolynomielle untere Schranke für dies Art von Automaten gibt und entsprechend damit keine solche Reduktion durchführbar ist. Das schließt eine Reduktion über andere Verfahren aber nicht aus, entsprechend bleibt das Bestimmen der Komplexität von Paritätsspielen ein offenes Problem.

Struktur Ziel dieser Arbeit ist es nun, einen Überblick über diese verschiedenen Konzepte zu geben und sie in Verbindung zueinander zu setzten. Dazu werden zuerst in Kap.2 die Grundlagen erklärt. Insbesondere ist dabei wichtig, dass eine einheitliche Schreibweise

genutzt wird, da sich diese in den verschiedenen Arbeiten zu den Themen unterscheiden. Dabei wird auf Automaten, Spiele und universelle Graphen und Bäume eingegangen. In Kap.3 wird dann erklärt, wie *good-for-Games* und *good-for-small-Games* Automaten definiert sind und auch die ersten Ergebnisse dazu vorgestellt. In Kap. 4.1 werden zuerst die Registerspiele vorgestellt. An dieser Stelle ist zu beachten, dass Lehtinen die Konstruktion von Registerspielen, mit der sie den quasipolynomiellen Algorithmus für Paritätsspiele gezeigt hatte, [4] in einem späteren Paper [6] nochmal überarbeitet hatte, um Beweise einfacher und nachvollziehbarer zu gestalten. Hier wird die zweite Variante vorgestellt, sowie einige der daraus folgenden Beweise insbesondere natürlich für den Zeitbedarf gezeigt. In Kap. 4.2 wird dann der von Czerwinski et al. entwickelte Separatoransatz vorgestellt, mit welchem die drei Verfahren für Paritätsspiele dargestellt werden können. Außerdem wird der Bewies erläutert, warum dafür eine untere Schranke existiert. Allerdings bezieht sich Czerwinski noch auf die erste Version der Registerspiele, weshalb dies nicht mehr mit der neuen Version übereinstimmt. Deshalb werden in Kap. 4.3 Registerspiele und Separatoren aneinander angepasst, um sie besser miteinander in Bezug zu setzten. In Kap. 5 werden dann verschiedene Ergebnisse von Colcombet und Fijalkow [7] [8] genutzt, um zu zeigen, dass Registerspiele eine Form der *good-for-small-Games* Automaten sind, sowie dass die von Czerwinski gezeigten Laufzeitschranken auf diese anwendbar sind.

2 Grundlagen

Zunächst einmal werden die Grundlagen, die hinter dieser Arbeit stehen, besprochen, dabei wird auch die Notation der verschiedenen benutzten Konzepte erläutert, da es in der Literatur durchaus Abweichungen gibt. Im Folgenden werden Automaten, Spiele und universelle Bäume und Graphen besprochen.

Automaten: Als Erstes wären da Automaten. Ein Automat $A = (\Sigma, Q, s, \delta, \alpha)$ besteht aus einem endlichen Alphabet Σ, einer endlichen Zustandsmenge Q, einem Startzustand s, einer Übergangsfunktion δ und einer Akzeptanzbedingung α. Der Automat liest dabei Worte $w \in \Sigma^\omega$, dass heißt, das der Automat vom Startzustand aus buchstabenweise die Zustände so wechselt, wie es in der Übergangsfunktion definiert wird. Man spricht auch vom Lauf eines Automaten über ein Wort w. Die Akzeptanzbedingung bestimmt dabei, ob ein Lauf vom Automaten akzeptiert wird. Die Menge aller Wort, deren Lauf akzeptiert wird bildet dabei die Sprache $L(A)$ des Automaten. Dabei stellt die Übergangsfunktion den Unterschied zwischen verschiedenen Arten von Automaten da, bei deterministische Automaten gilt $\delta : Q \times \Sigma \to Q$, während bei nichtdeterministischen Automaten $\delta : Q \times \Sigma \to P(Q)$ gilt. Dabei ist $P(Q)$ die Potenzmenge über Q, also die Menge aller möglichen Mengen der Zustände. Wenn dabei an einem Zustand mehrere Übergänge möglich sind, wird einer davon zufällig gewählt. Bei Nichtdeterminismus muss nur ein Lauf von w über die möglichen Übergänge zur Akzeptanzbedingung führen. Des Weiteren gibt es noch alternierende Automaten, diese sind eine Unterform der nichtdeterministischen Automaten. Dort werden die Zustände in zwei disjunkte Teilmengen unterteilt $Q = Q_\forall \cup Q_\exists$, auch universell und existenziell genannt. $Q_\exists$ verhält sich genau so, wie bei nichtdeterministischen Automaten, währen bei $Q_\forall$ gilt, dass alle möglichen Übergänge zu einer Akzeptanzbedingung führen müssen, damit die Eingabe akzeptiert wird. Dabei ist zu beachten, dass diese Automaten gleich mächtig sind, also die gleiche Art von Sprachen erkennen, der Unterschied besteht in der Anzahl der benötigten Zustände und das man bei Beweisen bestimmte Eigenschaften von nichtdeterministischen beziehungsweise alternierenden Automaten ausnutzen kann.

Im Rahmen dieser Arbeit werden Automaten betrachtet, die über unendlichen Worten $w = w_0 w_1 \ldots \in \Sigma^\omega$ operieren. Dies beeinflusst die Art der Akzeptanzbedingungen: während bei endlichen Worten die Bedingung häufig mit dem Zustand zusammenhängt,der nach der letzten Eingabe erreicht wird, ist dies hier nicht möglich. Daher ist die Bedingung hier häufig, dass eine bestimmte Eigenschaft unendlich oft beziehungsweise nur endlich oft auftritt. Eine der einfachsten ist *Safety*, diese enthält eine Menge ablehnender Zustände und ein Wort wird akzeptiert, wenn diese Zustände nie erreicht werden. Eine andere, die im Verlauf der Arbeit häufiger genutzt wird, ist *Parität*. Dabei wird jedem Zustand ein

Wert, genannt Priorität, aus dem endlichen Intervall $[i,d]$ zugewiesen und ein Durchlauf wird genau dann akzeptiert, wenn die höchste Priorität, die unendlich oft auftritt, gerade ist.

Zu Spielen: Als nächstes betrachten wir die Grundlagen von Spielen. Ein Spiel ist ein gerichteter Graph $G = (V, R, L)$ mit Knoten V, Kanten R und einer Labelfunktion L. Dabei wird die Knotenmenge in zwei disjunkte Teilmengen unterteilt, die den beiden Spielern zugewiesen werden, entsprechend gilt $V = V_A \cup V_E$. Außerdem verfügt er über eine Menge von Buchstaben Σ, diese wird mit der Labelfunktion L den Kanten zugewiesen; also $L : R \to \Sigma$. In solchen Spielen gibt es zwei Spieler, im folgenden Adam und Eva genannt, häufig A bzw. E abgekürzt, wobei V_A zu Adam und V_E zu Eva gehört. Ein Spielablauf ist eine Abfolge von Knoten, im Folgenden Positionen genant, $p = p_0 p_1 p_2 ...$ wobei die einzelnen Spielzüge $(p_i, p_i + 1) \in R$ jeweils von dem Spieler gewählt werden, dem die aktuelle Position gehört. Wenn es von einer Position aus keinen Zug gibt, wird diese als deadlocked bezeichnet. Ein Spiel das keine solchen Positionen besitzt ist deadlockfrei. Im Rahmen dieser Arbeit werden nur deadlockfreie Spiele betrachtet. Ein Spiel G' wird als Unterspiel zu einem Spiel G bezeichnet, wenn alle Positionen und Züge, die in G' vorkommen, auch in G vorkommen. G kann aber Positionen und Züge besitzen, die nicht in G' auftauchen.

An dieser Stelle sei noch angemerkt, dass man die Labels auch den Knoten zuweisen kann, die Funktion also folgendermaßen definiert wird: $L : V \to \Sigma$. Beide Varianten werden in der Literatur benutzt und tauchen auch in den Arbeiten auf, auf die sich später bezogen wird. Dies ist aber nicht problematisch, da sie gleichwertig sind. Man kann ein Spiel immer von der einen Form in die andere Form überführen. Wenn die Prioritäten an den Knoten stehen, muss man sie für die Umformung nur von einem Knoten an alle aus diesem Knoten ausgehenden Kanten setzen. Die Umformung in die andere Richtung ist etwas aufwendiger. Für jede Kante wird ein neuer Knoten eingefügt, der zwischen den beiden Knoten liegt, die von der Kante verbunden wurden. Der neue Knoten erhält die Priorität, die vorher die Kante besaß, die anderen Knoten erhalten eine Priorität, die das Spiel nicht beeinflusst.

Des Weiteren besitzt ein Spiel eine Gewinnbedingung. Eine mögliche Bedingung ist *Safety*, hierbei wird eine Menge an Positionen ausgewählt und Eva gewinnt, wenn diese Positionen in einem unendlichen Spiel nie erreicht werden, ansonsten gewinnt Adam. Eine andere Bedingung ist Parität, dabei bestehen die Labels Σ aus den Werten des Intervalls $[i, d]$ mit i und d aus den natürlichen Zahlen, die Labels werden im Folgenden als Priorität bezeichnet, und Eva gewinnt, wenn die höchste unendlich oft auftauchende Priorität gerade ist.

Die Beispiel 2.1a und 2.1b zeigen zwei Paritätsspiele. Dabei sind die roten Quadrate Knoten aus V_E und die grünen Kreise Knoten aus V_A und v_0 ist der Startknoten des Spiels. Dabei ist leicht zu sehen, dass das linke Spiel von Eva gewonnen wird, da die Kante mit Priorität 2 immer genommen werden muss. Beim rechten Spiel hingegen hat Adam eine Gewinnstrategie, nämlich in v_1 immer den Loop zu wählen und damit Priorität 1 unendlich oft auftauchen zu lassen.

(a) Spiel mit Eva als Gewinner (b) Spiel mit Adam als Gewinner (c) Automat zu Spiel b

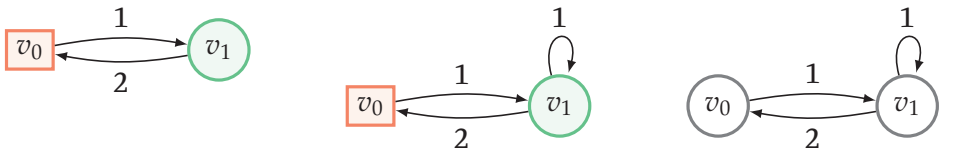


Abbildung 2.1: Beispiele für Spiele und Automat

Hier sieht man, dass es zwischen Automaten und Spielen Überschneidungen gibt. Es kann nämlich für ein Spiel ein Automat konstruiert werden, der entscheidet, ob eine gegebenes Wort eine korrekte Zugfolge ist, die für einem der Spieler zum Sieg führt. In 2.1c ist ein Parity-Automat zu sehen, der entscheidet ob ein Wort in der Sprache des Spiels aus 2.1b ist.

Definition 1. *Eine Strategie für einen Spieler ist eine Funktion $\pi : V^* \to V$, die einer endlichen Abfolge von Knoten $v_0, ..., v_i$ einen Nachfolgeknoten zuweist. Dabei gehört v_i zu dem entsprechenden Spieler.*

Die Strategie gibt also an, wie sich der Spieler in einer bestimmten Spielsituation verhalten soll. Man spricht davon, dass einer der Spieler eine Gewinnstrategie für eine Position x besitzt, wenn er eine Strategie besitzt, die es ihm bei jeden möglichen Zug des Gegners ermöglicht, die Gewinnbedingung zu erreichen. Die Definition lässt sehr leicht erkennen, dass nicht beide Spieler gleichzeitig eine Gewinnstrategie für eine Position besitzen können, allerdings gibt es auch Positionen in denen keiner der Spieler eine Gewinnstrategie besitzt. Die Menge an Positionen für die Adam bzw. Eva eine Gewinnstrategie besitzt wird Gewinnregion W_A bzw. W_E genant. Wenn dabei die Startposition in der Gewinnregion eines Spielers liegt, spricht man davon, dass der Spieler das Spiel gewinnt. Ein Spiel, in dem es für jede Position einen Gewinner gibt, wird determiniert genannt. Eine Strategie wird positional genannt, wenn nur die aktuelle Position nicht aber die vorherigen Züge für die Auswahl des aktuellen Zuges entscheidend sind.

Eine der wichtigen Aussagen über Paritätsspiele ist, das sie positional determiniert sind. Dies wurde unter anderem von Emerson and Jutla [9] bewiesen.

Zu (n,d)-Bäumen und (n,d)-Graphen: Eine weitere Struktur die auf Graphen beruht, sind Bäume. Graphtheoretisch sind Bäume zusammenhängende Graphen ohne Kreise. Dabei betrachten wir gewurzelte Bäume mit geordneten Kindern. Dabei besitzt jeder Knoten ein Label, mit dem er identifiziert wird. Von diesem Knoten gehen wiederum Kinder ab, wobei die Kinder eines Knotens total geordnet sind. Man kann einen beliebigen Knoten also an der Abfolge der Labels seiner Vorgänger identifizieren. Die Höhe eines Knotens beschreibt, wie viele Knoten er von der Wurzel entfernt liegt. Eine mögliche Ordnung ist die lexikografische, dabei sind die Kinder wie im Alphabet in einer festen Reihenfolge. Wenn man nun zwei beliebige Knoten vergleichen will, schaut man sich das Wort an, dass von der Wurzel ausgehend bis zum Knoten gebildet wird und der erste Buchstabe, der sich

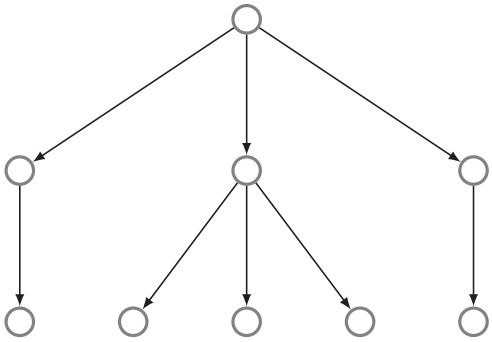


Abbildung 2.2: ein (3,4)-universeller Baum

unterscheidet, bestimmt die Ordnung. Zum Beispiel gilt $\langle a,c,e\rangle < \langle a,d,b\rangle$. Knoten die keine Kinder haben werden Blätter genannt. Ein Baum S ist in einem Baum T enthalten, wenn man durch Entfernen von Knoten in T den Baum S erzeugen kann. Man spricht von einem (n,d)-Baum, wenn der Baum maximal n Blätter hat und maximale Höhe $d/2$ besitzt. Der Einfachheit halber wird im Folgenden davon ausgegangen, dass alle Blätter eines Baumes die gleiche Höhe haben.

Definition 2. *Ein Baum wird als (n,d)-universell bezeichnet, wenn er alle (n,d)-Bäume enthält.*

Abbildung 2.2 zeigt dabei den kleinsten (3,4)-universellen Baum. Für (3,4)-Bäume gibt es verschiedene Möglichkeiten, wie sie aufgebaut sind. Entweder haben alle Blätter verschiedene Vorgänger, alle haben den gleichen Vorgänger oder zwei haben den gleichen und das dritte hat einen anderen Vorgänger. Alle Möglichkeiten können bei beliebiger Ordnung der Blätter in den gezeigten Baum enthalten sein.

Ebenfalls gibt es (n,d)-Graphen, dies ist ein gerichteter Graph mit n Knoten und Prioritäten bis d. Dabei wird jeder Kante eine Priorität zugewiesen also $(v,i,v') \in R$. Ein (n,d)-Graph bildet also die grundlegende Struktur für ein Spiel mit n Positionen und Prioritäten bis d, es fehlt nur noch die Unterteilung der Knoten in die von Adam und Eva. Ein Pfad in einem Graphen ist eine Sequenz von aufeinanderfolgenden Kanten. Man spricht davon, dass ein Graph Parität erfüllt, wenn alle Pfade Parität erfüllen. Diese Aussage ist gleichwertig dazu, dass die größte Priorität in alle Zyklen gerade ist. Wenn ein Pfad nämlich keinen Zyklus aufweist, kann auch keine Priorität unendlich oft auftauchen. Ein Homomorphismus ϕ zwischen zwei Graphen G und G' ist eine Abbildung die allen Kanten aus G Kanten aus G' zuweist, unter Beibehaltung der Prioritäten: $(v,i,v') \in R \rightarrow (\phi(v),i,\phi(v') \in R'$

Definition 3. *Ein Graph ist (n,d)-universell, wenn er die Parity-Bedingung erfüllt und es für jeden (n,d)-Graphen, der ebenfalls Paritäterfüllt, einen Homomorphismus in diesen Graphen gibt.*

3 good-for-small-Games

Produkt aus Spiel und Automat Good-for-small-Games Automaten sind eine neue Idee von Fijalkow und Colcombet [8], sie sind eine Erweiterung der bereits bekannten good-for-Games Automaten. Um zu klären, was ein good-for-Games Automat ist, muss zuvor das Produkt eines Automaten A und eines Spiels G definiert werden. Die Idee dabei ist, dass abwechselnd Züge im Spiel und im Automaten stattfinden, währenddessen wird der Zustand im jeweils Anderen beibehalten. Dabei liest der Automat die Priorität die zuvor im Spiel auftaucht als seine aktuelle Eingabe. Dies setzt voraus, dass das Alphabet Σ des Automaten mit den Labels des Spiels übereinstimmt. Gleichzeitig wird bei nichtdeterministischen Automaten, die Entscheidung, welcher der zulässigen Übergänge genutzt wird, an Eva übergeben. Das Produkt $G \times A$ wird formal folgendermaßen gebildet:

Definition 4. *Für Zustände Q aus A und Positionen V sowie Zügen R aus G bildet $Q \times (V \uplus R)$ die neue Menge an Positionen. Die neue Startposition ist das Produkt des Startzustands Q_i und der Startposition V_i. Die Positionen können dabei in zwei Arten unterteilt werden. Einmal die Spielpositionen $(q, v) \in Q \times V$ und zum Anderen die Automatenpositionen $(q, (v, p, w)) \in Q \times R$. Spielpositionen sind dem Spieler zugeordnet, dem auch die Position v im Ursprungsspiel gehörte, Automatenpositionen hingegen sind immer Eva zugeordnet.*

Die möglichen Züge sind wie folgt definiert, für alle Züge $(v, p, w) \in R$ des Ursprungsspiels existiert ein Zug $((q, v), \epsilon, (q, (v, p, w)))$ und für alle Übergänge $(q, (p, d), r)$ existiert ein Zug $(q, (v, p, w), d, (r, w))$.

Das Produkt zu bilden, hat zwei Gründe. Zum Einen kann dadurch ein Spiel in eine andere Gewinnbedingung überführt werden. Zum Beispiel ist das Produkt einen Paritätsspiels und eine Safety-Automaten ebenfalls ein Safety-Spiel. Zum Anderen kann damit ein eventuell auftretender Nichtdeterminismus des Automaten aufgelöst werden, da Eva jetzt die Kontrolle über die entsprechenden Positionen erhält.

In Abbildung 3.1 ist nun ein Beispiel für ein solches Produkt zu sehen. Abbildung 3.1a zeigt dabei das ParitätssSpiel G. In diesem hat Eva eine Gewinnstrategie und zwar muss sie immer von v_1 aus den Zug nach v_0 wählen, da sonst Adam in v_2 unendlich oft die 1 erzeugen kann. In Abbildung 3.1b wiederum ist der Safety-Automat A zu sehen. In diesem bildet q_2 den ablehnenden Zustand. Da A ein Safety-Automat ist, besitzt er keine Prioritäten, die an den Kanten stehenden Zahlen stellen die am entsprechenden Übergang gelesene Eingabe dar. Die von A akzeptierte Sprache besteht aus der unendlichen Abfolge der Sequenz (2,1) und erfüllt damit auch die Paritätsbedingung. In Abbildung 3.1c ist nun das Produkt $G \times A$ zu sehen. Dies ist nun ein Safety-Spiel mit der Gewinnbedingung aus A, entsprechend sind die Zustände (q_2, v_2) und $(q_2, (v_2, 1, v_2))$ ablehnend. Da A als

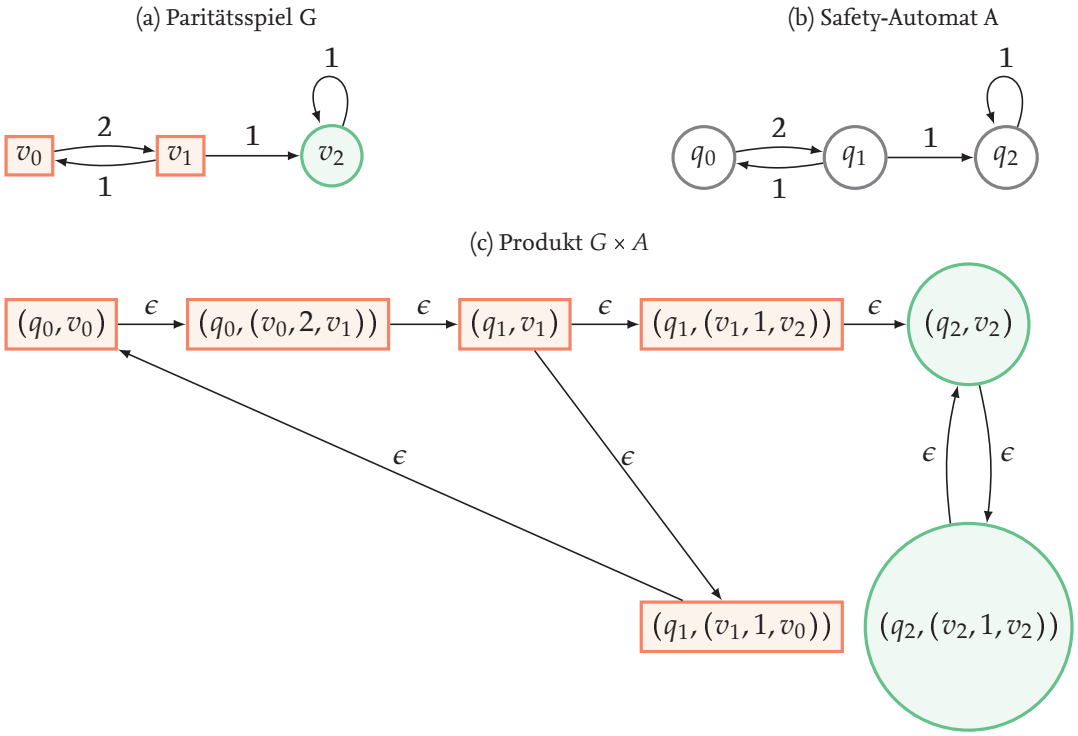


Abbildung 3.1: Beispiel für ein Produkt

Safety-Automat keine Prioritäten besitzt, sind alle Prioritäten im Produkt ϵ, die von A abgeleitet werden, auch ϵ, was dazu passt, dass das Produkt ein Safety-Spiel ist. Die Position (q_1, v_1) ist dabei die Stelle, an der Eva den Nichtdeterminismus von Zustand q_1 aus A auflöst. Nach der obigen Definition existieren noch mehr Positionen wie (q_0, v_2), da diese aber von der Startposition aus nicht erreichbar sind, müsse diese nicht beachtet werden.

Ein schon seit längerer Zeit bekanntes Ergebnis hierzu lautet, dass für ein Spiel G mit einer beliebigen Gewinnbedingung V und einem Automaten A, dessen akzeptierte Sprache $L \subseteq V$ ist, gilt, dass, wenn Eva $G \times A$ gewinnt, sie auch G gewinnt. Hierfür wird die Strategie S_E, die Eva für $G \times A$ besitzt, zu einer Strategie für G umgeformt. Und zwar wird jeder Zug $((q, v), \epsilon, (q, (v, p, w)))$ aus $G \times A$ zu einem Zug (v, p, w) in G.

good-for-Games Nach dem nun das Produkt von einem Spiel und einem Automaten definiert wurde, kann nun definiert werden, wann ein Automat *good-for-Games* ist.

Definition 5. *Ein Automat A wird als good-for-Games bezeichnet, wenn für eine beliebige Gewinnbedingung V gilt, dass für alle Spiele G mit der Gewinnbedingung V gilt, dass G und $G \times A$ den gleichen Gewinner haben.*

Bei deterministischen Automaten gilt diese Beziehung immer, denn da der Automat für jede Eingabe nur einen Übergang besitzt, existiert in den Automatenpositionen immer nur ein möglicher Zug für Eva. Sie kann also das Spiel dort gar nicht beeinflussen, wodurch G und $G \times A$ den selben Gewinner haben. Es sind also die nichtdeterministischen

Fälle, die interessant sind. Good-for-small-Games Automaten sind ähnlich definiert, nur das die Bedingung hier nur für Spiele G mit maximal n Positionen gelten muss. Für Spiele mit mehr als n Positionen können sich die Gewinner von G und $G \times A$ unterscheiden. Dabei kann n beliebig aus den positiven ganzen Zahlen gewählt werden.

Definition 6. *Ein Automat A wird als good-for-small-Games bezeichnet, wenn für eine beliebige Gewinnbedingung V gilt, dass für alle Spiele G mit der Gewinnbedingung V und mit maximal n Positionen gilt, dass G und $G \times A$ den gleichen Gewinner haben.*

Die Idee dahinter ist, dass, statt einen Automaten zu finden, der für alle Spiele gilt, man nur einen suchen muss, der bis zur Größe der zu betrachtenden Spiele gilt. Sie sind also eine Näherungslösung zu den good-for-Games Automaten und je größer man das n wählt, desto genauer wird die Näherung. Daher wird auch die alternative Bezeichnung *good-for-n-Games* genutzt. Ein Automat ist *strongly good-for-n-Games*, wenn zusätzlich die vom Automaten A akzeptierte Sprache $L(A)$ in V enthalten ist.

Eine wichtige Erkenntnis an dieser Stelle betrifft das Produkt zweier Automaten. Sei A ein good-for-Games Automat, der die Sprache V akzeptiert, und sei B ein good-for-Games Automat, der die Sprache W akzeptiert, dann ist $A \times B$ ebenfalls ein good-for-Games Automat, der die Sprache W akzeptiert.

Obige Definition ist nur eine Möglichkeit, *good-for-Games* Automaten zu definieren, in der Literatur sind 5 verschiedenen Möglichkeiten bekannt. Boker und Lehtinen[10] haben sich in einer ihrer Arbeiten die 5 Definitionen angeschaut, sie teilweise für alternierende Automaten verallgemeinert und nachgewiesen, dass sie tatsächlich das gleiche beschreiben. Die 5 Definitionen sind *good-for-Games composition*, *good-for-Automata composition*, *compliant with the letter games*, *history determinism* und *good-for-Trees*. *good-for-Games composition* ist dabei die Definition, die bisher genutzt wurde und auch die, die im Rest der Arbeit genutzt wird, wobei in der allgemeinen Definition auch unendlich große Spiele zulässig sind. *good-for-Automata composition* ist ganz ähnlich definiert, nur das hier nicht für Spiele sondern für Automaten B gelten soll, dass B und $B \times A$ äquivalent sein sollen. *Compliant with the letter games* ist folgendermaßen definiert. Dies ist ein Spiel über nichtdeterministische Automaten, in dem Adam ein Wort buchstabenweise erzeugt und Eva gleichzeitig den auftretenden Nichtdeterminismus auflöst, sie also nicht wissen kann, welche Eingaben später noch auftreten werden. Wenn Eva das Spiel gewinnt ist die Definition zutreffend. *History determinism* bedeutet, dass es eine Möglichkeit gibt, den Nichtdeterminismus aufzulösen, bei der nur die bisher erfolgte Eingabe zu betrachten ist. Man sieht hier einfach die Verknüpfung zur vorherigen Definition. Und schließlich bedeutet *good-for-Trees*, dass ein Automat erweitert werden kann, um auch über Bäume eingesetzt zu werden.

Die Beweise für das Übereinstimmen der Definitionen stammen teilweise bereits aus anderen Arbeiten und sind zu umfangreich, um sie hier darzustellen. Allerdings ist ein zu beachtender Punkt, dass teilweise von unendlich großen Spielen ausgegangen wird, entsprechend gelten die Gleichheiten nicht alle uneingeschränkt für *good-for-small-Games* Automaten, da diese ja nur eine beschränkte Größe besitzen.

Wenn ein Paritätsspiel G über n Zustände und d Prioritäten verfügt, kann ein strongly good-for-n-games Automat A konstruiert werden, sodass $G \times A$ über $m = n^{\log d + O(1)}$ Zustände und $d' = 1 + \lfloor \log n \rfloor$ Prioritäten verfügt. Ein solches Spiel zu lösen, gelingt in quasipolynomieller Zeit. Dies wird in einem späteren Kapitel gezeigt.

Verbindung zu anderen Strukturen Später wurde von Colcombet und Fijalkow [8] gezeigt, dass für eine Gewinnbedingung, für die Eva eine positionale Gewinnstrategie besitzt, wie zum Beispiel Parität, folgende Werte übereinstimmen:

1. die Anzahl an Zuständen eines deterministischen strongly (n,d)-separierenden Automaten
2. die Anzahl an Zuständen eines strongly good-for-n-Games Automaten
3. die Anzahl an Zuständen eines strongly (n,d)-separierenden Automaten
4. die Anzahl der Knoten eines (n,d)-universal Graphen.

Da es dabei nur um die Anzahl der Knoten und Zustände geht, hat die Größe der Prioritäten d keinen Einfluss auf die Gleichheit.

Ein separierender Automat ist wie folgt definiert. Für zwei disjunkte Sprachen J und K wird eine Sprache S als Separator bezeichnet, wenn $J \subseteq S$ und $S \cap K = \varnothing$. Ein Automat A separiert zwei Sprachen, wenn seine Sprache $L(A)$ die Sprachen separiert.

Die Idee von Czerwinski et al. [5] ist Paritätsautomaten auf Safety separierende Automaten zu reduzieren. Hierzu wird ein Separator für $EvenCycles_{n,d}$ und $OddCycles_{n,d}$ konstruiert. Hierzu zunächst ein paar Begriffserklärungen. Man sagt ein Zyklus in einem Graphen ist gerade, wenn die höchste darin auftauchende Priorität gerade ist, und ein Graph ist gerade, wenn alle Zyklen gerade sind, analog für ungerade. Die Sprache eines Graphen ist die Menge aller Wörter, die über die Prioritäten an den Pfaden im Graphen entstehen. $EvenCycles_{n,d}$ bzw. $OddCycles_{n,d}$ ist die Sprache bestehend aus unendlichen Wörtern in geraden bzw. ungeraden Graphen mit maximal n Knoten und Prioritäten bis maximal d. (Nach Czerwinski et al.) $LimsupEven_d$ bzw. $LimsupOdd_d$ sind die Sprachen über unendliche Wörter, in denen die höchste unendlich oft auftretende Zahl gerade bzw. ungerade ist. Dabei besteht das Alphabet der Sprachen aus den ganzen Zahlen des Intervalls [1,...,d]. Diese Definition ist erstmal von Graphen getrennt. Da die Graphen bei $EvenCycles_{n,d}$ bzw. $OddCycles_{n,d}$ maximal n Positionen besitzen, folgt daraus, dass zwischen zwei Auftreten der höchsten Priorität maximal $n-1$ andere Prioritäten liegen können. Bei $LimsupEven_d$ bzw. $LimsupOdd_d$ hingegen können beliebig viele Prioritäten dazwischen liegen. Entsprechend gilt für alle natürlichen n das $EvenCycles_{n,d} \subset LimsupEven_d$ und $OddCycles_{n,d} \subset LimsupOdd_d$. Ein Automat A wird als *(n,d)-Separator* bezeichnet, wenn er $EvenCycles_{n,d}$ und $OddCycles_{n,d}$ separiert, und als *strong (n,d)-Separator*, wenn er $EvenCycles_{n,d}$ und $LimsupOdd_d$ separiert.

An dieser Stelle kann man wieder die Verbindung zu Spielen aufbauen. Denn wenn Eva ein Paritätsspiel gewinnt, ist die höchste unendlich oft auftauchende Priorität gerade, die entsprechende Sprache muss also in $LimsupEven_d$ liegen und darf nicht in $LimsupOdd_d$ liegen. Dabei ist d die höchste Priorität, die im Spiel auftritt. Gleichzeitig sind Paritätsspiele positional, also hat Eva in ihrer Strategie für jeden Knoten genau eine Kante, die

benutzt wird. Wenn nun ein neuer Graph G' erzeugt wird, indem alle nicht von Eva genutzten Kanten gelöscht werden, sowie alle dadurch nicht mehr erreichbaren Knoten, dann ist G' gerade. Wenn er nicht gerade wäre, gäbe es mindestens einen ungeraden Zyklus, diesen könnte Adam dann unendlich oft durchlaufen und würde somit gewinnen, was ein Widerspruch dazu wäre, das Eva das Spiel gewinnt. Da G' endlich ist, liegt die erzeugte Sprache in $EvenCycles_{n,d}$, wobei n die Anzahl an Knoten in G' ist. Entsprechend lässt sich ein Parity-Spiel über einen *strong (n,d)-Separator* beschreiben.

Lemma 1. *Nun gilt, dass für ein Parity-Spiel G mit n Knoten und Prioritäten bis d und einen deterministischen safety (n,d)-separierenden Automaten A, dass Eva genau dann eine Gewinnstrategie in G hat, wenn sie auch eine im safety Spiel $G \times A$ hat.*

Beweis. Wie bereits erklärt, sind Gewinnstrategien für Parity-Spiel positional. Entsprechend gilt mit dem gleiche Argument wie oben, dass, wenn Eva das Spiel gewinnt, ein gerader Graph mit maximal n Knoten erzeugt werden kann. Somit sind alle Wort die im Spiel erzeugt werden in $EvenCycles_{n,d}$ und werden von A akzeptiert. Wenn Eva nun ihre Strategie für G in den Spielpositionen von $G \times A$ nutzt, gewinnt sie das Spiel. Denn da A deterministisch ist, hat sie in den Automatenpositionen immer nur einen möglichen Zug, wodurch die Strategie nicht beeinflusst wird. Selbiges gilt, wenn Adam eine Gewinnstrategie besitzt. □

4 Registerspiele und Separierende Automaten

Im folgenden Abschnitt kommen wir nun zu einem der Verfahren, mit dem Paritätsspiele in quasipolynomieller Zeit gelöst werden können. Von den drei Methoden *play summaries, progress measure* und *Registerspiele* schauen wir uns dabei die Registerspiele an. Dies liegt daran, das Registerspiele nicht zwingend Determinismus voraussetzen und somit am kompliziertesten mit den Separatoren in Zusammenhang zu setzten sind. Neben dem formalen Aufbau wird dabei natürlich auch gezeigt, dass sie auch den Zweck erfüllen, für den sie gedacht sind. Außerdem wird auf den Registerindex eingegangen, da dieser der entscheidende Punkt für die Laufzeit ist. Anschließend befassen wir uns mit den Separatoren, welche eine Alternative für die Funktionalität der Registerspiele darstellen. Gleichzeitig kann damit als alternative für den quasipolynomiellen Algorithmus der Registerspiele eine Reduktion von Parität auf Safety-Spiele gezeigt werden. Nun wurden die Registerspiele aber inzwischen überarbeitet, wodurch die Übereinstimmung von Registerspiel und Separator nicht so einfach zu erkennen ist. Daher werden abschließend noch die aktuellen Definitionen aneinander angepasst.

4.1 Registerspiele

Lethinen [4] hatte in ihrer Arbeit die Registerspiele als neuen Methode vorgestellt, welche später von Colcombt und Fijalkow [5] als eine mögliche Darstellung von good-for-small-Games Automaten erkannt wurde. Dies ist eine der drei bisher bekannten Techniken, mit denen Paritätsspiele in quasipolynomieller Zeit lösbar sind. In einem späteren Paper [6] hat Lehtinen zusammen mit Boker die Registerspiele nochmal überarbeitet, im Folgenden wird daher auf die überarbeitete Fassung eingegangen.

Definition Registerspiel Dabei ist ein Registerspiel eine Variante eines Paritätsspiels, bei dem das Spiel um eine Anzahl an Registern r_0 bis r_k ergänzt wird. Damit können zum Beispiel vorher aufgetretene Prioritäten gespeichert werden. Im Folgenden wird die Definition der Registerspiele von Lethinen erläutert, dabei wird davon ausgegangen, dass im ursprünglichen Spiel G die Prioritäten an den Knoten liegen. Zu Spielbeginn sind diese Register leer sprich $r_i = 0$ und werden im Laufe des Spiels folgendermaßen genutzt. In jedem Zug wählt einer der Spieler wie gehabt einen Zug mit Priorität p aus. Nun wählt Eva einen Index i, $0 \leq i \leq k$ aus, welcher die Updates der Register folgendermaßen bestimmt: für $j < i : r_j = 0$, für $j = i : r_j = p$ und für $j > i : r_j = max(r_j, p)$. Vor dem Update findet eine Ausgabe statt, die wie folgt entsteht, wenn $max(r_i, p)$ gerade ist, wird $2i$ aus-

gegeben, ansonsten $2i+1$. Eva gewinnt das Spiel, wenn die größte Ausgabe, die unendlich oft auftaucht, gerade ist. Entsprechend muss die Gewinnstrategie für Eva nicht nur die Positionen auswählen, sondern auch die Indexe der Register.

Bevor wir fortfahren, sollte noch kurz gezeigt werden, dass diese Erweiterung die Spiele nicht mächtiger macht, als sie es bisher sind. Da das Spiel eine maximale Priorität p besitzt, gibt es nur endlich viele Werte die in den Registern auftauchen können. Entsprechend ist die Anzahl aller möglichen Registerzustände endlich. Wenn man nun die Register als einen Vektor auffasst, dann ist das Kreuzprodukt aus den Positionen des Spiel und den möglichen Vektoren immer noch endlich, und man kann des Kreuzprodukt als neu Menge an Positionen für eine bereits bekannte Spielart nutzen und das Registerspiel simulieren. Damit sind Registerspiele nicht mächtiger als die bereits bekannten Spiele.

Im Folgenden wird die Definition Formal erklärt.

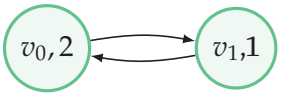
Definition 7. *Zu den Paritätsspiel $G = (V^G, R^G, L^G)$, wobei I die Prioritäten $L^G : V^G \to I$ darstellt, bildet $R_E^k(G) = (V, R, L)$ das zugehörige Registerspiel mit k Registern, bei dem Eva die Register kontrolliert. Die Labelfunktion, die den Kanten die Prioritäten zuweist, ist formell durch $L : R \to [0, ..., 2k+1]$ definiert. Außerdem wird noch eine Variable $t \in \{0,1\}$ eingefügt, die aussagt, ob der nächste Zug ein Zug im zugrundeliegenden Spiel G ist (t=1) oder Evas Auswahl des Registers (t=0) beinhaltet. Die Positionen V in $R_E^k(G)$ sind dabei ein Produkt aus den Positionen von G, dem Registerzuständen, welche aufsteigend sortiert sind, und der Variable t, also $(v, \vec{r}, t) \in V^G \times I^{k+1} \times \{0,1\}$. In der überarbeiteten Fassung wird die Startposition des Registerspiels nicht explizit genannt, da aber auch keine Änderungen zur alten Variante genannt werden, ist die Startposition die gleiche wie in der alten Variante $(v_i, \vec{0}, 0)$. Dies ist ein Resetzustand, er dient dazu, dass die Priorität, die in G am Startzustand vorliegt, berücksichtigt wird, bevor der Spielzug simuliert wird. V_A besteht dabei aus alle Positionen $(v, \vec{r}, 1)$ für die $V \in V_A^G$ gilt, und V_E besteht aus allen anderen Positionen, Eva kontrolliert entsprechend die Zustände, in denen ein Register ausgewählt wird. Für die Kanten gilt $R = E_{move} \cup E_i$ für alle $i \in [0, ..., k]$. Dabei stellt E_{move} die Kanten dar, die die Züge in G simulieren und besteht aus $((v, \vec{r}, 1), (w, \vec{r}, 0))$ für $(v, w) \in R^G$. E_i hingegen simuliert das Updaten der Register und zwar folgendermaßen: Für alle $i \in [0, ..., k]$ gilt $((v, \vec{r}, 0), (v, \vec{r}', 1))$ mit $r'_j = 0$ für $j < i$, $r'_j = L^G(v)$ für $j = i$ und $r'_j = max(r_j, L^G(v))$ für $j > i$. Die Prioritäten werden wie folgt gebildet. Kanten aus E_{move} haben die Priorität 0 und Kanten aus E_i der Form $((v, \vec{r}, 0), (v, \vec{r}', 1))$ haben die Priorität $2i$ wenn $max(r_i, L^G(v))$ gerade ist und $2i+1$ wenn es ungerade ist.*

Nun ist eine solche Konstruktion nicht hilfreich, wenn sie nicht den Zweck erfüllt, für denn sie gedacht ist, daher wurden von Lethinen zwei wichtige Eigenschaften gezeigt.

Lemma 2. *Erstens wenn Adam eine Gewinnstrategie für G von v aus hat, dann hat er auch eine für $R_E^k(G)$ von v aus für jedes beliebige k. Zweitens wenn Eva eine Gewinnstrategie für G mit Prioritäten I von v aus hat, dann hat sie auch eine für $R_E^k(G)$ von v aus für $k > i$ wobei $2i$ die größte gerade Priorität in I ist.*

Beweis. Beim ersten Fall ist die grundlegende Beweisidee, dass die größte Priorität p, die unendlich oft auftritt, ungerade ist, da Adam eine Gewinnstrategie hat, und dass es einen

(a) Spiel mit Registerindex 0



(b) Spiel mit Registerindex 1

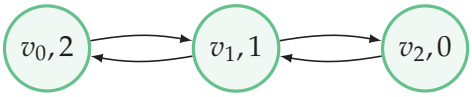


Abbildung 4.1: Beispiel für zwei Spiel

Register r_i gibt, den Eva unendlich oft auswählt. Dieser existiert, weil es nur endlich viele Register gibt, das Spiel aber unendlich lang ist. Es gibt also einen Zug, ab dem keine größere Priorität als p und kein größerer Register als r_i auftaucht. Immer wenn nun p auftaucht wird r_i der Wert p zugewiesen und da keine größeren Prioritäten mehr auftauchen, bleibt dieser Wert in r_i bis Eva r_i updatet. Wenn nun r_i upgedatet wird, wird $2i+1$ als Ausgabe erzeugt. Da dies unendlich oft passiert, ist $2i+1$ die größte unendlich oft auftauchende Ausgabe und da es ungerade ist, gewinnt Adam das Spiel.

Beim zweiten Fall ist die Idee, dass Eva genug Register zur Verfügung hat, damit die höchste gerade Priorität, die unendlich oft auftritt, ab eine bestimmten Zeitpunkt als einziger Wert in einem Register auftaucht. Da es eine größte gerade Priorität der Form $2i$ für ein i aus den natürlichen Zahlen gibt, hat Eva für $R_E^k(G)$ mit $k \geq i$ eine Gewinnstrategie, die wie folgt aussieht. Für die Auswahl der Kanten geht sie genauso vor wie in G. Die Register wählt sie folgendermaßen aus: wenn die Prioritäten $2j$ bzw. $2j+1$ auftritt, wählt sie Register j aus. Da Eva eine Gewinnstrategie für G hat, ist die höchste Priorität, die unendlich oft auftaucht gerade, besitzt also die Form $2l$ für ein l aus den natürlichen Zahlen, entsprechend wird Register l unendlich oft upgedatet. Ab dem Punkt, wo keine größeren Prioritäten als $2l$ mehr auftauchen, wird der Wert in Register l immer $2l$ bleiben, da die kleineren Prioritäten immer in niedrigeren Registern gespeichert werden. Da Register l unendlich oft ausgewählt wird, wird auch die Ausgabe $2l$ unendlich oft erzeugt, entsprechend gewinnt Eva das Registerspiel. □

Der Registerindex Entsprechend des obigen Beweises wurde mit dem Register-Index eine Möglichkeit gegeben, die Komplexität solcher Spiele zu bestimmen. Der Register-Index k beschreibt den höchsten Index eines Registers, bei der kleinsten Anzahl an nötigen Registern, die Eva braucht, um ein Registerspiel zu gewinnen, für dessen zugrundeliegendes Parity-Spiel sie eine Gewinnstrategie hat. Formaler ausgedrückt bedeutet Register-Index k, dass Eva für das Spiel $R_E^k(G)$ eine Gewinnstrategie besitzt, für $R_E^{k-1}(G)$ aber keine Gewinnstrategie besitzt. Dabei ist anzumerken, dass die im obigen Beweis genutzte Anzahl an nötigen Registern nicht immer das Optimum darstellt.

Beispielsweise besteht eine Strategie, in einem Spiel, in dem Eva verhindern kann, das ungerade Prioritäten unendlich oft besucht werden, daraus, immer Index 0 auszuwählen. Dadurch können nur gerade Ausgaben unendlich oft auftreten und somit ist auch die

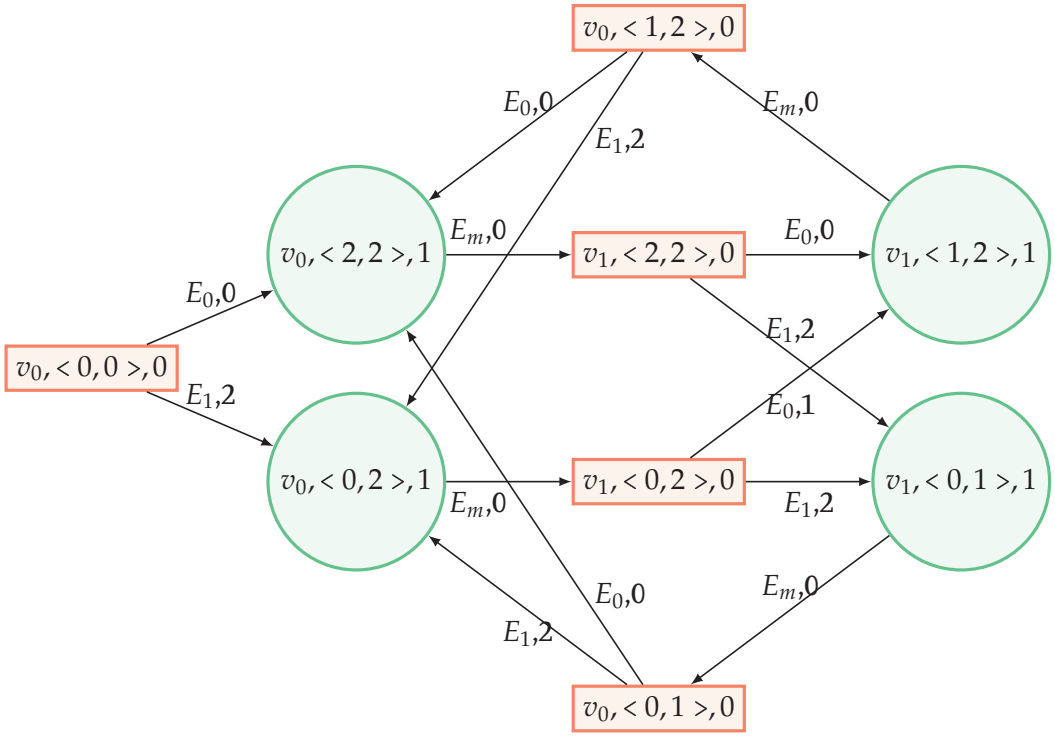


Abbildung 4.2: $R^2_E(G)$

höchste unendlich oft auftauchende Ausgabe gerade. Eva braucht für solche Spiele also nur das Register r_0 und dieses Spiel hat den Register-Index 0. In den Abbildungen 4.1a und 4.1b sind weiter Beispiele zu sehen. Im ersten Fall beträgt der Registerindex ebenfalls 0, denn da $max(p, r_0)$ immer 2 ist, ist die Ausgabe immer gerade. Im zweiten Fall hingegen hat Adam bei Registerindex 0 eine Gewinnstrategie, nämlich beim Erreichen von v_1 die Kante nach v_2 zu wählen und danach zwischen v_1 und v_2 zu wechseln, wodurch Eva die Ausgabe 1 unendlich oft erzeugen muss. Hat Eva aber zwei Register, so kann sie Priorität 1 immer in r_0 und Priorität 0 immer in r_1 speichern und somit die Ausgabe 2 unendlich oft erzeugen.

Dabei gilt, dass es für jedes natürliche n ein Spiel mit Register-Index n existiert, dieses lässt sich nämlich wie von Lehtinen gezeigt [6] rekursiv erzeugen. Dafür werden Spiele betrachtet, in denen die Prioritäten an den Kanten liegen, außerdem gehören alle Positionen zu Adam. Dann bildet das Spiel, das nur aus einem Knoten besteht und nur eine Kante mit Priorität 0 besitzt, die eine Schleife an dem Knoten bildet, den Anfang. Dieses Spiel G_0 besitzt definitiv Registerindex 0. Wenn nun ein Spiel G_n mit Registerindex n zur Verfügung steht, wird ein Spiel mit Index $n+1$ wie folgt konstruiert. Man nimmt zwei Kopien des Spiels G_n und verbindet ihre Startpositionen p_i und $p_{i'}$ miteinander. Und zwar geht eine Kante mit Priorität $2n+1$ von p_i nach $p_{i'}$ und eine mit Priorität $2(n+1)$ von $p_{i'}$ nach p_i. Das so konstruierte Spiel hat Registerindex $n+1$, das bedeutet, dass Adam in diesem Spiel mit n Registern eine Gewinnstrategie besitzt, was im Folgenden bewiesen wird. Hierbei ist entscheidend, dass G_n nur Prioritäten bis $2n$ enthält. Für G_0 gilt diese

Aussage eindeutig und wenn sie für G_n gilt, dann nach Konstruktion auch für G_{n+1}. Die Strategie sieht wie folgt aus. Zuerst benutzt Adam die Kante von p_i nach $p_{i'}$, dadurch wird Eva gezwungen den Wert $2n+1$ in den Register r_n zu speichern, da r_n der größte Register des Spiels ist und nach Definition des Registerupdates das Maximum aus aktuellen Wert im Register und der Priorität erhält. Anschließend spielt Adam in diesem Teilspiel G_n weiter. Um nun im Teilspiel nicht zu verlieren, muss Eva ihrer Strategie für G_n folgen, diese setzt aber voraus, dass der Register r_n benutzt wird. Da in G_n alle Prioritäten kleiner als $2n+1$ sind, ist Eva somit gezwungen $2n+1$ als Ausgabe zu erzeugen. An diesem Punkt bewegt Adam sich zurück zu $p_{i'}$ und nimmt die Kante mit Priorität $2(n+1)$ nach p_i, wodurch r_n nun $2(n+1)$ als Wert enthält. Jetzt spielt Adam im anderen Teilspiel weiter und auch hier wird Eva irgendwann den Register r_n nutzen müssen. Zu diesem Zeitpunkt wird zwar die Ausgabe $2n$ erzeugt, aber jetzt enthält r_n einen Wert der kleiner als $2n+1$ ist. Nun kann Adam nach p_i zurückkehren und die Kante mit Priorität $2n+1$ nach $p_{i'}$ benutzen, dadurch enthält r_n nun den Wert $2n+1$. Dieses Vorgehen kann Adam unendlich oft wiederholen und somit die Ausgabe $2n+1$ unendlich oft erzeugen. Adam gewinnt also bei n Registern. Stehen dagegen $n+1$ Register zur Verfügung kann Eva gewinnen. Dazu muss sie nur, wenn Adam von $p_{i'}$ nach p_i wechselt den Register $n+1$ auswählen und somit die Ausgabe $2(n+1)$ unendlich oft erzeugen. Somit hat das konstruiert Spiel Registerindex $n+1$.

In Abbildung 4.2 ist das zu Abbildung 4.1a gehörende Registerspiel mit n=2 Registern zu sehen. Die Knoten sind so benannt, wie es weiter oben beschrieben wurde, an den Kanten steht zuerst, welche Art von Kante es ist, und dann ihre Priorität. In diesem Beispiel wurden aus den 2 Knoten von G insgesamt 8 Knoten in $R_E^1(G)$. Allgemein erhöht die Konstruktion eines Registerspiels die Anzahl der Positionen stark, diese liegen bei einem Spiel G mit z Knoten und der größten Priorität p für $R_E^k(G)$ nämlich in $O(zp^k)$. Dies entsteht dadurch, dass die neuen Zustände das Produkt der alten Zustände mit den Vektor, der die Registerinhalte abbildet, bilden. Der Vektor enthält so viele Einträge, wie es Register gibt, und jeder Eintrag kann einen Wert von 0 bis p enthalten. Entsprechent gibt es p^k verschiedene Kombinationen für den Vektor. Dies mit der Anzahl der Zustände ergibt zp^k viele Möglichkeiten. Dafür werden die Prioritäten kleiner, was in diesem Beispiel nicht zu sehen ist. Die höchste Priorität hängt im Registerspiel nämlich nur von der Anzahl der Register ab und ist deshalb maximal $2k+1$, wobei k der Registerindex ist. Der Registerindex ist aber auch von der Knotenzahl abhängig und lässt sich damit genauer abschätzen. Für ein Spiel mit z Knoten beträgt er maximal $\lfloor 1+\log z \rfloor$. Im Folgenden kann man, um das Spiel G zu lösen, stattdessen das Spiel $R_E^{\lfloor 1+\log z \rfloor}(G)$ lösen. Wenn man hierfür einen Algorithmus benutzt, bei dem die Prioritäten den exponentielle Faktor bilden, hebt sich dies durch den Logarithmus wieder auf, dadurch erhält man für das ursprüngliche Spiel G einen quasipolynomiellen Algorithmus.

Lemma 3. *Zu einem Paritätsspiel mit z Zuständen und der größten Priorität p, existiert ein Registerspiel mit Registerindex $\lfloor 1+\log z \rfloor$.*

Entscheidend ist hier der von Lethinen [6, Kap. 4.1] durchgeführte Beweis, dass der

Registerindex tatsächlich logarithmisch beschränkt ist, welcher im folgenden dargestellt wird. In diesem Beweis werden defensive Gewinnstrategien und der defensive Registerindex genutzt, diese sind wie folgt definiert. Eine Strategie δ für $R_E^k(G)$ ist defensiv, wenn im Unterspiel G' von G mit höchster Priorität p gilt, dass von der Position $(v, \vec{r}, 0)$ mit $p \leq r_k$ und r_k gerade, ein Zug der mit δ übereinstimmt niemals $2k+1$ ausgibt. Ein Spiel G hat defensiven Registerindex k, wenn Eva für alle Positionen in $R_E^k(G)$ eine defensive Gewinnstrategie besitzt. An der Definition ist einfach zu sehen, dass der defensive Registerindex mindestens so groß wie der normale Registerindex sein muss, also gilt jede obere Schranke, die für den defensiven Registerindex gefunden wird, auch für den normalen Registerindex. Der eigentliche Beweis bezieht sich auf die Anzahl disjunkter Zyklen, da aber für endliche Graphen die Anzahl der Zyklen durch die Anzahl der Positionen beschränkt ist, gilt die logarithmische Schranke auch für die Anzahl an Positionen[1]. Zu beweisen ist also, dass ein endliches Parity-Spiel mit z disjunkten Zyklen maximal Registerindex $\lfloor 1 + \log z \rfloor$ besitzt.

Beweis. Zu Beginn kann man alle Spiele, für die Eva eine Gewinnstrategie besitzt, vereinfachen, indem man an allem Positionen die zu Eva gehören alle Kanten bis auf diejenige, welche in der Gewinnstrategie genutzt wir, löscht. Anschließend können alle Positionen an Adam übergeben werden, wodurch Evas Strategie nur noch die Auswahl der Register beinhaltet und nur noch diese im Beweis betrachtet werden müssen. Der Beweis ist eine Induktion über die Anzahl der Positionen n im Spiel G. Der Induktionsanfang mit $n = 1$ ist trivialerweise erfüllt. Da es nur eine Position gibt, kann es auch nur einen Zyklus geben. Dieser besteht aus einer einzelnen Kante, die eine Schleife über der Position bildet. Da diese die einzige auftauchende Priorität beinhaltet, braucht Eva auch nur einen Register um zu gewinnen. Im Induktionsschritt betrachte das Spiel G indem p die größte gerade Priorität ist, die in einem Zyklus auftritt. Sei G' das Spiel welches von den Prioritäten bis $p-2$ aus G induziert wird. Dann lässt sich G' zerlegen in die *maximal strongly connected subgames* $G_1, \ldots, G_j$, wobei gilt $j \leq z$, und seien dabei $k_1, \ldots, k_j$ die entsprechenden defensiven Registerindizes, wovon m der größte ist. Im Falle das es keine solchen Teilspiele gibt, enthalten alle Zyklen in G die Priorität p und Eva gewinnt defensiv in $R_E^1(G)$. Dazu muss sie nur r_1 wählen, wenn p auftaucht, und ansonsten r_0, dadurch ist die größte unendlich oft auftauchende Ausgabe gerade. Wenn es hingegen solche Teilspiele gibt, muss man drei verschiedene Fälle betrachten, erstens wenn $m = 0$ gilt; zweitens wenn $m > 0$ gilt und es genau ein i mit $k_i = m$ gibt; drittens wenn es i, j mit $i \neq j$ und $k_i = k_j = m$ gibt. In allen drei Fällen ist die Grundidee, dass für die Teilspiele G_j nur die unteren Register benötigt werden und der größte Register nur benutzt wird, wenn p auftaucht.

1. Fall $m = 0$: In diesem Fall kann Eva das $R_E^1(G)$ Registerspiel gewinnen. Dazu nutzt sie innerhalb eines Teilspiels G_j einfach ihre Strategie für das 0-Registerspiel und immer wenn die Priorität p in einem Zyklus auftritt wählt sie Register 1. Entweder bleibt das Spiel immer in einem Teilspiel und folgt dort einer Gewinnstrategie oder die Priorität p

[1] Lethinen hat auch einen direkten Beweis geführt [4, Kap. 4.3], dieser bezog sich aber noch auf die alte Konstruktion der Registerspiele.

taucht unendlich oft auf, wodurch der Wert 2 unendlich oft ausgegeben wird, der Wert 3 aber nur endlich oft. Wenn der höchste Register von Beginn an eine Priorität größer oder gleich der größten Priorität in G enthält, wird 3 niemals ausgegeben und damit ist die Strategie defensiv.

2. Fall $m > 0$ und es gibt genau ein i mit $k_i = m$: In diesem Fall ist der defensive Registerindex von G nicht größer als m und nach Induktionsvoraussetzung ist $m \leq 1 + \log z$. Evas Gewinnstrategie ist wie folgt. Innerhalb eines Teilspiels G_j nutzt sie ihre defensive Gewinnstrategie für G_j und nutzt dabei nur die unteren Register bis k_j. Außerhalb der Teilspiele wählt sie r_m wenn die Priorität p auftaucht und ansonsten r_0. Sollte die Startbelegung von Register r_m ein ungerader Wert größer als p enthalten, so wird beim ersten auftauchen von p der Wert $2k_m + 1$ ausgegeben und r_m erhält den Wert p. Sollte die Startbelegung ein gerader Wert größer p sein, ist die Ausgabe $2k_m$ und r_m erhält ebenfalls den Wert p. Da alle anders Wert, die außerhalb der Teilspiele auftauchen, immer in r_0 gespeichert werden, enthält r_m ab diesen Punkt immer wenn ein Teilspiel G_j betreten wird den Wert p. Da die Strategie für G_j defensive ist, wird auch in G_j nie $2m + 1$ ausgegeben. Wenn nun die Priorität p auftaucht enthält r_m entweder bereits p oder eine kleinere Priorität aus G_j, entsprechend ist $max(r_m, p)$ gerade und es wird $2k_m$ ausgegeben. Das Spiel kann nun auf zwei Arten ablaufen, entweder bleibt es ab einem Punkt unendlich lange in einem Teilspiel G_j, wofür Eva eine defensive Gewinnstrategie besitzt, oder es wird unendlich oft zwischen den Teilspielen gewechselt. Im zweiten Fall taucht in den Zyklen die Priorität p unendlich oft auf und erzeugt die Ausgabe $2k_m$ unendlich oft. Entsprechend hat Eva eine defensive Gewinnstrategie.

3. Fall i, j mit $i \neq j$ und $k_i = k_j = m$: In diesem Fall ist der defensive Registerindex von G nicht größer als $m + 1$. Da nach der Induktionsvoraussetzung G_i und G_j jeweils höchstens $2^{k_m - 1}$ viele disjunkte Zyklen haben und G_i und G_j disjunkt sind, enthält G maximal 2^{k_m} viele disjunkte Zyklen, weshalb $m + 1$ genügt. Evas Strategie ist ähnlich wie im vorherigen Fall. Im Teilspiel G_j nutzt sie ihre Gewinnstrategie für das k_j-Registerspiel mithilfe der Register bis k_j, außerhalb wählt sie r_{m+1}, wenn p in einem Zyklus auftaucht, und ansonsten r_0. Entweder das Spiel bleibt irgendwann innerhalb eines Teilspiels, dann gewinnt Eva dort. Oder es wechselt unendlich oft zwischen den Teilspielen, dann taucht die Priorität p unendlich oft auf. Weil r_{m+1} nach dem ersten Auftauchen von p nur noch den Wert p enthält, ist $2(m + 1)$ die größte Ausgabe, die unendlich oft auftaucht. Da die Strategien für G_j defensiv sind, wird dort keine Ausgabe größer als $2m$ erzeugt, somit ist die Gesamtstrategie auch defensiv.

□

Wie weiter oben bereits erläutert, hängen Spiele und Automaten oft eng zusammen, entsprechend gibt es für Registerspiele auch Registerautomaten, welche die Sprachen der Spiele erkennen. Ein solcher Registerautomat ist nicht *good-for-Games*, da der Automat einen Register-Index k zugewiesen hat, aber Eva auch in Spielen mit Register-Index größer als k eine Gewinnstrategie besitzen kann. Er ist lediglich *good-for-small-Games* da 2^{k-1} die Größe der Spiele beschränkt, für die eine Gewinnstrategie von Eva garantiert werden

kann.

4.2 Separator

Czerwinski et al. [5] haben einen Separator-Ansatz entwickelt und Registerspiele darauf zurückgeführt, wobei sie sich auf die erste Variante von Lehtinen [4] beziehen und nicht auf die Erweiterte [6] mit anderer Resetmechanik. Eine Schwierigkeit hierbei ist, dass Registerspiele nichtdeterministisch sein können, aber (n,d)-separierende Automaten grundsätzlich nur auf deterministischen funktionieren. Das Problem ist, das sich beim Bilden des Kreuzprodukt Safety-Spiels die Gewinnstrategie nicht übertragen lässt, da es nötig sein kann einen zukünftigen Zug des Gegenspielers zu raten, wodurch sich der Nichtdeterminismus nicht auflösen lässt. Allerdings gibt es eine Klasse von nichtdeterministischen Automaten, bei denen das möglich ist, und zwar die *good-for-Games* Automaten. Dies liegt an der in Kapitel 3 erläuterten Äquivalenz zwischen *good-for-Games* und *history determinism*. Damit die Bedingung von good-for-Games erfüllt wird, muss die Definition von separierenden Automaten angepasst werden. Ein nichtdeterministischer Automat A ist ein *good-for-separation strong (n,d) Separator*, wenn er jedes Wort in $LimsupOdd_d$ ablehnt und Eva für jedes gerade Spiel G mit n Knoten und Prioritäten bis d eine Gewinnstrategie in $G \times A$ hat.

Der hier gewählte Ansatz ist, sich die Graphenstruktur anzuschauen. Dabei wird ein nichtdeterministischer Paritätsautomat $R_{n,d}$ derart konstruiert, dass die Konstruktion $R_E^k(G)$ von Lehtinen zum Paritätsspiel G grundlegend mit $G \times R_{n,d}$ übereinstimmt. Dabei stellt $R_{n,d}$ einen *good-for-separation strong (n,d) separator* dar. Es gibt eine quasipolynomielle untere Schranke für die Größe dieser Automaten, welche später nachgewiesen wird.

Formal ist der nichtdeterministische-Paritätsautomat $R_{n,d}$ über dem Alphabet Σ_d, welches die ganzen Zahlen von 1 bis d darstellt, wie folgt definiert.

Definition 8. *Die Zustände bilden die Sequenz der Register $\langle r_{\lfloor 1+\lg(n)\rfloor}, ..., r_2, r_1\rangle$ mit $r_i \leq r_{i+1}$ und $r_i \in \{1, 2, ..., d\}$. Der Startzustand ist dabei $\langle 1, 1, ..., 1\rangle$. Von jedem Zustand $s = \langle r_{\lfloor 1+\lg(n)\rfloor}, ..., r_2, r_1\rangle$ ausgehend gibt es zwei Arten von Zuständen, wobei im Folgenden immer $p \in \Sigma_d$ und $k, 1 \leq k \leq \lfloor 1 + \lg(n)\rfloor$ definiert sind. Erstens für jedes p den als Update von s mit p bezeichneten Zustand $\langle r_{\lfloor 1+\lg(n)\rfloor}, ..., r_k, p, ..., p\rangle$, wobei das kleinste k gewählt wird, sodass $r_k > p$ gilt. Zweitens für jedes k den als k-Reset von s bezeichneten Zustand $\langle r_{\lfloor 1+\lg(n)\rfloor}, ..., r_{k+1}, r_{k-1}, ..., r_1, 1\rangle$. Bei den Zustandsübergängen existieren auch wieder zwei verschiedene Arten. Zum Einen existieren die nicht-reset Übergänge von s nach s', wenn s' das Update von s mit p bildet, der Form $(s, p, 1, s')$ mit Priorität 1. Zum Anderen existieren die Reset von Register k Übergänge. Wenn zum Zustand s s' das Update von s mit p bildet und s'' der k-Reset von s' ist, existiert ein Übergang der Form $(s, p, 2k, s'')$, wenn r_k gerade ist, bzw. $(s, p, 2k + 1, s'')$, wenn r_k ungerade ist, dabei wird r_k von s' betrachtet, also dem Zustand in dem der Reset stattfindet. Im Grunde sind es also zwei Übergänge, die zusammen abgearbeitet werden.*

In Abbildung 4.3b ist nun der Separator $R_{2,2}$ zum Spiel aus Abbildung 4.3a zu sehen. Die Zustände geben dabei die Register an, dabei ist zu bedenken, dass es sich um einen Automaten und kein Spiel handelt, die Zustände werden also nicht zwischen Adam und

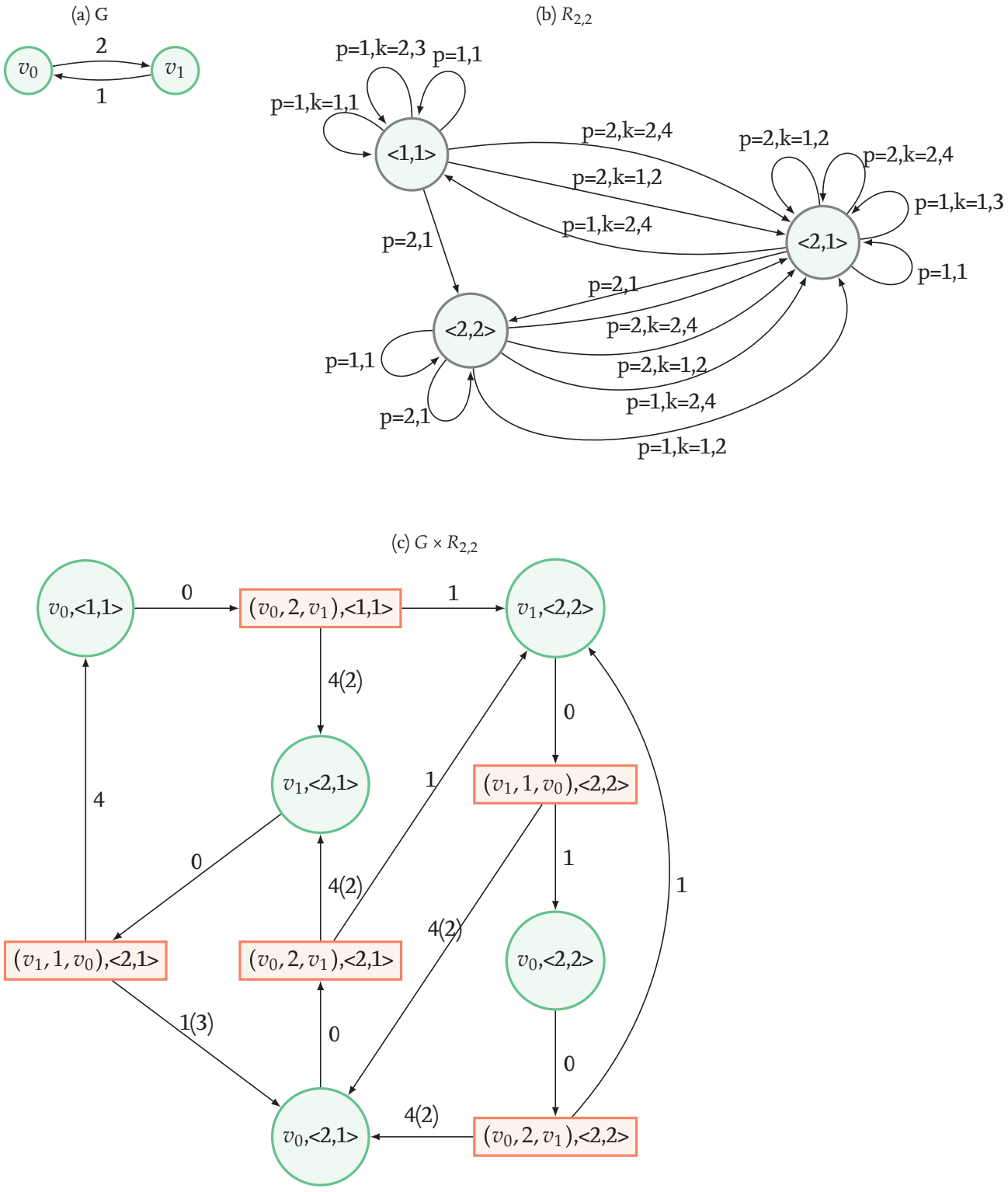


Abbildung 4.3: Entstehung eines Separators

Eva unterschieden. Die Werte an den Kanten geben an, welche Priorität p auftrat, welcher Register k benutzt wurde und welche Priorität die Kante hat. Wenn nur p angegeben ist handelt es sich um einen Übergang, in dem nicht resettet wird, wenn p und k angegeben sind, ist es ein Reset von Register k. Der Übergang $p = 2, k = 1, 2$ von $\langle 1, 1\rangle$ nach $< 2, 1 >$ ist zum Beispiel ein Reset von Register 1. Dabei sind $s = \langle 1, 1\rangle$ $s' = \langle 2, 2\rangle$ und $s'' = \langle 2, 1\rangle$ und für die Priorität wird r_1 aus $\langle 2, 2\rangle$ betrachtet. Zu beachten ist, dass nur die Abfolge der Prioritäten die Eingabe für den Automaten bildet, die Entscheidung welcher Register resettet wird ist aus Automatensicht entsprechend nichtdeterministisch.

In Abbildung 4.3c wiederum ist das Produkt $G \times R_{2,2,}$ zu sehen. Da Eva im ursprünglichem Spiel keine Positionen besitzt, sind Evas einzige Positionen hier die Automatenpositionen. Im Paper von Czerwinski et al. [5, Kap 4.3] wird nicht explizit angesprochen, dass Eva diese Positionen bekommt. Aber es heißt, dass die Wahl, die den Nichtdeterminismus auflöst, an Eva gegeben wird. Dies und die Tatsache, dass in der Definition des Produkts ebenfalls die Automatenpositionen an Eva gegeben werden, machen deutlich, dass es hier ebenfalls so gedacht ist. Laut Definition des Produkts haben die Übergänge von Spielpositionen zu Automatenpositionen die Priorität ϵ. Da in unserem Zusammenhang nicht eindeutig ist, wie das Spiel mit einer Priorität umgeht, die keine ganze Zahl ist, wurden diese Übergänge stattdessen mit der Priorität 0 versehen. Da 0 kleiner als alle möglichen Prioritäten ist, die auftauchen können, wird dadurch auch das Ergebnis nicht verändert. Denn da zwischen zwei Übergängen mit Priorität 0 immer ein anderer Übergang mit höherer Priorität liegt, kann nur dann unendlich oft die 0 auftreten, wenn auch eine höhere Priorität unendlich oft auftritt. Im Graphen sieht man außerdem mehrere Übergänge, an denen mehrere Zahlen stehen. In diesem Fall führt die Auswahl verschiedener Registerindizes zur gleichen Belegung der Register, es existieren also theoretisch verschiedene Übergänge. Da Eva die Wahl zwischen den Übergängen hat, kann sie immer den für sie vorteilhafteren nehmen und der andere kann aus dem Spiel entfernt werden, ohne das Ergebnis zu verändern. Dabei wählt Eva bei geraden Prioritäten immer die größere und bei ungeraden die niedrigere, sollten gerade und ungerade gleichzeitig auftreten, was in diesem Beispiel nicht der Fall ist, wählt Eva immer die gerade. Die ausgewählte Priorität ist in diesem Fall die erste Zahl, während in der Klammer die andere Priorität der Vollständigkeit halber aufgeführt ist.

Da wie gesagt R_G und $G \times R_{n,d}$ grundlegend übereinstimmen, können alle von Lehtinen bewiesenen Aussagen über R_G auch auf $G \times R_{n,d}$ angewandt werden. Wenn Adam also eine Gewinnstrategie in G hat, besitzt er auch eine in $G \times R_{n,d}$, und für Gewinnstrategien von Eva in G gilt für $G \times R_{n,d}$ der Registerindex, also das Eva maximal $\lfloor 1 + \lg n \rfloor$ Register braucht, um in $G \times R_{n,d}$ zu gewinnen. Man sieht, dass der Registerindex bereits bei der Konstruktion von $R_{n,d}$ genutzt wurde, entsprechend besitzt Eva eine Gewinnstrategie in $G \times R_{n,d}$. Gleichzeitig besitzt Adam keine Gewinnstrategie, also werden die Wörter aus $LimsupOdd_d$ abgelehnt. Daraus folgt, dass $G \times R_{n,d}$ tatsächlich ein *good-for-separation strong (n,d) separator* ist.

Reduktion auf Safety Wie in Kapitel 3 bereits gesagt, besteht die Idee darin, den Parität auf einen Safety Automaten zu reduzieren, denn für Safety sind bereits effiziente Algorithmen zum Lösen bekannt, allerdings ist $R_{n,d}$ noch ein Paritätsautomat. Czerwinski et al. [5, Kap 4.4] haben aber auch gezeigt, dass das Produkt von $R_{n,d}$ mit einem *deterministischen safety strong (n',d')-separator* S den *safety good-for-separation strong (n,d)-separator* $R_{n,d} \times S$ ergibt. Dabei gilt das d' die höchstmögliche Priorität und das n' die Anzahl der Zustände in $R_{n,d}$ sind. Wie bereits gezeigt besitzt Eva eine Gewinnstrategie δ für $G \times R_{n,d}$, diese ist sogar positional. Wenn man nun den Untergraphen G' dieser Strategie bildet, ist dieser gerade, hat maximal n' Knoten und Prioritäten bis p'. Da S konkret als *deterministischen safety strong (n',d')-separator* gewählt wurde, wird für jeden Pfad in G' die Sequenz der Prioritäten von S akzeptiert. Entsprechend gewinnt Eva auch immer das Spiel $(G \times R_{n,d}) \times S$, wenn sie die Strategie δ in $G \times R_{n,d}$ benutzt, und gewinnt damit auch das äquivalente Spiel $G \times (R_{n,d} \times S)$. Da $R_{n,d}$ als *strong (n,d)-separator* konstruiert wurde, lehnt er jedes Wort aus $LimsupOdd_d$ ab, folglich ist die höchste unendlich oft auftauchende Priorität in einem solchen Wort ungerade. Entsprechend ist auch jedes solche Wort, das der Automat $R_{n,d} \times S$ erhält in $LimsupOdd'_d$ und wird wegen der Auswahl von S als *deterministischen safety strong (n',d')-separator* abgelehnt. Insgesamt erfüllt damit $R_{n,d} \times S$ die Bedingungen eines *safety good-for-separation strong (n,d)-separator*.

Außerdem haben Czerwinski et al. [5, Kap. 4.1] einen Vorschlag für einen solchen safety Automaten S gegeben. Dabei wird ein Automat $U_{n,d}$ konstruiert, wobei die Idee ist, dass $U_{n,d}$ zählt, wie oft eine ungerade Priorität p auftaucht, ohne das eine größere Priorität auftaucht. Wenn eine Priorität öfter auftaucht, als das Spiel Positionen besitzt, bedeutet das, dass die Priorität in einem Zyklus auftritt. Da keine größere Priorität auftritt, ist die ungerade Priorität die größte in diesem Zyklus, entsprechend gibt es im Spiel eine Möglichkeit, dass die größte unendlich oft auftretende Priorität ungerade ist. Dabei beruht $U_{n,d}$ auf einen *(n,d)-universellen* Baum, in dem die Blätter lexikografisch geordnet sind. Die Zustände in $U_{n,d}$ bestehen dabei aus den Blättern vom Baum sowie einen ablehnenden Zustand a. Den Startzustand bildet das Blatt mit der höchsten Ordnung. Die Übergangsfunktion δ ist für alle $p \in \Sigma_d$ wie folgt definiert. $\delta(a, p) = a$, ist also der ablehnende Zustand einmal erreicht, kann er nicht mehr verlassen werden. Für alle $s \neq a$ gilt $\delta(s, p) = s'$ mit, entweder s' das größte Blatt mit $s|_p = s'|_p$ wenn p gerade, s' das größte Blatt mit $s|_p > s'|_p$ wenn p ungerade oder $s' = a$ wenn p ungerade und kein solches Blatt existiert. Dabei ist $s|_p$ die sogenannte p-truncation mit $1 \leq p \leq d$. Für ein S der Form $\langle k_{d-1}, k_{d-3}, \ldots, k_1 \rangle$ bildet $s|_p$ $\langle k_{d-1}, k_{d-3}, \ldots, k_p \rangle$ oder $\langle k_{d-1}, k_{d-3}, \ldots, k_{p+1} \rangle$, wenn p ungerade bzw. gerade ist. $U_{n,d}$ ist nun ein *strong (n,d)-separator* und die Idee für den Beweis dazu wird im Folgender erläutert.

Das der Automat Worte aus $LimsupOdd_d$ ablehnt, ist dadurch gegeben, dass bei ungeraden Prioritäten die Übergangsfunktion $\delta(s, p) = s'$ mit $s|_p > s'|_p$ genutzt wird. Wenn die größte unendlich oft auftretende Priorität ungerade ist, würde dadurch eine unendliche Kette an kleiner werdenden Elementen entstehen. Da der zugrunde liegende Baum aber endlich ist, wird irgendwann kein kleineres Blatt mehr existieren und der Automat in den ablehnenden Zustand a übergehen. Das der Automat Worte aus $EvenCycles_{n,d}$ akzep-

tiert, ist dadurch gegeben, dass für einen geraden Graphen G mit n Knoten und Prioritäten bis d ein *progress measure* μ existiert, das die Knoten in die Menge der Blätter eines (n,d)-universellen Baumes überträgt. Ein progress measure ist das Labeln eines Baumes, wenn für jede Kante (v,u) mit Priorität p gilt: Wenn p gerade ist, gilt $\mu(v) \geq \mu(u)$, und wenn p ungerade ist, gilt $\mu(v) > \mu(u)$. Wenn man den Durchlauf $\langle q_1, q_2, q_3, \ldots\rangle$ über das Wort betrachtet, gilt nach Definition des Initialzustandes $q_1 \geq \mu(v_1)$. Daraus folgt induktiv $q_i \geq \mu(v_i)$, wodurch der Durchlauf niemals den ablehnenden Zustand a erreicht und somit akzeptiert.

Da dies nun gezeigt ist, ist $R_{n,d} \times U_{n',d'}$ ein *safety good-for-separation strong (n,d)-separator* mit dem ein Paritätsspiel G auf ein Safety-Spiel reduziert werden kann. Und da sowohl $R_{n,d}$ also auch $U_{n',d'}$ eine quasipolynamiale Schranke für die Anzahl der Zustände besitzen, besitzt auch $R_{n,d} \times U_{n',d'}$ eine solche Schranke. Benutzt man nun einen Algorithmus, dessen Laufzeit von der Anzahl der Zustände abhängt, kann man das so entstandene Safety-Spiel in quasipolynomieller Zeit lösen.

Auch die anderen beiden Methoden, um Paritätsspiele in quasipolynomieller Zeit zu lösen nämlich Play-Summaries und progress measures lassen sich über Separatoren erklären. Progress measures wurde oben bereits kurz angesprochen und der Beweis dafür funktioniert ähnlich. Im Falle von Play-Summaries sind nur das Vertauschen der Spieler und eine Anpassung der Gewinnbedingung nötig, um einen Separator zu erhalten.

Der Nachweis dieser Schranke, welche auch für die beiden anderen anfangs erwähnten Algorithmen, mit denen man Paritätsspiele quasipolynomiell lösen kann, gilt, beruht darauf, dass alle gezeigten Strukturen einen universellen Baum beinhalten. In diesem Fall enthält jeder safety strong (n,d)-separator einen universellen Baum, dies wird in Kap. 5 genauer erläutert.

4.3 Vergleich und Anpassung

Wie gesagt sollen für die erste Variante der Registerspiel R_G und $G \times R_{n,d}$ übereinstimmen, allerdings sieht man an den Abbildungen 4.2 und 4.3c, das dies für die zweite Variante nicht der Fall ist. Dies liegt zum Einen an Unterschieden in den Definitionen und zum Anderen daran, dass nachdem die Registerspiele überarbeitet wurden, die Resetmechanik nicht mehr übereinstimmt. Daher werden im Folgenden die Definitionen von den Registerspielen und den Separatoren überarbeitet und aneinander angepasst, um sie sinnvoll zu vergleichen.

Die erste Anpassung betrifft die Tatsache, dass bei Lehtinen im Spiel G 0 als Priorität zugelassen wird, bei den Separatoren aber nicht, dadurch wird bei den Registerspielen beim Reset der Register auf 0 gesetzt statt auf 1 wie beim Separator. Die Anpassung besteht darin, dass bei den Registerspielen die Register zu Anfang mit 1 belegt sind und der Reset den Wert auf 1 setzt. Gleichzeitig geht man davon aus, dass das Ausgangsspiel G nur positive Prioritäten enthält, sollte es doch 0 als Priorität aufweisen, werden alle Prioritäten um den Wert 2 erhöht, dies verändert den Gewinner und die Strategie für das Spiel nicht. Formell wird der Registerreset also wie folgt definiert: für alle $j \in [1, \ldots, d]$ gilt $r_j = 1$

für $j < i$, $r_j = p$ für $j = i$ und $r_j = max(r_j, p)$ für $j > i$.

Die zweite Anpassung ist nötig, weil bei den Registerspielen davon ausgegangen wird, das in G die Prioritäten an den Knoten liegen, während sie bei den Separatoren an den Kanten liegen. Zur besseren Vergleichbarkeit werden nun in beiden Fällen die Prioritäten an die Kanten gelegt und zwar so, dass, wenn eine Priorität p an einem Knoten v liegt, sie an alle von v ausgehenden Kanten gelegt wird. Dies führt aber dazu, dass bei den Resetschritten im Registerspiel nicht mehr die zu nutzende Priorität am Knoten abgelesen werden kann, entsprechend muss sie mit im Zustand gespeichert werden. Deshalb wird in den Zuständen, die die Registerauswahl von Eva darstellen die Priorität mit gespeichert. Also werden aus Zuständen der Form $(v, \vec{r}, 0)$ welche mit $(v, L^G(w, v), \vec{r}, 0)$, wobei w der Zustand ist, der vom vorherigen R_{move} aus nach v führte. Dadurch muss nun auch der Startzustand angepasst werden. Ursprünglich war der Startzustand nämlich von der Form $(v_i, \vec{0}, 0)$, sprich Eva konnte den Registerreset beruhend auf der Priorität von v_i durchführen. Da aber jetzt in G die Prioritäten an den Kanten liegen, steht noch keine Priorität zur Verfügung, der erste Zustand von $R_E^k(G)$ muss also den Zug in G simulieren und ist entsprechend $(v_i, \vec{1}, 1)$. Zusätzlich werden wie oben beschrieben die Register am Anfang mit 1 belegt. Außerdem werden die Register nun absteigend statt aufsteigend sortiert. Wobei dies nur eine Vereinfachung für den Leser und keine Inhaltliche Anpassung ist.

Die weiteren Anpassungen betreffen den Separator $R_{n,d}$. Im originalen $R_{n,d}$ werden die Register von 1 an indiziert, da sie in der neuen Variante von $R_E^k(G)$ aber bei 0 anfangen, werden im Separator nun die Indizes um eins gesenkt also $< r_{\lfloor \lg(n) \rfloor}, ..., r_1, r_0 >$. Dies hat auf die eigentliche Funktionsweise keinen Einfluss, aber die Prioritäten aus $R_E^k(G)$ und $G \times R_{n,d}$ sind dadurch einfacher zu vergleichen.

Die wichtigere Änderung betrifft die Art, wie Update und Reset funktionieren. In dem alten Registerspiel wurden das Update der Register mit der Priorität p und das Resetten des Registers k separat voneinander durchgeführt und es wurde auch nur der Kleinste Register auf 0 gesetzt. Daher wurde dies auch für den Separator so übernommen. In der neuen Variante werden aber Update und Reset zusammen durchgeführt und alle kleineren Register werden auf 0 gesetzt. Außerdem war der Registerreset optional, entsprechend muss dies alles beim Separator abgeändert werden. Beim Update wurde der kleinste Register r_k gesucht, dessen Wert größer als die Priorität p war, und alle kleineren Register erhielten den Wert p. Da die Register sortiert waren, hatten auch alle größeren Register als r_k einen wert größer als p. Es lässt sich also auch so ausdrücken, dass alle Register das Maximum von r_i und p zugewiesen bekommen. Desweiteren wurde nur der kleinste Register auf 1 gesetzt und der Rest verschoben, nun werden alle Register, die kleiner als der gewählte sind auf 1 gesetzt. Der Reset für eine Priorität p funktioniert nun folgendermaßen, für jedes i gilt $r'_k = max(r_k, p)$ für $k > i$, $r'_k = p$ für $k = i$ und $r'_k = 1$ für $k < i$.

Durch diese Änderung muss nun aber auch die Bildung der Prioritäten angepasst werden, in der ursprünglichen Variante wurde nur der Registerinhalt dafür betrachtet. Da dies aber nach dem Update stattfand, enthielt der Register bereits das Maximum aus dem Registerinhalt und der Priorität, in der geänderten Variante enthält er nur den alter Wert.

Für die Prioritäten muss entsprechend gelten, $2r_i$ wenn $max(r_i, p)$ gerade ist und $2r_i + 1$ wenn $max(r_i, p)$ ungerade ist.

Die angepassten Graphen zu dem oben bereits genutzten Spiel G sind nun in den Abbildungen 4.4a, 4.4b und 4.4c zu sehen. Man sieht, dass bis auf Benennung die Graphen von $G \times R_{2,2}$ und $R_E^1(G)$ übereinstimmen.

Wichtig ist natürlich, dass die Anpassungen keinen der bisherigen Beweise verändern. Das G und $R_E^k(G)$ den gleichen Gewinner haben Lemma 2, gilt auch weiterhin. Denn sowohl für Adams als auch für Evas Strategien ist nur entscheidend, dass die Register über dem zum Reset ausgewählten immer mindestens die aktuelle Priorität enthalten, was bei den Anpassungen nicht verändert wird. Welche Werte die kleineren Register enthalten, ist nicht von Bedeutung, solange sie nur kleiner als die aktuelle Priorität sind, was auch nach der Anpassung gegeben ist. Selbiges gilt für den Beweis von Lemma 3, dass der Registerindex logarithmisch durch die Anzahl der Knoten aus G beschränkt ist. Auch hier ist nicht von Bedeutung welche Werte die Register, die kleiner sind als der Ausgewählte, zugewiesen bekommen.

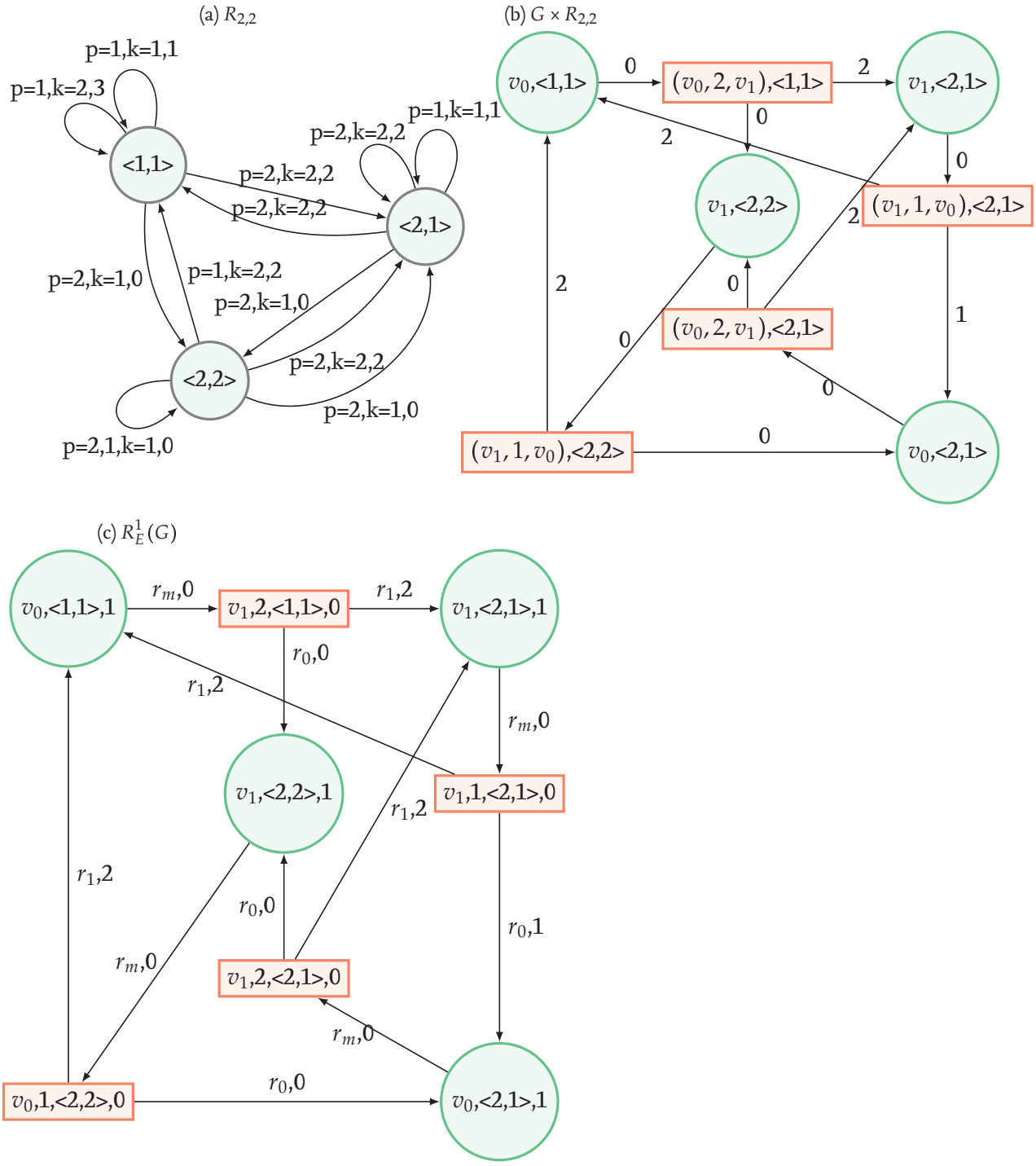


Abbildung 4.4: Angepasste Versionen von Registerspiel und Separator

5 Schranken und Übereinstimmungen

Der Sinn dahinter zu zeigen, dass verschiedene Konstruktionen übereinstimmen, besteht darin, dass auch verschieden Eigenschaften übereinstimmen. Wenn es für die eine Konstruktion einen Algorithmus gibt, der eine Eigenschaft nachweist, so ist dieser auch auf die andere Konstruktion anwendbar. Wenn es für die eine Konstruktion eine Schranke bezüglich einer Größe gibt, so gilt diese Schranke auch für die andere Konstruktion.

In 4.3 wurde abschließend gezeigt, dass Registerspiele und Separator-Automaten das gleiche Ergebnis liefern. In 4.1 wiederum wurde der Beweis, dass der Registerindex logarithmisch von der Knotenanzahl des Ausgangsspieles abhängt, gezeigt. Daraus folgt, dass das Ausgangsspiel in quasipolynomieller Zeit zu lösen ist. Entsprechend gilt diese Laufzeit auch für Separator-Automaten. Dies war auch der Gedanke hinter dem Paper von Czerwinski et al., zu zeigen das alle drei bisher bekannte Verfahren, von denen Registerspiele eines sind, Paritätsspiele in quasipolynomieller Zeit lösen, über Separatorautomaten auf universelle Bäume zurückzuführen sind und somit gleichwertig sind. Dort wurde gezeigt, dass universelle Bäume eine quasipolynomielle obere Schranke besitzen, was sich mit der Schranke für Registerspiele deckt. Gleichzeitig wurde auch eine quasipolynomielle untere Schranke nachgewiesen, wodurch folgt, dass es mit Registerspielen nicht möglich ist, eine Laufzeit, die besser als quasipolynomiel ist, zu erreichen.

Mit einem ähnlichen Gedanken haben Colcombt und Fijalkow verschiedene andere Äquivalenzen gezeigt, einmal [7] eine Erweiterung von Czerwinski et al., dass die Größe der kleinsten universellen Graphen, separierenden Automaten und universellen Bäume gleich ist, und zum anderen [5], dass good-for-n-Games Automaten, (n,d)-separierende Automaten und (n,d)-universelle Graphen zusammenhängen.

Bäume, Graphen und separierende Automaten Befassen wir uns erstmal mit der ersten Äquivalenz. Also das folgende Größen gleich sind:

1. die des kleinsten universellen Baumes
2. die des kleinsten separierenden Automaten und
3. die des kleinsten universellen Graphen.

In diesem Fall werden die Kanten des Graphen wie folgt aufgefasst. Es existieren d viele Kanten E_j mit $j \in [0, d-1]$, wobei mit $(v, v') \in E_j$ gemeint ist, dass die Kante zwischen v und v' die Priorität j besitzt oder formal $(v, j, v') \in E$.

In Bäumen gilt, dass jeder Knoten bis auf die Wurzel einen eindeutigen Vorgänger besitzt. Im Folgenden werden von den Blättern an die verschiedenen Ebenen des Baumes

von 0 bis $d-1$ durchnummeriert. Dabei stehen gerade Nummern für die Knoten und ungerade für die Kanten. Die Kombination aus einem Blatt u und einer Nummer ist nun eindeutig einem Knoten oder einer Kante im Baum zugewiesen. Zum Beispiel liefert $(u,3)$ die Kante (v,v') für die gilt, dass v' der Vorgänger von u und v der von v' ist. Diese nutzen wir nun, um einen Baum T in einen Graphen $G(T)$ umzuwandeln. Dabei sind die Knoten die Blätter von T. Die Kanten werden wie folgt gebildet, es existiert die Kante $(v,j,v') \in E$ wenn für gerade j die Vorgängerknoten u von v und u' von v' auf Ebene j gilt $u \leq u'$ oder für ungerade j die Vorgängerkanten w und w' auf Ebene j gilt $w < w'$. Hierbei beziehen sich $\leq$ und $<$ auf die Ordnung im Baum.

(Siehe [7] Kap. 3)Um nun die obigen Äquivalenzen zu beweisen wird zunächst eine Hilfsdefinition gebraucht, die baumähnlichen Graphen.

Definition 9. *Ein Graph ist baumähnlich, wenn er die folgenden 4 Bedingungen erfüllt:*

1. wenn $(v,i,v') \in E$ *und* $(v',j,v'') \in E$ *gelten, dann gilt* $(v,max(i,j),v'') \in E$*, sprich die Kanten sind transitiv*

2. für i gerade gilt, dass E_i *total ist,* $(v,i,v') \in E$ *oder* $(v',i,v) \in E$

3. für i gerade ist E_i *reflexiv,* $(v,i,v) \in E$

4. für i ungerade gilt, dass $(v,i,v') \in E$ *genau dann wenn* $(v,i-1,v') \in E$ *und* $(v',i-1,v) \notin E$

Wenn nun G ein baumähnlicher Graph ist, kann man daraus einen Baum $T(G)$ konstruieren, dessen Blätter die Knoten von G sind und die Knoten auf den Ebenen mit geraden i werden durch die Äquivalenzklassen von E_i gebildet. Aus dieser Definition folgen nun folgende Aussagen. Für Bäume t, T gilt, dass t genau dann in T enthalten ist, wenn ein Homomorphismus $\phi : G(t) \to G(T)$ existiert. Und wenn t ein Baum ist, dann ist $G(t)$ baumähnlich und $T(G(t)) = t$. Man kann also Bäume immer als baumähnliche Graphen auffassen.

Die Beweisidee für die Äquivalenzen sieht nun wie folgt aus. Ein (n,d)-universeller Baum wird zu einem (n,d)-separierenden Automaten umgebaut, ein (n,d)-universeller Graph zu einem (n,d)-universellen Baum und ein (n,d)-separierender Automat zu einem (n,d)-universellen Graphen, wobei die Konstruktionen die gleichen Größen aufweisen. Dafür wird mehrfach als Hilfsergebnis genutzt, dass ein maximaler Graph, der die Paritätsbedingung erfüllt, baumähnlich ist. Daraus folgt fast direkt, dass ein maximaler (n,d)-universeller Graph ein (n,d)-universeller Baum ist. Im Fall vom Separator zum universellen Graphen wird die *Saturation*-Technik benutzt. Dabei werden zu einem (n,d)-Graphen G der die Paritätsbedingung erfüllt solange Kanten hinzugefügt, bis die Bedingung nicht mehr erfüllt werden könnte. Der so entstandene Graph t ist die *Saturation* von G. Wenn nun ein Graph G die Paritätsbedingung erfüllt, was bei separierenden Automaten der Fall ist, dann ist jede *Saturation* davon universell. Bei der Konstruktion von universellen Bäumen zu Automaten wird der Automat auf eine bestimmte Weise konstruiert und anschließend nachgewiesen, dass es sich um einen Separator handelt.

Separierende und good-for-n-Games Automaten Nun gibt es noch die andere Gruppe von Äquivalenzen, dass die Anzahl an Zuständen in strongly good-for-n-Games, an Zuständen

in strongly (n,d)-separierenden Automaten und an Knoten in (n,d)-universellen Graphen gleich sind. Die letzten beiden sind bereits in der vorherigen Aussage enthalten, sind hier aber etwas anders aufgebaut, da sie hier allgemein für alle Gewinnbedingungen gezeigt werden, für die Eva eine positionale Gewinnstrategie besitzt. Außerdem gilt es für jede beliebige Größe der Prioritäten d. Deshalb wird dieser Teil nicht nochmal explizit erläutert. In unserem Fall beschäftigen wir uns nur mit der Paritätsbedingung. Der erste Teil ist nochmal unterteilt und zwar gilt, dass die Anzahl an Zuständen

1. eines *strongly (n,d)-separierenden deterministischen* Automaten,
2. eines *strongly good for n-games* Automaten und
3. eines *strongly (n,d)-separierenden* Automaten

gleich sind.

(Siehe [7] 3.2 Lemma 7) Das dabei 3. $\Rightarrow$ 1. gilt, ist sofort offensichtlich. Entscheidend ist der Beweis, dass 1. $\Rightarrow$ 2. $\Rightarrow$ 3. gilt. Für den ersten Teil bedeutet das, wenn ein *strongly (n,d)-separierenden deterministischen* Automaten A gegeben ist, ist A auch *strongly good-for-n-Games*. Also ist zu zeigen, dass Eva $G \times A$ genau dann gewinnt, wenn sie G gewinnt. Das aus einem Sieg in $G \times A$ ein Sieg in G folgt, ist seit langem allgemein für Automaten bewiesen. Die andere Richtung ist aufwendiger zu beweisen. Wenn Eva das Spiel G gewinnt, dann existiert ein Graph S_E, der die positionale Gewinnstrategie von Eva darstellt. Dabei gilt $L(S_E) \subseteq L(A)$, woraus folgt, dass für jede Zugfolge ϕ aus S_E ein Durchlauf in A über den labels von ϕ existiert. Nun wird eine neue Strategie für Eva in $G \times A$ konstruiert. In den Spielpositionen spielt Eva nach S_E, in den Automatenpositionen gibt es aufgrund obigen Ergebnisses und dem Determinismus nur einen möglichen Zug, diesen wählt Eva. Dies ist eine Gewinnstrategie für $G \times A$ und somit ist A ein *strongly good-for-n-Games* Automat.

Für den zweiten Teil bedeutet es, dass wenn ein *strongly good-for-n-games* Automat A gegeben ist, ist A auch ein *strongly (n,d)-separierenden* Automaten. Wenn nun A ein *strongly good-for-n-games* Automat für Parität ist, dann liegt $L(A)$ in Parität. Nun betrachtet man ein Wort $u \in P_n$, wobei P_n die Vereinigung aller Sprachen ist, die von safety Automaten mit n Zuständen akzeptiert werden, die Teilsprachen von Parität akzeptieren. Dann existiert ein safety Automat B mit maximal n Zuständen sodass $u \in L(B)$ und $L(B)$ liegt in Parität. Dieser Automat kann als Paritätsspiel G aufgefasst werden, in dem alles Positionen zu Adam gehören. Da $L(B)$ in Parität liegt, gewinnt Eva G. Da A ein *strongly good for n-games* Automat ist hat Eva somit auch eine Gewinnstrategie S_E für $G \times A$. Wenn Adam nun in $G \times A$ nach den Buchstaben von u spielt, erzeugt S_E nun einen akzeptierenden Durchlauf für u in A. Damit ist $u \in L(A)$ und somit A ein *strongly (n,d)-separierenden* Automaten.

Wie in 4.2 bereits gesagt, haben Czerwinski et al. [5, Kap. 2.4] nachgewiesen, dass ein universellen Baum mindestens eine quasipolynomielle Anzahl an Blättern hat. Genauer gesagt hat jeder (n,d)-universeller Baum mindestens $n^{\lg((d/2)/\lg(n))-1}$ viele Blätter, wenn $d \leq n$ gilt. Hierbei ist zu beachten, dass bei Czerwinski ein (n,d)-Baum mit Höhe d statt $d/2$ definiert wird, weswegen sich die Formel leicht unterscheidet und d hat mindestens den Wert 2. Zur Vereinfachung wird $h = d/2$ gesetzt und in den Formeln mit h gearbeitet, die Abschätzung lautet dann $n^{\lg(h/\lg(n))-1}$, wenn $2h \leq n$ gilt. Die Abschätzung erfolgt in meh-

reren Schritten und nutzt mehrere Hilfsformeln. Im ersten Schritt wird die Hilfsfunktion $g(n,h) = \sum_{\delta=1}^{n} g(\lfloor l/\delta \rfloor, h-1)$ mit $g(l,1) = l$ und $g(1,h) = 1$ genutzt. Diese Funktion stellt eine untere Schranke für die Anzahl der Blätter im Baum dar und dies wird per Induktion bewiesen. Im nächsten Schritt wird gezeigt, dass die Abschätzung $g(n,h) \leq \binom{\lfloor \lg n \rfloor + h - 1}{\lfloor \lg n \rfloor}$ gilt. Dafür werden wieder zwei Hilfsfunktionen genutzt, zuerst $G(p,h) = g(2^p,h)$ für $p \geq 0$ und $h \geq 1$. Man sieht sofort, dass $g(n,h) \leq G(n,h)$ gilt. Für die Funktion G gilt nun $G(p,h) \geq \sum_{k=0}^{p} G(p-k,h-1), G(P,1) \geq 1$ und $G(0,h) = 1$. Anschließend wird eine andere Funktion als untere Schranke für G konstruiert, nämlich H mit $H(p,h) = H(p,h-1) + H(p-1,h)$, $H(p,1) = 1$ und $H(0,h) = 1$. Dabei ist H(p,h) eine andere Schreibweise für den Binomialkoeffizienten $\binom{p+h-p}{p}$, was anhand der Pascalschen Identität leicht zu sehen ist. Setzt man nun $p = \lfloor \lg n \rfloor$ liefern dies Hilfsfunktion $g(n,h) \geq \binom{\lfloor \lg n \rfloor + h - 1}{\lfloor \lg n \rfloor}$. Im letzten Schritt ist nur noch zu zeigen, dass $\binom{\lfloor \lg n \rfloor + h - 1}{\lfloor \lg n \rfloor} \geq n^{\lg(h/\lg(n))-1}$ gilt. Dafür wird genutzt, dass die Ungleichung $(k/l)^l \leq \binom{l}{k}$ gilt. Diese wird auf $\binom{\lfloor \lg n \rfloor + h - 1}{\lfloor \lg n \rfloor}$ angepasst, anschließend wird auf beiden Seiten der Logarithmus lg angewandt und abschließend durch Anwendung von $n \geq 2$ und $2h \leq n$ zur Abschätzung die Ungleichung gezeigt. Insgesamt ist somit gezeigt, dass ein (n,d)-universeller Baum mindestens $n^{\lg((d/2)/\lg(n))-1}$ viele Blätter besitzt.

Da bereits gezeigt wurde, dass universelle Bäume und separierende Automaten die gleiche untere Schranke besitzen, gilt die nun gezeigte Schranke auch für einen (n,d)-separierenden Automaten. Da universelle Bäume die Grundlage der drei bisher bekannten Algorithmen bilden, die Paritätsspiele in quasipolynomieller Zeit lösen, gilt diese untere Schranke für alle drei. Entsprechend können mit diesen Verfahren keine allgemein schnelleren Algorithmen gewonnen werden.

Lethinen hat außerdem gezeigt [6], dass man zu einen Registerspiel auch einen Registerautomaten konstruieren kann, der die Sprache des Registerspiels akzeptiert. Nach den oben gezeigten Äquivalenzen ist dieser Automat *good-for-small-Games*. Der Registerindex ist logarithmisch zur Anzahl an Positionen im ursprünglichen Spiel, entsprechend kann umgekehrt ein Automat mit Registerindex k nur Paritätsspiele mit höchstens 2^{k-1} Positionen erkennen. Es gibt aber immer Spiel, die größer sind, für die Eva eine Gewinnstrategie besitzt. Entsprechen sind Registerautomaten und daraus folgend auch separierende Automaten nicht *good-for-Games*. Wie schon gesagt, ist es mit Registerspielen nicht möglich, einen Algorithmus zu entwickeln, der schneller als quasipolynomiell ist, um Paritätsspiele zu lösen. Dies gilt auch für alle anderen Verfahren, die universelle Bäume oder *good-for-small-Games* Automaten nutzen.

6 Fazit und Ausblick

Die Registerspiele und der Separatoransatz wurden in dieser Arbeit so aneinander angepasst, dass eindeutig zu erkennen ist, dass sie das gleiche leisten. Auch wurden damit untere und obere Schranken für die Laufzeit nachgewiesen. Außerdem wurden verschiedene Ergebnisse in Bezug auf *good-for-small-Games* Automaten zusammengetragen und gezeigt, dass Registerspiele eine Variante davon sind. Zusätzlich ist damit gezeigt, dass die gezeigten untere Schranke entsprechend für alle Verfahren, die *good-for-small-Games* Automaten oder universelle Bäume nutzen, gilt.

Es wurde zwar eine untere Schranke für drei neue Verfahren gezeigt, allerdings ist noch unbekannt, ob es nicht andere Verfahren gibt, mit denen Paritätsspiele schneller gelöst werden können oder ob diese Schranke allgemein für Paritätsspiele gilt. Außerdem kann noch genauer untersucht werden, ob Spezialfälle existieren, bei denen Methoden wie die Registerspiele eine Verbesserung gegenüber anderen Methoden ergeben. In der Anwendung ist es häufig so, dass auch wenn verschiedene Algorithmen allgemein die gleiche Laufzeit haben, es für ein bestimmtes Problem sinnvoll sein kann, einen von ihnen zu bevorzugen. So gibt es bereits Beispiele, in denen für andere Algorithmen schwer zu lösende Spiele einen kleinen Registerindex aufweisen und somit über Registerspiele verhältnismäßig schnell zu lösen sind. Auf der anderen Seite haben Registerspiele und auch die anderen Verfahren wie Separatoren den Nachteil, das sie die Anzahl der benötigten Zustände quasipolynomiell ansteigt. Die bessere Laufzeit wird also mit höheren Speicherbedarf erkauft. Diese Problematik zu untersuchen, wäre sicher ein interessantes Gebiet, Lehtinen hat dazu bereits Ansätze aufgezeigt. Zu Beispiel wäre zu untersuchen, ob der Registerindex nur in entsprechen konstruierten Fällen beliebig groß werden kann und in den für die Anwendung interessanten Fällen klein bleibt, oder ob es auch in der Anwendung Fälle gibt, die einen hohen Registerindex aufweisen, und wie man diese Fälle erkennen kann.

Ein anderer interessanter Punkt wäre, wie es sich mit anderen Gewinnbedingungen für Spiele verhält. Es gibt zum Beispiel die Büchi und CoBüchi Bedingungen. Diese sind eine Variante der Paritätsbedingung, bei der das Intervall der Prioritäten auf [1,2] bzw. [0,1] eingeschränkt wird. Es wäre zu untersuchen, inwiefern sich Registerspiele darauf übertragen lassen und ob das Vorteile beim Lösen der Spiele bringt. Da der Vorteil von Registerspielen darin liegt, dass sie die Größe der Prioritäten verkleinern, diese bei Büchi und CoBüchi aber bereits klein sind, wäre an der Stelle auf den ersten Blick keine Verbesserung zu sehen, zumal es für diese bereits polynomielle Algorithmen gibt. Allerdings wäre zu überprüfen, ob sich durch den *good-for-small-Games* Ansatz nicht auch für andere Bedingungen Konstruktionen finden lassen, die bessere Algorithmen ermöglichen.

Desweiteren beziehen sich Registerspiele auf nichtdeterministische Automaten, wie genau sie sich auf alternierende Automaten übertragen lassen wäre noch zu untersuchen. Für das Konzept von *good-for-Games* Automaten hat Lehtinen [10] bereits Untersuchungen angestellt. Dabei hat sie ermittelt, dass für manche Bedingungen alternierende Automaten genauso ausdrucksstark wie deterministische Automaten sind und für andere Bedingungen alternierende Automaten ausdrucksstärker sind. Allerdings konnte sie es nicht für alle Fälle abschließend beantworten.

Literaturverzeichnis

[1] Wieslaw Zielonka. Infinite games on finitely coloured graphs with applications to automata on infinite trees. *Theor. Comput. Sci.*, 200(1-2):135–183, 1998.

[2] Cristian S. Calude, Sanjay Jain, Bakhadyr Khoussainov, Wei Li, and Frank Stephan. Deciding parity games in quasipolynomial time. In *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2017, Montreal, QC, Canada, June 19-23, 2017*, pages 252–263, 2017.

[3] Marcin Jurdzinski and Ranko Lazic. Succinct progress measures for solving parity games. In *32nd Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2017, Reykjavik, Iceland, June 20-23, 2017*, pages 1–9, 2017.

[4] Karoliina Lehtinen. A modal μ perspective on solving parity games in quasi-polynomial time. In *Proceedings of the 33rd Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2018, Oxford, UK, July 09-12, 2018*, pages 639–648, 2018.

[5] Wojciech Czerwinski, Laure Daviaud, Nathanaël Fijalkow, Marcin Jurdzinski, Ranko Lazic, and Pawel Parys. Universal trees grow inside separating automata: Quasi-polynomial lower bounds for parity games. *CoRR*, abs/1807.10546, 2018.

[6] Udi Boker and Karoliina Lehtinen. Register games. *CoRR*, abs/1902.10654, 2019.

[7] Thomas Colcombet and Nathanaël Fijalkow. Parity games and universal graphs. *CoRR*, abs/1810.05106, 2018.

[8] Thomas Colcombet and Nathanaël Fijalkow. Universal graphs and good for games automata: New tools for infinite duration games. In *Foundations of Software Science and Computation Structures - 22nd International Conference, FOSSACS 2019, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2019, Prague, Czech Republic, April 6-11, 2019, Proceedings*, pages 1–26, 2019.

[9] E. Allen Emerson and Charanjit S. Jutla. Tree automata, mu-calculus and determinacy (extended abstract). In *32nd Annual Symposium on Foundations of Computer Science, San Juan, Puerto Rico, 1-4 October 1991*, pages 368–377, 1991.

[10] Udi Boker and Karoliina Lehtinen. Good for games automata: From nondeterminism to alternation. In *30th International Conference on Concurrency Theory, CONCUR 2019, August 27-30, 2019, Amsterdam, the Netherlands.*, pages 19:1–19:16, 2019.