Gottfried Wilhelm Leibniz Universität Hannover
Institut für Theoretische Informatik

Komplexität von Spielen

Bachelorarbeit

Leonard Kerner
Matrikelnr. 10014314

Hannover, den 26. Juni 2023

Erstprüfer: PD Dr. rer. nat. habil. Arne Meier
Zweitprüfer: Prof. Dr. rer. nat. Heribert Vollmer
Betreuerin: Vivian Holzapfel, M. Sc.

Erklärung der Selbstständigkeit

Hiermit versichere ich, die vorliegende Bachelorarbeit/Masterarbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet zu haben. Die Arbeit hat in gleicher oder ähnlicher Form noch keinem anderen Prüfungsamt vorgelegen.

Hannover, den 26. Juni 2023

_____

Leonard Kerner

Inhaltsverzeichnis

1 Einleitung

In der folgenden Bachelorarbeit wird die Komplexität des Findens von Lösungswegen der Spiele „Tic-Tac-Toe“ „Super Mario Bros.“ untersucht. Im Anschluss werden die beiden Spiele miteinander verglichen, um Gemeinsamkeiten und Unterschiede der Spiele herauszustellen. Das Ziel der Arbeit besteht darin, herauszuarbeiten, wie sehr sich die Lösungsfindung der beiden Spiele ähnelt und welche Gemeinsamkeiten und Unterschiede die Beweisführungen aufweisen. Um einen angemessenen Rahmen für diese Bachelorarbeit zu bewahren, werden die technischen Beweise nicht eigenständig erbracht, sondern anhand ihrer grundlegenden Ideen beschrieben. Außerdem wird nicht auf verschiedene Möglichkeiten für die Generalisierung der Spiele eingegangen, da es insbesondere bei „Super Mario Bros.“ verschiedene Ansätze gibt.

In der Bachelorarbeit werden zunächst die wichtigsten Grundlagen erläutert, um ein einheitliches Verständnis der Begriffe vorauszusetzen. Im Anschluss werden die beiden verwendeten Spiele „Tic-Tac-Toe“ und „Super Mario Bros.“ für einen Einblick in den Spielablauf beschrieben. Zudem wird auf das Spiel „Geographie“ eingegangen, da für die Einordnung von „Tic-Tac-Toe“ wichtig ist. Daraufhin werden die jeweiligen Nachweise für die PSPACE-Vollständigkeit der Lösungssuche innerhalb der Spiele vorgestellt. Als nächster Punkt werden die Spiele sowie die Beweise der PSPACE-Vollständigkeit ihrer Lösungssuche miteinander verglichen, um Gemeinsamkeiten und Unterschiede herauszustellen. Zum Schluss folgt eine Zusammenfassung der wichtigsten Erkenntnisse sowie ein Ausblick über mögliche weiterführende Forschungsthemen.

2 Grundlagen

Im folgenden Kapitel werden die verwendeten Grundlagen und Definitionen vorgestellt, um ein einheitliches Verständnis der jeweiligen Begriffe zu setzen. Diese Grundlagen dienen als theoretische Basis für die Bachelorarbeit.

2.1 Turingmaschinen

Der nachfolgende Abschnitt bezieht sich auf das Buch „*Komplexität von Algorithmen*“ von Meier und Vollmer und definiert deterministische und nichtdeterministische Turingmaschinen [MV15, S. 15 ff.].

Eine (deterministische) Turingmaschine ((D)TM) ist ein mathematisches Modell zum Betrachten algorithmischer Probleme. Anschaulich besteht die Turingmaschine aus einem in Zellen eingeteilten, beschreibbaren Band sowie einem Lese-Schreibkopf, welcher sich nach links oder rechts bewegen. Die Zellen auf dem Band können von dem Lese-Schreibkopf eingelesen und beschrieben werden. Formal stellt eine Turingmaschine einen Algorithmus dar und wird nach Meier und Vollmer wie folgt definiert [MV15, S.15, 16]:

Definition 1 ((Deterministische) Turingmaschine) *Eine* (deterministische) Turingmaschine *ist ein Tupel* $M = (Z, \Gamma, \delta, z_0, A, V)$ *mit den folgenden Bestandteilen:*

Z : *Die Menge der Zustände, welche die Turingmaschine erreichen kann*

Γ : *Die Menge der zulässigen Zeichen auf dem Band der Turingmaschine (Bandalphabet)*

δ : *Die Übergangsfunktion* $\delta : Z \times \Gamma \to Z \times \Gamma \times \{L, N, R\}$

z_0 : *Der Startzustand der Turingmaschine*

A : *Die Menge der akzeptierenden Zustände*

V : *Die Menge der verwerfenden Zustände*

Eine *nichtdeterministische Turingmaschine* (NTM) hat bei der Übergangsfunktion δ nicht immer nur einen einzigen Folgezustand. Dies lässt sich durch eine veränderte Übergangsfunktion ausdrücken:

$$\delta_{NTM} : Z \times \Gamma \to \mathcal{P}(Z \times \Gamma \times \{L, N, R\})$$

Anschaulich führt die Turingmaschine in diesem Fall alle möglichen Schritte gleichzeitig aus.

2.2 Quantifizierte Boolesche Formeln

Da für die späteren Beweise jeweils eine Reduktion mittels quantifizierter Boolescher Formeln (QBF) verwendet wird, sollen hier die grundlegenden Aspekte erläutert werden. Dieser Abschnitt bezieht sich auf „*Aussagenlogik - Deduktion und Algorithmen*“ von Kleine Büning und Lettman.

Eine *aussagenlogische (oder Boolesche) Formel* besteht aus einer oder mehreren Variablen oder Atomen (zum Beispiel $A_0, A_1, \dots$), der Negation „$\neg$“, der Disjunktion „$\wedge$“ und der Konjunktion „$\vee$“ [KL94, Def. 1.1.1, S. 2]. Eine quantifizierte Boolesche Formel wird nach Kleine Büning und Lettman [KL94, Def. 7.1.1, S. 362] wie folgt definiert:

Definition 2 (Quantifizierte Boolesche Formel)

1. *Jede aussagenlogische Formel ist eine quantifizierte Boolesche Formel*
2. *Sei Φ eine quantifizierte Boolesche Formel und seien x bzw. y aussagenlogische Variablen, dann sind auch $\exists y\Phi$ und $\forall x\Phi$ quantifizierte Boolesche Formeln.*
3. *Nur gemäß 1. und 2. gebildete Formeln sind quantifizierte Boolesche Formeln.*

Mit der Abkürzung TQBF (aus dem Englischen übersetzt für „True Quantified Boolean Formula“) sei das folgende Problem gemeint:

Definition 3 (Erfüllbarkeitsproblem Quantifizierter Boolescher Formeln) *Sei Φ eine Quantifizierte Boolesche Formel. Ist Φ erfüllbar?*

2.3 Komplexitätsklassen

Der nachfolgende Abschnitt bezieht sich ebenfalls auf Meier und Vollmer und führt verschiedene Komplexitätsklassen ein [MV15, S. 20 ff.].

Probleme können mithilfe einer Turingmaschine in verschiedene Komplexitätsklassen eingeteilt werden. Dabei spielt die Anzahl der benötigten Schritte (Zeit) oder die Anzahl der verwendeten Speicherzellen (Platz) eine wichtige Rolle. Es wird für das jeweilige Problem immer die am schwersten zu lösende Version (worst case) betrachtet. Anhand der benötigten Zeit oder des benötigten Platzes kann dann ein Problem in Abhängigkeit von der Eingabegröße des Problems in eine Komplexitätsklasse eingeteilt werden. Die für diese Arbeit verwendeten Klassen werden nun kurz vorgestellt.

Für die Formalen Definitionen werden zunächst die Klassen DTIME [AB09, Def.1.19, S. 27] und (N)SPACE [AB09, Def. 4.1, S. 76] benötigt:

Definition 4 (Die Klasse DTIME) *Sei* $T : \mathbb{N} \to \mathbb{N}$ *eine beliebige Funktion. Dann ist* $DTIME(T(n))$ *die Menge aller booleschen Funktionen, welche sich für ein konstantes* $c > 0$ *in* $c \cdot T(n)$ *Zeiteinheiten berechnen lassen.*

Definition 5 (Die Klassen SPACE und NSPACE) *Sei* $S : \mathbb{N} \to \mathbb{N}$ *und* $L \subseteq \{0,1\}^*$. *Es gilt* $L \in SPACE\ (s(n))$ *(beziehungsweise* $L \in NSPACE\ (s(n))$*), wenn es eine Konstante c und eine TM (bezieungsweise NTM) M gibt, welche L über jede Eingabe* $x \in \{0,1\}$ *so entscheidet, dass die Anzahl der Speicherzellen, welche während der Berechnung durch M nicht leer sind, höchstens* $c \cdot s(|x|)$ *beträgt.*

In der Komplexitätsklasse P befinden sich alle Probleme, welche von einer deterministischen Turingmaschine in polynomieller Zeit entschieden werden können. Demnach werden bei einem Problem mit Eingabelänge n und einer Konstanten $c \geq 0$ höchstens n^c Arbeitsschritte benötigt, um das Problem zu lösen. Formal wird sie von Arora und Barak für ein beliebiges, aber festes c wie folgt definiert [AB09, Def. 1.20, S. 27]:

Definition 6 (Die Klasse P)

$$P = \cup_{c \geq 1} DTIME(n^c)$$

In der Komplexitätsklasse NP befinden sich alle Probleme, welche von einer nichtdeterministischen Turingmaschine in polynomieller Zeit entschieden werden können. Demnach gibt es bei einem Problem mit Eingabelänge n und einer Konstanten $c \geq 0$ eine Lösung, welche höchstens n^c Arbeitsschritte benötigt. Formal lässt sie sich folgendermaßen definieren [AB09, Def. 2.1, S. 40]:

Definition 7 (Die Klasse NP) *Eine Sprache* $L \subseteq \{0,1\}^*$ *liegt in NP, wenn es ein Polynom* $p : \mathbb{N} \to \mathbb{N}$ *und eine Polynomialzeit Turingmaschine M gibt, sodass für alle* $x \in \{0,1\}^*$ *gilt:*

$$x \in L \Leftrightarrow \exists u \in \{0,1\}^{p(|x|)},\ mit\ M(x,u) = 1$$

Erfüllen $x \in L$ *und* $u \in \{0,1\}^{p(|x|)}$ *die Bedingung* $M(x,u) = 1$*, dann wird u das Zertifikat für x genannt.*

Analog zu den Klassen P und NP befinden sich in den Komplexitätsklassen PSPACE und NPSPACE alle Probleme, welche von einer deterministischen (PSPACE) oder nichtdeterministischen (NPSPACE) Turingmaschine mit polynomiellen Platz gelöst werden können. Demnach werden bei einem Problem mit Eingabelänge n und $c \geq 0$ höchstens n^c Speicherzellen benötigt, um das Problem zu entscheiden [AB09, Def.4.5, S. 78].

Definition 8 (Die Klassen PSPACE und NPSPACE)

$$PSPACE = \cup_{c \geq 0} SPACE(n^c)$$
$$NPSPACE = \cup_{c \geq 0} NSPACE(n^c)$$

Für die beiden letzten Begriffe wird zunächst die Reduzierbarkeit benötigt. Meier und Vollmer definieren diese wie folgt [MV15, Def.14, S.79f.]

Definition 9 (Reduzierbarkeit) *Seien Σ und Δ zwei Alphabete, sodass $A \subseteq \Sigma^*, b \subseteq \Delta^*$ zwei Sprachen sind. A heißt auf B* (in Polynomialzeit) reduzierbar*, falls es eine Funktion $f : \Sigma^* \to \Delta^*$ mit den folgenden Eigenschaften gibt:*

- *Es gibt eine (D)TM M, die in Polynomialzeit arbeitet und bei Eingabe eines Wortes $x \in \Sigma^*$ das Wort $f(x) \in \Delta^*$ produziert. M hat dazu ein spezielles Ausgabeband, auf dem sich am Ende der Rechnung das Ausgabewort befindet. Mit anderen Worten: Die Funktion f ist in Polynomialzeit berechenbar.*
- *Für alle $x \in \Sigma^*$ gilt: $x \in A$ gdw. $f(x) \in B$*

Die Funktion f wird auch (Polynomialzeit-) Reduktion *von A auf B, in Zeichen $A \leq_m^P B$, genannt.*

Mit der Reduzierbarkeit lassen sich nun auch schwere und vollständige Probleme definieren. Diese werden nach Meier und Vollmer wie folgt definiert [MV15, Def. 15, S. 82]:

Definition 10 (NP-schwer, NP-vollständig)

(i) Eine Sprache B ist NP-schwer*, falls für alle $A \in NP$ gilt: $A \leq_m^P B$.*

(ii) Eine Sprache B ist NP-vollständig*, falls B NP-schwer ist und $B \in NP$.*

2.4 Gelöste Spiele

Dieser Abschnitt bezieht sich auf „Searching for Solutions in Games and Artificial Intelligence“ von Allis [All94].

Spiele, bei denen der Ausgang bei perfekter Spielweise aller Parteien bekannt ist, werden als „gelöst“ bezeichnet. Diese Spiele sind für Komplexitätsbetrachtungen gut geeignet, da es nur wenige oder keine Ungewissheiten im Spielverlauf gibt.

Eine genauere Definition und Abstufung wurde von Allis vorgestellt [All94, S. 7f.]:

Definition 11 (Gelöstes Spiel)

- *Ein Spiel heißt ultra-schwach gelöst, wenn von den Startpositionen aus der theoretische Ausgang des Spiels bekannt ist.*
- *Ein Spiel heißt schwach gelöst, wenn von den Startpositionen eine Strategie bekannt ist, welche zum theoretischen Ausgang des Spieles führt.*
- *Ein Spiel heißt stark gelöst, wenn zu jeder erlaubten Spielposition eine Strategie bekannt ist, welche zum theoretischen Ausgang des Spieles führt.*

3 Spiel 1: „Tic-Tac-Toe“

Im folgenden Kapitel wird das Spiel „Tic-Tac-Toe“ kurz beschrieben und generalisiert. Anschließend wird das Lösungsproblem für „Tic-Tac-Toe“ formuliert, sowie ein Beweis für die PSPACE-Vollständigkeit dieses Problems vorgestellt.

„Tic-Tac-Toe“ (TTT) ist ein Brettspiel für zwei Personen. Es wird üblicherweise auf einem 3×3 Feld gespielt. Abwechselnd markieren die Spielenden ein Feld mit ihrem Symbol (häufig „X “ und „O “). Die erste Person, die drei ihrer Symbole in einer horizontalen, vertikalen, oder diagonalen Reihe anordnen kann, gewinnt die Partie. Dabei ist es nicht möglich, eine bereits gesetzte Markierung zu entfernen oder zu ändern. Eine Partie ist somit immer nach spätestens 9 Zügen vorbei.

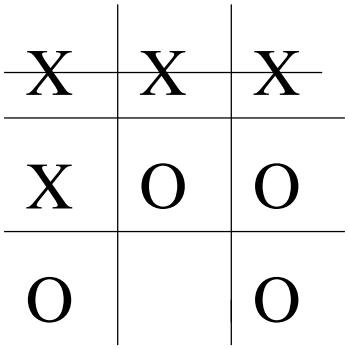

Abbildung 3.1: Eine beendete Partie „Tic-Tac-Toe“. Diese Partie hat „X“ gewonnen.

„Tic-Tac-Toe“ gehört zu den m, n, k Spielen, bei denen auf einem $m \times n$ Feld gespielt wird, mit dem Ziel, eine Reihe der Länge k mit den eigenen Markierungen zu bilden. Es ist somit das $3, 3, 3$ Spiel. Ein weiteres Beispiel für ein m, n, k Spiel ist „Gobang“, das $19, 19, 5$ Spiel. „Tic-Tac-Toe“ ist auch das kleinste m, n, k Spiel, welches beiden Parteien gewonnen werden kann. Es gehört außerdem zu den stark gelösten Spielen. Bei perfekter Spielweise beider Parteien ist das Ergebnis immer ein Unentschieden.

Als Generalisierung für „Tic-Tac-Toe“ wird das $n, n, 3$ Spiel betrachtet.

3.1 „Tic-Tac-Toe“ zu gewinnen ist PSPACE-schwer

Folgender Satz wird in diesem Abschnitt gezeigt:

Satz 1 *Eine Zugfolge zu finden, mit welcher „X“ in generalsiertem Tic-Tac-Toe gewinnen kann, ist PSPACE-schwer.*

Das Problem aus Satz 1 wird von nun an als „GTTT“ bezeichnet. Der Beweis von Satz 1 erfolgt in zwei Schritten. Zunächst wird gezeigt, dass TQBF auf eine Generalisierung des Spiels „Geographie“ reduzierbar ist. Anschließend wird die Reduktion von generalisiertem „Geographie“ auf GTTT gezeigt. Durch die Transitivität der Reduzierbarkeit ist TQBF damit reduzierbar auf GTTT und somit ist GTTT PSPACE-schwer.

3.1.1 Generalisiertes „Geographie“ zu gewinnen ist PSPACE-schwer

In diesem Teil wird das Spiel „Geographie“ vorgestellt, generalisiert. Außerdem wird das Lösungsproblem für diese Generalisierung formuliert. Anschließend wird ein Beweis für die Reduzierbarkeit von TQBF auf dieses Problem vorgestellt und anhand eines Beispiels verdeutlicht.

„Geographie“ ist ein Spiel für zwei Personen, bei dem abwechselnd Städte aufgezählt werden und jede Stadt nur einmal genannt werden darf. Person A nennt eine Stadt, zum Beispiel „Hannover“. Person B antwortet mit einer anderen Stadt, welche mit dem letzten Buchstaben der Stadt von Person A beginnt, wie zum Beispiel „Rostock“. Das Spiel endet, sobald eine Person keine Stadt mehr nennen kann. Diese Person hat das Spiel verloren.

Eine beliebige Partie von „Geographie“ lässt sich als gerichteter Graph darstellen, bei der jede Stadt einen Knoten darstellt. Von jedem Knoten aus geht eine Kante zu jeder zulässigen Stadt, welche den Regeln des Spiels genügt.

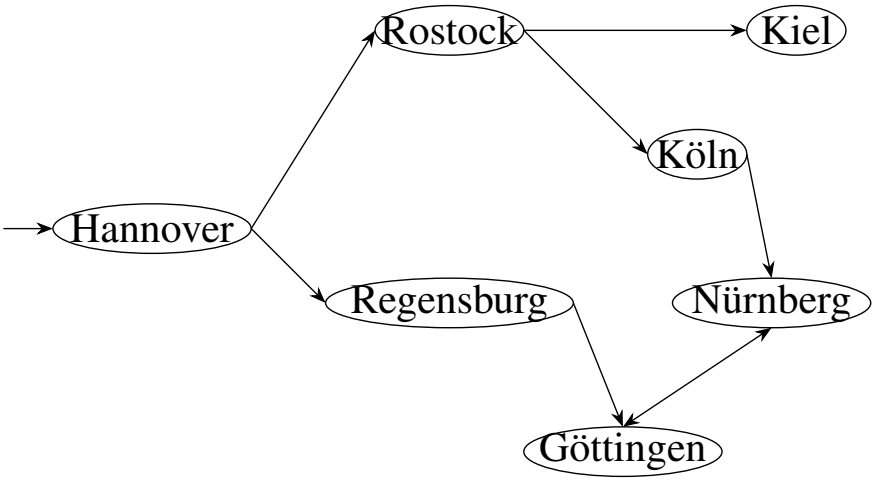

Abbildung 3.2: Beispiel für eine Partie „Geographie“

Das Spiel „Geographie“ kann als gerichteter Graph generalisiert werden, wie Lichtenstein und Sipser in „Go Is PSPACE Hard“ zeigen [LS78, Def.3, S. 394]:

Definition 12 (Generalisiertes Geographie) *Generalisiertes Geographie (GG) ist ein Spiel für zwei Personen, welches auf einem gerichteten Graphen gespielt wird. Die erste Person markiert einen unmarkierten Knoten auf dem Graphen. Anschließend markieren die teilnehmenden Personen abwechselnd einen freien Knoten auf dem Graphen, auf den einen Kante des vorherigen markierten Knotens zeigt. Die Person, welche keinen gültigen Zug mehr machen kann, hat verloren.*

Es wird nun der folgende Satz gezeigt (Beweis nach Schaefer in „On the Complexity of Some Two-Person Perfect-Information Games“ [Sch78, Theo. 3.1, S.194]:

Satz 2 *Eine Zugfolge zu finden, mit welcher Person A in GG gewinnen kann, ist PSPACE-schwer.*

Beweis. Gegeben seien eine QBF der Form $\exists x, \forall y, \exists z \ldots \varphi(x, y, z, ...)$, mit φ in KNF und ein gerichteter Graph G, bei dem die Kanten einen Spielzug und die Knoten einen Spielzustand darstellen. Person A spielt die $\forall$-Quantoren und versucht die QBF „falsch“ werden zu lassen. Person B spielt die $\exists$-Quantoren und versucht die QBF „wahr“ werden zu lassen. Die Belegungen der Wahrheitswerte der Variablen durch die Quantoren werden durch Teilgraphen in Diamantenform realisiert.

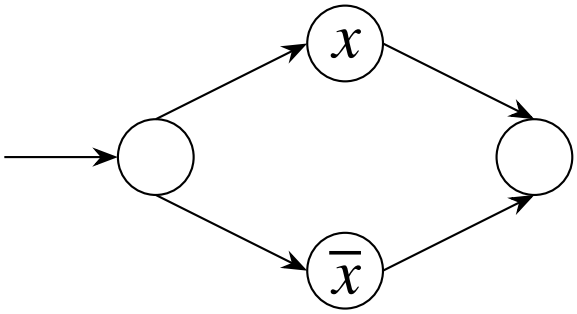

Abbildung 3.3: Belegung der Variable x

Die Spielenden belegen abwechselnd ihre jeweiligen, durch Knoten repräsentierten, Variablen mit Wahrheitswerten. Wurden alle Variablen belegt, wird die Erfüllbarkeit von φ geprüft. Hierzu führen Kanten für „$\forall$“ zu der jeweiligen Negation, der in jeder Klausel von φ vorkommenden Variable. Das Spiel endet, sobald einer der Spielenden keine gültigen Züge mehr hat.

Da der Graph durch „Dummyknoten“ so konstruiert werden kann, dass jede mögliche Anzahl von Variablen und jede mögliche Formel φ simuliert werden kann, ist TQBF auf das Problem aus Satz 2 reduzierbar. Damit es PSPACE-schwer, eine Zugfolge zu finden, welche zum Sieg der Partie führt. □

Beispiel Gegeben sei ein gerichteter Graph, welcher der QBF $\exists x \forall y \exists z : (x \vee y \vee z) \wedge (x \vee \overline{y} \vee z)$ entspricht. Die Spielenden Personen sind „$\exists$“ (gennant E) und „$\forall$“ (genannt A). Das Ziel von E ist es, die QBF „wahr“ werden zu lassen. A möchte das Gegenteil erreichen. Das Spiel beginnt am Start-Knoten und da die QBF mit einem Existensquantor beginnt, ist E zuerst am Zug. Durch die Wahl eines Knotens belegt E die Variable x nun mit einem Wahrheitswert. In Abbildung 3.4 entspricht der Knoten x der Belegung der Variable x mit „wahr“. Analog wird die Belegung von x mit „falsch“ durch den Knoten $\overline{x}$ dargestellt. E wählt den Knoten x und damit ist A am Zug. Vom Knoten x aus gibt es nur eine Kante, die zu einem anderen Knoten führt. A hat somit keine Wahl und belegt den Knoten A_1. Gleiches wiederholt sich mit E und dem Knoten E_1. Anschließend kann A eine Variable belegen. Dieses Muster setzt sich fort, bis alle Variablen belegt wurden. Nachdem der Knoten A_3 erreicht wurde, ist E am Zug. Nun kann E einen der Knoten K_1 oder K_2 wählen. Diese Knoten stellen die Klauseln dar. K_1 entspricht in diesem Besipiel $(x \vee y \vee z)$ und K_2 enstpricht $(x \vee \overline{y} \vee z)$. Angenommen, A hat y mit „wahr“ belegt. Wenn E nun den Knoten K_1 wählt, ist A am Zug und hat keine gültigen Züge mehr. Damit hat A verloren. Belegt A y mit „falsch“ so wählt E den Knoten K_2 und das Ergebnis bleibt dasselbe. E hat immer dann eine Gewinnstrategie, wenn die QBF erfüllbar ist. Wenn die QBF nicht erfüllbar ist oder falsch belegt wurde, dann hätte A bei jeder Klausel immer einen Knoten, der noch nicht belegt wurde. In dem Fall wäre E am Zug, aber hätte keine gültigen Züge mehr. Dann hätte A gewonnen.

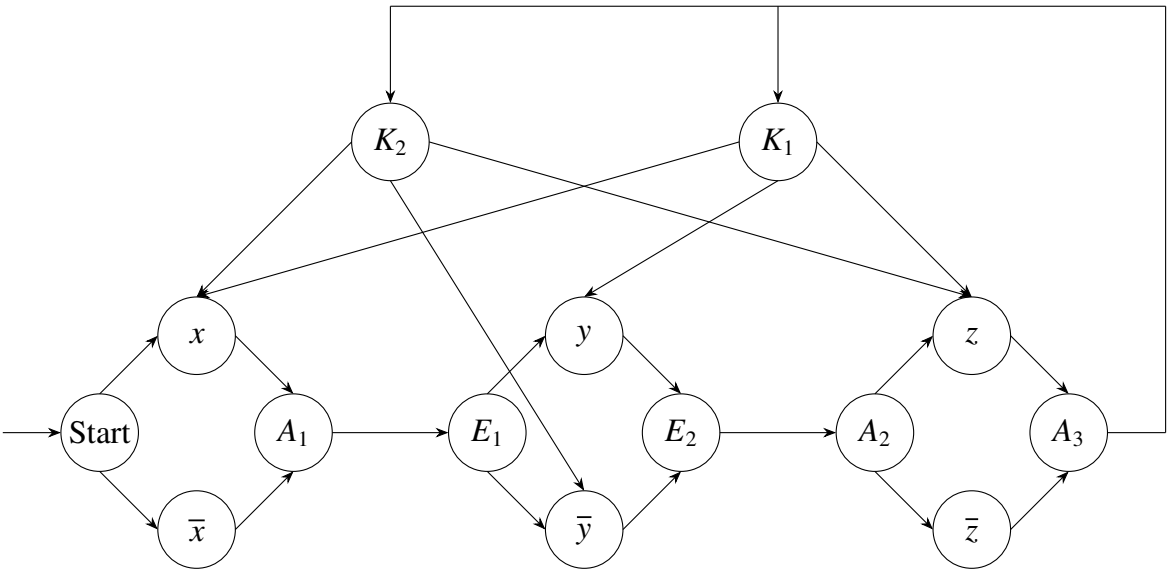

Abbildung 3.4: Generalisiertes Geographie für die QBF $\exists x \forall y \exists z : (x \vee y \vee z) \wedge (x \vee \overline{y} \vee z)$

Es wurde von Lichtenstein und Sipser gezeigt, dass die Reduktion durch Umstrukturierung des GG Graphen auch gilt, wenn man GG auf bipartite, planare Graphen beschränkt, bei denen jeder Knoten höchstens mit drei Kanten inzidiert [LS78, Theo. 3, S. 395f.]. Diese Eigenschaft wird in dem nächsten Schritt verwendet.

3.1.2 Generalisiertes „Tic-Tac-Toe “ zu gewinnen ist PSPACE-schwer

In diesem Abschnitt wird die Reduzierbarkeit von GG auf GTTT gezeigt, wodurch gleichzeitig die Reduzierbarkeit von TQBF auf GTTT gezeigt wird. GTTT ist damit PSPACE-schwer.

Durch die PSPACE-Schwere von GG kann nun durch Reduktion der Beweis von Satz 1 geführt werden (Beweis nach Reisch in „Gobang ist PSPACE-vollständig“ [Rei80, Lemma II.4, S.62ff.]):

Beweis. Um in generalisiertem TTT den bipartiten, planaren Graphen mit höchstens drei Kanten pro Knoten zu simulieren, werden alle möglichen Kombinationen aus eingehenden und ausgehenden Kanten in Stellungen beim generalisiertem TTT übersetzt. Dazu werden die „X“ und „O“ so gelegt, dass immer ein Feld von der Person, welche am Zug ist, belegt werden muss. Tut sie dies nicht, so verliert sie die Partie. Diese offenen Felder Stellen die Knoten dar. Die bereits im Vorfeld belegten Felder stellen die Kanten dar. So lassen sich alle Eigenschaften des GG Graphen in generalisiertem TTT simulieren. Daher ist GG reduzierbar auf GTTT und GTTT ist PSPACE-schwer. □

3.2 Generalisiertes „Tic-Tac-Toe “ zu gewinnen ist PSPACE-vollständig

In diesem Abschnitt wird gezeigt, dass GTTT in PSPACE liegt. Mit dem Ergebnis des vorherigen Abschnitts folgt daraus, dass GTTT PSPACE-vollständig ist. Betrachtet wird der folgende Satz:

Satz 3 *Eine Zugfolge zu finden, mit welcher „X“ in generalsiertem Tic-Tac-Toe gewinnen kann, ist ein polynomielles Platzproblem.*

Beweis. [Rei80, Lemma II.2, S.61] Gegeben sei eine Partie generalisiertes TTT auf einem $n \times n$ Spielbrett. Jede Zelle auf dem Spielbrett kann genau eine von 3 Zuständen annehmen: Leer, „X “ und „O “. Es gibt demnach 3^{n^2} mögliche Zustände, welche das Spielbrett annehmen kann. Nach jedem Zug wird einer dieser Zustände angenommen und kann im Spielverlauf nicht mehr geändert werden. Daher gibt es $n^2!$ mögliche Spielzüge, die auf dem Spielbrett gespielt werden können [1]. Alle diese Spiele lassen sich in einem Zustandsgraphen darstellen. Da es sich um $n^2!$ Zustände handelt, ist der Graph von polynomieller Größe.

Es gibt für jeden möglichen ersten Zug einen Startknoten. Außerdem gibt es für jeden möglichen Spielausgang einen Endknoten. Nun lässt sich mittels einer NTM von jedem

[1] Tatsächlich lässt sich die Anzahl der möglichen Spielzüge noch weiter Einschränken, wenn illegale Spielzüge, Spiegelungen und Drehungen des Spielbrettes aus der Zählung ausgeschlossen werden. Da dies keinen Einfluss auf den Beweis hat, wurde darauf verzichtet diese Aspekte in den Beweis mit aufzunehmen.

Startknoten aus ein Weg zu einem entsprechenden Endknoten finden, sofern dieser existiert. Es gibt also einen nichtdeterministischen Algorithmus, der das Problem aus Satz 3 löst, wodurch das Problem in NPSPACE liegt. Durch den Satz von Savitch liegt das Problem in PSPACE [MV15, Satz 8, S. 31]. □

Aus Satz 1 und Satz 3 folgt somit, dass GTTT PSPACE-vollständig ist.

4 Spiel 2: „Super Mario Bros.“

Im folgenden Kapitel wird das Spiel „Super Mario Bros.“ kurz beschrieben und generalisiert. Anschließend wird das Lösungsproblem für die gewählte Generalisierung von „Super Mario Bros.“ formuliert, sowie ein Beweis für die PSPACE-Vollständigkeit dieses Problems vorgestellt.

„Super Mario Bros.“ (SMB) ist ein zweidimensionales „Jump-’n’-Run“ Spiel, welches Nintendo entwickelte.

Ziel des Spiels ist es den Charakter „Super Mario“ an verschiedenen Hindernissen und Feinden vorbei ins Ziel zu steuern.

Um „Super Mario Bros.“ komplexitätstheoretisch zu betrachten, muss das Spiel zunächst generalisiert werden. Eine mögliche Generalisierung zeigt sich in „*Super Mario Bros. is Harder/Easier than We Thought*“ von Demaine, Viglietta und Williams [DVW16, Abschnitt 2.1]. Diese wurde für den Beweis der PSPACE-Vollständigkeit angepasst [DVW16, Abschnitt 4]. Dabei werden die folgenden, vom realen Spiel abweichenden, Annahmen getroffen:

- Die Level sind beliebig groß und haben beliebig viele Objekte
- Die Zeit und Anzahl der Leben steigen linear im Verhältnis zur Größe des Levels
- Der Bildschirm bewegt sich nicht nur in eine festgelegte Richtung
- Objekte, die den Bildschirm verlassen, setzen sich nicht zurück, insbesondere die Position der Monster

Diese Version des Spiels wird als *generalisiertes Super Mario Bros.* bezeichnet. Unter diesen Annahmen wird der folgende Satz betrachtet:

Satz 4 *Die Flagge in generalisiertem Super Mario Bros. zu erreichen, ist PSPACE-vollständig.*

4.1 Generalisiertes „Super Mario Bros.“ zu gewinnen ist PSPACE-schwer

In diesem Abschnitt wird der Folgende Satz gezeigt:

Satz 5 *Die Flagge in generalisiertem Super Mario Bros. zu erreichen, ist PSPACE-schwer.*

Das Problem aus Satz 5 wird mit „GSMB“ abgekürzt. Für zweidimensionale Plattformspiele wurden verschiedene Meta-Theoreme aufgestellt, welche Lösungsprobleme verschiedener Spiele unter bestimmten Bedingungen in P-schwer, NP-schwer und PSPACE-schwer einteilen. Besonders relevant für diese Arbeit ist das unter Metatheorem 4.c zu findende von Viglietta in „*Gaming Is a Hard Job, But Someone Has to Do It!*“ [Vig12, S.8f.]:

Satz 6 *Wenn in einem Spiel Türen und Druckplatten vorkommen und der Spielavatar einen Ausgang erreichen muss, dann gilt:*

[…]

(c) *Wenn jede Tür von zwei Druckplatten gesteuert wird, ist das Spiel PSPACE-schwer.*

Da in „Super Mario Bros.“ keine Druckplatten existieren, wurde für den Nachweis der PSPACE-Schwere von Demaine et al. eine abgewandelte Form verwendet, welche Viglietta in „*Lemmings is PSPACE-complete*“ vorstellte [Vig15, Metatheorem 3]:

Satz 7 *Wenn in einem Spiel alleinstehende Türen und Pfade mit Kreuzungen (ohne Pfadwechselmöglichkeit) vorkommen und der Spielavatar von einem Startpunkt aus ein Ziel erreichen muss, dann ist das Spiel PSPACE-schwer.*

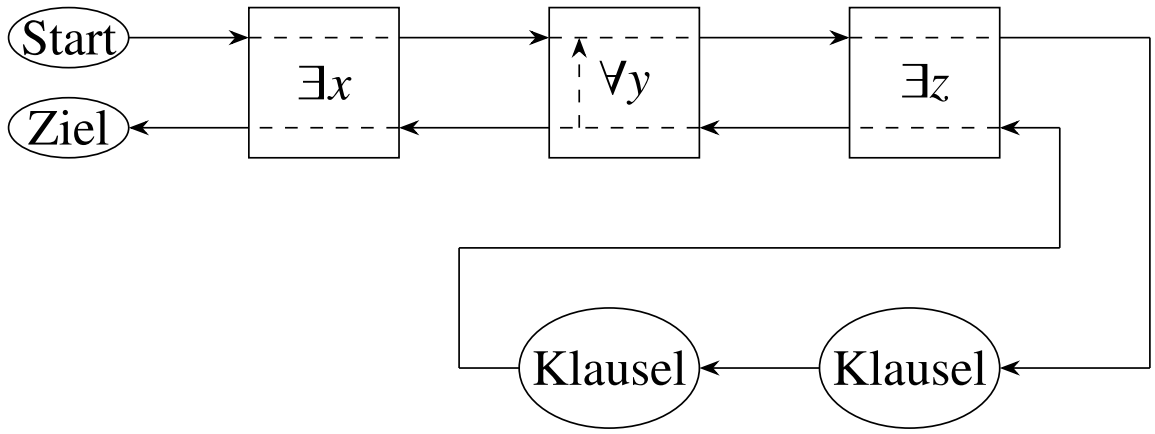

Abbildung 4.1: Aufbau des Beweises von Satz 7

Beweis. [Alo+15, S.6] Für den Beweis werden Eigenschaften der quantifizierten Booleschen Formeln in „Gadgets“ übersetzt, um die Formel im Spiel zu simulieren. Voraussetzung ist eine QBF der Form $\exists x, \forall y, \exists z \ldots \varphi(x, y, z, ...)$, mit φ in 3-KNF. Um das Ziel des Levels zu erreichen, muss die QBF den Wahrheitswert „wahr“ annehmen. Im Kontext der Gadgets bedeutet „wahr“ eine offene und „falsch“ eine verschlossene Tür. Zuerst werden alle Quantoren-Gadgets durchschritten. Die Allquantoren setzen beim ersten Durchlaufen ihre jeweilige Variable auf „wahr“. Bei den Existenzquantoren kann der Wahrheitswert der entsprechenden Variable gewählt werden. Dies geschieht durch das Öffnen der Türen, welche von dieser Variable repräsentiert werden. Nun müssen alle Klausel-Gadgets durchschritten werden. Da die Klauseln in 3-KNF vorliegen, können sie nur passiert werden, wenn mindestens eine Variable „wahr“ ist. Nachdem alle Klauseln erfolgreich durchlaufen wurden, müssen die Quantoren nochmals durchschritten werden. Dabei wechseln die jeweiligen Allquantoren von „wahr“ auf „falsch“ und die Klauseln müssen mit den jeweiligen neuen Wahrheitswerten erneut durchlaufen werden. Diese Abfolge wiederholt sich für jeden Allquantor. Wird eine Kombination von Wahrheitswerten der Existensquantoren gefunden, sodass sich die Klauseln für alle möglichen Wahrheitswerte der Allquantoren passieren lassen, so öffnet sich die Tür zum Ziel und das Level kann abgeschlossen werden.[1] □

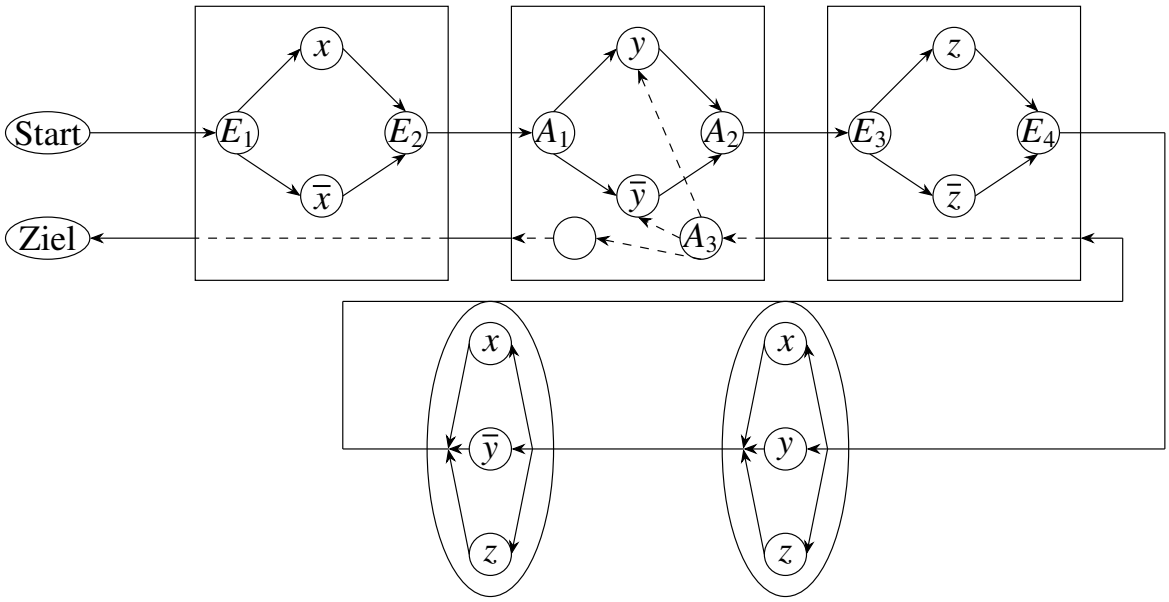

Abbildung 4.2: Generalisiertes SMB für die QBF $\exists x \forall y \exists z : (x \vee y \vee z) \wedge (x \vee \bar{y} \vee z)$

Beispiel Gegeben sei die QBF aus dem letzten Beispiel:$\exists x \forall y \exists z : (x \vee y \vee z) \wedge (x \vee \bar{y} \vee z)$. Es wurde in generalisiertem SMB ein Level konstruiert, welches der QBF entspricht. Die Kanten stellen dabei Gänge dar. Mario beginnt am Startknoten und begibt sich zum Knoten

[1] Ausführlichere Beweisführungen und Beispiele lassen sich in „Classic Nintendo games are (computationally) hard“ [Alo+15] und „Gaming Is a Hard Job, But Someone Has to Do It!“[Vig12] finden.

E_1. Hier kann er die Variable x mit „wahr“ oder „falsch“ belegen und seinen Weg zum Knoten A_1 fortsetzen. Dieser Vorgang wiederholt sich, bis Mario beim Knoten E_4 angekommen ist. Wurden die Variablen so belegt, dass die Klauseln „wahr“ sind, kann Mario mindestens einen Knoten der folgenden Dreiergruppen passieren. Die Knoten entsprechen dabei Türen. Diese sind nur dann geöffnet, wenn die entsprechende Belegung der jeweiligen Variable „wahr“ ergibt. Konnten alle Klauseln passiert werden, kommt Mario am Knoten A_3 an. An diesem Punkt wird er durch verschlossene Türen gezwungen, die Belegung von y zu ändern und alle Klauseln erneut zu durchlaufen. Wird A_3 ein weiteres Mal erreicht, kann Mario passieren und somit das Ziel erreichen. Wäre die QBF nicht erfüllbar, gäbe es keine Möglichkeit das Ziel zu erreichen, da die Klauseln nicht für alle Belegungen passiert werden könnten.

Für den Beweis von Satz 5 werden in „Super Mario Bros.“ die folgenden Elemente benötigt:

- Einen steuerbaren Spielavatar zum Navigieren des Levels
- Ein Start und ein Ziel
- Passierbare Pfade in einer Ebene
- Ein „Crossover-Gadget“ mit dem sich Pfade kreuzen können, ohne dass die anderen Pfade passiert werden können, zum Realisieren der Quantoren
- Türen, welche sich öffnen und schließen lassen, um die Wahrheitswerte darzustellen
- Genügend Zeit, um das Level abzuschließen

All diese Elemente werden von Demaine et al. [DVW16, Kap. 4, S. 10ff.] realisiert und vorgestellt. Nach Satz 7 zeigt sich demnach, dass sich TQBF auf das Problem GSMB reduzieren lässt. Damit ist Satz 5 bewiesen.

4.2 Generalisiertes „Super Mario Bros.“ zu gewinnen ist PSPACE-vollständig

In diesem Abschnitt wird gezeigt, dass das Problem einen Zielpfad in generalisiertem „Super Mario Bros.“ zu finden, in PSPACE liegt. In Kombination mit dem vorherigen Abschnitt wäre damit die PSPACE-Vollständigkeit des Problems gezeigt.

In „*Super Mario Bros. is Harder/Easier than We Thought*“ zeigen Demaine et al. folgenden Satz [DVW16, Abschnitt 2.3, Theorem 1]:

Lemma 1 *Die Flagge in einem Level von SMB-Standard zu erreichen ohne Leben zu verlieren ist ein polynomielles Zeitproblem.*

Bei SMB-Standard handelt es sich um eine Generalisierung von SMB, welche sich nur in einem Punkt von generalisiertem SMB unterscheidet. Bei SMB-Standard werden alle Objekte, welche den Bildschirm verlassen, zurückgesetzt. Die wichtigsten Punkte des Beweises von Lemma 1 sind auch für den Beweis der PSPACE-Vollständigkeit von generalisiertem SMB relevant:

- Die Anzahl der Positionen, welche Mario im Bildschirm einnehmen kann, sind endlich
- Mario und alle Monster und Objekte besitzen eine konstante Höchstgeschwindigkeit
- Alle Zustandsübergänge und Interaktionen zwischen Objekten können in konstanter Zeit berechnet werden, da der Bildschirm eine konstante Größe hat und damit auch eine maximale Anzahl an Objekten, welche miteinander interagieren könnten
- Zufällige Ereignisse werden von einer pseudo-Zufallszahl konstanter Größe generiert
- Es gibt eine Schleife, welche die aktiven Objekte zählt, in polynomieller Größe
- Es gibt eine Schliefe, welche auf Fehlschläge prüft (Zeit abgelaufen, Leben verloren etc.)
- Es existieren Zustandsübergänge für alle möglichen Eingaben für Mario (wird durch die Anzahl an möglichen Eingaben pro Frame begrenzt)
- Alle möglichen Zustandsübergänge zwischen zwei Frames lassen sich in polynomieller Zeit berechnen

Aus diesen Punkten ergibt sich ein gerichteter Zustandsgraph polynomieller Größe, bei dem jede Kante eine mögliche Aktion und das Verstreichen einer Zeiteinheit darstellt. Es gibt einen Startknoten für jede pseudo-Zufallszahl und einen dazugehörigen Endknoten.

Dieser Graph kann nun mittels Breitensuche nach dem kürzesten Pfad durchsucht werden.

Da sich SMB-Standard und generalisiertes SMB nur geringfügig unterscheiden, lässt sich dieser Beweis für den folgenden Satz abwandeln:

Satz 8 *Die Flagge in einem Level von generalisiertem SMB zu erreichen ohne Leben zu verlieren ist ein polynomielles Platzproblem.*

Beweis. Gegeben sei der gerichtete Zustandsgraph polynomieller Größe aus dem Beweis für Lemma 1. Da sich in gerealisiertem SMB die Position und der Zustand aller Objekte und Monster gemerkt wird, müssen alle möglichen Postitionen und Interaktionen von Objekten gespeichert werden. Auch jene, welche sich nicht im Bildschirm befinden. Daher können die Zustandsübergänge nicht mit konstantem, sondern mit polynomiellem Platz gespeichert werden. Da es sich um eine konstante Anzahl an Objekten handelt, welche im Verlauf

des Spiels nur ab- und nicht zunehmen kann, wird kein exponentieller Platz benötigt. Der Zustandsgraph wächst um einen polynomiellen Faktor an, bleibt also polynomiell groß. Es lässt sich somit mittels einer NTM ein Pfad vom Start zum Ziel finden, welcher mit polynomiellem Platz auskommt. Daher gilt, dass das Problem aus Satz 8 in NPSPACE liegt. Der Satz von Savitch besagt, dass NPSPACE = PSPACE gilt. Daher liegt das Problem aus Satz 8 in PSPACE. □

Das Ziel in generalisiertem Super Mario Bros. zu erreichen, ist demnach PSPACE-schwer und liegt in PSPACE, wodurch es PSPACE-vollständig ist.

5 Vergleich

In diesem Kapitel erfolgt ein Vergleich der behandelten Spiele „Tic-Tac-Toe“ und „Super Mario Bros.“ sowie der auf ihnen formulierten Probleme.

Auf den ersten Blick erscheinen die beiden Spiele sehr unterschiedlich. Das zeigt sich im Spielablauf darin, dass in SMB eine Figur ins Ziel gesteuert wird, während bei TTT Markierungen auf ein Spielbrett gelegt werden. SMB ist ein Spiel für eine Person, wohingegen TTT von zwei Personen gespielt wird. TTT ist rundenbasiert. In SMB können jedoch kontinuierlich Eingaben erfolgen. TTT wirkt in seinen möglichen Zügen recht beschränkt. In SMB scheinen die möglichen Aktionen sehr viel komplexer zu sein. Aufgrund der Einfachheit von TTT ist das Spiel sowohl analog, zum Beispiel als Brettspiel oder mit Stift und Papier, als auch digital spielbar. SMB hingegen ist ein Videospiel, welches sich in digitaler Form am leichtesten spielen lässt.

Bei genauerer Betrachtung fallen allerdings ein paar Gemeinsamkeiten auf. Es lässt sich zum Beispiel argumentieren, dass SMB ein Spiel für zwei Personen ist. Eine Person spielt Mario, die andere steuert die Hindernisse und Monster. Dieser Teil wird in der Regel von einem Programm gesteuert. Darüber hinaus sind die Eingaben bei SMB durch die Frame Rate beschränkt. Definiert man die jeweiligen Frames als Runden, so wird SMB zu einem rundenbasierten Spiel für zwei Personen, bei dem beide Spielende ihre Züge gleichzeitig machen. Bei den Generalisierungen wurden die jeweiligen Spielwelten beliebig vergrößert. Für TTT wurden die Grenzen des Spielfeldes entfernt. Bei SMB wurde die Levelbegrenzung entfernt und die Beschränkung, dass sich der Bildschirm nur in eine Richtung bewegt, aufgehoben.

Die Beweise der PSPACE-Schwere weisen mehrere Gemeinsamkeiten auf. In beiden Fällen ist die Grundlage eine QBF in KNF. Beide Beweise verwenden speziell konstruierte „Gadgets“, um die Eigenschaften der QBF zu simulieren. Der theoretische Aufbau und die Zusammensetzung der Gadgets sind dabei fast identisch: Zuallererst werden die Wahrheits-

werte der Variablen festgelegt. Im Anschluss werden die Klauseln der QBF durchlaufen. Danach werden die Wahrheitswerte der Variablen mit Allquantoren umgekehrt und für jeden geänderten Wert werden die Klauseln erneut durchlaufen.

Darüber hinaus werden die Beweise in zwei Schritten geführt. Für GTTT wird erst eine Reduktion von TQBF auf GG geführt und anschließend eine Reduktion von GG auf GTTT. Bei SMB wird zunächst das Meta-Theorem bewiesen und anschließend werden die benötigten Eigenschaften in SMB gezeigt. Daraus ergibt sich, dass sich die Beweise in einigen Punkten voneinander unterscheiden. Beim Beweis der PSPACE-Schwere von GTTT wird ein „Umweg“ über GG genommen. Zwar wird bei GSMB auch das Meta-Theorem verwendet, allerdings lässt dieses direkt mit GSMB beweisen.

Die Beweise der Mitgliedschaft in PSPACE beider Spiele sind nahezu identisch. In beiden Fällen wird von den jeweils möglichen Zuständen ein Zustandsgraph erstellt, der mittels einer NTM navigiert werden kann. Daraus folgt eine Mitgliedschaft in NPSPACE, woraus im nächsten Schritt die Mitgliedschaft in PSPACE folgt. Unterschiedlich sind in diesem Fall ausschließlich die notwendig getroffenen Annahmen, da sich die möglichen Spielzustände deutlich unterscheiden.

Anhand der Aufführungen hat sich gezeigt, dass die betrachteten Spiele mehr Gemeinsamkeiten haben, als es zunächst den Anschein hatte.

6 Zusammenfassung und Ausblick

In diesem abschließenden Kapitel werden die Vorgehensweise und Ergebnisse der Bachelorarbeit zusammengefasst. Zum Schluss werden Fragen, die während der Bearbeitung aufkamen und offen blieben, in einem Ausblick thematisiert.

Zu Beginn wurden die wichtigsten Grundlagen und Definitionen erläutert, um der Arbeit eine theoretische Basis zu geben.

Anschließend wurde das Spiel „Tic-Tac-Toe “ präsentiert und generalisiert. Im Zuge dessen wurde ein Beweis der PSPACE-Schwere vorgestellt, welcher mittels Reduktion von dem Erfüllbarkeitsproblem quantifizierter Boolescher Formeln auf das generalisierte „Geographie “ Spiel mit anschließender Reduktion von generalisiertem Tic-Tac-Toe auf generalisiertes Geographie geführt wurde. Daraufhin wurde ein Beweis dafür erbracht, dass das Entscheidungsproblem, ob eine Person in generalisiertem „Tic-Tac-Toe “ gewinnen kann, in PSPACE liegt. Damit wurde die PSPACE-Vollständigkeit nachgewiesen.

Im folgenden Kapitel wurde mit dem gleichen Aufbau das „Spiel Super Mario Bros.“ vorgestellt, generalisiert sowie ein Beweis für die PSPACE-Vollständigkeit dargestellt. Die

PSPACE-Schwere des Lösungsproblems wurde durch ein Meta-Theorem bewiesen, welches selbst durch Reduktion von dem Erfüllbarkeitsproblem quantifizierter Boolescher Formeln als PSPACE-schwer eingeteilt wurde. Anschließend wurden die von dem Meta-Theorem geforderten Eigenschaften nachgewiesen, was die PSPACE-Schwere von GSMB nachwies. Es folgte ebenfalls ein Beweis für die Mitgliedschaft des Problems in PSPACE, woraus wiederum die PSPACE-Vollständigkeit folgte.

Im letzten Teil wurden zunächst die beiden genannten Spiele selbst verglichen. Im Vergleich stellte sich heraus, dass die Spiele mehr Gemeinsamkeiten aufweisen, als es auf den ersten Blick erkennbar war. So zeigten sich die Spiele in ihren äußeren Erscheinungen beziehungsweise Rahmenbedingungen zwar recht unterschiedlich, jedoch ergaben sich anhand der Beweisführungen viele Gemeinsamkeiten. Zum Beispiel äußerten sich der Aufbau der jeweiligen verwendeten Gadgets sowie die Beweise für die Mitgliedschaft in PSPACE der Lösungsprobleme als nahezu identisch. Das Ziel der Arbeit wurde somit erfüllt.

Während der Bearbeitung ergaben sich fortführende Fragen, die in der Bachelorarbeit aus Gründen der Einschränkungen des Umfangs, nicht bearbeitet werden konnten:

Gibt es Spiele, welche sich nicht als Zwei-Personen-Spiele interpretieren lassen, die sich trotzdem mittels Reduktion von QBF als PSPACE-schwer einteilen lassen? Wie lässt sich ein direkter Beweis für die PSPACE-schwere von „Tic-Tac-Toe“ führen? Weisen alle Beweise für die PSPACE-Vollständigkeit bei Zwei-Personen-Spielen die gefundenen Gemeinsamkeiten auf?

Diese und ähnliche Fragen liefern somit eine gute Basis für weitere Untersuchungen.

Literatur

[LS78] David Lichtenstein und Michael Sipser. „GO Is PSPACE Hard“. In: *19th Annual Symposium on Foundations of Computer Science, Ann Arbor, Michigan, USA, 16-18 October 1978*. IEEE Computer Society, 1978, S. 48–54. DOI: 10.1109/SFCS.1978.17. URL: https://doi.org/10.1109/SFCS.1978.17.

[Sch78] Thomas J. Schaefer. „On the Complexity of Some Two-Person Perfect-Information Games“. In: *J. Comput. Syst. Sci.* 16.2 (1978), S. 185–225. DOI: 10.1016/0022-0000(78)90045-4. URL: https://doi.org/10.1016/0022-0000(78)90045-4.

[Rei80] Stefan Reisch. „Gobang ist PSPACE-vollständig“. In: *Acta Informatica* 13 (1980), S. 59–66. DOI: 10.1007/BF00288536. URL: https://doi.org/10.1007/BF00288536.

[All94] L.V. Allis. „Searching for solutions in games and artificial intelligence“. English. Diss. Maastricht University, Jan. 1994. ISBN: 9090074880. DOI: 10.26481/dis.19940923la.

[KL94] Hans Kleine Büning und Theodor Lettmann. *Aussagenlogik - Deduktion und Algorithmen*. Leitfäden und Monographien der Informatik. Teubner, 1994. ISBN: 978-3-519-02133-9.

[AB09] Sanjeev Arora und Boaz Barak. *Computational Complexity - A Modern Approach*. Cambridge University Press, 2009. ISBN: 978-0-521-42426-4. URL: http://www.cambridge.org/catalogue/catalogue.asp?isbn=9780521424264.

[Vig12] Giovanni Viglietta. „Gaming Is a Hard Job, But Someone Has to Do It!“ In: *Fun with Algorithms - 6th International Conference, FUN 2012, Venice, Italy, June 4-6, 2012. Proceedings*. Hrsg. von Evangelos Kranakis, Danny Krizanc und Flaminia L. Luccio. Bd. 7288. Lecture Notes in Computer Science. Springer, 2012, S. 357–367. DOI: 10.1007/978-3-642-30347-0_35. URL: https://doi.org/10.1007/978-3-642-30347-0%5C_35.

[Alo+15] Greg Aloupis u. a. „Classic Nintendo games are (computationally) hard“. In: *Theor. Comput. Sci.* 586 (2015), S. 135–160. DOI: 10.1016/j.tcs.2015.02.037. URL: https://doi.org/10.1016/j.tcs.2015.02.037.

[MV15] Arne Meier und Heribert Vollmer. *Komplexität von Algorithmen*. Bd. 4. Mathematik für Anwendungen. Lehmanns Media, 2015. ISBN: 978-3-86541-761-9. URL: http://www.lehmanns.de/shop/mathematik-informatik/32129287-9783865417619-komplexitaet-von-algorithmen.

[Vig15] Giovanni Viglietta. „Lemmings is PSPACE-complete“. In: *Theor. Comput. Sci.* 586 (2015), S. 120–134. DOI: 10.1016/j.tcs.2015.01.055. URL: https://doi.org/10.1016/j.tcs.2015.01.055.

[DVW16] Erik D. Demaine, Giovanni Viglietta und Aaron Williams. „Super Mario Bros. is Harder/Easier Than We Thought“. In: *8th International Conference on Fun with Algorithms, FUN 2016, June 8-10, 2016, La Maddalena, Italy*. Hrsg. von Erik D. Demaine und Fabrizio Grandoni. Bd. 49. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2016, 13:1–13:14. DOI: 10.4230/LIPIcs.FUN.2016.13. URL: https://doi.org/10.4230/LIPIcs.FUN.2016.13.