Einsatz von Endspieldatenbanken in Verifikationsspielen

Studienarbeit
von
Okimoto Tenda

Betreut von Robert Mattmüller

Albert-Ludwigs-Universität Freiburg
Fakultät für Angewandte Wissenschaften
Institut für Informatik

2007

Inhaltsverzeichnis

1 Motivation

Zuerst möchte ich als Motivation erklären, wieso das Thema Verifikation mit Hilfe von Spielen [1] besonders interessant ist, was für ein Spiel ich in meiner Studienarbeit betrachtet habe und wie man mit diesem Spiel umgehen kann. Bevor ich solche Fragen antworte, sehen wir erst Mal, was ein Spiel überhaupt ist, was man unter dem Begriff Spiel verstehen kann. Wir haben in unserem Leben tausende Spiele wie zum Beispiel Schach, Othello, Kartenspiel, Sport, sogar einen Vertrag zwischen zwei Firmen oder mehreren Firmen bis zum politischen Problem können wir als ein Spiel betrachten. Es gibt sogar ein wissenschaftliches Gebiet dafür, die Spieltheorie [2]. Wir wissen, dass es für jedes Spiel Spielregeln gibt, unter welchen ein Spiel durchgeführt wird. In unserer Gesellschaft gibt es auch Regeln und unter diesen Regeln leben wir in unserer Gesellschaft. Hier sieht man schon die Ähnlichkeit zwischen dem Spiel und der Gesellschaft. Wir können ein bestimmtes Problem in unserem Leben als ein bestimmtes Spiel betrachten, so dass man manche Probleme besser verstehen kann und somit in der Zukunft eine bessere Lösung bzw. bessere Strategie verwenden kann. Um ein Spiel zu definieren, muss man zuerst Spielregeln überlegen. Auf Unterschied zwischen strategischen Spielen und extensiven Spielen (nur ein Zug oder mehrere Züge) eingehen. Erstens gibt es Spieler. Eine grundlegende Frage ist, welche Spieler an dem Spiel beteiligt sind. Ein Spieler kann eine einzelne Person sein, oder eine Organisation, die aus mehreren einzelnen Personen besteht, eine Firma, oder eine ganze Nation. Ohne Gegenspieler kann man kein Spiel formalisieren, also ist es auch nötig, die Anzahl der Teilnehmer an einem Spiel klar zu stellen. Zweitens gibt es Aktionen. Jeder Spieler hat bestimmte Aktionen. Der Unterschied zwischen Aktion und Strategie ist folgendes: Eine Aktion ist eine „Wahlmöglichkeit“ (z.B. ein Nachfolgerknoten), eine Strategie ist eine Festlegung auf eine bestimmte Aktion in jeder Situation. Meisten existieren mehrere Aktionen und somit hat jeder Spieler Aktionen, eine bestimmte Strategie zu wählen. Es könnte sein, dass ein Spieler unendliche viele Strategien hat. Auf jedem Fall trifft man die Entscheidung, eine bestimmte Aktion zu wählen, so dass ein Spiel weiter durchgeführt werden kann. Drittens gibt es Spielzeit und Anfangszustand. Es ist auch wichtig klar zu stellen, ob ein Spiel nur einmalig oder mehrmals oder mit der festen Zeit durchgeführt wird. Es ist wichtig, ob es überhaupt einen Endspielzustand gibt und wie der Anfangszustand aussieht. Als weitere Regeln kann man zum Beispiel Utility und Kooperation nennen, die für meine Studienarbeit jedoch nicht unbedingt nötig werden.

Nun, was ist dann ein Spiel? In Johan Huizinga [3] steht, dass Spiel eine freiwillige Handlung oder Beschäftigung ist, die innerhalb gewisser festgesetzter Grenzen von Zeit und Raum nach freiwillig angenommen, aber unbedingt bindenden Regeln verrichtet wird, ihr Ziel in sich selber hat. Es gibt unglaublich viele verschiedene Spiele und ihre Regeln. Ich will deswegen zuerst klar stellen, mit welchen Spielen ich mich hier beschäftigt habe. Sie heißen „weak parity games“, also „schwache Paritätsspiel“. Eine formale Erklärung und Definition findet man im Abschnitt 3. Hier möchte ich nur eine kleine Einführung geben, damit man zumindest einen besten Eindruck von schwachen Spielen gewinnen kann. Zuerst betrachten wir die Regeln des schwachen Spiels. Es gibt zwei Spieler, einen Anfangszustand und man weiß genau, wer in diesem Anfangszustand am Zug ist. Wir wissen, wer am Ende dieses Spiel gewinnt oder verliert. Unser Ziel ist zuerst, dass man bei jedem Zustand repräsentieren kann, wer am Zug ist, welche Aktionen existieren und wer gewinnt oder

verliert. Ein weiteres Ziel ist, dass man zum Schluss repräsentieren kann, wer am Anfangszustand gewinnt oder verliert. Um ein schwaches Spiel zu repräsentieren, habe ich hier Binäre Entscheidungsdiagramme verwendet. Was ein Binäres Entscheidungsdiagramm ist, sehen wir gleich im Abschnitt 2 genauer. Was ich noch hier erklären muss, ist über Herangehensweise, welche ich hier verwendet habe. Im allgemeinen gibt es zwei verschiedene Herangehensweisen und zwar Vorwärts-Herangehensweise und Rückwärts-Herangehensweise. Schwache Spiele kann man beide Herangehensweisen lösen. Wir lösen schwache Spiele durch Rückwärts-Herangehensweise. Mit anderem Wort versucht man zuerst durch Binäre Entscheidungsdiagramme ein gegebenes Spiel zu repräsentieren und zu repräsentieren, wer dieses Spiel am Ende gewinnt oder verliert, wie es aussieht und wer im Anfangszustand eine Gewinnstrategie besitzt. Die Spiele zu lösen ist nicht nur ein Spielanalyse, sondern man kann dadurch viel lernen. Zum Beispiel, es gibt bestimmte Spiele bzw. Fälle, in denen man überhaupt keine alternative Wahl hat. Das heißt, dass man schon eine gewinnende oder verlierende Wahl erkennt. Solche Situationen zu kennen spielt natürlich in unserem Leben eine große Rolle. Wenn man z.B. verlierende Wahl kennt, lässt sich bestimmte Risiko in unserem Leben vermeiden. Das heißt, dass man bestimmte Wette aussteigen, auf einen bestimmten Vertrag verzichten, oder an bestimmten Spielen nicht teilnehmen und so weiter. Anders gesagt, wenn man gewinnende Wahl kennt, hat man bei bestimmte Fälle immer einen Vorteil.

Was sind die Spiele noch Mal? Die Spiele sind eine Art, Verifikationsprobleme zu repräsentieren. In meiner Studienarbeit repräsentieren wir ein Verifikationsproblem als Spiel und lösen das Spiel. Zuerst betrachten wir Binäre Entscheidungsdiagramme, durch welche sich ein schwaches Spiel repräsentieren lässt. Wie BDD definiert und was z.B. ein minimales OBDD ist, lernen wir in Abschnitt 2 kennen. Danach erkläre ich, wie ein schwaches Spiel definiert ist. In Abschnitt 4 lernen wir Algorithmus kennen, durch welchen sich schwache Spiele lösen lassen. Auf meiner Studienarbeit habe ich zwei verschiedene Algorithmen vorbereitet. Durch einige Beispiele sehen wir genau, was unser Algorithmus tut. In Abschnitt 5 habe ich ein Experiment bzw. ein großes Beispiel eines schwachen Spiel angegeben. Im letzten Abschnitt habe ich eine Zusammenfassung meiner Forschung beschrieben und möchte zusätzlich interessante zukünftige Forschung beschreiben.

2 Binäre Entscheidungsdiagramme

Bevor wir uns mit dem schwachen Spiel beschäftigen, möchte ich hier kurz über Binäre Entscheidungsdiagramme, kurz BDDs, sprechen. In vielen Bereichen der Informatik, deren Schwerpunkt auf der Darstellung und Manipulation von booleschen Funktionen liegt, spielen BDDs eine bedeutende Rolle. Sie sind eine sehr geeignete und vor allem kompakte Darstellungsmöglichkeit für Mengen von Belegungen boolescher Variablen, die bei geschickter Implementierung leistungsfähige Funktionen zur Verfügung stellen. Eine andere Möglichkeit der Darstellung boolescher Funktionen sind binäre Entscheidungsbäume. Die folgende Abbildung1 zeigt einen Entscheidungsbaum für logische Formel $F = (x \wedge y) \vee z$. Ein Knoten repräsentiert eine Entscheidung über die jeweiligen Variablen durch seine beiden ausgehenden Kanten. Ein Pfad durch einen solchen Entscheidungsbaum repräsentiert eine Belegung der Funktion, deren Wert dem Terminalknoten entnommen wird. Dabei ist auf einem Pfad jede Variable genau einmal vertreten.

Geht man also auf einem solchen Pfad jeden linken Sohn eines Knotens weiter, so ist die Belegung der Variablen des Knotens gleich 0, im anderen Falle entsprechend 1. Man sieht, dass dieser Entscheidungsbaum viele Informationen mehrfach enthält. Wenn man diese überflüssige Redundanz entfernt, erhält man ein Binäres Entscheidungsdiagramm. Zuerst lernen wir die Definition von BDDs kennen und danach, wie man ein gegebenes BDD ordnen und sogenannte OBDD reduzieren kann.

x	y	z	F
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

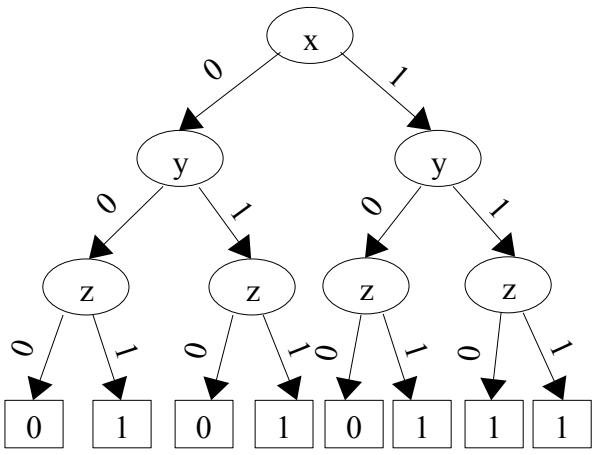


Abbildung 1: Entscheidungsbaum für $(x \wedge y) \vee z$

2.1 Definition

Sei A eine Menge von aussagenlogische Variablen. Ein Binäres Entscheidungsdiagramm über A ist ein gerichteter azyklischer Graph $G = (A, E)$ mit Knoten A, Kanten $E \subseteq A \times A$ und einer Wurzel, der folgende Eigenschaften erfüllen:

- Es gibt genau einen Knoten ohne eingehende Kante, also eine Wurzel.
- Alle Blätter besitzen keine ausgehende Kanten und sind mit einem Wert entweder 0 oder 1 markiert.
- Jeder innere Knoten $a \in A$ besitzt genau zwei ausgehende Kanten, die mit entweder 0 oder 1 markiert sind.
- Wir definieren zwei Funktionen var und ind durch var: $A \rightarrow \{x1,x2,\ldots,xn\} = V$ und ind : $\{x1,x2,\ldots,xn\} \rightarrow \{1,2,\ldots,n\}$. Dann hat Jeder Nichtterminalknoten v als Attribut einen Index ind, der auf eine Eingangsvariable der Menge var verweist.
- zwei unmittelbare Nachfolger low (v): $V \rightarrow V$ und high (v): $V \rightarrow V$.
- Ein Terminalknoten (Knoten ohne Nachfolger) hat als Attribut einen Wert val (v) $\in \{0,1\}$.

Offensichtlich ist für ein BDD B und einen Knoten n von B das Teildiagramm B', das durch alle von n erreichbare Knoten induziert wird, auch ein BDD. Die folgende Abbildung 2 zeigt mögliche BDDs für die aussagenlogische Formel $(x \wedge y) \vee z$. Schon auf den ersten Blick ist ersichtlich, dass BDDs im allgemeinen nicht kanonische Repräsentation sind. Das heißt, dass es für die Formel verschiedene Darstellungsmöglichkeiten von BDDs gibt. Wann ist dann ein BDD eine kanonische

Repräsentation, die wir eigentlich haben möchten? Ein BDD ist eine kanonische Repräsentation, wenn es ein reduziertes geordnetes BDD ist. Zuerst betrachten wir geordnete BDDs und danach lernen wir kennen, wie sich ein geordnetes BDD reduzieren lässt.

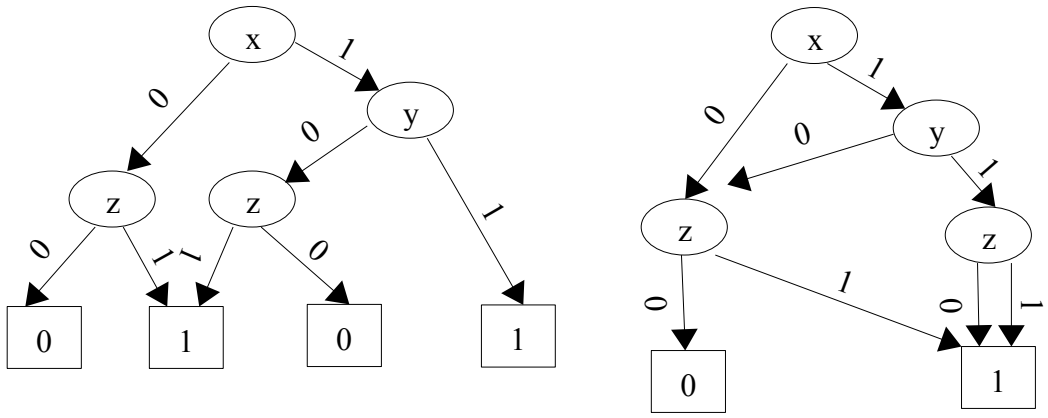
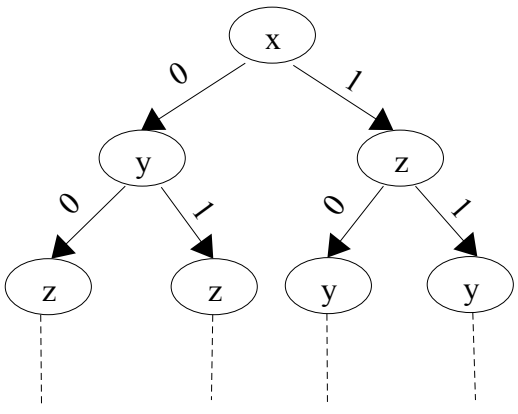


Abbildung 2: BDDs für (x ∧ y) ∨ z

2.2 Geordnete BDDs

Ein OBDD (geordnetes BDD) ist ein gerichteter azyklischer Graph G = (A,E) mit genau einem Wurzelknoten (Knoten ohne Vorgänger). Für ein beliebiges Paar von Nichtterminalknoten der Form (v, low (v)) bzw. (v, high (v)) gilt: ind (v) < ind (low (v)) bzw. ind (v) < ind (high (v)). Also, wenn eine feste Ordnung über der Variablenmenge definiert ist, so spricht man von einem OBDD. Zum Beispiel, wenn wir für die Variablen x, y, z in Abbildung 2 die Ordnung ind (x) < ind (y) < ind (z) betrachten, dann sind beide BDDs in Abbildung 2 geordnete BDDs. Was sind dann nicht geordnete BDDs? Die Abbildung 2' zeigt z.B. ein nicht geordnetes BDD. Wir können für die Variablen x, y, z die Ordnung ind (x) < ind (y) < ind (z) aber auch ind (x) < ind (z) < ind (y) betrachten. Die beiden Ordnungen gelten nicht gleichzeitig. Somit kann man nicht dem BDD in Abbildung 2' die Ordnung geben. Nun haben wir kennengelernt, was ein geordnetes BDD ist. Im nächstem Abschnitt sehen wir weiter, wie sich ein OBDD reduzieren lässt.

Abbildung 2': nicht geordnetes BDD



2.3 Reduktion

Die Idee ist, dass die Information von BDDs möglichst kompakt dargestellt wird. In einem OBDD kann es Knoten geben, deren ausgehende Kanten auf den gleichen Knoten verweisen, und somit keine Entscheidung im eigentlichen Sinne darstellen. Wir wenden auf OBDD Reduktionen an, die genau solche Knoten eliminieren. Ein OBDD wird als reduziert bezeichnet, wenn er keinen Knoten v mit low (v) = high (v) enthält und wenn keine zwei Knoten v und v' existieren, so dass die Teildiagramme mit Wurzelknoten v und v' isomorph sind. Wir lernen Entfernen von Redundanz durch folgende zwei Regeln kennen.

- Eliminationsregel: Überflüssige Knoten werden entfernt.
- Verschmelzungsregel: Isomorphe (=gleichartige) Teil-BDDs werden zusammengefasst.

Zuerst, was ist isomorphe BDDs bzw. Teildiagramme? Zwei BDDs sind isomorph, wenn es zwischen den sie repräsentierenden Knotenmengen eine bijektive Abbildung gibt, die im folgenden Sinne auch strukturerhaltend sein muss:

- Definiere label Funktion für Index i durch: $label_i : E_i \rightarrow \{0,1\}$
- Ist $B_1 = < V_1, E_1, ind_1, var_1, val_1, label_1>$ das erste und $B_2 = < V_2, E_2, ind_2, var_2, val_2, label_2 >$ das zweite BDD mit Wurzeln w_1 bzw. w_2 und $\varphi : V_1 \rightarrow V_2$ die Bijektion, dann muss zusätzlich gelten, dass
  - $\varphi(w_1) = w_2$
  - $ind_1 (v_1) = ind_2 (\varphi(v_1))$ für alle $v_1 \in V_1$
  - $(v_1, v_1') \in E_1$ gdw. $(\varphi(v_1), \varphi(v_1')) \in E_2$ für alle $v_1, v_1' \in V_1$
  - Kantenlabels müssen gleich sein, d.h. wenn $(v1, v'1) \in E$,so $label_1 (v1, v'1) = label_2 (\varphi(v_1), \varphi(v_1'))$.

Die erste Reduktion besteht darin, dass sich die Knoten mit gleichem 0- und 1-Nachfolger entfernen lassen und die eingehende Kante danach zum Nachfolger gehen. Die zweite Reduktion besteht darin, dass sich isomorphe Teildiagramme verschmelzen lassen. Die Abbildung 3 zeigt die Reduktion des in Abbildung 2 dargestellten BDD.

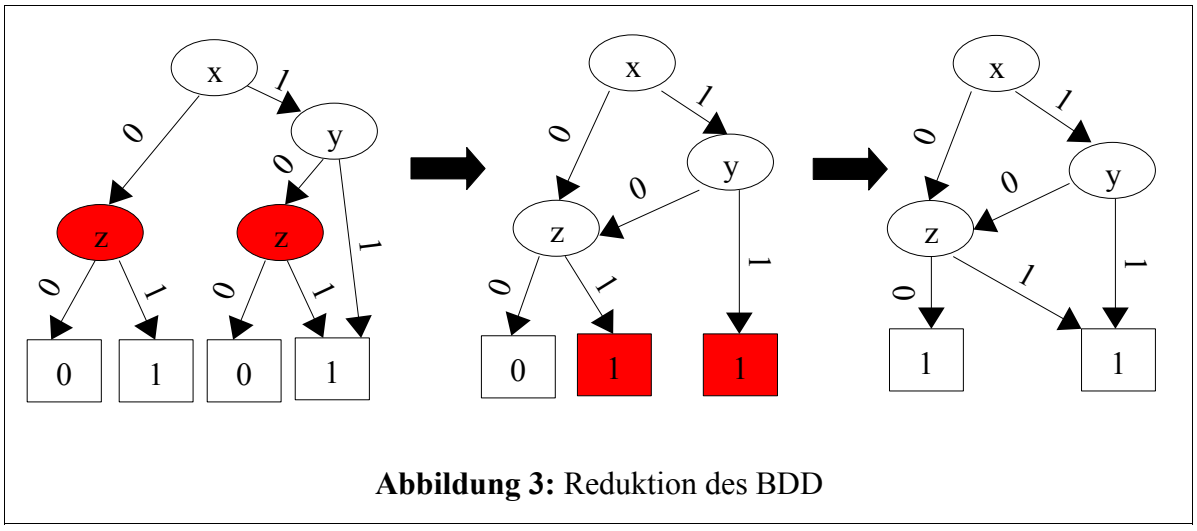


Abbildung 3: Reduktion des BDD

Jetzt haben wir BDDs mit der kanonischen Repräsentation kennengelernt. Zum Schluss dieses Abschnittes stelle ich einen Satz von Bryant (1986) als eine Zusammenfassung der Reduktion vor. Zwei Boolesche Funktionen f1 und f2 sind äquivalent, wenn bei gleichen Bewertungen der Eingangsvariablen die Ausgangsvariablen gleich sind.

Satz von Bryant: Zwei Boolesche Funktionen sind funktional äquivalent genau dann, wenn ihre reduzierten geordneten BDDs für eine gegebene Variablenordnung isomorph sind.

3 Ein schwaches Spiel

In diesem Abschnitt sehen wir genau, was ein schwaches Spiel ist, wie sich es definieren lässt. Wir haben schon in Motivation kurze Zusammenfassung eines schwaches Spiels gesehen. Hier sehen wir genauer, wie ein schwaches Spiel definiert werden kann, und den neuen Begriff „Force set". Dazu noch lernen wir ein kleines Beispiel eines schwaches Spiels kennen.

3.1 Definition

Ein schwaches Spiel ist definiert durch ein Tupel $G = < V_{even}, V_{odd}, E, v_0, \alpha >$, wobei

- $V = V_{even} \uplus V_{odd}$ ist eine endliche Menge von Knoten.
- $v_0 \in V$ ist ein Anfangszustand.
- $E \subseteq V \times V$ ist eine Menge von Kanten.
- $\alpha : V \rightarrow \mathbb{N}$ ist eine Farbfunktion für die gilt: $(v, w) \in E \rightarrow \alpha(v) \leq \alpha(w)$.

Jeder Knoten $v \in V$ hat Ausgangsgrad mindestens eins im Graph (V, E). Für einen Knoten $v \in V_{even}$ sagen wir, dass even der Besitzer von v ist und es gilt owner (v) = even. Für einen Knoten $v \in V_{odd}$ sagen wir, dass odd der Inhaber von v ist und es gilt owner (v) = odd. Wir sagen, dass das Niveau von v even ist, also level (v) = even genau dann, wenn α(v) gerade ist und das Niveau von v ungerade ist, also level (v) = odd genau dann, wenn α(v) ungerade ist. Für jedes $n \in \alpha(V)$ im Wertebereich von der Farbfunktion α, sagen wir level. Die gewinnende Bedingung für ein schwaches Spiel ist definiert durch Lauf. Ein Lauf in einem Spiel ist eine unendliche Folgen $v_0v_1v_2.....$ in V^ω, so dass $(v_i, v_{i+1}) \in E$ eine Kante ist. Bei einem Lauf gewinnt der Spieler even bzw. odd, wenn die höchste Farbe von Knoten, die unendlich häufig im Lauf auftreten, gerade bzw. ungerade ist. Aufgrund der Monotoniebedingung für Knotenfarben haben fast alle Knoten in einem Lauf die gleiche Farbe und bei jedem Lauf gewinnt entweder der Spieler even oder odd. Ein schwaches Spiel ist eine spezielle Form der allgemeine Parity-Games und infolgedessen gewinnt ein Spieler mit einer „memoryless strategy". Das heißt, dass die Strategie immer nur von aktuellen Zustand s abhängt und nicht von der Folge von Zuständen, die durch laufen werden, um s zu erreichen. Eine solche Strategie für einen Spieler $p \in \{even, odd\}$ ist eine Abbildung $s_p : V_p \rightarrow V$, so dass $(v, v') \in E$ gilt, wenn $s_p(v) = v'$ gilt. Ein Lauf $v_0v_1v_2.....$ ist in der Übereinstimmung mit einer Strategie s_p genau dann, wenn gilt: Für alle $i \in \mathbb{N}$ und $v_i \in V_p$ gilt $v_{i+1} = s_p(v_i)$. Eine Strategie s_p ist eine Gewinnstrategie für einen Spieler p genau dann, wenn der Spieler p bei allen Läufen in Übereinstimmung mit s_p gewinnt. Bei einem Knoten v gewinnt ein Spieler p genau dann, wenn der

Spieler p eine Gewinnstrategie im Spiel $< V_{even}, V_{odd}, E, v, \alpha >$ besitzt und ein Spiel wird von einem Spieler p gewonnen genau dann, wenn der Spieler bei dem Anfangsknoten v_0 gewinnt. Ein Spiel zu lösen bedeutet die Bestimmung davon, welcher Spieler es gewinnt.

3.2 Force Set

In Abschnitt 3.1 haben wir die Definition der schwachen Spiele gesehen. Jetzt kommt die Frage, ob wir gewinnende Zustände markieren können, ohne das Spieldiagramm vollständig zu erforschen. Da die Spiele immer in dem Sinne entscheidbar sind, dass jeder Knoten, insbesondere auch die Wurzel, von genau einem der beiden Spieler gewonnen wird, und dass die Gewinner aller Knoten in endlich vielen Berechnungsschritten bestimmt werden können, gibt es folgende möglichen Fälle: Erstens ist, wenn ein Knoten bereits als gewonnen markiert worden ist. Dann spielt bei diesem Fall keine Rolle, wer nun am Zug ist. Spieler1 oder Spieler2. Zweitens ist, wenn Gegenspieler am Zug ist und alle Nachfolger bereits als gewonnen markiert worden sind. Dann hat Gegenspieler keine Wahl, gewinnende Zustände für Spieler1 zu vermeiden. Schließlich ist, wenn Spieler1 am Zug ist und es mindestens einen Nachfolger gibt, der bereits als gewonnen markiert worden ist. So nimmt Spieler1 den Nachfolger, um diese Spiel zu gewinnen. Auf jedem Fall ist es möglich, ein gewinnendes Kriterium für alle Knoten zu definieren, welche gewinnend für Spieler1 sind, weil sie zu Niveau von Spieler1 gehören und der Gegenspieler im Durchlauf nicht vermeiden kann, dieses Niveau zu verlassen, ohne verlierend oder gewinnend Knoten für Spieler1 zu spielen. Zu diesem Zweck definieren wir einen Force Set F von Spieler $p \in \{even, odd\}$ als Menge von Knoten im gleichen Niveau ($level(F) = \{p\}$) mit folgenden Eigenschaften:

- $\forall v \in F\ (owner(v) = p \Rightarrow \exists w \in succ(v): (w \in F \vee Won^{\wedge}p(w)))$
- $\forall v \in F\ (owner(v) \neq p \Rightarrow \forall w \in succ(v): (w \in F \vee Won^{\wedge}p(w)))$

Die Knoten in Force Set F von Spieler p sind gewinnend für Spieler p, wenn Spieler p bei seinem Zug in Force Set F bleibt oder die bereits als „gewonnen“ markierte Knoten für Spieler p bei seinem Zug wählt. Also, Force Set sind eigentlich nur interessant, wenn man den Wert eines Knotens nicht anders entscheiden kann. Was man noch für Force Set F beachten muss, ist dass, die Farben von F nicht dem Spieler entsprechen. Das heißt, in Force Set F müssen alle Knoten die gleiche Farbe haben, aber muss nicht der gleiche Spieler am Zug sein. Als eine wichtige Bemerkung für Force Set ist die Monotonie.

3.3 Beispiel

Wir haben jetzt die Definition von schwache Spiele und dazu gehörte wichtige Force Set kennengelernt. Nun stelle ich hier ein Beispiel eines schwachen Spiels vor.

3.3.1 Aufgabenbeschreibung

- Es gibt zwei Spieler. Spieler odd und Spieler even.
- Es gibt am Anfang 2n+1 viele endliche Kugeln, wobei $n \in \mathbb{N}$
- Jeder Spieler kann beliebig viele Kugeln nehmen.

- Spieler odd ist am Zug.
- Jeder spielt abwechselnd.
- Nach jeweiligen Zügen von Spieler odd und even zählt man die üblich bleibenden Kugeln.
  (1) Falls es gerade viele Kugeln gibt, gewinnt Spieler even.
  (2) Falls es ungerade viele Kugeln gibt, gewinnt Spieler odd.

Sei etwa n = 5. So haben wir am Anfang 11 Kugeln. Wenn Spieler odd z.B. einen Kugel am Anfang und Spieler even an seinem Zug zwei Kugeln nimmt, so gewinnt Spieler even, da es gerade viele Kugeln (8 Kugeln) üblich bleibt, aber wenn Spieler even an seinem Zug auch einen Kugel nimmt, so gewinnt Spieler odd, da es ungerade viele Kugeln (9 Kugeln) üblich bleibt.

3.3.2 Analyse

Zuerst versuchen wir das oben definierte schwaches Spiel formal zu beschreiben. D.h. wir beschreiben, wie sich das gegebene Beispiel bzw. Spiel mit Hilfe von Definition im Abschnitt 3.1 definieren lässt. Danach sehen wir das Spiel anschaulich. Ich habe für das schwache Spiel ein Spieldiagramm und eine Tabelle angegeben, durch die man sofort sehen kann, wer das Spiel gewinnt. Gegeben sei ein schwaches Spiel $G = < V_{even}, V_{odd}, E, v_0, \alpha >$, wobei

- $V = \{v_0, v_1, v_2\}$, wobei $V_{odd} = \{v_0\}$ und $V_{even} = \{v_1, v_2\}$, d.h. owner (v_0) = odd und owner (v_i) = even, i = 1,2.
- $v_0 \in V$ ist der Anfangszustand
- $E = \{(v0, v1), (v0,v2), (v1, v0), (v1, v2), (v2, v2)\}$ ist die Menge der Kanten.
- $\alpha : V \rightarrow \mathbb{N}$ ist eine Farbfunktion mit $\alpha(v_0) = \alpha(v_1) = 1$ und $\alpha(v_2) = 2$, d.h. level (v_2) = even und level (v_i) = odd, i = 0,1.
- Sei K eine endliche Menge von Kugeln mit der Mächtigkeit $| K | = 2n+1$, wobei $n \in \mathbb{N}$.

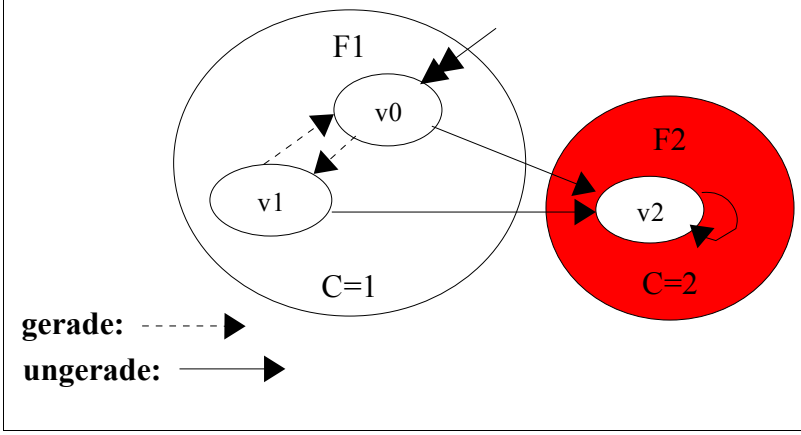
In diesem Spiel ist es vorausgesetzt, dass jeder Spieler optimale Aktion wählt. Die folgende Abbildung 4 zeigt der oben definierte Spielgraph. Zusätzlich gebe ich eine Tabelle an, mit der sich unser Spiel anders analysieren lässt und man besser verstehen kann, wer das Spiel gewinnt. Zuerst sehen wir die folgende Tabelle.

	Even nimmt gerade viele Kugeln	Even nimmt ungerade viele Kugeln
Odd nimmt gerade viele Kugeln	Odd gewinnt	Even gewinnt
Odd nimmt ungerade viele Kugeln	Even gewinnt	Odd gewinnt

Man sieht sofort , dass Spieler even dieses Spiel gewinnt, solange er für gerade ungerade und für ungerade gerade viele Kugeln nimmt. D.h. Falls Spieler odd ungerade viele Kugeln nimmt, nimmt Spieler even gerade viele Kugeln, so dass es immer gerade viele Kugeln üblich bleibt. Analog geht bei umgekehrtem Fall auch. Als Konsequenz kann man sagen, dass der Spieler, wer zuerst am Zug

ist, dieses Spiel verliert. Mit der Abbildung 4 sieht man auch, dass Spieler even den Force Set F2, der die gerade Farbe besitzt, schließlich immer erreichen kann und somit Spieler even das Speil gewinnt. Im nächsten Abschnitt lernen wir kennen, wie sich schwache Spiele lösen lassen.

Abbildung 4: Ein schwaches Spiel



4 Lösen von schwachen Spiele

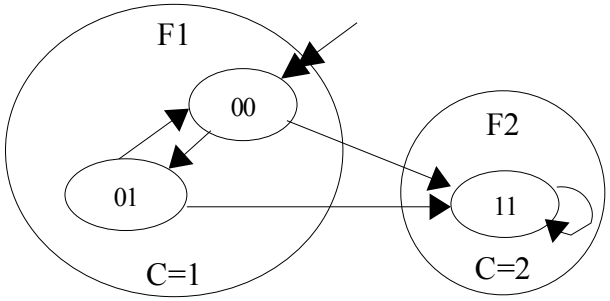
Wir haben gerade in Abschnitt 3 ein Beispiel eines schwachen Spiels gesehen. In diesem Abschnitt lernen wir kennen, wie sich ein schwaches Spiel lösen lässt. Ein Spiel zu lösen bedeutet die Bestimmung davon, welcher Spieler es gewinnt. Zuerst sehen wir, welche Relationen nötig ist, um schwache Spiele zu lösen und wie sie sich binär kodieren lässt. Danach lernen wir unseren Algorithmus für Gewinn- bzw. verlierende Bedingung kennen. Wir sehen insgesamt zwei verschiedene Algorithmen, in den Force Set jeweils anders berechnet wird. Die beiden Algorithmen sind Rückwärts-Algorithmus. Es gibt neben unserer symbolischen rückwärts-gerichteten Methode auch explizite vorwärts-gerichtete Algorithmen zum Lösen von schwachen Spielen. Man kann aber unsere symbolische rückwärts-gerichtete Methode auch zur Erzeugung einer Endspieldatenbank für den Vorwärts-Algorithmus verwenden. Es ist deswegen reicht, Rückwärts-Algorithmus kennenzulernen.

4.1 Kodierung

Gegeben sei ein schwaches Spiel $G = < V_{even}, V_{odd}, E, \{v_0\}, \alpha >$. Die Zustände und Farben werden binär kodiert. Seien $N = | V |$ und $M = \max(\alpha(v))+1$. Dann brauchen wir $n = \lceil \log_2(N) \rceil$ und $m = \lceil \log_2(M) \rceil$ viele Bits für aktuelle Zustände und Farben. Seien $C = \{y_{m-1},.....,y_0\}$ die Menge von color Variablen und $V = \{x_{n-1},.....,x_0\}$ die Menge von Zustandsvariablen. Dann repräsentieren ihre Belegungen $c : C \rightarrow \{0,1\}$ und $v : V \rightarrow \{0,1\}$ den Zustand und die dazugehörige Farbe. Wir lernen gleich durch ein Beispiel kennen, wie man die Zustände bzw. Farben kodieren kann. Gegeben sei wieder unser Spielgraph. Wir kodieren zuerst nur den aktuellen Zustand v. D.h. ein Zustand v, der binär dargestellt ist, enthält die Information für sich und welche Farbe er hat. In Abbildung 4 gibt es insgesamt drei Zustände bzw. binäre Darstellungen 00, 01 und 11, also N = 3. Die maximal Zahl von Farbfunktion ist C = 2, also M = 3. Wegen $n = \lceil \log_2(3) \rceil = 2$ und $m = \lceil \log_2(3) \rceil = 2$ brauchen

wir zwei Bits für Zustände und zwei Bits für die Farbe. Also, die Zustände 00, 01 und 11 werden mit 00 = (¬v1, ¬v0), 01 = (¬v1, v0), 11 = (v1, v0) und die Farben C = 1 und C = 2 werden mit color ≡ (¬c1 ∧ c0 ∧ ¬v1 ∧ ¬v0) ∨ (¬c1 ∧ c0 ∧ ¬v1 ∧ v0) ∨ (c1 ∧ ¬c0 ∧ v1 ∧ v0) ≡ (¬c1 ∧ c0 ∧ ¬v1) ∨ (c1 ∧ ¬c0 ∧ v1 ∧ v0) binär kodiert.

Abbildung 4: Ein schwaches Spiel

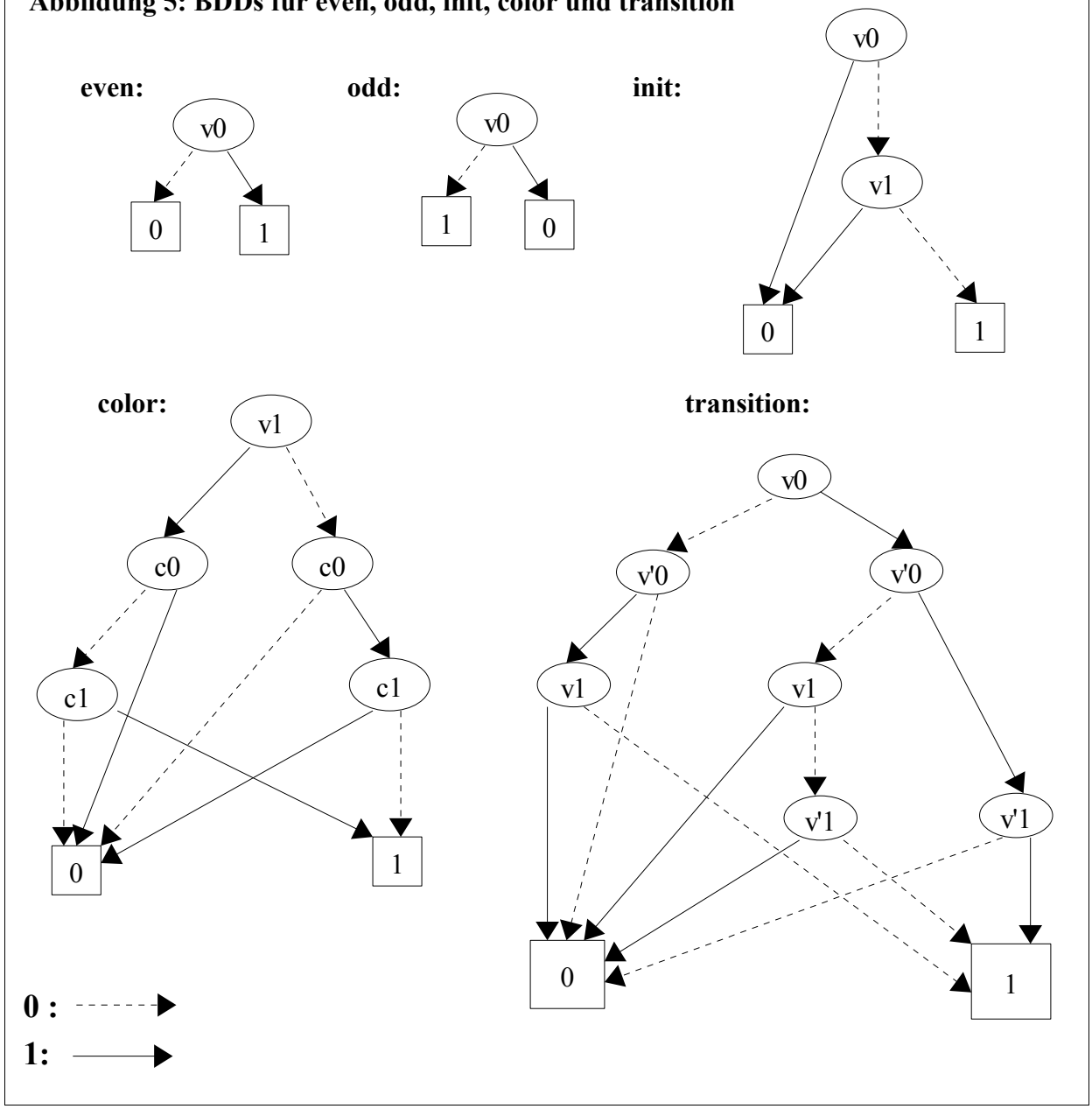


4.2 Vorbereitung

Sei G ein schwaches Spiel. Aus G lassen sich die Relationen even, odd, init, color und leq berechnen, die wichtige Eigenschaften sind, um schwache Spiele zu lösen. Um owner von einem Knoten v zu bestimmen, definieren wir zuerst even (v) und odd (v). Eine Relation even (v) definieren wir dadurch, dass even (v) genau dann gilt, wenn $v \in V_{even.}$ Analog gilt für odd(v), d.h. odd (v) gilt genau dann, wenn $v \in V_{odd.}$ Eine Kante zwischen einem Knoten v und seinem Nachfolger v' definieren wir eine Relation transition (v, v') dadurch, dass transition (v, v') genau dann gilt, wenn (v, v') ∈ E. Die Relation init (v) hat die Eigenschaft, dass init (v) genau dann gilt, wenn $v = v_0$. Weitere Relationen sind für Force Set besonders wichtig. Zuerst definieren wir eine Farbrelation. Die Farbrelation color (v, c) ist definiert durch: color (v,c) gilt genau dann, wenn α(v) = c. Um diese Farbe von color (v, c) zu bestimmen, bereiten wir zwei weitere Relationen evenNumber (c) und oddNumber (c) vor. Eine Relation evenNumber (c) gilt genau dann, wenn c eine gerade Zahl ist und für oddNumber (c) eine ungerade Zahl. Eine weitere wichtige Relation ist leq (c', c), welche die Farben von Force Set berechnet und Monotonie Eigenschaft erfüllt. Eine Relation leq (c', c) hat die Eigenschaft, dass leq (c',c) genau dann gilt, wenn $c' \leq c$. Somit haben wir wichtige Relationen bzw. even, odd, init, transition, color und leq kennengelernt, die für unseren Algorithmus wichtig sind. Wir werden sie auch binär kodieren. Die Kodierungen sind folgendes und die Abbildung 5 zeigt die von (1) bis (5) repräsentierte BDDs.

(1) even (v1,v0) ≡ v0
(2) odd (v1,v0) ≡ ¬v0
(3) init (v1,v0) ≡ ¬v1 ∧ ¬v0
(4) transition (v1,v,v'1,v'0) ≡ (¬v1 ∧ ¬v0 ∧ v'0) ∨ (¬v1 ∧ v0 ∧ (v'1 ↔ v'0)) ∨
(v1 ∧ v0 ∧ v'1 ∧ v'0)
(5) color (c1,c0,v1,v0) ≡ (v1 → c1 ∧ ¬c0) ∧ (¬v1 → ¬c1 ∧ c0)
(6) leq (c'1,c'0,c1,c0) ≡ (¬c'1 ∧ c1) ∨ ((c'1 ↔ c1) ∧ (¬c'0 ∧ c0)) ∨ ((c'1 ↔ c1) ∧ (c'0 ↔ c0))

Abbildung 5: BDDs für even, odd, init, color und transition



4.3 Repräsentation vom Algorithmus

Unser Algorithmus besteht aus zwei Teilalgorithmen bzw. Algorithmus für Gewinnbedingung und verlierende Bedingung. Wenn ich schreibe, dass ein Knoten als „gewonnen“ bzw. „verloren“ markiert ist, meine ich immer für Spieler even. Also, wenn der Wurzelknoten als „gewonnen“ bzw. „verloren“ markiert ist, bedeutet es, dass Spieler even das Spiel gewinnt bzw, verliert. Zuerst sehen wir die zwei Teilalgorithmen genau, wie sich die jeweilige Bedingung definieren lässt und danach sehen wir unser Algorithmus für Lösen von schwachen Spiele.

4.3.1 Algorithmus für Gewinnbedingung

Gegeben sei Spieler P∈{even, odd}. Unser Algorithmus für Gewinnbedingung berechnet durch

Iteration, ob ein nicht markierter Knoten als gewonnen markiert werden kann. Dabei werden drei Methoden bzw. WonByOwnMove, WonByOppMove und WonByForceSet verwendet. In unserem Algorithmus ist es besonders wichtig, wie sich Force Set definieren lässt. Die Idee von Force Set ist folgendes: Wenn wir einen Knoten v als „gewonnen“ für Spieler even markieren können, dann muss sein Nachfolger v' schon als „gewonnen“ markiert sein. Wenn wir aber einen Nachfolger v" finden, der als weder „gewonnen“ noch „verloren“ markiert ist, dann markieren wir ihn als „nicht verloren“. Wir müssen hier darauf beachten, dass die Markierung „nicht verloren“ nicht unbedingt „gewonnen“ bedeutet. Da der Nachfolger v" nicht zu gleichem Force Set von dem Knoten v gehören kann, muss er zu einem anderen Force Set gehören. Wenn der Nachfolger v" zu dem gleichen Force Set von dem Knoten v gehören könnte, so müsste er als „gewonnen“ makiert werden können. Also, wir geben einen anderen Force Set für den Nachfolger v" mit der Eigenschaft von Monotonie der Farbfunktion an, also es gilt für neuen gegebenen Force Set noch $\alpha(v'') < \alpha(v)$.
Nun sehen wir genau, wie unsere drei Methoden für Gewinnbedingung definiert sind. Ich gebe zusätzlich an, wie sich die Gewinnbedingungen mathematisch beschreiben lassen. Die Mengen won^i können induktiv berechnet werden. Dabei müssen die folgenden möglichen Gewinnbedingungen in Iterationsschritt i+1 berücksichtigt werden:

- Ein Zustand war schon im vorherigen Iterationsschritt als gewonnen markiert.
  (1) won^i (v)

- In Zustand v ist Spieler even am Zug und es gibt mindestens einen Nachfolger v' von v, der bereits im letzten Schritt als gewonnen markiert war.
  (2) $\text{wonByOwnMove}^{i+1}(v) = \text{Even}(v) \wedge \exists v' (R(v, v') \wedge \text{won}^i(v'))$

- In Zustand v ist Spieler odd am Zug und alle Nachfolger v' von v waren bereits im letzten Schritt als gewonnen markiert war.
  (3) $\text{wonByOppMove}^{i+1}(v) = \text{Odd}(v) \wedge \forall v' (R(v, v') \rightarrow \text{won}^i(v'))$

- Ein Zustand liegt in einem trivialen Force-Set für Spieler even, das heißt,
  $\text{wonByForcsSet}^{i+1}(v) =$ (4) ∧ (5) ∧ (6) ∧ (7) ∧ (8) mit
  - $\alpha(v)$ ist gerade.
  (4) $\exists c (\text{Color}(v, c) \wedge \text{EvenNumber}(c))$

  - v ist im vorherigen Iterationsschritt noch nicht als verloren markiert,
  (5) $\neg \text{lost}^i(v)$

  - alle v' mit $\alpha(v) < \alpha(v')$ sind schon als gewonnen oder verloren markiert,
  (6) $\text{allHigherColoredDecided}^{i+1}(v) =$
  $\exists c \text{ Color}(v, c) \wedge \forall c' \forall v' (\text{Color}(v', c') \rightarrow (\text{Leq}(c', c) \vee \text{won}^i(v') \vee \text{lost}^i(v')))$

  - in v ist Spieler even am Zug und es gibt einen Nachfolger v' von v, der im letzten Schritt noch nicht als verloren markiert war oder in v ist Spieler odd am Zug und es gibt keinen Nachfolger v' von v, der im letzten Iterationsschritt schon als verloren markiert war,
  (7) $\text{notLostByOwnMove}^{i+1}(v) = \text{Even}(v) \rightarrow \exists v' (R(v, v') \wedge \neg \text{lost}^i(v'))$
  (8) $\text{notLostByOppMove}^{i+1}(v) = \text{Odd}(v) \rightarrow \forall v' (R(v, v') \rightarrow \neg \text{lost}^i(v'))$

4.3.2 Algorithmus für verlierende Bedingung

Unser Algorithmus für verlierende Bedingung ist ähnlich wie für Gewinnbedingung definiert. Wir sehen hier nur mathematische Beschreibungen für drei Methoden, die im Algorithmus für verlierende Bedingung verwendet werden.

- lostByOppMove^i+1 (v) = Odd (v) ∧ ∃v′(R(v, v′) ∧ lost^i (v′))
- lostByOwnMove^i+1 (v) = Even (v) ∧ ∀v′(R(v, v′) → lost^i (v′))
- lostByForcsSet^i+1 (v) = (4') ∧ (5') ∧ (6') ∧ (7') ∧ (8') mit
  - (4') ∃c (Color(v, c) ∧ OddNumber (c))
  - (5') ¬ won^i (v)
  - (6') allHigherColoredDecided^i+1 (v) = ∃c Color(v, c) ∧
    ∀c′ ∀v′(Color(v′, c′) → (Leq (c′, c) ∨ won^i (v′) ∨ lost^i (v′))
  - (7') notWonByOwnMove^i+1 (v) = Even (v) → ∀v′(R(v, v′) → ¬ won^i (v′))
    (8') notWonByOppMove^i+1 (v) = Odd (v) → ∀v′(R(v, v′) ∧ ¬ won^i (v′))

4.3.3 Algorithmus für Lösen von schwachen Spiele

Wir haben gerade Algorithmen für Gewinn- und verlierende Bedingung kennengelernt. Dabei ist es besonders wichtig, wie sich Force Set definieren lässt. Nun sehen wir, wie sich durch zwei Algorithmen schwache Spiele lösen lassen. Mit der ersten while-Schleife werden alle Knoten, die noch nicht markiert sind, als „gewonnen“ oder „verloren“ markiert. Mit der zweiten while-Schleife wählt Spieler seinen optimalen Zug. So werden alle Knoten bis zum Wurzelknoten nach Iterationen entweder „gewonnen“ oder „verloren“ markiert. In unserem Algorithmus finden wir eine Stelle, wo mit ☺ gekennzeichnet wird. In dieser Stelle gebe ich im nächsten Abschnitt 4.4 einen wichtigen Satz für alternativen Algorithmus an.

```
function SolveWeakGame ():
    represent game symbolically
    initialize won and lost with ⊥
    repeat until won^i+1 = won^i and lost^i+1 = lost^i
            repeat until fixpoint
              oldWon^j+1 ← oldWon^j ∨ wonByOwnMove^j+1 ∨ wonByOppMove^j+1
              oldLost^j+1 ← oldLost^j ∨ lostByOwnMove^j+1 ∨ lostByOppMove^j+1
            end
            ☺
            won^i+1 ← won^i ∨ wonByForceSet^i+1
            lost^i+1 ← lost^i ∨ lostByForceSet^i+1
    end
    return won(v0)
end
```

4.4 Alternativer Algorithmus

Wir haben schon Algorithmus für Lösen von schwachen Spiele kennengelernt. Wie sich ein Force Set definieren lässt, spielt dabei besonders eine Rolle. Die Idee war, dass ein Knoten, der maximale Farbe hat und noch nicht als „gewonnen“ bzw. „verloren“ markiert ist, den Nachfolger hat, der sich als „gewonnen“ oder „verloren“ markieren lässt oder zu einem anderen Force Set gehört. Wir wollen in diesem Abschnitt versuchen, einen alternativen Algorithmus zu definieren. Das Schlüsselwort für alternativen Algorithmus ist „Zyklus“ und durch „Zyklus“ versuchen wir neuen Force Set zu definieren. Bevor wir unseren alternativen Algorithmus sehen, lernen wir einen wichtigen Satz kennen. Im letzten Abschnitt 4.3.3 haben wir Algorithmus für Lösen von schwachen Spielen kennengelernt. Im Algorithmus habe ich eine Zeile angegeben, die mit ☺ gekennzeichnet wird. Der folgende Satz gilt an dieser Stelle. D.h. es gibt momentan nur die Knoten, die nicht genau als „gewonnen“ oder „verloren“ markiert werden können. Mit dem anderen Wort haben wir nur die unentschiedenen Knoten.

4.4.1 Satz von Zyklus-Force Set

Sei V eine endliche Knotenmenge von schwache Spiele $G = < V, E, v_0, \alpha >$. Seien $M \subseteq V$ die Menge aller schon entschiedenen Knoten von V und $N \subseteq V \setminus M$ das Komplement von M. Jedes Element aus V hat mindestens einen Nachfolger. Dann gilt folgender Satz:

Satz: Ist $N \neq \emptyset$, so existiert $\{n_1,.....,n_k\} \subseteq N$, so dass $(n_i, n_{(i+1) \bmod k}) \in E$ für $i = 1,2,...,k$ $(k \geq 1)$.

Beweis durch Induktion: Sei $m_1,....., m_s$ endlich viele Elemente aus M. Sei noch $n_1,....., n_r$ endlich viele Elemente aus N und nicht leer.
Induktionsanfang: N besteht aus einem einzigen Element. Sei $n \in N$. Da n aber auch Element von V ist, hat mindestens einen Nachfolger. Wenn n nur Nachfolger in M hat, so ist n als „gewonnen“ oder „verloren“ markiert. D.h. n ist nicht mehr Element von N. Also, n muss einen Nachfolger in N haben. Da es aber nur einen Element n in N existiert, muss n als Nachfolger sich selbst nehmen.
Induktionsschritt: Sei $N' = \{n_1,....., n_i\}$ für $2 \leq i \leq r$.
Annahme: N' bildet keinen Zyklus. Da N' eine endliche Teilmenge von V ist, hat jedes Element aus N' mindestens einen Nachfolger. Jedes Element aus N' ist noch nicht als „gewonnen“ oder „verloren“ markiert bzw. entschieden. Deswegen muss jedes Element aus N' mindestens einen Nachfolger wieder in N' haben, sonst gehört das Element zu M. Sind alle Nachfolger von n_i schon als „gewonnen“ oder „verloren“ markiert, dann lässt sich das Element n_i auch als „gewonnen“ oder „verloren“ markieren. Wähle ni einen Nachfolger von n_{i-1}, wobei $2 \leq i \leq r$, dann hat jedes Elemente von $n_{i-1},.....,n_1$ seinen Nachfolger in N'. Da $n_i \in N'$ mindestens einen Nachfolger hat und es nach Annahme keinen Zyklus in N' geben muss, hat n_i seine Nachfolger nur in M. Dann ist n_i entweder als „gewonnen“ oder „verloren“ markiert. Das ist aber Widerspruch, dass n_i Element aus N' ist. Also, Annahme ist falsch, somit gibt es einen Zyklus in N'. qed.

Bemerkung: Besteht N aus einem einzigen Element n, so gibt es einen sogenannten „Selbstzyklus“, der sich selbst Zyklus macht. (Induktionsanfang)

4.4.2 Force Set

Wir überlegen zuerst, in welcher Situation Spieler even vermeiden kann, in einem schon als „verloren“ markierten Nachfolger zu ziehen, so dass er das Spiel nicht verliert. Der Spieler sucht alternative Wege, bis er entweder einen Knoten erreicht, der als „gewonnen“ markiert ist, oder in einem Zyklus bleibt. Den ersten Fall kümmert unsere Gewinnbedingung, die wir schon im Abschnitt 4.3.1 gesehen haben. Wenn der Spieler even aber keinen solchen Knoten bis zur Wurzel findet, so hat er keine Möglichkeit, diese verlierende Situation zu vermeiden. Bei dem zweiten Fall definieren wir neuen Froce Set durch Zyklus. Mit Hilfe vom Satz in Abschnitt 4.4.1 können wir ohne Kontrolle feststellen, dass sich ein Zyklus durch die nicht als „verloren“ markierten Knoten definieren lässt. Ohne Kontrolle meine ich, dass man nicht braucht, die Existenz von Zyklus zuerst prüfen zu müssen. Für neuen Force Set für Gewinnbedingung muss es aber zuerst klar sein, mit welchem Knoten ein Zyklus anfängt und welche Farbe der neue Force Set hat. Der Anfangsknoten von Zyklus ist für neuen Force Set vorausgesetzt. Wir sehen in diesem Abschnitt nicht ganzen Algorithmus, sondern nur, wie sich Force Set definieren lässt, weil die sonstige Methoden gleich wie im Algorithmus im Abschnitt 4.3 definiert sind. Force Set für Gewinnbedingung im alternativen Algorithmus besteht aus drei Methoden bzw. wonZykleAnfangByOwnMove, wonZykleSchrittByOwnMove und wonZykleSchrittByOppMove. Nun sehen wir genau, wie sich der Force Set definieren lässt und zusätzlich noch, wie er mathematisch beschrieben werden kann. Die Menge won^i kann induktiv berechnet werden. Dabei müssen die folgenden möglichen Gewinnbedingungen in Iterationsschritt i+1 von Force Set berücksichtigt werden:

- Wenn ein Zustand v mit der geraden Farbe nicht bereits als verloren markiert ist und mindestens eine von drei folgenden Bedingungen gilt, dann liegt er in einem wonByForceSet. wonByForceSet^i+1 (v) = ¬ lost^i (v) ∧ (1) ∧ (2) ∧ ((3) ∨ (4)) mit

  - α(v) ist gerade

  (1) ∃c (Color(v, c) ∧ EvenNumber (c))

  - In Zustand v ist Spieler even am Zug und es gibt mindestens einen Nachfolger v' von v, der bereits im letzten Schritt als verloren markiert war und es gibt mindestens noch einen Nachfolger v' von v, der bereits im letzten Schritt nicht als verloren markiert war.

  (2) wonZykleAnfangByOwnMove^i+1 (v) =
  Even (v) ∧ ∃v´(R(v, v') ∧ lost^i (v')) ∧ ∃v´(R(v, v') ∧ ¬ lost^i (v'))

  - In Zustand v ist Spieler even am Zug und es gibt mindestens einen Nachfolger v' von v, der bereits im letzten Schritt nicht als verloren oder (3) markiert war. Der Nachfolger v' von v hat eine Farbe c' mit der geraden Zahl.

  (3) wonZykleSchrittByOwnMove^i+1 (v) = Even (v) ∧ ∃v´∃c' (R(v, v') ∧ Color(v', c') ∧
  EvenNumber (c') ∧ (¬ lost^i (v') ∨ wonZykleAnfangByOwnMove^i (v'))

  - In Zustand v ist Spieler odd am Zug und alle Nachfolger v' von v waren nicht bereits im letzten Schritt als verloren oder (3) markiert war. Alle Nachfolger v' von v hat eine Farbe c' mit der geraden Zahl.

(4) wonZykleSchrittByOppMove^i+1 (v) = Odd (v) ∧ ∀v´(R(v, v') → ∃c' (Color(v', c') ∧ EvenNumber (c') ∧ (¬ lost^i (v') ∨ wonZykleAnfangByOwnMove^i (v'))

Force set für verlierende Bedingung ist ähnlich wie Gewinnbedingung definiert deswegen sehen wir nun, wie sich durch zwei Algorithmen, die wir gerade kennengelernt haben, schwache Spiele lösen lassen. Die erste und zweite Schleife sind analog wie im Algorithmus 1 definiert. Mit der dritten while-Schleife wählt Spieler seinen optimalen Zug im wonByForceSet.

```
function SolveWeakGame ():
    represent game symbolically
    initialize won and lost with ⊥
    repeat until won^i+1 = won^i and lost^i+1 = lost^i
        repeat until fixpoint
            won^j+1 ← won^j ∨ wonByOwnMove^j+1 ∨ wonByOppMove^j+1
            lost^j+1 ← lost^j ∨ lostByOwnMove^j+1 ∨ lostByOppMove^j+1
        end
        ☺
            initialize zwon and zlost with ⊥
            repeat until foxpoint
            zwon^k+1 ← zwon^k ∨ wonZykleSchrittByOwnMove^k+1 ∨
                                    wonZykleSchrittByOppMove^k+1
            zlost^k+1 ← zlost^k ∨ loseZykleSchrittByOwnMove^k+1 ∨
                                    loseZykleSchrittByOppMove^k+1
            end
        won^i+1 ← zwon^i
        lost^i+1 ← zlost^i
        won^i+1 ← zwon^i ∨ wonByForceSet^i+1
        lost^i+1 ← zlost^i ∨ lostByForceSet^i+1
    end
    won^i+1
    lost^i+1
    return won(v0)
end
```

4.5 Beispiele

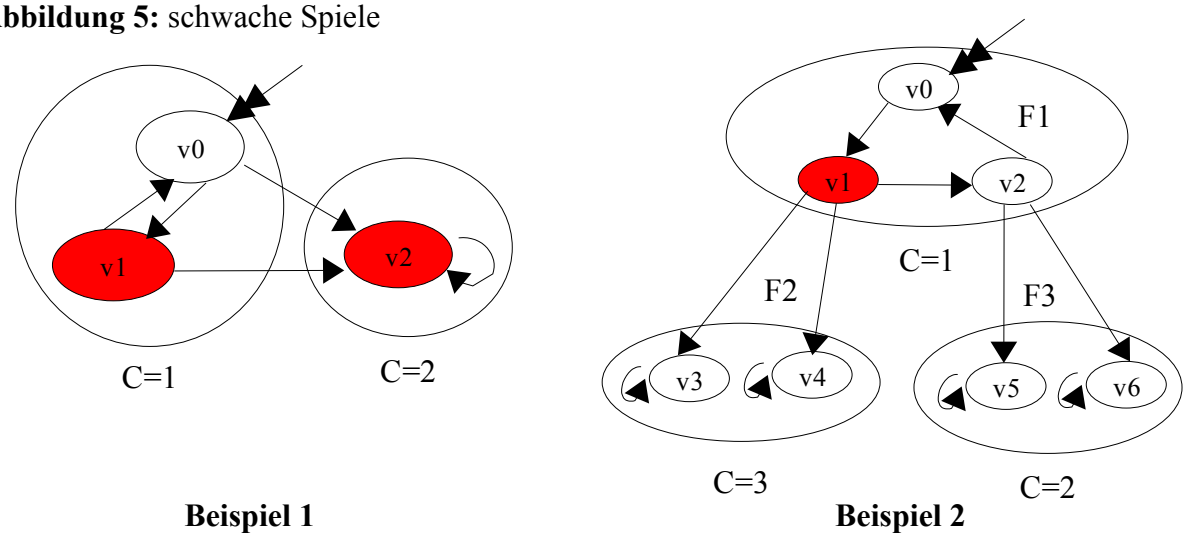
4.5.1 Aufgabenbeschreibung

Die folgende Abbildung 5 zeigt zwei verschiedene Beispiele. Das Beispiel 1 ist uns schon bekannt. Die Aufgabebeschreibung vom Beispiel 1 hatten wir schon im Abschnitt 3.3.2. Deswegen sehen wir hier nur das Beispiel 2 genauer. Spieler even ist nur bei dem Zustand v1 am Zug. Da die Anzahl von Knoten sieben und die maximale Farbe C = 3 ist, brauchen wir drei Bits für die Zustände und zwei Bits für die Farbe, also N = 7, n = $\lceil \log_2(7) \rceil$ = 3 und M = 4, m = $\lceil \log_2(4) \rceil$ = 2. Wir kodieren

dann jeden Zustand v mit drei Bits, also v = (x2,x1,x0). Die Kodierung von allen Zuständen in Beispiel 2 sind folgendes: v0 = (0,0,0), v1 = (0,0,1), v2 = (0,1,0), v3 = (1,0,0), v4 = (1,0,1), v5 = (1,1,0) und v6 = (1,1,1).

- V = {v0, v1, v2, v3, v4, v5, v6, v7}, wobei V_{even} = {v1} und V_{odd} = V \ {v1}, d.h. owner (v1) = even und owner (vi) = odd, i = 0,2,3,4,5,6,7.
- v0 ∈V ist der Anfangszustand
- E = {(v0, v1), (v1, v2), (v1, v3), (v1, v4), (v2, v0), (v2, v5), (v2, v6), (v3, v3), (v4, v4), (v5, v5), (v6, v6)}
- α : V → ℕ ist eine Farbfunktion mit α(v0) = α(v1) = α(v2) = 1, α(v5) = α(v6) = 2 und α(v3) = α(v4) = 3, d.h. level (v5) = level (v6) = even und level (v_i) = odd, i = 0,1,2,3,4.

Abbildung 5: schwache Spiele



4.5.2 Ergebnis und Analyse

Das erwartete Ergebnis von Beispiel 1 ist, dass Spieler even gewinnt. Bei dem Anfangsknoten v0 ist Spieler odd am Zug. Wenn er seinen Nachfolger v2 wählt, verliert er sofort. Also, er wählt den Nachfolger v1. Dann ist Spieler even bei dem Knoten v1 am Zug und wählt seinen Nachfolger v2, so dass er gewinnt. Für beide Fälle hat Spieler odd keine Gelegenheit zu gewinnen. Somit gewinnt Spieler even dieses Spiel. Wie sieht es dann mit dem Beispiel 2 ?

Wir erwarten, dass Spieler odd dies Mal gewinnt. Bei dem Anfangsknoten v0 ist Spieler odd am Zug und hat nur eine Möglichkeit, den Nachfolger v1 zu wählen. Bei dem Knoten v1 ist Spieler even am Zug und hat drei Möglichkeiten, seine Nachfolger v2, v3 oder v4 zu ziehen. Eine Möglichkeit besteht darin, dass er v3 oder v4 wählt. Da die beiden Knoten zu einem Force Set F2 gehören und F2 die ungerade Farbe C = 3 hat, verliert Spieler even sofort, wenn er einen von beiden Nachfolger wählt. Also, er wählt besser den Nachfolger v2. Bei dem Knoten v2 ist Spieler odd wieder am Zug. Er hat drei Möglichkeiten, seine Nachfolger v0, v5 oder v6 zu ziehen. Er hat gleiche Situation, wie Spieler even bei dem Knoten v1 hatte. Da er keinen Nachfolger von einem Force Set F3 wählt, bleibt bei ihm nur eine Wahl, den Nachfolger v0 zu ziehen. Dann sind wir wieder am Anfang und haben beide Spieler wieder gleiche Situationen bzw. Aktionen. Hier besteht

somit einen Zyklus. Da die beide Spieler in diesem Zyklus bleiben, sind sie in einem Force Set F1 und F1 hat ungerade Farbe C = 1. Also, Spieler odd gewinnt tatsächlich dies Mal, wie wir erwartet haben. Nun sehen wir, was unsere Algorithmen ausgeben. Als Ausgabe habe ich angegeben, durch welche Methode sich jeder Knoten markieren lässt und wer das Spiel gewinnt. Die erste zwei Ergebnisse sind vom Beispiel 1 und die letzte zwei vom Beispiel 2. Beachte hier, dass die Markierungen für Spieler even sind. D.h. falls ein Knoten v als „gewonnen“ bzw. „verloren“ markiert ist, gewinnt bzw. verliert Spieler even bei diesem Knoten v.

Ergebnisse

Beispiel 1 mit 1.Algorithmus:

wonByOwnMove = {v1}	loseByOwnMove = Ø
wonByOppMove = {v0}	loseByOppMove = Ø
wonByForceSet = {v2}	loseByForceSet = Ø

Ergebnis = Even gewinnt

Beispiel 1 mit 2.Algorithmus:

wonByOwnMove = {v1, v2}	loseByOwnMove = Ø
wonByoppMove = {v0}	loseByOppMove = Ø
wonByForceSet = Ø	loseByForceSet = Ø

Ergebnis = Even gewinnt

Beispiel 2 mit 1.Algorithmus:

wonByOwnMove = Ø	loseByOwnMove = Ø
wonByOppMove = Ø	loseByOppMove = Ø
wonByForceSet = {v5, v6}	loseByForceSet = {v0, v1, v2, v3, v4}

Ergebnis = Odd gewinnt

Beispiel 2 mit 2.Algorithmus:

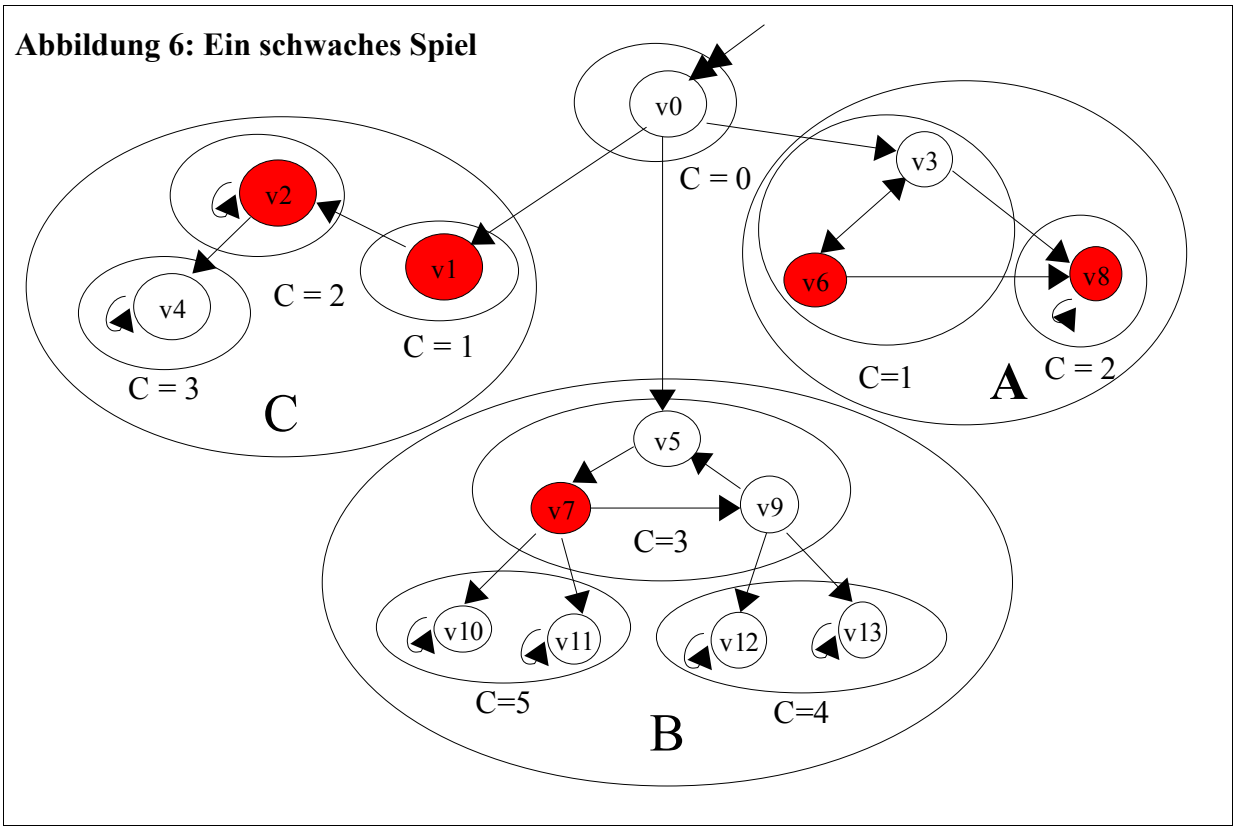
wonByOwnMove = Ø	loseByOwnMove = {v1}
wonByoppMove = {v5, v6}	loseByOppMove = {v0, v3, v4}
wonByForceSet = Ø	loseByForceSet = {v2}

Ergebnis = Odd gewinnt

5. Experiment

Wir haben im Abschnitt 3 gesehen, wie man schwache Spiele definieren kann. Danach haben wir im Abschnitt 4 zwei verschiedene Algorithmen kennengelernt, wie sich schwache Spiele lösen lassen. Dabei spielt besonders große Rolle, wie sich Force Set in jeweiligem Algorithmus definieren lässt. Wir haben mit zwei kleinen Beispielen getestet, was unser Algorithmus jeweils berechnet. Als Ausgabe habe ich gewählt, welcher Spieler das gegebene schwache Spiel gewinnt und durch welche Methode sich jeder Knoten markieren lässt. Falls der Anfangsknoten als „gewonnen“ markiert ist, bedeutet das, dass Spieler even das Spiel gewinnt. Sonst gewinnt Spieler odd. Die beiden Algorithmen haben ausgegeben, was wir als Ergebnis erwartet haben. In diesem Abschnitt 5 gebe ich ein großes Beispiel als Experiment an. Mit dem in Abbildung 6 dargestellten Spielgraph möchten wir beschäftigen. Wenn wir genau den Spielgraph sehen, bemerken wir, dass er aus drei sogenannte Teilspiele A, B und C besteht. Zwei Teilspiele A und B haben wir schon im Abschnitt 4 als kleine Beispiele getestet und wissen jeweils, dass Spieler even im Teilspiel A und Spieler odd im Teilspiel B gewinnt. Wir wollen hier die Ergebnisse nutzen, um unseres Beispiel einfacher berechnen zu können. D.h. wir geben einen neuen Spielgraph an, der das gleiche Ergebnis von Abbildung 6 hat. Natürlich kommt dann die Frage, ob es solche Automatisierung gibt. D.h. Wie sich ein großes Spiel teilen lässt, wie kann man die geteilte Teilspiele erkennen und wie kann man einen neuen durch die Ergebnisse von Teilspielen rekonstruierten vereinfachten Spiel gewinnen ? Das ist als eine weitere Forschung interessant aber wir wollen hier nur wissen, wer das Spiel gewinnt. Also, wir nutzen die Ergebnis von Teilspiel A und B. Als Ergebnis gebe ich wieder alle Methoden durch die sich jeder Knoten markieren lässt und dadurch markierte Knoten wie im Abschnitt 4.5.2 an und wer das Spiel gewinnt.

Abbildung 6: Ein schwaches Spiel



5.1 Aufgabenbeschreibung

Gegeben sei folgender Spielgraph. Wir definieren das Spiel durch:

- V = {v0,.....,v13}, wobei V_{even} = {v1, v2, v6, v7, v8} und V_{odd} = V \ {v1, v2, v6, v7, v8}, d.h. owner (v) = even, v∈ V_{even} und owner (w) = odd, w∈ V_{odd}.
- v0 ∈V ist der Anfangsknoten und owner (v0) = odd.
- E = {(v0, v1), (v0, v3), (v0, v5), (v1, v2), (v2, v2), (v2, v4), (v3, v6), (v3, v8), (v4, v4), (v5, v7), (v6, v8), (v7, v9), (v7, v10), (v7, v11), (v8, v8), (v9, v5), (v9, v12), (v9, v13), (v10, v10), (v11, v11), (v12, v12), (v13, v13)}
- α : V → ℕ ist eine Farbfunktion mit α(v0) = 0, α(v1) = α(v3) = α(v6) = 1, α(v2) = α(v8) = 2, α(v4) = α(v5) = α(v7) = α(v9) = 3, α(v12) = α(v13) = 4 und α(v10) = α(v11) = 5, also level (vi) = even, i = 0,2,8,12,13 und level (vj) = odd, j = 1,3,4,5,6,7,9,10,11.

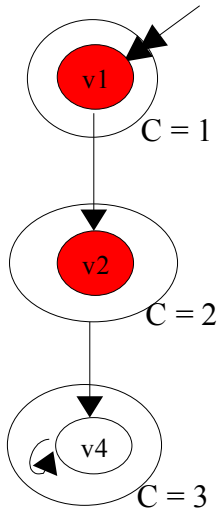
Es gilt N = 14 und M = 6 in Abbildung 6, also n = ⌈ $\log_2(14)$ ⌉ = 4 Bits für die Knoten v0,.....,v13 und m = ⌈ $\log_2(6)$ ⌉ = 3 Bits für die Farbe. Jeder Knoten lässt sich durch v = (v3, v2, v1, v0) und jede Farbe durch c = (c2, c1, c0) kodieren. Wir versuchen das schwache Spiel nicht direkt zu lösen. D.h. wir nutzen die Ergebnisse von Teilspiele A und B, die wir schon im Abschnitt 4.5.2 gesehen haben, so dass sich das gegebene schwache Spiel vereinfachen lässt. Mit dem Wort „vereinfachen“ meine ich, dass sich ein schwaches Spiel mit wenigen Zuständen bzw. Knoten und somit wenige Bits Kodierung darstellen lässt. Beachte hier auch, wenn sich ein Knoten als „gewonnen“ bzw. „verloren“ markieren lässt, meine ich für Spieler even. Aus den Ergebnissen von A und B lassen sich die Knoten v3 als „gewonnen“ und v5 als „verloren“ markieren. Im nächsten Abschnitt 5.2 analysieren wir zuerst den Teilspiel C und mit den Ergebnissen von A, B und C gebe ich das vereinfachte Spielgraph und wir lösen das vereinfachte Spiel statt das große, um nur das Ergebnis zu wissen, wer das Spiel von Abbildung 6 gewinnt.

5.2 Ergebnis und Analyse

Wir wollen zuerst das Teilspiel C analysieren, ob sich der Wurzelknoten v1 als „gewonnen“ oder „verloren“ markieren lässt. Die Abbildung 7 zeigt das Teilspiel C von Abbildung 6 und sein Ergebnis. Ich habe hier unseren zweiten Algorithmus angewendet. Das Teilspiel C ist folgendes definiert:

- owner (v1) = owner (v2) = even und owner (v4) = odd.
- v1 ist der Anfangsknoten.
- E = {(v1, v2), (v2, v2), (v2, v4), (v4, v4)}
- α(v1) = 1, α(v2) = 2 und α(v4) = 3, also level (v1) = level (v4) = odd und level (v2) = even.

Abbildung 7: Das Teilspiel C und Ergebnis



wonByOwnMove = {v1}
wonByoppMove = Ø
wonByForceSet = {v2}

loseByOwnMove = Ø
loseByOppMove = {v4}
loseByForceSet = Ø

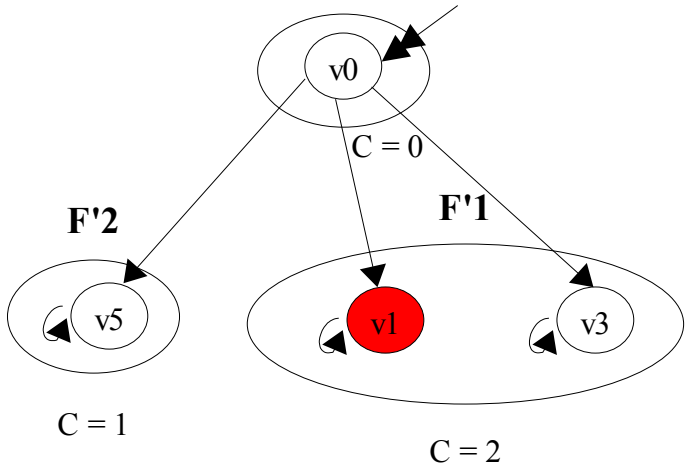
Ergebnis = Even gewinnt

Man sieht sofort, dass Spieler even beim Knoten v2 am Zug ist und in Force Set mit der geraden Farbe C = 2 immer bleiben kann, da es einen Zyklus existiert. Somit gewinnt Spieler even im Teilspiel C. Also, der Knoten v1 ist als „gewonnen“ markiert. Unser erwartete Ergebnis ist, dass Spieler even gewinnt. Aus dem Ergebnis gewinnt Spieler even tatsächlich das Spiel bzw. Teilspiel C. Nun haben wir alle Ergebnisse der Teilspiele A, B und C. Die Knoten v3 im Teilspiel A und v1 im Teilspiel C sind als „gewonnen“ markiert. Wir definieren einen neuen Force Set F'1, der v1 und v3 enthält und gerade Farbe C = 2 hat. Der Knoten v5 im Teilspiel B ist als „verloren“ markiert. Wir geben wieder einen anderen neuen Force Set F'2, der v5 enthält und ungerade Farbe C = 1 hat. Jetzt können wir das schwache Spiel von Abbildung 6 in einfacher Form definieren.

- V = {v0, v1, v3, v5}, wobei V_{even} = {v1} und V_{odd} = V \ {v1}, d.h. owner (v) = even, v ∈ V_{even} und owner (w) = odd, w ∈ V_{odd}.
- v0 ∈V ist der Anfangsknoten und owner (v0) = odd.
- E = {(v0, v1), (v0, v3), (v0, v5), (v1, v1), (v3, v3), (v5, v5)}
- α : V → ℕ ist eine Farbfunktion mit α(v0) = 0, α(v1) = α(v3) = 2, α(v5) = 1.
- v1, v3 ∈F'1 und v5 ∈F'2 sind die neu definierte Force Sets.

Die Abbildung 8 zeigt den vereinfachten Spielgraphen. Mit dem vereinfachen Spielgraph haben wir N = 4 und M = 3. Also, wir brauchen n = ⌈ $\log_2(4)$ ⌉ = 2 Bits für die Knoten und m = ⌈ $\log_2(3)$ ⌉ = 2 Bits für die Farbe. Wir haben 2 Btis weniger im Vergleich mit der Kodierung v = (v3, v2, v1, v0) in Abbildung 6 für die Knoten und 1 Bits weniger für die Farbe. Aus der Abbildung 8 sieht man sofort, dass Spieler odd das vereinfachte Spiel bzw. unseres schwaches Spiel gewinnt. Am Anfang ist Spieler odd am Zug und er wählt den Nachfolger v5. Da der Knoten v5 in einem Force Set mit der ungeraden Farbe liegt, ist v5 als „verloren“ markiert. Also, Spieler odd gewinnt. Beachte wieder , dass die Markierung „gewonnen“ bzw. „verloren“ für Spieler even ist. Nun, wir testen mit unserem Algorithmus, ob unser erwartete Ergebnis richtig ist. Dies Mal wird 1.Algorithmus verwendet. Das Ergebnis zeigt uns, dass Spieler odd tatsächlich das Spiel gewinnt, wie wir erwartet haben.

Abbildung 8: Das vereinfachte schwache Spiel



Ergebnis

WonByOwnMove = Ø	loseByOwnMove = Ø
wonByOppMove = Ø	loseByOppMove = {v0}
wonByForceSet = {v1, v3}	loseByForceSet = {v5}

Ergebnis = Odd gewinnt

6 Zusammenfassung und Ausblick

In meiner studienarbeit habe ich Algorithem für Lösen von schwachen Spielen repräsentiert. D.h. wir haben bei jedem Zustand repräsentiert, wer am Zug ist, welche Aktionen existieren und ob sich jeder Zustand als „gewonnen“ oder „verloren“ markieren lässt. Dabei haben wir betrachtet, dass die Markierung für Spieler even ist. Zuerst haben wir kennengelernt, was schwache Spiele sind und wie sie sich definieren lassen. Als eine wichtige Eigenschaft von den schwachen Spielen wäre ich zu nennen, dass es zwei Spieler even und odd gibt. Bei allen Spielen ist es als grundlegende Regel wichtig, die Anzahl der Teilnehmer an einem Spiel klar zu stellen. Eine weitere wichtige Eigenschaft von den schwachen Spielen ist Force Set. Wir haben einen Force Set F von Spieler p durch Farbfunktion als Menge von Knoten im gleichen Niveau (level (F) = {p}) definiert. Die Farben von Force sets sind mit Monotonie-Eigenschaft gegeben.

Um schwache Spiele zu repräsentieren, habe ich BDDs verwendet. Im Abschnitt 2 haben wir BDDs mit der kanonischen Repräsentation kennengelernt. BDDs sind vor allem kompakte Darstellungsmöglichkeit für Mengen von Belegungen boolescher Variablen, die bei geschickter Implementierung leistungsfähige Funktionen zur Verfügung stellen und kanonische BDDs sind die reduzierten minimal geordneten BDDs. Im Abschnitt 4 haben wir einen Algorithmus kennengelernt, die schwache Spiele lösen. Ein Spiel zu lösen bedeutet die Bestimmung davon, welcher Spieler es gewinnt. Für die Gewinnbedingung von Spieler even haben wir durch Iterationen die noch nicht als

„gewonnen“ oder „verloren“ markierten Knoten mit Hilfe von unseren drei Hauptmethoden markiert, die wonByOwnMove, wonByOppMove und wonByForceSet sind. Wir haben danach versucht, einen alternativen Algorithmus durch „Zyklus“ zu definieren. Dabei haben wir einen wichtigen Satz kennengelernt. Der Unterschied zwischen dem 1.Algorithmus im Abschnitt 4.3 und 2.Algorithmus im Abschnitt 4.4 ist, wie sich Force Set in jeweiligem Algorithmus definieren lässt. Die Hauptidee von Force Set bei dem ersten Algorithmus besteht darin, dass sich jeder Nachfolger des aktuellen Knotens, der noch nicht als „gewonnen“ bzw. „verloren“ markiert ist, entweder schon als „gewonnen“ oder „verloren“ markieren lässt, oder er gehört zu einem anderen Force Set. Dabei spielt die Farbe von Force Set eine große Rolle, da sich die verschiedenen Force Sets durch verschiedene Farbe unterscheiden lassen. Die Haputidee von Force Set bei dem zweiten Algorithmus ist, dass man mit den unentschiedenen Knoten einen Zyklus definieren kann. Das habe ich als ein Satz im Abschnitt 4.4.1 bewiesen. Mit den verschiedenen Beispielen habe wir gesehen, durch welche Methode sich jeder Knoten markieren lässt und welche Knoten schließlich als „gewonnen“ oder „verloren“ markieren lassen. Die Entscheidung, wer das Spiel gewinnt, haben wir mit dem Anfangsknoten getestet, ob er sich als „gewonnen“ bzw. „verloren“ markieren lässt. Wir müssen nicht vergessen, dass die Aussage bzw. die Markierung „gewonnen“ bzw. „verloren“ für Spieler even ist. Als Experiment haben wir ein großes Beispiel behandelt. Was passiert, wenn man ein großes schwaches Spiel hat ? Wir haben das große schwache Spiel in kleinen Teilspielen geteilt und zuerst sie gelöst. Durch ihre Ergebnisse haben wir das große schwache Spiel in einfacher Form rekonstruiert und das vereinfachte Spiel gelöst. Dabei gab es paar Fragen, ob man überhaupt so was automatisieren kann. Da wir aber nur das Ergebnis des schwachen Spiels wissen wollten, haben wir einfach es in einfacher Form rekonstruiert und gelöst.

Zum Schluss möchte ich kurz über weitere interessanten Forschungen schreiben. Wir haben ganze Zeit in meiner Studienarbeit mit Schwache Spiele beschäftigt. Die Frage, die wir bei Experiment hatten, zu antworten kann eine interessante Forschung sein. Was passiert dann, wenn wir z.B. die Regel von der Anzahl der Spieler nicht zwei sondern drei ersetzen ? Kann man solche schwache Spiele definieren und lösen ? Wie sieht die Gewinn- bzw. verlierende Bedingung aus ? Solche Erweiterung zu forschen wäre eine interessante Forschung. Wann ist sie lösbar und wann nicht ? Wie sehen schwache Spiele in Komplexitätstheorie ? Solche Frage zu antworten könnte auch eine interessante Forschung sein.

7 Literaturverzeichnis

[1] Erich Grädel, Wolfgang Thomas und Thomas Wilke: Automata, Logics, and Infinite Games. Lecture Notes in Computer Science 2500 Springer 2002.

[2] Martin Osborne: An Introduction to Game Theory. Oxford University Press, 2004.

[3] Johan Huizinga. Homo Ludens. 1938/1991, S.37

[4] Ernst A.Heinz. Endgame Databases and efficient Index Schemes. In ICCA Journal, Vol22, No.1, pages 22-32, March 1999.

[5] J.Schaeffer, Y.Björnsson, N.Burch, A.Kishimoto, M.Müller, R.Lake, P.Lu und S.Sutphen. Solving Checkers. Department of Computing Science. In International Joint Conference on Artificial Intelligence (IJCAI), pages 292-297, 2005.

[6] J.Schaeffer. The Games Computers (and People) Play. " Academic Press, Vol. 50 (ed Zelkowitz M.V.), pp 189-266, 2000

[7] M.Helmert, R.Mattmüller und S.Schewe. Selective Approaches for Solving Weak Game, , in: Proceedings of the Fourth International Symposium on Automated Technology for Verification and Analysis (ATVA 2006), pp. 200-214, 2006.

[8] M.Helmert und S.Schewe. Heuristic Game Solving. AVACS S1 Workshop, Saarbrücken.

[9] S.Edelkamp. Symbolic Exploration in Two-Player Games: Preliminary Results, In Proceedings of the International Conference on AI Planning and Scheduling (AIPS'02) Workshop on Model Checking, 2002.