Ao.Univ.Prof. Dipl.-Ing. Dr.techn. Alois Panholzer

D I P L O M A R B E I T

Komplexität und Varianten von Minesweeper

ausgeführt am Institut für
Diskrete Mathematik und Geometrie
der Technischen Universität Wien

unter Anleitung von
Ao.Univ.Prof. Dipl.-Ing. Dr.techn. Alois Panholzer

durch
Martin Heinrich
Bräuhausgasse 64/9
1050 Wien

Wien, am 8. Mai 2006

Martin Heinrich

Abstract

This diploma thesis deals with *"Complexity and Variants of Minesweeper"*, the well-known computer game distributed with the Microsoft Windows[1] operating system. We use MW as an abbreviation for Minesweeper rather than MS, because the latter is usually associated with Microsoft.

The work is based on Richard Kaye's proof ([8]) of *MW's NP-completeness*. We start our chain of proof by introducing a definition of the Turing Machine and reviewing its mathematical basis. We find good explicit bounds for Kaye's reduction from SAT to the Minesweeper-(Consistency-)Problem (MWP) by using a slightly different approach:

Given a Boolean Formula F in Conjunctive Normal Form (CNF) containing c literals using s variables, we construct a MW board whose size must not exceed $(4c-1) \times (3s+7)$ and which is consistent if and only if F is satisfiable.

We also find a *regular expression* for one-dimensional MW-strings and prove that a *context-sensitive grammar* exists for MWP in any dimensional variant.

Minesweeper on graphs (MWG) is a generalization of the idea of Minesweeper applied to finite directed multigraphs. They offer an *arbitrary, weighted neighborhood-function* for some set of nodes, containing all variants of Minesweeper, including MW on spheres and any discrete approximation of manifolds or fractals. We show that MWG is in NP as well—thus *not harder* than MWP.

Next we try to find limits for simple and already hard MWG-instances by exploring *categories of graphs* and we can prove that graphs with bounded vertex-degree $d \leq 2$ are simple and that a context-free grammar for MW on trees exists. Minesweeper is hard—in the sense of NP-Completeness—for *simple, locally starlike*[2] graphs.

We can also reduce the problems SAT, "Vertex Cover", "Clique", "three-dimensional Matching" and non-negative "Knapsack" to MWG. These reductions prove that we do not lose NP-completeness of MWG when allowing one of the following sets of restrictions in addition to the properties mentioned above:

- plane, in-degrees $d^- \leq 3$, out-degrees $d^+ \leq 4$, values displayed on nodes $\kappa \leq 2$
- in-degrees $d^- \leq 3$, values displayed on nodes $\kappa \equiv 1$

It is unclear if bounding vertex degrees with $d \leq 3$ causes simple instances. This question could depend on the famous P versus NP Problem ([4]).

[1] *Microsoft* and *Windows* are registered trademarks. No statement within any scientific work on Minesweeper should be interpreted as a comment on any Microsoft product or the company itself.

[2] *Locally starlike* means that every vertex is starlike which means that it is either the source or the target of all its edges.

Inhaltsverzeichnis

Vorwort

Bereits sehr früh im Verlauf meines Studiums wusste ich, dass mein Diplomarbeitsthema aus dem Bereich der diskreten Mathematik sein soll. *Algorithmen* wecken dabei mein ganz besonderes Interesse. Obwohl ich nie anfällig auf den Suchtfaktor Computerspiel war, gab es immer wieder Zeiten, in denen ich sogar die (Denk-)Spiele, die standardmäßig mit dem Betriebssystem installiert werden, von meinem Rechner entfernt habe um meinen Arbeitserfolg nicht zu gefährden.

Minesweeper ist so ein Denkspiel und hat mich fasziniert, weil scheinbar kleine Entscheidungen über die Position einer fehlenden Mine große Auswirkungen auf weitere Felder haben kann. Mein erster Gedanke zur Analyse war die Frage: Wie viel muss ein Computer wissen um Minesweeper spielen zu können? – Eigentlich doch „nur“ wie *wahrscheinlich* es ist, dass ein Feld vermint ist.

Es stellte sich heraus, dass Minesweeper bereits wissenschaftlich untersucht wurde. Insbesondere war für mich der Ansatz von Richard Kaye von Interesse, unabhängig von tatsächlich auftretenden Bildern, die *Konsistenz* allgemeiner Minesweeper-Konfigurationen zu überprüfen. In seinem Artikel ([8]), in dem er die NP-Vollständigkeit dieses Problems beweist, rückt er mit dem Nebensatz, Wahrscheinlichkeiten seien nicht einfacher als Konsistenz, eine Lösung zu meinem Ansatz weit in die Ferne.

Der Beweis selbst ist sehr schön und im Überblick einfach zu verstehen, aber – und das bitte keineswegs negativ zu verstehen – in seiner Art ein *Existenzbeweis*, der außerdem Elemente enthält, deren Beschreibung klar, aber deren Herleitung nicht offensichtlich ist. Der Anspruch an die vorliegende Arbeit war nun, den Beweis für das konkrete Computermodell Turing-Maschine lückenlos nachzuzeichnen.

Mit dem Hintergrund der verschiedenen Arbeiten zum Thema Minesweeper, die mir genügend Inhalt liefern können, ersuchte ich Alois Panholzer um die Betreuung der Arbeit und beschloss den Arbeitstitel *„Komplexität und weitere Aspekte von Minesweeper“*. Diese weiteren Aspekte traten durch erfolgreiche Überlegungen, die Einschränkung eines rechteckigen Spielfeldes fallen zu lassen, immer weiter in den Hintergrund.

Abgesehen von dem NP-Vollständigkeitsbeweis nach Kaye mit guten expliziten Schranken für die Reduktion des Erfüllbarkeitsproblems auf Minesweeper enthält die Diplomarbeit auch sprachtheoretische Betrachtungen des Problems und eine Analyse von Minesweeper auf endlichen *allgemeinen Graphen*, sowie einen sehr kurzen Überblick über weitere Aspekte. Der Inhalt ist in einem Abstract (siehe Seite ii) und auf deutsch etwas ausführlicher am Ende der Arbeit (Abschnitt 4.2 auf Seite 85) zusammengefasst.

Ein Wort zur Literatur...

Die im Literaturverzeichnis angeführten Titel sind nach dem Nachnamen des (ersten angegebenen) Autors sortiert. Es umfasst einerseits *Lehrbücher* zu den benötigten Fachgebieten und andererseits einige *Artikel* und *Berichte* von wissenschaftlichen Arbeiten

zum Thema Minesweeper. Bei letzteren habe ich versucht, möglichst viele Informationen zur Verfügung zu stellen – vor allem, wenn sie nicht erschienen sind.

Internet-Links sind vom *Stand Anfang April 2006*.

Die Vielzahl junger Berichte soll auch einen Anhaltspunkt für Recherchen zum Umfeld der vorliegenden Arbeit geben ohne eine vollständige Aufzählung garantieren zu können. Es fällt auf, dass die Erscheinungsjahre um 1980 und nach 2000 liegen: Minesweeper ist eine neue Anwendung oder Zugang zu einem relativ alten und gut erforschten Bereich der Mathematik.

Die Wahl der Lehrbücher kam hauptsächlich durch deren Verfügbarkeit in den Bibliotheken zustande. Diestels *Graphentheorie* ([5]) gibt einen guten Einstieg zum Thema, der umfangreicher als für die vorliegende Arbeit notwendig ist, aber hauptsächlich ungerichtete Graphen behandelt. Es gibt bereits mindestens zwei neuere Auflagen als die angegebene. Zur *Komplexitäts-* und *Sprachtheorie* bietet sich [16] als gut zu lesender Überblick an. Die Werke [6] und [10] zeichnen sich durch vollständige Beweise aus.

Mein Hauptwerk zum Einstieg in die Komplexitätstheorie war „Data Structures and Algorithms 2“ von Kurt Mehlhorn ([11]) und davon besonders der zweite Band „Graph Algorithms and NP-Completeness“ und das Kapitel VI auf den Seiten 171–208. Es enthält nahezu alle Beweise zu den Turing-Maschinen und angeführten Problemen.

. . . und zu meinen Helfern

Mein besonderer Dank gilt *Alois Panholzer* für die unkomplizierte Betreuung. Er hat mir viel Freiraum in meiner Arbeitsweise und trotzdem die notwendige Unterstützung gegeben und war auch dann noch geduldig und nachsichtig, als meine gewünschte Einreichfrist sehr nahe kam.

Michael Drmota konnte für mich klären, dass das von mir gewählte Thema noch nicht in dieser Form an einer Wiener Universität behandelt wurde, und bot sich ebenfalls für die Betreuung an. *Martin Goldstern* ermöglichte mir über Kaye's Beweis im Rahmen des Forschungsseminars der Algebra-Gruppe zu referieren. Bei diesem Vortrag erkannte er auch das Auffüllen eines Minesweeper-Spielfelds mit Minen als Konzept zur Isolation von Information (siehe Bemerkung 2.19). *Günther Eigenthaler* machte mich bei dieser Gelegenheit auf einen Fehler in der Metrik bei der Modellierung des Standardspiels aufmerksam, und *Wolfgang Eppenschwandtner* schlug vor, das Konzept der relativen Explosivität zu formalisieren (siehe Definition 2.17).

Ebenso hilfreich waren die *Kollegen und Freunde*, die die Arbeit Korrektur gelesen haben. Ihre Mühe hat jedenfalls zu einer Verbesserung vor allem des englischen Abstracts geführt, trotzdem bin ich für verbleibende Fehler alleine verantwortlich.

Besonders gefreut hat mich die kurze aber fruchtbare E-Mail-Korrespondenz mit *Richard Kaye*. Er hat mich dazu animiert auch andere als das Erfüllbarkeitsproblem auf Minesweeper zu reduzieren.

Ihnen allen, und jenen, die mich während des ganzen Studiums unterstützten, ein herzliches Dankeschön!

Wien, am 5. Mai 2006 Martin Heinrich

Voraussetzungen und Notationen

Wie bereits im Vorwort erwähnt, sollte mathematisches Grundwissen für das Verständnis der Arbeit ausreichen. Vor allem wird die Kenntnis von Mengen, Relationen und Funktionen, sowie deren Notationen und Verwendungen vorausgesetzt. Vorwissen über Graphen ist von Vorteil für das dritte Kapitel.

$\mathbf{N}$ bezeichnet die natürlichen Zahlen (mit 0), die wiederum (auch) als Menge ihrer Vorgänger aufgefasst werden können: $\forall k \in \mathbf{N} : k = \{l \in \mathbf{N} : l < k\} = \{0, \ldots, k-1\}$.

$\mathbf{Z}$ und $\mathbf{R}$ stehen für die ganzen und reellen Zahlen; $\mathbf{P}(X)$ für die Potenzmenge von X; B^D für die Funktionen von D nach B; $R[x]$ für die Polynome über x mit Koeffizienten aus R.

Wir wollen kurze Schreibweisen zulassen, falls sie klar sind. Zum Beispiel sei einerseits mit einer Funktion $f : D \to B$ auch gleichzeitig ihre Wirkung auf Teilmengen der Definitionsmenge $f : \mathbf{P}(D) \to \mathbf{P}(B)$ mit $f(X) = \{f(x) : x \in X\}$ erklärt. Andererseits schreiben wir für ein Funktionsargument $f(\{x\})$, das eine einelementige Menge ist, auch abkürzend $f(x)$.

Satz 0.1 (Beispielsatz) *Definitionen, Erkenntnisse und Bemerkungen sind mit der selben Nummerierung versehen. Sätze und Lemmata sind im Anschluss bewiesen, wobei auf zuvor geführte Argumentationen gegebenenfalls hingewiesen wird.*

Beweis 0.1 Diese Arbeit ist eine endliche Aneinanderreihung von Zeichen. Ein fachkundiger Mensch kann die Aussage in linearer Zeit überprüfen. □

Außerdem behaupten wir noch, wir hätten die Abkürzung MW für Minesweeper absichtlich gewählt, weil MS bereits mehrfach belegt ist und vor allem im Zusammenhang mit Computern für Microsoft steht.

O-Notation

Sei $f : D \to B$ eine Funktion mit Bildern in (einer Teilmenge von) $\mathbf{N}$ oder $\mathbf{R}$, so bezeichne $O(f)$ die Menge $O(f) = \left\{g \in B^D : \exists c \in B : g < cf \text{ auf fast ganz } D\right\}$. D ist meistens $\mathbf{N}$ oder $\mathbf{N}^k$ für ein $k \in \mathbf{N}$. Wir erweitern *keine* Operatoren auf diese Mengen und verwenden daher normale Mengenschreibweisen $(\in, \subset, \subseteq, \ldots)$ mit dieser Notation. Analog bezeichne $\Omega(f) = \left\{g \in B^D : \exists c \in B : g > cf \text{ auf fast ganz } D\right\}$ die Menge der entsprechenden von unten beschränkten Funktionen. Die Verwendung ist intuitiv eindeutig.

Projektionen

Mit $\pi_i(x)$ bezeichnen wir für $i \in I$ aus einer Indexmenge die i-te Projektion von $x \in X = \Pi_{i \in I} X_i$ einem Tupel in die entsprechende Menge X_i. Dem entsprechend sei auch für eine Funktion $f : D \to X = \Pi_{i \in I} X_i$ die Projektion $\pi_i \circ f : D \to X_i$ und für eine Zeichenfolge $x \in X^*$ und $i \in \mathbf{N}^+$ das i-te Zeichen $\pi_i(x)$ definiert.

1 Komplexität

Dieses Kapitel soll eine Einführung zu dem Begriff Zeitkomplexität geben. Bereits ein kleiner Blick auf die Kapazitäten des menschlichen Gehirns lässt uns erkennen, dass Umfang und Vorgehensweise eines solchen Modells die Analyse der Wege zur Problemlösung unmöglich macht. Das Thema dieses Kapitels kann also nur *automatisiertes* Lösen von Problemen sein.

Wir definieren die 1936 von A. Turing eingeführte Turing-Maschine als allgemeinen Computer. Danach können wir die wichtigen Problemklassen P, NP und NP-vollständig betrachten. Außerdem werden wir SAT als erstes Problem der Klasse NP-vollständig vorstellen und gleichzeitig Techniken erarbeiten, die uns einfachere Argumentationen im Bereich der Komplexität ermöglichen.

1.1 Turing-Maschine

1.1.1 Grundlagen

Dieser einfache Computer besteht aus einem rechtsseitig endlosen Band mit einem Schreib-/Lesekopf für Ein- und Ausgabe. Allerdings fordern wir eine *endliche* Eingabe. Die Zeichen auf dem Band seien aus einem *endlichen* Alphabet und der Schreib-/Lesekopf sei genau ein Zeichen breit. Die Kontrolleinheit der Turing-Maschine enthält deren Zustand, der ebenfalls aus einer *endlichen* Menge vorgegeben ist. Die Maschine liest in jedem Schritt genau ein Zeichen und schreibt abhängig von diesem und dem Zustand ein Zeichen. Außerdem ändert sie ihren Zustand und verschiebt den Lesekopf auf ein Nachbarzeichen.

Wir programmieren diesen Computer indem wir die Übergangsfunktion festlegen, die die Kontrolleinheit zur Wahl des zu schreibenden Zeichens und des Folgezustands verwendet. Um allgemeine Resultate zu erhalten, legen wir den Folgezustand aber im Allgemeinen nicht genau fest, sondern lassen mehrere Möglichkeiten offen. Dieses Konzept wird als nicht-deterministische Turing-Maschine bezeichnet.

Definition 1.1 (Turing-Maschine) *Eine Turing-Maschine (kurz: TM) ist ein Tupel* $\langle \Gamma, Q, \delta, b, q_0, q_+ \rangle$ *mit*

- Γ *einem endlichen Alphabet*
- Q *einer endlichen Menge Zustände mit* $Q \cap \Gamma = \emptyset$
- $\delta : \Gamma \times Q \to \mathbf{P}(\Gamma \times Q \times \{-1, 1\})$ *der Übergangsfunktion mit den Projektionen* δ_{write} *auf das zu schreibende Zeichen,* δ_{state} *auf den neuen Zustand und* δ_{step} *auf die Bewegungsrichtung des Lesekopfs*

- $b \in \Gamma$ *dem Leerzeichen am Band*
- $q_0 \in Q$ *dem Startzustand*
- $q_+ \in Q$ *einem positiven Endzustand*

Definition 1.2 (deterministische Turing-Maschine) *Eine TM ist genau dann* deterministisch*, wenn*

$$\forall \langle c, q\rangle \in \Gamma \times Q : |\delta(c, q)| \leq 1$$

Um den Ablauf eines Programms auf einer TM nachvollziehen zu können, definieren wir ein Standbild. Dadurch ergeben sich in natürlicher Weise Ergänzungen zur Übergangsfunktion. Wir geben an, wie man sie auf ein Standbild anwendet, und definieren die durch sie induzierte Relation als Nachfolgerelation.

Definition 1.3 (Standbild) *Zu einer gegebenen TM $M = \langle \Gamma, Q, \ldots\rangle$ ist ein Tripel $\langle k, q, t\rangle$ ein Standbild mit*

- $k \in \mathbf{N}$ *der Position des Schreib-/Lesekopfs,*
- $q \in Q$ *dem Zustand der TM und*
- $t : \mathbf{N} \to \Gamma$ *dem Inhalt des Bandes.*

$\mathcal{S}^*_M := \mathbf{P}\left(\mathbf{N} \times Q \times \Gamma^{\mathbf{N}}\right)$ *bezeichne die Menge aller Standbilder der TM.*

Bezeichnung 1.4 *Die Übergangsfunktion $\delta(c \in \Gamma, q \in Q)$ aufgefasst als Funktion auf den Standbildern sei $\delta(k, q, t) := \delta(t(k), q)$.*

Definition 1.5 (Nachfolgerelation) *Ein Standbild $S_1 \in \mathcal{S}^*_M$ ist (direkter) Nachfolger des Standbilds S_0 genau dann wenn S_1 durch die Übergangsfunktion direkt aus S_0 abgeleitet wird:*

$$\begin{aligned} S_0 \vdash S_1 \quad :\Longleftrightarrow \quad & \exists\ \langle write, state, step\rangle \in \delta(S_0) : \\ & k_1 = k_0 + step, \\ & q_1 = state, \\ & t_1(k_0) = write \text{ und} \\ & t_1 \equiv t_0 \text{ auf } \mathbf{N}\backslash\{k_0\} \end{aligned}$$

Obwohl uns nun auch die reflexive, transitive Hülle dieser Relation interessiert, wollen wir Konvergenzfragen vermeiden. Wir definieren rekursiv:

Defintion und Lemma 1.6

$$\begin{aligned} \vdash^0 &:= \{\langle S_0, S_0\rangle : S_0 \in \mathcal{S}^*_M\} \\ \forall i \in \mathbf{N} : \vdash^{i+1} &:= \left\{\langle S_0, S_{i+1}\rangle : S_0, S_{i+1} \in \mathcal{S}^*_M \wedge \exists S_i \in \mathcal{S}^*_M : S_0 \vdash^i S_i \wedge S_i \vdash S_{i+1}\right\} \\ \vdash^* &:= \bigcup_{i \in \mathbf{N}} \vdash^i \end{aligned}$$

Wir sehen sofort, dass damit $\vdash^1 = \vdash$. Außerdem beschreibt $\vdash^i$ genau die Relation, die durch exakt i-maliges Anwenden der Übergangsfunktion abgeleitet wird, also:

$$\vdash^i = \{\langle S_0, S_i\rangle : S_0, S_i \in \mathcal{S}^*_M \wedge \exists S_1, \ldots, S_{i-1} \in \mathcal{S}^*_M : \forall j = 0, \ldots, i-1 : S_j \vdash S_{j+1}\}$$

Bevor wir eine TM starten muss die Eingabe festgelegt werden. Die endliche Eingabe aus Γ^n mit der Länge $n \in \mathbf{N}$ soll dabei immer auf den Positionen 1 bis n stehen. Die restlichen Positionen sollen leer sein, also b enthalten. Die Eingabe selbst darf b nicht enthalten. Dies ist aber keine Einschränkung, weil dem Alphabet Γ ein weiteres Leerzeichen $\bar{b} \neq b$ hinzugefügt werden kann, wenn Problemstellungen mit einem Leerzeichen in der Eingabe auftreten. Das linkeste Zeichen $b\,(0)$ am Band soll leer sein um den Anfang des Bandes zu kennzeichnen – die korrekte Handhabung sei dem Programmierer überlassen.

Bezeichnung 1.7 (Startbild) *Ein Standbild $S = \langle k, q, t\rangle$ mit*

$$k = 1,\ q = q_0,\ t\,(0) = b \text{ und } \exists n \in \mathbf{N} : (\forall i = 1, \ldots n : t\,(i) \neq b \text{ und } \forall i > n : t\,(i) = b)$$

nennen wir Startbild. Für eine Eingabe $x \in \Gamma^*$ sei $S\,(x)$ das dazugehörige Startbild.

Bemerkung 1.8 (Länge der Eingabe) *Das für ein Startbild angegebene n ist eindeutig und heißt* Länge der Eingabe *$|x| = n$. Es gilt dann $x \in \Gamma^n$.*

In weiterer Folge interessieren uns nur mehr jene Standbilder, die im Laufe einer Berechnung auftreten. Die dadurch entstehenden Einschränkungen auf $\vdash$ und damit auf $\vdash^*$ sind erwünscht und betreffen nicht die Menge der interessanten Standbilder:

Definition 1.9 (erreichbare Standbilder) *Eine TM enthält nur*

$$\mathcal{S}_M := \{S \in \mathcal{S}_M^* : \exists S_0 \textit{ Startbild} : S_0 \vdash^* S\}$$

Lemma 1.10 *Das Band der TM ist immer fast überall leer.*

$$\forall S = \langle k, q, t\rangle \in \mathcal{S}_M : |\{i \in \mathbf{N} : t\,(i) \neq b\}| \in \mathbf{N}$$

Beweis 1.10 Wegen $S \in \mathcal{S}_M$ existiert ein Startbild $S_0 \vdash^* S$. Für ein Startbild gilt die Behauptung definitionsgemäß. Nach Definition 1.6 existiert ein $i \in \mathbf{N}$ mit $S_0 \vdash^i S$. Durch i-maliges Anwenden der Übergangsfunktion werden aber höchstens i Leerzeichen überschrieben. □

Bezeichnung 1.11 (Endbild, Ergebnis, Berechnung) *Wir sagen*

- *Ein Standbild $S = \langle k, q, t\rangle \in \mathcal{S}_M$ mit $\delta\,(S) = \emptyset$ ist ein* Endbild.
- *Ein Endbild mit $q = q_+$, $t\,(0) = b$ und*

  $$\exists n \in \mathbf{N} : (\forall i = 1, \ldots n : t\,(i) \neq b \text{ und } \forall i > n : t\,(i) = b)$$

  heißt Ergebnis.
- *Der nicht-leere Inhalt des Bandes eines Ergebnisses ist die* Ausgabe.
- *Für ein Startbild-Endbild-Paar $S_0 \vdash^* S_n$ heißt die endliche Folge der Standbilder, mit deren Hilfe S_n abgeleitet wird,* Berechnung.

Die TM beendet ihre Berechnung also, wenn es keinen Nachfolger gibt. Wir können aber feststellen, ob dies gewollt war oder passiert ist. Wir wollen annehmen, dass ein Endbild bereits dann ein Ergebnis ist, wenn $q = q_+$. Dies ist keine Erweiterung, wenn wir gleichzeitig fordern, dass das Band immer nur einen zusammenhängenden nicht-leeren Inhalt an den Positionen $1, 2, \ldots$ hat. Sollte ein Algorithmus Leerzeichen vorsehen, können wir dies durch ein weiteres Zeichen in Γ bewerkstelligen.

1.1.2 Beispiel: Gleichheit von Zeichenfolgen

Wir wollen eine TM anführen, die berechnet, ob zwei Zeichenfolgen aus $\{0,1\}^*$ gleich sind. Die Eingabe soll aus den zwei Zeichenfolgen getrennt durch ein Fragezeichen (?) bestehen. Wenn diese Konvention nicht eingehalten wird, soll das Band gelöscht werden. Bei einer gültigen Eingabe soll die Ausgabe gleich der Eingabe sein, wobei das Fragezeichen durch ein Gleich- oder Ungleichheitszeichen ersetzt wird. Zur erfolgreichen Absolvierung dieser Aufgabe werden wir Zeichen für Zeichen miteinander vergleichen. Dabei müssen wir aber die bearbeiteten Zeichen markieren. Das Alphabet unserer TM lautet also

$$\Gamma := \{0, 1, ?, =, \neq, \bar{0}, \bar{1}, b\}$$

Nun müssen wir aber zunächst sicherstellen, dass die Eingabe nur gültige Zeichen enthält. Dazu werden wir einfach alle Zeichen durchlaufen. Wenn wir einen Fehler finden, wechseln wir den Zustand um alle Zeichen zu löschen. Wir verwenden vorerst folgende Zustände und definieren gleich die Übergangsfunktion auf der jeweils auf diesen Zustand eingeschränkten Definitionsmenge[1]:

$$q_{scan} = q_0: \quad \delta(c, q_{scan}) = \begin{cases} \langle c, q_{scan}, 1\rangle \ \ falls\ c \in \{0,1\} \\ \langle ?, q_{scan?}, 1\rangle \ \ falls\ c = ? \\ \langle c, q_{error}, 1\rangle \ \ sonst \end{cases}$$

$$q_{scan?}: \quad \delta(c, q_{scan?}) = \begin{cases} \langle c, q_{scan?}, 1\rangle \ \ falls\ c \in \{0,1\} \\ \langle b, q_{learn}, -1\rangle \ \ falls\ c = b \\ \langle c, q_{error}, 1\rangle \ \ sonst \end{cases}$$

$$q_{error}: \quad \delta(c, q_{error}) = \begin{cases} \langle b, q_{clear}, -1\rangle \ \ falls\ c = b \\ \langle c, q_{error}, 1\rangle \ \ sonst \end{cases}$$

$$q_{clear}: \quad \delta(c, q_{clear}) = \begin{cases} \langle b, q_{+}, 1\rangle \ \ falls\ c = b \\ \langle b, q_{clear}, -1\rangle \ \ sonst \end{cases}$$

Analog können wir folgenden Algorithmus implementieren:

1. Merke das gelesene Zeichen und markiere es
2. Suche links das Fragezeichen
3. Suche links das erste nicht markierte Zeichen
4. Wenn die Zeichen nicht übereinstimmen gehe zu $q_{\neq}$
5. Sonst suche rechts das Fragezeichen
6. Suche rechts das letzte nicht markierte Zeichen und gehe zu 1

Hierbei bezeichne $q_{\neq}$ den Zustand, der die Markierungen von den Zeichen löscht und das Ungleichheitszeichen setzt. Die einzelnen Befehle entsprechen verschiedenen Zuständen, jedoch muss für jedes zu merkende Zeichen ein eigener Zustand pro Befehl eingeführt werden. Diese Vorgehensweise ist essentiell zur Programmierung von TM.

δ	0	1	$\bar{0}$	$\bar{1}$
q_{scan}	$0, q_{scan}, 1$	$1, q_{scan}, 1$	$\bar{0}, q_{error}, 1$	$\bar{1}, q_{error}, 1$
$q_{scan?}$	$0, q_{scan?}, 1$	$1, q_{scan?}, 1$	$\bar{0}, q_{error}, 1$	$\bar{1}, q_{error}, 1$
q_{error}	$0, q_{error}, 1$	$1, q_{error}, 1$	$\bar{0}, q_{error}, 1$	$\bar{1}, q_{error}, 1$
q_{clear}	$b, q_{clear}, -1$	$b, q_{clear}, -1$	$b, q_{clear}, -1$	$b, q_{clear}, -1$
$q_{=}$	$0, q_{=}, 1$	$1, q_{=}, 1$	$0, q_{=}, 1$	$1, q_{=}, 1$
$q_{\neq}$	$0, q_{\neq}, 1$	$1, q_{\neq}, 1$	$0, q_{\neq}, 1$	$1, q_{\neq}, 1$
q_{learn}	$\bar{0}, q_{find0}, -1$	$\bar{1}, q_{find1}, -1$	$\emptyset$	$\emptyset$
q_{find0}	$0, q_{find0}, -1$	$1, q_{find0}, -1$	$\emptyset$	$\emptyset$
q_{find1}	$0, q_{find1}, -1$	$1, q_{find1}, -1$	$\emptyset$	$\emptyset$
$q_{find0?}$	$\bar{0}, q_{next}, 1$	$1, q_{\neq}, 1$	$\bar{0}, q_{find0?}, -1$	$\bar{1}, q_{find0?}, -1$
$q_{find1?}$	$0, q_{\neq}, 1$	$\bar{1}, q_{next}, 1$	$\bar{0}, q_{find1?}, -1$	$\bar{1}, q_{find1?}, -1$
q_{next}	$\emptyset$	$\emptyset$	$\bar{0}, q_{next}, 1$	$\bar{1}, q_{next}, 1$
$q_{next?}$	$0, q_{next?}, 1$	$1, q_{next?}, 1$	$\bar{0}, q_{learn}, -1$	$\bar{1}, q_{learn}, -1$
$q_{learn?}$	$0, q_{\neq}, 1$	$1, q_{\neq}, 1$	$\bar{0}, q_{learn?}, -1$	$\bar{1}, q_{learn?}, -1$

δ	$?$	$=$	$\neq$	b
q_{scan}	$?, q_{scan?}, 1$	$=, q_{error}, 1$	$\neq, q_{error}, 1$	$b, q_{clear}, -1$
$q_{scan?}$	$?, q_{error}, 1$	$=, q_{error}, 1$	$\neq, q_{error}, 1$	$b, q_{learn}, -1$
q_{error}	$?, q_{error}, 1$	$=, q_{error}, 1$	$\neq, q_{error}, 1$	$b, q_{clear}, -1$
q_{clear}	$b, q_{clear}, -1$	$b, q_{clear}, -1$	$b, q_{clear}, -1$	$b, q_{ready}, 1$
$q_{=}$	$=, q_{=}, 1$	$\emptyset$	$\emptyset$	$b, q_{ready}, -1$
$q_{\neq}$	$\neq, q_{\neq}, 1$	$\emptyset$	$\emptyset$	$b, q_{ready}, -1$
q_{learn}	$?, q_{learn?}, -1$	$\emptyset$	$\emptyset$	$\emptyset$
q_{find0}	$?, q_{find0?}, -1$	$\emptyset$	$\emptyset$	$\emptyset$
q_{find1}	$?, q_{find1?}, -1$	$\emptyset$	$\emptyset$	$\emptyset$
$q_{find0?}$	$\emptyset$	$\emptyset$	$\emptyset$	$b, q_{\neq}, 1$
$q_{find1?}$	$\emptyset$	$\emptyset$	$\emptyset$	$b, q_{\neq}, 1$
q_{next}	$?, q_{next?}, 1$	$\emptyset$	$\emptyset$	$\emptyset$
$q_{next?}$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$
$q_{learn?}$	$\emptyset$	$\emptyset$	$\emptyset$	$b, q_{=}, 1$

Tabelle 1.1: Übergangsfunktion zum Beispiel: Gleichheit von Zeichenfolgen

Außerdem benötigen wir noch einen Zustand $q_=$ analog zu $q_\neq$ und wir müssen alle Fälle betrachten, die am Ende auftreten können. Die weiteren Überlegungen sind in Tabelle 1.1 auf Seite 5 zusammengefasst.

Die hier konstruierte TM lautet also

$$\begin{aligned} M := \langle \quad & \Gamma := \{0, 1, ?, =, \neq, \bar{0}, \bar{1}, b\}, \\ & Q := \{q_{scan}, q_{scan?}, q_{error}, q_{clear}, q_=, q_\neq, q_{learn}, \\ & q_{find0}, q_{find1}, q_{find0?}, q_{find1?}, q_{next}, q_{next?}, q_{learn?}, q_{ready}\}, \\ & \delta, b, q_0 := q_{scan}, q_+ := q_{ready}\rangle \end{aligned}$$

mit 8 Zeichen und 15 Zuständen. Diese TM ist deterministisch und jedes Endbild ist ein Ergebnis. Das ist aber nicht notwending, wenn uns ausschließlich interessiert, ob die Eingabe gültig ist *und* die Zeichenfolgen gleich sind. Diese Überlegung führt uns zu weiteren Definitionen im folgenden Abschnitt.

1.1.3 Sprache, Laufzeit

Interessante Aufgabenstellungen für TM sind

- herauszufinden ob eine *Eingabe* eine gewisse *Eigenschaft* hat und
- aus einer *Eingabe* eine *Ausgabe* zu ermitteln.

Die folgenden Definitionen liefern uns dazu die Rahmenbedingungen:

Definition 1.12 (Sprache) *Zu einer TM M bezeichnen wir mit*

$$L(M) := \{x \in \Gamma^* : \exists E \; Ergebnis : S(x) \vdash^* E\}$$

die von M akzeptierte Sprache.

Im angeführten Beispiel ist $L(M) = \Gamma^*$ und daher nicht zum Trennen bezüglich der Eigenschaft „*die Zeichenfolgen sind gleich*“ geeignet. Wir haben – wenn uns ausschließlich diese Entscheidung interessiert – sogar zu viel Aufwand betrieben: Die TM

$$\begin{aligned} M := \langle \quad & \Gamma := \{0, 1, ?, \bar{0}, \bar{1}, b\}, \\ & Q := \{q_{scan}, q_{scan?}, q_=, q_{learn}, q_{find0}, q_{find1}, q_{find0?}, q_{find1?}, q_{next}, q_{next?}, q_{learn?}\}, \\ & \delta \mid_{\Gamma \times Q}, b, q_0 := q_{scan}, q_+ := q_=\rangle \end{aligned}$$

mit 6 Zeichen und 11 Zuständen liefert genau $L(M) = \{x?x : x \in \{0,1\}^*\}$. Die Einschränkung von δ bezieht sich hierbei auch auf das Bild, d. h. Elemente der Bilder, die nicht in $\Gamma \times Q \times \{-1, 1\}$ liegen, werden entfernt – dadurch entstehen in diesem Fall Endbilder, die keine Ergebnisse sind.

Bezeichnung 1.13 *Eine TM M, die immer ein Ergebnis liefert, also $\forall x \in \Gamma^* \; \exists E \; Ergebnis : S(x) \vdash^* E$ oder $L(M) = \Gamma^*$, nennen wir* total.

Bezeichnung 1.14 *Eine* deterministische *und* totale *TM heißt* berechnend.

[1] Die Bilder seien die einelementigen Mengen, die das jeweils angegebene Tupel enthalten.

Definition 1.15 (Funktion) *Zu einer* berechnenden *TM M bezeichnet*

$$\begin{array}{rccc} f_M: & \Gamma^* & \to & \Gamma^* \\ & \mathit{Eingabe} & \mapsto & \mathit{Ausgabe} \end{array}$$

die von M berechnete Funktion*. Diese Funktion ist wohldefiniert, weil M deterministisch und total ist.*

Als nächstes interessiert uns der Aufwand, den wir zur Erfüllung einer solchen Aufgabenstellung betreiben müssen.

Definition 1.16 (Laufzeit, Zeitkomplexität) *Die Zeitkomplexität einer TM M ist*

$$T_M(n) := \max_{x \in L(M) \cap \Gamma^n} \min \left\{ k \in \mathbf{N} : \exists E \; \mathit{Ergebnis} : S(x) \vdash^k E \right\}$$

falls M nicht-deterministisch *und für eine* deterministische *TM*

$$T_M(n) := \max_{x \in \Gamma^n} \left\{ k \in \mathbf{N} : \exists E \; \mathit{Endbild} : S(x) \vdash^k E \right\}.$$

Außerdem ist $T_M(n) := \infty$ falls es ein $x \in \Gamma^n$ gibt, für das M nicht hält.

Hierbei ist es wichtig, dass uns bei nicht-deterministischen TM nur Ergebnisse interessieren, weil die Sprache über die Existenz eines solchen definiert ist. Selbst wenn bereits ein Endbild vorliegt, kann nicht gesagt werden ob ein Ergebnis existiert. Wir gehen jedoch davon aus, dass wir eine nicht-deterministische TM nach spätestens $T_M(n)$ Schritten anhalten.

Lemma 1.17 (Speicherkomplexität) *Die tatsächlich benötigte Länge des Bandes ist durch die Zeitkomplexität beschränkt:*

$$\forall i, n \in \mathbf{N} : \forall x \in \Gamma^n : \forall S = \langle k, q, t \rangle \in \mathcal{S}_M : S(x) \vdash^* S \wedge i > \max(n, T_M(n)) \Rightarrow t(i) = b$$

für M deterministisch. Für nicht-deterministisches M sei zusätzlich $x \in L(M)$.

Beweis 1.17 Gelingt mit dem selben Argument wie Beweis 1.10: Aus der Definition von $T_M(n)$ folgt $\exists N \leq T_M(n) : S(x) \vdash^N S$. In $S(x)$ steht der Kopf an Position 1, in S folglich höchstens an Position $T_M(n) + 1$ ohne diese beschrieben zu haben. □

Bezeichnung 1.18 (Komplexität) *Wir können also die* Zeit*komplexität kurz mit* Komplexität *bezeichnen.*

Die Anzahl der Zeichen und Zustände interessiert uns dabei wenig, da diese fixe natürliche Zahlen sind, die nicht von der Eingabe abhängen. Ebenso ist der Umfang der Übergangsfunktion dadurch beschränkt: Ihre Bilder haben insgesamt höchstens $2|\Gamma|^2 |Q|^2 \in \mathbf{N}$ Elemente.

Die Komplexität der im Beispiel angeführten TM ist aus $O(n^2)$.

1.1.4 Komlexitätsklassen

Zentrale Fragen der Komplexitätstheorie beziehen sich auf Komplexitätsklassen. Diese definiert man auf *Sprachen* bezüglich der Existenz eines Computers von entsprechender Komplexität. Die folgenden Klassen werden über TM definiert:

Definition 1.19 (P) *Unter Problemen mit polynomieller Laufzeit verstehen wir*

$$P := \{L \subseteq \Gamma^* : \ \Gamma \textit{ beliebig, endlich } \wedge \exists M \textit{ deterministische } TM : \exists p \in \mathbf{N}[x] : \\ L = L(M) \wedge \forall n \in \mathbf{N} : T_M(n) \leq p(n)\}$$

Definition 1.20 (NP) *Probleme mit nicht-deterministisch polynomieller Laufzeit sind*

$$NP := \{L \subseteq \Gamma^* : \ \Gamma \textit{ beliebig, endlich } \wedge \exists M \ TM : \exists p \in \mathbf{N}[x] : \\ L = L(M) \wedge \forall n \in \mathbf{N} : T_M(n) \leq p(n)\}$$

Es werden dabei Sprachen über beliebigen endlichen Alphabeten betrachtet. Trotzdem ist das Verhältnis der beiden Mengen zueinander scheinbar einfach beschrieben:

Lemma 1.21 ($\mathbf{P} \subseteq \mathbf{NP}$) *NP enthält P.*

Beweis 1.21 Für eine Sprache aus P existiert eine deterministische TM M und eine obere Schranke für die Laufzeit. M ist aber gleichzeitig eine (nicht-deterministische) TM: Wenn wir die Komplexität in diesem Sinne messen, wenden wir das Maximum auf eine Teilmenge der Elemente an, während das Minimum nicht zum Tragen kommt – die Komplexität wird also nicht größer. Daher erfüllt M auch die Forderungen in NP. □

Ob $NP = P$, ist eine immer noch offene Frage und gehört zu den bekanntesten. Ihre Beantwortung ist mit einer Million Dollar dotiert ([18],[4]). Allgemein wird angenommen, dass $NP \neq P$, weil sehr viele schwierige Problemstellungen aus vielen verschiedenen Bereichen in NP liegen, für die aber kein effizienter Algorithmus gefunden werden konnte. Diese schwierigsten Probleme sind äquivalent zueinander – in einem Sinn, den Abschnitt 1.2.1 beschreibt – und alle bisher gefundenen Algorithmen haben nicht-polynomielle Laufzeit (in den jeweils schlechtesten Fällen).

Die offensichtliche Simulation nicht-deterministischer TM unter realen Umständen lässt sogar exponentielle Laufzeit vermuten. Zuerst legen wir uns ein paar Techniken zurecht, die es uns in Hinblick auf die eben definierten Problemklassen auch erlauben, „normale“ Computer zu verwenden. Dazu geben wir an, wie wir Pseudocode-Befehle mit einer deterministischen TM simulieren. Diese Elemente können natürlich auch auf nicht-deterministische TM angewendet werden. Darüber hinaus geben wir die Komplexität an: Alle Befehle mit polynomieller Laufzeit können wir verwenden, um eine TM zum Beweis einer Zugehörigkeit zu P oder NP zu konstruieren.

Eingabealphabet

Wir werden häufig Zeichen am Band markieren müssen um sie wiederzufinden oder nicht noch einmal zu bearbeiten. Das ist kein Problem, weil wir zu einem ursprünglichen Alphabet endlich viele Markierungen hinzufügen können. Diese Anzahl muss für den Algorithmus – nicht für die Eingabe – fest gewählt sein. Sicherzustellen, dass jedes Zeichen $c \in \Gamma$ der Eingabe nicht markiert oder allgemeiner aus einem Eingabealphabet Σ mit $b \notin \Sigma \subset \Gamma$ ist, kostet n Schritte, wenn $n = |x|$ die Länge der Eingabe ist.

Syntax, Semantik

Bei der Gleichheit von Zeichenfolgen haben wir an die Eingabe noch eine weitere Anforderung gestellt: Die Eingabe durfte nur genau ein Fragezeichen enthalten. Eine solche Anforderung nennen wir *Syntax*. Die Syntax ist einfach zu überprüfen und soll (viele) Elemente aus Γ^* erkennen, die nicht in L liegen. Ein Eingabealphabet gehört zur Syntax.

Erst bei einer korrekten Syntax können wir der Eingabe eine Bedeutung geben. Im Beispiel haben wir die Teile vor und nach *dem einen* Fragezeichen als Zeichenfolgen interpretiert. Diese Bedeutung heißt *Semantik*. Insgesamt wird eine Eingabe akzeptiert, wenn sie die Syntax *und* semantische Anforderungen erfüllt, zum Beispiel die Gleichheit der Zeichenfolgen.

Eine obere Schranke für die Laufzeit muss Syntax- und Semantikprüfung berücksichtigen.

Variablen

Wir können durch Erweiterung der Zustände eine endliche Anzahl einzelner Zeichen aus dem Alphabet „speichern“ und diese wiedergeben oder für eine Entscheidung heranziehen. Auch diese Anzahl darf aber nicht von der Eingabe abhängen.

Um uns eine variable Anzahl Zeichen zu merken, können wir diese am Ende der Eingabe auf das Band schreiben. Dazu verwenden wir einfach ein anderes Alphabet für jede neue Variable. Wir gehen davon aus, dass wir die Länge einer Variable unseres Algorithmus immer mit einer Funktion l_{var} in Abhängigkeit von der Länge der Eingabe beschränken können. Dann kostet das Kopieren der gesamten Eingabe in eine Variable $O\left(n\left(n + l_{var}\left(n\right)\right)\right)$ Schritte, weil die Anzahl der Variablen konstant sein muss.

Das Vergleichen zweier Variablen kostet $O\left(n + l_{var}\left(n\right)^2\right)$. Das n benötigt man um von der aktuellen Position am Band zur Variable zu wechseln und anschließend zurückzukehren. Den selben Aufwand hat das Addieren zweier Variablen, wenn man die gespeicherte Zahl als Länge der Variable codiert. Eine Multiplikation braucht in diesem Fall $O\left(n + l_{var}\left(n\right)^3\right)$ Schritte. Die Codierung in Stellen über einem Alphabet erhöht den Aufwand nicht. So benötigt etwa ein Algorithmus, der die Vorkommen bestimmter Zeichen zählt und als dekadische Zahl am Ende der Eingabe niederschreibt $O\left(n\left(n +_{10} log\left(n\right)\right)\right) \subset O\left(n^2\right)$ Schritte: n für das Durchlaufen der Eingabe und jeweils $n +_{10} log\left(n\right)$ für das Kopieren der Information.

Schleifen

Ebenso sehen wir, dass es kein Problem ist bestimmten Code mehrfach – nämlich auch entsprechend des Wertes in einer Variable – zu wiederholen. Wir können dazu die Werte bei jedem Durchlauf direkt dekrementieren oder mit Markierungen zur selben Basis versehen. Dieses Konzept kann sogar dahingehend erweitert werden, dass Variablen an den selben Stellen am Band gespeichert werden. Nun interessiert uns aber welche Werte wir nur in Abhängigkeit der Länge einer Eingabe oder einer Variable berechnen können:

Definition 1.22 (Schrittfunktion) *Eine Funktion $T : \mathbf{N} \to \mathbf{N}$ heißt* Schrittfunktion, *wenn es eine deterministische TM gibt, die für alle Eingaben aus Γ^n genau nach $T\left(n\right)$ Schritten hält.*

Beispiele für Schrittfunktionen ergeben sich aus den eben beobachteten Laufzeiten. Nun können wir auch den Komplexitätszusammenhang zwischen nicht-deterministischen und deterministischen TM angeben:

Satz 1.23 (Simulation nicht-deterministischer TM) *Sei N eine* nicht-deterministische *TM mit $T_N(n) \in O(T(n))$ einer durch eine Schrittfunktion beschränkten Komplexität und $L = L(N)$ der von N akzeptierten Sprache. Dann existiert eine Konstante c und eine* deterministische *TM M, die L in $T_M(n) \in O\left(c^{T(n)}\right)$ akzeptiert.*

Beweis 1.23 Sei x eine beliebige Eingabe mit Länge n und $k = \max\{|\delta(y,q)| : y \in \Gamma_N \wedge q \in Q_N\}$. Dann gibt es höchstens $k^{T_N(n)}$ Berechnungen die vor dem Zeitpunkt $T_N(n)$ enden. $x \in L$ gilt dann und nur dann, wenn eine dieser Berechnungen mit einem Ergebnis endet.

Sei $m \in \mathbf{N}$ ein Faktor, der die Schranke $T_N(n) \leq mT(n)$ erfüllt. M schreibt nun zunächst $k^{mT(n)}$ in der Basis k in eine Variable A und kopiert die Eingabe in eine andere. Anschließend durchläuft M eine Schleife in der jedesmal die Variable A dekrementiert wird und bis A gleich 0 folgende Befehle ausführt:

1. Kopiere die Eingabe wieder an den richtigen Platz.

2. Simuliere N. Wähle dabei im i-ten Schritt jeweils jenes Element der Übergangsfunktion, das der i-ten Stelle von Variable A entspricht.

Das Kopieren und die Wahl der Übergangsfunktion in jedem Schritt benötigen jeweils $O(n(n + l_{var}(n)))$ Schritte falls N das Band nicht über die Länge der Eingabe benötigt, $O(l_{var}(n)(n + l_{var}(n)))$ sonst. Insbesondere wird ihre Laufzeit also durch ein Polynom $p(mT(n))$ beschränkt. Daher dauert die gesamte Simulation höchstens $p(mT(n)) \cdot k^{mT(n)} \in O\left(c^{mT(n)}\right)$ Schritte, wobei c eine Konstante ist. □

1.2 NP-Vollständigkeit

Nachdem wir einen Zusammenhang zwischen allgemeinen TM und existierenden Computern hergestellt haben, wollen wir eine Verbindung zwischen Sprachen und realen Problemen herstellen:

Bezeichnung 1.24 (Entscheidungsproblem) *Eine Sprache $L \subseteq \Sigma^*$ über einem endlichen Eingabealphabet Σ nennen wir* Entscheidungsproblem *oder kurz* Problem. *Die Fragestellung eines Entscheidungsproblems lautet:*

Gegeben eine Zeichenfolge $x \in \Sigma^*$. Gilt $x \in L$?

Der Zusammenhang zwischen einer Sprache als Teilmenge beliebiger Eingaben und einem verbal formulierten Problem ist die auf Seite 9 beschriebene Semantik. Die Syntax wird verwendet um das Problem für den Computer zu übersetzen. Wir verwenden aber die Begriffe Problem und Sprache beliebig.

Bezeichnung 1.25 (Probleminstanz) *Zu einem Entscheidungsproblem $L \subseteq \Sigma^*$ ist $x \in \Sigma^*$ eine (Problem-)*Instanz*. Die Länge von x heißt* Größe *der Instanz.*

Bemerkung 1.26 *P entspricht also genau den Problemen, die auf existierenden Computern in* polynomieller Zeit *abhängig von der Größe der Instanz gelöst werden können.*

Wenn wir eine ähnliche Aussage über NP machen wollen, müssen wir zu einem Trick greifen, weil exponentielle Laufzeiten – unabhängig davon ob die analoge Aussage dann überhaupt richtig wäre – in der Praxis uninteressant sind:

Bemerkung 1.27 *NP entspricht genau den Problemen, die auf existierenden Computern mit einem* Kobold *in polynomieller Zeit abhängig von der Größe der Instanz gelöst werden können. Diesen Kobold darf der Computer bei einer Verzweigung im Code befragen, welcher Teil ausgeführt werden soll: Der Kobold antwortet immer mit der richtigen Entscheidung, falls es diese gibt.*

Die Intension dieser Bemerkung ist klar: Der Kobold implementiert die *Existenz* einer Berechnung mit Ergebnis indem er immer das entsprechende Element der Übergangsfunktion verwendet. Die Fragen an den Kobold dürfen aber nicht von der Instanz abhängen. Daher verfolgen wir diesen Gedanken auch nicht weiter oder formalisieren ihn, er dient nur dem Verständnis.

1.2.1 Reduktion von Problemen

Die Definitionen von P und NP lassen vermuten, dass diese unvorstellbar groß sind. Eine Klassifizierung ist wünschenswert. Zu diesem Zweck müssen wir einen Zusammenhang zwischen Problemen festlegen:

Definition 1.28 (Transformation) *Seien Σ_1, Σ_2 endliche Alphabete.*

- *Eine Funktion $f : \Sigma_1^* \to \Sigma_2^*$ heißt* Transformation*, wenn es eine* berechnende *TM M gibt mit $f_M = f$.*
- *Eine Transformation ist* polynomiell*, wenn es ein Polynom $p \in \mathbf{N}[n]$ gibt, sodass $T_M(n) \in O(p(n))$ für eine der TM M gilt.*

Defintion und Lemma 1.29 (Reduzierbarkeit) *Seien $L_i \subseteq \Sigma_i^*,\ i = 1, 2$ Sprachen über endlichen Alphabeten. L_1 heißt (polynomiell) reduzierbar auf L_2, $L_1 \leq_{(p)} L_2$, wenn es eine (polynomielle) Transformation $f : \Sigma_1^* \to \Sigma_2^*$ gibt, die die Zugehörigkeit zur jeweiligen Sprache für alle Zeichenfolgen über Σ_1 erhält:*

$$L_1 \leq_{(p)} L_2 : \iff \exists f : \Sigma_1^* \to \Sigma_2^*\ (polynomielle)\ Transformation : \\ \forall x \in \Sigma_1^* : x \in L_1 \leftrightarrow f(x) \in L_2$$

In der vorliegenden Arbeit wollen wir Reduzierbarkeit als polynomielle Reduzierbarkeit interpretieren.

$\leq$ ist reflexiv und auf endlichen Teilmengen transitiv.

Beweis 1.29 Die Identität auf einem beliebigen Alphabet ist eine Transformation mit Laufzeit konstant 0, daher ist $\leq$ reflexiv. Zur Transitivität benötigen wir zwei Argumente:

1. Wir können die Hintereinanderausführung zweier TM M_1, M_2 in einer simulieren ohne den Aufwand zu erhöhen, indem jedes Ergebnis von M_1 zu einem Startbild von M_2 wird.

2. Die Länge der Ausgabe von M_i $i = 1, 2$ ist durch $T_{M_i}(n) \leq p_i(n)$ beschränkt. Daher ist die Gesamtdauer der Simulation durch das Polynom $p_1(n) + p_2(p_1(n))$ beschränkt.

Dieses Argument kann auf endlich viele TM erweitert werden, nicht aber auf beliebige Reduzierbarkeit. □

Wenn wir ein Problem L_1 auf L_2 reduzieren können, enthält L_2 in einem gewissen Sinn *alle* Instanzen von L_1. L_2 muss daher schwieriger bzw. nicht leichter sein. Nun können wir NP-vollständig als die schwierigsten Probleme in NP definieren:

1.2.2 Die Klasse NP-vollständig

Definition 1.30 (NP-vollständig) *Eine Problem L liegt in oder ist* NP-vollständig*, wenn es in NP liegt und alle Probleme aus NP enthält:*

$$\textit{NP-vollständig} := \{L \in NP : \forall L' \in NP : L' \leq L\}$$

In Abschnitt 1.3.3 werden wir zeigen, dass diese Klasse nicht leer ist. Sobald wir aber ein Problem aus NP-vollständig gefunden haben, können wir die Zugehörigkeit weiterer Probleme einfacher zeigen.

Lemma 1.31 *Sei L_0 aus NP-vollständig.*

$$\forall L_1 \in NP : L_0 \leq L_1 \Rightarrow L_1 \in \textit{NP-vollständig}$$

Beweis 1.31 Wir müssen zeigen, dass $\forall L \in NP : L \leq L_1$. Wir haben aber $L \leq L_0 \leq L_1$ als Voraussetzung und die Transitivität auf $\{L, L_0, L_1\}$. □

Darüber hinaus können wir mit dieser Klasse die Diskussion von $N \neq NP$ aus Abschnitt 1.1.4 unterstreichen. Es gilt nämlich nicht nur – wie Lamma 1.31 zeigt –, dass ein Problem schwierig ist, wenn es *ein beliebiges* schwieriges Problem enthält, sondern auch, dass *alle* schwierigen Probleme einfach sind genau dann, wenn eines von ihnen einfach ist:

Satz 1.32 *Sei L_0 aus NP-vollständig.*

$$L_0 \in P \Longleftrightarrow P = NP$$

Beweis 1.32 Wenn $P = NP$ gilt klarerweise $L_0 \in P$. Umgekehrt sei $L \in NP$ beliebig. Es gilt also $L \leq L_0$, weil dieses aus NP-vollständig ist. Die deterministische TM, die L akzeptiert, braucht also nur die polynomielle Transformation auf L_0 durchzuführen und die berechnete Instanz zu entscheiden. Die Argumente aus Beweis 1.29 liefern eine insgesamt polynomielle Laufzeit. □

Beispiel-Probleme

In diesem Abschnitt sind NP-vollständige Beispielprobleme aufgeführt und kurz beschrieben. Beweise zur Komplexität sind unter vielen anderen in [11] zu finden. Allgemein gilt, dass die meisten dieser Probleme auch in der Praxis bedeutend sind, und dass Lösungsansätze ausführlich untersucht wurden.

SAT

Hierbei geht es um die *Erfüllbarkeit* aussagenlogischer Formeln. In Abschnitt 1.3 werden wir das Problem exakt definieren.

Zu beachten gilt, dass je nach Autor Formeln bestimmter Art zugelassen werden. Die Bandbreite reicht ausgehend von beliebigen logischen Operatoren (Negation, Konjunktion, Disjunktion, Äquivalenz, Kontravalenz, Implikation) über Normalformen (zum Beispiel CNF[2], k-CNF) bis hin zur ausschließlichen Verwendung von Negation und entweder Konjunktion oder Disjunktion. Alle diese Ausprägungen können ineinander übergeführt werden.

CLIQUE

Die Fragestellung dieses Problems lautet: Gibt es zu einem gegebenen ungerichteten Graphen und einer natürlichen Zahl k einen *vollständigen* Untergraphen der Größe k? Unter einer CLIQUE versteht man dabei eben jene Teilmenge der Knoten, in der *je zwei Knoten adjazent* sind. Dieses Problem wird in Abschnitt 3.3.4 definiert.

VC

Das Problem „Vertex Cover“ ist eng mit CLIQUE verwandt. Unter den gleichen Vorgaben sucht man eine Teilmenge, in der zu jeder Kante *mindestens ein inzidenter Knoten* liegt. Auch dieses Problem wird in Abschnitt 3.3.4 genauer vorgestellt.

TSP

Unter dem „Traveling Salesman Problem“ versteht man das Problem eines Handlungsreisenden, seine *Reisekosten* zu beschränken: Gegeben ein einfacher gerichteter Graph mit durch natürliche Zahlen bewerteten Kanten und eine natürliche Zahl k, gibt es einen Pfad, der alle Knoten (einmal) durchläuft, sodass die *Summe der Bewertungen* durch k beschränkt ist?

Für TSP gibt es die Varianten mit ungerichteten Graphen, also symmetrischen Bewertungen, und mit Bewertungen, die die Dreiecksungleichung erfüllen. Aber auch diese Spezialfälle sind genauso schwierig.

HAM

Gegeben ein Graph, gibt es einen *Hamiltonkreis*? Darunter versteht man einen einfachen Kreis, der alle Knoten mindestens einmal, aber keine Kante öfter trifft. Die Varianten

[2] siehe Definition 1.33

gerichteter und ungerichteter Graph sind insofern gleich schwierig, als sie beide in NP-vollständig liegen.

3DM

Beim „3-dimensionalen Matching“ (Zusammenführen/Anpassen) versucht man von einer gegebenen *dreistelligen Relation* auf gleich mächtigen Mengen eine Teilmenge anzugeben, die nicht mächtiger ist als diese Mengen, aber *jedes ihrer Elemente* in einem Tupel enthält. Auch dieses Problem wird in Abschnitt 3.3.4 exakt definiert.

KNAPSACK

Das Rucksack-Problem findet zum Beispiel in der Kryptologie Anwendung: Gegeben ganze Zahlen $a_1, \ldots, a_n, b \in \mathbf{Z}$, gibt es eine Teilmenge der a_i, deren *Summe* gleich b ist? Das selbe Problem auf natürlichen Zahlen $a_1, \ldots, a_n, b \in \mathbf{N}$ ist ebenfalls bereits schwierig. Wir werden das in Abschnitt 3.3.4 beweisen.

Verwandte Probleme

Es ist wichtig zu beachten, dass die hier angeführten Fragen immer Entscheidungsfragen sind, die positiv oder negativ beantwortet werden sollen. Daher gibt es zu jedem dieser Probleme eine[3] Sprache im hier behandelten Sinn.

Oft werden dieselben Bezeichnungen für verwandte Optimierungs- oder Abzählprobleme verwendet:

- Wieviele erfüllende Belegungen, Hamiltonkreise, Matchings oder Partitionen einer ganzen Zahl gibt es?
- Wie groß ist die größte CLIQUE?
- Wie klein sind die kleinsten VC oder Reisekosten?

Es ist klar, dass diese Fragestellungen *nicht einfacher* zu beantworten sind. Diese Probleme entsprechen der Berechnung einer Funktion (Probleminstanz $\rightarrow$ Antwort) und werden auch anhand ihrer Laufzeit analysiert, entsprechen aber nicht unserer Kategorisierung.

1.3 SAT

In Kapitel 2 werden wir SAT als Hilfsmittel verwenden um zu beweisen, dass Minesweeper NP-vollständig ist. Außerdem garantiert uns dieser Abschnitt, dass NP-vollständig nicht leer ist. Für beide Belange benötigen wir aber nur ein beliebiges NP-vollständiges Problem und werden SAT nicht in vollem Umfang diskutieren.

[3] Es gibt unendlich viele Sprachen/Darstellungen für jedes Problem!

1.3.1 Grundlagen

Zur Definition des Erfüllbarkeitsproblems SAT benötigen wir Formeln der Aussagenlogik.

Definition 1.33 (Formel) *Wir definieren rekursiv:*

- x_i *heißt (Wahrheits-)*Variable *für alle* $i \in \mathbf{N}$.
- *Für eine Variable* x_i *sind* x_i *und* $\overline{x}_i$ Literale.
- *Für Literale* $y_1, \ldots, y_n$ *ist* $y_1 \vee \ldots \vee y_n$ *eine* Klausel.
- *Für Klauseln* $C_1, \ldots, C_m$ *ist* $C_1 \wedge \ldots \wedge C_m$ *eine* Formel.

Diese Form aussagenlogischer Formeln nennt man *Konjunktive Normalform* (CNF). Weitere Normalformen oder allgemeine Formeln mit weiteren Operatoren werden wir hier nicht benötigen.

Bezeichnung 1.34 (Belegung) *Eine Funktion* $\varphi : \mathbf{N} \to \{0, 1\}$ *heißt* Belegung.

Definition 1.35 (Auswertung) *Die Erweiterung einer Belegung* φ *auf Formeln nennen wir* Auswertung. *Wir definieren rekursiv entsprechend Definition 1.33:*

- $\varphi(x_i) := \varphi(i)$ *und* $\varphi(\overline{x}_i) := 1 - \varphi(i)$ *für Literale*
- $\varphi(y_1 \vee \ldots \vee y_n) := \max\{\varphi(y_1), \ldots, \varphi(y_n)\}$ *für Klauseln*
- $\varphi(C_1 \wedge \ldots \wedge C_m) := \min\{\varphi(C_1), \ldots, \varphi(C_m)\}$ *für Formeln*

Bemerkung 1.36 *Die Auswertung entspricht genau den bekannten Junktoren* und *und* oder *der Aussagenlogik, wobei Klauseln – also „*$\vee$*“ – zuerst ausgewertet werden. Aus Gründen der Übersichtlichkeit können wir Klammern um Klauseln setzen. Diese unterstreichen aber nur die semantische Bedeutung und werden in der Syntax einfach ignoriert.*

Problem 1.37 (SAT) *Gegeben eine aussagenlogische Formel* F *(in CNF). Existiert eine Belegung* φ *deren Auswertung* $\varphi(F) = 1$ *erfüllt?*

Diese Formulierung ist offensichtlich semantisch. Zur Klassifizierung von SAT müssen wir den Zusammenhang zur Sprache herstellen.

1.3.2 Umsetzung am Computer

Das Problem bei der direkten Umsetzung des verbal formulierten Problems SAT auf einer TM ist der unendliche Vorrat an Variablen x_i. Wir müssen diese in einem endlichen Alphabet codieren. Der auch später zur Angabe eines Algorithmus einfachste Weg ist es, Literalzeichen mit der Vielfachheit ihrer Indizierung zu wiederholen.

$$\Sigma_{SAT} := \{x, \bar{x}, \vee, \wedge\}$$

Als nächstens werden wir eine TM M_{syn} angeben, die alle syntaktisch richtigen Eingaben für SAT akzeptiert. Hierbei ist es nur wichtig, dass $\vee$ und $\wedge$ weder am Anfang, noch am Ende, noch direkt aufeinanderfolgend auftreten. Außerdem müssen aufeinanderfolgende Literalzeichen gleich sein.

δ	x	$\bar{x}$	$\vee$	$\wedge$	b
q_x	$x, q_x, 1$	$\emptyset$	$\vee, q_{op}, 1$	$\wedge, q_{op}, 1$	$b, q_+, -1$
$q_{\bar{x}}$	$\emptyset$	$\bar{x}, q_{\bar{x}}, 1$	$\vee, q_{op}, 1$	$\wedge, q_{op}, 1$	$b, q_+, -1$
q_{op}	$x, q_x, 1$	$\bar{x}, q_{\bar{x}}, 1$	$\emptyset$	$\emptyset$	$\emptyset$

Tabelle 1.2: Übergangsfunktion von M_{syn}

Lemma 1.38 (Syntax von SAT) *Die Syntax von SAT kann mit*

$$M_{syn} := \langle \Gamma := \{x, \bar{x}, \vee, \wedge, b\}, Q := \{q_x, q_{\bar{x}}, q_{op}, q_+\}, \delta, b, q_0 := q_{op}, q_+ \rangle$$

in linearer Zeit überprüft werden: $T_{M_{syn}}(n) \in O(n)$.

Beweis 1.38 Die Gültigkeit der Syntax und die Laufzeit können unmittelbar von δ in Tabelle 1.2 auf Seite 16 abgelesen werden. □

Da die Codierung eines Literals y_i genau i Zeichen benötigt, folgt sofort:

Korollar 1.39 *Eine Auswertung φ einer Formel als Eingabe der Länge n hängt höchstens von den Werten $\varphi(1), \ldots, \varphi(n)$ ab.*

Der offensichtliche Weg die Erfüllbarkeit einer solchen Eingabe zu testen ist das Auswerten aller 2^n unterschiedlichen Belegungen. Diese Beobachtung lässt uns nur dann auf einen Algorithmus mit polynomieller Laufzeit hoffen, wenn uns eine nicht-deterministische TM zur Verfügung steht.

Satz 1.40 (SAT $\in$ NP) *SAT kann in quadratischer Zeit auf einer TM getestet werden.*

Beweis 1.40 Wie schon gezeigt kann die Syntax in n sichergestellt werden. Die weitere Vorgehensweise ist das nicht-deterministische Generieren *einer* Belegung und die anschließende Auswertung. Dadurch generieren wir implizit *alle* Belegungen und finden die richtige falls diese existiert. Der Algorithmus ist mit Laufzeitschranken in Tabelle 1.3 auf Seite 17 angegeben. □

Der in Tabelle 1.3 angegebene Algorithmus zeigt die Auswertung sehr klar, indem er alle Rekursionsebenen der Definition 1.35 auf dem Band wiedergibt. Nach dem Kopieren der ersten Zeichen der Belegung in die Literale könnte man die Auswertung auch direkt – analog zur Syntax – in n Schritten durchführen, weil jede Klausel, die zu 1 ausgewertet wird „vergessen“ werden kann, während eine Klausel ohne ein 1 am Ende irgendeines Literals, sofort $x \notin SAT$ liefert.

Wir sehen, dass die Auswertung *deterministisch* ausgeführt wird. Die Existenz einer Belegung wird nicht-deterministisch in *polynomieller* Zeit überprüft. Es wird sich zeigen, dass diese Möglichkeit charakteristisch für NP ist.

Bezeichnung 1.41 (Vorhersagerelation, Zeuge) *Für zwei* disjunkte *endliche Alphabete Σ, Σ' nennen wir eine beliebige Relation $R \subseteq (\Sigma^* \times \Sigma'^*)$* Vorhersagerelation. *Als Interpretation seien die Zeichenfolgen aus Σ'* Zeugen *für Eigenschaften der Zeichenfolgen aus Σ.*

Befehl mit Laufzeit $O(f(n))$	$f(n)$
Überprüfe die Syntax.	n
Kopiere jedes Literal – also jeweils die Anzahl aufeinanderfolgender Literalzeichen – in die (einzige) Variable. Diese enthält anschließend das Maximum und ist nicht länger als n.	n^2
Überschreibe jedes Zeichen der Variable nicht-deterministisch wahlweise mit 0 oder 1. Dies ist die gesuchte Belegung.	n
Für jedes Literal: Kopiere die ersten Zeichen der Belegung auf das Literal und kopiere anschließend das letzte dieser Zeichen bzw. die andere Belegung im Falle von $\bar{x}$ auf alle Zeichen des Literals. Zur Laufzeit: der erste Teilbefehl entspricht einem „kopiere Eingabe“.	n^2
Für jede Klausel (also jeweils ein Bereich zwischen Zeichen aus $\{b, \wedge, Variable\}$): Überschreibe alle Zeichen mit 0; sobald ein Zeichen 1 gefunden wird, alle – auch die vorigen in diesem Bereich – mit 1. Zur Laufzeit: kein Zeichen wird öfter als 3 Mal gescannt.	n
Die Formel ist genau dann erfüllbar, wenn auf den ersten n Zeichen kein 0 (mehr) vorkommt.	n

Tabelle 1.3: Algorithmus zu SAT

Definition 1.42 (Vorhersagesprache) *Für $R \subseteq (\Sigma^* \times \Sigma'^*)$ eine Vorhersagerelation ist*

$$L_R := \{xy : \ x \in \Sigma^*, \ y \in \Sigma'^*, \ \langle x, y\rangle \in R\}$$

die zu R gehörende Vorhersagesprache.

Definition 1.43 (polynomielle Vorhersagerelation) *Eine Vorhersagerelation R ist* polynomiell, *wenn $L_R \in P$.*

Unseren Beobachtungen zufolge gilt mit obigen Bezeichnungen folgender Satz, dessen Beweisideen alle bereits erarbeitet wurden:

Satz 1.44 (Zeugen für NP)

$$\begin{aligned} NP = \{L \subseteq \Gamma^* : \ & \Gamma \textit{ beliebig, endlich } \wedge \exists p \in \mathbf{N}[n] : \exists \Sigma \textit{ endlich} : \\ & \exists R \subseteq (\Gamma^* \times \Sigma^*) \textit{ polynomielle Vorhersagerelation} : \\ & \forall x \in \Gamma^* : x \in L \Leftrightarrow \exists y \in \Sigma^{p(|x|)} : \langle x, y\rangle \in R\Big\} \end{aligned}$$

Beweis 1.44 „$\supseteq$“: Gegeben ein Polynom $p(n)$ und eine polynomielle Vorhersagerelation $R \subseteq (\Gamma^* \times \Sigma^*)$ zu einer Sprache L suchen wir eine TM, die L in polynomieller Zeit akzeptiert. Diese TM generiert zunächst nicht-deterministisch ein $y \in \Sigma^{p(|x|)}$ in der (einzigen) Variable. Die Laufzeit liegt in $O(|x|(|x| + p(|x|)))$. Anschließend verwenden wir die TM von R um den generierten Zeugen zu überprüfen.

„$\subseteq$“: Gegeben eine nicht-deterministische TM M, die L akzeptiert, mit polynomieller Laufzeitschranke $p \in \mathbf{N}[n]$ suchen wir eine polynomielle Vorhersagerelation und ein Polynom als Schranke der Länge eines Zeugen. Analog zu Beweis 1.23 suchen wir uns die maximale Anzahl k nicht-deterministischer Wahlmöglichkeiten von M. Σ sei beliebig mit k Zeichen, wir interpretieren wieder jede Berechnungsmöglichkeit von M eindeutig einem $y \in \Sigma^{p(n)}$ zugeordnet und legen fest, dass

$$\langle x, y\rangle \in R :\Longleftrightarrow \text{„}y \text{ ist eine } \textit{Berechnung mit Ergebnis} \text{ auf } M \text{ mit Eingabe } x\text{“}.$$

Das Berechnen der Relation entspricht genau der deterministischen Simulation *einer* Berechnungsmöglichkeit von M. Die Existenz einer Berechnung, die $x \in L_M$ liefert, entspricht nun genau der Existenz eines Zeugen. □

1.3.3 SAT ist NP-vollständig

SAT hat uns in natürlicher Weise eine Charakterisierung von NP geliefert, und das Testen von SAT hat keinen – zumindest keinen offensichtlichen – Algorithmus der in polynomieller Zeit auf einer *deterministischen* TM läuft. Wir werden zeigen, dass SAT schwierig ist.

Da Satz 1.40 bereits $SAT \in NP$ liefert, benötigen wir für den Nachweis der NP-Vollständigkeit von SAT nur, dass es alle Probleme aus NP enthält.

Bemerkung 1.45 *Wir wollen zeigen, dass für eine beliebige Sprache $L \in NP$ gilt: $L \leq SAT$. Dazu müssen wir gemäß Definition 1.29 eine polynomielle Transformation angeben, die zu jeder Eingabe x über dem Alphabet von L eine Formel (in CNF) liefert, die genau dann erfüllbar ist, wenn $x \in L$.*

Bezeichnung 1.46 *Folgende Objekte und Bezeichnungen seien für die weitere Konstruktion fest gewählt:*

- $L \in NP$ *eine beliebige Sprache*
- $M = \langle \Gamma = \{c_0, \ldots, c_h\}, Q = \{q_0, \ldots, q_{s-1}\}, \delta, b = c_0, q_0, q_+ = q_1\rangle$ *eine TM, die L akzeptiert: $L(M) = L$*
- $p \in \mathbf{N}[n]$ *ein Polynom, das $T_M(n) < p(n)$ beschränkt*
- $x = c_{j_1} c_{j_2} \cdots c_{j_n} \in \Gamma^*$ *eine beliebige Eingabe mit Länge $|x| = n$.*

Zu diesen Vorgaben müssen wir nur noch Variablen angeben, die M und ihr Verhalten auf x hinreichend beschreiben, und diesen Zusammenhang in der gesuchten Formel sicherstellen. Zunächst erweitern wir M, sodass die TM bei Endbildern nicht hält. Dadurch können wir exakt zum Zeitpunkt $t = p(n)$ feststellen, ob $x \in L$:

Wir fügen Q einen neuen Zustand q_s hinzu und definieren für jene $\langle c \in \Gamma, q \in Q\rangle$ mit $\delta(c, q) = \emptyset$:

$$\delta(c, q) := \begin{cases} \langle c, q_+, 1\rangle & falls\ q = q_+ \\ \langle c, q_s, 1\rangle & sonst \end{cases}$$

Die beiden Zustände q_+ und q_s haben und hatten keine Bedeutung für den Algorithmus. Sie speichern einfach die Entscheidung von M für immer – insbesondere also bis $t = p(n)$.

Tabelle 1.4 auf Seite 19 zeigt die Variablen, denen wir das ebenfalls angegebene Verhalten aufzwingen wollen. Wegen Lemma 1.17 benötigen wir für die Positionen k am Band nur den selben Indexbereich wie für die Zeit t.

Im Folgenden werden wir eine Formel $\mathcal{F}$ angeben, die das auch leistet. Sie wird aus $\mathcal{F} = \mathcal{S} \wedge \mathcal{K} \wedge \mathcal{U} \wedge Q_{p(n),1}$ zusammengesetzt, wobei die Teilformeln die Bedeutungen

Name	Indexbereich	Anzahl	gewünschte Bedeutung (wenn = 1)
$Q_{t,i}$	$0 \leq t \leq p(n)$ $0 \leq i \leq s$	$(p(n)+1) \cdot$ $\cdot (s+1)$	M ist zum Zeitpunkt t im Zustand q_i
$C_{t,k,j}$	$0 \leq t,k \leq p(n)$ $0 \leq j \leq h$	$(p(n)+1)^2 \cdot$ $\cdot (h+1)$	$t(k)$ der Inhalt des Bandes an Postion k ist zum Zeitpunkt t gleich c_j
$K_{t,k}$	$0 \leq t,k \leq p(n)$	$(p(n)+1)^2$	Der Kopf ist zum Zeitpunkt t an Position k
$D_{t,i,j,l,w,p}$	$1 \leq t \leq p(n)$ $0 \leq i,l \leq s$ $0 \leq j,w \leq h$ $p \in \{-1,1\}$ $\langle c_w, q_l, p\rangle \in \delta(c_j, q_i)$	$\leq 2 \cdot p(n) \cdot$ $\cdot (s+1)^2 \cdot$ $\cdot (h+1)^2$	Zum Zeitpunkt $t-1$ wird $\langle c_w, q_l, p\rangle \in \delta(c_j, q_i)$ angewendet
$D_{t,\mathbf{d}}$	$\mathbf{d} \in \mathcal{D}$		$D_{t,\mathbf{d}} := D_{t,i,j,l,w,p}$ mit $\mathbf{d} := \langle i,j,l,w,p\rangle$

Tabelle 1.4: Variablen zum Beweis der Komplexität von SAT

- $\mathcal{S}$: Startbild mit Eingabe x
- $\mathcal{K}$: Konsistenz der TM (zu jeder Zeit muss *genau* eine der jeweiligen Variablen den Wert 1 haben)
- $\mathcal{U}$: Übergangsfunktion (das Anwenden muss konsistent in Bedingungen und Ausführung sein)

haben. Die Klausel $Q_{p(n),1}$ garantiert, dass die Eingabe akzeptiert wird, also dass die TM zum Zeitpunkt $p(n)$ im Zustand $q_1 = q_+$ ist.

Lemma 1.47 (Startbild-Bedingung) *Die Formel*

$$\mathcal{S} := Q_{0,0} \wedge K_{0,1} \wedge C_{0,0,0} \wedge C_{0,1,j_1} \wedge \ldots \wedge C_{0,n,j_n} \wedge C_{0,n+1,0} \wedge \ldots \wedge C_{0,p(n),0}$$

leistet das gewünschte und enthält $p(n)+3$ Vorkommen von Variablen.

Beweis 1.47 Ein Startbild enthält den Zustand q_0, der Kopf ist an Position 1 und die Zeichen am Band sind $\langle c_0 = b, c_{j_1}, \ldots, c_{j_n}, b, \ldots\rangle$. $\mathcal{S}$ ist offensichtlich eine Formel nach unserer Definition. □

Lemma 1.48 (Genau-1-Bedingung) *Für beliebige Variablen $x_1, \ldots, x_N$ garantiert die Formel*

$$\mathbf{1}(x_1, \ldots, x_N) := (x_1 \vee \ldots \vee x_N) \wedge \bigwedge_{1 \leq i < j \leq N} (\overline{x}_i \vee \overline{x}_j),$$

dass genau eine der Variablen 1 ist. $\mathbf{1}(x_1, \ldots, x_N)$ enthält N^2 Vorkommen von Variablen.

Beweis 1.48 Die Klausel $x_1 \vee \ldots \vee x_N$ garantiert, dass *mindestens* eine der Variablen 1 ist, während die Klauseln $\overline{x}_i \vee \overline{x}_j$ garantieren, dass von zwei beliebigen Variablen mindestens eine 0 – also *höchstens* eine 1 – ist. Die Anzahl der Vorkommen ist $N + 2\frac{N(N-1)}{2} = N^2$. □

Lemma 1.49 (Konsistenz-Bedingung) *Die Formel*

$$\mathcal{K} := \bigwedge_{1 \leq t \leq p(n)} \mathbf{1}\left(D_{t,\mathbf{d}} : \mathbf{d} \in \mathcal{D}\right) \quad \wedge$$

$$\bigwedge_{0 \leq t \leq p(n)} \left(\mathbf{1}\left(Q_{t,0}, \ldots, Q_{t,s}\right) \wedge \mathbf{1}\left(K_{t,0}, \ldots, K_{k,p(n)}\right) \wedge \bigwedge_{0 \leq k \leq p(n)} \mathbf{1}\left(C_{t,k,0}, \ldots, C_{t,k,h}\right) \right)$$

leistet das gewünschte und enthält $O\left(p\left(n\right)\left(s^4h^4 + s^2 + p\left(n\right)^2 + p\left(n\right) \cdot h^2\right)\right)$ *Vorkommen von Variablen.*

Beweis 1.49 Bei jedem Berechnungsschritt wird genau ein Übergang angewendet, und zu jedem Zeitpunkt ist die TM in genau einem Zustand, liest genau eine Position am Band und enthält an jeder Position genau ein Zeichen. Die Beschränkung der Vorkommen ist mit Lemma 1.48 klar. □

Lemma 1.50 (Übergang-Bedingung) *Die Formel*

$$\mathcal{U} := \bigwedge_{\substack{1 \leq t \leq p\left(n\right) \\ 0 \leq k \leq p\left(n\right) \\ 0 \leq j \leq h}} \left(K_{t-1,k} \vee \overline{C}_{t-1,k,j} \vee C_{t,k,j}\right) \quad \wedge$$

$$\bigwedge_{\substack{1 \leq t \leq p\left(n\right) \\ \langle i,j,l,w,p \rangle = \mathbf{d} \in \mathcal{D}}} \left(\left(\overline{D}_{t,\mathbf{d}} \vee Q_{t-1,i}\right) \wedge \left(\overline{D}_{t,\mathbf{d}} \vee Q_{t,l}\right)\right) \quad \wedge$$

$$\bigwedge_{\substack{1 \leq t \leq p\left(n\right) \\ 0 \leq k \leq p\left(n\right) \\ \langle i,j,l,w,p \rangle = \mathbf{d} \in \mathcal{D}}} \left\{\left(\overline{D}_{t,\mathbf{d}} \vee \overline{K}_{t-1,k} \vee K_{t,k+p}\right) \wedge \left(\overline{D}_{t,\mathbf{d}} \vee \overline{K}_{t-1,k} \vee C_{t-1,k,j}\right) \wedge \left(\overline{D}_{t,\mathbf{d}} \vee \overline{K}_{t-1,k} \vee C_{t,k,w}\right)\right\}$$

leistet das gewünschte und enthält $O\left(p\left(n\right)^2 \cdot s^2h^2\right)$ *Vorkommen von Variablen.*

Beweis 1.50 Die einzelnen Klauseltypen sind wie folgt zu interpretieren:

- $[A\vee]\,\overline{B} \vee X$: *Wenn [*$\varphi\left(A\right) = 0$ *und]* $\varphi\left(B\right) = 1$, *dann muss* $\varphi\left(X\right) = 1$ gelten, damit die Klausel – und damit die Formel – zu 1 ausgewertet wird.

- $K \vee \overline{C} \vee C$: Wenn der Schreibkopf nicht da war und das Band hier dieses Zeichen enthalten hat, dann muss es dieses jetzt wieder enthalten.
- $\overline{D} \vee Q$: Wenn wir diesen Übergang anwenden wollen, dann muss der Zustand vor und nach der Anwendung diesem entsprechen.
- $\overline{D} \vee \overline{K} \vee X$: Wenn wir diesen Übergang anwenden wollen und der Schreibkopf da war, dann müssen jeweils der Kopf und das gelesene und das geschriebene Zeichen unseren Vorstellungen entsprechen.

Diese Interpretation der Klauseln als Implikation kann durch einfache Umwandlungen logischer Operatoren überprüft werden. Die Beschränkung der Vorkommen ist klar. □

Lemma 1.51 ($\mathcal{F}$ ist einfach) *Es existiert eine polynomielle Transformation f mit*

$$\begin{aligned} f : \Gamma^* &\rightarrow \Sigma^*_{SAT} \\ x &\mapsto \mathcal{F}(x) = \mathcal{S}(x) \wedge \mathcal{K}(|x|) \wedge \mathcal{U}(|x|) \wedge Q_{p(|x|),1} \end{aligned}$$

Beweis 1.51 Zunächst halten wir fest, dass die Anzahl *verschiedener* Variablen in $\mathcal{F}$ in $O\left(p(n)^2\right)$ liegt (siehe Tabelle 1.4 auf Seite 19). Lemmata 1.47–1.50 zeigen, dass die Anzahl der *Vorkommen* von Variablen in $\mathcal{F}$ in $O\left(p(n)^3\right)$ liegt. Die TM der Transformation generiert also eine Zeichenfolge einer Länge $l_{var}(n) \in O\left(p(n)^5\right)$.

Nun müssen wir nur mehr alle Wahrheitsvariablen durchzählen, also mit natürlichen Zahlen indizieren. Ein solcher Index hängt nur von $p(n) = p(|x|)$ ab und kann durch Additionen und Multiplikationen berechnet werden. Von x selbst hängen genau n Variablen aus $\mathcal{S}$ ab, wobei diese Abhängigkeit durch eine endliche Schleife mit jeweils $O(|\Gamma|)$ Durchläufen implementiert wird.

Die Syntax von SAT ist linear, also kann die Laufzeit wegen der Multiplikationen als Operation mit der längsten Laufzeit – ohne weitere Rechnung – mit $O\left(p(n)^{15}\right)$ beschränkt werden. Da Multiplikationen aber nur für die Indizes bis zu einer Größe von $O\left(p(n)^2\right)$ ausgeführt werden, reduziert sich diese Schranke auf $O\left(p(n)^6 + p(n)^5\right)$. □

Bemerkung 1.52 *Ein Algorithmus, der L löst indem er das Problem auf SAT reduziert, hat die Laufzeitschranke $O\left(p_{SAT}\left(p(n)^6\right)\right)$. Diese Vorgehensweise scheint nicht effizient, selbst wenn wir uns durch Kenntnis von Eigenheiten des Problems L deutliche Verbesserungen der Schranke erhoffen dürfen.*

Diese Überlegung wurde in [13] empirisch nachgewiesen und sogar verschärft: Die Verbesserung der Laufzeit für speziell strukturierte Instanzen, wirkt sich deutlich negativ auf die Gesamtleistung in den jeweils schlechtesten Fällen aus.

Es bleibt zu zeigen, dass diese Transformation auch wirklich die Sprache L auf SAT reduziert. Das beweisen wir anhand der folgenden Lemmata:

Lemma 1.53 ($\mathbf{x \in L \Rightarrow \mathcal{F}(x) \in SAT}$) *Wenn $x \in L$ gilt, dann ist $\mathcal{F}(x)$ erfüllbar.*

Beweis 1.53 Wenn $x \in L$, dann existiert eine Berechnung $S_0 \vdash S_1 \vdash \ldots \vdash S_{p(n)}$ mit $S_0 = S(x)$ und der Zustand in $S_{p(n)}$ ist $q_+ = q_1$.

Die erfüllende Belegung definieren wir auf den Variablen mit 1, wenn die jeweils angegebene Bedeutung in Tabelle 1.4 auf Seite 19 richtig ist, mit 0 sonst. Die Auswertung zu $\varphi(\mathcal{F}(x)) = 1$ ist einfach nachzurechnen. □

Lemma 1.54 ($\mathcal{F}(\mathbf{x}) \in \mathbf{SAT} \Rightarrow \mathbf{x} \in \mathbf{L}$) *Wenn $\mathcal{F}(x)$ erfüllbar ist, dann gilt $x \in L$.*

Beweis 1.54 Sei φ eine Belegung, sodass gilt $\varphi(\mathcal{F}(x)) = 1$. Durch φ wird auch jede Teilformel $\mathcal{S}$, $\mathcal{K}$ und $\mathcal{U}$ und jede Klausel zu 1 ausgewertet.

Wegen $\varphi(\mathcal{K}(x)) = 1$ definieren die Variablen $Q_{t,i}, C_{t,k,j}, K_{t,k}$ für jeden Zeitpunkt t eindeutig die relevanten Einträge eines Standbildes $S_t = \left\langle k, q = q_i, t = \left\langle c_{j(0)}, \ldots, c_{j(p(n))} \right\rangle \right\rangle$.

Wegen $\varphi(\mathcal{S}(x)) = 1$ ist $S_0 = S(x)$ das Startbild mit der Eingabe x.

Wegen $\varphi(\mathcal{U}(x)) = 1$ folgt $S_{t-1} \vdash S_t$ für alle $1 \leq t \leq p(n)$.

Schlußendlich ist die Folge $S(x) = S_0 \vdash S_1 \vdash \ldots \vdash S_{p(n)}$ wegen $\varphi\left(Q_{p(n),1}\right) = 1$ eine Berechnung mit Ergebnis auf der Eingabe x. Daher gilt $x \in L$. □

Satz 1.55 (SAT $\in$ NP-vollständig) *SAT ist schwierig.*

Beweis 1.55 *(Zusammenfassung)* Die in Bemerkung 1.45 aufgezeigte Vorgehensweise, eine *polynomielle Transformation* zu konstruieren, die eine beliebige Eingabe in eine im Sinne der Reduktion von Problemen äquivalente Formel umwandelt, haben wir mit Lemma 1.51 abgeschlossen.

Den Beweis dieser Äquivalenz liefern die Lemmata 1.53 und 1.54. Damit ist für ein beliebiges $L \in NP$ $L \leq SAT$ nachgewiesen. □

2 Minesweeper

Dieses Kapitel beschreibt das Spiel Minesweeper und liefert anschließend eine Betrachtung seiner Komplexität im Sinne des Kapitels 1. Die Beweisideen wurden erstmals im Jahr 2000 veröffentlicht (siehe [8]). Kaye hat seinen ursprünglichen Beweis mehrfach überarbeitet und mit neuen Ideen ergänzt, die er im Rahmen von Vorträgen präsentierte. Die Gesamtheit seiner Arbeit ist Vorlage für den Beweis, den wir hier erarbeiten.

2.1 Das Spiel

Minesweeper ist ein kleines Computer-(Denk-)Spiel das mit dem Microsoft Betriebssystem Windows[1] geliefert wird. Es existiert nahezu unverändert seit Versionen, in denen Verzeichnisse und Dateien noch in verschiedenen Steuerelementen dargestellt wurden, und ist daher einer breiten Masse von Computerbenutzern bekannt.

2.1.1 Grundlagen

Dem Spieler wird zunächst ein leeres, rechteckiges Minenfeld aus kleinen Feldern gezeigt. Seine Aufgabe besteht nun darin, alle sicheren Felder zu finden ohne dabei ein Feld mit einer Mine zu erwischen.

Abbildung 2.1 auf Seite 24 zeigt das Spiel nachdem der Spieler bereits einige Informationen gewonnen hat: Durch Öffnen einzelner sicherer Felder erfährt der Spieler, wieviele Minen auf den acht angrenzenden Feldern liegen. Ist der Spieler sicher, eine Mine lokalisiert zu haben, kann er das entsprechende Feld mit einer Fahne markieren um ein unbeabsichtigtes Öffnen zu verhindern. Die Anzeige oben links gibt an, wieviele Minen noch versteckt sind, d. h. die Anzahl der Minen im Minenfeld abzüglich der durch Fahnen markierten Felder. Die andere Anzeige zeigt die Spieldauer in Sekunden an, die Dauer eines erfolgreichen Spieles wird zur Wertung herangezogen.

Es ist klar, dass einzelne Ziffern nicht genügend Information zur Angabe weiterer sicherer Felder geben. Der Spieler muss hoffen, durch Glück (zum Beispiel durch Öffnen eines sicheren Feldes ohne angrenzende Minen) verschiedene Informationen über gleiche Felder zu erlangen, die er dann verknüpfen kann. Ein einfacher Fall ist folgender: Ein Feld hat genau so viele ungeöffnete Nachbarfelder, wie die Anzahl fehlender Minen, gefolgt von einem Feld, dessen verminte Nachbarn durch diese Information bestimmt sind.

Es lassen sich viele weitere *Standardsituationen* finden, deren Auflistung in diesem Kapitel aber nicht erforderlich ist. Vielmehr ist ein allgemeines Schema von Interesse: Der Spieler kann sich fragen, ob es zu einem Widerspruch führt, wenn ein bestimmtes

[1] *Microsoft* und *Windows* sind eingetragene Warenzeichen. Keine Aussage der wissenschaftlichen Arbeiten über Minesweeper soll als Kommentar über irgendein Produkt von Microsoft oder die Firma selbst gewertet werden.

Abbildung 2.1: Minesweeper

Feld vermint ist. Hat er einen solchen Widerspruch gefunden, kann er von einem sicheren Feld ausgehen. (Wir gehen davon aus, dass der Spieler nur sicher verminte Minen durch Fahnen markiert.)

Bemerkung 2.1 *Diese Fragestellung entspricht dem am Ende des folgenden Abschnitts definierten Minesweeper-Problem: Gibt es keinen Widerspruch in einem betrachteten Bild eines Minesweeper-Spiels?*

Offensichtlich können wir Minesweeper automatisiert spielen, wenn uns ein Algorithmus zur Beantwortung dieser Frage zur Verfügung steht. Um sicherzugehen, dass ein Feld sicher vermint ist, muss das aktuelle Bild mit allen neun möglichen Werten $0, \ldots, 8$ auf dem fraglichen Feld negativ getestet werden.

2.1.2 Modellierung

Wir suchen eine Modellierung, die bekannte mathematische Objekte verwendet um das Spiel zu beschreiben. Gleichzeitig vergeben wir Namen um leichter von diesen Objekten sprechen zu können:

Bezeichnung 2.2 *$M, N \in \mathbf{N}^+$ seien die Anzahl der* Spalten *und* Zeilen*. Wir nehmen an, dass $M \geq N > 1$. Ersteres ohne Beschränkung der Allgemeinheit, zweiteres um triviale Fälle auszuschließen*[2]*.*

Definition 2.3 (Spielfeld)

$$\mathcal{F} := M \times N = \mathbf{N}^{<M} \times \mathbf{N}^{<N}$$

Die Elemente von $\mathcal{F}$ heißen Felder*. Für ein Feld x bezeichnen $\langle x_m, x_n \rangle$ die* Koordinaten *des Feldes.*

[2]Diese Aussage impliziert *nicht*, dass *alle* Fälle mit $N = 1$ trivial sind. In Abschnitt 3.1 werden wir aber zeigen, dass sie einfach sind.

Defintion und Lemma 2.4 (Abstand, Distanz, *Metrik*)

$$\begin{aligned} \delta : \mathcal{F} \times \mathcal{F} &\to \mathbf{R} \\ \langle x, y \rangle &\mapsto \sqrt{|x_m - y_m|^2 + |x_n - y_n|^2} \end{aligned}$$

Die Bezeichnung Metrik *(auf $\mathcal{F}$) ist auch im mathematischen Sinn korrekt: δ ist*

(a) $\forall x, y \in \mathcal{F} : \delta(x, y) \geq 0 \ \wedge \ \delta(x, y) = 0 \Leftrightarrow x = y$ *positiv definit,*

(b) $\forall x, y \in \mathcal{F} : \delta(x, y) = \delta(y, x)$ *symmetrisch,*

(c) $\forall x, y, z \in \mathcal{F} : \delta(x, y) \leq \delta(x, z) + \delta(z, y)$ *und erfüllt die Dreiecksungleichung.*

Beweis 2.4 δ ist die Einschränkung der Euklidischen Metrik des $\mathbf{R}^2$ auf $\mathcal{F}$. Die Eigenschaften gelten. □

Definition 2.5 (Nachbarschaft)

$$\begin{aligned} \mathcal{N} : \mathcal{F} &\to \mathbf{P}(\mathcal{F}) \\ x &\mapsto \{y \in \mathcal{F} : \delta(x, y) < 2\} \end{aligned}$$

Wir verwenden die Bezeichnungen $\mathcal{N}(x)$ und $\mathcal{N}(X) := \bigcup_{x \in X} \mathcal{N}(x)$ um die Nachbarschaft für Felder und Mengen von Feldern anzugeben. Die folgende Charakterisierung liefert eine anschauliche Interpretation dieser Mengen:

Lemma 2.6 (Nachbarschaft)

$$\mathcal{N}(x) = \{y \in \mathcal{F} : \max\{|x_m - y_m|, |x_n - y_n|\} < 2\}$$

Beweis 2.6 Klar mit den Abschätzungen $\sqrt{2^2 + 0^2} \geq 2$ und $\sqrt{1^2 + 1^2} < 2$. □

Die *Minen* sind nun eine Teilmenge $\mathcal{M} \subseteq \mathcal{F}$ des Spielfeldes. Für eine gegebene Menge $\mathcal{M}$ definieren wir weitere Funktionen, die wir aber nur im Falle einer Mehrdeutigkeit mit dem – oder wie den – Bezeichner der Minen indizieren werden.

Bezeichnung 2.7 *Die folgenden Abkürzungen nennen wir* Minenzahl *und* Minenindikator*:*

(a) $K := |\mathcal{M}|$

(b) $\mu : \mathcal{F} \to 2 = \{0, 1\} \qquad \mu(x) := \chi_{\mathcal{M}}(x)$

Definition 2.8 (Umgebungsgefahr)

$$\begin{aligned} \sigma : \mathcal{F} &\to \mathbf{N} \\ x &\mapsto \sum_{y \in \mathcal{N}(x)} \mu(y) \end{aligned}$$

Definition 2.9 (Explosivität)

$$\begin{aligned} \epsilon : \mathcal{F} &\rightarrow \mathbf{N}^{\infty} \\ x &\mapsto \frac{\sigma(x)}{1-\mu(x)} \end{aligned}$$

Die Explosivität soll die am Spielfeld angezeigten Werte wiedergeben. Hierbei gilt zu beachten, dass aus einem sicher verminten Feld keine direkte Information über Nachbarn gewonnen werden kann. Die Punkte (e) und (f) des folgenden Lemmas rechtfertigen die Definition in dieser Form.

Lemma 2.10 (Definitionslemma) *Es gilt:*

(a) $0 \leq K \leq M \cdot N$

(b) $\sum_{x \in \mathcal{F}} \mu(x) = \sum_{x \in \mathcal{M}} \mu(x) = K$

(c) $\forall x \in \mathcal{F} : |\mathcal{N}(x)| \in \{4, 6, 9\}$

(d) $\forall x \in \mathcal{F} : 0 \leq \sigma(x) \leq 9$

(e) $\forall x \in \mathcal{F} : x \in \mathcal{M} \Leftrightarrow \epsilon(x) = \infty$

(f) $\forall x \in \mathcal{F} \setminus \mathcal{M} : \epsilon(x) = \sigma(x) \leq 8 < \infty$

Beweis 2.10

(a) Klar mit $\mathcal{M} \subseteq \mathcal{F}$.

(b) Klar mit $\mu(x) = \chi_{\mathcal{M}}(x)$.

(c) Die Werte treten auf, wenn x in der „Ecke“, am „Rand“ oder irgendwo in der „Mitte“ liegt. Weitere Überlegungen mit Lemma 2.6.

(d) Folgt direkt aus (c).

(e) Klar wegen des Nenners der Explosivität.

(f) Die Gleichheit folgt wegen des Nenners der Explosivität, die Ungleichheit mit (d) und $x \in \mathcal{F} \setminus \mathcal{M}$. □

Wenn wir im Spielverlauf das Spielfeld betrachten, können wir aber noch nicht den vollen Zusammenhang zu den bisher definierten Funktionen herstellen, daher definieren wir unabhängig von $\mathcal{M}$:

Definition 2.11 (Konfiguration) *Für ein Spielfeld $\mathcal{F}$ nennen wir ein Tripel $\mathcal{C} = \langle \mathcal{F}, \mathcal{D}, \kappa \rangle$ mit $\mathcal{D} \subseteq \mathcal{F}$ und $\kappa : \mathcal{D} \rightarrow \{0, \ldots, 8, \infty\}$* Konfiguration.

Zu beachten gilt, dass die formale Definition einer *Konfiguration* keine Aussage über ihre Gültigkeit macht, d. h. eine gegebene Konfiguration sagt nichts über die Position der Minen aus, und die Funktion κ hat a priori nichts mit der Explosivität ϵ zu tun.

Diese Beobachtung führt uns direkt zu dem in [8] formulierten Minesweeper-Problem:

Problem 2.12 (MWP) *Gegeben eine Konfiguration $\mathcal{C} = \langle \mathcal{F}, \mathcal{D}, \kappa \rangle$. Existiert eine Menge $\mathcal{M} \subseteq \mathcal{F}$ Minen, sodass $\kappa = \epsilon|_{\mathcal{D}}$?*

2.2 Komplexität

Auch hier müssen wir analog zu SAT erst einen Zusammenhang zwischen dem semantisch formulierten MWP und der Sprache herstellen:

2.2.1 Umsetzung am Computer

Das Problem bei der Umsetzung von MWP auf einer TM ist das *zweidimensionale* Spielfeld, das auf dem eindimensionalen Band repräsentiert werden soll. Klarerweise werden wir eine zeilen- oder spaltenweise Anordnung wählen – wir geben eine *Zeile* nach der anderen an. Weiters entscheiden wir uns dafür, die Zeilen nicht durch ein eigenes Zeichen zu trennen, sondern die Zeichen der letzten Spalte zu markieren. Dadurch erreichen wir $n = M \cdot N$.

Definition 2.13

$$\begin{aligned} \Sigma_\kappa &:= \{0, \ldots, 8, ?, \infty\} \\ \Sigma_{\bar{\kappa}} &:= \left\{\bar{0}, \ldots, \bar{8}, \bar{?}, \bar{\infty}\right\} \\ \Sigma_{MWP} &:= \Sigma_\kappa \cup \Sigma_{\bar{\kappa}} \end{aligned}$$

Lemma 2.14 (Syntax von MWP) *Die Syntax von MWP kann in $O(n^2)$ überprüft werden.*

Beweis 2.14 Der Algorithmus muss abgesehen vom Eingabealphabet nur testen, ob jede Zeile gleich lang ist. Dies entspricht genau $N - 1$ Vergleichen von Zeichenfolgen der Länge M. Die Laufzeit ist also beschränkt durch $O(M^2 \cdot N)$. □

Die Charakterisierung von NP in Satz 1.44 gibt uns ein einfaches Werkzeug für den Beweis, dass MWP in NP liegt. In diesem Fall werden wir aber die polynomielle Vorhersagerelation direkt in der Eingabe generieren. Dadurch können wir den weiteren Algorithmus leichter angeben.

Satz 2.15 (MWP $\in$ NP) *MWP kann in quadratischer Zeit auf einer TM getestet werden.*

Beweis 2.15 Wie schon gezeigt kann die Syntax in $O(n^2)$ sichergestellt werden. Der gesamte Algorithmus ist mit Laufzeitschranken in Tabelle 2.1 auf Seite 28 angegeben. Das Programm endet sofort, falls eine Subtraktion nicht durchgeführt werden kann. □

Der Algorithmus generiert einfach auf den nicht vorgegebenen Feldern die gesuchte Anordnung der Minen und gleichzeitig die Werte auf den sicheren Feldern. Anschließend überprüft er, ob die Zahlen auf *allen* sicheren Feldern mit der Zahl der benachbarten Minen übereinstimmt. Es ist klar, dass nicht-deterministisch viele offensichtlich falsche Werte generiert werden können, aber auch hier kommt die Stärke einer solchen TM voll zum Tragen: Wenn ein Zeuge existiert, wird er gefunden.

Befehl mit Laufzeit $O(f(n))$	$f(n)$
Überprüfe die Syntax.	n^2
Überschreibe jedes ? nicht-deterministisch mit einem der anderen Zeichen aus Σ_κ und jedes $\bar{?}$ mit einem anderen Zeichen aus $\Sigma_{\bar{\kappa}}$.	n
Für jedes Feld ohne Mine: Wenn das Zeichen links eine Mine und nicht in der letzten Spalte ist, ziehe 1 ab. Wenn das Zeichen nicht in der letzten Spalte steht und rechts eine Mine ist, ziehe 1 ab.	n
Für jedes Feld: Zähle die Zahl der Minen vom vorigen bis zum folgenden Zeichen ($\in \{0,1,2,3\}$) und ziehe diesen Wert vom selben Feld in der nächsten Zeile ab, falls dieses keine Mine ist. Markiere jeweils die aktuellen Felder der Zeile. Zur Laufzeit: Der Vorgang entspricht genau einem Kopieren oder Vergleichen von Information einer Zeile in die nächste – wie bei der Syntax.	n^2
Analog ziehe die Zahl der Minen auf den „unteren Nachbarn“ ab.	n^2
Die Konfiguration ist konsistent, wenn auf den ersten n Zeichen alle sicheren Felder den Wert 0 enthalten.	n

Tabelle 2.1: Algorithmus zu Minesweeper

2.2.2 Minesweeper ist NP-vollständig

In diesem Abschnitt werden wir beweisen, dass MWP schwierig ist. Da wir bereits gezeigt haben, dass Minesweeper in NP liegt (Satz 2.15), können wir Lemma 1.31 anwenden, das – in dieser Anwendung – besagt, dass MWP NP-vollständig ist, wenn SAT als schwieriges Referenzproblem auf Minesweeper reduzierbar ist.

Bemerkung 2.16 *Wir wollen gemäß Definition 1.29 eine polynomielle Transformation angeben, die zu jeder Formel F in CNF eine Konfiguration $\mathcal{C} = \langle \mathcal{F}, \mathcal{D}, \kappa \rangle$ liefert, die genau dann konsistent ist, wenn F erfüllbar ist.*

Das bedeutet, dass wir die gegebene Formel und mögliche Belegungen am Spielfeld codieren müssen. Die einfachste Codierung ist jeder Variable ein Feld zuzuordnen und als Belegung

$$\varphi(x) := \mu(x)$$

zu wählen. Hierbei bezieht sich der Minenindikator auf die generierten Positionen der Minen.

Anschließend brauchen wir noch ein Feld, dass abgeleitet von den Feldern der Variablen genau die Auswertung der Formel repräsentiert. Von diesem verlangen wir dann noch, dass es eine Mine enthält, und erreichen dadurch den Zusammenhang zwischen Konsistenz und Erfüllbarkeit.

In diesem Abschnitt werden wir einige Konfigurationen angeben müssen. Die offensichtlichste Darstellung ist jeweils eine Tabelle, die das Spielfeld darstellt. Der Übersicht wegen geben wir aber κ nicht direkt an, sondern ziehen von diesem Wert die angegebenen Minen der Nachbarfelder ab:

Definition 2.17 (relative Explosivität) *Für die abgebildeten Konfigurationen definieren wir*

$$\kappa(x) := \begin{cases} \infty & falls\ x = \infty\ oder\ x\ leer \\ x + |\{y \in \mathcal{N}(x) : \kappa(y) = \infty\}| & falls\ x \in \{0, \ldots, 8\} \\ undefiniert & sonst \end{cases}$$

Bemerkung 2.18 *κ ist in 2.17 wohldefiniert, weil die Definition auf sicheren vorgegebenen Feldern nur von* vorgegebenen Minen *abhängt. Der Definitionsbereich $\mathcal{D}$ von κ ist hier ebenfalls angegeben. Die Felder außerhalb von $\mathcal{D}$ werden wir mit Variablen beschriften um sie benennen zu können.*

Bemerkung 2.19 *Da Minen dazu geeignet sind, unkontrollierten Informationsfluss zu verhindern, werden wir sie sehr häufig verwenden, aber eben nicht immer mit ∞ angeben. Diese Aussage ist klar, weil Minen keine direkte Information über ihre Nachbarfelder liefern.*

Das eben bemerkte Konzept ist ganz wesentlich um Information von den Variablenfeldern gezielt zusammenzuführen. Im Folgenden geben wir an, wie wir das konkret bewerkstelligen:

Lemma 2.20 (Informationsweiterleitung) *Die Konfigurationen*

x_1	x_2	x_3	x_4	x_5
x_{16}	∞	∞	∞	x_6
x_{15}	∞	x_0	∞	x_7
x_{14}	∞	∞	∞	x_8
x_{13}	x_{12}	x_{11}	x_{10}	x_9

und

	y_0	1	y_1	y_2			y_4	
			1		1	1		
					y_3			

zeigen, dass die Information, ob ein Feld eine Mine enthält, gezielt *auf ein anderes Feld übertragen werden kann.*

Beweis 2.20 In der ersten Konfiguration erkennen wir, dass Information von Minen abgeschirmt wird. Ob die Felder x_i für $0 < i \leq 16$ Minen enthalten, hängt offensichtlich nicht davon ab, ob das Feld x_0 eine Mine enthält. Daher geben bei uns leere Felder Minen vor.

Das zweite Beispiel zeigt, dass aufgrund der Werte 1 immer nur *genau* eines der benachbarten Felder eine Mine enthalten kann. Es gilt $y_0 = \bar{y}_1 = y_2 = \bar{y}_3 = y_4$. Entscheidend hierbei ist ausschließlich, dass genau die zwei gewünschten *Variablen* Nachbarn des jeweiligen Feldes mit dem Wert 1 sind. □

Bemerkung 2.21 *Die Lage der Felder zwischen denen die Information ausgetauscht werden soll, ist bei zunehmender Distanz egal. Hierbei kann die Information auch negiert werden.*

Da in einer Formel eine Variable beliebig oft – in positiver und negierter Form – vorkommen kann, ist es auch wichtig, das Kopieren von Information sicherzustellen.

Lemma 2.22 (Informationsreplikation) *Die Konfiguration*

x_1			x_2	
	1	1	∞	x_3
	1	x_0	1	
x_6	∞	1	1	
	x_5			x_4

zeigt, dass Information (beliebig) kopiert werden kann.

Beweis 2.22 Jeder Wert 1 hat genau zwei benachbarte Variablenfelder. Es gilt daher $x_1 = \ldots = x_6 = \bar{x}_0$. Die Information *kann* an beliebigen (nicht zu nahe aneinanderliegenden) Stellen einer Informationsleitung kopiert werden. □

Entscheidend ist nun noch ein passendes Konzept um die Informationen mehrerer Variablen einer Klausel zum Wert dieser Klausel zusammenzuführen. Dabei kommt uns entgegen, dass die logische ODER-Verknüpfung „$\vee$“ assoziativ ist[3]. Daher suchen wir einen Weg auf einem vorgegebenen Feld eine Mine zu fordern, wenn auf einem von *zwei* anderen Feldern eine Mine liegt. Andernfalls wollen wir dies verhindern. Ein einfacher Ansatz ist folgender:

	$\bar{x}$	
$\bar{y}$	2	
		z

- Wenn x eine Mine enthält – also $\bar{x}$ nicht –, muss auch z eine enthalten,
- und wenn y eine Mine enthält, muss z ebenso eine enthalten.

Allerdings gibt diese Konfiguration vor, dass höchstens eines der Felder x oder y eine Mine enthalten kann. Wir führen daher eine Hilfsvariable ein, die die fehlende Mine im Fall $x = y = 1$ also $\bar{x} = \bar{y} = 0$ aufnehmen kann:

	$\bar{x}$	
$\bar{y}$	2	z
	v	

Die möglichen Belegungen sind nun jeweils zwei Minen aus vier Variablen:

$\bar{x}$	$\bar{y}$	v	z	$x \vee y$	x	y
1	1	0	0	0	0	0
1	0	1	0	1	0	1
1	0	0	1	1	0	1
0	1	1	0	1	1	0
0	1	0	1	1	1	0
0	0	1	1	1	1	1

Wir sehen, dass die Auswertung von z und $x \vee y$ genau dann nicht übereinstimmt, wenn $v = 1$ und $z = 0$ gilt. Wenn es gelingt, diese – und *nur* diese – Belegungen zu verhindern,

[3]Streng nach Definition 1.35 benötigen wir die Möglichkeit, das Maximum endlich vieler Zahlen iterativ zu ermitteln. Das ist klarerweise gegeben.

dann leistet die Konfiguration genau das gewünschte. Wir fordern also, dass höchstens eines der Felder v oder $\bar{z}$ eine Mine enthält. Eine solche Konfiguration steht uns aber bereits zur Verfügung:

$\bar{v}$	u	z
	2	

$\bar{v}$	u	z	v
1	1	0	0
1	0	1	0
0	1	1	1

In Hinblick auf Bemerkung 1.52 versuchen wir in der Konstruktion die Konfiguration zu einer gegebenen Formel klein zu halten. Folgendes Lemma gibt eine kompakte Zusammenfassung des eben Erarbeiteten wieder.

Lemma 2.23 (ODER-Verknüpfung) *Die Konfiguration*

1			1		
	$\bar{x}$	$\bar{y}$	∞	1	
	∞	2	z	∞	
	v	∞	2	∞	
	1	$\bar{v}$	u	∞	

ist genau dann konsistent, wenn $z = x \vee y$.

Beweis 2.23 Wir geben genau die konsistenten Belegungen an:

$\bar{x}$	$\bar{y}$	v	z	u	$\bar{v}$
1	1	0	0	1	1
1	0	0	1	0	1
0	1	0	1	0	1
0	0	1	1	1	0

Durch konsequentes Auflisten oder mit der vorhergegangenen Argumentation sehen wir, dass diese Belegungen genau jene sind, bei denen genau zwei der ersten vier *und* genau zwei der letzten drei *und* genau eines der Felder v und $\bar{v}$ eine Mine enthält. Die Behauptung ist damit klar. □

Bemerkung 2.24 *Die drei Felder mit dem Wert* 1 *in den ersten zwei Zeilen garantieren, dass die Information aus dieser Konfiguration auch wirklich verwendet werden kann. Außerdem zeigt diese Konfiguration bereits vor, wie allgemeine Klauseln umgesetzt werden können:*

1			1				1					1		
	$\bar{x}_1$	$\bar{x}_2$	∞	1	$\bar{z}_2$	$\bar{x}_3$	∞	1	$\cdots$	$\bar{z}_{n-1}$	$\bar{x}_n$	∞	1	
	∞	2	z_2		∞	2	z_3		$\cdots$	∞	2	z_n		
	v_2	∞	2		v_3	∞	2		$\cdots$	v_n	∞	2		
	1	$\bar{v}_2$	u_2		1	$\bar{v}_3$	u_3		$\cdots$	1	$\bar{v}_n$	u_n		

Die logische UND-Verknüpfung können wir sofort angeben:

1			1		
	x	y	∞	1	
	∞	2	$\bar{z}$	∞	
	v	∞	2	∞	
	1	$\bar{v}$	u	∞	

oder

1			1		
	$\bar{x}$	$\bar{y}$	∞	1	
	∞	2	v	∞	
	z	∞	1	∞	
	u	2	$\bar{v}$	∞	

Die erste Konfiguration erhalten wir durch Anwenden der Gleichung[4]

$$z = \min\{x, y\} = 1 - \max\{1 - x, 1 - y\}$$

für $x, y \in \{0, 1\}$, die zweite durch die Beobachtung, dass in der ODER-Konfiguration $v = x \wedge y$ gilt[5]. Diese weiteren Verknüpfungsmöglichkeiten werden wir aber gar nicht benötigen, weil wir die Auswertung der Formel nicht wirklich ausführen müssen um die Existenz einer erfüllenden Belegung nachzuweisen.

Da eine Formel genau dann zu 1 ausgewertet wird, wenn jede Klausel 1 ergibt, fordern wir einfach – statt einer Mine auf einem der Formel entsprechenden Feld – jeweils eine Mine am Ausgang z_n jeder Klausel.

Es bleibt aber noch die Frage der Anordnung dieser Klauseln am Spielfeld für eine beliebige Formel in CNF. Da jede Variable mehrfach vorkommen kann, bietet sich eine Lösung an, bei der die Inhalte der Variablen und die Klauseln parallel angeordnet sind und dann durch querverlaufende Informationsleitungen verbunden werden. Abbildung 2.2 auf Seite 33 zeigt das grundlegende Schema. Hierbei bedeuten Knoten (runde Markierungen auf Kreuzungen), dass die Information kopiert wird, während die anderen Überkreuzungen ohne wechselseitige Beeinflussung ausgeführt werden müssen.

Da eine einzelne ODER-Konfiguration die Breite 4 hat, bietet sich für die horizontal verlaufenden Variablen eine abwechselnde Anordnung von x und $\bar{x}$ an – jeweils getrennt von 1. Dadurch ist ein Zyklus von einem x zum nächsten ebenso 4 Spalten breit.

Lemma 2.25 (Kreuzung ohne Wechselwirkung) *Die Konfiguration*

		1		
	1	a		
	2	b		
x	2	y	2	z
		c		
		1		

ist genau dann konsistent, wenn $x = \bar{y} = z$ und $a = \bar{b} = c$.

Beweis 2.25 Wir suchen zunächst *alle* konsistente Belegungen bei beliebiger Vorgabe von a und x, sehen aber, dass die anderen Variablen dadurch bereits festgelegt sind: Der

[4] Es gilt $\overline{x \wedge y} = \bar{x} \vee \bar{y}$ (De Morgan). Analog findet man eine Konfiguration zur hier nicht behandelten Implikation $x \to y := \bar{x} \vee y$.

[5] Ebenso ist $u = x \leftrightarrow y$ die logische Äquivalenz und dem entsprechend $\bar{u}$ das Ausschließende Oder.

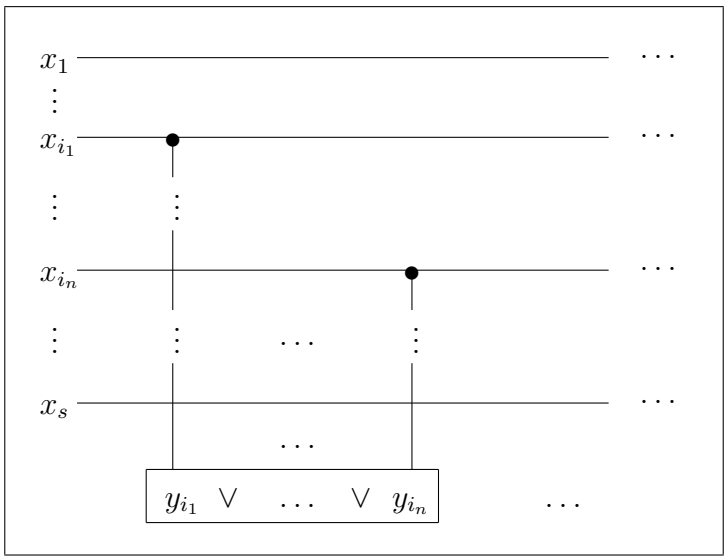

Abbildung 2.2: Formel am Spielfeld

x_i	1	$\bar{x}_i$	1

Variable horizontal

x_i	1	$\bar{x}_i$	1
1			
$\bar{x}_i$			

mit Verzweigung

x_i	2	$\bar{x}_i$	1
x_j	2		
$\bar{x}_j$	2		

mit Kreuzung

Der Wert 1 in der rechtesten Spalte wird jeweils durch 2 ersetzt, falls rechts davon eine Kreuzung steht. Umgekehrt steht in der letzten Zeile einer Kreuzung mit der untersten Variable nicht der Wert 2, sondern 1.

Abbildung 2.3: Konfigurationen für die Variable-Klausel-Zuordnung

Wert 1 in der zweiten Zeile garantiert $b = \bar{a}$. Unabhängig von deren Belegung fehlt nun eine Mine für 2 in der dritten Zeile. Daher gilt $y = \bar{x}$. Es folgt, dass $c = \bar{b}$ und $z = \bar{y}$ in dieser Reihenfolge.

Umgekehrt sehen wir leicht, dass alle Belegungen, die die Gleichungen erfüllen, konsistent sind. □

Bemerkung 2.26 *Durch Fortsetzen der obigen Argumentation kann bereits in der (nicht mehr aufscheinenden) siebenten Zeile eine weitere Variablenleitung gekreuzt werden.*

Im Folgenden werden wir diese Schlussfolgerungen von unten nach oben durchführen und in einem Algorithmus wiedergeben. Die offensichtlich auftretenden „Bausteine“ für die Zusammensetzung der zur gegebenen Formel äquivalenten Konfiguration haben im oberen Bereich der Variablen eine Breite von 4 Spalten und eine Höhe von 3 Zeilen. Abbildung 2.3 auf Seite 33 zeigt eine vollständige Auflistung dieser Teilkonfigurationen.

Abgesehen davon, dass die allerletzte Spalte entfernt werden muss um mit dem Feld 1 nicht $\bar{x}_i = 1$ festzulegen, fehlt noch eine Angabe der Schnittstelle zwischen diesen Bau-

steinen und den Klauseln aus Bemerkung 2.24. Diese muss unter anderem sicherstellen, dass das jeweils richtige Literal x_i oder $\bar{x}_i$ verwendet wird.

Lemma 2.27 (SAT $\leq$ MWP) *Es existiert eine polynomielle Transformation $f : \Sigma_{SAT} \to \Sigma_{MWP}$, die genau die erfüllbaren Formeln in konsistente Konfigurationen überführt.*

Beweis 2.27 Für eine gegebene Formel F sei c die Anzahl der Vorkommen von Variablen und s die Anzahl verwendeter Variablen. Abbildung 2.3 auf Seite 33 zeigt die Bausteine zu einer Konfiguration der Größe $M = 4c - 1$ mal $N = 3s$, deren erste Spalte in der ersten und dann jeder dritten Zeile als Variable interpretiert werden kann.

Die letzte Zeile dieser Konfiguration hat in einer konsistenten Belegung in der ersten und dann jeder vierten Spalte genau dann eine Mine, wenn die dem entsprechenden an dieser Stelle vorkommenden Literal y_i zugehörige Variable x_i den Wert $\varphi(x_i) = \mu(x_i) = 0$ hat.

Wir fügen dieser Konfiguration weitere zwei Zeilen hinzu und ergänzen auf den letzten drei Zeilen

$\bar{x}_i$	$*$		
	1		
x_i			

bzw.

$*$	$\bar{x}_i$	1	
		x_i	
$\bar{x}_i$	1		

wenn $y_i = x_i$ bzw. $y_i = \bar{x}_i$. Die erste Zeile entspricht der letzten Zeile der ursprünglichen Konfiguration. Das Sternchen $(*)$ ist dabei der Wert 1 im Falle einer „Kreuzung“ oder ∞ falls $i = s$ die Variable mit höchstem Index herangezogen wird. In keinem dieser Fälle entstehen aber irgendwelche Wechselwirkungen. Auch die fehlende letzte Spalte wird nicht benötigt.

In einer neuen Zeile fügen wir jeweils direkt unter dem ersten Literal y_1 jeder Klausel ein Feld 1 hinzu. Falls dies die einzige Klausel ist, garantiert dieses bereits eine Mine also die Auswertung zu 1. Für jedes weitere Literal der Klausel fügen wir in vier weiteren Zeilen eine Konfiguration aus Lemma 2.23 so ein, dass das rechteste Feld 1 unterhalb der Klausel in Zeile $3s + 2$ steht. Die Schnittstellen treffen dadurch aufeinander und der rechteste Wert 1 einer Klausel garantiert deren Auswertung zu eins.

Die Laufzeitschranken zum Algorithmus sind in Tabelle 2.2 auf Seite 35 angegeben. Die Äquivalenz prüfen wir leicht, weil eine konsistente Minenbelegung am Spielfeld bereits durch die unabhängige Belegung besagter Felder der ersten Spalte generiert wird. Die im unteren Bereich geforderten Minen sind genau bei einer erfüllenden aussagenlogischen Belegung gegeben. □

Satz 2.28 (MWP $\in$ NP-vollständig) *Minesweeper ist schwierig.*

Beweis 2.28 *(Zusammenfassung)* Der Satz folgt direkt aus Lemma 1.31 *(Reduktion schwieriger Probleme)* mit den Sätzen „*SAT ist schwierig*“ (1.55) und „*MWP ist polynomiell lösbar*“ (2.15) sowie Lemma „*SAT ist auf MWP reduzierbar*“ (2.27). □

Befehl mit Laufzeit $O(f(c,s)) \subseteq O(f(n,n))$	$f(c,s)$
Berechne c, als Zahl der Junktoren plus 1, und s, als Maximum der Variablenindizes. Zur Laufzeit: Es gilt $\max\{c,s\} \leq n \leq c(s+1)$. Die Schranke ist n^2.	c^2s^2
Generiere die erste Zeile der Länge $M = 4c-1$ mit den Werten ? und 1 abwechselnd. Markiere die erste und letzte Spalte (jeweils ein Fragezeichen) unterschiedlich.	c^2
Erstelle zwei weitere Zeilen, die ausschließlich Minen enthalten. Erste und letzte Spalte sind wieder markiert.	c^2
Kopiere den erstellten Block der Länge $12c-3$ genau $s-1$ Mal und die letzte Zeile weitere sieben Mal. Zur Laufzeit: Die Länge der Variablen ist durch $l_{var} \in O(cs)$ beschränkt.	c^2s^2
Das Spielfeld hat nun die Größe $M = 4c-1$ mal $N = 3s+7$. Abgesehen von der letzten ist auch die erste als „aktuelle“ Spalte markiert.	
Für jedes Literal: Führe die Befehle der unteren Tabelle aus. Die Laufzeit wird mit dem c-fachen der größten Schranke abgeschätzt, weil die Iteration explizit angegeben ist.	c^2s^2
Für jedes sichere Feld: Erhöhe den Wert um die Anzahl der benachbarten Minen. Zur Laufzeit: Der Befehl funktioniert analog zum deterministischen Teil des Minesweeper-Algorithmus (siehe Tabelle 2.1 auf Seite 28).	c^2s^2
Merke, ob das Literal x oder $\bar{x}$ ist und, ob es das erste einer Klausel ist, also rechts von b oder $\wedge$ steht.	cs
Finde den richtigen Variablenstrang. Zähle dazu auf der markierten aktuellen Spalte die ? ab.	cs^2
Setze auf die folgenden zwei Felder der aktuellen Spalte 1 und ?.	c
Solange auf dem nächsten Feld der aktuellen Spalte ein ? steht, setzte dessen Zeilennachbarn auf 2 und die drei darunterliegenden Felder der zwei nächsten zwei Zeilen auf $\infty\|?\|2$.	cs
Gehe eine Zeile zurück und erstelle die richtige der im Beweis 2.27 dargestellten Verbindungen zwischen Literal und Klausel.	c
In der folgenden Zeile füge den Wert 1 hinzu für das erste Literal einer Klausel bzw. eine ODER-Konfiguration sonst.	c
Verschiebe die aktuelle Spalte um vier nach rechts.	cs

Tabelle 2.2: Algorithmus zu Reduktion von SAT auf Minesweeper

3 Varianten

Minesweeper ist offensichtlich ein zweidimensionales Objekt. Ein naheliegender Gedanke ist die Erweiterung oder Reduktion der Dimensionalität. So entspricht etwa das nulldimensionale Minesweeperproblem einem Feld mit einem der Werte aus $\Sigma_{MW0} = \{0, \infty, ?\}$. Jede Konfiguration ist konsistent.

Dieses Kapitel beinhaltet beide und noch mehr Varianten des Spiels bzw. des Problems Minesweeper aber auch alternative Betrachtungsweisen der ursprünglichen Version.

3.1 Lineares Minesweeper

Betrachten wir zunächst die eindimensionale Variante von Minesweeper. Die einfachste Definition ist wohl das Einschränken des Spielfeldes auf eine Zeile: $N = 1$. Dabei muss aber beachtet werden, dass wir in Bezeichnung 2.2 von $N > 1$ ausgehen und daher die Sätze aus Abschnitt 2.1.2 (*Modellierung von Minesweeper*) nicht notwendig gelten.

Besonderes Interesse wollen wir aber der Tatsache schenken, dass die Syntax des Linearen Minesweeper Problems ohne Markierungen oder sonstigen Sonderzeichen auskommt, die ursprünglich zur Trennung der Zeilen dienten. Diese Sprache, also die Menge aller konsistenten eindimensionalen Minesweeperkonfigurationen, entspricht eher unserer Vorstellung einer Sprache als Menge von Wörtern, als dies beim Standardspiel der Fall war. Zu einer entsprechenden Definition benötigen wir aber noch die passenden Begriffe.

3.1.1 Automaten

Die Turing-Maschine als allgemeinen Automaten haben wir bereits definiert. Dieser Abschnitt beschreibt nun kurz und formal spezielle Automaten, die von generellem Interesse sind, sowie deren Sprachen.

Bezeichnung 3.1 (Leerwort) *Das Wort ϵ mit Länge $|\epsilon| = 0$ heißt* Leerwort.

Definition 3.2 (Endlicher Automat) *Ein Endlicher Automat (kurz: EA) ist ein Tupel $\langle \Sigma, Q, \delta, q_0, Q^+ \rangle$ mit*

- *Σ einem endlichen Eingabealphabet*
- *Q einer endlichen Menge Zustände mit $Q \cap \Sigma = \emptyset$*
- *$\delta : \Sigma \times Q \to \mathbf{P}(Q)$ der Übergangsfunktion.*
- *$q_0 \in Q$ dem Startzustand*
- *$\emptyset \neq Q^+ \subseteq Q$ einer Menge Endzustände*

Anders als bei einer TM wird die Eingabe eines EA einfach zeichenweise vorgenommen: Bei jedem Zeichen wechselt der EA in Abhängigkeit vom gelesenen Zeichen x und dem internen Zustand q in einen Zustand aus $\delta(x, q)$. Die folgenden Begriffsbildungen sind zu überdenken, aber leicht verständlich:

Definition 3.3 *Für einen EA M gilt:*

(a) M *ist* deterministisch *genau dann, wenn* $\delta : \Sigma \times Q \to Q$ *total und eindeutig ist.*

(b) M *heißt* ϵ-EA *(ϵEA) genau dann, wenn* δ *sogar auf* $(\Sigma \cup \{\epsilon\}) \times Q$ *definiert ist*[1].

(c) Als Standbild *genügt das Paar* $\langle x, q\rangle \in \Sigma^* \times Q$*, das den noch nicht gelesenen Teil der Eingabe und den aktuellen Zustand enthält.*

(d) $S(x) := \langle x, q_0\rangle$ *heißt* Startbild *zu* x*.*

(e) $\langle x, q\rangle$ *ist ein* Endbild*, wenn* $x = \epsilon$ *oder* $\delta(\pi_1(x), q) = \emptyset$*.*

(f) $\langle \epsilon, q\rangle$ *ist ein* Ergebnis*, wenn* $q \in Q^+$*.*

(g) Die Sprache von M *ist wieder* $L(M) := \{x \in \Sigma^* : \exists E\ Ergebnis : S(x) \vdash^* E\}$

(h) M *ist* äquivalent *zum EA* N *genau dann, wenn sie das selbe Eingabealphabet* $\Sigma = \Sigma_M = \Sigma_N$ *haben und die selben Wörter akzeptieren, also* $L(M) = L(N)$ *gilt.*

Lemma 3.4 (Simulation von ϵEA) *Zu jedem ϵEA gibt es einen äquivalenten (nichtdeterministischen) EA ohne ϵ-Übergänge.*

Beweis 3.4 Wir betrachten die durch δ induzierte Relation $R = \{\langle x, q, p\rangle \in (\Sigma \cup \{\epsilon\}) \times Q \times Q : p \in \delta(x, q)\}$. Zunächst entfernen wir alle Tupel der Form $\langle \epsilon, q, q\rangle$ (ϵ-Schleife), weil diese ja keine Auswirkung haben. Nun nehmen wir ein beliebiges Tupel der Form $\langle \epsilon, q, p\rangle$, entfernen es und

- falls $p \in Q^+$ setzen wir $Q^+ := Q^+ \cup \{q\}$ und
- für jedes Tupel $\langle x, p, s\rangle$ mit $x \in \Sigma \cup \{\epsilon\}$, $s \in Q$ fügen wir $\langle x, q, s\rangle$ zu R hinzu, wenn wir dieses nicht bereits entfernt hatten.

Diese Ersetzungen führen wir solange fort, bis kein Tupel mehr ϵ enthält. Mit den folgenden Überlegungen ist der Beweis fertig:

- Diese Vorgehensweise führt zu einem Ende: Die Menge $\mathbf{P}((\Sigma \cup \{\epsilon\}) \times Q \times Q)$ ist endlich, daher können wir nicht unendlich viele Tupel hinzufügen, die wir nicht bereits entfernt hatten.
- Der EA mit der durch die neue Relation R definierten Übergangsfunktion $\delta(x, q) := \bigcup_{\langle x,q,p\rangle \in R} \{p\}$ ist zum ursprünglichen äquivalent: Wir ersetzen jeden ϵ-Übergang gefolgt von einem beliebigen weiteren durch die Kombination dieser beiden. Zusammen mit der Erweiterung der Endzustände können genau die erfolgreichen Berechnungen wieder in solche übergeführt werden. □

[1] Ein Zustandswechsel *kann* also auch eintreten, wenn kein Zeichen gelesen wird.

Lemma 3.5 (Simulation nicht-deterministischer EA) *Zu jedem nicht-deterministischen EA gibt es einen äquivalenten deterministischen EA.*

Beweis 3.5 Sei $N = \langle \Sigma, Q, \delta, q_0, Q^+ \rangle$ nicht-deterministisch, dann konstruieren wir $M := \left\langle \Sigma, Q_M := \mathbf{P}(Q), \delta_M, \{q_0\}, Q_M^+ := \{q \in Q_M : q \cap Q^+ \neq \emptyset\} \right\rangle$ mit

$$\delta_M(x, q) := \bigcup_{p \in q} \delta(x, p).$$

Wegen $\emptyset \in Q_M$ ist δ_M überall definiert und M damit deterministisch. $\emptyset \in Q_M$ ist außerdem ein Fixpunkt von δ_M, M erreicht also sicher kein Ergebnis, falls N hält bevor die Eingabe ganz gelesen wurde.

Der Zustand von M der nach k Zeichen erreicht wird, enthält *genau alle möglichen* Zustände, die N nach k Zeichen erreichen kann. Dies gilt insbesondere für $k = n = |x|$ die Länge der Eingabe. Es gilt $x \in L(N)$ genau dann, wenn einer dieser Zustände in Q^+ liegt. Das entspricht genau der Definition von Q_M^+ und daraus folgt, dass $L(M) = L(N)$ M und N äquivalent sind. □

Der so konstruierte deterministische EA M enthält $|Q_M| = 2^{|Q|}$ Zustände. Diese exponentielle Steigerung konnte bei der Simulation nicht-deterministischer TM nicht in der Größe des Automaten kompensiert werden, weil die Wahlmöglichkeit durch den Schreibvorgang zu explodieren scheint – oder eigentlich: durch die Möglichkeit (nicht-deterministisch) geschriebene Zeichen wieder und wieder zu lesen und (nicht-deterministisch) erneut zu verarbeiten, ist der Aufwand doppelt exponentiell. Bevor wir aber auf den Zusammenhang zu TM genauer eingehen, folgt aus den Lemmata 3.4 und 3.5 noch ein einfaches

Korollar 3.6 (Äquivalenz Endlicher Automaten) *Die drei EA-Typen*

- *deterministischer EA ohne ϵ-Übergänge (DEA),*
- *nicht-deterministischer EA ohne ϵ-Übergänge (NEA),*
- *ϵEA*

können untereinander äquivalent ineinander übergeführt werden.

Lemma 3.7 (Simulation durch TM) *Zu jedem EA A gibt es eine deterministische TM M, die $L(A) = L(M)$ in $T_M(n) = n + 1$ akzeptiert.*

Beweis 3.7 Wir nehmen einen zu A äquivalenten DEA $N = \langle \Sigma, Q, \delta, q_0, Q^+ \rangle$ sowie ein $b \notin \Sigma$ und ein $q_+ \notin Q$ und definieren $M := \langle \Sigma \cup \{b\}, Q \cup \{q_+\}, \delta_M, b, q_0, q_+ \rangle$. Mit der Übergangsfunktion

$$\delta_M(x, q) := \begin{cases} \{\langle x, p, 1 \rangle\} & falls\ p \in \delta(x, q) \\ \{\langle b, q_+, -1 \rangle\} & falls\ x = b\ und\ q \in Q^+ \\ \emptyset & sonst \end{cases}$$

folgen die Aussagen unmittelbar. □

Wir sehen also, dass die Laufzeit für alle EA durch die Länge der Eingabe beschränkt werden kann[2]. Wegen Korollar 3.6 hängt die folgende, zu P und NP analoge Definition nicht vom Typ des EA ab:

Definition 3.8 (EA) *Die Problemklasse Endlicher Automaten ist*

$$EA := \{L \subseteq \Sigma^* : \Sigma \text{ beliebig, endlich } \wedge \exists M\ EA : L = L(M)\}$$

Lemma 3.9 (EA $\subset$ P) *P enthält EA echt.*

Beweis 3.9 Lemma 3.7 liefert sofort, dass P EA enthält. Die Gleichheit von Zeichenfolgen, oder einfacher $L := \{0^n 1^n : n \in \mathbf{N}\}$ Zeichenfolgen gleicher Länge liegen in P nicht aber in EA: Ein akzeptierender EA dürfte beim Lesen von 0^n nie zweimal den selben Zustand einnehmen, weil sonst die Information n verloren geht. Sobald also $n \geq |Q|$, schlägt jeder gewählte Algorithmus fehl. Q darf nicht von der Eingabe abhängen. □

Als Erweiterung eines EA bietet sich ein „Speicher“ an, auf dem der Automat schreiben kann. Zunächst soll dieser aber nur auf einer Seite beschrieben und gelesen werden können – beide Operationen auf der selben Seite. Einen solchen Speicher nennt man Kellerspeicher:

Definition 3.10 (Kellerautomat) *Ein Kellerautomat (kurz: KA) ist ein Tupel $\langle \Sigma, K, Q, \delta, q_0, k_0, Q^+ \rangle$ mit*

- Σ *einem endlichen Eingabealphabet*
- K *einem endlichen Kelleralphabet*
- Q *einer endlichen Menge Zustände mit* $Q \cap \Sigma = \emptyset$
- $\delta : (\Sigma \cup \{\epsilon\}) \times K \times Q \to \mathbf{P}(\Sigma \times K^*)$ *der Übergangsfunktion.*
- $q_0 \in Q$ *dem Startzustand*
- $k_0 \in K$ *dem Keller-Startzeichen*
- $\emptyset \neq Q^+ \subseteq Q$ *einer Menge Endzustände*

Die Übergangsfunktion interpretieren wir wie folgt: Der KA darf – muss aber nicht – ein Zeichen der Eingabe lesen und ersetzt das oberste/erste Zeichen im Kellerspeicher durch eine Zeichenfolge – kann also sowohl das Zeichen löschen als auch weitere Zeichen hinzufügen.

Bemerkung 3.11 *Wir dürfen davon ausgehen, dass die geschriebene Zeichenfolge die Länge* 0, 1 *oder* 2 *hat. Längere Zeichenfolgen können durch (mehrfaches) ignorieren der Eingabe der Reihe nach hinzugefügt werden.*

[2]Nur ϵ-Übergange können diese Schranke verhindern, sind aber nicht notwendig.

Außerdem darf das Keller-Startzeichen nie gelöscht oder geändert werden. Dies überlassen wir aber wieder dem Programmierer. Zunächst müssen auch hier einige Begriffe angepasst werden:

Definition 3.12 *Für einen KA M gilt:*

(a) M *ist* deterministisch *genau dann, wenn* $|\delta| \leq 1$*, und* δ *aufgefasst als Funktion auf* $\Sigma \cup \{\epsilon\}$ *– also für feste* $k \in K$ *und* $q \in Q$ *– immer entweder* ϵ *oder alle anderen Zeichen auf* $\emptyset$ *abbildet.*

(b) Ein Tripel $\langle x, k, q\rangle \in \Sigma^* \times K^* \times Q$*, das den noch nicht gelesenen Teil der Eingabe, den Inhalt des Speichers und den aktuellen Zustand enthält, heißt* Standbild.

(c) $S(x) := \langle x, k_0, q_0\rangle$ *ist* Startbild *zu* x.

(d) $\langle x, q\rangle$ *ist ein* Endbild*, wenn* $x = \epsilon$ *oder* $\delta(x_1, k_1, q) = \emptyset$.

(e) $\langle \epsilon, k, q\rangle$ *ist ein* Ergebnis*, wenn* $q \in Q^+$ *– unabhängig von* k.

(f) Die Sprache von M *ist wieder* $L(M) := \{x \in \Sigma^* : \exists E\ Ergebnis : S(x) \vdash^* E\}$

Beispiel 3.13 *Gegeben ein Eingabealphabet* Σ *und ein fest gewähltes Zeichen* $a \in \Sigma$*, dann kann die Sprache* $L := \{x_1 \dots x_n a x_n \dots x_1 : x_i \in \Sigma\}$ *offensichtlich von einem* deterministischen *KA entschieden werden.*

Beispiel 3.14 *Die Sprache* $L := \{x_1 \dots x_n x_n \dots x_1 : x_i \in \Sigma\}$ *kann offensichtlich von einem* nicht-deterministischen *KA entschieden werden.*

Definition 3.15 (DKA, KA) *Die Problemklasse (deterministischer) KA ist*

$$\begin{aligned} DKA &:= \{L \subseteq \Sigma^* : \Sigma \text{ beliebig, endlich } \wedge \exists M \text{ deterministischer } KA : L = L(M)\} \\ KA &:= \{L \subseteq \Sigma^* : \Sigma \text{ beliebig, endlich } \wedge \exists M\ KA : L = L(M)\} \end{aligned}$$

Natürlich wollen wir auch die Problemklassen von TM angeben. Dabei gilt jedoch zu beachten, dass diese nicht identisch mit P oder NP sind, weil diese Laufzeitschranken vorgeben. Ohne solche Beschränkungen gilt aber wie bei EA, dass deterministische TM nicht-deterministische simulieren können, also gleich mächtig sind.

Definition 3.16 (TM) *Die Problemklasse der TM ist*

$$TM := \{L \subseteq \Sigma^* : \Sigma \text{ beliebig, endlich } \wedge \exists M\ TM : L = L(M)\}$$

Definition 3.17 (Linear beschränkter Automat) *Ein linear beschränkter Automat (kurz: LBA) ist eine TM deren Speicherbedarf immer durch eine lineare Funktion beschränkt werden kann:*

$$\exists c \in \mathbf{N} : l_{var}(n) \leq c \cdot n$$

Definition 3.18 (LBA) *Die Problemklasse der LBA ist*

$$LBA := \{L \subseteq \Sigma^* : \Sigma \text{ beliebig, endlich } \wedge \exists M\ LBA : L = L(M)\}$$

Satz 3.19 (Sprachen der Automaten) *Es gilt:*

$$EA \subset DKA \subset KA \subset LBA \subset TM$$

Und es existiert eine Sprache, die nicht in TM liegt.

Beweis 3.19 Die ersten beiden Inklusionen gelten offensichtlich. Beispiel 3.13 zeigt, dass die erste echt ist. Wir zeigen, dass die Sprache aus Beispiel 3.14 *nicht* in DKA liegt: Ein KA kann auf einmal geschriebene Information (eine Folge mehrerer Speicherzeichen) nur zugreifen, wenn er später geschriebene Information vollständig entfernt. Die Entscheidung, ob zwei gleiche aufeinanderfolgende Zeichen die Wortmitte darstellen, muss also ohne Grundlage getroffen werden. Zu jedem Ansatz gibt es ein Gegenbeispiel.

Die Simulation eines KA durch einen LBA muss argumentiert werden, weil theoretisch beliebig viele Operationen ausgeführt werden können, ohne ein (weiteres) Zeichen der Eingabe zu lesen. Da der KA aber nur endlich viele Zustände und Kellerzeichen hat, ist die Zahl dieser Operationen entweder doch linear beschränkt oder unnötig, sodass es einen einfacheren KA gibt, der die selbe Sprache akzeptiert.

Die Sprache $L := \{a^n b^n c^n : a, b, c \in \Sigma\}$ liegt nicht in KA, weil die beim Lesen von a^n gewonnene Information n beim Überprüfen von b^n benötigt und daher zerstört wird. Daher ist $KA \subset LBA$ echt.

Die letzte Inklusion ist wieder offensichtlich, deren Echtheit wird zum Beispiel in [6] und in [16] nachgewiesen – ebenso wie die zusätzliche Aussage, dass bei richtiger Interpretation $TM \subset \mathbf{P}(\Sigma^*)$ echt gilt. □

3.1.2 Reguläre Sprachen

Dieser Abschnitt behandelt Sprachen regulärer Ausdrücke und stellt einen Zusammenhang zu Automaten her. Im Abschnitt 3.1.3 geben wir dann nicht nur Beweise, sondern auch genaue Darstellungen der Sprache des Linearen Minesweeper Problems an. Als Sprache verstehen wir dabei intuitiv eine Menge von Zeichenfolgen $L \subseteq \Sigma^*$ über einem Alphabet, und dieses Verständnis ist für die Betrachtung von Minesweeper exakt genug.

Definition 3.20 (Operationen) *Seien L_1 und L_2 Mengen (von Zeichenfolgen), dann bezeichne*

- $L_1 \cup L_2 := \{x : x \in L_1 \vee x \in L_2\}$ *die Vereinigung,*
- $L_1 L_2 := \{xy : x \in L_1 \wedge y \in L_2\}$ *das Produkt und*
- $L_1^* := \bigcup_{k \in \mathbf{N}} L_1^k = \{\epsilon\} \cup L_1 \cup L_1 L_1 \cup \ldots$ *die Iteration*

von L_1 (und L_2).

Definition 3.21 (Regulärer Ausdruck) *Ein* regulärer Ausdruck *(kurz: RA) ist ein Paar $\langle \Sigma, R \rangle$ wobei $R \in RA(\Sigma)$ rekursiv aus den Zeichen $\Sigma \cup \{\emptyset, \epsilon, (,), +, *\}$ aufgebaut ist:*

- $\emptyset \in RA(\Sigma)$

- $\epsilon \in RA(\Sigma)$
- $\Sigma \subset RA(\Sigma)$
- *Für alle* $r, s \in RA(\Sigma)$ *gilt auch:*
  - $(r + s) \in RA(\Sigma)$
  - $(rs) \in RA(\Sigma)$
  - $(r*) \in RA(\Sigma)$

Definition 3.22 (Sprache eines RA) *Für einen RA* $A = \langle \Sigma, R \rangle$ *definieren wir seine* Sprache $L(A)$ *entsprechend der Rekursion von Definition 3.21:*

- $L(\emptyset) := \emptyset$
- $L(\epsilon) := \{\epsilon\}$
- $L(a) := \{a\}$ *für* $a \in \Sigma$
- $L((r + s)) := L(r) \cup L(s)$
- $L((rs)) := L(r) L(s)$
- $L((r*)) := L(r)^*$

Wir sagen $L(A)$ *ist eine* reguläre Sprache.

Ein regulärer Ausdruck ist also ein Alphabet und eine Zeichenfolge einer bestimmten Form, die eine Sprache über diesem Alphabet definiert. Hierbei müssen wir nun drei Ebenen unterscheiden: die geschriebene Sprache dieser Arbeit, die Sprache aller RA eines Alphabets und die Sprache eines bestimmten RA. Diese Differenzierung wird aber stets aus dem Zusammenhang hervorgehen, weil RA die Zeichen + und $*$ (Sprache aller RA) statt $\cup$ und * (Sprache dieser Arbeit) enthalten und die Sprache eines bestimmten RA nur Zeichen aus Σ enthält.

Wir wollen allerdings Operatorreihenfolge und Assoziativität nützen um RA zu verkürzen: Wir vereinbaren, dass die Iteration $*$ vor dem Produkt und dieses vor der Vereinigung + ausgewertet wird. Außerdem sind $*$ und + assoziativ und können ohne Klammern geschrieben werden.

Beispiel 3.23 $A = \langle \Sigma = \{a, \ldots, z\}, R = a(m + pq)*(x + z) \rangle$ *definiert die Sprache aller Wörter aus Kleinbuchstaben, die mit a beginnen, auf x oder z enden und dazwischen aus einer beliebigen (auch leeren) Folge von m und pq bestehen.*

$$L(A) \supset \{ax, ammmz, ampqx, apqpqz\}$$

Lemma 3.24 (EA für RA) *Für jeden RA A gibt es einen* ϵ*EA M, der* $L(A) = L(M)$ *akzeptiert.*

Beweis 3.24 Wir geben entsprechend der rekursiven Zusammensetzung regulärer Ausdrücke ϵEA mit genau einem Endzustand an, die wir anschließend durch ϵ-Übergänge verbinden:

- $\emptyset \in RA(\Sigma)$: Der Automat hat einen Start- und einen Endzustand. Die Übergangsfunktion bildet immer auf die leere Menge $\emptyset$ ab: kein Wort wird akzeptiert, weil der Startzustand kein Endzustand ist.

- $\epsilon \in RA(\Sigma)$: Der Automat besteht nur aus einem Startzustand, der gleichzeitig Endzustand ist. Die Übergangsfunktion bildet immer auf die leere Menge $\emptyset$ ab: es wird kein Wort einer positiven Länge akzeptiert.

- $\Sigma \subset RA(\Sigma)$: Der Automat, der ein Wort der Länge 1 akzeptiert, hat einen Start- und einen Endzustand. Der Übergang in den Endzustand wird genau durch das zu akzeptierende Zeichen gewährleistet, während alle anderen Bilder leer sind.

- $(r+s) \in RA(\Sigma)$: Vom Startzustand ermöglichen wir je einen ϵ-Übergang zum Startzustand der Automaten von r und s, deren Endzustände wir wiederum durch zwei ϵ-Übergänge auf einen Endzustand verbinden.

- $(rs) \in RA(\Sigma)$: Wir führen einen ϵ-Übergang vom Endzustand des Automaten von r zum Startzustand von s ein.

- $(r*) \in RA(\Sigma)$: Der Automat enthält einen Startzustand, der gleichzeitig Endzustand ist. Wir fügen je einen ϵ-Übergang zum Startzustand des Automaten von r und von dessen Endzustand zurück zum neuen Zustand hinzu.

Bei der Verwendung eines zuvor konstruierten ϵEA entfernen wir dessen Endzustand aus Q^+, sodass immer genau ein Endzustand existiert. Die Korrektheit ist offensichtlich. □

Lemma 3.25 (RA für EA) *Für jeden DEA M gibt es einen RA A, der $L(M) = L(A)$ akzeptiert.*

Beweis 3.25 Wir nehmen zunächst einen beliebigen DEA $M = \langle \Sigma, Q, \delta, q_0, Q^+ \rangle$, dessen Zustände wir nach und nach entfernen, bis wir einen DEA erhalten, dessen RA direkt erfasst werden kann. Dazu betrachten wir den DEA $N = \langle RA(\Sigma), Q, \delta_{RA}, q_0, Q^+ \rangle$, der RA akzeptiert:

Für zwei Zustände q_i und q_j definieren wir genau einen Übergang $\delta_{RA}(R, q_i) = \{q_j\}$ mit

$$R := \begin{cases} (x_1 + \ldots + x_l) & falls\ x_1, \ldots, x_l \in \Sigma\ alle\ Übergänge\ \delta(x_k, q_i) = q_j\ sind \\ \emptyset & falls\ kein\ solcher\ existiert \end{cases}$$

Wie man sich leicht überzeugen kann, akzeptiert N genau alle RA, die irgendeine von M akzeptierte Zeichenfolge beschreibt. Außerdem hat N für je zwei Zustände genau einen Übergang und für jeden Zustand einen Übergang auf sich selbst.

Wir wollen nun unter Beibehaltung dieser Prämisse Zustände entfernen: Für einen festen Zustand s betrachten wir alle Paare $\langle p, q \rangle \in (Q \setminus \{s\})^2$ mit $\delta_{RA}(R, p) = \{q\}$, $\delta_{RA}(P, p) = \{s\}$, $\delta_{RA}(T, s) = \{q\}$ und $\delta_{RA}(S, s) = \{s\}$, ersetzen R durch $(R + PS*T)$ und entfernen anschließend den Zustand s.

Insgesamt entfernen wir für jeden Zustand $q \in Q^+$ alle Zustände außer q_0 und q, sodass jeweil ein EA mit einem oder zwei Zuständen entsteht. Wir vereinigen/summieren

die RA dieser EA zu dem gesuchten RA, wobei im Falle $q = q_0$ mit dem Übergang $\delta_{RA}(R, q) = \{q\}$ der RA gleich $R\circledast$ ist. Ansonsten seien die Übergänge $\delta_{RA}(R, q_0) = \{q_0\}$, $\delta_{RA}(S, q_0) = \{q\}$, $\delta_{RA}(T, q) = \{q\}$ und $\delta_{RA}(U, q) = \{q_0\}$ mit dem beschreibenden RA $(R + ST\circledast U)\circledast ST\circledast$.

Die zuletzt beschriebenen EA akzeptieren genau alle RA, die eine Zeichenfolge beschreiben, die M mit dem Zustand q akzeptiert. Die Konstruktion ist korrekt. □

Defintion und Satz 3.26 (Reguläre Sprachen) *Die Sprachen der EA sind genau die regulären Sprachen:*

$$RA := \{L \subseteq \Sigma^* : \Sigma \textit{ beliebig, endlich } \wedge \exists A\ RA : L = L(A)\} = EA$$

Beweis 3.26 Klar mit den vorhergehenden Lemmata 3.24 und 3.25. □

3.1.3 Definition und Klassifizierung

Lineares Minesweeper können wir uns leicht als einzeilige Variante des bekannten Spiels vorstellen. In diesem Abschnitt wollen wir aber speziell die Zeichenfolgen untersuchen, die bei dieser Variante auftreten, und das Problem daher auch so definieren. Bei der Klassifizierung betrachten wir zunächst Zeugen im Sinne von Bezeichnung 1.41 und werden anschließend zeigen, dass die gewonnenen Beschreibungen direkt auf das lineare Minesweeper-Problem (MW1) umgelegt werden kann.

Definition 3.27 (MW-Zeichenfolge) *Eine MW-Zeichenfolge ist eine Zeichenfolge ungleich ϵ über $\Sigma = \{0, 1, 2, \infty\}$ die nicht mit 2 beginnt oder endet und deren Zeichen 0, 1 und 2 jeweils angeben, wieviele der bis zu zwei Nachbarzeichen ∞ sind.*

Problem 3.28 (MW1) *Gegeben eine Zeichenfolge m über $\Sigma = \{0, 1, 2, \infty, ?\}$, gibt es eine Interpretation der Fragezeichen (?), sodass m eine MW-Zeichenfolge ist?*

Offensichtlich ist die Informationsweiterleitung in MW-Zeichenfolgen sehr eingeschränkt und insbesondere nicht rückkoppelbar. Abgesehen vom Zeichen 1 ist unmittelbar vorgegeben welche der Nachbarzeichen Minen enthalten. Wenn wir nun während der Verarbeitung unterscheiden ob *die* Mine eines Wertes 1 bereits davor steht oder erst folgen muss, können wir durch einfaches Scannen des Wortes – oder eben durch Anwendung eines EA – feststellen, ob es sich um eine MW-Zeichenfolge handelt.

Bemerkung 3.29 *Wir unterscheiden in der Notation der Beweise mitunter $\overline{1}$ und $\underline{1}$ um das unveränderte Zeichen 1 in einem Wort zu interpretierten.*

Lemma 3.30 (MW-Zeichenfolgen sind regulär) *Es gilt:*

$$\{m : m \textit{ ist MW-Zeichenfolge}\} \in EA$$

δ	0	1	2	∞
q_0	q_1	q_2	–	q_3
q_1	q_1	q_2	–	–
q_2	–	–	–	q_3
q_3	–	q_1	q_2	q_3

Tabelle 3.1: Übergangsfunktion für MW-Zeichenfolgen

δ	0	1	2	∞	?
q_0	q_1	q_2	–	q_3	q_1, q_2, q_3
q_1	q_1	q_2	–	–	q_1, q_2
q_2	–	–	–	q_3	q_3
q_3	–	q_1	q_2	q_3	q_1, q_2, q_3

Tabelle 3.2: Übergangsfunktion für MW1

Beweis 3.30 Wir können einen (D)EA M angeben, der genau die MW-Zeichenfolgen akzeptiert, indem wir jeweils die Menge der erlaubten Zeichen als Zustand verwenden:

$$\begin{aligned} M \quad := \langle \quad & \Sigma = \{0, 1, 2, \infty\}, \\ & Q := \left\{ q_0 := \{0, \underline{1}, \infty\}, q_1 := \{0, \underline{1}\}, q_2 := \{\infty\}, q_3 := \left\{\overline{1}, 2, \infty\right\}\right\}, \\ & \delta, q_0, Q^+ := \{q_1, q_3\}\rangle \end{aligned}$$

In Tabelle 3.1 auf Seite 45 ist die Übergangsfunktion[3] angegeben. Da immer nur erlaubte Zeichen gelesen werden, ist die Korrektheit klar. □

Von einer MW-Zeichenfolge gelangen wir zu einer gültigen MW1-Instanz, indem wir beliebige Zeichen durch Fragezeichen ersetzen:

Satz 3.31 (MW1 ∈ EA) *Lineares Minesweeper kann von EA entschieden werden.*

Beweis 3.31 Wir nehmen den EA M aus Beweis 3.30 und fügen zu jedem Übergang einen ?-Übergang hinzu. In Tabelle 3.2 auf Seite 45 ist diese *nicht-deterministische* Übergangsfunktion angegeben. □

Korollar 3.32 (MW1 ∈ P) *Lineares Minesweeper ist einfach.*

Da wir wissen, dass es einen direkten Zusammenhang zwischen EA und RA gibt, wollen wir auch einen RA für MW1 angeben. Einerseits kann dies natürlich auf die in Beweis 3.25 angegebene Weise durchgeführt werden, andererseits wird sich aber wieder eine Konstruktion über MW-Zeichenfolgen als günstig erweisen.

[3]Falls ein DEA gesucht wird, müssen die fehlenden Werte in einen weiteren – zurückweisenden – Zustand überführen.

Lemma 3.33 (RA für MW-Zeichenfolgen) *Der RA*

$$00* + (\epsilon + 0* 1)\,\infty\,(\infty + 2\infty + 10* 1\infty)*(\epsilon + 10*)$$

über $\Sigma = \{0, 1, 2, \infty\}$ *beschreibt genau die MW-Zeichenfolgen.*

Beweis 3.33 Es handelt sich offensichtlich um einen RA. Wir überlegen uns die Korrektheit anhand der Minen (∞) im Ausdruck: Entweder hat die MW-Zeichenfolge keine Mine ($00*$) oder es gibt eine erste Mine ($(\epsilon + 0* 1)\,\infty$), vor der entweder nichts oder nur beliebig viele 0 gefolgt von genau einem 1 stehen.

Der nächste Teilausdruck enthält nun genau die nächste Mine und alle Zeichenfolgen ohne Mine, die zwischen zwei Minen auftreten dürfen: $\infty + 2\infty + 10* 1\infty = (\epsilon + 2 + 10* 1)\,\infty$. Nach der letzten Mine steht wieder nichts oder ein 1 gefolgt von beliebig vielen 0. □

Bemerkung 3.34 *Bei der Umwandlung des EA in einen RA entsteht wegen der Konstruktion über die akzeptierenden Zustände ein Ausdruck, der unterscheidet, ob die MW-Zeichenfolge auf* ∞ *endet oder nicht. Diese Unterscheidung ist aber nicht zweckmäßig.*

Satz 3.35 (RA für MW1) *Der RA*

$$(0{+}?)\,(0{+}?)* + (\epsilon + (0{+}?)*\,(1{+}?))\ldots$$

$$\ldots(\infty{+}?)\,(? + \infty + 2\infty + (1{+}?)\,(0{+}?)*\,(1{+}?)\,(\infty{+}?))*(\epsilon + (1{+}?)\,(0{+}?)*)$$

über $\Sigma = \{0, 1, 2, \infty, ?\}$ *beschreibt genau MW1.*

Beweis 3.35 Der angegebene RA entspricht genau jenem aus Lemma 3.33. Jedes einzelne Zeichen y kann aber auch ein ? sein und wird also durch $(y{+}?)$ ersetzt. Bei der Ausnahme $(? + \infty + 2\infty + \ldots)*$ kann man sich einfach überlegen, dass diese Vereinfachung korrekt ist. □

3.2 Mehrdimensionales Spielfeld

In diesem Abschnitt wenden wir uns zunächst dem Begriff der Grammatik zu. Ähnlich wie bei MW1 können wir auch einen sprachlichen Aufbau von MWP vermuten. Dieses Thema werden wir aber nur kurz beschreiben und aus der Literatur zitieren und anschließend das Werkzeug für Spielfelder beliebiger (endlicher) Dimension erarbeiten.

3.2.1 MWP ist kontextsensitiv

Der formale Begriff einer *Grammatik* wurde im 20. Jahrhundert vom norwegischen Mathematiker Thue und dem amerikanischen Sprachwissenschaftler Chomsky geprägt. Hierbei werden Sprachen gebildet, indem fest vorgegebene Ersetzungen auf Wörter angewendet werden.

Bezeichnung 3.36 (Semi-Thue-System) *Eine nichtleere, endliche Relation* $\emptyset \neq R \subseteq \Sigma^* \times \Sigma^*$ *über einem Alphabet* Σ *nennen wir* Semi-Thue-System. *Ein Element* $\langle x, y \rangle \in R$ *heißt* Regel *oder* Produktion *und wird auch* $x \to y$ *geschrieben.*

Definition 3.37 (Ableitungsrelation) *Für ein Semi-Thue-System* R *über* Σ *ist ein Wort* $w \in \Sigma^*$ *aus* $v \in \Sigma^*$ *(direkt)* ableitbar*, wenn ein Teilwort entsprechend einer Regel ersetzt wird:*

$$v \vdash w :\Longleftrightarrow \exists\ u, x, y, z \in \Sigma^* : v = uxz \wedge w = uyz \wedge x \to y \in R$$

Bezeichnung 3.38 *Analog zu Definition 1.6 bezeichnen wir die reflexive (endlich-)transitive Hülle der Ableitungsrelation mit* $\vdash^*$.

Definition 3.39 (Sprache eines Semi-Thue-Systems) *In der Sprache eines Semi-Thue-Systems* R *mit einer* Startmenge $I \subseteq \Sigma^*$ *sind alle Wörter, die aus* I *durch endliche Ableitungen erreicht werden können:*

$$L(R, I) := \{w \in \Sigma^* : \exists v \in I : v \vdash^* w\}$$

Dieses Konzept ist sehr allgemein und für Klassifizierungen kaum handhabbar. Daher betrachtet man eingeschränkte Ableitungssysteme, für die wir weitere Begriffe einführen:

Definition 3.40 (Chomsky-Grammatik) *Eine Chomsky-Grammatik ist ein Tupel* $\langle \Sigma, \Gamma, R, I \rangle$ *mit*

- Σ *einem endlichen Alphabet (*Terminalsymbole*)*
- Γ *einer endlichen Menge* Nonterminalsymbole *mit* $\Sigma \cap \Gamma = \emptyset$
- R *einem Semi-Thue-System über* $\Sigma \cup \Gamma$*:* $R \subset \left\{x \to y \in (\Sigma \cup \Gamma)^2 : x \neq \epsilon\right\}$
- $I \in \Gamma$ *einem* Startsymbol

Wir übernehmen die Begriffe *Ableitung* und *Sprache* von Semi-Thue-Systemen für Chomsky-Grammatiken, wobei in der Sprache einer solchen nur Wörter aus Terminalsymbolen sind.

Definition 3.41 (Sprache einer Chomsky-Grammatik) *Gegeben* $G = \langle \Sigma, \Gamma, R, I \rangle$ *eine Chomsky-Grammatik, so bezeichnet*

$$L(G) := \{w \in \Sigma^* : I \vdash_R^* w\}$$

die Sprache *von* G.

Definition 3.42 (Chomsky-Hierachie) *Sei* $G = \langle \Sigma, \Gamma, R, I \rangle$ *eine Chomsky-Grammatik, dann heißt* G

- Typ-3- *oder* rechtslineare Grammatik*, wenn alle Regeln eine der folgenden Formen haben:*
$$A \to \epsilon, A \to a \text{ oder } A \to aB \quad \text{mit } a \in \Sigma,\ A, B \in \Gamma$$

- Typ-2- *oder* kontextfreie Grammatik*, wenn alle Regeln die folgende Form haben:*

$$A \to \psi \quad mit\ A \in \Gamma,\ \psi \in (\Sigma \cup \Gamma)^*$$

- Typ-1- *oder* kontextsensitive Grammatik*, wenn alle Regeln eine der folgenden Formen haben:*

$$\begin{aligned} \varphi_1 A \varphi_2 \to \varphi_1 \psi \varphi_2 \quad & mit\ A \in \Gamma,\ \varphi_1, \varphi_2, \psi \in (\Sigma \cup \Gamma)^*\ und\ \psi \neq \epsilon, \\ I \to \epsilon \quad & falls\ I\ in\ keiner\ rechten\ Seite\ auftritt \end{aligned}$$

- Typ-0- *oder (allgemeine)* Chomsky-Grammatik*, wenn keine weitere Einschränkung vorgegeben ist.*

Bezeichnung 3.43 (Typ-i-Sprache) *Die Sprache einer Typ-i-Grammatik ($0 \leq i \leq 3$) heißt* Typ-i-Sprache*.*

Definition 3.44 (Klasse der Typ-i-Sprachen) *Analog zu den Automaten sei*

$$\mathcal{L}_i := \{L \subseteq \Sigma^* : \Sigma\ beliebig,\ endlich\ \wedge \exists G\ Typ\text{-}i\text{-}Grammatik : L = L(G)\}$$

die Klasse der Typ-i-Sprachen.

In weiterer Folge wollen wir die Bezeichnung *Hierarchie* untermauern und einen Zusammenhang zu den Automaten herstellen, indem wir folgenden Satz zitieren[4]:

Satz 3.45 (Überblick über Sprachklassen [16]) *Es gilt:*

$$\mathcal{L}_3 = EA = RA \subset DKA \subset \mathcal{L}_2 = KA \subset \mathcal{L}_1 = LBA \subset \mathcal{L}_0 = TM\,[\subset \mathbf{P}(\Sigma^*)]$$

Letzteres bei Betrachtung der Klassen über einem festen Alphabet.

Beweis 3.45 *(Auszüge der Ideen)* Die Inklusionen wurden bereits in Satz 3.19 bewiesen, sobald die Äquivalenzen nachgewiesen sind:

$\mathcal{L}_3 = EA$: Für die Konstruktion einer rechtslinearen Grammatik aus einem DEA $\langle \Sigma, Q, \delta, q_0, Q^+ \rangle$ führt

$$\left\langle \Sigma, \Gamma := Q, R := \{p \to xq : \delta(x,p) = q\} \cup \left\{q \to \epsilon : q \in Q^+\right\}, I := q_0 \right\rangle$$

zum Ziel. Umgekehrt muss bei der analogen Konstruktion eines NEA ein zusätzlicher Endzustand für die Regeln der Form $A \to a$ (siehe Definition 3.42) eingeführt werden.

Auch die weiteren Konstruktionen von nicht-deterministischen Automaten aus einer Grammatik funktioniert nach dem gleichen Prinzip, benötigen aber kompliziertere Automaten. Die vollständigen Beweise können in [6] und [16] nachgelesen werden. □

Satz 3.46 (MWP ist kontextsensitiv) *Es existiert eine* kontextsensitive*, aber* keine kontextfreie Grammatik *zu MWP.*

[4]Die Betonung des Zitates soll den fehlenden Beweis entschuldigen.

Beweis 3.46 Zunächst halten wir fest, dass der Algorithmus zu MWP (siehe Tabelle 2.1 auf Seite 28) keine Variablen benötigt, also das Band nicht über die Ausgabe hinaus benötigt. Daher ist er für einen LBA geeignet.

Umgekehrt kann MWP nicht auf einem KA entschieden werden, da die Syntax gleich $\left(\Sigma_\kappa^{(M-1)}\Sigma_{\bar{\kappa}}\right)^N$ (siehe Definition 2.13) ist, die zunächst erarbeitete Information M aber auch bei $N > 2$ nur einmal wiederverwendet werden kann. □

Bemerkung 3.47 *Dieses Ergebnis war zu erwarten, weil einerseits MWP schwierig in NP ist und andererseits Sprachen aus $\mathcal{L}_0 \setminus \mathcal{L}_1$ in der Literatur explizit konstuiert werden mussten – sich also nicht aus praktischen Problemen ergeben haben. Eine Typ-1-Grammatik anzugeben ist aber nicht-trivial.*

3.2.2 Mehrdimensionale Turing-Maschinen

Dieser Abschnitt stellt das Konzept der *d-dimensionalen Turing-Maschinen* ($d \in \mathbf{N}^+$) vor und liefert dadurch die Basis für d-dimensionale Varianten, aber auch ein einfaches Werkzeug für allgemeines Minesweeper (Abschnitt 3.3.2). Exakte und noch viel weitergehende Ausführungen dieses Konzepts sind in [17] und den dort zitierten Quellen zu finden.

Definition 3.48 (d-dimensionale Turing-Maschine) *Eine d-dimensionale Turing-Maschine (kurz: dTM) ist eine TM mit einem d-dimensionalen Ein-/Ausgabeband, also ein Tupel $\langle \Gamma, Q, \delta, b, q_0, q_+ \rangle$ mit*

$$\delta : \Gamma \times Q \to \mathbf{P}\left(\Gamma \times Q \times \left(\{-1,0,1\}^d \setminus \{\mathbf{0}\}\right)\right)$$

einer erweiterten Übergangsfunktion.

Bezeichnung 3.49 *Wir nennen 1TM auch* lineare TM*, 2TM* quadratische TM *und 3TM auch* kubische TM*.*

Das Band einer dTM ist entlang jeder Dimension in einer Richtung unendlich. Ein *Standbild* enthält also das Band als Funktion $t : \mathbf{N}^d \to \Gamma$. Weiters gilt wieder die Konvention, dass der nicht-leere Bereich am Band immer ein d-dimensionales Rechteck ist, dessen Ecke mit den kleinsten Koordinaten in $\langle 1, \ldots, 1 \rangle$ liegt. Die *Eingabe* hat die Länge $n = n_1 \cdot \ldots \cdot n_d$, die Anzahl der nicht-leeren Felder des Startbildes, also das d-dimensionale Volumen des Rechtecks.

Bemerkung 3.50 *Die Definition der Länge der Eingabe kann auch über das Maximum oder die Summe der Kantenlängen erfolgen, weil wir wieder polynomielle Laufzeiten betrachten wollen. Das Produkt bietet sich aber in Hinblick auf die Simulation durch „normale“ eindimensionale TM an.*

Um besagte Konvention zu wahren, benötigt das Beschreiben eines Feldes, das in der Dimension j außerhalb der Eingabe liegt, einen zusätzlichen Aufwand von $O\left(\frac{n}{n_j}\right)$ zum Ausfüllen des neuen umgebenden Rechtecks. Im Allgemeinen wird diese Schranke sogar

$O(l_{var})$ für den *zusätzlichen* Aufwand aufgrund der höheren Dimension betragen, wenn l_{var} die maximale Anzahl nicht-leerer Felder beschränkt.

Der Aufwand für das (wieder)finden eines markierten Element liegt dann ebenfalls in $O(l_{var})$ – bei der Programmierung muss ein Pfad durch alle nicht-leeren Felder vorgegeben werden. Allerdings interessiert uns ohnehin der Zusammenhang zu den bekannten Problemklassen P und NP:

Definition 3.51 (Sprachen einer dTM) *Sei f eine vorgegebene* polynomiell berechenbare, bijektive Funktion *einer dTM, deren Ausgabe sich ausschließlich entlang einer Dimension erstreckt.*

Die Sprache einer dTM M unter f *ist dann das Bild unter f aller von M akzeptierten Eingaben.*

Für endliche Dimensionen d und eine endliche Eingabe existiert immer eine fest vorgegebene *Interpretation* der Eingabe als Zeichenfolge – wie in Definition 3.51 benötigt.

Es existieren für eine dTM sogar immer unendlich viele Interpretationen der Eingaben als Zeichenfolgen, weil die Eingabe zwar endlich ist aber beliebig groß sein kann. Jedoch führt eine weitere Diskussion über die Idee dieser Arbeit hinaus. Es ergibt sich meistens intuitiv eine zum Problem passende Interpretation, deren Bijektivität wir durch Begrenzungszeichen – wie für MW in Abschnitt 2.2.1 beschrieben – garantieren können.

Bezeichnung 3.52 (Sprache einer dTM) *Wir bezeichnen mit* Sprache einer dTM *die Sprache einer dTM* unter einer geeigneten Interpretation f.

Lemma 3.53 (Simulation einer 2TM) *Für eine quadratische TM N mit Laufzeit $T_N(n) \geq n$ gibt es eine lineare TM M, welche die Sprache von N in $T_M(n) \in O\left(T_N(n)^2\right)$ akzeptiert.*

Beweis 3.53 Wir vereinbaren ohne Beschränkung der Allgemeinheit, dass die zweidimensionale Eingabe von N „zeilenweise“ in einer vorgegebenen Richtung in M vorliegt. Auch die leeren Felder mit einer 0-Koordinate seien aus beweistechnischen Gründen in dieser Eingabe unter Verwendung eines weiteren[5] Leerzeichens enthalten, sodass wir keine Spezialfälle betrachten müssen und der Algorithmus von N exakt übernommen werden kann.

Zunächst muss die Konvention bezüglich des nicht-leeren Bereichs überprüft werden. Das entspricht – wie die Syntaxprüfung von MWP – einer Laufzeit von $O(n^2)$, weil alle Zeilen auf gleiche Länge getestet werden müssen. Gleichzeitig markieren wir die erste als aktuelle Spalte, also jeweils das Zeichen nach einem Zeilenwechsel – dieser ist durch ein solches neues Leerzeichen (erstes Zeichen einer Zeile) gekennzeichnet.

Nun benötigen wir nur mehr die Simulation aller Lesekopf-Verschiebungen inklusive jener aus dem nicht-leeren Rechteck hinaus. Wir werden zeigen, dass das aufgrund der Markierung der aktuellen Spalte in $O(T_N(n))$ möglich ist. Die Zustände von M sind dann aus dem Produkt der Zustände von N und jenen für die Verschiebungen. Dadurch kann der ursprüngliche Algorithmus schrittweise – getrennt durch, aber unabhängig von

[5] Es gibt ein Leerzeichen am Band, eines um die nicht benötigten Felder im nicht-leeren Rechteck zu markieren und dieses neue Leerzeichen.

diesen Lesekopf-Operationen – ablaufen. Die Laufzeit ist dann klar, weil für jeden Schritt in N einmal der Kopf verschoben werden muss.

Jede Lesekopf-Operation hat zwei Komponenten, die wir getrennt ausführen wollen. Zunächst jene, die innerhalb einer Zeile arbeitet: Falls das Zeilenende dadurch nicht erreicht wird, beträgt der Aufwand $O(l_{var}) \subseteq O(T_N(n))$ Schritte, weil die Markierung der aktuellen Spalte verschoben werden muss.

Wenn das Zeilenende erreicht wird, verschieben wir alle Zeichen in M nach der aktuellen Position um eine Stelle weiter und fügen das neue Zeichen *dieser* Zeile ein. Da N nach unserer Konvention nun alle weiteren Positionen dieser neuen Spalte besucht um sie für N als nicht-leer zu kennzeichnen, beträgt der zusätzliche Aufwand immer nur $O(l_{var})$ Schritte, wenn wir auch die Bewegung in eine andere Zeile dadurch beschränken können:

Die zweite Bewegungskomponente kann aber sehr schnell und sogar schon während des Verschiebens der aktuellen Spalte durchgeführt werden, weil in M nur das vorige oder nächste Zeichen der markierten Spalte gefunden werden muss. Insgesamt ist die Vorgehensweise damit vollständig beschrieben und beschrankt. □

Bemerkung 3.54 *Die Beschränkung der Laufzeit kann im konkreten Fall meist wesentlich verbessert werden, weil die Ausdehnung der Variablen im Allgemeinen deutlich kleiner als die Laufzeit sein wird.*

Bemerkung 3.55 *Lemma 3.53 und der folgende Satz 3.56 gelten sowohl für* deterministische *als auch* nicht-deterministische *Versionen der genannten Maschinen, weil der zeitliche Mehraufwand* deterministisch *ist.*

Satz 3.56 (Simulation einer dTM) *Für eine dTM N mit Laufzeit $T_N(n) \geq n$ gibt es eine lineare TM M, welche die Sprache von N in $T_M(n) \in O\left(T_N(n)^{2^{d-1}}\right)$ akzeptiert.*

Beweis 3.56 Wir führen einen Induktionsbeweis nach der Dimension d: Lemma 3.53 liefert uns unmittelbar die Anfangsbedingung für $d = 2$ und auch den Induktionsschritt:

Angenommen, die Aussage gilt für d. Wir wollen einen $(d+1)$TM-Algorithmus auf einer TM ausführen. Wir verwenden zunächst eine 2TM und simulieren die ersten d Dimensionen in einer Dimension dieser 2TM. Eine Lesekopf-Operation können wir nun ausführen, indem wir zuerst den Schritt in der letzten Dimension simulieren: $\pi_{d+1}(\delta_{step}(c,q))$, $c \in \Gamma, q \in Q$ hat keinen zusätzlichen Aufwand in der 2TM.

Anschließend müssen wir in der anderen Dimension, welche die Dimensionen 1 bis d enthält, $\pi_{1,\ldots,d}(\delta_{step}(c,q))$, $c \in \Gamma, q \in Q$ durchführen, was nach Induktionsvoraussetzung einen Aufwand von $O\left(T_N(n)^{2^{d-1}}\right)$ hat. Diese 2TM simulieren wir nun in einer linearen TM M in $O\left(\left(T_N(n)^{2^{d-1}}\right)^2\right) = O\left(T_N(n)^{2^{(d+1)-1}}\right)$ Schritten. Damit ist der Induktionsschritt und daher die Aussage für alle $d \in \mathbf{N}$ gezeigt. □

Bemerkung 3.57 *Wenn $T_N(n)$ ein Polynom ist, so klarerweise auch $T_N(n)^{2^{d-1}}$.*

Korollar 3.58 (dTM beweisen P und NP) *Für den Beweis der Zugehörigkeit eines Problems zu P bzw. NP kann eine (deterministische) TM beliebiger Dimension mit polynomieller Laufzeit angegeben werden.*

$$\begin{aligned} NP = \{L \subseteq \Gamma^* : \quad & \Gamma \text{ beliebig, endlich } \wedge \exists M \; dTM : \exists p \in \mathbf{N}[x] : \\ & L = L(M) \wedge \forall n \in \mathbf{N} : T_M(n) \leq p(n)\} \\ P = \{L \subseteq \Gamma^* : \quad & \Gamma \text{ beliebig, endlich } \wedge \exists M \; dTM \text{ deterministisch} : \exists p \in \mathbf{N}[x] : \\ & L = L(M) \wedge \forall n \in \mathbf{N} : T_M(n) \leq p(n)\} \end{aligned}$$

3.2.3 Folgen für Minesweeper-Varianten

Die dTM liefern uns ein einfaches Mittel zur Behandlung mehrdimensionaler MW-Varianten. Wir passen daher zunächst das MW-Modell aus Abschnitt 2.1.2 an. Natürlich gelten die Sätze aus diesem Abschnitt dann im Allgemeinen nicht mehr.

Bezeichnung 3.59 (Dimension, Länge) *Wir bezeichnen im Folgenden die Dimension des Spielfeldes immer mit d, und dessen Ausdehnung oder Länge in der Dimension $1 \leq k \leq d$ mit N_k.*

Definition 3.60 (Spielfeld)

$$\mathcal{F} := N_1 \times \ldots \times N_d$$

Die Elemente von $\mathcal{F}$ heißen Felder*. Für ein Feld x bezeichnen $\langle x_1, \ldots, x_d \rangle$ die* Koordinaten *des Feldes.*

Obwohl sich im zweidimensionalen Fall die diskrete Einschränkung der Euklidischen Metrik als brauchbar erwiesen hat, stellt sich heraus, dass die Charakterisierung über die *Maximumsmetrik* (siehe Lemma 2.6) eher unserer Vorstellung entspricht:

Definition 3.61 (Abstand, Distanz, *Metrik*)

$$\begin{aligned} \delta : \mathcal{F} \times \mathcal{F} &\to \mathbf{R} \\ \langle x, y \rangle &\mapsto \max_{1 \leq k \leq d} |x_k - y_k| \end{aligned}$$

Definition 3.62 (Nachbarschaft)

$$\begin{aligned} \mathcal{N} : \mathcal{F} &\to \mathbf{P}(\mathcal{F}) \\ x &\mapsto \{y \in \mathcal{F} : \delta(x, y) < 2\} \end{aligned}$$

Wenn man die Felder als aneinander gesetzte abgeschlossene Einheitswürfel[6] eingebettet in den $\mathbf{R}^d$ betrachtet, dann sind zwei Felder genau dann Nachbarn, wenn ihr Schnitt nicht leer ist – also zumindest einen gemeinsamen (Eck-)Punkt enthält.

Die *Minen* sind wieder eine Teilmenge $\mathcal{M} \subseteq \mathcal{F}$ des Spielfeldes. Die weiteren Begriffe bleiben ebenfalls analog erhalten.

[6] vergleiche Abbildung 2.1 auf Seite 24

Definition 3.63 (Umgebungsgefahr)

$$\begin{aligned} \sigma : \mathcal{F} &\to \mathbf{N} \\ x &\mapsto \sum_{y \in \mathcal{N}(x)} \mu(y) \end{aligned}$$

Definition 3.64 (Explosivität)

$$\begin{aligned} \epsilon : \mathcal{F} &\to \mathbf{N}^{\infty} \\ x &\mapsto \frac{\sigma(x)}{1 - \mu(x)} \end{aligned}$$

Die Explosivität soll die am Spielfeld angezeigten Werte wiedergeben. Allerdings kann die Anzahl der Minen in der Nachbarschaft höher sein:

Definition 3.65 (Konfiguration) *Für ein Spielfeld $\mathcal{F}$ nennen wir ein Tripel $\langle \mathcal{F}, \mathcal{D}, \kappa \rangle$ mit $\mathcal{D} \subseteq \mathcal{F}$ und $\kappa : \mathcal{D} \to 3^d \cup \{\infty\} = \left\{0, \ldots, 3^d - 1, \infty\right\}$* Konfiguration.

Problem 3.66 (MWd) *Gegeben eine Konfiguration $\mathcal{C} = (\mathcal{F}, \mathcal{D}, \kappa)$. Existiert eine Menge $\mathcal{M} \subseteq \mathcal{F}$ Minen, sodass $\kappa = \epsilon \mid_{\mathcal{D}}$?*

Wir können nun ein solches d-dimensionales Spielfeld einfach in einer dTM darstellen, weil das Eingabealphabet $\Sigma = 3^d \cup \{\infty, ?\}$ endlich ist. Trennzeichen werden nicht benötigt. Außerdem folgt sofort:

Lemma 3.67 (MWd $\in$ NP) *MWd kann in* linearer Zeit *auf einer dTM getestet werden.*

Beweis 3.67 Die dTM muss zunächst die ganze Eingabe durchlaufen und nichtdeterministisch die Fragezeichen (?) ersetzen. Das Überprüfen dieses möglichen Zeugen funktioniert aber ebenfalls in $O(n) = O\left(\prod_{k=1}^{d} N_k\right)$, weil jeder der maximal 3^d Nachbarn eines Feldes in jeweils maximal einem Schritt erreicht werden kann. □

Lemma 3.68 (MWd $\in$ LBA $= \mathcal{L}_1$) *MWd hat eine* kontextsensitive *Interpretation.*

Beweis 3.68 Die Simulation von dTM durch lineare TM benötigt keine zusätzlichen Variablen. Daher folgt die Aussage direkt aus Lemma 3.67. □

Lemma 3.69 (MWP $\leq$ MW2) *MWd enthält MWP für $d \geq 2$.*

Beweis 3.69 Angenommen MWP ist mit dem Eingabealphabet Σ_{MWP} laut Definition 2.13 entlang einer Dimension einer 2TM gegeben. Das Kopieren der Zeilen in eine zweidimensionales Struktur benötigt $O(n^2)$ Schritt, ebenso wie das gleichzeitige Erstellen und anschließende Löschen des überschüssigen nicht-leeren Bereichs zur Wahrung unserer Rechteck-Konvention.

Insbesondere ist MW2 aus MWP polynomiell berechenbar, selbst wenn man MWP nicht ohnedies als Interpretation von MW2 ansieht. Klarerweise ist MW2 ein Spezialfall von MWd für alle $d \geq 2$. □

Korollar 3.70 (MWd $\in$ NP-vollständig) *MWd ist schwierig für $d \geq 2$.*

3.3 Minesweeper auf Graphen

Wir haben bei MWd gesehen, dass sich die Definitionen von Umgebungsgefahr und Explosivität, aber auch jene der Konfiguration und die Fragestellung des Problems nicht ändern, wenn wir eine andere Struktur für Spielfeld und Nachbarschaft wählen. Als nächsten Schritt wollen wir diese Struktur weiter verallgemeinern, aber die Idee von Minesweeper – ein Feld gibt die Anzahl der Minen auf seinen Nachbarn bekannt – im Auge behalten.

3.3.1 Grundlagen

Wir können uns zum Beispiel einen *Graphen* vorstellen, dessen *Knoten* die Felder darstellen und dessen *Kanten* die Nachbarschaftsrelation festlegt. Der allgemeinste Fall ist wohl ein Graph, der auch *Mehrfachkanten* und *Schlingen*[7] enthalten darf. Diese Begriffe wollen wir aber zunächst definieren.

Definition 3.71 (allgemeiner Graph) *Ein allgemeiner Graph ist ein Tupel* $G = \langle V, E, \eta \rangle$ *mit*

- V *einer* Knoten*menge (Vertices)*
- E *einer* Kanten*menge (Edges) mit* $V \cap E = \emptyset$
- $\eta : E \to V^2$ *der* Inzidenzfunktion*, die jeder Kante zwei Knoten zuordnet, mit ihren Projektionen* η_A *auf den* Anfangs- *und* η_E *auf den* Endknoten*.*

Definition 3.72 (Isomorphismus für allgemeine Graphen) *Für zwei allgemeine Graphen* $G = \langle V_G, E_G, \eta \rangle$ *und* $H = \langle V_H, E_H, \rho \rangle$ *heißen zwei* bijektive *Abbildungen* $\varphi_V : V_G \to V_H$ *und* $\varphi_E : E_G \to E_H$*, welche die* Inzidenzfunktion respektieren

$$\varphi_V \circ \eta_A \equiv \rho_A \circ \varphi_E \quad \wedge \quad \varphi_V \circ \eta_E \equiv \rho_E \circ \varphi_E$$

Isomorhismus $\varphi : G \to H$ *von* G *nach* H*.*

Zwei allgemeine Graphen heißen isomorph *(*$G \cong H$*) genau dann, wenn es einen Isomorphismus zwischen ihnen gibt.*

Definition 3.73 (Abkürzungen für einen allgemeinen Graphen) *Wir sagen:*

(a) $\alpha_0 := |V|$ *ist die Anzahl der Knoten,* $\alpha_1 := |E|$ *die Anzahl der Kanten.*

(b) Für $X, Y \subseteq V$ *sind* $E(X, Y) := \{e \in E : \eta_A(e) \in X \wedge \eta_E(e) \in Y\}$ *die Kanten von der Knotenmenge* X *in die Knotenmenge* Y*.*

(c) Für $x \in V$ *ist* $d^+(x) := |E(x, V)|$ *der* Weggrad *und* $d^-(x) := |E(V, x)|$ *der* Hingrad *eines Knotens.*

(d) Für $x \in V$ *sind* $\mathcal{N}(x) := \{y \in V : E(x, y) \neq \emptyset\}$ *die* Nachbarn *eines Knotens.*

[7] Wir werden sehen, dass Schlingen keine zusätzliche Information liefern können.

Definition 3.74 (Kantenfolge) *Unter einer* Kantenfolge *verstehen wir eine endliche Folge $e_1, \ldots, e_l$ von Kanten $e_i \in E,\ 1 \leq i \leq l$ mit*

$$\forall\, 2 \leq i \leq l : \eta_A(e_i) = \eta_E(e_{i-1})$$

jeweils einer Verbindung durch einen Knoten. Diese Kantenfolge $e_1, \ldots, e_l$ hat die Länge *l und die Knoten $v_0 := \eta_A(e_1)$ und $v_i := \eta_E(e_i),\ 1 \leq i \leq l$. v_0 heißt* Anfangs- *und v_l* Endknoten.

Bezeichnung 3.75 *Wir sagen:*

(a) Anfangs- und Endknoten einer Kante sind zu dieser inzident *und zueinander* adjazent *oder benachbart.*

(b) Der Endknoten einer Kantenfolge ist von deren Anfangsknoten erreichbar.

(c) Eine Kante e geht *von $\eta_A(e)$* aus. *Sie* führt von $\eta_A(e)$ nach $\eta_E(e)$.

(d) Eine Kante e mit $\eta_A(e) = \eta_E(e)$ gleichem Anfangs- und Endknoten heißt Schlinge.

(e) Gilt für verschiedene Kanten $e_1 \neq e_2$, dass jeweils die Anfangs- und Endknoten gleich sind ($\eta_A(e_1) = \eta_A(e_2) \wedge \eta_E(e_1) = \eta_E(e_2)$), so sind diese parallel *oder* Mehrfachkanten.

(f) Gilt für verschiedene, nicht parallele Kanten $e_1 \neq e_2$, dass die mit ihr inzidenten Knoten gleich sind ($\eta_A(e_1) = \eta_E(e_2) \wedge \eta_E(e_1) = \eta_A(e_2)$), so sind diese gegenläufig.

(g) Ein Knoten x, der mit keinem anderen Knoten benachbart ist $E(x, V \setminus \{x\}) = E(V \setminus \{x\}, x) = \emptyset$, heißt isoliert.

(h) Ein Knoten x, von dem nur Kanten ausgehen oder zu dem nur Kanten hinführen, heißt sternförmig*:* $\min\{d^+(x), d^-(x)\} = 0$

Bezeichnung 3.76 (spezielle Kantenfolgen) *Wir sagen:*

(a) Eine Kantenfolge, deren letzte und erste Kante über einen Knoten verbunden sind $\eta_A(e_1) = \eta_E(e_l)$, heißt Kreis.

(b) Eine Kantenfolge, deren Kanten $e_i = e_j \Leftrightarrow i = j,\ 1 \leq i, j \leq l$ paarweise disjunkt sind, heißt Kantenzug *oder* einfach.

(c) Eine Kantenfolge, deren Anfangs- und Endknoten der Kanten

$$\eta_A(e_i) \neq \eta_A(e_j) \quad \wedge \quad \eta_E(e_i) \neq \eta_E(e_j) \qquad 1 \leq i < j \leq l$$

jeweils paarweise disjunkt sind, heißt Bahn *oder* kreuzungsfrei.

(d) Ein kreuzungsfreier Kreis heißt Zyklus.

Definition 3.77 (Eigenschaften allgemeiner Graphen) *Sei $G = \langle V, E, \eta \rangle$ ein allgemeiner Graph. G kann u.a. folgende Eigenschaften haben:*

(a) endlich: *V und E sind endlich:* $\alpha_0, \alpha_1 \in \mathbf{N}$

(b) schlingenlos: *G hat keine Schlingen.*

(c) eindeutig: *G hat keine Mehrfachkanten.*

(d) einfach: *G ist eindeutig und schlingenlos.*

(e) lokal sternförmig: *Jeder Knoten ist sternförmig:* $\forall x \in V : \min\left\{d^+(x), d^-(x)\right\} = 0$

(f) symmetrisch: *Zu jeder Kante gibt es eine gegenläufige Kante:*

$$\forall e \in E : \exists e' \in E : \eta_A(e') = \eta_E(e) \wedge \eta_E(e') = \eta_A(e)$$

(g) ungerichtet: *G ist symmetrisch und gegenläufige Kanten treten paarweise auf.*

(h) vollständig: *Von jedem Knoten gibt es je eine Kante zu jedem anderen Knoten:*

$$\forall x, y \in V : x \neq y \rightarrow (\exists e \in E : \eta_A(e) = x \wedge \eta_E(e) = y)$$

(i) stark zusammenhängend: *Jeder Knoten ist von jedem anderen Knoten* erreichbar*:*

$$\forall x, y \in V : x \neq y \rightarrow \exists e_1, \ldots, e_l \; Kantenfolge : x = \eta_A(e_1) \wedge y = \eta_E(e_l)$$

(j) zusammenhängend: *Es gibt eine Kante zwischen je zwei nicht-leeren Knotenmengen, deren Vereinigung alle Knoten enthält:*

$$\forall X, Y \subseteq V : (X, Y \neq \emptyset \wedge X \cup Y = V) \rightarrow (E(X, Y) \cup E(Y, X) \neq \emptyset)$$

(k) Baum: *G ist* minimal zusammenhängend*, also zusammenhängend, aber nach Entfernen einer beliebigen Kante nicht mehr.*

(l) Wald: *Jede Zusammenhangskomponente von G ist ein Baum.*

(m) zyklenfrei: *G hat keinen* Zyklus*.*

(n) eben: *Die Knoten von G sind Punkte der reellen Ebene $V \subseteq \mathbf{R}^2$, und die Kanten können als – bis auf Anfangs- und Endpunkte – zu diesen und zueinander disjunkte Polygonzüge zwischen ihren inzidenten Knoten dargestellt werden*[8]*.*

(o) planar: *Es gibt einen zu G isomorphen ebenen Graphen.*

Wir wollen den Umgang mit ungerichteten Graphen möglichst intuitiv gestalten. Insbesondere gilt dies für die Definition der speziellen Kantenfolgen aus Bezeichnung 3.76.

Bemerkung 3.78 (ungerichtete Graphen) *Für ungerichtete Graphen interpretieren wir die Paare gegenläufiger Kanten als* eine ungerichtete *Kante. Aus dieser Zusammenführung folgt zum Beispiel:*

[8] Wenn alle Punkte $V = \mathbf{R}^2$ Knoten sind, muss klarerweise die Kantenmenge leer sein.

- *Der* Grad *eines Knotens* $x \in V$ *ist* $d(x) := d^+(x) = d^-(x)$.
- *Es gibt* ungerichtete Bäume *mit mehr als einem Knoten.*
- *Ein Paar gegenläufiger Kanten ist* kein Zyklus.

Allerdings interessiert uns für *endliche Graphen* eine Darstellung von Mehrfachkanten, die einen Graphen bis auf Isomorphie eindeutig wiedergibt. Diese Darstellung soll auf die für uns unnötige Auflistung/Benennung der Kanten in einer Menge E verzichten.

Offensichtlich ist die Inzidenzfunktion $\eta : E \to V^2$ für *eindeutige* Graphen injektiv. Für je einen Anfangs- und einen Endknoten gibt es also maximal eine Kante, die diese verbindet. Wenn wir nun noch angeben, wieviele verschiedene solche Kanten für zwei Knoten eines allgemeinen Graphen existieren, haben wir eine Darstellung gefunden:

Defintion und Lemma 3.79 (Adjazenzmatrix für endliche Graphen) *Sei* $G = \langle V, E, \eta \rangle$ *ein endlicher Graph und* $\psi : \alpha_0 \to V$ *eine Nummerierung seiner Knoten, so heißt die Funktion*

$$\begin{array}{rccl} A: & \alpha_0 \times \alpha_0 & \to & \mathbf{N} \\ & \langle i, j \rangle & \mapsto & |E(\psi(i), \psi(j))| \end{array}$$

(allgemeine) Adjazenzmatrix *von* G. G *kann aus* A *bis auf Isomorphie* rekonstruiert *werden.*

Beweis 3.79 Gegeben eine Adjazenzmatrix $A : \alpha_0 \times \alpha_0 \to \mathbf{N}$: Wir definieren

$$H := \left\langle V_H := \alpha_0, E_H := \bigcup_{i,j=0}^{\alpha_0 - 1} (\{i\} \times \{j\} \times A(i,j)), \eta_H := \rho := \langle \pi_1(E_H), \pi_2(E_H) \rangle \right\rangle.$$

Offensichtlich kann $\psi = \varphi_V$ als Teil des Isomorphismus von H auf G herangezogen werden. $\{i\} \times \{j\} \times A(i,j)$ enthält genau $A(i,j) = |E_G(\psi(i), \psi(j))|$ Elemente und die ersten beiden Komponenten geben den Start- bzw. Endknoten an. Das garantiert die Existenz der gesuchten Funktion $\varphi_E : E_H \to E_G$ auf den Kanten und damit einen Isomorphismus zwischen den beiden Graphen. □

Bemerkung 3.80 *Die endlichen Graphen können also auch als endliche,* vollständige, eindeutige *Graphen mit einer* Bewertungsfunktion *nach* $\mathbf{N}$ *für die „Breite“ einer Kante interpretiert werden.*

Bemerkung 3.81 *Die verbale Definition eines* gerichteten *Graphen („paarweises Auftreten gegenläufiger Kanten“) kann durch die* Symmetrie der Adjazenzmatrix $A = A^T$ *und* gerade Diagonaleinträge $A(i,i) \in 2\mathbf{N}$, $i \in \alpha_0$ *formalisiert werden.*

3.3.2 Allgemeines Minesweeper

Für Minesweeper ergeben sich aus dem vorigen Abschnitt die möglichen Verallgemeinerungen

- *beliebige Nachbarschaft* und
- *bewertete Nachbarschaft.*

Wir wollen gleich beide Möglichkeiten wahrnehmen, aber in Hinblick auf die Klassifizierung nur endliche Varianten definieren um diese auf einer TM lösen zu können. Außerdem verwenden wir die Darstellung eines endlichen Graphen $G = \langle V, A \rangle$ durch seine Adjazenzmatrix. Insbesondere verfügen wir also über eine *kanonische* Nummerierung der Knoten.

Bezeichnung 3.82 (Allgemeines Minesweeper) *Ein endlicher Graph $G = \langle \mathcal{F}, \mathcal{N} \rangle$ heißt* Spielfeld *und jeder Knoten $x \in \mathcal{F}$* Feld*. $\mathcal{N}$ definiert die bewertete Nachbarschaft in dem Sinn, dass Minen auf Nachbarfeldern, zu denen mehrere Kanten führen, gefährlicher sind.*

Bezeichnung 3.83 (Minen) Minen *sind eine Teilmenge $\mathcal{M} \subseteq \mathcal{F}$ der Felder.*

$$\begin{array}{rccl} \mu: & \mathcal{F} & \to & 2 = \{0,1\} \\ & x & \mapsto & \chi_{\mathcal{M}}(x) \end{array}$$

heißt Minenindikator.

Definition 3.84 (Umgebungsgefahr)

$$\begin{array}{rccl} \sigma: & \mathcal{F} & \to & \mathbf{N} \\ & x & \mapsto & \displaystyle\sum_{y \in \mathcal{F}} \mathcal{N}(x,y)\,\mu(y) \end{array}$$

Die Explosivität müssen wir diesmal explizit statt analytisch definieren: In der Definition von MWP war jedes Feld sein eigener Nachbar. Dadurch konnte der undefinierte Ausdruck $0/0$ nicht auftreten.

Definition 3.85 (Explosivität)

$$\begin{array}{rccl} \epsilon: & \mathcal{F} & \to & \mathbf{N}^{\infty} \\ & x & \mapsto & \begin{cases} \sigma(x) & falls\ \mu(x) = 0 \\ \infty & sonst \end{cases} \end{array}$$

Lemma 3.86 (Definitionslemma) *Es gilt:*

(a) $\forall x \in \mathcal{F} : 0 \leq \sigma(x) \leq (|\mathcal{F}| \max \mathcal{N}) \in \mathbf{N}$

(b) $\forall x \in \mathcal{F} : x \in \mathcal{M} \Leftrightarrow \epsilon(x) = \infty$

Beweis 3.86 Die erste Aussage (a) ist eine triviale Abschätzung der Summe $\sum_{y \in \mathcal{F}} \mathcal{N}(x,y)\,\mu(y)$. Letztere (b) haben wir absichtlich so gewählt. □

Definition 3.87 (Konfiguration) *Für ein Spielfeld $G = \langle \mathcal{F}, \mathcal{N} \rangle$ nennen wir ein Tupel $\mathcal{C} = \langle \mathcal{F}, \mathcal{N}, \mathcal{D}, \kappa \rangle$ mit $\mathcal{D} \subseteq \mathcal{F}$ und $\kappa : \mathcal{D} \to \mathbf{N}^\infty$* Konfiguration.

Problem 3.88 (MWG) Minesweeper auf Graphen: *Gegeben eine Konfiguration $\mathcal{C} = \langle \mathcal{F}, \mathcal{N}, \mathcal{D}, \kappa \rangle$. Existiert eine Menge $\mathcal{M} \subseteq \mathcal{F}$ Minen, sodass $\kappa = \epsilon\,|_{\mathcal{D}}$?*

In weiterer Folge wollen wir die Komplexität von MWG untersuchen. Dazu benötigen wir zuerst eine Darstellung für eine Maschine. Das komplexeste Objekt einer Konfiguration $\mathcal{C} = \langle \mathcal{F}, \mathcal{N}, \mathcal{D}, \kappa \rangle$ ist sicher die Nachbarschaft $\mathcal{N} : \mathcal{F} \times \mathcal{F} \to \mathbf{N}$, weil sie *drei* nicht a priori beschränkte Komponenten hat.

Wir wählen daher für MWG – und allgemein für die Darstellung von Graphen – eine 3TM. Das Quadrat mit Seitenlänge $\alpha_0 = |\mathcal{F}|$ am Anfang der Ebene der ersten beiden Dimensionen interpretieren wir als *Adjazenzmatrix*, deren Einträge in der dritten Dimension angegeben sind.

Für MWG hängen wir entlang der ersten Dimension die Werte von κ an die Adjazanzmatrix an. Dafür verwenden wir ein anderes Alphabet, sodass wir uns diese beiden Darstellungen sogar *ineinander* vorstellen können. Natürliche Zahlen seien wieder durch Aufzählen codiert, dann ergibt sich

$$\Sigma = \{1, 0\} \times \{\bar{1}, \bar{0}, \infty, ?\}$$

als Eingabealphabet für MWG auf einer 3TM.

Lemma 3.89 (Darstellung von MWG) *Eine Instanz von MWG kann auf einer 3TM als Eingabe der Länge $n = \alpha_0^2 \max \mathcal{N}$ dargestellt werden. Die Syntax kann in linearer Zeit $O(n)$ überprüft werden.*

Beweis 3.89 Wir übernehmen die eben beschriebene und in Abbildung 3.1 auf Seite 60 angegebene Idee, die Beschriftung der Felder κ dem Quader der Adjazenzmatrix einzuschreiben. Lemma 3.86(a) garantiert uns, dass dieser ausreicht: die gefaltete Fläche zur Darstellung von κ hat eine Höhe von $\alpha_0 \cdot \max \mathcal{N}$.

Die Syntax beinhaltet genau die folgenden Forderungen: Der Quader hat eine quadratische Grundfläche. Funktionswerte werden immer in einem – möglicherweise gefalteten – *eindimensionalen* Teil der 3TM angegeben: Natürliche Zahlen k sollen durch k aufeinanderfolgende Werte 1 angegeben sein, die bis zur maximalen Höhe durch 0 ergänzt sind. Die Funktionswerte *Mine* (∞) und *unbestimmtes Feld* (?) sollen immer die volle Höhe ausfüllen.

Die Forderung nach der quadratischen Grundfläche überprüfen wir mit einem Durchlauf entlang der Diagonale: dieser muss genau am Eck enden. Die korrekte Darstellung der Funktionen kann für jeden Wert unabhängig und egal ob von oben oder von unten getestet werden. Wenn wir nun die gefaltete Fläche spaltenweise – also zunächst in die Höhe – durchlaufen, stellen wir sowohl die Gültigkeit dieser Funktionswerte fest, also auch jener der Matrix: bei jedem „Knick“ wechseln wir in eine andere Säule der Adjazenzmatrix.

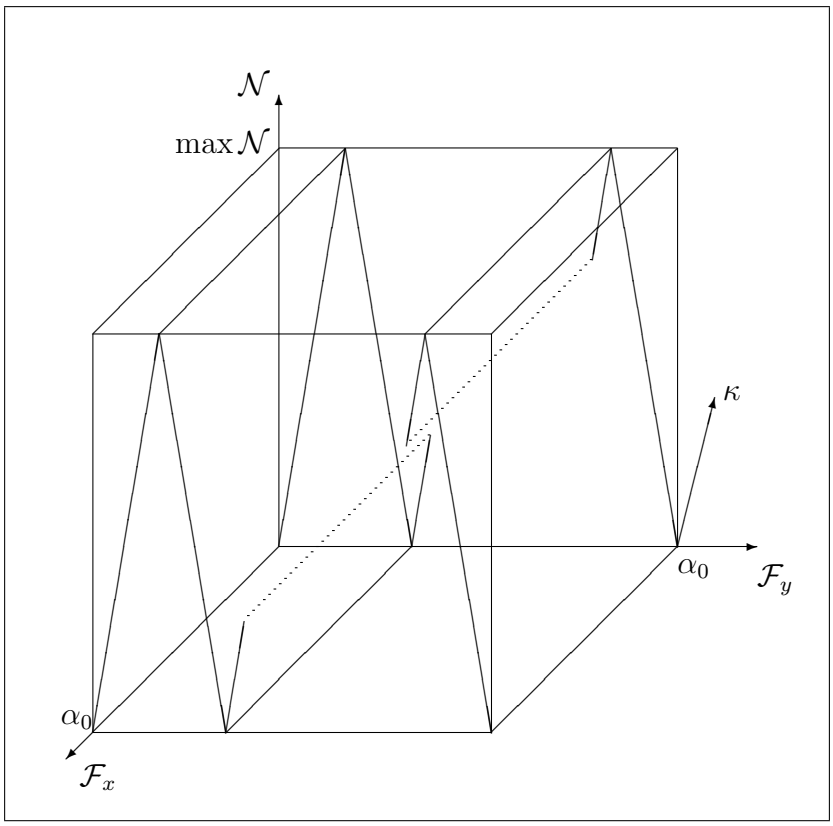

Abbildung 3.1: Darstellung von MWG auf einer 3TM

Der eben beschriebene Algorithmus liest einmal jedes Diagonalelement der Grundfläche, 3 weitere – leere – Bandfelder um die Ecke festzustellen und anschließend jedes Element genau einmal und $\alpha_0^2 + \alpha_0 + 1$ leere Bandfelder bei den Richtungswechseln. □

Bemerkung 3.90 (Darstellung eindeutiger Graphen) *Wenn das Spielfeld ein eindeutiger Graph ist, also* $\max \mathcal{N} = 1$ *gilt, kann MWG nach dem selben Schema auf einer 2TM dargestellt werden.*

Es ist klar, dass MWG nicht einfacher als MWP ist. Trotzdem werden wir einen Algorithmus angeben, der zu einer gegebenen MW2-Instanz den entsprechenden Graphen konstruiert:

Lemma 3.91 (MWP = MW2 ≤ MWG) *Auf einer 2TM kann der MW-Graph zu einem Standardbrett in* quadratischer Zeit *berechnet werden.*

Beweis 3.91 Gegeben ist ein rechteckiges Spielfeld über dem Alphabet $\Sigma_{MW2} = \{0, \ldots, 8, \infty, ?\}$. Die Aufgabe besteht nun aus den folgenden drei Schritten:

(1) Kopieren *aller* Felder in eine Dimension.

(2) Ausmessen des Quadrates und codieren von κ über $\pi_2(\Sigma_{MWG}) = \{\bar{0}, \bar{1}, \infty, ?\}$.

(3) Erstellen der *Adjazenzmatrix*.

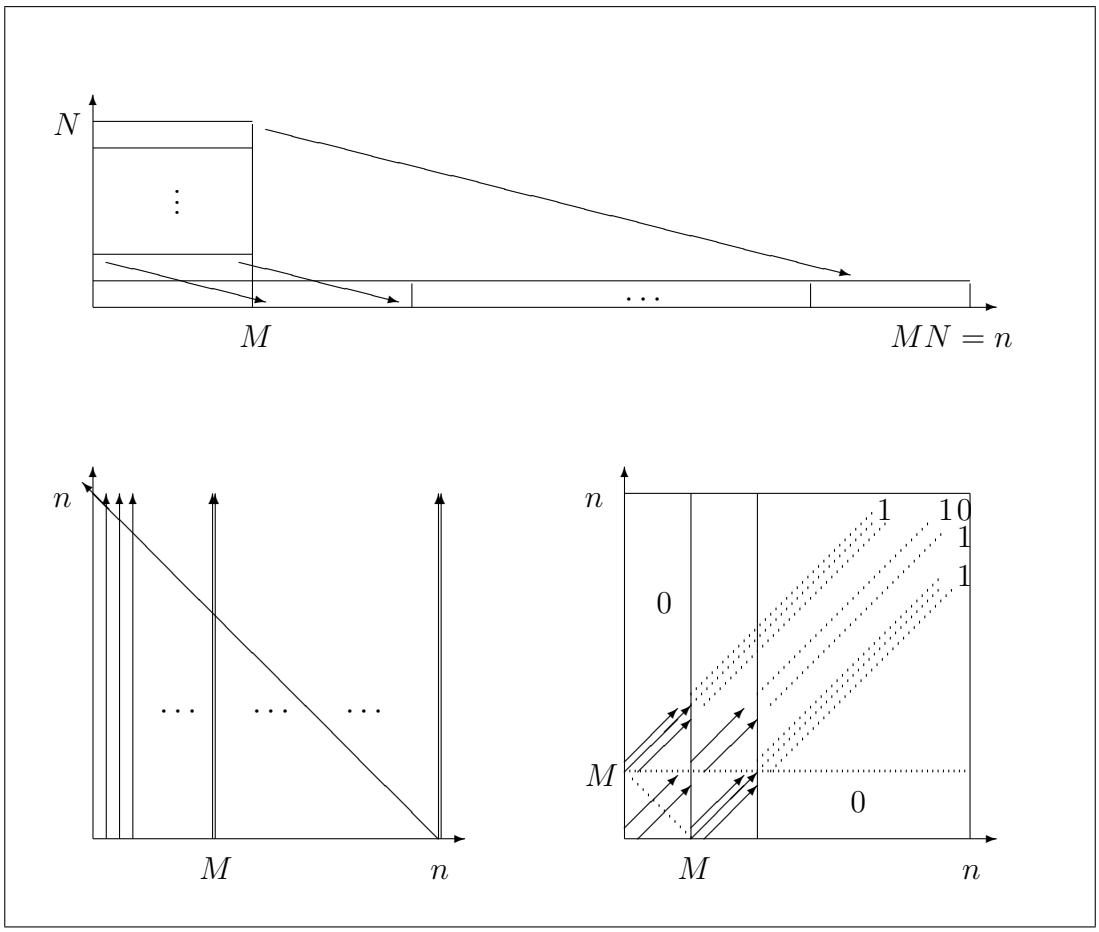

Abbildung 3.2: Algorithmus zur Reduktion von MW2 auf MWG

Es ist klar, dass zur Darstellung von MW2 ein *eindeutiger* Graph genügt. Die Länge $n = MN$ der Eingabe ist die Anzahl der Felder, jene der Ausgabe deren Quadrat n^2. Wenn beim Kopieren und Eintragen von κ auch die Information über das Zeilenende auf die ganze Höhe durch eine entsprechende Markierung übertragen wird, kann jeder der drei angegebenen Schritte, die in Abbildung 3.2 auf Seite 61 illustriert sind, *linear für die Ausgabe* durchgeführt werden – daraus resultiert die quadratische Laufzeit. □

Erstaunlicher als diese Reduktion ist die folgende Beobachtung, dass MWG auch nicht schwieriger als MWP ist, weil es auf einer nicht-deterministischen TM in polynomieller Zeit gelöst werden kann. Der Beweis gelingt für die Darstellung aus Lemma 3.89 auf einer 3TM:

Satz 3.92 (MWG ∈ NP) *MWG kann auf einer 3TM in* quadratischer Zeit *gelöst werden.*

Beweis 3.92 Offensichtlich können die fehlenden Werte außerhalb der Konfiguration $\kappa(\mathcal{F} \setminus \mathcal{D})$ nicht-deterministisch in $O(n)$ Schritten generiert werden, weil wir eine obere Schranke der Umgebungsgefahr σ haben. Es bleibt die Überprüfung des Zeugen:

Bei genauer Betrachtung der Darstellung in Abbildung 3.1 auf Seite 60 erkennen wir, dass die zu vergleichenden Werte $\kappa(x) = \sum_{y \in \mathcal{F}} \mathcal{N}(x, y)\, \mu(y)$ durch die Anzahl der

Befehl mit Laufzeit $O(f(\alpha_0, \max \mathcal{N}))$	$f(\alpha_0, \max \mathcal{N})$
Für jedes Feld x entlang der $\mathcal{F}_x$-Achse ohne Mine ($\kappa(x) \neq \infty$):	α_0
Finde x auf der $\mathcal{F}_y$-Achse durch eine Diagonalquerung mit $\delta_{step} = (-1, 1, 0)$	α_0
Setze alle Stellen von $\mathcal{N}$ in der gefundenen $\mathcal{F}_x \times \mathcal{N}$-Ebene gleich 0	$\alpha_0 \max \mathcal{N}$
Die Gesamtlaufzeit liegt in $O(n) = O(\alpha_0^2 \max \mathcal{N})$	$\alpha_0 (\alpha_0 + \alpha_0 \max \mathcal{N})$

Tabelle 3.3: Algorithmus zu Berechnung von $\mathcal{N}(x, y)\,\mu(y)$

Werte 1 und $\bar{1}$ in der jeweiligen $\mathcal{F}_y \times \mathcal{N}$-Ebene repräsentiert werden, wenn wir die Kanten $\mathcal{N}(x, y)$ zu sicheren Feldern $\mu(y) = 0$ entfernen. Diese Berechnung von $\mathcal{N}(x, y)\,\mu(y)$ ist in *linearer* Zeit möglich und in Tabelle 3.3 auf Seite 62 angegeben.

Die weitere Vorgehensweise ist einfach: Teste die Anzahl der Werte 1 und $\bar{1}$ in jeder $\mathcal{F}_y \times \mathcal{N}$-Ebene auf Gleichheit. Die Laufzeit ist $O\left(\alpha_0 (\alpha_0 \max \mathcal{N})^2\right) \subseteq O(n^2)$. □

Da der Zeuge wieder mit den n bereits durch die Eingabe belegten Feldern auskommt, und mit MW2 als Spezialfall von MWG (Lemma 3.91) folgen sofort zwei weitere Ergebnisse:

Korollar 3.93 (MWG $\in$ LBA $= \mathcal{L}_1$) *MWG hat eine* kontextsensitive *Interpretation.*

Korollar 3.94 (MWG $\in$ NP-vollständig) *MWG ist schwierig in NP.*

3.3.3 Spezielle Graphen

Die obere Schranke der Komplexität von MWG ist nun zufriedenstellend gefunden. Wie bei der eindimensionalen Variante, stellt sich aber die Frage, was *notwendig* ist, um alle Probleme aus NP simulieren zu können. Wir versuchen in diesem Abschnitt Klassen von Graphen zu finden, für die MWG *einfach* oder bereits *schwierig* im Sinne der NP-Vollständigkeit ist.

Lemma 3.95 (unnötige Kanten) *Kanten, die von einem* verminten Feld ausgehen *oder auf ein* sicheres Feld führen*, können ohne Informationsverlust aus einem MW-Graphen entfernt werden.*

Beweis 3.95 Betrachten wir zunächst verminte Felder x mit $\mu(x) = 1$: Die Explosivität $\epsilon = \infty$ enthält keine Information über die Nachbarfelder. Daher ist es egal ob und welche Felder Nachbarn einer Mine sind.

Auf der anderen Seite liefert $\mathcal{N}(x, y)\,\mu(y)$ keinen Beitrag zur Umgebungsgefahr $\sigma(x)$ des Anfangsfeldes einer Kante, falls $\mu(y) = 0$ das Endfeld sicher ist. □

Eine Schlinge geht vom selben Feld aus, zu dem sie führt. Daher geht sie immer entweder von einem verminten Feld aus oder führt auf ein sicheres Feld und es folgt sofort:

Korollar 3.96 (Schlingen sind unnötig) Schlingen *können ohne Informationsverlust aus einem MW-Graphen entfernt werden.*

Lemma 3.97 (mehr unnötige Kanten) *Kanten, die auf ein* vorgegebenes Feld $x \in \mathcal{D}$ führen*, können ohne Informationsverlust aus einem MW-Graphen entfernt werden.*

Beweis 3.97 Dieses Lemma muss wegen Lemma 3.95 nur mehr für vorgegebene *verminte* Endfelder $y \in \mathcal{D}$ bewiesen werden – und zwar für sichere Anfangsfelder. Falls auch das Anfangsfeld $x \in \mathcal{D}$ vorgegeben ist, subtrahieren wir einfach vom Feldwert $\kappa(x) \mapsto \kappa(x) - \mathcal{N}(x, y)$ die Zahl der Kanten, die wir entfernen und erhalten dadurch die Information für die anderen Nachbarn von x. Diese Vorgehensweise entspricht der relativen Explosivität für MWP aus Definition 2.17.

Wenn nun das Anfangsfeld $x \notin \mathcal{D}$ *nicht* vorgegeben ist, müssen wir zeigen, dass es eine – mit der Explosivität ϵ identische – konsistente Fortsetzung von κ auf das ganze Spielfeld $\mathcal{F}$ genau dann gibt, wenn eine solche nach dem Entfernen der Kanten existiert. Falls diese Fortsetzung $\kappa(x) = \infty$ enthält, kann die Kante wegen Lemma 3.95 entfernt werden. Ansonsten beweisen wir diese Äquivalenz in beide Richtungen:

Angenommen sie existiert für den ursprünglichen Graphen, dann muss $\kappa(x) \geq \mathcal{N}(x, y)$ gelten, und die Voraussetzungen bleiben für den neuen Graphen erhalten, wenn man die Kanten entfernt und deren Anzahl vom Funktionswert abzieht.

Angenommen die konsistente Fortsetzung existiert im neuen Graphen ohne die fraglichen Kanten, so können beliebig endlich viele Kanten bei gleichzeitiger Erhöhung des Funktionswertes problemlos hinzugefügt werden. Insbesondere können wir dadurch eine Entsprechung auf dem ursprünglichen Graphen finden. □

Lemma 3.98 (isolierte Felder) *Felder auf* isolierten Knoten *(oder alle anderen Knoten) können ohne Informationsverlust aus einem MW-Graphen entfernt werden.*

Beweis 3.98 Falls das isolierte Feld $x \in \mathcal{D}$ vorgegeben und vermint ist oder keine Umgebungsgefahr $\kappa(x) = 0$ voraussetzt, kann es entfernt werden, weil *dieses* Feld nicht über die Konsistenz entscheidet. Falls jedoch eine positive Umgebungsgefahr $\kappa(x) = k > 0$ gefordert wird, kann diese nie erreicht werden: Die Instanz ist ungültig und somit äquivalent zur einfachsten ungültigen Instanz, einem einzelnen Feld ohne Nachbarn aber mit einer positiven Beschriftung.

Ist das isolierte Feld nicht vorgegeben, so entscheidet es ebenfalls nicht über die Konsistenz, weil das Festsetzen seines Wertes auf eine Mine, die Funktion auf den anderen Feldern – wie bereits überlegt – nicht beeinflusst. □

Wir haben nun gezeigt, dass Kanten *von einer Mine* und *zu einem vorgegebenen Feld* und *isolierte Felder* entfernt werden können. Daraus folgt sofort:

Korollar 3.99 (Minen sind unnötig) *Felder mit* vorgegebenen Minen *können ohne Informationsverlust aus einem MW-Graphen entfernt werden.*

Wir haben jetzt bereits einige Vereinfachungen der MW-Graphen gefunden, die die Konsistenz einer Instanz nicht beeinflussen. Daher können diese Vereinfachungen eine untere Schranke der Komplexität nicht überschreiten. Der Beweis dieser Schranke – der Vollständigkeit in NP – für MWG ist mit der folgenden Reduktionskette beliebiger Probleme $NP \ni L \leq (CNF)SAT \leq MWP = MW2 \leq MWG$ sehr schnell gelungen. Es liefert also bereits die Reduktion von SAT-MWP-Instanzen auf MWG dessen

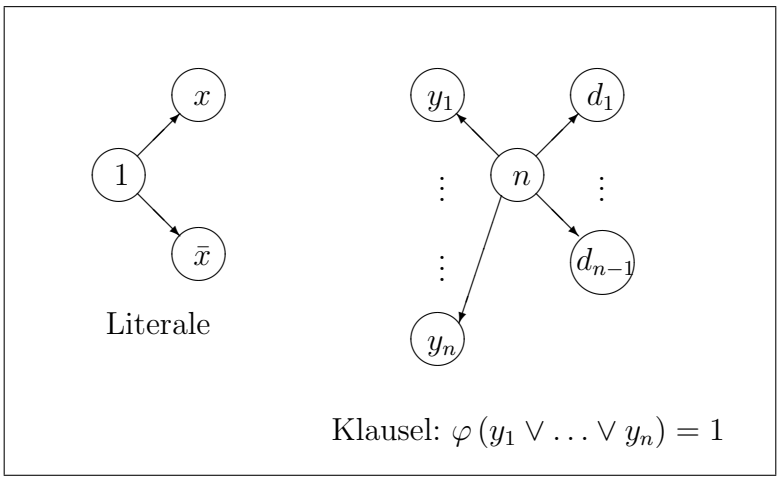

Abbildung 3.3: Einfaches Konzept für $SAT \leq MWG$

Schwierigkeit. Durch einen direkteren Beweis und unter Beachtung des gefundenen Einsparungspotentials können wir nun aber kleinere Klassen von Graphen angeben, die bereits NP-vollständig sind.

Lemma 3.100 ($\mathbf{SAT \leq MWG_{einfach,lokal\ sternförmig}}$) *Die Reduktion von SAT auf MWG benötigt nur* lokal sternförmige *Graphen ohne* Schlingen, parallelen *und* gegenläufigen Kanten[9]*.*

Beweis 3.100 Gegeben eine Formel F mit c Vorkommen von s verschiedenen Variablen in m Klauseln. Wir geben einen einfachen MW-Graphen mit lauter sternförmigen Knoten an, der genau dann eine konsistente Minenverteilung hat, wenn F eine erfüllende Belegung hat.

Dazu benötigen wir zwar keine *Informationsleitungen* und *Kreuzungen* mehr, wie in MWP, aber die *Negation* und die *Auswertung jeder Klausel* zu 1. Die Negation, also die Bereitstellung *aller* möglicher Literale, garantieren uns je drei Knoten pro Variable: $x \leftarrow 1 \rightarrow \bar{x}$. Wegen Lemma 3.97 sind alle Knoten sternförmig.

Für eine Klausel $y_1 \vee \ldots \vee y_n$ mit n Literalen benötigen wir ein Feld z, dass mit $\kappa(z) = n$ beschriftet ist und als einfache Nachbarn die entsprechenden Literale und genau $n-1$ „Dummy“-Felder hat. Diese funktionieren nach dem „Reise nach Jerusalem“-Prinzip: Jedes der Literale kann eine Mine enthalten, wenn aber jedes versucht diese los zu werden, bleibt eines über. Es gilt daher $\varphi(y_1 \vee \ldots \vee y_n) = 1$.

Dieses Konzept ist in Abbildung 3.3 auf Seite 64 dargestellt. Der Graph ist offensichtlich *einfach* und außerdem *lokal sternförmig*. Er enthält $\alpha_0 = 3s + c$ Knoten und $\alpha_1 = 2s + 2c - m$ Kanten und ist daher in polynomieller Zeit generierbar. □

Es fällt sofort auf, dass die durch den eben geführten Beweis generierten Graphen, zwar einfach und klein sind, dafür aber Knoten mit hohen *Hin-* und *Weggraden* enhalten können. Bei dem Umweg über MWP sind diese zumindest mit 8 beschränkt. Es wird sich aber zeigen, dass diese Schranke deutlich verkleinert werden kann.

[9]Sternförmige Knoten sind offensichtlich nicht zu gegenläufigen Kanten inzident.

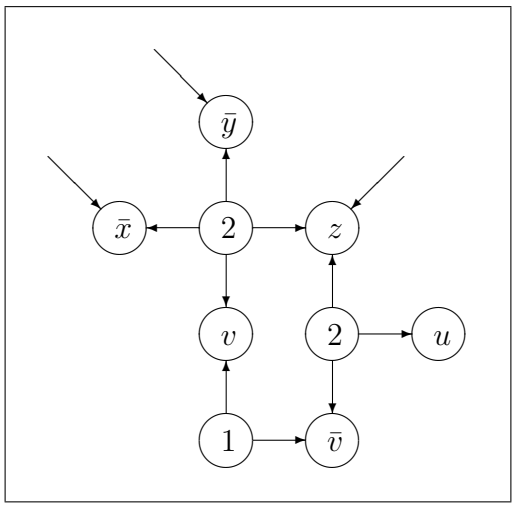

Abbildung 3.4: ODER-Verknüpfung für MWG

Lemma 3.101 (Informationsreplikation) *Die Information, ob ein gegebenes unbestimmtes Feld in einer konsistenten Minenverteilung* eine Mine enthält*, kann beliebig oft unter Einhaltung der Schranken*

$$\forall x \in \mathcal{F} : d^+(x) \leq 2 \wedge d^-(x) \leq 3$$

ausgewertet werden.

Beweis 3.101 Wir kehren einfach wieder zum Konzept der Informationsleitung zurück und sehen, dass hierbei *jeder* Knoten genau zwei inzidente Kanten hat:

$$\cdots \to x \leftarrow 1 \to \bar{x} \leftarrow 1 \to x \leftarrow \cdots$$

Jeder Knoten $y \in \{x, \bar{x}\}$ darf also noch Nachbar eines weiteren Feldes sein, ohne die Schranke $d^-(y) \leq 3$ zu überschreiten. □

Lemma 3.102 (ODER-Verknüpfung) *Die Disjunktion zweier Felder kann auf einem MW-Graphen unter Einhaltung der Schranken*

$$\forall x \in \mathcal{F} : d^+(x) \leq 4 \wedge d^-(x) \leq 3$$

umgesetzt werden.

Beweis 3.102 Der in Abbildung 3.4 auf Seite 65 angegebene Teilgraph zeigt die MWP-Konfiguration aus Lemma 2.23 nach Anwendung der erarbeiteten Vereinfachungen. Dieser leistet das gewünschte und respektiert die Schranken. □

Da die Konzepte aus den Lemmata 3.101 und 3.102 auch einfache und lokal sternförmige Graphen garantieren, folgt sofort:

Korollar 3.103 ($\mathbf{SAT \leq MWG_{einfach, lokal\ sternförmig, d^+ \leq 4, d^- \leq 3}}$) *Die Reduktion von SAT auf MWG benötigt nur* lokal sternförmige *Graphen ohne* Schlingen*,* parallelen *und* gegenläufigen Kanten*, deren* Weggrade *mit* $d^+ \leq 4$ *und deren* Hingrade *mit* $d^- \leq 3$ *beschränkt sind.*

Eine weitere Eigenschaft kommt durch das Illustrieren der bisher verwendeten Konzepte in Frage: Ist auch die Einschränkung auf *ebene*[10] Graphen noch ohne Verlust der NP-Vollständigkeit machbar?

Die Frage kann mit einem schon von Kaye in [8] verwendeten Konzept für planare Schaltkreise bejaht werden. Hierbei macht man sich die absolute Symmetrie von logischer Äquivalenz ($\leftrightarrow$) oder Antivalenz ($+$) zu nutze. Diese sind durch die folgenden Verknüpfungstafeln gegeben:

x	y	$x \leftrightarrow y$
0	0	1
0	1	0
1	0	0
1	1	1

x	y	$x + y$
0	0	0
0	1	1
1	0	1
1	1	0

Wir sehen sofort, dass genau alle Zeilen mit *einem oder drei* bzw. *keinem oder zwei* Werten 1 verwendet werden. Daher gilt stets, dass die entsprechende Verknüpfung zweier beliebiger Spalten die dritte ergibt:

$$(x \leftrightarrow y) \leftrightarrow x = y \qquad (x + y) + x = y$$
$$(x \leftrightarrow y) \leftrightarrow y = x \qquad (x + y) + y = x$$

Zur Umsetzung erinnern wir uns an die Beobachtung, dass die konsistenten Belegungen der ODER-Verknüpfung (siehe Beweis 2.23) die logische Äquivalenz im Hilfsfeld u ergeben hat. Außerdem gilt natürlich $x \leftrightarrow y = \bar{x} \leftrightarrow \bar{y}$. Wir wollen aber das folgende Konzept nicht vergessen, weil wir es in Abschnitt 3.3.4 noch verwenden werden:

Bemerkung 3.104 (variable Explosivität) *Die absolute Symmetrie der logischen Äquivalenz liefert eine einfache Umsetzung auf MW-Graphen, wenn uns ein Feld mit* variabler Explosivität *„1 oder 3“ zur Verfügung steht. Die Lösung ist in Abbildung 3.5 auf Seite 67 für allgemeine und eindeutige Graphen dargestellt.*

Da die angeführten Teilgraphen aber einen Knoten mit Weggrad $d^+ = 5$ haben, greifen wir doch auf die adaptierte ODER-Lösung zurück, die aber bei genauer Betrachtung auch nur eine „variable Explosivität“ in der *Summe* der beiden Felder mit dem Wert 2 zur Verfügung stellt, und erhalten:

Satz 3.105 ($\mathbf{SAT \leq MWG_{einfach,lokal\ sternförmig,eben,d^+\leq 4,d^-\leq 3,\sigma\leq 2}}$) *Die Reduktion von SAT auf MWG benötigt nur* ebene, lokal sternförmige *Graphen ohne* Schlingen, parallelen *und* gegenläufigen Kanten*, deren* Weggrade *mit $d^+ \leq 4$ und deren* Hingrade *mit $d^- \leq 3$ beschränkt sind. Außerdem ist die Umgebungsgefahr aller Felder x in einer konsistenten Belegung mit $\sigma(x) \leq 2$ beschränkt.*

Beweis 3.105 Da die aktuellen ODER-Graphen und Informationsleitungen eben – oder zumindest planar – sind, fehlt nur noch die Angabe einer ebenen Kreuzung um die Anordnung der Knoten in der Ebene wie bei MWP (siehe Abbildung 2.2 auf Seite 33)

[10] Die Eigenschaft *planar* kann durch die Äquivalenz zu einem ebenen Graphen für MWG sofort durch die stärkere Eigenschaft *eben* ersetzt werden.

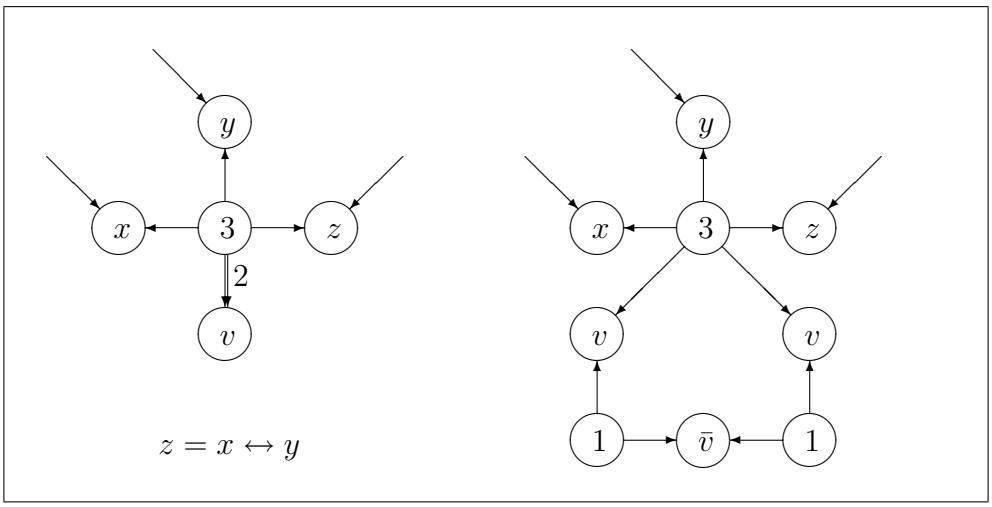

Abbildung 3.5: Logische Äquivalenz durch variable Explosivität

zu ermöglichen. Das Konzept ist vollständig erklärt. Wir geben die ebene Kreuzung der Vollständigkeit halber in Abbildung 3.6 auf Seite 68 an.

Die Beschränkung der Umgebungsgefahr ist leicht einzusehen, weil in allen bisher vorgestellten Konzepten vorgegebene Felder die Forderung $\sigma \leq 2$ erfüllen und unbestimmte Felder keine Nachbarn haben. Daher haben konsistente Belegungen dieser Graphen auf den zunächst unbestimmten Feldern einen der Werte 0 oder ∞. □

Korollar 3.106 ($\mathbf{MWG_{einfach,lokal\,sternf.,eben,d^+\leq 4,d^-\leq 3,\sigma\leq 2}} \in$ NP-vollständig) *Das Problem MWG ist bereits schwierig für MW-Graphen mit den angegebenen Eigenschaften.*

Aus Beweis 3.105 sehen wir, dass die aus SAT konstruierten Graphen sogar die Eigenschaft haben, dass *alle Felder* mit 1, 2 oder ? beschriftet sind. *Zeugen* für die Konsistenz können ausschließlich unter den *zweistelligen* Belegungen

$$\varphi : \mathcal{F} \setminus \mathcal{D} \to \{0, \infty\}$$

gefunden werden.

In gewisser Weise sind diese MWG-Instanzen also als Erfüllbarkeitsproblem für aussagenlogische Formeln interpretierbar. Diese Formeln sind eine Konjunktion 2-, 3- und 4-stelliger Operationen[11], welche die Forderung „genau 1“ bzw. „genau 2“ wiedergeben. In einer solchen Formel kommt außerdem keine Variable öfter als drei Mal vor.

Einfache Beispiele

Nach einigen Vorstößen Richtung unterer Grenze der NP-Vollständigkeit suchen wir nun MW-Graphen, die einfach zu lösen sind. *Einfach* bedeutet hierbei deterministische polynomielle Laufzeit oder mindestens eine Typ-2-Grammatik. Weil der Zusammenhang

[11] Die Stelligkeit wird genau durch den Weggrad d^+ angegeben.

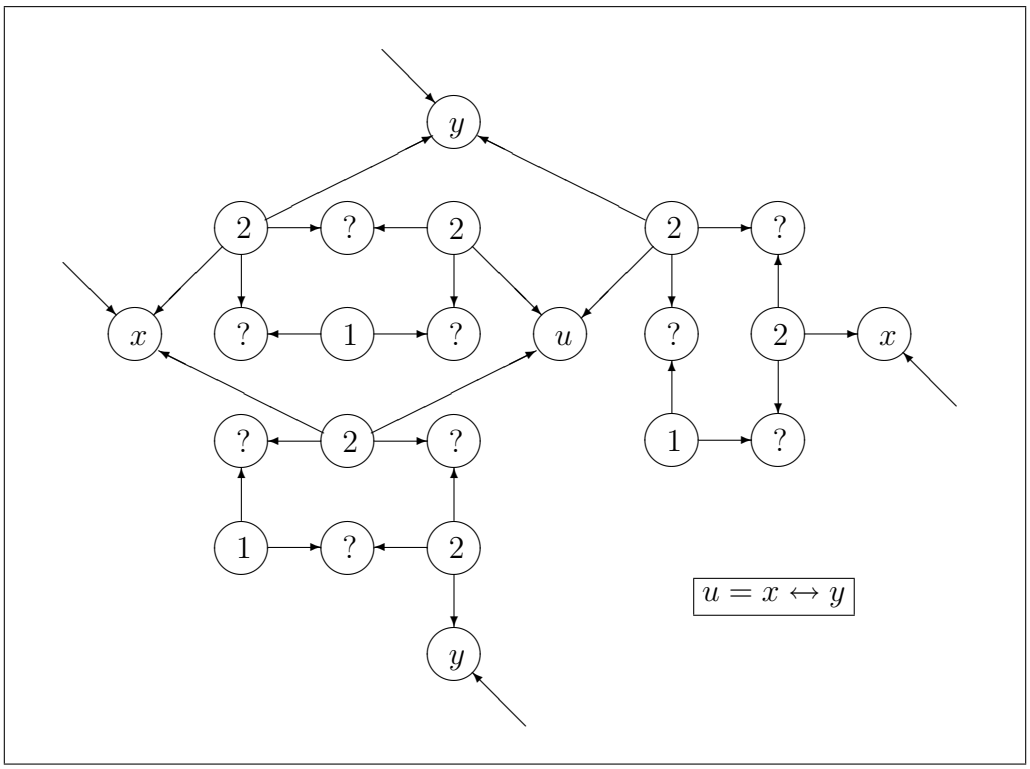

Abbildung 3.6: Ebene Kreuzung für MWG

zwischen der Laufzeitklasse P und der Chomsky-Hierachie nicht klar ist, werden wir immer beide Zugehörigkeiten argumentieren.

Wir werden uns in diesem Abschnitt zwei wesentlichen Einschränkungen widmen, die uns bei den bisherigen Betrachtungen aufgefallen sind:

- Der ODER-Graph enthält einen Knoten mit Grad $d^+ = 4$.
- Die Komplexität in SAT entsteht durch mehrfaches Auftreten gleicher Variablen.

Letztere führt nicht-planaren MW-Graphen oder zu planaren, deren Kanten die reelle Ebene $\mathbf{R}^2$ in mehrere disjunkte Mengen teilen. Wir werden sehen, dass *Bäume* (und Wälder) als MW-Graphen und solche mit sehr *geringen Knotengraden* ($d \leq 2$) einfache Grammatiken oder kleine Laufzeiten haben – vorausgesetzt man wendet einen auf die Klasse zugeschnittenen Algorithmus an.

Zunächst betrachten wir endliche ungerichtete Graphen, deren Knoten jeweils nur mit maximal zwei (ungerichteten) Kanten inzident sind:

Lemma 3.107 (eindimensionale Graphen) *Zusammenhangskomponenten endlicher* ungerichteter *Graphen mit beschränkten Knotengraden $d \leq 2$ sind* kreuzungsfreie Kantenfolgen*. Wir nennen sie daher* eindimensional*.*

Beweis 3.107 Eine solche Komponente ist zusammenhängend und ungerichtet, es gibt also eine Kantenfolge von jedem Knoten zu jedem anderen. Wir unterscheiden nach Knotengraden: Wenn es einen Knoten x mit $d(x) = 0$, so ist dies der einzige, da sonst $E(x, V \setminus x) = \emptyset$ leer wäre.

Wenn es einen Knoten x mit $d(x) = 1$ gibt, erreichen wir von ihm aus genau *einen* Knoten y, der entweder Grad $d(y) = 1$ hat, oder wieder zu einem eindeutigen *weiteren* Knoten benachbart ist. Diese Folge muss irgendwann in einen Knoten mit Grad $d(y) = 1$ enden, weil V endlich ist und alle Kanten der bereits besuchten Knoten schon „verwendet“ wurden.

Wenn es nur Knoten x mit Grad $d(x) = 2$ gibt, so durchlaufen wir alle Knoten nach obiger Vorgehensweise und Argumentation von einem Startknoten aus mit einer seiner beiden Kanten beginnend genau einmal bevor wir zum *Startknoten* – und keinem anderen – zurückkehren. In jedem Fall erhalten wir also eine kreuzungsfreie Kantenfolge, die auch ein Zyklus – also geschlossen – sein kann. □

Im Wesentlichen bedeutet dies, dass wir es hier mit einer kleinen Verschärfung von MW1 zu tun haben: Wir betrachten einerseits auch geschlossene MW-Zeichenfolgen, deren erstes und letztes Zeichen benachbart sind, und andererseits mehrere solcher Zusammenhangskomponenten gleichzeitig.

Lemma 3.108 (RA für spezielle $\mathbf{MWG_{d\leq 2}}$) *Der RA*

$$00*(\circ + \bullet) + \infty\,((\epsilon + 2 + 10* 1)\,\infty)*(\epsilon + 2 + 10* 1)\circ +$$

$$+ (\epsilon + 0* 1)\,\infty\,((\epsilon + 2 + 10* 1)\,\infty)*(\epsilon + 10*)\bullet$$

erzeugt bis auf Isomorphie *der Graphen eine* Interpretation *aller konsistenter, vollständig vorgegebener, zusammenhängender, eindimensionaler MW-Graphen.*

Beweis 3.108 Zunächst benötigen wir eine bijektive, polynomiell berechenbare Interpretation für eindimensionale MW-Graphen. Wegen Lemma 3.107 müssen wir nur die Werte der Knoten entlang einer Kantenfolge notieren. Das Alphabet ist also $\Sigma = \{0, 1, 2, \infty\}$. Wir gehen davon aus, dass der Graph in einer 2TM vorliegt, weil die jeweils mit 2 beschränkte Zahl der Kanten und die Umgebungsgefahr in je einem Zeichen codiert werden kann. Der Algorithmus mit Laufzeitschranken ist in Tabelle 3.4 auf Seite 70 angegeben und kann in linearer Zeit ausgeführt werden.

Es fehlt die Korrektheit des RA: Die Ausdrücke, die auf $\bullet$ enden, sind genau jene für MW-Zeichenfolgen und somit richtig. Ein Zyklus endet auf $\circ$ und beginnt in unserer Interpretation immer mit einer Mine, falls er eine enthält. Die nächsten beiden Ausdrücke geben nun genau die weiteren Minen jeweils mit den dazwischenliegenden Feldern sowie die möglichen Felder zwischen letzter und erster Mine wieder. Diese Ausdrücke entsprechen ebenfalls den Teilausdrücken der MW-Zeichenfolgen. Das Wiederherstellen des ursprünglichen Graphen bis auf Isomorphie ist einfach. □

Satz 3.109 (EA für $\mathbf{MWG_{ungerichtet,d\leq 2}}$) *Es existieren eine Interpretation bis auf Isomorphie und ein EA, die $MWG_{ungerichtet,d\leq 2}$ lösen.*

Beweis 3.109 Klarerweise können die einzelnen Zusammenhangskomponenten des Graphen durch Aneinanderfügen mehrerer Interpretationen aus Lemma 3.108 angegeben werden. Die Reihenfolge ist hierbei wegen der Isomorphietoleranz irrelevant. Dieses Verketten eines oder mehrerer kann auch im RA umgesetzt werden.

Befehl mit Laufzeit $O(f(\alpha_0))$	$f(\alpha_0)$
Teste die Syntax (MW-Graph, $d \leq 2$, $\kappa : V \to \{0, 1, 2, \infty\}$)	α_0^2
Suche ein Feld x mit *Knotengrad* 1 entlang der $\mathcal{F}_x$-Achse. Markiere es als Startfeld – wenn kein solches gefunden, dann eines mit *Mine*, sonst ein beliebiges.	α_0^2
Notiere seinen Wert in der Ausgabe	α_0
Gehe in $\mathcal{F}_x$-Richtung zur ersten Kante (Adjazenzeintrag ungleich 0)	α_0
Falls es keine gibt handelt es sich nicht um einen Zyklus: notiere ein $\bullet$ am Ende der Ausgabe.	α_0
Entferne in $\mathcal{F}_y$-Richtung alle (maximal eine weitere) Kanten, weil das Feld vom Nachfolger aus nicht mehr besucht werden darf.	α_0
Suche das Feld auf der $\mathcal{F}_x$-Achse durch eine Diagonalquerung.	α_0
Falls es das Startfeld ist, notiere ein $\circ$ am Ende der Ausgabe.	α_0
Sonst fahre mit „Notiere seinen Wert…“ fort.	
Die Gesamtlaufzeit liegt in $O(n) = O(\alpha_0^2)$, weil jeder der mehrfach ausgeführten Befehle genau für alle Knoten einmal vorkommt.	α_0^2

Tabelle 3.4: Algorithmus zu Interpretation eindimensionaler MW-Graphen

Die Eigenschaft „vollständig vorgegeben“ aus dem selben Lemma können wir analog zu Vorgehensweise bei MW1 durch optionales Ersetzen jedes Zeichens mit einem Fragezeichen (?) umgehen (siehe Beweis 3.35). Natürlich gibt es für diesen endgültigen RA dann einen EA. Allerdings schlägt die Anforderung an die Interpretation möglichst mit Minenen zu beginnen im Algorithmus fehl, wenn eine Zusammenhangskomponente keine Minen (∞) aber Fragezeichen (?) enthält:

Da die Interpretation deterministisch berechenbar sein muss, können wir nur implementieren, dass der Algorithmus die Zeichenfolge mit dem *ersten* gefundenen ? beginnt, falls er keine Mine (∞) findet. Wir können aber verlangen, dass diese Wahl nicht auf ein anderes als das Erste gegebenenfalls aufeinanderfolgender Fragezeichen trifft, ohne die lineare Laufzeit zu gefährden, indem wir rückwärts das erste andere Zeichen suchen und dessen Nachfolger heranziehen. Unser RA deckt nun bereits die Fälle ab, in denen ein Zeuge der Komponente keine Mine enthält und in denen dieses erste ? als Mine interpretiert wird. Es fehlen folgende Fälle, die wir gleich reparieren:

- Das Fragezeichen steht für den Wert 2 in $\infty 2 \infty$: Da im Ausdruck keine Minen (∞) vorkommen stehen an dieser Stelle mindesten drei aufeinanderfolgende Fragezeichen (???) und wir haben *nicht* mit dem mittleren begonnen.

- Das Fragezeichen steht für einen der Werte aus $10*1$: Wie im vorigen Fall argumentiert kann nicht der erste Wert 1 betroffen sein. Wir ergänzen den RA mit:

$$\ldots +? (\epsilon + (0+?)* (1+?))? ((\epsilon + 2 + (1+?) (0+?)* (1+?))?)* (1+?) (0+?)* \circ + \ldots$$

Falls im ersten Klammerausdruck das Leerwort ϵ verwendet wird, stehen die beiden Fragezeichen ?? für 1∞, ansonsten wird das Erste als 0 interpretiert. Die Konsistenz ist jetzt mit einem EA auf der Interpretation überprüfbar. □

Obwohl ein entsprechender Algorithmus problemlos auch direkt angegeben werden kann, liefert dieser Satz mit den Laufzeiten aus Tabelle 3.4 auf Seite 70 sofort:

Korollar 3.110 ($\mathbf{MWG_{ungerichtet,d\leq 2}} \in P$) *MWG ist einfach für ungerichtete Graphen mit beschränktem Grad $d \leq 2$. Die Laufzeit ist* linear.

Mit ein paar einfachen Überlegungen sehen wir, dass es auch nicht schwieriger sein kann gerichtete und sogar gemischte eindimensionale Graphen zu betrachten:

Satz 3.111 ($\mathbf{MWG_{d\leq 2}} \in P$) *MWG ist einfach für* gemischte *Graphen mit beschränktem Grad $d \leq 2$.*

Beweis 3.111 Aus den bereits angestellten Überlegungen folgt sofort, dass die Syntax auch für gemischte eindimensionale Graphen linear ist. In der Interpretation geben wir nun jeweils gemeinsam mit dem Wert des Knotens die Richtung der weiterführenden Kante ($\{0, 1, 2, \infty\} \times \{\leftarrow, \rightarrow, \leftrightarrow\}$) an. Weiters denken wir zurück an den EA für MW-Zeichenfolgen, dessen Zustände die erlaubten Folgewerte angegeben haben (Beweis 3.30):

Analog können wir uns hier die entsprechenden Einschränkungen überlegen. Exemplarisch geben wir an, dass auf $(\infty, \leftarrow)$ eines der Zeichen[12]

$$\left\{\left(\overline{1}, *\right), (\infty, *), (2, \rightarrow), (2, \leftrightarrow)\right\} \not\ni (2, \leftarrow)$$

und auf $(\infty, \rightarrow)$ eines der Zeichen

$$\{(0, *), (\infty, *), (\underline{1}, \rightarrow), (\underline{1}, \leftrightarrow)\} \not\supset \{(2, *)\}$$

folgen darf. Der EA wird zwar deutlich komplizierter – auch deshalb, weil er sich das erste Zeichen durch disjunkte Teilautomaten mit unterschiedlichen Endzuständen merken muss – aber es bleibt für gemischte MW-Graphen bei der *linearen Laufzeit.* □

Bei Bäumen gehen wir wie bei den eindimensionalen Graphen vor: Zunächst überlegen wir uns eine Darstellung für ungerichtete Bäume, dann betrachten wir die Zeugen auf diesen. Zum Schluss verallgemeinern wir die Ergebnisse auf beliebige Konfigurationen und gemischte Graphen.

Wir können bei Bäumen nicht mehr mit einer eindeutigen Kantenfolge ohne Verzweigungen rechnen und die Nachbarn eines Knotens müssen einerseits unmittelbar hintereinander abgefragt werden um die Umgebungsgefahr zu verifizieren, andererseits wird ihr Wert jeweils gefragt, wenn wir deren Nachbarn betrachten.

Da wir die Umgebungsgefahr nicht mehr beschränken können, ist keine Darstellung in einem Zeichen mehr möglich und es scheint klar, dass unser Computer zum Lösen des Problems nicht ohne Speicher auskommt. Aus diesem Grund streben wir auch für die reine Darstellung eines Baumes keinen RA an, sondern geben uns mit einer einfach verständlichen Typ-2-Grammatik zufrieden.

Lemma 3.112 (Bäume sind zyklenfrei) *Ein ungerichteter Baum hat* keinen Zyklus *und es gibt von jedem zu jedem Knoten* genau eine kreuzungsfreie *Kantenfolge.*

[12] Das Sternchen ($*$) steht hierbei für alle Möglichkeiten.

Beweis 3.112 Bäume sind minimal zusammenhängende Graphen. Daraus folgt im ungerichteten Fall sofort, dass eine Kantenfolge von jedem zu jedem Knoten existiert. Angenommen der Graph hätte einen Zyklus, so könnte *eine beliebige* Kante aus diesem entfernt werden ohne den Zusammenhang zu stören.

Angenommen es gäbe für ein Knotenpaar zwei (oder mehr) Kantenfolgen zueinander, so gibt es einen Zyklus auf dem die beiden Knoten liegen, an denen sich diese Kantenfolgen trennen und wieder kreuzen. Damit sind alle Aussagen bewiesen. □

Lemma 3.113 (Typ-2-Grammatik für MW-Bäume) *Die* kontextfreie *Grammatik*

$$\begin{aligned}
\mathcal{G} := \quad & \langle \quad \Sigma := \{0,1,\infty,+,-\}, \Gamma := \{T,V,E,N\}, R, I := T \rangle \qquad \textit{mit} \\
R := \quad & \{ \quad T \to V, \quad T \to VE, \\
& \qquad E \to +T-, \quad E \to +T- E, \\
& \qquad V \to 0, \quad V \to \infty, \quad V \to N, \\
& \qquad N \to 1, \quad N \to 1N\}
\end{aligned}$$

erzeugt bis auf Isomorphie *der Graphen eine* Interpretation *aller vollständig vorgegebener, ungerichteter MW-Bäume.*

Beweis 3.113 Zunächst halten wir als Folgerung von Lemma 3.112 fest, dass Bäume rekursiv aufgebaut sind: Wenn wir einen beliebigen Knoten x wählen, so hat dieser zu jedem anderen Knoten genau eine kreuzungsfreie Kantenfolge. Dasselbe gilt auch für seine Nachbarknoten. Wenn wir also die $d(x)$ Kanten von x enfernen, erhalten wir $d(x)+1$ disjunkte Bäume, die zusammen den ursprünglichen Graphen ergeben. Diese „Tiefensuche“ endet in einzelnen Knoten[13], weil der Baum keinen Zyklus hat und endlich ist.

Wegen der Isomorphietoleranz ist es auch egal, welchen Knoten wir als Startknoten wählen und in welcher Reihenfolge wir seine Kanten abarbeiten. Wir interpretieren die Nonterminalsymbole Γ wie folgt: $T \ldots$ *Baum*, $V \ldots$ *Knoten*, $E \ldots$ *Kanten* und $N \ldots$ *natürliche Zahl*. Die Regeln besagen nun, dass ein Baum ein einzelner Knoten oder ein Knoten mit Kanten ist. Kanten eines Knotens sind eine Kante, an der ein Baum hängt, und optional weitere Kanten. Ein Knoten ist mit 0, ∞ oder einer natürlichen Zahl beschriftet. Letztere besteht aus einer positiven Anzahl Einser 1.

Die Grammatik beschreibt also genau die MW-Bäume. Diese Interpretation ist bijektiv, weil wir nach jedem Knoten angeben, ob es eine neue Kante gibt, die wir durchlaufen $(+)$, oder ob wir zum Vorgängerknoten zurückkehren $(-)$. Auf unserer 2TM (Bäume sind offensichtlich eindeutig) gehen wir entlang einer Kantenfolge – analog zum eindimensionalen Fall – durch iteratives Suchen einer Kante und Auffinden des Knotens auf der $\mathcal{F}_x$-Achse mit einer Diagonalquerung.

Wenn wir wieder alle anderen *ankommenden* Kanten entfernen, können wir diese Kantenfolge auch wieder zurückgehen um eine weitere Kante eines Knotens zu finden. Insgesamt ist die Laufzeit wie im eindimensionalen Fall *linear*. Das Wiederherstellen eines isomorphen Graphen ist einfach. □

[13] Diese letzten Knoten nennt man *Blätter*, den Startknoten *Wurzel* und einen Baum mit einem derart ausgezeichneten Knoten *Wurzelbaum*.

$\pi_1(\kappa(y))$	$\mu(x) = 0$	$\mu(x) = 1$
0	notiere „+“ als Knotentrennzeichen im Speicher	gefundene Inkonsistenz
1	notiere „+1“ im Speicher	notiere „+“ im Speicher: dadurch haben wir 1 vom neuen Knoten abgezogen
∞	ersetze den letzten Wert 1 aus dem Speicher mit „$+\infty$“: inkonsistent, falls er nicht 1 ist	notiere „$+\infty$“ im Speicher

Tabelle 3.5: Informationsaustausch an einer Kante eines MW-Baumes

Wir haben jetzt *alle* vollständig angegebenen MW-Bäume dargestellt und müssen noch die konsistenten Zeugen herausfiltern. Hierzu werden wir aber nur die *Existenz* einer kontextfreien Grammatik zeigen, indem wir die Zeugen auf einem DKA testen:

Satz 3.114 (DKA für Zeugen auf ungerichteten MW-Bäumen) *Zeugen auf ungerichteten MW-Bäumen können mit einem deterministischen KA überprüft werden.*

Beweis 3.114 Wir erinnern uns, dass eine einfache Fallunterscheidung in den Zuständen codiert werden kann und daher keinen Platz am Kellerspeicher benötigt. Über eine Kante geht aber auch nicht mehr Information als der Minenindikator μ, dieser aber für beide inzidenten Knoten.

Unser KA notiert zunächst den ersten Knoten x (0, $k \in \mathbf{N}$, ∞) im Kellerspeicher und merkt sich, ob er eine Mine enthält. Beim Einlesen einer Kante + liest er zusätzlich noch das *erste* Zeichen des neuen Knoten y und merkt es sich. Die weitere Aktion hängt von den gemerkten Informationen ab und ist in Tabelle 3.5 auf Seite 73 angegeben.

Jetzt ist der Informationsaustausch zwischen den beiden Knoten abgeschlossen. Außerdem ist die Information, ob der erste Knoten eine Mine enthält nach wie vor im Kellerspeicher, sodass wir sie beim Zurückkehren wieder auslesen können. Wir schreiben gegebenenfalls die *weiteren* Einser 1 des neuen Knoten in den Speicher und merken uns statt der alten Information, ob der *neue* Knoten eine Mine enthält. Die rekursive Vorgehensweise in die Tiefe ist damit erklärt.

Wenn der KA eine Kante ($-$) liest, die zurück führt, müssen alle Minen des betrachteten Knotens gefunden sein. Daher steht am Ende des Speichers „$+\infty$“ oder „+“ (der Wert 0 ist durch fehlende Zeichen 1 oder ∞ zwischen zwei Knotentrennzeichen gekennzeichnet), wenn die Belegung konsistent ist. Wir können diese Zeichen entfernen und erneut abfragen, ob der alte Knoten eine Mine war um diese Information für eine weitere Kante zu verwenden. □

Wenn wir nun als Knotenbeschriftung auch das Fragezeichen (?) zulassen, müssen wir einfach einen Wert raten, den wir im Kellerspeicher ablegen. Von diesem Wert müssen wir nur wissen, *dass* er beschränkt ist: Es müssen nicht *alle* Berechnungsmöglichkeiten halten. Es folgt sofort:

Korollar 3.115 ($\mathbf{MWG_{ungerichtet,Baum} \in NKA}$) *MW auf* ungerichteten Bäumen *kann mit einem nicht-deterministischen KA gelöst werden.*

Wir wissen natürlich sogar, dass die Umgebungsgefahr eines Knotens x eines eindeutigen Graphen mit $d(x) \leq \alpha_0$ beschränkt ist. Der KA kann diese Schranke zwar nicht umsetzen, aber wir halten ihn nach der entsprechenden Laufzeit an, falls kein Zeuge gefunden wurde.

Satz 3.116 ($\mathbf{MWG_{Baum}} \in \mathbf{NKA}$) *MW auf* gemischten Bäumen *kann mit einem nicht-deterministischen KA gelöst werden.*

Beweis 3.116 Wir gehen analog zum Fall der eindimensionalen Graphen vor: In der Interpretation sei(en) die Richtung(en) der Kante jeweils im ersten Zeichen eines neuen Knoten codiert. Tabelle 3.5 auf Seite 73 wird dadurch ein wenig umfangreicher, aber nicht komplizierter. Die Vorgehensweise ist klar. □

Korollar 3.117 ($\mathbf{MWG_{Baum}} \in \mathcal{L}_2$) *Es gibt eine Interpretation beliebiger konsistenter MW-Bäume und eine* kontextfreie Grammatik*, die diese beschreibt.*

3.3.4 Weitere Reduktionen

Als vorläufigen Abschluss der Betrachtungen von MWG werden wir für einige NP-vollständige Probleme zeigen, dass sie einfach und anschaulich auf MWG reduzierbar sind. Nach der Definition des jeweiligen Problems folgt immer die informelle Beschreibung einer möglichen Darstellung am Computer, woraus sofort die Zugehörigkeit zu NP mit Hilfe einer polynomiellen Vorhersagerelation folgt.

Anschließend beweisen oder argumentieren wir jeweils die Vollständigkeit in NP durch die Reduktion eines bekannten NP-vollständigen Problems. Die Reduktion auf MWG ist meistens leicht verständlich und am besten anhand einer Skizze erklärt.

Problem 3.118 (kSAT) *Jede Instanz von (CNF)SAT, deren Klauseln* nicht mehr *als $k \in \mathbf{N}$* Literale *enthalten, ist eine Instanz von kSAT. Die Fragestellung ist gleich.*

Offensichtlich ist jede Instanz von kSAT eine Instanz von nSAT für alle $n \geq k$ und von SAT. Damit ist einerseits die Darstellung geklärt und andererseits stellt sich die Frage nach dem kleinsten k, sodass kSAT schwierig – also NP-vollständig – ist. Ohne Beweis wollen wir festhalten, dass 2SAT in P liegt.

Satz 3.119 ($\mathbf{SAT} \leq \mathbf{3SAT}$) *Jede Instanz von SAT kann in eine Instanz von 3SAT übergeführt werden, sodass immer beide oder keine von beiden erfüllbar sind.*

Beweis 3.119 Es genügt zu zeigen, dass eine Klausel $C = y_1 \vee \ldots \vee y_n$ mit einer beliebigen Anzahl Literale in eine Formel F in 3-CNF mit polynomiellen Schranken umgewandelt werden kann. Wir definieren mit nur n *neuen* Variablen $z_1, \ldots, z_n$:

$$F := (y_1 \vee \bar{z}_1) \wedge (z_1 \vee y_2 \vee \bar{z}_2) \wedge \ldots \wedge (z_{n-1} \vee y_n \vee \bar{z}_n) \wedge z_n.$$

Angenommen es existiert in der ursprünglichen Belegung ein $1 \leq i \leq n$ mit $\varphi(y_i) = 1$, so können wir diese erfüllend fortsetzen, indem wir $\varphi(z_j) := 0$ festlegen genau dann, wenn $j < i$ gilt.

Angenommen es existiert eine erfüllende Belegung auf $\{y_1, \ldots, y_n, z_1, \ldots, z_n\}$, so kann $\varphi(C) = 0$ nicht gelten: Wegen $\varphi(y_1) = 0$ folgte $\varphi(z_1) = 0$. Ebenso könnten wir $\varphi(z_2) = \ldots = \varphi(z_n) = 0$ folgern mit Widerspruch zu $\varphi(F) = 1$. □

3SAT kann auf MW-Graphen wie SAT dargestellt werden. Dabei ergibt sich für das Konzept aus Beweis 3.100 bereits eine Schranke für die Weggrade und Knotenbeschriftungen. Diese führen zu kleinen Graphen, können aber mit den Schranken $d^+ \leq 4$ und $\sigma \leq 2$ aus Satz 3.105 nicht mithalten. Auch für die folgenden Beweise in denen SAT auf andere Probleme reduziert wird, ist es nicht notwendig auf eine Beschränkung der Klausellänge zu bestehen.

Wir haben sogar bereits argumentiert, das auch beliebige SAT-Instanzen zugelassen werden können, weil der ODER-Graph Felder mit der Konjunktion und Äquivalenz seiner „Eingaben“ enthält. Die Anordnung – wie sie bei MWP zu beachten war – ist bei allgemeinen Graphen kein Thema.

Problem 3.120 (CLIQUE) *Gegeben ein endlicher, einfacher, ungerichteter Graph $G = \langle V, A\rangle$ ($V = \alpha_0$, $A : V^2 \to 2$, $A^T = A$, $\forall i \in \alpha_0 : A(i,i) = 0$) und eine natürliche Zahl $k \in \mathbf{N}$. Gibt es eine eine k-elementige Teilmenge $U \subseteq V$ der Knoten mit $|U| = k$, sodass die Knoten aus U paarweise benachbart sind?*

$$\forall x, y \in U : x \neq y \to A(x,y) = 1$$

Eine solche Teilmenge nennt man CLIQUE. Die Darstellung ist auf einer 2TM leicht möglich. Ein Zeuge sei die Markierung der Knoten aus U entlang einer Achse: Wir markieren jeweils die ganze Spalte und stellen zeilenweise für die gleichen Knoten (Diagonalquerung!) fest, ob alle Kreuzungen mit markierten Spalten den Wert 1 enthalten.

Satz 3.121 (SAT $\leq$ CLIQUE) *Mit CLIQUE kann SAT gelöst werden.*

Beweis 3.121 Gegeben eine Formel $F = C_1 \wedge \ldots \wedge C_m$, $m \in \mathbf{N}$ mit Klauseln $C_i = y_{i_1} \vee \ldots \vee y_{i_{n_i}} = x_{i_1}^{g_{i_1}} \vee \ldots \vee x_{i_{n_i}}^{g_{i_n}}$, $1 \leq i \leq m$, $n_i \in \mathbf{N}^+$, $g_{i_j} \in 2$ mit $g_{i_j} = 0$, wenn $y_{i_j} = \bar{x}_{i_j}$, und $g_{i_j} = 1$, wenn $y_{i_j} = x_{i_j}$.

Wir definieren den folgenden einfachen ungerichteten Graphen, der einen Knoten für jedes Vorkommen einer Variable $c := \sum_{i=1}^{m} n_i$ enthält. Kanten definieren wir für Literale aus verschiedenen Klauseln, die nicht die Negation voneinander sind:

$$V := \left\{ v_{i_j} : 1 \leq i \leq m,\ 1 \leq j \leq n_i \right\}$$

$$\begin{array}{rcl} A: \quad V^2 & \to & 2 \\ \left(v_{i_j}, v_{k_l}\right) & \mapsto & \begin{cases} 1 & falls\ i \neq k \wedge \left(x_{i_j} \neq x_{k_l} \vee g_{i_j} = g_{k_l}\right) \\ 0 & sonst \end{cases} \end{array} .$$

Dieser Graph ist offensichtlich symmetrisch. Er hat eine CLIQUE der Größe m genau denn, wenn F erfüllbar ist: In einer CLIQUE sind weder gleichzeitig zwei Literale aus einer Klausel, noch gleichzeitig ein Literal und sein Komplement, wegen der Größe m aber immer ein Literal aus jeder Klausel. Wir finden also leicht für jede erfüllende Belegung eine CLIQUE und umgekehrt. □

Problem 3.122 (VC) *Gegeben ein endlicher, einfacher, ungerichteter Graph $G = \langle V, A\rangle$ ($V = \alpha_0$, $A : V^2 \to 2$, $A^T = A$, $\forall i \in \alpha_0 : A(i,i) = 0$) und eine natürliche Zahl $k \in \mathbf{N}$. Gibt es eine eine k-elementige Teilmenge $U \subseteq V$ der Knoten mit $|U| = k$, sodass mit jeder Kante zumindest einer ihrer inzidenten Knoten in U liegt?*

$$\forall x, y \in V : A(x,y) = 1 \to (x \in U \vee y \in U)$$

Eine solche Teilmenge nennt man „Vertex Cover“. Die Darstellung ist auf einer 2TM leicht möglich. Ein Zeuge sei die Markierung der Knoten aus U entlang einer Achse: Wir markieren jeweils die ganze Spalte und für die gleichen Knoten (Diagonalquerung!) auch die Zeilen. Jeder Wert 1 muss jetzt markiert sein.

Satz 3.123 (CLIQUE = VC) *CLIQUE und VC können bijektiv ineinander übergeführt werden.*

Beweis 3.123 Für den Graphen $G = \langle V, A\rangle$ einer CLIQUE-Instanz mit einem Zeugen $U \subseteq V$ mit $|U| = k$ gilt: $V \setminus U$ ist ein VC der Größe $\alpha_0 - k$ für den Graphen $G' = \langle V, 1 - A - E_{\alpha_0}\rangle$. E_{α_0} bezeichne die $\alpha_0 \times \alpha_0$-Einheitsmatrix, G' nennt man *Komplement* von G. Die Komplementbildung ist für Graphen selbstinvers und die Aussage gilt auch in die andere Richtung.

Da im Komplement keine Kante zwischen zwei beliebigen Knoten einer CLIQUE U existiert, ist $V \setminus U$ wirklich ein VC. Umgekehrt gilt für ein VC U, dass keine Kanten zwischen beliebigen Knoten aus $V \setminus U$ verlaufen dürfen. Daher ist diese Menge eine CLIQUE im Komplement. □

Satz 3.124 (VC $\leq$ $\mathbf{MWG}_{\text{einfach,lokal sternförmig}}$) *VC kann mit einfachen, lokal sternförmigen MW-Graphen gelöst werden.*

Beweis 3.124 Gegeben ein endlicher, einfacher, ungerichteter Graph $G = \langle V, A\rangle$ und eine natürliche Zahl $k \in \mathbf{N}$. Wir wollen eine Konfiguration $\mathcal{C} = \langle \mathcal{F}, \mathcal{N}, \mathcal{D}, \kappa\rangle$ angeben, die genau dann konsistent ist, wenn G ein VC der Größe k hat.

Wir benötigen also im besten Fall auch eine Entsprechung der Zeugen und beginnen daher mit einem *unbestimmten* Feld $V \subset \mathcal{F}$ für jeden Knoten in G mit $V \cap \mathcal{D} = \emptyset$. Diese Felder sollen die *charakteristische Funktion* des VC U darstellen, müssen also genau k Minen enthalten. Darüber hinaus muss für je zwei benachbarte Knoten aus V *mindestens* eines der beiden Felder vermint sein.

Die Lösung ist in Abbildung 3.7 auf Seite 77 angegeben. Die Äquivalenz der Instanzen folgt aus der Interpretation besagter Minenindikatoren als charakteristische Funktion zusammen mit den notwendigen und hinreichenden Einschränkungen für diese Minen. Die Konfiguration hat $|\mathcal{F}| = \alpha_0 + 2\alpha_1 + 1$ Felder und $\Sigma\mathcal{N} = \alpha_0 + 3\alpha_1$ explizit angegebene einfache Nachbarschaften. κ ist mit $\max\{2, k\}$ beschränkt. □

Satz 3.125 (CLIQUE $\leq$ $\mathbf{MWG}_{\text{einfach,lokal sternförmig}}$) *CLIQUE kann mit einfachen, lokal sternförmigen MW-Graphen gelöst werden.*

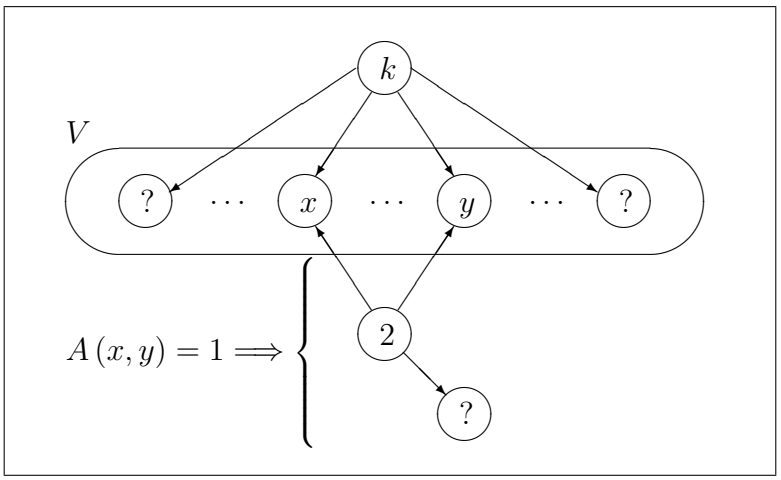

Abbildung 3.7: Reduktion von VC auf MWG

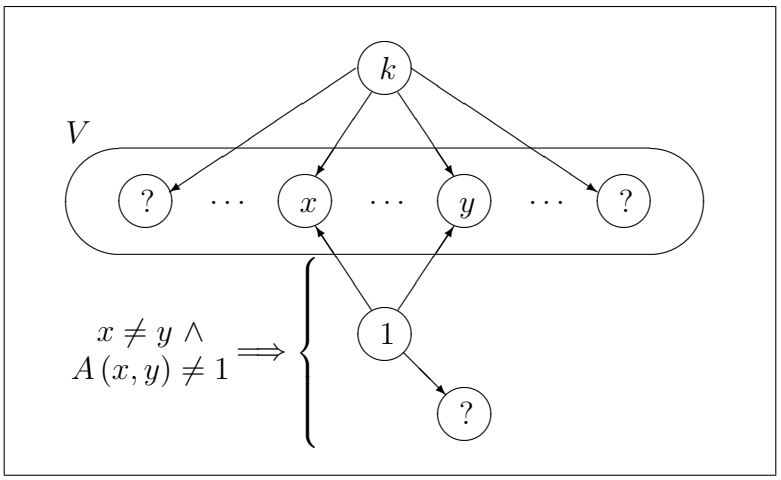

Abbildung 3.8: Reduktion von CLIQUE auf MWG

Beweis 3.125 Wir übernehmen im Wesentlichen das Spielfeld aus Beweis 3.124, müssen aber die Einschränkung für eine CLIQUE umformulieren:

$$\begin{aligned}
&\forall x, y \in U: && x \neq y && \rightarrow A(x,y) = 1 \\
\Leftrightarrow\ &\forall x, y \in V: && (x \neq y \wedge x \in U \wedge y \in U) && \rightarrow A(x,y) = 1 \\
\Leftrightarrow\ &\forall x, y \in V: && (x \neq y \wedge A(x,y) \neq 1) && \rightarrow (x \notin U \vee y \notin U)
\end{aligned}$$

Die Lösung ist in Abbildung 3.8 auf Seite 77 angegeben. Die Äquivalenz der Instanzen folgt aus der Interpretation besagter Minenindikatoren als charakteristische Funktion zusammen mit den notwendigen und hinreichenden Einschränkungen für diese Minen. Die Konfiguration hat $|\mathcal{F}| = 2\alpha_0^2 - \alpha_0 - 2\alpha_1 + 1$ Felder und $\Sigma\mathcal{N} = 3\alpha_0^2 - 2\alpha_0 - 3\alpha_1$ explizit angegebene einfache Nachbarschaften. κ ist mit $\max\{1, k\}$ beschränkt. □

Da das Konzept der charakteristischen Funktion der Knotenmenge vorgegebener Größe bei den beiden Reduktionen von CLIQUE und VC auf MWG identisch ist, und sich

die überprüfenden Kanten-Knoten-Kanten-Spangen nicht widersprechen, kann MWG beide Fragestellungen für den selben Graphen synchron lösen:

Bemerkung 3.126 (synchrone CLIQUE und VC) *Existiert auf G mit $\alpha_0 := |V|$ und $\alpha_1 := |E|$ eine k-elementige Knotenmenge, die* gleichzeitig *CLIQUE und VC ist? Diese Frage entscheidet eine entsprechende Überlagerung der Spielfelder aus Beweis 3.124 und 3.125 mit $2\alpha_0^2 - \alpha_0 + 1$ Feldern und $3\alpha_0^2 - 2\alpha_0$ Nachbarschaften.*

Bemerkung 3.127 (CLIQUE = VC) *Je nach der Dichte $\alpha_1 / (\alpha_0^2 - \alpha_0)$ des gegebenen einfachen ungerichteten Graphen kann das Verwenden des MW-Graphen von VC für CLIQUE und umgekehrt unter der Angabe von $\alpha_0 - k$ statt k einen kleineren MW-Graphen ergeben. Das entspricht genau der Komplementbildung.*

Problem 3.128 (3DM) *Gegeben drei gleichmächtige endliche Mengen X, Y und Z mit $k := |X| = |Y| = |Z|$ und eine dreistellige Relation $R \subseteq X \times Y \times Z$ auf diesen. Existiert eine k-elementige Teilmenge $U \subseteq R$ der Relation mit $|U| = k$, die jeden Wert aus jeder Menge X, Y und Z in einem Tupel enthält?*

$$\forall x \in X, y \in Y, z \in Z : \exists u_x, u_y, u_z \in U : x = \pi_1(u_x) \wedge y = \pi_2(u_y) \wedge z = \pi_3(u_z)$$

Die Darstellung ist auf einer 3TM leicht möglich: Jede Achse stellt eine Menge dar, der aufgespannte Würfel die Relation mit $\Sigma = 2$. Die Syntax kann durch eine Querung entlang der Raumdiagonale überprüft werden. Wenn ein Zeuge durch markierte Werte 1 der Relation angegeben wird, so muss in *jeder* $X \times Y$-, $X \times Z$- und $Y \times Z$-Ebene genau ein markierter Einser $\bar{1}$ stehen.

Satz 3.129 (SAT $\leq$ 3DM) *Mit 3DM kann SAT gelöst werden.*

Beweis 3.129 Gegeben eine Formel $F = C_1 \wedge \ldots \wedge C_m$ mit s Variablen $x_1, \ldots, x_s$. Wir geben die erste Mengen der Größe $2ms$ an, die jede Variable in beiden möglichen Literalen für jede Klausel einmal enthält: $X := \left\{x_{i_j}, \bar{x}_{i_j} : 1 \leq i \leq s, 1 \leq j \leq m\right\}$.

Die anderen beiden Mengen enthalten zunächst jede Klausel einmal:

$$Y \supset \{C_1, \ldots, C_m\} \subset Z$$

Wir simulieren für jede Klausel die „Wahl“ eines ihrer Literale, das zu 1 ausgewertet wird, mit den Tupeln der Relation

$$R \supset \left\{\left(y_{i_j}, C_j, C_j\right) : 1 \leq j \leq m,\ y_{i_j} \in \left\{x_{i_j}, \bar{x}_{i_j}\right\} \text{ kommt in } C_j \text{ vor}\right\}.$$

Diese Wahl darf aber nie ein Literal und seine Negation gleichzeitig treffen. Mit

$$R \supset \{(x_{i_1}, a_{i_1}, b_{i_1}), (\bar{x}_{i_1}, a_{i_1}, b_{i_2}), \ldots, (x_{i_m}, a_{i_m}, b_{i_m}), (\bar{x}_{i_m}, a_{i_m}, b_{i_1}) : 1 \leq i \leq s\}$$

können wir das verhindern. Es entspricht einer zyklischen MW-Informationsleitung, weil für jedes $1 \leq i \leq s$ jedes a_i und jedes b_i ein x_i oder ein $\bar{x}_i$ benötigen um in $U \subseteq R$ getroffen zu werden: Es bleiben entweder genau die Variablen oder ihre Negationen gleichzeitig über um von einer Klausel gewählt zu werden.

Jetzt enthalten die Mengen Y und Z jeweils $m+ms$ Elemente, die a_i und b_i definieren jeweils eine Belegung auf den x_i und die C_j suchen ihre Auswertung zu 1. Falls F also nicht erfüllbar ist, gibt es kein entsprechendes Matching. Wenn F aber erfüllbar ist, können wir eine Untermenge $U \subseteq R$ der Relation so angeben, dass alle a_{i_j}, b_{i_j} und C_j und genau $m+ms$ der $2ms$ Literale getroffen werden. Diese können wir nun noch durch die fehlenden Elemente in Y und Z abdecken:

$$R \supset \left\{ \left(x_{i_j}, d_r, d_r\right), \left(\bar{x}_{i_j}, d_r, d_r\right) : 1 \leq i \leq s, 1 \leq j \leq m, 1 \leq r \leq ms - m \right\}.$$

Dadurch enthalten alle drei Mengen X, Y und Z jeweils $2ms$ Elemente. Für jedes Matching gilt, dass die a und b für jede m-fache Ausführung einer Variable x synchron alle x oder alle $\bar{x}$ abgedeckt werden. Jede Klausel deckt ein weiteres für sie bestimmtes Literal ab und die $ms - m$ „Dummy"-Elemente d genau die fehlenden Literale. Die Matchings entsprechen also eineindeutig den erfüllenden Belegungen. □

Satz 3.130 (3DM $\leq$ MWG$_{\text{einfach,lokal sternförmig,d}^- \leq 3, \sigma \leq 1, \kappa \equiv 1}$) *3DM kann mit einfachen, lokal sternförmigen MW-Graphen mit beschränkten Hingraden $d^- \leq 3$ und Knotenbeschriftungen $\sigma \leq 1$ gelöst werden.*

Beweis 3.130 Gegeben drei gleichmächtige endliche Mengen X, Y und Z mit $k := |X| = |Y| = |Z|$ und eine dreistellige Relation $R \subseteq X \times Y \times Z$ auf diesen mit $m := |R|$. Wie bei VC wollen wir wieder die charakteristische Funktion vom Zeugen $U \subseteq R$ darstellen. Das Tupel werden wir als Nachbarschaft zu den entsprechenden Elementen der drei Mengen modellieren und die Bedingung, dass jedes Element in einem Tupel vorkommen muss, durch die Werte 1 auf den entsprechenden Feldern.

Die Lösung ist in Abbildung 3.9 auf Seite 80 angegeben. Zu beweisen bleibt, dass jeder konsistente Zeuge *genau k* Minen enthält. Zu jedem Feld $r \in R$ führt aber *genau eine* Kante aus jeder der Mengen X, Y und Z. Außerdem führen *alle Kanten nach R*, daher gilt zum Beispiel

$$k = \sum_{x \in X} \kappa(x) = \sum_{x \in X} \sigma(x) = \sum_{x \in X} \sum_{r \in R} \mathcal{N}(x, r)\, \mu(r) = \sum_{r \in R} \left(\sum_{x \in X} \mathcal{N}(x, r) \right) \mu(r) = \sum_{r \in R} \mu(r).$$

Die Konfiguration hat $|\mathcal{F}| = 3k+m$ Felder und $\Sigma\mathcal{N} = 3m$ explizit angegebene einfache Nachbarschaften. σ ist mit 1 beschränkt. □

Problem 3.131 (KNAPSACK$^+$) *Gegeben $k+1$ natürliche Zahlen a_i, $i \in k$ und $b = a_k$, existiert eine Teilmenge $U \subseteq k$, sodass sich die entsprechenden Zahlen zu b aufsummieren?*

$$\sum_{i \in U} a_i = b$$

Die Funktion $a : k+1 \to \mathbf{N}$ kann leicht auf einer 2TM mit Zahlen zu einer beliebigen Basis dargestellt werden. Zeugen in Form von Markierungen an der Achse, werden durch aufsummieren in der letzten Spalte überprüft. Die selbe Vorgehensweise kann natürlich auch für ganze Zahlen $a : k+1 \to \mathbf{Z}$ verwendet werden. Außerdem sehen wir sofort $KNAPSACK^+ \leq KNAPSACK$. Letzteres steht für das Problem mit gleicher Fragestellung, das negative Zahlen nicht ausschließt.

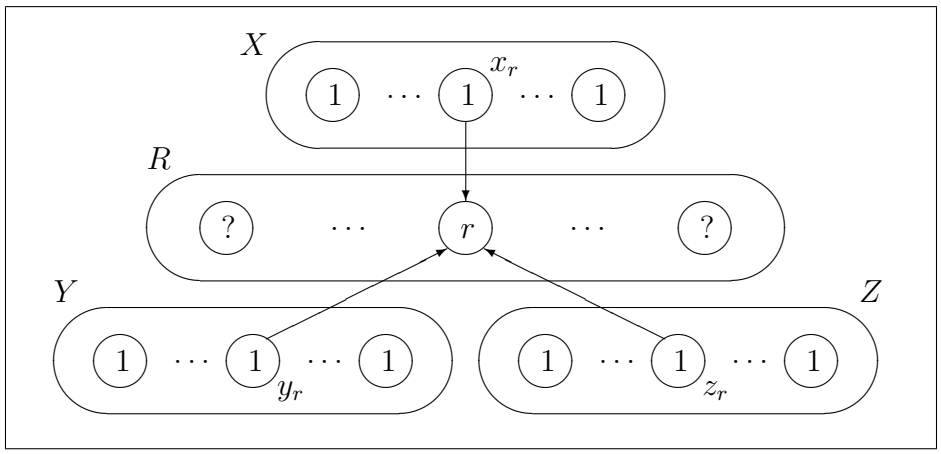

Abbildung 3.9: Reduktion von 3DM auf MWG

Satz 3.132 (3DM $\leq$ KNAPSACK$^+$) *Mit $KNAPSACK^+$ kann 3DM gelöst werden.*

Beweis 3.132 Gegeben drei gleichmächtige endliche Mengen X, Y und Z mit $k := |X| = |Y| = |Z|$ und eine dreistellige Relation $R \subseteq X \times Y \times Z$ auf diesen mit $m := |R|$. Die Elemente der endlichen Mengen X, Y, Z und R seinen nummeriert.

Wir geben die Zahlen in der zu konstruierenden $KNAPSACK^+$-Instanz als *Binärzahlen* mit $3km$ Stellen an: Jeder Block der Länge m stellt dabei ein Element aus X, Y oder Z dar. Wir veranschaulichen uns diese Überlegung mit der Definition der $a_i,\ i \in m$:

$$a_i := 2^{m(2k+u-1)} + 2^{m(k+v-1)} + 2^{m(w-1)} \quad mit\ \ (x_u, y_v, z_w) =: r_i \in R$$

Jedes a_i enthält nun genau drei Einser, die jeweils am Ende des Blockes der Länge m jenes Elements stehen, das sie repräsentieren. Die drei großen Blöcke der Länge $k \cdot m$ stehen für die drei Mengen X, Y und Z in dieser Reihenfolge. In einem Matching muss jeder der kleinen Blöcke genau einmal getroffen werden, also den Wert 1 enthalten:

$$a_m = b := \sum_{j=0}^{3k-1} 2^{mj}$$

Für ein Matching $U \subseteq R$ gilt offensichtlich $b = \sum_{r_i \in U} a_i$. Da die kleinen Blöcke aber selbst bei Summierung über alle Tupel der Relation $|R| = m$ keinen Überlauf produzieren, gilt auch, dass jede solche Summierung ein Matching angibt. Die Konstruktion ist sicher *nicht* exponentiell, wenn die Werte 2^{mj} nicht durch Potenzieren, sondern durch Abzählen der Stellen berechnet werden. □

Satz 3.133 (KNAPSACK$^+ \leq$ MWG$_{\text{einfach,lokal sternförmig,eben}}$) *$KNAPSACK^+$ kann mit einfachen, lokal sternförmigen, ebenen MW-Graphen gelöst werden.*

Beweis 3.133 Gegeben $k+1$ natürliche Zahlen $a_i,\ i \in k$ und $b = a_k$. Wir formen die KNAPSACK-Bedingung für eine Teilmenge $U \subseteq k$ um:

$$b = \Sigma_{i \in U} a_i = \Sigma_{i \in k} a_i \chi_U(i) =: \Sigma_{i \in k} \mathcal{N}(v_b, i)\, \mu(i).$$

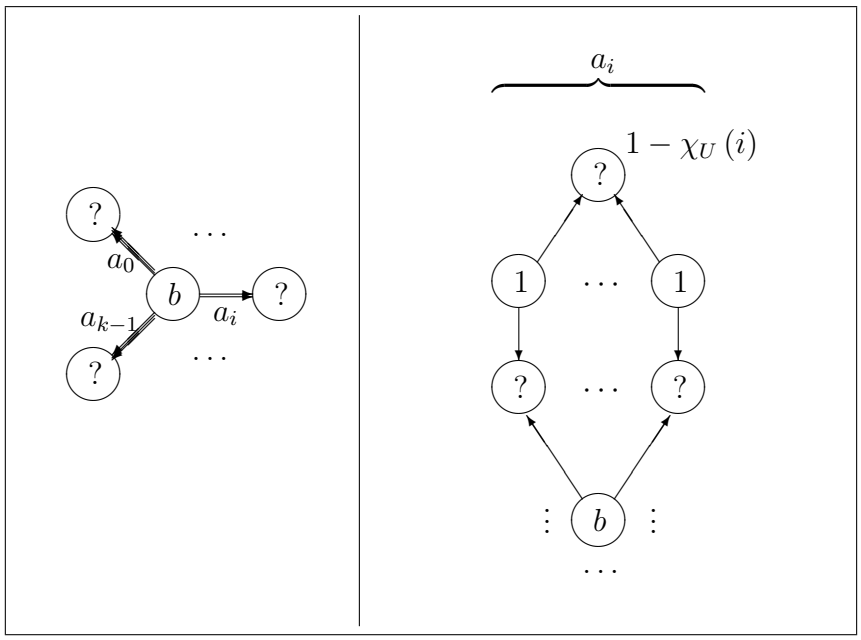

Abbildung 3.10: Reduktion von $KNAPSACK^+$ auf MWG

Das Konzept ist dadurch klar vorgegeben. Wir zeigen in Abbildung 3.10 auf Seite 81 auch eine Lösung für einfache MW-Graphen. Der MW-Graph mit *Mehrfachkanten* hat $k+1$ Felder und k Mehrfachnachbarschaften mit jeweils bis zu $\max_{i \in k} a_i$ parallelen Kanten. Er ist also relativ klein und sehr übersichtlich.

Der einfache Graph benötigt $O\left(k + 1 + 2\sum_{i \in k} a_i\right)$ Felder und $O\left(3\sum_{i \in k} a_i\right)$ einfache Nachbarschaften. Die Angabe erfolgt als obere Schranke, weil der Fall $a_i = 1$ einfacher modelliert werden kann. □

Durch Ersetzen der Beschriftung $\kappa(v_b) = b$ mit ? und Hinzufügen eines Feldes mit Wert 0 und Nachbar v_b, kann ein MW-Graph angegeben werden, dessen konsistente Belegungen genau alle möglichen Summen $\Sigma_{i \in U} a_i,\ U \in k$ in v_b generieren. Diese und weitere denkbare Anwendungen sind aber keine Probleme mit Entscheidungsfragen mehr. Die Anmerkung soll nur darauf hinweisen, dass in dieser Arbeit lange nicht alle Aspekte der Erweiterung von Minesweeper auf Graphen ausgeschöpft wurden. Einige Anregungen werden wir noch in Abschnitt 4.2.2 geben.

4 Umfeld und Ausblick

Dieses Kapitel soll einen kurzen Überblick über Minesweeper geben. Das Minesweeper-(Konsistenz-)Problem ist zwar ein guter wissenschaftlicher Ansatz zur Komplexitätsanalyse aber für menschliche Spieler wenig hilfreich.

Das Ende des Kapitels enthält eine Zusammenfassung der Arbeit und der gefundenen Ergebnisse sowie einen Ausblick auf weiterführende Aspekte von Minesweeper im Allgemeinen und MWG im Speziellen.

4.1 Praktisches Spielen

Wie schon bei der Einführung zu Minesweeper erwähnt, ist die Bewertung für einen Spieler die Zeit, die er zum Aufdecken aller sicheren Felder benötigt. Die Felder mit Minen und deren Anzahl sind dabei a priori festgelegt, der Computer garantiert aber ein sicheres erstes Feld. Das heißt die Festlegung geschieht erst nachdem sich der Spieler für das Feld entschieden hat, das er zuerst erkunden will.

Da das Spielen eine mechanische Komponente hat, nämlich das Navigieren über das Spielfeld mit Hilfe der Computermaus und das Erkunden/Öffnen der Felder durch Klicken der linken Maustaste oder durch klicken beider Maustasten auf einem Feld mit *relativer Explosivität* gleich 0, stellt sich die Frage, welche Mindestdauer für ein Spiel möglich ist und wie diese von der Aufteilung der Minen abhängt. Wir wollen hier das – empirisch getestete – Konzept des ehemaligen Minesweeper-Spielers Stephan Bechtel vorstellen:

4.1.1 Einfache Spiele

Wie schnell ein Spiel gespielt werden kann, hängt davon ab – aber nicht ausschließlich –, wie viele Klicks mit der Maus ausgeführt werden müssen. Dieser Wert ist einfach zu berechnen, wenn angenommen wird, dass ein Spieler Felder nur mit der linken Maustaste öffnet und nicht die Möglichkeit verwendet, alle Nachbarn eines Feldes ohne relative Explosivität auf einmal zu öffnen. Sehrwohl beachten wir aber die Tatsache, dass der Computer alle Nachbarn eines Feldes ohne reale Umgebungsgefahr sofort und rekursiv öffnet.

Bezeichnung 4.1 (3BV) B*echtel's* B*oard* B*enchmark* V*alue (3BV), also Bechtels Richtwert für ein Spiel, ist die* minimale Anzahl *von Linksklicks, die benötigt wird um alle sicheren Felder eines Spielfeldes mit einer vorgegebenen Anordnung der Minen zu öffnen, wenn ausschließlich Linksklicks zu diesem Zweck verwendet werden.*

Bemerkung 4.2 (non-flagging Style) *Wenn ein Spieler* keine Fahnen *setzt um gefundene Minen zu markieren, nennt man das* non-flagging Style. *Offensichtlich kann*

3BV auch als minimale Anzahl Klicks zum Lösen eines Spiels im non-flagging Style definiert werden, weil die relative Explosivität für ein Feld stets gleich der tatsächlichen ist.

Wir wollen nun 3BV mit den Bezeichnungen der Modellierung des Standardspiels aus Abschnitt 2.1.2 angeben. Wir setzen die dortigen Definitionen und Erkenntnisse fort.

Defintion und Lemma 4.3 (Insel) $\mathcal{I} \subseteq \mathcal{F}$ *heißt* Insel *genau dann, wenn*

(a) $\mathcal{I} \neq \emptyset$,

(b) $\forall x \in \mathcal{I} : \sigma(x) = 0$,

(c) $\forall x, y \in \mathcal{I} : \ \exists l \in \mathbf{N} : \ \exists x = x_0, x_1, \ldots, x_l = y \in \mathcal{I} : \ \forall i \in l : x_{i+1} \in \mathcal{N}(x_i)$,

(d) $\mathcal{I}$ *ist maximal mit (a)–(c).*

Inseln *sind also genau die maximalen Mengen benachbarter Felder ohne Umgebungsgefahr. Jedes solche Feld liegt* auf genau einer *Insel.*

Beweis 4.3 Die Maximalität einer Insel widerspricht der Annahme, ein Feld ohne Umgebungsgefahr könnte auf mehr als einer Insel liegen. □

Lemma 4.4

$$\mathcal{I} \text{ Insel } \Rightarrow \ \mathcal{I} \cap \mathcal{N}(\mathcal{M}) = \emptyset$$

Beweis 4.4 Angenommen $\exists x \in \mathcal{N}(\mathcal{M}) : \sigma(x) = 0$, wähle $b \in \mathcal{M}$ mit $x \in \mathcal{N}(b)$. Wegen der Symmetrie von δ folgt $b \in \mathcal{N}(x)$ und daraus direkt der Widerspruch mit $\sigma(x) \geq 1$. □

Bezeichnung 4.5 (Rand, Strand) *Für ein* $X \subseteq \mathcal{F}$ *bezeichnen wir die Menge* $\mathcal{R}_X := \mathcal{N}(X) \setminus X$ *als* Rand. *Für Inseln* $\mathcal{I}$ *sagen wir auch* Strand $\mathcal{S}_\mathcal{I}$.

Satz 4.6 *Es gilt*

(a) $(\mathcal{S}_\mathcal{I} \cap \mathcal{M}) \subseteq (\mathcal{N}(\mathcal{I}) \cap \mathcal{M}) = \emptyset$ *(Am Strand gibt es keine Bomben)*

(b) $\mathcal{F} = \mathcal{N}(\mathcal{M}) + \sum \{\mathcal{I} \subseteq \mathcal{F} : \text{Insel}\}$

(c) $\forall x \in \mathcal{F} : x \in \mathcal{S}_\mathcal{I} \ \Rightarrow \ 0 < \epsilon(x) = \sigma(x) \leq 5$

(d) $\mathcal{R}_{\sum \mathcal{I}} = \bigcup \mathcal{S}_\mathcal{I} \subseteq \mathcal{R}_\mathcal{M}$ *(*$\sum$ *und* $\bigcup$ *durchlaufen* $\{\mathcal{I} \subseteq \mathcal{F} : \text{Insel}\}$*)*

Beweis 4.6 (a) und (b) folgen direkt aus der Definition mit der Symmetrie der Metrik und Lemma 4.4.

(c) Die scharfe Ungleichung folgt aus der Maximalität der Insel. Nun nehmen wir an $x \in \mathcal{S}_{\mathcal{I}}$. Sei $y \in \mathcal{N}(x)$ mit $\sigma(y) = 0$. Wegen $|\mathcal{N}(x) \cap \mathcal{N}(y)| \geq 4$ (kein Feld der angegebenen Menge ist aus $\mathcal{M}$) folgt $\sigma(x) \leq 9 - 4 = 5$.

(d) Die Inklusion folgt direkt aus (c). $\mathcal{R}_{\sum \mathcal{I}} \subseteq \bigcup \mathcal{S}_{\mathcal{I}}$ ist klar. Für die Gegenrichtung ist nur mehr zu zeigen, dass kein Feld des Strandes einer Insel auf einer anderen Insel liegt. Das ist klar wegen der Maximalität der Insel. □

3BV wird – unter Verwendung der hier definierten Bezeichnungen – berechnet als Anzahl der Inseln plus Anzahl der sicheren Felder, die nicht an einem Strand liegen:

Defintion und Lemma 4.7 (3BV) *Der in Bezeichnung 4.1 eingeführt 3BV ist*

$$\begin{aligned} 3BV(\mathcal{F},\mathcal{M}) &= \left|\mathcal{R}_{\mathcal{M}} \setminus \bigcup_{\{\mathcal{I}\subseteq\mathcal{F}:\text{Insel}\}} \mathcal{S}_{\mathcal{I}}\right| + |\{\mathcal{I} \subseteq \mathcal{F} : \text{Insel}\}| = \\ &= \left|\mathcal{R}_{\mathcal{M}} \setminus \bigcup_{\{\mathcal{I}\subseteq\mathcal{F}:\text{Insel}\}} \mathcal{N}(\mathcal{I})\right| + |\{\mathcal{I} \subseteq \mathcal{F} : \text{Insel}\}| = \\ &= |\mathcal{R}_{\mathcal{M}} \setminus \mathcal{N}(\{x \in \mathcal{F} : \sigma(x) = 0\})| + |\{\mathcal{I} \subseteq \mathcal{F} : \text{Insel}\}| = \\ &= M \cdot N - K - |\mathcal{N}(\{x \in \mathcal{F} : \sigma(x) = 0\})| + |\{\mathcal{I} \subseteq \mathcal{F} : \text{Insel}\}|. \end{aligned}$$

Beweis 4.7 Die Ausdrücke der ersten drei Zeilen sind offensichtlich gleich, der Schritt zur letzten Zeile folgt aus Lemma 4.6. $MN - K$ ist die Anzahl der sicheren Felder. Solche die auf einer Insel oder einem Strand liegen, werden durch einen einzelnen Klick auf die zugehörige Insel geöffnet. Die letzen beiden Summanden berücksichtigen diese Überlegung. □

Diese Funktion $3BV(M,N,K)$ hängt außer von der *Spielfeldgröße* und der Anzahl der versteckten *Minen* auch von der *Anzahl*, *Größe* und *Form* der Inseln ab. Eine Analyse der Funktion für eine (fest) vorgegebene Verteilung der $\binom{MN}{K}$ möglichen Aufteilungen der Minen wäre wünschenswert, scheint aber sehr schwierig. Ebenso wären andere Funktionen zur Bestimmung der Einfachheit eines Spieles denkbar.

Für MWP-Konfigurationen, die aus der Reduktion von SAT nach der in Lemma 2.27 argumentierten Funktion gewonnen werden, ist die untere Schranke für diese Bewertung $3BV \in \Omega(cs) = \Omega(MN)$. Die Konfigurationen enthalten keine Inseln $\mathcal{N}(\mathcal{M}) = \mathcal{F}$, und obwohl es $K \geq \frac{1}{3}MN$ viele Minen gibt, sind drei Viertel der Felder in den horizontalen Informationsleitungen sicher. Das reicht für die angegebene untere Schranke.

4.1.2 Strategien

Es gibt in Minesweeper-Konfigurationen, die aus einem Spiel entstehen, viele *Standardsituationen*, die sich aus der relativen Explosivität mehrerer benachbarter Felder ergeben. Als klassische Beispiele seien

a	b	c	d	e
∞	1	2	1	∞
∞	∞	∞	∞	∞

und

u	v	w	x	y	z
∞	1	2	2	1	∞
∞	∞	∞	∞	∞	∞

angeführt. Angenommen c wäre vermint, so bleibt weder b noch d für die zweite Mine des mittleren Feldes. Daher sind genau b und d in der linken Konfiguration vermint, während a, c und e sicher sind. Angenommen v wäre vermint, so wäre die relative Explosivität der Felder 2|2|1 in der rechten Konfiguration gleich 1|2|1. Daraus folgt, dass v sicher ist mit Widerspruch. Aus Symmetriegründen folgt, dass genau w und x aus den Feldern $u, \ldots, z$ vermint sind.

Wie bereits erwähnt steht uns mit einer effizienten Lösung von MWP ein Weg zur Verfügung, die Sicherheit eines Feldes oder mit 9-fachem Aufwand auch eine Mine zu bestätigen, *falls dies eindeutig ist*. Das entspricht jeweils *sicheren Ereignissen*, wenn man Wahrscheinlichkeiten für die Sicherheit eines Feldes aufstellen will. Offensichtlich ist das Berechnen oder Bestätigen solcher Werte daher *schwieriger* als MWP – wie bereits von Kaye in [8] erwähnt. Der direkte Weg ist das Ermitteln *aller* konsistenten Aufteilungen der Minen und Abzählen jener, für die das gefragte Feld sicher bzw. vermint ist.

Hier haben wir wieder einen Punkt gefunden, in dem MWP als Konsistenz-Problem nicht exakt das Spielen von Minesweeper nachzeichnet: Der Spieler kennt die Anzahl K der Minen, die auf dem Spielfeld verteilt sind. Ohne diese „Einschränkung“ beträgt die Wahrscheinlichkeit, dass ein Feld $x \in \mathcal{F} \setminus \mathcal{N}(\mathcal{D})$, über das keine Information vorliegt, vermint ist, genau $1/2$, weil es zu jeder konsistenten Aufteilung mit einer Mine auf gefragtem Feld eine solche gibt, in der dieses sicher ist, und umgekehrt. Durch diese Vorgabe werden die Wahrscheinlichkeiten für vollkommen unbestimmte Felder zwar immer noch gleich – aber im Allgemeinen verschieden von 50% – sein.

Spielstrategien wurden unter Anwendung verschiedenster Werkzeuge bereits mehrfach untersucht. Weiterführend zu diesem Thema sind zum Beispiel [1], [3], [7], [12] und [15].

4.2 Zusammenfassung

In *diesem Abschnitt* sind Verweise auf Inhalte der vorliegenden Arbeit am rechten Seitenrand ohne weitere Erwähnung angegeben. Kapitel und Abschnitte referenzieren wir in runden, Definitionen und *Erkenntnisse* in eckigen Klammern jeweils unter Angabe von fortlaufender Nummer und Seitenzahl. (4.2, 85) [0.1, vii]

Die Arbeit bietet einen direkten Einstieg in die Komplexitätstheorie über die *Turing-Maschinen*. Wir behandeln diese ausführlich um eine lückenlose Beweisführung für spätere Resultate zu gewährleisten. Das Ziel des ersten Kapitels ist es, die Grundlagen für die Komplexitätsanalyse von Minesweeper ausgehenden von mathematischen Grundkenntnissen zu wiederholen. Die Begriffe Komplexitätsklasse (NP), *Reduktion* und *NP-Vollständigkeit* werden definiert und erklärt. Außerdem führen wir den Beweis, dass NP-vollständig nicht leer ist anhand des Referenzproblems (CNF)SAT, das wir auch für den Vollständigkeitsbeweis von Minesweeper in NP benötigen. (1.1, 1) (1.2.1, 11) [1.30, 12] (1.3, 14)

Im zweiten Kapitel stellen wir das Spiel Minesweeper kurz vor und *modellieren* es durch mathematische Objekte. Anschließend wird die Theorie aus dem ersten Kapitel angewendet um die Komplexität von Minesweeper zu untersuchen. Der in den Grundzügen nachgezeichnete Beweis von Kaye in [8] wird durch eine *exakte Darstellung* auf Turing-Maschinen ergänzt. Mit einem geringfügig *neuen Konzept* können wir auch *gute explizite Schranken* für die Spielfeldgröße auf Minesweeper reduzierter SAT-Instanzen angeben. (2.1.2, 24) (2.2.1, 27) [2.19, 29] [2.27, 34]

In weiterer Folge werden Varianten von Minesweeper betrachtet: Das *lineare* und [3.28, 44]

mehrdimensionale Spiel wurden bereits in anderen Arbeiten analysiert, hier ergänzen wir die Betrachtungen durch ausführliche *automaten-* und *sprachtheoretische Aspekte*, die aufgrund der fundierten Grundlagen der ersten Kapitel vergleichsweise schnell angeführt werden können. (3.2.3, 52) (3.1.1, 36) [3.42, 47]

In der zunächst gefundenen mathematischen Darstellung für Minesweeper erkennen wir parallelen zur Definition von *Graphen*. Ein Grundverständnis für Graphen wird angenommen, und wir geben nur einen kurzen Begriffsüberblick vor der Definition von *allgemeinem Minesweeper* auf Graphen, das alle Varianten von Minesweeper – wie zum Beispiel Minesweeper auf Sphären und diskreten Netzen von beliebigen Mannigfaltigkeiten und Fraktalen – enthält. Anschließend analysieren wir die *Komplexität* dieser Erweiterung und untersuchen die Grenze zwischen einfachen und schwierigen Instanzen auf *speziellen Graphen*. [3.71, 54] (3.3.2, 58) [3.92, 61] (3.3.3, 62)

Dieses letzte Kapitel bietet außer der Zusammenstellung der Ergebnisse einen kurzen Überblick über andere Sichtweisen auf Minesweeper. Im Vordergrund steht dabei die Ansicht eines *menschlichen Spielers*. Die *Modellierung* des Standardspiels wird für diese Belange erweitert ohne Ergebnisse hierfür zu erwarten. (4.1, 82) [4.7, 84]

4.2.1 Ergebnisse

Wir wollen hier *alle* wesentlichen Ergebnisse, die in dieser Arbeit enthalten und bewiesen sind, auflisten. Die Reihenfolge entspricht etwa dem Verlauf des Textes.

- Die *Probleme* SAT, 3SAT, CLIQUE, VC, 3DM und KNAPSACK sind NP-vollständig. (1.3, 14) (3.3.4, 74)

- Das Minesweeper-(Konsistenz-)Problem (MWP) ist *NP-vollständig* und die Reduktion von SAT können wir unter Einhaltung folgender Schranken durchführen: Gegeben eine (CNF)SAT-Instanz, deren s Variablen insgesamt c Mal in der Formel auftreten, so kann eine äquivalente MWP-Instanz der Größe $(4c-1) \times (3s+7)$ konstruiert werden. (2.2.2, 28) [2.27, 34]

- *Lineares Minesweeper* kann mit einem endlichen Automaten gelöst werden. Wir finden einen einfach interpretierbaren Regulären Ausdruck, der dessen Sprache beschreibt. (3.1.3, 44) [3.35, 46]

- *Mehrdimensionales Minesweeper* (MWd) ist nicht schwieriger als das Standardspiel. Es existiert eine kontextsensitive Grammatik, die genau die Sprache der konsistenten Instanzen beschreibt. (3.2.3, 52) [3.68, 53]

- Minesweeper auf endlichen *allgemeinen Graphen* (MWG) ist nicht schwieriger als das Standardspiel. (3.3.2, 58) [3.92, 61]

- Minesweeper auf Graphen mit *beschränktem Knotengrad* $d \leq 2$ ist nicht schwieriger als lineares Minesweeper. Es existiert eine kontextfreie Grammatik, die genau die Sprache der konsistenten Instanzen von Minesweeper auf *Bäumen* beschreibt. [3.107, 68] [3.111, 71] [3.116, 74]

- Minesweeper auf *einfachen, lokal sternförmigen Graphen* ist bereits schwierig, also vollständig in NP. Diese Graphen sind auch schlingenlos, eindeutig und bipartit. (3.3.3, 62) [3.100, 64]

Letzteres folgt aus lokal sternförmig. Die beiden Knotenmengen, innerhalb derer keine Kanten verlaufen, sind die bestimmten und die unbestimmten Felder. Zusätzlich gilt, dass alle Kanten von den *bestimmten Feldern wegführen*. [3.97, 63]

- Minesweeper auf *einfachen, lokal sternförmigen Graphen* ist sogar bereits schwierig wenn wahlweise eine der folgenden Mengen von Einschränkungen verlangt wird:
  - *eben*, $d^+ \leq 4$, $d^- \leq 3$, $\sigma \leq 2$: beschränkte Weg- und Hingrade und Umgebungsgefahren sowie eine ebene Zeichnung [3.105, 66]
  - $d^- \leq 3$, $\sigma \leq 1$, $\kappa \equiv 1$: beschränkte Hingrade und Umgebungsgefahren, die nur zwei Werte annehmen können (3.3.4, 74) [3.130, 79]

  In letzterem Fall sind Felder nur mit 1 und ? beschriftet, und konsistente Zeugen entscheiden über 0 und Mine (∞) auf den zunächst unbestimmten Feldern.

4.2.2 Weiterführende Fragen

Ganz allgemein ist die Frage, ob $P = NP$ gilt, offen und ihre Beantwortung mit einer Million Dollar dotiert ([4]). Minesweeper kann ein Weg dazu sein.

Weitere Fragen ergeben sich sowohl aus der Strategie- als auch aus der Komplexitätsanalyse. Zunächst sei an die Funktion [4.7, 84]

$$3BV(\mathcal{F}, \mathcal{M}) = M \cdot N - K - |\mathcal{N}(\{x \in \mathcal{F} : \sigma(x) = 0\})| + |\{\mathcal{I} \subseteq \mathcal{F} : \text{Insel}\}|$$

erinnert, sowie an die Möglichkeit, eine objektive Untergrenze für ein Standardspiel durch eine andere Funktion zu beschreiben. Eine Formalisierung und Klassifikation des folgenden verbal beschriebenen Problems ist auch offen. Man kann weder ausschließen, dass es äquivalent zu einem bereits untersuchten Problem ist, noch dass es schwierig ist.

Problem 4.8 (Besuch aller sicheren Felder) *Gegeben sei ein Standard-Minesweeper-Spielfeld $\mathcal{F}$ und eine* bekannte *Aufteilung der Minen $\mathcal{M} \subseteq \mathcal{F}$ sowie eine natürliche Zahl $k \in \mathbf{N}$. Gibt es eine Möglichkeit alle sicheren Felder mit (höchstens) k Mausklicks zu öffnen? Dabei gelten das direkte Öffnen, das Markieren einer Mine und das Öffnen aller Nachbarn eines Feldes ohne relative Explosivität jeweils als ein Klick.*

Für das Problem MWG stellen sich weitere Fragen: [3.88, 59]

- Kann Minesweeper auf *Bäumen* deterministisch in polynomieller Zeit gelöst werden? Ein Ansatz könnte sein, bei Blättern beginnend analog zu linearem MW auf erlaubte Nachfolger zu testen.

- Graphen mit beschränkten Knotengraden $d \leq 2$ ergeben einfache MW-Instanzen, solche mit $d \leq 4$ sind im Allgemeinen bereits schwierig. Kann eine Aussage über Graphen mit Knotengraden $d \leq 3$ gemacht werden? Gibt es einen Zusammenhang zwischen dieser Frage und jener bezüglich $P = NP$?

Abgesehen davon gibt es viele weitere Eigenschaften von Graphen, von denen nicht geklärt ist, ob und wie sie die Komplexität von MWG beeinflussen. Und schlussendlich könnten andere oder weitere Verallgemeinerungen der Idee von Minesweeper auch für andere mathematische Fachrichtungen interessant sein.

Abbildungsverzeichnis

Tabellenverzeichnis

Literaturverzeichnis

[1] Andrew Adamatzky (1997): *How Cellular Automaton Plays Minesweeper*, Applied Mathematics and Computation 85, Seiten 127–137
`http://uncomp.uwe.ac.uk/adamatzky/papers.htm`

[2] Lourdes Peña Castillo, Stefan Wrobel (2002): *Multirelational Active Learning for Games*, Machine Learning Workshop FGML[1] 2002
`http://kd.cs.uni-magdeburg.de/~pena/papers/FGML02.pdf`

[3] Lourdes Peña Castillo, Stefan Wrobel (2003): *Learning Minesweeper with Multirelational Learning*, Proceedings of IJCAI[2] 2003
`http://kd.cs.uni-magdeburg.de/~pena/papers/IJCAI03.pdf`

[4] Stephen Cook (2000): *The P versus NP Problem*, University of Toronto; Official Problem Description, Clay Mathematics Institute
`http://www.claymath.org/millennium/P_vs_NP`

[5] Reinhard Diestel (1996): *Graphentheorie*, Springer-Verlag Berlin Heidelberg

[6] John Hopcroft, Rajeev Motwani, Jeffrey Ullman (2002): *Einführung in die Automatentheorie, Formale Sprachen und Komlexitätstheorie*, Pearson Studium, Pearson Education Deutschland GmbH

[7] Meredith Kadlac (2003), *Explorations of the Minesweeper Consistency Problem*, Humboldt State University, Arcata, CA
`http://www.math.oregonstate.edu/~math_reu/REU2003/Proceedings.htm`

[8] Richard Kaye (2000): Minesweeper is NP-complete, *The Mathematical Intelligencer*, Volume 22, Number 2/2000, Seiten 9–15
`http://web.mat.bham.ac.uk/R.W.Kaye/publ.htm`

[9] Richard Kaye (2000): *Some Minesweeper Configurations*, The University of Birmingham, School of Mathematics and Statistics, B15 2TT
`http://web.mat.bham.ac.uk/R.W.Kaye/minesw/minesw.pdf`

[10] Peter Linz (2001): *An Introduction to Formal Languages and Automata*, Jones and Bartlett Publishers

[11] Kurt Mehlhorn (1984): *Data Structures and Algorithms*, 3 Bände, Springer-Verlag Berlin Heidelberg

[1] Treffen der GI-Fachgruppe 1.1.3 Maschinelles Lernen
[2] International Joint Conference on Artificial Intelligence

[12] Preslav Nakov, Zile Wei (2003): *Minesweeper, #Minesweeper*, Report
`http://www.cs.berkeley.edu/~zile/CS294-7-Nakov-Zile.pdf`

[13] Jürgen Oberhofer (2003): *The architechture of SAT solvers and their applicability to NP-complete problems*, Diplomarbeit: Technische Universität Wien, Institut für Informationssysteme

[14] Wolfgang Paul, Rüdiger Reischuk (1981): On Time versus Space II, *Journal of Computer and System Sciences*, Volume 22, Number 3/1981, Seiten 312–327.
`http://www.tcs.uni-luebeck.de/pages/reischuk/Publikationen.html`

[15] Kasper Pedersen (2004): *The complexity of Minesweeper and strategies for game playing*, Project 2003-2004: University of Warwick, Department of Computer Science
`http://www.dcs.warwick.ac.uk/~kasper/Minesweeper/Files/report.pdf`

[16] Peter Sander, Wolffried Stucky, Rudolf Herschel (1995), *Automaten Sprachen Berechenbarkeit*, B. G. Teubner Stuttgart

[17] Joel Seiferas (1977): Linear-Time Computation by Nondeterministic Multidimensional Iterative Arrays, *SIAM Journal on Computing*, Volume 6, Number 3/1977, Seiten 487–504.
`http://locus.siam.org/fulltext/SICOMP/volume-06/0206035.pdf`

[18] *Weitere Internet-Links*:

Minesweeper wissenschaftlich: `http://www.heinrichmartin.com/mw`
`http://www.dcs.warwick.ac.uk/~kasper/Minesweeper`
`http://web.mat.bham.ac.uk/R.W.Kaye/minesw/minesw.htm`

Clay Mathematics Institute: `http://www.claymath.org/millennium`

Wikipedia: `http://en.wikipedia.org/wiki/3BV`
`http://en.wikipedia.org/wiki/NP-complete`
`http://en.wikipedia.org/wiki/Minesweeper_%28computer_game%29`

Minesweeper Spieler-Seiten: `http://www.dotti.at/ms`
`http://www.stephan-bechtel.de`
`http://stb.st.funpic.de/3bv.htm`
`http://www.metanoodle.com/minesweeper`
`http://www.chez.com/demineur/ring/index.html`
`http://www.planet-minesweeper.com/class_home.php`