THEORETISCHE INFORMATIK 2

Ergänzungen zum Skript von 2017

Jürgen Koslowski

Theoretical Computer Science

Technische Universität Braunschweig

Juni 2019

Inhaltsverzeichnis

0 Einführung **0**

1 Entscheidbarkeit und Berechenbarkeit **0**

1.0 Turing-Maschinen und Entscheidbarkeit 0
1.0.1 Entscheidungs- und Wortprobleme 0
1.0.2 Turing-Maschinen 1
1.0.3 Varianten von Turing-Maschinen und Determinisierung 6
1.0.4 Entscheidbarkeit und Abschlußeigenschaften 10
1.1 Berechenbarkeit 12
1.2 Unentscheidbarkeit 19
1.2.1 Codierung von Turing-Maschinen 19
1.2.2 Die universelle Turing-Maschine 21
1.2.3 Das Halteproblem 21
1.2.4 Reduktionen 22
1.3 Weitere unentscheidbare Probleme 25
1.3.1 Das Post´sche Korrespondenzproblem 25
1.3.2 Der Satz von Rice 29
1.4 Unentscheidbare Probleme kontextfreier Sprachen 32

2 Komplexitätstheorie **35**

2.0 Zeit- und Raumkomplexität 35
2.0.1 Zeitkomplexität 35
2.0.2 Raumkomplexität 36
2.0.3 Die robusten Komplexitätsklassen 37
2.0.4 Komplementklassen 39
2.0.5 Simple Relationen zwischen Komplexitätsklassen 40
2.1 Eine Landkarte der Komplexität 41
2.2 $\boldsymbol{L}$ und $\boldsymbol{NL}$ 42
2.2.1 Probleme in $\boldsymbol{L}$: Arithmetik 42
2.2.2 Das Pfadproblem in $\boldsymbol{NL}$ 42
2.2.3 Reduktionen, Härte und Vollständigkeit 44
2.2.4 $\boldsymbol{Path}$ ist $\boldsymbol{NL}$-vollständig 48
2.3 co$\boldsymbol{NL}$ und der Satz von Immermann und Szelepcsényi 49
2.3.1 2-SAT 50
2.3.2 Der Satz von Immermann und Szelepcsényi 54

copyright: Jürgen Koslowski, TU Braunschweig, 19. Juli 2019

2.4 ***P*** 58
2.4.1 Das Circuit-Value-Problem 58
2.5 ***NP*** 63
2.5.1 ***NP***-vollständige Probleme 64
2.5.2 Zertifikatssprachen 68
2.6 ***PSPACE*** und der Satz von Savitch 70

copyright: Jürgen Koslowski, TU Braunschweig, 19. Juli 2019

0 Einführung

[altes Skript]

1 Entscheidbarkeit und Berechenbarkeit

1.0 Turing-Maschinen und Entscheidbarkeit

Ziel: Welche Probleme sind *algorithmisch* lösbar?

Zunächst beschränken wir uns auf Probleme mit einer Lösung im Boole'schen Wahrheitsbereich $\mathbb{B} = \{0, 1\}$, sog. *Entscheidungsprobleme.* Später werden wir auch Antworten in Δ^* zulassen, für ein endliches Alphabet Δ. Dann spricht man von *Berechnungsproblemen.* Wegen $\mathbb{B} \subseteq \mathbb{B}^*$ ist jedes E-Problem auch ein W-Problem.

1.0.1 Entscheidungs- und Wortprobleme

1.0.00 Definition. Gegeben sei eine (typischerweise) unendliche disjunkte Vereinigung $I = I_{\text{yes}} + I_{\text{no}}$, deren Elemente wir *Instanzen* nennen. Das zugehörige *Entscheidungsproblem* fragt für eine Instanz $i \in I$, ob $i \in I_{\text{yes}}$ gilt. In kompakter Form:

Entscheidungsproblem zu $I = I_{\text{yes}} + I_{\text{no}}$ **Gegeben:** eine Instanz $i \in I$ **zu entscheiden:** ob $i \in I_{\text{yes}}$ gilt

$\triangleleft$

I.A. kann das nicht durch bloßes Hinschauen beantwortet werden, sondern erfordert einen gewissen Aufwand an (Speicher-)Platz und Zeit. Oft gelingt es, den Instanzen eine *Größe* $|i| \in \mathbb{N}$ zuzuordnen und den Aufwand in Abhängigkeit von $|i|$ abzuschätzen ($\rightarrow$ Komplexität).

Viele interessante Probleme haben Instanzen, die aus Texten, Zahlen, Formeln, Quellcodes *etc.* bestehen, sich also als *Wörter* über einem geeigneten endlichen Alphabet Σ darstellen lassen.

1.0.01 Definition. Gegeben sei eine Teilmenge $L \subseteq \Sigma^*$, also eine formale Sprache über Σ. Das zugehörige *Wortproblem* fragt für eine Instanz $w \in \Sigma^*$, ob $w \in L$ gilt.

Wortproblem zu $L \subseteq \Sigma^*$ **Gegeben:** eine Instanz $w \in \Sigma^*$ **zu entscheiden:** ob $w \in L$ gilt

$\triangleleft$

Damit ist ein Wortproblem ein Entscheidungsproblem bzgl. $\Sigma^* = L + (\Sigma^* - L) = L + \overline{L}$. Als Länge einer Instanz kann man in diesem Fall die Wortlänge verwenden.

Fast jedes E-Problem läßt sich als W-Problem über einem geeigneten Alphabet darstellen.

copyright: Jürgen Koslowski, TU Braunschweig, 19. Juli 2019

1.0.02 Beispiel. Aus TheoInf 1 kennen wir bereits folgende Wortprobleme und dafür geeignete Algorithmen:

▷ das Wortproblem für *reguläre Sprachen*; zu lösen durch Simulation eines passenden deterministischen Automaten;

▷ das Wortproblem für *kontext-freie Sprachen*, die mittels CFG in (einer Variante der) Chomsky-Normalform (CNF) spezifiziert sind; zu lösen mittels CYK-Algorithmus. ◁

Im Gegensatz fallen die aus TheoInf 1 bekannten Algorithmen zur Eliminierung spontaner τ-Übergänge, die Umformung eines NFA in einen EA sowie dessen Minimierung, oder die Umformung einer CFG in eine in CNF nicht in die Klasse der E-Probleme, sondern es handelt sich um Berechnungsprobleme, siehe Abschnitt 1.1.

Viele andere Wortprobleme sind algorithmisch lösbar, die sich nicht als kontext-freie Sprachen formulieren lassen. Das erfordert aber ein mächtigeres Automatenmodel.

1.0.2 Turing-Maschinen

Während endliche Automaten (ob deterministisch oder nicht) und PDAs beide über einen endlichen *internen Speicher* in Form von *Zuständen* verfügen, können PDAs zudem auf einen potentiell(!) unendlichen, aber zu jedem Zeitpunkt endlichen *externen Speicher* in Form eines *Stacks* (auch *Stapelspeicher* oder *Keller* genannt) zugreifen. Die Möglichkeit des Zugriffs ist allerdings eingeschränkt: der Schreib-Lese-Kopf kann nur das Top-Element des Stacks sehen und ggf. durch ein Wort aus dem Stack-Alphabet ersetzen, wodurch der Stack seine Größe verändern kann.

Turing Maschinen wurden bereits 1936 von **Allan Turing** (1912–1954) eingeführt. Sie besitzen einen endlichen internen sowie einen unendlichen externen Speicher in Form eines beidseitig unbeschränkten *Turing-Bandes* auf dessen einzelne Zellen ein Schreib-Lese-Kopf unbeschränkten Zugriff hat, und sich folglich bewegen kann. Sie existieren in diversen Varianten, die letztendlich alle äquivalent sind.

PDAs fanden dagegen erst Ende der 50'er Jahren des 20. Jahrhunderts Verbreitung (1957 schlug **Charles Hamblin** (1922–1985) die umgekehrt Polnische Notation (RPN) zur Verwendung mit einem Stack vor; im gleichen Jahr beantragten **Friedrich L. Bauer** (1924–2015) und **Klaus Samelson** (1918–1980) ein Patent zur Nutzung des Stapelspeichers bei der Übersetzung von Programmiersprachen, das 1962 erteilt wurde; 1959 wurden PDAs unter dem Namen *General Problem Solver* (G.P.S.) von A. Newell, J.C. Shaw und H.A. Simon vorgestellt). Das Prinzip des Stacks wurde zwar schon 1945/1946 ebenfalls von Turing im Rahmen militärischer Forschungen zum Bau eines "richtigen" Computers entdeckt, die aber lange der Geheimhaltung unterlagen.

Insofern sollte man TMs *nicht* als Verallgemeinerungen von PDAs betrachten. Sie unterscheiden sich zudem auch in der Handhabung der Eingabe.

copyright: Jürgen Koslowski, TU Braunschweig, 19. Juli 2019

Alle seither entwickelten Formalisierungen des Konzepts algorithmischer Lösbarkeit (ohne Quanteneffekte!) haben sich als höchstens so mächtig wie TMs herausgestellt, $\to$ **Church-Turing-These**:

Alles, was intuitiv algorithmisch berechnet werden kann, läßt sich mit einer TM berechnen.

1.0.03 Definition. Eine Turing-Maschine $M = \langle Q, \Sigma, \Gamma, \delta, q_0, q_{\text{acc}}, (q_{\text{rej}})\rangle$ besteht aus

- ▷ einer endlichen Zustandsmenge Q;
- ▷ einem endlichen Eingabe-Alphabet Σ;
- ▷ einem endlichen Band-Alphabet Γ mit $\Sigma + \{\sqcup\} \subseteq \Gamma$; $\sqcup$ heißt *Leerzeichen* oder *Blank*;
- ▷ ausgezeichneten Zuständen: dem *Anfangszustand* q_0, dem *akzeptierenden Zustand* q_{acc}, und, optional, dem *abweisenden Zustand* $q_{\text{rej}} \neq q_{\text{acc}}$; die beiden letzteren fasst man auch als *Haltezustände* zusammen;
- ▷ einer Übergangsrelation $Q \times \Gamma \xrightarrow{\delta} Q \times \Gamma \times \{L, N, R\}$, die folgender Bedingung genügt: für alle $a \in \Gamma$ gilt

$$\langle q_{\text{acc}}, a\rangle \text{ und } \langle q_{\text{rej}}, a\rangle \quad \text{erlauben genau einen Übergang nach} \quad \langle q_{\text{rej}}, a, N\rangle \text{ bzw. } \langle q_{\text{rej}}, a, N\rangle$$

M heißt *deterministisch*, sofern δ eine partielle (!) Funktion ist. ◁

Die Zeichen L, R und N sollen die Bewegung des Kopfes relativ zum Band andeuten, nach links, rechts oder keine Bewegung (Not). Das wird erst in Definition 1.0.04(2) der Transitionen zwischen Konfigurationen einer TM deutlich. Man beachte, dass keine externen Eingabesymbole in die Übergänge einfließen, es sind nur spontane oder τ-Übergänge definiert. Die Zusatzbedingung besagt, dass nach Erreichen eines Haltezustands die Maschine in diesem und der Kopf auf der aktuellen Bandzelle verweilt (ideling).

Eine an endliche Automaten angelehnte graphische Darstellung von TMs verwendet die Zustände als Knoten und Kanten der Form

mit $\langle c, X\rangle \in \mathcal{B} \times \{N, L, R\}$. Anfangs- und Akzeptanzzustand werden markiert wie zuvor. Man beachte, dass man derartige Pfeile nicht beliebig komponieren kann, da sie keine Information über den Inhalt benachbarter Speicherzellen beinhalten.

Dafür bedarf es Konfiguration, die den gesamten aktuellen Bandinhalt, die Kopfposition und den momentanen Zustand beschreiben. Sofern man mit einer endlichen Eingabe auf einem sonst leeren Band startet, sind immer nur endlich viele Speicherzellen nicht leer. Damit lassen sich

copyright: Jürgen Koslowski, TU Braunschweig, 19. Juli 2019

Konfigurationen als Tripel $\langle u|q|v\rangle \in \Gamma^* \times Q \times \Gamma^*$ realisieren, wobei u mindestens alle links der Kopfposition auftretenden Elemente aus $\Gamma - \{\sqcup\}$ abdeckt, und $v \neq \varepsilon$ alle rechts davon stehenden Elemente aus $\Gamma - \{\sqcup\}$ umfasst, sowie die Kopfposition selbst als linken Rand. Insbesondere besteht keine Symmetrie zwischen u und v! (In der Literatur findet man auch andere Konventionen.) Aus pragmatischen Gründen wollen wir nicht darauf bestehen, dass u und v minimal sind; sie dürfen links bzw. rechts überflüssige Blanks enthalten.

1.0.04 Definition.

(0) Eine *Konfiguration* ist ein Äquivalenzklasse von $\Gamma^* \times Q \times \Gamma^*$ unter der Relation, die alle Tripel der Form $\langle u|q|v\rangle$, $\langle \sqcup u|q|v\rangle$, $\langle u|q|v\sqcup\rangle$ sowie $\langle u|q|v\sqcup\rangle$ identifiziert; wir werden im Folgenden nur mit Repräsentanten rechnen. Konfigurationen der Form $\langle \varepsilon|q_0|, w\rangle$ mit $w \in \Sigma^*$ heißen *Startkonfigurationen*.

(1) δ definiert folgende *Transitionen* zwischen Konfigurationen:

$$\begin{array}{cccl} \langle u|q|av\rangle & \longrightarrow & \langle ub|p|v\rangle & \text{falls } \langle q,a\rangle\,\delta\,\langle p,b,R\rangle \\ \langle uc|q|av\rangle & \longrightarrow & \langle u|p|cbv\rangle & \text{falls } \langle q,a\rangle\,\delta\,\langle p,b,L\rangle \\ \langle u|q|av\rangle & \longrightarrow & \langle u|p|bv\rangle & \text{falls } \langle q,a\rangle\,\delta\,\langle p,b,N\rangle \end{array}$$

Damit bilden die Konfigurationen und die Transitionen einen gerichteten Graphen, den *Konfigurationsgraphen*.

(2) Unter *Haltekonfigurationen* versteht man solche, in denen entweder ein Haltezustand auftritt, oder aus denen keine Transitionen herausführen. Haltekonfigurationen mit dem Zustand q_{acc} heißen *akzeptierend*, alle anderen *abweisend*. $\lhd$

1.0.05 Definition. Die maximalen Pfade im Konfigurationsgraphen, die mit Startkonfigurationen beginnen, wollen wir als *Berechnungen* interpretieren. Diese können vier mögliche Verhalten zeigen:

- ▷ sie erreichen in endlich vielen Schritten eine akzeptierende Konfiguration (und ideln dann dort); in diesem Fall heißt auch die Berechnung *akzeptierend*;
- ▷ sie erreichen in endlich vielen Schritten eine abweisende Konfiguration (und ideln dann dort); in diesem Fall heißt auch die Berechnung *abweisend*;
- ▷ sie haben endliche Länge, erreichen also eine Konfiguration, aus der keine Transition heraus führt; dann ist per Definition der Übergangsrelation kein Haltezustand erreicht. Solche Berechnungen heißen ebenfalls *abweisend*;
- ▷ sie bleiben nicht stecken, aber erreichen auch nie einen Haltezustand. Solche Berechnungen sollen *nicht-terminierend* heißen.

copyright: Jürgen Koslowski, TU Braunschweig, 19. Juli 2019

M *akzeptiert* $w \in \Sigma^*$ genau dann, wenn es eine akzeptierende Berechnung mit Startkonfiguration $\langle \varepsilon | q_0 | w \rangle$ gibt.

$$\mathcal{L}(M) := \{\, w \in \Sigma^* \;:\; \exists u, v \in \Gamma^*.\, \langle \varepsilon | q_0 | w \rangle \longrightarrow^* \langle u | q_{\text{acc}} | v \rangle \,\} \qquad \lhd$$

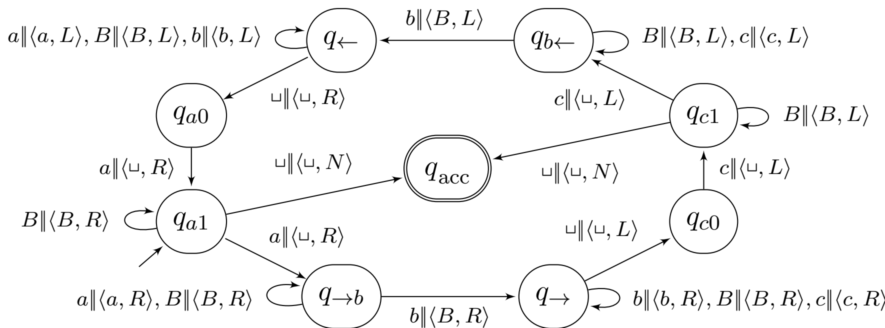
In einer deterministischen TM gibt es von jeder Konfiguration aus genau einen Pfad; der kann endlich oder unendlich sein.

1.0.06 Beispiel. Um $L = \{\, a^n b^n c^n \;:\; n \in \mathbb{N} \,\} \subseteq \{a, b, c\}^*$ mit einer TM zu akzeptieren, verwenden wir das Bandalphabet $\Gamma = \{a, b, c, B, \#\}$, wobei B dazu dient anzuzeigen, dass an dieser Stelle ein b bearbeitet wurde.

Idee: Pro Durchlauf vom linken zum rechten Rand sollen drei Zeichen a, b und c in dieser Reihenfolge bearbeitet werden: im Anfangszustand q_{a1} das erste a von links löschen, das erste b von links suchen und mit B überschreiben, den rechten Rand suchen und ggf. ein links davon stehendes c löschen; dieses erfordert zwei Hilfszustände q_{c0} und q_{c1}.

Dann kann ggf. ein entsprechender Durchlauf vom rechten zum linken Rand folgen, bei dem ein c, ein b und ein a in dieser Reihenfolge bearbeitet werden sollen. (Alternativ könnte man auch zum linken Rand zurückkehren.)

Akzeptanz, wenn bei einem Durchlauf vom einen zum anderen Rand nur noch ein Wort aus $\{B\}^*$ gefunden wird. Beachte die Symmetrie!



Diese TM ist deterministisch im oben angegebenen Sinn, d.h., δ ist eine partielle Funktion. Wollte man auf einer totalen Funktion δ bestehen, würde die TM sehr unübersichtlich werden. $\lhd$

1.0.07 Definition. Eine Sprache $L \subseteq \Sigma^*$ heißt *semi-entscheidbar* (alternativ auch *rekursiv aufzählbar* oder *Turing-erkennbar*), wenn es eine TM M gibt mit $L = \mathcal{L}(M)$. $\lhd$

Der Begriff der semi-entscheidbaren Sprache provoziert mehrere Fragen:

- ▷ Gibt es Sprachen, die nicht semi-entscheidbar sind? Das folgt mit Hilfe eines leichten Abzählbarkeitsarguments, s.u..
- ▷ Können deterministische TMs alle semi-entscheidbaren Sprachen erkennen? Das ist korrekt, aber der Beweis ist aufwändiger als im Fall der EAs.

- ▷ Wie steht es mit der Ausdrucksfähigkeit von Maschinen mit mehr als einem Band oder einem einseitig unendlichen Band?
- ▷ Was sind "entscheidbare Sprachen"?

1.0.08 Satz. *Für jedes endliche Alphabet Σ gibt es nur abzählbar unendlich viele semi-entscheidbare Sprachen, während überabzählbar viele Sprachen nicht semi-entscheidbar sind.*

Beweis. Wie in der Logik-VL (oder auch in TheoInf-1) gezeigt, ist $\Sigma^* = \bigcup\{\, \Sigma^n : n \in \mathbb{N} \,\}$ als abzählbare Vereinigung endlicher Sprachen abzählbar, die Potenzmenge $P(\Sigma^*)$ aber nicht.

Andererseits gibt es bis auf Isomorphie, d.h., Umbenennung der Zustände und der Bandsymbole, nur abzählbar viele TMs mit dem Alphabet Σ: bei fester Zustandszahl und fester Größe des Bandalphabets gibt es nur endlich viele Übergangsrelationen. Also haben wir es wieder mit einer abzählbaren Vereinigung endlicher Mengen semi-entscheidbarer Sprachen zu tun. □

Eine konkrete nicht-semi-entscheidbare Sprache anzugeben muß aber noch etwas warten.

Aufgrund geschichtlichen Entwicklung ist der Zusammenhang zwischen TMs und PDAs etwas kompliziert.

1.0.09 Bemerkungen.

(0) Um einem PDA zu einer TM zu verallgemeinern, muß man die Einschränkungen beim Speicherzugriff aufheben um den Kopf auf dem Stack wandern zu lassen. Solange der Kopf überall lesen, aber nur nur am Stackende schreiben darf, spricht man von *Stack-Automaten*; diese sind schwächer als TMs. Also muß man das Schreiben überall zulassen. Um das lokale Verlängern oder Verkürzen des Stacks vermeiden (was an Masochismus grenzt), muß der verfügbare Speicher zum Ausgleich von vornherein einseitig unendlich sein, mit bis auf endlich viele Ausnahmen leeren Speicherzellen; man kann ihn als einseitig unendliches Band betrachten. Da der Kopf das Band nicht verlassen darf, braucht man ein spezielles end-of-tape-Symbol, ganz ähnlich wie das end-of-stack-Symbol, das bei seitwärtiger Betrachtung des Stacks dessen Ende signalisierte.

Weiterhin ist bei einer TM zunächst die gesamte Eingabe einzulesen und auf das Band zu schreiben; die eigentliche Arbeit erfolgt anschließend mittels spontaner oder τ-Übergänge.

(1) Um bei PDAs den leeren Stack zuzulassen und die mit dem end-of-stack-Symbol einhergehenden praktischen Probleme zu umgehen, muß der Kopf den Stack von vorne statt von der Seite betrachten. Erlaubt man nun die Bewegung des Kopfes, so erhält man einen 2PDA mit zwei in entgegengesetzte Richtungen weisenden Stacks, deren Endpunkte der Kopf sehen kann. Zudem kann in beiden Richtungen ein nichtleeres Wort geschrieben oder das aktuelle Symbol gelöscht werden; insbesondere erlaubt das die Simulation von Bewegung des Kopfes. 2PDAs erweisen sich nur als äquivalent zu TMs. Auch wenn sie eleganter zu sein scheinen und etwas effizienter sind, konnten sie die etablierten TMs aber nicht mehr verdrängen.

copyright: Jürgen Koslowski, TU Braunschweig, 19. Juli 2019

1.0.3 Varianten von Turing-Maschinen und Determinisierung

Für eine TM M und eine Anfangskonfiguration $\langle \varepsilon | q_o | w \rangle$ lassen sich die Berechnungen im Konfigurationsgraphen als geschichteter Baum mit dieser Anfangskonfiguration as Wurzel in Schicht 0 darstellen. Die Schichten entsprechen der Länge der Berechnung. Achtung: á priori kann dieselbe Konfiguration in einer Schicht mehrfach vorkommen; fasst man diese Vorkommen zusammen, entstehen keine gerichteten Kreise, auch wenn der Begriff "Baum" nun etwas seltsam anmutet. Eine Determinisierung von M sollte diesen Baum systematisch danach durchsuchen, ob irgendwo eine akzeptierende Konfiguration auftaucht. Da unendliche Berechnungen möglich sind, ist zwingend eine Breiten-Suche anzuwenden.

Eine 1-Band-TM, die dieses Programm ausführt, dürfte recht unübersichtlich werden und schwer zu verifizieren sein. Mit einer Mehrband-TM ließe sich das viel einfacher umsetzen. Dazu ist zunächst die Äquivalenz zwischen 1-Band und Mehrband-TMs zu etablieren. Als Zwischenschritt verwenden wir Mehrspur-TMs (Stichwort: Tonbandgeräte).

1.0.10 Definition. Für $n > 0$ entspricht eine *(deterministische) n-Spur-TM* über Σ mit Bandalphabet $\Gamma \supseteq \Sigma + \{\sqcup\}$ einer (deterministischen) TM über $\Sigma \times \{\#\}^{n-1}$ mit Bandalphabet Γ^n. Als Blanksymbol dieser 1-Band Maschine dient das n-Tupel $\langle \sqcup, \ldots, \sqcup \rangle$.

Der Kopf liest/schreibt immer in allen Spuren gleichzeitig, und die Bewegung erfolgt auch in allen Spuren synchron in derselben Richtung.

1.0.11 Satz. *Von Mehrspur-TMs akzeptierte Sprachen sind semi-entscheidbar.* □

1.0.12 Definition.

(0) Für eine *n-Band TM* $\boldsymbol{M} = \langle Q, \Gamma, n, \Sigma, \delta, q_0, q_{\text{acc}}, (q_{\text{rej}}) \rangle$ hat δ die Form

$$Q \times \Gamma^n \xrightarrow{\delta} Q \times (\Gamma \times \{L, R, N\})^n$$

(1) Konfigurationen sind nun Äquivalenzklassen von $(\Gamma^*)^n \times Q \times (\Gamma^*)^n$, und Anfangskonfigurationen haben die Eingabe auf Band 0, alle übrigen Bänder sind leer. Die graphische Darstellung verwendet Label der Form $\beta \| A_0; A_1; \ldots; A_{n-1}$ mit $\beta \in \Gamma^n$ und n Aktionen $A_i \in \Gamma \times \{L, N, R\}$. ◁

Wesentlich ist, dass die n Köpfe unabhängig auf den n Bändern agieren können. Hier zeigt sich der Nutzen der Option N zum Verweilen auf dem aktuellen Feld. Ohne sie müssten alle Köpfe bewegt werden, wenn nur auf einem Band gearbeitet werden soll.

1.0.13 Satz. *Von n-Band TMs akzeptierte Sprachen sind semi-entscheidbar.*

copyright: Jürgen Koslowski, TU Braunschweig, 19. Juli 2019

Beweis. Idee: Simulation durch eine $2n$-Spur-TM, bei der die geraden Spuren den Bändern entsprechen, und auf deren ungeraden Spuren über die jeweiligen Kopfpositionen des Bandes auf der Vorgängerspur buchgeführt wird, durch Markierungen in der relevanten Spalte. Nicht terminierende Berechnungen dieser Simulation resultieren zwingend von ebensolchen der Originalmaschine. □

Wie effizient ist diese Simulation einer n-Band TM durch eine 1-Band-TM? Der Bedarf ein Speicherplatz verdoppelt sich im Wesentlichen, während parallele Arbeitsschritte der n-Band TM nun sequentiell abzuarbeiten sind, und zwar an potentiell n Stellen auf dem n-spurigen Band. Der größte Zusatzaufwand besteht dann im Hin- und Herfahren des Kopfes der n-Spur Maschine zu den aktuell markierten Kopfpositionen für die einzelnen Bänder. Deren maximaler Abstand entspricht der 2-fachen Schrittzahl der Ursprungsmaschine wodurch sich die Laufzeit der Simulation im Wesentlichen quadrieren kann.

1.0.14 Beispiel. Die Sprache $L = \{\, w \in \{0,1\}^* \,:\, |w|_0 = |w|_1^2 \,\}$ ist semi-entscheidbar:

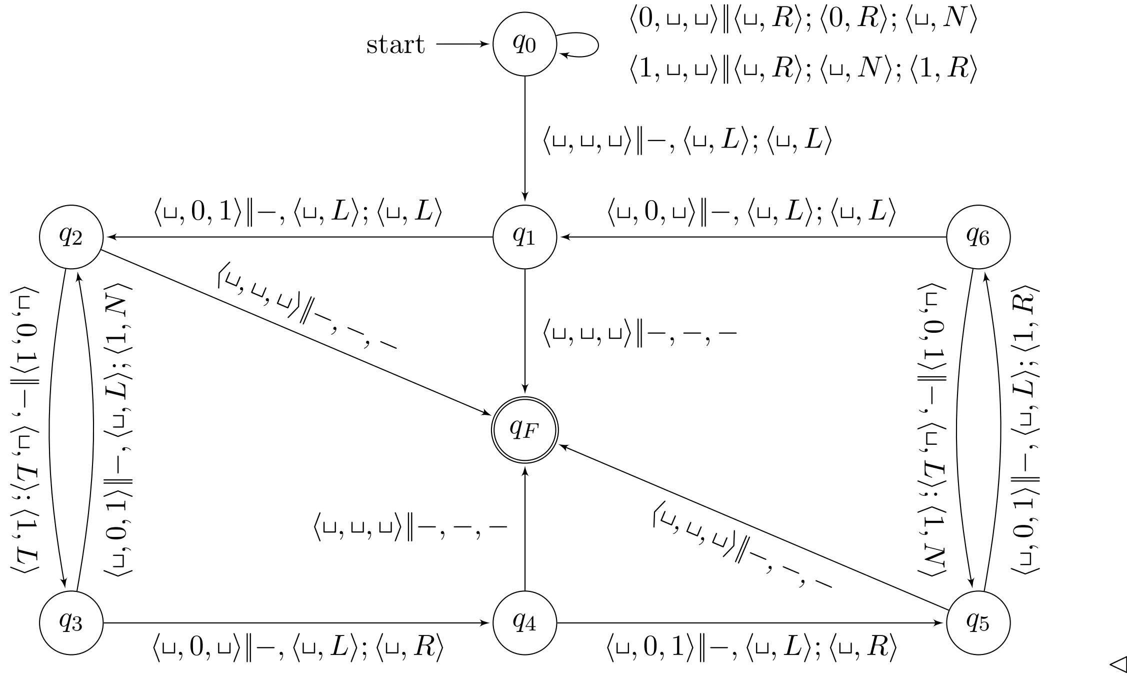
Idee: Wir verwenden eine 3-Band dTM, bei der die Nullen und Einsen der Eingabe auf B_0 zunächst auf die Bänder B_1 bzw. B_2 verschoben werden. Dort finden alle weiteren Berechnungen statt. Um das Problem mit n Einsen und n^2 Nullen auf das Problem mit $n-1$ Einsen und $(n-1)^2$ Nullen zurückzuführen, wollen wir eine Eins löschen. Wegen

$$(n-1)^2 = n^2 - 2n + 1 = n^2 - 2(n-1) - 1$$

sind dann $2(n-1)+1$ viele Nullen zu löschen. Eine davon können wir parallel mit der Eins löschen. Dann sind für jede der verbliebenen $n-1$ Einsen noch je zwei Nullen zu löschen, was z.B. in einem Durchlauf durch die Einsen auf B_2 von einem zum anderen Ende geschehen kann. (Alternativ wäre auch möglich, die Einsen auf B_2 erst von rechts nach links and dann wieder zurück zu durchlaufen.) In der nächsten Runde werden die Einsen auf B_2 in der anderen Richtung durchlaufen, während die Nullen auf B_1 immer in derselben Richtung gelöscht werden. Nur wenn die Löschung aller Symbole gelingt, wird die Eingabe akzeptiert.

Bei der folgenden Realisierung sind Aktionen, die weder Kopfposition noch Bandinhalt

verändern, durch “ − ” abgekürzt.



◁

In Zukunft werden wir nicht alle TMs so detailliert beschreiben können, sondern uns meist auf informelle Spezifikationen beschränken.

1.0.15 Satz. *Jede semi-entscheidbare Sprache wird von einer dTM akzeptiert.*

Beweis. Betrachte eine 1-Band TM $\boldsymbol{M} = \langle Q, \mathcal{B}, \Sigma, \delta, q_0, q_{\mathrm{acc}}, (q_{\mathrm{rej}})\rangle$ Für jedes Paar $\langle q, b\rangle \in Q \times \mathcal{B}$ bezeichne $\rho\langle q, b\rangle$ die Anzahl der verfügbaren Übergänge; diese werden von 0 bis $\rho\langle q, b\rangle - 1$ durchnummeriert. Setze

$$r := \max\{\, \rho\langle q, b\rangle \,:\, \langle q, b\rangle \in Q \times \mathcal{B}\,\} \quad , \quad R := \lceil \log_2 r \rceil \quad \text{und} \quad \mathbb{Z}_R := \{\, n \in \mathbb{N} \,:\, n < R\,\}$$

Falls $r > 1$ ist $\boldsymbol{M}$ nicht deterministisch und wir konstruieren eine deterministische 4-Band Maschine $\boldsymbol{M}'$ mit $\mathcal{L}(\boldsymbol{M}') = L$:

- ▹ B_0 enthält die Eingabe $w \in \Sigma^*$;
- ▹ auf B_1 werden systematisch die Zahlen $k \in \mathbb{N}$ erzeugt, unär mit einem geeigneten Bandsymbol;
- ▹ auf B_2 werden systematisch die k-Tupel $\varphi \in (\mathbb{Z}_R)^k \cong \big(\{0,1\}^R\big)^k$ erzeugt, durch geeignete neue Bandsymbole getrennt;

copyright: Jürgen Koslowski, TU Braunschweig, 19. Juli 2019

▷ Band B_3 dient zur Simulation des durch φ spezifizierten Anfangsstücks einer $\boldsymbol{M}$-Berechnung der Länge $\leq k$ mit Eingabe w; als Schrittzähler fungiert der String auf B_1.

Das Akzeptanzverhalten ist wie folgt spezifiziert:

(0) Erscheint im Laufe der durch φ bestimmten Simulation von M eine akzeptierende Konfiguration von M, so akzeptiert auch M'.

(1) Erscheint im Laufe der durch φ bestimmten Simulation von M eine abweisende oder keine akzeptierende Konfiguration von M, so wird die Simulation abgebrochen und auf B_2 das nächste Tupel erzeugt.

Existiert in einer Konfiguration kein Übergang mit der durch φ bestimmten Nummer, so kann die entsprechende φ-Komponente auf 0 gesetzt und die nachfolgende φ-Komponente um 1 erhöht werden.

(2) Wurden alle k-Tupel durchprobiert ohne dass M' akzeptiert hat, so wird der String auf B_1 um ein Symbol verlängert.

(3) Mit Hilfe einer besonderen Zustandskomponente wird registriert, ob alle bisherigen Simulationen der Tiefe k den abweisenden Zustand erreicht haben oder steckengeblieben sind. Ist das der Fall, so begibt sich M' in den abweisenden Zustand, denn weitere Simulationen von M größerer Länge können nicht mehr zum Erfolg führen.

Diese Arbeitsweise entspricht einer Breiten-Durchsuchung des Berechnungsbaums für die gegebene Eingabe. Existiert eine akzeptierende Berechnung, so wird sie nach endlich vielen Schritten gefunden. Andernfalls hält M' im abweisenden Zustand oder terminiert nicht. Falls M auf jede Eingabe terminiert, so gilt das wegen (3) auch für M'. □

Im Gegensatz zu früheren Simulationen ist die hier beschriebene Determinisierung einer nTM i.A. nicht effizient: Wir nehmen an, dass für jede Eingabe der Länge n die Tiefe des Berechnungsbaums durch eine Funktion $f(n)$ beschränkt ist. Bei maximalem Verzweigungsgrad r kann er bis zu $(f(n))^r$ Knoten haben. Dagegen fällt das Problem, dass nach jeder Vergrößerung der Tiefe viele der früher ausgeführten Berechnungsschritte erneut auszuführen sind, kaum ins Gewicht.

TM-Varianten mit einer endlichen Anzahl von Anfangs- bzw. Akkzeptanzzuständen werden offensichtlich nicht mächtiger sein können als die bisher betrachteten Modelle. Aber der externe Speicher einer TM könnte grundsätzlich anders organisiert werden, vergl. Bemerkung 1.0.09(1), wo der Stack eines PDA zu einem einseitig unendlichen Band mutiert. Dass auch in diesem Fall eine effiziente Simulation möglich ist, überlassen wir den HAn.

Wie steht es mit der Größe des Alphabets? Reale Computer arbeiten mit dem Binäralphabet. Dass dies keinen Einfluss auf ihre Mächtigkeit hat folgt aus der Tatsache, dass jedes Alphabet

copyright: Jürgen Koslowski, TU Braunschweig, 19. Juli 2019

Σ derart binär codieren läßt, etwa durch eine Injektion $\Sigma \xrightarrow{\kappa} \{0,1\}^*$, so dass die eindeutige Erweiterung zu einem Monoid-Homomorphismus $\Sigma^* \xrightarrow{\bar{\kappa}} \{0,1\}^*$ ebenfalls injektiv ist. Dazu genügt es, dass alle κ-Werte die gleiche Länge haben, etwa $k = \lceil \log_2 |\Sigma| \rceil$.

1.0.16 Satz. *Jede TM $M = \langle Q, \Sigma, \Gamma, \delta, q_0, q_{\mathrm{acc}}, (q_{\mathrm{rej}}) \rangle$ kann von einer TM M' über dem Binäralphabet simuliert werden.*

Beweis. (Skizze). Eine Codierung κ wie oben sei gegeben. Jeder Schritt der Maschine M erfordert die Kenntnis von k aufeinanderfolgenden Speicherzellen von M'. Die Zustandsmenge von M' möge die Form $R \times Q \times (\Gamma + \{?\})$ haben; die dritte Komponente gibt Auskunft darüber, ob der Speicherinhalt der aktuellen Zelle von M bekannt ist, und wenn ja, welchen Wert er hat.

Die effiziente(!) Simulation erfordert nun folgende Schritte:

- ▷ in einem Zustand der Form $\langle r, q, ? \rangle$ lese k Zellen und speichere das entsprechende Γ-Symbol im Zustand $\langle r, q, b \rangle$;
- ▷ Modifiziere die gerade gelesenen k Zellen gemäß δ und κ.
- ▷ Führe ggf. k Schritte zur Seite aus und ändere den Kontrollzustand von M.

Nach Konstruktion gilt nun $w \in \mathcal{L}(M) \subseteq \Sigma^*$ gdw. $\kappa(w) \in \mathcal{L}(M') \subseteq \{0,1\}^*$. □

1.0.4 Entscheidbarkeit und Abschlußeigenschaften

Damit wir mit Hilfe einer TM in der Praxis entscheiden können, ob ein Wort $w \in \Sigma^*$ zu $L \subseteq \Sigma^*$ gehört, brauchen wir eine definitive Antwort, ja oder nein. Dafür sind echt nichtdeterministische TMs ungeeignet, da Eingaben w mit mehreren Berechnungen existieren, die nicht notwendig zum gleichen Ergebnis führen müssen. Selbst wenn eine akzeptierenden Berechnung darunter ist, besteht keine Garantie, dass diese bei einem Testlauf ausgewählt wird (Würfeln). Erst ihre Determinisierung gemäß Satz 1.0.15 wird die entsprechende akzeptierende Haltekonfiguration garantiert in endlich vielen Schritten erreichen.

Allgemein können in deterministischen TMs unendliche nicht haltende Berechnungen möglich sein. Solange keine Haltekonfiguration erreicht wird, oder die Berechnung nicht stecken bleibt, ist i.A. nicht klar, ob dies in der Zukunft der aktuellen Berechnung doch noch geschehen kann.

1.0.17 Definition.

- ▷ Eine dTM heißt *Entscheider*, wenn jede ihrer Berechnungen auf eine endliche Eingabe "hält", d.h., eine Haltekonfiguration erreicht.
- ▷ Eine Sprache $L \subseteq \Sigma^*$ heißt *entscheidbar*, wenn ein Entscheider M mit $L = \mathcal{L}(M)$ existiert.

1.0.18 Bemerkungen.

copyright: Jürgen Koslowski, TU Braunschweig, 19. Juli 2019

(0) Jede kontextfreie Sprache ist entscheidbar, siehe TheoInf 1.

(1) Jede dTM kann als partielle Funktion von Σ^* nach $\mathbb{B} = \{0, 1\}$ aufgefasst werden, die im Falle der Akzeptanz 1 ausgibt, im abweisenden Fall oder nach Steckenbleiben 0, und im Fall einer nicht terminierenden Berechnung undefiniert ist. Die Entscheider sind genau diejenigen dTMs, die auf diese Weise eine totale Funktion induzieren. $\triangleleft$

Leider kann man es der Spezifikation einer dTM nicht ansehen, ob sie ein Entscheider ist. Das wäre harmlos, wenn jede semi-entscheidbare Sprache auch von einem Entscheider akzeptiert werden könnte, aber die Terminologie deutet an, dass dies nicht der Fall ist. Das wirft neue Probleme auf:

– Wie kann man festzustellen, ob eine dTM ein Entscheider ist? Dieses Problem stellt sich in der Tat als nicht algorithmisch lösbar und somit unentscheidbar heraus, siehe Unterabschnitt 1.2.3

– Gibt es eine semi-entscheidbare Sprache, die nicht entscheidbar ist? Beide Sprachklassen sind abzählbar unendlich, dies liefert also keine Handhabe zur Unterscheidung.

1.0.19 Satz. *Eine Sprache $L \subseteq \Sigma^*$ ist genau dann entscheidbar, wenn L und ihr Komplement $\bar{L} := \Sigma^* - L$ semi-entscheidbar sind.*

Beweis. Ist L entscheidbar, so existiert eine dTM $\boldsymbol{M}$ mit $\mathcal{L}(\boldsymbol{M}) = L$, die immer hält. Insbesondere ist L damit semi-entscheidbar.

Wir modifizieren M derart, dass nun $\bar{L} = \Sigma^* - L$ akzeptiert wird. Falls q_{rej} existiert, vertauschen wir die Rollen von q_{acc} und q_{rej} und fügen neue Übergänge $\langle p, b\rangle\,\bar{\delta}\,\langle q_{\text{rej}}, \langle b, N\rangle\rangle$ hinzu, falls $\langle p, b\rangle\delta = \emptyset$. Andernfalls fügen wir einen neuen akzeptierenden Zustand q_* ein, und dazu Übergänge $\langle p, b\rangle\,\bar{\delta}\,\langle q_*, \langle b, N\rangle\rangle$ falls $\langle p, b\rangle\delta = \emptyset$ oder $p = q_*$. Das liefert dann eine dTM $\boldsymbol{M'}$ mit $\mathcal{L}(\boldsymbol{M'}) = \bar{L}$.

Umgekehrt lassen wir dTMs $\boldsymbol{M}$ und $\bar{\boldsymbol{M}}$ mit $\mathcal{L}(\boldsymbol{M}) = L$ und $\mathcal{L}(\bar{\boldsymbol{M}}) = \bar{L}$ mittels Interleaving dieselbe Eingabe bearbeiten. Die resultierende Maschine $\boldsymbol{K}$ ist immer noch deterministisch und hält genau dann, wenn eine der Teilmaschinen hält, und sie möge genau dann akzeptieren, wenn $\boldsymbol{M}$ hält und akzeptiert, oder wenn $\bar{\boldsymbol{M}}$ hält und nicht akzeptiert. Dann hält $\boldsymbol{K}$ immer und erfüllt $\mathcal{L}(\boldsymbol{K}) = L$. $\square$

1.0.20 Satz. *Die Klasse der (semi-)entscheidbaren Sprachen ist abgeschlossen unter*

▷ *endlicher Vereinigung und endlichem Durchschnitt;*

▷ *Konkatenation und Iteration (Kleene Stern);*

▷ *Spiegelung und Shuffle;*

▷ *Residuierung bzgl. endlicher Sprachen.*

copyright: Jürgen Koslowski, TU Braunschweig, 19. Juli 2019

- ▷ *homomorphen Bildern und Urbildern.*

Darüberhinaus sind die entscheidbaren Sprachen unter Komplementbildung abgeschlossen, die semi-entscheidbaren Sprachen aber nicht.

Beweis. Grobe Idee, vergl. HA.

- ▷ Endliche Vereinigung und Kleene Stern: durch Konstruktion geeigneter Maschinen, vergl. HA.
- ▷ Endlicher Durchschnitt, Konkatenation, Shuffle: durch geeignete Abwandlung der Konstruktionsprinzipien der obigen Maschinen.
- ▷ Spiegelung, Residuierung mit einzelnen Wörtern, homomorphe Urbilder: Modifikation der Eingabe durch
  - Spiegelung derselben;
  - Anfügen eines Prä- bzw. Postfixes;
  - Anwendung des Homomorphismus auf die Eingabe.

  Die Residuierung mit endlichen Sprachen läßt sich als endlicher Durchschnitt ausdrücken.
- ▷ homomorphe Bilder: nichtdeterministische oder systematische Suche nach einem Urbild der Eingabe, Eingabe dieses Urbilds in die Maschine für die Ausgangssprache.

Der Abschluss der entscheidbaren Sprachen unter Komplementbildung folgt aus Satz 1.0.19, indem man die Rollen der beiden Maschinen in der Akzeptanzbedingung vertauscht. □

1.1 Berechenbarkeit

Das allgemeine Berechenbarkeistproblem besteht darin für eine partielle Funktion $A \xrightarrow{f} B$ festzustellen, ob für jedes $a \in A$ der Funktionswert $f(a)$ algorithmisch bestimmt werden kann, sofern er existiert. Andernfalls soll der Algorithmus mit einer Fehlermeldung oder gar nicht terminieren.

Hier wird *nicht* verlangt, dass der entsprechende Algorithmus jetzt schon bekannt ist, denn dann wäre die Antwort auf das Problem von der Zeit abhängig!

1.1.00 Beispiel.

- ▷ Die überall undefinierte partielle Funktion $\mathbb{N} \xrightarrow{\perp} \mathbb{N}$ ist berechenbar, und zwar durch einen Algorithmus, der nie terminiert, also etwa durch eine absichtliche Endlosschleife in einer Programmiersprache, oder durch eine TM mit einelementigem Alphabet (unäre Darstellung von $\mathbb{N}$), deren Kopf im nicht akzeptierenden Anfangszustand nur nach links wandert.

copyright: Jürgen Koslowski, TU Braunschweig, 19. Juli 2019

- ▷ Jede konstante Funktion ist berechenbar, selbst wenn wir den Funktionswert (noch) nicht kennen:
$$\mathbb{N} \xrightarrow{f} \{0,1\}^* \quad , \quad n \mapsto \begin{cases} 1 & \text{falls } \boldsymbol{P} = \boldsymbol{NP} \\ 0 & \text{sonst} \end{cases}$$ ◁

Auch in diesem Abschnitt wollen wir uns auf den Fall beschränken, dass A und B freie Monoide über endlichen Mengen sind, also nur den Fall $\Sigma^* \xrightarrow{f} \Delta^*$ betrachten.

Historisch gibt es diverse Versuche, den Begriff der Berechenbarkeit zu formalisieren, z.B.

- ▷ primitiv rekursive und μ-rekursive Funktionen (Kurt Gödel 1965, Jacques Herbrand);
- ▷ den λ-Kalkül (Alonzo Church 1933, Stephen C. Kleene 1935);
- ▷ kombinatorische Logik (Moses Schönfinkel 1924, Haskell B. Curry 1929).

Das Analogon zur Church-Turing-These nimmt an, dass sich all diese Berechnungsmodelle mit Turing-Maschinen simulieren lassen.

Wie in Bemerkung 1.0.18(1) angedeutet, realisieren dTMs wie oben eingeführt zumindest partiell die charakteristischen Funktionen der von ihnen akzeptierten Sprachen (auf dem Komplement müssen sie nicht vollständig definiert sein, das gilt nur für Entscheider). Die Ausgabe liegt in $\{0,1\}$ und damit auch in $\{0,1\}^*$. Das legt die Spezifikation eines Ausgabealphabets Δ nahe, um Ausgaben in Δ^* zu erhalten. Fragt sich, in welcher Form die von der TM bereitgestellt werden sollen. Im Gegensatz zu Moore- und Mealy-Automaten und im Hinblick auf die Eingabe, die auch zunächst auf dem Band stand, soll die Ausgabe ebenfalls auf dem Band stehen, was $\Delta + \{\sqcup\} \subseteq \Gamma$ erfordert, wie beim Eingabealphabet. Hier wollen wir ein spezielles Ausgabeband dafür vorsehen. Damit endet schon die Symmetrie zur Eingabe. Manche Autoren betrachten stattdessen das längste Wort aus Δ^*, das etwa rechts der Kopfposition auf dem Band steht (Wätjen).

Im Hinblick auf den Speicherplatzbedarf, der uns im Abschnitt zur Komplexitätstheorie interessieren wird, wollen wir zwei technische Zusatzannahmen treffen, die garantieren, dass der Platz, den Eingabe und Ausgabe einnehmen, nicht mitgezählt wird, wenn es um den Platzbedarf der eigentlichen Rechnung geht (hier macht sich der Unterschied zu EAs und PDAs bemerkbar, deren Eingabe von einer externen Quelle kam). Andernfalls kann es keinen sub-linearen Platzbedarf geben, was für eine Funktion, die Eingaben auf Leerheit prüft, durchaus Sinn ergibt: man muß nicht die ganze Eingabe kennen um festzustellen, ob sie leer ist.

- ▷ Das Eingabe-Band B_0 ist **read-only**, kann also nicht überschrieben aber wiederholt inspiziert werden;
- ▷ das Ausgabeband B_k ist **one-way** oder **write-only**, kann also nur in einer Richtung beschrieben werden, und der Anfang der Ausgabe kann nicht mehr inspiziert werden.

copyright: Jürgen Koslowski, TU Braunschweig, 19. Juli 2019

Diese Bedingungen sind nicht symmetrisch hinsichtlich Ein-und Ausgabe: Ein **one-way** Ausgabe-Band ist nur deshalb **write-only**, weil das Band zu Beginn leer ist. Ein **one-way** Eingabe-Band (in der Richtung, in der die Eingabe erwartet wird) braucht nicht **read-only** zu sein; man kann darauf schreiben, aber die geschriebenen Symbole sind für den Fortgang der Berechnung irrelevant. Damit entspricht ein **one-way** Eingabe-Band eher der Art und Weise, wie EAs und PDAs ihre Eingabe verarbeiten.

1.1.01 Definition. Eine partielle Funktion $\Sigma^* \xrightarrow{f} \Delta^*$ heißt *Turing-berechenbar*, sofern eine dTM existiert, die obige Einschränkungen erfüllt und auf jeder Eingabe $w \in \Sigma^*$

- ▷ nicht hält (was Steckenbleiben beinhaltet) oder sie abweist, sofern $f(w)$ undefiniert ist;
- ▷ andernfalls nach endlich vielen Schritten akzeptiert mit dem Wert $f(w)$ auf dem ansonsten leeren letzten Band.

Gelegentlich hilft es, bei partiellen Funktionen $A \xrightarrow{f} B$ zwischen abzählbaren Mengen zu *Darstellungen* der Elemente von A bzw. B als Strings über geeigneten Alphabeten Σ und Δ überzugehen, sofern das effizient möglich ist.

1.1.02 Beispiel. Die natürlichen Zahlen $\mathbb{N} = \{0, 1, 2, \dots\}$ formen zwar selber ein freies Monoid, $\mathbb{N} = \{1\}^*$, aber diese *unäre* Darstellung ist oft unangemessen, wenn es um Berechenbarkeitsfragen geht, etwa von Funktionen $\mathbb{N} \xrightarrow{f} \mathbb{N}$.

(0) Aufgrund der leicht berechenbaren Binärdarstellung $\{1\}^* \cong \mathbb{N} \xrightarrow{\boldsymbol{bin}} \{0,1\}^*$, die injektiv aber nicht surjektiv ist (führende Nullen!), läßt sich die Berechenbarkeit von f auf die der folgenden partiellen Funktion zurückführen:

$$\{0,1\}^* \xrightarrow{f_{\boldsymbol{bin}}} \{0,1\}^* \quad , \quad \boldsymbol{bin}(n) \mapsto \begin{cases} \bot & \text{falls } fn \text{ undefiniert ist} \\ \boldsymbol{bin}(f(n)) & \text{sonst} \end{cases}$$

Auf Binärstrings, die keine Binärdarstellungen natürlicher Zahlen sind, ist $f_{\boldsymbol{bin}}$ ebenfalls undefiniert.

(1) Betracht die Funktion

$$\mathbb{N} \xrightarrow{f_\pi} \{0,1\} \quad , \quad n \mapsto \begin{cases} 1 & \text{falls } n \text{ Präfix der Dezimaldarstellung von } \pi \text{ ist} \\ 0 & \text{sonst} \end{cases}$$

Diese Funktion ist berechenbar (und damit entscheidbar), weil es Algorithmen gibt, die π auf beliebig viele Dezimalstellen genau bestimmen können.

(2) Wie steht es mit der Berechenbarkeit der analog zu f_π definierten Funktion f_a für eine beliebige positive reelle Zahl a? Da es überabzählbar viele reelle Zahlen gibt, aber nur

copyright: Jürgen Koslowski, TU Braunschweig, 19. Juli 2019

abzählbar viele Berechnungsalgorithmen, können nicht alle dieser Funktionen berechenbar sein. In der Tat sind nur abzählbar viele dieser Funktionen berechenbar. Leider ist es schwierig, eine konkrete reelle Zahl a anzugeben, für die f_a nicht berechenbar ist. (Eigentlich ist dies ein Beispiel dafür, dass bestimmte Teilmengen von $\mathbb{N}$ nicht entscheidbar sind!)

(3) Im Fall einer unstrukturierten abzählbaren Menge A ist es nicht möglich, eine berechenbare Darstellung $A \xrightarrow{d} \Sigma^*$ mit einem endlichen Alphabet Σ anzugeben.

1.1.03 Satz. *Für endliche nichtleere Alphabete Σ und Δ gibt es nicht-berechenbare partielle Funktionen $\Sigma^* \xrightarrow{f} \Delta^*$.*

Beweis.

Wir betrachten eine Aufzählung (= surjektive Funktion mit Domain $\mathbb{N}$, vergl. frühe Logik-HA) M_i , $i \in \mathbb{N}$ aller TMs mit Ein- und Ausgabealphabet Σ bzw. Δ, die den obigen Einschränkungen genügen. Die zugehörigen partiellen Funktionen mögen m_i , $i \in I$ heißen. Weiter sei s_i , $i \in \mathbb{N}$, eine Aufzählung der Elemente von Σ^* , z.B. lexikographisch bzgl. einer beliebigen linearen Ordnung auf Σ.

Annahme: Jede partielle Funktion $\Sigma^* \xrightarrow{f} \Delta^*$ hat die Form m_i für ein $i \in \mathbb{N}$.

Wähle $d \in \Delta \neq \emptyset$ und konstruiere die totale Funktion $\Sigma^* \xrightarrow{g} \Delta^*$ wie folgt:

$$f(w_i) = \begin{cases} \varepsilon & \text{, falls } m_i(w_i) \text{ undefiniert ist} \\ d \cdot m_i(w_i) & \text{, sonst} \end{cases}$$

Behauptung: f stimmt mit keiner der partiellen Funktionen m_i , $i \in \mathbb{N}$, überein.

Falls $f = m_k$, muß m_k total sein. Aber an der Stelle w_k ist der Funktionswert von f um eins länger als der von m_k , im Widerspruch zu unserer Annahme. □

Diese Art der *Diagonalisierung* werden wir noch mehrfach verwenden.

Wir können nun den alternativen Namen *rekursiv aufzählbar* für semi-entscheidbare Sprachen rechtfertigen:

1.1.04 Satz. *Eine nichtleere Sprache $L \subseteq \Sigma^*$ ist semi-entscheidbar, wenn es eine berechenbare Funktion $\mathbb{N} \xrightarrow{f} \Sigma^*$ mit Bild L gibt.*

Beweis.

($\Leftarrow$) Ein Algorithmus, der $w \in L$ semi-entscheidet, berechnet der Reihe nach die Werte $f(i)$, $i \in \mathbb{N}$, in jeweils endlich vielen Schritten und vergleicht sie mit w. Bei Wörtern $w \in \Sigma^* - L$ terminiert er nicht. (Dies funktioniert auch für $L = \emptyset$.)

copyright: Jürgen Koslowski, TU Braunschweig, 19. Juli 2019

(⇒) Betrachte eine TM M mit $\mathcal{L}(M) = L \neq \emptyset$ und eine bijektive Aufzählung s_i, $i < n$, der Symbole aus Σ. Dies induziert eine lexicographische Ordnung auf Σ^*.

Nun untersuchen wir, iterativ für jede Berechnungstiefe $i \in \mathbb{N}$, alle Wörter aus Σ^* der Länge $\leq i$ gemäß ihrer lexicographischen Ordnung, ob sie mit höchstens i Berechnungsschritten von M akzeptiert werden, und geben diejenigen Wörter aus, für die das funktioniert. Dass hierbei Wiederholungen auftreten ist kein Problem. Die Reihenfolge der Ausgaben bestimmt die gesuchte Funktion.

Um sie zu implementieren, muß die simulierende TM noch mit einem Ausgabezähler versehen werden, der bei Eingabe von $n \in \mathbb{N}$ die Simulation nach $n+1$ Ausgaben stoppt und die letzte Ausgabe der Simulation als Ergebnis liefert. □

Wir können nun den schon oben angedeuteten Zusammenhang zwischen Entscheidbarkeit und Berechenbarkeit konkretisieren:

1.1.05 Satz. *Gegeben sei eine Sprache $L \subseteq \Sigma^*$.*

(0) ***L*** *ist genau dann semi-entscheidbar, wenn ihre* partielle charakteristische Funktion

$$\Sigma^* \xrightarrow{\dot{\chi}} \{0,1\} \quad , \quad w \mapsto \begin{cases} 1 & \text{if } w \in L; \\ \bot & \text{if } w \notin L. \end{cases}$$

berechenbar ist.

(1) ***L*** *ist genau dann entscheidbar, wenn ihre* charakteristische Funktion

$$\Sigma^* \xrightarrow{\chi} \{0,1\} \quad , \quad w \mapsto \begin{cases} 1 & \text{if } w \in L; \\ 0 & \text{if } w \notin L. \end{cases}$$

berechenbar ist. □

Wir fassen zusammen:

1.1.06 Corollar. *Für eine beliebige Sprache $L \subseteq \Sigma^*$ sind folgende Aussagen äquivalent*

(0) *L ist semi-entscheidbar (auch bekannt als rekursiv aufzählbar oder Turing-erkennbar);*

(1) *L wird von einer TM akzeptiert;*

(2) *L ist das Bild einer totalen berechenbaren Funktion $\mathbb{N} \xrightarrow{f} \Sigma^*$;*

(3) *die partielle charakteristische Funktion $\dot{\chi}_L$ von L ist berechenbar;*

(4) *L ist der Definitionsbereich einer partiellen berechenbaren Funktion $\Sigma^* \xrightarrow{g} \Delta^*$.* □

copyright: Jürgen Koslowski, TU Braunschweig, 19. Juli 2019

1.1.07 Bemerkung. In der Logik-VL wurde die *Abzählbarkeit* einer Menge X mittels der Existenz einer injektiven Funktion $X \xrightarrow{\varphi} \mathbb{N}$ definiert. Es wurde weiterhin nachgewiesen, dass diese Bedingung äquivalent dazu ist, dass X entweder leer ist, oder eine surjektive Abbildung $\mathbb{N} \xrightarrow{\psi} X$ zulässt; letztere wird auch als *Aufzählung* von X bezeichnet. Da ψ nicht injektiv zu sein braucht, können sich Elemente von X dabei wiederholen.

Da für endliches Alphabet Σ die Menge Σ^* und somit jede ihrer Teilmengen wieder abzählbar ist (Verknüpfung mit der entsprechenden injektiven Inklusionsfunktion), besitzt jede nichtleere Sprache eine solche Aufzählung, Rein technisch könnte man sie „aufzählbar“ nennen, was wir aber lieber vermeiden wollen. Der Unterschied zu nichtleeren *rekursiv aufzählbaren* Sprachen besteht darin, dass für letztere die Existenz einer *berechenbaren* Aufzählung verlangt wird.

Achtung: die rekursive Aufzählbarkeit vererbt sich im Gegensatz zur Abzählbarkeit nicht notwendig auf nichtleere Teilmengen! (Warum nicht?)

Wir schließen diesen Abschnitt mit einem recht bekannten konkreten Beispiel einer nicht berechenbaren Funktion.

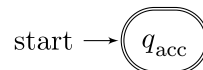
1.1.08 Definition. Die sog. *Busy Beaver* Funktion $\mathbb{N} \xrightarrow{\beta} \mathbb{N}$ bildet $n \in \mathbb{N}$ ab auf

▷ die größte Zahl $\beta(n)$ nichtleerer Felder, die eine 1-Band dTM über einem Singleton-Alphabet $\{\star\}$, mit n Nicht-Haltezuständen und einen Haltezustand q_{acc} ausgehend vom leeren Band auf diesem hinterlassen kann, wenn sie hält (d.h., einen Haltezustand erreicht).

Achtung: Nach dem Halt der Maschine dürfen zwischen den Symbolen $\star$ Lücken auftreten. Damit kann sich die größte unär codierte Zahl k, die nach dem Halt auf dem Band steht, durchaus von $\beta(n)$ unterscheiden.

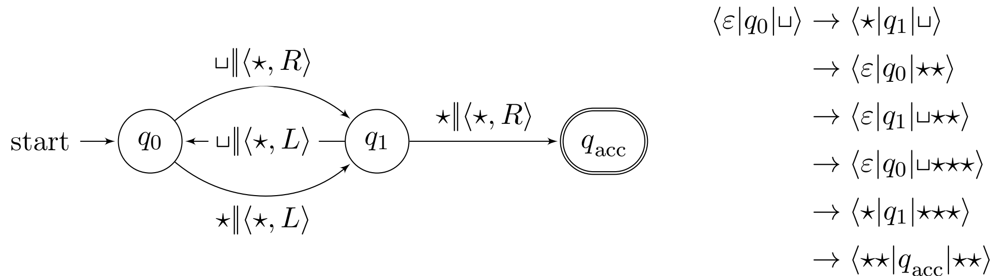
1.1.09 Beispiele. Wir verzichten hier auf die Leerlauf-Schleifen bei q_{acc}. Alles Übrige kann ohne die neutrale Bewegung N bewerkstelligt werden.

$\beta(0) = 0$:



$\beta(1) = 1$:

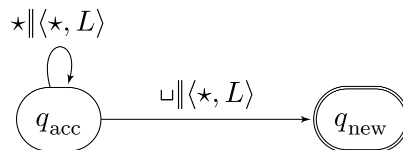
$\beta(2) = 4$:



$\beta(3)$ sollten Sie selber bestimmen.

1.1.10 Proposition. *Die Busy Beaver Funktion wächst streng monoton, d.h.,* $\beta(n) < \beta(n+1)$ *für jedes* $n \in \mathbb{N}$.

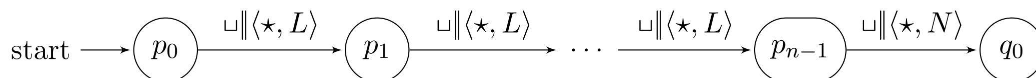
Beweis. Die dTM $\boldsymbol{M}$ mit Akzeptanz-Zustand q_{acc} möge $\beta(n)$ realisieren. Füge einen neuen Akzeptanz-Zustand q_{new} hinzu und die Übergänge



Die resultierende Maschine $\boldsymbol{M'}$, mit q_{acc} als Nicht-Haltezustand, sucht nach Erreichen von q_{acc} links ein freies Feld, schreibt $\star$ und hält. Also gilt $\beta(n) < \beta(n) + 1 \leq \beta(n+1)$. □

1.1.11 Proposition. *Für jede Turing-berechenbare totale Funktion* $\mathbb{N} \xrightarrow{f} \mathbb{N}$ *existiert eine Zahl* r_f *, so dass für alle* $n \in \mathbb{N}$ *gilt* $f(n) \leq \beta(n + r_f)$.

Beweis. Die Funktion f möge von der dTM $\boldsymbol{M}$ mit r_f Zuständen und unärer Codierung berechnet werden. Für $n \in \mathbb{N}$ fügen wir n neue Zustände p_i , $i < n$, zu $\boldsymbol{M}$ hinzu, sowie folgende Übergänge:



$\boldsymbol{M}_n$ produziert ausgehend vom leeren Band n Sterne und führt dann $\boldsymbol{M}$ mit dieser unären Eingabe aus. Mit $\boldsymbol{M}$ hält auch $\boldsymbol{M}_n$, und dann stehen mindestens $f(n)$ Sterne auf dem Band, also folgt $f(n) \leq \beta(n + r_f)$. □

copyright: Jürgen Koslowski, TU Braunschweig, 19. Juli 2019

1.1.12 Satz. *β ist nicht Turing-berechenbar.*

Beweis. Wir nehmen an, β ist Turing-berechenbar vermöge der dTM $\boldsymbol{M}$. Dann ist auch $\mathbb{N} \xrightarrow{f} \mathbb{N}$ mit $n \mapsto \beta(2n)$ Turing-berechenbar (Verknüpfung mit einer einfachen Maschine, die die Eingabe verdoppelt). Für jedes $n \in \mathbb{N}$ gilt nach obigem Satz $\beta(2n) = f(n) \leq \beta(n + r_f)$. Aber $n = r_f + 1$ liefert einen Widerspruch zur strengen Monotonie von β. □

1.2 Unentscheidbarkeit

Wir wollen uns nun konkreten Beispielen nicht (semi-)entscheidbarer Sprachen zuwenden. Dafür benötigen wir eine Reihe neuer Konzepte:

(a) Um Entscheidungsprobleme hinsichtlich TMs als Wortprobleme behandeln zu können, benötigen wir eine Codierung von TMs. (Selbstreferenz, Gödel)

(b) Wenn eine TM den Code einer weiteren TM als Eingabe verarbeiten soll um etwas über das Verhalten der Eingabe-Maschine herauszufinden, liegt es nahe, die Eingabe-TM zu simulieren. Dass dies möglich ist, folgt aus der Existenz einer sog. „universellen TM“.

(c) Unter Verwendung des schon bekannten Diangonalisierungstricks wird es dann möglich sein, bestimmte „Halteprobleme“, die danach fragen ob eine bestimmte TM bei einer bestimmten Eingabe hält (was das Steckenbleiben einschließt) oder nicht, als unentscheidbar nachzuweisen.

(d) Die Unentscheidbarkeit weiterer Probleme braucht nun nicht mehr "from scratch" beweisen zu werden, sondern läßt sich auf die Unentscheidbarkeit des Halteproblems „reduzieren“. Diese Vorgehensweise ist von größer konzeptioneller Wichtigkeit und wird uns in der Komplexitätstheorie wieder begegnen.

1.2.1 Codierung von Turing-Maschinen

Wir wollen eine dTM $M = \langle Q, \Sigma, \Gamma, \delta, q_0, q_{\text{acc}}, q_{\text{rej}} \rangle$ letztendlich binär codieren, verwenden zunächst aber noch ein Trennsymbol $\#$, also das Codierungsalphabet $\{0, 1, \#\}$.

Ohne Beschränkung der Allgemeinheit (ObdA) gehen wir davon aus, dass Zustandsmenge und Bandalphabet durchnummeriert sind:

$$Q = \{ q_i : i < n \} \quad \text{und} \quad \Gamma = \{ a_j : j < m \}$$

Die festgelegten Symbole mögen alle zu Beginn der Nummerierung auftreten, und die ersten k nicht-Blank-Symbole mögen das Eingabealphabet Σ ausmachen, d.h.,

$$q_1 = q_{\text{acc}} \quad , \quad q_2 = q_{\text{rej}} \quad \text{sowie} \quad a_0 = \sqcup \quad \text{und} \quad \Sigma = \{ a_i : 1 \leq i \leq k \}$$

Indem wir einen Zustand q_{rej} zulassen, können und wollen wir das Steckenbleiben bei Berechnungen vermeiden. Zu verlangen, dass die Zustände q_0, q_{acc} und a_{rej} paarweise verschieden sind, stellt keine echte Einschränkung dar, denn dTMs, die dieser Bedingung nicht genügen, lassen sich leicht entsprechend umbauen, auf Kosten einer festen Anzahl von Berechnungsschritten.

Den möglichen Kopfbewegungen L, R und N ordnen wir die Zahlen 0, 1 und 2 zu.

Mittels der aus Beispiel 1.1.02(0) bekannten Binärdarstellungsfunktion $\mathbb{N} \xrightarrow{\text{bin}} \{0.1\}^*$ soll die Zuweisung

$$\delta\langle q_i, a_j\rangle = \langle q_{i'}, a_{j'}, d\rangle$$

zunächst dargestellt werden durch das Wort

$$w_{i,j,i',j',d} := \#\#\,\text{bin}(i)\#\,\text{bin}(j)\#\,\text{bin}(i')\#\,\text{bin}(j')\#\,\text{bin}(d)$$

1.2.00 Definition. Aus der Spezifikation von M extrahieren wir zunächst die Anzahlen n der Zustände, k der Eingabesymbole und $m > k$ der Bandsymbole. Die Konkatenation von

$$\#\#\#\,\text{bin}(n)\#\,\text{bin}(k)\#\,\text{bin}(m) \quad \text{mit} \quad \bigodot_{i<n} \bigodot_{j<m} w_{i,j,i',j',d}$$

liefert eine ternäre Codierung von M. Der durch $0 \mapsto 00$, $1 \mapsto 01$ und $\# \mapsto 11$ bestimmte injektive Homomorphismus $\{0, 1, \#\}^* \longrightarrow \{0, 1\}^*$ liefert schließlich die gewünschte binäre Codierung $\langle M\rangle$.

1.2.01 Bemerkungen.

(0) Wegen $\Sigma \subseteq \Gamma$ liefert die ternäre Codierung von Eingaben

$$\{1, 2, \ldots, k\}^t \ni w \mapsto \bigodot_{\ell<t} \#\,\text{bin}(x_\ell)$$

nach Anwendung des obigen Homomorphismus eine binäre Codierung $\langle x\rangle$ derselben. Aber es ist sinnvoller, für jedes Eingabesymbol $\lceil \log_2 k + 1\rceil$ Bits zu verwenden, damit bei der Simulation immer genug Platz zum schreiben des Codes eines Σ-Symbols vorhanden ist!

(1) Die oben angegebene Codierung ist natürlich nicht surjektiv, besitzt also nur eine partielle Rechtsinverse. Um später Fallunterscheidungen zu vermeiden(?), wollen wir diese partielle Funktion zu einer totalen Funktion aufwerten, indem wir allen Nicht-Codes eine feste dTM $M_\emptyset$ zuordnen, die die leere Sprache akzeptiert.

(2) Wir haben hier nur 1-Band dTMs betrachtet. Während für Mehrband-Maschinen die Simulation durch 1-Band Maschinen effizient ist, gilt das für die Determinisierung von nTMs nicht. Insofern ist eine eigenständige Codierung von nTMs anzustreben.

copyright: Jürgen Koslowski, TU Braunschweig, 19. Juli 2019

1.2.2 Die universelle Turing-Maschine

Wir wollen das Akzeptanzproblem für dTMs untersuchen:

Akzeptanzproblem (ACCEPT)
Gegeben: eine dTM M und eine Eingabe w
zu entscheiden: ob M die Eingabe w akzeptiert, d.h., ob $w \in \mathcal{L}(M)$ gilt.

Als Wortproblem über $\{0, 1, \#\}$ formuliert erhalten wir

$$\text{ACCEPT} = \{\, w\#x \,:\, w, x \in \{0,1\}^* \text{ und } x \in \mathcal{L}(M_w) \,\}$$

Die Idee, dieses Problem mit einer *universellen* TM zu lösen, die die Maschine M_w als "1Programm“ mit der Eingabe x simuliert, geht auf Allan Turing zurück (1936). Mit John von Neumanns Arbeit von 1945 liegt hier die Basis für moderne Computer.

1.2.02 Satz. *Es gibt eine sog. „universelle TM“ U, die das Akzeptanzproblem semi-entscheidet.*

Beweis. (Skizze) Man verwendet eine 4-Band-Maschine, mit der Eingabe auf dem read-only Band B_0. Der erste Teil w der Eingabe wird ternär decodiert und auf Band B_1 geschrieben, das Programm-Band. Falls es sich dabei um $M_\emptyset$ handelt, weist U die Eingabe gleich ab.

Anschließend wird die Eingabe x auf Band B_2 decodiert, und schließlich Band B_2 mit dem Ternär-Code von q_0 initialisiert.

Aufgrund des ersten Σ-Symbol-Codes der Eingabe und q_o wird auf Band 2 nach einer passenden Übergangsregel gesucht; es kann höchstens eine geben, Im Erfolgsfall wird ein Simulationsschritt ausgeführt und die Bänder B_2 und B_3 entsprechend verändert, andernfalls wird die Eingabe abgewiesen. □

1.2.3 Das Halteproblem

Wir wenden uns nun dem berühmten Halteproblem für dTMs zu:

(Allgemeines) Halteproblem (HP)
Gegeben: eine dTM M und eine Eingabe w
zu entscheiden: ob M mit der Eingabe w hält.

Das Wortproblem formuliert also

$$\text{HP} = \{\, w\#x \,:\, w, x \in \{0,1\}^* \text{ und } M_w \text{ hält bei Eingabe } x \,\}$$

1.2.03 Satz. (Turing, 1936) HP *ist semi-entscheidbar, aber nicht entscheidbar.*

Beweis. Der Beweis des ersten Teils verwendet U zur Simulation von M_w auf x; diese Simulation hält genau dann, wenn M_w auf x hält.

Für den interessanteren Teil verwendet man eine Variante des Diagonalisierungstricks. Annahme: HP würde durch H entscheiden. Definiere D mit Hilfe von H, die immer hält.

copyright: Jürgen Koslowski, TU Braunschweig, 19. Juli 2019

- ▷ D erhält nur eine Eingabe w;
- ▷ sie schreibt dahinter ein $\#$ und eine Kopie von w, um dann zum linken Rand zurückzukehren;
- ▷ nun läuft sie wie H mit der Eingabe $w\#w$, bis ein Haltezustand von H erreicht wird;
- ▷ wenn H den Zustand $_{\mathrm{acc}}$ erreicht, läuft D unendlich in einem Nicht-Haltezustand weiter;
- ▷ wenn H den Zustand q_{rej} erreicht, akzeptiert D.

Nun betrachten wir das Verhalten von D auf die Eingabe $\langle D\rangle$:

Fall 0: D hält auf $\langle D\rangle$. Dann weist H die Eingabe $\langle D\rangle\#\langle D\rangle$ ab, und folglich hält D nicht auf $\langle D\rangle$; Widerspruch.

Fall 1: D hält nicht auf $\langle D\rangle$. Dann akzeptiert H die Eingabe $\langle D\rangle\#\langle D\rangle$, und folglich hält D auf $\langle D\rangle$; Widerspruch.

Also existiert keine derartige Maschine H. □

Der obige Beweis zeigt sogar die Unentscheidbarkeit des

Speziellen Halteproblems (SHP)
Gegeben: eine dTM M
zu entscheiden: ob M mit der Eingabe $\langle M\rangle$ hält.

Also rührt die Unentscheidbarkeit des Halteproblems nicht daher, dass wir beliebige Eingaben zulassen.

Die Unentscheidbarkeit des Akzeptanzproblems sollte mit einer Modifikation des obigen Beweises gelingen, HA.

Aufgrund von Satz 1.0.19 liefern alle drei bisher als semi-entscheidbar aber nicht entscheidbar nachgewiesenen Probleme sogar nicht semi-entscheidbare Probleme:

1.2.04 Satz. *Die Komplemente* CO-ACCEPT, CO-HP *und* CO-SHP *von* ACCEPT, HP *bzw.* SHP *sind nicht semi-entscheidbar.* □

Zum Abschluss formulieren wir CO-ACCEPT als Wortproblem:

$$\textsc{co-accept} = \{\, \mathrm{w} : \mathrm{w} \notin \mathcal{L}(\mathrm{M_w})\,\}$$

1.2.4 Reduktionen

Um nicht für jedes neue unentscheidbare Problem einen eigenen Diagonalisierungsbeweis führen zu müssen, versucht man, neue Probleme auf bereits bekannte Probleme zurückzuführen. Dadurch erhält man eine Quasi-Ordnung (= reflexive und transitive Relation) auf der Klasse aller Probleme. Die gleiche Technik wird sich auch in der Komplexitätstheorie als nützlich erweisen.

copyright: Jürgen Koslowski, TU Braunschweig, 19. Juli 2019

1.2.05 Definition. Sind $K \subseteq \Sigma^*$ und $L \subseteq \Delta^*$ Sprachen, so heißt eine berechenbare Funktion $\Sigma^* \xrightarrow{f} \Delta^*$ *(many-one-)Reduktion* von K auf L, falls für alle $w \in K$ gilt

$$w \in K \quad \text{gdw.} \quad f(w) \in L$$

Weiter heißt K *(many-one-)reduzierbar* auf L, wenn eine derartige (many-one-)Reduktion existiert. Schreibweise $K \leq KL$.

Die Qualifikation "many-one" deutet an, dass die Funktion f n nicht injektiv zu sein braucht. In der Literatur treten noch andere Sorten von Reduktionen auf, die wir hier nicht verwenden werden.

1.2.06 Lemma. *Die Relation $\leq$ auf Problemen ist reflexiv und transitiv, also eine Quasi-Ordnung..*

Beweis. Klar, weil Identitätsfunktionen ebenso wie die Komposition berechenbarer Funktionen wieder berechenbar ist. □

1.2.07 Lemma. *Die (semi-)entscheidbaren Probleme formen einen* unteren Abschnitt *bzgl. $\leq$, d.h., falls L (semi-)entscheidbar und $K \leq L$, dann ist auch K (semi-)entscheidbar.*

Beweis. Die Funktion f werde von M_f berechnet, und M_l sei ein (Semi-)Entscheider für L. Wir konstruieren einen (Semi-)Entscheider für K, indem wir die Eingabe w zunächst mittels M_f in $f(w)$ umwandeln, und dieses als Eingabe von M_l verwenden. Im Wesentlichen handelt es sich also um die „Komposition" von M_f mit M_l. □

Typischerweise wird die Kontraposition dieses Ergebnisses angewendet:

1.2.08 Corollar. *Die nicht (semi-)entscheidbaren Probleme formen einen* oberen Abschnitt *bzgl. $\leq$, d.h., falls K nicht (semi-)entscheidbar ist und $K \leq L$, dann ist auch L nicht (semi-)entscheidbar.* □

1.2.09 Bemerkung. Betrachtet man anstelle von E-Problemen die B-Probleme der entsprechenden charakteristischen Funktionen, dann sind die obigen Ergebnisse Ausdruck der Tatsache, dass (partielle) berechenbare Funktionen unter Komposition abgeschlossen sind. ◁

1.2.10 Beispiel. Um das Problem

Halten bei leerer Eingabe (HP_ε) **Gegeben:** eine dTM M. **zu entscheiden:** ob M mit der Eingabe ε hält.

einordnen zu können, weisen wir $\text{HP} \leq \text{HP}_\varepsilon$ nach.

Einer Eingabe $w\#x$ für das allgemeine Halteproblem ist der Code $\langle M_w^x \rangle$ einer Maschine M_W^x zuzuordnen, die wie folgt spezifiziert ist:

▷ M_w^x löscht zunächst das Eingabeband mit $\sqcup$-Zeichen;

▷ schreibt dann das Wort x auf das leere Eingabeband, und bewegt sich zum ersten Symbol;

▷ simuliert M_w mit Code $\langle w \rangle$ auf dieser Eingabe.

Man überlegt sich leicht, dass $w\#x \mapsto \langle M_w^x \rangle$ tatsächlich berechenbar ist:

▷ die Maschine M_w ist mit einem deterministischen Präprozessor zu versehen;

▷ dieser benötigt $|x| + 2$ viele Zustände: einen neuen Anfangszustand, der die Eingabe löscht; je einen Zustand zum Schreiben der Symbole in x, und einen Zustand zur Suche des linken Randes; anschließender Übergang in den Anfangszustand von M_w;

▷ die neue Maschine möge genau dann halten, wenn M_w das tut. ◁

Wir hatten oben erwähnt, dass man die Unentscheidbarkeit von ACCEPT ähnlich wie die von HP nachweisen kann. Aber nun sind wir in der Lage, diese Ergebnis mit Hilfe einer Reduktion zu zeigen.

1.2.11 Satz. *Folgende Probleme sind semi-entscheidbar, aber nicht entscheidbar:*

(a) *Das* Allgemeine Akzeptanzproblem

$$\text{ACCEPT} = \{\, w\#x \in \{0,1,\#\}^* \;:\; w, x \in \{0,1\}^* \,\}$$

(b) *Das* Selbstakzeptanzprobmen, *auch* spezielles Akzeptanzproblem *genannt*

$$\text{SELF-ACCEPT} = \{\, w \in \{0,1\}^* \;:\; w \in \mathcal{L}(M_w) \,\}$$

(c) *Das* ε-Akzeptanzproblem

$$\text{ACCEPT}_\varepsilon = \{\, w \in \{0,1\}^* \;:\; \varepsilon \in \mathcal{L}(M_w) \,\}$$

Beweis. Als Sprache der universellen Turing-Maschine U ist ACCEPT semi-entscheidbar.

Wegen Lemma 1.2.07 folgt die Semi-Entscheidbarkeit von SELF-ACCEPT und $\text{ACCEPT}_\varepsilon$, wenn wir diese Probleme unterhalb von ACCEPT verorten können: die Reduktion wandelt eine Instanz w von SELF-ACCEPT oder $\text{ACCEPT}_\varepsilon$ in die Instanz $w\#w$ bzw. $w\#$ von ACCEPT um.

Zum Beweis der Nichtentscheidbarkeit kann man gemäß Korollar 1.2.08 die als nicht entscheidbar bekannten Versionen des Halteproblems auf die passenden Varianten des Akzeptanzproblems reduzieren. Wir beschränken uns hier auf $\text{HP} \leq \text{ACCEPT}$:

Für eine Instanz $w\#x$ des Halteproblems suchen wir eine Instanz $w'\#x'$ des Akzeptanzproblems mit

$$w\#x \in \text{HP} \quad \text{gdw.} \quad w'\#x' \in \text{ACCEPT}$$

copyright: Jürgen Koslowski, TU Braunschweig, 19. Juli 2019

Es dürfte am einfachsten sein, nur die TM M_w derart zu $M_{w'}$ abzuwandeln, so $M_{w'}$ immer akzeptiert, wenn M_w hält. Dann kann die Eingabe x unverändert bleiben, d.h., $x' = x$. Die Maschine $M_{w'}$ möge

– sich so verhalten wir M_w, solange kein Haltezustand erreicht ist;

– akzeptieren, wenn M_w akzeptiert;

– akzeptieren, wenn M_w abweist.

Man überlegt sich wieder leicht, dass $w \mapsto w'$ in der Tat berechenbar ist: Man ersetzt in der Spezifikation von M_w den Zustand q_{rej} durch q_{acc}, und fügt ggf. Übergänge nach q_{acc} hinzu, wenn M_w sonst steckenbleiben würde. □

1.3 Weitere unentscheidbare Probleme

Die bisherigen Reduktionen waren recht einfach. Im Fall des sog. Post'schen Korrepondenzproblems wird die Reduktion wesentlich aufwändiger sein. Das liegt auch daran, dass das Problem nicht via TMS spezifiziert ist. Gerade deshalb eignet es sich in konkreten Fällen manchmal besser als Start einer Reduktion auf ein unbekanntes vermutlich unentscheidbares Problem.

Der Satz von Rice klassifiziert nahezu alle Probleme, die mit Eigenschaften der Sprache einer TM zu tun haben, als unentscheidbar.

1.3.1 Das Post´sche Korrespondenzproblem

1.3.00 Definition. Gegeben sei ein Alphabet Σ.

Post'sches Korrespondenzproblem (PCP)
Gegeben: ein Wort $\left\langle \begin{bmatrix} x_i \\ y_i \end{bmatrix} : i < t \right\rangle \in \Sigma^* \times \Sigma^*$.
zu entscheiden: ob ein Index-Wort $w \in t^+$ existiert mit

$$x_{w_0} x_{w_1} \dots x_{w_{n-1}} = y_{w_0} y_{w_1} \dots y_{w_{n-1}}$$

Wir stellen uns die Elemente der Eingabe als Sorten von Domino-Steinen vor; von jeder Sorte gibt es einen unbeschränkten Vorrat. Das zu lösende Problem besteht darin, Domino-Steine dieser Sorten so horizontal nebeneinander zu platzieren, dass oben wie unten dasselbe Wort aus Σ^* entsteht.

1.3.01 Beispiel. Eine Instanz des PCP über $\Sigma = \{0, 1\}$ ist etwa

$$K = \underbrace{\begin{bmatrix} 1 \\ \hline 101 \end{bmatrix}}_{0} \underbrace{\begin{bmatrix} 10 \\ \hline 00 \end{bmatrix}}_{1} \underbrace{\begin{bmatrix} 011 \\ \hline 11 \end{bmatrix}}_{2}$$

copyright: Jürgen Koslowski, TU Braunschweig, 19. Juli 2019

Man stelle sich die Komponenten der Instanz als Sorte von Domino-Steinen vor. Wenn jede Komponente unbeschränkt verfügbar ist, läßt sich eine Indexfolge angeben, so dass in der oberen Hälfte und der unteren Hälfte der entsprechenden Stein-Folge dasselbe Wort über Σ entsteht?

In unserem konkreten Fall leistet die Indexfolge 0212 das Gewünschte:

$$\underbrace{\left[\frac{1}{101}\right]}_{0}\underbrace{\left[\frac{011}{11}\right]}_{2}\underbrace{\left[\frac{10}{00}\right]}_{1}\underbrace{\left[\frac{011}{11}\right]}_{2}$$

Oben wie unten entsteht das Wort 101110011 .

1.3.02 Satz. (Emil Leon Post, 1946) PCP *ist unentscheidbar.*

Letztendlich wollen wir ACCEPT auf PCP reduzieren. Direkt wäre das schwierig, daher führen wir eine modifizierte Version des PCP ein und führen den Beweis in zwei Schritten.

1.3.03 Lemma. *Das*

<u>**modifizierte Post'sche Korrespondenzproblem**</u> *(* MPCP *)*
Gegeben: *ein Wort* $\left\langle \left[\frac{x_i}{y_i}\right] : i < t \right\rangle \in \Sigma^* \times \Sigma^*$.
zu entscheiden: *ob ein Index-Wort* $w \in t^*$ *existiert mit*

$$x_0 x_{w_0} x_{w_1} \dots x_{w_{n-1}} = y_0 y_{w_0} y_{w_1} \dots y_{w_{n-1}}$$

erfüllt

$$\text{MPCP} \leq \text{PCP}$$

Der Unterschied zwischen MPCP und PCP besteht darin, dass bei MPCP die Sorte des ersten Elements einer Lösung festgelegt ist, während eine Lösung von PCP mit jeder beliebigen Sorte beginnen darf.

Beweis.

Wir verschaffen uns etwas „Kitt“ zwischen den Buchstaben des Alphabets Σ, indem wir mit Hilfe eines neuen Symbols @ für jedes Wort $w = w0\, w_1 \dots w_{n-1} \in \Sigma^*$ drei Varianten betrachten:

$$\begin{aligned}
w^{\otimes} &:= @\, w_0 \,@\, w_1 \,@\, w_2 \,@ \dots @\, w_{n-1} \,@ \\
w^{+} &:= @\, w_0 \,@\, w_1 \,@\, w_2 \,@ \dots @\, w_{n-1} \\
w^{-} &:= \quad w_0 \,@\, w_1 \,@\, w_2 \,@ \dots @\, w_{n-1} \,@
\end{aligned}$$

Als Zielalphabet wollen wir $\Delta := \Sigma + \{@, \$\}$ verwenden. Als mögliche Lösungen des Bildes einer Instanz K des PCP über Σ sollen nur solche Indexfolgen in Betracht kommen, bei deren

entsprechenden Δ-Wörtern das Symbol @ mit Elementen aus Σ alterniert bis zum Schluss, der die Form @\$ mit dem neuen Symbol \$ haben soll. Damit sind insbesondere der 0-Komponente von K zwei Komponenten von $f(K)$ zuzuordnen, eine für den Start, und eine zweite, die im Inneren der Folge auftreten kann. Setze also

$$f(K) = \underbrace{\begin{bmatrix} x_0^{\otimes} \\ \hline y_0^{+} \end{bmatrix}}_{0} \underbrace{\begin{bmatrix} x_0^{-} \\ \hline y_0^{+} \end{bmatrix}}_{1} \underbrace{\begin{bmatrix} x_1^{-} \\ \hline y_1^{+} \end{bmatrix}}_{2} \cdots \underbrace{\begin{bmatrix} x_{t-1}^{-} \\ \hline y_{t-1}^{+} \end{bmatrix}}_{t} \underbrace{\begin{bmatrix} \$ \\ \hline @\$ \end{bmatrix}}_{t+1}$$

Nun gilt es zu zeigen

$$K \text{ hat eine Lösung, die mit } 0 \text{ beginnt} \qquad \text{gdw.} \qquad f(K) \text{ hat eine Lösung}$$

($\Rightarrow$) Aus einer Lösung $0w$ für K mit $w \in t^*$ gewinnen wir eine Lösung $0\langle w_i + 1 : 0 < i < t\rangle$ für $f(W)$.

($\Leftarrow$) Jede Lösung von $f(K)$ muß mit 0 beginnen und mit $t+1$ enden, denn das sind die einzigen Komponenten von $f(K)$ in denen das obere und untere Wort mit @ beginnen bzw. enden.

Betrachte eine Lösung $w \in (t+2)^*$ minimaler Länge. Dann treten 0 sowie $t+1$ jeweils genau einmal auf, am Anfang und am Ende. Folglich gilt $w_1 w_2 \ldots w_t \in \{1, 2, \ldots, t\}^*$, und $W' = 0\, w_1 - 1\, w_2 - 1 \ldots w_t - 1$ ist eine Lösung für K. □

1.3.04 Proposition. ACCEPT $\leq$ MPCP.

Beweis. Die Idee besteht darin, für eine Instanz $w\#x$ von ACCEPT die Konfigurationenfolge der Berechnung von M_w auf x mittels der Domino-Steine zu beschreiben, derart, dass eine akzeptierende Berechnung genau einer Lösung des so bestimmten PCP entspricht.

Wir erinnern an die Transitionen zwischen Konfigurationen in Definition 1.0.04, und definieren folgende Sorten von Domino-Steinen:

(0) $D_o = \begin{bmatrix} \# \\ \hline \# q_0 x \# \end{bmatrix}$ für den Start;

(1) $\begin{bmatrix} q a \\ \hline p b \end{bmatrix}$ für jeden Übergang $\langle q, a\rangle \mapsto \langle p, b, N\rangle$; sowie
$\begin{bmatrix} q \# \\ \hline p b \# \end{bmatrix}$ für jeden Übergang $\langle q, \sqcup\rangle \mapsto \langle p, b, N\rangle$;

(2) $\begin{bmatrix} q a \\ \hline b p \end{bmatrix}$ für jeden Übergang $\langle q, a\rangle \mapsto \langle p, b, R\rangle$; sowie
$\begin{bmatrix} q \# \\ \hline b p \# \end{bmatrix}$ für jeden Übergang $\langle q, \sqcup\rangle \mapsto \langle p, b, R\rangle$;

(3) $\begin{bmatrix} \gamma q a \\ \hline p \gamma b \end{bmatrix}$ für jeden Übergang $\langle q, a\rangle \mapsto \langle p, b, L\rangle$ und jedes $\gamma \in \Gamma$; sowie
$\begin{bmatrix} \# q \# \\ \hline p \sqcup b \# \end{bmatrix}$ für jeden Übergang $\langle q, a\rangle \mapsto \langle p, b, L\rangle$; sowie

copyright: Jürgen Koslowski, TU Braunschweig, 19. Juli 2019

$\left[\frac{\gamma\, q\, \#}{p\, \gamma\, b\, \#}\right]$ für jeden Übergang $\langle q, \sqcup\rangle \mapsto \langle p, b, L\rangle$ und jedes $\gamma \in \Gamma$; sowie
$\left[\frac{\#\, q\, \#}{\#\, p \sqcup b\, \#}\right]$ für jeden Übergang $\langle q, \sqcup\rangle \mapsto \langle p, b, L\rangle$;

(4) $\left[\frac{\gamma}{\gamma}\right]$ für jedes $\gamma \in \Gamma$;

(5) $\left[\frac{\gamma\, q_{\mathrm{acc}}}{q_{\mathrm{acc}}}\right]$ und $\left[\frac{q_{\mathrm{acc}}\, \gamma}{q_{\mathrm{acc}}}\right]$ für jedes $\gamma \in \Gamma$;

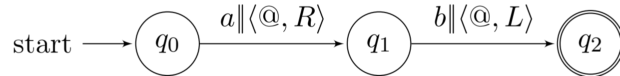
(6) $\left[\frac{\#}{\#}\right]$ and $\left[\frac{\#}{\sqcup\, \#}\right]$;

(7) $\left[\frac{q_{\mathrm{acc}}\, \#\, \#}{\#}\right]$ für den Abschluss.

Nach Konstruktion steht in der kürzesten Lösung des MPCP die Folge der Konfigurationen der akzeptierenden Berechnung, getrennt durch $\#$-Zeichen und mit zwei $\#$-Zeichen am Schluss.

Umgekehrt liefert jede Lösung des MPCP eine kürzeste Lösung, und die listet die Konfigurationenfolge einer akzeptierenden Berechnung auf, oben wie unten. □

1.3.05 Beispiel. Die Maschine M mit $\Sigma = \{a, b\}$ und $\Gamma = \{a, b, @, \sqcup\}$ habe die Übergänge



Zusammen mit der Eingabe $x = aba$ erhalten wir folgende Dominos für MPCP :

(0) $D_o = \left[\frac{\#}{\# q_0\, a\, b\, a\, \#}\right]$ für den Start;

(1) (keine Dominos);

(2) $\left[\frac{q_0\, a}{@\, q_1}\right]$;

(3) $\left[\frac{a\, q_1\, b}{q_2\, a\, @}\right]$, $\left[\frac{b\, q_1\, b}{q_2\, b\, @}\right]$, $\left[\frac{@\, q_1\, b}{q_2\, @\, @}\right]$, $\left[\frac{\sqcup\, q_1\, b}{q_2 \sqcup @}\right]$, $\left[\frac{\#\, q_1\, b}{q_2 \sqcup @}\right]$;

(4) $\left[\frac{a}{a}\right]$, $\left[\frac{b}{b}\right]$, $\left[\frac{@}{@}\right]$, $\left[\frac{\sqcup}{\sqcup}\right]$;

(5) $\left[\frac{a\, q_2}{q_2}\right]$, $\left[\frac{q_2\, a}{q_2}\right]$, $\left[\frac{b\, q_2}{q_2}\right]$, $\left[\frac{q_2\, b}{q_2}\right]$, $\left[\frac{@\, q_2}{q_2}\right]$, $\left[\frac{q_2\, @}{q_2}\right]$, $\left[\frac{\sqcup\, q_2}{q_2}\right]$, $\left[\frac{q_2 \sqcup}{q_2}\right]$;

(6) $\left[\frac{\#}{\#}\right]$ and $\left[\frac{\#}{\sqcup\, \#}\right]$;

(7) $\left[\frac{q_2\, \#\, \#}{\#}\right]$ für den Abschluss.

copyright: Jürgen Koslowski, TU Braunschweig, 19. Juli 2019

Die akzeptierende Berechnung

$$\langle \varepsilon | q_0 | aba \rangle \to \langle @ | q_1 | ba \rangle \to \langle \varepsilon | q_2 | @@a \rangle$$

liefert folgende Lösung des MPCP (nach jedem Berechnungsschritt löschen wir das oben und unten übereinstimmende Präfix bis auf das letzte # oben):

▷
$$\begin{bmatrix} \# \\ \hline \# \, q_0 \, a \, b \, a \, \# \end{bmatrix} \begin{bmatrix} q_0 \, a \\ \hline @ \, q_1 \end{bmatrix} \begin{bmatrix} b \\ \hline b \end{bmatrix} \begin{bmatrix} a \\ \hline a \end{bmatrix} \begin{bmatrix} \# \\ \hline \# \end{bmatrix} \longrightarrow \begin{bmatrix} \# \\ \hline \# \, @ \, q_1 \, b \, a \, \# \end{bmatrix}$$

▷
$$\begin{bmatrix} \# \\ \hline \# \, @ \, q_1 \, b \, a \, \# \end{bmatrix} \begin{bmatrix} @ \, q_1 \, b \\ \hline q_2 \, @ \, @ \end{bmatrix} \begin{bmatrix} a \\ \hline a \end{bmatrix} \begin{bmatrix} \# \\ \hline \# \end{bmatrix} \longrightarrow \begin{bmatrix} \# \\ \hline \# \, q_2 \, @ \, @ \, a \, \# \end{bmatrix}$$

▷
$$\begin{bmatrix} \# \\ \hline \# \, q_2 \, @ \, @ \, a \, \# \end{bmatrix} \begin{bmatrix} q_2 \, @ \\ \hline q_2 \end{bmatrix} \begin{bmatrix} @ \\ \hline @ \end{bmatrix} \begin{bmatrix} a \\ \hline a \end{bmatrix} \begin{bmatrix} \# \\ \hline \# \end{bmatrix} \longrightarrow \begin{bmatrix} \# \\ \hline \# \, q_2 \, @ \, a \, \# \end{bmatrix}$$

▷
$$\begin{bmatrix} \# \\ \hline \# \, q_2 \, @ \, a \, \# \end{bmatrix} \begin{bmatrix} q_2 \, @ \\ \hline q_2 \end{bmatrix} \begin{bmatrix} a \\ \hline a \end{bmatrix} \begin{bmatrix} \# \\ \hline \# \end{bmatrix} \longrightarrow \begin{bmatrix} \# \\ \hline \# \, q_2 \, a \, \# \end{bmatrix}$$

▷
$$\begin{bmatrix} \# \\ \hline \# \, q_2 \, a \, \# \end{bmatrix} \begin{bmatrix} q_2 \, a \\ \hline q_2 \end{bmatrix} \begin{bmatrix} \# \\ \hline \# \end{bmatrix} \longrightarrow \begin{bmatrix} \# \\ \hline \# \, q_2 \, \# \end{bmatrix}$$

▷
$$\begin{bmatrix} \# \\ \hline \# \, q_2 \, \# \end{bmatrix} \begin{bmatrix} q_2 \, \# \, \# \\ \hline \# \end{bmatrix} \longrightarrow \begin{bmatrix} \# \\ \hline \# \end{bmatrix}$$

1.3.06 Bemerkung. Man kann das PCP auf Instanzen vorgegebener Länge einschränken:

Post'sches Korrespondenzproblem der Länge k (PCP_k)
Gegeben: ein Wort $\langle \langle x_i, y_i \rangle : i < k \rangle \in (\Sigma \times \Sigma)^k$.
zu entscheiden: ob ein Wort $w \in t^+$ existiert mit

$$x_{w_0} x_{w_1} \dots x_{w_{n-1}} = y_{w_0} y_{w_1} \dots y_{w_{n-1}}$$

PCP_1 ist natürlich trivial, während PCP_2 entscheidbar ist. Dagegen ist PCP_9 als unentscheidbar bekannt. Für die Werte $k \in \{3, 4, 5, 6, 7, 8\}$ ist die Frage der Entscheidbarkeit noch offen.

1.3.2 Der Satz von Rice

Der Satz von Rice besagt im Wesentlichen, dass jedes *nicht-triviale* Problem hinsichtlich des Verhaltens von TMs unentscheidbar ist.

copyright: Jürgen Koslowski, TU Braunschweig, 19. Juli 2019

1.3.07 Definition. $\boldsymbol{RE}(\Sigma)$ bezeichnet die Menge der semi-entscheidbaren Sprachen über dem Alphabet Σ. *Eigenschaften* P, die solche Sprachen haben können, entsprechen *Teilmengen* von $\boldsymbol{RE}(\Sigma)$, bzw. charakteristischen Funktionen $\boldsymbol{RE}(\Sigma) \xrightarrow{P} \mathbb{B} = \{0,1\}$. Ist letztere konstant, heißt die Eigenschaft *trivial*.

Jede Sprachenin $\boldsymbol{RE}(\Sigma)$ hat eine endliche Darstellung in Form einer TM, die sie semi-entscheidet. Identifiziert man die Eigenschaft P mit der Sprache

$$\{\, w \in \{0,1\}^* \;:\; P(\mathcal{L}(M_w)) = 1 \,\}$$

reduziert man das Problem P zu entscheiden auf das Problem, diese Sprache zu entscheiden.

Achtung: ob eine Codierung in P liegt darf nur von der Sprache $L \in \boldsymbol{RE}(\Sigma)$ abhängen, nicht von der gewählten Maschine (davon gibt es abzählbar unendlich viele).

1.3.08 Beispiele. Die folgenden Eigenschaften semi-entscheidbarer Sprachen sind nicht trivial:

- ▷ $L = \mathcal{L}(M_w)$ ist endlich;
- ▷ $L = \mathcal{L}(M_w)$ ist regulär;
- ▷ $L = \mathcal{L}(M_w)$ ist kontextfrei;
- ▷ $L = \mathcal{L}(M_w)$ ist entscheidbar;
- ▷ L enthält 10110, d.h., M_w akzeptiert die Eingabe 10110;
- ▷ L ist universell, d.h., $L = \Sigma^*$;

Dagegen sind folgende Eigenschaften trivial:

- ▷ L ist Bild einer totalen berechenbaren Funktion;
- ▷ L ist nicht semi-entscheidbar;
- ▷ L wird von einer Maschine mit einer geraden Anzahl an Zuständen akzeptiert.

Und schließlich beziehen sich die folgenden Eigenschaften auf die akzeptierende Maschine, und nicht auf die akzeptierte Sprache:

- ▷ M_w hat 481 Zustände;
- ▷ Die Berechnung von M_w auf 10110 hält nach höchstens 10 Schritten;
- ▷ M_w ist ein Entscheider.
- ▷ Es gibt eine kleinere TM, die dieselbe Sprache wie M_w akzeptiert.

copyright: Jürgen Koslowski, TU Braunschweig, 19. Juli 2019

1.3.09 Satz. (Satz von Rice, Henry Gordon Rice, 1953) *Jede nicht-triviale Eigenschaft semi-entscheidbarer Sprachen ist unentscheidbar.*

Beweis.

Sei $P \in \boldsymbol{RE}(\Sigma)$ nicht-trivial, und oBdA $P(\emptyset) = 0$. Wegen der Nichttrivialität von P existiert $L \in \boldsymbol{RE}(\Sigma)$ mit $P(L) = 1$. Wähle eine dTM K mit $\mathcal{L}(K) = L$.

Behauptung: $\textsc{hp} \leq P = \{ w \in \{0,1\}^* : P(\mathcal{L}(M_w)) = 1 \}$.

Einer Instanz $w\#x$ von $\textsc{hp}$ wollen wir (den Code) eine(r) Maschine $M^K_{w,x}$ zuordnen mit

$$x \in \mathcal{L}(M_w) \quad \text{gdw.} \quad P(\mathcal{L}(M^K_{x,x})) = 1$$

Spezifikation von $M^K_{w,x}$: Bei Eingabe von y

- ▷ verschiebe y auf ein separates Band;
- ▷ schreibe x auf das Eingabeband;
- ▷ simulierte M_w auf x (braucht nicht zu terminieren);
- ▷ falls die obige Simulation terminiert, simulierte K auf y;
- ▷ akzeptiere y genau dann, wenn diese Simulation y akzeptiert.

Falls $x \notin \mathcal{L}(M_w)$ kommt es nicht zur Simulation von K auf y, daher gilt dann $\mathcal{L}(M^K_{w,x}) = \emptyset$. Andernfalls gilt $y \in \mathcal{L}(M^K_{w,x}$ gdw. $y \in L$.

Des weiteren ist klar, dass man aus den Codes $w = \langle M_w \rangle$ und $\langle K \rangle$ den Code einer Maschine $M^K_{w,x}$ berechnen kann. □

Achtung: der Satz von Rice macht keine Aussagen über Semi-Entscheidbarkeit oder das Komplement, co-Semi-Entscheidbarkeit. Es gibt nicht-trivial Eigenschaften, die semi-entscheidbar, und solche die co-semi-entscheidbar sind.

1.3.10 Definition. Eine Eigenschaft semi-entscheidbarer Sprachen über Σ heißt *monoton*, falls aus $L \subseteq L'$ folgt $P(L) \leq P(L')$.

1.3.11 Satz. (Rice, 1956) *Jede nicht-monotone Eigenschaft semi-entscheidbarer Sprachen ist nicht semi-entscheidbar.* □

copyright: Jürgen Koslowski, TU Braunschweig, 19. Juli 2019

1.4 Unentscheidbare Probleme kontextfreier Sprachen

Manche nicht-triviale Probleme kontextfreier Sprachen sind im Gegensatz zum semi-entscheidbaren Fall tatsächlich entscheidbar:

▷ Das Wortproblem für eine Sprache $L \subseteq \Sigma^*$, die durch eine kontextfreie Grammatik oder einen PDA spezifiziert ist: verwende den CYK-Algorithmus.

▷ Das Leerheitsproblem, ob $\mathcal{L}(G) = \emptyset$ gilt.

Auf die folgenden unentscheidbaren Probleme läßt sich jeweils das PCP reduzieren.

1.4.00 Satz. *Gegeben seien zwei kontextfreie Grammatiken G_i, $I < 2$. Folgende Probleme sind unentscheidbar:*

(1) *Gilt* $\mathcal{L}(G_0) \cap \mathcal{L}(G_1) = \emptyset$ *?*

(2) *Gilt* $|\mathcal{L}(G_0) \cap \mathcal{L}(G_1)| = \infty$ *?*

(3) *Ist* $\mathcal{L}(G_0) \cap \mathcal{L}(G_1)$ *kontextfrei?*

(4) *Gilt* $\mathcal{L}(G_0) \subseteq \mathcal{L}(G_1)$ *?*

(5) *Gilt* $\mathcal{L}(G_0) = \mathcal{L}(G_1)$ *?*

Beweis.

(1) Wir reduzieren das $\{0,1\}$-PCP. Sei

$$\left\langle \begin{bmatrix} x_i \\ \hline y_i \end{bmatrix} : i < k \right\rangle \in (\{0,1\}^* \times \{0,1\}^*$$

eine Instanz des PCP über $\{0,1\}$. Als Terminal-Alphabet für beide Grammatiken verwenden wir eine Menge mit $k+3$ Elementen: $\Delta = \{0,1,\$\} + \{\, a_i : i < k \,\}$.

Grammatik G_0 mit dem Nichtterminal-Alphabet $\{S, A, B\}$ habe die Produktionen

$$\begin{aligned} S &\to A\$B, \\ A &\to a_i A x_i \ , \ i < k \\ A &\to a_i x_i \ , \ i < k \\ B &\to y_i^{\mathrm{rev}} B a_i \ , \ i < k \\ B &\to y_i^{\mathrm{rev}} a_i \ , \ i < k \end{aligned}$$

wobei w^{rev} die Spiegelung von w ist. Man überzeugt sich leicht, dass

$$\mathcal{L}(G_0) = \{\, a_{i_{n-1}} \dots a_{i_0} x_{i_0} \dots x_{i_{n-1}} \$ y_{i_{m-1}}^{\mathrm{rev}} \dots y_{i_0}^{\mathrm{rev}} a_{i_0} \dots a_{i_{m-1}} \ : \ i, j \in k^* \,\}$$

copyright: Jürgen Koslowski, TU Braunschweig, 19. Juli 2019

wobei wir die Zahl k mit der Menge $\{0, 1, \ldots, k-1\}$ identifiziert haben.

G_1 mit dem Nichtterminal-Alphabet $\{S, T, \$\}$ hat die Produktionen

$$S \to T \mid a_i S a_i, i < k$$
$$T \to \$ \mid 0T0 \mid 1T1$$

und erzeugt somit die Sprache

$$\mathcal{L}(G_1) = \{uv\$v^{\text{rev}}u^{\text{rev}} \;:\; v \in \{0,1\}^*, u \in \{\, a_i \;:\; i < k\,\}^*\,\}$$

Die Instanz K des PCP hat genau dann eine Lösung $i \in k^+$, wenn $\mathcal{L}(G_0 \cap \mathcal{L}(G_1)$ nicht leer ist, nämlich das Wort

$$a_{i_{n-1}} \ldots a_{i_0} x_{i_0} \ldots x_{i_{n-1}} \$ y_{i_{n-1}}^{\text{rev}} \ldots y_{i_0}^{\text{rev}} a_{i_0} \ldots a_{i_{n-1}}$$

enthält. Die Reduktion besteht also in der Zuordnung $K \mapsto \langle G_0, G_1 \rangle$ und ist folglich berechenbar.

(2) Wenn eine Instanz eines PCP eine Lösung hat, dann hat sie auch unendlich viele Lösungen, denn jede Lösung kann iteriert werden. Daher können wir die obige Reduktion auch für das Problem der Unendlichkeit des Durchschnitts verwenden.

(3) Da nach Satz 1.0.19 die Klasse der entscheidbaren Probleme unter der Bildung von Komplementen abgeschlossen ist, gilt das auch für die Klasse der unentscheidbaren Probleme. Insofern genügt es, das Komplement der Kontextfreiheit des Durchschnitts als unentscheidbar nachzuweisen.

Aber das gelingt mit der Reduktion aus Teil (1). Im Fall des leeren und somit kontextfreien Durchschnitts hat das PCP keine Lösung. Es bleibt zu zeigen, dass im nichtleeren Fall der Durchschnitt $\mathcal{L}(G_0)\mathcal{L}(\mathcal{G}_\infty)$ nicht kontextfrei ist. Aber das gelingt z.B. mit dem Pumping-Lemma für kontextfreie Sprachen (vergl. TheoInf 1).

(4) Wir reduzieren das nach obiger Überlegung ebenfalls unentscheidbare Problem der Leerheit des Schnitts auf das Inklusionsproblem. Dazu identifiziert man die beiden Sprachen als deterministisch kontextfrei, d.h., sie werden von deterministischen PDAs akzeptiert. Da die Klasse der deterministisch kontextfreien Sprachen unter Komplement abgeschlossen ist (aber weder unter Vereinigung noch Durchschnitt), existiert eine kontextfreie Grammatik $\overline{G_1}$ für das Komplement $\overline{\mathcal{L}(G_1)}$. Wegen

$$\mathcal{L}(G_0) \cap \mathcal{L}(G_1) = \emptyset \quad \text{gdw.} \quad \overline{\mathcal{L}(G_1)} = \mathcal{L}(G_0) \subseteq \mathcal{L}(\overline{G_1})$$

ist die Zuordnung $\langle G_0, G_1 \rangle \mapsto \langle G_0, \overline{G_1} \rangle$ die gewünschte Reduktion.

(5) Kontextfreie Sprachen sind unter Vereinigung abgeschlossen, es läßt sich sogar leicht eine kontextfreie Grammatik G_2 konstruieren mit $\mathcal{L}(G_2) = \mathcal{L}(G_0) \cup \mathcal{L}(G_1)$. Wegen

$$\mathcal{L}(G_0) \in \mathcal{L}(G_1) \quad \text{gdw.} \quad \mathcal{L}(G_2) = \mathcal{L}(G_1)$$

ist die Zuordnung $\langle G_0, G_1 \rangle \mapsto \langle G + 2, G_1 \rangle$ die gewünschte Reduktion. □

copyright: Jürgen Koslowski, TU Braunschweig, 19. Juli 2019

1.4.01 Bemerkung. Die Probleme (1) – (4) aus Satz 1.4.00 sind aufgrund der Beobachtung im Beweis zu Teil (3) auch im deterministisch kontextfreien Fall unentscheidbar. Allerdings ist Sprachgleichheit im deterministisch -kontextfreien Fall entscheidbar, wie 2001 von Géraud Sénizergues bewiesen wurde [Sé01], [Sé02]. Dafür wurde ihm 2002 der Gödel-Preis verliehen.

1.4.02 Bemerkung. Mit Hilfe der obigen Grammatiken lassen sich auf folgende Probleme eine kontextfreie Grammatik betreffend als unentscheidbar nachweisen:

- ▷ Ist G eindeutig?
- ▷ Ist $\overline{\mathcal{L}(G)}$ kontextfrei?
- ▷ Ist $\mathcal{L}(G)$ regulär?
- ▷ Ist $\mathcal{L}(G)$ deterministisch kontextfrei?
- ▷ Ist $\mathcal{L}(G)$ universell?

copyright: Jürgen Koslowski, TU Braunschweig, 19. Juli 2019

2 Komplexitätstheorie

Bisher haben wir nur untersucht, ob ein Problem überhaupt von einer TM semi-entschieden oder entschieden werden kann. Für Entscheider, die immer halten, kann man zusätzlich fragen, wieviel Zeit und wieviel Speicherplatz bei der Lösung einer konkreten Instanz benötigt wird. Wenn es gelingt, jeder Eingabe auf sinnvolle Weise eine „Größe“ zuzuordnen, kann man diese zum zeitlichen und räumlichen Aufwand in Beziehung setzen. Das ermöglicht es, entscheidbare Probleme feiner zu klassifizieren.

2.0 Zeit- und Raumkomplexität

Sofern nicht anders spezifiziert, halten in diesem Kapitel alle TMs, deterministisch oder nicht, auf jede Eingabe. Der Begriff „Entscheider“ war oben für dTMs eingeführt worden, die immer halten. Wir wollen diese Einschränkung beibehalten und im nicht-deterministischen Fall nicht von „Entscheidern“ sprechen (vergl. HA).

Wir untersuchen den Ressourcenverbrauch zunächst für spezifische Eingaben, dann für Eingaben spezifischer syntaktischer Länge, und schließlich bezogen auf spezielle Schrankenfunktionen, bzw. Mengen von solchen.

2.0.1 Zeitkomplexität

2.0.00 Definition. Wir betrachten eine TM M, die immer hält, potentiell nicht-deterministisch und potentiell mit mehreren Bändern, über Σ.

(1) Der *Zeitverbrauch*, oder die *Rechenzeit*, von M auf Eingabe $x \in \Sigma^*$ ist

$$\boldsymbol{Time}_M(x) = \max\{\, n \;:\; n \text{ ist die Länge einer haltenden Berechnung von } M \text{ auf } x \,\}$$

wobei die Länge der Berechnung die Anzahl der Schritte bis zum Erreichen eines Haltezustands ist. Gemäß der Voraussetzung terminiert jede Berechnung.

(2) Die *Zeitkomplexität* von M ist Worst-Case-Verhalten von M auf Eingaben vorgegebener Länge:

$$\boldsymbol{TIme}_M(n) = \max\{\, \boldsymbol{Time}_M(x) \;:\; |x| = n, \}$$

(3) Für eine Funktion $\mathbb{N} \xrightarrow{t} \mathbb{N}$ heißt M t*-zeitbeschränkt*, wenn $\boldsymbol{TIme}_M(n) \leq t(n)$ für alle $n \in \mathbb{N}$. ***DTIME NTIME***

Nun können wir die grundlegenden Zeitkomplixitätsklassen definieren:

2.0.01 Definition. Für eine Funktion $\mathbb{N} \xrightarrow{t} \mathbb{N}$ setzen wir

$$\boldsymbol{DTIME}_k(t) = \{\, \mathcal{L}(M) \;:\; M \text{ ist eine } t\text{-zeitbeschränkte } k\text{-Band dTM} \,\}$$

$$\boldsymbol{NTIME}_k(t) = \{\, \mathcal{L}(M) \;:\; M \text{ ist eine } t\text{-zeitbeschränkte } k\text{-Band nTM} \,\}$$

copyright: Jürgen Koslowski, TU Braunschweig, 19. Juli 2019

Für eine Menge $T \subseteq \mathbb{N}^{\mathbb{N}}$ sind $\boldsymbol{DTIME}_k(T)$ bzw. $\boldsymbol{NTIME}_k(T)$ die entsprechenden Vereinigungen über alle $t \in T$.

2.0.02 Bemerkungen.

- ▷ Wenn multiplikative Konstanten keine Rolle spielen, verwendet man oft die Klasse $T = \mathcal{O}(t)$.
- ▷ Sublineare Zeitschranken $\mathbb{N} \xrightarrow{t} \mathbb{N}$ ergeben wenig Sinn, a entsprechend zeitbeschränkte Maschinen nicht einmal ihre Eingabe lesen könnten.

2.0.2 Raumkomplexität

Hier kommt die Zusatzbedingung hinsichtlich des speziellen **read-only** Eingabebandes zum Tragen, siehe Abschnitt 1.1. Unter einer k-Band Maschine verstehen wir eine Maschine mit k *Arbeitsbändern*, das Eingabeband zählt nicht mit.

2.0.03 Definition. Wir betrachten eine TM M, die immer hält, potentiell nicht-deterministisch und potentiell mit mehreren Bändern, über Σ.

(1) Der *Platzverbrauch* von M in Konfiguration c ist

$$\boldsymbol{space}_M(c) = \max\{\, |w| \ : \ w \text{ ist der Inhalt eines Arbeitsbandes in Konfiguration } c \,\}$$

wobei Blankzeichen außerhalb des Bereichs, in dem andere Bandsymbole auftraten, nicht mitzählen, vergl. die Äquivalenzrelation auf Konfigurationen vor Definition 1.0.04.

(2) Der *Platzverbrauch* von M bei der Berechnung von x ist

$$\boldsymbol{space}_M(x) = \max\{\, \boldsymbol{space}_M(c) \ : \ c \text{ ist Konfiguration in einer Berechnung von } M \text{ auf } a \,\}$$

(3) Die *Platzkomplexität* von M ist Worst-Case-Verhalten von M auf Eingaben vorgegebener Länge:

$$\boldsymbol{space}_M(n) = \max\{\, \boldsymbol{space}_M(x) \ : \ |x| = n, \}$$

(4) Für eine Funktion $\mathbb{N} \xrightarrow{s} \mathbb{N}$ heißt M s*-platzbeschränkt*, wenn $_M(n) \leq s(n)$ für alle $n \in \mathbb{N}$.

Nun können wir auch hier die grundlegenden Platzkomplexitätsklassen definieren:

2.0.04 Definition. Für eine Funktion $\mathbb{N} \xrightarrow{s} \mathbb{N}$ setzen wir

$$\boldsymbol{DSPACE}_k(s) = \{\mathcal{L}(M) \ : \ M \text{ ist eine } s\text{-platzbeschränkte } k\text{-Band dTM}\,\}$$
$$\boldsymbol{NSPACE}_k(s) = \{\mathcal{L}(M) \ : \ M \text{ ist eine } s\text{-platzbeschränkte } k\text{-Band nTM}\,\}$$

Für eine Menge $S \subseteq \mathbb{N}^{\mathbb{N}}$ sind $\boldsymbol{DSPACE}_k(S)$ bzw. $\boldsymbol{NSPACE}_k(S)$ die entsprechenden Vereinigungen über alle $s \in S$.

copyright: Jürgen Koslowski, TU Braunschweig, 19. Juli 2019

2.0.05 Lemma. *Für jede Funktion $r \in \mathbb{N}^{\mathbb{N}}$ und jedes $k > 0$ gilt*

$$\boldsymbol{DTIME}_{k+1}(r) \subseteq \boldsymbol{DSPACE}_k(r) \quad \textit{und} \quad \boldsymbol{NTIME}_{k+1}(r) \subseteq \boldsymbol{NSPACE}_k(r)$$

2.0.06 Beispiel. Über $\Sigma = \{a, b\}$ betrachten wir die Sprache mit gleich vielen Symbolen a und b:

$$L = \{\, x \in \{a, b\}^* \,:\, |x|_a = |x|_b \,\}$$

Behauptung: $L \in \boldsymbol{DSPACE}_1(\mathcal{O}(\log n))$, d.h., L läßt sich deterministisch mit logarithmischem (und insbesondere sublinearem) Platzverbrauch lösen.

Eine entsprechende Maschine ist wie folgt spezifiziert: Auf dem einzigen Arbeitsband wird ein binärer Zähler mit 0 initialisiert und dann beim Durchlauf über die Eingabe bei jedem Auftreten von a um eins vergrößert, und von b um eins verkleinert. Sein Wert ist auf den Bereich von $-|x|$ bis $|x|$ beschränkt und benötigt $\lceil |x| \rceil + 2$ Zellen zur Speicherung.

Eine weniger effiziente Maschine könnte die Eingabe auf das Arbeitsband kopieren, und in mehreren Durchläufen jeweils ein a und ein b löschen, bis die kopierte Eingabe nach einem Durchlauf vollständig überschrieben wurde. Das verbraucht linearen Platz und quadratische Zeit.

2.0.3 Die robusten Komplexitätsklassen

Wir können nun die sog. *robusten* Komplexitätsklassen definieren. Manche von ihnen tragen noch descriptive Namen:

copyright: Jürgen Koslowski, TU Braunschweig, 19. Juli 2019

2.0.07 Definition.

$$
\begin{aligned}
\boldsymbol{L} &:= \boldsymbol{DSPACE}(\mathcal{O}(\log n)) && \text{(aka } \boldsymbol{LOGSPACE}) \\
\boldsymbol{NL} &:= \boldsymbol{NSPACE}(\mathcal{O}(\log n)) && \text{(aka } \boldsymbol{NLOGSPACE}) \\
\boldsymbol{P} &:= \bigcup_{k\in\mathbb{N}} \boldsymbol{DTIME}(\mathcal{O}(n^k)) && \text{(aka } \boldsymbol{PTIME}) \\
\boldsymbol{NP} &:= \bigcup_{k\in\mathbb{N}} \boldsymbol{NTIME}(\mathcal{O}(n^k)) && \text{(aka } \boldsymbol{NPTIME}) \\
\boldsymbol{PSPACE} &:= \bigcup_{k\in\mathbb{N}} \boldsymbol{DSPACE}(\mathcal{O}(n^k)) \\
\boldsymbol{NPSPACE} &:= \bigcup_{k\in\mathbb{N}} \boldsymbol{NSPACE}(\mathcal{O}(n^k)) \\
\boldsymbol{EXP} &:= \bigcup_{k\in\mathbb{N}} \boldsymbol{DTIME}(\mathcal{O}(2^{\mathcal{O}(n^k)}) && \text{(aka } \boldsymbol{EXPTIME}) \\
\boldsymbol{NEXP} &:= \bigcup_{k\in\mathbb{N}} \boldsymbol{NTIME}(\mathcal{O}(2^{\mathcal{O}(n^k)}) && \text{(aka } \boldsymbol{NEXPTIME}) \\
\boldsymbol{EXPSPACE} &:= \bigcup_{k\in\mathbb{N}} \boldsymbol{DSPACE}(\mathcal{O}(2^{\mathcal{O}(n^k)}) \\
\boldsymbol{NEXPSPACE} &:= \bigcup_{k\in\mathbb{N}} \boldsymbol{NSPACE}(\mathcal{O}(2^{\mathcal{O}(n^k)})
\end{aligned}
$$

Die Klasse $\boldsymbol{P}$ umfasst alle Problem, die sich deterministisch in Plynomialzeit lösen lassen. Eine Sprache L gehört also zu $\boldsymbol{P}$, wenn es einen Exponenten $k \in \mathbb{N}$ gibt, so dass L bereits in $\boldsymbol{DTIME}(\mathcal{O}(n^k))$ liegt. Wichtig ist, dass der Exponent nicht von der Länge der Eingabe abhängt.

2.0.08 Bemerkungen.

(0) Man kann auch größere Klassen betrachten, etwa die Klassen $m\boldsymbol{EXP}$ und $m\boldsymbol{EXPSPACE}$ der Probleme, die sich mit m-fach exponentiellem Zeit-/Platzverbrauch lösen lassen, etwa

$$L \in 2\boldsymbol{EXP} \quad \text{gdw.} \quad \exists k \in \mathbb{N}.\, L \in \boldsymbol{DTIME}\left(2^{2^{\mathcal{O}\left(n^k\right)}}\right)$$

Schließlich sei

$$\boldsymbol{ELEMENTARY} := \bigcup_{m>0} m\boldsymbol{EXP}$$

Wir werden später sehen, dass bei dieser Definition die Unterscheidung zwischen deterministischen und nicht-deterministischen Maschinen irrelevant ist, ebenso die Unterscheidung zwischen Zeit- und Platzbedarf.

(1) Es gibt nicht-elementare Probleme, bei denen Entscheidungsverfahren mit Eingaben der Länge n eine Laufzeit haben, die $f(n)$-fach exponentiell ist, wobei f eine super-lineare Funktion ist.

copyright: Jürgen Koslowski, TU Braunschweig, 19. Juli 2019

(2) Allgemein wird $\boldsymbol{P}$ als die Klasse der *effizient lösbaren* Probleme betrachtet; für jede Sprache $L \in \boldsymbol{P}$ existiert ein $k \in \mathbb{N}$ so dass die Laufzeit einer Eingabe der Länge n im Wesentlichen durch n^k beschränkt ist. Man glaubt, dass sich auch größere Instanzen mit erträglichem Zeitaufwand lösen lassen. (Na ja, der AKS-Primalitätstest hat in der aktuellen Version einen Zeitbedarf von $\tilde{\mathcal{O}}\left((\log n)^6\right)$, und wird daher kaum verwendet. Andere Tests sind i.A. viel schneller, aber deren worst-case Verhalten ist noch nicht bekannt.)

Während die grundlegenden Klassen mit der Anzahl der (Arbeits-)Bänder parametrisiert waren, entfällt das bei den robusten Klassen:

2.0.09 Lemma. *Für jede Funktion* $\mathbb{N} \xrightarrow{f} \mathbb{N}$ *und jede Zahl* $k > 0$ *gilt*

$$\begin{array}{ll} \boldsymbol{DTIME}_k(f) \subseteq \boldsymbol{DTIME}_1(f \cdot f) & \boldsymbol{DSPACE}_k(f) \subseteq \boldsymbol{DSPACE}_1(f) \\ \boldsymbol{NTIME}_k(f) \subseteq \boldsymbol{NTIME}_1(f \cdot f) & \boldsymbol{NSPACE}_k(f) \subseteq \boldsymbol{NSPACE}_1(f) \end{array}$$

Beweis.

Aufgrund der Beweisskizze für Satz 1.0.13 wissen wir, wie eine n-Band Maschine durch eine $2n$-Spur Maschine und somit durch eine 1-Band-Maschine simuliert werden kann, unabhängig davon, of die Maschine deterministisch ist oder nicht. Jeder Schritt der n-Band-Maschine erfordert n Schritte auf der $2n$-Spur-Maschine, und die relativen Kopfpositionen auf den ungradzahligen Spuren können sich nach k Schritten der n-Band Maschine um bis zu $2k$ Positionen voneinander entfernen, was $\mathcal{O}\left(k^2\right)$ Bewegungen bei der Simulation erfordert. Ein separates **read-only** Eingabeband ändert nichts an diesem Argument. □

2.0.10 Corollar. *Die Definitionen der robusten Komplexitätsklassen ist unabhängig von der Anzahl der verwendeten Bänder, z.B.*

$$\boldsymbol{L} = \bigcup_{t>0} \boldsymbol{DSPACE}_t\left(\mathcal{O}\left(\log n\right)\right) \quad \textit{und} \quad \boldsymbol{P} = \bigcup_{t>0} \bigcup_{k\in\mathbb{N}} \boldsymbol{DTIME}_t\left(\mathcal{O}\left(n^k\right)\right)$$

2.0.4 Komplementklassen

Der Übergang von einem Problem, also einer Sprache $L \subseteq \{0,1\}^*$ zum Komplement $\overline{L} = \{0,1\}^* - L$ schlägt sich in den Komplexitätsklassen wie folgt nieder:

2.0.11 Definition. Die *Komplementklasse* einer Komplexitätsklasse $\mathcal{C}$ besteht aus den Komplementen der Probleme in $\mathcal{C}$, d.h.,

$$\text{co}\,\mathcal{C} = \{\, \overline{L} \,:\, L \in \mathcal{C} \,\}$$

2.0.12 Beispiel. Man kann zeigen (Übungen?), dass das bekannte Kopierproblem

$$\textsc{copy} = \{\, w\#w \,:\, w \in \{0,1\}^* \,\}$$

copyright: Jürgen Koslowski, TU Braunschweig, 19. Juli 2019

zu $\boldsymbol{DTIME}_1\left(\mathcal{O}\left(n^2\right)\right) \subseteq \boldsymbol{P}$ und zu $\boldsymbol{DSPACE}_1(\mathcal{O}(\log n)) \subseteq \boldsymbol{L}$ gehört. Daher läßt sich

$$\overline{\mathrm{COPY}} = \{0,1\}^* \cup \{0,1,\#\}^*\#\{0,1,\#\}^*\#\{0,1,\#\}^* \cup \{\, u\#v \,:\, u,v \in \{0,1\}^* \wedge u \neq v \,\}$$

in co$\boldsymbol{DTIME}_1\left(\mathcal{O}\left(n^2\right)\right) \subseteq$ co$\boldsymbol{P}$ sowie co$\boldsymbol{DSPACE}_1(\mathcal{O}(\log n)) \subseteq$ co$\boldsymbol{L}$ verorten.

2.0.13 Lemma. *Für jede Funktion $\mathbb{N} \xrightarrow{f} \mathbb{N}$ und alle $k > 0$ gilt*

$$\boldsymbol{DSPACE}_k(f) = co\boldsymbol{DSPACE}_k(f) \quad \textit{sowie} \quad \boldsymbol{DTIME}_k(f) = co\boldsymbol{DTIME}_k(f)$$

Insbesondere sind diese Klassen unter Komplementbildung abgeschlossen.

Beweis.

Ein deterministischer Entscheider für $L \subseteq \Sigma^*$ wird zu einem deterministischen Entscheider für $\overline{L} = \Sigma^* - L$, indem man die Rollen von q_{acc} und q_{rej} vertauscht. Dabei ändern sich weder Platz noch Zeitverbrauch. □

2.0.14 Corollar.

$$\boldsymbol{L} = co\boldsymbol{L} \quad , \quad \boldsymbol{PSPACE} = co\boldsymbol{PSPACE} \quad , \quad \boldsymbol{P} = co\boldsymbol{P}$$ □

Wie sich nichtdeterministische Komplexitätsklassen unter Komplementbildung verhalten ist eine nicht-triviale Frage. Wir werden später $\boldsymbol{NL} =$ co$\boldsymbol{NL}$ und $\boldsymbol{NPSPACE} =$ co$\boldsymbol{NPSPACE}$ zeigen. Ob $\boldsymbol{NP}$ mit co$\boldsymbol{P}$ übereinstimmt ist bisher nicht bekannt, wird aber allgemein bezweifelt.

2.0.5 Simple Relationen zwischen Komplexitätsklassen

2.0.15 Lemma. *Für jede Funktion $\mathbb{N} \xrightarrow{f} \mathbb{N}$ gilt*

$$\boldsymbol{DTIME}(f) \subseteq \boldsymbol{NTIME}(f) \quad \textit{und} \quad \boldsymbol{DSPACE}(f) \subseteq \boldsymbol{NSPACE}(f)$$

Beweis. Der Begriff der nTM subsumiert den der dTM. □

Aufgrund von Lemma 2.0.05 gilt außerdem:

2.0.16 Corollar.

$$\boldsymbol{P} \subseteq \boldsymbol{PSPACE} \quad \textit{und} \quad \boldsymbol{NP} \subseteq \boldsymbol{NSPACE}$$ □

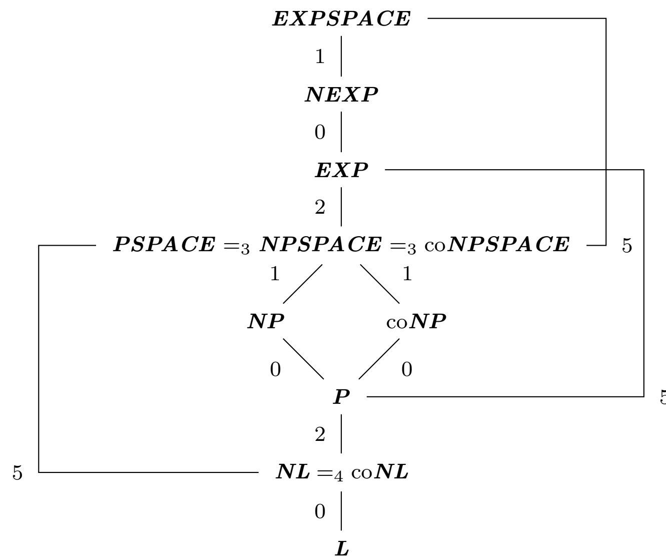
copyright: Jürgen Koslowski, TU Braunschweig, 19. Juli 2019

2.1 Eine Landkarte der Komplexität

Im Rest der VL verfolgen wir zwei Ziele:

- ▷ Die Untersuchung der robusten Komplexitätsklassen anhand typischer Probleme und Analyse; welche Algorithmen lassen sich mit gegebenem Zeit- und Platzverbrauch umsetzen.
- ▷ In welchem Verhältnis stehen die robusten Klassen zueinander?

Schlussendlich ergibt sich folgender Graph, dessen Kanten Inklusionen von unten nach oben sind:



(0) wegen Lemma 2.0.15

(1) Jede Zeit- ist auch eine Platzschranke. Die Simulation sogar nicht-deterministischer t-zeitbeschränkter Maschinen gelingt mit t-platzbeschränkten deterministischen Maschinen.

(2) Eine Maschine mit Platzverbrauch t läßt sich, selbst im nicht-deterministischen Fall, durch eine deterministische Maschine mit Zeitschranke 2^t simulieren.

(3) Der Satz von Savitch zeigt $\boldsymbol{NSPACE}(f) \subseteq \boldsymbol{DSPACE}(f \cdot f)$. Daraus folgt $\boldsymbol{PSPACE} = \boldsymbol{NPSPACE}$, und aufgrund von $\boldsymbol{PSPACE} = \mathrm{co}\boldsymbol{PSPACE}$ die Gleichheit aller drei Klassen. Aus dem Satz läßt sich aber nicht $\boldsymbol{L} = \boldsymbol{NL}$ folgern; dieses Problem ist immer noch offen.

(4) Der Satz von Immerman und Szelepcsényi zeigt $\boldsymbol{NL} = \mathrm{co}\boldsymbol{NL}$.

(5) Die Hierarchie-Resultate für Zeit und Platz zeigen, dass man mit exponentiell mehr Zeit bzw. Platz echt mehr Probleme lösen kann. Das impliziert die Echtheit der folgenden Inklusionen

$$\boldsymbol{NL} \subset \boldsymbol{PSPACE} \quad , \quad \boldsymbol{P} \subset \boldsymbol{EXP} \quad \text{und} \quad \boldsymbol{PSPACE} \subset \boldsymbol{EXPSPACE}$$

Die Striktheit aller übrigen Inklusionen ist offen, insbesondere $\boldsymbol{P} \stackrel{?}{=} \boldsymbol{NP} \stackrel{?}{=} \mathrm{co}\boldsymbol{NP}$.

copyright: Jürgen Koslowski, TU Braunschweig, 19. Juli 2019

2.2 $\boldsymbol{L}$ und $\boldsymbol{NL}$

Wir stellen einige Probleme in $\boldsymbol{L}$ bzw. $\boldsymbol{NL}$ vor. Die Frage, ob beide Klassen übereinstimmen, ist bisher offen. Um Kandidaten zu finden, die diese Klassen evtl. voneinander abgrenzen können, führen wir einen passende Begriff von many-one Reduktionen ein und das damit einhergehende Konzept der Vollständigkeit.

2.2.1 Probleme in $\boldsymbol{L}$: Arithmetik

Addition und Multiplikation bestimmen zwei grundlegende E-Probleme der Arithmetik

Addition (ADD)
Gegeben: natürliche Zahlen i, j, ℓ.
zu entscheiden: ob die Summe $i+j$ mit ℓ übereinstimmt.

Multiplikation (MULT)
Gegeben: natürliche Zahlen i, j, ℓ.
zu entscheiden: ob das Produkt $i \cdot j$ mit ℓ übereinstimmt.

Als Wortprobleme über $\{0,1\}$ also

$$\begin{aligned} \text{ADD} &= \{\, x\#y\#z \;:\; \exists\, i,j,\ell \in \mathbb{N}.\, x = \operatorname{bin}(i),\, y = \operatorname{bin}(j),\, z = \operatorname{bin}(\ell) \wedge x+y=z \,\} \\ \text{MULT} &= \{\, x\#y\#z \;:\; \exists\, i,j,\ell \in \mathbb{N}.\, x = \operatorname{bin}(i),\, y = \operatorname{bin}(j),\, z = \operatorname{bin}(\ell) \wedge x\cdot y=z \,\} \end{aligned}$$

2.2.00 Lemma. ADD *und* MULT *liegen in* $\boldsymbol{L}$.

Beweis.

Der Trick besteht darin, im Fall, dass keines der Argumente 0 ist, die einzelnen Bits von $x+y$ bzw. $x\cdot y$ zu berechnen, mit dem entsprechenden Bit von z zu vergleichen, und das Ergebnis dann zu verwerfen. Danach kann derselbe Platz zur Berechnung des nächsten Bits wiederverwendet werden. Außerdem braucht man eine konstante Anzahl an Bits für einen eventuellen Übertrag, sowie einen Zähler für die aktuelle Position des zu berechnenden Bits. Der Platzbedarf dieses Zählers ist logarithmisch in der Länge der Eingabe, also doppelt logarithmisch in den Werten x, y und z. Die weiteren Details des Beweises seien der lesenden Person überlassen. □

2.2.2 Das Pfadproblem in $\boldsymbol{NL}$

Die wichtigsten Probleme in $\boldsymbol{NL}$ betreffen die Existenz/Nicht-Existenz von Pfaden in einem gerichteten Graphen.

Pfadexistenz (PATH)
Gegeben: gerichteter Graph $G = \langle V, E\rangle$ mit $E \subseteq V \times V$, Knoten $s, t \in V$
zu entscheiden: ob ein Pfad von s nach t in G existiert

Zwecks Umwandlung in ein binäres Wortproblem ist festzulegen, wie der Graph binär codiert

copyright: Jürgen Koslowski, TU Braunschweig, 19. Juli 2019

werden soll. Als Knotenmenge kann man oBdA die Menge $n = \{0, 1, \ldots, n-1\}$ mit der entsprechenden Binärdarstellung verwenden. Als Codierung der Kanten kommen etwa binäre $n \times n$-Matrizen in Frage, oder Ajazenzlisten, die jedem Knoten $i < n$ die Liste der von i aus erreichbaren direkten Nachbarn zuordnet. Die zweite Variante braucht ./A. etwas weniger Platz, da man sich die Nullen in der Adjazenzmatrix spart und nur tatsächlich vorhandene Kanten codiert werden. (Dafür wird der Übergang zum Komplementärgraphen schwieriger, der in bestimmten Problemen hilfreich sein kann.) Die Adjazenzliste für Knoten i kann man als

$$\boldsymbol{ad}(i) := \operatorname{bin}(i)\#\operatorname{bin}(j_0)\#\ldots\#\operatorname{bin}(j_{k-1})$$

schreiben, und den Graphen mit n Knoten als

$$\boldsymbol{AD}(G) := \boldsymbol{ad}(0)\#\#\boldsymbol{ad}(1)\#\#\ldots\boldsymbol{ad}(n-1)\#\#$$

Damit erhalten wir PATH als Wortproblem:

$$\text{PATH} = \{\, \boldsymbol{AD}(G)\#\operatorname{bin}(s)\#\operatorname{bin}(t) \ : \text{im Graphen } G \text{ existiert ein Pfad voin } s \text{ nach } t \,\}$$

2.2.01 Lemma. PATH *liegt in* $\boldsymbol{NL}$.

Beweis.

Wenn es einen Pfad von s nach t gibt, dann auch einen Pfad mit Länge $< n$, in dem kein Knoten doppelt auftritt.

Ein Algorithmus kann zufällig eine Knotenfolge κ der Länge $< n-2$ aus $V - \{s, t\}$ raten und überprüfen, ob dadurch ein Pfad von s nach t bestimmt wird. Man braucht aber nicht die gesamte Folge zu kennen, um mit der Überprüfung zu beginnen, sondern kann sich Glied für Glied „weiterhangeln“:

(0) initialisiere den Zähler z mit 0;

(1) setze $\kappa_z = s$;

(2) teste, ob $\kappa_z = t$; falls ja, akzeptieren;

(3) teste, ob t in $\boldsymbol{ad}(s)$ vorkommt; falls ja, akzeptieren;

(4) solange $z < n-1$:

(5) $z := z + 1$

(6) wähle zufällig κ_{z+1} in $\boldsymbol{ad}(\kappa_z)$, nach Voraussetzung gilt $\kappa_{z+1} \neq t$;

(7) GOTO (3)

copyright: Jürgen Koslowski, TU Braunschweig, 19. Juli 2019

Der Algorithmus ist korrekt: wenn es einem Pfad der Länge $<n$ von s nach t in G gibt; kann dieser geraten werden, und der Algorithmus akzeptiert. Umgekehrt garantiert eine akzeptierende Berechnung die Existenz eines Pfades von s nach t.

Der Zähler z, die obere Schranke $n-1$ und die beiden Knoten κ_z und κ_{z+1} benötigen jeweils $\log n$ Platz. Dasselbe gilt für den Testknoten t, was es in $\boldsymbol{ad}(\kappa_z)$ zu suchen gilt. Dafür wird maximal eine konstante Anzahl weiterer Zeiger auf die Eingabe benötigt, mit demselben Platzbedarf.

Da jede Adjazenzliste nicht-leer ist, sie trägt zumindest den Namen ihres Knotens, hat die Eingabe mindestens die Länge n. □

2.2.3 Reduktionen, Härte und Vollständigkeit

Bisher ist unbekannt, ob die Inklusion $\boldsymbol{L} \subseteq \boldsymbol{NL}$ echt ist, oder nicht. Um zumindest eine Idee zu bekommen, welche Probleme in $\boldsymbol{NL}$ möglicherweise nicht in $\boldsymbol{L}$ liegen könnten, wollen wir eine Quasi-Ordnung auf der Klasse aller Probleme definieren, bzgl. der die robusten Klassen nicht nur untere Abschnitte sind, sondern sogar ein bis auf Äquivalenz größtes Element, d.h., eine kleinste obere Schranke enthalten; derartige Elemente heißen aus historischen Gründen „vollständig“ in ihrer Klasse. Das Problem, $\boldsymbol{NL}$ von $\boldsymbol{L}$ zu trennen, besteht nun darin zu untersuchen, ob es ein $\boldsymbol{NL}$-vollständiges Problem gibt, das zu $\boldsymbol{L}$ gehört. Beide Klassen stimmen genau dann überein, wenn ein solches Problem existiert.

In Kapitel 1.2 haben wir bereits mit Hilfe von (many-one)-Reduktionen (Definition 1.2.05) eine Quasi-Ordnung auf der klasse aller Probleme definiert (Lemma 1.2.06). Die dortige Quasiordnung ist allerdings zu grob für das hier beschriebene Problem, wir müssen sie verfeinern.

2.2.02 Definition. Sei $\boldsymbol{R}$ eine Klasse von Funktionen zwischen freien Monoiden, die alle Identitätsfunktionen enthält und unter Komposition abgeschlossen ist. Ein Problem $K \subseteq \Sigma^*$ heißt *$\boldsymbol{R}$-(many-one)-reduzierbar* auf $L \subseteq \Delta^*$, wenn es eine (many-one)-Reduktion im Sinne von Definition 1.2.05 von K auf L gibt, die zu R gehört. Schreibweise $K \leq^{\boldsymbol{R}} L$.

Aufgrund unserer Forderung an $\boldsymbol{R}$ ist unmittelbar klar:

2.2.03 Lemma. *$\leq^{\boldsymbol{R}}$ ist eine Quasi-Ordnung und $\sim^{\boldsymbol{R}} := \leq^{\boldsymbol{R}} \cap \geq^{\boldsymbol{R}}$ ist eine Äquivalenzrelation auf der Klasse aller Probleme.* □

Bzgl. Quasiordnungen kann ganz analog wie für Halbordnungen (= reflexiven, transitiven und antisymmetrischen Relationen) von oberen und unteren Schranken sprechen:

2.2.04 Definition. Wir betrachten eine Klasse $\mathcal{C}$ von Problemen.

▷ Ein Problem B heißt *$\mathcal{C}$-hart* bzgl. $\boldsymbol{R}$, wenn es eine obere Schranke von $\mathcal{C}$ bzgl. $\leq^{\boldsymbol{R}}$ ist, d.h., wenn

$$C \leq^{\boldsymbol{R}} B \quad \text{für alle} \quad C \in \mathcal{C}$$

copyright: Jürgen Koslowski, TU Braunschweig, 19. Juli 2019

▷ Ein Problem $B \in \mathcal{C}$ heißt $\mathcal{C}$ *-vollständig* bzgl. $\boldsymbol{R}$, wenn es $\mathcal{C}$-hart bzgl. $\boldsymbol{R}$ ist.

Die Klassen der $\mathcal{C}$-harten und $\mathcal{C}$-vollständigen Probleme wollen wir in dieser VL mit $\mathcal{C}$-$\boldsymbol{H}$ bzw. $\mathcal{C}$-$\boldsymbol{V}$ bezeichnen. Insbesondere gilt $\mathcal{C}\text{-}\boldsymbol{V} = \mathcal{C} \cap \mathcal{C}\text{-}\boldsymbol{H}$.

Intuitiv sind $\mathcal{C}$-harte Probleme bzgl. $\boldsymbol{R}$ mindestens so schwierig wie alle Probleme in $\mathcal{C}$, während $\mathcal{C}$-vollständige Probleme bzgl. $\boldsymbol{R}$ die schwierigsten Probleme in $\mathcal{C}$ sind.

Die $\mathcal{C}$-harten Probleme bzgl. $\boldsymbol{R}$ bilden immer einen *oberen Abschnitt* bzgl. der Quasi-Ordnung $\leq^{\boldsymbol{R}}$: ist B $\mathcal{C}$-hart und gilt $B \leq^{\boldsymbol{R}} A$, so ist A erst recht eine obere Schranke von $\mathcal{C}$ bsgl. $\boldsymbol{R}$, und damit ebenfalls $\mathcal{C}$-hart bzgl. $\boldsymbol{R}$.

Á priori ist allerdings nicht klar, dass der Durchschnitt dieses oberen Abschnitts mit $\mathcal{C}$ nicht leer ist! Das wird ganz entscheidend von der konkreten Klasse $\boldsymbol{R}$ and erlaubten Reduktions-Funktionen abhängen. Man kann aber immer die Menge der unteren $\leq^{\boldsymbol{R}}$-Schranken der $\mathcal{C}$-harten Probleme bzgl. $\boldsymbol{R}$ betrachten, das ist ein unterer $\leq^{\boldsymbol{R}}$-Abschnitt, der $\mathcal{C}$ umfasst. Mit anderen Worten, alle Probleme aus $\mathcal{C}$ sind untere Schranken der $\mathcal{C}$-harten Probleme bzgl. $\boldsymbol{R}$.

In der Komplexitätstheorie werden zwei spezielle Reduktionsklassen $\boldsymbol{R}$ und die von ihnen induzierten Quasiordnungen eine Rolle spielen:

▷ Reduktionen, die in Polynomialzeit berechenbar sind ($\leq^{\text{poly}}$), und

▷ Reduktionen, die in logarithmischem Platz berechenbar sind ($\leq^{\log}$).

Der Ressourcenverbrauch von Berechnern wird ganz analog zu dem von Entscheidern definiert, wobei die Besonderheiten von Berechnern hinsichtlich Ein- und Ausgabeband zu beachten sind, siehe Abschnitt 1.1: ein **read-only** Eingabeband, und ein **one-way** Ausgabeband, die beide beim Platzbedarf nicht mitgezählt werden, das sie nicht zum Rechnen verwendet werden können. Insbesondere sind Berechner dTMs, die immer halten.

2.2.05 Definition. M sei ein totaler Berechner.

▷ Sein Zeitverbrauch wird durch eine Funktion $\mathbb{N} \xrightarrow{t} \mathbb{N}$ beschränkt, wenn für jedes $n \in \mathbb{N}$ und jede Eingabe x der Länge $|x| = n$ der Berechner nach höchstens $t(n)$ Schritten hält.

▷ Sein Platzverbrauch wird durch eine Funktion $\mathbb{N} \xrightarrow{s} \mathbb{N}$ beschränkt, wenn für jedes $n \in \mathbb{N}$ und jede Eingabe x der Länge $|x| = n$ der Berechner auf jedem Arbeitsband höchstens $s(n)$ Zellen nutzt.

2.2.06 Definition. Gibt es für eine totale Funktion $\Sigma^* \xrightarrow{f} \Delta^*$ einen Berechner M, der zu jeder Eingabe $x \in \Sigma^*$ nach endlich vielen Schritten hält und die Ausgabe $f(x)$ auf das Ausgabeband geschrieben hat, so heißt f

▷ *logspace-berechenbar*, wenn Speicherverbrauch von M durch $\mathcal{O}(\log(n))$ beschränkt ist;

copyright: Jürgen Koslowski, TU Braunschweig, 19. Juli 2019

▷ *in Polynomialzeit berechenbar*, wenn eine Konstante $k \in \mathbb{N}$ existiert, so dass der Zeitverbrauch von M durch $\mathcal{O}(n^k)$ beschränkt ist.

2.2.07 Bemerkung.

(0) Sobald wir den Abschluss dieser Funktionsklassen unter Komposition nachgewiesen haben (siehe Lemmata 2.2.08 und 2.2.09), die Identitätsfunktionen sind automatisch enthalten, wird $K \subseteq \Sigma^*$ *logspace-* bzw. *in Polynomialzeit reduzierbar* auf $L \in \Delta^*$ heißen, geschrieben $K \leq^{\log} L$ bzw. $K \leq^{\text{poly}} L$, sofern eine totale Funktion $\Sigma^* \xrightarrow{f} \Delta^*$ des passenden Typs existiert mit

$$x \in K \quad \text{gdw.} \quad f(x) \in L$$

(1) Intuitiv(?) sind logspace-Reduktionen dazu gedacht, Probleme in $\boldsymbol{L}$ von denen in größeren Klassen zu separieren, während Polynomialzeit-Reduktionen eine entsprechende Rolle für Probleme in $\boldsymbol{P}$ spielen sollen. Um Probleme innerhalb von $\boldsymbol{L}$ bzw. $\boldsymbol{P}$ zu unterscheiden, sind sie jedoch zu „grob“.

2.2.08 Lemma. *Die Komposition logspace-berechenbarer Funktionen ist wieder logspace-berechenbar.*

Beweis. Die Funktionen $\Sigma^* \xrightarrow{f} \Delta^* \xrightarrow{g} \Omega^*$ mögen Berechner M_f und M_g mit logarithmischem Platzverbrauch haben. Wir konstruieren einen Berechner $M_{g \circ f}$ mit ebenfalls logarithmischem Platzverbrauch.

Achtung: ein Zwischenspeichern von $f(x)$ bei Eingabe x auf einem der Arbeitsbänder von $M_{g \circ f}$ verletzt i.A., die Platzschranke. Daher sollen die einzelnen Symbole von $f(x)$ immer nur bei Bedarf berechnet und der Simulation von M_g zur Verfügung gestellt werden.

▷ Die Maschine $M_{g \circ f}$ wird einen Zähler benötigen, der die gerade benötigte Position von $f(x)$ angibt. Damit dieser Zähler den Platzbeschränkungen genügt, ist nachzuweisen, dass $f(x)$ höchstens polynomiell groß in $n = |x|$ ist, denn $\log(n^d) = d \cdot \log(n)$.

Eine Konfiguration von M_f bei der Berechnung von $f(x)$ hängt von folgenden Parametern ab:

– dem Zustand, $|Q|_f$ Möglichkeiten;
– der Kopfposition auf dem Eingabeband, $n = |x|$ Möglichkeiten;
– dem Inhalt der Arbeitsbänder: auf jedem der k Bänder können maximal $a \cdot \log n + b$ viele Zellen mit Nicht-Blanks beschrieben sein; dazwischen können aber auch Blanks vorkommen, also gibt es $|\Gamma|^{a \cdot \log n + b}$ Möglichkeiten;
– die Kopfposition auf jedem der Arbeitsbänder ist ebenfalls durch $a \cdot \log n + b$ beschränkt;
– eer Inhalt des Ausgabebandes und die dortige Kopfposition sind irrelevant.

copyright: Jürgen Koslowski, TU Braunschweig, 19. Juli 2019

Damit gibt es zur Eingabe x mit $n = |x|$ maximal

$$|Q| \cdot n \cdot k \left(|\Gamma|^{a \cdot \log n + b} + a \cdot \log n + b \right) = |Q| \cdot n \cdot k \left((n^a \cdot 2^b)^{\log |\Gamma|} + a \cdot \log n + b \right)$$

Konfigurationen, was sich in $\mathcal{O}(n^c)$ für eine geeignete Konstante c verorten läßt.

Als dTM, die immer hält, kann der Berechner M_f keine dieser Konfigurationen wiederholen, hält also spätestens nach entsprechend vielen Schritten. Aber das beschränkt dann auch die Länge der Ausgabe $f(x)$. Der angesprochene Zähler benötigt also $\mathcal{O}(\log n)$ viel Platz.

▷ Die Berechnung des Symbols in Position i von $f(n)$ ist mit logarithmischem Platzbedarf möglich: man ergänzt M_f un einen entsprechenden Zähler und unterdrückt alle Ausgaben in Positionen $j < i$ von $f(x)$; unmittelbar nach Ausgabe der gewünschten Position kann die Simulation von M_f abgebrochen werden.

Die Simulation von M_g im Berechner $M_{g \circ f}$ unterliegt keinen Einschränkungen, daher gibt $M_{g \circ f}$ bei Eingabe dieselben Ergebnisse aus wie M_g bei Eingabe $f(x)$. □

Das entsprechende Ergebnis fur in Polynomialzeit berechenbare Funktionen ist viel einfacher:

2.2.09 Lemma. *Die Komposition in Polynomialzeit berechenbarer Funktionen ist wieder in Polynomialzeit berechenbar.*

Beweis. Die Funktionen $\Sigma^* \xrightarrow{f} \Delta^* \xrightarrow{g} \Omega^*$ mögen Berechner M_f und M_g mit polynomialem Zeitverbrauch haben. Der Berechner $M_{g \circ f}$, der das Ausgabeband von M_f als Arbeitsband und dann als Eingabeband für M_g verwendet, hat ebenfalls polynomialem Zeitverbrauch, da Polynome unter Substitution abgeschlossen sind. □

Damit können wir nun die in Bemerkung 2.2.07(0) vorweggenommenen Reduktionsbegriffe verwenden.

2.2.10 Lemma. ***NL****ist ein unterer Abschnitt bzgl.* $\leq^{\log}$ *und* ***NP*** *ist ein unterer Abschnitt bzgl.* $\leq^{\text{poly}}$.

Beweis. Betrachte $K \leq^{\log} L \in \boldsymbol{NL}$, einen Berechner M_f, der die Reduktion realisiert und eine Maschine M_L, die L in logarithmischem Platz löst. Anstelle M_f und M_L direkt zu verknüpfen, was zuviel Platz für das Resultat von f auf einem Arbeitsband benötigen würde, verfährt man wie im obigen Beweis und berechnet die nötigen Bits von $f(x)$ on-the-fly bei Bedarf. Das liefert einen Entscheider für K mit logarithmischem Platzbedarf.

Wie zuvor ist das Problem im Fall $K \leq^{\log} L \in \boldsymbol{NP}$ einfacher, da wir die Maschinen direk verknüpfen können. □

copyright: Jürgen Koslowski, TU Braunschweig, 19. Juli 2019

2.2.4 *Path* ist *NL*-vollständig

Wir hatten die Reduktions-Quasi-Ordnungen eingeführt, um untersuchen zu können, ob sich innerhalb von Klassen wie ***NL*** oder ***NP*** die „schwierigsten“ Probleme von den „einfachen“ in ***L*** bzw. ***P*** separieren lassen. Nun gilt es zunächst zu zeigen, dass es innerhalb von ***NL*** tatsächlich schwierige, vulgo ***NL***-harte Probleme bzgl. $\leq^{\log}$ gibt, vergl. Definition 2.2.04:

2.2.11 Satz. PATH *ist* ***NL****-hart bzgl.* $\leq^{\log}$.

Beweis.

Um ein beliebiges Problem $K \in \boldsymbol{NL}$ auf PATH zu reduzieren, betrachten wir einen Entscheider M für K mit logarithmischem Platzbedarf. Für jede Instanz x von K. d.h., für jede Eingabe x führt M eine eindeutige Berechnung aus, also folgt einem eindeutigen Pfad von der Initialkonfiguration $\langle \varepsilon | q_0 | x \rangle$ zu einer Haltekonfiguration.

Á priori kann es im Konfigurationsgraphen für M mehrere akzeptierende Haltekonfigurationen geben, die sich im Bandinhalt unterscheiden und in den Kopfpositionen. Und falls $x \in K$ ist nicht klar, welche davon letztendlich erreicht wird.

Daher ist M zunächst derart zu modifizieren, dass es nur noch eine Haltekonfiguration gibt: am einfachsten dürfte es sein zu verlangen, dass alle Arbeitsbänder leer sind und der Kopf auf dem Eingabeband wieder auf dem ersten Eingabesymbol steht. Um die Arbeitsbänder leeren zu können, müssen vor Arbeitsbeginn Anfangs- und Endmarkierer, etwa \$ und @, auf jedem Arbeitsband nebeneinander geschrieben werden. Alle zu schreibenden Symbole sind dann dazwischen zu positionieren; dazu ist ggf. \$ nach links und @ nach rechts zu verschieben. Verschiebungen in die andere Richtung sind nicht nötig. Nach Halt der ursprünglichen Maschine M kann dann auf jedem Arbeitsband ausgehend vom linken Rand sämtlicher Bandinhalt bis zum rechten Rand gelöscht werden. Den linken Rand der Eingabe zu finden, ist trivial.

Wir nehmen nun oBdA an, M hat nur eine akzeptierende Konfiguration. Einer Instanz x von K wollen wir eine Instanz $f(x)$ von PATH zuordnen, deren Graph ein Teilgraph des Konfigurationsgraphen von M ist, mit $s = \langle \varepsilon | q_0 | x \rangle$ und $t = \langle eps | q_{\text{acc}} | \varepsilon \rangle$. Es verbleibt, die von $\langle \varepsilon | q_0 | x \rangle$ aus erreichbaren Konfigurationen zu codieren. Zustand und Kopfpositionen relativ zum Bandinhalt lassen sich durch Konstanten bzw. $k \cdot \log |x|$ Bits speichern, bei k Arbeitsbändern. Da der Platzverbrauch von M durch $\mathcal{O}(\log |x|)$ beschränkt ist, existiert ein positives $d \in \mathbb{N}$, so dass sich der Inhalt jedes Arbeitsbandes mit höchstens $d \cdot \log |x|$ darstellen läßt. Man beachte, dass x nicht in den Konfigurationen von M genannt werden braucht, x existiert auf dem Eingabeband des Berechners M_f von f, und aus der Kopfposition auf dem Eingabeband ließe sich der linke und rechte Bandinhalt bei Bedarf rekonstruieren.

Zusammenfassend: für die fest gewählte Maschine M lassen sich Konstanten r und s finden, so dass jede Konfiguration für die Berechnung einer beliebigen Eingabe x mit $r \cdot \log |x| + s$ Zeichen dargestellt werden kann.

copyright: Jürgen Koslowski, TU Braunschweig, 19. Juli 2019

M_f möge wie folgt arbeiten:

- ▷ M_f verfüge über den Code von M; das braucht konstanten Platz;
- ▷ zunächst werden systematisch alle (codierten) Wörter aus dem Bandalphabet von M der Länge $< r \cdot \log|x| + s$ auf ein Arbeitsband geschrieben und daraufhin untersucht, ob sie syntaktisch korrekte Konfigurationen von M darstellen; im positiven Fall wird ein Zähler inkrementiert; dann wird die potentielle Konfiguration von der nächsten überschrieben. Auf diese Weise erfahren wir die Zustandszahl N eines relevanten Teilgraphen des Konfigurationsgraphen von M, ein Wert mit $\mathcal{O}(\log\log|x|)$ Bits, der somit gespeichert werden kann.
- ▷ Anschließend werden in zwei geschachtelten Schleifen die Konfigurationen Nr. i und Nr. j von Neuem erzeugt und daraufhin untersucht, ob ein direkter Übergang möglich ist; im Positiven Fall wird j der Adjazenzliste von i hinzugefügt.

Auf diese Weise benötigt die Berechnung der Instanz $f(x)$ für PATH nur logarithmischen Platz in $|x|$. □

2.2.12 Corollar. PATH *ist* ***NL*** *-vollständig.*

Man kann das Problem sogar auf azyklische gerichtete Graphen einschränken, in denen es keine Kreise gibt, denn die betrachteten Konfigurationsteilgraphen haben diese Eigenschaft, sonst wäre M kein Entscheider.

Pfadexistenz in azyklischen Graphen (ACPATH)
Gegeben: gerichteter azyklischer Graph $G = \langle V, E\rangle$ mit $E \subseteq V \times V$, Knoten $s, t \in V$
zu entscheiden: ob ein Pfad von s nach t in G existiert

Allerdings braucht der gesamte Konfigurationsgraph von M nicht azyklisch zu sein, insofern wird noch eine Reduktion PATH $\leq^{\log}$ ACPATH benötigt, was als Übungsaufgabe betrachtet werden sollte.

2.3 co*NL* und der Satz von Immermann und Szelepcśenyi

In der Klasse co***NL*** der Komplemente von Problemen in ***NL*** wird sich das Problem 2 SAT der Erfüllbarkeit aussagenlogischer Formeln in KNF mit Klauseln der Länge ≤ 2 als vollständig erweisen. Da aufgrund des Satzes von Immermann und Szelepcśenyi co***NL*** mit ***NL*** übereinstimmt, ist 2 SAT auch ***NL***-vollständig.

Zunächst stellen wir fest:

2.3.00 Lemma. *Ist $\mathcal{C}$ eine Komplexitätsklasse, $\boldsymbol{R}$ eine unter Identitäten und Komposition abgeschlossene Klasse von Funktionen zwischen freien Monoiden, dann ist $L \subseteq \Sigma^*$ genau dann $\mathcal{C}$ -hart/$\mathcal{C}$ -vollständig, wenn $\overline{L} = \Sigma^* - L$ co$\mathcal{C}$ -hart/co$\mathcal{C}$ -vollständig ist.*

copyright: Jürgen Koslowski, TU Braunschweig, 19. Juli 2019

Beweis. HA. □

Insbesondere sind damit

Nichtexistenz von Pfaden in Graphen (NOPATH)
Gegeben: gerichteter Graph $G = \langle V, E\rangle$ mit $E \subseteq V \times V$, Knoten $s, t \in V$
zu entscheiden: ob kein Pfad von s nach t in G existiert

und

Nichtexistenz von Pfaden in azyklischen Graphen (NOACPATH)
Gegeben: gerichteter azyklischer Graph $G = \langle V, E\rangle$ mit $E \subseteq V \times V$, Knoten $s, t \in V$
zu entscheiden: ob kein Pfad von s nach t in G existiert

co$\boldsymbol{NL}$-vollständig.

2.3.01 Bemerkung. Das Komplement $\{0,1\}^*$-PATH zerfällt in zwei Teilen: Binärwörter, die keinen Graphen mit zwei ausgezichneten Knoten codieren, und solche, die zwar einen Graphen mit ausgezeichneten Knoten s und t codieren, bei denen t von s aus nicht erreicht werden kann; dies ist das oben beschriebene Problem NOPATH . Die Frage, ob eine valide Graphencodierung vorliegt, ist ein Syntaxcheck und läßt sich deterministisch mit logarithmischem Platzverbrauch lösen, liegt also in $\boldsymbol{L}$. Da $\boldsymbol{NL}$ unter Vereinigungen abgeschlossen ist, liegt das Kompplement $\{0,1\}^*$-PATH genau dann in $\boldsymbol{NL}$ wenn NOPATH in $\boldsymbol{NL}$ liegt. Daher wird gelegentlich zwischen diesen beiden Problemen nicht genau genug unterschieden. Analog verhält es sich mit $\{0,1\}^*$-ACPATH und NOACPATH .

2.3.1 2-SAT

Wir erinnern an den Begriff der „Konjunktiven Normalform“ einer aussagenlogischen Formel:

2.3.02 Definition. Wir betrachten nur die Junktoren $\neg$ (Negation), $\wedge$ (Konjunktion) und $\vee$ (Disjunktion).

- ▷ Atomare Formeln und ihre Negationen heißen *Literale.*
- ▷ *Klauseln* sind Disjunktionen von Literalen. Wir sprechen von k- Klauseln, $k \in \mathbb{N}$, wenn darin höchstens k verschiedene Literale auftreten. Man identifiziert Klausen häufig auch mit der (endlichen) Menge ihrer Literale.
- ▷ Eine Formel in *konjunktiver Normalform (KNF)* ist eine Konjunktion von Formeln, gern auch als endliche Menge von endlichen Literalmengen geschrieben. Man spricht von k- *KNF*, falls nur k- Klauseln vorkommen. Man identifiziert Formeln in KNF häufig mit der endlichen Menge ihrer Klauseln.

Damit ergeben sich mehrere Entscheidungsprobleme, zuvorderst:

copyright: Jürgen Koslowski, TU Braunschweig, 19. Juli 2019

Erfüllbarkeit aussagenlogischer Formeln in KNF (SAT) **Gegeben:** aussagenlogische Formel F in KNF **zu entscheiden:** ob F erfüllbar ist, d.h., existiert eine Belegung φ mit $\hat{\varphi}(F) = 1$?

Man interessiert sich ebenfalls für die Abhängigkeit der Härte dieses Problems von diversen Parametern, hier speziell für jedes $k \in \mathbb{N}$:

Erfüllbarkeit aussagenlogischer Formeln in k- KNF (k - SAT) **Gegeben:** aussagenlogische Formel F in k- KNF **zu entscheiden:** ob F erfüllbar ist, d.h., existiert eine Belegung φ mit $\hat{\varphi}(F) = 1$?

Offenbar sind 0-SAT und 1-SAT trivial. In späteren Kapiteln werden wir auch k-SAT für $k > 2$ betrachten.

Zwecks sinnvoller Codierung von Formeln F in KNF oder betrachten wir eine Aufzählung p_i , $i \in \mathbb{N}$, der Atome. Sofern in F höchstens Atome p_i mit $i < N$ auftreten so braucht man zur Codierung der möglichen Literale eine disjunkte Vereinigung der Menge $\text{bin}[N] = \{ \text{bin}(i) : i < N \}$ mit sich selbst, etwa rote und blaue Binärdarstellungen für positive bzw. negative Literale. Klauseln der Länge n sind dann Tupel $X_0\#X_1\#\ldots\#X_{n-1} \in (\text{bin}[N] + \text{bin}[N])^n$ (oBdA ohne Wiederholungen), während Formeln in KNF2 endliche Listen von Klauseln sind (ebenfalls ohne Wiederholungen), getrennt etwa durch $\#\#$ und mit dem Präfix $\text{bin}(N)\#\#\#$ versehen. Im Fall von k-KNF ist die Länge der Klauseln durch k beschränkt.

2.3.03 Satz. *2-SAT ist co**NL**-vollständig.*

Für den Beweis ist zu zeigen, dass sowohl 2-SAT $\in$ co***NL*** als auch 2-SAT $\in$ co***NL-H*** gilt. Der Nachweis von 2-SAT $\in$ ***NL*** kann prinzipiell auf zwei verschiedene Arten erfolgen:

▷ Entweder konstruiert man eine nTM M , die 2-NOSAT mit logarithmischem Platzverbrauch löst. Die Länge der Eingabe ist durch $N^2 \cdot \log N$ beschränkt, also steht maximal $2 \log N \cdot \log\log N$ Platz zum Arbeiten zur Verfügung. Aber eine Belegung der n ersten Atome benötigt bereits N Speicherzellen, und damit zu viel Platz. Insofern ist nicht unmittelbar klar, wie eine deratige Maschine arbeiten könnte.

▷ Alternativ folgt mit einem zu einer HA analogen Argument, dass ***NL*** bzgl $\leq^{\log}$ ein unterer Abschnitt ist. Daher genügt es, 2-SAT auf ein bekanntes Problem in co***NL*** zu reduzieren. Diese Strategie werden wir verfolgen, sie erfordert aber einige Vorarbeiten.

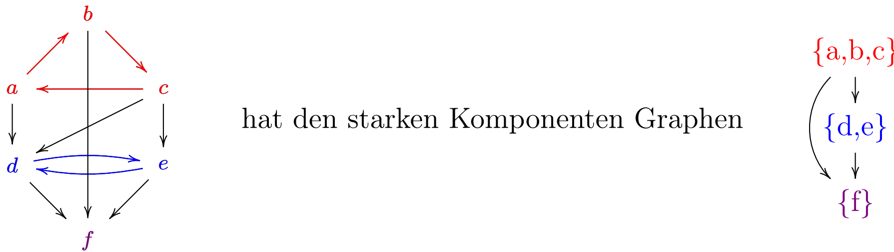
2.3.04 Definition. $G = \langle V, E \rangle$ sei ein gerichteter Graph.

(0) Der *Graph* $\tilde{G}$ *der starken Komponenten* von G hat als Knoten die Äquivalenzklassen bzgl. der Erreichbarkeitsrelation in G , und die von G induzierten Kanten. Genauer: $\langle K, K' \rangle$ ist genau dann eine Kante von $\tilde{G}$, wenn eine G -Kante von einem K -Knoten zu einem K' -Knoten existiert.

copyright: Jürgen Koslowski, TU Braunschweig, 19. Juli 2019

(1) Ist G azyklisch, so versteht man unter einer *topologischen Sortierung* von G eine Abbildung $V \xrightarrow{t} \mathbb{N}$, so dass aus $\langle u, v\rangle \in E$ folgt $t(u) < t(v)$.

2.3.05 Beispiel.



Die starken Komponenten von G bilden die Knoten eines Graphen $\bar{G}$: . Konstruktionsbedingt ist $\tilde{G}$ azyklisch, also topologisch sortierbar.

2.3.06 Bemerkungen.

▷ Die starken Komponenten eines gerichteten Graphen sind berechenbar:

Initialisiere den Komponentenzähler i zu 0
while $V \neq \emptyset$ **do**
  Wähle Knoten $v \in V$.
  Entferne v aus V und starte eine neue Komponente $K_i = \{v\}$.
  for Knoten $w \in V$ **do**
    Setze $\varphi(L') = 1$.
    if es gibt einen Pfad von v nach w und zurück **then**
      Verschiebe den Knoten w von V nach K_i .
    end if
  end for
  Setze $i := i + 1$.
end while

▷ Eine topologische Sortierung eines azyklischen Graphen kann algorithmisch bestimmt werden:

Initialisiere den zuzuweisenden Wert i zu 0 .
Initialisiere die Menge W der noch zu bearbeitenden Konten zu V .
while W enthält einen Knoten x ohne Vorgänger in W **do**
  $t(x) = i$;
  $i := i + 1$;
  $W := W - \{x\}$
end while

Es gibt effizientere Algorithmen, aber das ist hier irrelevant.

copyright: Jürgen Koslowski, TU Braunschweig, 19. Juli 2019

2.3.07 Lemma. *Das Problem*

Funktionale Nichtexistenz von Pfaden *(* SINVKOMP *)* **Gegeben:** *gerichteter Graph* $\boldsymbol{G} = \langle V, E\rangle$ *und eine selbstinverse Abbildung* $V \xrightarrow{f} V$ *ohne Fixpunkte* **zu entscheiden:** *ob jeder Knoten* $u \in V$ *in einer anderen starken Komponente liegt als* $f(u)$

liegt in co$\boldsymbol{NL}$.

Beweis. Man kann eine nTM, die NOPATH in logarythmischem Platz löst, iterativ einsetzen um für jeden Knoten $u \in V$ zu testen, ob in G kein Pfad von u nach $f(u)$ oder kein Pfad von $f(u)$ nach u existiert. □

2.3.08 Lemma. *2-*SAT $\leq^{\log}$ SINVKOMP

Beweis. Die Reduktion $\{0,1\}^* \xrightarrow{\gamma} \{0,1\}^*$ bildet eine Formel in 2-KNF auf ihren sog. *Implikationsgraphen* $\gamma(F)$ ab: dessen Knoten sind die potentiellen Literale zu allen in F auftretenden Atomen, und jede KLausel $m \vee \ell$ induziert zwei Kanten $\neg m \to \ell$ und $\neg \ell \to m$. Alle übrigen Binärwörter werden auf den Graphen mit einerm Knoten und einer Kante abgebildet.

Behauptung: Die Reduktion ist in logarithmischem Platz berechenbar: die Maximalzahl an Atomen N ist zu Verdoppeln, was durch Anhängen einer 0 geschieht; Inspektion der Klauseln liefert die Adjazenzlisten der Knoten in $\gamma(F)$.

Behauptung: Die Reduktion ist korrekt, d.h., F ist genau dann erfüllbar, wenn in $\gamma(F)$ die Literale x und $\neg x$ in verschiedenen starkten Komponenten liegen:

($\Rightarrow$) Wenn eine Belegung der Variablen existiert, die jeder Klausel $(m \vee \ell)$ den Wert 1 zuordnet, muß insbesondere m oder ℓ mit 1 belegt sein. Damit existiert in $\gamma(F)$ keine Kante von einem mit 1 zu einem mit 0 belegten Literal. Da je zwei Knoten einer starken Komponente auf einem gerichteten Kreis liegen, können somit die verschieden belegten Literale x und $\neg x$ nicht in derselben starken Komponente auftreten.

($\Leftarrow$) Wähle eine topologische Sortierung t des Graphen der starken Komponenten $[\ell]$ von $\gamma(F)$, ℓ Literal in F, und setze für jedes Atom p

$$\varphi(p) = \begin{cases} 1 & \text{falls } t[\neg p] < t[p] \\ 0 & \text{sonst} \end{cases}$$

Ist $(m \vee \ell)$ eine Klausel mit $\varphi(m) = 0$, dann gilt wegen der Kanten $\neg \ell \to m$ und $\neg m \to \ell$ in $\gamma(F)$

$$t[\neg \ell] \leq t[m] < t[\neg m] \leq \to [\ell]$$

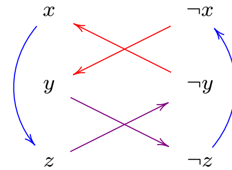
und folglich $\varphi(\ell) = 1$. □

2.3.09 Corollar. *2-*SAT $\in$ co$\boldsymbol{NL}$ □

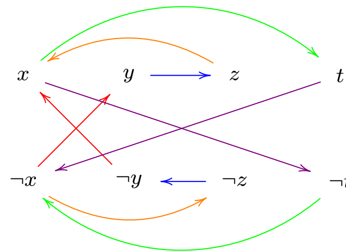
copyright: Jürgen Koslowski, TU Braunschweig, 19. Juli 2019

2.3.10 Beispiele.

▷ $F = (x \vee y) \wedge (\neg x \vee z) \wedge (\neg z \vee \neg y)$ ist erfüllbar, denn $\gamma(F)$ hat zwei starke Komponenten, die alle Atome und ihre Negationen trennen:



▷ Andererseits ist $F = (y \vee x) \wedge (\neg y \vee z) \wedge (\neg z \vee x) \wedge (\neg x \vee \neg t) \wedge (t \vee \neg x)$ nicht erfüllbar, da $\gamma(F)$ nur eine starke Komponente besitzt:



2.3.11 Lemma. *2-SAT $\in$ co**NL-H** (bzgl. $\leq^{\log}$-Reduktionen).*

Beweis.

Wir zeigen NOACPATH $\leq^{\log}$ 2-SAT.

Einer Instanz $G\#s\#t$ für NOACPATH liefert eine Formel $\Phi(G,s,t)$ in 2-KNF wie folgt: Atome sind die Knoten von G, jede Kante $u \to v$ des Graphen liefert eine Klausel $\neg u \vee v$, und dazu kommen Unit-Klauseln s und $\neg t$ für den Start- und Zielknoten.

Behauptung: $\Phi(G,s,t)$ ist genau dann erfüllbar, wenn in G kein Pfad von s nach t existiert.

Deren Beweis ist eine leichte HA, ebenso wie der Nachweis, dass $G\#s\#t \mapsto \Phi(G,s,t)$ in logarithmischem Platz berechnet werden kann. □

2.3.2 Der Satz von Immermann und Szelepcśenyi

Falls eines der bisher als co**NL**-vollständig erkannten Probleme in ***NL*-*V*** verortet werden kann, würde ***NL*** = co***NL*** folgen.

Die Frage war lange ungeklärt (die Komplexitätstheorie hat ihre Anfänge 1965), aber man glaubte, dass diese Klassen nicht übereinstimmten. Überraschenderweise konnten Neil Immerman (Professor an der University of Massachusetts Amherst) und Róbert Szelepcsényi (Student in Bratislava in der Slovakei) 1987 unabhängig voneinander zeigen, dass Gleichheit gilt. Sie wurden

copyright: Jürgen Koslowski, TU Braunschweig, 19. Juli 2019

für ihr Resultat 1995 mit dem Gödel-Preis ausgezeichnet. Der Beweis führte zudem eine wichtige neue Beweistechnik ein, das *induktive Zählen.*

2.3.12 Theorem. (Immerman and Szelepcsényi, 1987) *Für eine Platzschranke* $\mathbb{N} \xrightarrow{s} \mathbb{R}$, *die nach unten durch* log *beschränkt ist, gilt*

$$\boldsymbol{NSPACE}(s) = co\boldsymbol{NSPACE}(s)$$

2.3.13 Corollar. $\boldsymbol{NL} = co\boldsymbol{NL}$. □

Wir zeigen zunächst das folgende Theorem.

2.3.14 Satz. NOPATH $\in \boldsymbol{NL}$.

Aus diesem Ergebnis folgt der Satz von Immerman und Szelepcsényi: man wende das Theorem auf den Konfigurationsgraphen einer gegebenen TM, die oBdA nur eine akzeptierende Haltekonfiguration besitzt (vergl. Beweis von Satz 2.2.11). Eine Eingabe x wird genau dann nicht akzeptiert, wenn es keinen Pfad von der durch x spezifizierten Startkonfiguration zu der einzigen akzeptierenden Haltekonfiguration gibt.

Beweis. (von Theorem 2.3.14) Die naive Idee, alle von s aus erreichbaren Knoten aufzulisten und t auf Zugehörigkeit zu untersuchen, scheitert am Platzbedarf für diese Knotenmenge.

Ähnlich wie es im Beweis von Satz 2.2.11 wichtig war, zunächst die Anzahl der Knoten im relevanten Teilgraphen des Konfigurationsgraphen zu bestimmen, wird es hier darauf ankommen, die Anzahl N der von s aus erreichbaren Knoten zu bestimmen.

Behauptung 0: Ist N bekannt, so gibt es genau dann eine erfolgreiche Berechnung des folgenden nicht-deterministischen Algorithmus, wenn es in G keinen Pfad von s nach t gibt.

2.3.15 Algorithmus. unreach(G,s,t)

```
 1: count := 0
 2: for Knoten v do
 3: │ Rate, ob v von s aus erreichbar ist
 4: │ if Ja then
 5: │ │ Rate einen Pfad von s nach v der Länge ≤ n
 6: │ │ if Falls das geratene kein gültiger Pfad nach v ist then
 7: │ │ │ return false            // Erreichbarkeit oder Pfad falsch geraten
 8: │ │ end if
 9: │ │ if v = t then
10: │ │ │ return false            // t ist erreichbar
11: │ │ end if
12: │ │ count++                   // Erreichbarer Knoten gefunden
13: │ end if
```

14: **end for**
15: **if** $count \neq N$ **then**
16: **return FALSE** // für mindestens einen Knoten falsch geraten
17: **else**
18: **return TRUE** // immer richtig geraten und t wirklich unerreichbar
19: **end if**

Der Algorithmus kann nur dann „**TRUE**“ zurückgeben, wenn wir genau die erreichbaren Knoten als erreichbar raten:

- ▷ Wenn wir einen unerreichbaren Knoten als erreichbar raten, schlägt die Verifikation (Zeile 4 bzw. Zeile 7) fehl, egal welchen Pfad wir raten.
- ▷ Wenn wir zu wenige Knoten als erreichbar raten, schlägt die Überprüfung der Anzahl in Zeile 15 fehl.

Wenn t wirklich nicht erreichbar ist, dann gibt es eine Berechnung, nämlich diese Berechnung, die „**TRUE**“ zurückgibt.

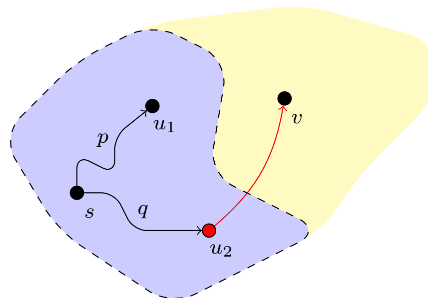
Angenommen es gibt eine Berechnung, die „**TRUE**“ zurückgibt. Dann kann t nicht erreichbar sein: Wir haben alle erreichbaren Knoten identifiziert, und t war nicht darunter, sonst hätten wir in Zeile 9/10 „false“ zurückgegeben. Damit ist Behauptung 0 beweisen.

Wir definieren nun $R(i)$ als die Anzahl der von s in $\leq i$ Schritten erreichbaren Knoten von G. Offenbar gilt $R(0) = 1$, nur s ist in 0 Schritten von s aus erreichbar, und $R(|V|) = N$.

Behauptung 1: Die Funktion R ist rekursiv in logarithmischem Platz berechenbar.

Da logarithmischer Platz i.A. nicht ausreicht, um alle Knoten zu speichern, die in $< i$ Schritten von s erreicht werden können, verwendet man wieder den obigen Trick um sicherzustellen, dass nur diese Knoten geraten werden können.

Die folgende Grafik stellt diese Idee dar.



Im blauen Bereich mögen die Knoten liegen, die in $< i$ Schritten von s aus erreichbar sind, ihre Anzahl beträgt $R(i-1)$. Bei der Überprüfung, ob v in $\leq i$ Schritten von s aus erreichbar ist, ist bei allen Knoten u zunächst nicht-deterministisch zu entscheiden, ob man sie im blauen Bereich

copyright: Jürgen Koslowski, TU Braunschweig, 19. Juli 2019

vermutet. Falls Ja, ist ein Pfad der Länge $< i$ von s nach u zu raten. Ist der Pfad korrekt, stellt sich die Frage, ob $u = v$ oder $\langle u, v\rangle \in E$. Bei einem Testknoten u_o im gelben Bereich wird man keinen hinreichend kurzen Pfad von s aus finden. Im Fall von u_1 und u_2 existieren derartige Pfade, aber v ist kein Nachfolger von u_1. Spätestens nachdem man $R(i-1)$-mal erfolgreich einen Pfad der Länge $< i$ geraten hat, kann man die Untersuchung von v beenden.

2.3.16 Algorithmus. #reach(G,s)

```
 1: R(0) = 1                                   // Nur s selbst erreichbar in 0 Schritten.
 2: for i = 1,...,n do
 3:   R(i) := 0                                                      // Initialisierung
 4:   for alle Knoten v do
 5:     // Wir wollen überprüfen, ob v in ≤ i Schritten erreichbar ist.
 6:     // Wir finden alle Knoten u, die in ≤ i − 1 Schritten erreichbar sind
 7:     // und überprüfen, ob v Nachfolger ist.
 8:     count := 0
 9:     for alle Knoten u do
10:       Entscheide, ob u weiter bearbeitet werden soll
11:       if Ja then
12:         Rate eine Knotenfolge von s nach u der Länge ≤ i − 1
13:         if Falls dies kein gültiger Pfad nach u ist then
14:           return false                   // falsch entschieden oder geraten
15:         end if
16:         count++
17:         if u = v oder ⟨u, v⟩ ∈ E then
18:           R(i)++
19:           goto nächste Iteration von v-Schleife
20:         end if
21:       end if
22:     end for
23:     if count ≠ R(i − 1) then
24:       return false          // Knoten falsch als unerreichbar geraten für ein u
25:     end if
26:   end for
27: end for
28: return R(n)
```

Die Korrektheit von **#reach(G,s)** beweist man mittels Induktion über i.

Induktionsanfang: Der Fall $i = 0$ ist klar.

Annahme: Für $i-1$ möge die Behauptung stimmen.

Induktionsschritt: Die Berechnung liefert genau dann nicht „false“, wenn in jedem Durchlauf

copyright: Jürgen Koslowski, TU Braunschweig, 19. Juli 2019

genau die in $< i$ Schritten von s aus erreichbaren Knoten und korrekte Pfade zu ihnen geraten werden. Der temporäre Wert für $R(i)$ wird nur dann erhöht, wenn v wirklich mit einem der obigen Knoten übereinstimmt oder sein Nachfolger ist.

Der finale Algorithmus berechnet zunächst $N := \#$**reach(G,s)** und mit diesem Wert **unreach(G,s,t)**.

Um logarithmischen Platzbedarf zu garantieren, ist beim Raten der Pfade ähnlich vorzugehen, wie im Beweis von Lemma 2.2.01: anstatt den Pfad in Gänze zu raten, braucht immer nur die aktuell letzte Kante betrachtet zu werden. Insofern sind immer nur zwei Knoten zu speichern, bzw. der schon berechnete Wert $R(i-1)$ und der momentane Wert von $R(i)$. □

2.4 *P*

Wir wenden uns nun den Problemen zu, die gemeinhin hinsichtlich ihres Zeitbedarfs als effizient lösbar gelten. Wir beginnen mit der Klasse ***P***, also der Klasse der Probleme, die durch eine dTM in polynomialer Zeit gelöst werden können.

Üblicherweise lassen sich Probleme in ***P*** als Auswertungen oder Überprüfungen formulieren. Dies beinhaltet das Prüfen der Korrektheit von Beweisen oder die Evaluation einer Funktionen an einem bestimmten Wert. Konkret werden wir ein erstes ***P***-vollständiges Problem betrachten. Der Beweis der Vollständigkeit geht auf Ladner zurück.

2.4.1 Das Circuit-Value-Problem

2.4.00 Definition. Unter einem **Booleschen Schaltkreis (Circuit)** der Länge $n+1$ versteht ein Sequenz von endlich vielen Zuweisungen (oder *Gattern*) der Form

$$P_k = 0 \,|\, 1 \,|\, P_i \vee P_j \,|\, P_i \wedge P_j \,|\, \neg P_i,$$

mit $i, j < k < n$. OBdA mögen die nullären vor allen unären und binären Zuweisungen auftreten. Jedes P_k darf nur einmal definiert werden, also auf der linken Seite einer Zuweisung stehen. $\lhd$

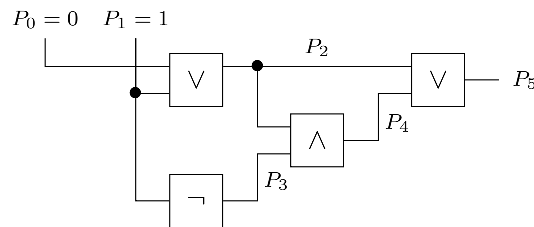
Natürlich könnten auch weitere Junktoren wie $\to$ und $\leftrightarrow$ berücksichtigt werden, aber in der Praxis begnügt man sich zumeist mit den hier angegebenen. Aufgrund der nullären Zuweisungen kann man auch von *bewerteten Schaltkreisen* sprechen. Wir werden später auch *unbewertete* Schaltkreise betrachten.

2.4.01 Beispiel. Betrachte den Schaltkreis, der durch folgende Zuweisungen gegeben ist.

$$C: \quad \begin{array}{ll} P_0 = 0, & P_3 = \neg P_1, \\ P_1 = 1, & P_4 = P_3 \wedge P_2, \\ P_2 = P_1 \vee P_0, & P_5 = P_2 \vee P_4. \end{array}$$

copyright: Jürgen Koslowski, TU Braunschweig, 19. Juli 2019

Wir können Schaltkreise als gerichtete Graphen auffassen. Dabei sind die P_k, welchen ein Wert 0 oder 1 zugewiesen wird, die Eingabesignale. Die anderen P_k stellen Ausgabesignale bestimmter Gatter dar, die mit den Junktoren $\neg$, $\vee$ oder $\wedge$ markiert sind und entsprechend viele Eingaben haben. Damit ein Signal an mehreren Gattern als Eingabe verwendet werden kann, sind explizite Verzweigungen möglich (die wir gegenüber mehreren Ausgaben bevorzugen). Dies erinnert dann an Schaltkreise aus der Elektrotechnik. Für obiges Beispiel ergibt sich der folgende Graph:

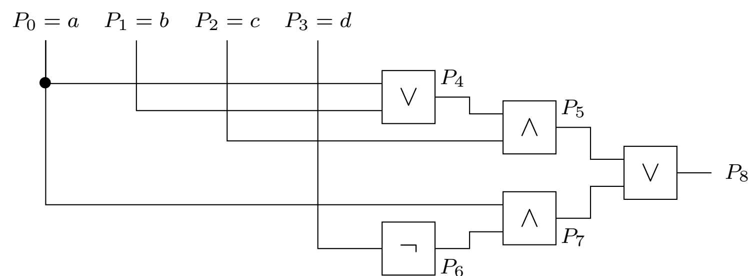


Aussagenlogische Formeln liefern *symbolische* Schaltkreise: Die Eingabewerte sind zwar fest, aber unbekannt, und Verzweigungen dürfen nur bei den ursprünglichen Eingaben erfolgen, nicht hinter einem Gatter.

2.4.02 Beispiel. Wir können die Formel $F = ((a \vee b) \wedge c) \vee (\neg d \wedge a)$ wie folgt als Schaltkreis interpretieren (bottom-up in der Baumdarstellung der Formel):

$$C: \quad \begin{array}{lll} P_0 = a, & P_3 = d, & P_6 = \neg P_3, \\ P_1 = b, & P_4 = P_0 \vee P_1, & P_7 = P_6 \wedge P_0, \\ P_2 = c, & P_5 = P_4 \wedge P_2, & P_8 = P_5 \vee P_7. \end{array}$$

Im zugehörigen Graphen gibt es hinter den Gattern keine Verzweigungen mehr:



Löste man die Verzweigungen vor den Gattern auf, erhielte man einen Baum. Jede Belegung φ der Atome a, b, c und d mit Werten aus $\mathbb{B} = \{0, 1\}$ wandelt den symbolischen Schaltkreis in einen tatsächlichen Schaltkreis um.

Belegte aussagenlogische Formeln sind also spezielle Schaltkreise. Die Berechnung von P_8 entspricht der Auswertung von $\hat{\varphi}(F)$. Das Problem, ob $\hat{\varphi}(F)$ den Wert 1 liefert, gehört zu $\boldsymbol{L}$, während das Auswerten beliebiger Schaltkreise sich als $\boldsymbol{P}$-vollständig herausstellen wird, bzgl.

copyright: Jürgen Koslowski, TU Braunschweig, 19. Juli 2019

$\leq^{\log}$. Wir haben es also mit folgenden Problemen zu tun:

Auswertung belegter aussagenlogischer Formeln (BOOLEVAL) **Gegeben:** aussagenlogische Formel F und dafür „passende“ partielle Belegung φ ; **zu entscheiden:** ob $\hat{\varphi}(F) = 1$ gilt

„Passend“ bedeutet hier, dass φ zumindest auf den in F auftretenden Atomen definiert ist.

Circuit Value Problem (CVP) **Gegeben:** Boole'scher Schaltkreis C als Liste von Zuweisungen an P_i , $i < n + 1$ **zu entscheiden:** ob $P_n = 1$ gilt

2.4.03 Beispiel. Die Auswertung in Beispiel 2.4.01 erfolgt in der Reihenfolge der Indizes: aus den gegebenen Werten $P_0 = 0$ und $P_1 = 1$ erhält man rekursiv:

$$P_2 = 1 \quad , \quad P_3 = 0 \quad , \quad P_4 = 0 \quad , \quad P_5 = 1$$

2.4.04 Satz. (Richard E. Ladner, 1975) CVP *ist* ***P****-vollständig (bezüglich* $\leq^{\log}$ *).*

Wie üblich müssen wir CVP $\in$ ***P*** und CVP $\in$ ***P-H*** zeigen. Die erste Aufgabe löst man durch Konstruktion einer dTM, aber im zweiten Fall steht noch kein ***P***-vollständiges Problem zur Verfügung, das man auf CVP reduzieren könnte. Daher muß an dieser Stelle jedes ***P***-Problem auf CVP reduziert werden.

2.4.05 Lemma. CVP $\in$ ***P***

Beweis. Wir nehmen zunächst vereinfachend an, dass ganze Zuweisungen auf den Feldern des Bandes stehen können. Dann sollen in der Startkonfiguration die Zuweisungen P_i , $i < n + 1$, getrennt durch Blank-Zeichen auf dem Band stehen. Bei der ersten unausgewerteten Zuweisung P_k von links sind ein oder zwei Indizes $< k$ der Bestandteile zu extrahieren, die zugehörigen Werte von den passenden Feldern links der aktuellen Position auszulesen und auf bis zu zwei Hilfsbändern zur Auswertung bereitzustellen. Der Zeitbedarf liegt in $\mathcal{O}(n^2)$.

Nun wollen wir zu im Wesentlichen binär codierten Zuweisungen übergehen, etwa

$$P_t = \neg P_s \mapsto \text{bin}(t) = \neg\, \text{bin}(s) \quad \text{und} \quad P_t = P_r \star P_s \mapsto \text{bin}(t) = \text{bin}(r) \star \text{bin}(s)$$

mit $r, s < t$ und $\star \in \{\wedge, \vee\}$ sowie expliziter Verwendung der Junktoren und des Gleichheitszeichens auf dem Band. Beim Auswerten ist die rechte Seite der entsprechenden Gleichung durch 0 oder 1 gefolgt von Trennzeichen # zu überschreiben. Die Laufwege des Kopfes wachsen proportional zu der Verlängerung der Eingabe, insofern ändert sich nichts an der quadratischen Zeitkomplexität. □

copyright: Jürgen Koslowski, TU Braunschweig, 19. Juli 2019

2.4.06 Bemerkung. Ein erster Versuch, alle Zuweisungen mit konstantem Platz aufzuschreiben, aber ohne die Angabe von $\mathrm{bin}(t)$, hat sich als nicht zielführend erwiesen.

2.4.07 Satz. CVP *ist* $\boldsymbol{P}$*-hart bzgl.* $\leq^{\log}$.

Beweis.

Wir betrachten ein Problem $A \in \boldsymbol{P}$, das von einer dTM $M = \langle Q, \Sigma, \Delta, q_0, q_{\text{acc}}, q_{\text{rej}} \rangle$ in polynominaler Zeit entschieden wird, etwa mit Zeitschranke $t(n) = n^c$ wobei c konstant ist. OBdA mögen für M dieselben Vereinbarungen gelten wie zu Beginn von Unterabschnitt 1.2.1 festgelegt.

Zur Berechnung von $x \in \Sigma^n$, sind höchstens n^c Schritte nötig; da nach unserer Definition einer TM in Haltezuständen noch Leerlauf möglich ist (Verharren des Kopfes auf derselben Position im selben Haltezustand unter Beibehaltung des Bandsymbols), können wir die Berechnung auch auf genau n^c Schritte ausdehnen.

Steht der Kopf zu Beginn auf Position -1 (Randmarkierung), so ist in n^c Schritten maximal Position $n^c - 1$ erreichbar. Die Konfigurationen im Verlauf der Berechnung entsprechen also bijektiv Wörtern aus $\Delta^* \times (Q \times \Delta) \times \Delta^*$ der Gesamtlänge $n^c + 1$, die den Bandinhalt dieses Bereichs und den Zustand an der Position des Kopfes darstellen. Von Schritt i zu Schritt $i+1$ verändert sich ein derartiges Wort aber an höchstens zwei Stellen, der alten Kopfposition und einem von ihren Nachbarn: indem evtl. ein neues Bandsymbol aus Δ geschrieben wird, und sich evtl. der Kopf bewegt und potentiell seinen Zustand ändert.

Á priori braucht das Band nach Ende der Berechnung nicht leer zu sein, und wir müssen die Position des Kopfes relativ zu seiner Startposition nicht kennen.

Für die zu konstruierende Instanz von CVP führen wir folgende Atome ein:

- ▷ $P^a_{i,j}$ möge genau dann wahr sein, wenn in Schritt $i \leq n^c$ an Position $-n^c \leq j \leq n^c$ das Symbol $a \in \Delta$ steht;
- ▷ $Q^p_{i,j}$ möge genau dann wahr sein, wenn in Schritt $i \leq n^c$ der Kopf an Position $-n^c \leq j \leq n^c$ steht und den Zustand $p \in Q$ hat;
- ▷ Q^{qcc} möge genau dann wahr sein, wenn $x \in \mathcal{L}(M)$ gilt.

Die Initialkonfiguration mit Eingabe x liefert damit:

- ▷ $P^{x_j}_{0,j} = 1$ und $P^a_{0,j} = 0$ sofern $a \neq x_j$ für $0 \leq j < n$;
- ▷ $P^{\sqcup}_{0,j} = 1$ und $P^a_{0,j} = 0$ sofern $a \neq \sqcup$ für $-n^c \leq j < 0$ und $n \leq j \leq n^c$;
- ▷ $Q^{q_o}_{i,0} = 1$ und $Q^p_{i,0} = 0$ sofern $p \neq q_0$, und $Q^p_{i,j} = 0$ falls $j \neq 0$ und $p \in Q$.

copyright: Jürgen Koslowski, TU Braunschweig, 19. Juli 2019

In der Form

$$P\,\mathrm{bin}(i) P\,\mathrm{bin}(j) P\,\mathrm{bin}\, a = b \quad \text{und} \quad Q\,\mathrm{bin}(i) Q\,\mathrm{bin}(j) Q\,\mathrm{bin}\, p = b$$

mit $b \in \{0,1\}$ lassen sich diese Zuweisungen auf das Ausgabeband des Berechners M_f der Reduktion schreiben, getrennt durch $\#$-Zeichen. Das erfordert nur konstanten Platz auf den Arbeitsbändern.

Sofern die Konfiguration in Schritt i schon in Zuweisungen umgewandelt wurde, erhalten wir in Schritt $i+1$ für den Bandinhalt

$$P^b_{i+1,j} = \bigvee_{\delta(p,a)=(q,b,X)} \left(Q^p_{i,j} \wedge P^a_{i,j} \right) \vee \left(P^b_{i,j} \wedge \bigwedge_{p \in Q} \neg Q^p_{i,j} \right)$$

mit $X \in \{L,N,R\}$, für jedes $b \in \Delta$, denn entweder schreibt M in Schritt i an Position j ein b, oder in dort stand schon ein b und M schreibt nicht an dieser Position. Aber diese Zuweisungen haben noch nicht die verlangte Form. Daher führen wir Hilfsvariable mittels binärer Zuweisungen ein

$$R^{p,a}_{i,j} = Q^p_{i,j} \wedge P^a_{i,j} \quad \text{und} \quad S^{p,a}_{i,j} = \neg Q^p_{i,j} \wedge P^a_{i,j}$$

DA der Berechner der Reduktion über den Code der Maschine M verfügen muß, können wir ebenso annehmen, dass neben der Zustandsmenge Q auch die Mengen der Übergänge

$$T_b := \{\, \langle p, a, q, X \rangle \,:\, \delta(p,q) = (q, b, X) \,\}$$

Nun gilt

$$P^b_{i,j} \wedge \bigwedge_{p \in Q} \neg Q^p_{i,j} = \bigwedge_{p \in Q} S^{p,a}_{i,j} = \left(\left(\left(S^{0,a}_{i,j} \wedge S^{1,a}_{i,j} \right) \wedge S^{2,a}_{i,j} \right) \wedge \ldots \right) \wedge S^{|Q|-1,a}_{i,j}$$

Auf ähnliche Weise mit Hilfe einer linearen Ordnung auf T_b lässt sich das linke Argument der Disjunktion auf binäre Disjunktionen zurückführen.

Für Kopfposition und Zustand ehalten wir

$$\begin{aligned} Q^q_{i+1,j} &= \bigvee_{\delta(p,a)=(q,b,L)} \left(Q^p_{i,j+1} \wedge P^a_{i,j+1} \right) \vee \bigvee_{\delta(p,a)=(q,b,N)} \left(Q^p_{i,j} \wedge P^a_{i,j} \right) \vee \\ &\quad \bigvee_{\delta(p,a)=(q,b,R)} \left(Q^p_{i,j-1} \wedge P^a_{i,j-1} \right) \\ &= \bigvee_{\delta(p,a)=(q,b,L)} R^{p,q}_{i,j+1} \vee \bigvee_{\delta(p,a)=(q,b,N)} R^{p,q}_{i,j} \vee \bigvee_{\delta(p,a)=(q,b,R)} R^{p,q}_{i,j-1} \end{aligned}$$

für jedes $p \in Q$, denn entweder bewegt sich der Kopf aus seiner Position in Schritt i nach rechts, gar nicht, oder nach links.

copyright: Jürgen Koslowski, TU Braunschweig, 19. Juli 2019

Der Berechner möge oBdA auch folgende Mengen kennen:

$$T_{q,X} := \{ \langle p, a, b \rangle : \delta(p, a) = (q, b, X) \} \text{ für } X \in \{L, N, R\}$$

Diese ermöglichen es, auch $Q^q_{i+1,j}$ auf binäre Zusweisungen zurückzuführen.

Schließlich brauchen wir eine letzte entscheidende Zuweisung

$$Q^{\mathrm{acc}} = \bigvee_{-n^c \le j \le n^c} Q^{q_{\mathrm{acc}}}_{n^c,j}$$

Die Umformung in binäre Zuweisungen erfolgt auch hier nach dem obigen Schema. □

Zusammenfassend gilt also:

$$\boldsymbol{L} \ni \text{BOOLEVAL} \subseteq \text{CVP} \in \boldsymbol{NP\text{-}H} \tag{2.4-00}$$

2.4.08 Bemerkung. Ein berühmtes und in der Praxis (Cryptographie) wichtiges Problem ist

Primalität (PRIMES)
Gegeben: eine natürliche Zahl $n > 1$.
zu entscheiden: ist n eine Primzahl?

Es wurde 2002 von Manindra Agrawal, Neeraj Kayalu und Nitin Saxena vom *Indian Institute of Technology Kanpur* als zu $\boldsymbol{P}$ gehörig nachgewiesen. Die ursprüngliche obere Schranke $\tilde{\mathcal{O}}\left(\log(n)^{12}\right)$ für die Laufzeit konnte 2005 von Carl Pomerance und H. W. Lenstra, Jr auf $\tilde{\mathcal{O}}\left(\log(n)^{6}\right)$ reduziert werden.

Praktische Primalitätstests, die entweder nicht deterministisch, oder nicht polynomiell zeitbeschränkt oder nicht allgemeingültig sind, haben bisher i.A. immer noch eine viel kürzere Laufzeit.

Leider liefert der Algorithmus im negativen Fall keine Aussagen über mögliche Primfaktoren.

2.5 *NP*

Im Hinblick auf 2.4-00 sollte das Erfüllbarkeitsproblem für Boole'sche Formeln höchstens so schwierig sein wie das Erfüllbarkeitsproblem für Boolesche Schaltkreise ohne Bewertung, d.h., im Wesentlichen formale Schaltkreise.

Interessanterweise gilt hier sogar die umgekehrte Relation bzgl. $\le^{\mathrm{poly}}$. Zunächst müssen wir aber das Erfüllbarkeitsproblem für unbewertete Boole'sche Schaltkreise genauer formulieren:

Circuit Value Problem mit variablem Input (CVPVI)
Gegeben: Schaltkreis C der Länge $n+1$ mit speziellen Zuweisungen $P_i = ?$, $i < k$.
zu entscheiden: ob ein Binärtupel $y \in \{0,1\}^k$ existiert, so dass die Substitution von y_i für ? in $P_i = ?$, $i < k$, ein CVP $C(y)$ mit $P_n = 1$ liefert.

copyright: Jürgen Koslowski, TU Braunschweig, 19. Juli 2019

2.5.00 Proposition. CVPVI $\leq^{\text{poly}}$ SAT

Beweis. Für eine CVPVI-Instanz C der Länge $n+1$ mit speziellen Zuweisungen $P_i = ?$, $i < k$, verwendet F_C die Variablen x_i , $i < k$. Die übrigen Zuweisung, mit von ? verschiedener rechter Seite (einschließlich $\to$), liefern folgende Klauseln, unter Verwendung der Mengenschreibweise

$$\begin{aligned} P_i = \neg P_j &\mapsto (x_i \leftrightarrow \neg x_j) \boxminus \{\neg x_i, \neg x_j\}\{x_j, x_i\} \\ P_i = P_j \wedge P_k &\mapsto (x_i \leftrightarrow x_j \wedge x_k) \boxminus \{\neg x_i, x_j\}\{\neg x_i, x_k\}\{\neg x_j, \neg x_k, x_i\} \\ P_i = P_j \vee P_k &\mapsto (x_i \leftrightarrow x_j \vee x_k) \boxminus \{\neg x_i, x_j, x_k\}\{\neg x_j, x_i\}\{\neg x_k, x_i\} \\ P_i = P_j \to P_k &\mapsto (x_i \leftrightarrow \neg x_j \vee x_k) \boxminus \{\neg x_i, \neg x_j, x_k\}\{x_j, x_i\}\{\neg x_k, x_i\} \end{aligned}$$

F_C besteht nun aus der Konjunktion dieser Formeln in KNF mit der Unit-Klausel x_n . Diese sogenannte *Tseytin-Transformation* (G. S. Tseytin, 1966) erfordert offenbar polynomiale Zeit, und liefert eine zu F erfüllbarkeitsäquivalente Formel F_C , d.h., F_C ist genau dann erfüllbar, wenn $y \in \{0,1\}^k$ existiert, so dass in $C(y)$ gilt $P_n = 1$. □

2.5.01 Beispiel. Der unbewertete Schaltkreis

$$\begin{aligned} C:\ & P_0 = ?, \\ & P_1 = ?, \\ & P_2 = P_0 \vee P_1, \\ & P_3 = \neg P_1, \\ & P_4 = P_2 \wedge P_3 \end{aligned}$$

liefert die Formel

$$\begin{aligned} F_C &= (x_2 \leftrightarrow x_0 \vee x_1) \wedge (x_3 \leftrightarrow \neg x_1) \wedge (x_4 \leftrightarrow x_2 \wedge x_3) \wedge x_4 \\ &= \{\neg x_2, x_0, x_1\}\{\neg x_0, x_2\}\{\neg x_1, x_2\}\{\neg x_3, \neg x_1\}\{x_3, x_1\}\{\neg x_4, x_2\}\{\neg x_4, x_3\}\{\neg x_2, \neg x_3, x_4\}\{x_4\} \end{aligned}$$

Dann ist $C(1,0)$ eine CVP-Instanz, und die partielle Belegung $\varphi(x_0) = 1$ und $\varphi(x_1) = 0$ läßt sich zu einer erfüllenden Belegung von F_C erweitern vermöge $\varphi(x_2) = \varphi(x_3) = \varphi(x_4) = 1$. ◁

2.5.02 Bemerkung. Die Tseytin-Transformation erlaubt die Umwandlung beliebiger aussagenlogischer Formeln in erfüllbarkeitsäquivalente KNF-Formeln in polynomialer Zeit, während die Umwandlung in äquivalente KNF-Formeln mit Hilfe der De'Morganschen Regeln und eines Distributivgesetzes exponentiell viel Platz und Zeit benötigen kann. Auf diese Weise können beliebige Formeln mit SAT-Solvern behandelt werden.

2.5.1 *NP*-vollständige Probleme

2.5.03 Satz. (Cook 1971, Levin 1973) SAT *ist **NP**-vollständig.*

copyright: Jürgen Koslowski, TU Braunschweig, 19. Juli 2019

Beweis.

Um die Zugehörigkeit zu ***NP*** nachzuweisen, konstruiert man eine nTM M, die SAT in polynomialer Zeit akzeptiert. M findet die auftretenden Atome und rät nichtdeterministisch eine Belegung. Die resultierende Instanz von BOOLEVAL läßt sich nun in polynomialer Zeit lösen.

Behauptung: SAT ist ***NP***-hart.

Für ein ***NP***Problem A, das von einer nTM akzeptiert wird, die n^c-zeitbeschränkt ist, skizzieren wir eine Reduktion $A \leq^{\text{poly}} \text{SAT}$. OBdA können wir annehmen, dass der Konfigurationsgraph von A strikt binär ist, d.h., jedes Binärwort $y \in \{0,1\}^{n^c}$ beschreibt eine mögliche Berechnung für eine Eingabe x der Länge n.

Für die Eingabe x mit $|x| = n$ möge die dTM $M^{(x)}$ Eingaben der Form $y \in \{0,1\}^*$ mit $|y| = n^c$ erwarten und dann M auf x entlang der Vorgabe y deterministisch simulieren. Dann akzeptiert M die Eingabe x genau dann, wenn ein $y \in \{0,1\}^{n^c}$ existiert, das von $M^{(x)}$ akzeptiert wird.

Nach dem Satz von Ladner gibt es einen Schaltkreis $C^{(x)}(y)$, der die Berechnung von $M^{(x)}$ auf y modelliert, d.h., $M^{(x)}$ akzeptiert y genau dann, wenn $C^{(x)}(y)$ eine positive CVP-Instanz ist. Ersetzt man die Zuweisungen $P_i = y_i$ durch $P_i = ?$, $i < n^c$, erhalten wir eine CVPVI-Instanz $C^{(x)}$, so dass M genau dann x akzeptiert, wenn ein $y \in \{0,1\}^{n^c}$ existiert, so dass $C^{(x)}(y)$ positiv ist.

Gemäß Proposition 2.5.00 ist die Formel $F_{C^{(x)}}$ genau dann erfüllbar, wenn M die Eingabe x akzeptiert. □

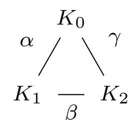
Wir betrachten nun ein wichtiges graphentheoretisches Problem

Hamilton'scher Kreis (HC)
Gegeben: Ungerichteter Graph $G = \langle V, E \rangle$
zu entscheiden: Gibt es einen Rundweg, der alle Knoten genau einmal berührt?

2.5.04 Satz. HC *ist* ***NP****-vollständig.*

Beweis. Die folgende Lösung basiert auf dem Beweis von Theorem 9.7 in [Pap94]:

Idee: die Literale einer Boole'schen Formel F in 3-KNF sollen auf bestimmte Kanten eines Graphen $R(F)$ abgebildet werden. Und da in 3-SAT jede Klausel $K = (\alpha \vee \beta \vee \gamma)$ aus genau drei Literalen besteht, sollen die zugehörigen Kanten ein ausgezeichnetes Dreieck $\Delta(K)$ bilden:



(2.5-00)

Bei c Klauseln liefert dies $3c$ Knoten.

Nun wollen wir erreichen, daß in jedem der Dreiecke $\Delta(K)$ Kanten genau dann zu einem Hamilton'schen Kreis gehören, wenn sie mit 0 bewertet werden. Insbesondere können dann nicht alle drei Seiten von $\Delta(K)$ vorkommen.

Zu diesem Zweck ordnen wir in einer Vorstufe $\bar{R}(F)$ von $R(F)$ jeder in F vorkommenden Variablen x_i, $i < n$, zusätzlich zwei parallele(!) Kanten zu, die die Belegungen von x_i mit 1 bzw. 0 repräsentieren sollen. Dies erfordert $n+1$ zusätzliche Knoten, die wir einfach mit $i \leq n$ durchnummerieren:

$$i \underset{x_i \text{ false}}{\overset{x_i \textbf{ TRUE}}{=\!=\!=}} i+1 \tag{2.5-01}$$

Das Problem paralleler Kanten löst sich im Folgenden automatisch, sobald sie durch weitere Hilfsknoten aufgeteilt werden. In $\bar{R}(F)$ bilden alle Knoten der ausgezeichneten Dreiecke $\Delta(K)$ mit den Knoten 0 und n sowie zwei weiteren Knoten A und B eine Clique mit $3c+4$ Knoten.

Wird x_i mit $i < n$ mit 1 belegt, so wollen wir die entsprechende Kante von i nach $i+1$ für einen Hamilton'schen Kreis verwenden; dieser darf dann keine der Kanten für das Literal x in den ausgezeichneten Dreiecken enthalten. Entsprechend darf die andere Kante von i nach $i+1$ nicht in dem Hamilton'schen Kreis auftreten, während alle Kanten für das Literal $\neg x_i$ in den ausgezeichneten Dreiecken dazugehören sollen. Bei einer Belegung von x_i mit 0 ist entsprechend umgekehrt zu verfahren: genau die mit 0 bezeichnete Kante von i nach $i+1$ soll zum Hamilton'schen Kreis gehören, ebenso jede x_i entsprechende Kante in den ausgezeichneten Dreiecken.

Wir haben somit bestimmte Paare von disjunkten Kanten, von denen wir sicherstellen wollen, dass genau eine von ihnen zu einem Hamilton'schen Kreis gehören kann.

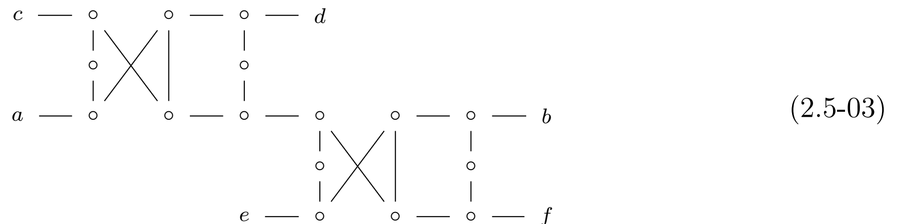
Folgende Hilfskonstruktion setzt diese Nebenbedingung für disjunkte Kanten um, indem diese durch Hilfsknoten aufgeteilt werden:

$$\begin{array}{ccc} c \text{ ——— } d & & c \text{ — } \circ \quad \circ \text{ — } \circ \text{ — } d \\ & \mapsto & \vert \;\; \times \;\; \vert \quad\;\; \circ \\ a \text{ ——— } b & & a \text{ — } \circ \quad \circ \text{ — } \circ \text{ — } b \end{array} \tag{2.5-02}$$

Ist der rechte Graph Teil eines Graphen G, wobei von den nicht bezeichneten Hilfsknoten keine weiteren Kanten ausgehen, so gibt es nur zwei Möglichkeiten, wie ein Hamilton'scher Kreis für G diesen Teilgraphen durchlaufen kann: mäandernd zwischen a und b, oder mäandernd zwischen c und d. Jede Abweichung von einem dieser Wege führt zwangsläufig dazu, dass ein Knoten ausgelassen wird. (Man beachte, dass sechs Hilfsknoten und zwei vertikale Verbindungen nicht ausreichen, um diese Verhalten zu garantieren.) Enthält umgekehrt ein Graph H disjunkte Kanten $\{a,b\}$ und $\{c,d\}$, so kann man sie durch den rechten Graphen ersetzen und somit erreichen, daß ein Hamilton'scher Kreis im derart modifizierten Graphen H' entweder den mäandernden Weg zwischen a und b, entsprechend der Kante $\{a,b\}$ in H, oder den mäanadernden Weg zwischen c und d, entsprechend der Kante $\{c,d\}$ in H enthält.

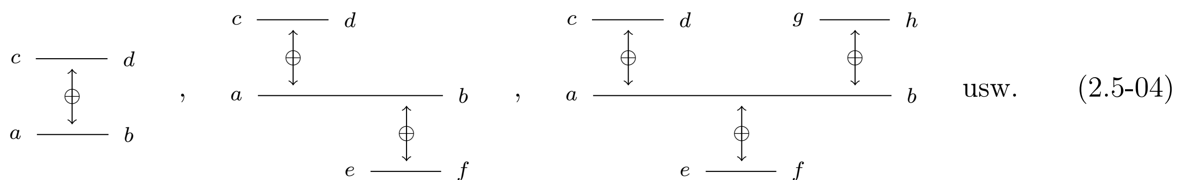
copyright: Jürgen Koslowski, TU Braunschweig, 19. Juli 2019

Sollen im Graphen H entweder $\{a,b\}$ oder die beiden Kanten $\{c,d\}$ und $\{e,f\}$ in einem Hamilton'schen Kreis liegen (alle Kanten paarweise disjunkt), so läßt sich das z.B. mit folgendem Hilfsgraphen realisieren



Analog kann man verfahren, wenn die Alternative zur Kante $\{a,b\}$ aus mehr als zwei Kanten besteht.

Der Übersichtlichkeit halber werden wir diese Konstruktion in $R(F)$ durch folgende Markierung der zu verbindenden Kanten im Graphen $\bar{R}(F)$ kenntlich machen:



Um $R(F)$ zu erhalten verbinden wir alle dem Literal x_i entsprechenden Kanten der ausgezeichneten Dreiecke mittels der obigen Konstruktion (2.5-04) mit der Kante $x_i\,\boldsymbol{true}$ von i nach $i+1$, und alle dem Literal $\neg x_i$ entsprechenden Kanten der ausgezeichneten Dreiecke mit der Kante $x_i\,\boldsymbol{false}$ von i nach $i+1$. Da mindestens eine der ursprünglich parallelen Kanten von i nach $i+1$ gemäß (2.5-04) mit mindestens einer ausgezeichneten Dreieckskante verbunden ist, hat $R(F)$ keine parallelen Kanten mehr.

$\bar{R}(F)$ besitzt $3c+n+3$ Knoten, und jede Kante der ausgezeichneten Dreiecke wird mittels 8 Hilfsknoten mit einer der Kanten aus (2.5-01) verbunden. $R(F)$ besitzt daher $27c+n+3$ Knoten. Offenbar kann $R(F)$ in polynomialer Zeit in $n+c$ konstruiert werden.

Korrektheit der Reduktion: Die graphische Abkürzung (2.5-04) erlaubt uns, im Wesentlichen im Graphen $\bar{R}(F)$ zu argumentieren, der übersichtlicher ist als $R(F)$. Jeder Hamilton'sche Kreis C in $R(F)$ induziert zwei Pfade C_0 und C_1 von 0 nach n, von denen C_0 alle Knoten 1 $\dots n-1$ enthält, während in C_1 neben A und B ausschließlich Knoten der ursprünglichen Dreiecke vorkommen. Der Verlauf von C_0 bestimmt dabei einen Wahrheitswert für die Variablen x_i, $i<n$, entsprechend der ursprünglichen parallelen Kanten von i nach $i+1$. Dies schließt aus, dass die gemäß (2.5-04) mit dieser verbundenen und mit 1 bewerteten Literal-Kanten der ausgezeichneten Dreiecke in C vorkommen, während die mit 0 bewerteten Literal-Kanten dieser Dreiecke in C vorkommen müssen. Aber in keinem ausgezeichneten Dreieck können alle drei Kanten zu C gehören, sonst würde ein Knoten mehrfach auftreten. Folglich muß mindestens ein Literal pro Klausel den Wert 1 haben, und somit ist F erfüllbar.

copyright: Jürgen Koslowski, TU Braunschweig, 19. Juli 2019

Ist umgekehrt F erfüllbar, etwa mit einer Belegung $\{ x_i : i < n \} \xrightarrow{\varphi} \mathbb{B} = \{0,1\}$ der Variablen, so müssen wir einen Hamilton'schen Kreis in $R(F)$ konstruieren. Von 0 nach n durchlaufen wir die durch φ bestimmten Kanten der Vorstufe von $R(F)$, was in den ausgezeichneten Dreiecken Pfade der Länge < 3 auswählt (entweder zwei mit 0 belegte Kanten, oder eine mit 0 belegte Kante und den leeren Pfad am gegenüberliegenden Knoten). Da ansonsten alle Dreiecksknoten untereinander sowie mit 0, n, A und B verbunden sind, können diese unabhängigen Pfade zu einem Pfad von n nach 0 zusammengefügt werden. Eine einzelne Klausel aus drei positiven Literalen zeigt, warum die Hilfsknoten A und B nötig sind (vergl. HA). □

Das folgende Problem ist die Entscheidungsvariante eines Optimierungsproblems:

Travelling Salesman Problem (E-TSP)
Gegeben: Zahlen $n, K \in \mathbb{N}$ und eine $n \times n$-Matrix C mit Einträgen aus $\mathbb{N} + \{\infty\}$
zu entscheiden: Gibt es eine Permutation π auf n, mit $\sum_{i<n} C\big(\pi(i), \pi(i+1 \bmod n)\big) < K$?

2.5.05 Corollar. E-TSP *ist **NP**-vollständig.*

Beweis.

Zum Nachweis der Zugehörigkeit zu ***P*** konstruieren wir eine nTM M wie folgt: M rät nichtdeterministisch eine Permutation π der Menge $n = \{0, 1, \ldots, n-1\}$. Anschließend werden Kosten zwischen den dadurch bestimmten benachbarten Knoten $\pi(i)$ und $\pi(i+1 \bmod n)$ summiert und mit K verglichen.

Umgekehrt zeigen wir HC $\leq^{\text{poly}}$ E-TSP mit Hilfe folgender Reduktion: wir überführen den Graphen G mit n Knoten einer Instanz von HC in eine $n \times n$-Matrix C , wobei

$$C(i,j) = \begin{cases} 1 & \text{falls } \{i,j\} \in E \\ \infty & \text{sonst} \end{cases}$$

Was in maximal quadratischer Zeit möglich ist.

Da ein Hamilton'scher Kreis in einem Graphen mit n Knoten genau n Kanten enthält, ist die Korrektheit der Reduktion unmittelbar klar. □

2.5.2 Zertifikatssprachen

Alle bisherigen Konstruktionen nicht-deterministischer TMs folgten dem gleichen Schema: zunächst wird nicht-deterministisch eine potentielle Lösung für das zu entscheidende Problem geraten, und dann deterministisch daraufhin überprüft, ob es tatsächlich eine Lösung ist. Es ist kein Zufall, dass man sich auf eine nicht-deterministische Phase zu Beginn beschränken kann. In der Tat kann man die Sprachen in ***NP*** auf diese Weise charakterisieren.

2.5.06 Definition. Für $L \subseteq \Sigma^*$ heißt $L_{\text{check}} \subseteq \Gamma^*$ mit $\Sigma + \{\#\} \subseteq \Gamma$ *Zertifikatssprache*, falls

copyright: Jürgen Koslowski, TU Braunschweig, 19. Juli 2019

- $L_{\text{check}} \in \boldsymbol{P}$, eine entsprechende dTM heißt *Verifizierer*;
- es existiert ein Polynom q so dass für jedes $w \in \Sigma^*$ gilt

$$w \in L \quad \text{gdw} \quad w\#z \in L_{\text{check}} \quad \text{für ein Wort } z \in \Gamma^* \text{ mit } |z| \leq q(|w|)$$

Ein solches Wort z heißt *Zertifikat* für w.

2.5.07 Satz. *$\boldsymbol{NP}$ besteht genau aus den Sprachen, die Zertifikatssprachen besitzen.*

Beweis. Ein Verifizierer $\mathcal{M}_{\text{check}}$ mit Zeitkomplexität p, für L_{check}, mit Längenkomplexität q, liefert eine L akzeptieren nTM $\mathcal{M}$, die immer hält:

- B_0 enthält die Eingabe w; ihre Länge $|w|$ und $q(|w|)$ werden auf auf B_3 gespeichert, was $O(q(|w|))$ Schritte erfordert;
- auf B_1 wird nichtdeterministisch ein Wort $z \in \Gamma^*$ mit $|z| \leq q(|w|)$ erzeugt ("geraten"); dies erfordert $|z| \in O(q(|w|))$ Schritte;
- auf B_2 wird $\mathcal{M}_{\text{check}}$ mit Eingabe $w\#z$ simuliert.

$\mathcal{M}$ hält und akzeptiert genau dann, wenn die Simulation von $\mathcal{M}_{\text{check}}$ dies tut und hat Zeitkomplexität in $O(p(q(n)))$.

Umgekehrt akzeptiere die nTM $\mathcal{M}$ die Sprache L mit Zeitkomplexität t. Der $\boldsymbol{M}$-Konfigurationssbaum mit der Initialkonfiguration für w als Wurzel hat eine Tiefe $\leq t(|w|)$. Die maximale Verzweigungszahl sei $r \in \mathbb{N}$; oBdA können wir annehmen, für jeden Übergang gäbe es r Alternativen. Jeder Weg von der Initialkonfiguration zu einer Haltekonfiguration, also jede Berechnung, kann nun durch ein Wort in $\{0, \ldots, r-1\}^*$ der Länge $\leq t(|w|)$ eindeutig beschrieben werden. $L_{\text{check}} \subseteq L\{\#\}r^*$ besteht dann aus den Wörtern $w\#z$ mit $w \in L$ und einer Spezifikation z für eine akzeptierende Berechnung für w in $\mathcal{M}$.

Die Maschine $\boldsymbol{M}_{\text{check}}$ simuliert bei Eingabe $w\#z$ den Ablauf der Maschine $\boldsymbol{M}$ mit Eingabe w gemäß der durch z spezifizierten Berechnung und ist somit deterministisch. Das erfordert $|z| \leq t(|w|)$ Simulationsschritte von $\boldsymbol{M}_{\text{check}}$. □

Das obige *maschinenbezogene* Zertifikat wird in der Praxis meist sehr unhandlich sein. Dort verwendet man meistens *problembezogene* Zertifikate, wie schon in den obigen Beispielen.

2.5.08 Beispiel. Das Problem

Unabhängige Menge oder **Independent Set** (UM)
Gegeben: ein ungerichteter Graph $G = \langle V, E \rangle$ und eine Zahl k
zu entscheiden: Gibt es eine Teilmenge $U \subseteq V$ mit $|U| \geq k$ die keine Kante enthält?

liegt in $\boldsymbol{NP}$. Die unabhängigen Mengen der Größe $\geq k$ selbst können als Zertifikate dienen; es sind Lösungen des Berechnungsproblems, eine unabhängige Menge mit mindestens k Elementen zu finden.

copyright: Jürgen Koslowski, TU Braunschweig, 19. Juli 2019

2.5.09 Beispiel. Das Problem

Subset Sum (SSSUM)
Gegeben: eine Menge $N = \{ n_i : i < k \} \subseteq \mathbb{N} - \{0\}$ positiver natürlicher Zahlen und $S > 0$
zu entscheiden: Gibt es eine Teilmenge $M \subseteq N$ mit $\sum M = S$?

liegt in $\boldsymbol{NP}$. Hier können die Teilmengen $M \subseteq N$ mit Summe S als Zertifikate dienen.

Die beiden gerade genannten Probleme sind sogar $\boldsymbol{NP}$-vollständig, was im ersten Fall in den HA zu zeigen ist.

2.6 *PSPACE* und der Satz von Savitch

Während hinsichtlich der Zeit-Komplexitätsklassen das Problem, ob $\boldsymbol{P}$ mit $\boldsymbol{NP}$ übereinstimmt, noch offen ist, wurde das entsprechende Platzproblem bereits 1970 gelöst:

2.6.00 Theorem. (Walter Savitch, 1970) *Ist* $\mathbb{N} \xrightarrow{s} \mathbb{N}$ *eine Platzschranke mit* $\log n \leq s(n)$ *für alle* n *. Dann gilt*

$$\boldsymbol{NSPACE}(s) \subseteq \boldsymbol{DSPACE}(s^2)$$

Damit quadriert sich der Platzbedarf einer nicht-deterministische Maschine, wenn sie determinisiert wird. Als unmittelbare Folgerungen erhalten wir

2.6.01 Corollar.

$$\boldsymbol{PSPACE} = \boldsymbol{NPSPACE} \quad \text{und} \quad \boldsymbol{EXPSPACE} = \boldsymbol{NEXPSPACE} \qquad \square$$

Beachte, dass $\boldsymbol{PSPACE}$ und $\boldsymbol{EXPSPACE}$ under Quadrierung abgeschlossen sind. Wegen $\boldsymbol{PSPACE} = \text{co}\boldsymbol{PSPACE}$ folgt auch die Übereinstimmung von $\text{co}\boldsymbol{NPSPACE}$ mit $\boldsymbol{PSPACE}$.

2.6.02 Bemerkung. Leider können wir aus dem Satz von Savitch *nicht* $\boldsymbol{L} = \boldsymbol{NL}$ folgern, da aus $s \in \mathcal{O}(\log(n))$ zwar $s^2 \in \mathcal{O}((\log(n))^2)$, aber nicht notwendig $s^2 \in \mathcal{O}(\log(n))$ folgt.

Beweis. [Satz von Savitch] Wie in Lemma 2.2.08 und in Abschnitt 2.3.2 wollen wir das Akzeptanzproblem für eine nTM M als ein Pfad-Problem in ihrem Konfigurationsgraphen auffassen. Entsprechend möge die Akzeptanz nur bei leerem Band erfolgen, denn nur dann haben wir eine eindeutige akzeptierende Konfiguration.

Behauptung: Das Problem PATH läßt sich für Graphen G mit n Knoten deterministisch mit Platzverbrauch in $\mathcal{O}((\log n)^2)$ lösen.

Gibt es in G einen Pfad von s nach t , so gibt es auch einen Pfad der Länge $\leq n$, und ein solcher läßt sich in zwei Pfade der Länge $\leq n/2$ zerlegen, von s zu einem Zwischenknoten v , und von dort nach t . Man muß also nur rekursiv überprüfen, ob für jedes $v \in V$

copyright: Jürgen Koslowski, TU Braunschweig, 19. Juli 2019

▷ ein Pfad der Länge $\leq n/2$ von s nach v existiert, und

▷ ein Pfad der Länge $\leq n/2$ von v nach t existiert.

Zu jedem Zeitpunkt sind nur drei Knoten und die gewünschte Pfadlänge zu betrachten, was $4\log n$ Bits benötigt. Diese wollen wir auf einen Stack schreiben; sonstiger Platz zur Lösung des aktuellen Problems am Ende des Stacks kann wiederverwendet werden. Da sich bei jedem Schritt die Pfadlänge halbiert, ist die Stacktiefe durch $\log n$ beschränkt. Also benötigt die Buchführung $\mathcal{O}\big((\log n)^2\big)$ viel Platz. Die Suche nach Pfaden der Länge ≤ 1 ist in konstantem Platz zu bewältigen, da nur die Adjazenzlisten des gegebenen Graphen durchsucht werden müssen. In Pseudocode etwa:

2.6.03 Algorithmus. (savitch(G, s, t, k)) Eingabe: Graph G, Startknoten s, Zielknoten t, Schranke $k \in \mathbb{N}$
Ausgabe: TRUE genau dann, wenn es in G einen Pfad von s nach t der Länge $\leq k$ gibt

```
 1: if k = 0 then
 2: |  return (s = t)
 3: end if
 4: if k = 1 then
 5: |  return (s → t)
 6: end if
 7: if k > 1 then
 8: |  for v Knoten von G do
 9: |  |  if savitch(G, s, v, ⌈k/2⌉) und savitch(G, v, t, ⌊k/2⌋) then
10: |  |  |  return TRUE
11: |  |  end if
12: |  end for
13: |  return FALSE
14: end if
```

Nun sei eine nTM $M = \langle Q, \Sigma, \Gamma, \delta, q_0, q_{\mathrm{acc}}, q_{\mathrm{rej}}\rangle$ mit Raumkomplexität $s(n) \geq \log n$ gegeben, die nur mit leerem Band akzeptiert. Für eine Eingabe x der Länge n ist die Anzahl κ der von der Initialkonfiguration aus erreichbaren Konfigurationen abschätzen durch

$$\kappa \leq |Q| \cdot |\Gamma|^{s(n)} \cdot s(n) < |Q| \cdot \left(|\Gamma|^{s(n)}\right)^2 < 2^{C \cdot s(n)}$$

für eine Konstante $C \in \mathbb{N}$. Die nicht-deterministische Lösung des Pfadproblems braucht nun $\mathcal{O}\big(s(n)\big)$ Platz, während die deterministische Lösung quadratisch soviel Platz benötigt. □

Zum Abschluß präsentieren wir ein ***PSPACE***-vollständiges Problem.

copyright: Jürgen Koslowski, TU Braunschweig, 19. Juli 2019

2.6.04 Definition. Die Syntax *quantifizierter Boole'scher Formeln* mit Gleichheit (kurz *qBF*) hat die BNF

$$F ::= x \mid \top \mid \bot \mid x = y \mid \neg F \mid F \wedge F \mid F \vee F \mid F \rightarrow F \mid \forall x.F \mid \exists x.F$$

wobei x und y Variablen oder atomare Formeln sind. Die Menge der *freien Variablen* einer qBF F ist gegeben durch

$$\begin{aligned} \boldsymbol{FV}(\top) = \boldsymbol{FV}(\bot) &:= \emptyset \\ \boldsymbol{FV}(x) &:= \{x\} \\ \boldsymbol{FV}(x = y) &:= \{x, y\} \\ \boldsymbol{FV}(\neg F) &:= \boldsymbol{FV}(F) \\ \boldsymbol{FV}(F \wedge G) = \boldsymbol{FV}(F \vee G) = \boldsymbol{FV}(F \rightarrow G) &:= \boldsymbol{FV}(F) \cup \boldsymbol{FV}(G) \\ \boldsymbol{FV}(\forall x.F) = \boldsymbol{FV}(\exists x.F) &:= \boldsymbol{FV}(F) - \{x\} \end{aligned}$$

Nicht freie Variablen sind durch einen Quantor *gebunden*.

F heißt *geschlossen*, falls jede auftretenden Variable gebunden ist.

Achtung: Hier handelt es sich *nicht* um die Syntax der Prädikatenlogik, sondern um eine Erweiterung der Aussagenlogik um Quantoren. Wir betrachten hier *keine* Signaturen, zugehörige Terme oder Strukturen. Die atomaren Formeln (oder Variablen) x sind frei mit Wahrheitswerten belegbar. Dennoch gilt auch hier:

2.6.05 Fakt. Jede geschlossene qBF F kann in sogenannter *Prenex-Normalform* (kurz *PNF*) dargestellt werden

$$F \mathrel{\models\!\!\!\dashv} Q_0 x_0.\, Q_1 x_1. \ldots Q_{n-1} x_{n-1}.\, G(x_0, \ldots, x_{n-1})$$

wobei G keine Quantoren enthält. Mit Hilfe von "Dummy-Variablen" ist es zudem möglich, *alternierende Prenex-Normalform* (kurz *aPNF*) zu erreichen, bei der Existenz- und All-Quantoren alternieren:

$$F \mathrel{\models\!\!\!\dashv} \exists x_0.\, \forall x_1. \ldots Q_{m-1} x_{m-1}.\, G(x_0, \ldots, x_{m-1})$$

Die Semantik quantifizierter Boole'scher Formeln ist durch eine Ergänzung der aussagenlogischen Semantik gegeben. Wir erinnern zunächst an den Substitutions-Operator $[x/G]$, der jedes freie(!) Auftreten der Variable x in einer Formel F durch die Formel G ersetzt. Im Fall quantifizierter Boole'scher Formeln ist darauf zu achten, dass keine freie Variable $y \neq x$ von G in den Einflußbereich eine Quantors $\forall y$ oder $\exists y$ in F gerät. Uns interessiert hier aber nur der Fall $G \in \{\bot, \top\}$, so dass dieses Problem nicht auftritt.

2.6.06 Definition. Für eine Belegung φ der Variablen setze

$$\hat{\varphi}(\forall x.\ F) := \inf\{\hat{\varphi}(F[x/\top]), \hat{\varphi}(F[x/\bot])\} \quad \text{und} \quad \hat{\varphi}(\exists x.\ F) := \sup\{\hat{\varphi}(F[x/\top]), \hat{\varphi}(F[x/\bot])\}$$

copyright: Jürgen Koslowski, TU Braunschweig, 19. Juli 2019

Wir nennen F *wahr*, wenn $\hat{\varphi}(F) = 1$ für jede Belegung φ der Variablen (dies erweitert den Begriff der Tautologie).

2.6.07 Satz. *Das E-Problem*

quantifizierte Boole'sche Formel *(* QBF *)*
Gegeben: *eine geschlossene quantifizierte Boolesche Formel* F *in PNF*
zu entscheiden: *ob* F *wahr ist.*

ist ***PSPACE****-vollständig.*

Beweis. Die Zugehörigkeit zu ***PSPACE*** folgt, indem man eine Maschine betrachtet, die im Wesentlichen den Davis-Putnam-Algorithmus implementiert: sukzessive werden Quantoren von links nach rechts entfernt, und in der verbliebenen Formel die bisher gebundene Variable durch $\top$ bsw. $\bot$ substituiert. Auf diese Weise erhalten wir einen Baum mit quantorenfreiern Boole'scher Formeln als Blättern, deren Auswertung logarithmischen Platz braucht, vergl. BOOLEVAL . Sobald ein Zweig bearbeitet ist, kann sein Platz wiederverwendet werden. Bei einer Baumtiefe von n liegt der Platzbedarf also bei $\mathcal{O}(n + \log n) = \mathcal{O}(n)$.

Der Nachweis, dass QBF ***PSPACE***-hart ist orientiert sich an [Sip96]. Betrachte eine Sprache $L \subseteq \Sigma^*$ in ***PSPACE***, die von einer dTM M mit polynomialer Raumkomplexität mit leerem Band akzpetiert wird. Wie im Beweis des Cooke'schen Satzes können die Konfigurationen von M durch Boolesche Formeln in KNF ausgedrückt werden.

Mit Hilfe neuer Variabler c_i für Konfigurationen lassen sich damit gemäß der Halbierungsidee im Beweis des Satzes von Savitch quantifizierte Boolesche Formeln $\rho(c_i, c_j, k)$ bilden, die ausdrücken, dass c_j von c_i aus in höchstens 2^k Schritten erreicht werden kann.

Für eine Eingabe $x \in \Sigma^*$ ist die Laufzeit von der Ordnung $\mathcal{O}\left(2^{C \cdot s(|x|)}\right)$, aber der Platz für die Auswertung eines Zweiges des Auswertungsbaums kann beim nächsten Zweig wiederverwendet werden. Die Auswertung quantorenfreier Boole'scher Formeln braucht nur logarithmischen Platz in der Anzahl der Variablen (vergl. BOOLEVAL), die hier durch die Anzahl der Konfigurationen beschränkt ist. Wegen $\log\left(2^{C \cdot s(|x|)}\right) = C \cdot s(|x|)$ ist besagte Auswertung polynomial in $|x|$. Außerdem ist $x \in L$ äquivalent zur Wahrheit der qBF $\rho\left(c_x, c_F, C \cdot s(|x|)\right)$, wobei c_x die x entsprechende Initialkonfiguration und c_F die einzige akzeptierende Haltekonfiguration ist.

Behauptung: Die Berechnung $x \mapsto \rho\left(c_x, c_F, C \cdot s(|x|)\right)$ braucht polynominale Zeit in $|x|$.

Idee: Abschätzung der Zeit durch Abschätzung der Länge.

$i = 0$; Wie zuvor ist der Fall $\rho_{\langle c_0, c_1, 0 \rangle}$ trivial.

Annahme: die Behauptung stimmt für alle $j \leq i$.

$i + 1$ mittels Zwischenkonfigurationen:

$$\rho_{\langle c_0, c_1, i+1 \rangle} = \exists m. \left(\rho_{\langle c_0, m, i \rangle} \wedge \rho_{\langle m, c_1, i \rangle}\right) \qquad \text{(vorläufig)}$$

Das Einfügen von Zwischenkonfigurationen verdoppelt aber die Länge der Formeln, was zu exponentiellem Wachstum führt. Das läßt sich mit Hilfe universeller Quantifizierung und geschickter

copyright: Jürgen Koslowski, TU Braunschweig, 19. Juli 2019

Anwendung der Gleichheit verhindern:

$$\rho_{\langle c_0, c_1, i+1\rangle} \boxminus \exists m.\,\forall u.\forall v.\big((\langle u,v\rangle = \langle c_0, m\rangle \vee \langle u,v\rangle = \langle m, c_1\rangle) \to \rho_{\langle x,y,i\rangle}\big)$$

was die Länge pro Schritt nur noch um eine Konstante vergrößert. Damit ist die Behauptung nachgewiesen. □

QBF kann als Erweiterung von SAT aufgefaßt werden: eine KNF-Formel $F(x_0, \ldots, x_{t-1})$ ist genau dann erfüllbar, wenn folgende Formel wahr ist

$$\exists x_0.\, \exists x_1.\, \ldots \exists x_{t-1}.\, F(x_0, \ldots, x_{t-1})$$

qBF-Formeln in aPNF lassen sich auch *spieltheoretisch* interpretieren: zwei Spieler, Eloise und Abelard, wählen abwechsend Wahrheitswerte für die Variablen in

$$G = \exists x_0.\, \forall x_1.\, \ldots Q_{m-1} x_{m-1}.\, \varphi(x_0, \ldots, x_{m-1})$$

Eloise für die durch Existenzquantoren gebundenen Variablen, Abelard für die übrigen. Eloise *gewinnt*, falls in der so bestimmten Belegung F wahr ist, andernfalls gewinnt Abelard:

FormelSpiel (∃∀GAME) **Gegeben:** eine geschlossene quantifizierte Boolesche Formel F in aPNF **zu entscheiden:** ob Eloise eine Gewinnstrategie hat.

Man überzeugt sich leicht davon (HA?), dass ∃∀GAME ***PSPACE***- vollständig ist.

Literatur

[Pap94] Christos H. Papadimitriou. *Computational complexity.* Addison-Wesley, 1994.

[Sip96] Michael Sipser. *Introduction to the Theory of Computation.* International Thomson Publishing, 1996.

[Sé01] G. Sénizergues. L(a)=l(b)? decidability results from complete formal systems. *Theoretical Computer Science*, 251(1):1 – 166, 2001.

[Sé02] G. Sénizergues. L(a)=l(b)? a simplified decidability proof. *Theoretical Computer Science*, 281(1):555 – 608, 2002. Selected Papers in honour of Maurice Nivat.

copyright: Jürgen Koslowski, TU Braunschweig, 19. Juli 2019

Jürgen Koslowski (koslowj@tu-bs.de)
Theoretical Computer Science
TU Braunschweig
Mühlenpfordstr. 23
D-38106 Braunschweig
Germany

copyright: Jürgen Koslowski, TU Braunschweig, 19. Juli 2019