Logische Methoden in der Informatik

Sommersemester 2022

Stephan Kreutzer
Technische Universität Berlin

Inhaltsverzeichnis

Kapitel 1.

Einleitung

Als einer der zentralen Grundlagen der Informatik finden sich Anwendungen der Logik in der gesamten Breite der Informatik. Unter den kommerziell erfolgreichsten Beispielen für Gebiete der Informatik, die aus der Logik heraus entwickelt wurden, sind sicherlich relationale Datenbanken zu nennen, die nach dem Vorbild der Prädikatenlogik und des Begriffs der logischen Struktur modelliert wurden. Ebenso zählt der Bereich der computerunterstützten Verifikation, hier vor allem das sogenannte *Model Checking*, sicherlich zu den bekanntesten und erfolgreichsten Anwendungen der Logik in der Informatik. Als letztes Beispiel sei die Semantik von Programmiersprachen genannt, ein Gebiet, das fest in der Logik verankert ist.

Viele dieser Anwendungen wurden aus der Logik heraus entwickelt und haben dann umgekehrt wichtige Forschungsrichtungen innerhalb der Logik begründet, in denen die notwendige Theorie für diese Anwendungen weiter entwickelt wurde.

Aus diesen Arbeiten heraus haben sich sehr enge Beziehungen der Logik zu anderen klassischen Konzepten der Informatik ergeben aus denen zum Teil extrem elegante und wichtige Theorien entwickelt wurden.

Im Rahmen dieser Vorlesung werden wir einige der schönsten und elegantesten Beispiele für solche Theorien kennen lernen. Auf diese Weise werden wir sowohl Methoden behandeln, die hinter einigen wichtigen Anwendungen der Logik stehen, als auch viele Querbezüge zu anderen Bereichen der Informatik kennen lernen, die uns letztlich helfen, diese miteinander in Beziehung zu setzen.

1.1. Inhaltsübersicht

Im ersten Teil der Vorlesung behandeln wir den Zusammenhang zwischen Logik und der Theorie formaler Sprachen. Mit Hilfe logischer Formeln kann man auf naheliegende Weise formale Sprachen beschreiben. Es stellt sich daher die Frage, welche Klasse formaler Sprachen z.B. durch Formeln der Prädikatenlogik definiert werden kann. Erstaunlicherweise hat sich gezeigt, dass die Klasse der regulären Sprachen genau die Sprachen sind, die durch Formeln der *monadischen Logik zweiter Stufe* (MSO) definierbar sind, einer einfachen Erweiterung der Prädikatenlogik. Formeln dieser Logik sind also letztlich äquivalent

zu endlichen Automaten und wir können Formeln und Automaten effektiv ineinander übersetzen.

Die Logik bietet also eine einfache Art, Sprachen zu definieren, und diese Definitionen können dann effektiv in äquivalente Automaten übersetzt werden, die sich sehr viel besser für algorithmische Anwendungen eignen.

Wir werden diesen engen Zusammenhang zwischen Logik und Automatentheorie in der Vorlesung genauer behandeln.

Ein weiterer Vorteil dieser Analogie zwischen Logiken und Automatenmodellen ist, dass logische Formeln natürlich nicht auf Sprachen endlicher Wörter beschränkt sind, sondern wir ebensogut Sprachen über endlichen Bäumen, oder Sprachen unendlicher Wörter oder Bäume definieren können. Wenn MSO-Formeln genau die regulären Sprachen definieren, könnte man hoffen, dass die durch MSO-Formeln definierbaren Baumsprachen vielleicht ebenso gute Eigenschaften aufweisen, wie reguläre Sprachen. Und dass man vielleicht auch nützliche Automatenmodelle entwickeln kann, die auf endlichen Bäumen statt Wörtern arbeiten und wiederum äquivalent zu MSO-Formeln sind.

Das ist in der Tat der Fall und wir werden entsprechende Automatenmodelle kennen lernen.

Die oben schon erwähnte Logik MSO ist eine recht einfache Erweiterung der Prädikatenlogik, die es aber erlaubt, viele algorithmische Probleme z.B. über Graphen auf sehr einfache Weise zu formalisieren. Man kann MSO also auch als eine high-level Programmiersprache verwenden. Aus diesem Grund gewinnt die Frage an Bedeutung, wie schwer es eigentlich ist (im Sinne der Komplexitätstheorie), logische Formeln in gegebenen Eingabestrukturen auszuwerten. Auch dieser Frage werden wir in der Vorlesung nachgehen.

In diesem Zusammenhang zwischen Logik und Komplexität hat sich herausgestellt, dass es einen sehr engen Zusammenhang zwischen Logik und Komplexitätstheorie gibt. Ein bekannter Satz aus diesem Gebiet zeigt z.B., dass die Klasse NP, also der in nichtdeterministischer Polynomialzeit entscheidbarer Probleme, genau die Klasse aller Probleme ist, die in der existentiellen Logik zweiter Stufe ($\exists$SO) beschrieben werden können. D.h. $\exists$SO liefert eine rein logische Beschreibung der Komplexitätsklasse NP. Dies hat viele Vorteile gegenüber der Definition über Turing-Maschinen, z.B. ist es unentscheidbar, ob eine gegebene Turing-Maschine polynomiell zeitbeschränkt ist, wohingegen man natürlich sehr leicht entscheiden kann, ob eine Formel eine syntaktisch korrekte $\exists$SO Formel ist.

Neben NP können viele andere Komplexitätsklassen durch Logiken beschrieben werden.

Insgesamt bietet die Vorlesung eine Einführung in logische Anwendungen in der Informatik. Sie bildet aber auch eine Klammer, in der Themen der anderen Theorievorlesungen an der TU Berlin noch einmal aufgegriffen und miteinander in Beziehung gesetzt werden.

Teil I.

Reguläre Sprachen und die monadische Logik zweiter Stufe

Kapitel 2.

Formale Sprachen und endliche Automaten

In diesem Kapitel wiederholen wir zunächst einige bekannte Konzepte der Automatentheorie, wie sie zum Beispiel im Modul *Formale Sprachen und Automaten* behandelt werden.

2.1 Definition (Alphabete, Wörter, Sprachen) *Sei Σ eine endliche Menge, genannt* Alphabet*, deren Elemente wir Buchstaben nennen. Ein* Wort *über Σ ist eine endliche Folge $w = a_1 \dots a_n$ von Buchstaben $a_i \in \Sigma$. Wir bezeichnen mit Σ^* die Menge aller Wörter über Σ. Eine* Sprache *über Σ ist eine Menge $\mathcal{L} \subseteq \Sigma^*$.*

2.1. Anwendungen formaler Sprachen

Ein wichtiges Einsatzgebiet formaler Sprachen ist natürlich die Analyse von Programmen einer Programmiersprache. Hierbei werden genau die Arten formaler Sprachen verwendet, wie sie typischerweise in Einführungsvorlesungen zur Automatentheorie oder Theorie formaler Sprachen behandelt werden.

Formale Sprachen bieten darüber hinaus auch eine erste Abstraktionsebene im Umgang mit algorithmischen Problemen. Jedes Problem über Graphen, Strukturen, usw. kann durch eine geeignete Kodierung als formale Sprache dargestellt werden. Die Kodierung von, z.B., Graphen durch Wörter über einem festen Alphabet entspricht dabei dem Vorgehen in der Programmierung. Auch hier muss ein Graph ja in irgendeiner Form im Speicher eines Computers repräsentiert werden und damit letztlich als ein Wort über dem Alphabet $\{0, 1\}$. Dementsprechend werden auch die üblichen Komplexitätsklassen wie P oder NP formal als Klassen formaler von Sprachen definiert. Umgekehrt kann man wiederum auch jede Klasse formaler Sprachen als eine Art von Komplexitätsklasse auffassen. Die Klassifikation algorithmischer Probleme entsprechend ihrer Komplexität ist also letztlich eine Klassifikation formaler Sprachen. Natürlich unterscheiden sich die in diesem Kontext auftretenen Sprachen deutlich von den Sprachen, die bei der Analyse von Programmcode verwendet werden.

Durch die oben schon erwähnte implizite Repräsentation von Graphen, oder auch nur Mengen, durch Wörter über $\{0, 1\}$ im Speicher eines Computers oder einer Turing-

Machine ändern wir allerdings die Eingabe in subtiler Form. Da ja die Elemente einer Menge in einer bestimmten Reihenfolge in der Kodierung auftreten, wird die Menge zwangläufig geordnet. Normalerweise spielt das keine Rolle, aber in der Charakterisierung von Komplexitätsklassen durch logische Formeln in Part III wird uns diese implizite Ordnung große Probleme bereiten.

Wir werden uns im ersten Teil der Vorlesung aber zunächst mit den klassischen formalen Sprachen der Chomsky-Hierarchie beschäftigen, vor allem den regulären Sprachen.

2.2. Endliche Automaten und reguläre Sprachen

Die *regulären Sprachen* sind eine der einfachsten Arten von Sprachen die wir betrachten, sie bilden die unterste Stufe in der *Chomsky Hierarchie*. Die Bedeutung der regulären Sprachen unter anderem in der Textverarbeitung und Programmierung liegt an den sehr guten Eigenschaften regulärer Sprachen. Zum einen sind sie so allgemein, dass sie für viele Anwendungen völlig ausreichen. Andererseits aber sind sie unter sehr vielen Operationen abgeschlossen und für viele Entscheidungsprobleme, die reguläre Sprachen betreffen, gibt es sehr effiziente Algorithmen. Reguläre Ausdrücke, deterministische und nichtdeterministische endliche Automaten und Typ-3 Grammatiken sind bekannte äquivalente Beschreibungsmöglichkeiten für reguläre Sprachen. Wir werden in diesem Kapitel eine äquivalente Charakterisierung durch die Logik kennenlernen.

Wir wiederholen zunächst kurz die Definition der regulären Sprachen durch endliche Automaten.

2.2 Definition (endliche Automaten) *Ein* endlicher nicht-deterministischer Automat *(abgekürzt: NFA) ist ein Tupel* $\mathcal{A} \coloneqq (Q, \Sigma, q_0, \Delta, F)$*, wobei*

- Q *eine endliche Menge von Zuständen,*
- Σ *ein endliches Alphabet,*
- q_0 *der* Anfangszustand*,*
- $F \subseteq Q$ *die Menge der* Endzustände *und*
- $\Delta \subseteq Q \times \Sigma \times Q$ *die* Übergangsrelation *ist.*

Der Automat ist deterministisch *(abgekürzt: DFA), wenn* Δ *funktional ist, d.h. wenn es für jeden Zustand* $q \in Q$ *und jedes Symbol* $a \in \Sigma$ *genau ein* $q' \in Q$ *gibt mit* $(q, a, q') \in \Delta$*. Wir schreiben dann üblicherweise* Δ *als Funktion* $\delta : Q \times \Sigma \to Q$*.*

2.3 Definition (Läufe, akzeptierte Sprache) *Ein* Lauf ρ *eines NFA* $\mathcal{A} \coloneqq (Q, \Sigma, q_0, \Delta, F)$ *auf einem Wort* $w \coloneqq a_1 \dots a_n \in \Sigma^*$ *ist eine Folge* $q_0 q_1 \dots q_n$ *von Zuständen, so dass für alle* $1 \leqslant i \leqslant n$ *gilt:* $(q_{i-1}, a_i, q_i) \in \Delta$*. Der Lauf ist* akzeptierend*, wenn* $q_n \in F$*.*

Die von einem Automaten $\mathcal{A}$ akzeptierte Sprache*, geschrieben $\mathcal{L}(\mathcal{A})$, ist die Menge aller Wörter $w \in \Sigma^*$, so dass es einen akzeptierenden Lauf von $\mathcal{A}$ auf w gibt.*

2.4 Definition (reguläre Sprache) *Sei Σ ein endliches Alphabet. Eine Sprache $\mathcal{L} \subseteq \Sigma^*$ heißt* regulär*, wenn es einen NFA $\mathcal{A}$ mit $\mathcal{L}(\mathcal{A}) = \mathcal{L}$ gibt.*

Wir könnten uns auch auf deterministische endliche Automaten einschränken, wie der folgende Satz zeigt, den wir ohne Beweis notieren.

2.5 Satz *Eine Sprache $\mathcal{L} \subseteq \Sigma^*$ ist genau dann regulär, wenn es einen deterministischen Automaten gibt, der $\mathcal{L}$ akzeptiert.*

2.3. Abschlusseigenschaften regulärer Sprachen

Die Klasse der regulären Sprachen ist abgeschlossen unter den gewöhnlichen Mengenoperationen Vereinigung, Schnitt und Komplement. Darüber hinaus gilt die Abgeschlossenheit auch für die Konkatenation und den Kleene-Stern. Für unsere logische Charakterisierung benötigen wir auch den Abschluss unter Projektion.

2.6 Lemma (Abschluss unter Schnitt, Komplement und Vereinigung) *Sei Σ ein endliches Alphabet.*

1. *Sind $\mathcal{L}_1, \mathcal{L}_2 \subseteq \Sigma^*$ reguläre Sprachen, so sind auch $\mathcal{L}_1 \cup \mathcal{L}_2$ und $\mathcal{L}_1 \cap \mathcal{L}_2$ regulär.*
2. *Ist $\mathcal{L} \subseteq \Sigma^*$ regulär, so ist auch $\Sigma^* \setminus \mathcal{L}$ regulär.*

Beweis. (Skizze) Angenommen, $\mathcal{L}_1, \mathcal{L}_2, \mathcal{L}$ werden durch deterministische endliche Automaten $\mathcal{A}_1, \mathcal{A}_2, \mathcal{A}$ erkannt. Dann wird $\mathcal{L}_1 \cup \mathcal{L}_2$ durch den Vereinigungsautomat $\mathcal{B}_1$ erkannt, der aus der disjunkten Vereinigung der beiden Automaten $\mathcal{L}_1 \dot\cup \mathcal{L}_2$ zusammen mit einem neuen Anfangszustand entsteht.

$\mathcal{L}_1 \cap \mathcal{L}_2$ wird durch den Produktautomaten erkannt.

$\Sigma^* \setminus \mathcal{L}$ wird durch das Komplement von $\mathcal{A}$ erkannt, also durch den Automaten, der genauso wie $\mathcal{A}$ definiert ist, nur dass Endzustände und Nicht-Endzustände vertauscht werden. □

Als letzte Abschlusseigenschaft brauchen wir noch den Abschluss unter *Projektion.*

2.7 Definition *Sei Σ ein endliches Alphabet und sei $\Gamma \coloneqq \Sigma \times \{0,1\}$. Sei $\mathcal{L} \subseteq \Gamma^*$ eine Sprache. Die* Projektion *von $\mathcal{L}$ auf Σ ist definiert als die Sprache $\mathcal{L}_\pi \subseteq \Sigma^*$ mit*

$$\mathcal{L}_\pi \coloneqq \{a_1 \dots a_n \in \Sigma^* : \quad \text{es ex. Folge } i_1 \dots i_n \text{ mit } i_j \in \{0,1\} \text{ für alle } j \text{ mit } 1 \leqslant j \leqslant n \text{ und } (a_1, i_1) \dots (a_n, i_n) \in \mathcal{L}\}.$$

2.8 Beispiel Sei $\Sigma \coloneqq \{a, b\}$ und $\Gamma \coloneqq \Sigma \times \{0, 1\}$. Sei $\mathcal{L} \subseteq \Gamma^*$ die Menge aller Wörter $(a_1, i_1) \dots (a_n, i_n)$, so dass

- $i_n = 0$ und
- für alle $1 \leqslant j \leqslant n$ gilt $i_j = 0$ genau dann, wenn $|\{a_l : l \leqslant j \text{ und } a_l = a\}|$ gerade ist.

Zum Beispiel ist das Wort $(a, 1)(b, 1)(b, 1)(a, 0)(b, 0)$ in $\mathcal{L}$, das Wort $(a, 0)$ aber nicht.

Die Projektion von $\mathcal{L}$ auf Σ ist die Sprache aller Wörter über Σ, die eine gerade Anzahl von a's enthalten.

2.9 Lemma *Sei Σ ein endliches Alphabet und sei $\Gamma \coloneqq \Sigma \times \{0, 1\}$. Sei $\mathcal{L} \subseteq \Gamma^*$ eine Sprache. Wenn $\mathcal{L}$ regulär ist, dann ist auch die Projektion $\mathcal{L}_\pi$ von $\mathcal{L}$ auf Σ regulär.*

Beweis. Sei $\mathcal{A} \coloneqq (Q, \Gamma, q_0, \Delta, F)$ ein NFA mit $\mathcal{L}(\mathcal{A}) = \mathcal{L}$. Wir definieren einen NFA $\mathcal{B} \coloneqq (Q, \Sigma, q_0, \Delta', F)$ mit den gleichen Zuständen wie $\mathcal{A}$ und mit der Übergangsrelation

$$\Delta' \coloneqq \{(q, a, q') : (q, (a, 1), q') \in \Delta \text{ oder } (q, (a, 0), q') \in \Delta\}.$$

Sei nun $w = a_1 \cdots a_n \in \Sigma^*$. Der Automat $\mathcal{B}$ akzeptiert das Wort w genau dann, wenn es eine Folge $i_1, \dots, i_n$ mit $i_j \in \{0, 1\}$ gibt, so dass $\mathcal{A}$ das Wort $w' \coloneqq (a_1, i_1) \cdots (a_n, i_n)$ akzeptiert. Also ist $w \in \mathcal{L}_\pi$ genau dann, wenn $w' \in \mathcal{L}$. □

Kapitel 3.

Die Monadische Logik zweiter Stufe

In diesem Abschnitt führen wir die *monadische Logik zweiter Stufe*, kurz MSO, ein. Die Logik erweitert die Prädikatenlogik durch die Möglichkeit, über Mengen von Elementen zu quantifizieren.

3.1. Syntax und Semantik

3.1 Definition (Variablen und Terme) *Wir fixieren eine abzählbar unendliche Menge* Var_1 *von* Variablen erster Stufe *sowie eine abzählbar unendliche Menge von* monadischen Variablen zweiter Stufe mVar_2. *Wir werden die Variablen erster Stufe durch Kleinbuchstaben* $x, y, z, x_i ...$ *bezeichnen, die der zweiten Stufe entsprechend mit Großbuchstaben.*

Sei σ *eine Signatur. Die Menge* $\mathcal{T}_\sigma$ *der* σ*-Terme ist induktiv wie folgt definiert.*

- $x \in \mathcal{T}_\sigma$ *für alle* $x \in \mathrm{Var}_1$.
- $c \in \mathcal{T}_\sigma$ *für alle Konstantensymbole* $c \in \sigma$.
- *Ist* $f \in \sigma$ *ein* k*-stelliges Funktionssymbol und* $t_1, \ldots, t_k \in \mathcal{T}_\sigma$*, dann ist*

$$f(t_1, \ldots, t_k) \in \mathcal{T}_\sigma \,.$$

Ein Term, in dem keine Variablen vorkommen, heißt Grundterm.

3.2 Definition (Syntax der Monadischen Logik zweiter Stufe) *Sei* σ *eine Signatur. Die Menge* $\mathrm{MSO}[\sigma]$ *der Formeln der monadischen Logik zweiter Stufe über* σ *ist induktiv wie folgt definiert.*

- $t = t' \in \mathrm{MSO}[\sigma]$ *für alle Terme* $t, t' \in \mathcal{T}_\sigma$.
- $X = Y \in \mathrm{MSO}[\sigma]$ *für alle* $X, Y \in \mathrm{mVar}_2$.
- $X(x) \in \mathrm{MSO}[\sigma]$ *für alle* $x \in \mathrm{Var}_1$ *und* $X \in \mathrm{mVar}_2$.
- $R(t_1, \ldots, t_k) \in \mathrm{MSO}[\sigma]$ *für alle* k*-stelligen Relationssymbole* $R \in \sigma$ *und alle* $t_1, \ldots, t_k \in \mathcal{T}_\sigma$.

- *Wenn* $\varphi \in \mathrm{MSO}[\sigma]$*, dann* $\neg\varphi \in \mathrm{MSO}[\sigma]$*.*
- *Wenn* $\varphi, \psi \in \mathrm{MSO}[\sigma]$*, dann* $(\varphi \vee \psi) \in \mathrm{MSO}[\sigma]$*,* $(\varphi \wedge \psi) \in \mathrm{MSO}[\sigma]$*,* $(\varphi \to \psi) \in \mathrm{MSO}[\sigma]$ *und* $(\varphi \leftrightarrow \psi) \in \mathrm{MSO}[\sigma]$*.*
- *Wenn* $\varphi \in \mathrm{MSO}[\sigma]$ *und* $x \in \mathrm{Var}_1$*, dann* $\exists x\varphi \in \mathrm{MSO}[\sigma]$ *und* $\forall x\varphi \in \mathrm{MSO}[\sigma]$*.*
- *Wenn* $\varphi \in \mathrm{MSO}[\sigma]$ *und* $X \in \mathrm{mVar}_2$*, dann* $\exists X\varphi \in \mathrm{MSO}[\sigma]$ *und* $\forall X\varphi \in \mathrm{MSO}[\sigma]$*.*

Wir definieren $\mathrm{MSO} \coloneqq \bigcup\{\mathrm{MSO}[\sigma] : \sigma \text{ ist eine Signatur }\}$*.*

3.3 Bemerkung Oft wird auch $x \in X$ geschrieben statt $X(x)$. Wir werden, abhängig vom Kontext, zwischen diesen Schreibweisen wechseln und uns an die übliche Schreibweise des jeweiligen Forschungsfeldes halten.

Die Semantik von MSO ist analog zur Semantik der Prädikatenlogik definiert. Wie in der Prädikatenlogik definieren wir den Begriff der freien Variablen *frei*(φ) einer Formel $\varphi \in \mathrm{MSO}$. Allerdings können in MSO sowohl Variablen erster als auch zweiter Stufe frei vorkommen.

3.4 Definition (Freie und gebundene Variablen) *Sei* σ *eine Signatur. Die Menge frei*(φ) *der* freien Variablen *einer Formel* $\varphi \in \mathrm{MSO}[\sigma]$ *ist induktiv definiert durch:*

- *Für* $t_1, t_2 \in \mathcal{T}_\sigma$ *ist frei*$(t_1 = t_2) \coloneqq \mathrm{Var}_1(t_1) \cup \mathrm{Var}_1(t_2)$*.*
- *Für* $X, Y \in \mathrm{mVar}_2$ *ist frei*$(X = Y) \coloneqq \{X, Y\}$*.*
- *Wenn* $\varphi = R(t_1, \ldots, t_k)$ *für* $R \in \sigma$ *und* $t_1, \ldots, t_k \in \mathcal{T}_\sigma$*, dann frei*$(\varphi) \coloneqq \bigcup_{i=1}^{k} \mathrm{Var}_1(t_i)$*.*
- *Wenn* $\varphi = x \in X$ *für* $x \in \mathrm{Var}_1$ *und* $X \in \mathrm{mVar}_2$*, dann frei*$(\varphi) \coloneqq \{x, X\}$*.*
- *frei*$(\neg\varphi) \coloneqq$ *frei*(φ) *für alle* $\varphi \in \mathrm{MSO}[\sigma]$*.*
- *frei*$\big((\varphi * \psi)\big) \coloneqq$ *frei*$(\varphi) \cup$ *frei*(ψ) *für alle* $* \in \{\vee, \wedge, \to, \leftrightarrow\}$ *und* $\varphi, \psi \in \mathrm{MSO}[\sigma]$*.*
- *Wenn* $\varphi = \exists x\psi$ *oder* $\varphi = \forall x\psi$*, für* $x \in \mathrm{Var}_1$ *and* $\psi \in \mathrm{MSO}[\sigma]$*, dann frei*$(\varphi) \coloneqq$ *frei*$(\psi) \setminus \{x\}$*.*
- *Wenn* $\varphi = \exists X\psi$ *oder* $\varphi = \forall X\psi$*, für* $X \in \mathrm{mVar}_2$ *und* $\psi \in \mathrm{MSO}[\sigma]$*, dann frei*$(\varphi) \coloneqq$ *frei*$(\psi) \setminus \{X\}$*.*

Eine Formel φ *mit frei*$(\varphi) = \emptyset$ *heißt ein Satz. Eine Variable, die in* φ *vorkommt, aber nicht frei ist, heißt* gebunden*. Wir schreiben* $\varphi(v_1, \ldots, v_k)$*, um zu sagen, dass frei*$(\varphi) \subseteq \{v_1, \ldots, v_k\}$*.*

3.5 Definition (Belegungen und Interpretationen) *Sei* σ *eine Signatur und sei* $\mathcal{A}$ *eine* σ*-Struktur.*

1. *Eine* Belegung *in* $\mathcal{A}$ *ist eine Funktion* $\beta : \mathrm{dom}(\beta) \to A \cup \mathrm{Pow}(A)$ *mit* $\beta(x) \in A$, $\beta(X) \in \mathrm{Pow}(A)$ *für alle* $x \in \mathrm{Var}_1$ *und* $X \in \mathrm{mVar}_2$ *und* $\mathrm{dom}(\beta) \subseteq \mathrm{Var}_1 \cup \mathrm{mVar}_2$. *Wir sagen, dass* β *für* $\varphi \in \mathrm{MSO}[\sigma]$ passend *ist, wenn frei*$(\varphi) \subseteq \mathrm{dom}(\beta)$.

2. *Eine* σ-Interpretation *ist ein Paar* $(\mathcal{A}, \beta)$, *bestehend aus einer* σ*-Struktur* $\mathcal{A}$ *und einer Belegung* β *in* $\mathcal{A}$. $\mathcal{I} \coloneqq (\mathcal{A}, \beta)$ *ist passend für* $\varphi \in \mathrm{MSO}[\sigma]$, *wenn* β *zu* φ *passt.*

Im Folgenden sei σ eine Signatur.

3.6 Definition *Sei* $\mathcal{A}$ *eine* σ*-Struktur.*

1. *Ist* β *eine Belegung,* $x \in \mathrm{Var}_1$ *und* $a \in A$ *ein Element, dann definieren wir eine neue Belegung* $\beta[x/a]$ *mit* $\mathrm{dom}(\beta[x/a]) \coloneqq \mathrm{dom}(\beta) \cup \{x\}$ *durch*

$$\beta[x/a](y) \coloneqq \begin{cases} a & \text{wenn } x = y \\ \beta(y) & \text{sonst.} \end{cases}$$

   Die Belegung $\beta[x/a]$ *ist also genauso wie* β *definiert, mit dem Unterschied, dass* x *nun durch* a *interpretiert wird.*

2. *Ist* $\mathfrak{I} \coloneqq (\mathcal{A}, \beta)$ *eine* σ*-Interpretation,* $x \in \mathrm{Var}_1$ *und* $a \in A$ *ein Element, dann definieren wir* $\mathfrak{I}[x/a]$ *als* $(\mathcal{A}, \beta[x/a])$.

3. *Völlig analog definieren wir auch* $\beta[X/S]$ *für eine Variable* $X \in \mathrm{mVar}_2$ *und ein* $S \subseteq A$, *ebenso wie* $\mathfrak{I}[X/S]$.

Wir können nun die Semantik von MSO definieren.

3.7 Definition (Interpretation von Termen) *Sei* σ *eine Signatur. Induktiv über den Formelaufbau definieren wir eine Funktion* $[\![\cdot]\!]$, *die jedem Term* $t \in \mathcal{T}_\sigma$ *und jeder* σ*-Interpretation* $\mathfrak{I} \coloneqq (\mathcal{A}, \beta)$ *für* t *einen Wert* $[\![t]\!]^{\mathfrak{I}} \in A$ *zuweist.*

- $[\![v_i]\!]^{\mathfrak{I}} \coloneqq \beta(v_i)$ *für alle* $v_i \in \mathrm{Var}_1$
- $[\![c]\!]^{\mathfrak{I}} \coloneqq c^{\mathcal{A}}$ *für alle Konstantensymbole* $c \in \sigma$.
- *Ist* $f \in \sigma$ *ein* k*-stelliges Funktionssymbol und* $t_1, \ldots, t_k \in \mathcal{T}_\sigma$ *dann gilt*

$$[\![f(t_1, \ldots, t_k)]\!]^{\mathfrak{I}} \coloneqq f^{\mathcal{A}}\big([\![t_1]\!]^{\mathfrak{I}}, \ldots, [\![t_k]\!]^{\mathfrak{I}}\big).$$

3.8 Definition (Semantik MSO) *Sei* σ *eine Signatur. Induktiv über den Formelaufbau definieren wir eine Funktion* $[\![\cdot]\!]$, *die jeder Formel* $\varphi \in \mathrm{MSO}[\sigma]$ *und jeder* σ*-Interpretation* $\mathfrak{I} \coloneqq (\mathcal{A}, \beta)$ *für* φ *einen Wahrheitswert* $[\![\varphi]\!]^{\mathfrak{I}} \in \{0, 1\}$ *zuordnet.*

- *Für alle Terme $t, t' \in \mathcal{T}_\sigma$ definieren wir*

$$[\![t = t']\!]^{\mathfrak{I}} := \begin{cases} 1 & \text{wenn } [\![t]\!]^{\mathfrak{I}} = [\![t']\!]^{\mathfrak{I}} \\ 0 & \text{sonst.} \end{cases}$$

- *Für alle Formeln $X(x)$ mit $X \in \mathrm{mVar}_2, x \in \mathrm{Var}_1$ definieren wir*

$$[\![X(x)]\!]^{\mathfrak{I}} := \begin{cases} 1 & \text{wenn } [\![x]\!]^{\mathfrak{I}} \in [\![X]\!]^{\mathfrak{I}} \\ 0 & \text{sonst.} \end{cases}$$

- *Für alle k-stelligen Relationssymbole $R \in \sigma$ und alle Terme $t_1, \ldots, t_k \in \mathcal{T}_\sigma$ definieren wir*

$$[\![R(t_1, \ldots, t_k)]\!]^{\mathfrak{I}} := \begin{cases} 1 & \text{wenn } ([\![t_1]\!]^{\mathfrak{I}}, \ldots, [\![t_k]\!]^{\mathfrak{I}}) \in R^{\mathcal{A}} \\ 0 & \text{sonst.} \end{cases}$$

- *Die Semantik der Verknüpfungen $\neg, \vee, \wedge, \rightarrow, \leftrightarrow$ ist wie in der Aussagenlogik definiert, z.B. wenn $\varphi := \neg\psi \in \mathrm{MSO}[\sigma]$ dann definieren wir*

$$[\![\varphi]\!]^{\mathfrak{I}} := 1 - [\![\psi]\!]^{\mathfrak{I}}.$$

- *Wenn $\varphi := \exists x\psi \in \mathrm{MSO}[\sigma]$, dann definieren wir*

$$[\![\varphi]\!]^{\mathfrak{I}} := \begin{cases} 1 & \text{es gibt } a \in A, \text{ so dass } [\![\psi]\!]^{\mathfrak{I}[x/a]} = 1 \\ 0 & \text{sonst.} \end{cases}$$

- *Wenn $\varphi := \forall x\psi \in \mathrm{MSO}[\sigma]$, dann definieren wir*

$$[\![\varphi]\!]^{\mathfrak{I}} := \begin{cases} 1 & [\![\psi]\!]^{\mathfrak{I}[x/a]} = 1 \text{ für alle } a \in A \\ 0 & \text{sonst.} \end{cases}$$

- *Wenn $\varphi := \exists X\psi \in \mathrm{MSO}[\sigma]$, dann definieren wir*

$$[\![\varphi]\!]^{\mathfrak{I}} := \begin{cases} 1 & \text{es gibt } S \subseteq A, \text{ so dass } [\![\psi]\!]^{\mathfrak{I}[X/S]} = 1 \\ 0 & \text{sonst.} \end{cases}$$

- *Wenn $\varphi := \forall X\psi \in \mathrm{MSO}[\sigma]$, dann definieren wir*

$$[\![\varphi]\!]^{\mathfrak{I}} := \begin{cases} 1 & [\![\psi]\!]^{\mathfrak{I}[X/S]} = 1 \text{ für alle } S \subseteq A \\ 0 & \text{sonst.} \end{cases}$$

Wie üblich schreiben wir $\mathfrak{I} \models \varphi$ wenn $[\![\varphi]\!]^{\mathfrak{I}} = 1$.

3.2. Beispiele

Graphen sind Strukturen über der Signatur $\sigma_{graph} = \{E\}$, wobei E ein zweistelliges Relationssymbol ist. Wir betrachten nur ungerichtete, einfache Graphen, d.h. wir nehmen an dass

- $(v, v) \notin E^{\mathcal{G}}$ für jeden Graphen $\mathcal{G} = (V, E^{\mathcal{G}})$ und alle $v \in V$ und dass
- $(v, w) \in E^{\mathcal{G}}$ impliziert $(w, v) \in E^{\mathcal{G}}$ für alle $v, w \in V$.

3.9 Beispiel Sei $\mathcal{G} = (V, E)$ ein Graph, sei $v \in V$ und seien $R, G, B \subseteq V$.

- Sei

$$\varphi(X,Y,Z) \coloneqq \quad \forall x \, \big(\bigvee_{U \in \{X,Y,Z\}} \big(\quad U(x) \wedge \bigwedge_{W \in \{X,Y,Z\} \setminus \{U\}} \neg W(x) \big) \big) \wedge \\ \forall x \forall y (E(x,y) \to \bigwedge_{U \in \{X,Y,Z\}} \neg (U(x) \wedge U(y))) \big).$$

  Es gilt $\mathcal{G} \models \varphi(R, G, B)$ genau dann, wenn R, G, B eine 3-Färbung von $\mathcal{G}$ ist.

- Sei $\varphi(X, Y) \coloneqq \forall x(X(x) \to Y(x))$. Es gilt $\mathcal{G} \models \varphi(R, G)$ genau dann, wenn $R \subseteq G$. Ebenso können wir Formeln definieren, die $R \subsetneq G$, $R \supseteq G$ etc. definieren.

- $\varphi(x, X, Y) \coloneqq X(x) \wedge Y(y)$ definiert, dass $x \in X \cap Y$ ist. Analog können wir Formeln definieren, die sagen, dass $x \in X \setminus Y$, $x \in X \cup Y$ etc liegt.

Wir werden daher in den folgenden Beispielen die üblichen Mengenoperationen in den Formeln benutzen, also Formeln den Art $\exists x(x \in X \setminus Y \wedge ...)$ schreiben. Es ist dann klar, dass diese Formeln äquivalent in MSO ausgedrückt werden können.

- Zusammenhang:

$$\neg \exists X \exists Y \Big(X \neq \emptyset \wedge Y \neq \emptyset \wedge \forall x (x \in X \vee x \in Y) \wedge \\ \forall x \forall y \big((x \in X \wedge y \in Y \to \neg E(x,y)) \big) \Big)$$

- Dominierende Menge:

$$\varphi(X) \coloneqq \forall y \big(y \in X \vee \exists x (x \in X \wedge E(x,y)) \big)$$

Die bisherigen Beispiele verwendeten die Standardkodierung von Graphen als relationale Strukturen über der Signatur $\sigma_{graph} = \{E\}$, die nur die Kantenrelation enthält. Im Zusammenhang mit MSO über Graphen ist oft folgende alternative Modellierung als sogenannte *Inzidenzstruktur* sinnvoll.

3.10 Definition (Inzidenzstruktur) *Sei $\sigma_{inc} := \{V, E, I\}$, wobei V, E einstellige Relationssymbole sind und I ein zweistelliges Relationssymbol ist. Wir können dann einen Graph $G = (V, E)$ als σ_{inc}-Struktur $\mathcal{G} = (G, V^{\mathcal{G}}, E^{\mathcal{G}}, I^{\mathcal{G}})$ modellieren, wobei $G = V \cup E$, $V^{\mathcal{G}} = V$, $E^{\mathcal{G}} = E$ und $I^{\mathcal{G}} := \{(v, e) : v \in V, e \in E, v \in e\}$. Wir nennen diese Art der Modellierung die* Inzidenzkodierung *von Graphen und die übliche Modellierung als Strukturen über σ_{graph} die* Standardkodierung.

Für die Prädikatenlogik spielt es keine Rolle, ob Graphen als Inzidenzstrukturen oder über die Standardkodierung modelliert werden. Für MSO sind die beiden Arten aber grundverschieden, denn in der Standardkodierung kann nur über Mengen von Knoten quantifiziert werden, in der Inzidenzkodierung aber auch über Mengen von Kanten.

3.11 Beispiel Man betrachte die Formel

$$\varphi(X, Y) := \exists F \subseteq E \begin{pmatrix} \forall e \in F(\exists x \in X \exists y \in Y(I(x, e) \wedge I(y, e))) \wedge \\ \forall x \in X \exists^{=1} e \in F(I(x, e)) \wedge \\ \forall y \in Y \exists^{=1} e \in F(I(y, e)) \end{pmatrix}.$$

Hierbei steht $\exists^{=1} e \varphi(e)$ für die Formel $\exists e \varphi(e) \wedge \forall f(\varphi(f) \rightarrow e = f)$ die besagt, dass es genau ein e gibt, dass die Formel φ erfüllt. Wir werden diese Schreibweise im Weiteren öfters benutzen.

Wenn $\mathcal{G}$ ein Graph mit Universum G in der Inzidenzkodierung ist und $A, B \subseteq G$ mit $A \cap B = \emptyset$ ist, dann gilt $(\mathcal{G}, A, B) \models \varphi$ genau dann, wenn es ein perfektes Matching zwischen A und B in $\mathcal{G}$ gibt, d.h. eine Kantenmenge, die jedem $a \in A$ genau ein $b \in B$ zuordnet.

3.12 Übung *Geben Sie eine σ_{inc}-Formel $\varphi(X)$ an die genau dann für eine Kantenmenge F in einem Graph $\mathcal{G}$ gilt, wenn F ein Pfad ist.*

3.3. MSO-definierbare Sprachen

3.13 Definition *Sei Σ ein endliches Alphabet. Wir definieren eine Signatur $\sigma_\Sigma := \{\leqslant, s, (P_a)_{a \in \Sigma}\}$, wobei $\leqslant$ und s zweistellige Relationssymbole und die P_a, für $a \in \Sigma$, einstellige Relationssymbole sind.*

Ein Wort $w = a_1 \dots a_n \in \Sigma^+$ können wir wie folgt als σ_Σ-Struktur $\mathcal{W}_w := (W, \leqslant^{\mathcal{W}_w}, s^{\mathcal{W}_w}, (P_a)^{\mathcal{W}_w})$ auffassen.

- *Wir wählen als Universum $W := \{1, \dots, n\}$.*
- *Die Relation $\leqslant^{\mathcal{W}_w}$ ist die übliche Ordnung auf $1, ..., n$ und die Relation $s^{\mathcal{W}_w}$ ist die Nachfolgerrelation bzgl. $\leqslant^{\mathcal{W}_w}$, d.h. $s^{\mathcal{W}_w} := \{(i, i+1) : i < n\}$.*

- *Die Relationen $P_a^{\mathcal{W}_w}$ sind definiert durch*

$$P_a^{\mathcal{W}_w} \coloneqq \{i : a_i = a\}.$$

*Umgekehrt entspricht jeder σ_Σ-Struktur $\mathcal{W}$, so dass die Mengen $P_a^{\mathcal{W}}$ das Universum von $\mathcal{W}$ partitionieren, $\leq^{\mathcal{W}_w}$ eine lineare Ordnung und $s^{\mathcal{W}_w}$ die Nachfolgerrelation bzgl. $\leq^{\mathcal{W}_w}$ ist, ein Wort aus Σ^+. Wir nennen die σ_Σ-Strukturen mit dieser Eigenschaft σ_Σ-*Wortstrukturen*. Die Klasse der σ_Σ-Strukturen entspricht genau der Menge Σ^+.*

Für eine Menge $\mathcal{W}$ von σ_Σ-Wortstrukturen definieren wir

$$\mathrm{Lang}(\mathcal{W}) \coloneqq \{w \in \Sigma^+ : \mathcal{W}_w \in \mathcal{W}\}$$

und für eine Sprache $\mathcal{L}$ definieren wir

$$\mathrm{Models}(\mathcal{L}) \coloneqq \{\mathcal{W}_w : w \in \mathcal{L}\}.$$

Wir schränken uns auf Worte aus Σ^+ ein, da wir stets fordern, dass das Universum von Strukturen nicht leer ist.

3.14 Beispiel Sei $\Sigma \coloneqq \{a, b\}$ und sei $w \coloneqq abbaa$. Dann ist $\mathcal{W}_w$ die σ_Σ-Struktur mit Universum $W \coloneqq \{1, \ldots, 5\}$, $\leqslant^{\mathcal{W}_w}$ und $s^{\mathcal{W}_w}$ wie oben angegeben und $P_a^{\mathcal{W}_w} \coloneqq \{1, 4, 5\}$ und $P_b^{\mathcal{W}_w} \coloneqq \{2, 3\}$.

Wenn es klar ist, um welche Struktur es sich gerade handelt, lassen wir üblicherweise den Index $^{\mathcal{W}_w}$ weg, schreiben also kurz P_a statt $P_a^{\mathcal{W}_w}$.

Wir können also nun Sprachen $\mathcal{L} \subseteq \Sigma^*$ auch durch logische Formeln definieren.

3.15 Definition *Sei Σ ein endliches Alphabet. Sei $\varphi \in \mathrm{MSO}[\sigma_\Sigma]$ ein Satz. Die durch φ definierte Wortsprache $\mathcal{L}(\varphi) \subseteq \Sigma^+$ ist definiert als*

$$\mathcal{L}(\varphi) = \{w \in \Sigma^+ : \mathcal{W}_w \models \varphi\}.$$

Eine Sprache $\mathcal{L} \subseteq \Sigma^+$ ist MSO-definierbar*, wenn es ein $\varphi \in \mathrm{MSO}[\sigma_\Sigma]$ gibt mit $\mathcal{L}(\varphi) = \mathcal{L}$.*

3.16 Beispiel Sei $\Sigma \coloneqq \{a, b\}$.

- Wir geben nun eine Formel an, die die Sprache a^*b^* über dem Alphabet $\Sigma \coloneqq \{a, b\}$ definiert.

$$\begin{aligned}\varphi_{a^*b^*} \coloneqq &\exists x \forall y \big((y \leqslant x \to P_a(y)) \wedge (y > x \to P_b(y))\big) \vee \\ &\exists x \forall y \big((y < x \to P_a(y)) \wedge (y \geqslant x \to P_b(y))\big).\end{aligned}$$

  Hierbei steht $y > x$ für $y \geqslant x \wedge \neg y = x$. Nach unserer Konvention ist $\varepsilon \notin \mathcal{L}(a^*b^*)$.

- Wir können das kleinste *min* und größte *max* Element der Ordnung durch $min(x) \coloneqq \forall y(x \leqslant y)$ und $max(x) \coloneqq \forall y(y \leqslant x)$ definieren. In den folgenden Formeln können wir daher *min* und *max* als Konstantensymbole benutzen, die durch das kleinste und größte Element interpretiert sind. Wir können diese Formeln dann leicht in MSO umschreiben.
- Die folgende Formel

  $$\varphi_{even} \coloneqq \exists X\big(\forall x \forall y(s(x,y) \to (x \in X \leftrightarrow y \notin X) \wedge min \in X \wedge max \notin X\big)$$

  gilt in einem Wortmodell $\mathcal{W}_w$, wenn es eine Menge X gibt, die jede zweite Position in dem Wort w enthält, so dass der erste Buchstabe in X ist, der letzte aber nicht. Das ist genau dann der Fall, wenn w gerade Länge hat.
- Sei $\Sigma \coloneqq \{a, b, c\}$. Die Formel

  $$\forall y\big(P_a(y) \to \exists z(z > y \wedge P_b(z))\big)$$

  definiert die Menge der Wörter über Σ, in denen nach jedem a noch mindestens ein b folgt.

3.17 Beispiel Sei $\Sigma \coloneqq \{a, b\}$. Die Sprache $\{a^n b^n : n > 0\}$ ist nicht definierbar in MSO$[\sigma_\Sigma]$, da sie nicht regulär ist. Dies wird aus Satz 4.1 folgen. Erweitern wir jedoch die Wortmodelle um weitere arithmetische Prädikate, so kann die Sprache wieder definiert werden. Erlauben wir z.B. die Addition in Formeln (als dreistellige Relation), dann kann die Sprache durch

$$\exists x\big(x + x = max \wedge \forall z(z \leqslant x \to P_a(x)) \wedge \forall z(x < z \to P_b(z))\big)$$

definiert werden, also sogar in FO.

Mit Ausnahme des letzten Beispiels waren die Sprachen in den vorherigen Beispielen alle regulär. Wir werden als Nächstes zeigen, dass die Klasse der MSO-definierbaren Sprachen genau die regulären Sprachen sind.

3.18 Übung *Sei $\Sigma \coloneqq \{a, b, c\}$. Geben Sie* MSO*-Formeln an, die die folgenden Sprachen definieren. Erläutern Sie wo nötig Ihre Formeln umgangssprachlich.*

1. a^*ac
2. $b(a^*bb)^*$
3. *Die Sprache* $\mathcal{L} \coloneqq \{w \in \{a\}^* :$ *die Anzahl der a's in w ist gerade*$\}$.

3.19 Übung *Zeigen Sie, dass die letzte Sprache aus der vorhergehenden Übung nicht in* FO *definierbar ist.*

Kapitel 4.

Äquivalenz von MSO und Automaten

Ziel dieses Kapitels ist es zu zeigen, dass die MSO-definierbaren Sprachen genau die regulären Sprachen sind.

4.1 Satz (Büchi, Elgot 1961) *Eine Sprache $\mathcal{L} \subseteq \Sigma^+$ ist genau dann regulär, wenn sie* MSO*-definierbar ist.*

4.2 Bemerkung Wir betrachten hier nur Sprachen ohne das leere Wort. Das leere Wort ε würde einer Wortstruktur mit leerem Universum entsprechen. Da solche Strukturen in der Logik nicht zugelassen sind, kann das leere Wort formal gesehen nicht definiert werden. Wir werden daher im Weiteren das leere Wort nicht weiter betrachten.

Wir zeigen dazu zunächst folgendes Lemma.

4.3 Lemma *Sei Σ ein endliches Alphabet. Jede reguläre Sprache über Σ^+ ist* MSO*-definierbar.*

Beweis. Sei $\mathcal{L} \subseteq \Sigma^*$ eine reguläre Sprache. Dann gibt es einen NFA $\mathcal{A} \coloneqq (Q, \Sigma, q_0, \Delta, F)$ mit $\mathcal{L}(\mathcal{A}) = \mathcal{L}$. Wir werden eine Formel konstruieren, die genau dann in einem Wortmodell $\mathcal{W}_w$ gilt, wenn der Automat $\mathcal{A}$ einen akzeptierenden Lauf ρ auf w hat.

Wir erinnern dazu an die Formeln $min(x)$ und $max(x)$ aus Beispiel 3.16.

Als Erstes definieren wir eine Formel $\varphi_{part}(X_0, \ldots, X_n)$ die besagt, dass die Mengen $X_0, \ldots, X_n$ eine Partitionierung des Universums bilden, d.h. das jeder Knoten in genau einer dieser Mengen vorkommt. Man beachte, dass die übliche Definition einer Partitionierung auch fordert, dass jede der Mengen nichtleer ist. Wir lockern für unsere Zwecke diese Definition und erlauben hier auch leere Mengen.

$$\varphi_{part}(X_0, \ldots, X_n) \coloneqq \forall y \bigvee_{i=0}^{n} \Big(y \in X_i \wedge \bigwedge_{j \neq i} y \notin X_j \Big)$$

Sei $Q \coloneqq \{q_0, \ldots, q_n\}$. Wir möchten nun einen akzeptierenden Lauf des Automaten auf w erraten. Der Lauf wird dargestellt als eine Partitionierung von W, dem Universum des Wortmodells, in Mengen $X_0, \ldots, X_n$, die verträglich mit der Übergangsrelation ist. Für jede Position $x \in W$ des Wortes (und das sind Elemente der Struktur $\mathcal{W}_w$) sagen

wir, dass $x \in X_i$ gilt, wenn der Automat sich im Zustand q_i befindet, nachdem er den Buchstaben a_x gelesen hat.

$$\exists X_0 \dots \exists X_n \Big(\exists x_{min} \exists x_{max} min(x_{min}) \wedge max(x_{max}) \wedge \varphi_{part}(X_0, \dots, X_n) \wedge \tag{1}$$

$$\bigwedge_{0 \leqslant i \leqslant n} \big(x_{min} \in X_i \rightarrow \bigvee_{a \in \Sigma, (q_0, a, q_i) \in \Delta} P_a(x_{min})\big) \wedge \tag{2}$$

$$\forall x \forall y \big(s(x,y) \rightarrow \bigwedge_{0 \leqslant i,j \leqslant n} \big(x \in X_i \wedge y \in X_j \rightarrow \bigvee_{(q_i, a, q_j) \in \Delta} P_a(y)\big)\big) \wedge \tag{3}$$

$$\bigvee_{q_i \in F} x_{max} \in X_j \Big) \tag{4}$$

Hierbei sind die Formeln *min* und *max* wie zuvor definiert.
Die Zeilen besagen dabei Folgendes:

1. Zeile (1) definiert x_{min} und x_{max} als kleinstes und größtes Element, also als die erste und letzte Position im Wort. Weiterhin wird definiert, dass die Variablen $X_0, \dots, X_n$ so gewählt werden müssen, dass jede Position in genau einer Menge vorkommt.

2. Die nächste Zeile sagt, dass die erste Position x_{min} nur in einer Variablen X_i vorkommen darf, wenn der Automat vom Startzustand q_0 durch Lesen des ersten Buchstabens in den Zustand q_i übergehen kann.

3. Entsprechend definiert die nächste Zeile, dass die geratenen Zustände auch wirklich einem Lauf entsprechen.

4. Die letzte Zeile sagt dann noch, dass der letzte Zustand ein Endzustand sein muss.

Es gilt also, dass die Formeln genau dann in einem Modell $\mathcal{W}_w$ gilt, wenn w durch $\mathcal{A}$ akzeptiert wird. □

Für die Rückrichtung möchten wir zeigen, dass für jeden MSO[σ_Σ]-Satz φ, die Sprache $\mathcal{L}(\varphi) = \{w \in \Sigma^+ : \mathcal{W}_w \models \varphi\}$ regulär ist. Wir nehmen ohne Beschränkung der Allgemeinheit an, das φ in Pränexnormalform ist, d.h. die Form $Q_1 x_1 \dots Q_n x_n \, \varphi'$ hat, wobei φ' quantorenfrei und in disjunktiver Normalform ist. Analog wie für FO kann man für MSO zeigen, dass man jede Formel in eine äquivalente Formel in Pränexnormalform umwandeln kann. Die Pränexnormalform wird dabei wie für FO definiert, ohne Quantoren zu unterscheiden, die Variable der ersten oder der zweiten Stufen quantifizieren. Wir beobachten weiter, dass ein Wortmodell einer Formel $\varphi(X_1, \dots, X_k)$ die Form $(\mathcal{W}_w, A_1, \dots, A_k)$ hat, wobei $A_i \subseteq W$ für $1 \leqslant i \leqslant k$ (die freien Variablen werden interpretiert durch die Mengen A_i). Wir können derartige Modelle wiederum als Wörter

über dem Alphabet $\Gamma = \Sigma \times \{0,1\}^k$ auffassen. Der Bitvektor kodiert auf natürliche Weise die charakteristische Funktion der Mengen A_i (zum Beispiel wird $(aba, \{1,2\}, \{2,3\})$ durch $(a,1,0)(b,1,1)(a,0,1)$ kodiert).

Wir erinnern an Lemma 2.9, welches besagt, dass für jedes Alphabet Σ und für $\Gamma = \Sigma \times \{0,1\}$, falls $\mathcal{L} \subseteq \Gamma^*$ regulär ist, dann ist auch die Projektion $\mathcal{L}_\pi$ von $\mathcal{L}$ auf Σ regulär. Das Lemma erlaubt uns, induktiv Quantoren zu eliminieren.

4.4 Lemma *Sei $\varphi(X_1, \ldots, X_k) \in \mathrm{MSO}$, sei $\mathcal{K} = \{\mathcal{W} = (\mathcal{W}_w, A_1, \ldots, A_k) : \mathcal{W} \models \varphi\}$ und sei $\mathcal{L} = \mathrm{Lang}(\mathcal{K})$ die entsprechende Sprache über $\Sigma \times \{0,1\}^k$. Sei $\mathcal{L}'$ die Projektion von $\mathcal{L}$ auf $\Sigma \times \{0,1\}^{k-1}$. Dann ist $\mathrm{Models}(\mathcal{L}') = \{\mathcal{W}' = (\mathcal{W}_w, A_1, \ldots, A_{k-1}) : \mathcal{W}' \models \exists X_k\, \varphi\}$.*

Als direkte Folgerung erhalten wir das folgende Lemma.

4.5 Lemma *Definiert $\varphi(X_1, \ldots, X_k)$ eine reguläre Sprache über $\Sigma \times \{0,1\}^k$, so definiert $\exists X_k \varphi$ eine reguläre Sprache über $\Sigma \times \{0,1\}^{k-1}$.*

Unser nächstes Ziel ist es, Quantoren erster Stufe zu eliminieren. Wir definieren eine Logik MSO_0, die nur Mengenquantoren erlaubt. Wir werden zeigen, dass, eingeschränkt auf Wortmodelle, MSO_0 äquivalent zu MSO ist.

4.6 Definition (Syntax der Logik MSO_0) *Sei Σ ein Alphabet und sei σ_Σ die entsprechende Wortsignatur. Die Menge der atomaren Formeln der Logik MSO_0 über σ_Σ ist induktiv wie folgt definiert.*

- $X \subseteq Y \in \mathrm{MSO}_0$ *für alle* $X, Y \in \mathrm{mVar}_2$.
- $X \subseteq P_a \in \mathrm{MSO}_0$ *für alle* $X \in \mathrm{mVar}_2, P_a \in \sigma_\Sigma$.
- $\mathrm{Sing}(X) \in \mathrm{MSO}_0$ *für alle* $X \in \mathrm{mVar}_2$.
- $S(X,Y) \in \mathrm{MSO}_0$ *für alle* $X, Y \in \mathrm{mVar}_2$.
- $X \leqslant Y \in \mathrm{MSO}_0$ *für alle* $X, Y \in \mathrm{mVar}_2$.

Die Menge der MSO_0-Formeln ist nun der Abschluss der atomaren Formeln unter den Booleschen Operatoren $\neg, \wedge, \vee, \rightarrow, \leftrightarrow$, sowie unter Quantifizierung von Mengenvariablen $\exists X, \forall X$ für alle $X \in \mathrm{mVar}_2$.

Die Semantik von MSO_0 ist so definiert wie erwartet. Dabei soll $\mathrm{Sing}(X)$ bedeuten, dass X eine einelementige Menge ist, $S(X,Y)$ soll bedeuten, dass $X = \{x\}, Y = \{y\}$ und $s(x,y)$ gilt und $X \leqslant Y$ soll bedeuten, dass $X = \{x\}, Y = \{y\}$ und $x \leqslant y$.

4.7 Lemma *Jede MSO-Formel $\varphi(X_1, \ldots, X_k)$ ist über Wortmodellen äquivalent zu einer MSO_0-Formel.*

BEWEIS. Per Induktion über den Formelaufbau konstruieren wir zu $\varphi(y_1, \ldots, y_m, X_1, \ldots, X_k) \in$ MSO eine Formel $\varphi^*(Y_1, \ldots, Y_m, X_1, \ldots, X_k) \in \mathrm{MSO}_0$, welche äquivalent zu φ ist.

- $\varphi = s(y_i, y_j)$ wird übersetzt in $\varphi^* = S(Y_i, Y_j)$.
- $\varphi = (y_i = y_j)$ wird übersetzt in $\varphi^* = (Y_i \subseteq Y_j \wedge Y_j \subseteq Y_i)$.
- $\varphi = (y_i \leqslant y_j)$ wird übersetzt in $\varphi^* = (Y_i \leqslant Y_j)$.
- $\varphi = P_a(y_i)$ für $a \in \Sigma$ wird übersetzt in $\varphi^* = (Y_i \subseteq P_a)$.
- $\varphi = y_i \in Z$ für $Z \in \mathrm{mVar}_2$ wird übersetzt in $\varphi^* = Y_i \subseteq Z$.

Der Induktionsschritt für die Booleschen Operatoren und Mengenquantoren ist klar.

Sei nun die zu $\varphi(y_1, \ldots, y_m, X_1, \ldots, X_k)$ äquivalente Formel $\varphi^*(Y_1, \ldots, Y_m, X_1, \ldots, X_k)$ bereits konstruiert. Dann wird

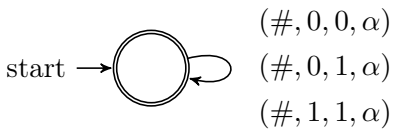
- $\exists y_m \varphi(y_1, \ldots, y_m, X_1, \ldots, X_k)$ übersetzt zu $\exists Y_m(\mathrm{Sing}(Y_m) \wedge \varphi^*)$ und
- $\forall y_m \varphi(y_1, \ldots, y_m, X_1, \ldots, X_k)$ wird übersetzt zu $\forall Y_m(\mathrm{Sing}(Y_m) \to \varphi^*)$. □

Es genügt im Folgenden also MSO_0-Formeln zu betrachten.

4.8 Lemma *Sei* $\varphi(X_1, \ldots, X_k) \in \mathrm{MSO}_0[\sigma_\Sigma]$ *und sei* $\mathcal{K} = \{\mathcal{W} = (\mathcal{W}_w, A_1, \ldots, A_k) : \mathcal{W} \models \varphi\}$. *Dann ist* $\mathrm{Lang}(\mathcal{K})$ *regulär.*

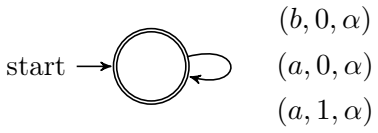
BEWEIS. (SKIZZE) Per Induktion über den Formelaufbau konstruieren wir aus einer Formel $\varphi(X_1, \ldots, X_k) \in \mathrm{MSO}_0$ einen NFA $\mathcal{A}_\varphi$, der die Sprache $\mathcal{L}(\varphi)$ akzeptiert. Wir gehen hierbei davon aus, dass die Formel $\varphi(X_1, \ldots, X_k)$ in einer Normalform gegeben ist, in welcher ausschließlich die Operatoren $\subseteq$, $\mathrm{Sing}(.)$, $S(.,.)$, $\leqslant$, $\neg$, $\vee$ und $\exists$ verwendet werden. Die Existenz dieser Normalform für MSO_0 lässt sich einfach per struktureller Induktion über den Formelaufbau beweisen. Zur Vereinfachung der Notation nehmen wir an, dass die atomaren Formeln nur die Variablen X_1 und X_2 benutzen (statt allgemeiner alle Variablen $X_1, \ldots, X_k$).

- Der Automat für $X_1 \subseteq X_2$ ist



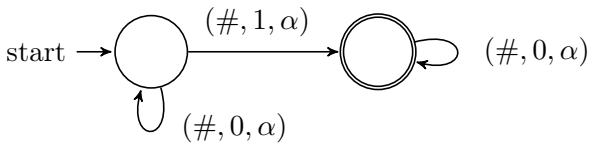
  Hier und im Folgenden steht # für beliebiges $a \in \Sigma$ und α für ein beliebiges Wort aus $\{0,1\}^{k-2}$ oder $\{0,1\}^{k-1}$.

- Der Automat für $X_1 \subseteq P_a$ ist

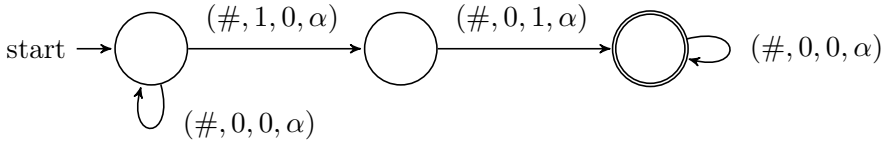


  In diesem Fall sei $b \in \Sigma \setminus \{a\}$.

- Der Automat für $\mathrm{Sing}(X_1)$ ist



- Der Automat für $S(X_1, X_2)$ ist



- Der Automat für $X_1 \leqslant X_2$ wird ähnlich konstruiert.

- Im Induktionsschritt seien für φ_1 und φ_2 die Automaten $\mathcal{A}_{\varphi_1}$ und $\mathcal{A}_{\varphi_2}$ schon konstruiert. Dann ist der Automat für $\neg\varphi_1$ der Komplementautomat von $\mathcal{A}_{\varphi_1}$, der Automat für $\varphi_1 \vee \varphi_2$ der Automat für $\mathcal{L}(\mathcal{A}_{\varphi_1}) \cup \mathcal{L}(\mathcal{A}_{\varphi_2})$ und für $\exists X_k \varphi_1(X_1, \ldots, X_k)$ bildet man den Projektionsautomaten von $\mathcal{A}_{\varphi_1}$.

Aus Lemma 4.7 und Lemma 4.8 folgt nun die Rückrichtung von Satz 4.1.

Aus unserem Wissen über reguläre Sprachen können wir nun leicht einige Eigenschaften von MSO ableiten. Das existentielle Fragment von MSO, geschrieben $\exists$MSO ist die Menge aller MSO-Formeln, die für die Mengenvariablen mVar_2 nur Existenzquantoren benutzt. Somit folgt aus unserer Konstruktion im Beweis zu Lemma 4.3 die folgende Aussage.

4.9 Lemma *Über Wortmodellen ist jede* MSO$[\sigma_\Sigma]$*-Formel äquivalent zu einer* $\exists$MSO*-Formel.*

4.10 Lemma MSO *kann nicht zählen, z.B. die Sprache* $\{a^n b^n : n \in \mathbb{N}\}$ *ist nicht* MSO*-definierbar. Addition ist nicht* MSO$[\{\leqslant\}]$*-definierbar.*

Weiterhin beobachten wir, dass die Übersetzung von Formeln in Automaten und die Übersetzung von Automaten in Formeln effektiv ist. Wir folgern, dass das Erfüllbarkeitsproblem und das Äquivalenzproblem für MSO auf Wortmodellen entscheidbar ist.

Das Erfüllbarkeitsproblem für MSO ist das Problem zu entscheiden, ob eine Formel $\varphi \in \mathrm{MSO}$ ein endliches Wortmodell hat.

Das Äquivalenzproblem ist das Problem zu zwei gegebenen Formeln $\varphi, \psi \in \mathrm{MSO}$ zu entscheiden, ob φ und ψ dieselben Wortmodelle haben.

Die Probleme sind allerdings nicht effizient zu lösen. Wir zitieren den Satz von Meyer und Stockmeyer ohne Beweis. Wir definieren die Tower-Funktion (Potenzturm) wie folgt. Sei $\mathrm{tow}(2,1,n) \coloneqq 2^n$ und $\mathrm{tow}(2,k+1,n) = 2^{\mathrm{tow}(2,k,n)}$.

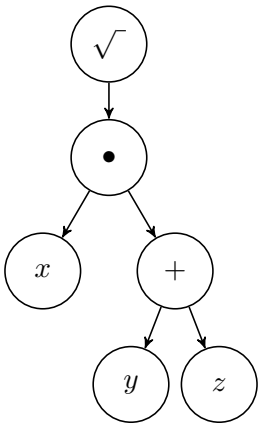
4.11 Satz (Meyer, Stockmeyer (1971)) *Für* kein *festes $k \in \mathbb{N}$ kann man zu einer beliebigen* MSO*-Formel φ die Zustandszahl des äquivalenten Automaten $\mathcal{A}_\varphi$ durch* $\mathrm{tow}(2,k,n)$ *beschränken, wobei $n = |\varphi|$.*

Kapitel 5.

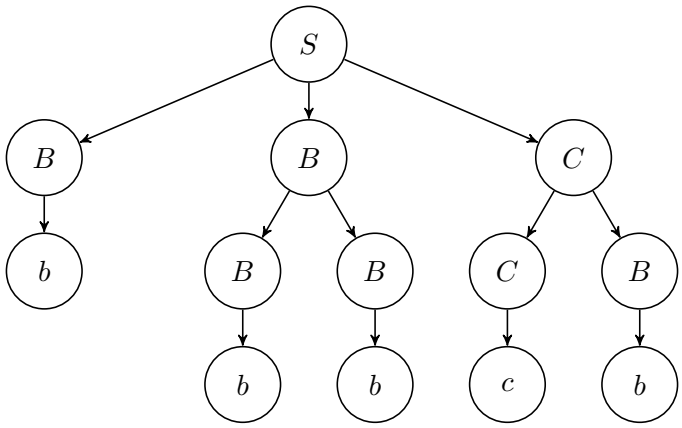
Baumautomaten

Viele Objekte der Informatik haben eine Baumstruktur. Ziel dieses Kapitels ist es, die MSO-Theorie der endlichen Wörter auf die Theorie der endlichen Bäume zu erweitern.

5.1 Beispiel Der Term $\sqrt{x \cdot (y + z)}$ hat die Baumstruktur



5.2 Beispiel Für die reguläre Grammatik $S \to BBC$, $B \to b \,|\, BB$, $C \to c \,|\, CB$ ergibt sich ein möglicher Ableitungsbaum für das Wort *bbbcb*.



5.1. Endliche Bäume und Baumautomaten

5.3 Definition *Ein* Rangalphabet *ist eine nicht leere endliche Menge* Γ *von Symbolen, wobei jedem* $a \in \Gamma$ *eine endliche Menge* $\mathrm{rg}(a) \subset \mathbb{N}$ *von* Rängen *oder* Stelligkeiten *zugeordnet wird. Zu* Γ *und* $i \in \mathbb{N}$ *definieren wir* $\Gamma_i \coloneqq \{a \in \Gamma : i \in \mathrm{rg}(a)\}$.

Sei $m = \max\{i : \Gamma_i \neq \emptyset\}$ der maximal auftretende Rang. Das Maximum existiert, da Γ und $\mathrm{rg}(a)$ endlich für alle $a \in \Gamma$ sind. Dann ist $\Gamma = \bigcup_{0 \leqslant i \leqslant m} \Gamma_i$.

5.4 Beispiel In Beispiel 5.1 ist $\Gamma = \{x, y, z, \surd, \cdot, +\}$, $\Gamma_0 = \{x, y, z\}$, $\Gamma_1 = \{\surd\}$ und $\Gamma_2 = \{\cdot, +\}$.

5.5 Definition *Sei* $\Gamma = \bigcup_{0 \leqslant i \leqslant m} \Gamma_i$ *ein Rangalphabet.*

1. *Die Menge aller* Γ-Bäume, *kurz* $\mathcal{T}_\Gamma$, *ist die kleinste Menge, die*
   - *jedes* $a \in \Gamma_0$ *und*
   - *für alle* $i > 0$ *und* $s \in \Gamma_i$ *und* $t_1, \ldots, t_i \in \mathcal{T}_\Gamma$ *auch* $s(t_1, \ldots, t_i)$ *enthält.*

   Γ*-Bäume werden also induktiv konstruiert.*

2. *Eine* Γ-Baumsprache *oder* Baumsprache *über* Γ *ist eine Teilmenge* $L \subseteq \mathcal{T}_\Gamma$.

Die algebraische Schreibweise ist oft praktisch (insbesondere für die Repräsentation eines Baumes für einen Algorithmus). Wir fassen Bäume dennoch oft als beschrifteten Graphen auf und zeichnen den Graphen wie in den obigen Beispielen. Eine Beschriftung von $T \in \mathcal{T}_\Gamma$ ist eine Funktion $\beta : V(T) \to \Gamma$. Dabei benutzen wir für T stets eine Knotenmenge $V(T) \subseteq \{1, \ldots, m\}^*$, die unter Präfixen abgeschlossen ist, d.h.

(i) falls $ui \in V(T)$, dann auch $u \in V(T)$ für $u \in \{1, \ldots, m\}^*, i \in \{1, \ldots, m\}$ und

(ii) falls $ui \in V(T)$ und $1 \leqslant j < i$, dann auch $uj \in V(T)$ für $u \in \{1, \ldots, m\}^*, i \in \{1, \ldots, m\}$ und

(iii) falls $u \in V(T)$ und u genau i Nachfolger hat, dann ist $\beta(u) \in \Gamma_i$.

Wir definieren Baumautomaten nun ähnlich zu Wortautomaten. Ein wichtiger Unterschied ist, dass wir Läufe auf verschiedene Arten definieren können.

5.6 Definition *Ein* nichtdeterministischer Bottom-Up-Baumautomat *(*$\mathrm{NBA}_\uparrow$*)* $\mathcal{A}$ *ist ein Tupel* $\mathcal{A} = (Q, \Gamma, \Delta, F)$, *wobei*

- Q *eine endliche Zustandsmenge ist,*
- Γ *ein Rangalphabet ist,*
- $F \subseteq Q$ *eine Menge von Endzuständen ist und*

- $\Delta \subseteq \bigcup_{0 \leqslant i \leqslant m} (Q^i \times \Gamma_i \times Q)$ *ist, wobei* m *der maximale Rang von* Γ *ist (und* $Q^0 \times \Gamma_0 \times Q \cong \Gamma_0 \times Q$*)*

$\mathcal{A}$ *ist* deterministisch *(*$\mathrm{DBA}_\uparrow$*), wenn* Δ *funktional ist, d.h. für alle* $0 \leqslant i \leqslant m$ *und* $q_1, \ldots, q_i \in Q$ *und* $a \in \Gamma_i$ *genau ein* q *existiert mit* $((q_1, \ldots, q_i), a, q) \in \Delta$*.*

Wir präsentieren Δ oft als eine Liste von Regeln $((q_1, \ldots, q_i), a) \to q$.

5.7 Definition *Sei* Γ *ein Rangalphabet,* $\mathcal{A}$ *ein* $\mathrm{NBA}_\uparrow$ *und* $T \in \mathcal{T}_\Gamma$*. Ein* Lauf *von* $\mathcal{A}$ *auf* T *ist eine Abbildung* $\rho : V(T) \to Q$ *so dass*

- *für alle Blätter* $t \in V(T)$ *gilt:* $(\beta(t), \rho(t)) \in \Delta$ *und*
- *für alle* $t \in V(T)$ *und* $i > 0$ *mit* $\beta(t) \in \Gamma_i$ *und Nachfolgern* $(t_1, \ldots, t_i)$ *gilt:*

$$\Big(\big(\rho(t_1), \ldots, \rho(t_i)\big), \beta(t), \rho(t) \Big) \in \Delta.$$

Der Lauf ρ *ist* akzeptierend*, wenn* $\rho(w) \in F$ *für die Wurzel* w *von* T*. Die von* $\mathcal{A}$ akzeptierte Sprache $\mathcal{L}(\mathcal{A}) \subseteq \mathcal{T}_\Gamma$ *ist die Menge der Bäume* $T \in \mathcal{T}_\Gamma$*, so dass* $\mathcal{A}$ *einen akzeptierenden Lauf auf* T *hat. Eine Baumsprache* $L \subseteq \mathcal{T}_\Gamma$ *ist* regulär*, wenn es einen* $\mathrm{NBA}_\uparrow$ *gibt, der* L *akzeptiert.*

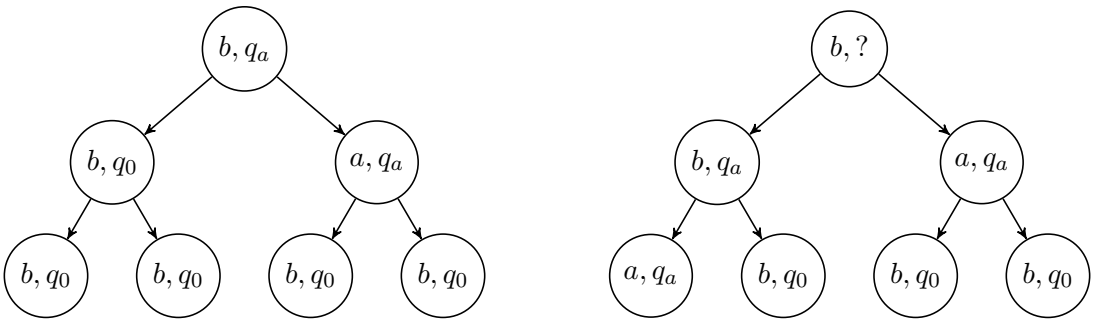
5.8 Beispiel Sei $\Gamma = \{0, 1, +, \cdot\}$, mit $\mathrm{rg}(0) = \mathrm{rg}(1) = \{0\}$ und $\mathrm{rg}(+) = \mathrm{rg}(\cdot) = \{2\}$. Wir definieren einen Automaten, der alle Terme über Γ akzeptiert, die zu $0 \bmod 3$ auswerten.
Wähle

- $Q = \{q_0, q_1, q_2\}$, $F = \{q_0\}$ und
- $\Delta = \{0 \mapsto q_0, 1 \mapsto q_1, ((q_i, q_j), +) \mapsto q_{i+j \mod 3}, ((q_i, q_j), \cdot) \mapsto q_{i \cdot j \mod 3}\}$.

5.9 Beispiel Sei $\Gamma = \{a, b\}$, $\mathrm{rg}(a) = \mathrm{rg}(b) = \{0, 1, 2\}$. Sei $L \subseteq \mathcal{T}_\Gamma$ die Sprache der Bäume, die genau ein a enthalten. Wir definieren einen Automaten, der L akzeptiert.

- $Q = \{q_0, q_a\}$, $F = \{q_a\}$,
- $\Delta = \{b \mapsto q_0, a \mapsto q_a, (q_0, a) \mapsto q_a, (q_0, b) \mapsto q_0, ((q_0, q_0), a) \mapsto q_a, ((q_a, q_0), b) \mapsto q_a,$ $((q_0, q_0), b) \mapsto q_0, ((q_0, q_a), b) \mapsto q_a\}$.

Beachte, dass z.B. keine Transition für (q_a, a) definiert ist, der nichtdeterministische Automat also "stecken bleibt" und nicht akzeptieren wird, wenn er auf diese Beschriftung trifft. Wir geben auf den folgenden Bäumen die Beschriftung β an erster Stelle und die Beschriftung, die sich aus dem Lauf ρ ergibt an zweiter Stelle an.



Die folgenden Sätze geben wir ohne Beweis an. Die Beweise sind analog zu den entsprechenden Sätzen für Wortautomaten.

5.10 Satz *Zu jedem* $\mathrm{NBA}_\uparrow$ $\mathcal{A}$ *kann man effektiv einen* $\mathrm{DBA}_\uparrow$ $\mathcal{B}$ *definieren mit* $\mathcal{L}(\mathcal{A}) = \mathcal{L}(\mathcal{B})$.

5.11 Satz *Die Klasse der regulären Baumsprachen ist unter Komplement, Vereinigung, Schnitt und Projektion abgeschlossen.*

5.12 Satz *Das Leerheitsproblem für* $\mathrm{NBA}_\uparrow$ *ist in Polynomialzeit entscheidbar.*

5.13 Definition *Sei* Γ *ein Rangalphabet und* m *der maximale Rang von* Γ. *Ein* nichtdeterministischer Top-Down-Automat *(*$\mathrm{NBA}_\downarrow$*)* $\mathcal{A}$ *ist ein Tupel*

$$\mathcal{A} = (Q, \Gamma, Q_0, \Delta),$$

wobei

- Q *eine endliche Zustandsmenge ist,*
- $Q_0 \subseteq Q$ *eine Menge von Startzuständen ist und*
- $\Delta \subseteq \bigcup_{0 \leqslant i \leqslant m} (Q \times \Gamma_i \times Q^i)$ *eine Transitionsrelation ist.*

5.14 Definition *Ein Lauf von* $\mathcal{A}$ *auf* $T = (V(T), \beta)$ *ist eine Abbildung* $\rho : V(T) \to Q$ *mit*

- $\rho(\varepsilon) \in Q_0$
- *für* $u \in V(T)$ *mit Nachfolgern* $(u1, \ldots, ui)$, $i > 0$ *ist* $\Big(\rho(u), \beta(u), \big(\rho(u1), \ldots, \rho(ui)\big)\Big) \in \Delta$ *und*
- *für ein Blatt* $u \in V(T)$ *ist* $\big(\rho(t), \beta(t)\big) \in \Delta$.

$\mathcal{A}$ akzeptiert T, wenn es einen Lauf gibt. (Es wird keine explizite Endzustandsmenge benötigt!) Ein $\text{NBA}_\downarrow$ *ist* deterministisch *(ein* $\text{DBA}_\downarrow$*), wenn $Q_0 = \{q_0\}$ und Δ funktional ist.*

Anders als für Bottom-Up-Automaten haben deterministische und nichtdeterministische Top-Down-Automaten nicht dieselbe Mächtigkeit.

5.15 Lemma *Die Baumsprache $L = \{f(a,b), f(b,a)\}$ ist* $\text{NBA}_\downarrow$*-, aber nicht* $\text{DBA}_\downarrow$*-erkennbar.*

Beweis. Offensichtlich ist L $\text{NBA}_\downarrow$-erkennbar. Für die zweite Aussage nehmen wir an, es existiert ein $\text{DBA}_\downarrow$ $\mathcal{A} = (Q, \Gamma, \{q_0\}, \Delta)$, der L erkennt. Da $\mathcal{A}$ den Baum $f(a,b)$ akzeptiert, existiert ein Lauf ρ mit $\rho(\varepsilon) = q_0$, $\rho(1) = q_1$ und $\rho(2) = q_2$ für $q_1, q_2 \in Q$, so dass $((q_0, f), (q_1, q_2)) \in \Delta$ die eindeutige Transition für (q_0, f) ist und so dass $(q_1, a), (q_2, b) \in \Delta$. Da $\mathcal{A}$ den Baum $f(b,a)$ akzeptiert, und $((q_0, f), (q_1, q_2)) \in \Delta$ die eindeutige Transition für (q_0, f) ist, folgt, dass auch $(q_1, b), (q_2, a) \in \Delta$ gilt. Dann akzeptiert $\mathcal{A}$ aber auch den Baum $f(a,a) \notin L$. Also akzeptiert $\mathcal{A}$ entgegen unserer Annahme nicht L, ein Widerspruch. □

5.16 Satz *Die durch* $\text{DBA}_\downarrow$ *erkannte Sprachklasse ist echt kleiner als die durch* $\text{NBA}_\downarrow$ *erkannte Sprachklasse.*

Wiederum ohne Beweis geben wir folgenden Satz an.

5.17 Satz $\text{NBA}_\downarrow$*,* $\text{NBA}_\uparrow$ *und* $\text{DBA}_\uparrow$ *erkennen dieselbe Sprachklasse.*

5.2. Äquivalenz von MSO und Baumautomaten

Wir erhalten dann alle Sätze, die wir für MSO auf Worten gezeigt haben, auch für MSO auf Bäumen. Wir erweitern zunächst die monadische Logik zweiter Stufe dahingehend, dass ihre Formeln über Bäumen interpretiert werden.

5.18 Definition *Sei Γ ein Rangalphabet und sei m der maximale Rang von Γ. Wir definieren das Vokabular $\sigma_\Gamma \coloneqq \{(P_a)_{a\in\Gamma}, S_1, \ldots, S_m, \leqslant\}$, wobei die P_a für $a \in \Gamma$ einstellige Relationssymbole und die S_i und $\leqslant$ zweistellige Relationssymbole sind. Wir interpretieren einen beschrifteten Baum $(T, \beta) \in \mathcal{T}_\Gamma$ als σ_Γ-Struktur, wobei $(s,t) \in S_i$ wenn t der i-te Nachfolger von s ist und $\leqslant$ als Präfixordnung interpretiert wird, d.h. $s \leqslant t$ wenn der eindeutige Pfad in T von der Wurzel zu t durch s läuft.*

Wir schließen diesen Teil der Vorlesung ab mit dem folgenden Satz.

5.19 Satz (Doner, und unabhängig Thatcher and Wright, 1968) *Eine Baumsprache $\mathcal{L}$ ist genau dann regulär, wenn sie MSO-definierbar ist.*

Teil II.

Auswerten von Formeln

Kapitel 6.

Komplexität logischer Auswertungsprobleme

Ziel dieses Kapitels ist die Untersuchung der Komplexität der Auswertung logischer Formeln in Strukturen. Wir nennen zwei wichtige Anwendungen, in denen logische Formeln ausgewertet werden (müssen).

- In der Datenbanktheorie bildet die Prädikatenlogik den Kern der Anfragesprache SQL auf relationalen Datenbanken. Abgesehen von zusätzlichen Zählfunktionen kann die Auswertung einer SQL-Anfrage interpretiert werden als die Auswertung einer prädikatenlogischen Formeln auf einer Datenbank, interpretiert als relationale Struktur. Die effiziente Auswertung solcher Formeln ist also essentiell wichtig. Wir werden sehen, dass bereits die Auswertung existentieller Formeln NP-vollständig ist, das allgemeine Auswertungsproblem ist PSPACE-vollständig.
- In der parametrisierten Komplexität wird das model-checking Problem der Prädikatenlogik und monadischen Logik zweiter Stufe untersucht. Beide Logiken können wichtige NP-Probleme und PSPACE-Probleme formulieren. Man erhält Komplexitätsresultate nicht nur für spezielle Probleme, sondern für ganze Klassen von Problemen. Häufig wird auch die Graphklasse eingeschränkt, auf der das model-checking Problem gelöst werden soll. So erhält man für viele Klassen eine reichhaltige Strukturtheorie, die das effiziente Lösen von schweren Graphproblemen erlauben. Wir werden Courcelles Satz kennenlernen, der besagt, dass jede MSO-Eigenschaft von Graphen (z.B. 3-Färbbarkeit) auf Graphen mit beschränkter Baumweite in Linearzeit gelöst werden kann.

Im Folgenden sei σ eine Signatur.

6.1 Definition *Sei $\mathcal{C}$ eine Klasse von σ-Strukturen und sei $\mathcal{L}$ eine Logik. Das* Auswertungsproblem *von $\mathcal{L}$ auf $\mathcal{C}$,* $\mathbf{MC}(\mathcal{L}, \mathcal{C})$*, ist das Problem zu einer gegebenen Struktur $\mathcal{A} \in \mathcal{C}$ und Formel $\varphi \in \mathcal{L}$ zu entscheiden, ob $\mathcal{A} \models \varphi$ gilt.*

Wir haben bereits gesehen, dass sich in MSO NP-vollständige Probleme ausdrücken lassen. Also ist $\mathbf{MC}$(MSO) mindestens NP-schwer. Wir werden zeigen, dass es sogar PSPACE-vollständig ist.

Zur Erinnerung: PSPACE ist die Klasse aller Probleme/Sprachen, die durch eine polynomiell platzbeschränkte deterministische Turingmaschine entschieden werden können. NPSPACE ist die Klasse aller Probleme/Sprachen, die durch eine polynomiell platzbeschränkte nicht-deterministische Turingmaschine entschieden werden können. Polynomielle Platzbeschränkung bedeutet, dass zu jeder Eingabe der Länge n höchstens $p(n)$ viele Zellen auf dem Arbeitsband besucht werden (für ein Polynom p). Diese Zellen dürfen aber beliebig oft besucht werden.

Savitch hat 1970 gezeigt, dass jedes von einer nicht-deterministischen Turingmaschine mit Platz $s(n)$ lösbare Problem auf einer deterministischen Turingmaschine mit Platz $s(n)^2$ gelöst werden kann. Es folgt sofort, dass PSPACE=NPSPACE.

Wir zeigen zunächst den folgenden Satz.

6.2 Satz *Das model-checking Problem für* MSO*,* $\mathbf{MC}(\mathrm{MSO}, \mathcal{C})$*, ist in* PSPACE *lösbar für jede Klasse* $\mathcal{C}$ *von endlichen* σ*-Strukturen.*

BEWEIS. Sei $\mathcal{A}$ eine Struktur mit n Elementen und sei $\psi(X_1, \ldots, X_m, x_1, \ldots, x_n) \in \mathrm{MSO}$. Seien $U_1 \subseteq A, \ldots, U_m \subseteq A, u_1 \in A, \ldots, u_n \in A$ Belegungen für die freien Variablen. Wir werten die Formel rekursiv aus.

- Falls ψ quantorenfrei ist können wir sie in Linearzeit auswerten.
- Falls $\psi = \exists X_{m+1}\, \psi'$, so testen wir für alle $U_{m+1} \subseteq A$ ob $\psi'(U_1, \ldots, U_{m+1}, u_1, \ldots, u_n)$ gilt. Wenn ein ψ' zu wahr auswertet, so akzeptieren wir. Wir verwerfen, wenn kein ψ' zu wahr auswertet.
- Falls $\psi = \forall X_{m+1}\psi'$, so testen wir für alle $U_{m+1} \subseteq A$ ob $\psi'(U_1, \ldots, U_{m+1}, u_1, \ldots, u_n)$ gilt. Wenn ein ψ' zu falsch auswertet, so verwerfen wir. Wir akzeptieren, wenn alle ψ' zu wahr auswertet.
- Falls $\psi = \exists x_{n+1}\, \psi'$, so testen wir für alle $u_{n+1} \in A$ ob $\psi'(U_1, \ldots, U_m, u_1, \ldots, u_{n+1})$ gilt. Wenn ein ψ' zu wahr auswertet, so akzeptieren wir. Wir verwerfen, wenn kein ψ' zu wahr auswertet.
- Falls $\psi = \forall x_{n+1}\psi'$, so testen wir für alle $u_{n+1} \in A$ ob $\psi'(U_1, \ldots, U_m, u_1, \ldots, u_{n+1})$ gilt. Wenn ein ψ' zu falsch auswertet, so verwerfen wir. Wir akzeptieren, wenn alle ψ' zu wahr auswertet.

Beachte, dass auf diese Weise eine „Tiefensuche durch die Formel“ ausgeführt wird. Falls die Formel q Mengenquantoren und k Elementquantoren hat, müssen zu jedem Zeitpunkt also höchstens q Mengen und k Elemente gespeichert werden. Es muss weiterhin eine Funktion implementiert werden, die Mengen so ordnet, dass aus jeder Menge die nächste Menge errechnet werden kann. Somit muss nicht gespeichert werden welche Mengen schon ausgewertet wurden. Der Speicherbedarf des Algorithmus ist somit $\mathcal{O}(n \log n)$ für jeden Mengenquantor (die Kodierung jedes Elements braucht $\log n$ viel Speicher) und

$\mathcal{O}(\log n)$ für jeden Elementquantor. Insgesamt ist der Speicherbedarf also beschränkt durch $\mathcal{O}(|\psi| \cdot n \log n)$. □

In der Hausaufgabe haben wir das Konzept der alternierenden Turingmaschinen eingeführt und gezeigt, dass obiger Algorithmus von einer alternierenden Turingmaschine in Polynomialzeit ausgeführt werden kann. Wir haben gezeigt, dass APTime = PSPace. Alternierende Turingmaschinen sind oft das natürlichere Maschinenmodell für die Auswertung von Formeln.

Mit folgender Definitionen und folgendem Satz ist es eine leichte Übung die passende untere Schranke, nämlich, dass $\mathbf{MC}(\mathrm{FO}, \mathcal{C})$ PSPace-hart ist auf der Klasse $\mathcal{C}$ aller σ-Strukturen. Es genügt sogar eine Klasse $\mathcal{C}$ mit einer einzigen Struktur mit zwei Elementen und einer unären Relation zu betrachten. Dazu reduzieren wir das model-checking Problem auf das PSPace-vollständige Problem *Auswertung quantifizierter Boolescher Formeln* (QBF).

6.3 Definition *Eine* quantifizierte Boolesche Formel *ist eine Formel der Form*

$$Q_1 X_1 \dots Q_n X_n \psi,$$

wobei ψ eine aussagenlogische Formel mit Variablen $X_1, \dots, X_n$ und $Q_i \in \{\forall, \exists\}$ für $1 \leqslant i \leqslant n$.

Die Semantik für quantifizierte Boolesche Formeln ist induktiv definiert wie erwartet. Jede Formel wertet entweder zu wahr oder zu falsch aus.

6.4 Definition QBF *ist das Problem zu einer gegebenen quantifizierten Booleschen Formel φ zu entscheiden ob φ wahr ist.*

6.5 Satz QBF *ist* PSPace*-vollständig.*

Im Allgemeinen können wir also nicht effizient testen, ob eine FO- oder MSO-Formel auf einer Struktur gilt. Dennoch können wir eine feinere Analyse vornehmen und in vielen Fällen eine effiziente Lösung finden. Dazu untersuchen wir zunächst, ob die Härte des Problems aus der Formel oder aus der Struktur entsteht. Dieser Ansatz wurde von Vardi 1982 vorgeschlagen.

Wir unterscheiden zwischen der *kombinierten Komplexität (combined complexity)*, *Datenkomplexität (data complexity)* und *Formelkomplexität (expression complexity)* des Model-checking Problems. Für die kombinierte Komplexität hat der Model-checking Algorithmus als Eingabe eine Struktur $\mathcal{A}$ und eine Formel φ und wir fragen nach der Komplexität des Problems bzgl. der Größe $|\mathcal{A}| + |\varphi|$. Für die Datenkomplexität fixieren wir φ, d.h. der Algorithmus hat als Eingabe nur eine Struktur $\mathcal{A}$ und fragen nach der Komplexität des Problems bzgl. der Größe $|\mathcal{A}|$. Eine Laufzeit von $|\mathcal{A}|^{|\varphi|}$ ist für festes φ hier also polynomiell. Für die Formelkomplexität betrachten wir entsprechend die Struktur $\mathcal{A}$ als fest.

6.6 Satz

- *Die kombinierte Komplexität und Formelkomplexität von* FO *ist* PSPACE*-vollständig.*
- *Die Datenkomplexität von* FO *ist in* PTIME *(*$\mathcal{O}(|\mathcal{A}|^{|\psi|})$.

Für MSO ist die Datenkomplexität nicht in PTIME, da schon feste Formeln NP-schwere Probleme beschreiben können.

Z.B. um die Komplexität einer Datenbankanfrage zu beschreiben eignet sich die Datenkomplexität eher als die kombinierte Komplexität oder die Formelkomplexität, da die Formel, die die Datenbankanfrage beschreibt, im Vergleich zur Datenbank verschwindend klein ist. In der Praxis ist aber selbst eine polynomielle Auswertung für große Datenbanken zu langsam, Linearzeitanfragen sind die einzig realistisch auswertbaren Anfragen.

Kapitel 7.

Parametrische Komplexität

Wir greifen das Datenbankbeispiel aus dem letzten Kapitel auf. Dort haben wir festgestellt, dass sich die Datenkomplexität am besten eignet um die Komplexität einer praktischen Datenbankanfrage zu beschreiben, da die Formeln sehr klein sind im Vergleich zur Datenbank. In der parametrischen Komplexität möchten wir eine noch feinere Analyse vornehmen und die Komplexität eines Problems bezüglich eines oder mehrerer Parameter ausdrücken.

7.1 Definition *Ein parametrisiertes Problem über einem Alphabet Σ ist eine Sprache $\mathcal{L} \subseteq \Sigma^* \times \mathbb{N}$. Für eine Instanz (w, k) nennen wir k den Parameter.*

7.2 Beispiel Das parametrisierte FO-model-checking Problem p-$\mathbf{MC}(\mathrm{FO}, \mathcal{C})$ hat als Eingabe eine (kodierte) Struktur $\mathcal{A} \in \mathcal{C}$ und eine (kodierte) Formel φ. Der Parameter ist $|\varphi|$. Das Problem ist zu bestimmen ob $\mathcal{A} \models \varphi$. Formal ist

$$p\text{-}\mathbf{MC}(\mathrm{FO}, \mathcal{C}) = \{(enc(\mathcal{A}) \# enc(\varphi), k) : k = |\varphi|, \mathcal{A} \text{ } \sigma\text{-Struktur}, \varphi \in \mathrm{FO}[\sigma] \text{ und } \mathcal{A} \models \varphi\}$$

für eine geeignete Kodierungsfunktion enc.

7.3 Beispiel Wir können viele Entscheidungsprobleme parametrisieren durch die Größe der Lösung. So hat etwa das parametrisierte dominating set Problem als Eingabe einen Graphen G und $k \in \mathbb{N}$, der Parameter ist k und das Problem ist zu bestimmen ob G ein dominating set der Größe höchstens k hat. Dies ist eine sinnvolle Parametrisierung immer dann, wenn wir annehmen können, dass die Lösung klein ist im Vergleich zum Eingabegraphen.

7.4 Definition *Ein parametrisiertes Problem ist* fixed-parameter tractable*, wenn es ein $c \in \mathbb{N}$ und eine berechenbare Funktion $f : \mathbb{N} \to \mathbb{N}$ und einen Algorithmus gibt, der auf Eingabe (w, k) in Zeit $f(k) \cdot |w|^c$ entscheidet, ob $(w, k) \in \mathcal{L}$.*

Wir schreiben FPT *für die Klasse aller parametrisierten Probleme, die fixed-parameter tractable (fpt) sind.*

Achtung: die Zugehörigkeit eines Problems zur Klasse FPT hängt von der Parametrisierung ab.

In der parametrisierten Komplexität sprechen wir von einem effizient lösbaren Problem, wenn das Problem fpt ist. Die Klasse FPT nimmt also die Rolle von PTime in der klassischen Komplexität ein. Die Rolle von NP wird übernommen von der sogenannten W-Hierarchie. Es gilt $\text{FPT} \subseteq W[1] \subseteq W[2] \subseteq \ldots$. Wir geben keine formale Definition der W-Hierarchie an, sondern beschränken uns auf die Angabe einiger Beispiele für vollständige Probleme für die untersten Stufen der Hierarchie. Die Probleme sind jeweils parametrisiert durch die Größe der Lösung. Das Vertex-cover Problem liegt in FPT, das independent set Problem ist hart für $W[1]$ und das dominating set Problem ist hart für $W[2]$.

Härte für eine Klasse wird über fpt-Reduktionen definiert.

7.5 Definition *Seien $\mathcal{P} \subseteq \Sigma^* \times \mathbb{N}$ und $\mathcal{Q} \subseteq \Gamma^* \times \mathbb{N}$ parametrisierte Probleme. Eine fpt-Reduktion von $\mathcal{P}$ auf $\mathcal{Q}$ ist ein Tupel (A, f, g, c), wobei $c \in \mathbb{N}$, $f, g : \mathbb{N} \times \mathbb{N}$ berechenbare Funktionen sind und A ein Algorithmus ist, der auf Eingabe $(w, k) \in \Sigma^* \times \mathbb{N}$ ein Wort $(w', k') \in \Gamma^* \times \mathbb{N}$ in Zeit $f(k) \cdot |w|^c$ berechnet, so dass*

1. *$(w, k) \in \mathcal{P} \iff (w', k') \in \mathcal{Q}$ und*
2. *$k' \leqslant g(k)$.*

Wir schreiben $\mathcal{P} \leqslant_{fpt} \mathcal{Q}$.

Wie in der klassischen Komplexitätstheorie werden Reduktionen benutzt um Komplexitätsaussagen für bekannte Probleme zu übertragen auf andere Probleme.

7.6 Lemma *Seien $\mathcal{P} \subseteq \Sigma^* \times \mathbb{N}$ und $\mathcal{Q} \subseteq \Gamma^* \times \mathbb{N}$ parametrisierte Probleme.*

1. *Wenn $\mathcal{Q} \in \text{FPT}$ und $\mathcal{P} \leqslant_{fpt} \mathcal{Q}$, dann auch $\mathcal{P} \in \text{FPT}$.*
2. *Wenn $\mathcal{P}$ $W[i]$-hart ist und $\mathcal{P} \leqslant_{fpt} Q$, dann ist auch $\mathcal{Q}$ $W[i]$-hart für alle $i \in \mathbb{N}$.*

7.7 Definition *Ein parametrisiertes Problem $\mathcal{P} \subseteq \Sigma^* \times \mathbb{N}$ ist in XP, wenn es eine berechenbare Funktion $f : \mathbb{N} \to \mathbb{N}$ und einen Algorithmus gibt, der $\mathcal{P}$ in Zeit $|w|^{f(k)}$ entscheidet.*

Wir beobachten, dass das parametrisierte Model-checking Problem für FO in XP liegt. Wir beobachten weiterhin, dass viele natürliche Probleme in ihrer Standardparametrisierung (Parameter enspricht der Größe der Lösung) in FO definierbar sind und somit eine Standardreduktion auf das FO-Model-checking Problem vorliegt. Wir verallgemeinern dieses Konzept wie folgt.

7.8 Definition *Sei $\mathcal{L}$ eine Logik und σ eine Signatur. Wir schreiben σ-Struct für die Menge aller σ-Strukturen und fixieren eine geeignete Kodierung über Σ. Ein parametrisiertes Problem $\mathcal{P} \subseteq \sigma$-Struct $\times \mathbb{N}$ ist parametrisiert $\mathcal{L}$-definierbar, wenn ein Algorithmus*

A existiert, der auf Eingabe k eine Formel φ_k berechnet, so dass für alle Eingaben $(\mathcal{A}, k)$ gilt

$$(\mathcal{A}, k) \in P \iff \mathcal{A} \models \varphi_k.$$

Da φ_k nur von k abhängt, berechnet der Algorithmus A in obiger Definition eine fpt-Reduktion von $\mathcal{P}$ auf das parametrisierte $\mathcal{L}$ Model-checking Problem. Wir folgern, dass das parametrisierte FO-Model-checking Problem nicht fpt ist, da etwa das parametrisierte dominating set Problem FO-definierbar ist und dieses Problem $W[2]$-hart ist. Genauer, p-**MC**(FO) ist vollständig für die parametrisierte Klasse AW[$*$] und es gilt $W[i] \subseteq \text{AW}[*]$ für alle $i \in \mathbb{N}$.

Wir werden uns im folgenden Kapitel mit der Frage beschäftigen, ob es eingeschränkte Klassen gibt, auf denen das parametrisierte FO- oder MSO-Model-checking Problem fpt ist und ob wir diese Klassen möglicherweise genau charakterisieren können.

Kapitel 8.

Baumweite und der Satz von Courcelle

In diesem Kapitel lernen wir die Baumweite kennen, die ein strukturelles Maß für die Ähnlichkeit eines Graphen zu einem Baum misst. Graphen mit beschränkter Baumweite sind eine wichtige Klasse von Graphen, auf denen das parametrisierte MSO model-checking fixed-parameter tractable ist (sogar fixed-parameter linear).

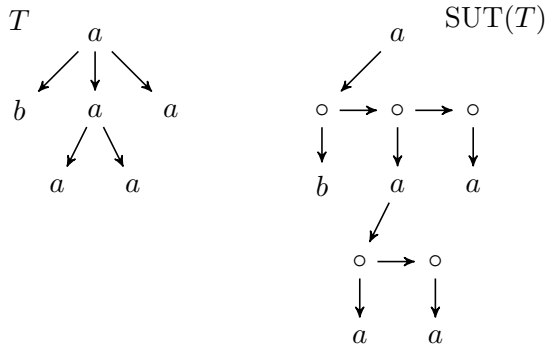
8.1. Algorithmen auf Bäumen

Wir möchten zunächst zeigen, dass MSO_2 model-checking auf Bäumen fixed-parameter tractable ist. MSO_2 ist definiert wie MSO, erlaubt aber zusätzlich die Quantifizierung über Teilmengen der Kantenrelation. Den Ansatz werden wir erweitern für das model-checking auf Graphen mit beschränkter Baumweite. Die Idee ist, eine MSO_2-Formel in einen Baumautomaten zu übersetzen und dann auf dem Baum laufen zu lassen. Dazu kodieren wir Bäume als *Sibling Unranked Trees*, kurz SUTs.

Sei Σ ein endliches Alphabet und sei T ein Σ-beschrifteter Wurzelbaum mit beliebiger Stelligkeit. Ein Sibling Unranked Tree ist ein gerichteter Binärbaum über $\Sigma' \coloneqq \Sigma \dot\cup \{\circ\}$.

- Jeder Knoten, der mit $a \in \Sigma$ beschriftet ist, ist entweder ein Blatt oder hat einen $\circ$-Knoten als einzigen Nachfolger.
- Jeder $\circ$-Knoten hat genau einen Σ-Nachfolger und höchstens einen $\circ$-Nachfolger.

So kodieren wir z.B. den Baum T folgendermaßen.



Sei nun $\sigma = \{E\}$ oder $\sigma = \{V, E, I\}$ und sei $\varphi \in \mathrm{MSO}[\sigma]$. Jeder Σ-beschriftete unbeschränkte Baum kann wie üblich als σ-Struktur $\mathcal{A}_T$ kodiert werden. Außerdem kann φ übersetzt werden in eine Formel $\varphi' \in \mathrm{MSO}[\sigma_{\Sigma'}, E_1, E_2]$, so dass

$$\mathrm{SUT}(T) \models \varphi' \iff \mathcal{A}_T \models \varphi.$$

Hier steht E_1 für den linken und E_2 für den rechten Nachfolger. Beachte, dass $\mathrm{SUT}(T)$ die Kinder eines Knotens ordnet, während T ein ungeordneter Baum ist. Umgekehrt kann man also nicht jede Formel $\varphi' \in \mathrm{MSO}[\sigma_{\Sigma'}, E_1, E_2]$ in eine Formel $\varphi \in \mathrm{MSO}[\sigma]$ mit obiger Eigenschaft übersetzt werden. Beachten Sie außerdem, dass auch eine FO-Formel in eine MSO-Formel und nicht in eine MSO-Formel übersetzt wird.

8.1 Satz *Das parametrisierte* MSO_2 *Model-Checking-Problem auf endlichen Bäumen ist fixed-parameter tractable.*

Beweis. Sei $\varphi \in \mathrm{MSO}_2$ und T ein Σ-Baum.

- Wir berechnen $T' = \mathrm{SUT}(T)$ bzw. wir nehmen an, dass T schon durch Nachfolgerlisten kodiert ist.
- Wir transformieren φ in eine Formel φ', so dass

  $$T' \models \varphi' \iff T \models \varphi.$$

  Die Laufzeit hierfür ist $f(|\varphi|)$ für eine Funktion $f\colon \mathbb{N} \to \mathbb{N}$ und hängt somit nur von der Formel, genauer von der Länge der Formel, ab.
- Wir wandeln φ' um in einen $\mathrm{DBA}_\uparrow$ $\mathcal{A}_{\varphi'}$ so dass

  $$T' \models \varphi' \iff T' \in \mathcal{L}(\mathcal{A}_{\varphi'}).$$

  Die Laufzeit hierfür ist $f'(|\varphi'|)$ für eine Funktion $f'\colon \mathbb{N} \to \mathbb{N}$.
- Wir verifizieren $T' \in \mathcal{L}(\mathcal{A}_{\varphi'})$. Die Laufzeit hierfür ist $f''(|\mathcal{A}_{\varphi'}|) \cdot n$ für eine Funktion $f''\colon \mathbb{N} \to \mathbb{N}$. Dabei bezeichnet n die Größe des Baumes. □

8.2. Baumweite

Wir werden unser letztes Ergebnis nun erweitern auf die Klasse der Graphen mit beschränkter Baumweite. Wir identifizieren die Eigenschaft eines Baumes, dass jeder Knoten ein Trenner des Baumes ist, als die relevante Eigenschaft, die effiziente dynamische Programmierung bzw. den Baumautomatenansatz erlaubt. Diese Trennereigenschaften wollen wir erweitern.

8.2 Definition *Sei G ein Graph. Eine* Baumzerlegung *von G ist ein Paar $\mathcal{T} = (T, \beta)$, wobei T ein Baum ist und $\beta : V(T) \to 2^{V(G)}$, so dass*

- *für alle $v \in V(G)$ ist*

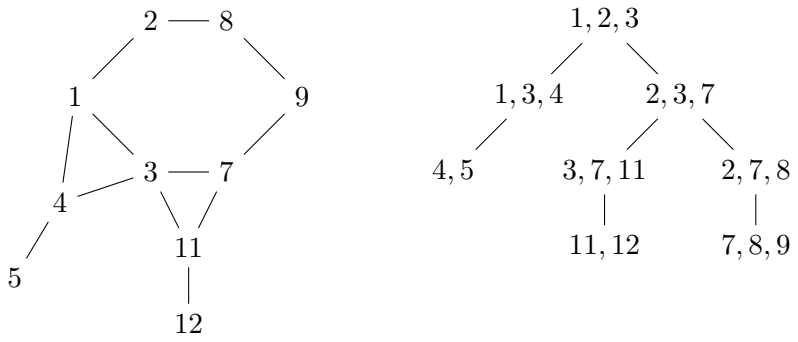
  $$\beta^{-1}(v) \coloneqq \{t \in V(T) : v \in \beta(t)\} \neq \emptyset$$

  und induziert einen Unterbaum von T.

- *für jede Kante $e \in E(G)$ existiert ein $t \in V(T)$ mit $e \subseteq \beta(t)$.*

Wir nennen die $\beta(t)$ Taschen (engl. bags).

8.3 Beispiel Ein Graph G und eine Baumzerlegung von G.



8.4 Definition *Sei G ein Graph und $\mathcal{T} = (T, \beta)$ eine Baumzerlegung von G. Die Weite von $\mathcal{T}$ ist*

$$w(\mathcal{T}) = \max\{|\beta(t)| : t \in V(T)\} - 1$$

und die Baumweite *von G ist*

$$\mathrm{bw}(G) = \min\{w(\mathcal{T}) : \mathcal{T} \text{ Baumzerlegung von } G\}.$$

Eine Klasse $\mathcal{C}$ von Graphen hat beschränkte Baumweite, wenn es ein $k \in \mathbb{N}$ gibt mit $\mathrm{bw}(G) \leqslant k$ *für alle $G \in \mathcal{C}$.*

Um die strukturelle Komplexität einer relationalen σ-Struktur $\mathcal{A} = (A, R_1^{\mathcal{A}}, \ldots)$ zu messen, betrachen wir den *Gaifman-Graphen* $G(\mathcal{A})$ von $\mathcal{A}$ mit Knotenmenge A und $E(G(\mathcal{A})) = \{\{a, b\} :$ es existiert $R \in \sigma$, R k-stellig, und $(a_1, \ldots, a_k) \in R^{\mathcal{A}}$ und $a_i = a, a_j = b$ für ein i und j, $1 \leqslant i, j \leqslant k\}$.

8.5 Beispiel

- Serien-parallele Graphen haben Baumweite höchstens 2.
- Kontrollflussgraphen (engl.: control flow graphs) von C- oder Java-Programmen haben Baumweite höchstens 8 (und höchstens 6, wenn sie kein goto benutzen).
- Backbone-Netze großer Internetprovider haben eine Baumweite von etwa 40.

Wir betrachten nun die angekündigten Separatoreigenschaften von Graphen mit beschränkter Baumweite.

8.6 Definition *Sei G ein Graph und seien $X, Y, S \subseteq V(G)$. S ist ein X-Y-*Trenner*, wenn es keinen Pfad in $G - S$ von X nach Y gibt. Eine* Separation *in G ist ein Paar (A, B), $A, B \subseteq V(G)$ mit*

- $A \cup B = V(G)$.
- $A \setminus B \neq \emptyset$ *und* $B \setminus A \neq \emptyset$.
- $A \cap B$ *ist ein A-B-Trenner.*

Die Ordnung von (A, B) *ist* $|A \cap B|$.

8.7 Lemma *Sei $\mathcal{T} = (T, \beta)$ eine Baumzerlegung von G. Sei $e = \{s, t\} \in E(T)$. Seien T_s, T_t die beiden Komponenten von $T - e$, so dass $s \in V(T_s)$ und $t \in V(T_t)$. Sei $\beta(T_s) = \bigcup\{v \in V(G) : \text{es existiert } t \in V(T_s) \text{ mit } v \in \beta(t)\}$ und analog für $\beta(T_t)$. Sei $S = \beta(s) \cap \beta(t)$. Wenn $\beta(T_s) \setminus \beta(T_t) \neq \emptyset$ und $\beta(T_t) \setminus \beta(T_s) \neq \emptyset$, dann ist $(\beta(T_s), \beta(T_t))$ eine Separation mit Separator S und S ist ein $\beta(T_s)$-$\beta(T_t)$-Trenner.*

8.3. Der Satz von Courcelle

Wir möchten einen Graphen mit beschränkter Baumweite in einem Baum kodieren, indem wir den Baum passend beschriften. Für jede feste Baumweite geht dies mit einer festen Anzahl von Farben. Dazu gehen wir zunächst zu einer geordneten Baumzerlegung über, d.h. in jeder Tasche hat jedes Element eine feste Position (die Tasche wird nicht mehr als Menge dargestellt). Aus technischen Gründen nehmen wir auch an, dass sich in jeder Tasche genau $k + 1$ Elemente befinden. Falls dies nicht der Fall ist, füllen wir die letzten Positionen der Tasche einfach mit Kopien des letzten Elements auf. Nun können wir mit einer festen Anzahl von Farben alle Informationen kodieren.

Zu jeder Tasche von Knoten t geben wir folgende Informationen an:

- Falls t das Kind von t' ist, geben wir die Indizes der Elemente an, die sich in beiden Taschen befinden;

- die Indizes von Knoten der Tasche von t, zwischen denen sich Kanten befinden;
- die Indizes von Knoten der Tasche von t, die gleich sind.

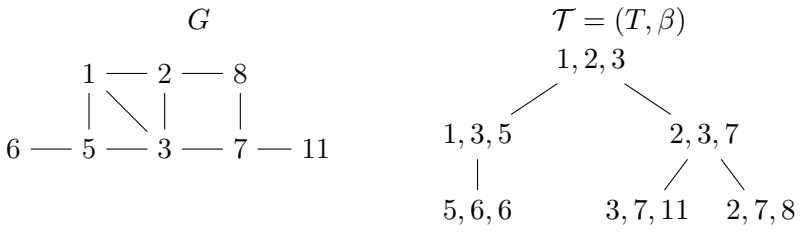
Formal definieren wir für $k \in \mathbb{N}$ die Signatur $\sigma_k := \{overlap(i,j), edge(i,j), eq(i,j) : 1 \leq i < j \leq k+1\}$ und $\Sigma_k := 2^{\sigma_k}$. Sei $\mathcal{T} = (T, \beta)$ eine Baumzerlegung von G mit Weite k. Die geordnete Baumzerlegung $\mathcal{T}' = (T, \beta')$ zu $\mathcal{T}$ ist definiert durch Angabe von $\beta' : V(T) \to (V(G))^{k+1}$ wie folgt:

$$\text{ist } \beta(t) = \{v_1, \ldots, v_j\},\ j \leq k+1, \text{ so definiere } \beta'(t) = (v_1, \ldots, v_j, \underbrace{v_j, \ldots, v_j}_{k+1-j \text{ mal}}).$$

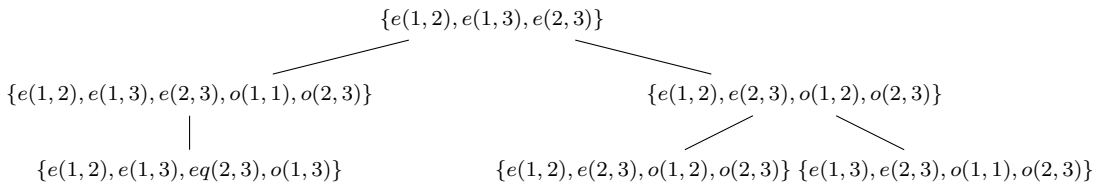
Wir kodieren nun $\mathcal{T}' = (T, \beta')$ als Σ_k-beschrifteten Baum T_Σ wie folgt:

- $V(T_\Sigma) = V(T)$ und $E(T_\Sigma) = E(T)$.
- Wir fixieren eine Wurzel $w \in V(T_\Sigma)$ und fassen T_Σ als (partiell) geordneten Baum auf (bzgl. Kindrelation).
- Wir definieren die Beschriftungsfunktion $\sigma : V(T_\Sigma) \to \Sigma_k$, so dass für $t \in V(T)$ mit $\beta'(t) = (v_1, \ldots, v_{k+1})$:
  - $edge(i,j) \in \sigma(t)$ gdw. $\{v_i, v_j\} \in E(G)$.
  - $eq(i,j) \in \sigma(t)$ gdw. $v_i = v_j$.
  - Wenn s mit $\beta'(s) = (u_1, \ldots, u_{k+1})$ der Vorgänger von t ist, dann ist $overlap(i,j) \in \sigma(t)$ gdw. $v_i = u_j$.

8.8 Beispiel



$$\mathcal{T}' = (T, \beta')$$



Man beachte, dass in $\mathcal{T}'$ die *Indizes* der Knoten der Bags aus $V(T)$ vermerkt sind. Z.B. bedeutet $edge(2,3)$ im rechten Nachfolger der Wurzel nicht, dass $\{2,3\}$ eine Kante von G wäre, sondern dass G eine Kante zwischen dem zweiten und dritten Element des Bags von $\mathcal{T}$ hat - also die Kante $\{3,7\}$.

8.9 Lemma *Sei G ein Graph und $\mathcal{T}' = (T, \beta')$ eine geordnete Baumzerlegung mit Weite k und sei T_Σ die Kodierung von $\mathcal{T}'$ als Σ_k-Baum. Zu jedem $\varphi \in \mathrm{MSO}_2[\sigma_{graph}]$ kann man effizient ein $\varphi' \in \mathrm{MSO}[\sigma_{\Sigma_k}]$ konstruieren, sodass*

$$G \models \varphi \Leftrightarrow T_\Sigma \models \varphi'.$$

8.10 Satz (Bodlaender) *Es existiert eine Funktion $f\colon \mathbb{N} \to \mathbb{N}$ und ein Algorithmus, der zu jedem Eingabegraphen G und jeder Zahl $k \in \mathbb{N}$ in Zeit $f(k) \cdot |V(G)|$ entweder eine Baumzerlegung von G mit Weite $\leqslant k$ konstruiert oder feststellt, dass $\mathrm{bw}(G) > k$.*

8.11 Satz (Courcelle) *Es gibt eine berechenbare Funktion $f\colon \mathbb{N}^2 \to \mathbb{N}$, so dass zu jedem Graphen G mit Baumweite höchstens k und jeder Formel $\varphi \in \mathrm{MSO}_2$ in Zeit $f(|\varphi|, k) \cdot |V(G)|$ entschieden werden kann, ob $G \models \varphi$. D.h. das parametrisierte MSO_2-Model-Checking-Problem ist fixed-parameter tractable auf allen Klassen mit beschränkter Baumweite.*

Courcelles Satz ist ein extrem wichtiges theoretisches Ergebnis. Kürzlich ist sogar ein Model-Checker effizient implementiert worden. Wir verweisen auf das Projekt *Sequoia* der RWTH Aachen[1].

Außerdem wurde gezeigt **(K., Tazari 2010)**, dass effizientes MSO_2-Model-Checking nicht für wesentlich größere Klassen möglich ist, falls diese unter Subgraphen abgeschlossen sind (unter einer Standardannahme aus der Komplexitätstheorie).

8.4. Cliquenweite

Die Cliquenweite verallgemeinert die Baumweite, Klassen mit beschränkter Cliquenweise sind im Allgemeinen aber nicht unter Subgraphen abgeschlossen, wie es für die Baumweite der Fall ist. Auf Klassen mit beschränkter Cliquenweite ist das MSO_1-Model-Checking-Problem effizient lösbar.

[1] http://sequoia.informatik.rwth-aachen.de/sequoia/

8.12 Definition *Sei $k \in \mathbb{N}$. Die Menge der k-*Ausdrücke *ist induktiv wie folgt definiert.*

- *i ist ein k-Ausdruck für alle $1 \leq i \leq k$.*
- *$t_1 \oplus t_2$ ist ein k-Ausdruck für alle k-Ausdrücke t_1, t_2.*
- *$edge_{i-j}(t)$ ist ein k-Ausdruck für alle k-Ausdrücke t und alle $1 \leq i < j \leq k$.*
- *$rename_{i \to j}(t)$ ist ein k-Ausdruck für alle k-Ausdrücke t und alle $1 \leq i, j \leq k$.*

Sei Σ ein Alphabet. Ein Σ-gefärbter Graph *ist ein Tupel $(V(G), E(G), c = c(G))$, wobei $(V(G), E(G))$ ein ungerichteter Graph ist und $c\colon V(G) \to \Sigma$ eine Färbungsfunktion. Ein k-Ausdruck φ beschreibt einen $\{1, \ldots, k\}$-gefärbten Graphen $G(\varphi)$ wie folgt.*

- *$G(i)$ ist der Graph $\big(\{v\}, \emptyset, \{(v, i)\}\big)$, d.h. v ist mit Farbe i gefärbt.*
- *$G(t_1 \oplus t_2) \coloneqq G(t_1) \oplus G(t_2)$ ist die disjunkte Vereinigung von $G(t_1)$ und $G(t_2)$.*
- *$G(edge_{i-j}(t))$ ist der gefärbte Graph $\Big(V(G(t)), E, c\big(G(t)\big)\Big)$ mit*

$$E = E(G(t)) \cup \big\{\{v, w\} \ : \ c(G(t))(v) = i, c(G(t))(w) = j\big\},$$

  d.h. man erhält $G(edge_{i-j}(t))$ aus $G(t)$, indem man zusätzliche Kanten zwischen allen Knoten mit Farbe i und j einfügt (wenn nötig).

- *$G(rename_{i \to j}(t))$ ist der Graph, den man aus $G(t)$ erhält, wenn man alle Knoten mit Farbe i mit Farbe j färbt:*

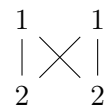
$$G(rename_{i \to j}(t)) \coloneqq (V(G(t)), E(G(t))), c'\,.$$

  mit $c'(v) = c(v)$, falls $c(v) \neq i$, und $c'(v) = j$, falls $c(v) = i$.

8.13 Beispiel Der 2-Ausdruck

$$edge_{1-2}((1 \oplus 1) \oplus (2 \oplus 2))$$

erzeugt den Graphen



8.14 Definition *Sei G ein Graph. Die* Cliquenweite $\mathrm{cw}(G)$ *von G ist die kleinste Zahl $k \in \mathbb{N}$ für die ein k-Ausdruck φ existiert mit $G = G(\varphi)$.*

8.15 Satz (Courcelle, Rotics, Makowski) $\mathbf{MC}(\mathrm{MSO}_1, \mathcal{C})$ *ist fixed-parameter tractable für alle Klassen* $\mathcal{C}$ *mit beschränkter Cliquenweite.*

Im Allgemeinen ist das Erfüllbarkeitsproblem der Prädikatenlogik, und insbesondere der monadischen Logik zweiter Stufe, unentscheidbar. In einem früheren Kapitel haben wir die Frage, ob eine Formel ein endliches Baummodell hat auf den Leerheitstest für Baumautomaten zurückgeführt, dieses eingeschränkte Erfüllbarkeitsproblem ist also für MSO entscheidbar. Dieses Ergebnis lässt sich wie folgt erweitern.

8.16 Satz (Seese 1992)

1. *Für alle* $k \in \mathbb{N}$ *gibt es einen Algorithmus, der entscheidet, ob eine gegebene* MSO_2*-Formel ein Modell mit Baumweite höchstens* k *hat.*

2. *Wenn eine Klasse* $\mathcal{C}$ *unbeschränkte Baumweite hat, dann gibt es keinen Algorithmus, der zu jeder Eingabeformel* $\varphi \in \mathrm{MSO}_2$ *entscheidet, ob es ein Modell von* φ *aus* $\mathcal{C}$ *gibt.*

8.17 Satz (Courcelle) *Für alle* $k \in \mathbb{N}$ *gibt es einen Algorithmus, der entscheidet, ob eine gegebene* MSO_1*-Formel ein Modell mit Cliquenweite höchstens* k *hat.*

8.18 Vermutung (Seese) *Wenn* $\mathcal{C}$ *unbeschränkte Cliquenweite hat, dann gibt es keinen Algorithmus, der zu jeder Eingabeformel* $\varphi \in \mathrm{MSO}_1$ *entscheidet, ob es ein Modell von* φ *aus* $\mathcal{C}$ *gibt.*

Kapitel 9.

Lokalität der Prädikatenlogik und das Auswertungsproblem von FO

Im letzten Kapitel haben wir die Komplexität des Auswertungsproblems für die monadische Logik zweiter Stufe untersucht. Das Problem hat sich als effizient lösbar auf baumartigen Strukturen herausgestellt.

- $\mathbf{MC}(\mathrm{MSO}_2, \mathcal{C}) \in \mathrm{FPT}$, wenn $\mathcal{C}$ beschränkte Baumweite hat und
- $\mathbf{MC}(\mathrm{MSO}_1, \mathcal{C}) \in \mathrm{FPT}$, wenn $\mathcal{C}$ beschränkte Cliquenweite hat.

Der Satz von Kreutzer und Tazari hat gezeigt, dass diese Klassen auf denen effizientes MSO-model-checking möglich ist, nicht wesentlich erweitert werden können. Um mit den globalen Eigenschaften von Graphen umzugehen, die MSO formulieren kann, benötigen wir gute Trennereigenschaften, die wir nur in den baumartigen Strukturen vorfinden.

In diesem Kapitel werden wir die Komplexität des Model-Checking-Problems der Prädikatenlogik untersuchen. Wir werden zunächst den Satz von Gaifman beweisen, der die Basis für unseren Model-Checking-Ansatz liefert. Er besagt, dass in der Prädikatenlogik nur lokale Eigenschaften formuliert werden können. Unser Ziel ist wie für die monadische Logik zweiter Stufe die größte Klasse $\mathcal{C}$ zu finden, auf der das FO-Model-Checking-Problem fixed-parameter tractable ist.

Das Model-Checking-Problem für FO wurde zuerst von Seese im Kontext der parametrisierten Komplexität untersucht.

9.1 Satz (Seese 1996) *Das Problem* $\mathbf{MC}(\mathrm{FO}, D_k)$*, wobei* D_k *die Klasse aller Graphen mit Maximalgrad* $\leqslant k$ *ist, ist fixed-parameter tractable.*

9.1. Der Satz von Gaifman

Wir fixieren eine endliche, relationale Signatur σ ohne Konstantensymbole. (In der englischsprachigen Literatur spricht man von einer „purely relational signature“, im Gegensatz zu einer „relational signature“, die auch Konstantensymbole enthalten darf).

9.2 Definition *Sei $\mathcal{A} = (A, R_1, \ldots, R_k)$ eine σ-Struktur. Der* Gaifmangraph $G_\mathcal{A}$ *ist der Graph mit Knotenmenge A und allen Kanten $\{a, b\}$, so dass $a, b \in A$, $a \neq b$ und a und b kommen gemeinsam in einem Tupel einer der Relationen R_i vor.*

Der Gaifmangraph erlaubt uns die üblichen Konzepte der Graphentheorie auf Strukturen zu übertragen. Zum Beispiel ist die Distanz zwischen zwei Elementen $a, b \in A$ definiert als die Distanz der Elemente im Gaifmangraph. Ähnlich kann man Trenner für Strukturen und Baumzerlegungen definieren.

9.3 Definition *Sei $r \in \mathbb{N}$, $\mathcal{A} = (A, R_1, \ldots, R_k)$ eine σ-Struktur und $a \in A$.*

1. *Die* r-Nachbarschaft *von a in $\mathcal{A}$ ist die Menge $N_r^\mathcal{A}(a) \coloneqq \{b \in A : $ Distanz von a zu b im Gaifman-Graphen $\mathcal{G}(\mathcal{A})$ ist $\leq r\}$.*
2. *Wir schreiben $\mathcal{A}_r(a)$ für die von $N_r^\mathcal{A}(a)$ induzierte Substruktur von $\mathcal{A}$, d.h. die Struktur mit Universum $N_r^\mathcal{A}(a)$ und für jedes s-stellige Relationssymbol $R \in \sigma$ ist $R^{\mathcal{A}_r(a)} = \{(a_1, \ldots, a_s) : a_i \in N_r^\mathcal{A}(a), (a_1, \ldots, a_s) \in R^\mathcal{A}\} = R^\mathcal{A} \cap N_r^\mathcal{A}(a)^s$.*

9.4 Definition *Sei $\varphi(x) \in FO[\sigma]$. φ ist* r-lokal um x, *wenn für alle σ-Strukturen $\mathcal{A}$ mit Universum A und alle $a \in A$ gilt $\mathcal{A} \models \varphi(a) \iff \mathcal{A}_r(a) \models \varphi(a)$.*

9.5 Beispiel Sei $\sigma = \{R, E\}$, wobei R ein einstelliges und E ein zweistelliges Relationssymbol ist.

- $\varphi(x) = R(x)$ ist 0-lokal
- $\varphi(x) = \exists y \ (E(x, y) \wedge R(y))$ ist 1-lokal
- Für alle $r \in \mathbb{N}$ ist $\varphi(x) = \exists y \ R(y)$ nicht r-lokal.

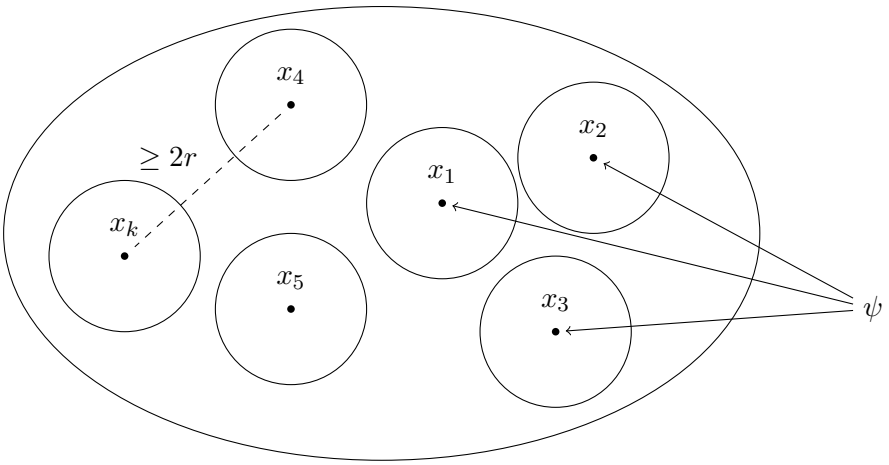
Wir können Distanzen in σ-Strukturen mit Formeln ausdrücken, d.h. es gibt für jedes $r \in \mathbb{N}$ eine Formel $dist_{\leqslant r}(x, y)$ mit $\mathcal{A} \models dist_{\leqslant r}(a, b) \iff$ die Distanz zwischen $a, b \in A$ im Gaifman-Graphen von $\mathcal{A}$ ist höchstens r. Wir schreiben der Lesbarkeit halber $dist(x, y) \leqslant r$ für die Formel $dist_{\leqslant r}(x, y)$ und $dist(x, y) > r$ für $\neg dist_{\leqslant r}(x, y)$.

9.6 Definition *Ein* basis-lokaler Satz *ist ein Satz, der die Form*

$$\varphi = \exists x_1 \ldots \exists x_k \left(\bigwedge_{1 \leq i < j \leq k} dist(x_i, x_j) > 2r \wedge \bigwedge_{i=1}^{k} \psi(x_i) \right)$$

hat, wobei $\psi(x)$ eine r-lokale Formel ist für ein $r \in \mathbb{N}$.

Ein basis-lokaler Satz beschreibt die Existenz von k Elementen, deren r-Nachbarschaften disjunkt sind, so dass jedes Element die Formel ψ erfüllt. Der Wahrheitswert von ψ hängt nur von der r-Nachbarschaft des Elements ab.



9.7 Satz *[Gaifman '82] Jeder Satz $\varphi \in FO$ ist äquivalent zu einer Boole'schen Kombination von basis-lokalen Sätzen.*

Wir sagen, dass φ äquivalent ist zu einem Satz in *Gaifman-Normalform*. Der Satz besagt weiterhin, dass die Transformation von φ in einen äquivalenten Satz in Gaifman-Normalform berechenbar ist. Wir beweisen in der Vorlesung nur die nicht-konstruktive Version des Satzes, da der Beweis leichter zu verstehen ist.

9.8 Lemma *Für jedes $m \in \mathbb{N}$ gibt es ein $M \in \mathbb{N}$, so dass für alle σ-Strukturen $\mathcal{A}, \mathcal{B}$ gilt: Falls $\mathcal{A}$ und $\mathcal{B}$ dieselben basis-lokalen* FO[σ]*-Sätze vom Quantorenrang höchstens M erfüllen, dann gilt $\mathcal{A} \equiv_m \mathcal{B}$.*

Wir verschieben den Beweis des Lemmas und überlegen uns, was das Lemma für den Beweis von Satz 9.7 bringt. Offensichtlich gibt es für jedes $M \in \mathbb{N}$ bis auf logische Äquivalenz nur endlich viele verschiedene basis-lokale Sätze vom Quantorenrang höchstens M. Sei $\chi_1^M, \ldots, \chi_{h(M)}^M$ eine vollständige Liste dieser Sätze.

9.9 Lemma *Sei φ ein Satz, $m := \mathrm{qr}(\varphi)$ und sei M wie in Lemma 9.8. Dann ist φ äquivalent zur folgenden Booleschen Kombination basis-lokaler Sätze:*

$$\tilde{\varphi} := \bigvee \left\{ (\bigwedge_{i \in I} \chi_i^M \wedge \bigwedge_{i \notin I} \neg \chi_i^M) : \begin{array}{l} \text{es existiert } I \subseteq \{1, \ldots, h(M)\} \\ \text{und eine } \sigma\text{-Struktur } \mathcal{A}, \text{ so dass } \mathcal{A} \models \varphi \\ \text{und für alle } 1 \leqslant i \leqslant M\text{: } \mathcal{A} \models \chi_i^M \iff i \in I \end{array} \right\}$$

Beweis. Sei $\mathcal{B}$ eine σ-Struktur. Wir zeigen, dass $\mathcal{B} \models \varphi$ genau dann, wenn $\mathcal{B} \models \tilde{\varphi}$. Die Hinrichtung ist dabei klar. Zum Beweis der Rückrichtung gelte also $\mathcal{B} \models \tilde{\varphi}$. Also existiert eine Indexmenge $I \subseteq \{1, \ldots, h(M)\}$ so dass

1. $\mathcal{B} \models \bigwedge_{i \in I} \chi_i^M \wedge \bigwedge_{i \notin I} \neg \chi_i^M$, d.h. $\mathcal{B}$ erfüllt genau die Sätze χ_i^M für die $i \in I$, und

2. es gibt eine σ-Struktur $\mathcal{A}$ mit $\mathcal{A} \models \varphi$ und $\mathcal{A} \models \chi_i^M$ genau dann, wenn $i \in I$.

Dies bedeutet, dass $\mathcal{A}$ und $\mathcal{B}$ dieselben basis-lokalen Sätze vom Quantorenrang höchstens M erfüllen. Nach Lemma 9.8 gilt also $\mathcal{A} \equiv_m \mathcal{B}$ und somit $\mathcal{B} \models \varphi$, da $\mathrm{qr}(\varphi) = m$ und $\mathcal{A} \models \varphi$. □

Aus Lemma 9.9 folgt sofort der Satz von Gaifman. Wir müssen nun also nur noch Lemma 9.8 beweisen.

Beweis. (Beweis von Lemma 9.8.) Zunächst erinnern wir an die Definition der n-Hintikka-Formel $\varphi^n_{\mathcal{A},\overline{a}}$. Diese Formel war so definiert, dass für alle $\mathcal{A}', \overline{a}'$ gilt: $\mathcal{A}' \models \varphi^n_{\mathcal{A},\overline{a}}(\overline{a}')$ genau dann, wenn $(\mathcal{A}', \overline{a}') \equiv_n (\mathcal{A}, \overline{a})$, oder äquivalent, wenn Duplikatorin das n-Runden EF-Spiel auf $(\mathcal{A}', \overline{a}')$ und $(\mathcal{A}, \overline{a})$ gewinnt. Der Quantorenrang von $\varphi^n_{\mathcal{A},\overline{a}}$ ist gerade n.

Wir schreiben $\psi^{7^j,n}_{\overline{a}}(\overline{x})$ für die Formel $\varphi^n_{\mathcal{A},\overline{a}}$, deren Quantoren relativiert sind auf die 7^j-Umgebung von $\overline{a}$. Die Formel besagt in einer Struktur $\mathcal{A}'$, dass die 7^j-Umgebungen von $\overline{a}'$ (eingesetzt für $\overline{x}$) in $\mathcal{A}'$ n-äquivalent zur 7^j-Umgebung von $\overline{a}$ in $\mathcal{A}$ ist.

Wir definieren induktiv eine Funktion $g : \{0, \dots, m\} \to \mathbb{N}$. Für den Basisfall wählen wir $g(0) := 1$. Für den Induktionsschluß von $g(j)$ auf $g(j+1)$ werden im Verlaufe des Beweises verschiedene Bedingungen angegeben. Die genauen Werte auszurechnen ist jedoch mühsam und soll an dieser Stelle daher auch nicht erfolgen. Als erste Bedingung an g fordern wir die folgende.

Bedingung 0. Für alle $j < m$ gilt $g(j) < g(j+1)$.

Zu jedem $m \in \mathbb{N}$ definieren wir $M := g(m)$. Seien $\mathcal{A}$ und $\mathcal{B}$ zwei σ-Strukturen, die dieselben basis-lokalen Sätze vom Quantorenrang höchstens M erfüllen. Wir müssen zeigen, dass $\mathcal{A}$ und $\mathcal{B}$ m-äquivalent sind. Dazu zeigen wir, dass Duplikatorin unter dieser Voraussetzung das m-Runden Spiel auf $\mathcal{A}$ und $\mathcal{B}$ gewinnt.

Wir schreiben $\mathrm{Part}(\mathcal{A}, \mathcal{B})$ für die Menge der partiellen Isomorphismen zwischen $\mathcal{A}$ und $\mathcal{B}$ und definieren für $j \in \{0, \dots, m\}$

$$I_j := \{\overline{a} \mapsto \overline{b} \in \mathrm{Part}(\mathcal{A}, \mathcal{B}) : \mathcal{A}_{7^j}(\overline{a}) \equiv_{g(j)} \mathcal{B}_{7^j}(\overline{b}) \text{ und } |\overline{a}| = |\overline{b}| \leqslant m - j\}.$$

Wir vereinbaren für $|\overline{a}| = 0$, dass $\mathcal{A}_{7^j}(\overline{a}) = \emptyset$ und dass $\emptyset \equiv_{g(j)} \emptyset$.

Wir zeigen, dass im m-Runden-Spiel auf $\mathcal{A}$ und $\mathcal{B}$ Duplikatorin so spielen kann, dass die nach j Runden gewählten Elemente $\overline{a}$ und $\overline{b}$ einen partiellen Isomorphismus $\overline{a} \mapsto \overline{b}$ in I_{m-j} definieren. Dann erhalten wir insbesondere nach m Runden einen partiellen Isomorphismus.

Nach obiger Vereinbarung gilt $\emptyset \mapsto \emptyset \in I_m$, also ist für 0 Runden nichts zu zeigen.

Sei nun $0 \leqslant j < m$ und seien $\overline{a}$ und $\overline{b}$ die gewählten Elemente, d.h. $\overline{a} \mapsto \overline{b} \in I_{j+1}$. Aus Symmetriegründen dürfen wir annehmen, dass Herausforderer ein Element $a \in A$ wählt. Wir suchen $b \in B$, sodass $\overline{a}a \mapsto \overline{b}b \in I_j$. Nach Definition von I_{j+1} gilt

$$\mathcal{A}_{7^{j+1}}(\overline{a}) \equiv_{g(j+1)} \mathcal{B}_{7^{j+1}}(\overline{b}). \tag{9.1}$$

Wir unterscheiden zwei Fälle, je nachdem wo das neue Element a in Bezug auf die schon gewählten Elemente liegt.

Fall 1. Sei $a \in N^{\mathcal{A}}_{2\cdot 7^j}(\overline{a})$. Dann gilt

$$\mathcal{A}_{7^{j+1}}(\overline{a}) \models \exists z \ \mathrm{dist}_{\leqslant 2\cdot 7^j}(\overline{a}, z) \wedge \psi^{7^j, g(j)}_{\overline{a},a}(\overline{a}, z), \tag{9.2}$$

denn offensichtlich erfüllt a als Wahl für z die Formel. Dies führt zur zweiten Bedingung an die Funktion g.

Bedingung 1. $g(j+1)$ is größer als der Quantorenrang der Formel in (9.2).

Aus (9.1) folgt somit

$$\mathcal{B}_{7^{j+1}} \models \exists z \ \mathrm{dist}_{\leqslant 2\cdot 7^j}(\overline{b}, z) \wedge \psi^{7^j, g(j)}_{\overline{a}a}(\overline{b}, z),$$

Folglich existiert ein $b \in N^{\mathcal{B}}_{2\cdot 7^j}(\overline{b})$, so dass $\mathcal{B}_{7^{j+1}} \models \psi^{7^j, g(j)}_{\overline{a},a}(\overline{b}, b)$. Somit gilt $\mathcal{A}_{7^j}(\overline{a}, a) \equiv_{g(j)} \mathcal{B}_{7^j}(\overline{b}, b)$ und nach Definition von I_j also $\overline{a}a \mapsto \overline{b}b \in I_j$.

Fall 2. Sei nun $a \notin N^{\mathcal{A}}_{2\cdot 7^j}(\overline{a})$, d.h. $N_{7^j}(\overline{a}) \cap N_{7^j}(a) = \emptyset$. Für $1 \leqslant s \in \mathbb{N}$ sei

$$\delta^j_s(x_1, \ldots, x_s) := \bigwedge_{1 \leqslant l < k \leqslant s} \neg \mathrm{dist}_{\leqslant 4\cdot 7^j}(x_l, x_k) \wedge \bigwedge_{l=1}^{s} \psi^{7^j, g(j)}_a(x_l).$$

Die Formel besagt, dass es s verschiedene Elemente $x_1, \ldots, x_s$ gibt, deren Abstand paarweise jeweils größer als $4 \cdot 7^j$ ist, deren 7^j-Umgebungen jedoch alle $g(j)$-äquivalent zur 7^j-Umgebung um a sind.

Sei nun $e_{\mathcal{A}}$ so gewählt, dass

$$\mathcal{A}_{7^{j+1}}(\overline{a}) \models \exists x_1 \ldots \exists x_{e_{\mathcal{A}}} \ \delta^j_{e_{\mathcal{A}}}(x_1, \ldots, x_{e_{\mathcal{A}}}) \wedge \bigwedge_{l=1}^{e_{\mathcal{A}}} \mathrm{dist}_{\leqslant 2\cdot 7^j}(\overline{a}, x_l) \tag{9.3}$$

jedoch

$$\mathcal{A}_{7^{j+1}}(\overline{a}) \not\models \exists x_1 \ldots \exists x_{e_{\mathcal{A}}+1} \ \delta^j_{e_{\mathcal{A}}}(x_1, \ldots, x_{e_{\mathcal{A}}+1}) \wedge \bigwedge_{l=1}^{e_{\mathcal{A}}+1} \mathrm{dist}_{\leqslant 2\cdot 7^j}(\overline{a}, x_l). \tag{9.4}$$

$e_{\mathcal{A}}$ ist also die maximale Anzahl solcher Elemente, die nicht weiter als $2 \cdot 7^j$ von $\overline{a}$ entfernt sind. Da es in der $2 \cdot 7^j$-Umgebung um ein $a_i \in \overline{a}$ keine zwei Elemente mit Abstand größer als $4 \cdot 7^j$ geben kann, folgt $e_{\mathcal{A}} \leqslant |\overline{a}| \leqslant m - (j+1) \leqslant m$.

Analog zur Definition von $e_{\mathcal{A}}$ definieren wir $e_{\mathcal{B}}$ als die entsprechende Zahl in $\mathcal{B}$. Nach (9.1) gilt $\mathcal{A}_{7^{j+1}}(\overline{a}) \equiv_{g(j+1)} \mathcal{B}_{7^{j+1}}(\overline{b})$. Weiterhin stellen wir folgende Bedingung an die Funktion g.

Bedingung 2. $g(j+1)$ ist größer als der Quantorenrang der Formeln in (9.3) und (9.4).

Es folgt, dass $\mathcal{B}_{7^{j+1}}(\bar{b})$ die Formel in (9.3) erfüllt, die Formel in (9.4) jedoch nicht. Also gilt $e_{\mathcal{A}} = e_{\mathcal{B}}$.

Wir setzen $e := e_{\mathcal{A}} = e_{\mathcal{B}}$. Wegen (9.3) gilt auch

$$\mathcal{A} \models \exists x_1 \ldots \exists x_e \; \delta_e^j(x_1, \ldots, x_e). \tag{9.5}$$

Gilt ferner auch noch

$$\mathcal{A} \models \exists x_1 \ldots \exists x_e \exists x_{e+1} \; \delta_e^j(x_1, \ldots, x_{e+1}), \tag{9.6}$$

so setzen wir $e'_{\mathcal{A}} := e+1$. Andernfalls setzen wir $e'_{\mathcal{A}} := e$. Wiederum wird $e'_{\mathcal{B}}$ analog in $\mathcal{B}$ definiert und es gilt $e'_{\mathcal{A}} = e'_{\mathcal{B}}$. Dies folgt, da die Formeln in (9.5) und (9.6) basis-lokal sind und der Quantorenrang dieser Formeln höchstens dem Quantorenrang der Formeln in (9.3) und (9.4) entspricht. Wegen Bedingung 2 an g ist $g(j+1)$ also größer als dieser Quantorenrang. Nach Voraussetzung des Lemmas erfüllen $\mathcal{A}$ und $\mathcal{B}$ dieselben basis-lokalen Formeln vom Quantorenrang höchstens $M \geqslant g(j+1)$. Also erfüllt $\mathcal{B}$ ebenfalls die Formel in (9.5), nicht jedoch die in (9.6). Es folgt $e'_{\mathcal{A}} = e'_{\mathcal{B}}$. Wir schreiben $e' := e'_{\mathcal{A}} = e'_{\mathcal{B}}$.

Wir unterscheiden nun zwei Fälle, je nachdem ob $e = e'$ oder $e+1 = e'$.

Fall 2.1 Sei $e = e'$, d.h. alle Elemente $a' \in A$, die $\psi_a^{7^j, g(j)}(x)$ erfüllen, haben Abstand von $\overline{a}$ höchstens $6 \cdot 7^j < 7^{j+1}$. Denn gäbe es ein solches Element a' mit Abstand größer als $6 \cdot 7^j$ von $\overline{a}$, so wären die e Elemente in $N_{2 \cdot 7^j}(\overline{a})$ zusammen mit a' insgesamt $e+1$ Elemente, die die Formel δ_{e+1}^j erfüllen. Dies wäre aber ein Widerspruch zu $e = e'$.

Offensichtlich erfüllt a die Formel $\psi_a^{7^j, g(j)}(x)$. Es gilt also (mit der Voraussetzung des Falles 2) $a \in (N_{\mathcal{A}}(6 \cdot 7^j, \overline{a}) \backslash (N_{\mathcal{A}}(2 \cdot 7^j, \overline{a}))$ und somit

$$\mathcal{A}_{7^{j+1}}(\overline{a}) \models \exists z \; \neg \mathrm{dist}_{\leqslant 2 \cdot 7^j}(\overline{a}, z) \wedge \mathrm{dist}_{\leqslant 6 \cdot 7^j}(\overline{a}, z) \wedge \psi_a^{7^j, g(j)}(z) \wedge \psi_{\overline{a}}^{7^j, g(j)}(\overline{a}). \tag{9.7}$$

Dies liefert die dritte und letzte Bedingung an die Funktion g.

Bedingung 3. $g(j+1)$ ist größer als der Quantorenrang der Formel in (9.7).

Da nach (9.1) $(\mathcal{A}_{7^{j+1}}(\overline{a}), \overline{a}) \equiv_{g(j+1)} (\mathcal{B}_{7^{j+1}}(\bar{b}), \bar{b})$, folgt

$$\mathcal{B}_{7^{j+1}}(\bar{b}) \models \exists z \; \neg \mathrm{dist}_{\leqslant 2 \cdot 7^j}(\bar{b}, z) \wedge \mathrm{dist}_{\leqslant 6 \cdot 7^j}(\bar{b}, z) \wedge \psi_a^{7^j, g(j)}(z) \wedge \psi_{\overline{a}}^{7^j, g(j)}(\bar{b}).$$

Es existiert also ein $b \in B$ mit $2 \cdot 7^j < \mathrm{dist}(\bar{b}, b) \leqslant 6 \cdot 7^j$, so dass $(\mathcal{A}_{7^j}(\overline{a}), \overline{a}) \equiv_{g(j)} (\mathcal{B}_{7^j}(\bar{b}), \bar{b})$ und $(\tilde{N}_{\mathcal{A}}(7^j, a), a) \equiv_{g(j)} (\tilde{N}_{\mathcal{B}}(7^j, b), b)$. Ferner gilt $N_{7^j}^{\mathcal{A}}(\overline{a}) \cap N_{7^j}^{\mathcal{A}}(a) = \emptyset$ sowie $N_{7^j}^{\mathcal{B}}(\bar{b}) \cap N_{7^j}^{\mathcal{B}}(b) = \emptyset$ und somit $(\mathcal{A}_{7^j}(\overline{a}a), \overline{a}a) \equiv_{g(j)} ((\mathcal{B}_{7^j}(\bar{b}\bar{b}), \bar{b}\bar{b})$. Nach Definition von I_j gilt daher $\overline{a}a \mapsto \bar{b}\bar{b} \in I_j$.

Fall 2.2 Sei $e+1=e'$. Dann gilt $\mathcal{B} \models \exists x_1 \ldots \exists x_{e+1}\ \delta^j_{e+1}(x_1,\ldots,x_{e+1})$. Es gibt also ein $b \in$ mit $\mathcal{B} \models \psi_a^{7^j,g(j)}(b)$ und $\mathbb{N}^{\mathcal{B}}_{7^j}(\bar{b}) \cap N^{\mathcal{B}}_{7^j}(b) = \emptyset$. Insbesondere gilt also $\mathcal{A}_{7^j}(a) \equiv_{g(j)} \mathcal{B}_{7^j}(b)$. Analog zum Fall 2.1 gilt $N_{\mathcal{A}}(7^j, \bar{a}a), \bar{a}a) \equiv_{g(j)} (\tilde{N}_{\mathcal{B}}(7^j, \bar{b}b), \bar{b}b)$ und somit $\bar{a}a \mapsto \bar{b}b \in I_j$.

Per Induktion folgt also Lemma 9.8 und der Satz von Gaifman ist bewiesen. □

Wie oben erwähnt gibt es auch einen Algorithmus, der jede Formel φ in eine äquivalente Formel φ' in Gaifman-Normalform umwandelt. Allerdings kann diese Umformung niemals effizient sein, es wurde nämlich von Dawar, Grohe, Kreutzer und Schweickardt (2006) bewiesen, dass es keine elementare Schranke für die Länge von φ' in Abhängigkeit von der Länge von φ gibt. (D.h. es gibt keine Funktion $f_k(x) = 2^{2^{\cdots^{2^x}\}_k}}$ mit einem festen k, so dass $|\varphi'| \leq f_k(|\varphi|)$ gilt.) Dies stellt eine Hürde für die praktische Umsetzung von Model-Checking-Algorithmen dar, die auf Gaifmans Satz basieren. Eine genaue Analyse von speziellen Formeln kann aber durchaus zu kurzen lokalen Formeln führen.

9.10 Satz (Dawar, Grohe, Kreutzer, Schweikardt, 2006) *Für jedes $h \in \mathbb{N}$ gibt es einen* FO[E]*-Satz φ_h der Größe $\mathcal{O}(h^4)$, so dass jeder zu φ_h äquivalente* FO[E]*-Satz in GNF (Gaifman-Normalform) mindestens die Größe* tow(h) *hat, wobei*

$$\mathrm{tow}(0) := 1 \quad \textit{und} \quad \mathrm{tow}(n+1) := 2^{\mathrm{tow}(n)} \quad \textit{für alle } n \in \mathbb{N}.$$

Wir können nun den Satz von Seese beweisen.

9.11 Satz (Seese 1996) *Das Problem* $\mathbf{MC}(\mathrm{FO}, D_k)$*, wobei D_k die Klasse aller Graphen mit Maximalgrad $\leqslant k$ ist, ist fixed-parameter tractable.*

Insbesondere kann man für jeden festen FO-Satz φ zu gegebenem Graphen G vom Grad $\leqslant k$ in Linearzeit entscheiden, ob $G \models \varphi$.

Beweis. Sei $d \in \mathbb{N}$ fest, sei φ der gegebene FO[E]-Satz, und sei $G = (V, E)$ der gegebene Graph vom Grad $\leqslant d$. Wir schreiben n, um die Größe (der Kodierung) von G zu bezeichnen.

Zunächst übersetzen wir φ in einen äquivalenten Satz φ' in *Gaifman-Normalform*. Dann testen wir nacheinander für jeden einzelnen basis-lokalen Satz χ, der in φ' vorkommt, ob $G \models \chi$. Am Ende können wir dann die Resultate für die einzelnen basis-lokalen Sätze χ einfach kombinieren, um zu ermitteln, ob $G \models \varphi'$.

Sei im Folgenden χ ein basis-lokaler Satz der Form

$$\exists x_1 \cdots \exists x_\ell \Big(\Big(\bigwedge_{1 \leqslant i < j \leqslant \ell} \mathrm{dist}(x_i, x_j) > 2{\cdot}r \Big) \wedge \Big(\bigwedge_{i=1}^{\ell} \psi(x_i) \Big) \Big),$$

wobei $\ell, r \in \mathbb{N}$ und $\psi(x)$ r-lokal um x ist. Natürlich gilt $G \models \chi$ genau dann, wenn es mindestens ℓ Knoten vom paarweisen Abstand $> 2r$ in G gibt, so dass die r-Nachbarschaften dieser ℓ Knoten allesamt die r-lokale Formel ψ erfüllen. Um zu entscheiden, ob $G \models \chi$, gehen wir in 2 Schritten vor:

1. Bestimme für jeden Knoten $v \in V^G$, ob $G[N_r^G(v)] \models \psi(v)$, und markiere diejenigen Knoten v, für die $G[N_r^G(v)] \models \psi(v)$ gilt, als *rot*.
2. Entscheide, ob es in G ℓ *rote* Knoten vom paarweisen Abstand $> 2r$ gibt.

Zum Lösen von Schritt 1 beachte, dass für jeden Knoten v in einem Graphen G vom Grad $\leqslant d$ gilt:

$$|N_r^G(v)| \leqslant 1 + d + d^2 + \cdots + d^r \leqslant d^{r+1},$$

d.h. die Größe von $N_r^G(v)$ hängt nicht von der Größe von G ab, sondern nur von den Zahlen d und r. Schritt 1 kann daher mit einem brute-force Algorithmus gelöst werden.

Zum Lösen von Schritt 2 müssen wir nun noch ℓ rote Knoten vom paarweisen Abstand $> 2r$ finden. Dazu verwenden wir zunächst eine Greedy-Strategie, um aus den roten Knoten in G eine Menge X von roten Knoten mit paarweisem Abstand $> 2r$ auszuwählen. Dazu wird in einer Schleife jeweils ein beliebiger Knoten v aus R ausgewählt und danach werden alle Knoten mit Abstand $\leqslant 2r$ von v aus R gelöscht.

Wurde auf diese Weise eine Menge X mit (mindestens) ℓ roten Knoten vom paarweisen Abstand $> 2r$ gefunden, so können wir anhalten und G akzeptieren.

Enthält die Menge X jedoch weniger als ℓ Knoten, so können wir daraus noch nicht schließen, dass wir G verwerfen können, denn wir könnten ja in der Schleife die Knoten v für X einfach ungeschickt ausgewählt haben. Wir wissen jedoch auf jeden Fall, dass *jeder* rote Knoten von G in der $2r$-Nachbarschaft eines Elements aus X liegt. Daraus folgt auch, dass die $2r$-Nachbarschaft jedes roten Knotens in der $4r$-Nachbarschaft eines Elements aus X liegt. Da $|X| \leqslant \ell{-}1$ ist und da G ein Graph vom Grad $\leqslant d$ ist, wissen wir, dass

$$|N_{4r}^G(X)| \leqslant (\ell{-}1) \cdot d^{4r+1}.$$

Daher können wir nun einen Brute-Force-Algorithmus anwenden, um in der $4r$-Nachbarschaft von X das Independent-Set-Problem zu lösen. Die Laufzeit hängt nicht von der Größe von G sondern nur von einer Zahl $f_d(\ell, r)$ ab (für eine geeignete Funktion $f_d \colon \mathbb{N} \times \mathbb{N} \to \mathbb{N}$).

Insgesamt erhalten wir einen Algorithmus, der bei Eingabe eines basis-lokalen Satzes χ der Größe k und eines Graphen G vom Grad $\leqslant d$ in Zeit $f_d(k) \cdot n$ testet, ob $G \models \chi$ (wobei $f_d : \mathbb{N} \to \mathbb{N}$ eine geeignete Funktion ist). □

Derselbe Trick funktioniert auch für planare Graphen. Hier braucht man nur für den Test auf $G[N_r^G(v)] \models \psi(v)$ (um im 1. Schritt die Färbung der Knoten zu erhalten) Courcelles Theorem (Satz 8.11) anzuwenden. Dass Courcelles Theorem dabei anwendbar ist, folgt aus dem folgenden Satz:

9.12 Satz (Robertson, Seymour 1984) *Ein planarer Graph G mit Radius r hat Baumweite $\leq 3r + 2$.*

Nun kann man analog beweisen, dass das FO-Model-Checking-Problem fixed-parameter tractable ist auf der Klasse der planaren Graphen.

Die größte Klasse von Graphen, die bezüglich Untergraphen abgeschlossen ist und auf der effizientes FO-Model-Checking möglich ist, wurde kürzlich gefunden.

9.13 Satz (Grohe, Kreutzer, Siebertz '14) *Sei $\mathcal{C}$ eine Graphklasse, die unter Untergraphen abgeschlossen ist. Dann ist* $\mathrm{MC}(\mathrm{FO}, \mathcal{C}) \in \mathrm{FPT}$ *gdw. $\mathcal{C}$ nowhere dense ist.*

9.2. Anwendung von Gaifmans Satz für Approximationsalgorithmen

Im letzten Kapitel haben wir eine Anwendung des Satzes von Gaifman für das Auswertungsproblem der Prädikatenlogik auf Graphen (oder allgemeinen relationalen Strukturen) mit bestimmten Struktureigenschaften gesehen. Ähnliche Anwendungen gibt es auch für *Optimierungsprobleme*.

Bei einem Optimierungsproblem wollen wir nicht nur entscheiden, ob z.B. ein Graph eine dominierende Menge der Größe k enthält, sondern eine dominierende Menge minimaler Größe berechnen. Formal ist ein NP-Optimierungsproblem wie folgt definiert.

9.14 Definition *Ein* Optimierungsproblem *ist ein Tupel $(\mathcal{I}, sol, cost, opt)$, wobei*

- *$\mathcal{I}$ die Menge der möglichen Instanzen ist (als Wörter über einem Alphabet kodiert)*
- *sol ist eine Funktion, die jeder Instanz eine Menge zulässiger Lösungen zuordnet*
- *cost ist eine Funktion, die jedem Paar (w, s) aus einer Instanz w und zulässiger Lösung s eine natürliche Zahl c, die Kosten der Lösung, zuordnet und*
- *opt ist entweder min oder max, das Optimierungsziel.*

Ein Optimierungsproblem $(\mathcal{I}, sol, cost, opt)$ ist ein NP-Optimierungsproblem*, wenn*

- *die Menge $\mathcal{I}$ in Polynomialzeit entscheidbar ist,*
- *es ein festes Polynom $p(n)$ gibt, so dass für alle $x \in \mathcal{I}$ und alle $y \in sol(x)$ gilt: $|y| \leqslant p(|x|)$,*
- *für alle $x \in \mathcal{I}$ ist die Menge $sol(x)$ in Polynomialzeit entscheidbar und*
- *die cost-Funktion ist in Polynomialzeit berechenbar.*

Die Klasse aller NP-Optimierungsprobleme wird mit NPO bezeichnet.

9.15 Beispiel Sei G ein Graph. Eine *dominierende Menge* in G ist eine Menge $S \subseteq V(G)$, so dass für alle $v \in V(G)$ gilt: $v \in S$ oder es existiert ein $u \in S$ mit $\{v, u\} \in E(G)$.

Dann ist MIN-DOMINATING-SET $\coloneqq (\mathcal{I}, sol, cost, opt)$ ein NP-Optimierungsproblem, wobei

- $\mathcal{I}$ die Klasse aller endlicher Graphen und
- für jeden solchen Graphen G $sol(G)$ die Menge aller dominierender Mengen S von G ist,
- $cost(G, S) \coloneqq |S|$ und
- $opt = \min$.

Wir können bestimmte Optimierungsprobleme auch durch logische Formeln beschreiben.

9.16 Definition *Sei σ eine Signatur und sei $X \notin \sigma$ ein r-stelliges Relationssymbol. Eine Formel $\varphi(X) \in \mathrm{FO}[\sigma \dot\cup \{X\}]$ heißt* positiv in X*, oder* X-positiv*, wenn jedes Vorkommen von X in φ nur unter einer geraden Zahl von Negationssymbolen steht. Anders gesagt, wandelt man φ in Negationsnormalform um, kommt X nur positiv vor.*

Analog heißt $\varphi(X)$ negativ in X*, oder* X-negativ*, wenn jedes Vorkommen von X unter einer ungeraden Zahl von Negationssymbolen steht.*

Wir nehmen hierbei an, dass φ nur die Booleschen Operatoren $\wedge, \vee$ und $\neg$ benutzt, da $\rightarrow$ und $\leftrightarrow$ implizit Negationen enthalten.

X-positive, bzw. X-negative, Formeln definieren auf naheliegende Weise Optimierungsprobleme, wie die folgende Definition zeigt.

9.17 Definition *Sei σ eine Signatur und $X \notin \sigma$ ein r-stelliges Relationssymbol. Eine X-positive Formel $\varphi(X) \in \mathrm{FO}[\sigma \dot\cup \{X\}]$ definiert ein Minimierungsproblem $Min_{\varphi(X)}$, in dem für jede σ-Struktur $\mathcal{A}$ eine Menge $S \subseteq (V^{\mathcal{A}})^r$ minimaler Kardinalität gesucht wird, so dass $(\mathcal{A}, S) \models \varphi$.*

Analog definieren X-negative Formeln Maximierungsprobleme $Max_{\varphi(X)}$.

9.18 Beispiel Das MIN-DOMNATING-SET Problem wird durch die Formel

$$\varphi(X) \coloneqq \forall x \left(Xx \vee \exists y \ (y \in X \wedge E(x, y)) \right)$$

definiert.

9.19 Definition *Sei σ eine Signatur, $X \notin \sigma$ r-stellig und $\mathcal{A}$ eine σ-Struktur. Sei $\varphi(X)$ eine X-positive Formel und sei $X_{min} \subseteq (V^{\mathcal{A}})^r$ eine Relation minimaler Größe, so dass $(\mathcal{A}, X_{min}) \models \varphi(X)$.*

Sei $\varepsilon > 0$. Eine Lösung S für $Min_{\varphi(X)}$, d.h. eine Menge $S \subseteq (V^{\mathcal{A}})^r$ mit $(\mathcal{A}, S) \models \varphi$, heißt ε-nah*, wenn $|S| \leqslant (1+\varepsilon)|X_{min}|$.*

Ein Polynomialzeit Approximationsschema (PTAS) *für $Min_{\varphi(X)}$ ist eine uniforme Menge $(A_\varepsilon)_{\varepsilon>0}$ von Algorithmen, wobei A_ε auf Eingabe $\mathcal{A}$ eine ε-nahe Lösung für $Min_{\varphi(X)}$ in Polynomialzeit berechnet. Uniform heißt dabei, dass es einen Algorithmus gibt, der auf Eingabe ε den Algorithmus A_ε berechnet.*

Ein PTAS heißt effizient (EFTPAS)*, wenn der Grad des Polynoms nicht von ε abhängt, d.h. die Laufzeit aller A_ε wird durch Polynome festen Grads beschränkt, nur die Konstanten ändern sich.*

Wiederum sind die entsprechenden Begriffe für Maximierungsprobleme analog definiert.

Ebenso wie im vorherigen Kapitel, kann man durch eine Formulierung von Approximationsproblemen in der Prädikatenlogik für ganze Klassen von Optimierungsproblemen zeigen, dass sie auf bestimmten Klassen von Graphen effizient approximiert werden können. Ein Beispiel dafür liefert der folgende Satz, der auf Klassen von Graphen mit verbotenen Minoren arbeitet. Ein Beispiel solcher Klassen ist die Klasse der planaren Graphen.

9.20 Satz (Dawar, Grohe, Kreutzer, Schweikardt 2006) *Sei $\sigma := \{E\}$ die Signatur der Graphen, $X \notin \sigma$ ein r-stelliges Relationssymbol und sei $\varphi(X) \in \mathrm{FO}[\sigma \dot\cup \{X\}]$ eine X-positive Formel. Sei $\mathcal{C}$ eine Klasse von Graphen mit verbotenen Minoren, z.B. die Klasse aller planaren Graphen.*

Dann hat $Min_{\varphi(X)}$ ein EPTAS auf der Klasse $\mathcal{C}$ von Graphen. Die analoge Aussage gilt für X-negative Formel und $Max_{\varphi(X)}$.

Teil III.

Logik und Komplexität

Kapitel 10.

Existentielle Logik zweiter Stufe und NP

Zum Abschluss der Vorlesung führen wir die *existentielle Logik zweiter Stufe* (∃SO) ein und zeigen, dass sie die Komplexitätsklasse NP auf der Klasse aller endlichen Strukturen beschreibt. Dieses Ergebnis von Fagin (1974) hat das Gebiet der *deskriptiven Komplexitätstheorie* begründet. Dort wird die Komplexität eines Problems nicht durch die Zeit oder den Platz beschrieben, den eine Turingmaschine braucht, um es zu entscheiden, sondern durch die logischen Mittel, die für seine Formulierung benötigt werden. Eine der wichtigsten offenen Fragen aus der deskriptiven Komplexitätstheorie bleibt die Frage, ob es eine Logik gibt, die genau die PTime-Probleme beschreiben kann.

10.1. Die Logik zweiter Stufe

Die *Logik zweiter Stufe* erweitert die Logik erster Stufe bzw. die monadische Logik zweiter Stufe durch die Möglichkeit, über *Relationen* über dem Universum einer Struktur zu quantifizieren. Dazu besitzt sie *Variablen zweiter Stufe*, genannt *Relationsvariablen*, die wir im Folgenden immer mit Großbuchstaben wie beispielsweise X, Y, Z oder Varianten davon bezeichnen, und Formeln der Form $\exists X\, \varphi$ und $\forall X\, \varphi$, die es ermöglichen, über Relationen zu quantifizieren. Jede Relationsvariable X hat eine Stelligkeit $ar(X) \in \mathbb{N}$. Sei Var_2 die Menge aller Relationsvariablen. Im Folgenden sei σ eine Signatur.

10.1 Definition *Wir erweitern die Syntax der Prädikatenlogik um folgende Regeln:*

- *Für alle* $X \in \mathrm{Var}_2$*,* $k = ar(X)$ *und alle* $v_1, \ldots, v_k \in \mathrm{Var}_1 \cup \{c \in \sigma : c$ *ist ein Konstantensymbol*$\}$ *ist* $X(v_1, \ldots, v_k) \in \mathrm{SO}[\sigma]$*.*
- *Für alle* $\varphi \in \mathrm{SO}[\sigma]$ *und* $X \in \mathrm{Var}_2$ *sind* $\exists X\, \varphi$ *und* $\forall X\, \varphi$ *in* $\mathrm{SO}[\sigma]$*.*

Die Semantik der Logik zweiter Stufe *ist auf offensichtliche Weise definiert. Z.B. gilt für jede* σ*-Struktur* $\mathcal{A}$*, jede Relationsvariable* X *und jede* $\mathrm{SO}[\sigma]$*-Formel* φ*:*

$$\mathcal{A} \models \exists X\, \varphi \iff \text{es gibt ein } R \subseteq (V^{\mathcal{A}})^{ar(X)}, \text{ so dass } (\mathcal{A}, R) \models \varphi.$$

Wir erweitern die grundlegenden Notationen und Begriffe, z.B. den Quantorenrang, freie Variablen, die Modellklasse einer Formel, usw., die wir für die Logik erster Stufe eingeführt haben, in naheliegender Weise auf die Logik zweiter Stufe.

10.2. Der Satz von Fagin

In diesem Abschnitt beweisen wir den Satz von Fagin, der besagt, dass die existentielle Logik zweiter Stufe (∃SO) die Komplexitätsklasse NP beschreibt.

10.2 Definition *Sei σ eine Signatur. Die Menge* $\exists\mathrm{SO}[\sigma]$ *der Formeln der* existentiellen $\mathrm{SO}[\sigma]$-Formeln *besteht aus allen* $\mathrm{SO}[\sigma]$*-Formeln der Gestalt*

$$\exists X_1 \cdots \exists X_k \, \varphi,$$

wobei $k \in \mathbb{N}$, $X_1, \ldots, X_k \in \mathrm{Var}_2$ und $\varphi \in \mathrm{FO}[\sigma \cup \{X_1, \ldots, X_k\}]$.

Wir fixieren ein Alphabet Σ und für jede Klasse $\mathcal{C}$ von endlichen Strukturen eine Kodierungsfunktion $\mathit{code} : \mathcal{C} \to \Sigma^*$.

10.3 Definition *Sei $\mathcal{L}$ eine Logik, $\mathcal{K}$ eine Komplexitätsklasse und $\mathcal{C}$ eine Klasse endlicher Strukturen.* $\mathcal{L}$ beschreibt $\mathcal{K}$ auf $\mathcal{C}$*, falls*

> *für jede (unter Isomorphie abgeschlossene) Klasse $\mathcal{D} \subseteq \mathcal{C}$ ein $\mathcal{L}$-Satz φ existiert so, dass*
>
> $$(\mathcal{A} \in \mathcal{D} \iff \mathcal{A} \models \varphi) \iff \{\mathit{code}(\mathcal{A}) : \mathcal{A} \in \mathcal{D}\} \in \mathcal{K}.$$

Wir können nun den Satz von Fagin formulieren:

10.4 Satz (Fagin, 1974) ∃SO *beschreibt* NP *auf der Klasse aller endlichen Strukturen.*

Beweis. Sei σ eine Signatur und $\mathcal{D}$ eine unter Isomorphie abgeschlossene Klasse endlicher σ-Strukturen. Wir müssen zeigen:

> Es gibt einen $\exists\mathrm{SO}[\sigma]$-Satz φ mit
>
> $$(\mathcal{A} \in \mathcal{D} \iff \mathcal{A} \models \varphi) \iff \{\mathit{code}(\mathcal{A}) : \mathcal{A} \in \mathcal{D}\} \in \mathrm{NP}.$$

Für die Hinrichtung sei φ ein $\exists\mathrm{SO}[\sigma]$-Satz φ so, dass $\mathcal{A} \in \mathcal{D} \iff \mathcal{A} \models \varphi$. Wir müssen zeigen, dass $\{\mathit{code}(\mathcal{A}) : \mathcal{A} \in \mathcal{D}\} \in \mathrm{NP}$, d.h. dass es eine polynomiell zeitbeschränkte NTM gibt, die das Auswertungsproblem für φ auf endlichen Strukturen entscheidet.

Angenommen,

$$\varphi = \exists X_1 \cdots \exists X_k \, \psi,$$

wobei $k \in \mathbb{N}$ und $\psi \in \mathrm{FO}[\sigma \cup \{X_1, \ldots, X_k\}]$.

Nach Satz 6.6 existiert eine DTM M, die das Auswertungsproblem für ψ auf jeder festen $\sigma \cup \{X_1, \ldots, X_k\}$-Struktur $\mathcal{B}$ in Zeit $\mathcal{O}(|B|^{|\psi|})$ entscheidet. Eine NTM kann nun wie folgt vorgehen, um das Auswertungsproblem für φ zu lösen.

Eingabe: $code(\mathcal{A})$ für eine endliche σ-Struktur $\mathcal{A}$
Frage: Gilt $\mathcal{A} \models \varphi$?

1. Für alle $i \in \{1, \dots, k\}$ „rate“ eine Belegung $R_i \subseteq (V^{\mathcal{A}})^{ar(X_i)}$ für X_i.
2. Simuliere M auf Eingabe $code(\mathcal{A}, R_1, \dots, R_k)$. Falls M akzeptiert, so akzeptiere; sonst verwerfe.

Da jede Relation $R_i \subseteq (V^{\mathcal{A}})^{ar(X_i)}$ durch ein Wort der Länge $n^{ar(X_i)} \log n$, für $n := |A|$, über dem Alphabet $\{0,1\}$ kodiert werden kann, kann Schritt 1 in polynomieller Zeit ausgeführt werden. Da M polynomiell zeitbeschränkt ist, reicht auch für Schritt 2 eine polynomielle Anzahl von Schritten aus. Die beschriebene NTM ist also polynomiell zeitbeschränkt und damit $\{code(\mathcal{A}) : \mathcal{A} \in \mathcal{D}\} \in \mathrm{NP}$.

Sei umgekehrt $code(\mathcal{D}) = \{code(\mathcal{A}) : \mathcal{A} \in \mathcal{D}\} \in \mathrm{NP}$, und sei M eine NTM, die $code(\mathcal{D})$ entscheidet. Wir konstruieren einen $\exists\mathrm{SO}[\sigma]$-Satz φ, so dass für alle endlichen σ-Strukturen $\mathcal{A}$ gilt:

$$\mathcal{A} \models \varphi \iff M \text{ akzeptiert } code(\mathcal{A}). \qquad (\star)$$

Wir verfahren ähnlich wie im Beweis des Satzes von Büchi und Elgot, Satz 4.1. Wir können natürlich nicht die Eingabe mit Zuständen beschriften, da die Eingabe die Länge n hat, die Maschine M aber möglicherweise Laufzeit n^k für ein k. Außerdem reicht es nicht den Zustand zu speichern, sondern wir benötigen auch die Bandbeschriftung. Wir können aber Konfigurationen in hinreichend großen Relationen kodieren.

Sei $M = (Q, \Gamma, \Sigma, \Delta, q_0, F)$. O.B.d.A. gibt es ein $k \in \mathbb{N}$, so dass für alle endlichen σ-Strukturen $\mathcal{A}$ mit Universum A und $n := |A|$ gilt:

1. Jeder terminierende Lauf von M bei Eingabe $code(\mathcal{A})$ hat *genau* die Länge $n^k - 1$.
2. $n^k \geqslant |code(\mathcal{A})|$.

Im Folgenden halten wir eine endliche σ-Struktur $\mathcal{A}$ mit Universum A fest und setzen $n := |A|$. Die Konstruktion von φ wird aber unabhängig von $\mathcal{A}$ sein.

Schritt 1: Kodierung von Zeitpunkten, Bandpositionen und Berechnungen.

Wir kodieren die n^k möglichen Zeitpunkte und Bandpositionen durch k-Tupel über A. Dazu verwenden wir eine lineare Ordnung $\leqslant$ auf A und kodieren ein Element $a \in \{0, \dots, n^k - 1\}$ durch das Tupel $\overline{a} \in A^k$ vom Rang k bzgl. der lexigraphischen Ordnung auf k-Tupeln. Für die lineare Ordnung $\leqslant$, die entsprechende Nachfolgerrelation $Succ$ und das minimale Element 0 bzw. das maximale Element max reservieren wir die Relationsvariablen $X_{\leqslant}, X_{Succ}$ und die Elementvariablen z_0, z_{max}.

Zur Kodierung von Berechnungen verwenden wir zusätzlich die Relationsvariablen

$$Z_q \ \text{(für alle } q \in Q), \quad K, \quad B_a \ \text{(für alle } a \in \Gamma),$$

wobei $ar(Z_q) = k$ und $ar(K) = ar(B_a) = 2k$ sind. Wir werden die folgenden Interpretationen $Z_q^{\mathcal{A}}$, $K^{\mathcal{A}}$ und $B_a^{\mathcal{A}}$ von Z_q, K und B_a erzwingen:

$$\bar{t} \in Z_q^{\mathcal{A}} \iff \text{zum Zeitpunkt } \bar{t} \text{ befindet sich } M \text{ im Zustand } q, \tag{10.1}$$

$$(\bar{t}, \overline{p}) \in K^{\mathcal{A}} \iff \text{zum Zeitpunkt } \bar{t} \text{ steht der Schreib-/Lesekopf auf Bandposition } \overline{p}, \tag{10.2}$$

$$(\bar{t}, \overline{p}) \in B_a^{\mathcal{A}} \iff \text{zum Zeitpunkt } \bar{t} \text{ steht an Bandposition } \overline{p} \text{ das Symbol } a. \tag{10.3}$$

Schritt 2: Generieren von code($\mathcal{A}$) in den Relationen über $\mathcal{A}$.

Zur Beschreibung der Startkonfiguration von M auf Eingabe $code(\mathcal{A})$ benötigen wir noch Formeln $\gamma_0(\overline{x})$ und $\gamma_1(\overline{x})$, so dass für alle $b \in \{0,1\}$ und alle $\overline{p} \in A^k$ gilt:

$$(\mathcal{A}, Succ, 0) \models \gamma_b(\overline{p}) \iff \text{in } code(\mathcal{A}) \text{ steht an Position } \overline{p} \text{ ein } b. \tag{10.4}$$

Wir betrachten hier nur den Fall, dass σ aus einem 2-stelligen Relationssymbol R und einem Konstantensymbol c besteht. Andere Signaturen können analog behandelt werden.

Sei also $\sigma = \{R, c\}$. Dann ist

$$code(\mathcal{A}) \;=\; \underbrace{1^n 0^{n^2-n}}_{\text{Länge } n^2}\underbrace{code(R^{\mathcal{A}})}_{\text{Länge } n^2}\underbrace{code(c^{\mathcal{A}})}_{\text{Länge } n^2},$$

also insbesondere $|code(\mathcal{A})| = 3n^2$. Die Positionen $0, \ldots, 3n^2 - 1$ kodieren wir durch ihre n-äre Darstellung der Länge k, d.h. durch Tupel der Form

$$(\underbrace{0, \ldots, 0}_{k-3}, p_{k-2}, p_{k-1}, p_k) \in A^k$$

mit $p_{k-2} \in \{0,1,2\}$ und $p_{k-1}, p_k \in A$.

Wir definieren nun:

$$\begin{aligned}\gamma_1(x_1, \ldots, x_k) \;\coloneqq\; \textstyle\bigwedge_{i=1}^{k-3} x_i = z_0 \;\wedge\; ((\;\; ((\; &\underbrace{x_{k-2} = z_0 \wedge x_{k-1} = z_0}_{\text{die ersten } n \text{ Positionen sind 1en}} \;) \\ &\vee (\underbrace{(X_{Succ}(z_0, x_{k-2}) \wedge R(x_{k-1}, x_k))}_{\text{die 1en in } codeR} \\ &\vee (\underbrace{(\text{„}x_{k-2} = z_0 + 2\text{“} \wedge x_{k-1} = 0 \wedge x_k = c)}_{\text{die 1 in } codec})),\end{aligned}$$

wobei „$x_{k-2} = z_0 + 2$“ eine Abkürzung für $\exists y\,((X_{Succ}(z_0, y) \wedge X_{Succ}(y, x_{k-2}))$ ist. Weiterhin definieren wir:

$$\begin{aligned}&\gamma_0(x_1, \ldots, x_k) \;\coloneqq \\ &\bigwedge_{i=1}^{k-3} x_i = z_0 \wedge ((x_{k-2} = z_0 \vee X_{Succ}(z_0, x_{k-2}) \vee \text{„}x_{k-2} = z_0 + 2\text{“}) \wedge \neg\gamma_1(x_1, \ldots, x_k).\end{aligned}$$

Es ist leicht zu sehen, dass (10.4) für alle $b \in \{0,1\}$ und alle $\overline{p} \in (V^{\mathcal{A}})^k$ erfüllt ist.

Schritt 3: Konstruktion des $\exists\mathrm{SO}[\sigma]$-Satzes φ.
Der $\exists\mathrm{SO}[\sigma]$-Satz φ hat nun die Form

$$\exists X_{\leqslant}\, \exists X_{Succ}\, \exists (Z_q)_{q\in Q}\, \exists K\, \exists (B_a)_{a\in\Gamma}\, \exists z_0\, \exists z_{max}\, ((\varphi_{\leqslant,Succ,0,max} \tag{10.5}$$
$$\wedge\, \varphi_{\text{Zustand}} \wedge \varphi_{\text{Kopf}} \wedge \varphi_{\text{Band}} \tag{10.6}$$
$$\wedge\, \varphi_{\text{Start}} \wedge \varphi_{\text{Schritt}} \wedge \varphi_{\text{Akzeptiere}}). \tag{10.7}$$

Die Formel $\varphi_{\leqslant,Succ,0,max}$ besagt hierbei, dass $X_{\leqslant}$ durch eine lineare Ordnung auf A, X_{Succ} durch die entsprechende Nachfolgerrelation und z_0, z_{max} durch das minimale bzw. maximale Element der Ordnung interpretiert wird. Es ist leicht zu sehen, dass diese Eigenschaften FO-definierbar sind.

Für Tupel $\overline{x} = (x_1, \ldots, x_k)$ und $\overline{y} = (y_1, \ldots, y_k)$ von Variablen schreiben wir

$$\overline{x} = \overline{y} \quad \text{statt} \quad \bigwedge_{i=1}^{k} x_i = y_i, \qquad \exists \overline{x} \quad \text{statt} \quad \exists x_1 \cdots \exists x_k, \qquad \forall \overline{x} \quad \text{statt} \quad \forall x_1 \cdots \forall x_k.$$

Die verbleibenden Formeln besagen nun Folgendes:

- φ_{Zustand} besagt, dass M sich zu jedem Zeitpunkt in genau einem Zustand befindet:

$$\varphi_{\text{Zustand}} \coloneqq \forall \overline{t} \left(\left(\bigvee_{q\in Q} Z_q(\overline{t}) \right) \wedge \bigwedge_{q'\in Q\setminus\{q\}} \neg Z_{q'}(\overline{t}) \right). \tag{10.8}$$

- φ_{Kopf} besagt, dass der Schreib-/Lesekopf zu jedem Zeitpunkt auf genau einer Bandposition steht:

$$\varphi_{\text{Kopf}} \coloneqq \forall \overline{t} \exists \overline{p} \left(K(\overline{t}, \overline{p}) \wedge \forall \overline{p}' \left(K(\overline{t}, \overline{p}') \to \overline{p} = \overline{p}' \right) \right). \tag{10.9}$$

- φ_{Band} besagt, dass zu jedem Zeitpunkt auf jeder Bandposition genau ein Symbol steht:

$$\varphi_{\text{Band}} \coloneqq \forall \overline{t} \forall \overline{p}\, (\bigvee_{a\in\Gamma} (B_a(\overline{t}, \overline{p}) \wedge \bigwedge_{a'\in\Gamma\setminus\{a\}} \neg B_{a'}(\overline{t}, \overline{p})). \tag{10.10}$$

- φ_{Start} besagt, dass M sich zum Zeitpunkt 0 (repräsentiert durch das Tupel $\overline{z}_0 \coloneqq (\overbrace{z_0, \ldots, z_0}^{k \text{ Mal}})$) in der Startkonfiguration befindet:

$$\varphi_{\text{Start}} \coloneqq Z_{q_0}(\overline{z}_0) \wedge K(\overline{z}_0, \overline{z}_0) \wedge \forall \overline{p}\, ((((B_0(\overline{z}_0, \overline{p}) \leftrightarrow \gamma_0(\overline{p})) \tag{10.11}$$
$$\wedge\, ((B_1(\overline{z}_0, \overline{p}) \leftrightarrow \gamma_1(\overline{p})) \tag{10.12}$$
$$\wedge\, ((B_{\square}(\overline{z}_0, \overline{p}) \leftrightarrow \neg\gamma_0(\overline{p}) \wedge \neg\gamma_1(\overline{p}))). \tag{10.13}$$

- φ_{Schritt} besagt: Falls M sich zum Zeitpunkt $\bar{t}$ in einer Konfiguration befindet, für die eine Nachfolgekonfiguration existiert, so befindet sich M zum Zeitpunkt $\bar{t}+1$ in einer solchen Nachfolgekonfiguration. Dazu sei $\Delta_0 \coloneqq \{(q,a) \ : \ (q,a,q',a',m) \in \Delta\}$. Dann:

$$\begin{aligned}
&\varphi_{\text{Schritt}} \ \coloneqq \\
&\forall \bar{t} \forall \bar{p} \bigwedge_{(q,a)\in\Delta_0} \Big[Z_q(\bar{t}) \wedge K(\bar{t},\bar{p}) \wedge B_a(\bar{t},\bar{p}) \rightarrow \\
&\qquad \exists \bar{t}' \exists \bar{p}' \Big(\bigvee_{\substack{q',a',m \text{ mit} \\ (q,a,q',a',m)\in\Delta}} \Big(Succ_{lex}(\bar{t},\bar{t}') \wedge \chi_m(\bar{p},\bar{p}') \\
&\qquad\qquad \wedge Z_{q'}(\bar{t}') \wedge K(\bar{t}',\bar{p}') \wedge B_{a'}(\bar{t}',\bar{p}) \\
&\qquad\qquad \wedge \forall \bar{p}'' ((\neg \bar{p}'' = \bar{p} \rightarrow \bigwedge_{a''\in\Gamma} ((B_{a''}(\bar{t}',\bar{p}'') \leftrightarrow B_{a''}(\bar{t},\bar{p}'')))\Big) \Big].
\end{aligned}$$

  Hierbei ist

$$\chi_1(\bar{p},\bar{p}') \ \coloneqq \ Succ_{lex}(\bar{p},\bar{p}'), \quad \chi_{-1}(\bar{p},\bar{p}') \ \coloneqq \ Succ_{lex}(\bar{p}',\bar{p}), \quad \chi_0(\bar{p},\bar{p}') \ \coloneqq \ \bar{p} = \bar{p}'.$$

  Außerdem ist $Succ_{lex}(x_1,\ldots,x_k,y_1,\ldots,y_k)$ eine FO$[X_{\leqslant}]$-Formel, die die lexikographische Ordnung auf k-Tupeln definiert.

- $\varphi_{\text{Akzeptiere}}$ besagt, dass M zum Zeitpunkt n^k-1 in einem akzeptierenden Zustand ist:

$$\varphi_{\text{Akzeptiere}} \ \coloneqq \tag{10.14}$$

Damit ist die Konstruktion von φ abgeschlossen. Per Induktion über die Zeitpunkte $t \in [0, n^k-1]$ überzeugt man sich leicht, dass:

$$\mathcal{A} \models \varphi \iff \text{es gibt einen Lauf von } M \text{ bei Eingabe } code(\mathcal{A}), \text{ der nach genau } n^k-1 \text{ Schritten in einem akzeptierenden Zustand hält.} \tag{10.15}$$

Insgesamt gilt also ($\star$), was zu zeigen war. □

Die obige Formel rät eine Ordnung, damit sie Konfigurationen einer Turingmaschine geeignet kodieren kann. Das Raten dieser Ordnung stellt sich als eines der größten Probleme für eine Logik für PTIME heraus. Es kann nämlich nicht garantiert werden, dass die Logik die Ordnung nur benutzt um Konfigurationen zu kodieren und diese nicht mißbraucht um einen schweren Teil des Problems zu lösen. So kann z.B. mit Leichtigkeit das Hamiltonpfadproblem gelöst werden, wenn die Knoten in der Pfadreihenfolge in der Kodierung auftauchen.

Teil IV.

Logik in der Verifikation

Kapitel 11.

Verifikation

- Transitionssysteme
- LTL und CTL
- Büchi-Automaten (??)
- Satz angeben, dass LTL-Formeln in Büchi-Automaten übersetzt werden können (aber nicht beweisen, damit spart man sich die Abschlusseigenschaften)
- Leerheitstest für Büchi-Automaten und somit Entscheidbarkeit und Model-Checking für LTL
- Beispiel SPIN Model-Checker in den Übungen
- Anwendung in der Medizin, Beispiel SOAMED
- Unendliche Bäume und Baumautomaten, Baumabwicklungen