Mehr als Worte : Automaten und Formale Sprachen für Bäume, Bilder und andere Strukturen

Vorlesung

Sommersemester 2002

Johannes Waldmann, Institut für Informatik
Universität Leipzig, `joe@informatik.uni-leipzig.de`

0.1 Vorwort

Formale Sprachen sind Mengen von Objekten. Wir können sie auf drei Weisen spezifizieren:

- durch Grammatiken (oder Ausdrücke) *erzeugen*
- durch Automaten *akzeptieren*
- durch logische Formeln *beschreiben*

(Nach Wolfgang Thomas: die *heilige Dreifaltigkeit*, [Tho97])

Übung. Finde jeweils passende Darstellungen für die Sprache L über dem Alphabet $\Sigma = \{0, 1, 2\}$ aller Wörter, bei denen rechts von jeder 1 wenigstens noch eine 2 vorkommt.

Übung. Der erste Gesichtspunkt (Grammatiken) ist wesentlich *algebraisch*: die Sprache erfüllt ein (Un-)Gleichungssystem. Welche Lösungen hat die Gleichung

$$L = \{a, b\}^* \setminus a \cdot L \cdot b$$

Sprachfamilien. Durch Einschränkung der Beschreibungsmittel (Beispiele: nur rechtslineare Grammatiken, nur endliche Automaten, nur Formeln erster Ordnung mit bestimmten Relationssymbolen) erhält man Sprachfamilien (Beispiele: reguläre Sprachen). Durch fortschreitende Einschränkungen erhält man Hierarchien von Familien. (Beispiel: Chomsky-Hierarchie.)

Die Theorie formaler Sprachen untersucht solche Familien und Hierarchien. Typische Fragen dabei sind:

- Echtheit oder Kollaps von Hierarchien (Beispiel: die Familie der regulären Sprachen ist eine echte Teilmenge der Familie der kontextfreien Sprachen)
- Äquivalenz verschiedener Definitionen einer Familie (Beispiel: die Familie von linear beschränkten Automaten akzeptierten Sprachen ist genau die Familie der von kontextsensitiven Grammatiken erzeugten Sprachen)
- Abschlußeigenschaften (Beispiel: die Familie der regulären Sprachen ist komplement-abgeschlossen, aber die Familie der kontextfreien Sprachen nicht)
- Entscheidbarkeiten (Beispiel: es ist entscheidbar, ob eine für kontextfreie Grammatiken G gilt, daß $L(G) \neq \emptyset$, aber es ist nicht entscheidbar, ob $L(G) = \Sigma^*$.
- Komplexitäten (Beispiel: für gegebene reguläre Grammatiken $G_1, \ldots, G_n$ zu entscheiden, ob $L(G_1) \cap \ldots \cap L(G_n) \neq \emptyset$, ist PSPACE-vollständig.)

Ein *allgemeiner* Automat (Turingmaschine)

- liest
- schreibt
- ändert eigenen Zustand
- läuft

- hält (akzeptierend oder ablehnend)

Ein *endlicher* Automat kann das alles auch, mit einer Ausnahme: er *schreibt nicht*.

Das ist eine starke Einschränkung, mit Auswirkungen:

- ("negativ"): die erzeugte Sprachfamilie ist recht klein (reguläre Wortsprachen)
- ("positiv"): hat aber gute Abschluß- und Entscheidbarkeits-Eigenschaften.

Insbesondere haben wir effektiven Abschluß gegen alle Boolesche Operationen, und das beweisen wir genau durch Automatenkonstruktionen. Die funktionieren grade deswegen, weil mehrere (parallel laufende) endliche Automaten nicht auf den (Eingabe/Arbeits-)Speicher schreiben dürfen. (Gegenbeispiel: zwei Kellerautomaten kann man nicht durch einen einzigen simulieren.)

Übung: finde eine Operation, gegen die die regulären Sprachen *nicht* abgeschlossen sind!

Mehr als Worte...

Einerseits kann man zufrieden sein: wenn man eine Aufgabenstellung soweit reduziert hat, bis es eine Frage über Mengen von Wörtern ist, die durch endliche Automaten beschrieben werden, dann ist es praktisch gelöst. (Beispiele: Text-Matching, reguläre Ausdrücke in Skript-Sprachen, Scanner als Compiler-Bestandteile.)

Andererseits sind nicht alles Probleme reduzierbar auf Wörter. Wir betrachten diese, durch Anwendungen motivierte Bereiche:

- Bilder: z. B. Pixel-Grafiken
- Bäume: z. B. Term-Bäume, wegen Term-Ersetzung, (funktionaler) Programmierung
- unendliche Objekte: modellieren Prozesse, die nicht halten (sollen)

Wir fragen uns jeweils, wie wir die Idee "für endliche Automaten ist im Prinzip alles konstruier- und entscheidbar" dorthin transportieren können.

Wir werden feststellen: einige Aussagen lassen sich trivial übertragen (Beispiel: reguläre Baumsprachen sind abgeschlossen bzgl. Vereinigung), andere nur mit Einschränkungen (Beispiel: bottom-up-Baum-Automaten kann man deterministisch machen, aber top-down-Baum-Automaten nicht immer), andere nur mit großer Mühe (Beispiel: Satz von Rabin: zur Komplementierung von Automaten über unendlichen Wörtern) und wieder andere überhaupt nicht (Beispiel: es ist nicht entscheidbar, ob ein endlicher Bild-Automat überhaupt irgendein Bild akzeptiert).

Insgesamt lernen wir dabei klassische

- Gegenstände
- Fragestellungen
- Methoden
- Ergebnisse

der Theorie formaler Sprachen kennen.

Inhaltsverzeichnis

Literaturverzeichnis

[Bor99] Bernd Borchert. Two dimensional languages. http://math.uni-heidelberg.de/logic/bb/2dpapers.html, 1999.

[CDG+97] Hubert Comon, Max Dauchet, Rémi Gilleron, Denis Lugiez, Sophie Tison, and Marc Tommasi. *Tree Automata Techniques and Applications*. http://www.grappa.univ-lille3.fr/tata/, 1997.

[EF99] Heinz-Dieter Ebbinghaus and Jörg Flum. *Finite Model Theory*. Springer, 1999.

[EH99] Joost Engelfriet and Hendrik Jan Hoogeboom. *Tree-Walking Pebble Automata*, pages 72–83. 1999. http://www.wi.leidenuniv.nl/home/engelfri/pebble.ps.gz.

[Eng99] Joost Engelfriet. Derivation trees of ground term rewriting systems. *Information and Computation*, 152(1):1–15, 1999. technical report version http://www.wi.leidenuniv.nl/TechRep/1996/tr96-25.html.

[GR97] Dora Giammarresi and Antonio Restivo. *Two-Dimensional Languages*, pages 215–267. Volume 3 of Rozenberg and Salomaa [RS97], 1997.

[GS97] Ferenc Gécseg and Magnus Steinby. *Tree Languages*, pages 1–68. Volume 3 of Rozenberg and Salomaa [RS97], 1997.

[KM01] Jarkko Kari and Cristopher Moore. New results on alternating and nondeterministic two-dimensional finite-state automata. In *Symposium on Theoretical Aspects of Computer Science (STACS)*, page ??, 2001. http://www.santafe.edu/~moore/pubs/afa.html.

[NS00] Frank Neven and Thomas Schwentick. On the power of tree-walking automata. In *Automata, Languages and Programming*, pages 547–560, 2000. ftp://ftp.minet.uni-jena.de/pub/schwentick/NS00a.ps.

[Rei01] Klaus Reinhardt. The $\#a = \#b$ pictures are recognizable. In *Proceedings of the 18th STACS, Dresden*, pages 527–538, 2001. http://www-fs.informatik.uni-tuebingen.de/~reinhard/publ.html.

[RS97] G. Rozenberg and A. Salomaa, editors. *Handbook of Formal Languages*. Springer, 1997.

[Sip89] Michael Sipser. Halting space-bounded computations. *Theoretical Computer Science*, pages 335–338, 1989.

[Tho90] Wolfgang Thomas. *Automata on Infinite Objects*, pages 133–192. Volume B of van Leeuwen [vL90], 1990.

[Tho97] Wolfgang Thomas. *Languages, Automata and Logic*, pages 389–456. Volume 3 of Rozenberg and Salomaa [RS97], 1997.

[vL90] Jan van Leeuwen, editor. *Handbook of Theoretical Computer Science*. Elsevier Science Publishers B.V., 1990.

[Wal01] Johannes Waldmann. Automaten, die auf bäumen spazierengehen. Vortrag, Institut f. Informatik, Leipzig, 2001. http://www.informatik.uni-leipzig.de/~joe/talk/twa.ps.

Kapitel 1

Bildsprachen

Literatur:

[GR97] Dora Giammarresi und Antonio Restivo: *Two-Dimensional Languages*, Handbook of Formal Languages, Band III, Seiten 215 – 267.

[Bor99] Bernd Borchert: *Two Dimesional Languages*, (Sammlung von Links auf Papers und Personen, enthält u. a. Online-Version von Giammarresi/Restivo)

1.1 Definitionen und Beispiele

Definition Σ^{**}, $\mathrm{breit}(), \mathrm{hoch}()$

Beispiele:

Alle Bilder der Breite drei: $\mathrm{DREI}(\Sigma) := \{p \in \Sigma^{**} : \mathrm{breit}(p) = 3\}$

Alle Quadrate mit 1 auf der Hauptdiagonalen, und 0 anderswo: $\mathrm{DIAG} := \{p \in \{0,1\}^{**} : \mathrm{breit}(p) = \mathrm{hoch}(p) \wedge \forall 1 \leq x, y \leq \mathrm{breit}(p) : ((p_{xy} = 1) \iff (x = y))\}$

Alle ungeraden Quadrate mit 1 in der Mitte (und sonst 0): $\mathrm{CENTER}_0 := \{p \in \{0,1\}^{**} : \exists n \in \mathbb{N} : \mathrm{breit}(p) = \mathrm{hoch}(p) = 2n+1 \wedge (\forall x, y : (p_{xy} = 1) \iff (x = y = n)\}$

Alle ungeraden Quadrate mit 1 in der Mitte (und sonst egal): $\mathrm{CENTER} := \{p \in \{0,1\}^{**} : \exists n \in \mathbb{N} : \mathrm{breit}(p) = \mathrm{hoch}(p) = 2n+1 \wedge p_{nn} = 1\}$

Alle Bilder, bei denen erste und letzte Spalte übereinstimmen: $\mathrm{FIRSTLAST}(\Sigma) := \{p \in \Sigma^{**} : \forall 1 \leq y \leq \mathrm{hoch}(p) : p_{1,y} = p_{\mathrm{breit}(p),y}\}$

Alle Bilder, bei denen die erste mit *irgendeiner* Spalte übereinstimmt: $\mathrm{FIRSTSOME}(\Sigma) := \{p \in \Sigma^{**} : \exists 1 \leq x \leq \mathrm{breit}(p) : \forall 1 \leq y \leq \mathrm{hoch}(p) : p_{1,y} = p_{x,y}\}$

1.2 Kreuz-Automaten

Ein endlicher Automat, der in vier Himmelsrichtungen laufen kann.

Definition 1. *Ein Kreuz-Automat (4NFA)* $A = (\Sigma, Q, q_0, q_a, q_r, \delta)$ *besteht aus einem Alphabet* Σ*, einer endlichen Zustandsmenge* Q*, einem Startzustand* $q_0 \in Q$*, einem akzeptierenden und einem ablehnenden Endzustand* $q_a, q_r \in Q$*, und einer Übergangsrelation*

$$\delta \subseteq (\Sigma \times (Q \setminus \{q_a, q_r\})) \times (Q \times \{L, R, U, D\}).$$

Dabei sind L, R, U, D *(left, right, up, down) die vier Himmelsrichtungen.*

Der Automat heißt deterministisch *(4DFA), wenn* δ *sogar eine Funktion ist.*

Definition 2. *Eine Konfiguration* (b, p, q) *eines 4NFA* $A = (\Sigma, Q, q_0, q_a, q_r, \delta)$ *besteht aus einem Bild* b*, einer Position* q *darin, und einem Zustand* q*.*

Sie heißt Startkonfiguration*, wenn* $q = q_0$ *und* $p = (1, 1)$*, Sie heißt akzeptierende Endkonfiguration, wenn* $q = q_a$*, und ablehnede Endkonfiguration, wenn* $q = q_r$*.*

Die Übergangsrelation δ *beschreibt eine Relation* $\vdash$ *auf Konfigurationen durch* $(b, p, q) \vdash (b, p', q') \iff ((q, b_p), (q', d)) \in \delta)$ *und* $p' = p + d$*.*

Hierbei bezeichnet $p + d$ die Position "von p aus einen Schritt in Richtung d".

Definition 3. *Der 4NFA* A *akzeptiert ein Bild* b*, wenn es* p *gibt, so daß* $(b, (1, 1), q_0) \vdash^* (b, p, q_a)$*.*

Damit der Automat den Rand des Bildes erkennt, wird es umrahmt (entspricht Leerzeichen auf dem Band der Turingmaschine). Konvention: wenn der Rahmen betreten wird, *muß* er im nächsten Schritt wieder verlassen werden (oder der Automat hält).

Für $n \times 1$-Bilder ist das (zunächst) *mehr* als eine endlicher Automat auf Wörtern (weil auf dem Bild dann immer noch Bewegung nach Osten *und* Westen möglich ist).

Das mögliche Verhalten solcher Automaten ist:

- hält und *akzeptiert* durch Erreichen eines Zustandes q_a
- hält und *lehnt ab* durch Erreichen eines Zustandes q_r
- rechnet unendlich lange

Definition 4. $\mathcal{L}(4DFA), \mathcal{L}(4NFA)$ *heißen die durch entsprechende Automatenklasse akzeptierte Sprachfamilien.*

Aufgabe 5. *Welche der Beispielsprachen sind 4DFA, 4NFA?*

1.3 Eine nicht deterministische Bildsprache

Nach Definition ist trivialerweise $\mathcal{L}(4DFA) \subseteq \mathcal{L}(4NFA)$. Die Inklusion ist echt.

Satz 6. *Es gibt eine Sprache $L \in \mathcal{L}(4NFA) \setminus \mathcal{L}(4DFA)$.*

Proof. $L =$ CENTER.

CENTER $\in \mathcal{L}(4NFA)$: Ein 4NFA prüft zunächst, daß das Bild ein Quadrat ist. Dann fährt er aus einer Ecke diagonal los, bei einer 1 rät er, im rechten Winkel abzubiegen, und fährt in dieser Richtung weiter. Kommt er in einer Ecke an, akzeptiert er.

CENTER $\notin \mathcal{L}(4DFA)$: Wir betrachten einen 4DFA A, der, wenn er akzeptiert, das unten rechts tut.

Jeder $m \times m$-Block im Inneren eines $p \in L(A)$ wird (evtl. mehrfach) betreten und auch verlassen.

Weil A deterministisch ist, gehört zu jedem möglichen Block eine Abbildung von Eintrittspunkt und -zustand auf Austrittspunkt und -zustand.

Es nur gibt $(4m|Q|)^{4m|Q|}$ solche Abbildungen, aber 2^{mm} verschiedene Blöcke.

Wenn m groß genug ist, gibt es deswegen zwei verschiedene Blöcke, zu denen die gleiche Funktion gehört. Der Automat kann also diese beiden Blöcke nicht unterscheiden.

Wir können sie deswegen in jedem akzeptierten Bild gegeneinander austauschen, und das Resultat wird immer noch akzeptiert. Damit können wir aber auch (durch geeignete Plazierung) eine 1 in der Mitte gegen eine 0 tauschen.

Deswegen gibt es keinen 4DFA, der CENTER erkennt. □

Aufgabe 7. *Im vorigen Beweis: wie groß ist m zu wählen, wenn $|Q| = 10$?*

1.4 Operationen auf Bildsprachen

Wir definieren Verklebe-Operationen für Bilder:

Definition 8. *Für Bilder p, q mit* $\mathrm{hoch}(p) = \mathrm{hoch}(q)$ *bezeichnet $p \oplus q$ das Bild "p neben q".*

Für Bilder p, q mit $\mathrm{breit}(p) = \mathrm{breit}(q)$ *bezeichnet $p \ominus q$ das Bild "p über q".*

Das sind partielle Operationen (wenn die Argumente nicht passen, gibt es kein Ergebnis).

Diese Operationen werden auf Sprachen ausgedehnt:

Definition 9. $L_1 \oplus L_2 := \{p_1 \oplus p_2 : p_1 \in L_1, p_2 \in L_2\}$, $L_1 \ominus L_2 := \{p_1 \ominus p_2 : p_1 \in L_1, p_2 \in L_2\}$

Diese Verkettungen kann man iterieren:

Definition 10. $L^{0\ominus} := \lambda, L^{1\ominus} := L, L^{(n+1)\ominus} := L \ominus L^{n\ominus}$,

$L^{*\ominus} := \bigcup_{n\geq 0} L^{n\ominus}$

Entsprechend für $\oplus$.

Beachte $(L^{*\ominus})^{*\oplus} = (L^{*\oplus})^{*\ominus}$, wir schreiben dafür einfach L^{**}.

Definition 11. *In einem* regulären Ausdruck *für Bildsprachen über einem Alphabet Σ sind zugelassen (und bedeuten):*

- $\emptyset$ *(leere Sprache)*
- *jedes $x \in \Sigma$ (das 1×1-Bild mit Inhalt x)*
- $L_1 \ominus L_2, L_1 \oplus L_2$ *(Verkettungen, oben definiert)*
- $L^{*\ominus}, L^{*\oplus}$ *(iterierte Selbst-Verkettungen, oben definiert)*
- $L_1 \cup L_2, L_1 \cap L_2$ *(Vereinigung, Durchschnitt)*
- $\overline{L}$ *(Komplement bzgl. Σ^{**})*

Definition 12. *Die Menge aller regulären Ausdrücke heißt* RE, *die Klasse der dadurch beschriebenen Sprache ist* $\mathcal{L}(\mathrm{RE})$.

(Vorsicht, nicht verwechseln mit r.e. = recursively enumerable.)

Definition 13. Komplementfreie *reguläre Ausdrücke heißen* CFRE.

Aufgabe 14. *Welche der Beispielsprachen sind* $\mathcal{L}(\mathrm{CFRE})$ *oder* $\mathcal{L}(\mathrm{RE})$*?*

1.5 Abschluß-Eigenschaften

Von Wortsprachen her sind wir gewohnt, daß die Klasse der von endlichen Automaten akzeptierten Sprachen gegen alle "vernünftigen" Operationen abgeschlossen ist.

Tatsächlich nicht alle:

Definition 15. *Für $w \in \Sigma^*$:*

$$\text{cycle}(w) := \{w_k w_{k+1} \dots w_n w_1 w_2 \dots w_{k-1} : n = |w| \wedge 1 \leq k \leq n\}$$

Definition 16. $\text{cycle}(L) := \{w' : w \in L, w' \in \text{cycle}(w)\}$

Definition 17. *Für $w \in \Sigma^*$:*

$$\text{perm}(w) := \{w_{\pi(1)} w_{\pi(2)} \dots w_{\pi(n)} : n = |w| \wedge \pi : \{1 \dots n\} \leftrightarrow \{1 \dots n\}\}$$

Definition 18. $\text{perm}(L) := \{w' : w \in L, w' \in \text{perm}(w)\}$

Satz 19. *Wenn $L \in \text{REG}$, dann $\text{cycle}(L) \in \text{REG}$.*

Satz 20. *Es gibt $L \in \text{REG}$, so daß $\text{perm}(L) \notin \text{REG}$.*

Aufgabe 21. *Beweise!*

Durch Geradeaus-Konstruktionen (Übungsaufgabe) weist man tatsächlich nach, daß $\mathcal{L}(4DFA)$ und $\mathcal{L}(4NFA)$ abgeschlossen gegen Vereinigung und Durchschnitt sind.

1.6 4NFA ist nicht verkettungs-abgeschlossen

(Definitionen siehe Beispiele) Offensichtlich $\text{FIRSTLAST} \in \mathcal{L}(4DFA)$, und erst recht $\Sigma^{**} \in \mathcal{L}(4DFA)$. Es gilt $\text{FIRSTSOME} = \text{FIRSTLAST} \oplus \Sigma^{**}$, aber

Satz 22. $\text{FIRSTSOME} \notin \mathcal{L}(4NFA)$

Ohne Beweis. Wer will, kann ja was basteln. Es sollte ein kombinatorisches Argument (ähnlich zu $\text{CENTER} \notin \mathcal{L}(4DFA)$) geben.

Das Beispiel zeigt, daß weder 4DFA noch 4NFA gegenüber $\oplus, \ominus$ abgeschlossen sind.

1.7 4DFA ist komplement-abgeschlossen

Komplementabschluß ist nicht von vornherein klar. Die Automaten sind zwar deterministisch, aber das Gegenteil von "akzeptieren" ist nach Definition "ablehnen oder unendlich lange rechnen". Wir müssen beweisen, daß sich das Ablehnen durch Nicht-Halten ersetzen läßt durch Ablehnen in endlicher Zeit.

Satz 23. *Zu jedem 4DFA A gibt es einen 4DFA A' mit $L(A) = L(A')$ und der Eigenschaft, daß* alle *Rechungen von A' endlich sind.*

D. h. falls $b \in L(A)$, dann $(b,(1,1),q'_0) \vdash^* (b,?,q'_a)$ und falls $b \notin L(A)$, dann $(b,(1,1),q'_0) \vdash^* (b,?,q'_r)$.

Proof. Das geschieht nach einem Argument von Sipser [Sip89], das allgemein für platzbeschränkte deterministische Maschinen funktioniert.

Wir durchsuchen, beginnend mit allen akzeptierenden Konfigurationen, den *umgekehrten* Berechnungsbaum mit Tiefensuche.

Die wesentliche Beobachtungen sind:

- eine deterministische und akzeptierende Rechnung kann keine Schleife enthalten, deswegen ist der Baum endlich,
- wir können die Tiefensuche so organisieren, daß wir keinen unbeschränkten Hilfsspeicher (Stack) brauchen.

Wir gehen jeweils von einem Knoten (einer Konfiguration) (b,p,q) zu einem Vorgänger $(b,p',q') \vdash (b,p,q)$. Diese Vorgänger sind durch Invertieren der Übergangsrelation von A leicht zu finden.

Falls (b,p,q) eine Startkonfiguration ist, dann akzeptieren wir, denn $b \in L(A)$.

Falls (b,p,q) keine Vorgänger besitzt, dann müssen wir "backtracken". Dazu gehen wir in den Nachfolger (b,p',q') von (b,p,q) und von dort sofort in den nächsten (d. h. zu (b,p,q) benachbarten) Vorgänger (und falls wir selbst der letzte waren, dann das Ganze eins weiter oben). Das geht tatsächlich ohne Stack. Wir müssen feststellen, woher wir kommen. Dazu erzeugen vergleiche wir alle Vorgänger von $(b.p',q')$ (in der fixierten Reihenfolge) mit (b,p,q), und bemerken auf diese Weise, "wer wir sind". Dieses Verfahren läßt sich also in einem 4DFA implementieren.

Jetzt machen wir uns klar, daß es auch immer hält: Da das back-tracken korrekt implementiert ist, geht es höchstens darum, daß wir beim Voranschreiten (Absteigen) eine Konfiguration betreten, die wir (weiter oben im Baum) schon einmal sahen.

Genau das ist aber ausgeschlossen, da wir bei einer akzeptierenden Konfiguration begonnen hatten, und auf keinem Pfad dorthin sich eine Schleife befinden kann. □

Satz 24. *4DFA ist komplement-abgeschlossen:*
wenn $L \in \mathcal{L}(4DFA)$, dann $\Sigma^{**} \setminus L \in \mathcal{L}(4DFA)$.

Proof. Für einen 4DFA A für L konstruieren wir nach dem vorigen Satz den haltenden Automaten $A' = (\Sigma, Q', q'_0, q'_a, q'_r, \delta')$, der zu A äquivalent ist. Dann akzeptiert der 4DFA $(\Sigma, Q', q'_0, q'_r, q'_a, \delta')$ genau das Komplement von L. □

1.8 Alternierende Automaten

Im klassischen nichtdeterministischen Automaten sind die Zustände *existenziell*, d. h. wir akzeptieren, wenn *eine* akzeptierende Fortsetzung der Rechnung *existiert.*

Man kann nun zusätlich *universelle* Zustände einführen, von denen aus *alle* möglichen Forsetzungen der Rechnung akzeptieren sollen.

Weil also zwischen existentiellen und universellen Zuständen gewechselt wird, nennt man das Maschinenmodell *alternierender* (endlicher) Automat.

(Das ist ein ganz allgemein anwendbares Prinzip, es spielt dabei keine Rolle, ob der Automat endlich ist oder auf welchen Strukturen er operiert. Es gibt alternierende Turingmaschinen, usw.).

Definition 25. *Ein alternierender Kreuz-Automat (4AFA) $A = (A', s)$ besteht aus einem 4NFA A' und einer Abbildung $s : Q(A') \to \{\forall, \exists\}$.*

Eine Konfiguration von A ist eine Konfiguration von A'.

Definition 26. *Für einen 4AFA A ist Konfiguration (b, p, q) erfolgreich, wenn*

- $q = q_a$ *oder*
- $s(q) = \forall$ *und alle* (b', p', q') *mit* $(b, p, q) \vdash (b, p', q')$ *erfolgreich sind, oder*
- $s(q) = \exists$ *und wenigstens ein* (b', p', q') *mit* $(b, p, q) \vdash (b, p', q')$ *erfolgreich ist.*

Das Bild b wird von A akzeptiert, wenn $(b, (1, 1), q_0)$ erfolgreich ist.

Beachte dabei, daß die Bedingung im zweiten Fall auch dann erfüllt ist, wenn die Konfiguration *gar keine* Nachfolger besitzt.

Wir können uns das auch als Spiel vorstellen: wenn wir beweisen wollen, daß $b \in L(A)$, dann sind *wir* bei jeder Konfiguration (b, p, q) mit $s(q) = \exists$ am Zug und dürfen uns einen (erfolgreichen) Nachfolger aussuchen. Falls jedoch $s(q) = \forall$, dann ist "der andere" dran. Das Spiel endet, wenn wir q_a erreichen. Das Eingabe gilt dann als akzeptiert, wenn "wir" eine Gewinnstrategie besitzen (mit der wir q_a erreichen). Erreicht der andere q_r, oder eine Schleife, dann haben wir verloren.

Die klassischen nichtdeterministischen 4NFA entsprechen damit Solitaire-Spielen (der andere kommt nie dran).

Falls eine Konfiguration *genau einen* Nachfolger besitzt, dann ist es egal, ob es ein $\exists$- oder ein $\forall$-Knoten ist.

Offensichtlich gilt

Satz 27. $\mathcal{L}(4DFA) \subset \mathcal{L}(4NFA) \subseteq \mathcal{L}(4AFA)$.

Wir sehen demnächst, daß auch die zweite Inklusion echt ist.

1.9 Ein Beispiel einer 4AFA-Sprache

(Abgeschrieben aus [KM01].)

Definition 28. *Eine Permutations-Matrix enthält in jeder Spalte und in jeder Zeile genau eine 1, und sonst nur 0. Die Sprache aller Permutations-Matrizen nennen wir P.*

Satz 29. $P \in \mathcal{L}(4DFA)$.

Proof. Der Automat prüft, ob in jeder Zeile genau eine 1 steht, und danach, ob in jeder Spalte genau eine 1 steht. □

Bemerke: dann ist das bild automatisch quadratisch.

Definition 30. $P_2 = \{b \varobar 2^{+\ominus} \varobar b : b \in P\}$.

Das sind alle Bilder vom Format $(2n+1) \times n$, die aus zwei identischen Permutations-Matrizen über $\{0, 1\}$ bestehen, die durch eine Spalte 2 getrennt sind, beispielsweise:

$$\begin{pmatrix} 1 & 0 & 0 & 2 & 1 & 0 & 0 \\ 0 & 0 & 1 & 2 & 0 & 0 & 1 \\ 0 & 1 & 0 & 2 & 0 & 1 & 0 \end{pmatrix} \in P_2$$

Satz 31. $P_2 \in \mathcal{L}(4AFA)$.

Proof. Zunächst prüft der Automat, daß das vorgelegte Bild in $P \varobar 2^{*\ominus} \varobar P$ liegt. (Das kann sogar ein 4DFA.) Danach wissen wir, daß rechts und links zwei Permutationsmatrizen stehen, zu verifizieren ist noch, daß beide identisch sind.

Wir verzweigen *universell* zu allen Positionen im linken Bild, auf denen eine 1 steht, und führen dann die folgende (deterministische) Rechnung aus:

1. gehe wenigstens eine Spalte nach rechts
2. wenn Zeichen 2 oder Rand, dann akzeptiere
3. sonst suche die 1 in der aktuellen Spalte,
4. dann suche die andere 1 in der aktuellen Zeile (d. h. in der anderen Hälfte des Bildes)
5. wiederhole von 1.

Falls die Eingabe b tatsächlich $\in P_2$ war, dann halten alle diese Rechnungen akzeptierend, und deswegen wird die gesamten Eingabe akzeptiert.

Falls die beiden Permutationsmatrizen jedoch nicht übereinstimmen, dann wird wenigstens eine dieser Rechnungen nicht halten, und deswegen (nach Definition) die Eingabe abgelehnt. □

Aufgabe 32. *Beweise den zweiten Teil der Behauptung.*

Aufgabe 33. *Warum muß man universell verzweigen, d. h. die Rechnung auf* jeder *1 starten? Zeige, daß eine einzige Rechnung (beginnend in der äußersten linken Spalte) nicht immer das korrekte Resultat liefert.*

1.10 Reguläre Ausdrücke mit Projektionen

Die Sprache aller Quadrate über dem Alphabet $\Sigma = \{0\}$ kann man offenbar nicht durch einen regulären Ausdruck beschreiben.

Beispiel 34. *Mit einem größeren Alphabet schaffen wir wenigstens (einige) quadratische Bilder:*

$$L = (0^{*\ominus} \ominus 1 \ominus 2^{*\ominus})^{*\oplus} \cap (2^{*\oplus} \oplus 1 \oplus 0^{*\oplus})^{*\ominus}$$

erzeugt alle Bilder der Form

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 2 & 1 & 0 & 0 \\ 2 & 2 & 1 & 0 \\ 2 & 2 & 2 & 1 \end{pmatrix}$$

Mit der Abbildung (Projektion) $\pi : 0 \mapsto 0, 1 \mapsto 0, 2 \mapsto 0$ *gilt dann tatsächlich* $\mathrm{SQUARES}(0) = \pi(L)$.

Definition 35. *Ein komplementfreier regulärer AUsdruck mit Projektion ist ein Paar* (E, π), *wobei* $E \in \mathrm{CFRE}$ *und* $\pi : \Sigma \to \Sigma$.

Die Menge der durch PCFRE *definierbaren Bildsprachen heißt* $\mathcal{L}(\mathrm{PCFRE})$.

Aufgabe 36. *Welche der bisher genannten Beispielsprachen sind PCFRE?*

Autotool: Projektionen sind noch nicht implementiert. Ich weiß noch nicht, wie man das Suchen der Urbilder der Projektion sinnvoll organisieren kann.

1.11 Kachel-Systeme

Wir bleiben bei der Idee "regulärer Ausdruck mit Projektion", aber schränken den Ausdruck selbst noch ein, und gestatten nur Ausdrücke, die gewisse Teilbilder (von beschränkter Größe) verbieten. Die dann noch erlaubten Teilbilder dieser Größe heißen *Kacheln* (tiles).

Definition 37. *Für ein Bild* $p \in \Sigma^{**}$ *bezeichnet* $\mathrm{sub}_{b,h}(p)$ *die Menge aller* (b, j)*-Teilbilder. Für* $b = h = 2$ *schreiben wir einfach* $\mathrm{sub}(p)$*.*

Definition 38. *Für ein Bild* $p \in \Sigma^{**}$ *bezeichnet* $p_{\#}$ *das mit* $\#$ *gerahmte Bild.*

Definition 39. *Jede Menge* $E \subseteq (\Sigma \cup \{\#\})^{b \times h}$ *definiert die Sprache*

$$\mathrm{LOC}(E) := \{p \in \Sigma^{**} : \mathrm{sub}(p_{\#}) \subseteq E\}$$

Jede so definierte Sprache heißt (b, h)*-lokal. Die Menge aller so definierbaren Sprachen ist* $\mathcal{L}(\mathrm{LOC}(b, h))$*.*

Aufgabe 40. *(autotool) Vervollständige die Kachelmenge für die Sprache aus Beispiel 34*

$$E = \{ \begin{matrix} \#\# \\ \#1 \end{matrix}, \begin{matrix} \#\# \\ 10 \end{matrix}, \begin{matrix} \#\# \\ 0\# \end{matrix}, \begin{matrix} \#1 \\ \#2 \end{matrix}, \ldots \}$$

Definition 41. *Ein* (b, h)*-Kachel-(tiling-)System ist ein Paar* (E, π)*, mit einer Menge* $E \subseteq (\Sigma \cup \{\#\})^{b \times h}$ *und eine Projektion* $\pi : \Sigma \to \Sigma'$*. Es definiert die Sprache* $\mathrm{TS}(E, \pi) = \pi(\mathrm{LOC}(E)) \subseteq \Sigma'^{**}$*. Die Menge aller so definierbaren Sprachen heißt* $\mathcal{L}(\mathrm{TS}(b, h))$*.*

Ist (b, h) nicht angegeben, dann meinen wir $(2, 2)$.

1.12 Eigenschaften von TS

Satz 42. *Jede einelementige Sprache* $L = \{p\}$ *ist in* $\mathcal{L}(\mathrm{TS}(2,2))$.

Proof. Geradeaus-Konstruktion. □

Satz 43. $\mathcal{L}(\mathrm{TS})$ *ist abgeschlossen gegen* $\ominus, \ominus*, \oplus, \oplus*$.

Proof. Geradeaus-Konstruktion. □

Satz 44. $\mathcal{L}(\mathrm{TS})$ *ist abgeschlossen gegen* $\cap$ *und* $\cup$.

Proof. Geradeaus-Konstruktion. □

Corollar 45. *Alle endlichen Sprachen sind in* $\mathcal{L}(\mathrm{TS})$.

Corollar 46. $\mathcal{L}(\mathrm{TS}) \subseteq \mathcal{L}(\mathrm{PCFRE})$.

Proof. Das folgt sofort aus den vorigen Konstruktionen. □

Satz 47. $\mathcal{L}(\mathrm{TS})$ *ist nicht abgeschlossen gegen Komplement (Differenz).*

Proof. Siehe Definition 30. Es gilt $\{0,1,2\}^{**} \setminus P_2 \in \mathcal{L}(\mathrm{TS}))$ (beweise!), aber $P_2 \notin \mathcal{L}(\mathrm{TS})$. Letzteres zeigt man durch ein Anzahlargument. □

Satz 48. $\mathcal{L}(\mathrm{TS}(b,h)) \subseteq \mathcal{L}(\mathrm{TS}(2,2))$.

Proof. Moral: wir können die maximale Kachelgröße verringern, müssen aber dazu das Alphabet vergrößern. Das "macht nichts", da wir schließlich wieder auf das kleinere Alphabet projizieren können. □

1.13 4NFA und Tiling-Systeme

Satz 49. $\mathcal{L}(\mathrm{LOC}(b,h)) \subseteq \mathcal{L}(4DFA)$

Proof. Der 4DFA läuft über alle Positionen und prüft (durch komplettes Absuchen eines beschränkten Teilbildes), ob ein verbotenes Muster erscheint. □

Weder ein 4DFA noch ein 4NFA können jedoch das Urbild einer Projektion raten. (Wenn sie zum zweiten mal an einer Stelle vorbeikommen, wissen sie nicht mehr was sie beim ersten mal geraten hatten.) "Im Gegenteil", ein Kachel-System kann einen 4NFA simulieren. Das ist nicht von vornherein klar, denn der Automat läuft beliebig über das Bild, die Überdeckung wird aber nur einmal gewählt.

Satz 50. $\mathcal{L}(4NFA) \subseteq \mathcal{L}(\mathrm{TS})$.

Proof. Wir betrachten eine beliebige Sprache $L \in 4NFA(\Sigma)$ und einen Automaten $A = (\Sigma, Q, q_0, q_a, q_r, \delta)$ für L, und konstruieren daraus ein Domino-System für L.

Für die darin enthaltene lokale Sprache wählen wir ein Alphabet $\Sigma' = \Sigma \times H$ für eine geeignete endliche Menge H. Die Projektion π ist dann $(c, h) \mapsto c$. Das bedeutet, das Tiling-System rät für jede Position ein $h \in H$.

Wir haben dann zu zeigen, daß diese Aussagen äquivalent sind:

1. es gibt einen akzeptierenden Lauf von A auf dem Bild $b \in \Sigma^{**}$ (der oben links beginnt und unten rechts endet)
2. es gibt ein Bild $b' \in \Sigma'^{**}$ mit $\pi(b') = b$, und die Teilbilder von b' sind nur erlaubte Kacheln.

Die Komponenten h kodieren, auf welche Weise der Automat diese Position im Verlaufe einer Rechnung betrat und verließ. D. h. wir wählen $Q' = \{R, L, U, D\} \times Q \times \{R, L, U, D\}$ und $H =$ alle Mengen mit Elementen aus Q', bei denen die mittleren Elemente (Zustände) paarweise verschieden sind.

Die Kacheln kodieren, daß zwei nebeneinanderliegende Positionsfolgen kompatibel sind, das bedeutet, daß jede Rechnung nach vorn und hinten (in der Zeit) lokal eindeutig fortgesetzt werden kann.

Das Paar $((c_1, h_1), (c_2, h_2))$ ist als horizontale (liegende) Kachel erlaubt, falls $h_1 >_{c_1} h_2$ und $h_1 <_{c_2} h_2$, wobei $h_1 >_c h_2$ (h_1 und h_2 sind c-rechtskompatibel), falls eine umkehrbar eindeutige Zordnung ϕ zwischen der Menge aller $(*, q_1, R) \in h_1$ und der Menge aller $(R, q_2, *) \in h_2$ existiert, so daß $\phi(*, q_1, R) = (R, q_2, *) \Rightarrow \delta(c, q_1) \ni (q_2, R)$. Analog definierren wir links-kompatible $<_c$; und (für vertikale Kacheln) up- und down-kompatibel.

(1) $\to$ (2): Es genügt, akzeptierende Rechnungen des 4NFA zu betrachten, bei denen sich keine Konfiguration wiederholt. (Ansonsten könnten wir die Schleife löschen, und erhalten eine kürzere akzeptierende Rechnung.)

Es genügt weiterhin, nur Automaten zu betrachten, die oben links beginnen (das hatten wir sowieso schon festgelegt), und die unten rechts akzeptieren (das läßt sich durch zusätzliche Zustände leicht erreichen).

Jede solche Rechnung ist eine Folge

$$[(p_0, q_0, c_0), d_1, (p_1, q_1, c_1), d_2, \ldots, d_n, (p_n, q_a, c_n)]$$

(mit p_i : Position, q_i : Zustand, $d_i \in \{L, R, U, D\}$: Richtung) und alle q_i paarweise verschieden.

Aus dieser Folge kann man leicht rekonstruieren, welche Buchstaben $\in \Sigma'$ zu raten sind und welche Kacheln man zur Überdeckung auswählt.

(2) $\to$ (1): folgt aus der lokal eindeutigen Fortsetzbarkeit.

Wir es global so ein, daß oben links nur Buchstaben $(*, [(X, q_0, *)]) \in \Sigma^*$ stehen dürfen (X bezeichnet "hat keinen Eingang"), und unten rechts nur Buchstaben $(*, [(*, q_a, X)]) \in \Sigma^*$ ("kein Ausgang").

Damit muß oben links ein Pfad (im Startzustand) beginnen, und unten rechts ein Pfad (im akzeptierenden Zustand) enden. Auf allen anderen Punkten kann ein Rechnungspfad eintreten, muß dann aber wieder austreten. Am Bildrand darf kein Pfeil nach außen zeigen. Damit muß es einen durchgehenden Pfad vom Starten zum Akzeptieren geben. (Es kann außerdem mehrere geschlossene Kreise geben, und auch Felder, über die gar kein Pfad führt.) □

Aufgabe 51. *Für angenommenes $|\Sigma| = 3, |Q| = 5$, finde obere Schranken für $|\Sigma'|$ und die Anzahl der Kacheln, die in diesem Beweis konstruiert werden!*

Aufgabe 52. *Erkläre, wie durch diese Konstruktion Positionen behandelt werden, über die der Automat nie läuft.*

Aufgabe 53. *Erkläre genau, mit welchen Kacheln man die Ecken und Ränder behandelt.*

In diesem Beweis haben wir tatsächlich sehr spezielle Kacheln produziert:

Definition 54. *Ein* Domino-System *(DS) (E, π) ist ein Kachel-System, dessen Kacheln die Form 1×2 und 2×1 haben. Die von DS akzeptierten Sprachen heißen* $\mathcal{L}(\mathrm{DS})$.

Die oben bewiesene Aussage ist:

Satz 55. $\mathcal{L}(4NFA) \subseteq \mathcal{L}(\mathrm{DS})$ □

Satz 56. $\mathcal{L}(\mathrm{DS})$ *ist nicht komplement-abgeschlossen.*

Beweis: folgende Aufgabe

Aufgabe 57. *Die Sprache P_2 (Definition 30) ist nicht in* $\mathcal{L}(\mathrm{DS})$.

Das Komplement von P_2 ist in $\mathcal{L}(\mathrm{DS})$.

Satz 58. *Wenn $L \in \mathcal{L}(4AFA)$, dann $\Sigma^{**} \setminus L \in \mathcal{L}(\mathrm{DS})$.*

Beweis: siehe [KM01] □

1.14 Domino-Systeme und reguläre Ausdrücke

Satz 59. $\mathcal{L}(\mathrm{DS}) = \mathcal{L}(\mathrm{TS}(2,2))$.

Proof. Die Richtung $\mathcal{L}(\mathrm{DS}) \subseteq \mathcal{L}(\mathrm{TS})$ ist klar. Andersherum ist zu zeigen, wie wir 2×2-Kacheln durch Dominos simulieren. Das funktioniert deswegen, weil solche kleinen quadratischen Kacheln bei Überlappung (einer Zeile bzw. einer Spalte) von weitem wie Dominos aussehen. Genauer: [GR97] □

Jetzt können wir (endlich) die Verbindung zu regulären Ausdrücken herstellen:

Satz 60. $\mathcal{L}(\mathrm{DS}) \subseteq \mathcal{L}(\mathrm{PCFRE})$.

Proof. Für ein $L \in \mathcal{L}(\mathrm{DS})$ bezeichnen wir mit $H \subseteq \Sigma^{2,1}$ bzw. $V \subseteq \Sigma^{1,2}$ die Menge der liegenden bzw. stehenden Dominos. Wir konstruieren $L = L_H \cap L_V$, wobei L_H alle durch H-Dominos erlaubten Bilder sind, und L_V alle durch V-Dominos erlaubten.

Betrachten wir alle Zeilen, die in L_H vorkommen dürfen. Das sind alle String, die kein H enthalten. Das ist eine reguläre Sprache, für die es einen regulären Ausdruck A_H gibt (im klassischen Sinne, d. h. mit $\cdot$ und $*$ für Wörter, *und ohne Mengendifferenz*). Daraus können wir einen CFRE für die Bildsprache L_H bauen: $L_H = A_H^{*\ominus}$.

Analog konstuieren wir spaltenweise $L_R = A_R^{T*\oplus}$, und sind damit fertig. □

Die in diesem Beweis durchgeführt Konstruktion bekommt einen eigenen Namen:

Definition 61. *Für Wortsprachen $A_R, A_H \in \Sigma^*$ bezeichnet $A_R \oplus A_H$ die Bildsprache $L = A_R^{T*\oplus} \cap A_H^{*\ominus}$.*

Wegen $\mathcal{L}(\mathrm{PCFRE}) \subseteq \mathcal{L}(\mathrm{TS})$ und $\mathcal{L}(\mathrm{TS}) = \mathcal{L}(\mathrm{DS})$ folgt sogar

Satz 62. $\mathcal{L}(\mathrm{PCFRE}) = \mathcal{L}(\mathrm{TS}) = \mathcal{L}(\mathrm{DS})$.

Man nennt diese Sprachfamilie die *regulären* Bildsprachen, abgekürzt REG.

Wir haben gezeigt:

Satz 63. *Zu jeder regulären Bildsprache $L \in \Sigma^{**}$ existieren Wortsprachen $A_L, A_H \in \Sigma^*$ mit $L = A_R \oplus A_H$.* □

Reguläre Sprachen sind mächtiger als (existenzielle) Automaten:

Satz 64. $\mathcal{L}(LOC) \subseteq \mathcal{L}(4DFA) \subseteq \mathcal{L}(4NFA) \subseteq \mathcal{L}(\mathrm{TS})$ □

und alle Inklusionen sind tatsächlich echt.

Aufgabe 65. *Wie sieht die entsprechende Behauptung für Wortsprachen aus? Sind die Inklusionen (a) wahr, (b) echt?*

Durch Betrachtung der Bildsprachen haben wir demnach feine Unterschiede festgestellt, die für Wortsprachen verwischen.

1.15 Reguläre Bildsprachen und Zähler

Wir betrachten Sprachen über dem Alphabet $\Sigma = \{0, 1\}$:

- L_1: alle Quadrate, bei denen die unterste Zeile aus der Sprache $\{0^n 1^n : n \geq 0\}$ ist (andere Zeilen sind egal)
- L_2: alle Quadrate, bei denen die unterste Zeile aus der Sprache $\{w : \text{anzahl}_0(w) = \text{anzahl}_1(w)\}$ ist (andere Zeilen egal) (d. h. Zählen mit linear viel Platz)
- L_3: alle Rechtecke der Höhe h und der Breite $b = 2^h$, bei denen die unterste Zeile aus der Sprache $\{w : \text{anzahl}_0(w) = \text{anzahl}_1(w)\}$ ist (andere Zeilen egal) (d. h. Zählen mit logarithmisch viel Platz)
- L_4: alle Quadrate der Seitenlänge 2^n mit $\text{anzahl}_0(b) = \text{anzahl}_1(b)$. (d. h. *alle* Zeichen werden gezählt) (Hinweis: vorige Teilaufgabe)

Aufgabe 66. *Begründe, daß alle diese Sprachen regulär sind.*

Die Idee zu L_4 stammt von Klaus Reinhardt und ist hier beschrieben: [Rei01]. Es gibt dort auch ein Java-Applet, mit dem man sich die entsprechenden Tilings zusammenclicken kann. Sehr zu empfehlen!

Anstatt also Klaus' Artikel abzuschreiben, hier nur ein paar Erläuterungen.

Das Tiling-System für L_1 bekommt man leicht, indem man über dem Wort auf der untersten Zeile eine Pyramide errichtet.

Diese Pyramide sollte man sich tatsächlich als ein Ableitungsbaum (der Grammatik $S \to \epsilon \mid 0S1$) vorstellen.

Damit kann man auch L_2 lösen, ausgehend von der Grammatik $S \to \epsilon \mid 0S1S \mid 1S0S$.

Allerdings sind diese Bäume (ungefähr) so hoch wie breit, deswegen funktioniert der Ansatz nicht für L_3 mit nur logarithmischer Höhe.

Während bisher die Höhe (einer Pyramide bzw. eines Baumes) der Buchstabenanzahl (-differenz) entsprach, kodieren wir jetzt nicht unär, sondern binär: ein Element in Höhe h bedeutet eine Differenz von 2^h.

Dazu errichten wir über dem Wort in der untersten Zeile einen vollständigen binären Baum. Das Tiling-System "läuft" schichtweise von links nach rechts durch den Baum und inkrementiert/dekrementiert, wobei Überträge nach oben wandern, und "sein" Zustand stets $\in \{-1, 0, 1\}$ ist.

Nun haben wir bei Quadraten, in denen wir *alle* Buchstaben zählen sollen, aber gar keinen Platz für solche zusätzlichen Rechnungen.

Doch! Weil wir projizieren können, ist die Aufgabe ist äquivalent dazu, nur die Buchstaben auf ungeraden Positionen zu zählen. Damit haben wir alle Felder mit wenigstens einer geraden Koordinate frei für Zwischenrechnungen. Auf diese Felder legen wir Quadratgitter der Rastermaße $2^2, 2^3, 2^4, \ldots$.

Kapitel 2

Baumsprachen

Literatur:

[GS97] Ferenc Gegseg und Magnus Steinby, *Tree Languages*, Handbook of Formal Languages, Band III, Seiten 1 – 68.

[CDG+97] Hubert Comon et al. *Tree Automata Techniques and Applications*, http://www.grappa.univ-lille3.fr/tata/

2.1 Was ist überhaupt ein Baum?

2.1.1 Ungeordnete Bäume

Aus der Graphentheorie ist bekannt:

Definition 67. *Ein* Baum *ist ein kreisfreier zusammenhängender ungerichteter Graph.*

Aufgabe 68. *Beweise (Wdhl. 1. Semester!) Folgende Aussagen sind äquivalent:*

1. $G = (V, E)$ *ist ein Baum*
2. G *ist zusammenhängend und* $|E| = |V| - 1$
3. G *ist minimal zusammenhängend (d. h.: wenn man irgendeine Kante aus* G *löscht, ist das Resultat nicht mehr zusammenhängend)*
4. G *ist maximal kreisfrei (d. h: wenn man irgendeine Kante hinzufügt (zwischen bereits existierenden Knoten), dann entsteht ein Kreis.)*

Aufgabe 69. *Beweise: jeder Baum hat einen "Mittelpunkt": in jedem Baum* $G = (V, E)$ *gibt es einen Knoten* v*, so daß jede Zusammenhangskomponente von* $G \setminus v$ *höchstens* $|V|/2$ *Knoten enthält.*

In "graphentheoretischen" Bäumen gibt es kein Oben/Unten, kein Rechts/Links; also keine Wurzel, und keine Kinder-Reihenfolge.

2.1.2 Geordnete Bäume, tree domains

Oft braucht man aber genau das, d. h. eine festgelegte Zeichnung eines Baumes.

Wir benutzen Zahlenfolgen zur Adressierung von Knoten:

Definition 70. $\mathbb{N}_+^*$ *ist die Menge aller endlichen Listen von positiven Zahlen, und* $\leq$ *ist die Präfix-Halbordnung darauf:*

$$u \leq w :\iff \exists v : u \cdot v = w$$

Aufgabe 71. $(\mathbb{N}_+^*, \leq)$ *ist tatsächlich eine Halbordnung (aber keine totale Ordnung), und sogar ein unterer Halbverband (d. h. es existieren die Infima).*

Die Zahlenfolge $(x_1, x_2, \ldots)$ bezeichnet die Position, die wir erreichen, wenn wir von der Wurzel erst in das Kind Nummer x_1, dann von dort in das Kind Nummer x_2, usw. gehen.

Definition 72. *Eine Teilmenge* $M \subseteq \mathbb{N}_+^*$ *heißt* Baum-Bereich (tree domain), *wenn*

- *M ist eine präfix-abgeschlossene Teilmenge:* $\forall y : y \in M \Rightarrow \forall x \leq y : x \in M$.
- *M enthält keine Lücken:* $\forall w \in \mathbb{N}_+^*, l \in \mathbb{N}_+ : wl \in M \to \forall 1 \leq l' < l : wl' \in M$.

Definition 73. *Ein* angeordneter *(markierter) Baum ist eine partielle Abbildung* $b : \mathbb{N}_+^* \to \Sigma$, *wobei* $\mathrm{dom}(b)$ *ein Baum-Bereich ist.*

Bei den so definierten Bäumen können wir die Zeichen aus Σ beliebig auf die Knoten des Baumes schreiben.

2.1.3 Terme

Es ist aber oft so, daß aus der Knotenmarkierung bereits der "Typ" des Knotens (d. h. die Anzahl der Kinder) folgt.

Definition 74. *Eine* Signatur Σ *ist eine Folge* $(\Sigma_0, \Sigma_1, \ldots)$ *von Zeichenmengen, wobei* Σ_i *die Zeichen der Stelligkeit* i *enthält.*

Definition 75. *Zu gegebener Signatur* Σ *definieren wir die Menge* $\mathrm{Term}(\Sigma)$ *der* Terme *über* Σ *durch*

- *Wenn* $f \in \Sigma_i$, *und* $t_1, \ldots, t_i \in \mathrm{Term}(\Sigma)$, *dann* $f(t_1, \ldots, t_i) \in \mathrm{Term}(\Sigma)$.

Aufgabe 76. *Wo ist der Induktions-Anfang?*

Definition 77. *Zu jedem Term* t *gibt es einen angeordneten markierten Baum* b_t. *Die Menge der* Positionen $\mathrm{dom}(b_t)$ *nennen wir* $\mathrm{Pos}(t)$.

2.2 Lokale Baumsprachen

Völlig analog zu Bildsprachen können wir Sprachen durch lokale Überdeckungen und buchstabenweise Abbildungen definieren. Die Kacheln sind Teilbäume (nicht Teil-Terme, sondern "mittendrin", deswegen müssen wir zur Formulierung die Signatur erweitern).

Definition 78. *Eine* Baum-Kachel *zur Signatur* Σ *ist ein Baum der Signatur* $\Sigma' = \Sigma \cup \{\bot\}$*, mit einem neuen nullstelligen Symbol* $\bot$.

Definition 79. *Eine Baum-Kachel* K erscheint *im Baum* t *an der Position* p*, wenn alle Kachel-Symbole, die nicht* $\bot$ *sind, relativ zu* p *korrekt vorkommen:*

$$\forall q \in \mathrm{Pos}(K) : K[q] \neq \bot \to t[pq] = K[q].$$

Definition 80. *Ein Tripel* (N, V, W) *von Mengen von (notwendige, verbotene und Wurzel-Kacheln) definiert eine Sprache durch*

$$L(N, V, W) := \left\{ t : \begin{array}{ll} & \forall K \in N : \exists p \in \mathrm{Pos}(t) : K \textit{ erscheint in } t \textit{ an Position } p \\ \wedge & \forall K \in V : \neg\exists p \in \mathrm{Pos}(t) : K \textit{ erscheint in } t \textit{ an Position } p \\ \wedge & \exists K \in W : K \textit{ erscheint in } t \textit{ an Position } \epsilon \end{array} \right\}$$

Jede so definierbare Baumsprache heißt lokal.

Aufgabe 81. *Beweise, daß das Komplement einer lokalen Sprache wieder lokal ist. Vorsicht, das ist nicht so trivial, wie es scheint. Das Gegenteil von "jedes* $K \in N$ *ist notwendig" heißt* nicht *"jedes* $K \in N$ *ist verboten".*

2.3 Reguläre Baumsprachen

Wie bei Bildsprachen erhalten wir reguläre Sprachen als Bilder von lokalen Sprachen unter buchstabenweisen Abbildungen.

Definition 82. *Eine Projektion* $p : \Sigma \to \Gamma$ *ist eine Abbildung zwischen Signaturen, die die Stelligkeit der Symbole erhält (also* $p : \Sigma_i \to \Gamma_i$*).*

Definition 83. *Eine Baum-Sprache* $L \in \mathrm{Term}(\Sigma)$ *heißt regulär, wenn eine lokale Sprache* $L' \in \mathrm{Term}(\Gamma)$ *sowie eine Projektion* $p : \Gamma \to \Sigma$ *existieren, so daß* $L = p(L')$.

Bemerkung. In der Vorlesung hatte ich mit lokalen Sprachen ohne Wurzel-Kacheln begonnen. Wir hatten dann gesehen (Danke, Jörg), daß wir Wurzel-Kacheln durch eine weitere Projektion simulieren können, und somit den gleichen Begriff "reguläre Sprache" erhalten.

Wegen besserer Übereinstimmung mit den Standard-Bezeichnungen bei lokalen Wortsprachen (dort gibt es "Rechts- und Links-Kacheln") habe ich nun doch obige Formulierung gewählt. (Für den Rest der Vorlesung ist das alles nicht so bedeutsam.)

2.4 Reguläre Baum-Grammatiken

Wir geben eine andere Darstellung für reguläre Sprachen, indem wir die benutzten Projektionen und lokalen Kacheln einschränken. Wir benutzen nur:

- Projektionen $(q, x) \mapsto x$, wobei $x \in \Sigma, q \in Q$,
- (erlaubte) Kacheln der Form $(q, f)[(q_1, *), \ldots, (q_n, *)]$,

und fassen Q auf als Nichtterminale einer Grammatik:

Definition 84. *Eine reguläre Baum-Grammatik G ist ein Tupel (Σ, Q, S, R) mit Signatur Σ, Nichtterminal-Menge Q, Startsymbolmenge $S \subseteq Q$, und Regelmenge R. Dabei haben die Regeln in R die Form $q \to f(q_1, \ldots, q_n)$ mit $f \in \Sigma_n$ und $q, q_1, \ldots, q_n \in Q$.*

Ein Baum $t \in \mathrm{Term}(\Sigma)$ gehört genau dann zu $L(G)$, wenn es ein $s \in S$ sowie eine Ableitung $s \to_R^* t$ gibt.

Satz 85. *Eine Sprache ist genau dann regulär (im Sinne von Definition 83), wenn sie durch eine reguläre Grammatik erzeugt wird.*

Proof. (Teil 1) Zu einer gegebenen Grammatik G müssen wir die passenden Kachel-Mengen konstruieren. Die Wurzel-Kacheln sind $\{(s, \perp) : s \in S\}$, es gibt keine notwendigen Kacheln, und verboten sind alle Kacheln $(q, f)[(q_1, \perp), \ldots, (q_n, \perp)]$, für die es *keine* Regel $q \to f(q_1, \ldots, q_n)$ in R gibt.

(Teil 2) Umgekehrt müssen wir zu einer lokalen Sprache $L = L(N, V, W)$ und einer Projektion p eine passende Grammatik G für $p(L)$ angeben.

Dazu zeigen wir (Übungsaufgaben)

- man kann L auch als Projektion einer lokalen Sprache mit Kacheln der oben angegebenen Form $(q, f)[(q_1, *), \ldots, (q_n, *)]$ darstellen
- man kann jede lokale Sprache $L(N, V, W)$ mit Kacheln dieser Form auch als Projektion einer lokalen Sprache $L(\emptyset, V', W')$ (mit Kacheln der gleichen Form) darstellen (dort gibt es also keine notwendigen Kacheln mehr)
- aus dieser Form konstruiert man (wie im Teil 1) die erlaubten Kacheln (die "erlaubten" sind dann alle nicht verbotenen).

$\square$

Aufgabe 86. *Finde Grammatik für Terme in Signatur $(\Sigma_0 = \{0, 1\}, \Sigma_2 = \{\vee, \wedge\}$ mit "Wahrheitswert 1".*

Weitere Aufgaben. Siehe auch RX (online Grammatik-Basteln) http://www.informatik.uni-leipzig.de/~joe/rx/

2.5 Bottom-Up-Baumautomaten

Die Grammatik-Regeln $q \to f(q_1, \ldots)$ können wir als Top-Down-Automaten auffassen. (Beginnt in Wurzel, und vervielfacht sich selbst beim Laufen "nach unten".)

Wir drehen die Regel, erhalten $f(q_1, \ldots) \to q$, das ist ein Bottom-Up-Automat. (beginnt mit einem Exemplar in jedem Blatt, diese verschmelzen beim Laufen "nach oben".)

Definition 87. *Ein nichtdeterministische Bottom-Up-Baum-Automat ist* $A = (\Sigma, Q, \delta, F)$ *mit* $\Sigma =$ *Signatur,* $Q =$ *Zustandsmenge,* $F \subseteq Q$ *akzeptierende Endzustände,* $\delta =$ *Menge von Regeln der Form* $f(q_1, \ldots) \to q$.

Definition 88. *Ein Baum* $t \in \mathrm{Term}(\Sigma)$ *wird von* A *akzeptiert, falls es eine Ableitung* $t \to_\delta^* f \in F$ *gibt.*

Die Menge aller akzeptierten Terme heißt $L(A)$.

Eine Baumsprache L *heißt* regulär*, wenn es einen nichtdeterministischen Bottom-Up-Baum-Automaten* A *gibt, so daß* $L = L(A)$.

Die Menge aller regulären Baumsprachen heißt REG.

2.6 Beispiele für reguläre Baumsprachen

2.6.1 Spiele

Aufgabe 89. *Finde eine reguläre Grammatik für:*

- *Signatur:* I *nullstellig,* @ *zweistellig.*
- *Sprache: Alle Bäume mit einer geraden Anzahl von* @.

Bemerkung: das sind genau die Verlustpositionen im Term-Ersetzungs-Spiel mit der Regel $@(I, x) \to x$.

Siehe CGI-Skript http://theopc.informatik.uni-leipzig.de/~joe/trs/.

Vorsicht: längst nicht alle Verlustmengen von Termersetzungsspielen sind regulär. (Welche doch? Diplomarbeit!)

Siehe Paper http://www.informatik.uni-leipzig.de/~joe/projekte/trs-games/rta02/.

2.6.2 Randsprachen

Definition 90. *Für einen Baum* $t \in \mathrm{Term}(\Sigma)$ *ist das* Randwort $\mathrm{rand}(t) \in \Sigma_0^*$ *die Folge der Blattsymbole (von links nach rechts gelesen).*

Für eine Baumsprache $L \subseteq \mathrm{Term}(\Sigma)$ *ist* $\mathrm{rand}(L) := \{\mathrm{rand}(t) : t \in L\}$.

Aufgabe 91. *Finde eine reguläre Grammatik für:*

- *Signatur:* $\{a, b\}$ *nullstellig,* @ *zweistellig.*
- *Sprache:* $\{t : \mathrm{rand}(t) \in a^* b a^*\}$.

Aufgabe 92. *Finde eine reguläre Baumsprache* L *mit* $\mathrm{rand}(L) \notin \mathrm{REG}$.

Wir können Bäume einer regulären Sprache auffassen als Ableitungsbäume einer kontextfreien Grammatik:

Satz 93. *Für jede Baumsprache* L *gilt:* $L \in \mathrm{REG} \to \mathrm{rand}(L) \in \mathrm{CF}$.

Aufgabe 94. *Beweise. — Die Umkehrung gilt nicht. Finde ein Beispiel!*

Allerdings gilt: zu jeder Wortsprache $L' \in \mathrm{CF}$ *gibt es eine Baumsprache* $L \in \mathrm{REG}$ *mit* $L' = \mathrm{rand}(L)$.

2.6.3 Mehrheitssprachen

Definition 95. *Für jedes* $n \in \mathbb{N}$ *definieren wir* $\mathrm{MAJORITY}(n)$ *durch*

- *Signatur:* $\{0, 1\}$ *nullstellig,* M *: n-stellig.*
- *Bewertungsfunktion:*

  $$\mathrm{wert}(0) = 0, \mathrm{wert}(1) = 1, \mathrm{wert}(M(t_1, \ldots, t_n)) = \text{ wenn } |\{k : \mathrm{wert}(t_k) = 1\}| \geq n/2$$

- *Sprache:* $\mathrm{MAJORITY}(n) = \{t : \mathrm{wert}(t) = 1\}$.

Satz 96. *Jedes* $\mathrm{MAJORITY}(n)$ *ist regulär.*

Aufgabe 97. *Folgt aus* $t \in \mathrm{MAJORITY}(3)$ *notwendig, daß* $|t|_1 > |t|_0$ *(die Anzahl der 1 ist größer als die Anzahl der 0 in t)? Finde Schranken für das Anzahlverhältnis.*

2.7 Das Pumping-Lemma für reguläre Baumsprachen

Idee: wenn der Baum $t \in L(A)$ zu groß ist, dann gibt es einen Pfad länger als $|Q|$, auf dem ein Zustand doppelt benutzt wird, das entsprechende Teilstück können wir ab- und aufpumpen. Teilstücke sind Kontexte, d. h. Terme "mit Loch".

Definition 98. *Ein* Kontext *in einer Signatur* Σ *ist ein Term* $C \in \mathrm{Term}(\Sigma \cup \bullet)$, *wobei* $\bullet$ *ein neues nullstelliges Symbol* $\notin \Sigma$ *ist, das in* C *genau einmal vorkommt:* $|C|_\bullet = 1$.

Die Menge aller Kontexte in Σ *heißt* $\mathrm{Con}(\Sigma)$.

Definition 99. *Für* $C \in \mathrm{Con}(\Sigma), t \in \mathrm{Term}(\Sigma)$ *bezeichnet* $C[t] \in \mathrm{Term}(\Sigma)$ *den Term, den man erhält, wenn man in* C *das* $\bullet$ *durch* t *ersetzt.*

Mehrfache Einsetzung definieren wir durch

$$C^0[t] = t, C^{n+1}[t] = C[C^n[t]].$$

Definition 100. *Die Höhe* $\mathrm{height}(t)$ *eines Baumes* t *ist* $\max\{|p| : p \in \mathrm{Pos}(t)\}$.

Satz 101 (Pumping-Lemma (Schleifensatz)). *Wenn* $L \in \mathrm{REG}$, *dann existiert eine Zahl* n, *so daß für alle* $t \in L$ *gilt: Wenn* $\mathrm{height}(t) \geq n$, *dann existieren* $A, C \in \mathrm{Con}(\Sigma)$ *und* $e \in \mathrm{Term}(\Sigma)$, *so daß*

1. $A[C[e]] = t$
2. $C \neq \bullet$
3. $\forall k \geq 0 : A[C^k[e]] \in L$.

Proof. Wenn $L \in \mathrm{REG}$, dann gibt es einen Automaten A für L. Wir setzen $n = |Q(A)| + 1$.

Für jedes $t \in L$ gibt es einen akzeptierenden Lauf von A auf t, den wir uns als Markierung (der Knoten von t mit Zuständen aus Q) vorstellen. Weil der Baum hoch genug ist, enthält er einen Pfad länger als $|Q|$, auf diesem kommt dann (wenigstens) ein Zustand doppelt vor, sagen wir an Positionen p_1, p_2. Dann ist A der Kontext $t[p_1 := \bullet]$

(d. h. Baum t, aber an p_1 ein $\bullet$), C ist der Kontext $t[p_2 := \bullet]|_{p_1}$ (d. h. der Teilbaum ab p_1, mit $\bullet$ bei p_2), und $e = t|_{p_2}$.

Der Rest folgt leicht. □

Beispiel 102. *Signatur:* $\Sigma_0 = \{0, 1\}, \Sigma_2 = \{@\}$.

Die Sprache $\{t \in \mathrm{Term}(\Sigma) : |t|_1 = |t|_0\}$ *ist nicht regulär.*

Proof. Falls doch, dann setzen wir n als die Zahl aus dem Schleifensatz und betrachten $t_n = @(L, R)$, wobei L (bzw. R) ein vollständiger binärer Baum der Höhe n mit allen Blättern 0 (bzw. 1) ist. □

2.8 Operationen für Baumautomaten

Satz 103. *Für Automaten A^1, A^2 gibt es einen Automaten A mit*

$$L(A) = L(A^1) \cap L(A^2).$$

Proof. Kreuzprodukt-Konstruktion: $A = (\Sigma, Q^1 \times Q^2, \delta, F^1 \times F^2)$ wobei Σ die Vereinigung der beiden Signaturen ist und die Regelmenge

$$\delta = \{f((q_1^1, q_1^2), \ldots) \to (q^1, q^2) : f(q_1^1, \ldots) \to q^1 \in \delta^1 \wedge f(q_1^2, \ldots) \to q^2 \in \delta^2$$

Man kann leicht zeigen, daß $t \in L(A) \iff t \in L(A^1) \wedge t \in Ł(A^2)$. □

Satz 104. *Für Automaten A^1, A^2 gibt es einen Automaten A mit*

$$L(A) = L(A^1) \cup L(A^2).$$

Proof. auch Kreuzproduktkonstruktion, aber mit Finalzuständen

$$F = F^1 \times Q^2 \cup Q^1 \times F^2.$$

und außerdem müssen die Automaten zunächst vollständig gemacht werden. □

Aufgabe 105. *Was kann schiefgehen, wenn A^i nicht vollständig sind?*

Definition 106. *Ein Automat heißt* vollständig*, wenn es für jedes $t \in \mathrm{Term}(\Sigma)$* wenigstens ein *$q \in Q$ gibt mit $t \to_A^* q \in Q$.*

Satz 107. *Zu jedem Automaten A gibt es einen äquivalenten vollständigen Automaten A'.*

Proof. Einen neuen "Müll"-Zustand einführen, und passende Regeln dazu. □

Aufgabe 108. *Beschreibe exakt die Menge der nötigen Regeln.*

2.9 Deterministische Baumautomaten

Definition 109. *Ein Automat heißt deterministisch, wenn es für jedes $t \in \mathrm{Term}(\Sigma)$ höchstens ein $q \in Q$ gibt mit $t \to_A^* q$.*

Satz 110. *Zu jedem Automaten A gibt es einen äquivalenten deterministischen und vollständigen Automaten A'.*

Proof. Potenzmengenkonstruktion (wie für Wörter), $A' = (\Sigma, 2^Q, \delta', F')$. □

Satz 111. *Für jeden Automaten A gibt es einen Automaten A' mit $L(A') = \mathrm{Term}(\Sigma) \setminus L(A)$.*

Proof. Wir machen A deterministisch und vollständig und vertauschen dann akzeptierende mit ablehnenden Zuständen. □

Aufgabe 112. *Warum deterministisch? Warum vollständig?*

Satz 113. *Die Menge der regulären Baumsprachen ist abgeschlossen gegen die Booleschen Operationen (Vereinigung, Durchschnitt, Komplement).*

2.10 Deterministische Minimal-Automaten

Wir betrachten im folgenden nur deterministische vollständige Automaten.

Definition 114. *Die* syntaktische Kongruenz $\sim_A$ *eines (det. vollst.) Automaten A ist die Relation auf* $\mathrm{Term}(\Sigma)$ *mit* $t_1 \sim_A t_2 \iff t_1 \to_A^* q \leftarrow_A^* t_2$.

D. h. wwei Bäume sind A-kongruent, wenn sie in A zum gleichen Zustand führen.

Definition 115. *Die* syntaktische Kongruenz $\sim_L$ *einer Baumsprache* $L \subseteq \mathrm{Term}(\Sigma)$ *ist die Relation*

$$t_1 \sim_L t_2 \iff \forall C \in \mathrm{Con}(\Sigma) : C[t_1] \in L \iff C[t_2] \in L.$$

D. h. wwei Bäume sind L-kongruent, wenn sie durch keinen Kontext unterschieden werden können.

Satz 116. *Für jeden Automaten A für L gilt $\sim_A \subseteq \sim_L$.*

Proof. Wenn $t_1 \to_A^* q \leftarrow_A^* t_2$, dann gilt auch $C[t_1] \to_A^* q' \leftarrow_A^* C[t_2]$. Damit sind beide Seiten entweder beide in L (falls $q \in F$) oder beide nicht. □

Die Kongruenz irgendeines Automaten A für L ist also eine Verfeinerung der Kongruenz der Sprache L selbst. Damit erhalten wir andererseits $\sim_L$ als Vergröberung von $\sim_A$, d. h. auch, daß die $\sim_L$-Klassen Vereinigungen von $\sim_A$-Klassen sind.

Satz 117. *Die folgenden Aussagen sind für jede Sprache* $L \subseteq \mathrm{Term}(\Sigma)$ *äquivalent:*

- $L \in \mathrm{REG}$
- *Der Index von* $\sim_L$ *ist endlich*

D. h. $\sim_L$ teilt $\mathrm{Term}(\Sigma)$ in endlich viele Klassen.

Proof. Wenn $L \in \mathrm{REG}$, dann gibt es einen Automaten A mit $L = L(A)$. Wir können dazu einen äquivalenten vollst. det. Automaten A' für L konstruieren. Dann ist $\sim_L$ eine Vergröberung von $\sim_{A'}$, der Index von $\sim_L$ ist $\leq$ dem Index von $\sim_{A'}$, dieser ist aber gleich der Anzahl der Zustände von A'.

Wenn andererseits $\sim_L$ endlichen Index hat, dann können wir die Äquivalenzklassen als Zustände eines Automaten für L nehmen, der tatsächlich vollständig und deterministisch ist. □

Satz 118. *Zu jedem* $L \in \mathrm{REG}$ *gibt es (bis auf Isomorphie) genau einen minimalen vollst. det. Automaten für* L.

Proof. Das ist genau der eben konstruierte mit den Zustandsmenge $\mathrm{Term}(\Sigma)/\sim_L$. Jeder andere Automat für L ist eine Verfeinerung davon, müßte also mehr Zustände haben. □

Bemerkung: Diesen minimalen Automaten kann man tatsächlich durch Zusammenfassen von äquivalenten Zuständen irgendeines vollst. det. Automaten erhalten.

Bemerkung: Sehr sehr gern hätte man darüberhinaus eine Konstruktion eines minimalen (oder doch möglichst kleinen) *nicht*-deterministischen (Top-Down-)Automaten für L; damit z. B. Tools wie RX vernünftige Grammatiken ausgeben können. Das ist jedoch ein sehr sehr schweres Problem, für das es (bei vertretbarem Aufwand) nur Näherungslösungen gibt. (Diplomarbeit: ensprechendes Modul implementieren!)

2.11 (Homo)Morphismen in endliche Algebren

Definition 119. *Eine Σ-Algebra A ist eine Menge (Universum) U zusammen mit einer Abbildung, die jedem k-stelligen Symbol aus Σ eine Funktion $U^k \to U$ zuordnet.*

Funktion heißt natürlich *totale* Funktion.

Beispiel 120. *Die Boolesche Algebra hat $U = \{0, 1\}$ und Signatur $\Sigma_0 = \{F, T\}, \Sigma_1 = \{\neg\}, \Sigma_2 = \{\vee, \wedge\}$.*

Beispiel 121. *Die Term-Algebra* $\mathrm{Term}(\Sigma)$ *ist eine Σ-Algebra, die Funktionssymbole werden dabei "durch sich selbst" interpretiert. (Ausführlicher siehe Skripte von Dr. Hartwig zu Algebra/Semantik.)*

Satz 122. *Für jeden vollst. det. Automaten A für eine Sprache $L \subseteq \mathrm{Term}(\Sigma)$ kann man auf seiner Zustandsmenge Q eine Σ-Algebra erklären.*

Proof. Die Interpretation der Funktionssymbole wird durch die Überführungsfunktion des Automaten geliefert. □

Definition 123. *Ein (Homo)morphismus h von einer Σ-Algebra A in eine Σ-Algebra B ist eine Abbildung $h : U_A \to U_B$ mit*

$$\forall k : \forall f \in \Sigma_k : \quad h(f_A(t_1, \ldots, t_k)) = f_B(h(t_1), \ldots, h(t_n))$$

Hierbei ist f_A bzw. f_B die Interpretation des Symbols f in der Algebra A bzw. B.

Satz 124. *Eine Baumsprache $L \subseteq \mathrm{Term}(\Sigma)$ ist genau dann regulär, wenn eine endliche Σ-Algebra A, eine Teilmenge F des Universums U_A, sowie ein Morphismus $h : \mathrm{Term}(\Sigma) \to U$ existieren, so daß $L = h^{-1}(F)$.*

Proof. Wir nehmen als A irgendeinen det. vollst. Automaten für L, und als F seine akzeptierenden Zustände. □

Bemerkung: im Satz sehen wir ein allgemein anwendbares Definitionsprinzip, das det. vollst. Automaten (in beliebigen, d. h. nicht unbedingt freien Algebran) verallgemeinert: eine Menge heißt regulär, wenn sie Urbild eines Morphismus in eine endliche Algebra (der gleichen Signatur) ist.

2.12 Morphismen auf Kontexten

Dieser Abschnitt ist completely optional. Man kann auch einfach die Baum-Morphismus-Definition aus dem nächsten Abschnitt ohne Diskussion hinnehmen. (So steht es auch in Kapitel 1.4 aus [CDG+97].)

2.12.1 Morphismen von $\mathrm{Term}(\Sigma)$ nach irgendwo

Als Zielbereich U eines Morphismus $h : \mathrm{Term}(\Sigma) \to U$ können wir insbesondere auch eine (andere) Term-Algebra $U = \mathrm{Term}(\Gamma)$ wählen.

Beispiel 125. *Für* $\Sigma = \{a^0, b^0, f^2\}, \Gamma = \{Z^0, S^1\}$*, definiere*

$$h : \mathrm{Term}(\Sigma) \to \mathrm{Term}(\Gamma) : t \mapsto S^{|t|_a}(Z)$$

Wir zählen also die a in t und schreiben das Ergebnis unär hin. Die Interpretation von f_Σ ist dann die Abbildung

$$f_\Gamma : \mathrm{Term}(\Gamma)^2 \to \mathrm{Term}(\Gamma) : (S^x(Z), S^y(Z)) \mapsto S^{x+y}(Z)$$

Wir können leicht nachrechnen, daß h tatsächlich ein Morphismus ist, denn $h(f_\Sigma)(x, y)) = f_\Gamma(h(x), h(y))$.

Solch eine Abbildung h hängt jedoch wenig mit der inneren Struktur der Ziel-Algebra zusammen. Wir werden deswegen im folgenden zwischen Term-Algebren eine eingeschränkte Klasse von Morphismen betrachten.

2.12.2 Morphismen zwischen (Wort-)Monoiden

Zur Motivation betrachten wir die Situation bei Wörtern.

Dafür nehmen wir ein endliches Alphabet Σ von Buchstaben, und betrachten das Monoid $M = (\Sigma^*, \cdot, \epsilon)$. Ein Monoid-Morphismus geht nun von dort in irgendein anderes Monoid M'.

M' kann als Trägermenge auch ein Γ^* haben, aber dort gibt es viele verschiedene Monoid-Strukturen.

Beispiel 126. *Für* $\Gamma = \{c, d\}$ *ist* $(\Gamma^*, \odot, \epsilon)$ *mit*

$$u \odot v = \mathrm{sort}(uv),$$

ein Monoid, wobei sort *bezüglich der Ordnung* $c < d$ *gemeint ist. Beispiel:* $cdc \odot dc = \mathrm{sort}(cdcdc) = cccdd$.

Beispiel 127. *Für* $\Sigma = \{a, b\}$ *und* $M = (\Sigma^*, \dot{;}\epsilon)$ *erklären wir den Monoid-Morphismus* $h : M \to M'$ *durch*

$$h : a \mapsto c, b \mapsto d, h(u \cdot v) = h(u) \odot h(v).$$

Das ist ein Morphismus, aber es gilt *nicht* $h(u \cdot v) = h(u) \cdot h(v)$.

Man interessiert sich nun aber besonders für Abbildungen $f : \Sigma^* \to \Gamma^*$, die tatsächlich Monoid-Morphismen $f : (\Sigma^*, \cdot, \epsilon) \to (\Gamma^*, \cdot, \epsilon)$ sind, bei denen also die Verkettung in Σ^* in die Verkettung in Γ^* übersetzt wird.

Diese nennt man Wort-Morphismen, und sie sind bereits eindeutig durch ihren Wert auf einzelnen Buchstaben bestimmt.

2.12.3 Morphismen zwischen Kontexten

Die gleiche Situation tritt bei Abbildung von $\mathrm{Term}(\Sigma) \to \mathrm{Term}(\Gamma)$ auf: Eine Term-Algebra $\mathrm{Term}(\Gamma)$ kann auf ganz verschiedene Weisen zu einer Σ-Algebra erklärt werden, und damit gibt es auch viele, viele Homomorphismen.

Man interessiert sich aber besonders für solche, bei denen die Baum-Struktur in $\mathrm{Term}(\Sigma)$ in die Baum-Struktur von $\mathrm{Term}(\Gamma)$ übersetzt wird.

Bei Wortmengen war die "innerer Struktur" gerade die Monoid-Struktur, aber $\mathrm{Term}(\Sigma)$ ist kein Monoid.

Wo wir für Wörter

$$h(u \cdot v) = h(u) \cdot h(v)$$

schrieben, müßte für Bäume

$$h(C[t]) = h(C)[h(t)]$$

für einen Kontext C und einen Term t stehen, aber das ist zunächst nicht typ-korrekt (wir können h nicht auf einen Kontext anwenden).

Es geht aber doch, wenn h tatsächlich als eine Abbildung $h_* : \mathrm{Term}(\Sigma \cup \{x_1, \ldots, x_n\}) \to \mathrm{Term}(\Gamma \cup \{x_1, \ldots, x_n\})$ auffassen, die jeweils aus einem Σ-Term mit Variablen einen Γ-Term mit Variablen erzeugt. Eigentlich als Folge $h_0, h_1, \ldots$ von Abbildung für jede Variablenmenge $\{x_1, \ldots\}$. Insbesondere ist der Fall $n = 0$ (also gar keine Variablen) enthalten, und das ist der "eigentliche" Morphismus, an dem wir interessiert sind.

Wir nennen also jede Abbildung $h_* : \mathrm{Term}(\Sigma \cup \{x_1, \ldots, x_n\}) \to \mathrm{Term}(\Gamma \cup \{x_1, \ldots, x_n\})$ mit

$$\begin{aligned} \forall C \in \mathrm{Term}(\Sigma \cup \{x_1, \ldots, x_n\}) : \forall t_1, \ldots, t_n \in \mathrm{Term}(\Sigma) \\ h_0(C[x_1 := t_1, \ldots]) = h_n(C)[x_1 := h_0(t_1), \ldots] \end{aligned}$$

einen Baum-Morphismus von Σ nach Γ. (Hier benutzen wir links und rechts die null-stellige Variabte von h_*, und in der Mitte die n-stellige.)

Da wir Kontexte C aus Funktionssymbolen und Variablen zusammensetzen können, genügt es, Werte von h_* auf Kontexten C der Form $f(x_1, \ldots, x_n)$ anzugeben.

Damit bekommt also jedes k-stellige Funktionssymbol $f \in \Sigma_k$ einen Term $t \in \mathrm{Term}(\Gamma \cup \{x_1, \ldots, x_k\})$ zugeordnet (in dem die genannten Variablen evtl. mehrfach, oder auch gar nicht, erscheinen).

Aufgabe 128. *Zeige, daß der Morphismus aus Beispiel 125 kein Kontext-Morphismus ist.*

Proof. Es gibt keinen Kontext $C' \in \text{Term}(\{S, Z\} \cup \{x, y\})$, der dem Kontext $C = f(x, y)$ entspricht. (Es gibt ja überhaupt keine mehrstelligen Kontexte in der Ziel-Signatur.) □

Die ausführliche algebraische Behandlung von Baum-Morphismen erfordert den Begriff *Magmoid*, als Verallgemeinerung von *Monoid*, das bei Wörtern ausreicht.

2.13 Morphismen zwischen Term-Algebren

Im Speziellen können wir Baum-Morphismen von $\text{Term}(\Sigma)$ in eine andere Term-Algebra $\text{Term}(\Gamma)$ durch spezielle Term-Ersetzungs-Systeme (TRS) beschreiben.

Definition 129. *Ein TRS heißt linkslinear monadisch, falls alle Regeln von der Form $f(x_1, \ldots, x_n) \to r$ sind, wobei die x_i paarweise verschiedene Variablen sind.*

Rechts in der Regel steht ein beliebiger Term, der auch Variablen der linken Seite enthalten darf, sogar mehrfach.

Satz 130. *Jedes linkslinear monadische TRS R definiert auf folgende Weise einen Morphismus $h : \text{Term}(\Sigma) \to \text{Term}(\Gamma)$: wir betrachen das TRS*

$$R_h = \{h(l) \to r[x_1 := h(x_1), \ldots] : (l \to r) \in R\}$$

und definieren $h(t)$ als R_h-Normalform von t.

Aufgabe 131. *Zeige Existenz und Eindeutigkeit der Normalformen (d. h. Termination und Konfluenz von R_h).*

2.14 Abschluß-Eigenschaften bzgl. Morphismen

Satz 132. *Es gibt eine Sprache* $L \in \mathrm{REG}$ *und einen Morphismus* h*, so daß* $h(L) \notin \mathrm{REG}$.

Proof. Für $\Sigma = \{e^0, f^1\}$ und $\Gamma = \{e^0, g^2\}$ betrachte $h : e \mapsto e, f(x) \mapsto g(h(x), h(x))$. Es entsteht die Menge aller vollständigen Binärbaume, die nach einem einfachen Pump-Argument nicht regulär ist. □

Definition 133. *Ein Morphismus heißt* linear*, wenn das zugehörige Ersetzungssystem linear ist (d. h. auf der rechten Seite keine Variable doppelt erscheint).*

Satz 134. *Für jedes* $L \in \mathrm{REG}$ *und jeden* linearen *Morphismus* h *ist* $h(L) \in \mathrm{REG}$.

Beweis: Übung. Konstruktion eines (top-down-)Automaten.

Aufgabe 135. *Finde* $L \in \mathrm{REG}$ *und* nichtlineare *Morphismen* h*, für die trotzdem* $h(L) \in \mathrm{REG}$.

Hinweis: das geht zum Beispiel für solche L, bei denen die nichtlineare Regel nur an ganz wenigen Stellen (auf kleine Teilterme) paßt: $\Sigma = \{a^0, f^1, g^1\}$,

$$h : a \mapsto A, f(x) \mapsto F(h(x), h(x)), g(x) \mapsto G(G(h(x)))$$

und $L = g^* f f a$. Gibt es grundsätzlich andere Beispiele? (Ja.)

Satz 136. *Für jedes* $L \in \mathrm{REG}(\Gamma)$ *und* jeden *Morphismus* $h : \mathrm{Term}(\Sigma) \to \mathrm{Term}(\Gamma)$ *ist* $h^{-1}(L) \in \mathrm{REG}(\Sigma)$.

Proof. Jetzt darf h tatsächlich nichtlinear sein.

Wir nehmen irgendeinen det. vollst. Automaten $A = (\Sigma, Q, \delta, F)$ für L, und konstruieren daraus einen Automaten $A' = (\Gamma, Q, \delta', F)$. Dazu berechnen wir für jede Regel $f(x_1, \ldots) \to r$ des h definierenden TRS R_h und alle Zustände $q_1, \ldots \in Q$ den Zustand q mit $r[x_1 := q_1, \ldots] \to_A q$ und nehmen die Regel $f(q_1, \ldots) \to q$ in δ' auf. Dann kann man leicht zeigen, daß für alle $t \in \mathrm{Term}(\Sigma)$ gilt: $h(t) \in L(A) \iff t \in L(A')$. □

Aufgabe 137. *Bestimme nach diesem Verfahren* $h^{-1}(L)$ *für*

$$h : f(x) \to g(x, g(a, x))$$

und $L = L(q_0)$ *für den Automaten mit* $Q = \{q_0, q_1\}$ *und*

$$q_i \to (\textit{wenn } 2|i \textit{ dann } a \textit{ else } b) \mid \{g(q_j, q_k) : i \equiv j + k \pmod 2\}$$

Aufgabe 138. *Vergleiche a) den Begriff des Morphismus, b) die hier behandelten (Nicht-)Abschluß-Eigenschaften mit der Situation bei Wortsprachen!*

2.15 Normalformen von Ersetzungssystemen

Definition 139. *Für ein TRS R bezeichnet* $\mathrm{Nf}(R)$ *die Menge aller R-Normalformen.*

Satz 140. *Wenn R linkslinear ist, dann ist* $\mathrm{Nf}(R)$ *regulär.*

Proof. Die Menge aller R-Redexe (= Bäume, auf deren Wurzel eine R-Regel anwendbar ist), ist regulär. Man erhält dann $\mathrm{Nf}(R)$ als Komplement der Menge aller Bäume, die irgendeinen Redex enthalten. □

Aufgabe 141. *Bestimme die Menge aller Normalformen für die Ersetzungssysteme*

$$R_I = \{@(I,x) \to x\}, R_K = \{@(@(K,x),y) \to x\}$$
$$R_S = \{@(@(@(S,x),y),z) \to @(@(x,z),@(y,z))\}$$

Satz 142. *Das folgende Problem ist entscheidbar:*

- *Eingabe: ein TRS R*
- *Frage: ist* $\mathrm{Nf}(R) \in \mathrm{REG}$*?*

Es gibt (unabhängige, gleichzeitige!) Beweise von Kucherov und Tajine, Hofbauer und Huber, Gilleron und Vagvolgyi.

Bei linearen R ist die Antwort trivialerweise "Ja".

Aufgabe 143. *Beispiele für nichtlineare R mit* $\mathrm{Nf}(R) \in \mathrm{REG}$ *bzw.* $\mathrm{Nf}(R) \notin \mathrm{REG}$.

Aufgabe 144. *Ist* $\mathrm{Nf}(R_S^-)$ *für* $R_S^- = \{@(@(x,z),@(y,z)) \to @(@(@(S,x),y),z)\}$ *regulär?*

Die Normalformen $\mathrm{Nf}(R)$ sind die Terme, die keinen R-Nachfolger haben. Bei dieser Frage spielen die rechten Seiten der Ersetzungsregeln keine Rolle. Wir beschreiben damit eine Menge von Bäumen (Normalformen) durch verbotene Teilterme (Redexe).

2.16 Vorgänger/Nachfolgermengen

Viel interessanter, aber auch schwieriger zu beschreiben sind Mengen $R^{-*}(\mathrm{Nf}(R))$ = alle Mehrschritt-Vorgänger von Normalformen. (Dann brauchen wir natürlich die rechten Seiten.) Das ist auch für linkslineare Systeme nicht immer regulär, und wenn doch, dann schwer zu finden. (Meine Diss. war die Lösung dieser Aufgabe für R_S.)

Aufgabe 145. *Besitzt $n = @(X, X)$ mit $X = @(@(S, t), t), t = @(S, S)$ eine R_S-Normalform?*

Finde die Menge $R_S^{-*}(L)$ aller Mehrschritt-Vorgänger der Sprache $L \to S \mid @(S, S) \mid @(@(S, S), L)$. (Sie ist regulär.)

Diplomfrage: Sind in R_S Mehrschritt-Vorgängermengen von regulären Sprachen *immer* regulär?

Bemerkung. Es ist bis heute nicht bekannt, ob das folgende Problem entscheidbar ist:

- Eingabe: ein Morphismus h, eine Sprache $L \in \mathrm{REG}$
- Frage: ist $h(L) \in \mathrm{REG}$

Aufgabe 146. *Finde Beispiele (d. h. nichtlineare h) für beide Möglichkeiten!*

Aufgabe 147. *Zeige durch eine geeignete Reduktion, daß aus der Entscheidbarkeit von $h(L) \in \mathrm{REG}$ die Entscheidbarkeit von $\mathrm{Nf}(R) \in \mathrm{REG}$ folgt.*

Das ist Aufgabe 10 aus [CDG+97]. Auch die anderen Aufgaben (1 – 16 auf Seiten 35 ff.) sind wärmstens zu empfehlen.

2.17 Kontextfreie Baumsprachen

Das sind Regeln einer CF-Wort-Grammatik:

$$\{S \to \epsilon, S \to aSbS\}$$

Wie üblich fassen wir Wörter auf als Bäume über einer Signatur mit einstelligen Funktionssymbolen $\Sigma_1 = \{a, b\}$ und einem nullstelligen Wort-Ende-Zeichen $\Sigma_0 = \{e\}$.

Die Grammatik-Variablen sind dann selbst einstellige Symbole, und die Regelmenge lautet

$$\{S(x) \to x, S(x) \to a(S(b(S(x))))\}$$

Nun gestatten wir beliebig-stellige Variablen-Symbole und erhalten:

Definition 148. *Eine* kontextfreie Baum-Grammatik *ist ein Tupel* $G = (\Sigma, V, S, R)$, *wobei* Σ *("Terminale") und* V *("Variablen") disjunkte Signaturen sind,* $S \in V_0$*, und* R *eine Menge von Term-Ersertzungs-Regeln der Form*

$$v(x_1, \ldots, x_n) \to r$$

mit $v \in V_n$*, die* x_i *paarweise verschieden, und* $r \in \operatorname{Term}(\Sigma \cup V \cup \{x_1, \ldots, x_n\})$.

Definition 149. *Für eine CFG* G *ist* $L(G) = \{t \in \operatorname{Term}(\Sigma) : S \to_R^* t\}$.

Satz 150. *Das* membership*-Problem ist entscheidbar:*

- *Eingabe: eine CFG* G*, ein* $t \in \operatorname{Term}(\Sigma)$
- *Frage: gilt* $t \in L(G)$*?*

Bemerkung. Das ist nicht trivial! Man kann nicht einfach "alle Ableitungen für Terme, die kleiner als t sind", durchprobieren. Das sind evtl. unendlich viele (nicht Terme, aber Ableitungen), weil in der Grammatik kollabierende Regeln (Projektionen) sein könnnen (siehe oben, $S(x) \to x$). Bei Wortsprachen haben wir die CF-Grammatik deswegen zunächst Epsilon-frei gemacht. Es gibt aber *nicht zu jeder* CF-Baumgrammatik eine äquivalente ohne Projektionen.

Satz 151. *Das* emptiness*-Problem ist entscheidbar:*

- *Eingabe: eine CFG* G
- *Frage: ist* $\emptyset = L(G)$*?*

Wiederhole den entsprechenden Beweis für Wortsprachen (benutzt erreichbare und produkive Variablen). Ist er hier anwendbar?

Aufgabe 152. *Sind CF-Baumsprachen sind abgeschlossen unter Morphismen? (Ja.) Unter inversen Morphismen?*

Satz 153. *CF-Baumsprachen sind abgeschlossen unter Schnitt mit regulären Sprachen.*

Again: wiederhole die entsprechende Aussage für Wortsprachen.

Aufgabe 154. *Zeige, daß aus 153 und 151 die Behauptung 150 folgt.*

Es ist bis heute offen, ob das *finiteness*-Problem entscheidbar ist.

- Eingabe: eine CFG G
- Frage: ist $L(G)$ endlich?

Und again: wiederhole die entsprechende Aussage für Wortsprachen. (Hinweis: benutzt Pumping-Lemma). Ist die Beweisidee übertragbar (Hinweis: Pumping-Lemma benutzt Chomsky-Normalform)?

In der Literatur findet man den Begriff *indizierte Grammatik* für Wortsprachen. Die dadurch erzeugten “indizierten Sprachen” sind genau die Rand-Sprachen von CF-Baumsprachen. Das erklärt auch ihre (größtenteils angenehmen) Abschluß- und Entscheidbarkeits-Eigenschaften, obwohl die Sprachfamilie “deutlich oberhalb” von CF-Wortsprachen liegt (aber noch unterhalb der kontextsensitiven).

2.18 Ableitungsbäume von Grund-Term-Ersetzungs-Systemen

(Dieser Abschnitt komplett aus [Eng99]. Hier nur die Definitionen und Sätze. Für Beispiele, Beweise, Bilder, Erläuterungen siehe Original.)

Definition 155. *Ein Grund-Term-Ersetzungs-System (GTRS) ist ein TRS, dessen Regeln* keine *Variablen enthalten.*

Für ein GTRS R über der Signatur Σ kodieren wir Ableitungen $t_0 \to_R t_1 \to_R \ldots \to_R t_n$ (also Folgen von Bäumen) in einen *einzigen* Baum $t \in \mathrm{Term}(\Sigma_\#)$, wobei $\Sigma_\# = \Sigma \cup \#$ für ein neues zweistelliges Symbol $\#$.

Die Idee ist, daß jeder Teilbaum t' von t mit Wurzel $\#$ bedeutet: "hier stand vorher links(t'), und nachher rechts(t')", wobei wir die zwei Morphismen benutzen:

$$\begin{array}{ll} & \mathrm{links}(\#(x,y)) = \mathrm{links}(x) \\ \text{für } f \in \Sigma & \mathrm{links}(f(x_1,\ldots)) = f(\mathrm{links}(x_1),\ldots) \\ & \mathrm{rechts}(\#(x,y)) = \mathrm{rechts}(y) \\ \text{für } f \in \Sigma & \mathrm{rechts}(f(x_1,\ldots)) = f(\mathrm{rechts}(x_1),\ldots) \end{array}$$

Definition 156. *Ein Ableitungsbaum für das GTRS R ist ein Baum $t \in \mathrm{Term}(\Sigma_\#)$, so daß für jeden Teilbaum $t' = \#(x,y)$ von t gilt:* $\mathrm{rechts}(x) \to \mathrm{links}(y) \in R$.

Satz 157. *Die Menge D_R aller Ableitungsbäume für ein GTRS R ist eine reguläre Baumsprache.*

Satz 158. *Für jedes GTRS R und jedes $L \in \mathrm{REG}$ gilt: $R^*(L) \in \mathrm{REG}$.*

Proof. $R^*(L) = \mathrm{rechts}(\mathrm{links}^-(L) \cap D_R)$, und Abschluß-Eigenschaften von regulären Sprachen (gegen inverse Morphismen, Schnitt, und lineare Morphismen). □

Der Beweis funktioniert sogar für allgemeinere GTRS:

Definition 159. *Ein* extended *GTRS P ist eine endliche Menge von Paaren von regulären Baumsprachen $\{(L_1, R_1), \ldots, (L_n, R_n)\}$.*

Das zugehörige (im allg. unendliche) GTRS ist $L_1 \times R_1 \cup \ldots \cup L_n \times R_n$.

Definition 160. *Für eine Relation R auf $\mathrm{Term}(\Sigma)$ bezeichnet $R^{||}$ die Relation*

$$R^{||}(l,r) \iff \exists C \in \mathrm{Con}_k(\Sigma), \exists l_1, r_1, \ldots : l = C[l_1,\ldots] \wedge r = C[r_1,\ldots] \wedge R(l_1,r_1) \wedge \ldots$$

Definition 161. *Eine Relation R auf $\mathrm{Term}(\Sigma)$ heißt* ground tree transducer *(GTT), falls ein extended GTRS P existiert, so daß $R = P^{||}$.*

Satz 162. *Wenn R ein GTT ist, dann ist auch R^* ein GTT.*

Satz 163. *Wenn R ein GTT ist, und $L \in \mathrm{REG}$, dann sind $R(L) \in \mathrm{REG}$ und $R^*(L) \in \mathrm{REG}$.*

Zu GTTs siehe auch [CDG+97]. Dort werden die letzten beiden Sätze durch direkte Automaten-Konstruktionen bewiesen. Die hier gezeigte Methode der Ableitungsbäume liefert davon unabhängige Beweise.

2.19 Tree Walking Automata

(Dieser Abschnitt im wesentlichen nach [EH99] und [NS00]. Siehe auch Zusammenfassung in [Wal01].)

Mit regulären Grammatiken haben wir ein Verfahren zum Beschreiben von Baumsprachen, das im wesentlichen top-down und parallel arbeitet. Demgegenüber sind die endlichen Baum-Automaten ein bottom-up und parallel arbeitendes Modell.

Beides ist, wie wir gesehen haben, "eigentlich" algebraisch, und entspricht nicht der klassischen Vorstellung vom Automaten, die nämlich eine *sequentielle* ist (vergleiche Automaten auf Wörtern oder Bildern).

Bauen wir also einen sequentiellen Baum-Automaten. Er läuft auf dem Baum herum und heißt deswegen TWA (tree walking automaton). Wir beschreiben ihn in einer geeigneten Assemblersprache.

2.19.1 Eine Assemblersprache für endliche Automaten

Ein Programm eines TWA ist eine Liste von Befehlen `<stmt>` die eventuell Labels tragen. Ein Label muß `start` heißen.

- Wertzuweisung (Konstante) `<loc> := <val>;`
- dabei ist ein Wert `<val>` eine Konstante (Zeichenfolge) oder ein Zugriff: `<val> ::== Const <string> | Ref <string>`
- Halte akzeptierend: `Accept;` oder ablehnend: `Reject;`
- unbedingter Sprung: `Goto <label>;`
- bedingter Sprung: `If (<val>,<val>) <label>;`

Bemerkung: bis hierher ist alles noch völlig generisch: wir können bisher nur zwischen Programmplätzen hin- und herspringen sowie konstante Werte zuweisen und vergleichen.

Das ist ein endlicher Automat, und zwar ohne jeden externen Speicher. Die Zustandsmenge ist wirklich endlich, denn sie enthält den Programmzähler und die Belegung der Speicherstellen. Dort kann man aber nur endlich viele Dinge speichern. Es gibt mit Absicht keine Rechenbefehle (wie Addition oder Verkettung von Zeichenfolgen).

Das bisher beschriebene Steuerwerk soll nun mit der Außenwelt kommunizieren. Es kann fragen (d. h. einen externen Wert lesen) und Aktionen ausführen (lassen).

- `<val> ::== .. | Ask ( <question> )`
- `<stmt> ::== .. | Do ( <action> )`

Soweit ist immer noch alles generisch.

2.19.2 Spezielle TWA-Befehle

Ein TWA entsteht, wenn wir die erlaubten Fragen und Aktionen festlegen:

- `<question> ::==`
  - `Read` lies Zeichen auf aktueller Position?
  - `Exists <dir>` existiert Position in Richtung,
    wobei `<dir> ::= Up | Down <int>`, (Kind-Nummern $0, 1, \ldots$)
- `<action> ::==`
  - `Go <dir>` gehe in Richtung ...

Wie man sich nun leicht überlegen kann, ist dieser Befehlssatz noch nicht ausreichend: die so definierten Automaten können noch nicht mal einen Baum komplett durchlaufen (z. B. um in einem binären Baum zu prüfen, daß genau ein Blatt die Markierung b trägt).

Das ermöglichen wir durch einen neuen Test:

- `<question> ::==`
  - `Whoami` Wer bin ich? Liefert Antwort $0, 1, \ldots$ oder nichts.

Ich bin dabei, das für das Autotool zu implementieren, siehe http://theopc.informatik.uni-leipzig.de/~challenger/source/twodim/. Man sollte auch die Wort- und Bild-Automaten nach diesem Schema programmierbar machen. — Eine wunderschöne Übung ist es, noch eine "höhere" Sprache davorzuschalten, z. B. mit if-then-else und Schleifen und dann den entsprechenden Compiler zu schreiben. — Freiwillige vor!

Aufgabe 164. *Welche höheren Sprachelemente sind eigentlich erlaubt? Prozeduren und Funktionen auch? Es soll ja ein endlicher Automat bleiben.*

Aufgabe 165. *Programmiere damit einen TWA, der in der Signatur* $\Sigma = \{a^0, b^0, @^2\}$ *die Sprache* $L = \{t : \text{rand}(t) \in a^* b a^*\}$ *erkennt. (autotool: Subject TWAL)*

Aufgabe 166. *Konstruiere einen TWA, für die Sprache R mit der Grammatik*

$$R \to @(L, R) \mid a \mid b.$$

(autotool: Subject TWAR)

D. h. alle linken Kinder des rechten Rückgrats sind in L (vorige Aufgabe), das äußerste rechte Blatt ist egal.

2.20 TWA mit Pebbles

Die Sprache R aus Aufgabe 166 scheint nicht in TWA zu sein:

Es gibt für den Automaten keine Möglichkeit, während der Rechnung festzustellen, ob er sich auf dem rechten Rückgrat des Baumes befindet. Er müßte dazu zur Wurzel zurücklaufen, aber dann weiß er nicht mehr, welchen Teil des Rückgrats er bereits erfolgreich getestet hat.

Allerdings ist bis heute kein Beweis für $R \notin$ TWA bekannt. Wer einen findet, wird berühmt! Vielleicht gibt es eine verschärfte Pumping-Eigenschaft für TWA-Sprachen, die R nicht erfüllt?

Wenn wir den Automaten mit einem Rechenhilfsmittel ausstatten, dann kann er R leicht akzeptieren: wir geben ihm einen Stein (*pebble*), den er als Markierung auf einen Baumknoten legen darf. Später darf er dann fragen, ob der Pebble im aktuellen Knoten liegt, und ihn wieder aufheben.

Mit einem Pebble läßt sich R tatsächlich durch einen TWA akzeptieren (der Pebble markiert den Arbeitsstand auf dem rechten Rückgrat). Die Familie aller so akzeptierbaren Sprachen nennen wir pTWA(1).

Die Verallgemeinerung auf mehrere Pebble liegt nahe. Aber Vorsicht: mit zwei Pebbles kann der Automat bereits testen, ob zwei Pfade im Baum gleich tief sind! Damit kann er die Sprache $\{g(f^n(a), f^n(a)) : n \geq 0\}$ akzeptieren, die natürlich nicht regulär ist.

Deswegen ist folgende Einschränkung sinnvoll: Der Automat hat zwar mehrer Pebbles $1, 2, 3 \ldots$, aber deren Lebensdauern dürfen nicht überlappen, sondern müssen geschachtelt sein. Das heißt: er darf Pebble k nur dann benutzen (d. h. im Baum anbringen oder aus dem Baum entfernen), wenn er alle Pebbles mit größerer Nummer (noch oder wieder) in der Hand hält. (Vgl. mit Benutzung eines Stacks.)

Die Familie aller mit n Pebbles mit geschachtelter Lebensdauer akzeptierbaren Sprachen nennen wir pTWA(n), und die Vereinigung aller dieser Familien heißt pTWA.

Beachte: wenn $L \in$ pTWA, dann heißt das nicht, daß der Automat einen unbeschränkten Pebble-Vorrat hat, sondern $\exists n : L \in$ pTWA(n), d. h. es gibt eine Zahl n (unabhängig von der Baumgröße), so daß L mit n Pebbles akzeptierbar ist.

2.21 Trips on Trees

Satz 167. *(Engelfriet, 1998)* pTWA $\subseteq$ REG.

Proof. (Idee.) Wir erinnern uns daran, wie im Grundkurs zu jedem endlichen (Wort-)Automaten einen äquivalenter regulärer Ausdruck konstruiert wurde:

Man ordnet die Zustandsmenge Q des Automaten irgendwie linear, und betrachtet die Sprachen $L^h_{p,q}$, die alle Wörter $w = w_1 w_2 \dots w_n$ enthalten, für die der Automat einen Weg

$$p = p_0 \xrightarrow{w_1} p_1 \xrightarrow{w_1} \dots \xrightarrow{w_n} p_n = q$$

zurücklegt, wobei alle Zwischen-Zustände $p_1, \dots, p_{n-1}$ in der gewählten Ordnung $\leq h$ sein müssen.

Man hat $L^0_{p,q} = \{a \in \Sigma : p \xrightarrow{a} q\}$ (0 ist ein gedachter Zustand $< \min Q$, da gibt es gar keine Zwischenzustände) und verifiziert, daß

$$L^{h+1}_{p,q} = L^h_{p,q} \ \cup \ L^h_{p,h+1} \cdot (L^h_{h+1,h+1})^* \cdot L^h_{h+1,q}.$$

Die vom Automaten akzeptierte Sprache ist dann die Vereinigung aller $L^{\max Q}_{p,q}$ für alle Startzustände p und Endzustände q.

Genau diese Konstruktion führen wir für TWA aus. Wir betrachten dabei o.B.d.A. nur Automaten, die in der Wurzel beginnen (das ist sowieso klar) und auch in der Wurzel akzeptieren (das kann man leicht erreichen).

Damit gilt für jeden Teilbaum: falls ihn der Automat irgendwann betritt, verläßt er ihn auch wieder. Die Sprache $L^h_{p,q}$ bedeutet dann die Menge aller Bäume, die der Automat in Zustand p betritt, in Zustand q verläßt, und unterwegs nur Zustände $\leq h$ benutzt.

Völlig analog zum klassischen Beweis können wir reguläre Ausdrücke (d. h. hier Grammatiken) für $L^h_{p,q}$ kontruieren. Das beweist TWA $\subseteq$ REG.

Die Pebbles behandeln wir nach dem gleichen Schema: wir betrachten $L^{h,k}_{p,q}$ wobei $L^h_{p,q}$ wie bisher, und der zusätzliche Parameter k bedeutet, daß während der Rechnung nur Pebbles mit Nummern $\geq k$ benutzt wurden. Das läßt sich alles problemlos hinschreiben, und damit folgt die Behauptung. $\square$

Vermutung 168. TWA = pTWA(0) $\subset$ pTWA(1) $\subset \dots \subset$ pTWA $\subset$ REG.

Kommentar. Daß die Inklusion gelten, ist offensichtlich bzw. wurde eben gezeigt. Die Vermutung besteht darin, daß tatsächlich alle Inklusionen echt sind.

2.22 Mehrheitssprachen

Ein Kandidat für eine Sprache in REG$\setminus$pTWA sind T_k (für $k \geq 3$) über der Signatur mit $\Sigma_0 = \{0, 1\}$ und einem k-stelligen Symbol m.

Wir definieren den Wert eines Baums so:

$$\begin{aligned} &\mathrm{wert}(0) = 0, \mathrm{wert}(1) = 1 \\ &\mathrm{wert}(m(t_1, \dots, t_k)) = \text{ wenn (wenigstens die Hälfte der } \mathrm{wert}(t_i) = 1) \text{ dann 1 sonst 0} \end{aligned}$$

und setzen $M_k = \{t : \mathrm{wert}(t) = 1\}$.

Natürlich gilt:

Aufgabe 169. $\forall k : M_k \in \mathrm{REG}$.

Nach unserer Definition ist $\mathrm{wert}(m(x, y)) = x \vee y$, also ist $M_2 = \{t : \mathrm{rand}(t) \in (0 + 1)^*1(0 + 1)^*\}$, und deswegen verwandt mit der Sprache A, und deswegen in TWA.

Aber für mehrstellige Symbole reichen wohl beschränkt viele Pebbles nicht:

Vermutung 170. *(Neven, Schwentick, 2000)* $M_3 \notin$ pTWA

Auch hier gibt es zwar Indizien, aber keinen Beweis. Diplomarbeit!

Es ist zunächst erstaunlich, daß ein TWA (ohne Pebbles) sogar binäre Bäume mit $\vee$ und $\wedge$ auswerten kann:

Aufgabe 171. *In der Signatur mit* $\Sigma_0 = \{0, 1\}, \Sigma_2 = \{\vee, \wedge\}$*, zeige daß* $T = \{t : \mathrm{wert}(t) = 1\} \in$ TWA*, wobei die Auswertung nach den üblichen aussagenlogischen Regeln ablaufen soll.*

Autotool: probiere, TWAs zu schreiben für M_2 (subject TWAM2), M_3 (subject TWAM3) und T (subject TWAT). Beachte: das sollte nicht alles funktionieren. Falls doch, ist es wohl eher ein Bug im Tool.

pTWA habe ich nicht implementiert. Wer Lust hat …

Kapitel 3

Automaten und Logik

Wir wollen schließlich einen Zusammenhang zwischen Automatentheorie und Logik herstellen: Modellmengen von Formeln sind (je nach Signatur) Mengen von z. B. Wörtern oder Bäumen, und diese können wir (je nach Ausdrucksstärke der Logik) durch endliche Automaten beschreiben. Da wir mit den Automaten effektiv rechnen können, gewinnen wir daraus (für bestimmte Logiken) Entscheidungsverfahren (für Erfüllbarkeit, Allgemeingültigkeit).

3.1 Lokale Logik auf Bäumen

Wir fixieren eine endliche Menge von einstelligen Prädikat-Symbolen $a(), b(), \ldots$ sowie eine endliche Menge von zweistelligen Prädikatsymbolen $<_1, <_2, \ldots, <_k$; und das Symbol $=$ (das wir tatsächlich als Gleichheit interpretieren).

Die zu dieser Signatur passenden Strukturen können wir uns so vorstellen: Das Universum besteht aus den Adressen der Knoten eines Baumes, $a(p)$ ist wahr, wenn an Position p ein Zeichen a steht; und $p <_k q$ ist wahr, wenn q das k-te Kind von Position p ist.

Wir fixieren im folgenden immer eine maximale Stelligkeit k. Dann definieren wir die Abkürzungen

$$\begin{aligned} p < q &:\equiv p <_1 q \vee \ldots \vee p <_k q \\ \mathrm{blatt}(p) &:\equiv \neg\exists q : p < q \\ \mathrm{wurzel}(q) &:\equiv \neg\exists p : p < q \end{aligned}$$

Betrachten wir die Formeln

$$\begin{aligned} F_A &\equiv \forall p : \mathrm{blatt}(p) \to a(p) \\ F_L &\equiv \exists p : \mathrm{blatt}(p) \wedge b(p) \wedge \forall q : \mathrm{blatt}(q) \wedge (p \neq q) \to a(q) \end{aligned}$$

Es gilt $\mathrm{Mod}(F_L) = L$, die Modelle von F_L sind genau die Bäume aus der Sprache L.

Die Menge aller prädikatenlogischen Formeln dieser Signatur nennen wir LFO (lokale First-Order-Logik). Was für F_L klappte, funktioniert immer:

Satz 172. *(Engelfriet) Wenn für eine Baumsprache* $L = \mathrm{Mod}(F)$ *für* $F \in \mathrm{LFO}$, *dann* $L \in \mathrm{TWA}$.

Proof. (Idee.) Variablen, die in der Formel vorkommen, hängen entweder voneinander ab oder nicht. Tun sie es nicht, dann können wir die entsprechenden Teilformeln unabhängig voneinander prüfen. Für jeden Quantor machen wir dabei eine komplette Tiefensuche über den Baum.

Gibt es jedoch eine Abhängikeit, d. h. eine Kette $x = p_0 < p_1 < \ldots < p_n = y$, dann haben x und y voneinander höchstens den Abstand n. Es reicht also in diesem Falle, x über den gesamten Baum zu schicken, und y nur über lokale n-Umgebungen von x. Solche "lokalen Ausflüge" kann der TWA jedoch mit seiner endlichen Zustandsmenge durchführen (schlimmstenfalls braucht er Stelligkeit^n Zustände für jeden Ausflug). □

Aufgabe 173. *Was tun wir mit Prädikaten* $x \neq y$*?*

Weil für früher schon TWA $\subseteq$ REG hatten, und alle Übersetzungen LFO $\to$ TWA $\to$ REG effektiv waren, haben wir damit ein Entscheidungsverfahren für Erfüllbarkeit von LFO-Formeln (dieser Signatur): die Formel ist erfüllbar, wenn die Modellmenge nicht leer ist, d. h. wenn die vom Automaten akzeptierte Sprache nicht leer ist.

3.2 Globale Logik erster Ordnung

In LFO können wir nicht ausdrücken, daß p irgendwo oberhalb von p liegt, denn das ist eine "globale" Eigenschaft.

Deswegen können wir ein neues Prädikatsymbol $\ll$ hinzunehmen mit der Bedeutung

$$p \ll q \cong \exists p_1, \ldots, p_{n-1} : p = p_0 < p_1 < \ldots < p_{n-1} < p_n = q.$$

(Wir gestatten auch $n = 0$, d. h. $p = q$.) Beachte: das ist keine LFO-Formel (Pünktchen zwischen Quantoren gibt es nicht) sondern nur eine Erläuterung.

Jetzt können wir formulieren, daß q eine Position auf dem rechten Rückgrat eines Baumes ist.

Aufgabe 174. *Definiere (unter Benutzung von $\ll$) ein Prädikat* $\mathrm{rechts}(p)$*, das genau dann wahr ist, wenn sich p auf dem rechten Rückgrat der Wurzel (oder in der Wurzel selbst) befindet.*

Lösung:

$$\mathrm{rechts}(q) \equiv \forall p : p \ll q \wedge p \neq q \rightarrow \exists r : p <_2 r \wedge r \ll q$$

Betrachten wir nun die (bezüglich s) relativierte Formel

$$\begin{aligned} F_L^s \quad \equiv \quad & \exists p : \boxed{s \ll p \wedge} \mathrm{blatt}(p) \wedge b(p) \\ & \wedge \forall q : \boxed{s \ll p \rightarrow} \mathrm{blatt}(q) \wedge (p \neq q) \rightarrow a(q) \end{aligned}$$

Sie sagt, daß der Teilbaum mit s als Wurzel die Formel F_L erfüllen soll. Wir benutzen das so:

$$F_R \quad \equiv \quad \forall r : \mathrm{rechts}(r) \wedge \neg \mathrm{blatt}(r) \rightarrow \exists s : r <_1 s \wedge F_L^s$$

Damit gilt $\mathrm{Mod}(F_R) = R$ (die von früher bekannte Sprache). Es war ja $R \in \mathrm{pTWA}(1)$, und geht im Allgemeinen. Wir nennen die Menge aller prädikatenlogischen Formeln in obiger Signatur FO:

Satz 175. *(Engelfriet) Wenn eine Baumsprachen $L = \mathrm{Mod}(F)$ für $F \in \mathrm{FO}$, dann $L \in \mathrm{pTWA}$.*

Proof. (Idee.) Wie früher, aber für jede relativierte Variable legen wir einen Pebble auf den Startpunkt der Relativierung, und führen die Tiefensuche unterhalb davon aus. □

3.3 Monadische Logik Zweiter Ordnung

Nun ist bei Logik erster Ordnung noch lange nicht Schluß. Es ist bekannt, daß LFO $\subset$ FO $\subset$ REG, und alle Inklusionen echt sind. Wir beschäftigen uns jetzt mit der Logik MSO, für die wir REG = MSO erhalten. Das ist also "genau die richtige" für endliche Automaten.

"Zweiter Ordnung" bedeutet, daß wir prädikatenlogische Formeln schreiben dürfen, bei denen Quantifikation über "Objekte zweiter Ordnung" erlaubt ist. Was ist das? Im allgemeinen sind es alle Prädikate.

Im besonderen betrachten wir *monadische* Logik zweiter Ordnung, bei der wir nur über *einstellige* Prädikate quantifizieren dürfen.

Ein einstelliges Prädikat kann man aber als Menge auffassen (aller Objekte, für die das Prädikat wahr ist). Wir benutzen deswegen Mengen-Variablen, und bezeichnen sie mit Großbuchstaben (die first-order-Variablen (die Positionen im Baum bezeichnen) wie bisher mit Kleinbuchstaben).

Beispiel 176. *Wir können in MSO das Prädikat $\ll$ direkt ausdrücken. Wir definieren dazu ein Prädikat* $\mathrm{prefix}(X)$*, das genau dann wahr ist, wenn X eine präfix-abgeschlossene Menge ist.*

$$\begin{aligned} \mathrm{prefix}(X) &\equiv \forall s \in X : \forall r : r < s \to r \in X \\ p \ll q &\equiv \forall X : q \in X \wedge \mathrm{prefix}(X) \to p \in X \end{aligned}$$

Wir sehen, daß der lokal/global-Unterschied in MSO entfällt, da wir die globalen Aussagen durch Mengen-Variablen und lokale Aussagen umschreiben können.

Aufgabe 177. *Zeige, daß man in der gegebenen Signatur auch die Relationen $\subseteq$ (Teilmenge),* $\mathrm{leer}(X)$ *(X ist leere Menge),* $\mathrm{single}(X)$ *(X ist Einermenge) definieren kann.*

Lösungen.

$$\begin{aligned} X \subseteq Y &\equiv \forall x : x \in X \to x \in Y \\ \mathrm{leer}(X) &\equiv \neg \exists x : x \in X \end{aligned}$$

Wir schreiben die letzte Formel ohne Verwendung von first-order-Variablen so:

$$\begin{aligned} \mathrm{leer}(X) &\equiv \forall Y : Y \subseteq X \to Y = X \\ \mathrm{single}(X) &\equiv \forall Y : Y \subseteq X \to Y = X \vee \mathrm{leer}(Y) \end{aligned}$$

3.3.1 Der MSO-Modell-Begriff

Eine MSO-Formel enthält first-order- und second-order-Variablen. Die FO-Variablen bezeichnen Positionen, die MSO-Variablen Mengen von Positionen.

Eine Position ist einfach eine endliche Liste, also ein Element von $\{1, 2, \ldots, a\}^*$, wobei a die (fixierte) maximale Kinderzahl der Baumknoten ist. (D. h. die Prädikate $<_1, \ldots, <_a$ sind in der Signatur.)

Wir sehen jetzt, daß wir die einstelligen Prädikate $a(p)$ von früher gar nicht mehr brauchen, da wir $p \in A$ für eine MSO-Variable A schreiben können.

Definition 178. *Für eine MSO-Formel F mit freien FO-Variablen $x_1, \ldots, x_k$ und freien MSO-Variablen $X_1, \ldots, X_l$; und Positionen $p_1, \ldots, p_k$ sowie Mengen $P_1, \ldots, P_l$ von Positionen gilt*

$$(p_1, \ldots, p_k, P_1, \ldots, P_l) \models F,$$

wenn die Formel F bei der Interpretation von $\in$ durch "ist Element von", $p <_i q$ durch $pi = q$, x_i durch p_i und X_i durch P_i wahr ist.

Definition 179. *Die Menge aller Tupel $m = (p_1, \ldots, p_k, P_1, \ldots, P_l)$ mit $m \models F$ heißt* Modellmenge $\mathrm{Mod}(F)$ *von F.*

3.3.2 Ersetzen von FO- durch MSO-Variablen

Jetzt machen wir uns klar, daß wir auf FO-Variablen verzichten können, ohne die Ausdruckskraft der MSO einzuschränken.

Definition 180. *Formeln in MSO' benutzen nur*

- *MSO-Variablen $X_1, \ldots,$*
- *Prädikate $X \subseteq Y, \mathrm{single}(X), X <_i Y, \mathrm{wurzel}(X)$.*

In MSO' interpretieren wir $\subseteq$ wie erwartet, und $X <_i Y$ gilt genau dann, wenn X und Y Einermengen sind: $X = \{x\}, Y = \{y\}$ und $x <_i y$; und $\mathrm{wurzel}(X)$ genau dann, wenn $X = \{x\}$ und $\mathrm{wurzel}(x)$. Die dadurch erklärte Modell-Relation nennen wir $\models_2$.

Satz 181. *Zu jeder MSO-Formel F mit freien FO-Variablen $x_1, \ldots, x_k$ und freien MSO-Variablen $X_1, \ldots, X_l$ gibt es eine MSO'-Formel F' mit freien MSO-Variablen $X'_1, \ldots, X'_k, X_1, \ldots, X_l$ so daß*

$$(p_1, \ldots, p_k, P_1, \ldots, P_l) \models F$$

genau dann, wenn

$$(\{p_1\}, \ldots, \{p_k\}, P_1, \ldots, P_l) \models_2 F'.$$

Proof. Selberbasteln. Siehe vorige Übungsaufgabe. □

3.3.3 Codierung von Positionsmengen als Bäume

Definition 182. *Eine Menge M von k-Tupeln von endlichen Mengen von Positionen $\subseteq \{1, \dots, a\}^*$ heißt* MSO-definierbar*, wenn es eine MSO-Formel F mit freien MSO-Variablen $X_1, \dots, X_k$ gibt, so daß $(P_1, \dots, P_k) \in M$ genau dann gilt, wenn $(P_1, \dots, P_k) \models F$.*

Nun ist "Menge von Tupeln von Mengen von Positionen" ein etwas unhandlicher Begriff. Wir fassen ein solches Tupel als passend markierten Baum auf:

Als Alphabet wählen wir $\{0, 1, \perp\}^k$ (k ist die Länge der zu kodierenden Tupel) und alle diese Symbole sind a-stellig, bis auf $\perp^k$, das ist nullstellig (d. h. Blatt-Symbol).

Nennen wir diese Signatur Σ_k. Dann entspricht jedem Baum $t \in \operatorname{Term}(\Sigma_k)$ ein Tupel $\operatorname{val}(t) = (P_1, \dots, P_k)$ von Mengen von Positionen, und zwar besteht P_i aus allen Positionen, deren i-te Komponente eine 1 ist.

Umgekehrt können zu jedem Tupel $P = (P_1, \dots, P_k)$ von endlichen Mengen von Positionen einen passenden Baum $t = \operatorname{code}(P)$ mit $\operatorname{val}(t) = P$ konstruieren.

Warum die Unterscheidung zwischen 0 und $\perp$? Wir verwenden $\perp$ an i-ter Stelle der Position p genau dann, wenn es *kein* $p' > p$ gibt, für das $p' \in P_i$. Damit ist bei $\perp^k$ Schluß.

Aufgabe 183. *Kodiere das Mengen-Tupel* $(\{\epsilon\}, \{11, 2\}, \{121\})$.

3.3.4 MSO-Baumsprachen sind regulär

Satz 184. *Für jede MSO-definierbaren Menge M gilt* $\mathrm{code}(M) \in \mathrm{REG}$ *(d. h., ist eine reguläre Baumsprache.)*

Proof. Nach den Vorbetrachtungen ist es ausreichend, MSO'-definierbare Mengen M zu betrachten.

Wir müssen induktiv (über den Formelaufbau) zu jeder MSO'-Formel F einen Automaten A_F mit $L(A) = \mathrm{code}(\mathrm{Mod}(F))$ konstruieren.

Aufgabe 185. *Wie sieht das aus für $X \subseteq Y$, $X <_i Y$,* $\mathrm{wurzel}(X)$, $\mathrm{single}(X)$*?*

Behandeln wir den Fall der logischen Verknüpfung von Formeln. Hier ist wegen der de-Morganschen Regeln es ausreichend, $\vee$ und $\neg$ zu behandeln.

Wenn wir A für F gegeben haben, dann ergibt sich A' für $\neg F$ durch Komplementbildung (und anschließenden Schnitt mit der Menge der erlaubten Bäume, d. h. unter Beachtung der Regel für $\perp$ als Schluß-Symbole.)

Wenn wir Automaten A_1 und A_2 für Formeln F_1, F_2 mit freien MSO-Variablen $X_1, \ldots, X_k$ bzw. $Y_1, \ldots Y_l$ haben dann konstruieren wir zunächst den Automaten A_1' auf der Signatur $\{X_1, \ldots, X_k\} \cup \{Y_1, \ldots Y_l\}$. (Einige der Variablen könnten übereinstimmen, d. h. in F_1 und F_2 benutzt werden), so daß A_1' wie A_1 rechnet, und die zusätzlichen Komponenten ignoriert. (Ebenso für A_2'.) Der gewünschte Automat A für $F_1 \vee F_2$ entsteht also Vereinigung von A_1' und A_2'. Das ist konstruierbar, weil REG boolesch abgeschlossen ist.

Von den Quantoren $\forall X : F, \exists X : F$ (wir sind in MSO', also über MSO-Variablen!) genügt es wieder, $\exists X : F$ zu betrachten. Wenn X die i-te freie Variable in F ist, dann "löschen" wir im Automaten A für F einfache die i-te Komponente, d. h. der Automat A' für $\exists X : F$ rät sie. $\square$

Die Konstruktion ist effektiv ausführbar, und für Baum-Automaten A ist entscheidbar, ob $L(A) = \emptyset$

Aufgabe 186. *Gib dafür ein Entscheidungsverfahren an. Hinweis: vergleiche mit $L(G) = \emptyset$ für kontextfreie Grammatiken G.*

deswegen erhalten wir die Folgerung:

Satz 187. *Das folgende Problem ist entscheidbar:*

- *Eingabe: eine MSO-Formel F (in der Signatur mit $\in, <_1, \ldots, <_a$)*
- *Frage: hat F endliche Modelle (d. h. Mengenvariablen sind mit endlichen Mengen zu belegen)*

Wie komplex ist das gezeigte Verfahren? Beim Negieren müssen wir einen Komplement-Automaten konstruieren, und dabei kann die Zustandszahl explodieren. Das Negieren geschieht ja auch bei $\forall$-Quantoren, weil wir $\forall X : F$ als $\neg\exists X : \neg F$ behandeln. Wir rechnen also mit einer Laufzeit von $\exp(\exp(\ldots(\exp(a))))$, wobei wir so viele exp schachteln, wie Negationen oder Quantorwechsel vorkommen. (Das geht tatsächlich nicht besser, die Erfüllbarkeit solcher Formeln ist ein vollständiges Problem für eine sehr hohe Komplexitätsklasse.)

3.3.5 Reguläre Baumsprachen sind MSO-definierbar

Satz 188. *Jede reguläre Baumsprache L ist MSO-definierbar.*

(Bemerkung.) Wir müssen zunächst festlegen, was das ünerhaupt heißen soll, denn bis jetzt sind (kodierte) Modellmengen von Formeln Bäume in der speziellen Signatur $\{0, 1, \perp\}^k$.

Eine Baumsprache in einer beliebigen Signatur kodieren wir durch zusätzliche MSO-Variablen, und zwar für jedes Signaturzeichen f eine Variable X_f mit der gewünschten Interpretation: $p \in X_f$, falls auf Position p das Zeichen f steht.

Proof. (Idee.) Wenn wir einen Automaten A für L haben, dann konstruieren wir eine Formel

$$F = \exists X_1 : \exists X_2 : \ldots \exists Y_1 : \exists Y_2 : \ldots \exists Y_{|Q|} : F',$$

mit MSO-Variablen X_f für jedes Zeichen $f \in \Sigma$ (wie eben erklärt) sowie MSO-Variablen Y_q für jeden Zustand $q \in Q(A)$, mit der gewünschten Interpretation: $p \in X_q$, falls in p der Zustand q angenommen wird. In F' steht dann, daß

- partition$(X_1, \ldots, X_{|\Sigma|}$, d. h. jedes p gehört zu genau einem X_f
- partition$(Y_1, \ldots, Y_{|Q|}$, d. h. in jeder Position wird genau ein Zustand angenommen
- die Zustandsübergänge korrekt (entsprechend der Übergangsrelation des Automaten) ausgeführt wurden
- der Zustand in der Wurzel ein akzeptierender Zustand ist

□

Das läßt sich alles frei von Überraschungen durchziehen, und wir erhalten das versprochene Ergebnis: MSO auf $<_1, \ldots, <_a$ ist "dasselbe" wie endliche Baum-Automaten.

Aufgabe 189. *Definiere* partition$(X_1, \ldots, X_k)$ *durch eine MSO-Formel P_k.*

Dabei soll partition$(X_1, \ldots)$ *genau dann wahr sein, wenn die Mengen X_i paarweise disjunkt sind und ihre Vereinigung das gesamte Universum ist.*

3.4 Existentielle MSO auf Graphen

Wir hatten bis jetzt das Universum als Menge von Listen (d. h. Positionen) fixiert, und die Relationen $<_i$ erzeugen darauf eine Baumstruktur.

Das läßt sich verallgemeinern auf andere Objekte: beispielsweise können wir beliebige Graphen betrachten. Die FO-Variablen beziehen sich dann auf Positionen, d. h. Knoten, und MSO-Variablen auf Knotenmengen. Als Relation haben wir dann $N(p, q) \equiv$ die Knoten p und q sind direkt benachbart (sind durch eine Kante in G verbunden).

Wir stellen fest, daß viele interessante Graph-Eigenschaften als MSO-Formeln (in dieser Signatur) geschrieben werden können.

Wir interessieren uns dabei besonders für solche Formeln, bei denen die MSO-Variablen nur außen und *nur existentiell* quantifiziert vorkommen, d. h. die Formeln haben die Gestalt $\exists X_1 : \ldots : \exists X_k : F$ für eine FO-Formel F. Die entsprechende Logik nennen wir EMSO, und es gilt

Satz 190 (Fagin, 1974). *EMSO (auf Graphen)* ist *die Komplexitätsklasse NP (auf Graphen).*

Beispiel 191. *G besitzt eine 3-Färbung:*

$$\begin{aligned}\exists X_1, X_2, X_3 : \quad & \text{partition}(X_1, X_2, X_3) \\ \wedge \quad & \forall p, q \in X_1 : \neg N(p, q) \wedge \forall p, q \in X_2 : \neg N(p, q) \wedge \forall p, q \in X_3 : \neg N(p, q)\end{aligned}$$

Aufgabe 192. *Finde solche Darstellungen für andere Graph-Eigenschaften (aus dem Grundkurs Komplexität).*

Proof. (Idee.) Die Gültigkeit einer EMSO-Formel kann durch eine NP-Maschine geprüft werden, die die Belegung der Mengen-Variablen rät. Für jede solche Belegung ist der Wert der inneren (FO-)Formel sogar in P feststellbar ist. Der Trick dabei ist, daß das sogar in NLOGSPACE $\subseteq$ P geht, denn die geratene Belegung der FO-Variablen kann man ja binär hinschreiben.

Andersherum kann man zu gegebener NP-Maschine eine EMSO-Formel konstruieren, die genau dann erfüllt wird, wenn es eine Berechnungsbaum mit einem Pfad von der Wurzel zu einem akzeptierenden Blatt gibt (analog der Kodierung von $p \ll q$ in Beispiel 176). $\square$

Das muß man natürlich alles noch ein bißchen genauer hinschreiben, das heißt dann *Deskriptive Komplexitätstheorie* oder auch *Endliche Modelltheorie* [EF99]. (Dazu gibt es auch eine Vorlesung von Prof. Herre.)

3.5 Schwache und starke MSO, unendliche Bäume

Kehren wir von Graphen zu Bäumen zurück. Wir haben bisher nur *schwache* MSO betrieben, denn wir können nur Tupel von endlichen Mengen aus Bäume repräsentieren. Nun ist die Frage nach *starker* MSO, bei der also die Mengenvariablen auch mit unendlichen Positionsmengen belegt werden dürfen, durchaus gestattet und sinnvoll.

Ihre Beantwortung (nach dem vorgestellten Schema) erfordert allerdings eine Theorie von regulären Sprachen unendlicher (d. h. unendliche tiefer) Bäume.

Formal kann man sich auch auf unendlichen Bäumen endliche Top-Down-Automaten vorstellen: sie beginnen in der Wurzel und laufen nach unten und verzweigen.

Allerdings halten sie nicht, falls der Baum unendliche Pfade enthält. Deswegen kann man Akzeptanz nicht durch Endzustände erklären, sonder z. B. durch "Baum wird akzeptiert, wenn auf jedem unendlichen Pfad einer der Zustände $\{q_1, \ldots, q_k\}$ unendlich oft benutzt wird".

Damit entsteht aber das nächste Problem: wie rechnet man mit solchen Automaten? Wir wünschen uns ja effektiven Abschluß unter den Operationen, die wir in obigem Beweis brauchten. Da haben wir auch Komplemente gebildet, und das wird für Automaten auf unendlichen Bäumen schwer:

Im endlichen Fall konnten wir die Automaten mit der Potenzmengenkonstruktion vollständig und deterministisch machen (und dann zur Komplementierung akzeptierende und ablehnende Zustände vertauschen). Das ging aber nur für bottom-up-Automaten, und die haben wir eben im unendlichen Fall nicht.

Trotzdem läßt sich das Problem lösen (Satz von Rabin, 1977?) was aber hier nicht mehr ausgeführt werden kann.

Es gibt dazu eine sehr schöne Darstellung von Wolfgang Thomas: *Automata on Infinite Objects*, Handbook of Theoretical Computer Science, Vol. B, Seiten 133 – 192 [Tho90] .