

Skript zur Vorlesung "Algebraische
Automatentheorie", WS 06/07

Lutz Priese
Fachbereich Informatik
Universität Koblenz-Landau, Germany
priese@uni-koblenz.de

Vorwort

Dieses Skript gibt die wesentlichen Inhalte der Vorlesung "Algebraische Automatentheorie" im WS06/07 wieder. Im Skript wird teilweise von der Reihenfolge des Stoffes der Vorlesung abgewichen. Auch worden in der Vorlesung deutlich mehr Beispiele aufgezeigt und mehr Beweise ausgeführt.

1 Automaten über Wörter

Wir setzen in diesem Abschnitt Kenntnisse über endliche Automaten voraus und benutzen die Terminologie aus Erk, Priebe [1]. Automaten sind hier stets endlich, vollständig und determiniert. Ziel ist es zu zeigen, wie nützlich die algebraische Auffassung der Französischen Schule sein kann, indem wir Automaten über Wörter algebraisch analysieren.

Ein Alphabet Σ ist eine endliche, nicht-leere Menge, deren Elemente wir Buchstaben oder ähnlich (z.B. Labels) nennen.

Eine Äquivalenzrelation τ über einer Menge M ist eine reflexive, symmetrische, transitive Relation $\tau \subseteq M \times M$. τ saturiert eine "Sprache" $L \subseteq M$, falls L die Vereinigung einiger Äquivalenzklassen von τ ist. Gleichwertig dazu ist, dass für τ gilt

$$\forall w \in \Sigma^* : [w]_\tau \cap L \neq \emptyset \implies [w]_\tau \subseteq L, \text{ oder}$$

$$\forall w \in \Sigma^* : [w]_\tau \subseteq L \vee [w]_\tau \subseteq \Sigma^* - L.$$

Der Index einer Äquivalenzrelation ist die Anzahl der verschiedenen Äquivalenzklassen dieser Relation.

Für den Begriff einer Kongruenz brauchen wir zusätzlich eine algebraische Struktur über M . Es sei jetzt $M = \Sigma^*$ für ein endliches Alphabet mit der algebraischen Struktur eines Monoids, d.h. $M = (\Sigma^*, \circ, \varepsilon)$, wobei \circ die Konkatenation von Wörtern und ε das leere Wort ist. Statt $u \circ v$ wird nur uv geschrieben und Σ^* bezeichnet üblicherweise sowohl das Monoid M als auch dessen Trägermenge.

Definition 1.1 Eine Rechts-Kongruenz τ über Σ^* ist eine Äquivalenzrelation mit $\forall u, v, w \in \Sigma^* : (u\tau v \implies (uw)\tau(vw))$.

Eine Kongruenz ist eine Rechts-Kongruenz für die zusätzlich gilt:

$$\forall u, v, w \in \Sigma^* : (u\tau v \implies (wu)\tau(vw)).$$

Äquivalent für eine Kongruenz ist die Forderung, dass eine τ eine Äquivalenzrelation ist mit

$$\forall u, v, w, w' \in \Sigma^* : (u\tau v \implies (wuw')\tau(wvw')).$$

Example 1.1 1.) Für einen Automaten $A = (S, \Sigma, \delta, s_0, F)$ definieren wir eine Relation $\approx_A \subseteq (\Sigma^*)^2$ durch: $\forall u, v \in \Sigma^* :$

$u \approx_A v \iff \forall s \in S : \delta_A^*(s, u) = \delta_A^*(s, v)$. Damit gilt offensichtlich:

\approx_A ist eine Kongruenz von endlichem Index, die $L(A)$ saturiert.

2.) Für eine Sprache $L \subseteq \Sigma^*$ ist $\sim_L \subseteq \Sigma^* \times \Sigma^*$ definiert durch $\forall u, v \in \Sigma^* :$

$$u \sim_L v \iff \forall w \in \Sigma^* : (uw \in L \leftrightarrow vw \in L).$$

Offensichtlich ist \sim_L eine Rechts-Kongruenz auf Σ^* , und zwar die größte aller Rechts-Kongruenzen über Σ^* , die L saturieren. Denn ist τ eine Rechtskongruenz über Σ^* , die L saturiert, so folgt aus $u\tau v$ auch $uw\tau vw$, und damit $uw \in L \iff vw \in L$ für jedes $w \in \Sigma^*$, also auch $u \sim_L v$.

Definition 1.2 Es seien $(M_i, \circ_i, 1_i), 1 \leq i \leq 2$, zwei Monoide. Ein Monoidhomomorphismus h von M_1 nach M_2 ist eine Abbildung $h : M_1 \rightarrow M_2$ mit $h(1_1) = 1_2$ und $h(a \circ_1 b) = h(a) \circ_2 h(b), \forall a, b \in M_1$.

Theorem 1 Folgende Aussagen sind für eine Sprache L über Σ äquivalent:

- 1) L ist regulär
- 2) \exists endliches Monoid $(M, \circ_M, 1_M) : \exists B \subseteq M : \exists h : \Sigma^* \rightarrow M$ Monoidhomomorphismus mit $L = h^{-1}(B)$
- 3) \exists Kongruenz mit endlichem Index, die L saturiert.
- 4) \exists Rechts-Kongruenz mit endlichem Index, die L saturiert.
- 5) \sim_L hat endlichen Index.

Häufig nennt man eine Sprache, die Aussage 2) erfüllt auch *rational*, die 4) erfüllt *erkennbar*, und die von endlich Automaten akzeptiert werden auch *akzeptierbar*. Aber diese Begriffe werden in der Literatur auch relativ willkürlich verwendet. Wir werden im Folgenden die Begriffe "regulär", "rational", "erkennbar", "akzeptierbar" synonym verwenden.

Beweis:1) \iff 5) siehe Buch Erk, Priese.

1) \Rightarrow 2): Sei L regulär, so existiert ein endlicher Automat $A = (S, \Sigma, \delta, s_0, F)$, der L akzeptiert. Definiere

$M := S^S, \circ : S^S \times S^S \rightarrow S^S$ definiert als $(f \circ g)(q) := g(f(q)), 1 := id_S, B := \{f : S \rightarrow S \mid f(s_0) \in F\}, h : \Sigma^* \rightarrow S^S$ definiert als $h(w)(q) := \delta^*(q, w)$.

Damit ist (S^S, \circ, id) ein endliches Monoid und h ist ein Monoidhomomorphismus wegen $h(uv) = \lambda q. \delta^*(q, uv) = \lambda q. \delta^*(\delta^*(q, u), v) = \lambda q. h(v)(h(u)(q)) = h(u) \circ h(v)$ und $h(\epsilon) = \lambda q. \delta^*(q, \epsilon) = id_S$. Ferner gilt $h^{-1}(B) = \{w \in \Sigma^* \mid h(w)(s_0) \in F\} = \{w \in \Sigma^* \mid \delta^*(s_0, w) \in F\} = L(A) = L$.

2) \Rightarrow 3): Seien $(M, \circ, 1), B, h$ gegeben wie in Aussage 2). Konstruiere $\tau \subseteq \Sigma^* \times \Sigma^*$ als $u\tau v :\iff h(u) = h(v)$. τ ist Äquivalenzrelation von endlichem Index mit $u\tau v \Rightarrow h(u) = h(v) \Rightarrow h(w_1)h(u)h(w_2) = h(w_1)h(v)h(w_2) \Rightarrow h(w_1uw_2) = h(w_1vw_2) \Rightarrow (w_1uw_2)\tau(w_1vw_2)$, d.h. eine Kongruenz. Sei $B = \{x_1, \dots, x_k\}$, so gilt $L = h^{-1}(B) = \{w \in \Sigma^* \mid h(w) \in B\} = \{w \in \Sigma^* \mid h(w) = x_1\} \cup \dots \cup \{w \in \Sigma^* \mid h(w) = x_k\} = [w_1]_\tau \cup \dots \cup [w_k]_\tau$ für geeignete w_i mit $h(w_i) = x_i$. τ saturiert also L .

3) \Rightarrow 4) per Definition.

4) \Rightarrow 1): Sei τ wie in 4) gegeben. Konstruiere endlichen Automaten $A_L := (S, \Sigma, \delta, s_0, F)$ als

$S := \{[w]_\tau \mid w \in \Sigma^*\}, F := \{[w]_\tau \mid w \in L\}, s_0 := [\epsilon]_\tau, \delta([w]_\tau, a) := [wa]_\tau$, wobei δ unabhängig vom Repräsentanten ist: $[w_1]_\tau = [w_2]_\tau \implies w_1\tau w_2 \implies (w_1a)\tau(w_2a)$.

Offensichtlich folgt $\delta^*([w]_\tau, u) = [wu]_\tau$. Damit gilt

$L(A_L) = \{u \in \Sigma^* \mid \delta^*(s_0, u) \in F\} = \{u \in \Sigma^* \mid [\epsilon u]_\tau \in F\} = \{u \in \Sigma^* \mid u \in L\} = L$. ■

Definition 1.3 Ein Automaten-Homomorphismus $h : A_1 \rightarrow A_2$ von einem Automaten A_i nach A_2 mit $A_i = (S_i, \Sigma_i, \delta_i, s_i, F_i)$ ist eine Paar $h = (h_s, h_l)$ von zwei Abbildungen

$$h_s : S_1 \rightarrow S_2, h_l : \Sigma_1 \rightarrow \Sigma_2,$$

die die "Automatenstruktur" respektieren, d.h., für die gilt

$$h_s(s_1) = s_2, h_s(F_1) \subseteq F_2, h_s(S_1 - F_1) \subseteq S_2 - F_2, \text{ und}$$

$$\forall s \in S_1, a \in \Sigma_1 : \delta_2(h_s(s), h_l(a)) = h_s(\delta_1(s, a)).$$

Man sieht unmittelbar

Corollary 1.1 *Existiert ein Automaten-Homomorphismus (h_s, h_l) von A_1 nach A_2 , dann gilt bereits $h_l(L(A)) = L(B)$.*

Wir betrachten jetzt nur Automaten-Homomorphismen zwischen zwei Automaten mit dem gleichem Alphabet Σ wobei wir stets $h_l = id$ setzen. Existiert nun ein bijektiver Automaten-Homomorphismus zwischen zwei Automaten, so heißen diese isomorph. Isomorphe Automaten unterscheiden sich also höchstens in den Namen ihrer Zustände. Ein Automat heißt minimal, wenn es keinen Automaten mit weniger Zuständen gibt, der die gleiche Sprache akzeptiert. Es gilt

Lemma 1.1 *Es seien A ein Automat mit erreichbaren Zuständen, $L := L(A)$ und A_L der im Satz 1, 4) \rightarrow 1), konstruierte Automat zu \sim_L . Dann existiert ein surjektiver Automaten-Homomorphismus von A nach A_L . Ist A minimal, dann ist A bereits isomorph zu A_L .*

Beweis. Wir konstruieren $h_s : S_A \rightarrow K$, wobei S_A die Zustände von A und $K := \{[w]_{\sim_L} \mid w \in \Sigma^*\}$ die Zustände von A_L sind, für $s \in S_A$ als

$$h_s(s) := [w]_{\sim_L}, \text{ für ein Wort } w \text{ mit } \delta^*(s_o, w) = s.$$

Man sieht leicht, dass h_s wohl-definiert und ein surjektiver Automaten-Homomorphismus ist. Ist A zusätzlich minimal, so muss $|S_A| \leq |K|$ gelten und h_s ist zusätzlich noch injektiv. ■

2 Algebren und Graphen

2.1 Syntax: Signaturen

Definition 2.1 *Ein Signatur \mathcal{S} ist eine Paar*

$$\mathcal{S} = (S, Op),$$

- von einer Menge S , deren Elemente wir Sorten nennen, und
- einer Familie

$$Op = (Op_{w,s})_{w \in S^*, s \in S}$$

von Mengen $Op_{w,s}$ von Elementen, die wir Operatorensymbole nennen, mit $Op_{w,s} \cap Op_{w',s'} = \emptyset$ für $w \neq w'$.

Die Mengen $Op_{w,s}$ können leer, endlich oder unendlich sein. Ein $a \in Op_{w,s}$ heißt auch ($|w|$ -stelliges) Operatorsymbol der Arität w , der Sorte s und des Profils $w \rightarrow s$ und wird auch als $a_{w \rightarrow s}$ geschrieben. Das leere Wort wird dabei weggelassen, d.h. $a_{\rightarrow s}$ ist ein nullstelliges Operatorsymbol der Sorte s vom Profil $\varepsilon \rightarrow s$. Es ist nicht gefordert, dass $Op_{w,s} \cap Op_{w',s'} = \emptyset$ gilt für $w \neq w'$, d.h. das gleiche Symbol kann zwei Operatorensymbole unterschiedlicher Arität sein.

Definition 2.2 *Eine Signatur mit Variablen oder Generatoren ist ein Paar*

$$(\mathcal{S}, \mathcal{X})$$

von einer Signatur $\mathcal{S} = (S, Op)$ und einer Familie $(\mathcal{X} = (X_s)_{s \in S})$ von eventuell leeren Mengen X_s neuer Operatorensymbolen (die in Op nicht vorkommen) von einem Profil $\varepsilon \rightarrow s$. Ein Element in X_s heißt auch Variable oder Generator der Sorte s und wird meist mit x_s oder nur x bezeichnet.

$T(\mathcal{S}, \mathcal{X})$ ist die Familie $T(\mathcal{S}, \mathcal{X}) = (T_s(\mathcal{S}, \mathcal{X}))_{s \in S}$ aller Terme $T_s(\mathcal{S}, \mathcal{X})$ der Sorte s über $(\mathcal{S}, \mathcal{X})$. Sie ist induktiv definierbar als:

- $\forall s \in S : Op_{\rightarrow s} \cup X_s \subseteq T_s(\mathcal{S}, \mathcal{X})$,
- $\forall n \in \mathbb{N}, s_i, s \in S, t_i \in T_{s_i}(\mathcal{S}, \mathcal{X})$, für $1 \leq i \leq n > 0, \forall a \in Op_{s_1 \dots s_n \rightarrow s} : a(t_1, \dots, t_n) \in T_s(\mathcal{S}, \mathcal{X})$.

Eine Signatur \mathcal{S} mit Variablen \mathcal{X} ist natürlich selbst eine Signatur $\mathcal{S}_{\mathcal{X}}$, in der alle Variablen der Sorte $s \in S$ als nullstelliges Operatorensymbol zu $Op_{\varepsilon, s}$ hinzugenommen werden. Die Schreibweise $(\mathcal{S}, \mathcal{X})$ dient nur der Verdeutlichung einer gewissen Sonderrolle der Symbole in \mathcal{X} , sei es als Variablen oder Generatoren in den späteren Theorien (Kapitel 4.1).

Ein Term ohne vorkommende Variable heißt auch Grundterm. $T(\mathcal{S}) = (T_s(\mathcal{S}))_{s \in S}$ bezeichnet die Familie aller Grundterme, $T_s(\mathcal{S})$ die Menge der Grundterme der Sorte s . Kommen in einem Term t aus $T(\mathcal{S}, \mathcal{X})$ höchstens die Variablen x_1, \dots, x_n vor, so schreibt man auch $t(x_1, \dots, x_n)$. Sind t_i Grundterme der gleichen Sorte wie die Variable x_i , für $1 \leq i \leq n$, so bezeichnet $t(t_1, \dots, t_n)$ den Term, der aus dem simultanen Ersetzen aller Vorkommen einer Variablen x_i in t durch Et_i für $1 \leq i \leq n$ entsteht. Ein Term C in $T(\mathcal{S}, \mathcal{X})$ in dem nur eine einzige Variable an genau einer Stelle vorkommt, heißt auch Kontext. Ein Grundterm t

ist passend zum Kontext C , wenn die Sorte von t mit der der einzigen Variablen in C übereinstimmt. Wenn wir $C(t)$ schreiben, impliziert dies stets, dass t zu C auch passt.

Jeder Term $t \in T(\mathcal{S}, \mathcal{X})$ besitzt kanonisch eine Struktur als angeordneter Baum, indem wir in einem Term $f(t_1, \dots, t_n)$ f als Knoten mit den angeordneten Unterbäumen t_1, \dots, t_n auffassen. Bekannte Konzepte von Bäumen lassen sich leicht auf Terme übertragen.

Definition 2.3 Für Terme $t \in T(\mathcal{S}, \mathcal{X})$ definiert man induktiv:

Induktionsbeginn: $t = a \in T_0(\mathcal{S}, \mathcal{X})$, so ist a Wurzel und Blatt (im Term t),

$$\text{Teilterm}(t) := \{a\}, \text{Ast}(t) := \{(a, 0)\}, \text{Tiefe}(t) := 0, |t| := 1,$$

Induktionsschritt: $t = f(t_1, \dots, t_n)$, so ist f Wurzel von t und jedes Blatt von t_i ist auch Blatt von t , ferner ist f Vater von jeder Wurzel von t_i , und jede Wurzel von t_i ist (i -ter) Sohn von f , für $1 \leq i \leq n$,

$$\text{Teilterm}(t) := \{t\} \cup \bigcup_{1 \leq i \leq n} \text{Teilterm}(t_i),$$

$$\text{Ast}(t) := \bigcup_{1 \leq i \leq n} (f, i) \circ \text{Ast}(t_i),$$

$$\text{Tiefe}(t) := 1 + \max_{1 \leq i \leq n} \text{Tiefe}(t_i), |t| := 1 + \sum_{1 \leq i \leq n} |t_i|.$$

Ein Ast ist also ein Wort über $(Op \cup \mathcal{X}) \times \mathbb{N}$, das einen Weg von der Wurzel zu einem Blatt beschreibt. Ein Teilterm wird auch Unterbaum genannt.

Example 2.1 Es seien f 3-stellig, g 2-stellig, a und b 0-stellige Operatortensymbole geeigneter Profile, so dass $t = f(g(a, b), a, f(a, g(b, b), b))$ ein korrekter Term ist. $g(a, b)$ oder $f(a, g(b, b), b)$ sind dann Unterbäume und $(f, 3)(f, 2)(g, 2)(b, 0)$ ist der Ast von der Wurzel f zum Blatt b , der, von oben nach unten betrachtet, zuerst den 3., dann den 2. und am Ende nochmals den 2. Sohn als Weg wählt. Es ist $\text{Tiefe}(t) = 3$ und $|t| = 11$.

2.2 Semantik: Algebren

Definition 2.4 Eine (\mathcal{S} -) Algebra \mathcal{A} zur Signatur $\mathcal{S} = (S, Op)$ ist ein Paar

$$\mathcal{A} = ((\mathcal{A}_s)_{s \in S}, Op^{\mathcal{A}}),$$

- von einer Familie von nicht leeren Mengen \mathcal{A}_s von Elementen der Sorte s , und
- einer Familie $Op^{\mathcal{A}} = (Op_{w \rightarrow s}^{\mathcal{A}})_{w \in S^*, s \in S}$ von Mengen $Op_{w \rightarrow s}^{\mathcal{A}}$ von Abbildungen mit $|Op_{w, s}| = |Op_{w \rightarrow s}^{\mathcal{A}}|$, wobei jedem Operatorsymbol $f \in Op_{w, s}$ mit $w = s_1 \dots s_n$ genau eine Abbildung

$$f^{\mathcal{A}} : \mathcal{A}_{s_1} \times \dots \times \mathcal{A}_{s_n} \rightarrow \mathcal{A}_s$$

in $Op_{w \rightarrow s}^{\mathcal{A}}$ zugeordnet ist.

Algebren zur Signatur \mathcal{S} bilden die Semantik zur abstrakten Syntax \mathcal{S} : ein Operatorsymbol a in $Op_{w,s}$, $w = s_1 \dots s_n$ wird als die Abbildung $a^{\mathcal{A}} : \mathcal{A}_{s_1} \times \dots \times \mathcal{A}_{s_n} \rightarrow \mathcal{A}_s$ in $Op_{w \rightarrow s}^{\mathcal{A}}$ interpretiert. Nullstellige Operatorsymbole einer Sorte s werden damit als ein Element aus \mathcal{A}_s , Konstante der Sorte s , interpretiert. \mathcal{A}_s ist die Trägermenge der Sorte s , die Trägermenge einer Algebra \mathcal{A} ist die Vereinigung aller Trägermengen aller Sorten. Sie wird meist auch mit \mathcal{A} bezeichnet. Aus dem Zusammenhang sollte ersichtlich sein, ob mit \mathcal{A} gerade eine Algebra oder deren Trägermenge gemeint ist. e_s bezeichnet stets ein Element aus \mathcal{A}_s .

Definition 2.5 *Es seien \mathcal{A}, \mathcal{B} zwei Algebren zur gleichen Signatur $\mathcal{S} = (S, Op)$. Ein (Algebra-)Homomorphismus $h : \mathcal{A} \rightarrow \mathcal{B}$ ist eine Familie*

$$h = (h_s)_{s \in S}$$

von Abbildungen $h_s : \mathcal{A}_s \rightarrow \mathcal{B}_s$, so dass für alle $n \in \mathbb{N}$, $s_i, s \in S$, $a_i \in \mathcal{A}_{s_i}$, für $1 \leq i \leq n \geq 0$ und $a \in Op_{s_1 \dots s_n, s}$ gilt:

$$h_s(a^{\mathcal{A}}(a_1, \dots, a_n)) = a^{\mathcal{B}}(h_{s_1}(a_1), \dots, h_{s_n}(a_n)).$$

Insbesondere gilt $h_s(a^{\mathcal{A}}) = a^{\mathcal{B}}$ für ein nullstelliges Symbol a in Op der Sorte s . Für ein Element e der Sorte s schreiben wir statt $h_s(e)$ auch einfach $h(e)$.

Definition 2.6 *Es sei K eine Klasse von \mathcal{S} -Algebren. $\mathcal{A}_i \in K$ heißt initial für K , falls zu jeder Algebra \mathcal{A} in K genau ein \mathcal{S} -Algebra-Homomorphismus*

$$i : \mathcal{A}_i \rightarrow \mathcal{A}$$

existiert. i heißt dann der initiale Homomorphismus.

\mathcal{A}_i heißt initial (über \mathcal{S}), falls \mathcal{A}_i initial bzgl der Klasse aller \mathcal{S} -Algebren ist.

Insbesondere ist die Identität der einzige \mathcal{S} -Algebra-Homomorphismus von \mathcal{A}_i nach \mathcal{A}_i . Zwei bzgl K initiale Algebren sind isomorph.

Example 2.2 *Es sei \mathcal{S} eine Signatur, in der zu jeder Sorte S mindestens ein nullstelliges Operatorsymbol existiert. Die Familie der Grundterme einer Signatur $\mathcal{S} = (S, Op)$ bildet kanonisch selbst eine \mathcal{S} -Algebra $T(\mathcal{S})$, nämlich*

$$T(\mathcal{S}) := ((T_s(\mathcal{S}))_{s \in S}, Op),$$

in der die Elemente der Sorte s gerade alle Grundterme der Sorte s bilden und die Bedeutung eines Operatorsymbols $a_{s_1 \dots s_n, s}$ gerade die formale Anwendung von a auf n Terme aus $T_{s_1}(\mathcal{S}) \times \dots \times T_{s_n}(\mathcal{S})$ ist.

$T(\mathcal{S})$ ist die initiale Algebra über \mathcal{S} . Die Evaluierung $eval^{\mathcal{B}}$ ist der einzige Homomorphismus von $T(\mathcal{S})$ in eine \mathcal{S} -Algebra \mathcal{B} , in der alle Grundterme von \mathcal{S} in \mathcal{B} einfach ausgerechnet werden.

Definition 2.7 Für eine \mathcal{S} -Algebra \mathcal{B} ist die Evaluierung der Homomorphismus von $T(\mathcal{S})$ nach \mathcal{B} definiert durch

$$\text{eval}^{\mathcal{B}}(a) := a^{\mathcal{B}}$$

für alle nullstelligen Operatorsymbole, und

$$\text{eval}^{\mathcal{B}}(a(t_1, \dots, t_n)) := a^{\mathcal{B}}(\text{eval}^{\mathcal{B}}(t_1), \dots, \text{eval}^{\mathcal{B}}(t_n))$$

für alle n -stelligen Operatorsymbole a angewendet auf Grundterme t_i der korrekten Sorten.

Es seien eine Signatur $(\mathcal{S} = (S, \text{Op}), \mathcal{X})$ mit Variablen und eine \mathcal{S} -Algebra \mathcal{B} gegeben, so dass keine Variable ein Element von Op oder von \mathcal{B} ist. Wir erweitern die Evaluierung $\text{eval}^{\mathcal{B}}$ auf Terme in $T(\mathcal{S}, \mathcal{X})$ indem wir

$$\text{eval}^{\mathcal{B}}(x) := x$$

setzen. Es sei $t \in T_s(\mathcal{S}, \mathcal{X})$ ein Term, und (x_1, \dots, x_n) ein Vektor von Variablen, so dass alle Variablen in t in $\{x_1, \dots, x_n\}$ vorkommen. s_i sei die Sorte von x_i . Dann können wir t und (x_1, \dots, x_n) kanonisch als eine Abbildung

$$t^{\mathcal{B}} : \mathcal{B}_{s_1} \times \dots \times \mathcal{B}_{s_n} \rightarrow \mathcal{B}_s$$

auffassen, wobei für $a_i \in \mathcal{B}_{s_i}$ wir $t^{\mathcal{B}}(a_1, \dots, a_n)$ als das Element in \mathcal{B}_s definieren, indem jedes Vorkommen von x_i durch a_i für $1 \leq i \leq n$ in $t^{\mathcal{B}}$ ersetzt wird.

Eine Abbildung $\sigma : \mathcal{X} \rightarrow \mathcal{B}$ mit $\sigma(x_s) \in \mathcal{B}_s$ heißt auch Belegung. $\text{eval}_{\sigma}^{\mathcal{B}}$ erweitert $\text{eval}^{\mathcal{B}}$ durch

$$\text{eval}_{\sigma}^{\mathcal{B}}(x) := \sigma(x).$$

Jedem Ast w einer Länge n in einem Term $t \in T(\mathcal{S})$ kann man durch Induktion über $T(\Sigma)$ kanonisch eine Folge C_1, \dots, C_n von Kontexten wie folgt zuordnen:

- Induktionsbeginn: $t = a \in T_0(\Sigma)$. Dem einzigen Ast $(a, 0)$ wird $C_1 := x$ zugeordnet für eine Variable $x \notin T(\Sigma)$ der Sorte s ,

- Induktionsschritt: $t = f(t_1, \dots, t_n)$. Einem Ast $(f, i) \circ w$, wobei w ein Ast einer Länge k im Unterbaum t_i ist, dem bereits die Folge C_1, \dots, C_k zugeordnet ist, ordnen wir die Folge C_1, \dots, C_k, C_{k+1} mit $C_{k+1} := f(t_1, \dots, t_{i-1}, x, t_{i+1}, \dots, t_n)$ zu mit der Variablen an der i -ten Stelle, für $1 \leq i \leq n$.

Offensichtlich gilt damit

Lemma 2.1 Für jeden Baum t und jeden Ast w mit Blatt a in t gilt für die w zugeordnete Folge C_1, \dots, C_k mit $k = |w|$:

$$t = C_k(C_{k-1}(\dots(C_1(a))\dots)).$$

Für jede \mathcal{S} -Algebra \mathcal{A} ist

$$\text{eval}^{\mathcal{A}}(t) = C_k^{\mathcal{A}}(\dots C_i^{\mathcal{A}}(\text{eval}^{\mathcal{A}}(a))\dots).$$

Example 2.3 $(a^{(3)}, b^{(2)}, c^{(2)}, d^{(1)}, e^{(0)}, f^{(0)}, g^{(0)})$ sei ein gerangtes Alphabet, so erhält der Baum

$$t = a(b(c(d(f), e), d(b(g, f))), e, a(e, e, f))$$

die graphische Darstellung aus Figur 1. Dem Ast $(a, 1)(b, 1)(c, 2)(e, 0)$ werden die Kontexte

$$C_1 = x$$

$$C_2 = c(d(f), x)$$

$$C_3 = b(x, d(b(g, f)))$$

$$C_4 = a(x, e, a(e, e, f)) \text{ zugeordnet. Damit ist } t = C_4(C_3(C_2(C_1(e)))).$$

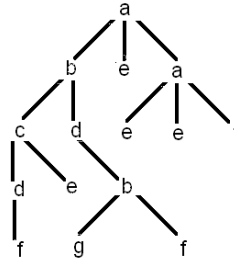


Figure 1: Der Baum t

Example 2.4 Wir betrachten die Signatur

$$Aut = (\{l, s, b\}, \{\delta_{sl,s}, f_{s,b}, s_{\varepsilon,s}, T_{\varepsilon,b}\}),$$

(genauer müssten wir $Aut = (\{l, s, b\}, \{\{\delta_{sl,s}\}, \{f_{s,b}\}, \{s_{\varepsilon,s}\}, \{T_{\varepsilon,b}\}\})$ schreiben, aber wir fassen solch eine triviale endliche Familie von endlichen Mengen selbst als endliche Menge auf)

bestehend aus drei Sorten (für letter, state, boolean). Es sei A eine Aut-Algebra. Wir fassen A_l als Alphabet, A_s als Zustandsmenge, A_b als Wahrheitsmenge, T^A als Wahrheitswert true, δ^A als Transitionsfunktion, s^A als Startzustand und Zustände $s \in A_s$ mit $f^A(s) = T^A$ als finale Zustände in A auf. Damit definieren wir

- $\delta_A^* : A_s \times A_l^* \rightarrow A_s$ induktiv durch $\delta_A^*(s, \varepsilon) := s$ und $\delta_A^*(s, wa) := \delta^A(\delta_A^*(s, w), a)$, und
- $L(A) := \{w \in A_l^* \mid f^A(\delta_A^*(s^A, w)) = T^A\}$.

Es seien A, B zwei Aut-Algebren und $h : A \rightarrow B$ ein Homomorphismus. So gilt offensichtlich

$$h_s(\delta_A^*(s^A, w)) = \delta_B^*(s^B, h_l(w)), \text{ und}$$

$$f^A(\delta^*(s^A, w)) = T^A \implies f^B(\delta_B^*(s^B, h_l(w))) = T^B, \text{ also}$$

$h_l(L(A)) \subseteq L(B)$. Wir können im Gegensatz zu den Automaten-Homomorphismen in Kapitel 1 nicht mehr $h_l(L(A)) = L(B)$ erwarten. Der Grund ist, dass in Kapitel 1 finale Zustände auf finale und nicht-finale auf nicht-finale abgebildet werden mußten. In der Signatur Aut ist aber nicht sichergestellt, dass die Sorte b auch tatsächlich Wahrheitswerte

sind und die Abbildung f zu jedem Zustand exakt sagen muß, ob er entweder final oder nicht-final ist, und dass h_b auch die "Nicht-Finalität" respektiert.

Der Kongruenzbegriff ist generell für Algebren definiert.

Definition 2.8 Eine Kongruenz τ auf einer Algebra \mathcal{A} über $\mathcal{S} = (S, Op)$ ist eine Familie $\tau = (\tau_s)_{s \in S}$ von Äquivalenzrelationen $\tau_s \subseteq \mathcal{A}_s \times \mathcal{A}_s$ so dass für alle $n \in \mathbb{N}$, $s, s_i \in S$, $a_i, a'_i \in \mathcal{A}_i$, für $1 \leq i \leq n$ und $a \in Op_{s_1 \dots s_n, s}$ gilt

$$a_i \tau_{s_i} a'_i \text{ für } 1 \leq i \leq n \implies a^{\mathcal{A}}(a_1, \dots, a_n) \tau_s a^{\mathcal{A}}(a'_1, \dots, a'_n).$$

Sind a, a' Elemente einer gleichen Sorte s , so schreiben wir auch $a \tau a'$ als eindeutige Abkürzung für $a \tau_s a'$.

Für eine beliebige Menge M und Äquivalenzrelation $\tau \subseteq M^2$ auf M ist $M/\tau := \{[m]_\tau \mid m \in M\}$ die Quotientenmenge. Für $\mathcal{A} = (\mathcal{A}_s)_{s \in S}$ definieren wir die Quotientenfamilie als

$$(\mathcal{A})/\tau := ((\mathcal{A}_s)/\tau_s)_{s \in S}.$$

Die Bedeutung einer Kongruenz τ für eine \mathcal{S} -Algebra liegt natürlich darin, dass \mathcal{A}/τ ebenfalls eine \mathcal{S} -Algebra bildet mit

$$f([u_1]_\tau, \dots, [u_n]_\tau) := [f(u_1, \dots, u_n)]_\tau \text{ für } f \in Op_{s_1 \dots s_n \rightarrow s}^{\mathcal{A}}, u_i \in \mathcal{A}_{s_i}.$$

Zwischen Kongruenzen und Homomorphismen besteht folgender elementare Zusammenhang:

Theorem 2 Für jeden Algebra-Homomorphismus $h : \mathcal{A} \rightarrow \mathcal{B}$ von einer \mathcal{S} -Algebra \mathcal{A} in eine \mathcal{S} -Algebra \mathcal{B} ist die Relation τ_h auf \mathcal{A}^2 definiert durch

$$a \tau_h a' :\Leftrightarrow h(a) = h(a')$$

(für Elemente a, a' gleicher Signatur in \mathcal{A}) eine \mathcal{S} -Algebra-Kongruenz über \mathcal{A} .
Für jede \mathcal{S} -Algebra-Kongruenz τ über einer \mathcal{S} -Algebra \mathcal{A} ist $h_\tau(a) := [a]_\tau$ ein \mathcal{S} -Algebra-Homomorphismus

$$h_\tau : \mathcal{A} \rightarrow \mathcal{A}/\tau.$$

Für Kongruenzen auf der initialen Algebra $T(\mathcal{S})$ mit $\mathcal{S} = (S, Op)$ gilt

Corollary 2.1 Eine Familie τ von Äquivalenzrelationen über $T_s(\mathcal{S})$ ist eine Kongruenz auf $T(\mathcal{S})$ genau dann, wenn für jeden Kontext C und für je zwei passende Grundterme t, t' aus $t \tau t'$ auch $C(t) \tau C(t')$ folgt.

Beweis. \implies : Es seien τ eine Kongruenz auf $T(\mathcal{S})$, und t, t' Grundterme mit $t \tau t'$, dann gilt auch $C(t) \tau C(t')$ gemäß folgender Induktion:

- Es sei $C = x$, so gilt $C(t) = t \tau t' = C(t')$, falls t, t' zu C passen,
- für $C = a$, a nullstellig, gilt $C(t) = a \tau a = C(t')$,

- für $C = f(T_1, \dots, T_n)$ für $T_i \in T(\mathcal{S}, \mathcal{X})$, wobei genau ein T_i ein Context und alle anderen Grundterme sind, gilt per Induktionvoraussetzung $T_i(t) \tau T_i(t')$, also auch $C(t) = f(T_1(t), \dots, T_n(t)) \tau C(T_1(t'), \dots, T_n(t')) = C(t')$.

\Leftarrow : Es gelte per Voraussetzung $C(t) \tau C(t')$ für jeden Kontext und passende Grundterme t, t' mit $t \tau t'$. Zu zeigen ist, dass für $f_{s_1 \dots s_n, s} \in Op$, Grundterme t_i, t'_i von der Sorte s_i mit $t_i \tau t'_i$ für $1 \leq i \leq n$ bereits $f(t_1, \dots, t_n) \tau f(t'_1, \dots, t'_n)$ gilt. Dazu betrachten wir die Folge $(C_i)_{1 \leq i \leq n}$ von Kontexten

$$C_i := f(t'_1, \dots, t'_{i-1}, x_i, t_{i+1}, \dots, t_n),$$

wobei x_i eine Variable der Sorte s_i ist. Es gilt nun

$$\begin{aligned} f(t_1, \dots, t_n) &= C_1(t_1) \tau C_1(t'_1) = f(t'_1, t_2, \dots, t_n) = C_2(t_2) \tau C_2(t'_2) = f(t'_1, t'_2, t_3, \dots, t_n) \\ &= C_3(t_3) \tau C_3(t'_3) = f(t'_1, t'_2, t'_3, t_4, \dots, t_n) \dots = C_n(t_n) \tau C_n(t'_n) = f(t'_1, \dots, t'_n). \end{aligned}$$

■

Definition 2.9 Eine Kongruenz τ saturiert eine Teilmenge $M \subseteq \mathcal{A}_s$ von Elementen einer Sorte s , falls für alle $a \in \mathcal{A}_s$ gilt, dass $[a]_\tau$ ganz in M oder ganz in $\mathcal{A}_s - M$ liegt.

Gleichwertig sind die Forderungen, dass von zwei τ -kongruenten Elementen der Sorte s nicht eines in M und das andere nicht in M liegen kann, bzw. dass M die Vereinigung einiger Äquivalenzklassen von τ ist.

2.3 Graphen, dags, Wälder, Bäume

Wir stellen hier, ohne in die Details zu gehen oder formale Definition zu geben, eine klassische mengentheoretische Vorgehensweise vor, Graphen, dags, Wälder und Bäume zu behandeln.

Ein Graph G ist ein Paar $G = (V, E)$ einer Menge V von Knoten und einer Menge $E \subseteq V \times V$ von gerichteten Kanten, wobei $(v, v') \in E$ eine Kante vom Vaterknoten v zum Sohnknoten v' beschreibt. G ist endlich, falls V endlich ist. Der in-degree (out-degree) eines Knoten ist die Anzahl seiner Väter (Söhne). Eine Wurzel ist ein Knoten mit in-degree 0, ein Blatt ein Knoten mit out-degree 0. Ein Weg (von v_1 nach v_n) ist eine Folge v_1, \dots, v_n von Knoten $v_i \in V$, so dass $(v_i, v_{i+1}) \in E$ für $1 \leq i < n$ gilt. Ein Kreis ist ein Weg von v_1 nach v_n mit $v_1 = v_n$. Ein dag (acyclic directed graph) ist ein Graph, in dem keine Kreise vorkommen. Ein Schnitt in einem dag ist eine Menge $S \subseteq V$ von Knoten, so dass jeder Weg von einer Wurzel zu einem Blatt genau einen Knoten aus S enthalten muss.

Ein Wald ist ein Graph, in dem es zwischen zwei Knoten maximal einen Weg gibt. Damit muss ein Wald auch ein dag sein. Ein Baum ist ein Wald mit genau einer Wurzel. Als Konsequenz muss es von der einzigen Wurzel zu jedem Nicht-Wurzelknoten genau einen einzigen Weg geben.

Ein knotengelabelter Graph ist ein Tripel (V, E, λ) , wobei $\lambda : V \rightarrow \Sigma$ eine Labelfunktion ist, die jedem Knoten einen Label aus einer Labelmenge Σ zuordnet. Ein kantengelabelter Graph ist ein Tripel (V, E, Σ) aus einer Knotenmenge V , einer Labelmenge Σ und einer Menge von gelabelter Kanten $E \subseteq V \times \Sigma \times V$.

Ein Graph ist angeordnet, wenn es zu jedem Knoten eine totale Anordnung auf der Menge seiner Söhne gibt. Der erste Sohn heißt dann auch der linkeste Sohn, der letzte der rechteste, motiviert von Bäumen, in denen man in graphischen Darstellungen die Söhne unterhalb des Vaters zeichnet. Ein gelabelter Graph heißt gerangt, wenn alle Knoten des Labels auch den gleichen out-degree besitzen. Üblicherweise werden die Begriffe "angeordnet" und "gerangt" nur für Bäume und Wälder, nicht aber für dags oder generelle Graphen verwendet.

Abmachung: In gesamten folgenden Text sind Graphen (und damit auch dags, Wälder und Bäume) stets endlich und knotengewichtet und isomorphe Graphen werden identifiziert.

Dabei heißen zwei Graphen $G_i = (V_i, E_i, \lambda_i), i = 1, 2$, isomorph, falls λ_i Labelfunktionen mit einem gleichen Wertebereich Σ sind und eine bijektive Funktion $h : V_1 \rightarrow V_2$ existiert mit $\forall v, v_1, v_2 \in V_1$ gilt

$$(v_1, v_2) \in E_1 \Leftrightarrow (h(v_1), h(v_2)) \in E_2, \text{ und } \lambda_1(v) = \lambda_2(h(v)).$$

D.h., wir betrachten nur abstrakte Graphen, das sind Äquivalenzklassen von isomorphen Graphen.

Wenn man diese systematische top-down Definitionen von Graphen zu Bäumen verlässt, kann man für manche Teilklassen von Graphen auch einfachere mengentheoretische Definitionen angeben. Betrachten wir z.B. angeordnete Bäume, so kann man auch definieren: Ein angeordneter Baum (über der Labelmenge Σ) ist ein Paar (Pos, λ) von einer Labelfunktion $\lambda : Pos \rightarrow \Sigma$ und einer präfixabgeschlossenen Menge $Pos \subseteq \mathbb{N}^*$ von Positionen.

Example 2.5 *Es sei $G_1 = (Pos_1, \lambda_1)$ mit $Pos_1 := \{\varepsilon, 1, 2, 3, 1 \circ 1, 1 \circ 1 \circ 1, 1 \circ 1 \circ 2\}$ und $\lambda_2 : \varepsilon \rightarrow a, 1 \rightarrow a, 2 \rightarrow b, 3 \rightarrow a, 1 \circ 1 \rightarrow c, 1 \circ 1 \circ 1 \rightarrow a, 1 \circ 1 \circ 2 \rightarrow b$ ein angeordneter Baum über $\{a, b, c\}$. Wir haben hier die Konkatenation in \mathbb{N}^* als \circ ausgeschrieben um die Zahl 11 klar von dem Wort 1o1 der beiden Zahlen 1 zu unterscheiden. Übersetzt in unsere mengentheoretische Definition ergibt dies den Graphen $G = (Pos, E, \lambda)$ mit*

$$(v, v') \in E \Leftrightarrow \exists i \in \mathbb{N} : v' = v \circ i,$$

wobei die Anordnung zweier Brüder $v \circ i, v \circ j$ als

$$v \circ i < v \circ j : \Leftrightarrow i < j$$

definiert ist. Allerdings ist hier noch nicht berücksichtigt, dass man eigentlich abstrakte Graphen untersuchen will. Definieren wir $G_2 = (Pos_2, \lambda_2)$ mit $Pos_2 := \{\varepsilon, 25, 26, 54, 25 \circ 25, 25 \circ 25 \circ 25, 25 \circ 25 \circ 26\}$ und $\lambda_2 : \varepsilon \rightarrow a, 25 \rightarrow a, 26 \rightarrow b, 54 \rightarrow a, 25 \circ 25 \rightarrow c, 25 \circ 25 \circ 25 \rightarrow a, 25 \circ 25 \circ 26 \rightarrow b$, so sind G_1 und G_2 isomorph. Beide erhalten als abstrakter Graph die gleiche "eindeutige" graphische Darstellung wie in Figure 2.

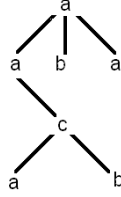


Figure 2: G_1 und G_2 als Baum gezeichnet

3 Angeordnete und gerangte Bäume

3.1 Angeordnete und gerangte Bäume

In diesem Kapitel ist unter Baum stets ein angeordneter und gerangter Baum gemeint. Diese lassen sich algebraisch sehr elegant behandeln.

Definition 3.1 *Ein gerangtes Alphabet Σ ist eine endliche Familie $\Sigma = (\Sigma_n)_{n \leq k}$ für ein $k \in \mathbb{N}$ von leeren oder endlichen Mengen Σ_n , deren Elemente Buchstaben der Arität n heißen.*

Wir identifizieren ein gerangtes Alphabet Σ mit der Signatur

$$\Sigma := (\{l\}, (Op_{l^n, l})_{n \in \mathbb{N}}),$$

wobei ein Buchstabe f der Arität n als ein Operatorsymbol vom Profil $l^n \rightarrow l$ in $Op_{l^n, l}$ aufgefasst wird. Mit $f^{(n)}$ kürzen wir die Schreibweise $f_{l^n \rightarrow l}$ ab. $f^{(n)}$ kürzt also ab, dass f ein Buchstabe der Arität n aufgefasst als ein Operatorsymbol der Arität l^n ist. In einer Σ -Algebra \mathcal{A} ist $f^{(n)\mathcal{A}}$ also eine Funktion von \mathcal{A}^n nach \mathcal{A} , wobei $\mathcal{A} := \mathcal{A}_l$ die einzige Trägermenge der einzigen Sorte l (wie letter) ist. Da wir gerangte Alphabete somit als Signatur auffassen, sind alle Begriffe wie Grundterm, Kontext, Kongruenz, etc., für gerangte Alphabete bereits erklärt.

Abmachung: Ein Baum ist in diesem Kapitel ein Grundterm $t \in T(\Sigma)$ über einem gerangtem Alphabet Σ .

Damit wird die Tatsache, dass wir abstrakte Graphen untersuchen wollen, automatisch erfüllt. Auch sind diese abstrakten Bäume automatisch gerangt und angeordnet. Begriffe wie Wurzel, Blatt, Vater von, Sohn von, Teilbaum, Ast, Tiefe, Größe $|t|$ sind damit in Kapitel 2 bereits definiert worden.

3.2 Automaten auf angeordneten, gerangten Bäumen

Ein **bottom-up** Baumautomat arbeitet intuitiv wie folgt: Zuerst wird jedes Blatt mit einem initialen Zustand zusätzlich annotiert. Sind alle n Söhne eines Knoten v des out-degrees n bereits mit Zuständen annotiert, v aber noch nicht, so erhält v einen Zustand,

der vom Label von v und dem Vektor der n Zustände seiner Söhne gemäß der Transitionrelation des Automaten abhängt. Erhält ein Vaterknoten einen Zustand, so werden alle Zustände seiner Söhne entfernt. Der Zustand, der der Wurzel zugeordnet wird, sagt, ob der Baum akzeptiert werden soll. Formal:

Definition 3.2 *Ein (endlicher bottom-up) indeterminierter Baumautomat*

$$A = (S, \Sigma, \Delta, F)$$

besteht aus

- einer endlichen Menge S von Zuständen der Arität 1,
- einem gerangtem Alphabet Σ , mit $S \cap \Sigma = \emptyset$,
- einer Relation $\Delta \subseteq \bigcup_n (\Sigma_n \times S^n) \times S$ von Transitionen,
- einer Menge $F \subseteq S$ von finalen Zuständen.

Die Tatsache, dass wir zu den Labels noch Zustände annotieren wollen, können wir algebraisch elegant ausdrücken, in den wir die Zustände als neue Buchstabe der Arität 1 auffassen und vor den betreffenden Knoten als neuen Vater hängen.

Definition 3.3 *Eine Konfiguration K von A ist ein Baum über dem gerangtem Alphabet $\Sigma \cup S$, in dem alle Zustände einen Schnitt bilden.*

Eine Startkonfiguration K_t zu einem Baum t über Σ ist eine Konfiguration, die man aus t erhält, indem simultan jede Wurzel $a^{(0)}$ durch einen Ast $q(a)$ (indeterminiert) ersetzt wird für ein $q \in S$ mit $(a, \varepsilon)\Delta q$.

Eine Konfiguration K' ist eine direkte Nachfolgekonfiguration von K , in Zeichen $K \vdash K'$, falls in K simultan jeder Teilterm einer Form $f(q_1(u_1), \dots, q_n(u_n))$ (indeterminiert) ersetzt wird durch $q(f(u_1, \dots, u_n))$ für ein q mit $(f, q_1, \dots, q_n)\Delta q$.

Eine Konfiguration, in der die Wurzel ein Zustand ist (der dann der einzige im Baum sein muss), heißt Endkonfiguration.

Wir schreiben $t \vdash_A^ q$, falls es eine Startkonfiguration K_t zu dem Baum t über Σ und eine Rechnung $K_t \vdash^* K$ von K_t zu einer Endkonfiguration K mit Wurzel $q \in S$ gibt. A akzeptiert einen Baum t , falls $t \vdash_A^* q$ für einen finalen Zustand $q \in F$ gilt. Die Menge aller von A akzeptierten Bäume ist die (Baum)Sprache $L(A)$ von A .*

Eine Sprache $L \subseteq T(\Sigma)$ heißt regulär, wenn sie von einem endlichen, indeterminierten bottom-up Baumautomaten akzeptiert wird.

Ein determinierter Baumautomat ist ein indeterminierter Baumautomat dessen Transitionrelation Δ eine Funktion, d.h. eine vollständige und funktionale Relation ist, $\Delta : \bigcup_n (\Sigma_n \times S^n) \rightarrow S$. In diesem fall wird die Transitionsfunktion Δ üblich als δ bezeichnet.

Wir können auch einen "langsameren" Rechnungsbegriff einführen, in dem $K \vdash K'$ besagt, dass in K indeterminiert nur ein Vorkommen eines Teilterms einer Form $f(q_1(u_1), \dots, q_n(u_n))$

durch $q(f(u_1, \dots, u_n))$ für ein q mit $(f, q_1, \dots, q_n)\Delta q$ ersetzt wird. Die Sprache eines Automaten ändert sich dadurch nicht.

Ein **top-down** Baumautomat arbeitet intuitiv wie folgt: Zuerst wird die Wurzel mit einem initialen Zustand annotiert. Ist ein Knoten v des out-degrees n bereits mit einem Zustand annotiert, seine n angeordneten Söhne v_1, \dots, v_n aber noch nicht, so erhalten alle Söhne v_i simultan (aber möglicherweise indeterminiert) einen Zustand q_i , der vom Zustand des Vaterknoten v und der Position (in der Anordnung) von v_i gemäß der Transitionrelation des Automaten abhängt. Dabei werden wieder die Zustände der Väter entfernt, sobald die Söhne Zustände erhalten. Formal:

Definition 3.4 *Ein indeterminierter top-down Automat*

$$A = (S, \Sigma, \Delta, I)$$

besteht aus

- einer endlichen Menge S von Zuständen der Arität 1,
- einem gerangtem Alphabet Σ , mit $S \cap \Sigma = \emptyset$,
- einer Relation $\Delta \subseteq \bigcup_n ((S \times \Sigma_n) \times S^n)$ von Transitionen,
- einer Menge $I \subseteq S$ von initialen Zuständen.

Eine Konfiguration ist jetzt ein Baum über $\Sigma \cup S$, in denen alle Zustände nur noch Teilmenge eines Schnitts zu sein brauchen. Eine Startkonfiguration K_t zu einem Baum t entsteht jetzt durch Ersetzen der Wurzel a durch den Ast $q(a)$ für einen Zustand $q \in I$. Eine direkte Nachfolgerkonfiguration K' einer Konfiguration K , $K \vdash K'$, entsteht jetzt durch simultanes Ersetzen aller Teilterme $q(f^{(n)}(u_1, \dots, u_n))$ durch $f(q_1(u_1), \dots, q_n(u_n))$ für einen Zustandsvektor (q_1, \dots, q_n) mit $(q, f)\Delta(q_1, \dots, q_n)$. Ein indeterminierter top-down Automat akzeptiert einen Baum t über Σ , falls eine Startkonfiguration K_t existiert mit $K_t \vdash^* t$. $L(A)$ ist die Menge aller von A akzeptierten Bäume.

Ein Zustand q , der Vater eines Blattes $a^{(0)}$ ist, kann verschwinden, wenn $(q, a)\Delta\varepsilon$ gilt, wobei ε der nullstellige Vektor in S^0 ist. Daher müssen "am Ende" einer Rechnung die Zustände keinen kompletten Schnitt mehr bilden und eine Rechnung von K_t aus kann mit dem Baum t über Σ selbst enden. In einer akzeptierenden Rechnung müssen am Schluss alle Zustände verschwunden sein.

Definition 3.5 *Das Reverse Δ^{-1} einer Relation $\Delta \subseteq \bigcup_n ((\Sigma_n \times S^n) \times S)$ ist*

$$\Delta^{-1} := \{((q, f^{(n)}), (q_1, \dots, q_n)) \mid (f^{(n)}, (q_1, \dots, q_n))\Delta q\},$$

und das Reverse Δ^{-1} einer Relation $\Delta \subseteq \bigcup_n ((S \times \Sigma_n) \times S^n)$ ist

$$\Delta^{-1} := \{((f^{(n)}, (q_1, \dots, q_n)), q) \mid (q, f^{(n)})\Delta(q_1, \dots, q_n)\}.$$

Das Reverse A^{-1} eines Automaten $A = (S, \Sigma, \Delta, B)$ ist

$$A^{-1} := (S, \Sigma, \Delta^{-1}, B).$$

Damit ist das Reverse eine top-down Automat ein bottom-up Automat und umgekehrt. B sind in einem Fall initiale, im anderen finale Zustände.

Lemma 3.1 Für jeden Baumautomaten A gilt $L(A) = L(A^{-1})$. Damit sind die Sprachklassen von indeterminierten bottom-up und top-down Automaten gleich.

Beweis. Für eine "langsame" Rechnung $K_{i-1} \vdash_A K_i \vdash_A K_{i+1}$ ist $K_{i+1} \vdash_{A^{-1}} K_i \vdash_{A^{-1}} K_{i-1}$ eine langsame Rechnung in A^{-1} . Damit gilt sofort

$$L(A) \subseteq L(A^{-1}), \text{ also auch} \\ L(A) \subseteq L(A^{-1}) \subseteq L((A^{-1})^{-1}) = L(A).$$

■

Dieses Resultat mag in ersten Moment etwas erstaunen, da ein bottom-up Automat bei der Zuordnung eines Zustandes zu einem Vaterknoten bereits alle Zustände aller Söhne, in denen auch deren Label codiert sein können, kennt, ein top-down Automat aber "vorwärtsblind" ist in dem Sinn, dass er die Weitergabe eines Zustandsvektors an seine Söhne aber nicht von deren Label abhängig machen kann. Indirekt kann er es aber doch durch den inhärenten Indeterminismus: es können die Label der Söhne geraten werden durch Zustandsvektoren, die keine weiteren Übergänge mehr erlauben, wenn die tatsächlich vorhandenen Label der Söhne von den geratenen verschieden sind. Man sieht relativ leicht, dass eine erfolgreiche geratene (akzeptierende) Rechnung top-down in einem top-down Baumautomaten A zu einer erfolgreichen bottom-up Rechnung in A^{-1} führt, und umgekehrt.

Ist in einem top-down Automaten A die Transitionsrelation Δ eine Funktion, so heißt A ein determinierter top-down Automat. Die Klasse aller von determinierten top-down Automaten akzeptierten Sprache ist allerdings eine echte Teilmenge der von beliebigen indeterminierten top-down Automaten akzeptierten Sprachen.

Example 3.1 Die Sprache $L_1 := \{f(a, b), f(b, a)\}$ über $\Sigma = \{f^{(2)}, a^{(0)}, b^{(0)}\}$ kann von einem indeterminierten top-down Automaten akzeptiert werden, allerdings nicht von einem determinierten. Jeder determinierte top-down Automat, der die Bäume $f(a, b), f(b, a)$ akzeptiert, muss auch den Baum $f(a, a)$ akzeptieren. Ändert man die Definition von determinierten top-down Automaten so ab, dass die "Vorwärts-Blindheit" behoben wird, indem die Übergabe eines Zustandsvektor an die Söhne sowohl von Zustand des Vaters als auch der Label aller Söhne abhängt, ändert sich die Situation nicht. Solche modifizierten determinierten top-down Automaten könnten zwar L_1 akzeptieren, aber nicht $L_2 := \{f(f(a, a), f(b, b)), f(f(b, b), f(a, a))\}$.

Bei bottom-up Automaten ist die Situation erfreulicher, da hier die Klassen der von indeterminierten und determinierten Automaten akzeptierten Sprachen übereinstimmen. Es gilt

Theorem 3 Eine Sprache L von Bäumen wird von einem indeterminierten bottom-up Automaten akzeptiert genau dann, wenn L von einem determinierten bottom-up Automaten akzeptiert wird.

Beweis: \Leftarrow ist trivial, da jeder determinierte Automat per Definition auch ein indeterminierter ist.

\Rightarrow : $A = (S, \Sigma, \Delta, F)$ akzeptiere $L \subseteq T(\Sigma)$. A^{det} sei definiert als

$$A^{det} := (2^S, \Sigma, \delta, F'), \text{ mit}$$

- $F' := \{N \subseteq S \mid N \cap F \neq \emptyset\}$,

- $\delta(f^{(n)}, (N_1, \dots, N_n)) := \{q \in S \mid \exists q_i \in N_i \text{ f\"ur } 1 \leq i \leq n : (f, (q_1, \dots, q_n)) \Delta q\}$.

Damit akzeptiert A^{det} ebenfalls L . ■

Abmachung: Im Rest dieses Kapitels ist ein Automat stets ein endlicher, determinierter, bottom-up Baumautomat.

3.3 Charakterisierungen von regulären Baumsprachen

Wir haben einen Algebra-Homomorphismus zwischen zwei Algebren einer gleichen Signatur definiert. Diese Homomorphismen verändern also die Signaturen nicht. Unter einem Baum-Homomorphismus versteht man in der Literatur aber häufig auch eine Veränderung der Signatur. Zur Vermeidung von Verwechslungen mit algebraischen Homomorphismen ziehen wir daher den Namen Signatur-Ersetzung hier vor.

Definition 3.6 *Es seien Σ, Σ' zwei gerangte Alphabete, x_i Variablen der Arität 0, die in $\Sigma \cup \Sigma'$ nicht vorkommen, $X_i := \{x_1, \dots, x_n\}$ und $\mathcal{X} := (X_i)_{i \in \mathbb{N}}$. Eine Signatur-Ersetzung h von Σ_1 nach Σ_2 ist eine Abbildung*

$$h : \Sigma_1 \rightarrow T(\Sigma_2, \mathcal{X}), \text{ mit } h(f^{(n)}) \in T(\Sigma_2, X_n).$$

Eine Substitution heißt linear, wenn jedes $h(f)$ ein linearer Term ist, in dem also jede Variable maximal einmal an einer Stelle vorkommt.

Man erweitert h zu einer Abbildung

$$h^* : T(\Sigma_1) \rightarrow T(\Sigma_2) \text{ durch}$$

$$h^*(a^{(0)}) := h(a), \quad h^*(f^{(n)}(t_1, \dots, t_n)) := h(f)(t_1, \dots, t_n).$$

Vorsicht: $f(t_1, \dots, t_n)$ ist per Definition ein Grundterm in $T(\Sigma_1)$, $h(f)$ ist ein Term in $T(\Sigma_2, X_n)$ mit Variablen x_1, \dots, x_n , und damit ist $h(f)(t_1, \dots, t_n)$ der Grundterm über Σ_2 der aus $h(f)$ durch simultanes Ersetzen aller Variabler x_i durch t_i entsteht.

Example 3.2 *Es seien $\Sigma = \{a^{(3)}, b^{(0)}, c^{(0)}\}$, $\Sigma' = \{a_1^{(2)}, a_2^{(2)}, b^{(0)}, c^{(0)}\}$, $h(b) = b$, $h(c) = c$, $h(a) = a_1(x_1(a_2(x_2, x_3)))$, so ist h eine lineare Signatur-Ersetzung, die ternäre Bäume über Σ auf binäre über Σ' abbildet. So ist z.B. $h(a(b, c, a(b, b, c))) = a_1(b, a_2(c, a_1(b, a_2(b, c))))$. h ist eine eindeutige Verschlüsselung von Ternär- in Binärbäume.*

Example 3.3 Für $\Sigma = \{a^{(2)}, b^{(1)}, c^{(0)}\}$ ist $h(a) = a$, $h(b) = a(x_1, x_1)$, $h(c) = c$ eine nicht lineare Substitution von Σ nach Σ , die z.B. die reguläre Baumsprache $L = \{b^n(c) \mid n \in \mathbb{N}\}$ aller Äste über b, c abbildet auf die nicht reguläre Baumsprache alle vollständigen Binärbäume über a, c , in der alle Äste eines Baumes die gleiche Länge besitzen müssen.

Es gilt

Theorem 4 Die Klasse aller regulären Baumsprachen ist abgeschlossen gegen Vereinigung, Durchschnitt, Komplement, lineare Signatur-Ersetzung, inverse Signatur-Ersetzung, aber nicht gegen beliebige Signatur-Ersetzung.

Die Beweise für Vereinigung, Durchschnitt, Komplement sind völlig analog zu denen für reguläre Sprachen über Wörter. Für Signatur-Ersetzung finden sie sich z.B. im TATA-Buch.

Definition 3.7 Es seien $A = (S, \Sigma, \delta, F)$ ein Automat und L eine Baumsprache $L \subseteq T(\Sigma)$ über einem gerangten Alphabets Σ . Die Relationen $\equiv_A, \equiv_L \subseteq T(\Sigma) \times T(\Sigma)$ über Bäumen sind definiert durch

$$t \equiv_A t' :\Leftrightarrow \exists s \in S : (t \vdash^* s \text{ und } t' \vdash^* s),$$

$$t \equiv_L t' :\Leftrightarrow \forall C \text{ Kontext in } T(\Sigma, \{x\}) : (C(t) \in L \leftrightarrow C(t') \in L).$$

Aus der Definition folgt unmittelbar:

Lemma 3.2 Es gilt

- \equiv_A und \equiv_L sind Kongruenzen,
- \equiv_A und \equiv_L saturieren beide L ,
- $\text{Index}(\equiv_A) \leq |S|$,
- ist τ eine Kongruenz, die L saturiert, so folgt aus $t \tau t'$ sofort $t \equiv_L t'$,
- \equiv_L ist die größte aller Kongruenzen, die L saturieren.

Theorem 5 Für eine Baumsprache $L \subseteq T(\Sigma)$ sind folgende Aussagen äquivalent

- (1) L ist regulär,
- (2) L wird von einer Baumkongruenz τ mit endlichem Index saturiert,
- (3) \equiv_L besitzt endlichen Index,
- (4) es existiert eine endliche Σ -Algebra \mathcal{E} , eine Teilmenge B von \mathcal{E} und ein $(\Sigma$ -Algebra-) Homomorphismus h von $T(\Sigma)$ nach \mathcal{E} mit $L = h^{-1}(B)$,
- (5) es existiert eine endliche Σ -Algebra \mathcal{E} und eine Teilmenge B von \mathcal{E} mit

$$L = \{t \in T(\Sigma) \mid \text{eval}^{\mathcal{E}}(t) \in B\}.$$

Historisch gesehen wurden Sprachen, die (2) erfüllen, "erkennbar", und solche die (4) erfüllen, "rational" genannt.

Beweis. (1) \implies (2): Sei A ein Automat, der L akzeptiert, dann ist z.B. \equiv_A eine Kongruenz von endlichem Index, die L saturiert.

(2) \implies (3): \equiv_L ist gröber, als jede Kongruenz, die L saturiert. Also $\text{Index}(\equiv_L) \leq$

$Index(\tau) < \infty$.

(3) \implies (4): Mit Satz 2 liefert die Kongruenz \equiv_L einen Homomorphismus $h_{\equiv_L} : T(\Sigma) \rightarrow T(\Sigma)_{/\equiv_L}$. Da \equiv_L einen endlichen Index hat, ist $T(\Sigma)_{/\equiv_L} = \{[t]_{\equiv_L} \mid t \in T(\Sigma)\}$ endlich. Mit $B := \{[t]_{\equiv_L} \mid t \in L\}$ gilt dann $L = h_{\equiv_L}^{-1}(B)$.

(4) \implies (5): Da $T(\Sigma)$ die initiale Σ -Algebra ist, ist $h = eval^{\mathcal{E}}$ der einzige Homomorphismus von $T(\Sigma)$ nach \mathcal{E} .

(5) \implies (1): Jede endliche Σ -Algebra \mathcal{E} definiert kanonisch einen Halbautomaten $A := (\mathcal{E}, \Sigma, \delta)$ ohne finale Zustände mittels

$$\delta(f^{(n)}, e_1, \dots, e_n) := f^{\mathcal{E}}(e_1, \dots, e_n).$$

Offensichtlich gilt für $t \in T(\Sigma)$:

$$t \vdash_A^* eval^{\mathcal{E}}(t).$$

Wählt man zu A den Zustandsraum B , so gilt $L(A) = \{t \in T(\Sigma) \mid eval^{\mathcal{E}}(t) \in B\} = (eval^{\mathcal{E}})^{-1}(B)$. ■

Theorem 6 *Jeder Automat über Σ ist eine endliche Σ -Algebra mit einer ausgezeichneten "finalen" Teilmenge der Trägermenge. Jede endliche Σ -Algebra ist ein Automat über Σ ohne definierten Zustandsraum.*

Beweis. Der Automat $A = (S, \Sigma, \delta, F)$ ist kanonisch eine endliche Σ -Algebra \mathcal{A} mit

$$\mathcal{A} := (S, \{f^{(n)\mathcal{A}} \mid f^{(n)} \in \Sigma\}) \text{ und } f^{(n)\mathcal{A}}(s_1, \dots, s_n) := \delta(f^{(n)}, s_1, \dots, s_n).$$

Damit gilt

$$t \vdash_A^* eval^{\mathcal{A}}(t),$$

und somit $L(A) = \{t \in T(\Sigma) \mid eval^{\mathcal{A}}(t) \in F\} = (eval^{\mathcal{A}})^{-1}(F)$. Wir wählen also F als ausgezeichnete Teilmenge in \mathcal{A} .

Umgekehrt definiert jede endliche Σ -Algebra $\mathcal{A} = (\mathcal{A}, \{f^{\mathcal{A}} \mid f \in \Sigma\})$ kanonisch den "Halb-" Automaten $A = (\mathcal{A}, \Sigma, \delta)$ ohne Zustände mit

$$\delta(f^{(n)}, e_1, \dots, e_n) := f^{\mathcal{A}}(e_1, \dots, e_n).$$
■

Example 3.4 *Zu einer regulären Sprache $L \subseteq T(\Sigma)$ sei der Automat $A_L = (S, \Sigma, \delta, F)$ definiert durch*

$$S := \{[t]_{\equiv_L} \mid t \in T(\Sigma)\}, F := \{[t]_{\equiv_L} \mid t \in L\}, \text{ und} \\ \delta(f^{(n)}, [t_1]_{\equiv_L}, \dots, [t_n]_{\equiv_L}) := [f(t_1, \dots, t_n)]_{\equiv_L}.$$

Offensichtlich ist δ wohl-definiert und es gilt für einen Baum $t \in T(\Sigma)$:

$$t \vdash_A^* [t]_{\equiv_L}.$$

Also $L(A) = \{t \in T(\Sigma) \mid [t]_{\equiv_L} \in F\} = L$.

Algebraisch ist A_L nichts anderes als die endliche Algebra $T(\Sigma)_{/\equiv_L}$ zusammen mit der Finalmenge L_{\equiv_L} .

Wir haben Bäume als Grundterme aus $T(\Sigma)$ erklärt und reguläre Sprachen nur für Mengen von solchen Grundtermen. Dies kann man mittels (4) auch liberaler handhaben und für beliebige Teilmengen der Trägermenge einer Σ -Algebren rationale Sprachen definieren.

Definition 3.8 Eine Teilmenge L der Trägermenge einer Σ -Algebra \mathcal{A} heißt *rational* (oder einfach auch *regulär*), falls eine endliche Σ -Algebra \mathcal{E} , eine Teilmenge $B \subseteq \mathcal{E}$ und ein Σ -Algebra-Homomorphismus h von \mathcal{A} nach \mathcal{E} existieren, so dass $L = h^{-1}(B)$ gilt. Das Paar (\mathcal{E}, h) heißt auch *Halbautomat* für \mathcal{A} und das Tripel (\mathcal{E}, h, B) auch *Automat* für \mathcal{A} . $L := h^{-1}(B)$ ist die von (\mathcal{E}, h, B) akzeptierte Sprache.

Wir haben Halbautomaten (ohne Zustände) über Σ mit endlichen Σ -Algebren identifiziert, und Automaten mit Paaren einer endlichen Algebra und einer finalen Teilmenge der Trägermenge. Damit ergibt sich ein kanonischer Begriff von Automaten-Homomorphismen:

Definition 3.9 Ein Automaten-Homomorphismus h von einem Automaten (A, F_A) nach den Automaten (B, F_B) ist ein Σ -Algebra-Homomorphismus h von den Halbautomaten A nach B , der die Finalmengen respektiert, d.h. für den

$$h(F_A) \subseteq F_B \text{ und } h(A - F_A) \subseteq B - F_B$$

gilt. Zwei Automaten heißen *isomorph*, falls ein bijektiver Σ -Algebra-Homomorphismus zwischen ihnen existiert.

Wir nennen einen Automaten wieder *minimal*, falls kein anderer Automat existiert, der mit weniger Zuständen die gleiche Sprache akzeptiert. Für den im Beispiel 3.4 definierten Automaten \hat{M}_L kann man leicht zeigen:

Theorem 7 Für jeden Automaten A , der L akzeptiert, existiert ein surjektiver Automaten-Homomorphismus von A nach A_L . A_L ist für eine reguläre Sprache L bis auf Isomorphie der einzige minimale Automat, der L akzeptiert.

Beweis. Sei $\mathcal{A} = (\mathcal{A}, \{f^A \mid f \in \Sigma\}, F \subseteq \mathcal{A})$ eine endliche Algebra mit $L = (eval^A)^{-1}(F)$. Dann definiert

$$h(s) := [t]_{\equiv_L} \text{ für ein } t \in T(\Sigma) \text{ mit } eval^A(t) = s$$

einen surjektiven Automaten-Homomorphismus $h_{/\equiv_L}^A$ von (\mathcal{A}, F) nach $A_L = (T(\Sigma)_{/\equiv_L}, L_{/\equiv_L})$. Ist \mathcal{A} minimal, so muss $h_{/\equiv_L}^A$ auch injektiv sein. ■

3.4 Pumping Lemma

Theorem 8 Zu jeder regulären Sprache $L \subseteq T(\Sigma)$ existiert eine Zahl n_L , so dass zu jedem Baum t in L einer Tiefe $> n_L$ zwei Kontexte C, C' und ein Baum $u \in T(\Sigma)$ existieren mit

- $t = C'(C(u))$,
- C ist nicht trivial, d.h. $|C| > 1$ und C besitzt eine Variable,
- $\forall n \in \mathbb{N}: C'(C^n(u)) \in L$, dabei ist $C^0(u) := u, C^{n+1} := C(C^n(u))$.

Beweis. Ist L regulär so gilt für eine endliche Σ -Algebra \mathcal{E} und Teilmenge B von \mathcal{E} gerade $L = \{t \mid \text{eval}^{\mathcal{E}}(t) \in B\}$. Es sei $n_L := |\mathcal{E}|$. Für $t \in L$ mit einer Tiefe $k > n_L$ wählen wir einen Ast w mit maximaler Tiefe k mit zugeordneter Folge C_1, \dots, C_k von Kontexten mit $t = C_k(\dots(C_i(b)\dots)$ für das Blatt a in w , vgl Lemma 2.1. Wegen $k > |\mathcal{E}|$ existieren zwei Zahlen i, j mit $1 \leq i < j \leq k$ und ein $e \in \mathcal{E}$ mit

$$\text{eval}^{\mathcal{E}}(C_i(\dots(C_1(a)\dots)) = \text{eval}^{\mathcal{E}}(C_j(\dots(C_1(a)\dots)) = e.$$

Für

$$C' := C_k(\dots(C_{j+1}(x)\dots), C := C_j(\dots(C_{i+1}(x)\dots), u := C_i(\dots(C_1(a)\dots))$$

gilt damit

$$t = C'(C(u)), \text{ und}$$

$$\text{eval}^{\mathcal{E}}(u) = e = C^{\mathcal{E}}(e) = C(C(e)) = (C^{\mathcal{E}})^n(e) \text{ für alle } n \in \mathbb{N}.$$

Also auch $\text{eval}^{\mathcal{E}}(C'(C^n(u))) = \text{eval}^{\mathcal{E}}(C'(C(u))) = \text{eval}^{\mathcal{E}}(t) \in B$. ■

3.5 Baumgrammatiken

Es existieren in der Literatur zahlreiche Konzepte für Graphgrammatiken. Wir beschäftigen uns hier nur mit dem wahrscheinlich einfachsten: reguläre Baumgrammatiken.

Definition 3.10 *Eine reguläre Baumgrammatik ist ein Tupel*

$$G = (V, T, s_0, R), \text{ von}$$

- einer endlichen Menge V von Variablen der Arität 0,
- einem endlichen gerangten Alphabet Σ von Terminalen mit $V \cap \Sigma = \emptyset$,
- einer Startvariablen $s_0 \in V$,
- einer endlichen Menge

$$R \subseteq V \times T(\Sigma, V)$$

von Regeln.

Für $(x, u) \in R$ schreibt man auch $x \rightarrow_R u$ oder nur $x \rightarrow u$. Es seien t, t' zwei Terme in $T(\Sigma, V)$, so heißt t' ein direkter Nachfolger von t , in Zeichen $t \vdash t'$, falls t' aus der simultanen indeterminierten Ersetzung aller Vorkommen aller Variablen $x \in V$ in t durch einen Term u_x mit $x \rightarrow_R u_x$ entsteht. Dabei dürfen zwei Vorkommen einer Variablen x durch zwei unterschiedliche Terme u_x, u'_x mit $x \rightarrow u_x$ und $x \rightarrow u'_x$ ersetzt werden.

$$L(G) := \{t \in T(\Sigma) \mid s_0 \vdash^* t\}$$

ist die von G generierte Sprache.

Gleichwertig für die Sprachgenerierung wäre ein "langsamerer" Rechnungsbegriff, in dem $t \vdash t'$ gilt, wenn in t genau ein Vorkommen einer Variablen x durch einen Tern u mit $x \rightarrow u$ ersetzt wird um t' zu erhalten. Eine Variable $x \in V$ ist nutzlos, falls kein Baum

$t \in T(\Sigma)$ existiert mit $x \vdash^* t$. Eine reguläre Baumgrammatik G heißt normiert, falls in ihr keine nutzlosen Variablen vorkommen und in jeder Regel $x \rightarrow u$ u ein Baum der Form $f^{(n)}(x_1, \dots, x_n)$ oder $a^{(0)}$ ist für $f, a \in \Sigma$ und $x_1, \dots, x_n \in V$. Zwei Grammatiken G_1, G_2 heißen äquivalent, wenn beide die gleiche Sprache erzeugen. Offensichtlich gilt das

Lemma 3.3 *Zu jeder regulären Baumgrammatik existiert eine äquivalente normierte.*

Theorem 9 *Eine Baumsprache ist genau dann regulär wenn sie von einer regulären Baumgrammatik erzeugt wird.*

Beweis. $L \subseteq T(\Sigma)$ sei regulär. Dann existiert ein indeterminierter **top-down** Baumautomat

$$A = (S, \Sigma, \Delta, I),$$

der L akzeptiert. Sei s_0 eine nullstellige Variable, die nicht in S vorkommt.

$$G := (S \cup \{s_0\}, \Sigma, s_0, R), \text{ mit}$$

$$R := \{s \rightarrow f^{(n)}(s_1, \dots, s_n) \mid (s, f) \Delta((s_1, \dots, s_n))\} \cup \{s_0 \rightarrow s \mid s \in I\}$$

generiert damit gerade L .

Wird umgekehrt L von einer regulären Grammatik

$$G = (V, \Sigma, s_o, R)$$

generiert, so können wir oE annehmen, dass G normiert ist. Der indeterminierte top-down Automat

$$A := (V, \Sigma, \Delta, \{s_o\}), \text{ mit}$$

$$(s, f^{(n)}) \Delta (s_1, \dots, s_n) :\Leftrightarrow x \rightarrow_R f(s_1, \dots, s_n)$$

für alle $f \in \Sigma, s_1, s \in V$ akzeptiert dann gerade L . ■

Normierte Grammatiken sind also nur eine andere Darstellungsform von indeterminierten top-down Baumautomaten.

3.6 Reguläre Ausdrücke für Bäume

Es sei Σ ein endliches, gerangtes Alphabet und X eine endliche Menge von Variablen der Arität 0 mit $\Sigma \cap X = \emptyset$. Wir betrachten jetzt Bäume und Sprachen über $\Sigma_X := \Sigma \cup X$.

Blatt-Substitution

Definition 3.11 *Es seien $L, L_1, \dots, L_k \subseteq T(\Sigma_X), t \in T(\Sigma_X), x_1, \dots, x_k \in X$, dann ist die Sprache $t(x_1/L_1, \dots, x_k/L_k)$ induktiv wie folgt definiert für $a \in \Sigma_0, a \notin \{x_1, \dots, x_k\}, f \in \Sigma_n, n > 0, s_i \in T(\Sigma_X)$:*

$$a(x_1/L_1, \dots, x_k/L_k) := \{a\},$$

$$x_i(x_1/L_1, \dots, x_k/L_k) := L_i,$$

$$f(s_1, \dots, s_n)(x_1/L_1, \dots, x_k/L_k) := \{f(t_1, \dots, t_n) \mid t_i \in s_i(x_1/L_1, \dots, x_k/L_k) \text{ für } 1 \leq i \leq n\}.$$

$$L(x_1/L_1, \dots, x_k/L_k) := \bigcup_{t \in L} t(x_1/L_1, \dots, x_k/L_k).$$

In $t(x/L)$ liegen somit alle Bäume, die man erhält indem alle Blätter x in t indeterminiert durch je einen Baum aus L ersetzt werden.

Theorem 10 *Sind L, L_1, \dots, L_k reguläre Baumsprachen, so auch $L(x_1/L_1, \dots, x_k/L_k)$.*

Beweis. Es seien L, L_i von den normierten Grammatiken $G = (V, \Sigma_X, s_0, R)$, $G_i = (V_i, \Sigma_X, s_i, R_i)$ generiert, wobei alle vorkommenden Mengen V, V_i paarweise disjunkt seien. R' entstehe aus R durch Ersetzen aller Vorkommen des terminalen Symbols x_i durch die i -te Startvariable s_i , d.h. durch Ersetzen der Regeln der Form $U \rightarrow x_i$ durch $U \rightarrow s_i$, für $1 \leq i \leq k$.

$$G' := (V \cup \bigcup_{1 \leq i \leq k} V_i, \Sigma_X, s_0, R' \cup \bigcup_{1 \leq i \leq k} R_i)$$

generiert dann $L(x_1/L_1, \dots, x_k/L_k)$. ■

Definition 3.12 *Für $L, M \subseteq T(\Sigma_X)$ ist die Konkatenation an der Stelle $x \in X$ definiert als*

$$L \circ_x M := L(x/M).$$

Die n -te Iteration nach $x \in X$ ist induktiv definiert als

$$L^{0,x} := \{x\},$$

$$L^{n+1,x} := L^{n,x} \circ_x L.$$

Die Iteration nach x ist

$$L^{*,x} := \bigcup_{n \in \mathbb{N}} L^{n,x}.$$

Lemma 3.4 *Sind L, M regulär, so auch $L \circ_x M, L^{n,x}$ und $L^{*,x}$.*

Beweis. Mit Satz 10 ist es für $L \circ_x M$, $L^{n,x}$ klar. $G = (V, \Sigma_X, s_0, R)$ generiere L . Fügt man zu jeder Regel $s \rightarrow_R x$ mit $s \in V$ noch die Regel $v \rightarrow s_0$ hinzu, so generiert man $L^{*,x}$. ■

$L^{*,x}$ ist wieder eine Sprache über Σ_X in der noch Blätter x vorkommen dürfen. Kommen in M allerdings keine Bäume mit Blättern x vor, dann auch in $L \circ_x M$ nicht.

Reguläre Ausdrücke

Definition 3.13 Die Menge der regulären Ausdrücke Reg_{Σ_X} ist die kleinste Menge mit

- $\Phi \in Reg_{\Sigma_X}$, $\Sigma_0 \cup X \subseteq Reg_{\Sigma_X}$,
- für $f \in \Sigma_n$, $E_1, \dots, E_n \in Reg_{\Sigma_X}$ gilt auch

$$f(E_1, \dots, E_n) \in Reg_{\Sigma_X},$$

- für $E, E_1, E_2 \in Reg_{\Sigma_X}$, $x \in X$ gilt auch

$$(E_1 + E_2), (E_1 \circ_x E_2), E^{*,x} \in Reg_{\Sigma_X}.$$

$\mathfrak{S} : Reg_{\Sigma_X} \rightarrow 2^{T(\Sigma)}$ interpretiert jeden regulären Ausdruck über Σ_X als eine Baumsprache über Σ_X wie folgt:

- $\mathfrak{S}(\Phi) := \emptyset$, $\mathfrak{S}(a) := \{a\}$, für $a \in \Sigma_0 \cup X$,
- $\mathfrak{S}(f(E_1, \dots, E_n)) := \{f(s_1, \dots, s_n) \mid s_i \in E_i \text{ für } 1 \leq i \leq n\}$,
- $\mathfrak{S}(E_1 + E_2) := \mathfrak{S}(E_1) \cup \mathfrak{S}(E_2)$, $\mathfrak{S}(E_1 \circ_x E_2) := \mathfrak{S}(E_1) \circ_x \mathfrak{S}(E_2)$, $\mathfrak{S}(E^{*,x}) := \mathfrak{S}(E)^{*,x}$.

Wir schreiben auch $E_1 = E_2$ falls $\mathfrak{S}(E_1) = \mathfrak{S}(E_2)$ gilt und nutzen die üblichen Klammerersparnisregeln, in denen $*$ stärker bindet als \circ , welches stärker als $+$ bindet. Die Konkatenation ist nicht assoziativ. So ist

$$(f(x, y) \circ_x h(y)) \circ_y a = f(h(a), a) \neq f(a, y) = f(x, y) \circ_x (h(y) \circ_y a).$$

Für festes x ist \circ_x allerdings assoziativ, da $t_1 \circ_x (t_2 \circ_x t_3) = (t_1 \circ_x t_2) \circ_x t_3$ gilt. Mit $t_1 \circ_{x_1} t_2 \circ_{x_2} \dots \circ_{x_{n_1}} t_n$ meinen wir stets die Linksklammerung $(\dots(t_1 \circ_{x_1} t_2) \circ_{x_2} \dots) \circ_{x_{n_1}} t_n$.

Example 3.5 Es sei $\Sigma := \{f^{(2)}, h^{(1)}, a^{(0)}, b^{(0)}\}$, dann beschreibt

$$E_1 := (f(x, x) + h(x))^{*,x} \circ_x (a + b)$$

gerade $T(\Sigma)$, und

$$E_2 := f(x, y) \circ_x h(x)^{*,x} \circ_y h(y)^{*,y} \circ_x a \circ_y b$$

beschreibt alle Bäume der Form $f(h^i(a), h^j(b))$.

Theorem 11 Eine Baumsprache ist genau dann regulär, wenn sie von einem regulären Ausdruck beschrieben wird.

Beweis. Es bleibt zu zeigen, dass für jede endliche Σ -Algebra \mathcal{E} und jedes $F \subseteq \mathcal{E}$ $L = (eval^{\mathcal{E}})^{-1}(F)$ durch einen regulären Ausdruck beschrieben wird. Wir werden L durch

einen regulären Ausdruck über $\Sigma_{\mathcal{E}}$ beschreiben, in dem auch Elemente aus \mathcal{E} vorkommen können.

$T(e, S, K)$ bezeichnet alle Bäume t in $T(\Sigma_K)$, deren Blätter in $\Sigma_0 \cup K$ liegen und die bei einer Evaluierung in \mathcal{E} zum Element $e \in \mathcal{E}$ führen und dabei in den echten Zwischenrechnungen nur Elemente aus S benutzen. Formal:

Für $e \in \mathcal{E}$, $S, K \subseteq \mathcal{E}$ ist

$$T(e, S, K) := \{t \in T(\Sigma_K) \mid \text{eval}^{\mathcal{E}}(t) = e \text{ und } \forall t' \in \text{Teilterm}(t) - (K \cup \{t\}) : \text{eval}^{\mathcal{E}}(t') \in S\}.$$

In $\text{Teilterm}(t) - (K \cup \{t\})$ liegen alle echten Unterbäume von t , die selbst kein Blatt in K sind.

Damit gilt

$$L = \bigcup_{e \in F} T(e, \mathcal{E}, \emptyset).$$

Wir zeigen, dass alle $T(e, S, K)$ bereits durch einen regulären Ausdruck über Σ_K beschrieben werden. Dies geschieht mittels Induktion über S :

$S = \emptyset$. In $T(e, \emptyset, K)$ liegen nur Bäume, deren echte Teilbäume, die nicht in K sind, zu "nichts" evaluieren. Also liegen nur Bäume t in $T(e, \emptyset, K)$, die keine echten Unterbäume besitzen außer Blätter in K . D.h. t liegt selbst in K oder hat die Form $f(a_1, \dots, a_n)$ für $f \in \Sigma_n$, $a_i \in K$, und evaluiert nach S . Damit ist $T(e, \emptyset, K)$ endlich und somit durch einen regulären Ausdruck über Σ_K beschreibbar.

$S' = S \cup \{e_0\}$, $e_0 \notin S$. Per Induktionsvoraussetzung gelte bereits für S und alle $e \in \mathcal{E}$, $K \subseteq \mathcal{E}$, dass $T(e, S, K)$ einen regulären Ausdruck $E(e, S, K)$ in $\text{Reg}(\Sigma_K)$ besitzt. Dann lässt sich $T(e, S', K)$ beschreiben durch

$$E(e, S', K) = E(e, S, K) + E(e, S, K \cup \{e_0\}) \circ_{e_0} E(e_0, S, K \cup \{e_0\})^{*, e_0} \circ_{e_0} E(e_0, S, K).$$

■

Example 3.6 Es seien $\Sigma := \{f^{(2)}, h^{(1)}, a^{(0)}, b^{(0)}\}$ und L_g die Sprache alle Bäume über Σ mit einer geraden Anzahl von Knoten. L_g wird von der endlichen Algebra

$$\mathcal{E} := (\{0, 1\}, f^{\mathcal{E}}(x, y) = x + y + 1(\text{mod } 2), h^{\mathcal{E}}(x) = x + 1(\text{mod } 2), a^{\mathcal{E}} = b^{\mathcal{E}} = 1)$$

mit $F = \{0\}$ erkannt.

Man rechnet leicht aus:

$$\begin{aligned} E(0, \emptyset, \emptyset) &= \Phi, \\ E(0, \emptyset, \{0\}) &= 0, \\ E(0, \emptyset, \{1\}) &= h(1), \\ E(0, \emptyset, \{0, 1\}) &= f(0, 1) + f(1, 0) + h(1) + 0, \\ E(1, \emptyset, \emptyset) &= a + b, \\ E(1, \emptyset, \{0\}) &= a + b + h(0) + f(0, 0), \\ E(1, \emptyset, \{1\}) &= a + b + f(1, 1) + 1, \\ E(1, \emptyset, \{0, 1\}) &= a + b + h(0) + f(0, 0) + f(1, 1) + 1. \end{aligned}$$

$$E(0, \{0\}, \emptyset) = E(0, \emptyset, \emptyset) + E(0, \emptyset, \{0\}) \circ_0 E(0, \emptyset, \{0\})^{*,0} \circ_0 E(0, \emptyset, \emptyset) = \Phi.$$

$$\begin{aligned} E(0, \{0\}, \{1\}) &= E(0, \emptyset, \{1\}) + E(0, \emptyset, \{0, 1\}) \circ_0 E(0, \emptyset, \{0, 1\})^{*,0} \circ_0 E(0, \emptyset, \{1\}) = \\ &= h(1) + (f(0, 1) + f(1, 0) + h(1) + 0) \circ_0 (f(0, 1) + f(1, 0) + h(1) + 0)^{*,0} \circ_0 h(1) = \\ &= (f(0, 1) + f(1, 0))^{*,0} \circ_0 h(1). \end{aligned}$$

$$\begin{aligned} E(1, \{0\}, \emptyset) &= E(1, \emptyset, \emptyset) + E(1, \emptyset, \{0\}) \circ_0 (E(0, \emptyset, \{0\})^{*,0} \circ_0 E(0, \emptyset, \emptyset) = \\ &= a + b + (a + b + h(0) + f(0, 0)) \circ_0 0^{*,0} \circ_0 \Phi = a + b. \end{aligned}$$

$$\begin{aligned} E(1, \{0\}, \{1\}) &= E(1, \emptyset, \{1\}) + E(1, \emptyset, \{0, 1\}) \circ_0 E(0, \emptyset, \{0, 1\})^{*,0} \circ_0 E(0, \emptyset, \{1\}) = \\ &= a + b + f(1, 1) + 1 + (a + b + h(0) + f(0, 0) + f(1, 1) + 1) \circ_0 (f(0, 1) + f(1, 0) + h(1) + 0)^{*,0} \circ_0 \\ &= h(1) = (h(0) + f(0, 0)) \circ_0 (f(0, 1) + f(1, 0))^{*,0} \circ_0 h(1) + a + b + f(1, 1) + 1. \end{aligned}$$

Damit berechnet sich ein regulärer Ausdruck E_g für L_g als

$$\begin{aligned} E_g &= E(0, \{0, 1\}, \emptyset) = E(0, \{0\}, \emptyset) + E(0, \{0\}, \{1\}) \circ_1 E(1, \{0\}, \{1\})^{*,1} \circ_1 E(1, \{0\}, \emptyset) = \\ &= \Phi + \left((f(0, 1) + f(1, 0))^{*,0} \circ_0 h(1) \right) \circ_1 \\ &= \left((h(0) + f(0, 0)) \circ_0 (f(0, 1) + f(1, 0))^{*,0} \circ_0 h(1) + a + b + f(1, 1) + 1 \right)^{*,1} \circ_1 (a + b) = \\ &= \left((f(0, 1) + f(1, 0))^{*,0} \circ_0 h(1) \right) \circ_1 \left((h(0) + f(0, 0)) \circ_0 (f(0, 1) + f(1, 0))^{*,0} \circ_0 h(1) + f(1, 1) \right)^{*,1} \\ &= G \circ_1 U^{*,1} \circ_1 (a + b). \end{aligned}$$

In $(f(0, 1) + f(1, 0))^{*,0}$ liegen nur Bäume über $f, 0, 1$ einer ungeraden Anzahl von Knoten, in denen genau ein Blatt den Wert 0 trägt. Dieses eine Blatt wird durch $\circ_0 h(1)$ durch den Baum $h(1)$ mit zwei Knoten ersetzt. Also liegen in

$$G := (f(0, 1) + f(1, 0))^{*,0} \circ_0 h(1)$$

nur Bäume mit einer geraden Anzahl von Knoten. In

$$U := (h(0) + f(0, 0)) \circ_0 (f(0, 1) + f(1, 0))^{*,0} \circ_0 h(1) + f(1, 1)$$

ebenso wie $U^{*,1}$ liegen hingegen nur Bäume einer ungeraden Anzahl von Knoten. Zu beachten ist, dass die Konkatenation ungerader Bäume ungerade bleibt. Daher liegen auch in $U^{*,1} \circ (a + b)$ nur Bäume mit ungerade vielen Knoten. Wird hingegen ein Baum t einer geraden Anzahl von Knoten mit einem Baum t' einer ungeraden Anzahl zu $t \circ t'$ konkateniert, so bleibt die Zahl der Knoten gerade. Konkateniert man an einen Baum t einen geraden Bäume t' , so kann das Ergebnis $t \circ t'$ gerade oder ungerade sein.

4 Ungerangte Bäume

4.1 Theorien

Definition 4.1 Eine Theorie (oder Algebraische Spezifikation) $\mathcal{T} = (S, Op, \mathcal{X}, E)$ ist eine Signatur $\mathcal{S} = (S, Op)$ mit Variablen \mathcal{X} und einer Menge E von Gleichungen der Form

$$t = t'$$

mit $t, t' \in T(\mathcal{S}, \mathcal{X})$ gleicher Sorte.

Eine \mathcal{T} -Algebra \mathcal{A} ist eine \mathcal{S} -Algebra, in der für jede Gleichung $t = t' \in E$ und Belegung $\sigma : \mathcal{X} \rightarrow \mathcal{A}$ gilt:

$$eval_{\sigma}^{\mathcal{A}}(t) = eval_{\sigma}^{\mathcal{A}}(t').$$

Es sei Σ eine S -sortierte Familie $\Sigma = (\Sigma_s)_{s \in S}$ von nullstelligen Symbolen (Generatoren) $a \in \Sigma_s$ der Sorte s , die wir den Grundtermen in $T(\mathcal{S})$ hinzufügen wollen. $T(\mathcal{S}, \Sigma)$ ist ja die Menge aller Terme über $Op \cup \Sigma$ (ohne Variable in \mathcal{X}).

Definition 4.2

$$\mathcal{F}(\mathcal{T}, \Sigma) = T(\mathcal{S}, \Sigma) / \equiv_E$$

heißt die freie \mathcal{T} -Termalgebra generiert von Σ . Dabei ist \equiv_E die von E erzeugte Kongruenz auf $T(\mathcal{S}, \Sigma)$, definiert durch:

$\forall n, i \in \mathbb{N} : \forall f^{(n)} \in Op : \forall t, t_i, t'_i \in T(\mathcal{S}, \Sigma) :$

1) $t \equiv_E t, t_1 \equiv_E t_2 \implies t_2 \equiv_E t_1, (t_1 \equiv_E t_2 \wedge t_2 \equiv_E t_3) \implies t_1 \equiv_E t_3,$

2) $t_i \equiv_E t'_i \implies f(t_1, \dots, t_n) \equiv_E f(t'_1, \dots, t'_n),$

3) $\forall t = t' \in E : \forall \text{Belegung } \sigma : \mathcal{X} \rightarrow T(\mathcal{S}, \Sigma) : eval_{\sigma}(t) \equiv_E eval_{\sigma}(t').$

1) erzwingt eine Äquivalenzrelation, 2) eine Kongruenz, und 3) das Erfüllen der Gleichungen in E .

Example 4.1 Die Theorie \mathcal{T}_w^1 aller Wörter über Σ (ein 1-sortiges Alphabet der Sorte w) ist gegeben durch

$S = \{w\}, Op = \{\circ^{(2)}, \varepsilon^{(0)}\}, \mathcal{S}_w^1 = (S, Op), \mathcal{X} = \{u, v, w\},$

$E = \{\circ(u, \circ(v, w)) = \circ(\circ(u, v), w), \circ(\varepsilon, u) = u, \circ(u, \varepsilon) = u\}.$

Die Theorie \mathcal{T}_w^2 aller Wörter über Σ (ein 1-sortiges Alphabet der Sorte l) ist gegeben durch $S' = \{l, w\}, Op' = \{\circ_{l \times w \rightarrow w}, \varepsilon_{\rightarrow w}\}, \mathcal{S}_w^2 = (S', Op').$

Damit gilt

$$\mathcal{F}(\mathcal{T}_w^1, \Sigma_w^*) = T(\mathcal{S}_w^1, \Sigma_w^*) / \equiv_E = \mathcal{F}(\mathcal{T}_w^2, (\Sigma_l, \emptyset_w)) = T(\mathcal{S}_w^2, (\Sigma_l, \emptyset_w)).$$

Hierbei bedeutet Σ_l (Σ_w) nur, dass wir die Elemente in Σ als Elemente der Sorte l (w) auffassen.

4.2 \mathcal{T} -Automaten

Im Folgenden sei eine Signatur \mathcal{S} mit Variablen oder Generatoren oder Gleichungen stets endlich. D.h. wir haben nur endlich viele Sorten, Operatorensymbole, Variable, Generatoren und Gleichungen. Es sei \mathcal{E} eine endliche \mathcal{S} -Algebra und $h : T(\mathcal{S}, \Sigma) \rightarrow \mathcal{E}$ ein \mathcal{S} -Algebra-Homomorphismus. Damit ist für $B \subseteq \mathcal{E}_s$ $L := h^{-1}(B) \subseteq T_s(\mathcal{S}, \Sigma)$ eine rationale Sprache der Sorte s . Dies kann man auch wie folgt als Automat charakterisieren:

Definition 4.3 *Ein \mathcal{S} -Halbautomat A ist ein Tupel*

$$A = (\mathcal{S}, \Sigma, Q, \delta) \text{ von}$$

- einer endlichen Signatur $\mathcal{S} = (S, Op)$,
- einem endlichen S -sortierten Alphabet Σ ,
- einer endlichen S -sortierten Familie Q von Zuständen, und
- einer Abbildung δ , die jedem
- - $a \in \Sigma_s$ ein Element $\delta(a) \in Q_s$, und
- - Operatorsymbol $f_{s_1 \times \dots \times s_n \rightarrow s}$ eine Abbildung

$$\delta(f) : Q_{s_1} \times \dots \times Q_{s_n} \rightarrow Q_s$$

zuordnet.

Ein \mathcal{S} -Automat ist ein \mathcal{S} -Halbautomat mit einer Menge $F \subseteq Q_s$ von finalen Zuständen einer Sorte $s \in S$.

\mathcal{S} -(Halb-)Automaten über Σ entsprechen endlichen \mathcal{S} -Algebren \mathcal{E} (mit einer ausgezeichneten Menge $F \subseteq \mathcal{E}_s$) und einem Homomorphismus $h : T(\mathcal{S}, \Sigma) \rightarrow \mathcal{E}$:

Zu A gehört die endliche \mathcal{S} -Algebra $\mathcal{E}_A = (Q, Op^{\mathcal{E}})$ mit $Op^{\mathcal{E}} = \{\delta(f) | f \in Op\}$ und $h_A(a) = \delta(a)$ für $a \in \Sigma$ generiert $h : T(\mathcal{S}, \Sigma) \rightarrow \mathcal{E}_A$.

Zu einer endlichen \mathcal{S} -Algebra \mathcal{E} und $h : T(\mathcal{S}, \Sigma) \rightarrow \mathcal{E}$ gehört der \mathcal{S} -Automat $A_{\mathcal{E}} = (\mathcal{S}, \Sigma, \mathcal{E}, \delta)$ mit $\delta(f) = f^{\mathcal{E}}$ für $f \in Op$ und $\delta(a) = h(a)$ für $a \in \Sigma$.

Definition 4.4 *Es sei $\mathcal{T} = (\mathcal{S}, \Sigma, \mathcal{X}, E)$ eine endliche Theorie. Ein \mathcal{T} -(Halb-)Automat A ist ein \mathcal{S} -(Halb-)Automat A , dessen assoziierte \mathcal{S} -Algebra eine \mathcal{T} -Algebra ist.*

Damit erfüllen alle Funktionen $\delta(f)$ für $f \in Op$ die Gleichungen in E .

Lemma 4.1 *Zu jedem \mathcal{T} -Halbautomaten existiert genau ein \mathcal{T} -Algebra-Homomorphismus k , so dass das folgende Diagramm kommutiert.*

$$\begin{array}{ccc}
 T(\mathcal{S}, \Sigma) & \xrightarrow{h_{\equiv_E}} & T(\mathcal{S}, \Sigma) / \equiv_E = \mathcal{F}(\mathcal{T}, \Sigma) \\
 & \searrow h_A & \swarrow k \\
 & & Q
 \end{array}$$

Damit ist für $F \subseteq \mathcal{E}_s = Q_s$

$$L_{\mathcal{S}}(A, F) = h_A^{-1}(F) \subseteq T_s(\mathcal{S}, \Sigma) \text{ mit } A \text{ aufgefasst als } \mathcal{S}\text{-Automat, oder}$$

$$L_{\mathcal{T}}(A, F) = k^{-1}(F) \subseteq \mathcal{F}_s(\mathcal{T}, \Sigma) \text{ mit } A \text{ aufgefasst als } \mathcal{T}\text{-Automat, und es gilt}$$

$$L_{\mathcal{T}}(A, F) = h_{\equiv_E}(L_{\mathcal{S}}(A, F)) = L_{\mathcal{S}}(A, F)/_{\equiv_E}.$$

Eine \mathcal{T} -Sprache $L \subseteq \mathcal{F}_s(\mathcal{T}, \Sigma)$ über Σ heißt regulär oder rational, wenn sie von einer endlichen \mathcal{T} -Algebra Q oder einem endlichen \mathcal{T} -Automaten als $L = L_{\mathcal{T}}(A, F)$ für ein $F \subseteq Q_s$ erkannt wird. Natürlich lässt sich L dann auch über \mathcal{T} -Algebra-Kongruenzen eines endlichen Index charakterisieren: Setzen wir

$$t \tau t' :\Leftrightarrow k(t) = k(t')$$

für k aus Lemma 4.1 und $t, t' \in \mathcal{F}(\mathcal{T}, \Sigma)$, so wird τ eine \mathcal{T} -Algebra Kongruenz von endlichem Index, die L saturiert (dann wird L oft als erkennbar bezeichnet).

Das ist der große Vorteil eines Vorgehens über \mathcal{T} -Algebren:

Wird eine Sprache von einer endlichen \mathcal{T} -Algebra erkannt, dann wird sie auch von endlichen (\mathcal{T} -) Automaten akzeptiert und von einer (\mathcal{T} -)Kongruenz endlichen Index' saturiert. D.h., die Begriffe "erkennbar", "akzeptierbar" und "rational" fallen automatisch zusammen.

4.3 Die Theorie ungerangter, ungeordneter Bäume

Definition 4.5 Die Theorie ungerangter, ungeordneter Bäume \mathcal{T}_{uu} ist gegeben durch

$S_{uu} = \{l, t, f\}$ für letter, tree, forest,

$Op_{uu} = \{pl \times f \rightarrow t, r t \rightarrow f, +_f \times f \rightarrow f, \theta \rightarrow f\}$, $\mathcal{S}_{uu} = (S_{uu}, Op_{uu})$.

$X_{uu} = \{x, y, z\}$ der Sorte f ,

$E_{uu} =$

$$(x + y) + z = x + (y + z),$$

$$x + y = y + x,$$

$$x + \theta = x.$$

Ferner sei $\equiv_{uu} := \equiv_{E_{uu}}$.

In der kanonischen mengentheoretischen Interpretation von Termen aus $T(\mathcal{S}_{uu}, \Sigma)$ für ein endliches Alphabet Σ nullstelliger Symbole der Sorte l als abstrakter Graph bedeutet

- θ der leere Wald $[(\emptyset, \emptyset, \emptyset)]_{\sim_{iso}}$,

- $p(a, f)$ der Baum, der entsteht, wenn man eine neue Wurzel mit Label a als Vater aller Wurzeln der Bäume des Waldes f wählt,

- $r(t)$ der Wald, der aus genau dem einen Baum t besteht,

- $f_1 + f_2$ der Wald, der aus der Vereinigung der Wälder f_1 und f_2 (aufgefasst als Multi-Mengen) entsteht.

$\mathcal{F}(\mathcal{T}_{uu}, \Sigma) := \mathcal{F}(\mathcal{T}_{uu}, (\Sigma_l, \emptyset_t, \emptyset_f))$ ist die freie \mathcal{T}_{uu} -Algebra $T(\mathcal{S}_{uu}, \Sigma)/\equiv_{uu}$ generiert von Σ . Wir definieren nun die Mengen $Tree_{uu}(\Sigma)$ ($Forest_{uu}(\Sigma)$) aller ungerangten, ungeordneten Bäume (Wälder) über Σ als

$$Tree_{uu}(\Sigma) := \mathcal{F}_t(\mathcal{T}_{uu}, \Sigma) \text{ und } Forest_{uu}(\Sigma) := \mathcal{F}_f(\mathcal{T}_{uu}, \Sigma).$$

Es genügen also Generatoren aus Σ der Sorte l um alle uu -Bäume und -Wälder über Σ darzustellen.

Example 4.2 *Der ungeordnete und ungerangte Baum α über $\{a, b\}$ aus Figur 2 kann durch die Terme*

$$t_1 = p(a, (r(p(a, r(p(c, (r(p(a, \theta))) + r(p(b, \theta)))))) + r(p(b, \theta)) + r(p(a, \theta))), \text{ oder}$$

$$t_2 = p(a, (r(p(a, \theta)) + r(p(b, \theta))) + r(p(a, r(p(c, r(p(b, \theta)) + r(p(a, \theta))))))$$

”beschrieben” werden. Kürzen wir einmal $p(a, \theta)$ durch a , $p(a, f)$ durch af ab und lassen r weg, so sieht man die Struktur von t_1, t_2 etwas klarer:

$$t_1 \simeq a(a(c(a + b) + b + a), t_2 \simeq a(a + b + a(c(b + a))).$$

Damit gilt

$$t_1 \neq t_2, \text{ aber } t_1 \equiv_{uu} t_2.$$

Damit wird α korrekt mit $[t_1]_{\equiv_{uu}}$ aus $Tree_{uu}(\{a, b\})$ dargestellt.

Wie sehen \mathcal{T}_{uu} -Automaten aus? Laut Definition 4.4 ist ein \mathcal{T}_{uu} -Automat A ein Tupel

$$A = (\mathcal{S}_{uu}, (\Sigma_l, \emptyset_t, \emptyset_f), (Q_l, Q_t, Q_f), \delta, F),$$

wobei δ die Gleichungen E_{uu} erfüllen muss. D.h. für alle $q, q', q'' \in Q_f$ muss gelten

$$\delta(+)(q, q') = \delta(+)(q', q),$$

$$\delta(+)(\delta(+)(q, q'), q'') = \delta(+)(q, \delta(+)(q', q'')),$$

$$\delta(+)(q, 1) = q,$$

für ein ausgezeichnetes Element $1 (= \delta(\theta) (= h_A(\theta)))$ in Q_f . Für $F \subseteq Q_t$ ist $L(A)$ eine Baum-Sprache, für $F \subseteq Q_f$ ist $L(A)$ eine Wald-Sprache. Der Fall $F \subseteq Q_l$ ist uninteressant. Es sei jetzt $F \subseteq Q_t$, d.h. wir untersuchen uu -Baumautomaten.

Allerdings sehen \mathcal{T}_{uu} -Automaten nicht ganz so aus, wie die uns gewohnten Automaten, in denen ja nicht Zustände unterschiedlicher Sorten vorkommen. Dennoch können wir mit einer einfachen Transformation aus \mathcal{T}_{uu} -Automaten äquivalente, uns eher vertraute Automaten erhalten.

Definition 4.6 *Es sei M eine Menge. Eine Funktion $f : M \times M \rightarrow M$ heißt*

- kommutativ, falls $f(x, y) = f(y, x)$, und
- assoziativ, falls $f(x, f(y, z)) = f(f(x, y), z)$ für alle $x, y, z \in M$ gilt.

Eine kommutative und assoziative Funktion f kann zu einer Funktion auf nicht-leeren Multi-Mengen über M

$$f : (\mathbb{N}^M - 0) \rightarrow M \text{ durch}$$

$$f(x) := x, f(m + x) := f(f(m), x) \text{ für } x \in M, m \in \mathbb{N}^M$$

verallgemeinert werden.

Definition 4.7 Ein uu -Baumautomat ist ein Tupel

$$B = (Q, \Sigma, \delta_I, \delta, \delta_i, F)$$

von endlichen Mengen Q, F, Σ mit $F \subseteq Q$ und drei totalen Abbildungen

$$\delta_I : \Sigma \rightarrow Q,$$

$$\delta : Q \times \Sigma \rightarrow Q,$$

$$\delta_i : Q \times Q \rightarrow Q,$$

wobei δ_i kommutativ und assoziativ ist.

Das Verhalten von B ist eine Abbildung

$$\delta_B^* : \mathcal{F}(\mathcal{T}_{uu}, \Sigma) \rightarrow Q,$$

definiert als

- $\delta^*(t) := \delta_I(a)$ für einen Baum $t = [p(a, \theta)]_{\equiv uu}$, der nur aus einem Blatt a besteht,

- $\delta^*(t) := \delta(\delta_i(\delta^*(t_1) + \dots + \delta^*(t_k)), a)$, für $t = [p(a, r(t_1) + \dots + r(t_k))]_{\equiv uu}$.

$L(B) := \{t \in \mathcal{F}(\mathcal{T}_{uu}, \Sigma) \mid \delta^*(t) \in F\}$ ist die von B akzeptierte Sprache.

Damit ist B ein bottom-up Baumautomat. δ_I regelt die initialen Zustände an den Blättern, δ_i die Zusammenfassung von Zuständen der Söhne, und δ die Übergabe an den Vaterknoten. Wie hängen nun uu -Baumautomaten B und \mathcal{T}_{uu} -Automaten A zusammen?

Es sei $A = (\mathcal{S}_{uu}, (\Sigma_l, \emptyset_t, \emptyset_f), (Q_l, Q_t, Q_f), \delta_A, F)$ ein \mathcal{T}_{uu} -Automat. OE sei $\delta_A(r) : Q_t \rightarrow Q_f$ injektiv. (Sonst ersetze man Q_f durch $Q_t \times Q_f$ und $\delta_A(r)$ durch $\delta'_A(r)$ mit $\delta'_A(r)(q) := (q, \delta_A(r)(q))$.) Damit definieren wir den uu -Baumautomaten B als

$$B := (\Sigma, Q_f, \delta_I, \delta_B, \delta_i, F'), \text{ mit}$$

$$\delta_I(a) := \delta_A(r)(\delta(p)(a, \delta(\theta))),$$

$$\delta_B(q, a) := \delta_A(r)(\delta(p(a, q))),$$

$$\delta_i(q, q') := \delta_A(+)(q, q'), \text{ und}$$

$$F' := \delta(r)(F).$$

Da $\delta_A(r)$ injektiv ist gilt $eval^A(t) \in F \Leftrightarrow \delta_A(r)(eval^A(t)) \in \delta_A(F) = F'$. Damit gilt $L(A) = L(B)$.

Umgekehrt sei $B = (\Sigma, Q_f, \delta_l, \delta_B, \delta_i, F)$ ein uu -Baumautomat. Dann konstruieren wir dazu einen äquivalenten \mathcal{T}_{uu} -Automaten

$$A = (\mathcal{S}_{uu}, (\Sigma_l, \emptyset_t, \emptyset_f), (Q_l, Q_t, Q_f), \delta_A, F), \text{ mit}$$

$$Q_f := Q, Q_t := Q, \text{ und } Q_l := \Sigma,$$

$$\delta_A(p)(a, q) := \delta_B(q, a),$$

$$\delta_A(r) := id,$$

$$\delta_A(+) := \delta_i, \text{ und}$$

$$\delta_A(\theta) := 1,$$

wobei 1 das neutrale Element in Q_f für $\delta(+)$ ist, um das Q_f nötigenfalls angereichert wird. Damit gilt wieder $L(A) = L(B)$.

(Hier wird der Aufbau von Bäumen, ohne die Sorte t selbst zu verwenden, nachgespielt wird. Sei jetzt $F \subseteq Q_f$, so untersuchen wir uu -Waldsprachen. Als uu -Waldautomat sollte man jetzt ein Tupel

$$B = (Q, \Sigma, \delta, \delta_i, s_0, F)$$

wählen können, wobei Q, Σ, F, δ und δ_i wie in uu -Baumautomaten gewählt sind und s_0 der Startzustand aus Q ist, der gleichzeitig neutrales Element bzgl δ_i ist. Dies spiegelt den Aufbau von Wäldern ohne Verwendung der Sorte t und die Korrespondenz

- $\theta \rightarrow s_0$,
- $f_1 + f_2 \rightarrow \delta_i : Q \times Q \rightarrow Q$, und
- $r(p(a, f)) \rightarrow \delta : Q \times \Sigma \rightarrow Q$ wider.

Übungsaufgabe 1: Kann man so uu -Waldautomaten erhalten, die zu \mathcal{T}_{uu} -Automaten gleichwertig sind?

Übungsaufgabe 2: Was ist mit indeterminierten uu -Baumautomaten? Sind sie zu determinierten äquivalent?

Übungsaufgabe 3: Wie sehen indeterminierte top-down uu -Baumautomaten aus? Sind sie zu indeterminierten bottom-up uu -Baumautomaten äquivalent?

Kleinere Studienarbeit: Wie sehen reguläre Kleene Ausdrücke für \mathcal{T}_{uu} -Sprachen aus?)

Natürlich kann man häufig die gleichen Objekte mittels unterschiedlicher Theorien $\mathcal{T}^1, \mathcal{T}^2$ beschreiben. Dabei kann es vorkommen, dass \mathcal{T}^1 -Automaten andere Sprachen erkennen als \mathcal{T}^2 -Automaten.

Example 4.3 *Es sei \mathcal{T}_{uu}^2 die Theorie $S_{uu}, Op_{uu}^2, E_{uu}$ mit S_{uu}, E_{uu} wie in \mathcal{T}_{uu} aber*

$$Op_{uu}^2 = \{p'_{f \times l \rightarrow t}, r_{t \rightarrow f}, +_{f \times f \rightarrow f}, \theta_{\rightarrow f}\}.$$

Wir interpretieren nun $p'(f, a)$ in \mathcal{T}_{uu}^2 genau wie $p(a, f)$ in \mathcal{T}_{uu} .

Der Baum α über $\{a, b\}$ aus Figur 2 wird nun statt durch t_1 wie in Beispiel 4.2 durch

$$t'_1 = p'((r(p'(r(p'((r(p'(\theta, a)) + r(p'(\theta, b))), c)))) + r(p'(\theta, b))) + r(p'(\theta, a)), a)$$

beschrieben. Da aber formal die Wurzel in der neuen Signatur $\mathcal{S}_{uu}^2 = (S_{uu}, Op_{uu}^2)$ am Ende stehen - in \mathcal{S}_{uu} könnte man die Bäume also mit den Wurzeln unten und den Pfeilen nach oben zeichnen - werden bei der Evaluierung zuerst die Wurzeln betrachtet. Ein \mathcal{T}_{uu}^2 -Automat ist also ein determinierter root-to-frontier Automat mit deutlich eingeschränkteren Akzeptanzmöglichkeiten als die determinierten frontier-to-root \mathcal{T}_{uu} -Automaten.

4.4 Die Theorie ungerangter, angeordneter Bäume

Definition 4.8 Die Theorie ungerangter, angeordneter Bäume \mathcal{T}_{ua} ist gegeben durch

$S_{ua} = \{l, t, h\}$ für letter, tree, hedge,

$Op_{ua} = \{pl \times h \rightarrow h, r_{t \rightarrow h}, \circ_{h \times h \rightarrow h}, \theta_{\rightarrow h}\}$, $\mathcal{S}_{ua} = (S_{ua}, Op_{ua})$.

$X_{ua} = \{x, y, z\}$,

$E_{ua} =$

$$(x \circ y) \circ z = x \circ (y \circ z),$$

$$x \circ \theta = x.$$

Ferner sei $\equiv_{ua} := \equiv_{E_{ua}}$.

Eine *Hecke* ist eine angeordnete Familie von abstrakten Bäumen. In der kanonischen mengentheoretischen Interpretation von Termen aus $T(\mathcal{S}_{ua}, \Sigma)$ für ein endliches Alphabet Σ nullstelliger Symbole der Sorte l als abstrakter Graph bedeutet

- θ die leere Hecke $[(\emptyset, \emptyset, \emptyset)]_{\sim_{iso}}$,

- $p(a, h)$ der angeordnete Baum, der entsteht, wenn man eine neue Wurzel mit Label a als Vater aller angeordneten Wurzeln der Bäume in der angeordneten Hecke h wählt,

- $r(t)$ die Hecke, die aus genau dem einen Baum t besteht,

- $h_1 \circ h_2$ die Hecke, die aus der angeordneten Konkatenation der Hecke h_1 mit der Hecke h_2 entsteht.

Die Signaturen \mathcal{S}_{uu} und \mathcal{S}_{ua} sind letztlich gleich, da der Name der Sorte f nur durch den Namen h und der Name des Operatorensymbols $+$ nur durch \circ ersetzt wurde. Es unterscheiden sich nur die Gleichungen E_{uu} und E_{ua} um die Forderung nach Kommutativität der Operation $+$ im Gegensatz zu \circ .

Obwohl damit die Syntax ungerangter, ungeordneter und angeordneter Bäume gleich ist (bis auf Wahl der Namen $+$ statt \circ und f statt h) unterscheiden sich $\mathcal{F}(\mathcal{T}_{uu}, \Sigma)$ und

$$\mathcal{F}(\mathcal{T}_{ua}, \Sigma) = T(\mathcal{S}_{ua}, \Sigma) / \equiv_{ua} = T(\mathcal{S}_{uu}, \Sigma) / \equiv_{ua}$$

erheblich. In $\mathcal{F}(\mathcal{T}_{ua}, \Sigma)$ kann man jedem Element $[t]_{\equiv_{ua}}$ einen eindeutigen Repräsentanten t_0 zuordnen, indem man t durch sukzessive Anwendung der folgenden Termersetzungsregeln in eine Normalform umwandelt:

$$u \circ \theta \rightarrow u,$$

$$\theta \circ u \rightarrow u,$$

$$u \circ (v \circ w) \rightarrow (u \circ v) \circ w.$$

Dies ist in $\mathcal{F}(\mathcal{T}_{uu}, \Sigma)$ wegen der Gleichung $u+v = v+u$ nicht möglich. Ein ausgezeichneter Repräsentant in $[t]_{=uu}$ existiert nicht.

Dennoch übertragen sich die automatentheoretischen Überlegungen für uu -Bäume 1-zu-1 auf ua -Bäume. Der natürliche Unterschied ist, dass in einem ua -Baumautomaten $B = (Q, \Sigma, \delta_I, \delta, \delta_i, F)$ die Funktion δ_i nur noch assoziativ, aber nicht mehr notwendig kommutativ, sein muss.

5 Finite Automata on Unranked and Unordered DAGs

Finite automata that operate on finite or infinite words and (ordered and ranked) trees and various equivalent concepts to regularity are known. Those are recognizability with congruences, rationality with magmas, expressibility with regular expressions, definability in certain classes of monadic second order logic, generation by certain right-linear grammars. A good overview on the tree results is given in the TATA book [2]. However, only ranked and ordered trees are considered there. Note, a graph is ranked if the degree of any node is determined by its label, leading to ranked alphabets. This is not the case in unranked graphs and the degree in an infinite set of such unranked graphs may be unbounded. Thus, a finite automaton operating on unranked graphs has to operate on nodes with an unknown number of incoming and/or outgoing arcs (let us call this the "problem of unbounded degree"). An unpublished but well-known report [3] of Brüggemann-Klein, Murata and Wood researches ordered, unranked trees in some detail. Unranked, ordered and unordered trees have been investigated as an algebra by Courcelle in [4]. He presents a very elegant characterization of acceptability by $T_{u,u}$ -magmas and frontier-to-root $T_{u,u}$ -automata. Unranked and unordered trees with arc labels instead of node labels allow for a simpler algebraic approach and are found in Boneva and Talbot [5]. Brüggemann-Klein, Murata, Wood, and also Boneva, Talbot, solve the problem of unbounded degree for their unranked trees by allowing infinite but regular sets of transitions for their automata while Courcelle uses an associative and commutative transition function on pairs of state that easily extends to unbounded multisets of states.

Some generalizations of automata to ranked graphs or to their sub-class of ranked directed acyclic graphs are known: Finite graph automata have been introduced by Thomas [6], automata over planar dags by Kamimura and Slutzki [7]. A Kleene theorem for planar dags has been presented by Bossut, Dauchet and Warin [8]. They describe planar dags by linear expression that follow a graphic lay-out and use seriell and parallel composition. Graph expressions have also been introduced by Courcelle [9] for hyper-graphs to define context-free graph grammars. Charatonik [10] has researched automata on t-dags where no isomorphic sub-trees are allowed.

However, there exists no satisfying concept of automata on unrestricted unranked, unordered graphs. Kaminski and Pinter [11] avoid the problem of unbounded degree as their automata define a bound on the degree of acceptable graphs. However, the language of all graphs over a fixed alphabet now is not accepted any more. Fanchon and Morin [12] define regular pomsets languages over unranked alphabets with auto-concurrency via congruences of finite index. Those congruences mirror a serial-parallel composition of

pomsets. They receive a concept of regularity that is closed under union but not under intersection or complement.

We will follow Courcelle’s approach towards automata - but without using algebras as a semantics. We introduce linear dag expressions that resemble $T_{u,u}$ as a syntax and give a set-theoretical semantics as graphs. Finite automata operating as well on (congruence classes of) dag expressions as on abstract dags are introduced. In contrast to trees, dags possess incoming and outgoing arcs and all problems of root-to-frontiers and of frontiers-to-root automata must appear in dags. It is known that on trees deterministic root-to-frontier automata are a proper sub-class of non-deterministic ones which are equivalent to deterministic or non-deterministic frontier-to-root automata. As a consequence, the ‘root-to-frontier’ part of dag automata should be non-deterministic. Our dag automata will therefore contain aspects of deterministic frontier-to-root and non-deterministic root-to-frontier automata.

5.1 Graphs and DAGs

Set-Theoretic Approach

A set-theoretical approach to graphs is simple:

A *graph* γ over an alphabet Σ is a triple $\gamma = (N, E, \lambda)$ of two finite sets N of *nodes* and $E \subseteq N \times N$ of *edges* and a labelling mapping $\lambda : N \rightarrow \Sigma$. Two graphs $\gamma_i = (N_i, E_i, \lambda_i)$ are *isomorphic*, $\gamma_1 \sim_{iso} \gamma_2$, if there exists a bijective function $h : N_1 \rightarrow N_2$ with $(v, v') \in E_1 \Leftrightarrow (h(v), h(v')) \in E_2$ and $\lambda_2(h(v)) = \lambda_1(v)$ holds for all v, v' in N_1 .

Thus, graphs in this paper are directed, unranked, unordered, finite and node labelled. We use the following rather standard notations.

$\bullet v := \{v' \in V \mid (v', v) \in E\}$, $v \bullet := \{v' \in V \mid (v, v') \in E\}$. For $V' \subseteq V$: $\bullet V' := \bigcup_{v \in V'} \bullet v$, $V' \bullet := \bigcup_{v \in V'} v \bullet$. Any node in $\bullet v$ ($v \bullet$) is a *father* (*son*) of v . $|\bullet v|$ ($|v \bullet|$) is the *in-* (*out-*)*degree* of v . A *root* (*leaf*) of a graph is a node with in-degree (out-degree) 0. A *connection* of length n between two nodes v, v' is a word $w = v_1 \dots v_{n+1}$ s.t. $v = v_1, v' = v_{n+1}$ and $(v_i, v_{i+1}) \in E \cup E^{-1}$ holds for $1 \leq i \leq n$. If $(v_i, v_{i+1}) \in E$ holds for all i w is called a *directed path* from v to v' . A *cycle* is a directed path of some length > 0 from one node to itself.

A *dag* (directed acyclic graph) is a graph without cycles. A *forest* is a dag where any node possesses at most one connection to at most one root. A *tree* is a forest with exactly one root. Thus, the empty graph $\varepsilon := (\emptyset, \emptyset, \emptyset)$ is a forest but not a tree. We usually identify isomorphic graphs and thus deal with *abstract* graphs.

By Σ^t , Σ^f , Σ^\dagger , and Σ^g we denote the sets of all abstract trees, forests, dags, and graphs, respectively, over Σ .

A graph is *ranked* if all nodes with the same label must also possess the same out-degree, and *double ranked* if the label defines both the in- and out-degree. It is *ordered* if a specific order between all sons of any node is given.

Algebraic Specification for Unranked Trees

Courcelle [4] defines a theory \mathcal{T}_{uu} for unranked, unordered trees that consists of a syntax of sorts $S_{uu} = \{l, t, f\}$ (for *letter, tree, forest*), operator symbols $Op_{uu} = \{p_{l \times f \rightarrow t}, r_{t \rightarrow f}, +_{f \times f \rightarrow f}, \theta_{\rightarrow f}\}$, and equations E_{uu} :

$$\begin{aligned} u + v &= v + u \\ (u + v) + w &= u + (v + w) \\ u + \theta &= u, \end{aligned}$$

for variables u, v, w of sort f .

Let Σ denote a set of 0-ary generators of sort l . Any unranked, unordered tree over Σ now simply becomes an element of sort t of $\mathcal{F}(\mathcal{T}_{uu}, \Sigma) = Term(\mathcal{S}_{uu}, \Sigma) / \equiv_{uu}$, where $Term(\mathcal{S}_{uu}, \Sigma)$ are all terms generated by $Op_{uu} \cup \Sigma$ and \equiv_{uu} is the \mathcal{S}_{uu} -algebra congruence induced by the equations E_{uu} . To get a theory \mathcal{T}_{uo} for unranked, ordered trees just drop the equation for commutativity.

We might try to follow this approach and define a theory \mathcal{T}_d for dags by adding to \mathcal{T}_{uu} a new sort s for *synchronization point* and a new operator symbol $q_{s \times t \rightarrow t}$ and study $Term(\mathcal{T}_d, \Sigma \cup \mathbb{N})$, where any integer $i \in \mathbb{N}$ is regarded as a 0-ary symbol of sort s . However, as we will not be able to use \mathcal{T}_d -algebras as a semantics for dags such an approach seems to be overloaded. In the following syntax of linear dag expression we mainly abbreviate $p(a, f)$ by af , $p(a, \theta)$ by a , $q(i, t)$ by it , and $r(t)$ by t and give a set-theoretic semantics.

Syntax of Graph Expressions

Let Σ denote a finite alphabet with $\Sigma \cap \mathbb{N} = \emptyset$. We define the sets $E_{\Sigma \cup \mathbb{N}}^t$ and $E_{\Sigma \cup \mathbb{N}}^f$ of *tree* and *forest expressions* over $\Sigma \cup \mathbb{N}$ as the smallest sets fulfilling the following requirements $\forall x \in \Sigma \cup \mathbb{N}$:

$$\begin{aligned} E_{\Sigma \cup \mathbb{N}}^t &\subseteq E_{\Sigma \cup \mathbb{N}}^f, \theta \in E_{\Sigma \cup \mathbb{N}}^f, x \in E_{\Sigma \cup \mathbb{N}}^t, \\ f, g \in E_{\Sigma \cup \mathbb{N}}^f &\implies xf \in E_{\Sigma \cup \mathbb{N}}^t, (f + g) \in E_{\Sigma \cup \mathbb{N}}^f. \end{aligned}$$

Define for $a \in \Sigma, i \in \mathbb{N}, f, g \in E_{\Sigma \cup \mathbb{N}}^f$:

$$int(\theta) := last(\theta) := first(\theta) := nol(\theta) := \emptyset,$$

$$int(a) := last(a) := nol(a) := first(a) := \emptyset, nol(i) := \emptyset, int(i) := last(i) = first(i) := \{i\},$$

$$first(af) := \emptyset, X(af) := X(f), \text{ for } X \in \{int, last, nol\},$$

$$first(if) := \{i\}, last(if) := last(f), X(if) := X(f) \cup \{i\}, \text{ for } X \in \{int, nol\},$$

$$X(f + g) := X(f) \cup X(g), \text{ for } X \in \{int, last, first, nol\}.$$

$int(f)$ tells which integers appear in f , $last(f)$ which integers appear as last elements, $nol(f)$ which integers as non-last elements, and $first(f)$ which integer as first elements of f . The binary relation \equiv on forest expressions is defined $\forall f, f', g, g', h \in E_{\Sigma \cup \mathbb{N}}^f, x \in \Sigma \cup \mathbb{N}$

by

$$1) f \equiv f, f \equiv g \implies g \equiv f, (f \equiv g \wedge g \equiv h) \implies f \equiv h,$$

$$2) f \equiv f' \implies xf \equiv xf', (f \equiv f' \wedge g \equiv g') \implies (f + g) \equiv (f' + g'),$$

3) $(f + g) \equiv (g + f)$, $(f + (g + h)) \equiv ((f + g) + h)$, $(f + \theta) \equiv f$.

By 1) \equiv becomes an equivalence relation, by 2) a congruence on our expressions, and fulfills by 3) the equations E_{uu} . The congruence \equiv_0 is defined as above but without the requirement $f + \theta \equiv_0 f$ in 3). $(f_1 + \dots + f_n)$ or $\sum_{1 \leq \nu \leq n} f_\nu$ abbreviates $(\dots(f_1 + f_2) + \dots) + f_n$.

Semantics of Graph Expressions

In a first, intermediate step we interpret a graph expression as a forest over $\Sigma \cup \mathbb{N}$.

$\forall x \in \Sigma \cup \mathbb{N}, f, g \in E_\Sigma^g$:

$\mathfrak{S}^o(\theta) := (\emptyset, \emptyset, \emptyset)$, $\mathfrak{S}^o(x) := (\{1\}, \emptyset, \lambda(1) := x)$,

$\mathfrak{S}^o(f) = (V, E, \lambda) \implies \mathfrak{S}^o(xf) := (V \cup \{v_{new}\}, E \cup \{(v_{new}, v) \mid v \in V \wedge \bullet v = \emptyset\}, \lambda \cup \lambda(v_{new}) := x)$,

$\mathfrak{S}^o(f + g) := \mathfrak{S}^o(f) + \mathfrak{S}^o(g)$, where $\alpha + \beta$ is the disjoint union of the two graphs α, β .

To get our intended interpretation as abstract graphs over Σ we regard all integers as synchronization points that must be synchronized (i.e., all occurrences of the same integer are identified) and deleted. Therefor we introduce the operation *Syd* (for "Synchronize and delete"):

$Syd_{i_1, \dots, i_k}(\alpha) := Syd_{i_1}(\dots(Syd_{i_k}(\alpha)\dots))$, with

$Syd_i(\alpha) := (V', E \cap (V' \times V') \cup E', \lambda|_{V'})$, for

$V' = \{v \in V \mid \lambda(v) \neq i\}$, $E' = \{(v, v') \mid \exists v_1, v_2 \in V : \lambda(v_1) = \lambda(v_2) = i \wedge (v, v_1), (v_2, v') \in E\}$, setting all nodes v as a father of all nodes v' if v possesses some son with label i and v' possesses some father with label i , deleting such all melted synchronization points.

The interpretation of an expression as an abstract graph is given as

$$\mathfrak{S}(f) := [Syd_{int(f)}(\mathfrak{S}^o(f))]_{\sim iso}.$$

Example 5.1 $\mathfrak{S}^o(f)$ and $\mathfrak{S}(f)$ for $f = a1b2a + a(1 + 2 + a) + ab$ are shown in figure 3.

The abstract dag α is the interpretation of the expressions $1a2c(1 + db2)$, $1c(a1 + db1)$, and $1db2c(a2 + 1)$. $\beta = \mathfrak{S}(d_i)$ for the expressions d_i for $1 \leq i \leq 5$ with

$$d_1 = a(1b + 2d) + c(1 + 2) + e2, \quad d_2 = a(1b + 2) + c(2d + 1) + e2, \quad d_3 = a1b + c12d + e2, \\ d_4 = a1(b + 2d) + c1 + e2, \quad d_5 = a1(b + 2d3) + 4c1 + 4e2.$$

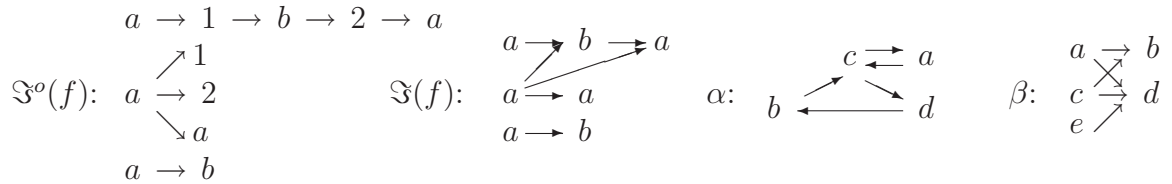


Figure 3: Some graphs of example 5.1

Of course, \equiv - or \equiv_0 -congruent expressions describe the same abstract graph.

DAG Expressions

Regard the relation $\sqsubseteq(f) \subseteq \text{int}(f) \times \text{int}(f)$ defined inductively for $i \in \mathbb{N}$, $a \in \Sigma$, $f, g \in E_{\Sigma \cup \mathbb{N}}^f$:

$$\sqsubseteq(\theta) := \sqsubseteq(a) := \sqsubseteq(i) := \emptyset,$$

$$\sqsubseteq(af) := \sqsubseteq(f), \sqsubseteq(if) := \sqsubseteq(f) \cup \{(i, j) \mid j \in \text{int}(f)\}, \sqsubseteq(f + g) := \sqsubseteq(f) \cup \sqsubseteq(g).$$

Obviously, the interpretation $\mathfrak{S}(f)$ of an expression f is an abstract dag if the transitive closure $\sqsubseteq(f)^+$ of $\sqsubseteq(f)$ is a partial order. Example 5.1 presents 5 rather different expressions d_i for the same abstract dag β of figure 3. However, only d_1 and d_2 are "smooth" expressions whilst d_3 to d_5 are against intuition. We need a formal definition for smoothness:

An expression $f \in E_{\Sigma \cup \mathbb{N}}^f$ is called *smooth* if f contains no sub-expressions

- θ , ij , ig ,
 - $(g + h)$ with $\text{not}(g) \cap \text{not}(h) \neq \emptyset$,
 - it with $i \in \text{not}(t)$,
 - i that occurs exactly once in f or with $i \in \text{int}(f) - \text{not}(f)$,
 - $\sum_{1 \leq \nu \leq n} t_\nu$ with two trees t_{ν_0}, t_{ν_1} for $1 \leq \nu_0 < \nu_1 \leq n$ with $\text{first}(t_{\nu_0}) = \text{first}(t_{\nu_1}) \neq \emptyset$,
- for $a \in \Sigma$, $i, j \in \mathbb{N}$, $t, t_\nu \in E_{\Sigma \cup \mathbb{N}}^t$, $g, h \in E_{\Sigma \cup \mathbb{N}}^f$.

In a smooth expression each synchronization point i must occur several times but only once as a non-last element. In this case i must precede some tree expression. No two synchronization points must follow each other. No brother trees must be synchronized. Tree and forest expressions over Σ are expressions without integers, a *graph expression* over Σ is a forest expression over $\Sigma \cup \mathbb{N}$, and a *dag expression* over Σ is a smooth graph expression where \sqsubseteq^+ is a partial order:

$$E_\Sigma^t := \{t \in E_{\Sigma \cup \mathbb{N}}^t \mid \text{int}(t) = \emptyset\}, E_\Sigma^f := \{f \in E_{\Sigma \cup \mathbb{N}}^f \mid \text{int}(f) = \emptyset\}, E_\Sigma^g := E_{\Sigma \cup \mathbb{N}}^g,$$

$$E_\Sigma^\dagger := \{f \in E_\Sigma^g \mid f \text{ is smooth and } \nexists i, j \in \text{int}(f) : (i \sqsubseteq(f)^+ j \wedge j \sqsubseteq(f)^+ i)\}.$$

All abstract trees, forest, dags, and graphs can be expressed:

Lemma 5.1 $\mathfrak{S}(E_\Sigma^t) = \Sigma^t$, $\mathfrak{S}(E_\Sigma^f) = \Sigma^f$, $\mathfrak{S}(E_\Sigma^\dagger) = \Sigma^\dagger$, $\mathfrak{S}(E_\Sigma^g) = \Sigma^g$.

Example 5.2 Let $f_1 = a1b2a7 + 6a(1 + 2 + \theta + 2 + 3a4) + 6 + ab(\theta + 7)$, $f_2 = 3a1 + 3 + a(1b2 + 2a4 + a5) + 6ab5$, and $d_6 = a1 + a(1b2 + 2a + a) + ab$. $\mathfrak{S}(f_1) = \mathfrak{S}(f_2) = \mathfrak{S}(d_6) = \mathfrak{S}(f)$ of figure 3. d_6 is a dag expression but f_1, f_2 are not as they violate smoothness.

5.2 DAG-Automata

Let M be a set. A function $f : M \times M \rightarrow M$ is *associative* and *commutative* if $f(x, y) = f(y, x)$ and $f(x, f(y, z)) = f(f(x, y), z)$ holds for all $x, y, z \in M$.

Such an associative and commutative function f is easily extended to

$$f^* : (\mathbb{N}^M - \{0\}) \rightarrow M$$

operating on non-empty multisets over M : $\forall a \in M : \forall m \in \mathbb{N}^M - \{0\}$:

$$f^*(1 \cdot a) := a, f^*(1 \cdot a + m) := f(a, f^*(m)).$$

A (root-to-frontier) *dag automaton*

$$A = (Q, \Sigma, \delta, \delta_i, \delta_o, \delta_I, \delta_F, I, F)$$

consists of

- a finite set Q of states with $Q \cap \mathbb{N} = \emptyset$,
- a finite alphabet Σ with $Q \cap \Sigma = \emptyset = \Sigma \cap \mathbb{N}$,
- a function $\delta : Q \times \Sigma \rightarrow 2^Q$,
- four associative and commutative functions

$$\delta_i, \delta_o, \delta_I, \delta_F : Q \times Q \rightarrow Q,$$

- a set $I \subseteq Q$ of initial states, and
- a set $F \subseteq Q$ of final states.

We now introduce the concept of configurations and computations of dag automata. By a simple scan through a dag expression d one can identify as *d-roots* those occurrences of letters in Σ that become roots in $\mathfrak{S}(d)$. A *configuration* C for A is a dag expression over $\Sigma \cup Q$ where exactly the states form the C-roots. Now, for a given dag d , put in front of each d-root in d some state of Q to get a configuration C_d . C_d is admissible if for the multiset m of added state $\delta_I^*(m) \in I$ holds. \mathcal{C}_d denotes the set of all those admissible configurations for d . Each $C \in \mathcal{C}_d$ is called a *start configuration* for d . As a configuration is a dag expression itself the congruences \equiv and \equiv_0 hold also for configurations.

A configuration C' is a *direct successor* of a configuration C , $C \vdash_A C'$, if there exists a configuration \hat{C} with $C \equiv_0 \hat{C}$ and C' is the result of replacing in \hat{C} a sub-expression

- sa by s' , with $s' \in \delta(s, a)$, or
- $s(f_1 + f_2)$ by $(s_1 f_1 + s_2 f_2)$, with $\delta_o(s_1, s_2) = s$, or
- $(s_1 + s_2)$ by s' , with $\delta_F(s_1, s_2) = s'$, or
- $(s_1 it + s_2 i)$ by $s' it$, with $\delta_i(s_1, s_2) = s'$, or
- $(s_1 i + s_2 i)$ by $s' i$, with $\delta_i(s_1, s_2) = s'$, or
- si by s , if i occurs exactly once in \hat{C} ,

for $a \in \Sigma, i \in \mathbb{N}, t \in \Sigma_{\Sigma \cup \mathbb{N}}^t, f_1, f_2 \in \Sigma_{\Sigma}^g, s, s_1, s_2 \in Q$. In addition, we write $d \vdash C$ for a dag expression d and a start configuration $C \in \mathcal{C}_d$. Only here δ_I plays a rôle. δ_o handles the transport of a state to outgoing arcs and δ_i of states from incoming arcs.

$eval^A(d) := \{s \in Q \mid d \vdash_A^* s\}$ is the set of all states into which a dag expression d may evaluate under A .

$D(A) := \{d \in E_{\Sigma}^{\dagger} \mid eval^A(d) \cap F \neq \emptyset\}$ is the dag language *accepted* by A . A dag language is *regular* if it is accepted by some dag automaton.

We may regard a dag automaton as operating on dag expressions, \equiv_0 -congruence classes of them, or on abstract dags, as any dag automaton operates identically on different dag expressions of the same abstract dag:

Theorem 12 *For all dag expressions d_1, d_2 and dag automata A over the same alphabet:*

$$\mathfrak{S}(d_1) = \mathfrak{S}(d_2) \implies eval^A(d_1) = eval^A(d_2).$$

Example 5.3 Let $\Sigma_{\text{even}}^\dagger$ denote the language of all dags over Σ with an even number of nodes. Σ_{even} is accepted by A_{even} with $Q := \{\mathbf{0}, \mathbf{1}\}$, where we use boldface integers as states, $I := F := \{\mathbf{0}\}$, $\delta(s, a) := s + \mathbf{1} \bmod 2$ for all $a \in \Sigma$, and $\delta_F := \delta_I := \delta_i := \delta_o := + \bmod 2$.

A non-deterministic computation with some dag expression d for the dag $\mathfrak{S}(f)$ of figure 3 is, e.g.

$$\begin{aligned} d &= a1b2a(3 + \theta) + \left(a((1 + 2) + a) + ab \right) \vdash \mathbf{1}a1b2a + \left(\mathbf{0}a((1 + 2) + a) + \mathbf{1}ab \right) (\in C_d) \\ \vdash_{(\delta)}^* \mathbf{0}1b2a + \mathbf{1}((1 + 2) + a) + \mathbf{0}b &\vdash_{(\delta_o, \delta)}^* \mathbf{0}1b2a + (\mathbf{1}(1 + 2) + \mathbf{0}a) + \mathbf{1} \vdash_{(\delta_o)} \\ \mathbf{0}1b2a + ((\mathbf{1}\mathbf{1} + \mathbf{0}\mathbf{2}) + \mathbf{0}a) + \mathbf{1} &\equiv_0 \left((\mathbf{0}1b2a + \mathbf{1}\mathbf{1}) + \mathbf{0}\mathbf{2} \right) + \mathbf{0}a + \mathbf{1} \vdash_{(\delta_i)} \\ ((\mathbf{1}\mathbf{1}b2a + \mathbf{0}\mathbf{2}) + \mathbf{0}a) + \mathbf{1} \vdash^* &(\mathbf{1}b2a + \mathbf{0}\mathbf{2}) + (\mathbf{1} + \mathbf{1}) \vdash^* (\mathbf{0}\mathbf{2}a + \mathbf{0}\mathbf{2}) + \mathbf{0} \vdash \mathbf{0}\mathbf{2}a + \mathbf{0} \vdash^* \mathbf{1}. \end{aligned}$$

Any computation for d leads to $\mathbf{1}$, thus $d \notin D(A_{\text{even}})$.

Example 5.4 Let $\Sigma_{\text{dis}}^\dagger$ denote the language of all disconnected dags over Σ and $\Sigma_{\text{con}}^\dagger$ those of all connected dags. We present an automaton A_d accepting $\Sigma_{\text{dis}}^\dagger$. When an abstract dag α consists of two disjunct dags α_1, α_2 an accepting computation of A_d guesses a state s_1 to be attached to all roots of α_1 and a different state s_2 to all roots of α_2 . A_d passes s_i through α_i . If α_1 and α_2 should have a common node the states s_1 and s_2 will meet and pass an error message to some leaf that forbids acceptance. Thus simply choose $Q := \{s_0, s_1, s_2, \checkmark, \perp\}$ with a sink state \perp (s.t. $\delta_\bullet(x, y) = \perp$ if $x = \perp$ or $y = \perp$ for all transition functions), $I := \{s_0\}$, $F := \{\checkmark\}$,

$$\delta_I(s_1, s_2) := s_0, \delta_I(s_1, s_1) := s_1, \delta_I(s_2, s_2) := s_2,$$

$$\delta(s_1, x) := s_1, \delta(s_2, x) := s_2, \text{ for } x \in \Sigma,$$

$$\delta_o(s_1, s_1) := s_1, \delta_o(s_2, s_2) := s_2,$$

$$\delta_i(s_1, s_1) := s_1, \delta_i(s_2, s_2) := s_2, \delta_i(s_1, s_2) := \perp,$$

$\delta_F(s_1, s_1) := s_1, \delta_F(s_2, s_2) := s_2, \delta_F(s_1, s_2) := \checkmark, \delta_F(\checkmark, s_i) := \checkmark$ for $i = 1, 2$, plus all required transitions to get commutative and associative mappings and make \perp a sink state. A "false" non accepting computation for the above dag expression d is shown in Figure 4.

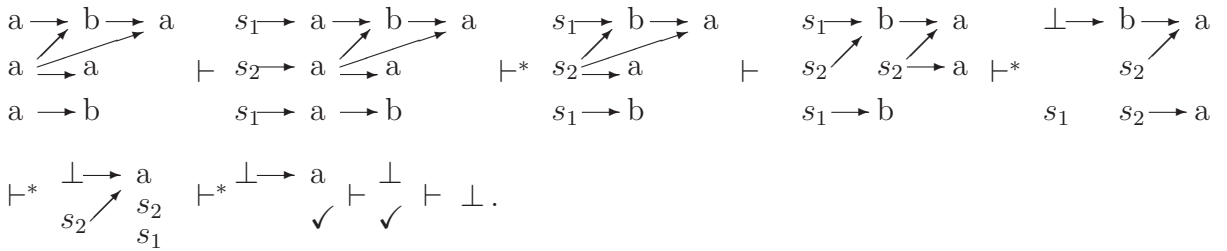


Figure 4: A non accepting computation of A_{dis}

Although the treatment of incoming and outgoing arcs in dag expressions is completely different (using a simple $+$ for outgoing arcs but an alphabet of infinitely many synchronization points for incoming arcs) they are treated symmetrically in dag automata. This is easily seen with reverse dags.

The *reverse* A^{rev} of an automaton $A = (Q, \Sigma, \delta, \delta_i, \delta_o, \delta_I, \delta_F, I, F)$ is

$$A^{rev} = (Q, \Sigma, \delta^{rev}, \delta_i^{rev}, \delta_o^{rev}, \delta_I^{rev}, \delta_F^{rev}, I^{rev}, F^{rev}),$$

with

$$\begin{aligned} I^{rev} &:= F, F^{rev} := I, \delta^{rev}(s, a) := \{s' \mid s \in \delta(s', a)\}, \\ \delta_i^{rev} &:= \delta_o, \delta_o^{rev} := \delta_i, \delta_I^{rev} := \delta_F, \delta_F^{rev} := \delta_I. \end{aligned}$$

Any successful computation in A for some α defines in reverse order immediately a successful computation for α^{rev} in A^{rev} . This implies $D(A)^{rev} \subseteq D(A^{rev})$. Further

$$D(A) = (D(A)^{rev})^{rev} \subseteq D(A^{rev})^{rev} \subseteq D((A^{rev})^{rev}) = D(A), \text{ thus}$$

Lemma 5.2 $D(A^{rev}) = D(A)^{rev}$

5.3 Regular and Non-Regular DAG Languages

Simple Regular DAG Languages and Closure Properties

Some typical examples of regular dag languages are - as expected - the language of all dags

- where no node possesses several sons with the same labels,
- where no node possesses several fathers with the same labels,
- where no node possesses several sons (fathers, respectively) with different labels,
- with exactly j roots (j leaves or j nodes, respectively),
- with 0 roots (0 leaves or 0 nodes, respectively) modulo some constant,
- where all non-roots (non-leaves) have the in-degree i (out-degree i , in-degree mod i is 0, out-degree mod i is 0, respectively).

Constructing dag automata that accept those languages is just a simple exercise.

A *maximal path* in a dag is a directed path from some root to some leaf. A path is identified with the word of labels of its nodes. $path(\alpha)$ is the set of all maximal paths in a dag α and $path(D) = \bigcup_{\alpha \in D} path(\alpha)$ for $D \subseteq \Sigma^\dagger$ defines a projection

$$path : 2^{\Sigma^\dagger} \rightarrow 2^{\Sigma^*}$$

from languages over dags into languages over words. In the opposite direction there are two canonical ways to embed languages over words into languages over dags:

- the *skinny* embedding of $L \subseteq \Sigma^*$ regards any word $w \in \Sigma^*$ as a path $w \in \Sigma^\dagger$ and is also denoted as $L (\subseteq \Sigma^\dagger)$,
- the *fat* embedding D_L of $L \subseteq \Sigma^*$ is the dag language

$$D_L := \{\alpha \in \Sigma^\dagger \mid path(\alpha) \subseteq L\}.$$

Lemma 5.3 *The skinny embedding L and the fat embedding D_L of a regular word language L are regular dag languages.*

The opposite statement holds for projections of a fat embedding but not for projections of general dag languages.

Lemma 5.4 *If D_L is a regular dag language then L is a regular word language. But there exist regular dag languages whose path projection is not even a context-free word language.*

Lemma 5.5 *The class of regular dag languages is closed under union and intersection.*

The proofs for all mentioned three lemmata are rather straight-forward. Interesting is a counter example for lemma 5.4 of a regular dag language with a non context-free path language: It is possible to construct a dag automaton A_D that accepts only dags of the form as shown in Figure 5 with an equal number of labels a, b, c and d where the order a

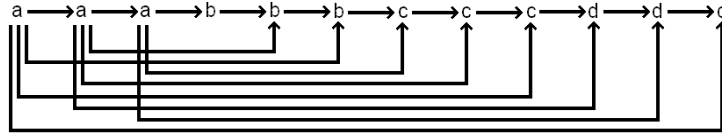


Figure 5: A dag accepted by A_D .

before b before c before d must be respected. This can be achieved by forcing all nodes with a label a to have four sons, labelled with a, b, c, d , but one who has three sons labelled with b, c and d . All nodes labelled with $x \in \{b, c, d\}$ are forced to have two fathers, one labelled with a and a second labelled with x with an exception for the first label x as shown. Thus, $path(D(A_D)) = \{a^i d^j | 1 \leq i, j\} \cup \{a^i c^j d^n | 1 \leq i, j \leq n\} \cup \{a^i b^j c^n d^n | 1 \leq i, j \leq n\}$, a non-context-free language. ■

Small Gaps between Regular and Non-Regular DAG Languages

There are simple example of non-regular dag languages as finite dag automata cannot count above some boundary: the language $D_{r=l}$ of all dags with the same number of roots and leaves, or $D_{n=}$ ($D_{r=}$, $D_{l=}$) over $\{a, b\}$ where equally many nodes (roots, leaves, respectively) are labelled with a and b . However, if one would change the concept of non-deterministic dag automata in such a way that also partial, non-associative functions δ_I and δ_F are allowed then $D_{r=}$ and $D_{l=}$ (in contrast to $D_{n=}$) become regular. To accept $D_{l=}$ choose $Q = \{s_0, s_a, s_b, \checkmark\}$, $I = \{s_0\}$, $F = \{\checkmark\}$, $\delta_I(s_0, s_0) = s_0$, $\delta(s_0, x) = s_x$ and $\delta_i(s_x, s_y) = s_0$ and $\delta_o(s_x, s_x) = s_x$ for $x, y \in \{a, b\}$ such that s_x tells that the last node visited has been labelled with x . Now, simply set $\delta_F(s_a, s_b) = \checkmark$, $\delta_F(\checkmark, \checkmark) = \checkmark$ and $\delta_F(., .)$ undefined elsewhere to accept $D_{l=}$. For $D_{r=}$ use the reverse automaton. However, such a trick is impossible with total associative and commutative functions δ_I, δ_F .

Theorem 13 Σ_{dis}^\dagger is regular but Σ_{con}^\dagger is not. Thus, regular dag languages are not closed under complement.

Proof. Regularity of Σ_{dis}^\dagger was shown in example 5.4. Non-regularity of Σ_{con}^\dagger is seen as follows: Suppose there exists an automaton A that accepts Σ_{con}^\dagger . Set

$$d := \sum_{1 \leq i \leq n} a(ia + i')$$

with $i' := i + 1$ for $1 \leq i < n$ and $n' := 1$, compare Figure 6 with $\alpha := \mathfrak{S}(d)$. α is

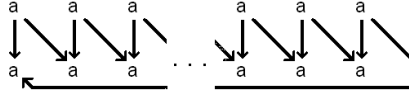


Figure 6: $\mathfrak{S}(d)$.

connected. Thus, A accepts α . For n large enough (i.e., longer than $(|Q| + 1)^2$) there must exist two different occurrences o'_1, o'_2 of labels a on the lower row and o_1, o_2 of their right fathers in the upper row where an accepting computation \mathcal{C} of A reaches in o_1 and o_2 the same state, say s , and in o'_1 and o'_2 a same state, say s' . Now, let β result from re-pointing the arc originally from o_1 to o'_1 now to o'_2 and the arc pointing originally from o_2 to o'_2 now to o'_1 . This doesn't introduce cycles and the same computation \mathcal{C} will still accept β - but β is disconnected (and still planar). ■

However, "bounded" connectivity becomes regular:

Lemma 5.6 *The languages of all connected dags with a fixed or bounded number of roots or leaves, respectively, are regular.*

Ladders of type 1 or 2 and beams are dags as presented in figure 7 $D_{1-ladder}$, $D_{2-ladder}$,

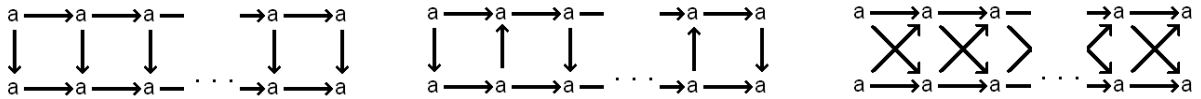


Figure 7: A type 1 ladder (left), type 2 ladder (middle) and beam (right)

D_{beam} denote the languages of all type 1 ladders, all type 2 ladders, and all beams, respectively, over a .

Theorem 14 *D_{beam} and $D_{2-ladder}$ are regular, $D_{1-ladder}$ is not.*

At a first sight, theorem 14 seems to point to a disadvantage of our concept of dag automata: type 1 ladders and type 2 ladders seem to be so similar that one might think that one type of ladders could result from the other by some "regular transformation" (and automata should preserve "regular transformations"). However, this is not the case. Type 1 and type 2 ladders have very distinct "synchronization properties": An automaton may

evaluate the upper row of a type-1-ladder ignoring the evaluation of the lower row, which is impossible for type 2 ladders. The situation is similar for Petri nets: There is a (rather simple) Petri net with $D_{2-ladder}$ as its true-concurrency dag semantics, but no Petri net can possess $D_{1-ladder}$ as its dag semantics, see [13].

One can prove theorem 14 by showing that any dag automaton accepting $D_{1-ladder}$ must also accept some connected but non-planar dag (that is not in $D_{1-ladder}$). Thus, $\mathcal{D}_{1-ladder}$ can't even be recognized relative to connected dags, i.e., if input dags are restricted to be connected. Both automata α, β of the proof of theorem 13 are planar. Thus, even if all input dags must be planar Σ_{con}^\dagger is not regular. This implies

Corollary 5.1 *The language Σ_{plan}^\dagger of all planar dags over Σ is not regular. Σ_{con}^\dagger is not regular relative to Σ_{plan}^\dagger and Σ_{plan}^\dagger is not regular relative to Σ_{con}^\dagger .*

The *shuffle*

$$D_1 \parallel D_2 := \{\alpha_1 + \alpha_2 \mid \alpha_i \in D_i\}$$

of two dag languages consists of disjoint unions of one dag from D_1 with one from D_2 . \parallel^i and the big shuffle \parallel^* are defined as

$$\parallel^0 D := \{\mathfrak{S}(\theta)\}, \quad \parallel^{n+1} := (\parallel^n D) \parallel D, \quad \parallel^* D := \bigcup_{i \geq 0} \parallel^n D.$$

It turns out that the big shuffle of even a single dag may be no regular language. Let \square be the beam of length 1, consisting of four nodes and being described by $a(1a+2a)+a(1+2)$. $d := a(a1a+a1)$ describes a dag $\diamond := \mathfrak{S}(d)$ with also only four nodes and four arcs as \square but only one root and leaf.

Theorem 15 *$\parallel^* \square$ and $\parallel^* \diamond$ are not regular.*

This immediately implies:

Lemma 5.7 *The class of regular dag languages is closed under \parallel^n for any n but not under \parallel^* .*

5.4 Deterministic DAG Automata

A dag automaton $A = (Q, \Sigma, \delta, \delta_i, \delta_o, \delta_I, \delta_F, I, F)$ is called *deterministic* if $|\delta(s, a)| = 1$ holds for $s \in Q, a \in \Sigma, I = \{s_0\}$ for one initial state s_0 , there exists a sink state $\perp \in Q$ and $\delta_I(s_0, s_0) = s_0, \delta_I(.,.) = \perp$ elsewhere, $\delta_o(s, s) = s$ for some states $s \in Q$ and $\delta_o(.,.) = \perp$ elsewhere. Thus, to ensure a deterministic computation a start configuration for a dag α receives by δ_I the same state s_0 attached to all roots and the same state must be prolonged by δ_o from a father to all sons. As in a parse tree for a context-free derivation, the order of where to apply a transition in a configuration is still free, but $|eval^A(d)| = 1$ will hold. A regular dag language is called *deterministic regular* if it is accepted by some deterministic dag automaton.

If one applies a deterministic (root-to-frontier) dag automata to the reverse α^{rev} of an unranked, unordered tree α it behaves exactly as Courcelle’s (frontiers-to-root) \mathcal{T}_{uu} -automata applied to α . Tree languages accepted by deterministic root-to-frontier tree automata are a proper subclass of those accepted by deterministic or non-deterministic frontier-to-root automata. Let $\triangleright = \mathfrak{S}(d)$ for $d = a1a + a1$. The trivial language $\{\triangleright\}$ is deterministic regular but $\{\triangleright\} \parallel \{\triangleright\}$ is not, as any deterministic automaton accepting $\triangleright + \triangleright = \mathfrak{S}(a1a + a1 + a2a + a2)$ must also accept $\square = \mathfrak{S}(a(1a + 2a) + a(1 + 2))$. Also, the regular languages Σ^{\dagger}_{even} and Σ^{\dagger}_{dis} are no longer deterministic regular. It is easily seen that the class of deterministic regular dag languages is closed under union, intersection and complement.

When a deterministic dag automaton passes a state from a father node to its sons it cannot react on the possibly different labels of the sons. Thus, deterministic dag automata are *forward blind*. One easily can define with the help of commutative and associative mappings a concept of non-forward-blind deterministic root-to-frontier dag automata where the state passed to a son may depend on the state of the father and the multiset of labels of all sons. An *nfb regular* dag language is a regular dag language accepted by a deterministic non-forward-blind dag automaton. Table 1 presents some properties of the classes of regular, deterministic regular, nfb regular and semi rational dag languages. A dag language is semi rational if it is the dag semantics of some Petri net, see [14]. In contrast to word languages, dag languages accepted by finite automata must not necessarily be Petri net dag languages, see the last two lines of table 1.

5.5 Comparison to Further Automata Concepts

There are several concepts in the literature of finite automata analyzing graphs or dags with ”local conditions”. Kaminski and Pinter [11] operate on rooted directed graphs over a double ranked alphabet, and, thus, with a global bound for the in- and out-degree. With their automata $D_{1-ladder}$ is also not recognizable, see Thomas [15]. (One easily may regard $D_{1-ladder}$ as a language over a double ranked alphabet by using different labels for the upper and lower row). Thomas introduces ”acceptors” on ranked graphs equivalent to existential monadic second-order logic (EMSL) on those graphs. Those acceptors simulate a tiling of a ranked graph with elementary graphs of a finite set of types plus some non-local constraints. $D_{1-ladder}$ becomes now acceptable relative to connected graphs. However, connectedness is expressible in MSL but not in EMSL, and thus not acceptable by those graph acceptors. It is hard to imagine how to generalize Thomas’ acceptor concept to unranked graphs and languages with no fixed bound of the in- and out-degree.

It is also known that $D_{1-ladder}$ is acceptable relativ to a class of planar dags (pdags) of Bossut, Dauchet and Warin [8]. They introduce algebraic pdag expressions built from two-sorted letters and operations (iterated) *parallel* and *serial* composition. They can present an automaton that accepts all connected pdag expressions, in contrast to our theorem 13, corollary 5.1 and inexpressibility of connectedness in EMSL. This contradiction is resolved if one notes that their pdag expression cannot describe all planar dags, especially not all planar dags of figure 6 that have been required for non-regularity of connectedness.

Closed under:	Reg_{det}^\dagger	Reg_{nfb}^\dagger	Reg^\dagger	$SemiRat^\dagger$
union	✓	✓	✓	✓
intersection	✓	✓	✓	✓
complement	✓	✓	no	no
reverse	no	no	✓	✓
shuffle	no	no	✓	✓
big shuffle	no	no	no	no
finite sets	no	no	✓	✓
fat embedding of \mathcal{L}_3	✓	✓	✓	✓
contains:				
Σ_{dis}^\dagger	no	no	✓	✓
$D_{r=l}$	no	no	no	✓
$D_{n=}$	no	no	no	✓
can count modulo i the:				
nodes	no	no	✓	✓
roots	no	no	✓	✓
leaves	✓	✓	✓	✓
incoming arcs	✓	✓	✓	(no)
outgoing arcs	no	✓	✓	(no)

Table 1: Closure Properties, () is a Conjecture.

Proofs for Chapter 5

Sketch of a proof for theorem 12. Let α be an abstract dag with some node v_o with an in-degree $n > 1$. Let d_1, d_2 be two different dag expression with the same interpretation α . From smoothness of dag expression one concludes that all incoming arcs into v_o must lead to one synchronization point, say i in d_1 and j in d_2 . The synchronization points i in d_1 and j in d_2 must occur exactly n times. Those synchronization points $i, j \in \mathbb{N}$ tell A which states of which fathers to be considered, but the operations of A are independent of the names "i" or "j" itself. Further, $n - 1$ occurrences of a synchronization point i in d_1 or j in d_2 are last elements and only after one occurrence the imbedded subtree in α with root v_o is expressed by some tree expression over $\Sigma \cup \mathbb{N}$. However, as δ_i is associative and commutative it is without any importance behind which occurrence of an i or j this is done.

Proof of lemma 5.3. This is obvious for the skinny embedding. For the fat embedding let the deterministic automaton $A_L = (Q_L, \Sigma, \delta_L, s_L, F_L)$ accept $L \subseteq \Sigma^*$. To accept D_L by $A = (Q, \Sigma, \delta, \delta_i, \delta_o, \delta_I, \delta_F, I, F)$ we set for $M, N \subseteq Q_L, x \in \Sigma$:
 $Q := 2^{Q_L} \cup \{\perp\}$ with a sink state \perp , $I := \{\{s_L\}\}$, $F := \{M \mid M \subseteq F_L\}$,
 $\delta(M, x) := \{\delta_L(s, x) \mid s \in M\}$, $\delta_i(M, N) := M \cup N$, $\delta_o(M, M) := M$,

$$\delta_I(s_L, s_L) = s_L \text{ and } \delta_I(.,.) = \perp, \text{ elsewhere, } \delta_F(M, N) := \begin{cases} F_L & ; \text{ if } M, N \subseteq F_L, \\ \perp & ; \text{ elsewhere.} \end{cases}$$

Proof of lemma 5.4. Let $A = (Q, \Sigma, \delta, \delta_i, \delta_o, \delta_I, \delta_F, I, F)$ accept the fat embedding D_L of some word language L . For $w \in L$ we regard w as a path with $\text{path}(w) = w$, thus w must be also an abstract graph in D_L . Therefor, A accepts w without using $\delta_i, \delta_o, \delta_I$, or δ_F . Thus, the non-deterministic automaton $A_L := (Q, \Sigma, \delta, I, F)$ accepts L .

For the counterexample of a regular dag language with a non-context-free path projection regard A_D with

$$\begin{aligned} Q &:= (\{s_a, s_a^o, s_b, s_c, s_d, s_b^i, s_c^i, s_d^i, s_{a,b}^i, s_{b,c}^i, s_{c,d}^i, s_1, s_1', s_2, \perp\} \text{ with a sink state } \perp, \\ \Sigma &:= \{a, b, c, d\}, I := \{s_a\}, F := \{s_d^i\}, \\ \delta(s_a, a) &= s_a^o, \delta(s_x, x) := s_x^i \text{ for } x \in \{b, c, d\}, \delta(.,.) := \perp \text{ elsewhere,} \\ \delta_i(s_b^i, s_{a,b}^i) &:= s_b, \delta_i(s_c^i, s_{a,c}^i) := \delta_i(s_b^i, s_{a,c}^i) := s_c, \delta_i(s_c^i, s_{a,d}^i) := \delta_i(s_d^i, s_{a,d}^i) := s_d, \\ \delta_o(s_a, s_{a,b}^i) &:= s_1, \delta_o(s_1, s_{a,c}^i) := s_2, \delta(s_b, s_{a,c}^i) := s_1', \delta_o(s_2, s_{a,d}^i) := \delta(s_1', s_{a,d}^i) := s_a^o, \\ \delta_I(q, q') &:= \delta_F(q, q') := \perp \text{ for } q, q' \in Q. \end{aligned}$$

(Note, to present a concrete dag automaton it suffices to present only the kernels that define the δ -functions. These kernel definitions have to be continued such that associativity and commutativity are achieved and that a sink state becomes a sink state.)

There holds $\delta_I^*(m) = s \in Q - \{\perp\} \Leftrightarrow m = 1 \cdot s$ which forces one root that achieves the state s_a in any start configuration. Analogously, δ_F forces exactly one leaf in state s_d^i in the final configuration of an accepting computation. Note, $\delta_o^*(m) = s_a \Leftrightarrow (m = 1 \cdot s_a + 1 \cdot s_{a,b}^i + 1 \cdot s_{a,c}^i + 1 \cdot s_{a,d}^i \text{ or } m = 1 \cdot s_b + 1 \cdot s_{a,c}^i + 1 \cdot s_{a,d}^i)$. Thus, δ_o forces any a -node to possess exactly four sons, labeled with a, b, c, d , or once three sons labelled with b, c, d . δ_i forces each node - but the first b -node - labelled with b, c or d to possess exactly two fathers, one of them an a -node. Thus, A_D accepts only dags with an equal number of labels a, b, c and d where the order a before b before c before d must be respected.

Proof of lemma 5.5. For closure under union and intersection the standard product of finite automata is simply generalized to dag automata:

For $A_j = (Q_j, \Sigma, \delta_j, \delta_{i,j}, \delta_{o,j}, \delta_{I,j}, \delta_{F,j}, I_j, F_j), j = 1, 2$, set

$$A_1 \times A_2 := (Q_1 \times Q_2, \Sigma, \delta_1 \times \delta_2, \delta_{i,1} \times \delta_{i,2}, \delta_{o,1} \times \delta_{o,2}, \delta_{I,1} \times \delta_{I,2}, \delta_{F,1} \times \delta_{F,2}, I_1 \times I_2, F'),$$

with

$$\delta_{F,1} \times \delta_{F,2}((x_1, x_2), (y_1, y_2)) := (\delta_{F,1}(x_1, y_1), \delta_{F,2}(x_2, y_2)), \text{ etc.}$$

To get the union $D(A_1) \cup D(A_2)$ choose $F' := (F_1 \times Q_2) \cup (Q_1 \times F_2)$. For the intersection set $F' := F_1 \times F_2$.

Proof of lemma 5.6. If the number of roots is bounded by l one can construct a finite automaton that guesses the number $k \leq l$ of roots and attaches to any root a different state s_1, \dots, s_k in an initial configuration. For $M, N \subseteq \{1, \dots, k\}$ a state s_M is passed along until it meets another state s_N , switching into $s_{M \cup N}$. s_M tells that up to now all roots in M can be connected by an undirected path. δ_F has to check that all roots are connected. One might try to define

$$\delta_F(s_M, s_N) := \begin{cases} s_{M \cup N} & ; \text{ if } M \cap N \neq \emptyset, \\ \perp & ; \text{ elsewhere,} \end{cases} \text{ with a sink state } \perp \text{ and set } F := \{s_{\{1, \dots, k\}}\}$$

to detect connectivity. However, such a δ_F is not associative. Therefore we need further states with subscripts \mathcal{M}, \mathcal{N} that are sets of subsets of $\{1, \dots, k\}$ and set $\delta_F(s_M, s_M) := s_{\{M\} * \{N\}}$, $\delta_F(s_{\mathcal{M}}, s_{\mathcal{N}}) := s_{\mathcal{M} * \mathcal{N}}$ with $A \in \mathcal{M} * \mathcal{N} : \iff (\exists B \in \mathcal{M} : \exists C \in \mathcal{N} : (B \cap C \neq \emptyset \wedge A = B \cup C))$ or $(A \in \mathcal{M} \cup \mathcal{N} \wedge \nexists B \in \mathcal{M} \cup \mathcal{N} : (A \neq B \wedge A \cap B \neq \emptyset))$. Now, with $F := \{s_{\{\{1, \dots, k\}\}}\}$ connectivity is detected. By symmetry the same holds for a fixed or bounded number of leaves.

Proof of theorem 14. Assume there exists an automaton A_l accepting exactly the type 1 ladders. For a type 1 ladder α long enough (i.e., longer than $(|Q| + 1)^2$) there must exist two occurrences o_1, o_2 of a in the upper line such that some accepting computation attaches to o_1 and to o_2 a same state, say s , and to their two sons in the lower row also a same state, say s' . Let β result from α by interchanging the lower sons of o_1, o_2 . β stays acyclic and the same computation accepts also β , but β is no type 1 ladder any more. A contradiction. As type-1-ladders are planar but β isn't this proves also **Corollary 5.1**.

Such an argument fails for beams or type 2 ladders as interchanging those two sons in a beam or type 2 ladder will introduce cycles. To prove that D_{beam} is regular we need the following topological property of beams: Beams are exactly those dags that

- possess exactly two roots and two leaves,
- all non-roots possess exactly two fathers,
- all non-leaves possess exactly two sons,
- there exist no three nodes v, v_1, v_2 such that v is the father of both v_1 and v_2 where v_1 is the second father of v_2 , i.e. no node is son and grandson of the same node.

All those properties are acceptable by a dag automaton A_b with

$Q := \{s_0, s_1, s_1^i, s_1^o, s_2, s_2^i, s_2^o, \checkmark, \perp\}$ with a sink state \perp , $I := \{s_0\}$, $F := \{\checkmark\}$,

$\delta_I(s_1, s_1) := s_0$, $\delta_I(., .) := \perp$, elsewhere,

$\delta(s_j, a) := s_j^o$, $\delta(s_j^i, a) := \delta(s_j^o, a) := \perp$ for $j = 1, 2$,

$\delta_o(s_j^i, s_j^i) := s_j^o$ for $j = 1, 2$ and $\delta_o(., .) := \perp$ elsewhere,

$\delta_i(s_1^i, s_1^i) := s_2$, $\delta_i(s_2^i, s_2^i) := s_1$, $\delta_i(., .) := \perp$ elsewhere,

$\delta_F(s_1^o, s_1^o) := \delta_F(s_2^o, s_2^o) := \checkmark$ and $\delta_F(., .) := \perp$ elsewhere.

δ_I forces two roots with an attached state s_1 in a start configuration. The change from s_j to s_j^o to s_j^i forces each node to possess exactly two sons and two fathers (or none). The change from sub-script 1 to 2 to 1 forces that no node is a son and grandson of one node.

A proof for the regularity of type 2 ladders is similar. Let an i - j -node be a node with i incoming and j outgoing arcs. Then type-2-ladders are uniquely described as those dags with

- there exists exactly one root that is a 0-2-node with its both sons being a 1-2-node and a 1-1-node that possesses itself one 2-1-node as son,
- each 1-2-node possesses exactly two 2-1-nodes or one 2-0-node and one 2-1-node as sons,
- each 2-1-node has one 1-2-node or 1-1-node as its son.

Those properties are acceptable by a dag automaton.

Proof of theorem 15. $\|\square$ is not regular: Assume some dag automaton A accepts $\|\square$. Let $\alpha \in \|\square$ be a union of a number of \square large enough that an accepting computation for

α must attach the same multiset of states to the roots of two different occurrences o_1, o_2 of \square . Let α' result from exchanging one son of o_1 with one son of o_2 . α' is still acyclic but not in $\|\ast\square$. However, the mentioned accepting computation for α cannot sense this change and still accepts α_1 . The argument for $\|\ast\diamond$ is similar.

References

- [1] K. Erk, L. Priese. *Theoretische Informatik*. Springer Verlag, 2000.
- [2] H. Comon, M. Daucher, R. Gilleron, S. Tison, and M. Tommasi. Tree automata techniques and application. Available on the Web from 13ux02.univ-lille.fr in directoty tata, 1998.
- [3] A. Brüggemann-Klein, M. Murata, and D. Wood. Regular tree and hedge languages of unranked alphabets. *Theor. Comp. Science Center Report HKUST-TCSC 2001-5*, pp29, 2001.
- [4] B. Courcelle. On recognizable sets and tree automata. In H. Aït-Kaci, M. Nivat, editor, *Resolution of Equations in Algebraic Structures*, volume 1, pages 93–126. Academic Press, 1989.
- [5] I. Boneva and J.-M. Talbot. Automata and logic for unranked and unordered trees. *LNCS*, 3467:500–515, 2005.
- [6] W. Thomas. Finite-state recognizability of graph properties. In D. Krob, editor, *Theorie des Automates et Applications*, volume 172, pages 147–159. l’Universite de Rouen, France, 1992.
- [7] T. Kamimura and G. Slutzki. Parallel and two-way automata on directed ordered acyclic graphs. *Inf. Control*, 49:10–51, 1981.
- [8] F. Bossut, M. Dauchet, and B. Warin. A Kleene theorem for a class of planar acyclic graphs. *Theor. Comp. Science Center Report HKUST-TCSC 2001-5*, 117:251–265, 1995.
- [9] B. Courcelle. A representation of graphs by algebraic expressions and its use for graph rewriting systems. In *Proc. 3rd Internat. Workshop on Graph-Grammars*, *LNCS*, pages 112–132. Springer Verlag, 1988.
- [10] W. Charatonik. Automata on dag representations of finite trees. Technical Report MPI-I-1999-2-001, MPI, Univ. Saarbrücken, 1999.
- [11] M. Kaminski and S. Pinter. Finite automata on directed graphs. *J. Comp. Sys. Sci.*, 44:425–446, 1992.
- [12] J. Fanchon and R. Morin. Regular sets of pomsets with auitoconcurrency. In *CONCUR 2002*, *LNCS 2421*, pages 402–417, 2002.
- [13] J.R. Menzel, L. Priese, and M. Schuth. Some examples of semi-rational dag languages. In *Developments in Language Theory: 10th International Conference, DLT 2006, St.Barbara, USA*, *LNCS 4036*, pages 351–362. Springer Verlag, 2006.
- [14] L. Priese. Semi-rational sets of dags. In *Developments in Language Theory: 9th International Conference, DLT 2005, Palermo, Italy*, *LNCS 3572*, pages 385–396. Springer Verlag, 2005.

- [15] W. Thomas. Automata theory on trees and partial orders. In *Proc. 7th Intern. Joint Conference CAAP/FALSE: TAPSOFT'97, LNCS*, volume 1214, pages 20–34. Springer Verlag, 1997.