Rheinisch-Westfälische Technische Hochschule Aachen
Lehrstuhl für Informatik VII
Prof. Dr. Wolfgang Thomas

Ausdrucksstärke von monadischem Datalog auf Bäumen

Seminar über Automaten für XML
WS 2005/06

Frank Radmacher
Matrikelnummer: 235573

Betreuung: Dr. Stefan Wöhrle
Lehrstuhl für Informatik VII, RWTH Aachen

Inhaltsverzeichnis

1 Einleitung

Eine der Hauptaufgaben bei der Arbeit mit Bäumen ist die Suche und Auswahl bestimmter Unterbäume und Knoten. Daher wollen wir hier eine weitere Möglichkeit vorstellen, *einstellige Anfragen über Bäumen* zu stellen, nämlich durch *monadische Datalog-Programme* [1].

Wir werden zunächst in Abschnitt 2 einige grundlegende Begriffe und Bezeichnungen einführen. In Abschnitt 3 werden wir monadisches Datalog als Anfragesprache vorstellen. In Abschnitt 4 werden wir die Äquivalenz dieser Anfragen zu MSO-definierbaren Anfragen zeigen. Dazu werden wir die in [2] eingeführten Anfrageautomaten durch monadische Datalog-Programme simulieren. Abschließend werden wir in Abschnitt 5 die Resultate noch einmal kurz zusammenfassen sowie einen kleinen Ausblick bieten.

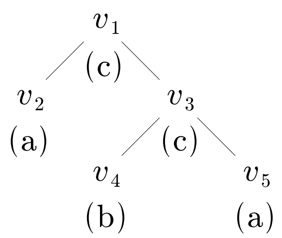
2 Grundlagen

In dieser Ausarbeitung beschränken wir uns auf endliche Bäume über einem endlichen Alphabet Σ. Dabei seien Bäume wie üblich definiert und bestehen aus mindestens einem Knoten, der Wurzel. Die Wurzel eines Baums t bezeichnen wir mit root_t. Die Nachfolger oder Kinder eines Knotens haben eine bestimmte Reihenfolge. Wir unterscheiden beschränkte und unbeschränkte Bäume. Beschränkte Bäume haben einen festen maximalen Knotengrad oder Rang K. Ein Knoten v mit Stelligkeit $k \leq K$ hat genau k Kinder. Wir schreiben dann $\mathrm{arity}(v) = k$. Knoten mit Stelligkeit 0 sind Blätter. Einen beschränkten Baum können wir als folgende relationale Struktur darstellen:

$$t_{\mathrm{rk}} = \langle \mathrm{dom}, \mathrm{root}, \mathrm{leaf}, (\mathrm{child}_k)_{k \leq K}, (\mathrm{label}_a)_{a \in \Sigma} \rangle$$

Dabei ist „dom“ (oder auch „dom_t“) die Menge aller Knoten des Baums t, „root“, „leaf“, „label_a“ sind einstellige Relationen und „child_k“ eine zweistellige Relation, deren Semantik deren intuitiven Bedeutung entspricht. „root“ enthält die Wurzel, „leaf“ die Blätter von t. „$\mathrm{child}_k(v_1, v_2)$“ besagt, dass v_2 der direkte k-te Nachfolger von v_1 in einem beschränkten Baum ist.

Beispiel 2.1. Betrachten wir folgenden Baum:



In der relationalen Struktur ist dann

$$\begin{aligned} \mathrm{dom} &= \{v_1, \ldots, v_5\}, & \mathrm{root} &= \{v_1\}, & \mathrm{leaf} &= \{v_2, v_4, v_5\}, \\ \mathrm{child}_1 &= \{\langle v_1, v_2\rangle, \langle v_3, v_4\rangle\}, & \mathrm{child}_2 &= \{\langle v_1, v_3\rangle, \langle v_3, v_5\rangle\}, & & \\ \mathrm{label}_a &= \{v_2, v_5\}, & \mathrm{label}_b &= \{v_4\}, & \mathrm{label}_c &= \{v_1, v_3\}. \end{aligned}$$

In unbeschränkten Bäumen darf jeder Knoten beliebig viele Kinder haben. Unbeschränkte Bäume können wir ebenfalls als relationale Struktur darstellen:

$$t_{\text{rk}} = \langle \text{dom}, \text{root}, \text{leaf}, \text{firstchild}, \text{nextsibling}, \text{lastsibling}, (\text{label}_a)_{a \in \Sigma} \rangle$$

Dabei entspricht die Semantik der Relationen wieder deren intuitiven Bedeutung [1]. Wir gehen im folgenden davon aus, dass beschränkte und unbeschränkte Bäume durch Strukturen der obigen Form gegeben sind, deren Signaturen wir mit τ_{rk} (für beschränkte Bäume) bzw. τ_{ur} (für unbeschränkte Bäume) bezeichnen werden. Wenn es unerheblich ist, ob wir einen beschränkten oder unbeschränkten Baum betrachten, werden wir auch häufig die Notationen $\tau_{\text{rk/ur}}$ für eine Signatur und $t_{\text{rk/ur}}$ für die entsprechende Interpretation verwenden.

Wir definieren die *monadische Logik zweiter Stufe* (MSO) auf Bäumen mit einer gebräuchlichen Syntax und der üblichen Semantik. Wie üblich, benutzen wir in MSO-Formeln Kleinbuchstaben $x, y, z, \ldots$ für Variablen über Knoten und Großbuchstaben $P_1, P_2, P_3, \ldots$ für Variablen über Mengen von Knoten. Als atomare Formeln sind die Relationssymbole aus τ_{rk} bzw. τ_{ur} und Formeln der Form $x = y$ und $P(x)$ bzw. $x \in P$ ausreichend.

Eine (einstellige) MSO-Anfrage ist eine MSO-Formel φ mit einer freien FO-Variablen. Die Antwort auf eine solche Anfrage $\varphi(x)$ zu einem Baum t ist die Knotenmenge $\{v \in \text{dom}_t \mid t \models \varphi(v)\}$.

3 Monadisches Datalog auf Bäumen

In diesem Abschnitt werden wir monadisches Datalog als Anfragesprache über Bäumen vorstellen. Wir werden kurz Syntax und Semantik vorstellen und dabei der Fixpunktsemantik besondere Aufmerksamkeit schenken. Eine ausführlichere und allgemeinere Einführung in Datalog findet sich in [3, 4].

Ein n-stelliges Prädikat in der Notation

$$p(a_1, \ldots, a_n)$$

heißt *Datalog-Ausdruck*, wobei a_i eine Konstante oder Variable ist (für $1 \leq i \leq n$), deren Wertebereich *dom* durch die relationale Struktur t_{rk} bzw. t_{ur} gegeben ist, also der Menge aller Baumknoten entspricht. Wir setzen voraus, dass Konstanten und Variablen durch ihre Notation voneinander unterscheidbar sind.

Wir unterscheiden zwischen extensionalen und intensionalen Prädikaten. Die *extensionalen Prädikate* sind die Prädikate aus der Signatur τ_{rk} bzw. τ_{ur} und werden entsprechend der gegebenen relationalen Struktur t_{rk} bzw. t_{ur} interpretiert. Darüber hinaus werden *intensionale Prädikate* durch Regeln eines *monadischen Datalog-Programms* wie folgt definiert:

Definition 3.1 (Monadisches Datalog-Programm)**.** Ein *monadisches Datalog-Programm* $\mathcal{P}$ ist eine endliche Menge von *monadischen Datalog-Regeln*. Eine solche Regel r hat dabei die Form

$$h(x) \leftarrow b_1(a_{1_1}, \ldots, a_{m_1}), \ldots, b_n(a_{1_n}, \ldots, a_{m_n}).,$$

wobei x eine Variable ist und die a_{i_j} Variablen oder Konstanten über *dom* sind. Mit *Vars*(r) bezeichnen wir die Menge aller in r auftretenden Variablen. Wir bezeichnen h als den *Kopf* und $b_1, \ldots, b_n$ als den *Rumpf* der Regel r. Prädikate im Kopf bezeichnen wir als *intensionalen Prädikate*, welche wir in der Menge $\mathrm{IPred}(\mathcal{P})$ zusammenfassen. Alle Regeln der obigen Form müssen folgende Eigenschaften erfüllen:

- $h \notin \tau_{\mathrm{rk/ur}}$, d. h. h ist nicht extensional.
- $b_i \in \tau_{\mathrm{rk/ur}}$ oder $b_i \in \mathrm{IPred}(\mathcal{P})$, d. h. b_i ist entweder extensional oder b_i steht im Kopf einer Regel.
- Die Variable x im Kopf $h(x)$ tritt auch im Rumpf in einem b_i auf.

Ein Beispiel für ein monadisches Datalog-Programm folgt später (Beispiel 3.3). Wir wollen zuvor jedoch noch auf die Semantik von monadischem Datalog eingehen. Zunächst werden wir dazu die Auswertung eines Datalog-Ausdrucks formalisieren.

Eine *Auswertung* ist eine Funktion $\phi : (\mathit{Vars}(r) \cup \mathit{dom}) \to \mathit{dom}$, die jede Variable auf ein Element aus *dom* abbildet und auf *dom* die Identität ist. Für einen Datalog-Ausdruck $p(a_1, \ldots, a_n)$ definieren wir $\phi(p(a_1, \ldots, a_n)) := p(\phi(a_1), \ldots, \phi(a_n))$. Ein ausgewerteter Datalog-Ausdruck $p(c_1, \ldots, c_n)$ enthält keine Variablen mehr und wird als Datalog-Fakt bezeichnet. Die Menge aller solcher *Grundatome* bezeichnen wir mit $\mathrm{Prim} := \{\phi(p(a_1, \ldots, a_n)) \mid \phi \text{ Auswertung}, p \in (\tau_{\mathrm{rk/ur}} \cup \mathrm{IPred}(\mathcal{P}))\}$.

Die *Fixpunktsemantik* eines monadischen Datalog-Programms $\mathcal{P}$ definieren wir im folgenden mit Hilfe von Transformationsfunktionen $\mathit{trans}^i_{p_j}$ für alle intensionalen Prädikate p_j von $\mathcal{P}$. Anschaulich beschrieben markieren wir von einem gegebenen Baum ausgehend durch die Transformationsfunktionen sukzessive die Baumknoten mit intensionalen Prädikaten.

Definition 3.2 (Fixpunktsemantik)**.** Sei $\mathcal{P}$ ein Datalog-Programm und seien $p_1, \ldots, p_n$ die intensionalen Prädikate von $\mathcal{P}$. Mit $\mathcal{T}^i_{\mathcal{P}}$ bezeichnen wir das Ergebnis der Berechnung nach der i-ten Ausführung von $\mathcal{P}$, d. h. die Menge aller nach dem i-ten Schritt gültigen Fakten. Die Menge der nach der i-ten Ausführung von $\mathcal{P}$ mit p_j markierten Baumknoten bezeichnen wir mit $T^i_{p_j}$. Anfangs sind $T^0_{p_1}, \ldots, T^0_{p_n} := \emptyset$, und die weiteren Markierungen ergeben sich sukzessive mit

$$\mathcal{T}^i_{\mathcal{P}} := \{p_j(v) \mid p_j \in \mathrm{IPred}(\mathcal{P}), v \in T^i_{p_j}\} \cup \{p(a_1, \ldots, a_n) \in t_{\mathrm{rk/ur}}\}$$

und

$$T^{i+1}_{p_j} := \mathit{trans}^i_{p_j}(\mathcal{T}^i_{\mathcal{P}}).$$

Dabei bestimmt $\mathcal{P}$ (für $1 \leq j \leq n$) die Transformationen $\mathit{trans}^i_{p_j} : 2^{\mathrm{Prim}} \to 2^{\mathrm{dom}}$ mit

$$\begin{aligned}\mathit{trans}^i_{p_j}(X) = T^i_{p_j} \cup \{\phi(x) \in \mathrm{dom}_t \mid{} & \text{es gibt eine Regel } p_j(x) \leftarrow b_1, \ldots, b_m. \in \mathcal{P},\\ & \text{so dass } \phi(b_1), \ldots, \phi(b_m) \in X\}.\end{aligned}$$

Die Fixpunkte $T^n_{p_j} = T^{n+1}_{p_j}$ der Folgen $T^0_{p_j}, T^1_{p_j}, T^2_{p_j}, \ldots$ bezeichnen wir mit $T^\omega_{p_j}$. Analog bezeichnen wir den Fixpunkt $\mathcal{T}^n_{\mathcal{P}} = \mathcal{T}^{n+1}_{\mathcal{P}}$ der Folge $\mathcal{T}^0_{\mathcal{P}}, \mathcal{T}^1_{\mathcal{P}}, \mathcal{T}^2_{\mathcal{P}}, \ldots$ mit $\mathcal{T}^\omega_{\mathcal{P}}$.

Die Funktionen $trans^i_{p_j}$ sind per Definition monoton. Nach dem Fixpunkt-Satz von Knaster-Tarski [4] hat jede monotone Funktion auf einem vollständigen Verband einen kleinsten Fixpunkt, und man erhält diesen Fixpunkt durch die Berechnung von $\mathcal{T}^\omega_\mathcal{P}$. Dieses minimale Modell $\mathcal{T}^\omega_\mathcal{P}$ von $\mathcal{P}$ ist der Schnitt aller Modelle von $\mathcal{P}$.

Eine *monadische Datalog-Anfrage* ist ein Datalog-Programm $\mathcal{P}$ mit einem als *Anfrageprädikat* ausgezeichneten $q \in \mathrm{IPred}(\mathcal{P})$. Die Antwort auf eine solche Anfrage über einem Baum t liefert die Knotenmenge $Q = \{v \in \mathrm{dom}_t \mid q(v) \in \mathcal{T}^\omega_\mathcal{P}\} = T_q$. Eine monadische Datalog-Anfrage kann als Auswahlfunktion gesehen werden, die eine Teilmenge von *dom* berechnet.

Beispiel 3.3. Angenommen, wir wollen alle Knoten eines binären (beschränkten) Baums berechnen, die in ihrem linken Teilbaum mindestens einen mit a beschrifteten Knoten und in ihrem rechten Teilbaum mindestens einen mit b beschrifteten Knoten haben. Wir entwerfen also ein monadisches Datalog-Programm über der Signatur τ_{rk}. Unser Programm mit einem Anfrageprädikat R besteht aus den Regeln:

$$A(x) \leftarrow \mathrm{label}_a(x). \tag{1}$$
$$A(x) \leftarrow \mathrm{child}_1(x, y), A(y). \tag{2}$$
$$A(x) \leftarrow \mathrm{child}_2(x, y), A(y). \tag{3}$$
$$B(x) \leftarrow \mathrm{label}_b(x). \tag{4}$$
$$B(x) \leftarrow \mathrm{child}_1(x, y), B(y). \tag{5}$$
$$B(x) \leftarrow \mathrm{child}_2(x, y), B(y). \tag{6}$$
$$R(x) \leftarrow \mathrm{child}_1(x, y_1), A(y_1), \mathrm{child}_2(x, y_2), B(y_2). \tag{7}$$

Betrachten wir nun wieder den binären Baum aus Beispiel 2.1. Die Fixpunktberechnung geschieht dann wie folgt. Der hochgestellte Index bezeichnet die Regel, mit der ein Datalog-Fakt jeweils gefolgert wurde.

$$\begin{aligned}
\mathcal{T}^0_\mathcal{P} = \{&\mathrm{root}(v_1), \mathrm{leaf}(v_2), \mathrm{leaf}(v_4), \mathrm{leaf}(v_5),\\
&\mathrm{child}_1(v_1, v_2), \mathrm{child}_1(v_3, v_4), \mathrm{child}_2(v_1, v_3), \mathrm{child}_2(v_3, v_5)\\
&\mathrm{label}_a(v_2), \mathrm{label}_a(v_5), \mathrm{label}_b(v_4), \mathrm{label}_c(v_1), \mathrm{label}_c(v_3)\}\\
\mathcal{T}^1_\mathcal{P} = \mathcal{T}^0_\mathcal{P} &\cup \{A(v_2)^{(1)}, A(v_5)^{(1)}, B(v_4)^{(4)}\}\\
\mathcal{T}^2_\mathcal{P} = \mathcal{T}^1_\mathcal{P} &\cup \{A(v_1)^{(2)}, A(v_3)^{(3)}, B(v_3)^{(5)}\}\\
\mathcal{T}^3_\mathcal{P} = \mathcal{T}^2_\mathcal{P} &\cup \{B(v_1)^{(6)}, R(v_1)^{(7)}\}
\end{aligned}$$

Dann ist $\mathcal{T}^3_\mathcal{P} = \mathcal{T}^4_\mathcal{P} = \mathcal{T}^\omega_\mathcal{P}$. Die Anfrageberechnung liefert uns folglich die Antwort $Q = \{v \mid R(v) \in \mathcal{T}^\omega_\mathcal{P}\} = T_R = \{v_1\}$.

4 Ausdrucksstärke

In diesem Abschnitt zeigen wir, dass monadische Datalog-Anfragen die gleiche Ausdrucksstärke wie MSO-Anfragen haben. Die Konstruktion einer MSO-Formel zu einer gegebenen monadischen Datalog-Anfrage ist einfach. Der Hauptteil dieses Abschnitts beschäftigt sich mit dem Beweis der Rückrichtung. Dazu werden wir *Anfrageautomaten* definieren, von denen wir wissen, dass diese gerade die in MSO definierbaren Anfragen beschreiben [2]. Dann brauchen wir nur noch Anfrageautomaten durch monadisches Datalog zu simulieren.

Zunächst wollen wir Datalog-Anfragen in MSO formulieren. Für diesen Zweck ist sogar Π_1-MSO ausreichend. Die Sprache Π_1-MSO ist ein MSO-Fragment, das nur universelle Sätze der Form $\forall P_1 \ldots \forall P_k \psi(P_1 \ldots P_k)$ enthält, wobei $\psi(P_1 \ldots P_k)$ eine prädikatenlogische Formel (FO-Formel) ist.

Satz 4.1 (Monadisches Datalog $\Longrightarrow$ Π_1-MSO)**.** *Jede monadische Datalog-Anfrage über beliebigen endlichen Strukturen ist* Π_1*-MSO-definierbar.*

Beweis. Sei $\mathcal{P}$ ein monadisches Datalog-Programm und o. B. d. A. sei P_1 das Anfrageprädikat. Dann können wir die durch $\mathcal{P}$ definierte Anfrage durch die MSO-Formel

$$\varphi(x) := \forall P_1 \ldots \forall P_n \ (\, \mathrm{SAT}(P_1, \ldots, P_n) \rightarrow x \in P_1 \,)$$

ausdrücken, wobei $\{P_1, \ldots, P_n\} = \mathrm{IPred}(\mathcal{P})$ und $\mathrm{SAT}(P_1, \ldots, P_n)$ die Konjunktion aller logischen Formeln ist, die jeweils einer Regel aus $\mathcal{P}$ entsprechen. Konkret entspricht hierbei eine Regel $h \leftarrow b_1, \ldots, b_m.$ der Formel

$$\forall z_1 \ldots \forall z_k \ (b_1 \wedge \ldots \wedge b_m \rightarrow h),$$

wobei $\{z_1, \ldots, z_k\}$ die Menge aller in der Regel vorkommenden Variablen ist.

Zu zeigen ist, dass $\mathcal{P}$ einen Knoten x selektiert genau dann, wenn $\varphi(x)$ wahr ist. Angenommen $\varphi(x)$ ist wahr. Da $P_1, \ldots, P_n$ allquantifiziert sind, wählt die MSO-Anfrage $\varphi(x)$ einen Knoten x aus, wenn ein Modell $P_1, \ldots, P_n$ mit $x \in P_1$ der Schnitt aller Modelle von $\mathrm{SAT}(P_1, \ldots, P_n)$ ist. Aus der Konstruktion von $\mathrm{SAT}(P_1, \ldots, P_n)$ folgt direkt, dass $P_1, \ldots, P_n$ dann auch der Schnitt aller Modelle von $\mathcal{P}$ ist. Da aus dem Fixpunkt-Satz von Knaster-Tarski folgt, dass das minimale Modell $\mathcal{T}_{\mathcal{P}}^{\omega}$ gleich dem Schnitt aller Modelle von $\mathcal{P}$ ist, stimmt das Modell $P_1, \ldots, P_n$ mit $\mathcal{T}_{\mathcal{P}}^{\omega}$ überein. Also selektiert auch $\mathcal{P}$ den Knoten $x \in P_1$.

Angenommen $\mathcal{P}$ selektiert $x \in P_1$ und $\mathcal{T}_{\mathcal{P}}^{\omega}$ sei der Fixpunkt von $\mathcal{P}$. Aus dem Fixpunkt-Satz folgt wieder, dass das minimale Modell $\mathcal{T}_{\mathcal{P}}^{\omega}$ der Schnitt aller Modelle von $\mathcal{P}$ ist. Daher ist ein Modell $P_1, \ldots, P_n$ von $\mathcal{P}$ auch stets Modell von $\mathrm{SAT}(P_1, \ldots, P_n)$, wenn $x \in P_1$ ist. Somit ist auch $\varphi(x)$ wahr. □

In den folgenden Abschnitten werden wir Anfrageautomaten definieren, die Neven und Schwentick in [2] eingeführt haben, und diese jeweils durch monadische Datalog-Programme simulieren. Wir unterscheiden zwischen beschränkten Anfrageautomaten (Abschnitt 4.1), die auf beschränkten Bäumen arbeiten, und starken unbeschränkten Anfrageautomaten (Abschnitt 4.2), die auf unbeschränkten Bäumen arbeiten.

4.1 Von beschränkten Anfrageautomaten zu monadischem Datalog

Wir definieren zunächst Anfrageautomaten über beschränkten Bäumen:

Definition 4.2 (QA^r)**.** Ein *beschränkter Anfrageautomat* (QA^r) ist ein *zwei Wege deterministischer beschränkter Baumautomat* (*2DTA*^{r}) [2] mit einer Auswahlfunktion λ. Formal ist ein QA^r ein Tupel

$$\mathcal{A} = \langle Q, \Sigma, F, s, \delta_\uparrow, \delta_\downarrow, \delta_{\text{root}}, \delta_{\text{leaf}}, \lambda \rangle$$

mit einer endlichen Zustandsmenge Q, einer Endzustandsmenge $F \subseteq Q$, einem Anfangszustand $s \in Q$, einem Rangalphabet Σ, Transitionsfunktionen δ und einer *Auswahlfunktion* $\lambda : Q \times \Sigma \to \{\bot, \top\}$. Seien U und D zwei disjunkte Teilmengen von $Q \times \Sigma$.

1. $\delta_\uparrow : U^{\leq K} \to Q$ ist die Transitionsfunktion für Aufwärtstransitionen.
2. $\delta_\downarrow : D \times \{1, \ldots, K\} \to Q^*$ ist die Transitionsfunktion für Abwärtstransitionen. Für jedes $i \leq K$ ist $\delta_\downarrow(q, a, i)$ ein Wort der Länge i.
3. $\delta_{\text{root}} : U \to Q$ ist die Transitionsfunktion für Wurzeltransitionen.
4. $\delta_{\text{leaf}} : D \to Q$ ist die Transitionsfunktion für Blatttransitionen.

Sei t ein beschränkter Baum. Ein *Schnitt* ist eine Teilmenge von dom_t, die genau einen Knoten jedes Pfads von der Wurzel zu einem Blatt enthält. Eine *Konfiguration* von $\mathcal{A}$ auf t ist eine Abbildung $c : C \to Q$ eines Schnitts C in die Zustandsmenge Q von $\mathcal{A}$. Der Automat $\mathcal{A}$ führt einen Berechnungsschritt $c_1 \to c_2$ zwischen zwei Konfigurationen $c_1 : C_1 \to Q$ und $c_2 : C_2 \to Q$ aus, wenn eine Aufwärts-, Abwärts-, Wurzel- oder Blatttransition ausgeführt wird:

1. $\mathcal{A}$ führt eine Aufwärtstransition von c_1 nach c_2 durch, wenn ein Knoten v mit Kindern $v_1, \ldots, v_n$ existiert, so dass
   a) $\{v_1, \ldots, v_n\} \subseteq C_1$,
   b) $C_2 = (C_1 - \{v_1, \ldots, v_n\}) \cup \{v\}$,
   c) $\delta_\uparrow(\langle c_1(v_1), \text{label}(v_1)\rangle, \ldots, \langle c_1(v_n), \text{label}(v_n)\rangle) = c_2(v)$ und
   d) c_2 ist identisch zu c_1 auf $C_1 \cap C_2$.
2. $\mathcal{A}$ führt eine Abwärtstransition von c_1 nach c_2 durch, wenn ein Knoten v mit Kindern $v_1, \ldots, v_n$ existiert, so dass
   a) $v \in C_1$,
   b) $C_2 = (C_1 - \{v\}) \cup \{v_1, \ldots, v_n\}$,
   c) $\delta_\downarrow(c_1(v), \text{label}(v), \text{arity}(v)) = c_2(v_1) \cdots c_2(v_n)$ und
   d) c_2 ist identisch zu c_1 auf $C_1 \cap C_2$.

3. $\mathcal{A}$ führt eine Wurzeltransition von c_1 nach c_2 durch, wenn
   a) $C_1 = C_2 = \{\text{root}_t\}$ und
   b) $\delta_{\text{root}}(c_1(\text{root}_t), \text{label}(\text{root}_t)) = c_2(\text{root}_t)$.
4. $\mathcal{A}$ führt eine Blatttransition von c_1 nach c_2 durch, wenn ein Blattknoten v existiert, so dass
   a) $v \in C_1$,
   b) $C_2 = C_1$,
   c) $\delta_{\text{leaf}}(c_1(v), \text{label}(v)) = c_2(v)$ und
   d) c_2 ist identisch zu c_1 auf $C_1 - \{v\}$.

Die Konfiguration $c : C \to Q$ mit $C = \{\text{root}_t\}$ und $c(\text{root}_t) = s$ ist die *Startkonfiguration.* Jede Konfiguration mit $c(\text{root}_t) \in F$ ist eine *akzeptierende* Konfiguration. Eine Konfigurationsfolge $c_1, \ldots, c_n$ nennen wir *Lauf*, wenn $c_1 \to \ldots \to c_n$ und c_1 die Startkonfiguration ist. Ein solcher Lauf *akzeptiert*, wenn c_n eine akzeptierende Konfiguration ist. Wir gehen in dieser Ausarbeitung davon aus, dass eine akzeptierende Konfiguration c_n keine Nachfolgekonfiguration besitzt, d. h. $\nexists c$ mit $c_n \to c$.

Obwohl $\mathcal{A}$ in einer Konfiguration durchaus verschiedene Transitionen ausüben kann, bezeichnen wir unsere Anfrageautomaten als deterministisch, da wegen der Disjunktheit der Teilmengen U und D von $Q \times \Sigma$ an jedem Knoten im Baum höchstens *eine* bestimmte Transition als nächstes ausgeführt werden kann. Dementsprechend ergibt sich für jeden Knoten stets in allen Läufen die gleiche Abfolge von Zuständen. Daher werden wir durchaus auch von *dem* Lauf und nicht von *einem* Lauf sprechen.

Ein Anfrageautomat $\mathcal{A}$ *markiert* einen Knoten v *in einer Konfiguration* $c : C \to Q$, wenn $v \in C$ und $\lambda(c(v), \text{label}(v)) = \top$. Durch die Auswahlfunktion λ *selektiert* $\mathcal{A}$ einen Knoten v eines Baums t, wenn ein Lauf $c_1, \ldots, c_n$ auf t akzeptiert und ein $1 \leq i \leq n$ existiert, so dass v in c_i markiert wird.

Im folgenden ist es unser Ziel, einen Anfrageautomaten nach Def. 4.2 durch ein monadisches Datalog-Programm mit einem Anfrageprädikat $query(x)$ zu simulieren. Vorbereitend wollen wir den Begriff der „imminent return situation“ einführen sowie ein kleines Lemma beweisen.

Definition 4.3. Seien $\mathcal{A}$ ein QA^r, $q, q_0 \in Q$, v_0 Elternknoten eines Knotens v und $c_1, \ldots, c_n$ ein Lauf von $\mathcal{A}$. Dann bezeichnen wir ein Tupel (q_0, q, v) als *imminent return situation*, wenn Indizes $i < j$ existieren, so dass $c_i(v_0) = q_0$, $\langle q_0, \text{label}(v_0)\rangle \in D$, $v_0 \notin C_k$ für $k \in [i+1, j]$, $c_j(v) = q$ und $\langle q, \text{label}(v)\rangle \in U$.

Anschaulich liegt eine *imminent return situation* (q_0, q, v) vor, wenn wir gerade von einem Knoten v mit Zustand q zu seinem Elternknoten v_0 zurückkehren und v_0 beim letzten Besuch den Zustand q_0 zugewiesen bekam. In diesem Fall lässt sich der Zustand q eindeutig durch seinen Knoten v und dem Zustand q_0 des Elternknotens bestimmen:

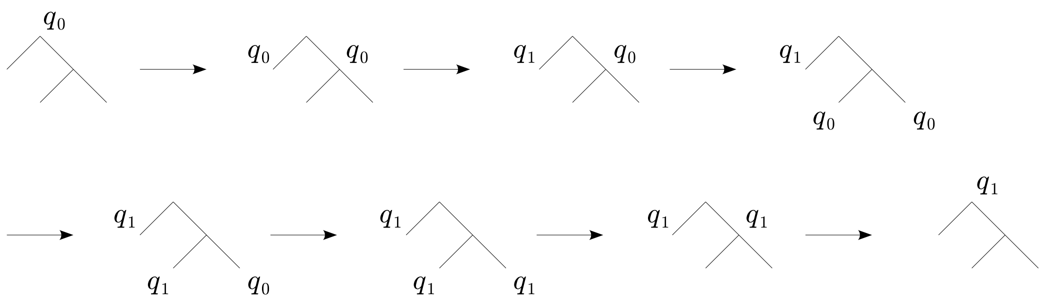
Lemma 4.4. *Gegeben seien ein Zustand q_0 und ein Knoten v. Dann gibt es höchstens einen Zustand q, so dass (q_0, q, v) eine* imminent return situation *ist.*

Die Behauptung folgt direkt aus der Tatsache, dass Anfrageautomaten deterministisch arbeiten. Dennoch findet sich in [1] ein detaillierterer Beweis.

Beispiel 4.5. Betrachten wir einen QA^r, der aus binären Bäumen die mit b markierten Knoten selektieren soll. Wir konstruieren dazu einen QA^r mit $Q = \{q_0, q_1\}$, $F = \{q_1\}$ und folgenden Transitionen:

- $\delta_{\downarrow}(q_0, *, 2) = \langle q_0, q_0 \rangle$
- $\delta_{\mathrm{leaf}}(q_0, *) = q_1$
- $\delta_{\uparrow}(\langle q_1, * \rangle, \langle q_1, * \rangle) = q_1$

Die Auswahlfunktion λ ist $\perp$ außer für $\lambda(q_1, b) = \top$. Betrachten wir den Lauf dieses QA^r auf dem Baum aus Beispiel 2.1:



In unserem Baum aus Beispiel 2.1 wird durch die Auswahlfunktion der einzige mit b beschriftete Knoten v_4 selektiert.

In diesem Beispiellauf ist unter anderem (q_0, q_1, v_3) eine *imminent return situation*. Da unsere Anfrageautomaten deterministisch arbeiten, ist der Zustand q_1 durch seinen Knoten v_3 und den Zustand q_0 des Elternknotens (die Wurzel v_1) eindeutig bestimmt.

Nun wollen wir uns unserem eigentlichen Ziel zuwenden:

Satz 4.6 ($QA^r \implies$ Monadisches Datalog)**.** *Sei $\mathcal{A}$ ein QA^r. Dann existiert ein monadisches Datalog-Programm $\mathcal{P}$, das eine zu $\mathcal{A}$ äquivalente Anfrage definiert.*

Um den Satz zu beweisen, werden wir nicht die einzelnen Konfigurationen in Datalog modellieren, sondern alle möglichen *Zustandszuweisungen*, die während eines Laufs von $\mathcal{A}$ auftreten können, in $\mathcal{P}$ formulieren. Zustandszuweisungen sind Paare (q, v), wobei v ein im Lauf besuchter Knoten ist, der den Zustand q zugewiesen bekommt. Formal bilden diese Paare die Menge

$$H = \{(q, v) \mid v \in C_i \text{ und } c_i(v) = q \text{ für ein } i\},$$

die in [1] auch „History“ genannt wird. Die Menge H beschreibt einen deterministischen Lauf im Anfrageautomaten eindeutig.

Unser Datalog-Programm soll nun alle Zustandszuweisungen berechnen, die im Lauf von $\mathcal{A}$ auftreten. Dann kann leicht ein Anfrageprädikat definiert werden, dass die gewünschte Anfrage definiert. Da wir nur an Zustandszuweisungen interessiert sind, die auch wirklich in einem akzeptierenden Lauf auftreten können, führen wir einen Datalog-Ausdruck $accept(x)$ ein, der beweisbar sein wird, wenn eine Zustandszuweisung schließlich der Wurzel x einen Endzustand zuweist. Der Datalog-Ausdruck $query(x)$ soll beweisbar sein, wenn der Knoten $x \in \text{dom}$ im Lauf markiert wird, und wenn $accept(y)$ für ein Knoten y beweisbar ist. Zustandszuweisungen werden dabei nur beweisbar sein, wenn sie in einem Lauf von $\mathcal{A}$ auftreten. Es ist daher wichtig, dass unsere Anfrageautomaten deterministisch arbeiten. Insbesondere sind entweder alle Zustandszuweisungen Teil eines akzeptierenden Laufs oder aber alle Zustandszuweisungen Teil eines nicht akzeptierenden Laufs.

Wir werden für alle vier Transitionsfunktionen aus Def. 4.2 Zustandszuweisungen entsprechende Datalog-Ausdrücke definieren, so dass die Korrektheit und Vollständigkeit der Konstruktion nahezu offensichtlich sind. Ein erster Versuch wäre es daher, für alle Paare (q, v) Datalog-Ausdrücke $q(v)$ entsprechend den Transitionen in $\mathcal{A}$ zu definieren. Für Abwärts-, Wurzel- und Blatttransitionen wäre eine solche Konstruktion ausreichend, da diese Transitionen nur vom Zustand eines Knotens abhängen und somit die entsprechenden Datalog-Regeln nur eine einzige Zustandszuweisung als Vorbedingung im Rumpf benötigen. Es können aus den Regeln keine Zustandszuweisungen abgeleitet werden, die nicht auch irgendwann in einem Lauf von $\mathcal{A}$ auftreten.

Um jedoch auch Korrektheit für Aufwärtstransitionen gewährleisten zu können, muss dagegen etwas mehr Aufwand in die Konstruktion gesteckt werden. Eine Aufwärtstransition hängt von mehreren Knoten und deren Zuständen ab. Daher muss bei einer Zustandszuweisung für eine Aufwärtstransition sichergestellt sein, dass die entsprechenden Zustandszuweisungen der Kinder auch wirklich in der gleichen Konfiguration auftreten können. Dieses ist der Fall, wenn beim letzten Mal als der entsprechende Elternknoten besucht wurde, dieser für alle seine Kinder den gleichen Zustand hatte. Daher erweitern wir unsere Zustandszuweisungen (q, v) unserer Konstruktion um den letzten Zustand q_0 des Elternknotens von v zu Tupeln (q_0, q, v). Wir müssen dann für Aufwärtstransitionen nur noch prüfen, ob für alle Zustandszuweisungen der Kinder die letzten Zustände der Elternknoten alle übereinstimmen. Wir werden die Notwendigkeit dieser Anpassung später noch an einem Beispiel verdeutlichen (Beispiel 4.7).

Im folgenden beschreiben wir die Simulation formal. Dazu verwenden wir Datalog-Ausdrücke in $(Q \cup \{\triangledown\}) \times Q$. Ein solches Prädikat der Form $\langle q_0, q\rangle(v)$ entspricht einer Zustandszuweisung (q_0, q, v). Das Symbol $\triangledown$ bezeichnet dabei einen Dummy-Zustand, den wir dem imaginären Elternknoten der Wurzel zuweisen, wenn eine Zustandszuweisung an der Wurzel erfolgt.

Beweis (von Satz 4.6). Das Datalog-Programm $\mathcal{P}$, das den QA^r $\mathcal{A}$ simuliert, enthält folgende Regeln für alle $q, q', q_1, \ldots, q_n \in Q$ und für alle $a, a_1, \ldots, a_n \in \Sigma$:

1. **Anfangszustand:** Wenn s der Anfangszustand von $\mathcal{A}$ ist, enthält $\mathcal{P}$ die Regel

$$\langle \triangledown, s\rangle(x) \leftarrow \text{root}(x).$$

2. **Aufwärtstransition:** Wenn $\delta_{\uparrow}(\langle q_1, a_1\rangle, \ldots, \langle q_n, a_n\rangle) = q'$, enthält $\mathcal{P}$ die Regeln

$$\begin{aligned}\langle q_0, q'\rangle(x) \leftarrow\ & \langle q_0, q\rangle(x), \\ & \mathrm{child}_1(x, x_1), \ldots, \mathrm{child}_n(x, x_n), \\ & \langle q, q_1\rangle(x_1), \ldots, \langle q, q_n\rangle(x_n), \\ & \mathrm{label}_{a_1}(x_1), \ldots, \mathrm{label}_{a_n}(x_n).\end{aligned}$$

für alle $q_0 \in (Q \cup \{\triangledown\})$.

3. **Abwärtstransition:** Wenn $\delta_{\downarrow}(q, a, n) = q_1 \ldots q_n$, enthält $\mathcal{P}$ die Regeln

$$\langle q, q_i\rangle(x_i) \leftarrow \langle q_0, q\rangle(x), \mathrm{child}_i(x, x_i), \mathrm{label}_a(x).$$

für alle $1 \leq i \leq n, q_0 \in (Q \cup \{\triangledown\})$.

4. **Wurzeltransition:** Wenn $\delta_{\mathrm{root}}(q, a) = q'$, enthält $\mathcal{P}$ die Regel

$$\langle \triangledown, q'\rangle(x) \leftarrow \langle \triangledown, q\rangle(x), \mathrm{label}_a(x), \mathrm{root}(x).$$

5. **Blatttransition:** Wenn $\delta_{\mathrm{leaf}}(q, a) = q'$, enthält $\mathcal{P}$ die Regeln

$$\langle q_0, q'\rangle(x) \leftarrow \langle q_0, q\rangle(x), \mathrm{label}_a(x), \mathrm{leaf}(x).$$

für alle $q_0 \in (Q \cup \{\triangledown\})$.

6. **Akzeptanzbedingung:** Wenn $q \in F$, enthält $\mathcal{P}$ die Regeln

$$\mathrm{accept}(x) \leftarrow \mathrm{root}(x), \langle q_0, q\rangle(x).$$

für alle $q_0 \in (Q \cup \{\triangledown\})$.

7. **Auswahlfunktion:** Wenn $\lambda(q, a) = \top$ enthält $\mathcal{P}$ die Regeln

$$\mathrm{query}(x) \leftarrow \langle q_0, q\rangle(x), \mathrm{label}_a(x), \mathrm{accept}(y).$$

für alle $q_0 \in (Q \cup \{\triangledown\})$.

Ein solches monadisches Datalog-Programm $\mathcal{P}$ zu einem QA^r $\mathcal{A}$ kann in LOGSPACE berechnet werden [1]. Zu zeigen bleiben Korrektheit und Vollständigkeit dieser Konstruktion. Für eine Menge $X \subseteq \mathcal{T}_{\mathcal{P}}^{\omega}$ definieren wir

$$\pi(X) := \{(q, v) \mid \exists q_0 \langle q_0, q\rangle(v) \in X\}.$$

Zu zeigen ist $\pi(\mathcal{T}_{\mathcal{P}}^{\omega}) = H$. Die Vollständigkeit $(\pi(\mathcal{T}_{\mathcal{P}}^{\omega}) \supseteq H)$ ist leicht einzusehen, denn per Konstruktion beinhalten die Zustandszuweisungen in unserem Fixpunkt $\mathcal{T}_{\mathcal{P}}^{\omega}$ die Zustandszuweisungen in H. Betrachten wir die Definitionen der Transitionen und Läufe in Def. 4.2, so sieht man, dass damit keine Übergänge möglich sind, die sich nicht auch in unseren Datalog-Regeln 1–5 widerspiegeln.

Der Beweis der Korrektheit $(\pi(\mathcal{T}_{\mathcal{P}}^{\omega}) \subseteq H)$ erfolgt durch Induktion über die Berechnung des Fixpunktes $\mathcal{T}_{\mathcal{P}}^{\omega}$:

Induktionsanfang Die Regel aus $\mathcal{P}$ für den Anfangszustand ist die einzige Regel, die kein Prädikat für eine Zustandszuweisung im Rumpf aufweist. Somit erhalten wir $\mathcal{T}_{\mathcal{P}}^1 = \mathcal{T}_{\mathcal{P}}^0 \cup \{\langle \nabla, s\rangle(\text{root}_t)\}$. Trivialerweise gilt $\pi(\mathcal{T}_{\mathcal{P}}^1) \subseteq H$.

Induktionsvoraussetzung Sei $X \subseteq \mathcal{T}_{\mathcal{P}}^\omega$ die Menge der Datalog-Fakten, die wir bereits berechnet haben, und gelte bereits $\pi(X) \subseteq H$.

Induktionsschluss 1. Fall: Regeln, die Wurzel-, Blatt- und Abwärtstransitionen entsprechen, haben nur genau eine Zustandszuweisung $\langle q_0, q\rangle(v)$ als Prämisse in ihrem Rumpf. Wenn diese Prämisse wahr ist, d. h. $\langle q_0, q\rangle(v) \in X$ und somit per Induktionsvoraussetzung $(q, v) \in H$, dann muss die gefolgerte Zustandszuweisung $\langle q_0', q'\rangle(v')$ auch wieder tatsächlich in einer Konfiguration im Lauf von $\mathcal{A}$ auftreten, also $(q', v') \in H$. Es folgt also $\pi(X \cup \{\langle q_0', q'\rangle(v')\}) \subseteq H$.

2. Fall: Bei Regeln, die Aufwärtstransitionen entsprechen, können dagegen mehrere Zustandszuweisungen in ihrem Rumpf auftreten. Wir wissen jedoch, dass bei einem Datalog-Ausdruck $\langle q, q_k\rangle(v_k) \in X$ mit $q \neq \nabla$ der Zustand q der zuletzt dem Elternknoten von v_k zugewiesene Zustand ist. Ist $\langle q_k, \text{label}(v_k)\rangle \in U$, so ist (q, q_k, v_k) eine *imminent return situation* gemäß Def. 4.3. Betrachten wir nun die Regel einer Aufwärtstransition, die mithilfe der Datalog-Ausdrücke $\langle q_0, q\rangle(v), \langle q, q_1\rangle(v_1), \ldots, \langle q, q_k\rangle(v_k) \in X$ im Rumpf auf $\langle q_0, q'\rangle(v)$ schließt. Dabei ist $\langle q_1, \text{label}(v_1)\rangle, \ldots, \langle q_k, \text{label}(v_k)\rangle \in U$ und die Knoten $v_1, \ldots, v_k$ sind die Kinder von v. Per Induktionsvoraussetzung gilt bereits $(q, v), (q_1, v_1), \ldots, (q_k, v_k) \in H$. Dann sind $(q, q_1, v_1), \ldots, (q, q_k, v_k)$ *imminent return situations* und nach Lemma 4.4 hängen die $q_1, \ldots, q_k$ nur vom Baum und vom Zustand q ab. Nach Induktionsvoraussetzung führt $\mathcal{A}$ also im Lauf irgendwann eine Abwärtstransition von Knoten v aus mit $c_i(v) = q$ für ein i. Die Folgetransitionen im Teilbaum von v sind dann von den *imminent return situations* $(q, q_1, v_1), \ldots, (q, q_k, v_k)$ bereits eindeutig und vollständig bestimmt. Da sich die zugehörigen Fakten in X befinden, existiert ein $j > i$, so dass für eine Konfiguration im Lauf von $\mathcal{A}$ irgendwann $c_j(v) = q'$ und somit auch $(q', v) \in H$ ist. Es folgt also $\pi(X \cup \{\langle q_0, q'\rangle(v)\}) \subseteq H$.

Damit ist unsere Behauptung $\pi(\mathcal{T}_{\mathcal{P}}^\omega) = H$ bewiesen.

Die Menge der durch die Auswahlfunktion λ selektierten Knoten haben wir durch das Anfrageprädikat *query*(x) in Teil 7 von $\mathcal{P}$ kodiert. Offensichtlich gilt für einen Baum t

$$\begin{aligned}
&\{v \mid \mathcal{A} \text{ selektiert } v \text{ in } t\} \\
\equiv\ &\{v \mid \mathcal{A} \text{ akzeptiert } t,\ (q, v) \in H \text{ und } \lambda(q, \text{label}(v)) = \top\} \\
\equiv\ &\{v \mid \text{query}(v) \in \mathcal{T}_{\mathcal{P}}^\omega\}.
\end{aligned}$$

Folglich ist die durch $\mathcal{P}$ definierte Anfrage zu der durch $\mathcal{A}$ definierten Anfrage äquivalent.

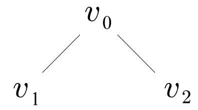
□

Beispiel 4.7. Gegeben sei ein QA^r mit $Q = \{q_0, q_1, q_2, q_3, q_f, q_+\}$, $F = \{q_f\}$ und folgenden Transitionen:

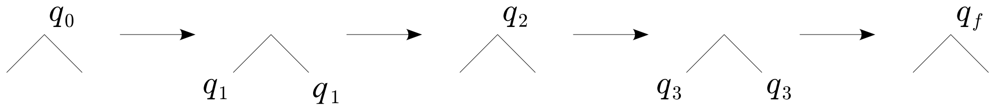
- $\delta_\downarrow(q_0, *, 2) = \langle q_1, q_1\rangle$

- $\delta_\uparrow(\langle q_1, *\rangle, \langle q_1, *\rangle) = q_2$
- $\delta_\downarrow(q_2, *, 2) = \langle q_3, q_3\rangle$
- $\delta_\uparrow(\langle q_3, *\rangle, \langle q_3, *\rangle) = q_f$
- $\delta_\uparrow(\langle q_1, *\rangle, \langle q_3, *\rangle) = q_+$
- $\delta_\uparrow(\langle q_3, *\rangle, \langle q_1, *\rangle) = q_+$

Die Auswahlfunktion λ ist $\perp$ außer für $\lambda(q_f, *) = \top$. Betrachten wir folgenden Baum:



Die Knotenbeschriftung ist für unser Beispiel unerheblich (seien einfach alle Knoten mit a beschriftet). Der deterministische Lauf des Automaten sieht folgendermaßen aus:



Durch die Auswahlfunktion λ wird also die Knotenmenge $Q = \{v_0\}$ selektiert. Konstruieren wir nun ein monadisches Datalog-Programm gemäß Satz 4.6, so folgt dieses Ergebnis auch aus der entsprechenden Fixpunktberechnung. Wir geben hier exemplarisch nur die Regeln an, die zum Ergebnis der Anfrage *query(x)* führen:

$$
\begin{aligned}
\langle\nabla, q_0\rangle(x) &\leftarrow \mathrm{root}(x). && \Longrightarrow \langle\nabla, q_0\rangle(v_0) \in \mathcal{T}_\mathcal{P}^\omega\\
\langle q_0, q_1\rangle(x_1) &\leftarrow \langle\nabla, q_0\rangle(x), \mathrm{child}_1(x, x_1), \mathrm{label}_a(x). && \Longrightarrow \langle q_0, q_1\rangle(v_1) \in \mathcal{T}_\mathcal{P}^\omega\\
\langle q_0, q_1\rangle(x_2) &\leftarrow \langle\nabla, q_0\rangle(x), \mathrm{child}_2(x, x_2), \mathrm{label}_a(x). && \Longrightarrow \langle q_0, q_1\rangle(v_2) \in \mathcal{T}_\mathcal{P}^\omega\\
\langle\nabla, q_2\rangle(x) &\leftarrow \langle\nabla, q_0\rangle(x), \mathrm{child}_1(x, x_1), \mathrm{child}_2(x, x_2),\\
&\qquad \langle q_0, q_1\rangle(x_1), \langle q_0, q_1\rangle(x_2), \mathrm{label}_a(x_1), \mathrm{label}_a(x_2). && \Longrightarrow \langle\nabla, q_2\rangle(v_0) \in \mathcal{T}_\mathcal{P}^\omega\\
\langle q_0, q_3\rangle(x_1) &\leftarrow \langle\nabla, q_2\rangle(x), \mathrm{child}_1(x, x_1), \mathrm{label}_a(x). && \Longrightarrow \langle q_0, q_3\rangle(v_1) \in \mathcal{T}_\mathcal{P}^\omega\\
\langle q_0, q_3\rangle(x_2) &\leftarrow \langle\nabla, q_2\rangle(x), \mathrm{child}_2(x, x_2), \mathrm{label}_a(x). && \Longrightarrow \langle q_0, q_3\rangle(v_2) \in \mathcal{T}_\mathcal{P}^\omega\\
\langle\nabla, q_f\rangle(x) &\leftarrow \langle\nabla, q_2\rangle(x), \mathrm{child}_1(x, x_1), \mathrm{child}_2(x, x_2),\\
&\qquad \langle q_2, q_3\rangle(x_1), \langle q_2, q_3\rangle(x_2), \mathrm{label}_a(x_1), \mathrm{label}_a(x_2). && \Longrightarrow \langle\nabla, q_f\rangle(v_0) \in \mathcal{T}_\mathcal{P}^\omega\\
\mathrm{accept}(x) &\leftarrow \mathrm{root}(x), \langle\nabla, q_f\rangle(x). && \Longrightarrow \mathrm{accept}(v_0) \in \mathcal{T}_\mathcal{P}^\omega\\
\mathrm{query}(x) &\leftarrow \langle\nabla, q_f\rangle(x), \mathrm{label}_a(x), \mathrm{accept}(y). && \Longrightarrow \mathrm{query}(v_0) \in \mathcal{T}_\mathcal{P}^\omega
\end{aligned}
$$

Der in diesem Beispiel definierte Anfrageautomat führt *keine* besonders sinnvolle Berechnung durch. Allerdings zeigt sich für diesen QA^r bei der Simulation durch monadisches Datalog sehr schön die Notwendigkeit der Erweiterung der Zustandszuweisungen zu

Paaren für Aufwärtstransitionen. Unser monadisches Datalog-Programm enthält unter anderem folgende Regeln (mit beliebigen $q_*, q_{*'} \in Q \cup \{\triangledown\}$):

$$\begin{aligned}
\langle q_*, q_2\rangle(x) \leftarrow\ & \langle q_*, q_0\rangle(x), \mathrm{child}_1(x, x_1), \mathrm{child}_2(x, x_2),\\
& \langle q_0, q_1\rangle(x_1), \langle q_0, q_1\rangle(x_2), \mathrm{label}_a(x_1), \mathrm{label}_a(x_2).\\
\langle q_*, q_f\rangle(x) \leftarrow\ & \langle q_*, q_2\rangle(x), \mathrm{child}_1(x, x_1), \mathrm{child}_2(x, x_2),\\
& \langle q_2, q_3\rangle(x_1), \langle q_2, q_3\rangle(x_2), \mathrm{label}_a(x_1), \mathrm{label}_a(x_2).\\
\langle q_*, q_+\rangle(x) \leftarrow\ & \langle q_*, q_{*'}\rangle(x), \mathrm{child}_1(x, x_1), \mathrm{child}_2(x, x_2),\\
& \langle q_{*'}, q_1\rangle(x_1), \langle q_{*'}, q_3\rangle(x_2), \mathrm{label}_a(x_1), \mathrm{label}_a(x_2).
\end{aligned}$$

Offensichtlich lässt sich der Datalog-Ausdruck $\langle q_*, q_+\rangle(\mathrm{root})$ mit unserem Beispiellauf nicht folgern. Würden wir jetzt allerdings nur Zustandszuweisungen der Form $q(x)$ als Prädikate verwenden, so ergäbe sich in unserem Datalog-Programm die Regel

$$\begin{aligned}
q_+(x) \leftarrow\ & \mathrm{child}_1(x, x_1), \mathrm{child}_2(x, x_2),\\
& q_1(x_1), q_3(x_2),\\
& \mathrm{label}_a(x_1), \mathrm{label}_a(x_2).
\end{aligned}$$

und $q_+(\mathrm{root})$ ließe sich fälschlicherweise folgern. Die verschiedenen Aufwärtstransitionen könnten nicht mehr unterschieden werden. Das Beispiel verdeutlicht, dass sich die Konstruktion aus Satz 4.6 entsprechend vereinfachen ließe, wenn man zusätzlich fordert, dass im QA^r an jedem Knoten nur eine Aufwärtstransition möglich sei.

4.2 Von starken unbeschränkten Anfrageautomaten zu monadischem Datalog

Wir definieren im folgenden Anfrageautomaten über unbeschränkten Bäumen, wobei wir die Definition eines *zwei Wege deterministischen endlichen Automaten* (*2DFA*) aus [2] voraussetzen:

Definition 4.8 (SQA^u)**.** Ein *starker unbeschränkter Anfrageautomat* (SQA^u) ist ein *starker zwei Wege deterministischer unbeschränkter Baumautomat* ($S2DTA^u$) [2] mit einer Auswahlfunktion λ. Formal ist ein SQA^u ein Tupel

$$\mathcal{A} = \langle Q, \Sigma, F, s, \delta_\uparrow, \delta_\downarrow, \delta_{\text{-}}, \delta_{\mathrm{root}}, \delta_{\mathrm{leaf}}, \lambda\rangle$$

mit Q, F,s,δ_{root},δ_{leaf} und λ wie in Def. 4.2. Seien U_{up} und U_{stay} zwei disjunkte reguläre Teilmengen von U^*. Die Transitionsfunktion für Aufwärtstransitionen ist nun von der Form $\delta_\uparrow : U_{\mathrm{up}} \to Q$ und die für Abwärtstransitionen von der Form $\delta_\downarrow : D \times \mathbb{N} \to Q^*$. Für alle $\langle q, a\rangle \in D$ muss $L_\downarrow(q, a) := \{\delta_\downarrow(q, a, i) \mid i \in \mathbb{N}\}$ regulär sein; für jedes $j \in \mathbb{N}$ muss $\delta_\downarrow(q, a, j)$ ein Wort der Länge j sein; und für jedes $q \in Q$ muss die Sprache $L_\uparrow(q) := \{w \in U^* \mid \delta_\uparrow(w) = q\}$ regulär sein. Aus dem Determinismus folgt, dass $L_\uparrow(q) \cap L_\uparrow(q') = \emptyset$ für alle $q \neq q'$.

Die Transitionsfunktion $\delta_{\text{-}} : U_{\mathrm{stay}} \to Q^*$ ist die so genannte *Verweiltransition*. Berechnet wird diese Funktion durch einen *2DFA* $\mathcal{B} = \langle S, \Sigma_{\mathcal{B}}, s_0, \delta_{\mathcal{B}}, F_{\mathcal{B}}, L, R\rangle$ über $\Sigma_{\mathcal{B}} = Q \times \Sigma$

mit einer Auswahlfunktion $\lambda_{\mathcal{B}} : S \times \Sigma_{\mathcal{B}} \to Q \cup \{\bot\}$. $\mathcal{B}$ arbeitet auf Wörtern der Form $\langle c_1(v_1), \text{label}(v_1)\rangle, \ldots, \langle c_1(v_n), \text{label}(v_n)\rangle$. Wir fordern, dass $\mathcal{B}$ immer hält und $\lambda_{\mathcal{B}}$ in einem Lauf jedem Knoten $v_1, \ldots, v_n$ *genau einen* Zustand aus Q zuweist. $\mathcal{A}$ führt eine Verweiltransition von $c_1 : C_1 \to Q$ nach $c_2 : C_2 \to Q$ durch, wenn ein Knoten v mit Kindern $v_1, \ldots, v_n$ existiert, so dass

a) $\{v_1, \ldots, v_n\} \subseteq C_1$,

b) $C_2 = C_1$,

c) $\delta_{\text{-}}(\langle c_1(v_1), \text{label}(v_1)\rangle, \ldots, \langle c_1(v_n), \text{label}(v_n)\rangle) = c_2(v_1) \cdots c_2(v_n)$ und

d) c_2 ist identisch zu c_1 auf $C_1 - \{v_1, \ldots, v_n\}$.

Wir fordern, dass an jedem Knoten höchstens eine Verweiltransition ausgeführt werden kann (diese Eigenschaft ist entscheidbar [2]).

Die restlichen Transitionen, Konfiguration, Lauf und Akzeptanz sind analog zu Def. 4.2 definiert.

Satz 4.9 ($SQA^u \implies$ Monadisches Datalog)**.** *Sei $\mathcal{A}$ ein SQA^u. Dann existiert ein monadisches Datalog-Programm $\mathcal{P}$, das eine zu $\mathcal{A}$ äquivalente Anfrage definiert.*

Beweis. Der Beweis funktioniert analog zu dem Fall der beschränkten Anfrageautomaten (Satz 4.6). Allerdings sind einige Änderungen in der Kodierung des Automaten notwendig:

1. **Abwärtstransitionen:** Für Abwärtstransition in einem Knoten v mit Kindern $v_1, \ldots, v_n$ müssen wir prüfen, ob es auch ein passendes Wort $q_1 \cdots q_n \in L_\downarrow(q, a) \subseteq Q^*$ gibt. Die Sprache $L_\downarrow(q, a)$ hat die *Dichte* 1, d. h. $|L_\downarrow(q, a) \cap \Sigma^i| \leq 1$, und jede Sprache mit konstanter Dichte kann als Vereinigung regulärer Ausdrücke der Form uv^*w (mit $u, v, w \in \Sigma^*$) geschrieben werden. Unser monadisches Datalog-Programm muss also prüfen, ob ein Matching $uv^kw \in L_\downarrow(q, a)$ der Länge n existiert, und bei Erfolg Zustandszuweisungen entsprechend des gematchten Wortes uv^kw durchführen.

2. **Aufwärtstransitionen:** Sei $\mathcal{B} = \langle Q, s_0, \delta, F\rangle$ ein endlicher Automat mit $\mathcal{L}(\mathcal{B}) = L_\uparrow(q)$. Unser Datalog-Programm muss also nur die Geschwister entsprechend der Relation nextsibling durchlaufen und die Zustandszuweisung durchführen, falls die Zustände der durchlaufenden Geschwister einem akzeptierenden Lauf von $\mathcal{B}$ entsprechen.

3. **Verweiltransitionen:** Auch hier werden ganz analog die Geschwister entsprechend der Relation nextsibling durchlaufen und es wird zunächst geprüft, ob es auf diesen einen akzeptierenden Lauf des 2DFAs gibt, welcher der Verweiltransition für die Knoten entspricht. Hat dieses Erfolg, werden die Zustandszuweisungen entsprechend der Auswahlfunktion $\lambda_{\mathcal{B}}$ durchgeführt. (Dieses funktioniert problemlos, da wir in Def. 4.8 vorausgesetzt haben, dass bei Terminierung jeder Knoten

genau einen Zustand aus Q zugewiesen bekommt. Weiterhin sind benötigte temporäre Prädikate eindeutig, da nach Voraussetzung nur eine Verweiltransition pro Knoten möglich ist.)

Für die konkreten Konstruktionen dieser Änderungen verweisen wir auf [1]. Der Beweis der Vollständigkeit und Korrektheit der Konstruktion funktioniert analog zu Satz 4.6. Das Resultat von Lemma 4.4 lässt sich problemlos auf den Fall für unbeschränkte Bäume übertragen. Die Konstruktion ist wiederum in LOGSPACE möglich. □

In [2] wurde die Äquivalenz von Anfrageautomaten zu MSO-Anfragen gezeigt:

Satz 4.10. *Eine einstellige Anfrage über beschränkten bzw. unbeschränkten Bäumen ist MSO-definierbar genau dann, wenn ein QA^r bzw. SQA^u existiert, der diese Anfrage berechnet.*

Durch unsere Simulationen (Sätze 4.6 und 4.9) und Satz 4.1 folgt auch die Äquivalenz von monadischem Datalog zu MSO-Anfragen:

Satz 4.11 (Monadisches Datalog $\iff$ MSO)**.** *Eine einstellige Anfrage über beschränkten bzw. unbeschränkten Bäumen ist MSO-definierbar genau dann, wenn ein monadisches Datalog-Programm über τ_{rk} bzw. τ_{ur} existiert, das die selbe Anfrage definiert.*

5 Zusammenfassung und Ausblick

Wir haben monadisches Datalog als Anfragesprache über Bäumen vorgestellt und deren Äquivalenz zu MSO-definierbaren Anfragen gezeigt (Satz 4.11). Dazu haben wir Anfrageautomaten eingeführt und diese durch monadische Datalog-Programme simuliert.

In [1] wird gezeigt, dass sich monadisches Datalog über Bäumen effizient auswerten lässt:

Satz 5.1. *Sowohl über τ_{rk} als auch über τ_{ur} lässt sich jede monadische Datalog-Anfrage in $O(|\mathcal{P}| \cdot |dom|)$ auswerten.*

Die in Abschnitt 4 vorgestellten Simulationen liefern monadische Datalog-Programme in der Größenordnung der gegebenen Anfrageautomaten. Bei diesen Simulationen bleibt daher auch die Effizienz der Anfrageauswertung in der gleichen Größenordnung.

Beispiel 4.7 verdeutlicht, dass sich die Simulation von Anfrageautomaten durch monadisches Datalog (und somit Beweis von Satz 4.6) durchaus noch vereinfachen ließe, wenn man die weitere Forderung an die Anfrageautomaten stellt, dass an jedem Knoten nur eine Aufwärtstransition möglich ist. Obwohl die Menge der durch deterministische zwei Wege Baumautomaten und deterministische Bottom-Up Baumautomaten erkannten Sprachen identisch ist ($\mathcal{L}(2DTA) = \mathcal{L}(\uparrow DTA)$) [2, 5], ist es jedoch nicht klar, ob die entsprechenden Anfrageautomaten mit nur einem Bottom-Up Durchlauf die gleiche Ausdrucksstärke wie herkömmliche QA^r bzw. SQA^u haben und sich die genannte Vereinfachung somit immer durchführen ließe.

Zu den weiteren Vorteilen von monadischem Datalog über Bäumen zählen die Existenz einer einfachen *Normalform* und die Anwendung für das *Visual Tree Wrapping*. Auf diese Aspekte wird in [1] näher eingegangen.

Literatur

[1] Georg Gottlob and Christoph Koch. Monadic datalog and the expressive power of languages for web information extraction. *JACM: Journal of the ACM*, 51, 2004.

[2] Frank Neven and Thomas Schwentick. Query automata over finite trees. *TCS: Theoretical Computer Science*, 275, 2002.

[3] Serge Abiteboul, Richard Hull, and Victor Vianu. *Foundations of Databases*. Addison-Wesley, 1995.

[4] Stefano Ceri, Georg Gottlob, and Letizia Tanca. *Logic programming and databases*. Springer-Verlag, Berlin, 1990.

[5] Etsuro Moriya. On two-way tree automata. *Information Processing Letters*, 50(3):117–121, 1994.