Endliche Baumautomaten

Christian Klein
ckwon@web.de

Vortrag im Rahmen des Seminars ' Logische Aspekte von XML '
im SS2003 an der Universität des Saarlandes

1 Übersicht

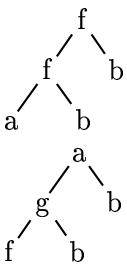
XML hat sich in den letzten Jahren als ein Standardformat etabliert. Da XML-Dokumente als Baum dargestellt werden können, ist es möglich, theoretische und praktische Ergebnisse aus dem Gebiet der Bäume in der theoretischen Informatik dafür anzuwenden.
Dieser Vortrag beschäftigt sich mit Automaten zum Erkennen von Mengen von Bäumen. Dazu können Konzepte von Wortautomaten benutzt werden. Die ersten Kapitel beschäftigen sich mit Bäumen, bei der sich die Anzahl der Kinder für jeden Knoten aus dem Knotenlabel erschliesst. Danach werden die unranked Bäume betrachtet, in deren Zusammenhang auch eine Anwendung für XML-Bäume behandelt wird, nämlich das lösen von XPATH-Anfragen. Zum Schluss stehen noch Tupelautomaten auf dem Programm, welche z.B. für die Korrespondenz zwischen Baumautomaten und WS2S benötigt werden.

2 Grundlagen

Ein (ranked) Alphabet ist eine Menge von Zeichen Σ und eine Funktion arity : $\Sigma \to \mathbb{N}$.
Ein Σ-Baum t ist ein Baum mit Knotenlabels aus Σ, so dass für jeden Knoten k gilt :
t(k) = σ $\Rightarrow$ die Anzahl der Kinder von k ist gleich arity(σ)
(wobei t(k) das Knotenlabel ist)

Der *shape* eines Baumes ist die Menge aller Knoten, wobei ϵ die Wurzel ist und für jeden Knoten k die Kinder die Positionen k1,...,kn haben.

Beispiel: Sei $\Sigma = \{f/2,\ g/1,\ a/0\}$.



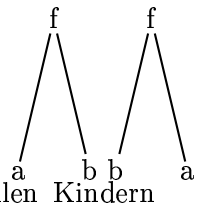
ist ein Σ-Baum mit shape $\{\epsilon, 1, 2, 11, 12\}$.

ist kein Σ-Baum, da die Stelligkeiten nicht eingehalten sind.

3 Motivation

Endliche Automaten können zum Erkennen von Mengen von Bäumen nur pfadweise eingesetzt werden.

Beispiel: Die Menge mit genau den beiden nebenstehenden Bäumen kann nicht von endlichen Automaten erkannt werden, da für das Akzeptieren eines Baumes Informationen über beide Kinder gleichzeitig von f benötigt werden.



Gewünscht ist eine Struktur, die den Zustand eines Knotens abhängig von allen Kindern bestimmt. Dies wird durch *Baumautomaten* erreicht.

4 Endliche Baumautomaten

4.1 Definition[1]

Ein *endlicher Baumautomat* ist ein Tupel (Q, Σ, F, Δ) mit Alphabet Σ, Zustandsmenge Q, einer Menge $F \subseteq Q$ und Übergangsrelation $\Delta \subseteq \bigcup_{n=1}^{\infty} Q \times \Sigma \times 2^{(Q^n)}$.
Die Regeln aus Δ sind von der Form $(q_0, \sigma) \rightarrow (p_1, \ldots, p_n) \cup \ldots \cup (q_1, \ldots, q_n)$ für $arity(\sigma) = n$. Diese Schreibweise ist top-down zu lesen, d.h. ein Knoten mit Markierung σ wird mit q_0 gelabelt, die n Kinder mit $q_1, \ldots, q_n$ unter Beibehaltung der Reihenfolge.

4.2 Weitere Definitionen

Ein *Lauf* r auf einem Baum t ist eine Funktion $shape(t) \rightarrow Q$, so dass für alle $p \in Pos(t)$ gilt: $n = arity(\ t(p)\ ),\ r(p) = q_0,\ r(p_i) = q_i$ für i=1,...,n $\Rightarrow \quad (q_0, t(p)) \rightarrow (q_1, \ldots q_n) \ \in \ \Delta$.

Ein Lauf r heißt *erfolgreich*, wenn $r(\epsilon) \in F$.

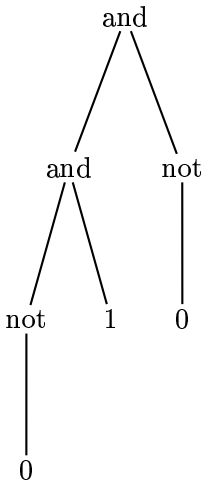
$L(M) \ := \{\ t \mid$ t ist Σ-Baum und es gibt einen erfolgreichen Lauf von M auf t $\}$ (*Sprache* des Automaten)

Eine Menge von Bäumen L heißt *erkennbar*, wenn es einen Automaten M gibt, so dass L = L(M).

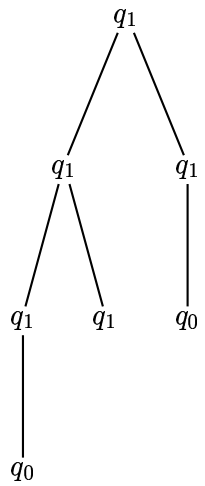
[1] Baumautomaten gehen zurück auf Doner (1965) und Thatcher & Wright (ebenfalls 1965)

Beispiel: Gegeben ist folgender Automat:
$\Sigma = \{and/2,\ not/1,\ 0/0,\ 1/0\}$, $Q = \{q_0,\ q_1\}$, $F = \{q_1\}$
$\Delta = \{(q_0, 0) \to \epsilon,\ (q_1, 1) \to \epsilon,$
$(q_0, not) \to q_1,\ (q_1, not) \to q_0,$
$(q_0, and) \to (q_0, q_0)\ \cup\ (q_0, q_1)\ \cup\ (q_1, q_0),\ (q_1, and) \to (q_1, q_1)$

Baum



erfolgreicher Lauf r:



Betrachte z.B. den Knoten 1. Er ist gelabelt mit 'and'. r(1) = q_1, r(11)=q_1 (linkes Kind von 1), r(12)=q_2 (rechtes Kind). Dies entspricht der letzten Regel in Δ. Da dies für alle Knoten gilt, ist r ein Lauf, da r(ϵ)= $q_1 \in F$ ist r ein erfolgreicher Lauf.
Die Sprache des Automaten ist die Menge aller gültigen Booleschen Formeln mit Konjunktion und Negation.
Beachte: Damit ein Lauf widergespiegelt wird, muss insbesondere für jedes Blatt eine Regel (q, σ) $\to \epsilon$ existieren, falls das Blatt mit q markiert ist.

4.3 Übersicht Traversierungsstrategien

Ein Lauf kann auf verschiedene Arten aus den Δ-Regeln berechnet werden. Die beiden naheliegendsten sind:

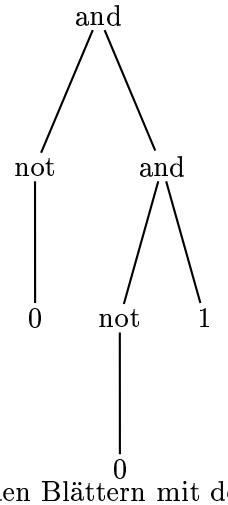
- **Bottom-up:** Beginne Berechnung bei den Blättern durch die Regeln der Konstanten, berechne aus den Zuständen der Kinder den Zustand eines Knotens. F beschreibt Endzustände; wenn ein Zustand aus F an der Wurzel steht, akzeptiere Baum.
  Bottom-up Berechnungen werden z.B. zur Auswertung logischer Formeln benutzt.

- **Top-down:** Beginne Berechnung an der Wurzel, labele die Kinder gemäß der rechten Seite der Δ-Regeln. F beschreibt Startzustände. Akzeptiere, wenn bei allen Blättern eine passende Regel für Zustand und Konstante existiert.
  Top-down Berechnungen werden z.B. beim Parsen von Programmen benutzt.

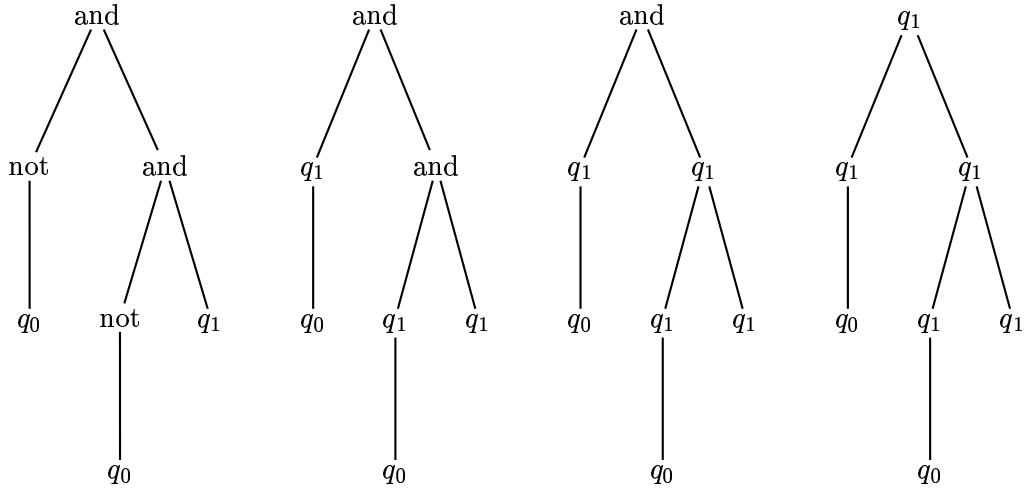
4.4 Bottom-up Berechnung

Beispiel: Wir benutzen denselben Automaten wie oben:
$\Sigma = \{and/2,\ not/1,\ 0/0,\ 1/0\}$, $Q = \{q_0,\ q_1\}$, $F = \{q_1\}$
$\Delta = \{(q_0, 0) \to \epsilon,\ (q_1, 1) \to \epsilon,\ (q_0, not) \to q_1,\ (q_1, not) \to q_0,$
$(q_0, and) \to (q_0, q_0)\ \cup\ (q_0, q_1)\ \cup\ (q_1, q_0),\ (q_1, and) \to (q_1, q_1)$

Berechnung für folgenden Baum:



Bottom-up Berechnung beginnt bei den Blättern mit den Regeln mit ϵ auf der rechten Seite für Konstanten; wenn Berechnung mit einem Endzustand an der Wurzel endet, so wird der Baum akzeptiert:



Alternative algebraische Charakterisierung

Bäume können durch Terme dargestellt werden. Bottom-up Berechnungen entsprechen dann Termersetzungen. Ein Term wird akzeptiert, wenn die Berechnung zu einem Zustand aus F führt.
Δ-Regel $(q_0, \sigma) \to (p_1, \ldots, p_n) \cup \ldots \cup (q_1, \ldots, q_n)$ wird interpretiert durch Termersetzungsregeln
$\sigma(p_1, \ldots, p_n) \to q_0$

$\vdots$
$\sigma(q_1, \ldots, q_n) \to q_0$

Beispiel:

Wir benutzen dasselbe Δ wie im obigen Beispiel.

and(not(and(1,1)), and(not(0), 1))
$\to$ and(not(and(q_1,q_1)), and(not(q_0), q_1))
$\to$ and(not(q_1, and(q_1, q_1))
$\to$ and(q_0, q_1)
$\to q_0$
Der Term and(not(and(1,1)), and(not(0), 1)) wird also nicht akzeptiert.

Bottom-up deterministische Baumautomaten

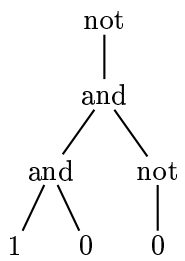
Ein Baumautomat heißt *bottom-up deterministisch*, wenn es in Δ keine zwei Regeln
$(q_0, \sigma) \to REGEXP_1 \cup (q_1, \ldots, q_n)$ und
$(p_0, \sigma) \to REGEXP_2 \cup (q_1, \ldots, q_n)$
gibt (d.h. es sind nie zwei Regeln bei einem Knoten anwendbar).

Satz: Für jede erkennbare Sprache L gibt es einen Bottom-up deterministischen Baumautomaten M, so dass L = L(M).
Beweis: Die Potenzmengenkonstruktion kann analog zu den endlichen Automaten angewandt werden.

4.5 Top-down Berechnung

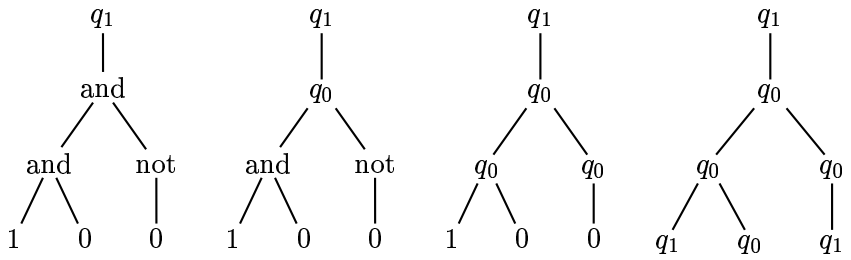
Beispiel: Es wird derselbe Automat wie im Beispiel zur Bottom-up Berechnung benutzt.

Baum :



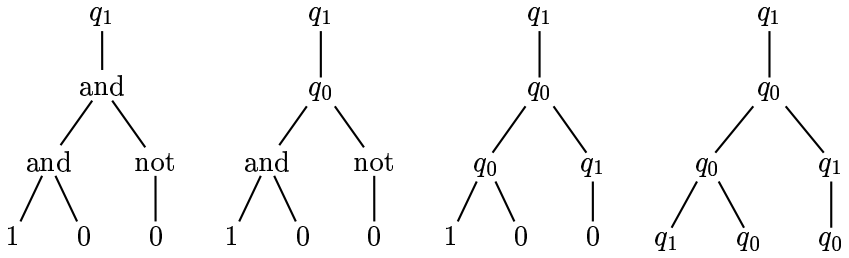
Berechnung beginnt mit Zustand aus F an der Wurzel. Der Automat ist nichtdeterministisch, angegeben ist eine nicht akzeptierende Berechnung und eine akzeptierende.

Erste Berechnung:



Dies ist eine nicht akzeptierende Berechnung, da es für das rechte Blatt keine Regel (q_1, 0) $\rightarrow \epsilon$ in Δ gibt.

Zweite Berechnung:



Dies ist eine akzeptierende Berechnung.

Bemerkung:

Bottom-up und top-down werden genau dieselben Bäume erkannt, da dieselben Läufe berechnet werden

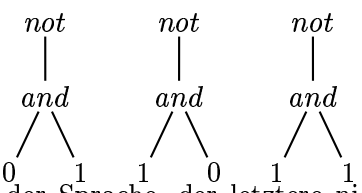
Top-down deterministische Automaten

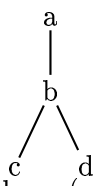
Ein Automat ist *top-down deterministisch*, wenn in Δ jede Regel die Form $(q_0, \sigma) \rightarrow (q_1, \ldots, q_n)$ hat, es keine zwei Regeln mit gleicher linker Seite gibt und $|F| < 2$.

Bemerkung:

Baumautomaten, die bottom-up deterministisch sind, können top-down nichtdeterministisch sein (s.Beispiel) und umgekehrt

Satz: Es existieren erkennbare Sprachen, die nicht von einem top-down deterministischen Automaten erkannt werden können.
Beweis: Die Sprache der gültigen booleschen Ausdrücke mit Konjunktion und Negation kann nicht top-down deterministisch erkannt werden.

Betrachte die drei Bäume 

.
Die ersten beiden sind in der Sprache, der letztere nicht. Jede Berechnung eines Top-down deterministischen Automaten führt bei allen drei zum Baum 

, da die Bäume oberhalb der Blätter gleich sind und daher zu denselben Zuständen führen (a, b, c und d müssen nicht paarweise verschieden sein).
Damit die ersten beiden Bäumen in der Sprache sind, muss es Regeln (c, 0) $\to \epsilon$, (c, 1) $\to \epsilon$, (d, 0) $\to \epsilon$, (d, 1) $\to \epsilon$ in Δ geben. Dadurch wird aber auch der dritte Baum akzeptiert, der nicht in der Sprache liegt.

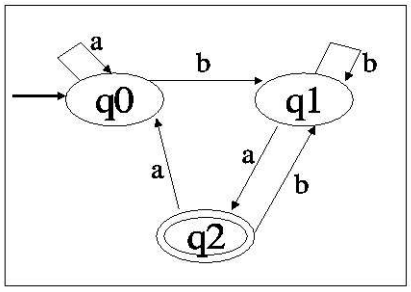
4.6 Umwandlung DEA → Top-down deterministischer Automat

Gegeben: DEA M=(Q,Σ,Δ,S,F)
Konstruiere Baumautomat M'=(Q',Σ',Δ',S',F')

- Q' = Q
- *Sigma*'=$\{ \sigma/1 \mid \sigma \in \Sigma \} \cup \{\perp/0\}$ (Buchstaben werden zu unären Funktionssymbolen, $\perp$ wird benötigt, um Wortende zu markieren)
- F'=S
- $\Delta' = \Delta \cup \{ (q, \perp) \to \epsilon \mid q \in F \}$

Wörter werden als Baum dargestellt, z.B. top-down, indem man den Wortanfang als Wurzel des Baumes setzt und jeweils als Kind den nächsten Buchstaben nimmt.
Aus aba wird z.B. 

Beispiel: Gegeben sei der endliche Automat M=($\{q_0, q_1, q_2\}$, $\{a, b\}$, Δ, S=$\{q_0\}$, F=$\{q_2\}$) mit $\Delta = \{(q_0, a) \to q_0, (q_0, b) \to q_1, (q_1, a) \to q_2, (q_1, b) \to q_1, (q_2, a) \to q_0, (q_2, b) \to q_1\}$
(Dieser Automat akzeptiert alle Wörter, die auf ba enden)



Für das Wort *abba* liefert der Baumautomat folgende top-down Berechnung (Beachte : wegen F=$\{q_2\}$ wird beim Baumautomat eine Regel $(q_2, \perp) \to \epsilon$ hinzugefügt) :

a	q_0	q_0	q_0	q_0	q_0
|	|	|	|	|	|
b	b	q_0	q_0	q_0	q_0
|	|	|	|	|	|
b	b	b	q_1	q_1	q_1
|	|	|	|	|	|
a	a	a	a	q_1	q_1
|	|	|	|	|	|
$\perp$	$\perp$	$\perp$	$\perp$	$\perp$	q_2

Das Wort wird also akzeptiert.

4.7 Übersicht Sprachklassen

DEA/NEA-Sprachen	$\subset$	deterministische Top-down Sprachen	$\subset$	erkennbare Sprachen

4.8 Abschlusseigenschaften

Satz: Erkennbare Sprachen sind abgeschlossen unter Schnitt, Vereinigung und Komplementbildung.
Beweis Automaten für Schnitt bzw. Vereinigung können durch Konstruktion des Produktautomaten erzeugt werden (vgl. endliche Wortautomaten). Man kann also in linearer Zeit prüfen, ob ein Baum im Schnitt/in der Vereinigung zweier erkennbarer Mengen liegt.
Einen Automaten für das Komplement einer Sprache erhält man, indem man den Automaten determinisiert und dann End- und Nichtendzustände vertauscht. Dies kann zu exponentiellem Wachstum in den Zuständen und Regeln führen.
Satz: Top-down deterministisch erkennbare Sprachen sind abgeschlossen unter Schnitt und Vereinigung.
Bemerkung: Top-down deterministisch erkennbare Sprachen sind *nicht* abgeschlossen unter Komplementbildung.

5 Unranked Automaten

Die im vorangegangenen Kapitel vorgestellten Baumautomaten sind beschränkt auf Alphabete, bei denen bereits vorher feststeht, wieviele Nachfolger jeder Knoten hat. Da es nun aber viele Baumanwendungen gibt, bei denen dies nicht der Fall ist (z.B. XML-Bäume), möchte

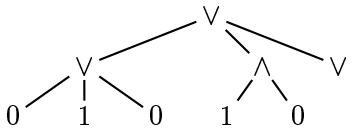
man nun das Konzept für unranked Alphabete erweitern. Ein möglicher Ansatz ist es, das Alphabet als Vereinigung von $\{\sigma/n \mid n \in \mathbb{N}\}$ für alle Zeichen σ aufzufassen. Da dies aber zu unendlich großen Alphabeten führt, betrachten wir hier einen anderen Ansatz.
Dazu machen wir folgende Änderungen an den Automaten [2]:

- Σ besteht aus einer Zeichenmenge, arity entfällt (d.h. jeder Knoten kann unabhängig seiner Markierung beliebig, aber endlich viele Nachfolger haben).
- Regeln aus Δ können auf der rechten Seite beliebige reguläre Ausdrücke über Q haben.

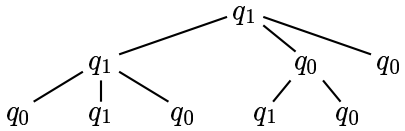
Beispiel: Automat für boolesche Ausdrücke mit Konjunktion und Disjunktion mit beliebig vielen Argumenten

$\Sigma = \{\vee, \wedge, 0, 1\}$,
Q=$\{q_0, q_1\}$,
F=$\{q_1\}$,
$\Delta = \{$
$(q_0, 0) \to \epsilon$,
$(q_1, 1) \to \epsilon$,
$(q_0, \wedge) \to (q_0 \cup q_1)^* q_0 (q_0 \cup q_1)^*$,
$(q_1, \wedge) \to 1^*$,
$(q_0, \vee) \to 0^*$,
$(q_1, \vee) \to (q_0 \cup q_1)^* q_1 (q_0 \cup q_1)^*\}$

Baum:



erfolgreicher Lauf:



5.1 Anwendung : XPATH Anfragen durch Unranked Automat

Unranked Automaten können dazu benutzt werden, in einem Baum nach gewissen Eigenschaften zu suchen, dazu gehören z.B. XPATH-Anfragen. XPATH ist eine Anfrage-Sprache für XML-Dokumente. Die Syntax wird an folgendem Beispiel erklärt:
Anfrage /a //b[/a] //b bedeutet
' Suche Wurzel a, die in einem Unterbaum einen Knoten b hat, der irgendwo in einem Unterbaum einen Knoten b hat und als Nebenbedingung ein Kind a hat '.

- /a ist erfüllt, wenn es ein Kind mit Label a gibt (am Anfang einer Anfrage bezieht sich dies auf die Wurzel)
- //a ist erfüllt, wenn es in irgendeinem Unterbaum einen Knoten mit Label a gibt
- Eckige Klammern [Anfrage] beschreiben eine Nebenbedingung des aktuellen Knotens, es kann eine komplette XPATH Anfrage enthalten sein

Bemerkungen:

- In eckigen Klammern können eigentlich auch Bedingungen an Attribute stehen. Da Attribute aber in der Baumdarstellung auch als Kinder angesehen werden können, muss man dies nicht extra betrachten (aus /a[@id=1] wird die Anfrage /a[/id/1]).

[2] erste Arbeiten zu unranked Automaten gehen auf Thatcher(1967) zurück

- Ausdrücke sind rechtsassoziativ geklammert, d.h. Bedingungen gelten immer für den gerade angegebenen Knoten.

Der Automat liefert als Ergebnis, ob der Eingabebaum die XPATH-Anfrage erfüllt.

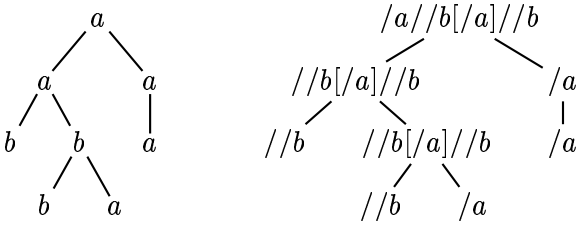
Konstruktion eines Automaten für eine Anfrage

Die Zustandsmenge des Automaten ist die Menge aller Teilausdrücke in der Anfrage, F ist die Menge mit der Anfrage als Element.
Ein Zustand an einem Knoten bedeutet, dass der entsprechende XPATH-Ausdruck für den Knoten darüber gilt; der Zustand der Wurzel gibt einen Ausdruck an, der für das gesamte Dokument gilt. Die Zustandsmenge kann dadurch vergrößert werden, dass Neben- und Hauptbedingung auch vom selben Knoten erfüllt werden können (s. folgendes Beispiel).
Automatenkonstruktion für folgende Anfrage: /a //b[/a] //b

Regelmenge:

(/a//b[/a]//b , a)	$\rightarrow Q^*$ (//b[/a] //b) Q^*	Wenn für irgendeinen Unterknoten (//b[/a] //b) gilt und ein a gelesen wird, dann gilt /a//b[/a]//b.
(//b[/a]//b, b)	$\rightarrow Q^*$ (//b[/a] //b) $Q^* \cup Q^*/aQ^*//bQ^* \cup Q^*//bQ^*/aQ^* \cup Q^*(/a//b)Q^*$	Neben- und Hauptbedingung gelten, wenn es für jede der Bedingungen einen Unterknoten gibt, der sie erfüllt, dies kann auch derselbe Knoten sein.
(/a//b, a)	$\rightarrow Q^*$ //b Q^*	/a//b gilt, wenn man a liest und ein Unterknoten //b erfüllt.
(//b, a)	$\rightarrow Q^*$ //b Q^*	Wenn man ein a liest, gilt //b, falls es bereits für einen Unterknoten gilt.
(//b, b)	$\rightarrow Q^*$	Wenn man b liest, gilt //b unabhängig von den Kindern.
(/a, a)	$\rightarrow Q^*$	a impliziert /a .

Beispielbaum und erfolgreicher Lauf:



Eine schwierige Stelle bei der Regelgenerierung ist der Fall, in dem eine Neben- und Hauptbedingung für denselben Unterknoten eintreten sollen.
Im folgenden wird angegeben, welche Bedingung dieser Unterknoten erfüllen muss, damit der Ausdruck wahr ist, d.h. für jede Bedingung kommt Q^* BEDINGUNG Q^* auf die rechte Seite

der Übergangsregel.
Dazu muss man im allgemeinen 3 Fälle betrachten :

1. Nebenbedingung (NB) [/aEXPR1] Hauptbedingung (HB) /bEXPR2

2. NB [/aEXPR1] HB //bEXPR2

3. NB [//aEXPR1] HB //bEXPR2

wobei b und a gleich sein können; EXPR1 und EXPR2 sind beliebige XPATH-Ausdrücke oder leere Bedingungen.
Der 4.Fall NB [//aEXPR1] HB /bEXPR2 kann durch Vertauschen von Haupt- und Nebenbedingung abgehandelt werden.

- Fall 1:
  Dies kann nur von einem Knoten erfüllt sein, wenn a=b. Der Zustand des Knotens muss dann /a[EXPR1]EXPR2 sein.

- Fall 2:
  Der Zustand des Knotens muss /a[EXPR1]//bEXPR2 sein oder /a[EXPR1]EXPR2, falls a=b.

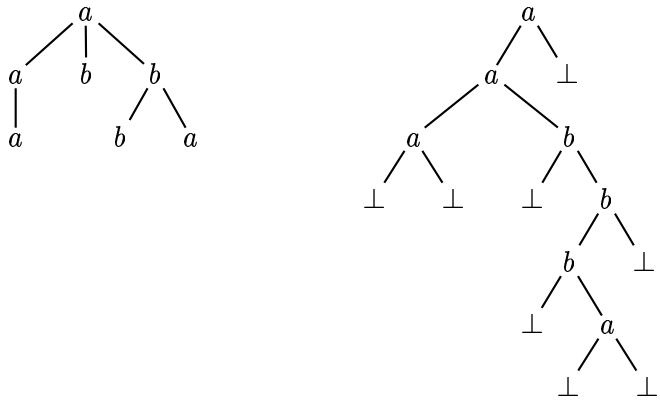
- Fall 3:
  Der Zustand des Knotens muss entweder //a[EXPR1]//bEXPR2 oder //b[EXPR2]//aEXPR1 sein. Hinzu kommt noch die Möglichkeit //a[EXPR1]EXPR2, falls a=b.
  So können die benötigten Zustände und die Regeln für Nebenbedingungen berechnet werden.

5.2 Unranked Baum $\rightarrow$ binärer Baum

Unranked Bäume können in binäre Bäume kodiert werden.

- alle Symbole aus Σ werden zu zweistelligen Funktionsoperatoren
- $\perp$ als neue Konstante für Blätter
- Kodierung : links unter einen Knoten werden seine Unterbäume kodiert, rechts nächstes Kind des Vaters

Unranked Baum mit Kodierung :



6 Automaten auf Baumtupeln

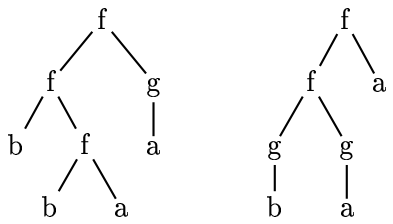
Analog zu den endlichen Automaten können auch Baumautomaten auf das Erkennen von Baumtupeln erweitert werden. Dabei ist als Schwierigkeit zu beachten, dass Bäume völlig unterschiedlichen Shape haben können.

6.1 Anpassung von Shapes

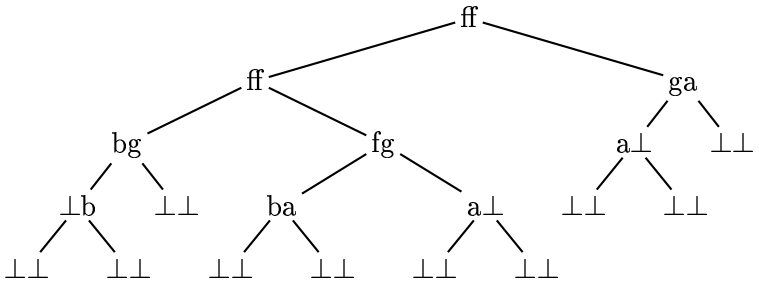
Wir betrachten hier nur binäre Bäume. Dass Verfahren im vorigen Kapitel kann bei Bäumen mit mehr Nachfolgern dazu benutzt werden, allgemeine Bäume in binäre Bäume umzuwandeln.
Der Shape des Tupelbaumes ist die Vereinigung der einzelnen Shapes. Die Knoten werden mit den Labels aller Bäume an der Stelle beschriftet. Wenn ein Baum an einer Stelle keinen Eintrag mehr hat, so wird dies mit $\perp$ signalisiert. Das Alphabet des Tupelbaumes ist dann das Kreuzprodukt der einzelnen Alphabete vereinigt mit $\perp$. Alle Tupel werden binäre Funktionssymbole, ausser dem Tupel, in dem nur $\perp$ enthalten ist. Dies wird die Konstante zum Anzeigen des Baumendes. Daher muss man zum Shape noch für jedes Blatt zwei Kinder hinzufügen.

Beispiel:

Sei f binär, g unär und a Konstante.
Gegeben die beiden Bäume



Der Tupelbaum sieht wie folgt aus :



Das erste Zeichen des Labels gibt die Beschriftung des ersten Baumes an dieser Stelle an, das zweite das des Zweiten.
Nun können Automaten für diese Tupel wie vorher angegeben werden.

Einfaches Beispiel für Tupelautomaten

Gesucht ist ein Automat, der die Sprache $\{[t,t] |$ t ist Σ-Baum $\}$ erkennt mit $\Sigma = \{f/2, g/1, a/0\}$.
Der folgende Automat löst dieses Probelm :
Alphabet des Automaten Σ' = $(\Sigma \cup \{\bot\})$ x $(\Sigma \cup \{\bot\})$ =
$\{ff/2, fg/2, fa/2, f\bot/2, gf/2, gg/2, ga/2, g\bot/2, af/2, ag/2, aa/2, a\bot/2, \bot f/2, \bot g/2, \bot a/2, \bot\bot/0\}$
Q = $\{q\}$ und F = $\{q\}$

Δ = {
(q,⊥⊥) → ϵ
(q, aa) → (q, q)
(q, gg) → (q, q)
(q, ff) → (q, q) }

Zylindrifikation und Projektion

An verschiedenen Stellen kann es vorkommen, dass man das Tupel um eins erweitern oder verkleinern will, ohne die eigentliche Sprache zu verändern (z.B. bei der Anwendung von Tupelautomaten für WS2S-Formeln, beim Abarbeiten von Existenz- oder Allquantoren)

Zylindrifikation an Stelle i: Füge für jede Regel (q, $s_1, \ldots s_{i-1}, s_i, \ldots s_k$) → REGEXP für jedes $\sigma \in \Sigma$ eine Regel (q, $s_1, \ldots s_{i-1}, \sigma, s_i, \ldots s_k$) → REGEXP ein. Dadurch bleibt die Sprache an sich gleich (d.h. wenn man sie auf die vorherigen Tupel beschränkt), die Regelmenge vergrößert sich aber um den Faktor $|\Sigma|$. Ausserdem wird ein bottom-up nichtdeterministischer Automat erzeugt. War der Ausgangsautomat top-down deterministisch, so bleibt diese Eigenschaft erhalten.

Projektion an Stelle i: Füge für Regeln
(q, $s_1, \ldots s_{i-1}, \sigma_1, s_{i+1}, \ldots s_k$) → $REGEXP_1$
⋮
(q, $s_1, \ldots s_{i-1}, \sigma_m, s_{i+1}, \ldots s_k$) → $REGEXP_m$
eine Regel (q, $s_1, \ldots s_{i-1}, s_{i+1}, \ldots s_k$) → $REGEXP_1 \cup \ldots \cup REGEXP_m$ hinzu.
Dadurch wird die Regelmenge verkleinert, der Automat ist i.A. top-down nichtdeterministisch, bottom-up deterministische Automaten bleiben bottom-up deterministisch.

Referenzen

- H.Comon, M.Dauchet, R.Gilleron, F.Jacquemard, D.Lugiez, S.Tison, M.Tommasi: *Tree Automata and its Applications* Kapitel 1 und 3
  Online Version: www.grappa.univ-lille3.fr/tata (1997, überarbeitete Fassung 2002)
- Frank Neven: *Automata Theory for XML Researchers*
  University of Limburg, 2002
- R.Wilhelm, D.Maurer *Übersetzerbau*
  Springer Verlag 1992 (2.Version 1997)