Radboud Universiteit, Nijmegen

Bachelorscriptie Informatica

Twee manieren voor het leren van een DTD bij voorbeelden van XML-documenten

Tom Evers
Januari 2016
Begeleider: Herman Geuvers
Tweede lezer: Frits Vaandrager

Inhoud

1 Inleiding

In 1996 is onder toezicht van het World Wide Web Consortium (W3C) het SGML Editorial Review Board opgericht [1]. Deze groep, later bekend als de XML Working Group, heeft de zogenaamde Extendible Markup Language ontworpen. Daarbij hadden ze een aantal doelstellingen. Zo moest XML bijvoorbeeld gemakkelijk bruikbaar zijn over het hele internet en een grote diversiteit aan programma's ondersteunen. Inmiddels wordt XML voor vanalles gebruikt, van webapplicaties tot specialistische software in fabrieksprocessen.

De syntaxis van XML-talen beschrijft de posities van *tags* ten opzichte van elkaar, aan de hand van een *document type definition (DTD)*. Een DTD bevat als het ware een beschrijving van welke tags in welke volgorde een "geldig" document vormen. Wanneer je attributen negeert, is een DTD eigenlijk een speciale vorm van een context-vrije grammatica. Jean Berstel en Luc Boasson hebben zulke grammatica's bestudeerd en geformaliseerd als XML-grammatica's. [2]

Soms is een DTD niet optimaal, bijvoorbeeld te groot en daardoor onleesbaar. In andere gevallen bestaat een DTD nog niet, maar zijn de ontwerpers van een XML-taal zelf niet bedreven genoeg om een grammatica te ontwerpen. In zulke gevallen is het handig om DTD's af te kunnen leiden aan de hand van XML-documenten zelf.

Hennig Fernau heeft hier veel onderzoek naar gedaan. Allereerst heeft hij een manier gevonden om een XML-document om te schrijven naar een set voorbeelden voor reguliere talen [3]. Vervolgens heeft hij twee manieren onderzocht om uit deze reguliere talen de gewenste grammatica af te leiden en zo hun DTD's te vinden. [3] [4]

Het omschrijven van een XML-taal naar een set van reguliere talen zal worden behandeld in sectie 2. In sectie 0 zal worden behandeld waarom sommige reguliere talen af te leiden zijn uit een eindig aantal voorbeelden van woorden die in de taal zitten. De twee manieren om dit te bereiken zullen woorden beschreven in secties 4 en 5, met een korte uitleg vooraf in sectie 3.3.

2 XML-talen naar reguliere talen

(1) **Voorbeeld:**

```
<inventaris>
  <boek>
      <schrijver>
          <voornaam>Tom</voornaam>
          <achternaam>Evers</achternaam>
      </schrijver>
      <titel>Scriptie schrijven voor dummies</titel>
      <prijs>€5,00</prijs>
  </boek>
  <boek>
      <schrijver>
          <voorletter>J</voorletter>
          <achternaam>Berstel</achternaam>
      </schrijver>
      <schrijver>
          <voorletter>L</voorletter>
          <achternaam>Boasson</achternaam>
      </schrijver>
      <titel>XML Grammars</titel>
      <prijs>€20,00</prijs>
  </boek>
</inventaris>
```

Het hierboven weergegeven stuk code is een voorbeeld van hoe de inventaris van een online boekenwinkel er uit zou kunnen zien. Voor het vinden van een grammatica bij dit stuk XML, is het zaak dat we het eerst omschrijven naar een set voorbeeldwoorden I_+, waarop we uiteindelijk onze algoritmes kunnen toepassen.
Daarbij komen er waarschijnlijk meteen een aantal vragen in je op. Hoe ziet een woord in een resulterende reguliere taal eruit? Als <boek> een letter is in een woord, bijvoorbeeld b, hoe behoud je dan het verband tussen b en de sluitende tag </boek>?
Antwoorden op deze vragen zijn gemakkelijker te vinden als gebruik wordt gemaakt van een tussenstap. Daarvoor moeten we eerst precies weten hoe een XML-grammatica is opgebouwd.

2.1 Een XML-grammatica

Berstel en Boasson geven in [2] de volgende formalisatie voor een XML-grammatica.

(2) **Definitie:** Een XML-grammatica G is een tupel $\langle A \cup \bar{A}, V, \{R_a \mid a \in A\}\rangle$ met:
- Terminaal alfabet $T = A \cup \bar{A}$ met $\bar{A} = \{\bar{a} \mid a \in A\}$
- Set variabelen $V = \{X_a \mid a \in A\}$
- Een speciale variabele, het *axioma* X met de speciale regel $X \to X_a, X_a \in V$
- Set reguliere expressies $R_a \subseteq V^*$ met $\forall_{a\in A}, \forall_{m\in R_a}, X_a \to am\bar{a}$

(3) **Definitie:** Voor een woord $w \in T^*$ en een XML-grammatica $G = \langle A \cup \bar{A}, V, \{R_a \mid a \in A\}\rangle$ geldt: G *genereert* w als $\exists_{a\in A}, X_a \to w$

(4) **Definitie:** $L \subseteq T^*$ wordt *gegenereerd* door XML-grammatica G wanneer:
$L = \{w \in T^* \mid G\ genereert\ w\}$ hetgeen genoteerd wordt als $L = L(G)$

(5) **Definitie:** $L \subseteq T^*$ is een *XML-taal* als er een XML-grammatica G is zodat $L = L(G)$

Laten we voorbeeld (1) gebruiken om deze definitie te verhelderen. Het eerste dat we nodig hebben is een alfabet. Hiervoor kiezen we een handige afkorting voor iedere verschillende tag, zodat we de volgende verzameling krijgen:

$$A = \{i, b, s, v, l, a, t, p\}$$

Aangezien iedere tag ook nog een sluitend component heeft, hebben we vanzelf ook:

$$\bar{A} = \{\bar{a} \mid a \in A\} = \{\bar{\imath}, \bar{b}, \bar{s}, \bar{v}, \bar{l}, \bar{a}, \bar{t}, \bar{p}\}$$

Het terminale alfabet T bestaat dus uit de volgende tekens:

$$T = A \cup \bar{A} = \{i, \bar{\imath}, b, \bar{b}, s, \bar{s}, v, \bar{v}, l, \bar{l}, a, \bar{a}, t, \bar{t}, p, \bar{p}\}$$

De tekst tussen de tags wordt genegeerd[1] in XML-grammatica's, dus het bovenstaande voorbeeld kunnen we als $ibsv\bar{v}a\bar{a}\bar{s}t\bar{t}p\bar{p}\bar{b}bsl\bar{l}a\bar{a}\bar{s}sl\bar{l}a\bar{a}\bar{s}t\bar{t}p\bar{p}\bar{b}\bar{\imath}$ opschrijven.

Vervolgens hebben we voor ieder element in A een variabele nodig, waarmee we kunnen beschrijven wat er allemaal tussen die tags kan staan. Formeel bepalen we dus als volgt de verzameling variabelen:

$$V = \{X_a \mid a \in A\}$$

Er is een speciale variabele X, het axioma. In DTD's wordt dit axioma vaak aangegeven in DOCTYPE-tags. Het axioma kun je zien als de belangrijkste variabele, de "root" van de grammatica: een XML-document begint en eindigt met tags van de a bij deze variabele.

$$X \to X_a$$

Iedere variabele X_a lijkt in essentie een reguliere taal te beschrijven, van alleen woorden die beginnen met een a en eindigen met een $\bar{a}$. Voor alles daartussen is er voor iedere $a \in A$ een verzameling mogelijkheden. Die verzameling is een subset van alles wat je kunt bouwen van alle variabelen uit V die je hebt. Dus geldt:

$$R_a \subseteq V^*$$
$$\forall_{a\in A}, \forall_{m\in R_a}, X_a \to am\bar{a}$$

Of, afgekort:

$$X_a \to aR_a\bar{a}$$

[1] Fernau heeft het in zijn artikel [3] over het symbool τ als "placeholder" voor arbitraire tekst. Dit gebruikt hij echter alleen in een voorbeeld van een XML-grammatica. Hierin gedraagt de τ zich min of meer als een λ: als de verzameling R_a voor een zekere $a \in A$ leeg is, dan zal er tussen de tags alleen tekst staan: $X_a \to a\tau\bar{a}$. Ook geldt hierbij $\tau\tau = \tau$.
In deze scriptie is ervoor gekozen de tekst volledig buiten beschouwing te laten. Daarbij nemen we aan dat als er tekst tussen tags staat, dat het enige is dat tussen die tags kan staan. Met andere woorden: R_a kan nooit τ bevatten.
Deze aanname lijkt Fernau ook te maken, aangezien hij nergens aankaart dat door tekst volledig te negeren, er wanneer je afgaat op een XML-grammatica, in principe op iedere plek in een XML-taal ruimte is voor arbitraire tekst.

Het voorbeeld van de inventaris in (1) zou met deze kennis al als XML-grammatica weer te geven moeten zijn. Zie daarvoor (6) hieronder.

(6) **Voorbeeld:** $X_i \to iR_i\bar{\imath}$ met $R_i = \{X_bX_b\}$
$X_b \to bR_b\bar{b}$ met $R_b = \{X_sX_tX_p, X_sX_sX_tX_p\}$
$X_s \to sR_s\bar{s}$ met $R_s = \{X_vX_a, X_lX_a\}$
$X_v \to v\bar{v}$
$X_l \to l\bar{l}$
$X_a \to a\bar{a}$
$X_t \to t\bar{t}$
$X_p \to p\bar{p}$

Het eerder genoemde verband tussen regels in XML-grammatica's en reguliere talen gaan we iets duidelijker proberen te leggen, zodat we uiteindelijk een XML-taal kunnen vangen in een contextvrije grammatica. Daarvoor zullen we eerst nog een aantal begrippen moeten introduceren. Uiteindelijk zullen we afzonderlijke woorden kunnen vinden die we als voorbeelden kunnen gebruiken voor een algoritme dat deze talen leert.

2.2 Woorden isoleren

2.2.1 Dyck primes

(7) **Definitie:** Taal D_A met $A = \{a_1, \dots, a_n\}$ uit alfabet $T = \{A \cup \bar{A}\}$ met axioma X is de taal van de zogeheten *Dyck primes*, en wordt gegenereerd door:
$X \to X_{a_1} \mid \dots \mid X_{a_n}$
$\forall_{a \in A}, X_a \to a\left(X_{a_1} \mid \dots \mid X_{a_n}\right)^* \bar{a}$

Berstel en Boasson [2] hebben het volgende bewezen:

(8) **Lemma:** Als L een XML-taal is, geldt $L \subseteq D_A$

Dyck primes zijn dus een belangrijk concept. Hierbij is het belangrijk om op te merken dat de taal D_A op zichzelf geen XML-taal is. Iedere variabele X_{a_i} genereert daarintegen de XML-taal D_{a_i}.

(9) **Definitie:** $D_{a_i} \coloneqq D_A \cap a_i(A \cup \bar{A})^*\bar{a}_i$
D_{a_i} is de taal van alle woorden in de taal van de *Dyck primes* die begonnen en afgesloten worden met de tag a_i.

(10) **Lemma:** D_{a_i} is een XML-taal voor alle $a_i \in A$ en $1 \leq i \leq n$

De taal D_A met ons voorbeeldalfabet zou er als volgt uitzien:

$$X \to X_i \mid X_b \mid X_s \mid X_v \mid X_l \mid X_a \mid X_t \mid X_p$$
$$X_i \to i\left(X_i \mid X_b \mid X_s \mid X_v \mid X_l \mid X_a \mid X_t \mid X_p\right)^*\bar{\imath}$$
$$X_b \to b\left(X_i \mid X_b \mid X_s \mid X_v \mid X_l \mid X_a \mid X_t \mid X_p\right)^*\bar{b}$$
$$X_s \to s\left(X_i \mid X_b \mid X_s \mid X_v \mid X_l \mid X_a \mid X_t \mid X_p\right)^*\bar{s}$$
$$\dots$$

Het woord bij voorbeeld (1) zit in de taal D_i.

$$ibsv\bar{v}a\bar{a}\bar{s}t\bar{t}p\bar{p}\bar{b}bsl\bar{l}a\bar{a}\bar{s}sl\bar{l}a\bar{a}\bar{s}t\bar{t}p\bar{p}\bar{b}\bar{\imath} \in D_i$$

In D_b zijn er al twee gevallen bekend.

$$bsv\bar{v}a\bar{a}\bar{s}t\bar{t}p\bar{p}\bar{b}, bsl\bar{l}a\bar{a}\bar{s}sl\bar{l}a\bar{a}\bar{s}t\bar{t}p\bar{p}\bar{b} \in D_b$$

2.2.2 Factoren

Om uit een langer woord verschillende kleinere woorden van andere talen te halen, introduceren we een nieuw concept.

(11) **Definitie:** De set van *factoren* van $L \subseteq \Sigma^*$ is $F(L) = \{y \in \Sigma^* \mid \exists_{(x,z\in\Sigma^*)}, xyz \in L\}$

Aannemende dat $\Sigma = T$ en $L = \{ibs\ \bar{}a\bar{a}\bar{s}t\bar{t}p\bar{p}\bar{b}bsl\bar{l}a\bar{a}\bar{s}sl\bar{l}a\bar{a}\bar{s}t\bar{t}p\bar{p}\bar{b}\bar{i}\}$, is hier een voorbeeld van elementen in $F(L)$:

$$\{i, b, s, ib, bs, sv, v\bar{v}, \bar{a}\bar{s}t\bar{t}p, sl\bar{l}a\bar{a}\bar{s}, bsl\bar{l}a\bar{a}\bar{s}\} \subset F(L)$$

$F(L)$ bevat alle combinaties van letters die ergens in die volgorde in een woord van L staan. Daaruit volgt dat $F(L)$ ook die woorden bevat die beginnen en eindigen met bij elkaar horende tags. Voor iedere tag beschrijven we deze verzameling zoals gedefinieerd in (12).

(12) **Definitie:** De set van factoren van $L \subseteq (A \cup \bar{A})$ die Dyck primes zijn, beginnend met de letter a is $F_a(L) = D_a \cap F(L)$

In principe bevat I_+ alleen woorden die beginnen met de meest alomvattende tag, het axioma. In het geval van voorbeeld (1) zijn dat woorden in D_i met $i \in A$. Deze woorden bevatten op hun beurt kleinere woorden die voorbeelden zijn voor andere talen D_{a_i} bij tags a_i. Nu hebben we dus een manier om voor die andere talen voorbeelden te vinden uit de grote woorden in I_+.

Een paar sets van factoren bij voorbeeld (1) zouden zijn:

$F_i(L) = \{ibsv\bar{v}a\bar{a}\bar{s}t\bar{t}p\bar{p}\bar{b}bsl\bar{l}a\bar{a}\bar{s}sl\bar{l}a\bar{a}\bar{s}t\bar{t}p\bar{p}\bar{b}\bar{i}\}$
$F_b(L) = \{bsv\bar{v}a\bar{a}\bar{s}t\bar{t}p\bar{p}\bar{b}, bsl\bar{l}a\bar{a}\bar{s}sl\bar{l}a\bar{a}\bar{s}t\bar{t}p\bar{p}\bar{b}\}$
$F_s(L) = \{sv\bar{v}a\bar{a}\bar{s}, sl\bar{l}a\bar{a}\bar{s}\}$

2.2.3 Sporen en oppervlakken

Terug naar het alfabet $A = \{a_1, \dots, a_n\}$ en de bijbehorende taal D_A. Voor een zekere $a \in A$ kan een woord $w \in D_a$ ontleed worden.

(13) **Definitie:** Woord $w \in D_a$ kan worden *ontleed*. De ontleding van w is
$d(w) = au_{a_1}u_{a_2} \dots u_{a_n}\bar{a}$ met $1 \leq i \leq n$ en $u_{a_i} \in D_{a_i}$

Merk hierbij op dat geldt $d(w) = w$. Voor ieder woord w hoort de volgorde van deelwoorden u, die uit $d(w)$ kan worden gevonden. Deze reeks noemen we een *spoor*.

(14) **Definitie**: Het *spoor* van woord $w \in D_a$ is
$s(w) = a_1 \dots a_n \in A^*$ met $d(w) = au_{a_1}u_{a_2} \dots u_{a_n}\bar{a}$

De verzameling van alle sporen voor een bepaalde $a \in A$ die aanwezig zijn in een gegeven taal L, noemen we het oppervlak van a.

(15) **Definitie:** *Oppervlak* $S_a(L)$ is de verzameling van alle sporen van woorden in $F_a(L)$

Zie sectie 2.3 voor de berekening van oppervlakken in voorbeeld (1).

Oppervlakken blijken handig bij het definiëren van een XML-grammatica bij een document $\mathcal{D}$. Stel, we hebben bij $\mathcal{D}$ een familie $\mathcal{S} = \{S_a \mid a \in A\}$. Dit is een verzameling van alle oppervlakken die in $\mathcal{D}$ voorkomen. Deze familie bevat dus alle sporen van alle mogelijke woorden uit $\mathcal{D}$ met $T = \{A \cup \bar{A}\}$.

(16) **Definitie:** Standaardgrammatica $G_{\mathcal{S}}$ bij familie $\mathcal{S}$ wordt als volgt gegenereerd:

- Maak voor iedere letter in A een variabele, zodat uiteindelijk de set variabelen $V = \{X_a \mid a \in A\}$
- Vul voor iedere letter de bijbehorende verzameling reguliere expressies met alle sporen voor die letter:
  $\forall_{a\in A}, X_a \to aR_a\bar{a}$ met $R_a = \{X_{a_1} \dots X_{a_n} \mid a_1 \dots a_n \in S_a\}$

$G_{\mathcal{S}}$ is per definitie een XML-taal, welk axioma we ook kiezen. We verwijzen terug naar voorbeeld (6) voor de standaardgrammatica bij voorbeeld (1).

(17) **Lemma:** Als L een XML-taal is, bestaat er een standaard XML-grammatica die L beschrijft.

Er is dus een één-op-één overeenkomst tussen reguliere oppervlakken en XML-talen.

2.3 Voorbeeld (1) uitgewerkt

$F_i(L) = \{ibsv\bar{v}a\bar{a}\bar{s}t\bar{t}p\bar{p}\bar{b}bsl\bar{l}a\bar{a}\bar{s}sl\bar{l}a\bar{a}\bar{s}t\bar{t}p\bar{p}\bar{b}\bar{\imath}\}$
$x_1 = ibs\ \bar{}a\bar{a}\bar{s}t\bar{t}p\bar{p}\bar{b}bsl\bar{l}a\bar{a}\bar{s}sl\bar{l}a\bar{a}\bar{s}t\bar{t}p\bar{p}\bar{b}\bar{\imath}$
$d(x_1) = iu_bu_b\bar{\imath}$
$\mathfrak{s}(x_1) = bb$
$S_i(L) = \{bb\}$

$F_b(L) = \{bsv\bar{v}a\bar{a}\bar{s}t\bar{t}p\bar{p}\bar{b}, bsl\bar{l}a\bar{a}\bar{s}sl\bar{l}a\bar{a}\bar{s}t\bar{t}p\bar{p}\bar{b}\}$
$x_1 = bsv\bar{v}a\bar{a}\bar{s}t\bar{t}p\bar{p}\bar{b}$
$d(x_1) = bu_su_tu_p\bar{b}$
$\mathfrak{s}(x_1) = stp$
$x_2 = bsl\bar{l}a\bar{a}\bar{s}sl\bar{l}a\bar{a}\bar{s}t\bar{t}p\bar{p}\bar{b}$
$d(x_2) = bu_su_su_tu_p\bar{b}$
$\mathfrak{s}(x_2) = \text{sstp}$
$S_b(L) = \{\mathfrak{s}(x_1), \mathfrak{s}(x_2)\} = \{stp, sstp\}$

$F_s(L) = \{sv\bar{v}a\bar{a}\bar{s}, sl\bar{l}a\bar{a}\bar{s}\}$
$x_1 = sv\bar{v}a\bar{a}\bar{s}$
$d(x_1) = su_vu_a\bar{s}$
$\mathfrak{s}(x_1) = va$
$x_2 = sl\bar{l}a\bar{a}\bar{s}$
$d(x_2) = su_lu_a\bar{s}$
$\mathfrak{s}(x_2) = la$
$S_s(L) = \{va, la\}$

Aan de volgende sets van factoren is triviaal te zien dat hun oppervlak leeg is:
$F_v(L) = \{v\bar{v}\}$
$F_l(L) = \{l\bar{l}\}$
$F_a(L) = \{a\bar{a}\}$
$F_t(L) = \{t\bar{t}\}$
$F_p(L) = \{p\bar{p}\}$

3 Reguliere talen leren

3.1 Leren in de limiet

Dana Angluin [5] geeft in haar paper over het afleiden van wat ze noemt “reversible languages” (omkeerbare talen) een goede uitleg over de moeilijkheden bij het leren van reguliere talen.
Gegeven een set strings, bijvoorbeeld $\{00011,011,001,001111\}$, zouden we een gok kunnen doen naar de taal waaruit ze komen. Deze set strings doet vermoeden dat we hier te maken hebben met een taal waarvan ieder woord eerst een aantal nullen bevat, en vervolgens een aantal enen.

$$I_+ = \{00011,011,001,001111\}$$
$$L(I_+) = 0^+1^+$$

Natuurlijk is er een oneindig aantal reguliere talen dat dit voorbeeld bevat, en is er dus geen enkele manier om te weten of zo’n vermoeden klopt. Wat we missen, is een criterium om te kunnen bepalen hoe “goed” een vermoeden is. Mark Gold [6] heeft hiervoor een belangrijk concept geformuleerd.
Stel dat je aan een set voorbeelden van een regel steeds maar woorden blijft toevoegen. Uiteindelijk bevat die set ieder voorbeeld voor een bepaalde regel. Als je voor zo’n set een afleidingsproces kunt geven, dat uiteindelijk de onderliggende regel van de verzameling voorbeelden vindt, dan kan dat proces de regel *leren in de limiet*. Golds leerconcept is als volgt geformaliseerd door Fernau [3].
Laat $\mathcal{L}$ een klasse van talen zijn die gedefinieerd wordt door een klasse taalbeschrijvende apparaten $\mathcal{D}$, bijvoorbeeld automaten of grammatica’s. $\mathcal{L}$ is identificeerbaar wanneer er een inferentiemachine IM bestaat met de volgende eigenschappen:

- $L \in \mathcal{L}$ kan worden opgesomd, mogelijk met herhalingen. Neem bijvoorbeeld enumeratie $E: \mathbb{N} \to L$
- IM krijgt de oneindige invoerstroom $E(1), E(2), \ldots$
- IM geeft daarop een uitvoerstroom van apparaten $D_i \in \mathcal{D}$
- Er is een $N(E)$ zodat $\forall_{n \geq N(E)}, D_n = D_{N(E)} \wedge L\left(D_{N(E)}\right) = L$

Met $L = \{w_1, w_2, w_3\}$ zou een mogelijke opsomming kunnen zijn:

$$E(1) = w_2, E(2) = w_1, E(3) = w_1, E(4) = w_3, E(5) = w_1, \ldots$$

Er is inderdaad een $N(E)$ die voldoet aan de voorwaarden. Met $N(E) = 4$ zou gelden dat $L(D_4) = \{w_1, w_2, w_3\}$, en dus dat bijvoorbeeld $D_5 = D_4$ met $L(D_5) = L = \{w_1, w_2, w_3\}$.
Omdat $E(n)$ met $n \geq 4$ alleen nog maar herhalingen bevat van $E(1)$, $E(2)$ en $E(4)$ zal de voorwaarde $\forall_{n \geq 4}, D_n = D_4 \wedge L(D_4) = L$ ook blijven gelden.

3.2 Leerbare talen

Er is veel onderzoek gedaan naar klassen van talen die identificeerbaar in de limiet zijn. Zo heeft Radhakrishnan in zijn artikelen [7] [8] onderzoek gedaan naar de zogenaamde terminal-distinguishable languages (terminaal-onderscheidbare talen). Hij toont aan dat deze klasse van talen valt te leren in de limiet. In sectie 4 maken we gebruik van deze eigenschap.
Fernau schrijft over de zogenaamde simple-looping languages (talen met simpele lussen) [4]. In sectie 5 zullen we hierover uitwijden. Ook Angluin heeft het in haar paper over een identificeerbare subklasse van reguliere talen, de k-omkeerbare talen [5]. In deze scriptie zullen we het daar uit ruimte-overwegingen niet over hebben.
In sectie 2 hebben we beschreven hoe een XML-document kan worden omgezet naar een set voorbeeldwoorden voor reguliere talen. Nu zullen we twee manieren beschrijven om uit deze voorbeelden de bijbehorende reguliere expressies te vinden, en daarmee een DTD bij de XML-documenten.

(18) **Opmerking:** Deterministische eindige automaten (DFA’s) hebben in dit werk een iets andere definitie dan gebruikelijk is. In de klassieke zin heeft een DFA altijd *precies één* transitie per teken uit het inputalfabet. In deze scriptie hoeft transitiefunctie van een DFA echter niet altijd in zijn geheel gedefinieerd te zijn. Een DFA met een tweeletterig inputalfabet kan bijvoorbeeld states bevatten met maar één uitgaande transitie. Zo’n automaat kan dan worden aangevuld tot een klassieke DFA, door vanuit zulke states een extra transitie te

maken naar iets als een eeuwig lussende niet-final state. Hierbij moet wel gelet worden op het volgende:
Sommige definities die in de volgende secties worden opgeschreven, leggen aan DFA's bepaalde voorwaarden op. Voldoet een DFA aan deze voorwaarden, dan valt deze binnen een speciale klasse. Deze voorwaarden zouden vaak wel gelden over de zojuist beschreven afwijkende DFA, maar niet over de klassieke DFA die het resultaat is van een toevoeging van eeuwige lussen. Bepalen of aan de gestelde voorwaarden wordt voldaan, doen we in deze scriptie daarom aan de hand van de afwijkende DFA.

(19) Definitie: DFA's in deze scriptie hebben *maximaal één* uitgaande transitie voor ieder teken uit het inputalfabet.

3.3 Twee methoden voor XML

3.3.1 f-onderscheidbaar

Fernau bekent dat "de klasse van alle XML-talen (over een vast alfabet) niet identificeerbaar is." ([3], *lemma 5*). Het vinden van identificeerbare subklassen van XML-talen is volgens hem toch mogelijk, door de één-op-één overeenkomst tussen XML-grammatica's en reguliere oppervlakken. In deze scriptie is die overeenkomst beschreven in lemma (17). In sectie 3.2 hierboven vertellen we over de vele pogingen die zijn gedaan om zulke leerbare subklassen te vinden. Fernau legt zich in zijn paper daarom toe op het introduceren van een zogenaamd raamwerk om zulke klassen te generaliseren.

In sectie 4 zullen we het hebben over dit raamwerk. We gebruiken de onderscheidende functies waarop het is gebaseerd bij het vinden van een grammatica uit een set van XML-documenten.

3.3.2 Simple-looping

In zijn paper uit 2009 [4] beschrijft Fernau een hele andere methode om talen te leren in de limiet, namelijk aan de hand van zijn algoritme: SL-*infer*. Pas vrij aan het eind van zijn publicatie schrijft hij dat dit algoritme ook zou kunnen worden gebruikt voor het leren van DTD's bij XML-documenten.

In sectie 5 zullen we over SL-infer uitwijden, en het testen op een voorbeeld in de praktijk.

4 F-onderscheidbaar

4.1 f-onderscheidbare talen

(20) **Definitie:** Laat F een eindige verzameling zijn. $f: T^* \to F$ is een *onderscheidende functie* als geldt: $\forall_{u,w,z \in T^*}, f(w) = f(z) \to f(wu) = f(zu)$

(21) **Definitie:** Laat f een onderscheidende functie zijn. Twee woorden $w, z \in T^*$ zijn *f-gelijk* als geldt $f(w) = f(z)$

Intuitief gezien zegt definitie (20) het volgende. Soms "ziet" een onderscheidende functie geen verschil tussen twee woorden w en z in T^*. Wanneer dit zo is, dan zal het resultaat van een verlenging van de twee woorden met dezelfde combinatie letters (wu en zu met $u \in T^*$) ook niet uit elkaar te houden zijn.

(22) **Definitie:** Laat f een onderscheidende functie zijn. $L \subseteq T^*$ is *f-onderscheidbaar* als geldt: $\forall_{u,v,w,z \in T^*}, f(w) = f(z) \wedge \{wu, wv\} \subseteq L \to (zu \in L \leftrightarrow zv \in L)$

Wanneer slechts één van de twee verlengingen van z in de taal zit, bijvoorbeeld zu, dan zou er een duidelijk onderscheid te maken zijn tussen zv (niet in de taal) en wv (wél in de taal). Door definitie (22) is dat onmogelijk, wil die taal f-onderscheidbaar zijn.

Daarin bouwt definitie (22) als het ware voort op (20). Stel dat een onderscheidende functie f twee woorden w en z in T^* niet uit elkaar kan houden. Wanneer twee verleningen van één van die woorden (wu en wv met $u, v \in T^*$) beiden in de taal zitten, zal f geen onderscheid meer kunnen maken tussen wu en zu of wv en zv. In een f-onderscheidbare taal L kan het dus niet zo zijn dat zowel $wu \in L$ als $wv \in L$ (beide verlengingen in L), maar tegelijkertijd $zu \notin L$ óf $zv \notin L$ (één van beide verlengingen niet in L).

(23) **Definitie:** Laat f een onderscheidende functie zijn. De familie van talen $L \subseteq T^*$ die onderscheidbaar zijn met functie f schrijven we op als (f, T)-DL

(24) **Voorbeeld:** $f(x) = \mathrm{Ter}(x) = \{a \in T \mid \exists_{u,v \in T^*}, uav = x\}$

Voorbeeld (24) hierboven houdt in dat wanneer als onderscheidende functie $\mathrm{Ter}(x)$ wordt gekozen, de familie (Ter, T)-DL die is van de Ter-onderscheidbare talen is. Deze klasse komt volgens Fernau overeen met die van de terminal-distinguishable talen die Radhakrishnan beschrijft in zijn papers uit 1987 [7] [8]. Een paar voorbeelden van het toepassen van deze functie: $\mathrm{Ter}(a) = \{a\}$, $\mathrm{Ter}(aaa) = \{a\}$, $\mathrm{Ter}(ab) = \{a, b\}$ en $\mathrm{Ter}(aababac) = \{a, b, c\}$.

4.2 f-onderscheidbare automaten

Niet alleen een taal kan f-onderscheidbaar zijn. Ook voor een automaat kan het een eigenschap zijn.

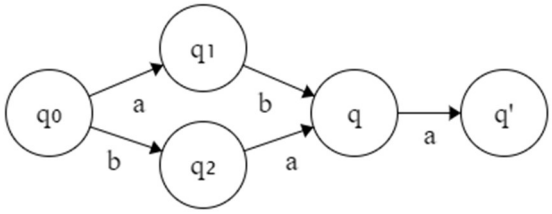
(25) **Definitie:** Met eindige automaat $\mathcal{A} = (Q, T, \delta, q_0, Q_F)$ en onderscheidende functie $f: T^* \to F$, is $\mathcal{A}$ f-onderscheidend als de volgende voorwaarden gelden:

1. $\mathcal{A}$ is deterministisch
2. $\forall_{q \in Q}, \forall_{x,y \in T^*}, \delta^*(q_0, x) = \delta^*(q_0, y) = q \to f(x) = f(y)$
3. $\forall_{q_1,q_2 \in Q}, \exists_{q_3 \in Q}, \exists_{a \in T}, \delta(q_1, a) = \delta(q_2, a) = q_3 \wedge f(q_1) = f(q_2)$ $\to q_1 = q_2 \vee q_1, q_2 \in Q_F$

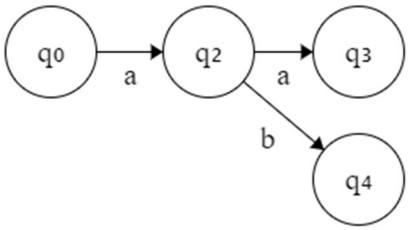
(26) **Definitie:** De familie eindige automaten $\mathcal{A}$ over alfabet T die onderscheidbaar zijn met functie f schrijven we op als (f, T)-DA

Een eindige automaat moet dus ten eerste een DFA zijn om f-onderscheidend te zijn.

Ten tweede: als vanuit de begintoestand met twee woorden dezelfde state q bereikt kan worden, moeten deze twee woorden f-gelijk zijn. Welke transitie $\delta(q, a) = q'$ er vervolgens ook wordt gekozen, $f(q')$ hangt niet af van de manier waarop q is bereikt. Zie figuur 1 voor een grafische weergave.



figuur 1 - Als deze automaat f-onderscheidend is, moet gelden:
$f(ab) = f(ba)$



figuur 2 - $Ter(q_0) = \emptyset, Ter(q_2) = \{a\}$, $Ter(q_3) = \{a\}$, $Ter(q_4) = \{a, b\}$

Dit is als het ware een vertaling van definitie (20) die gaat over een taal L. Volgens (20) zijn twee identieke voortzettingen van twee f-gelijke woorden, op hun beurt ook f-gelijk.
De derde eigenschap van een f-onderscheidende DFA: twee verschillende states mogen f-gelijk zijn, maar dan moeten ze ofwel beide eindtoestanden zijn, ofwel niet dezelfde derde state kunnen bereiken met hetzelfde teken.

Blik terug op voorwaarde 2 in definitie (25) hierboven. Wanneer er verschillende woorden naar één state q leiden, maar al deze woorden f-gelijk zijn, dan kunnen we functie f als het ware lichtelijk overloaden zodat hij ook werkt voor states. Laat $\mathcal{A}$ een f-onderscheidbare automaat zijn met stateset Q. Zie definitie (27) hieronder voor $f: Q \to F$.

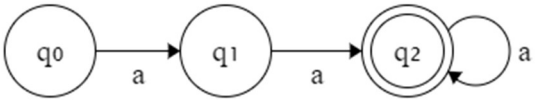
(27) Definitie: $f(q) \coloneqq f(x)$ met $\delta^*(q_0, x) = q$

Aangezien door de tweede voorwaarde in definitie (25) geldt $f(ab) = f(ba)$, zou dat in figuur 1 betekenen dat $f(q) = f(ab) = f(ba)$.

Je zou een onderscheidende functie f kunnen zien als iets waarmee je wat kunt zeggen over welke paden zijn afgelegd om in een zekere toestand te komen.

In figuur 3 geldt $q_1 \neq q_2$ en $\delta(q_1, a) = \delta(q_2, a) = q_2$. Omdat ook geldt dat $\mathrm{Ter}(q_1) = \mathrm{Ter}(q_2) = \{a\}$, voldoet deze automaat niet aan de derde voorwaarde voor f-onderscheidbare automaten met $f(x) = \mathrm{Ter}(x)$ en is hij dus niet Ter-onderscheidbaar.

$L(\mathcal{A})$ is ook niet Ter-onderscheidbaar: neem $w = aa$, $z = a$, $u = a$ en $v = \lambda$. Dan geldt $\mathrm{Ter}(aaa) = \mathrm{Ter}(aa) \wedge \{aaa, aa\} \subseteq L(\mathcal{A})$ en $zu = aa \in L(\mathcal{A})$ maar $zv = a \notin L(\mathcal{A})$.



figuur 3 - Automaat $\mathcal{A} = (\{q_0, q_1, q_2\}, \{a\}, \delta, q_0, \{q_2\})$ met $\delta(q_0, a) = q_1$, $\delta(q_1, a) = q_2$ en $\delta(q_2, a) = q_2$
Automaat $\mathcal{A}$ is niet Ter-onderscheidbaar

Volgens Fernau [3] (*theorem 2*) is een taal f-onderscheidbaar dan en slechts dan als hij wordt geaccepteerd door een f-onderscheidbare automaat. Een f-onderscheidbare automaat accepeert altijd een f-onderscheidbare taal, en een f-onderscheidbare taal heeft op zijn beurt altijd een f-onderscheidbare automaat die hem accepteert.

(28) Stelling: $L \in (f, T)\text{-}DL \leftrightarrow \exists_{\mathcal{A} \in (f,T)\text{-}DA}, L(\mathcal{A}) = L$

Fernau stelt in zijn paper [3] dat met onderscheidende functies een klasse van identificeerbare talen te definiëren is.

(29) Stelling: Voor ieder alfabet T en iedere onderscheidende functie $f: T \to F$ is de familie $(f,T)\text{-}DL$ identificeerbaar in de limiet.

Fernau breidt stelling (29) als volgt uit:

> *Bovendien is er een identificatie-algoritme dat, gegeven de eindige set voorbeelden $I_+ \subset T^*$, een hypothese geeft in de vorm van een eindige automaat $\mathcal{A}$ in orde-grootte $O(\alpha(|F|n)|F|n)$, met α is de inverse Ackermannfunctie en n is de totale lengte van alle woorden in I_+.*
> *De taal die geaccepteerd wordt door $\mathcal{A}$ is de kleinste f-onderscheidbare taal die I_+ bevat.*

Hoewel deze uitspraken erg interessant zijn vanuit algoritme-analitisch oogpunt, zullen we er niet verder op ingaan in deze scriptie.

4.3 Het algoritme: fuseren van toestanden (merging states)

Nu we een aantal definities hebben opgesteld over onderscheidende functies en onderscheidbare talen en automaten, kunnen we beginnen met het schetsen van een algoritme om daadwerkelijk een reguliere expressie te vinden uit een set voorbeelden.

(30) Algoritme: Het merging-state algorithm bestaat uit de volgende stappen:

1. Kies een geschikte onderscheidende functie f
2. Maak een zogenaamde skeletautomaat van de voorbeeldset
3. Voeg f-equivalente toestanden van deze skeletautomaat samen
4. Schrijf de resulterende automaat om naar een reguliere expressie

In de rest van sectie 4.3 zal met f een onderscheidende functie bedoeld worden.

4.3.1 Een skeletautomaat

(31) Definitie: Gegeven de volgende voorwaarden:

- $w_1, \dots, w_N \subseteq T^+$
- $w_i = a_{i,1} \dots a_{i,M_i}$
- $a_{i,j} \in T$
- $1 \leq i \leq N$
- $1 \leq j \leq M_i$

Een skeletautomaat $\mathcal{A}_S(I_+)$ over voorbeeldset $I_+ = \{w_1, \dots, w_N\}$ is gedefinieerd als $\mathcal{A}_S(I_+) = (Q_S, T, \delta_S, Q_0, Q_F)$ met:

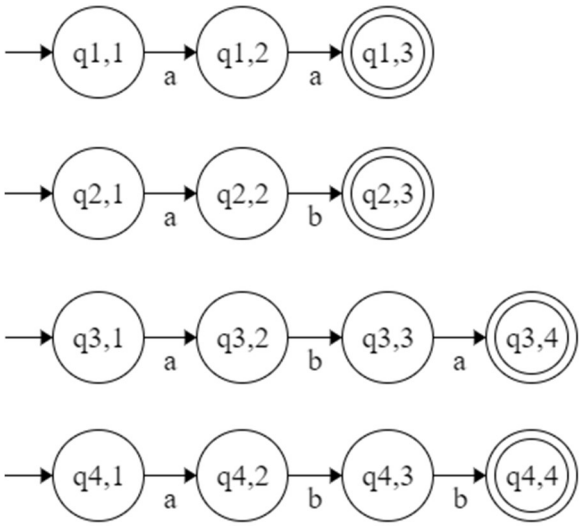
- $Q_S = \{q_{i,j} \mid 1 \leq i \leq N, 1 \leq j \leq M_i + 1\}$
- $\delta_S(q_{i,j}, \mathrm{a_{i,j}}) = q_{i,j+1}$ met $1 \leq i \leq N, 1 \leq j \leq M_i + 1$
- $Q_0 = \{q_{i,1} \mid 1 \leq i \leq N\}$
- $Q_F = \{q_{i,(M_i+1)} \mid 1 \leq i \leq N\}$

Definitie (31) hierboven is het gemakkelijkst te verhelderen aan de hand van een voorbeeld.

(32) Voorbeeld: Neem $I_+ = \{aa, ab, aba, abb\}$. Dan geldt $N = 4$, en hebben we de volgende invulling van de in (31) genoemde voorwaarden:

$w_1 = aa$	$w_2 = ab$	$w_3 = aba$	$w_4 = abb$
$(M_1 = 2)$	$(M_2 = 2)$	$(M_3 = 3)$	$(M_4 = 3)$
$a_{1,1} = a$	$a_{2,1} = a$	$a_{3,1} = a$	$a_{4,1} = a$
$a_{1,2} = a$	$a_{2,2} = b$	$a_{3,2} = b$	$a_{4,2} = b$
		$a_{3,3} = a$	$a_{4,3} = b$
$\delta_S(q_{1,1}, a) = q_{1,2}$	$\delta_S(q_{2,1}, a) = q_{2,2}$	$\delta_S(q_{3,1}, a) = q_{3,2}$	$\delta_S(q_{4,1}, a) = q_{4,2}$
$\delta_S(q_{1,2}, a) = q_{1,3}$	$\delta_S(q_{2,2}, b) = q_{2,3}$	$\delta_S(q_{3,2}, b) = q_{3,3}$	$\delta_S(q_{4,2}, b) = q_{4,3}$
		$\delta_S(q_{3,3}, a) = q_{3,4}$	$\delta_S(q_{4,3}, b) = q_{4,4}$

$Q_S = \{q_{1,1}, q_{1,2}, q_{1,3}, q_{2,1}, q_{2,2}, q_{2,3}, q_{3,1}, q_{3,2}, q_{3,3}, q_{3,4}, q_{4,1}, q_{4,2}, q_{4,3}, q_{4,4}\}$
$Q_0 = \{q_{1,1}, q_{2,1}, q_{3,1}, q_{4,1}\}$
$Q_F = \{q_{1,3}, q_{2,3}, q_{3,4}, q_{4,4}\}$S



figuur 4 – voorbeeld (32) grafisch weergegeven

Gegeven een skeletautomaat $\mathcal{A}_S$ als in definitie (31), definiëren we:

(33) **Definitie:** De *headstring* $\mathrm{HS}(q_{i,j}) = a_{i,1} \dots a_{i,j-1}$
De *frontierstring* $\mathrm{FS}(q_{i,j}) = a_{i,j} \dots a_{i,M_i}$

Definitie (33) levert respectievelijk de deelwoorden vóór en ná een bepaalde state $q_{i,j}$. In voorbeeld (32) (te zien in figuur 1) geldt dus onder andere $\mathrm{HS}(q_{2,2}) = a$ en $\mathrm{FS}(q_{3,2}) = ba$.

(34) **Opmerking:** In een skeletautomaat geldt dus $f(q) = f(\mathrm{HS}(q))$

4.3.2 State-equivalentie

Vervolgens willen we in een skeletautomaat bepaalde states kunnen samenvoegen. Om dat te kunnen doen, moeten we eerst kunnen beoordelen welke states equivalent zijn.

(35) **Definitie:** $$\forall_{q_{i,j},q_{k,\ell} \in Q_S}, q_{i,j} \rightleftharpoons_f q_{k,\ell} \leftrightarrow \left(\mathrm{HS}(q_{i,j}) = \mathrm{HS}(q_{k,\ell}) \vee \begin{pmatrix} \mathrm{FS}(q_{i,j}) = \mathrm{FS}(q_{k,\ell}) \\ \wedge \\ f(q_{i,j}) = f(q_{k,\ell}) \end{pmatrix} \right)$$

Definitie (35) introduceert het symbool $\rightleftharpoons_f$ om aan te geven dat twee states een zekere overeenkomst vertonen: ofwel hun headstrings zijn gelijk, ofwel ze worden gelijk beoordeeld door f en hun frontstrings komen overeen.
Deze overeenkomst is echter nog niet gesloten onder transitiviteit. Als geldt dat $a \rightleftharpoons b$ en $b \rightleftharpoons c$, dan wil dat nog niet zeggen dat $a \rightleftharpoons c$.

(36) **Definitie:** $\equiv_f \coloneqq (\rightleftharpoons_f)^+$

Definitie (36) introduceert equivalentierelatie $\equiv_f$, die ervoor zorgt dat $a \equiv c$ wanneer $a \rightleftharpoons b \rightleftharpoons c$.

(37) **Lemma:** Voor iedere onderscheidende functie f en iedere voorbeeldset I_+ is $\equiv_f$ een equivalentierelatie op de verzameling toestanden van $\mathcal{A}_S(I_+)$

In voorbeeld (32) en de bijbehorende grafische weergave (figuur 4) geldt onder andere:

- $q_{2,3} \rightleftharpoons_{\mathrm{Ter}} q_{3,3}$ omdat $\mathrm{HS}(q_{2,3}) = \mathrm{HS}(q_{3,3})$
- $q_{2,3} \rightleftharpoons_{\mathrm{Ter}} q_{3,4}$ omdat $\mathrm{FS}(q_{2,3}) = \mathrm{FS}(q_{3,4})$ en $\mathrm{Ter}(q_{2,3}) = \mathrm{Ter}(q_{3,4})$

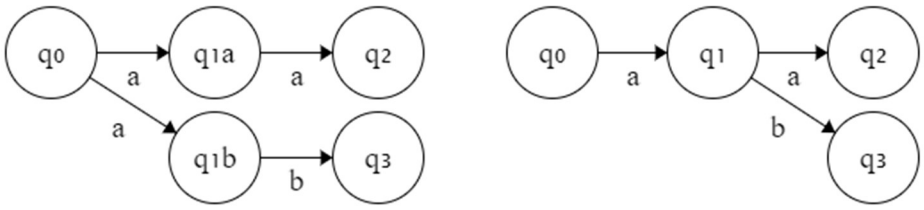
Volgens definitie (36) kunnen we nu concluderen dat $q_{3,3} \equiv_{\mathrm{Ter}} q_{3,4}$, omdat $q_{3,3} \rightleftharpoons_{\mathrm{Ter}} q_{2,3} \rightleftharpoons_{\mathrm{Ter}} q_{3,4}$.

(38) **Definitie:** Een *partitionering* π van set S is een verzameling niet-lege subsets van S, zodat ieder element $s \in S$ in precies één van die subsets zit

(39) **Definitie:** Voor een partitionering π van set S en een element $s \in S$ is definiëren we $B(s, \pi) = b$ als volgt: *Blok* $b \in \pi$ is het element van π dat s bevat.

(40) Definitie: Neem een automaat $\mathcal{A} = (Q, T, \delta, q_0, Q_F)$ en een partitionering π van verzameling toestanden Q. Dan definiëren we de *quotientautomaat* als volgt: $\pi^{-1}\mathcal{A} = (\pi^{-1}Q, T, \delta', B(q_0, \pi), \pi^{-1}Q_F)$ met

- $\pi^{-1}Q = \{B(q, \pi) \mid q \in Q\}$
- $\delta' = \{(B_1, a, B_2) \mid \exists_{q1 \in B_1}, \exists_{q_2 \in B_2}, (q_1, a, q_2) \in \delta\}$
- $\pi^{-1}Q_F = \{B(q, \pi) \mid q \in Q_F\}$



figuur 5 – Een voorbeeld van een normale automaat, en een quotientautomaat
links: een automaat $\mathcal{A}$, rechts: een automaat $\pi^{-1}\mathcal{A}$ met $q_1 = B(q_{1a}, \pi) = B(q_{1b}, \pi)$

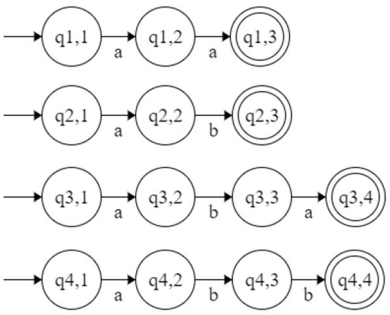
Definitie (40) is beschrijft de nieuwe indeling van een automaat, waarin transities niet bestaan tussen afzonderlijke states, maar tussen de blokken die de states bevatten. Van een normale automaat kan een speciale quotientautomaat gemaakt worden, als de partitionering wordt gebaseerd op een equivalentierelatie tussen states.
Volgens stelling (41) hieronder hoef je daarvoor slechts $\equiv_f$ te kiezen, en de resulterende automaat zal de kleinste f-onderscheidbare taal accepteren die de voorbeeldset bevat. [3] (*theorem 4*)

(41) Stelling: Voor iedere f-onderscheidbare functie f en iedere voorbeeldset I_+ is de automaat $\pi_f^{-1}\mathcal{A}_S(I_+)$ een f-onderscheidbare automaat (met π_f is de partitionering die is afgeleid aan de hand van equivalentierelatie $\equiv_f$).
$L\left(\pi_f^{-1}\mathcal{A}_S(I_+)\right)$ is de kleinste f-onderscheidbare taal die I_+ bevat.

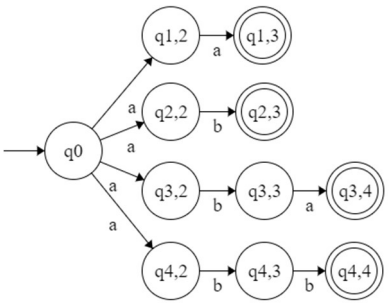
Om te illustreren hoe dit proces verloopt, gebruiken we de automaat uit voorbeeld (32). In de stappen hieronder fuseren we de toestanden van skeletautomaat $\mathcal{A}_S(I_+)$ tot we uitkomen op $\pi_{\mathrm{Ter}}^{-1}\mathcal{A}_S(I_+)$.
In figuur 6 is automaat $\mathcal{A}_S(I_+)$ weergegeven zoals die is volgens voorbeeld (32). Hier geldt $q_{1,1} \equiv_{\mathrm{Ter}} q_{2,1} \equiv_{\mathrm{Ter}} q_{3,1} \equiv_{\mathrm{Ter}} q_{4,1}$, omdat deze toestanden alle dezelfde headstring hebben, namelijk λ. Deze toestanden plaatsen we in hetzelfde blok $q_0 \in \pi_{\mathrm{Ter}}$ zodat $B(q_{1,1}, \pi_{\mathrm{Ter}}) = B(q_{2,1}, \pi_{\mathrm{Ter}}) = B(q_{3,1}, \pi_{\mathrm{Ter}}) = B(q_{4,1}, \pi_{\mathrm{Ter}}) = q_0$. Het resultaat is te zien in figuur 7.
Ook geldt $q_{1,2} \equiv_{\mathrm{Ter}} q_{2,2} \equiv_{\mathrm{Ter}} q_{3,2} \equiv_{\mathrm{Ter}} q_{4,2}$ omdat de headstring van deze vier states gelijk is aan a. Deze vier states plaatsen we in blok q_1. Zie figuur 8 voor het resultaat.
Nu zijn er nog een aantal relaties te vinden:

- $q_{2,3} \rightleftharpoons_{\mathrm{Ter}} q_{3,3} \rightleftharpoons_{\mathrm{Ter}} q_{4,3}$
  Omdat geldt: $\mathrm{HS}(q_{2,3}) = \mathrm{HS}(q_{3,3}) = \mathrm{HS}(q_{4,3}) = ab$
- $q_{2,3} \rightleftharpoons_{\mathrm{Ter}} q_{3,4} \rightleftharpoons_{\mathrm{Ter}} q_{4,4}$
  Omdat geldt: $\mathrm{FS}(q_{2,3}) = \mathrm{FS}(q_{3,4}) = \mathrm{FS}(q_{4,4}) = \lambda$
  en $\mathrm{Ter}(q_{2,3}) = \mathrm{Ter}(q_{3,4}) = \mathrm{Ter}(q_{4,4}) = \{a, b\}$

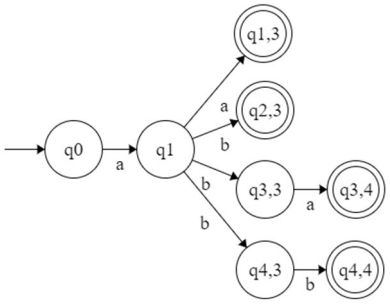
Vanwege de transitiviteit van de relatie $\equiv_{\mathrm{Ter}}$ die in definitie (36) is vastgesteld, geldt nu $q_{2,3} \equiv_{\mathrm{Ter}} q_{3,3} \equiv_{\mathrm{Ter}} q_{4,3} \equiv_{\mathrm{Ter}} q_{3,4} \equiv_{\mathrm{Ter}} q_{4,4}$. Deze stap wordt in figuur 9, figuur 10 en figuur 11 duidelijk gemaakt, door eerst $q_{2,3}$, $q_{3,3}$ en $q_{4,3}$ samen te voegen in blok q_2, vervolgens $q_{3,4}$ en $q_{4,4}$ samen te voegen in q_3 en uiteindelijk q_3 bij q_2 te voegen.
Uit de resulterende automaat $\pi_{\mathrm{Ter}}^{-1}\mathcal{A}_{\mathrm{S}}(\mathrm{I}_+)$ volgt de taal $L = a(a|(b(a|b)^*))$.



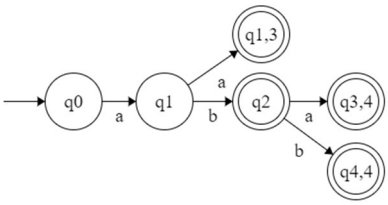
figuur 6 – automaat $\mathcal{A}_S(I_+)$, voorafgaand het fuseren



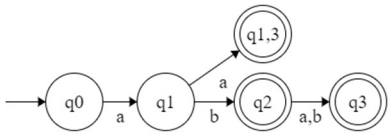
figuur 7 – $\mathcal{A}_S(I_+)$ na de eerste fusie



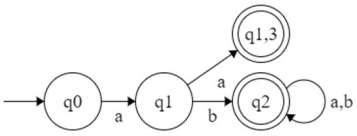
figuur 8 – $\mathcal{A}_S(I_+)$ na de tweede fusie



figuur 9 – de derde fusie, deel 1



figuur 10 – de derde fusie, deel 2



figuur 11 – de derde fusie, het resultaat: $\pi_{Ter}^{-1}\mathcal{A}_S(I_+)$

4.4 Voorbeeld (1) uitgewerkt

Uit sectie 2.3 volgt een aantal te leren oppervlakken. In deze sectie zullen we deze oppervlakken uitwerken volgens het op f-onderschijdbaarheid gebaseerde fuseren van toestanden. Voorbeeld (1) is helaas niet erg "spannend" wat de groottes van de oppervlakken van verschillende letters betreft.

4.4.1 Stap 1: het kiezen van een geschikte onderscheidende functie f

Bij het uitwerken van dit voorbeeld hebben we ervoor gekozen bij $f = \mathrm{Ter}$ te blijven, aangezien het vreemd zou zijn om nu een andere onderscheidende functie te introduceren. Bovendien volstaat Ter voor dit voorbeeld.

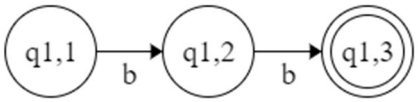
4.4.2 Stap 2: het maken van een skeletautomaat van de voorbeeldset

4.4.2.1 $I_+ = S_i(L)$

Alle formele beschrijvingen die nodig zijn bij het maken van een skeletautomaat met $I_+ = S_i(L) = \{bb\}$ zijn hier gegeven. In figuur 12 is deze skeletautomaat grafisch weergegeven.

$w_1 = bb$
$(M_1 = 2)$
$a_{1,1} = b$
$a_{1,2} = b$

$\delta_S(q_{1,1}, b) = q_{1,2}$
$\delta_S(q_{1,2}, b) = q_{1,3}$
$Q_S = \{q_{1,1}, q_{1,2}, q_{1,3}\}$
$Q_0 = \{q_{1,1}\}$
$Q_F = \{q_{1,3}\}$



figuur 12 –Skeletautomaat $\mathcal{A}_S(S_i(L))$

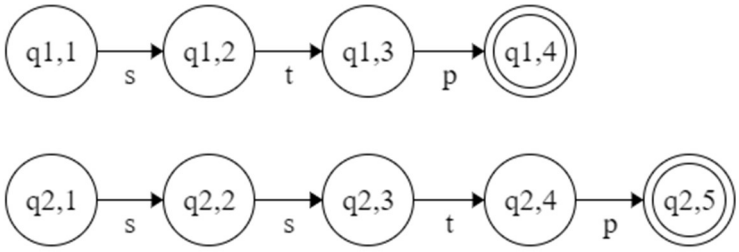
4.4.2.2 $I_+ = S_b(L)$

Wederom zijn de formele beschrijvingen gegeven voor de skeletautomaat met $I_+ = S_b(L) = \{stp, sstp\}$. De graaf is grafisch weergegeven in figuur 13.

$w_1 = stp$
$(M_1 = 3)$
$a_{1,1} = s$
$a_{1,2} = t$
$a_{1,3} = p$
$\delta_S(q_{1,1}, s) = q_{1,2}$
$\delta_S(q_{1,2}, t) = q_{1,3}$
$\delta_S(q_{1,3}, p) = q_{1,4}$

$w_2 = sstp$
$(M_2 = 4)$
$a_{2,1} = s$
$a_{2,2} = s$
$a_{2,3} = t$
$a_{2,4} = p$
$\delta_S(q_{2,1}, s) = q_{2,2}$
$\delta_S(q_{2,2}, s) = q_{2,3}$
$\delta_S(q_{2,3}, t) = q_{2,4}$
$\delta_S(q_{2,4}, p) = q_{2,5}$

$Q_S = \{q_{1,1}, q_{1,2}, q_{1,3}, q_{1,4}, q_{2,1}, q_{2,2}, q_{2,3}, q_{2,4}, q_{2,5}\}$
$Q_0 = \{q_{1,1}, q_{2,1}\}$
$Q_F = \{q_{1,4}, q_{2,5}\}$



figuur 13 – Skeletautomaat $\mathcal{A}_S(S_b(L))$

4.4.2.3 $I_+ = S_s(L)$

Hier gaat het om de skeletautomaat met $I_+ = S_s(L) = \{va, la\}$. In figuur 14 is deze skeletautomaat grafisch weergegeven.

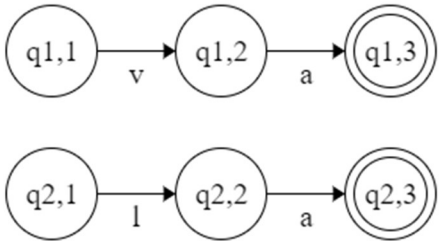
$w_1 = va$
$(M_1 = 2)$
$a_{1,1} = v$
$a_{1,2} = a$

$\delta_S(q_{1,1}, v) = q_{1,2}$
$\delta_S(q_{1,2}, a) = q_{1,3}$

$w_2 = la$
$(M_2 = 2)$
$a_{2,1} = l$
$a_{2,2} = a$

$\delta_S(q_{2,1}, l) = q_{1,2}$
$\delta_S(q_{2,2}, a) = q_{1,3}$

$Q_S = \{q_{1,1}, q_{1,2}, q_{1,3}, q_{2,1}, q_{2,2}, q_{2,3}\}$
$Q_0 = \{q_{1,1}, q_{2,1}\}$
$Q_F = \{q_{1,3}, q_{2,3}\}$

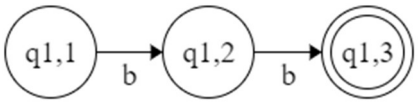


figuur 14 – Skeletautomaat $\mathcal{A}_S(S_s(L))$

4.4.3 Stap 3: het samenvoegen van equivalente states

4.4.3.1 $I_+ = S_i(L)$

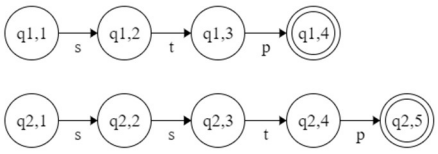
Helaas verandert er met het mergen van equivalente states niet veel aan de skeletautomaat van $S_i(L)$. Mocht er een tweede voorbeeldinventaris komen met bijvoorbeeld een andere hoeveelheid boeken, of ander materiaal (tijdschriften, DVD's) dan zal het algoritme hier waarschijnlijk wel een effect hebben.



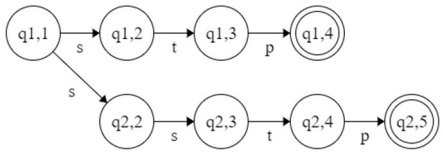
figuur 15 – $\pi_{Ter}^{-1}\mathcal{A}_S(S_i(L))$

4.4.3.2 $I_+ = S_b(L)$

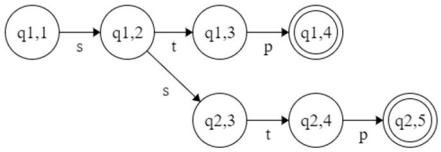
De eerste twee fusies zijn gebaseerd op gelijkheid van headstring. Vanaf de derde fusie verandert dat: omdat $\mathrm{FS}(q_{2,3}) = \mathrm{FS}(q_{1,2}) \wedge \mathrm{Ter}(q_{2,3}) = \mathrm{Ter}(q_{1,2})$ geldt $q_{2,3} \equiv_{\mathrm{Ter}} q_{1,2}$. De fusies na de derde zijn gebaseerd op gelijkheid van frontstring en Ter-waarde.



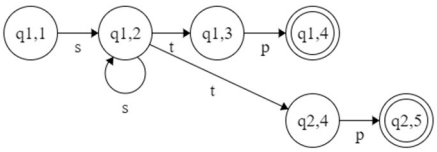
figuur 16 – Skeletautomaat $\mathcal{A}_S(S_i(L))$



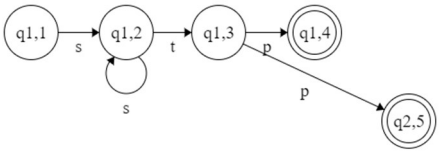
figuur 17 – De skeletautomaat na de eerste fusie, $q_{1,1} \equiv_{Ter} q_{2,1}$



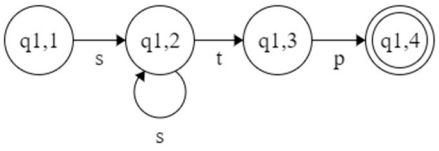
figuur 18 – De skeletautomaat na twee fusies $q_{2,2} \equiv_{Ter} q_{1,2}$



figuur 19 – De skeletautomaat na drie fusies $q_{2,3} \equiv_{Ter} q_{1,2}$



figuur 20 – De skeletautomaat na vier fusies $q_{2,4} \equiv_{Ter} q_{1,3}$

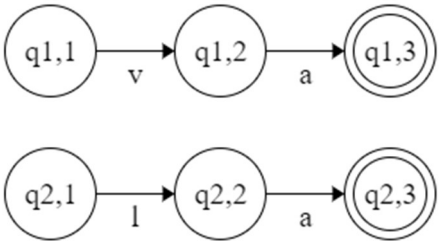


figuur 21 – Het resultaat $\pi_{Ter}^{-1}\mathcal{A}_S(S_b(L))$ na de laatste fusie: $q_{2,5} \equiv_{Ter} q_{1,4}$

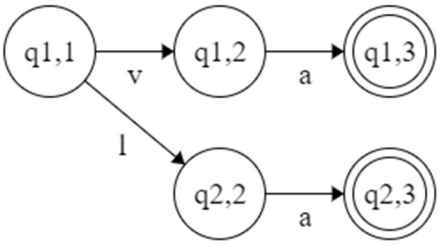
4.4.3.3 $I_+ = S_s(L)$

Net als in sectie 4.4.3.1, waar het ging om $I_+ = S_i(L)$, gebeurt er in het geval $I_+ = S_s(L)$ niet heel veel. Ook hier geldt: mocht de voorbeeldset uitgebreid worden met een schrijver die bijvoorbeeld een voornaam en een voorletter heeft, dan zal het algoritme hier meer effect hebben.

In figuur 22 – Skeletautomaat $\mathcal{A}_S(S_s(L))$figuur 22 en figuur 23 wordt de fusie van $\mathcal{A}_S(S_s(L))$ beschreven.



figuur 22 – Skeletautomaat $\mathcal{A}_S(S_s(L))$



figuur 23 – Na één fusie: $\pi_{Ter}^{-1}\mathcal{A}_S(S_s(L))$

4.4.4 Stap 4: het omschrijven naar een reguliere expressie

Hier gelden voor de voorbeeldsets de volgende reguliere expressies:

- $I_+ = S_i(L)$: bb
- $I_+ = S_b(L)$: ss^*tp
- $I_+ = S_s(L)$: $(va)|(la)$

4.4.5 Conclusie: de grammatica

Uiteindelijk is het vinden van een XML-grammatica bij voorbeeld (1) het einddoel. Deze grammatica kan direct afgeleid worden uit de gevonden reguliere expressies. Voor iedere $a \in A$ geldt dat X_a correspondeert met het bijbehorende teken a in de gevonden reguliere expressie. De grammatica die volgt uit de met het algoritme gevonden reguliere expressies:

$X_i \rightarrow iR_i\bar{\imath}$ met $R_i = \{X_bX_b\}$
$X_b \rightarrow bR_b\bar{b}$ met $R_b = \{X_sX_s^*X_tX_p\}$
$X_s \rightarrow sR_s\bar{s}$ met $R_s = \{X_vX_a, X_lX_a\}$
$X_v \rightarrow v\bar{v}$
$X_l \rightarrow l\bar{l}$
$X_a \rightarrow a\bar{a}$
$X_t \rightarrow t\bar{t}$
$X_p \rightarrow p\bar{p}$

5 Simple-looping

Een tweede manier om een reguliere expressie te vinden voor de voorbeelden binnen een XML-taal, is aan de hand van het $\mathcal{SL}$-*infer* algoritme dat Fernau in zijn eerdere paper [4] noemt. Hierin bewijst hij ook dat de simple-looping talen die hij beschrijft, identificeerbaar zijn in de limiet. Om goed te kunnen beschrijven wat een simple-looping taal is, zullen we eerst een aantal nieuwe zaken moeten definiëren.

Laat I een verzameling woorden $w \in \Sigma^*$ zijn, en laat a een letter zijn in Σ.

(42) Definitie: $\forall_{I\subseteq\Sigma^*}, \forall_{a\in\Sigma}, I(a) = \{w \in I \mid \exists_{v\in\Sigma^*}, w = av\}$

$I(a)$ bevat alle woorden uit I die beginnen met de letter a.

(43) Definitie: $\forall_{I\subseteq\Sigma^*}, \forall_{a\in\Sigma}, I^a = \{v \in b\Sigma^* \mid b \neq a \wedge (\exists_{n>0}, a^n v \in I(a))\}$

I^a bevat alle woorden die overblijven na het "verwijderen" van alle instanties van a aan het begin van de woorden in $I(a)$.

(44) Definitie: Gegeven verzameling $I \subset \Sigma^+$ en $I(a) = \{a^{n_1}v_1, a^{n_2}v_2, \dots a^{n_m}v_m\}$ met:

- $a \in \Sigma$
- $m = |I(a)|$
- $n_i \leq n_{i+1}$
- $v_i \in \{\lambda\} \cup (\Sigma \setminus \{a\})\Sigma^*$

Dan definiëren we het *a-spectrum* van I als $\mathbb{S}(I(a)) = (n_1, n_2, \dots n_m)$

Gegeven $\mathbb{S}(I(a)) = (n_1, n_2, \dots n_m)$, definiëren we $\alpha\left(\mathbb{S}(I(a))\right) = n_1$

Gegeven $\mathbb{S}(I(a)) = (n_1, n_2, \dots n_m)$, definiëren we $\omega\left(\mathbb{S}(I(a))\right) = n_m$

(45) Opmerking: De verzameling J^b met $J = I^a$ schrijven we ook als I^{ab}. Ook geldt dus $I^\lambda = I$

(46) Voorbeeld: Stel $I_+ = \{ababb, aabb, ababa, abc, bb, bbbb, baa\}$ met $I_+ \subset \Sigma^+$ en $\Sigma = \{a, b, c\}$, dan gelden onder andere de volgende zaken:

$I_+(a) = \{ababb, aabb, ababa, abc\}$	$I_+(b) = \{bb, bbbb, baa\}$
$\mathbb{S}(I(a)) = (1,1,1,2)$	$\mathbb{S}(I(b)) = (1,2,4)$
$\alpha\left(\mathbb{S}(I(a))\right) = 1$	$\alpha\left(\mathbb{S}(I(b))\right) = 1$
$\omega\left(\mathbb{S}(I(a))\right) = 2$	$\omega\left(\mathbb{S}(I(b))\right) = 4$
$I_+^a = \{babb, bb, baba, bc\}$	$I_+^b = \{aa\}$
$I_+^{ab} = \{abb, aba, c\}$	$I_+^{ba} = \emptyset$

5.1 Simple-looping talen

5.1.1 Links-uitgelijnde expressies

(47) **Definitie:** Een verenigingsvrije reguliere expressie is als volgt opgebouwd:
$\beta = b_1^{x_1} b_2^{x_2} \dots b_n^{x_n}$ waarbij bij $b_i \in \Sigma$ en $x_i \in \{1, *\}$
In het geval $n = 0$ geldt $\beta = \lambda$, het lege woord.

(48) **Definitie:** Laat $\beta = b_1^{x_1} b_2^{x_2} \dots b_n^{x_n}$ een verenigingsvrije reguliere expressie zijn, dan definiëren we $b_{n+1} = \$$ met $\$ \notin \Sigma$
Het dollarteken gebruiken we hier als een speciaal symbool om het eind van een woord te markeren.

(49) **Definitie:** Gegeven twee verenigingsvrije reguliere expressies β en γ met:
$\beta = b_1^{x_1} b_2^{x_2} \dots b_n^{x_n}$
$\gamma = c_1^{y_1} c_2^{y_2} \dots c_m^{y_m}$
Definieer $K \leq \min(n, m)$ als het getal waarvoor geldt:
$b_{K+1} \neq c_{K+1} \wedge (\forall_k, 1 \leq k \leq K \rightarrow b_k = c_k)$
β en γ zijn *links-uitgelijnd* dan en slechts dan als:
$\beta \neq \gamma \wedge b_K \notin \{b_{K+1}, c_{K+1}\} \wedge (\forall_k, 1 \leq k \leq K \rightarrow x_k = y_k)$
Definitie (49) heeft enige toelichting nodig. Volgens deze definitie zijn twee reguliere expressies links-uitgelijnd, wanneer ze tot een bepaalde letter a overeen komen. Deze letter a is echter verschillend in beide woorden, en verschilt ook van de letter ervoor.

(50) **Voorbeeld:** Een paar voorbeelden van links-uitgelijnde expressies:

$\beta = aaabd\$ = a^1a^1a^1b^1d^1\$$
$\gamma = aaac\$ = a^1a^1a^1c^1\$$
$K = 3$, want:
$b \neq c$
$(\forall_k, 1 \leq k \leq K \rightarrow b_k = c_k = a)$
Nu geldt:
$(b_3 = a) \notin \{b, c\}$
$k = 1: x_1 = y_1 = 1$
$k = 2: x_2 = y_2 = 1$
$k = 3: x_3 = y_3 = 1$
Dus β en γ zijn links-uitgelijnd

$\beta = ababc\$ = a^1b^1a^1b^1c^1\$$
$\gamma = ababd\$ = a^1b^1a^1b^1d^1\$$
$K = 4$, want:
$c \neq d$
$(\forall_k, 1 \leq k \leq K \rightarrow b_k = c_k)$
Nu geldt:
$(b_4 = b) \notin \{c, d\}$
$k = 1: x_1 = y_1 = 1$
$k = 2: x_2 = y_2 = 1$
$k = 3: x_3 = y_3 = 1$
$k = 4: x_4 = y_4 = 1$
Dus β en γ zijn links-uitgelijnd

(51) **Voorbeeld:** Natuurlijk zijn er ook gevallen die misschien links-uitgelijnd lijken, maar het niet zijn. Enkele voorbeelden:

$\beta = aaa\$$
$\gamma = aaa\$$
Hier geldt $\beta = \gamma$
Dus β en γ zijn niet links-uitgelijnd

$\beta = aac\$$
$\gamma = ab\$$
$K = 1$, want:
$a \neq \$$
$b_1 = c_1 = a$
Nu geldt echter: $(b_1 = a) \in \{a, b\}$
Dus β en γ zijn niet links-uitgelijnd

5.1.2 Simple-looping expressies

(52) Definitie: De familie simple-looping expressies schrijven we op als $SL\text{-}E$

(53) Definitie: Expressie $E \in SL\text{-}E$ wanneer deze een eindige vereniging is van paarsgewijs links-uitgelijnde expressies β met ofwel $\beta = \lambda$, ofwel de volgende normaalvorm:

$\beta = b_1^{k_1} b_1^{x_1} b_2^{k_2} b_2^{x_2} \dots b_n^{k_n} b_n^{x_n}$

Met de volgende voorwaarden:

- $1 \leq i \leq n, b_i \neq b_{i+1}$
- $\forall_{b_i}, b_i \in \Sigma$
- $(k_i \in \mathbb{N}) \geq 0$
- $x_i \in \{0,*\}$

Iedere eindige taal kan worden geschreven als een vereniging van verenigingsvrije expressies in de normaalvorm van definitie (53), maar deze expressies zullen vaak niet links-uitgelijnd zijn. De verenigingen $\beta \cup \gamma$ met β en γ uit voorbeeld (51) voldoen dan ook niet aan de voorwaarden.

(54) Definitie: De familie simple-looping talen schrijven we op als $SL\text{-}L$

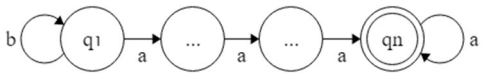
(55) Definitie: Gegeven $E \in SL\text{-}E$ definiëren we $SL\text{-}L \coloneqq \{L \mid \exists_{E \in SL\text{-}E}, L = L(E)\}$

Een taal L is simple-looping als er een simple-looping expressie bestaat die L beschrijft.

5.2 Simple-looping automaten

(56) Definitie: Laat DFA $\mathcal{A} = (Q, \Sigma, \delta, q_0, Q_F)$ een eindige automaat zijn. $\mathcal{P}_a = (q_1, q_2, \dots, q_n)$ met $a \in \Sigma$ is een *a-pad* wanneer:

1. $\delta(q_i, a) = q_{i+1}$ met $1 \leq i < n$
2. De enige uitgaande paden van states in $\mathcal{P}_a$ worden gegeven in voorwaarde 1, met als uitzondering de mogelijkheid $\delta(q_1, b) = q_1$ met $b \neq a$.
3. De enige inkomende paden van states in $\mathcal{P}_a$ worden gegeven in voorwaarde 1, met als uitzondering de mogelijkheid $\delta(q_n, a) = q_n$
4. $\forall_{q \in \{q_2, \dots, q_{n-1}\}}, q \notin Q_F$



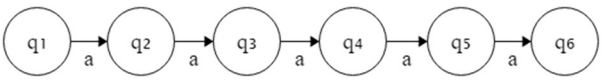
figuur 24 - $\mathcal{P}_a = (q_1, \dots, q_n)$ voldoet aan de voorwaarden van een a-pad

(57) Definitie: Een a-pad is *maximaal* als het niet deel is van een groter a-pad

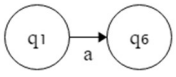
(58) Definitie: De *samentrekking* $C(\mathcal{A})$ van een automaat $\mathcal{A}$ wordt verkregen door voor iedere $a \in \Sigma$ het maximale a-pad $\mathcal{P}_a = (q_1, q_2, \dots, q_n)$ te vervangen door de transitie $\delta(q_1, a) = q_n$, en alle tussen-states $q_2, \dots, q_{n-1}$ te verwijderen.

In figuur 25 en figuur 26 wordt grafisch weergegeven hoe een a-pad wordt samengetrokken.

Omdat we alleen maximale a-paden samentrekken, kunnen we hierbij opmerken dat de volgorde



figuur 25 – $\mathcal{P}_a = (q_1, q_2, q_3, q_4, q_5, q_6)$ is een maximaal a-pad



figuur 26 - De samentrekking van $\mathcal{P}_a$ in figuur 25

waarin we die paden samentrekken, geen invloed heeft op het resultaat. [4] (*lemma 1*)

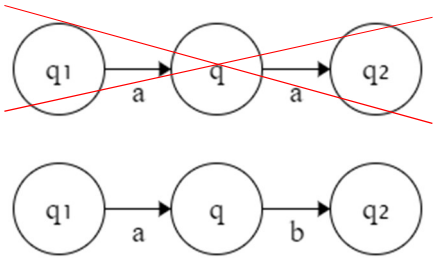
Uit het werk van Fernau [4] (*definition 3*) kunnen we het volgende concluderen:

(59) **Definitie:** Een *skeletgraaf*[2] is het resultaat van het verwijderen van alle loops en transitielabels uit een automaat $\mathcal{A}$.

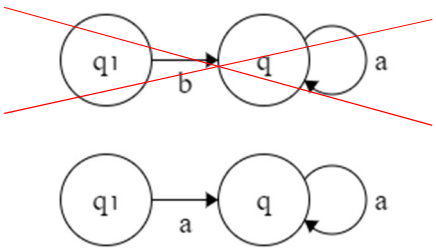
(60) **Definitie:** Een eindige automaat $\mathcal{A} = (Q, \Sigma, \delta, q_0, Q_F)$ is *simple-looping* dan en slechts dan als voor zijn samentrekking $C(\mathcal{A})$ geldt:

1. De skeletgraaf $\mathcal{S}$ is een gerichte boom met de volgende voorwaarden:
   - de graaf heeft één beginstate zonder inkomende transities
   - alle states zonder uitgaande transities zijn eindstates
2. $\forall_{a,b\in\Sigma}, \forall_{q_1,q,q_2\in Q}, q_1 \neq q \wedge q \neq q_2 \wedge \delta(q_1, a) = q \wedge \delta(q, b) = q_2 \rightarrow a \neq b$
3. $\forall_{a\in\Sigma}, \forall_{q_1,q\in Q}, q_1 \neq q \wedge \delta(q, a) = q \rightarrow \delta(q_1, a) = q$

Volgens de tweede voorwaarde van definitie (60) hebben inkomende en uitgaande transities, met uitzondering van loops, van iedere state paarsgewijs verschillende labels.
Uit de derde voorwaarde volgt dat een loop bij een state altijd hetzelfde label als de inkomende transitie van die state. Aangezien de skeletgraaf van $C(\mathcal{A})$ een gerichte boom is, zal iedere state altijd precies één inkomende transitie hebben, afgezien van loops.



figuur 27 – Definitie (60), voorwaarde 2 grafisch weergegeven.
De automaat boven voldoet niet aan de voorwaarde.
De automaat onder wel.



figuur 28 – Definitie (60), voorwaarde 3 grafisch weergegeven.
De automaat boven voldoet niet aan de voorwaarde.
De automaat onder wel.

(61) **Definitie:** De familie simple-looping automaten schrijven we op als $SL\text{-}A$

In figuur 29 wordt een simple-looping automaat $\mathcal{A}$ beschreven:

- Skeletgraaf $\mathcal{S}$ (figuur 31) van $C(\mathcal{A})$ (figuur 30) voldoet aan voorwaarde 1 van definitie (60)
- $C(\mathcal{A})$ voldoet aan voorwaarde 2:

$$\begin{pmatrix} q_0 \neq q_1 \wedge q_1 \neq q_2 \\ \wedge \\ \delta(q_0, a) = q_1 \wedge \delta(q_1, b) = q_2 \end{pmatrix} \rightarrow a \neq b$$

$$\begin{pmatrix} q_0 \neq q_1 \wedge q_1 \neq q_2 \\ \wedge \\ \delta(q_0, a) = q_1 \wedge \delta(q_1, c) = q_3 \end{pmatrix} \rightarrow a \neq c$$

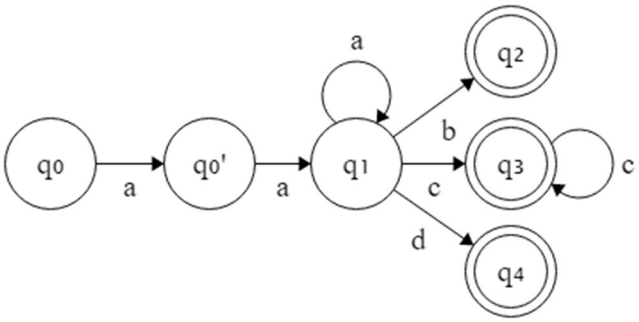
$$\begin{pmatrix} q_0 \neq q_1 \wedge q_1 \neq q_2 \\ \wedge \\ \delta(q_0, a) = q_1 \wedge \delta(q_1, d) = q_4 \end{pmatrix} \rightarrow a \neq d$$

- $C(\mathcal{A})$ voldoet aan voorwaarde 3:

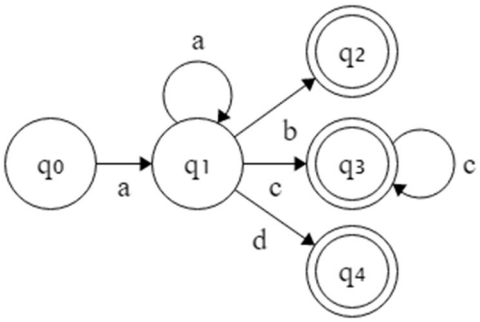
$$\begin{pmatrix} q_0 \neq q_1 \\ \wedge \\ \delta(q_1, a) = q_1 \end{pmatrix} \rightarrow \delta(q_0, a) = q_1$$

$$\begin{pmatrix} q_1 \neq q_3 \\ \wedge \\ \delta(q_3, c) = q_3 \end{pmatrix} \rightarrow \delta(q_1, c) = q_3$$

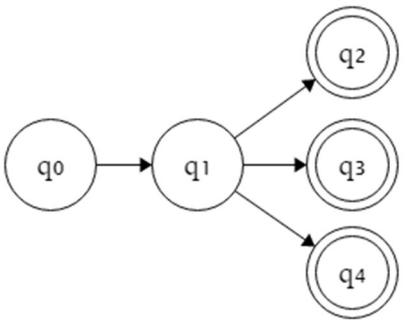
[2] Niet te verwarren met de skeletautomaat uit definitie (31)



figuur 29 – Automaat $\mathcal{A} = (Q, \Sigma, \delta, q_0, Q_F)$ met:
- $Q = \{q_0, q_0', q_1, q_2, q_3, q_4\}$
- $\Sigma = \{a, b, c, d\}$
- $\delta(q_0, a) = q_0'$, $\delta(q_0', a) = \delta(q_1, a) = q_1$, $\delta(q_1, b) = q_2$, $\delta(q_1, c) = \delta(q_3, c) = q_3$ en $\delta(q_1, d) = q_4$
- $Q_F = \{q_2, q_3, q_4\}$



figuur 30 – Automaat $C(\mathcal{A}) = (Q', \Sigma, \delta', q_0, Q_F)$ met:
- $Q = \{q_0, q_1, q_2, q_3, q_4\}$
- $\Sigma = \{a, b, c, d\}$
- $\delta'(q_0, a) = \delta'(q_1, a) = q_1$, $\delta'(q_1, b) = q_2$, $\delta'(q_1, c) = \delta'(q_3, c) = q_3$ en $\delta'(q_1, d) = q_4$
- $Q_F = \{q_2, q_3, q_4\}$



figuur 31 – De skeletgraaf $\mathcal{S}$ van $C(\mathcal{A})$

In zijn paper [4] bewijst Fernau de volgende stelling:

(62) Stelling: $\forall_{L \subseteq \Sigma^*}, L \in SL\text{-}L \leftrightarrow (\exists_{\mathcal{A} \in SL\text{-}A}, L(\mathcal{A}) = L)$

Met stelling (62) zegt hij dat voor iedere taal $L \subseteq \Sigma^*$, de noties "L is simple-looping" en "L wordt gegenereerd door een simple-looping automaat" equivalent zijn.

5.3 Het algoritme: SL-infer

Het algoritme dat Fernau beschrijft om uit voorbeeldsets automaten te leren noemt hij *SL-infer*, en bestaat uit twee delen: $create\text{-}tree$, beschreven in algoritme (63), en $generalize\text{-}simple$, beschreven in algoritme (65).

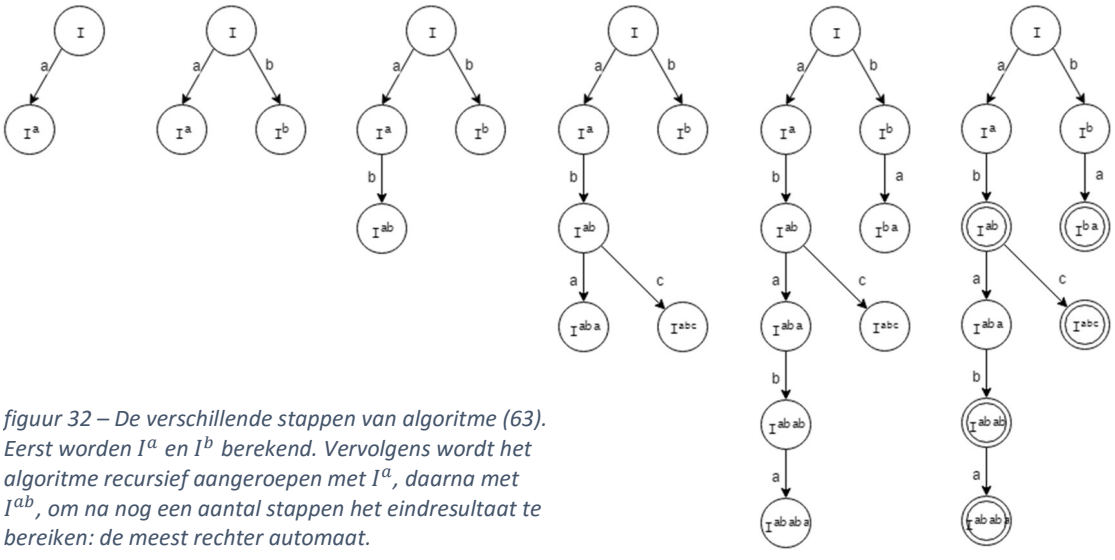
(63) Algoritme: Voor een set voorbeelden I definiëren we het algoritme $create\text{-}tree(I)$:

1. Bereken $I(a)$ en I^a voor iedere $a \in \Sigma$
2. Creëer voor iedere niet-lege set I^a een node met als label I^a
3. Voeg voor iedere $a \in \Sigma$ een transitie toe van I naar I^a met als label a
4. Roep voor iedere $a \in \Sigma$ $create\text{-}tree(I^a)$ aan
5. Maak voor iedere $x \in \Sigma \cup \{\lambda\}$ en $a \in \Sigma$ de state I^{xa} final als geldt: $|I^x(a)| > |I^{xa}|$

De werking van algoritme (63) wordt geïllustreerd aan de hand van voorbeeld (64).

(64) Voorbeeld: Neem $I = \{ababb, aabb, ababa, abc, bb, bbbb, baa\}$.
In de tabel hieronder is van de te berekenen verzamelingen de inhoud weergegeven. In figuur 32 is de voortgang van het algoritme grafisch in stappen weergegeven.

$a \in \Sigma$	$I(a)$	I^a
a	$I(a) = \{ababb, aabb, ababa, abc\}$	$I^a = \{babb, bb, baba, bc\}$
a	$I^a_+(a) = \emptyset$	
b	$I^a(b) = \{babb, bb, baba, bc\}$	$I^{ab} = \{abb, aba, c\}$
a	$I^{ab}(a) = \{abb, aba\}$	$I^{aba} = \{bb, ba\}$
a	$I^{aba}(a) = \emptyset$	
b	$I^{aba}(b) = \{bb, ba\}$	$I^{abab} = \{a\}$
a	$I^{abab}(a) = \{a\}$	$I^{ababa} = \emptyset$
b	$I^{abab}(b) = \emptyset$	
c	$I^{abab}(c) = \emptyset$	
c	$I^{aba}(c) = \emptyset$	
b	$I^{ab}(b) = \emptyset$	
c	$I^{ab}(c) = \{c\}$	$I^{abc} = \emptyset$
c	$I^a(c) = \emptyset$	
b	$I(b) = \{bb, bbbb, baa\}$	$I^b = \{aa\}$
a	$I^b(a) = \{aa\}$	$I^{ba} = \emptyset$
b	$I^b(b) = \emptyset$	
c	$I^b(c) = \emptyset$	
c	$I(c) = \emptyset$	



figuur 32 – De verschillende stappen van algoritme (63). Eerst worden I^a en I^b berekend. Vervolgens wordt het algoritme recursief aangeroepen met I^a, daarna met I^{ab}, om na nog een aantal stappen het eindresultaat te bereiken: de meest rechter automaat.

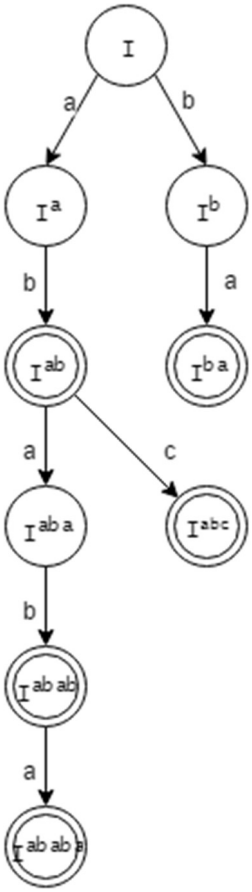
Algoritme (63) levert een automaat in de vorm van een boom T. Algoritme (65) voert enkele bewerkingen uit op deze boom.

(65) Algoritme: Voor een boom T definiëren we het algoritme $generalize\text{-}simple(T)$:
Voor iedere $x \in \Sigma \cup \{\lambda\}$ en $a \in \Sigma$, doe voor node I^{xa} het volgende:

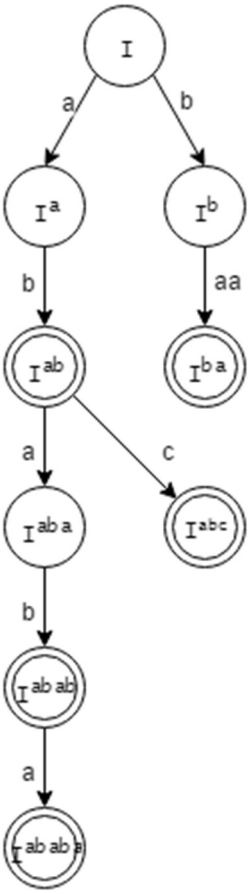
1. Geef de transitie die eindigt in I^{xa} het label a^n met $n = \alpha\left(\mathbb{S}\left(I^x(a)\right)\right)$
2. Als $\omega\left(\mathbb{S}\left(I^x(a)\right)\right) > \alpha\left(\mathbb{S}\left(I^x(a)\right)\right)$: voeg een loop met het label a toe aan I^{xa}

(66) Opmerking: Een transitie met een label van meerdere letters, bijvoorbeeld $\delta(q_1, aa) = q_2$, is een verkorte schrijfwijze voor een reeks van meerdere transities, in dit geval $\delta(q_1, a) = q_1'$ en $\delta(q_1', a) = q_2$ waarbij geldt dat de tussenliggende states geen enkele andere inkomende of uitgaande transitie hebben.

(67) Voorbeeld: Neem $T = create\text{-}tree(I)$ met het resultaat uit voorbeeld (66). Dat wil zeggen: $I = \{ababb, aabb, ababa, abc, bb, bbbb, baa\}$. De werking van $generalize\text{-}simple$ is grafisch weergegeven in figuur 33 tot en met figuur 35.

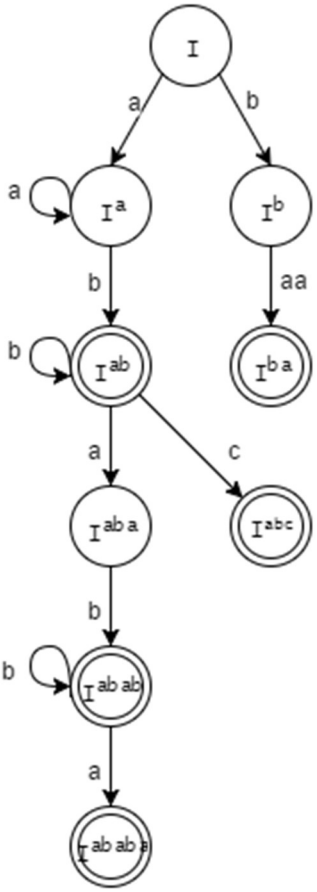


figuur 33 – Boom T, de input voor het generalize-simple algoritme



figuur 34 – Het resultaat van alleen de eerste stap van het algoritme, toegepast op alle nodes.

In dit geval is alleen het label van de transitie tussen I^b en I^{ba} veranderd.



figuur 35 – Vervolgens het resultaat van de tweede stap, toegepast op alle nodes.

In zijn paper [4] (*theorem 1*) zegt Fernau het volgende:

(68) Stelling: Afgezien van isomorfisme, is er maximaal één simple-looping DFA die een reguliere taal L genereert.

5.4 Voorbeeld (1) uitgewerkt

Uit sectie 2.3 volgt een aantal te leren oppervlakken. In deze sectie zullen we het voorbeeld uitwerken aan de hand van het SL-algoritme, dat bestaat uit $create\text{-}tree(I_+)$ en $generalize\text{-}simple(I_+)$.

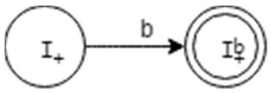
5.4.1 $create\text{-}tree(I_+)$

5.4.1.1 $I_+ = S_i(L)$

Aangezien $I_+ = S_i(L) = \{bb\}$ hoeft alleen geconstateerd te worden dat $I(b) = \{bb\}$ en $I^b = \emptyset$. De resulterende automaat $\mathcal{A}_{S_i} = (Q, A, \delta, q_0, Q_F)$ met $A = \{i, b, s, v, l, a, t, p\}$ kan worden beschreven als:

- $Q = \{I_+, I_+^b\}$
- $\delta(I_+, b) = I_+^b$
- $q_0 = I_+$
- $Q_F = \{I_+^b\}$

Deze automaat is grafisch weergegeven in figuur 36.



figuur 36 – $create\text{-}tree(S_i(L))$

5.4.1.2 $I_+ = S_b(L)$

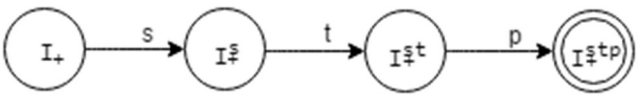
Met $I_+ = S_b(L) = \{stp, sstp\}$ vallen er iets meer verzamelingen te berekenen:

- $I_+(s) = \{stp, sstp\}$
- $I_+^s = \{tp\}$
- $I_+^s(t) = I_+^s = \{tp\}$
- $I_+^{st} = \{p\}$
- $I_+^{st} = I_+^{st}(p) = \{p\}$
- $I_+^{stp} = \emptyset$

Hetgeen resulteert in de volgende automaat $\mathcal{A}_{S_b} = (Q, A, \delta, q_0, Q_F)$ met $A = \{i, b, s, v, l, a, t, p\}$:

- $Q = \{I_+, I_+^s, I_+^{st}, I_+^{stp}\}$
- $\delta(I_+, s) = I_+^s$
- $\delta(I_+^s, t) = I_+^{st}$
- $\delta(I_+^{st}, p) = I_+^{stp}$
- $q_0 = I_+$
- $Q_F = \{I_+^{stp}\}$

Deze automaat is grafisch weergegeven in figuur 37.



figuur 37 – $create\text{-}tree(S_b(L))$

5.4.1.3 $I_+ = S_s(L)$

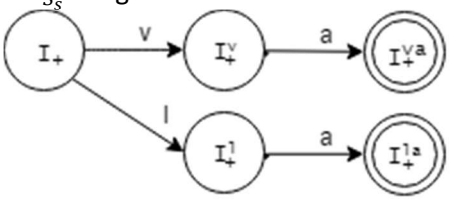
Voor $I_+ = S_s(L) = \{va, la\}$ moet worden berekend:

- $I_+(v) = \{va\}$
- $I_+^v = \{a\}$
- $I_+(l) = \{la\}$
- $I_+^l = \{a\}$
- $I_+^l(a) = I_+^v(a) = \{a\}$
- $I_+^{la} = I_+^{va} = \emptyset$

Met de resulterende automaat $\mathcal{A}_{S_s} = (Q, A, \delta, q_0, Q_F)$ met $A = \{i, b, s, v, l, a, t, p\}$:

- $Q = \{I_+, I_+^v, I_+^l, I_+^{va}, I_+^{la}\}$
- $\delta(I_+, v) = I_+^v$
- $\delta(I_+^v, a) = I_+^{va}$
- $\delta(I_+, l) = I_+^l$
- $\delta(I_+^l, a) = I_+^{la}$
- $q_0 = I_+$
- $Q_F = \{I_+^{va}, I_+^{la}\}$

Met een grafische weergave van $\mathcal{A}_{S_s}$ in figuur 38.

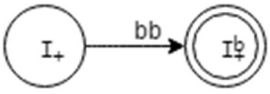


figuur 38 – create-tree$(S_s(L))$

5.4.2 *generalize-simple*(I_+)

5.4.2.1 $I_+ = S_i(L)$

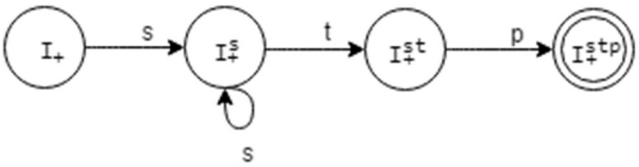
De transitie die eindigt in I_+^b krijgt het label b^n met $n = \alpha\left(\mathbb{S}(I_+(b))\right) = 2$.



figuur 39 – Het eindresultaat voor $I_+ = S_i(L)$

5.4.2.2 $I_+ = S_b(L)$

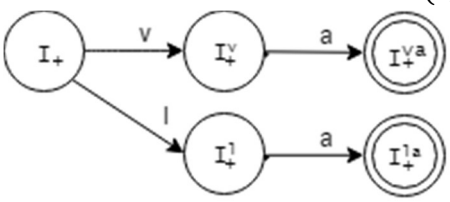
Omdat geldt $\left(\omega\left(\mathbb{S}(I_+(s))\right) = 2\right) > \left(\alpha\left(\mathbb{S}(I_+(s))\right) = 1\right)$ voegen we een lus met label s toe aan I_+^s.



1.1.1.1 figuur 40 - Het eindresultaat voor $I_+ = S_b(L)$

5.4.2.3 $I_+ = S_s(L)$

Aan het resultaat $T = create\text{-}tree\big(S_s(L)\big)$ verandert $generalize\text{-}simple(T)$ niets.



figuur 41 – Het eindresultaat voor $I_+ = S_s(L)$

5.4.3 Conclusie: de grammatica

Net als in sectie 4.4.5 geldt hier dat voor iedere $a \in A$ de variabele X_a correspondeert met het bijbehorende teken a in de gevonden reguliere expressies. De grammatica die volgt uit de met het SL-algoritme gevonden reguliere expressies:

$X_i \rightarrow iR_i\bar{\imath}$ met $R_i = \{X_bX_b\}$
$X_b \rightarrow bR_b\bar{b}$ met $R_b = \{X_sX_s^*X_tX_p\}$
$X_s \rightarrow sR_s\bar{s}$ met $R_s = \{X_vX_a, X_lX_a\}$
$X_v \rightarrow v\bar{v}$
$X_l \rightarrow l\bar{l}$
$X_a \rightarrow a\bar{a}$
$X_t \rightarrow t\bar{t}$
$X_p \rightarrow p\bar{p}$

6 Conclusie

Voordat we begonnen aan het uiteenzetten van de twee algoritmes, is de relatie tussen XML-talen en reguliere talen uitvoerig besproken. Er is duidelijk gemaakt hoe XML-documenten systematisch omgezet kunnen worden naar een set voorbeelden die geleerd kunnen worden door algoritmes die in eerste instantie ontworpen zijn voor reguliere talen.

Vervolgens zijn twee manieren besproken om uit voorbeelden van XML-documenten de bijbehorende grammatica voor die documenten te vinden. De ene manier generaliseert een triviale automaat met behulp van fusies, terwijl het andere algoritme vanuit het niets een automaat opbouwt.

Bij het algoritme op basis van onderscheidende functies is het resultaat sterk afhankelijk van de specifieke onderscheidende functie die wordt gekozen. In dit paper leggen we de nadruk op het uitleggen van het algoritme en kiezen we voor Ter als onderscheidende functie. Dit levert echter niet gegarandeerd het beste resultaat.
Fernau legt in zijn publicatie [3] uit dat het belangrijk is een goede functie te kiezen omdat de tijdcomplexiteit van het algoritme hier sterk vanaf hangt. Kiest de gebruiker een zeer geavanceerde functie, dan zal de complexiteit exponentieel toenemen. Wordt voor een simpele functie gekozen, dan zal het waarschijnlijk sterk overgeneraliseren. Fernau stelt voor de onderscheidende functie op een incrementele manier te kiezen, door bijvoorbeeld met het terminaal-onderscheidende Ter te beginnen en daaraan voorwaarden toe te voegen. In vervolgonderzoek zou voor het merging-state algoritme gezocht kunnen worden naar een specifieke onderscheidende functie die voor XML-talen het optimale resultaat geeft.

Het algoritme op basis van $simple\text{-}looping$ talen is daarentegen niet afhankelijk van een speciale functie en zal dus voor iedere set voorbeelden één uniek resultaat geven. Het nadeel van het algoritme is echter de klasse van talen die het leert. Neem bijvoorbeeld de set $I_+ = \{aabc, bbc, c\}$, een voorbeeld dat is gegenereerd met de reguliere expressie $E = a^*b^*c$ in het achterhoofd. Omdat een taal $L \in SL$ altijd moet bestaan uit paarsgewijs links-uitgelijnde expressies, zal het algoritme $L(E)$ officieel niet kunnen leren in de limiet. Het resultaat bij I_+ zal zijn $(aabc) \mid (bbc) \mid c$, en zal voor een uitgebreidere voorbeeldset van $L(E)$ uitkomen op iets als $(aa^*bb^*c) \mid (aa^*c) \mid (bb^*c) \mid c$. Dit is overigens ook het probleem dat het merging-state algoritme heeft bij een “verkeerde” keuze van onderscheidende functie.
Voor de klasse van reguliere talen die SL-infer wel kan leren, is het echter meer dan geschikt. Het algoritme is bovendien ook gemakkelijker te implementeren dan het merging-state algoritme.

Verrassend is het identieke resultaat van de twee algoritmes wanneer het gaat om de uitwerking van voorbeeld (1). Er zou onderzoek kunnen worden gedaan naar de overeenkomst tussen terminaal-onderscheidbare talen en simple-looping talen. Een interessant vraagstuk is of de verschillende algoritmes wellicht altijd een gelijk resultaat geven, wanneer bij het fuseren van nodes de functie Ter wordt gebruikt.

7 Referenties

[1] "Extensible Markup Language (XML) 1.0 (Fifth Edition)," [Online]. Available: http://www.w3.org/TR/xml/. [Accessed 13 12 2015].

[2] J. Berstel and L. Boasson, "XML grammars," in *Mathematical Foundations of Computer Science (MFCS'2000)*, vol. LNCS 1893, M. Nielsen and B. Rovan, Eds., Springer, 2000, pp. 182-191.

[3] H. Fernau, "Learning XML Grammars," in *Machine Learning and Data Mining in Pattern Recognition*, vol. LNAI 2123, P. Perner, Ed., Springer, 2001, pp. 73-87.

[4] H. Fernau, "Algorithms for learning regular expressions from positive data," *Information and Computation,* vol. 207, no. 4, pp. 521-541, 2009.

[5] D. Angluin, "Inference of Reversible Languages," *Journal of the ACM,* vol. 29, no. 3, pp. 741-765, 1982.

[6] M. Gold, "Language identification in the limit," *Information and Control,* vol. 10, no. 5, pp. 447-474, 1967.

[7] V. Radhakrishnan, "Grammatical Inference from Positive Data: An Effective Integrated Approach," PhD thesis, Department of Computer Science and Engineering, Indian Institute of Technology, Bombay (India), 1987.

[8] V. Radhakrishnan and G. Nagaraja, "Inference of regular grammars via skeletons," *IEEE Transactions on Systems, Man and Cybernetics,* vol. 17, no. 6, pp. 982-992, 1987.