Formalsprachliche Aspekte von XML

Matthias Wendlandt

5. Februar 2008

Ich möchte mich bei meinen Eltern Elke und Günter Wendlandt bedanken. Sie gaben mir in jeder Phase der Promotion den Mut und die Kraft zielstrebig zu arbeiten.

Ganz besonders möchte ich mich bei meinem Betreuer Herrn Prof. Dr. Martin Kutrib bedanken, dessen Ratschläge, Anregungen und kritische Äußerungen außerordentlich wertvoll für diese Arbeit waren.

Natürlich möchte ich mich auch bei allen anderen Personen bedanken, die an den Diskussionen über die vorliegende Arbeit beteiligt waren, insbesondere bei Katja Meckel.

Inhaltsverzeichnis

Abbildungsverzeichnis

Kapitel 1

Einleitung

1.1 Problemgegenstand

> *„I guess that XML will, by itself, solve the information interchange problem. In reality, XML just clears away some of the syntactical distractions so that we can get down to the big problem: how we arrive at common understandings about knowledge representation. That's the biggie. XML says, let's stop arguing about how we're going to represent trees and how we're going to represent attribute/value pairs. We'll just decide, let's do it as XML".*
>
> - Jon Bosak, Entwickler von XML [Bos05] -

Unternehmen tauschen Daten, wie Rechnungs- und Personaldaten, zwischen ihren Standorten aus. Die Raumfahrt liefert große Mengen an Informationen aus dem Weltraum und Konstruktionspläne von Maschinen werden archiviert.
Die zunehmende Globalisierung fordert eine einfache und schnelle Austauschbarkeit von Daten. Daher sind Firmenstandorte über Intra- oder Internet verbunden. Auch Universitäten arbeiten landesübergreifend zusammen und tauschen ihre Daten aus. Privatpersonen schicken Daten an entfernte Verwandte und Freunde.
Unmengen von Daten müssen in ihrer Darstellung klar strukturiert werden. Es muss nicht nur möglich sein das Design, sondern auch die inneren Beziehungen der Daten zu speichern, Hierarchien darzustellen und Eigenschaften Werte zuzuweisen. Vielmehr brauchen die Daten darüber hinaus ein einheitliches Format, das dennoch präzise genug ist, um die Daten mit verschiedenen Anwendungen bearbeiten zu können. Es muss ein einfacher Standard sein, für den sich interpretierende Software leicht entwickeln lässt.

Die herkömmlichen Datenformate dienten vor allem zur Beschreibung der

Darstellung der Daten und weniger zur Beschreibung ihrer Struktur. Ein einfacher Austausch war, bedingt durch die unterschiedlichen Formate der Anbieter, nicht möglich [Bos05].

In den 70er Jahren wurde SGML entwickelt. Dieses Format unterlag der generischen Kodierung, in der, statt der Formatierungscodes, deskriptive Tags verwendet wurden. Es fand vor allem Anwendung im Verlagswesen sowie im US Verteidigungsministerium. Für kleinere Firmen war es allerdings ungeeignet, da nur Spezialisten mit dieser Sprache umgehen konnten [Bos05].

In den 90er Jahren wurde HTML entwickelt. Es war der Anfang der generischen Kodierung für jedermann. HTML ist eine Anwendung von SGML, die eine feste Menge von Tags (deutsch: Abgrenzer) zur Beschreibung des Inhalts besitzt. Sie wurde für das Internet entwickelt und verbreitete sich der Einfachheit wegen sehr schnell. Es war kein Problem, interpretierende Software zu schreiben oder selber Dokumente mit ihr zu erzeugen.
Die Sprache entspricht jedoch nicht den allgemeinen Grundregeln der generischen Kodierung. Die feste Menge von Tags wird mit wenigen Ausnahmen nur zum Design des Dokumentes verwendet.

Es wurde eine Sprache benötigt, die der Mächtigkeit von SGML genügt und gleichzeitig die Einfachheit von HTML beibehält. C. M. Sperberg McQueen, John Bosak, Tim Bray und James Clark arbeiteten 1996 an einer SGML Lite Version [GP00]. Sie wurde 1998 als XML 1.0 (eXtensible Markup Language) [WWW] vom W3 Konsortium standardisiert [XML05].
XML hat sich seit der Veröffentlichung 1998 stark verbreitet. So ist es vom W3 Konsortium als Metasprache für Auszeichnungssprachen im Internet festgelegt worden und dient als Austauschformat jeglicher Daten, die über das Internet verschickt werden.
Die XML Dateien bestehen aus Tags und Fließtext zwischen den Tags. Es gibt zwei verschiedene Sorten von Tags: öffnende und schließende Tags. Ein öffnender Tag besteht aus einer öffnenden, spitzen Klammer <, einem Namen und einer schließenden, spitzen Klammer >. Ein schließender Tag besitzt zusätzlich nach der öffnenden, spitzen Klammer einen Slash /. Zu jedem öffnenden Tag muss ein schließender Tag existieren. Innerhalb eines Tagpaars können neue Tags oder Fließtext stehen. Es gibt genau einen Start- und genau einen Endtag.

Beispiel 1.1.1 Ein XML Dokument

```
<Dokument>
   <Buch>
      <Titel> Don Quijote </Titel>
         <Kapitel>
```

```
            <Ktitel> Junker Don Quijote </Ktitel>
            <Paragraph>...</Paragraph>
        </Kapitel>
    </Buch>
</Dokument>
```

□

XML ermöglicht es, Hierarchien darzustellen und einzelnen Elementen Attribute zuzuordnen. Zudem besitzt sie keine feste Menge von Tags, sondern die Menge kann selbst definiert werden. Um ein Standardformat von Dokumenten zu erfassen, können Dokumentklassen durch ein übergeordnetes Beschreibungssystem angelegt werden.
Diese Beschreibungssysteme sind ähnlich aufgebaut wie Grammatiken der Chomsky Hierarchie. Als Besonderheit verlangen sie einen Starttag zum Einleiten der rechten Seite einer Produktion und einen Endtag zum Beenden der Produktion.
Es existiert eine große Anzahl verschiedener Beschreibungssysteme, wie zum Beispiel die DTD [DTD07], das XML Schema [XML07], TREX [Claa] oder RELAX NG [Clab].
Die Beschreibungssysteme erläutern, wie die Tagpaare eines XML Dokumentes angeordnet sein müssen.

Eine DTD besteht aus einer besonderen Art von Tags. Diese Tags werden auch als Elemente bezeichnet. Jedes Element beschreibt ein Tagpaar aus dem später entstehendem XML Dokument. Sie beginnen mit einer spitzen Klammer <, darauf folgen ein Ausrufezeichen !, der Elementname, eingeklammert und durch Komma getrennt die Namen, der in diesem Element enthaltenen Tags, bzw. mit ♯PCDATA gekennzeichnet, die Möglichkeit Fließtext zu erlauben. Zum Schluss steht die schließende, spitze Klammer >.

Beispiel 1.1.2 Die DTD *Buch*

```
<!ELEMENT Buch (Titel, ISBN, Kommentare*, Autor+)>
<!ELEMENT Titel (♯PCDATA)>
<!ELEMENT ISBN (♯PCDATA)>
<!ELEMENT Kommentare (♯PCDATA)>
<!ELEMENT Autor (♯PCDATA)>
```

□

Alles was in den Elementen nach dem Namen in den Klammern aufgeführt wird, heißt Inhaltsmodell und kann aus einem regulären Ausdruck über den Namen der Elemente bestehen. Ein Stern bedeutet dabei, dass das Element in dem XML Dokument mehrere Male oder auch keinmal hintereinander aufgeführt werden kann. Das Plus bedeutet, dass es mindestens einmal enthalten sein muss, genauso gut aber auch mehrere Male vorkommen kann.

Ein großer Anwendungsbereich von XML findet sich im Datenaustausch. Durch die Beschreibungssysteme kann eine Struktur der Daten festgelegt werden. Die Beschreibungssyteme geben an, welche Tags existieren und in welcher Form sie angeordnet werden dürfen. Die Verarbeitung der Daten durch Programme wird hiermit vereinfacht.
Der Datenaustausch kann durch die allgemeine Struktur der Daten zwischen zwei gleichen Systemen, zum Beispiel zwei Datenbanken oder zwischen zwei unterschiedlichen Systemen, zum Beispiel einem Textverarbeitungssystem und einem Internet Content Management System, geschehen. Die Programme brauchen lediglich einen Algorithmus, der angibt, wie die Struktur, zum Beispiel eine hierarchische Beziehung, verarbeitet wird.
Ein weiterer Vorteil von XML im Datenaustausch ist, dass komplizierte Datenstrukturen, wie Objekte, Listen und Bäume, dargestellt und verschickt werden können [See03]. Die standardisierte Struktur macht es verarbeitenden Programmen einfach, die Struktur zu erkennen, obwohl sie womöglich den Inhalt der Daten nicht kennen.

XML kann beim Festlegen von Standarddokumenten hilfreich sein. Briefe und Firmendokumente erhalten damit eine einheitliche Form, die von verschiedenen Softwareprodukten bearbeitet werden kann. So ist der Firmenauftritt standardisiert.

In modernen Programmen werden XML Dateien verwendet, um Daten abzuspeichern. Spielstände von Computerspielen werden darin abgelegt oder Maße von Bauteilen gespeichert.
Die DOM (Document Object Model) (Spezifikation, [DOM07]) (nähere Informationen über den Gebrauch,[GP00]) und SAX (Simple API for XML) (Spezifikation,[SAX07]) (nähere Informationen über den Gebrauch, [Meg]) Schnittstellen erleichtern es den Programmierern, Daten aus den Dateien auszulesen. Die Programmierung eines komplizierten Dateiinhaltsverzeichnises gehört der Vergangenheit an. Die DOM Schnittstelle [Har99] liefert die Daten als Baum und bietet über Methoden einfache Zugriffsmöglichkeiten auf die Elemente. Die SAX Schnittstelle liefert die Daten seriell.
Die programmiertechnischen Vorteile können auch in der einfacheren Planung der Software genutzt werden. Es werden keine komplizierten Schnittstellen definiert, Daten werden durch XML Beschreibungsmodelle standardisiert und können somit einfacher verarbeitet werden.

Die Speicherung von Daten in Datenbanken kann durch XML modernisiert werden. So gibt es Ansätze, XML als eine Art Datenbank [See03] zu verwenden. Weil XML mächtigere semantische Beziehungen als relationale Datenbanken darstellen kann, lassen sich beispielsweise Hierarchien abbilden.

Im Bereich der Datenbankenforschung nimmt XML einen wichtigen Platz ein. In diesen Bereich fallen vor allem Verbindungen zu den relationalen Datenbanken, die die häufigste Speicherform von Daten in Datenbanken sind. Transformationsalgorithmen, welche dafür sorgen, dass eine XML Datei in ein relationales Datenbankkonzept umgewandelt wird, wurden in [DFS99] [FK99] [STZ+99] erforscht. Die Arbeit [STZ+99] gibt drei Algorithmen an, mit denen aus einer DTD ein relationales Modell erzeugt wird. In ihr wird vor allem die Anzahl der entstehenden Tabellen untersucht. [FK99] stellt acht verschiedene Algorithmen vor, in denen er auch DTDs in ein relationales Datenbankmodell umwandelt und berücksichtigt dabei vor allem die Leistungsmerkmale in Bezug auf Attribut-Werte Paare.
[DFS99] gibt einen Algorithmus an, der ein XML Dokument direkt in ein relationales Modell umwandelt, ohne eine DTD zu benötigen.
Alle Algorithmen setzen die Struktur der XML Daten in ein relationales Schema um, doch die Semantik der Daten bleibt unbehandelt.
Das folgende Beispiel zeigt eine Überführung einer DTD:

Beispiel 1.1.3 Überführung DTD *Tagung* in ein relationales Schema

<!ELEMENT Tagung (Titel,Gesellschaft,Jahr,Papier+)>
<!ELEMENT Papier (Pid,Titel,Zusammenfassung?)>

Es sei angenommen, der *Titel* und das *Jahr* in *Tagung* identifizieren eindeutig jedes Tupel in *Tagung*.
Wenn der Hybrid Inlining Algorithmus aus [STZ+99] auf die DTD angewendet wird, entsteht folgendes relationales Modell:

Tagung (Titel,Gesellschaft,Jahr)
Papier (Pid,Titel,Tagung$_{Titel}$,Tagung$_{Jahr}$,Zusammenfassung)

Die Struktur wird richtig übertragen, aber die Algorithmen fordern keine korrekte Semantik. Es könnte ein Tupel

$$t_0 :(100,\text{`DTD}\ldots\text{`},\text{`ER`},\ 3000,\ \text{`}\ldots\text{`})$$

in *Papier* eingefügt werden. Die DTD verlangt, dass ein *Papier* nur in Verbindung mit einer *Tagung* bestehen darf. Hierbei ist es möglich, dass eine *Tagung* nicht existiert. Das führt zu einem semantischen Fehler.
In relationalen Datenbankbedingungen könnte dies durch eine Teilmengenabhängigkeit gelöst werden.

□

Die Arbeiten [LMM00] [MLM01a] [LMCC02] beschäftigen sich mit diesen Problemen. Es werden Algorithmen vorgestellt, mit denen die XML Modelle semantisch korrekt dargestellt werden. Sie beschränken sich jedoch ausschließlich auf DTD Beschreibungsmodelle. Dazu werden Praxisstudien beschrieben.

In einer weiteren Arbeit [SSB+01] wird beschrieben, wie aus einem relationalen Modell Daten abgefragt werden und in einem XML Dokument präsentiert werden.
Dies dient vor allem der Darstellung bestehender Daten im Internet oder hilft bei der Übertragung von Daten via Internet. Dabei werden SQL Kommandos mit Zusätzen ergänzt. Das Ergebnis einer Anfrage ist dann eine XML Datei.

1.2 Problembereich

Produktionen der XML-artigen Grammatiken bestehen auf der rechten Seite aus regulären Ausdrücken. Die meisten praktischen Beschreibungsmodelle, wie zum Beispiel DTDs, dürfen keine mehrdeutigen regulären Ausdrücke enthalten. Jeder mehrdeutige reguläre Ausdruck kann zwar durch den in [BEGO71] vorgestellten Algorithmus in einen eindeutigen regulären Ausdruck umgewandelt werden, jedoch ergibt sich daraus eine maximale Laufzeit von $O(2^n)$. Algorithmen mit solchen Laufzeiten wären in der Praxis nicht einsetzbar.
Darüber hinaus ist die Eindeutigkeit, wie sie in [BEGO71] beschrieben wird, so definiert, dass es für einen eindeutigen regulären Ausdruck r für ein beliebiges Wort w der Sprache nur genau eine Ableitung geben darf. So ist der reguläre Ausdruck $r_1 = (a|b)^*aa^*$ mehrdeutig. Eine Indizierung eines regulären Ausdrucks bedeutet, dass jedem Zeichen des Ausdrucks ein Index beigefügt wird. Die Indizierung beginnt bei 1 und wird bei jedem Auftreten des Zeichens um eins erhöht. Wird r_1 indiziert, so wird er zu $(a_1|b_1)^*a_2a_3^*$. Daraus lassen sich für das Wort aaa folgende drei Ableitungen erzeugen $a_1a_1a_2$, $a_1a_2a_3$ und $a_2a_3a_3$.

Für einen deterministischen Kellerautomaten, der als Basis für Parser genutzt wird, ist es wichtig, dass er anhand der nächsten k Zeichen entscheiden kann, welche Überführung angewendet werden muss. In der Praxis wird normalerweise der Fall $k = 1$ gewählt.
Betrachten wir den regulären Ausdruck $r = (a|b)^*a$. Der Ausdruck r ist eindeutig. Wird jedoch der Ausdruck indiziert $(a_1|b_1)^*a_2$, so ist es für ein Wort $w = baaa, w \in L(r)$ nach Einlesen des Buchstabens b nicht klar, ob nach a_1 abgeleitet werden muss oder nach a_2.
Für die Inhaltsmodelle einiger XML-artiger Grammatiken sind nur solche regulären Ausdrücke erlaubt, bei denen anhand des nächsten Eingabezeichens (1-lookahead) entscheidbar ist, welcher Teil des regulären Ausdrucks benutzt werden muss, um das Eingabewort zu erzeugen. Diese regulären Ausdrücke wurden von Brüggemann-Klein definiert und untersucht [BK93b] [BKW92] [BK93a] [BKW98]. Sie nennen sich 1-eindeutige reguläre Ausdrücke. Sie sind eine echte Teilmenge der Menge der regulären Ausdrücke.

In einer Arbeit von 2006 [HW07] wird die 1-Eindeutigkeit erweitert, mit einem neuen Namen versehen und zu k-lookahead Determinismus und k-block Determinismus. Bei der k-lookahead Variante werden die nächsten k Zeichen gelesen, jedoch nur das nächste Zeichen verarbeitet, wohingegen bei der anderen Variante k Zeichen gelesen und alle verarbeitet werden.
Es wird gezeigt, dass eine echte Hierarchie innerhalb der k-block deterministischen Sprachen existiert und dass die k-block deterministischen Sprachen eine echte Teilmenge der k-lookahead deterministischen Sprachen sind. Es fehlt der Beweis, dass die k-lookahead Determinismus Hierarchie echt ist.

Im formalsprachlichen Bereich gibt es verschiedene Forschungsansätze, die sich mit den Beschreibungsmodellen von XML befassen. Die Beschreibungsmodelle werden als Grammatik aufgefasst. Im Wesentlichen gibt es dabei zwei verschiedene Ansatzpunkte:

- Beschreibungsmodelle werden als Zeichengrammatiken aufgefasst.
- Beschreibungsmodelle werden als Baumgrammatiken aufgefasst.

Die erste Strukturierung aus den bestehenden Beschreibungsmodellen in Grammatiken wurde in [MLM01b] und [LMM00] getroffen. Hierbei wurden die bestehenden Beschreibungsmodelle in vier verschiedene Baumgrammatiktypen eingeteilt: Local Baumgrammatiken, Single-Type Baumgrammatiken, Restrained Competition Baumgrammatiken und reguläre Baumgrammatiken.
Die Grammatiken bestehen dabei aus einem 4-Tupel $G = (N, T, S, P)$. Die Produktionen haben die Form

$$X \longrightarrow a(m), m \in Reg(N), a \in T, X \in N.$$

Die Produktionen werden hierbei als Baumproduktionen aufgefasst, wobei a ein Knoten ist, (m) die Nachfolgebäume sind und $Reg(N)$ ein regulärer Ausdruck über Nichtterminalzeichen ist.
Es wird eine Hierarchie zwischen den Grammatiken gezeigt, wobei die regulären Baumgrammatiken die ausdrucksmächtigsten Grammatiken sind und die Local Baumgrammatiken die ausdrucksschwächsten.
Desweiteren werden Algorithmen zum Parsen der entstehenden Modelle erzeugt. Dabei werden für die drei schwächeren Grammatiken deterministische Baumautomaten verwendet. Für die regulären Baumgrammatiken wird ein nichtdeterministischer Baumautomat angegeben.

Nähere Analysen von Baumgrammatiken werden in [CDGJ97] durchgeführt. Es werden nur die regulären Baumgrammatiken behandelt. Hier werden Abschlusseigenschaften, Entscheidbarkeitsfragen, Automatenmodelle und weiterführende Ergebnisse der regulären Baumgrammatiken präsentiert.

Es gibt in [MLM01b] und [LMM00] jedoch Unterschiede zwischen ihrem Modell und dem Modell von [CDGJ97]. In [MLM01b] und [LMM00] dürfen die regulären Ausdrücke der Inhaltsmodelle einen Kleenestern enthalten. Die regulären Baumgrammatiken sind zwar unter Iteration abgeschlossen, jedoch nur unter geschachtelter Iteration und nicht unter einer konkatenierten Iteration.

In einer weiteren Arbeit [LC00] werden sechs Beschreibungsmodelle auf ihre Zugehörigkeit zu den Grammatiktypen überpüft.

Murata [MLM01b] beschreibt einen Grammatiktyp, der den Baumgrammatiken ähnelt. Die Grammatiken heißen Hedge Grammatiken. Dabei muss eine Sprache nicht genau einen Wurzelknoten besitzen, sondern kann auch aus mehreren Bäumen bestehen.
Es werden sowohl Hedge Automaten vorgestellt, als auch die booleschen Abschlusseigenschaften ihrer Sprache. So sind sie sowohl unter Durchschnitt als auch unter Vereinigung und Komplement abgeschlossen.
In der Praxis finden die Hedge Grammatiken jedoch keine Anwendung. In der XML Spezifikation ist eindeutig definiert, dass der Anfang eines XML Dokumentes aus einem Starttag und das Ende aus dem dazugehörigen Endtag besteht [WWW]. Genauso sind HTML Dokumente aufgebaut, die in neuen Versionen eine Dokumentklasse von XML sind. Mit dieser Struktur wird ein Baum beschrieben. In der Spezifikation von XML wird dies als Wohlgeformtheit bezeichnet.

Ein wichtiger Forschungsbereich, mit dem sich diese Arbeit vor allem beschäftigt, sind XML Beschreibungsmodelle aus Sicht von Wortgrammatiken. Den Anstoß geben zwei Arbeiten von Boasson und Berstel [BB02b] [BB00], in denen ein Grammatiktyp namens XML Grammatik definiert wird. Diese Art von Grammatik ist vergleichbar mit DTDs, die durch die Formalisierung abstrahiert werden.
XML Grammatiken besitzen Produktionen mit einem Nichtterminalzeichen auf der linken Seite und einem Tagpaar auf der rechten Seite, das einen regulären Ausdruck über Nichtterminalen einschließt:

$$X \longrightarrow xReg(N)\overline{x}.$$

Die Produktionen haben die spezielle Form, dass zu jedem Terminalzeichen $x \in T, T = A \cup \overline{A}$, genau ein Nichtterminalzeichen existiert.
Für diese Form der Grammatiken wurde gezeigt, dass es zu jeder XML Sprache eine eindeutige minimale Grammatik gibt. Weiter wurde gezeigt, dass sie unter Durchschnitt, nicht aber unter Vereinigung und somit auch nicht unter Komplement abgeschlossen sind.

Eine weitere Arbeit befasst sich mit einer Verallgemeinerung der XML Grammatiken. Die Grammatiken heißen balancierte Grammatiken [BB02b]. Sie unterliegen nicht mehr der Einschränkung, dass jedem Terminalzeichen $x \in T$ genau ein Nichtterminalzeichen zugeordnet sein muss.
Wichtige Erkenntnisse ergeben sich durch den Beweis, dass es zu jeder balancierten Grammatik eine eindeutige Minimalgrammatik gibt. Die balancierten Sprachen sind unter allen booleschen Operationen abgeschlossen.
Darüber hinaus ist es entscheidbar, ob eine balancierte Grammatik eine reguläre bzw. eine kontextfreie Sprache erzeugt. Eine wichtige Eigenschaft, auch in Bezug auf die folgende Arbeit, ist, dass es für jede balancierte Grammatik eine codeterministische äquivalente Grammatik gibt. In älterer Literatur wird diese Eigenschaft als backwards deterministic bezeichnet. Die Eigenschaft legt zugrunde, dass es keine zwei Produktionen, $X \longrightarrow m$ und $X' \longrightarrow m$, in G gibt, ohne $X = X'$ zur Folge zu haben.

Es gibt einige ältere Ansätze, die sich mit ähnlichen Grammatiktypen befassen:

- Parenthesis Grammatiken
- Bracketed Grammatiken
- Chomsky-Schützenberger Grammatiken

Die Parenthesis Grammatiken wurden von Knuth [Knu67] und McNaughton [McN67] untersucht. Parenthesis Grammatiken sind Grammatiken $G = (N, T, S, P)$, wobei jede Produktion die Form

$$X \longrightarrow am\overline{a}$$

hat. Das Terminalalphabet T besteht aus einem Tagpaar $\{a, \overline{a}\}$ und einer zu T disjunkten Menge von Zeichen B, also $T = \{a, \overline{a}\} \cup B$. Die Inhaltsmodelle m in den Produktionen bestehen aus Nichtterminalen und Zeichen aus B, $m \in (N \cup B)^*$.
Es existiert nur ein Klammertyp und die Inhaltsmodelle sind endlich.
Ist $B = \emptyset$, wird die Grammatik als „streng“ bezeichnet.

Die Bracketed Grammatiken wurden von Ginsburg und Harrison [GH67] erforscht. Die Terminalzeichen bestehen aus der Menge der Zeichen $T = A \cup \overline{B} \cup C$. Die einzelnen Mengen sind disjunkt. Jede Produktion hat die Form

$$X \longrightarrow am\overline{b}, a \in A, \overline{b} \in \overline{B}.$$

Für das Inhaltsmodell m gilt $m \in (N \cup C)^*$. Es existiert eine Bijektion zwischen den Terminalen in A und den Produktionen. So kann in jedem Ableitungsschritt eindeutig erkannt werden, welche Produktion benutzt werden

muss.
Der Unterschied zu den XML Grammatiken besteht darin, dass die Inhaltsmodelle endlich sind.

Die Chomsky-Schützenberger Grammatiken werden in dem Chomsky-Schützenberger Theorem benutzt. Hierbei besteht das Terminalphabet aus den Zeichen $T = A \cup \overline{A} \cup B, m \in N^*$. Die Produktionen haben die Form

$$X \longrightarrow am\overline{a}.$$

Die Beziehung zwischen den Zeichen $a \in A$ und den Produktionen ist bijektiv.

1.3 Fragestellung und Aufbau

Diese Arbeit beschäftigt sich in zwei verschiedenen Richtungen mit XML-artigen Grammatiken.
Der erste Ansatz ist rein theoretisch und beschäftigt sich mit den formalsprachlichen Eigenschaften der Beschreibungsmodelle von XML. Schwerpunkt bilden die genaue Einordnung in die Chomsky Hierarchie, die Abschlusseigenschaften und die Entscheidbarkeitsfragen.
Die von Berstel und Boasson definierten Grammatiken werden, durch die in [MLM01b] gezeigten Grammatiken, ergänzt und bilden mit einem neuen Typ von Grammatiken, den RegDyck Grammatiken, die Basis der Arbeit:

- XML Grammatiken
- einfache balancierte Grammatiken
- konkurrenzverhindernde balancierte Grammatiken
- RegDyck Grammatiken
- balancierte Grammatiken.

Der zweite Ansatz dient dazu, einen praktischen Nutzen aus diesen Erkenntnissen zu ziehen. Es werden Parsergenerierungsalgorithmen und Parser konstruiert und deren Laufzeit bestimmt.

Aus diesen beiden Ansätzen ergibt sich die zentrale Fragestellung: Wie können die formalsprachlichen Eigenschaften der Beschreibungsmodelle von XML praktisch genutzt werden?
Im Einzelnen ergeben sich dadurch folgende Fragen:

- Gibt es neue, verbesserte XML-artige Grammatiken?
- Zu welchem speziellen Typ der Chomsky Hierarchie gehören die Familien der XML-artigen Sprachen?

- Gibt es eine Hierarchie zwischen den XML-artigen Sprachfamilien?
- Wie können Parser generiert werden?
- Mit welchen Methoden kann geparst werden?
- Welche Laufzeiten haben Parser und Parsergenerator?
- Welche Abschlusseigenschaften besitzen die Sprachen der Grammatiken?

Zuerst werden im zweiten Kapitel die grundlegenden Definitionen der Grammatiken beschrieben. Hierbei werden die bestehenden Definitionen der XML Grammatiken, der balancierten Grammatiken und die Inhaltsmodelleigenschaft des 1-lookahead Determinismus erläutert. Anschließend werden die bereits bestehenden Definitionen der Single-Type Baumgrammatiken und der Restrained Competition Baumgrammatiken in eine Definition der Wortgrammatiken übersetzt.

In Kapitel 3 wird ein neuer Grammatiktyp „RegDyck Grammatiken“ hergeleitet. Sie beschreiben den Durchschnitt zwischen regulären Sprachen und den Dyck Sprachen. Es wird dazu auch ein erkennender Automat angegeben.

Kapitel 4 beschäftigt sich mit der Einordnung der XML-artigen Grammatiken. Zuerst wird überprüft, in welche Klasse der Chomsky Hierachie die Grammatiken gehören. Danach wird eine Hierarchie innerhalb der XML-artigen Grammatiken gezeigt. Der letzte Teil des Kapitels beschäftigt sich mit Parsergeneratoren und Parsingmethoden. Es werden Laufzeitanalysen durchgeführt und neue Methoden zur Parserkonstruktion und zum Parsen erzeugt. Die Grundlagen hierfür legen die bis dahin erörterten theoretischen Grundlagen.

Im letzten Kapitel werden die Abschlusseigenschaften der Grammatiken erforscht: Vereinigung, Durchschnitt, erweitertes Komplement, erweiterte Konkatenation, erweiterte Iteration, erweiterte Substitution und erweiterter Homomorphismus. Danach folgt die Klärung von Entscheidbarkeitsfragen.

Kapitel 2

Grundlagen

2.1 Allgemeine Definitionen

Die Menge der natürlichen Zahlen wird mit $\mathbb{N} = \{1, 2, 3, \ldots\}$ bezeichnet. Die Vereinigung $\mathbb{N} \cup \{0\}$ wird mit $\mathbb{N}_0$ gekennzeichnet. Für die Potenzmenge einer endlichen Menge M wird 2^M geschrieben. Eine endliche, nichtleere Menge A von Zeichen wird als Alphabet bezeichnet. Ein Wort ist eine endliche Folge von Zeichen über einem Alphabet A. A^+ ist die Menge der nichtleeren Worte über A. Die Vereinigung $A^+ \cup \lambda$ wird durch A^* dargestellt. Die leere Menge wird mit $\emptyset$ bezeichnet. Es gilt $A^0 = \{\lambda\}$.

Die Länge eines Wortes $w \in A^*$ wird abkürzend als $|w|$ geschrieben, wobei gilt $|\lambda| = 0$. Die Spiegelung $u_n \ldots u_2 u_1$ eines Wortes $w = u_1 \ldots u_n$ wird durch w^R gekennzeichnet.

Es sei A ein Alphabet, dann ist $\overline{A}$ die zu A disjunkte Menge von Zeichen, für die gilt, wenn $a \in A$, dann ist $\overline{a} \in \overline{A}$ und wenn $a \in A, b \in A, a \neq b$, dann ist $\overline{a} \neq \overline{b}$.

Ein Dyckprime ist ein Wort $x\overline{x}$ mit $x \in A, \overline{x} \in \overline{A}$ oder ein Wort $xd_1 \ldots d_n\overline{x}$, $n \in \mathbb{N}$, wobei $d_1, \ldots, d_n$ Dyckprimes sind. Es sei T ein Alphabet mit $T = A \cup \overline{A}$, dann ist D_T die Menge aller Dyckprimes über diesem Alphabet.

Ein Dyckwort über dem Alphabet T hat die Form $v_1 \ldots v_n$, wobei $v_1, \ldots, v_n \in D_T, n \in \mathbb{N}$. Die Menge aller Dyckwörter über dem Alphabet T wird mit W_T bezeichnet.

Definition 2.1.1 Eine reguläre Grammatik ist eine Grammatik $G = (N, T, S, P)$, wobei gilt:

- N ist die Menge der Nichtterminalzeichen.

- T ist die Menge der Terminalzeichen mit $T \cap N = \emptyset$.
- S ist das Startsymbol.
- P ist die Menge der Produktionen der Form

$$X \longrightarrow Yz \text{ und}$$
$$X \longrightarrow z$$

  mit $X, Y \in N, z \in T^*$.

□

$L(G) = \{w \mid w \in T^* \wedge S \Rightarrow^* w\}$ ist die von der Grammatik G erzeugte reguläre Sprache.

Definition 2.1.2 (Pumping Lemma für reguläre Sprachen)
Es sei L eine reguläre Sprache. Dann existiert eine natürliche Zahl n, so dass sich alle Wörter $x \in L$ mit $|x| \geq n$ in $x = uvw$ zerlegen lassen, so dass folgende drei Bedingungen erfüllt sind:

1. $|v| \geq 1$
2. $|uv| \leq n$
3. $uv^i w \in L, \forall i \in \mathbb{N}_0$.

□

Definition 2.1.3 Eine kontextfreie Grammatik ist eine Grammatik $G = (N, T, S, P)$, wobei gilt:

- N ist die Menge der Nichtterminalzeichen.
- T ist die Menge der Terminalzeichen $N \cap T = \emptyset$.
- S ist das Startsymbol.
- P ist die Menge der Produktionen der Form

$$X \longrightarrow \alpha$$

  mit $X \in N, \alpha \in (N \cup T)^*$.

□

Definition 2.1.4 (selbsteinbettende Nichtterminalzeichen)
Es sei $G = (N, T, S, P)$ eine kontextfreie Grammatik. Ein Nichtterminalzeichen $X \in N$ wird als selbsteinbettend bezeichnet, wenn eine Ableitung $X \Rightarrow^* zXz_0, z, z_0 \in T^+$ existiert.

□

Die Ableitung eines Wortes $w \in L(G)$, wird bildlich durch einen Parsebaum beschrieben.

Definition 2.1.5 (Parsebaum)
Es sei $G = (N, T, S, P)$ eine kontextfreie Grammatik. Ein Parsebaum (Ableitungsbaum) ist ein Baum mit folgenden Eigenschaften:

- Die Wurzel wird mit dem Startsymbol S markiert.
- Jedes Blatt ist mit einem Zeichen $a \in T$ oder λ gekennzeichnet.
- Jeder innere Knoten ist mit einem Nichtterminalzeichen $X \in N$ gekennzeichnet.
- Es sei $X \in N$ die Beschriftung eines inneren Knoten und $X \longrightarrow X_0 \dots X_n$ die ausgewählte Produktion, dann sind $X_0, \dots, X_n \in (N \cup T)$ die Nachfolger des Knotens. Wenn $X \longrightarrow \lambda$ die gewählte Produktion ist, dann ist λ der einzige Nachfolger.

□

Definition 2.1.6 Ein nichtdeterministischer endlicher Automat (NFA) $\mathcal{R} = (S, A, s_0, \delta, F)$ ist ein System mit:

- S ist die Menge der Zustände.
- A ist die Menge der Eingabesymbole.
- s_0 ist der Startzustand.
- $F \subseteq S$ ist die Menge der Endzustände.
- $\delta : S \times A \longrightarrow 2^S$ ist die Überführungsfunktion.

Wir nehmen an, die Überführungsfunktion eines NFA sei total. Falls für eine Situation kein Folgezustand definiert ist, dann bildet der NFA in die leere Menge ab.

□

Die Erweiterung der Funktion δ für Wörter ist: $\Delta : S \times A^* \longrightarrow 2^S$.
Eine Konfiguration (s, w) von $\mathcal{R}$ ist ein Paar $S \times A^*$. Die Konfiguration (s, w) bedeutet, dass sich $\mathcal{R}$ im Zustand s befindet und w die noch nicht gelesene Eingabe ist.
$L(\mathcal{R}) = \{w \in A^* \mid \Delta(s, w) \in F\}$ ist die von dem nichtdeterministischen endlichen Automat $\mathcal{R}$ erkannte Sprache.

Definition 2.1.7 Ein nichtdeterministischer Kellerautomat $\mathcal{K} = (S, A, G, \triangle, s_0, \delta)$ ist ein System mit:

- S ist die Menge der Zustände.

- A ist die Menge der Eingabesymbole.
- G ist die Menge der Kellersymbole.
- $\triangle \in G$ ist das Kellerendesymbol.
- s_0 ist der Startzustand.
- Die Überführungsfunktion δ bildet von $S \times (A \cup \{\lambda\}) \times G$ in die endlichen Teilmengen von $S \times G^*$ ab.

Wir nehmen an, der Kellerautomat akzeptiert mit leerem Keller.

□

Eine Konfiguration (s, w, z) von $\mathcal{K}$ ist ein Tripel aus $S \times A^* \times G^*$, wobei sich $\mathcal{K}$ im Zustand s befindet, w die noch nicht gelesene Eingabe und z der Kellerinhalt ist.
$L(\mathcal{K}) = \{w \in A^* \mid (s_0, w, \triangle) \vdash^* (s', \lambda, \lambda) \wedge s' \in S\}$ ist die von dem Kellerautomat $\mathcal{K}$ erkannte Sprache.

Definition 2.1.8 Ein deterministischer Kellerautomat $\mathcal{K} = (S, A, G, \triangle, s_0, \delta, F)$ ist ein System mit:

- S ist die Menge der Zustände.
- A ist die Menge der Eingabesymbole.
- G ist die Menge der Kellersymbole.
- $\triangle \in G$ ist das Kellerendesymbol.
- s_0 ist der Startzustand.
- $F \subseteq S$ ist die Menge der Endzustände.
- Für jedes $s \in S$ und $g \in G$ ist $\delta(s, a, g)$ für alle $a \in A$ nicht definiert, wenn $\delta(s, \lambda, g)$ definiert ist.
  Für alle $s \in S, g \in G$ und $a \in (A \cup \{\lambda\})$ enthält $\delta(s, a, g)$ höchstens ein Element.

□

Definition 2.1.9 Es sei A ein Alphabet, das keines der Zeichen aus der Menge $V = \{|, ^*, ^+, ?, \emptyset, (,)\}$ enthält. Reguläre Ausdrücke über einem Alphabet A werden wie folgt festgelegt:

- Alle $p \in A$ sowie $\emptyset$ sind reguläre Ausdrücke.
- Wenn p und q reguläre Ausdrücke sind, dann ist auch die Konkatenation (pq) ein regulärer Ausdruck.

- Wenn p und q reguläre Ausdrücke sind, dann ist auch die Vereinigung $(p|q)$ ein regulärer Ausdruck.
- Wenn p ein regulärer Ausdruck ist, dann ist auch $(p)^*$ ein regulärer Ausdruck.

□

Die Klammern können weggelassen werden, wenn sich dadurch die Semantik des regulären Ausdrucks nicht ändert.
Im Folgenden gilt: $(p)^+ = (p)(p)^*$, $(p)? = (p|\emptyset)$ und $\lambda = \emptyset^*$. Die Zeichen $^*, ^+, ?, |$ werden als Operatoren bezeichnet.

Jeder reguläre Ausdruck r über einem Alphabet A legt eine Sprache $L(r)$ folgendermaßen fest:

- Die durch $\emptyset$ bezeichnete Sprache ist die leere Sprache.
- Die durch $a \in A$ bezeichnete Sprache enthält nur das Wort a.
- Für reguläre Ausdrücke p und q über dem Alphabet A gilt: $L(p|q) = L(p) \cup L(q), L(pq) = L(p)L(q), L(p^*) = (L(p))^*$.

Es sei A ein Alphabet, dann ist $Reg(A)$ die Menge der regulären Ausdrücke über diesem Alphabet.
Ein regulärer Ausdruck r über einem Alphabet A wird zu einem indizierten regulären Ausdruck r', indem jedes Vorkommen eines Zeichens $a \in A$ mit einem Index $i \in \mathbb{N}$ versehen wird, so dass die Kombination aus dem Zeichen a und dem Index i eindeutig in dem regulären Ausdruck r' ist. Die Indizierung beginnt bei 1. Es sei $r = uvuuwvwww, u, v, w \in A$ ein regulärer Ausdruck, dann ist der indizierte reguläre Ausdruck $r' = u_1v_1u_2u_3w_1v_2w_2w_3w_4$.
π ist eine injektive Abbildung, die für den gegebenen regulären Ausdruck r einen indizierten regulären Ausdruck r' zurückgibt.

Definition 2.1.10
Sei r ein regulärer Ausdruck über dem Alphabet A und sei $V = \{*, |, (,), \cdot\}$ die Menge der Operatoren. Die folgenden Abbildungen fassen den regulären Ausdruck als Wort über dem Alphabet A und den Operatoren V auf.

$$A' = \{a_i \mid a \in A, i \in \mathbb{N}\}$$

ist die Menge der indizierten Zeichen. Die folgenden Abbildungen

$$\gamma : (A \cup V)^* \times A \longrightarrow \mathbb{N}$$

$$\gamma(w, a) = |w|_a \text{ in } w$$

$$\psi : A \times \mathbb{N} \longrightarrow A'$$

$$\psi(a, i) = a_i$$

dienen dazu, die Anzahl des Vorkommens eines Buchstabens in einem Wort zu zählen und einem Buchstaben einen Index zuzuweisen.
Mit diesen Abbildungen können wir eine Abbildung π definieren, die zu einem regulären Ausdruck r, welcher als Wort aufgefasst wird, ein indiziertes Wort liefert, welches dann wieder als indizierter regulärer Ausdruck aufgefasst wird.

$\pi : (A \cup V)^* \longrightarrow (A' \cup V)^*$
$\pi(\epsilon) = \epsilon$
$x \in (A \cup V)^*, a \in A : \pi(xa) = \pi(x)\psi(a, \gamma(xa, a))$
$x \in (A \cup V)^*, y \in V : \pi(xy) = \pi(x)y$

□

Beispiel 2.1.4 Indizierter regulärer Ausdruck

$$r = (a|b)^*a(a|b)$$
$$\pi(r) = r' = (a_1|b_1)^*a_2(a_3|b_2)$$

□

Es sei π^{-1} die Abbildung, welche die Indizierung eines regulären Ausdrucks aufhebt: $\pi^{-1}(r') = r$. Ebenso hebt π^{-1} die Indizierung eines Wortes auf: $\pi^{-1}(w') = w, w' \in L(\pi(r)), w \in L(r)$.
Ein regulärer Ausdruck r heißt eindeutig, wenn gilt:

$$\forall w' \in L(\pi(r)) : \neg\exists w \in L(\pi(r)) : (\pi^{-1}(w) = \pi^{-1}(w') \wedge w \neq w').$$

2.2 k-lookahead deterministische reguläre Ausdrücke

Im Folgenden werden k-lookahead deterministische reguläre Ausdrücke definiert. Sie sind wichtig für die Untertypen von XML-artigen Grammatiken, die in dieser Arbeit untersucht werden. Eine XML-artige Grammatik $G = (N, T, S, P)$ besitzt Produktionen der Form

$$X \longrightarrow xm\overline{x},$$

wobei m ein regulärer Ausdruck über Nichtterminalen ist und $X \in N, x, \overline{x} \in T$.
In einigen XML-artigen Grammatiktypen ist nicht jeder beliebige reguläre Ausdruck erlaubt. Eine allgemeine Einschränkung der regulären Ausdrücke ist, dass sie eindeutig sein müssen. Die Eigenschaft, dass jeder reguläre Ausdruck in einen eindeutigen regulären Ausdruck in maximal $O(2^n)$ Zeit umgewandelt werden kann, ist allgemein bekannt, siehe [BEGO71]. Ein effektives Parsen verlangt Grammatiken, aus denen möglichst schnell ein XML Parser

konstruiert werden kann.
Eine präzisere Einschränkung der Inhaltsmodelle erfolgt durch die Einschränkung des k-Determinismus. Dieser hat zwei verschiedene Ausprägungen: Den k-lookahead Determinimus, welcher sich an die LL(1) Grammatiken im kontextfreien Bereich anlehnt und den k-block Determinismus [HW07]. In der Praxis sowie in dieser Arbeit wird nur der 1-lookahead Determinismus verwendet.

Definition 2.2.11 Es sei A ein Alphabet und r ein regulärer Ausdruck über A. σ ist eine Projektion auf das erste Zeichen eines Wortes $w \in A^*$. Es sei $w = u_1 \dots u_n$ mit $u_1, \dots, u_n \in A$, dann ist $\sigma(w) = u_1$. Es gilt $\sigma(\lambda) = \lambda$. Das Alphabet, das durch die Indizierung des regulären Ausdrucks r entsteht, wird mit A' bezeichnet.

Ein regulärer Ausdruck ist k-lookahead deterministisch, $k \in \mathbb{N}$, wenn gilt:

$$\forall u, v, w, x, y \in A'^* : (uxv \in L(\pi(r)) \wedge uyw \in L(\pi(r)) \wedge |x| = |y| = k) \\ \Longrightarrow ((\pi^{-1}(x) \neq \pi^{-1}(y)) \vee (\pi^{-1}(x) = \pi^{-1}(y) \wedge \sigma(x) = \sigma(y))).$$

□

Nicht jede reguläre Sprache kann durch einen k-lookahead deterministischen regulären Ausdruck beschrieben werden [HW07]. So ist die Sprache

$$L((a|b)^* a (a|b)^k), k \geq 1$$

nicht durch einen k-lookahead regulären Ausdruck erzeugbar.

2.3 Balancierte Grammatiken

Erweiterte kontextfreie Grammatiken sind Grammatiken, bei denen die rechte Seite der Produktionen aus einem regulären Ausdruck über Nichtterminalzeichen und Terminalzeichen besteht. Balancierte Grammatiken sind spezielle erweiterte kontextfreie Grammatiken. Die rechten Seiten der Produktionen bestehen aus einem Start- und einem Endtag und dazwischen befindet sich ein regulärer Ausdruck über Nichtterminalzeichen.
Die praktische Bedeutung dieser Grammatiken findet sich in den Beschreibungsmodellen von XML wieder. Wie in Kapitel 4 gezeigt wird, sind die balancierten Grammatiken die Obermenge der XML-artigen Grammatiken und wurden in [BB02b] definiert.
Ähnliche Grammatiken sind die Parenthesis Grammars, beschrieben von Knuth [Knu67] und McNaugthon [McN67]. Sie sind eine Untermenge der balancierten Grammatiken und besitzen eine endliche Menge von Produktionen und genau ein Klammerpaar.

Eine weitere Verbindung kann zu den regulären Baumgrammatiken [CDGJ97] gezogen werden. In dieser Sichtweise werden die Wörter nicht mehr als Klammerpaar gesehen, sondern in eine Baumstruktur aufgelöst. Hierbei gibt es Ähnlichkeiten zu regulären Baumgrammatiken. Lee, Mani und Murata schlagen diese Variante zum Parsen vor [LMM00] [MLM01b].

Aus formalsprachlicher Sicht sind die balancierten Grammatiken sehr interessant, denn sie besitzen sehr viele positive Abschlusseigenschaften und viele Entscheidbarkeitsfragen sind für sie entscheidbar.

Definition 2.3.12 Eine balancierte Grammatik ist ein 4-Tupel $G = (N, T, S, P)$, wobei gilt:

- N ist eine endliche Menge von Nichtterminalsymbolen.
- T ist eine endliche Menge von Terminalsymbolen mit

  $$T = A \cup \overline{A} \cup B \text{ und } \overline{A} = \{\overline{a} \mid a \in A\}.$$

  $A, \overline{A}$ und B sind disjunkt.
- S ist die Menge der Startsymbole.
- P ist die Menge von Produktionen der Form

  $$X \longrightarrow am\overline{a},$$

  mit $m \in Reg(N \cup B)$. Der Mittelteil m der Produktion wird als Inhaltsmodell bezeichnet. Für ein festes $X \in N$ und ein $a \in A$ gibt es genau eine Produktion

  $$X \longrightarrow am_{X,a}\overline{a}.$$

□

Das erste Terminalzeichen $a \in A$ auf der rechten Seite einer Produktion wird als Farbe der Produktion bezeichnet.
Wenn $B = \emptyset$ ist, dann ist die balancierte Grammatik „streng“ (engl. pure). Die entstehenden Wörter sind wohlgeformt und Dyckwörter. Wenn $B \neq \emptyset$, dann werden die entstehenden Wörter als Motzkinwörter [ST04] bezeichnet.

In dieser Arbeit werden ausschließlich „strenge“ balancierte Grammatiken behandelt. Der Zusatz „streng“ wird im Folgenden weggelassen.

Definition 2.3.13 Eine Sprache ist eine balancierte Sprache, wenn sie durch eine balancierte Grammatik erzeugt werden kann.

□

Es folgen ein paar Beispiele von balancierten Grammatiken und balancierten Sprachen:

Beispiel 2.3.5 Es sei $G = (N, T, \{S\}, P)$ eine balancierte Grammatik mit folgenden Produktionen:

$$\begin{array}{lcl} S & \longrightarrow & xY^*\overline{x} \\ Y & \longrightarrow & y\overline{y} \end{array}$$

Die Grammatik ist balanciert und erzeugt die Sprache $L = \{x(y\overline{y})^i\overline{x} \mid i \in \mathbb{N}_0\}$.

□

Beispiel 2.3.6 Es sei $G = (N, T, \{S\}, P)$ eine balancierte Grammatik mit folgenden Produktionen:

$$\begin{array}{lcl} S & \longrightarrow & xY\overline{x} \\ Y & \longrightarrow & y(S|A)\overline{y} \\ A & \longrightarrow & a\overline{a} \end{array}$$

Die Grammatik ist balanciert und erzeugt durch ihren selbsteinbettenden Teil eine kontextfreie Sprache $L = \{(xy)^n a\overline{a}(\overline{yx})^n \mid n \geq 1\}$.

□

Beispiel 2.3.7

$$L = \{x(y\overline{y})^n(z\overline{z})^n\overline{x} \mid n \geq 1\}$$

Behauptung: Die Sprache L ist nicht durch eine balancierte Grammatik erzeugbar.

Nehmen wir das Gegenteil an, dann muss es eine Ableitung

$$S \Longrightarrow xm\overline{x}, m \in Reg(N)$$

geben, aus welcher folgt

$$xm\overline{x} \Longrightarrow^* x(y\overline{y})^n(z\overline{z})^n\overline{x}.$$

Das Inhaltsmodell m hat die Gestalt

$$m = X_1X_2 \ldots X_nY_1Y_2 \ldots Y_n.$$

Jeder Dyckprime, ohne Beachtung der Operatoren, wird durch ein eigenes Nichtterminalzeichen erzeugt. Das Inhaltsmodell m ist somit X^nY^n. Es ist jedoch nicht durch einen regulären Ausdruck erzeugbar.

Beweis. Der Beweis folgt mittels Pumping Lemma. Die Pumping Lemma Konstante ist n. Wir wählen das Wort X^nY^n und die Länge $2n$. Aufgrund der 1. Bedingung des Pumping Lemmas darf v nicht leer sein, aufgrund der 2. Bedingung kann es nur aus Y bestehen.
Die 3. Bedingung lässt zu, dass auch die Wörter $X^{n-|v|}Y^n$ zulässig sind. Die Definition der Sprache erlaubt dies allerdings nicht. Der Ausdruck sowie das Inhaltsmodell sind somit nicht regulär. Die Sprache kann also nicht durch eine balancierte Grammatik erzeugt werden. □

Beispiel 2.3.8 Es sei $G = (N, T, S, P)$ eine kontextfreie Grammatik mit folgenden Produktionen:

$$\begin{array}{lcl} S & \longrightarrow & xY\overline{x} \\ Y & \longrightarrow & y\overline{y}(Y|\lambda)z\overline{z} \end{array}$$

Die durch G erzeugte Sprache ist nur eine balancierte Sprache, wenn $y = z$. Dann würde die Sprache durch die balancierte Grammatik

$$\begin{array}{lcl} S & \longrightarrow & x(YY)^*\overline{x} \\ Y & \longrightarrow & yy \end{array}$$

erzeugt werden.
Es würde andernfalls die Sprache $L = \{x(y\overline{y})^n(z\overline{z})^n\overline{x} \mid n \geq 1\}$ erzeugt werden, die durch obigen Beweis ausgeschlossen wurde.

□

Nun werden aus den balancierten Grammatiken, Grammatiken erzeugt, deren Produktionen indiziert sind.

Algorithmus:
Es sei $G = (N, T, S', P)$ eine balancierte Grammatik. Aus G entsteht folgendermaßen eine indizierte balancierte Grammatik G'. Die Konstruktion der indizierten balancierten Grammatik erfolgt über die Konstruktion eines Baumes.

1. Das Startsymbol S ist der Wurzelknoten des Baumes. Wenn eine Grammatik mehrere Startsymbole besitzt, dann wird für jedes Startsymbol ein eigener Baum konstruiert.

2. Die Nachfolger von S im Baum sind alle Produktionen von S. Die rechten Seiten der Produktionen werden indiziert. Jedes Nichtterminalzeichen X in jedem Inhaltsmodell der Produktionen wird mit einem Index versehen, so dass die Kombination aus Nichtterminalzeichen und Index in der indizierten balancierten Grammatik eindeutig ist. Der Index beginnt bei 1. Somit entsteht aus jedem Inhaltsmodell m ein indiziertes Inhaltsmodell m'.

3. Danach wird für jedes indizierte Nichtterminalzeichen X_i und der Produktion $X \longrightarrow xm\overline{x}$ eine Produktion erzeugt und die Nichtterminalzeichen im Inhaltsmodell wieder indiziert: $X_i \longrightarrow xm''\overline{x}$.

4. Nun gilt wieder, dass die Nachfolger eines indizierten Nichtterminalzeichens Y_i alle Produktionen von Y_i sind.

5. Die Schritte drei und vier werden solange wiederholt bis folgender Zustand eintritt: Es sei ψ eine Abbildung, die für ein indiziertes Nichtterminalzeichen Y_i die Indizierung löscht (Y). Falls für ein Nichtterminalzeichen Y_i ein Nachfolger konstruiert werden soll und ein Vorgänger Y_j im Baum $(Y_j \longrightarrow ym\overline{y})$ existiert, so dass $\psi(Y_i) = \psi(Y_j)$ gilt, dann wird eine Produktion $Y_i \longrightarrow ym\overline{y}$ konstruiert, für die keine weiteren Nachfolger konstruiert werden. Dies ist der Fall, wenn ein Nichtterminalzeichen selbsteinbettend auftritt.

6. Das Verfahren bricht ab, wenn keine Nachfolger mehr erzeugt werden müssen.

Die Produktionen der indizierten balancierten Grammatik werden aus den Knoten des Baums abgelesen. Daraus ergeben sich auch gleichzeitig die Nichtterminalzeichen, die Terminalzeichen und die Startsymbole.

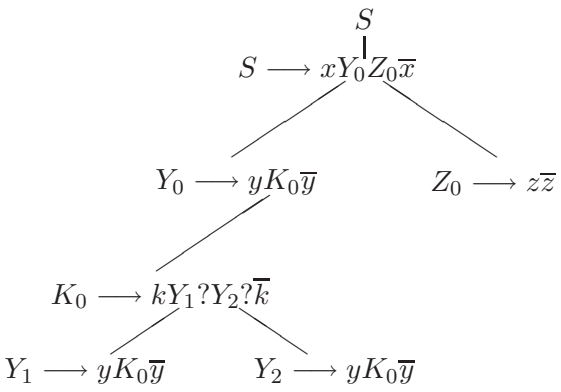
Definition 2.3.14
Eine indizierte balancierte Grammatik ist eine balancierte Grammatik, deren Produktionen durch den obigen Algorithmus indiziert wurden.

□

Beispiel 2.3.9 Gegeben sei die balancierte Grammatik $G = (N, T, \{S\}, P)$ mit den Produktionen:

$$\begin{array}{lcl} S & \longrightarrow & xYZ\overline{x} \\ Y & \longrightarrow & yK\overline{y} \\ K & \longrightarrow & kY?Y?\overline{k} \\ Z & \longrightarrow & z\overline{z} \end{array}$$

Für die Grammatik G wird folgende indizierte Grammatik G' konstruiert:



□

2.4 Untertypen der balancierten Grammatiken

Im Folgenden werden eingeschränkte balancierte Grammatiken definiert. Die Einschränkungen beziehen sich vor allem darauf, dass die entstehenden Grammatiken zu den LL(1) Grammatiken gehören, wie im vierten Kapitel gezeigt wird. Die balancierten Grammatiken können Mehrdeutigkeiten in ihren Inhaltsmodellen enthalten. So stellt sich zu allererst die Frage, ob sie zu den deterministisch kontextfreien Grammatiken gehören oder nichtdeterministisch kontextfrei sind. Falls sie deterministisch kontextfrei sind, stellt sich weiter die Frage, mit welchem Zeitaufwand ein deterministischer Kellerautomat aus einer Grammatik konstruiert werden kann.

In den Arbeiten [LMM00] und [MLM01b] wurden XML Baumgrammatiken definiert. Nun werden auf ähnliche Weise Wortgrammatiken definiert.

Definition 2.4.15 Es sei $G = (N, T, S, P)$ mit $T = A \cup \overline{A}$ eine balancierte Grammatik. Zwei Nichtterminale X und Y konkurrieren miteinander, wenn gilt:
$\exists X, Y \in N\ \exists m_0, m_1 \in Reg(N)\ \exists \overline{x} \in \overline{A}$:

$$\exists x \in A : (X \longrightarrow xm_0\overline{x} \in P \wedge Y \longrightarrow xm_1\overline{x} \in P \wedge X \neq Y)$$

□

Konkurrierende Nichtterminale erschweren das Parsen, da nicht mehr eindeutig anhand eines Terminals die Produktion hergeleitet werden kann. Somit müssen unter Umständen mehrere Produktionen auf ihre Gültigkeit hin überprüft werden, was zum Nichtdeterminismus und damit zu einer größeren Laufzeit führt.

Beispiel 2.4.10 Es sei $G = (N, T, \{S\}, P)$ eine balancierte Grammatik mit:

- $N = \{S, Y, Z\}$
- $T = \{x, \overline{x}, y, \overline{y}\}$
- $P = \{ \begin{array}{lcl} S & \longrightarrow & xY\overline{x}, \\ Y & \longrightarrow & yZ\overline{y}, \\ Z & \longrightarrow & y\overline{y}\} \end{array}$

Die Grammatik beinhaltet zwei Nichtterminale mit den gleichen Terminalen auf der rechten Seite der Produktion. Die Nichterminale Y und Z konkurrieren miteinander.

□

Definition 2.4.16 Eine XML Grammatik ist eine balancierte Grammatik ohne konkurrierende Nichtterminale und mit nur einem Startsymbol.
Für eine XML Grammatik $G = (N, T, S, P)$ mit $T = A \cup \overline{A}$ und dem Startsymbol S muss gelten:
Jede Produktion hat die Form:

$$X_a \longrightarrow am\overline{a}, \quad m \in Reg(N), \quad a \in A$$

und
$\forall X, Y \in N\ \forall m, m_0 \in Reg(N)\ \forall \overline{x} \in \overline{A}$:

$$\neg\exists x \in A : (X \longrightarrow xm\overline{x} \in P \wedge Y \longrightarrow xm_0\overline{x} \in P \wedge X \neq Y)$$

Zusätzlich besteht jedes Inhaltsmodell aus einem 1-lookahead deterministischen regulären Ausdruck.

□

Das bedeutet, es besteht eine bijektive Beziehung zwischen den Nichtterminalen N und den Terminalen A.

Definition 2.4.17 Eine balancierte Sprache ist eine XML Sprache, wenn sie durch eine XML Grammatik erzeugt werden kann.

□

Beispiel 2.4.11 Gegeben sei die Sprache

$$L = \{xyc\overline{cy}yd\overline{d}\overline{yx}\}.$$

Sie ist keine XML Sprache. Nehmen wir an, sie wäre eine XML Sprache, dann müsste eine XML Grammatik existieren, die L erzeugt.
Um L zu beschreiben wird eine balancierte Grammatik $G = (N, T, S, P)$ mit folgenden Produktionen benötigt:

$$P = \{ \begin{array}{lcl} S & \longrightarrow & xYZ\overline{x}, \\ Y & \longrightarrow & yC\overline{y}, \\ Z & \longrightarrow & yD\overline{y}, \\ C & \longrightarrow & c\overline{c}, \\ D & \longrightarrow & d\overline{d}\} \end{array}$$

Da die Sprache innerhalb der beiden $y \ldots \overline{y}$ Klammerpaare verschieden ist und deshalb zwei verschiedene Nichtterminalzeichen Y, Z mit den gleichen Terminalen auf der rechten Seite entstehen, gibt es zwei konkurrierende Nichtterminalzeichen.
Dies ist in einer XML Grammatik nicht erlaubt und die erzeugte Sprache stellt keine XML Sprache dar.

□

Definition 2.4.18 Eine einfache balancierte Grammatik ist eine balancierte Grammatik $G = (N, T, S, P)$ mit folgenden Eigenschaften:
$\forall X, Y, Z \in N \; \forall z, \overline{z}, \overline{x} \in T \; \forall m, m_0, m_1 \in Reg(N), \alpha, \beta, \gamma, \omega \in N^* :$

$$\neg \exists x \in A : (X \longrightarrow xm\overline{x} \in P \wedge Y \longrightarrow xm_0\overline{x} \in P \wedge X \neq Y \wedge Z \longrightarrow zm_1\overline{z} \in P \wedge \alpha X\beta \in L(m_1) \wedge \gamma Y\omega \in L(m_1))$$

S ist das einzige Startsymbol.
Zusätzlich besteht jedes Inhaltsmodell aus einem 1-lookahead deterministischen regulären Ausdruck.

□

Es existieren also keine Produktionen, deren Inhaltsmodelle konkurrierende Nichtterminale enthalten und es existiert nur ein Startsymbol.

Definition 2.4.19 Eine balancierte Sprache ist eine einfache balancierte Sprache, wenn sie durch eine einfache balancierte Grammatik erzeugt werden kann.

□

Beispiel 2.4.12 Es sei $G = (N, T, \{S\}, P)$ eine balancierte Grammatik mit:

- $N = \{S, Y, Z\}$
- $T = \{x, \overline{x}, y, \overline{y}\}$
- $P = \{ \begin{array}{lcl} S & \longrightarrow & xYZ\overline{x}, \\ Y & \longrightarrow & y\overline{y}, \\ Z & \longrightarrow & y\overline{y}\} \end{array}$

Die Grammatik beinhaltet zwei konkurrierende Nichtterminale Y und Z, welche in einem Inhaltsmodell enthalten sind. Dies ist in einer einfachen balancierten Grammatik nicht erlaubt.

□

Definition 2.4.20 Eine konkurrenzverhindernde balancierte Grammatik ist eine balancierte Grammatik $G = (N, T, S, P)$ mit folgenden Einschränkungen:
$\forall m, m_0, m_1 \in Reg(N) \; \forall X, Y, Z \in N \; \forall z, \overline{z}, \overline{x} \in T \; \forall U, V, W \in N^* :$

$$\neg\exists x \in A : (X \longrightarrow xm\overline{x} \in P \wedge Y \longrightarrow xm_0\overline{x} \in P \wedge X \neq Y \wedge$$
$$Z \longrightarrow zm_1\overline{z} \in P \wedge UXW \in L(m_1) \wedge UYV \in L(m_1))$$

S ist das einzige Startsymbol.
Zusätzlich besteht jedes Inhaltsmodell aus einem 1-lookahead deterministischen regulären Ausdruck.

□

Beispiel 2.4.13 Es sei $G = (N, T, S, P)$ eine balancierte Grammatik mit

- $N = \{X, Y, S\}$
- $T = \{x, \overline{x}, z, \overline{z}\}$
- $P = \{ \begin{array}{lcl} S & \longrightarrow & zX^*Y^*\overline{z} \\ X & \longrightarrow & x\overline{x} \\ Y & \longrightarrow & x\overline{x}\}. \end{array}$

Die Grammatik beinhaltet zwei Nichtterminale X und Y mit dem gleichen Terminal auf der rechten Seite der Produktion. Wir wählen U, V und W leer. Somit kann sowohl UXW, als auch UYV entstehen. Dies ist in einer konkurrenzverhindernden balancierten Grammatik nicht erlaubt.

□

Kapitel 3

Durchschnitt von regulären Sprachen und Dyckprimes

In der Praxis wurden die balancierten Grammatiken in verschiedene Untertypen unterteilt: XML Grammatiken, einfache balancierte Grammatiken und konkurrenzverhindernde balancierte Grammatiken. Die Einschränkungen werden, wie sich im nächsten Kapitel herausstellen wird, aus Gründen der besseren Parsebarkeit getroffen. Sie haben jedoch den Nachteil, dass bestimmte Sprachen nicht erzeugt werden können und einige Abschlusseigenschaften verloren gehen.
Es wird nun nach einem Untertyp gesucht, welcher einige Sprachen erzeugen kann, die die anderen Untertypen nicht erzeugen können und viele positive Abschlusseigenschaften besitzt.
Der Durchschnitt einer regulären Sprache mit der Menge D_T von Dyckprimes über einem Alphabet T, einer sogenannten RegDyck Sprache, scheint aus mehreren Gründen interessant zu sein. Er ist auf jeden Fall deterministisch kontextfrei und jede RegDyck Sprache ist auf einen regulären Ausdruck und eine Menge D_T von Dyckprimes zurückzuführen. Dabei wird ausgenutzt, dass der reguläre Ausdruck auch ein mehrdeutiger regulärer Ausdruck sein darf und für ihn ein NFA konstruiert werden kann, der in $O(n)$ Zeit parst.

3.1 Konstruktion von RegDyck Automaten

Der Durchschnitt einer regulären Sprache und der Menge D_T von Dyckprimes muss in den deterministisch kontextfreien Sprachen liegen, da der Durchschnitt einer regulären Sprache und einer deterministisch kontextfreien Sprache deterministisch kontextfrei ist.
Es wird daher überprüft, wie der Durchschnitt dieser beiden Sprachen aussieht und ob er möglicherweise ein Untertyp der balancierten Sprachen ist.
Zuerst wird ein RegDyck Automat aus einem beliebigen endlichen Automa-

ten und einem beliebigen Dyckprime Automaten konstruiert. Der RegDyck Automat hat die Besonderheit, dass seine Zustandsüberführungen zwar nichtdeterministisch sind, aber der Kellerinhalt immer eindeutig ist.

Es sei $\mathcal{R} = (S_R, A_R, s_{0R}, \delta_R, F_R)$ ein nichtdeterministischer endlicher Automat:

- $S_R = \{s_{0R}, s_1, s_2, \ldots, s_n\}$ ist die Menge der Zustände.
- $A_R = A \cup \overline{A} \cup B \cup \overline{B}$ ist das Eingabealphabet.
- s_{0R} ist der Startzustand.
- $F_R \subseteq S_R$ ist die Menge der Endzustände.
- δ_R ist die Überführungsfunktion.

Es sei $\mathcal{D} = (S_D, A_D, G_D, \triangle, s_{0D}, \delta_D, F_D)$ ein deterministischer Kellerautomat, der die Menge D_{A_D} von Dyckprimes erkennt:

- $S_D = \{s_{0D}, s_{1D}, s_{eD}\}$ ist die Menge der Zustände des Automaten.
- $A_D = A \cup \overline{A} \cup C \cup \overline{C}$ ist das Eingabealphabet.
- $G_D = \overline{A} \cup \overline{C} \cup \{\triangle\}$ ist das Kelleralphabet.
- s_{0D} ist der Startzustand.
- $F_D = \{s_{0D}, s_{eD}\}$ ist die Menge der Endzustände.
- δ_D ist die Überführungsfunktion:

$$\begin{array}{lcl}
(s_{0D}, a, \triangle) & \longrightarrow & (s_{1D}, \overline{a}\triangle)\ \forall a \in (A \cup C) \\
(s_{1D}, a, \overline{z}) & \longrightarrow & (s_{1D}, \overline{az})\ \forall a \in (A \cup C)\ \forall \overline{z} \in (\overline{A} \cup \overline{C}) \\
(s_{1D}, \lambda, \triangle) & \longrightarrow & (s_{eD}, \triangle) \\
(s_{1D}, \overline{a}, \overline{a}) & \longrightarrow & (s_{1D}, \lambda)\ \forall \overline{a} \in (\overline{A} \cup \overline{C}).
\end{array}$$

Nun wird ein nichtdeterministischer Kellerautomat $\mathcal{K} = (S_K, A_K, G_K, \triangle, s_{0K}, \delta_K)$ (der RegDyck Automat) konstruiert, der den Durchschnitt der Sprache des endlichen Automaten $\mathcal{R}$ und der Sprache des deterministischen Kellerautomaten $\mathcal{D}$ erkennt:

- $S_K = S_R \cup \{s_{eK}, s_{0K}, s_{1K}, s_E\}$ ist die Menge der Zustände.
- $A_K = A_R \cap A_D = A \cup \overline{A}$ ist das Eingabealphabet.
- $G_K = \overline{A} \cup \{\diamond, \triangle\}$ ist das Kelleralphabet.
- s_{0K} ist der Startzustand.
- Es wird eine Startüberführung zur Initialisierung benötigt:

$$(s_{0K}, \lambda, \triangle) \longrightarrow (s_{1K}, \diamond\triangle).$$

Der RegDyck Automat kellert ein Zeichen $\diamond$ ein. Danach arbeitet er wie der nichtdeterministische endliche Automat $\mathcal{R}$ und der deterministische Kellerautomat $\mathcal{D}$. Zum Schluss löscht er das Zeichen $\diamond$, wenn sowohl $\mathcal{R}$ als auch $\mathcal{D}$ akzeptieren.

Für jede Überführung in $\mathcal{R}$ der Form $(s_{0R}, a) \longrightarrow s_j$ mit $s_j \in S_R$ und $a \in A$ wird eine Überführung

$$(s_{1K}, a, \diamond) \longrightarrow (s_j, \overline{a}\diamond)$$

erzeugt. Dieser Zustand s_{1K} wird nur einmal benutzt und ist danach nicht mehr in dem Wertebereich der Überführungsfunktion enthalten. In allen von $\mathcal{K}$ übernommenen Zuständen des ursprünglichen Automaten $\mathcal{R}$ wird, wenn das Kellersymbol $\diamond$ an der obersten Stelle des Kellers steht, in einen Endzustand oder einen Fehlerzustand verwiesen. Das Zeichen $\diamond$ zeigt damit an, dass ein Dyckprime abgearbeitet worden ist.

Für jede Überführung in $\mathcal{R}$ der Form $(s_i, a) \longrightarrow s_j$ mit $s_i, s_j \in S_R$ und $a \in A$ werden Überführungen

$$(s_i, a, \overline{z}) \longrightarrow (s_j, \overline{a}\overline{z}) \; \forall \overline{z} \in \overline{A}, \overline{a} \in \overline{A}$$

erzeugt. Es wird festgelegt, dass für ein Eingabezeichen $a \in A$ der Automat $\mathcal{K}$ definiert ist, egal welches Zeichen $\overline{z} \in \overline{A}$ auf dem Keller liegt. Die einzige Bedingung ist, dass der endliche Automat $\mathcal{R}$ eine Zustandsüberführung im momentanen Zustand für das Eingabezeichen $a \in A$ besitzt.
Der Automat $\mathcal{D}$ ist für ein Zeichen $a \in A$ immer definiert, außer wenn er sich in dem Zustand s_{eD} oder s_E befindet. Er kellert das entsprechende Gegenstück aus der Menge $\overline{A}$ ein. Da der Automat $\mathcal{K}$ immer den gleichen Kellerinhalt wie $\mathcal{D}$ haben soll, muss auch $\mathcal{K}$ immer das entsprechende Gegenstück $\overline{a} \in \overline{A}$ zu $a \in A$ einkellern.

Für jede Überführung in $\mathcal{R}$ der Form $(s_i, \overline{a}) \longrightarrow s_j$ mit $s_i, s_j \in S_R$ und $\overline{a} \in \overline{A}$ wird eine Überführung

$$(s_i, \overline{a}, \overline{a}) \longrightarrow (s_j, \lambda)$$

erzeugt. Der Automat $\mathcal{K}$ ist somit für ein Zeichen $\overline{a} \in \overline{A}$ definiert, sofern der Automat $\mathcal{R}$ eine Überführung aus dem Zustand s_i für dieses Zeichen besitzt. Zusätzlich muss sich noch der Kellerinhalt verändern. Für jedes Eingabezeichen $\overline{a} \in \overline{A}$ löscht der Automat $\mathcal{D}$ das Zeichen $\overline{a}$

aus dem Keller, sofern $\overline{a}$ an der obersten Stelle im Keller liegt. Diese Überführung ist hiermit auch im Automat $\mathcal{K}$ konstruiert.

Für jede Überführung in $\mathcal{R}$ der Form $(s_i, \overline{a}) \longrightarrow s_j$ mit $s_i \in S_R, s_j \in S_R \setminus F_R$ und $\overline{a} \in \overline{A}$ wird eine Überführung

$$(s_j, \lambda, \diamond) \longrightarrow (s_E, \diamond)$$

erzeugt. Der Zustand s_E ist ein Fehlerzustand. Dieser Zustand wird erreicht, wenn ein Dyckprime abgearbeitet ist, jedoch kein Endzustand des endlichen Automaten erreicht ist. Das bedeutet, dass zwar der Automat $\mathcal{D}$ akzeptiert, nicht aber der Automat $\mathcal{R}$.

Es werden zusätzlich Überführungen für alle Endzustände $s_f \in F_R$ erzeugt:

$$(s_f, \lambda, \diamond) \longrightarrow (s_{eK}, \lambda).$$

Die Konstruktion beruht darauf, dass der Kellerautomat $\mathcal{D}$ und der Kellerautomat $\mathcal{K}$ zu jedem Zeitpunkt den gleichen Kellerinhalt haben. Weiter haben der Kellerautomat $\mathcal{K}$ und der endliche Automat $\mathcal{R}$ den gleichen Zustand nach dem Bearbeiten eines Eingabezeichens.
Der folgende Beweis teilt sich in die Betrachtung der Kellerinhalte von $\mathcal{K}$ und $\mathcal{D}$ und der Zustände von $\mathcal{K}$ und $\mathcal{R}$ auf.

Satz 3.1.1 Seien $\mathcal{R}$,$\mathcal{D}$ und $\mathcal{K}$ die Automaten aus der obigen Konstruktion, dann erkennt $\mathcal{K}$ den Durchschnitt von $\mathcal{R}$ und $\mathcal{D}$.

Beweis. Der erste Abschnitt des Beweises befasst sich mit den Kellern der Automaten $\mathcal{K}$ und $\mathcal{D}$.
Es sei $\mathcal{D}$ in der Konfiguration $(s_m, g, \triangle)$ und $\mathcal{K}$ in der Konfiguration $(s_i, g, \diamond\triangle)$. Dann folgt nach $|g|$ Schritten, dass $\mathcal{D}$ in der Konfiguration $(s_l, \lambda, \overline{v}\triangle)$ und $\mathcal{K}$ in der Konfiguration $(s_j, \lambda, \overline{v} \diamond \triangle)$ ist. Hierbei sind $s_m, s_l \in S_D, s_i \in S_K \setminus \{s_{0K}\}, s_j \in S_K \setminus \{s_{0K}, s_{1K}\}, g \in (A \cup \overline{A})^*$ und $\overline{v} \in \overline{A}^*$.

Der Beweis folgt per Induktion über die Länge eines Teilwortes $g'' \in (A \cup \overline{A})^*$ der Eingabe $g = g''g' \in (A \cup \overline{A})^*$ mit $g' \in (A \cup \overline{A})^*$.
Sei $g'' = \lambda$. Dann existiert für den Kellerautomaten $\mathcal{D}$ keine Überführung für $s_m \in S_D \setminus \{s_{1D}\}$. $\mathcal{D}$ bleibt also in der ursprünglichen Konfiguration $(s_m, g', \triangle)$. Ist $s_m = s_{1D}$, so wird die Überführung

$$(s_{1D}, \lambda, \triangle) \longrightarrow (s_{eD}, \triangle)$$

angewendet. $\mathcal{D}$ befindet sich dann in der Konfiguration $(s_{eD}, g', \triangle)$.
Der Kellerautomat $\mathcal{K}$ verwendet für $s_i \in S_K \setminus \{s_{0K}, s_{1K}, s_E, s_{eK}\}$ entweder die Überführung

$$(s_i, \lambda, \diamond) \longrightarrow (s_E, \diamond)$$

oder die Überführung

$$(s_i, \lambda, \diamond) \longrightarrow (s_{eK}, \lambda).$$

$\mathcal{K}$ befindet sich dann in der Konfiguration $(s_E, g', \diamond\triangle)$ oder $(s_{eK}, g', \triangle)$.
Der Fall $s_i = s_{0K}$ kann nach Konstruktion nicht auftreten. In den Fällen $s_i \in \{s_{1K}, s_E, s_{eK}\}$ ist der Automat für die Eingabe λ nicht definiert und bleibt in der Konfiguration $(s_i, g', \diamond\triangle)$.
Also steht in den Kellern der Automaten $\mathcal{K}$ und $\mathcal{D}$ (evtl. bis auf das Zeichen $\diamond$ im Keller von $\mathcal{K}$) dasselbe Wort.

Sei $g'' = a$. Dann verwendet $\mathcal{D}$ für $s_m = s_{0D}$ die Überführung

$$(s_{0D}, a, \triangle) \longrightarrow (s_{1D}, \overline{a}\triangle).$$

$\mathcal{D}$ befindet sich nun in der Konfiguration $(s_{1D}, g', \overline{a}\triangle)$.
Ist $s_m = s_{1D}$, so ist der Automat $\mathcal{D}$ nicht definiert und bleibt in der Konfiguration $(s_{1D}, a, \triangle)$.
Ist der Automat $\mathcal{K}$ in der Konfiguration $(s_{1K}, ag', \diamond\triangle)$, so verwendet er die Überführung

$$(s_{1K}, a, \diamond) \longrightarrow (s_w, \overline{a}\diamond)$$

und befindet sich anschließend in der Konfiguration $(s_w, g', \overline{a} \diamond \triangle)$, sofern $\mathcal{R}$ auch für a definiert ist.
Ist $s_i \neq s_{1K}$, so ist der Automat $\mathcal{K}$ nicht definiert und bleibt in der Konfiguration $(s_i, ag', \diamond\triangle)$.
Somit haben beide Automaten denselben Kellerinhalt (mit Ausnahme des Zeichens $\diamond$ im Keller von $\mathcal{K}$), da nur zwei Fälle eintreten können: Entweder ist $\mathcal{D}$ im Zustand s_{0D} und $\mathcal{K}$ im Zustand s_{1K} und beide Automaten sind definiert oder die Automaten befinden sich nicht in ihren Anfangszuständen und sind beide nicht definiert.

Sei $g'' = \overline{a}$. Dann ist für den Automaten $\mathcal{D}$ in keinem Zustand eine Überführung mit leerem Keller für diese Eingabe definiert. Ebenso ist für den Automaten $\mathcal{K}$ in keinem Zustand eine Überführung mit leerem Keller für eine solche Eingabe definiert. Folglich bleiben beide Automaten in ihrer Konfiguration und haben somit denselben Kellerinhalt.

Die Behauptung gelte nun für $|g''| \leq n \in \mathbb{N}_0$.

Es sei ag' oder $\overline{a}g'$ der Rest eines Eingabewortes $g = g''g'$ mit $g'' = wa$ bzw. $g'' = w\overline{a}, |w| = n, a \in A, \overline{a} \in \overline{A}$.

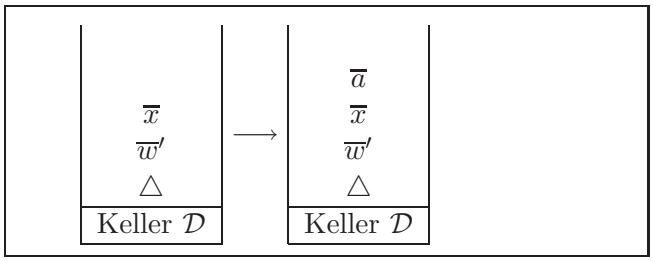


Abbildung 3.1: Kellerinhalt des Kellerautomaten $\mathcal{D}$ nach Abarbeitung des Eingabezeichens a.

Im Folgenden sei $\overline{w}' \in \overline{A}^*, \overline{x} \in \overline{A}$ mit $\overline{w} = \overline{xw}'$, wobei $\overline{w}$ das nach Abarbeitung von w in den Kellern stehende Wort ist.

1. Fall: $g'' = wa$
Nach Induktionsvoraussetzung und Definition ist der Kellerautomat $\mathcal{D}$ nach Abarbeitung von w in der Konfiguration $(s_{1D}, ag', \overline{xw}'\triangle)$ und der Automat $\mathcal{K}$ in der Konfiguration $(s_j, ag', \overline{xw}' \diamond \triangle)$, wobei $s_j \in S_K \setminus \{s_{0K}, s_{1K}\}$ (nach Konstruktion). Der Kellerautomat $\mathcal{D}$ benutzt die Überführung

$$(s_{1D}, a, \overline{x}) \longrightarrow (s_{1D}, \overline{ax})$$

und wechselt in die Konfiguration

$$(s_{1D}, g', \overline{axw}'\triangle). \text{ (Abbildung 3.1)}$$

Der Kellerautomat $\mathcal{K}$ benutzt für $s_j \in S_K \setminus \{s_{0K}, s_{1K}\}$ die Überführung

$$(s_j, a, \overline{x}) \longrightarrow (s_k, \overline{ax})$$

und befindet sich danach in der Konfiguration

$$(s_k, g', \overline{axw}' \diamond \triangle). \text{ (Abbildung 3.2)}$$

Die Fälle $s_j \in \{s_{0K}, s_{1K}\}$ können nach Konstruktion des Automaten nicht auftreten.
Somit haben die beiden Automaten immer denselben Kellerinhalt (bis auf das Zeichen $\diamond$ im Keller von $\mathcal{K}$). Falls $\mathcal{R}$ für eine Eingabe nicht definiert ist, dann gehört die Sprache aber auch nicht zum Durchschnitt der Sprachen von $\mathcal{R}$ und $\mathcal{D}$.
Sind die Keller leer, so sind beide Kellerautomaten nicht definiert, da sie nach Konstruktion nicht in ihre Anfangszustände s_{0D} und s_{0K} bzw. s_{1K} zurückkehren können. Nur für diese Zustände sind Überführungen mit leerem Keller und nicht leerer Eingabe definiert. Beide Automaten bleiben in der ursprünglichen Konfiguration und haben somit denselben Kellerinhalt.

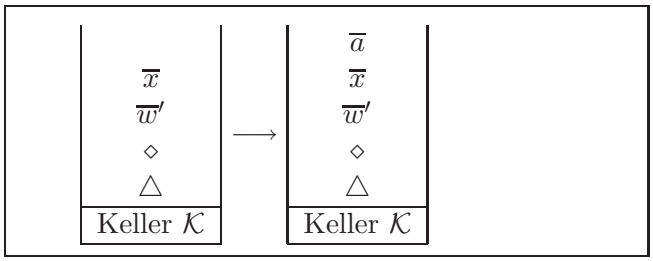


Abbildung 3.2: Kellerinhalt des Kellerautomaten $\mathcal{K}$ nach Abarbeitung des Eingabezeichens a.

2. Fall: $g'' = w\overline{a}$
Nach Induktionsvoraussetzung und Definition von $\mathcal{D}$ wird nach Abarbeitung von w ein Eingabezeichen $\overline{a}$ eingelesen und im Kellerautomaten $\mathcal{D}$ besteht eine Konfiguration $(s_{1D}, \overline{a}g', \overline{xw}'\triangle)$. Falls $\overline{x} = \overline{a}$, so verwendet der Kellerautomat $\mathcal{D}$ folgende Überführung:

$$(s_{1D}, \overline{a}, \overline{a}) \longrightarrow (s_{1D}, \lambda)$$

Es folgt die Konfiguration

$$(s_{1D}, g', \overline{w}'\triangle). \text{ (Abbildung 3.3)}$$

Ist $\overline{x} \neq \overline{a}$, so ist $\mathcal{D}$ nicht definiert und bleibt in der Konfiguration

$$(s_{1D}, \overline{a}g', \overline{xw}'\triangle).$$

Der Kellerautomat $\mathcal{K}$ ist nach Induktionsvoraussetzung und Definition nach Abarbeitung von w in der Konfiguration $(s_j, \overline{a}g', \overline{aw}' \diamond \triangle)$ mit $s_j \in S_K \setminus \{s_{0K}, s_{1K}\}$. Falls $\overline{x} = \overline{a}$, benutzt er die Überführung, wenn $\mathcal{R}$ auch dafür definiert ist

$$(s_j, \overline{a}, \overline{a}) \longrightarrow (s_k, \lambda)$$

und ist dann in der Konfiguration

$$(s_k, g', \overline{w}' \diamond \triangle). \text{ (Abbildung 3.4)}$$

Ist $\overline{x} \neq \overline{a}$, so ist $\mathcal{K}$ nicht definiert und bleibt in der Konfiguration

$$(s_j, \overline{a}g', \overline{xw}' \diamond \triangle).$$

In beiden Fällen steht im Keller von $\mathcal{K}$ dasselbe wie im Keller von $\mathcal{D}$ (bis auf das Zeichen $\diamond$ im Keller von $\mathcal{K}$).
Die Fälle $s_j \in \{s_{0K}, s_{1K}\}$ können nach Konstruktion des Automaten $\mathcal{K}$ nicht auftreten.
Sind die Keller leer, so sind beide Kellerautomaten nicht definiert. Sie bleiben in der ursprünglichen Konfiguration und haben somit denselben Kellerinhalt.

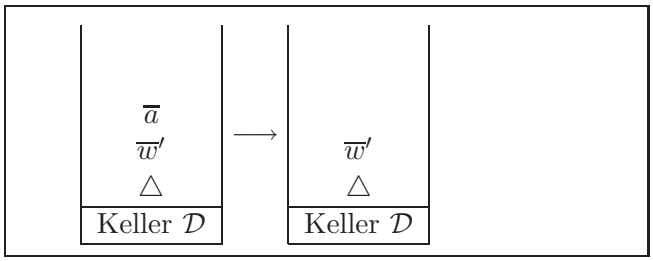


Abbildung 3.3: Kellerinhalt des Kellerautomaten $\mathcal{D}$ nach Abarbeitung des Eingabezeichens $\overline{a}$.

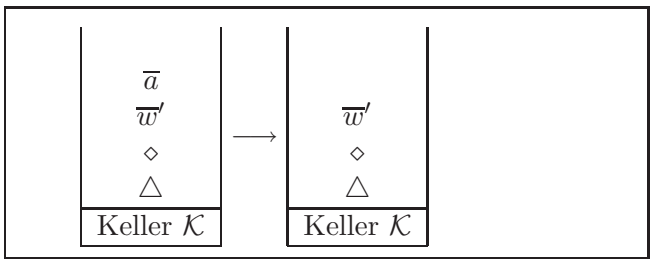


Abbildung 3.4: Kellerinhalt des Kellerautomaten $\mathcal{K}$ nach Abarbeitung des Eingabezeichens $\overline{a}$.

Befindet sich der Automat $\mathcal{K}$ in der Konfiguration $(s_i, ag', \overline{w} \diamond \triangle)$, dann wechselt er in die Konfiguration $(s_j, g', \overline{aw} \diamond \triangle)$. Der Automat $\mathcal{D}$ wechselt von der Konfiguration $(s_{1D}, ag', \overline{w}\triangle)$ in $(s_{1D}, g', \overline{aw}\triangle)$.
Ist $\mathcal{K}$ in der Konfiguration $(s_i, \overline{a}g', \overline{b}\overline{w}' \diamond \triangle)$ und $\mathcal{D}$ in der Konfiguration $(s_{1D}, \overline{a}g', \overline{b}\overline{w}'\triangle)$ mit $\overline{b} \in \overline{A}, \overline{a} \neq \overline{b}$, so gibt es für beide Automaten keine Überführung die angewendet werden kann. Somit sind beide Automaten in diesem Fall nicht definiert. Für $\overline{b} = \overline{a}$ wechselt $\mathcal{D}$ in die Konfiguration $(s_{1D}, g', \overline{w}\triangle)$ und $\mathcal{K}$ wechselt in die Konfiguration $(s_j, g', \overline{w} \diamond \triangle)$.

Der zweite Teil des Beweises folgt:
Es wird überprüft, ob sich der endliche Automat $\mathcal{R}$ und der RegDyck Automat $\mathcal{K}$ immer im gleichen Zustand befinden.
Es sei $\mathcal{R}$ in der Konfiguration (s_t, g) und $\mathcal{K}$ in der Konfiguration $(s_j, g, \diamond\triangle)$, $s_t \in S_R, s_j \in S_K \setminus \{s_{0K}\}$. Dann folgt nach $|g|$ Schritten, dass $\mathcal{R}$ in der Konfiguration $(S_i, \lambda), S_i \subseteq S_R$, und $\mathcal{K}$ in der Konfiguration $(S_i, \lambda, \overline{v} \diamond \triangle)$ ist, da $\mathcal{R}$ nichtdeterministisch sein kann. Hierbei ist $\overline{v} \in \overline{A}^*$.
Falls $\mathcal{K}$ in der Konfiguration $(s_{1K}, g, \diamond\triangle)$ ist, so ist $\mathcal{R}$ in der Konfiguration (s_{0R}, g). Da für den Zustand s_{1K} dieselben Überführungen konstruiert werden, wie sie für s_{0R} vorliegen, werden die beiden Zustände in dieselbe

Folgezustandsmenge $S_i \subseteq S_R$ überführt. Im Fall $s_j \neq s_{1K}$ gilt $s_j = s_t$ oder $s_j \in \{s_E, s_{eK}\}$.

Der Beweis folgt per Induktion über die Länge eines Teilwortes $g'' \in (A \cup \overline{A})^*$ der Eingabe $g = g''g'$ mit $g' \in (A \cup \overline{A})^*$.
Sei $g'' = \lambda$. Der Automat $\mathcal{R}$ ist für diese Eingabe nicht definiert und bleibt in der Konfiguration

$$(s_t, g').$$

Der Kellerautomat $\mathcal{K}$ verwendet für $s_j \notin \{s_{0K}, s_{1K}, s_E, s_{eK}\}$ entweder die Überführung

$$(s_j, \lambda, \diamond) \longrightarrow (s_E, \diamond)$$

oder die Überführung

$$(s_j, \lambda, \diamond) \longrightarrow (s_{eK}, \lambda).$$

$\mathcal{K}$ befindet sich dann entweder in der Konfiguration $(s_E, \lambda, \diamond)$ oder in der Konfiguration $(s_{eK}, \lambda, \triangle)$. $\mathcal{R}$ und $\mathcal{K}$ befinden sich nun in unterschiedlichen Zuständen. Dies ändert allerdings nichts daran, dass $\mathcal{K}$ den Durchschnitt der Sprachen der Automaten $\mathcal{R}$ und $\mathcal{D}$ erkennt, da dieser Fall nur auftritt, wenn entweder die Eingabe gleich λ ist oder die Eingabe bereits abgearbeitet wurde. In beiden Fällen bleibt für $\mathcal{R}$ nichts zu tun, allerdings muss $\mathcal{K}$ noch überprüfen, ob es sich bei der Eingabe um einen Dyckprime handelt.
Die Fälle $s_j \in \{s_{eK}, s_E, s_{0K}\}$ können nach vorheriger Überlegung nicht eintreten. Der Fall $s_j = s_{1K}$ wurde oben bereits betrachtet.

Die Behauptung gelte nun für $|g''| \leq n \in \mathbb{N}_0$.

$\mathcal{R}$ befindet sich nach der Abarbeitung eines Wortes w, wobei $g = g''g'$ mit $g'' = wa$ oder $g'' = w\overline{a}, |w| = n$, in einer Zustandsmenge $S_i \subseteq S_R$. Ebenfalls befindet sich der Kellerautomat $\mathcal{K}$ in der Zustandsmenge S_i. Die restliche Eingabe besteht aus einem Wort ag' oder $\overline{a}g'$.

Im Folgenden sei $\overline{w}' \in \overline{A}^*, \overline{x} \in \overline{A}$ mit $\overline{w} = \overline{x}\overline{w}'$, wobei $\overline{w}$ das nach Abarbeitung von w im Keller von $\mathcal{K}$ stehende Wort ist.

1. Fall: $g'' = wa$
Die Konstruktion beschreibt für jede Überführung

$$(s_m, a) \longrightarrow s_k$$

in $\mathcal{R}$ mit $s_m \in S_i, s_k \in S_R$ eine Überführung

$$(s_m, a, \overline{x}) \longrightarrow (s_k, \overline{ax})$$

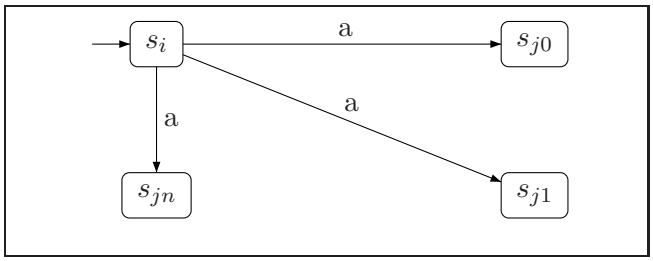


Abbildung 3.5: Automat $\mathcal{R}$ im Zustand s_i mit der Eingabe a.

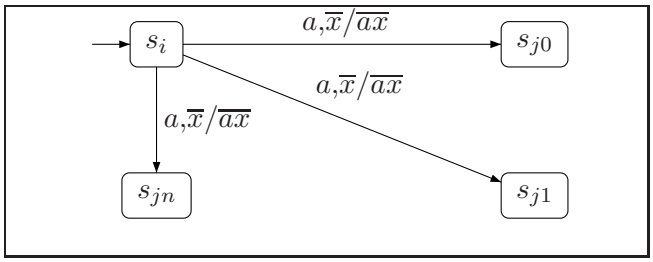


Abbildung 3.6: Automat $\mathcal{K}$ im Zustand s_i mit der Eingabe a. Die Beschriftung einer Kante mit $a,\overline{x}/\overline{ax}$ bedeutet, dass, wenn der Automat ein Zeichen a einliest und das Zeichen $\overline{x}$ auf dem Keller liegt, dann wechselt er in den Zustand zu dem die beschriftete Kante führt und legt $\overline{ax}$ auf dem Keller ab.

in $\mathcal{K}$, welche in den gleichen Zustand überführt. Da dies für alle Kellersymbole $\overline{x} \in G_K$ gilt, wird die Überführung unabhängig vom obersten Kellerzeichen ausgeführt. Gibt es in $\mathcal{R}$ mehrere Überführungen der Form $(s_m, a) \longrightarrow s_k$, so gibt es auch mehrere Überführungen in $\mathcal{K}$. Da $\mathcal{R}$ nichtdeterministisch sein kann, befinden sich sowohl $\mathcal{R}$ als auch $\mathcal{K}$ in einer Zustandsmenge S_j(Abbildung 3.5 und 3.6).

2. Fall: $g'' = w\overline{a}$
Für eine Überführung in $\mathcal{R}$ der Form

$$(s_m, \overline{a}) \longrightarrow s_k$$

mit $s_m \in S_i, s_k \in S_R$ hat der Automat $\mathcal{K}$ die Überführung

$$(s_m, \overline{a}, \overline{a}) \longrightarrow (s_k, \lambda).$$

Hierbei ist es für den Automaten $\mathcal{K}$ nicht gleichgültig, welches Zeichen auf der obersten Position im Keller liegt. Es muss das passende Zeichen $\overline{a}$ sein. Im Fall, dass es nicht das passende Zeichen ist, wäre der Automat $\mathcal{K}$ in einem anderen Zustand. Er würde nicht akzeptieren, genauso wie der Automat $\mathcal{D}$ nicht akzeptieren würde. Somit ist die erkannte Sprache wieder gleich. Ist $\overline{a}$ an der obersten Stelle des Kellers, befinden sich beide Automaten $\mathcal{K}$ und $\mathcal{R}$ in der Zustandsmenge S_j.

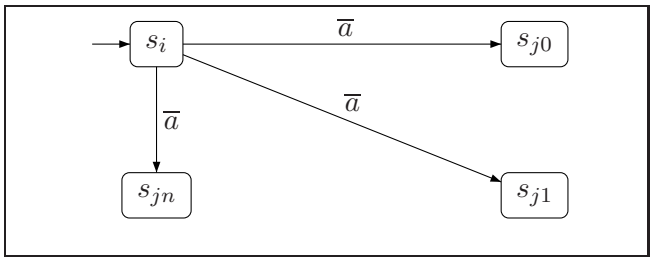


Abbildung 3.7: Automat $\mathcal{R}$ im Zustand s_i mit der Eingabe $\overline{a}$.

Es sei $(s_0, ag', \overline{w} \diamond \triangle)$ die Konfiguration von $\mathcal{K}$ und (s_0, ag') die Konfiguration von $\mathcal{R}$. Beide Automaten werden in die Zustandsmenge S_j überführt, da in $\mathcal{K}$ entsprechende Überführungen erzeugt worden sind. Der Kellerinhalt ist dabei beliebig.
Befinden sich die Automaten in den Konfigurationen $(s_0, \overline{a}g', \overline{b}\overline{w}' \diamond \triangle)$ und $(s_0, \overline{a}g')$, verweist $\mathcal{R}$ möglicherweise in einen Zustand und $\mathcal{K}$ akzeptiert die Eingabe nicht, falls $\overline{a} \neq \overline{b}$. $\mathcal{D}$ akzeptiert jedoch dann auch nicht, da $\mathcal{K}$ und $\mathcal{D}$ den gleichen Kellerinhalt haben.

Der Automat $\mathcal{K}$ akzeptiert eine Eingabe genau dann, wenn sein Keller leer ist, er also in der Konfiguration $(s_{eK}, \lambda, \triangle)$ ist. Diese Konfiguration erreicht der Automat genau dann, wenn sowohl $\mathcal{D}$ als auch $\mathcal{R}$ die Eingabe akzeptieren, also der Kellerautomat $\mathcal{D}$ einen leeren Keller hat und der endliche Automat $\mathcal{R}$ sich in einem Endzustand befindet.
Befindet sich der Automat $\mathcal{R}$ in einem Endzustand, aber der Keller von $\mathcal{K}$ ist (bis auf das Zeichen $\diamond$) nicht leer, so ist auch der Keller von $\mathcal{D}$ nicht leer. Folglich wird weder von $\mathcal{D}$ noch von $\mathcal{K}$ die Eingabe akzeptiert. Die Eingabe liegt damit auch nicht im Durchschnitt.

Ist der Keller von $\mathcal{D}$ leer, so ist der Keller von $\mathcal{K}$ bis auf das Zeichen $\diamond$ geleert, da sie immer denselben Kellerinhalt haben. Akzeptiert der Automat $\mathcal{R}$ die Eingabe allerdings nicht, so befindet sich $\mathcal{R}$ nicht in einem Endzustand, wenn er die Eingabe abgearbeitet hat. Hier existiert in $\mathcal{K}$ keine Überführung, die das Kellersymbol $\diamond$ im Keller löscht. Somit akzeptiert $\mathcal{K}$ die Eingabe ebenfalls nicht.

□

Nun wird noch die Gegenrichtung gezeigt. Es wird aus einem RegDyck Automaten $\mathcal{K}$ ein Kellerautomat $\mathcal{D}$ und ein endlicher Automat $\mathcal{R}$ konstruiert. Der Durchschnitt ihrer Sprachen soll gleich der Sprache von $\mathcal{K}$ sein. Dabei erkennt $\mathcal{D}$ eine Menge D_T von Dyckprimes über einem Alphabet T.

Es sei $\mathcal{K} = (S_K, A_K, G_K, \triangle, s_{0K}, \delta_K)$ ein RegDyck Automat mit:

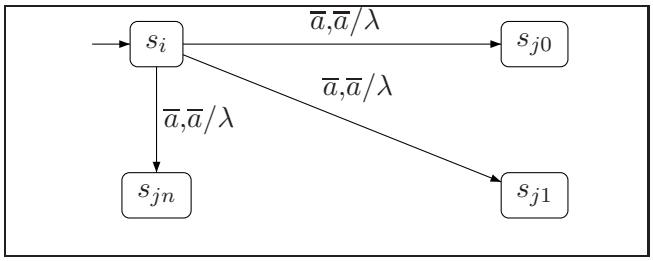


Abbildung 3.8: Automat $\mathcal{K}$ im Zustand s_i mit der Eingabe $\overline{a}$.

- $S_K = \{s_{0K}, s_{1K}, s_E, s_{eK}, s_1, \ldots, s_n\}$ ist die Menge der Zustände.
- $A_K = A \cup \overline{A}$ ist die Menge der Eingabesymbole.
- $G_K = \overline{A} \cup \{\triangle, \diamond\}$ ist die Menge der Kellerzeichen.
- s_{0K} ist der Startzustand.
- δ_K ist die Überführungsfunktion.

Nun werden die beiden Automaten $\mathcal{R}$ und $\mathcal{D}$ konstruiert, deren Durchschnitt ihrer Sprachen gleich der Sprache des RegDyck Automaten $\mathcal{K}$ ist.
Der nichtdeterministische endliche Automat $\mathcal{R} = (S_R, A_R, s_{1K}, \delta_R, F_R)$ wird folgendermaßen festgelegt:

- $S_R = S_K \setminus \{s_E, s_{0K}, s_{eK}\}$ ist die Menge der Zustände.
- $A_R = A \cup \overline{A}$ ist das Eingabealphabet.
- s_{1K} ist der Startzustand.
- $F_R = \{s_f \in S_K \mid \exists\ \delta_K(s_f, \lambda, \diamond) = (s_{eK}, \lambda)\}$ ist die Menge der Endzustände.
- δ_R:
  Für jede Überführung in δ_K der Form

  $$(s_i, a, \overline{z}) \longrightarrow (s_j, \overline{a}\overline{z}),\ a \in A, \overline{a}, \overline{z} \in \overline{A}, s_i, s_j \in S_K \setminus \{s_{0K}, s_{1K}, s_E, s_{eK}\},$$

  wird eine Überführung

  $$(s_i, a) \longrightarrow s_j$$

  erzeugt.
  Für jede Überführung der Form

  $$(s_i, \overline{a}, \overline{a}) \longrightarrow (s_j, \lambda),\ \overline{a} \in \overline{A}, s_i, s_j \in S_K \setminus \{s_{0K}, s_{1K}, s_E, s_{eK}\},$$

wird eine Überführung

$$(s_i, \overline{a}) \longrightarrow s_j$$

erzeugt.
Es wird außerdem für alle Überführungen der Form

$$(s_{1K}, a, \diamond) \longrightarrow (s_j, \overline{a}\diamond),\ a \in A, \overline{a} \in \overline{A}, s_j \in S_K \setminus \{s_{0K}, s_{1K}, s_E, s_{eK}\},$$

eine Überführung

$$(s_{1K}, a) \longrightarrow s_j$$

erzeugt.

Der deterministische Kellerautomat $\mathcal{D} = (S_D, A_D, G_D, \triangle, s_{0D}, \delta_D, F_D)$, der die Menge $D_{A\cup\overline{A}}$ von Dyckprimes erkennt, ist folgendermaßen definiert:

- $S_D = \{s_{0D}, s_{1D}, s_{eD}\}$ ist die Menge der Zustände.
- $A_D = A \cup \overline{A}$ ist das Eingabealphabet.
- $G_D = \overline{A} \cup \{\triangle\}$ ist das Kelleralphabet.
- s_{0D} ist der Startzustand.
- $F_D = \{s_{0D}, s_{eD}\}$ ist die Menge der Endzustände.
- δ_D ist definiert durch:

$$\begin{array}{lcl}
(s_{0D}, a, \triangle) & \longrightarrow & (s_{1D}, \overline{a}\triangle)\ \forall a \in A \\
(s_{1D}, a, \overline{z}) & \longrightarrow & (s_{1D}, \overline{a}\overline{z})\ \forall a \in A\ \forall \overline{z} \in \overline{A} \\
(s_{1D}, \lambda, \triangle) & \longrightarrow & (s_{eD}, \triangle) \\
(s_{1D}, \overline{a}, \overline{a}) & \longrightarrow & (s_{1D}, \lambda)\ \forall \overline{a} \in \overline{A}.
\end{array}$$

Satz 3.1.2 Es sei $\mathcal{K}$ der oben beschriebene RegDyck Automat über dem Alphabet $A \cup \overline{A}$, dann existiert ein Kellerautomat $\mathcal{D}$, welcher eine Dycksprache über dem Alphabet $A \cup \overline{A}$ erkennt und ein endlicher Automat $\mathcal{R}$, deren Durchschnitt ihrer Sprachen gleich der Sprache von $\mathcal{K}$ ist.

Beweis. Die Idee des Beweises ist, dass der Kellerinhalt des Kellerautomaten $\mathcal{K}$ und des Kellerautomaten $\mathcal{D}$ zu jeder Zeit gleich ist. Weiterhin befinden sich der Kellerautomat $\mathcal{K}$ und der endliche Automat $\mathcal{R}$ im gleichen Zustand.

Es sei $g \in (A \cup \overline{A})^*$ die Eingabe.
Es sei $\mathcal{D}$ in der Konfiguration $(s_0, g, \triangle)$ und $\mathcal{K}$ in der Konfiguration $(s_k, g, \diamond\triangle)$, wobei $s_0 \in S_D, s_k \in S_K \setminus \{s_{0K}\}$. Dann folgt nach $|g|$ Schritten, dass $\mathcal{D}$ in der Konfiguration $(s_d, \lambda, \overline{v}\triangle)$ und $\mathcal{K}$ in der Konfiguration $(s_i, \lambda, \overline{v} \diamond \triangle)$ ist,

$s_d \in S_D \setminus \{s_{0D}\}, s_i \in S_K \setminus \{s_{0K}, s_{1K}\}, \overline{v} \in \overline{A}^*$.

Der Beweis folgt per Induktion über die Länge eines Teilwortes $g'' \in (A \cup \overline{A})^*$ der Eingabe $g = g''g'$ mit $g' \in (A \cup \overline{A})^*$.
Die Behauptung ist trivial für $|g''| = 0$, da in diesem Fall $g'' = \lambda$ und $\mathcal{D}, \mathcal{K}$ in der angegebenen Konfiguration bleiben bzw. in $\mathcal{K}$ die Überführung

$$(s_k, \lambda, \diamond) \longrightarrow (s_E, \diamond)$$

oder die Überführung

$$(s_k, \lambda, \diamond) \longrightarrow (s_{eK}, \lambda)$$

verwendet wird. Dabei ändert sich allerdings der Kellerinhalt nicht (bis auf das Zeichen $\diamond$, was nicht relevant ist). Somit haben $\mathcal{D}$ und $\mathcal{K}$ für die Eingabe λ den gleichen Kellerinhalt.

Die Behauptung gelte nun für $|g''| \leq n \in \mathbb{N}_0$.

Es sei ag' oder $\overline{a}g'$ der Rest eines Eingabewortes $g''g'$ mit $g'' = wa$ bzw. $g'' = w\overline{a}, |w| = n$.

1. Fall: $g'' = wa$
Nach Induktionsvoraussetzung und Definition ist der Automat $\mathcal{K}$ nach Abarbeitung von w in der Konfiguration $(s_i, ag', \overline{xw}' \diamond \triangle)$ mit $s_i \in S_K \setminus \{s_{0K}, s_{1K}\}$.
Nach Voraussetzung hat der Kellerautomat $\mathcal{D}$ nach Abarbeitung von w den gleichen Kellerinhalt $\overline{xw}'$ und den gleichen Rest ag' des Eingabewortes wie $\mathcal{K}$. Er ist nach Definition in der Konfiguration $(s_{1D}, ag', \overline{xw}' \triangle)$.
Der Automat $\mathcal{K}$ benutzt für $s_i \notin \{s_E, s_{eK}\}$ die Überführung

$$(s_i, a, \overline{x}) \longrightarrow (s_j, \overline{ax}) \ \forall \overline{x} \in G_K$$

und wechselt in die Konfiguration $(s_j, g', \overline{axw}' \diamond \triangle)$.
Der Fall $s_i \in \{s_E, s_{eK}\}$ kann nicht auftreten, da die Eingabe bisher ungleich λ war und der Keller nicht leer ist. Nur in diesem Fall wechselt der Automat in einen der Zustände s_E oder s_{eK}.
Der Kellerautomat $\mathcal{D}$ benutzt die Überführung

$$(s_{1D}, a, \overline{x}) \longrightarrow (s_{1D}, \overline{ax}) \ \forall \overline{x} \in G_D$$

und wechselt in den Zustand $(s_{1D}, g', \overline{axw}' \triangle)$.
Somit haben $\mathcal{D}$ und $\mathcal{K}$ denselben Kellerinhalt nach Abarbeitung von g''.

2. Fall: $g'' = w\overline{a}$
Nach Induktionsvoraussetzung und Definition befindet sich der Automat $\mathcal{K}$ nach Abarbeitung von w in einer Konfiguration $(s_i, \overline{a}g', \overline{xw}' \diamond \triangle)$ mit $s_i \in S_K \setminus \{s_{0K}, s_{1K}\}$. Der Kellerautomat $\mathcal{D}$ befindet sich nach Induktionsvoraussetzung und Definition nach Abarbeitung von w in der Konfiguration

$(s_{1D}, \overline{a}g', \overline{xw}'\triangle)$.
Der Kellerautomat $\mathcal{K}$ benutzt für $s_i \notin \{s_E, s_{eK}\}$ und $\overline{x} = \overline{a}$ die Überführung

$$(s_i, \overline{a}, \overline{a}) \longrightarrow (s_j, \lambda)$$

und befindet sich danach in der Konfiguration $(s_j, g', \overline{w}' \diamond \triangle)$.
Der Fall $s_i \in \{s_E, s_{eK}\}$ kann nicht auftreten, da die Eingabe bisher ungleich λ war und der Keller nicht leer ist und nur in diesem Fall der Automat in einen der Zustände s_E oder s_{eK} wechselt.
Falls $\overline{x} \neq \overline{a}$ ist der Automat $\mathcal{K}$ nicht definiert und bleibt in der Konfiguration $(s_i, \overline{a}g', \overline{xw}' \diamond \triangle)$.
Der Automat $\mathcal{D}$ benutzt für $\overline{x} = \overline{a}$ die Überführung

$$(s_{1D}, \overline{a}, \overline{a}) \longrightarrow (s_{1D}, \lambda)$$

und befindet sich danach in der Konfiguration $(s_{0D}, g', \overline{w}'\triangle)$.
Für $\overline{x} \neq \overline{a}$ ist der Automat $\mathcal{D}$ nicht definiert und bleibt in der Konfiguration $(s_{1D}, \overline{a}g', \overline{xw}'\triangle)$.
Somit haben $\mathcal{D}$ und $\mathcal{K}$ denselben Kellerinhalt nach Abarbeitung von g''.
Befindet sich der Automat $\mathcal{K}$ in der Konfiguration $(s_0, ag', \overline{w} \diamond \triangle)$, dann wechselt er in die Konfiguration $(s_j, g', \overline{aw} \diamond \triangle)$. Der Automat $\mathcal{D}$ wechselt von der Konfiguration $(s_0, ag', \overline{w}\triangle)$ in $(s_0, g', \overline{aw}\triangle)$.
Ist $\mathcal{K}$ in der Konfiguration $(s_0, \overline{a}g', \overline{b}\overline{w}' \diamond \triangle)$ und $\mathcal{D}$ in der Konfiguration $(s_0, \overline{a}g', \overline{b}\overline{w}'\triangle)$ gibt es für $\overline{b} \neq \overline{a}$ in beiden Automaten keine Produktion die angewendet werden kann. Somit sind beide nicht definiert, sofern $\overline{b} \neq \overline{a}$. Sonst wechselt $\mathcal{K}$ in die Konfiguration $(s_j, g', \overline{w} \diamond \triangle)$ und $\mathcal{D}$ in die Konfiguration $(s_0, g', \overline{w}'\triangle)$.

Der zweite Teil des Beweises befasst sich mit den Zuständen des RegDyck Automaten $\mathcal{K}$ und den Zuständen des endlichen Automaten $\mathcal{R}$.
Es sei $\mathcal{R}$ in der Konfiguration (s_r, g) und $\mathcal{K}$ in der Konfiguration $(s_k, g, \diamond\triangle)$, $s_r \in S_R, s_k \in S_K \setminus \{s_{0K}\}$. Dann folgt nach $|g|$ Schritten, dass $\mathcal{R}$ in der Konfiguration $(S_i, \lambda), S_i \subseteq S_R$, und $\mathcal{K}$ in der Konfiguration $(S_i, \lambda, \overline{v} \diamond \triangle)$ ist, da $\mathcal{R}$ nichtdeterministisch sein kann. Hierbei ist $\overline{v} \in \overline{A}^*$.
Ist $s_k = s_{1K}$, so ist $s_r = s_{0R}$. Dann überführt sowohl $\mathcal{K}$ als auch $\mathcal{R}$ in die Menge $S_i \subseteq S_R$. Somit ist die Zustandsmenge nach der Überführung wieder gleich.
Im Fall $s_k \neq s_{1K}$ gilt $s_k = s_r$ oder $s_k \in \{s_E, s_{eK}\}$.

Der Beweis folgt per Induktion über die Länge eines Teilwortes $g'' \in (A \cup \overline{A})^*$ der Eingabe $g = g'', g' \in (A \cup \overline{A})^*$.
Sei $|g''| = 0$, also $g'' = \lambda$. Die Überführungsfunktion des Automaten $\mathcal{R}$ ist nicht definiert und der Automat bleibt in der angegebenen Konfiguration.
Falls $s_k \in S_R$, verwendet $\mathcal{K}$ entweder die Überführung

$$(s_k, \lambda, \diamond) \longrightarrow (s_E, \diamond)$$

oder die Überführung

$$(s_k, \lambda, \diamond) \longrightarrow (s_{eK}, \lambda).$$

Hier unterscheiden sich die Zustandsmengen von $\mathcal{K}$ und von $\mathcal{R}$. Dies ändert allerdings nichts daran, dass $\mathcal{K}$ den Durchschnitt der Sprachen von $\mathcal{D}$ und $\mathcal{R}$ erkennt. Denn in diesem Fall ist entweder ein Dyckprime abgearbeitet, der sowohl von $\mathcal{R}$ als auch von $\mathcal{D}$ akzeptiert wird, oder es ist ein Dyckprime, der nur von $\mathcal{D}$ akzeptiert wird. Dann akzeptiert allerdings $\mathcal{K}$ auch nicht, da $\mathcal{R}$ nicht akzeptiert, also s_k kein Endzustand des Automaten $\mathcal{R}$ ist.
Die Fälle $s_k \in \{s_E, s_{eK}\}$ können nicht auftreten, da dafür bereits mindestens eine Überführung vollzogen worden sein muss.
Somit befinden sich beide Automaten in derselben Zustandsmenge, sofern die Eingabe von $\mathcal{R}$ akzeptiert wird.

Die Behauptung gelte für $|g''| \leq n \in \mathbb{N}_0$.

Nach Induktionsvoraussetzung und Konstruktion befindet sich $\mathcal{R}$ nach der Abarbeitung eines Wortes w, wobei $g = g''g'$ mit $g'' = wa$ oder $g'' = w\overline{a}, |w| = n$, in einer Zustandsmenge $S_i \subseteq S_R$. Ebenfalls befindet sich der Kellerautomat $\mathcal{K}$ in der Zustandsmenge S_i und hat den Kellerinhalt $\overline{xw}'$, $\overline{x} \in \overline{A}, \overline{w}' \in \overline{A}^*$. Die restliche Eingabe besteht aus einem Wort ag' oder $\overline{a}g'$.

1. Fall: $g'' = wa$
Es sei $(s_i, ag', \overline{xw}' \diamond \triangle)$ eine Konfiguration des Automaten $\mathcal{K}$ und (s_i, ag') eine Konfiguration des Automaten $\mathcal{R}$ mit $s_i \in S_i$.
Der Automat $\mathcal{K}$ wendet die Überführung

$$(s_i, a, \overline{x}) \longrightarrow (s_j, \overline{ax}), \overline{x} \in G_K$$

an. Die Überführung wird für ein beliebiges oberes Zeichen des Kellers benutzt. Danach befindet sich der Automat in der Konfiguration $(s_j, g', \overline{axw}' \diamond \triangle)$. Der endliche Automat wendet die Überführung

$$(s_i, a) \longrightarrow s_j$$

an. Nach Konstruktion von $\mathcal{R}$ existiert für jedes Zeichen a eine solche Überführung, die in den gleichen Zustand führt wie $\mathcal{K}$. Somit befinden sich beide Automaten nach Abarbeiten des Zeichens a in dem gleichen Zustand s_j.

2. Fall: $g'' = w\overline{a}$
Es sei $(s_i, \overline{a}g', \overline{xw}' \diamond \triangle)$ eine Konfiguration des Automaten $\mathcal{K}$ und $(s_i, \overline{a}g')$ eine Konfiguration des Automaten $\mathcal{R}$ mit $s_i \in S_i$.
Falls $\overline{x} = \overline{a}$, wendet der Automat $\mathcal{K}$ die Überführung

$$(s_i, \overline{a}, \overline{a}) \longrightarrow (s_j, \lambda)$$

an und befindet sich danach in der Konfiguration $(s_j, g', \overline{w}' \diamond \triangle)$.
Sofern $\overline{x} = \overline{a}$, benutzt der Automat $\mathcal{R}$ nach Konstruktion die Überführung

$$(s_i, \overline{a}) \longrightarrow s_j$$

und befindet sich in dem Zustand s_j.
Beide Automaten befinden sich in dem gleichen Zustand.
Gilt allerdings $\overline{x} \neq \overline{a}$, so akzeptiert der Automat $\mathcal{K}$ nicht und der Automat $\mathcal{R}$ wechselt in den nächsten Zustand. Automat $\mathcal{D}$ akzeptiert allerdings auch nicht und somit wird diese Sprache weder von $\mathcal{R}$ noch $\mathcal{D}$ erkannt.
Befinden sich die Automaten in den Konfigurationen $(s_0, \overline{a}g', \overline{b}\overline{w}' \diamond \triangle)$ und $(s_0, \overline{a}g')$, verweist $\mathcal{R}$ möglicherweise in einen anderen Zustand und $\mathcal{K}$ akzeptiert die Eingabe nicht. Falls $\overline{b} \neq \overline{a}$, akzeptiert $\mathcal{D}$ ebenfalls nicht, da $\mathcal{K}$ und $\mathcal{D}$ den gleichen Kellerinhalt haben. Somit liegt die Eingabe nicht im Durchschnitt der Sprachen von $\mathcal{R}$ und $\mathcal{D}$. □

3.2 RegDyck Grammatiken

RegDyck Grammatiken lassen sich den balancierten Grammatiken unterordnen (siehe Kapitel 4), da sie lediglich Einschränkungen gegenüber den balancierten Grammatiken besitzen. Wie im Folgenden gezeigt wird, ist die Sprachfamilie deren Sprachen durch RegDyck Grammatiken erzeugt werden, eine Teilmenge der Sprachfamilie deren Sprachen durch RegDyck Automaten erkannt werden.
Die Vorteile gegenüber anderen Sprachfamilien der XML-artigen Sprachen liegen in der Möglichkeit, mehrdeutige reguläre Ausdrücke und konkurrierende Nichtterminalzeichen in den Inhaltsmodellen der Grammatiken zuzulassen. Die Parser der RegDyck Grammatiken bestehen aus zwei Einheiten: dem Dyck-Automaten und dem endlichen Automaten, welche beide in $O(n)$ Zeit parsen. Nachteile bestehen darin, dass zwei selbsteinbettende Nichtterminalzeichen X, Y mit gleichen Ableitungssequenzen nur bedingt erlaubt sind.
In der Praxis können Verbesserungen erreicht werden, da selbsteinbettende Nichtterminale, die gleiche Ableitungen erzeugen, nur sehr selten auftreten.

Definition 3.2.21 Es sei $G = (N, T, S, P)$ eine balancierte indizierte Grammatik. Eine Abbildung $FA : N^* \longrightarrow Reg(T)$ liefert für eine Folge von Nichtterminalzeichen einen regulären Ausdruck von Terminalzeichen, wie folgt:

- Für jede Produktion $X \longrightarrow xm\overline{x}$ aus P wird ein nichtdeterministischer endlicher Automat $\mathcal{R}_X$ konstruiert, der die rechte Seite der Produktion erkennt.

- In den Überführungen sind Nichtterminalzeichen und Terminalzeichen der Grammatik enthalten. Nun werden alle Nichtterminalzeichen aus

den Überführungen folgendermaßen entfernt: Es sei $(s_i, X) \longrightarrow (s_j)$ eine Überführung eines Automaten $\mathcal{R}_Y$, wobei s_i und s_j zwei Zustände des Automaten $\mathcal{R}_Y$ sind und X ein Nichtterminalzeichen ist. Dann wird ein λ-Übergang von s_i zum Startzustand von $\mathcal{R}_X$ und von den Endzuständen von $\mathcal{R}_X$ zu s_j erzeugt. Die Überführung $(s_i, X) \longrightarrow (s_j)$ wird in $\mathcal{R}_Y$ entfernt. Zu beachten ist, dass dies auch durchgeführt wird, wenn ein Nichtterminalzeichen selbsteinbettend auftritt.

- Wenn $FA(X), X \in N$, erzeugt werden soll, wird aus dem nichtdeterministischen endlichen Automaten $\mathcal{R}_X$ ein regulärer Ausdruck erzeugt.
- $FA(Z)$, wobei Z eine Folge $Z = Z_0 Z_1 \dots Z_n, n \in \mathbb{N}_0$, von Nichtterminalzeichen ist, ist die Konkatenation $FA(Z) = FA(Z_0)FA(Z_1)\dots FA(Z_n)$.

Für die Definition von RegDyck Grammatiken wird die Eigenschaft benutzt, dass zwei Nichtterminalzeichen in Konflikt stehen. Um in Konflikt zu stehen, müssen die beiden Nichtterminalzeichen selbsteinbettend sein oder es muss die Möglichkeit bestehen, aus ihnen in einer Folge von Ableitungsschritten ein selbsteinbettendes Nichtterminalzeichen abzuleiten. Die selbsteinbettenden Nichtterminalzeichen müssen zusätzlich gleiche Ableitungssequenzen erzeugen. Es sei X ein selbsteinbettendes Nichtterminalzeichen und $X \Longrightarrow^+ u_0 u_1 \dots u_n X v_n \dots v_1 v_0$ eine Ableitungsfolge, wobei u_i, v_i für $i \in \{0, \dots, n\}, u_i v_i \in D$, Folgen von Terminalzeichen sind. Dann erkennt der RegDyck Automat unabhängig voneinander die Folgen u_i und v_i durch Schleifen. Da der RegDyck Automat weder in seinem Keller, noch in seinen Zuständen kodieren kann, wieviele Schleifen von $u^{'}$s er durchlaufen hat, sondern nur welche Zeichen er noch erkennen muss, um einen Dyckprime zu erkennen, ist es unklar, wie oft er die Schleife von $v^{'}$s durchlaufen muss.
Wenn nun ein anderes selbsteinbettendes Nichtterminalzeichen Y mit der Ableitungsfolge $Y \Longrightarrow^+ u'_0 u'_1 \dots u'_m Y v'_m \dots v'_1 v'_0$ existiert und es eine Kombination $u_i v'_j \in D, j \in \{0, \dots, m\}$, gibt, kann es dazu kommen, dass der RegDyck Automat die durch die Grammatik beschriebene Sprache nicht korrekt erkennt, sondern zusätzlich noch andere Wörter.
Die Definition der RegDyck Grammatiken verhindert solche Konflikte von Nichtterminalzeichen.

Es gibt zwei Arten, dass zwei Nichtterminalzeichen in Konflikt stehen. Die erste Art kann auftreten, wenn zwei Nichtterminalzeichen in einem Inhaltsmodell enthalten sind. Die zweite Art kann auftreten, wenn ein Nichtterminalzeichen im Inhaltsmodell des anderen oder im Inhaltsmodell eines Nachfolgers des anderen enthalten ist. Dabei ist ein Nachfolger von X ein Nachfolger der Produktion von X im Baum der indizierten balancierten Grammatik. Die zweite Art muss genauer betrachtet werden. Es seien X und Y zwei Nichtterminalzeichen, wobei überprüft wird, ob X zu Y auf die zweite Art

in Konflikt steht. Dann muss die Produktion von Y ein Nachfolger von der Produktion von X im Baum der indizierten balancierten Grammatik sein. Nun müssen die Ableitungsfolgen untersucht werden, die X erzeugt, bis Y auftritt. In der Ableitungsfolge werden jedoch nur die Nachfolger untersucht, aus denen Y abgeleitet werden kann. Alle anderen Teile der Grammatik aus denen Dyckprimes erzeugt werden können, werden nicht untersucht. Deshalb werden in der Definition Produktionen verändert. Es werden nur die Ableitungsfolgen herausgefiltert, die zu einem Konflikt zwischen den beiden Nichtterminalzeichen führen könnten.

Definition 3.2.22 Im Folgenden wird definiert, wann zwei Nichtterminalzeichen in Konflikt stehen. Es sei $G = (N, T, S, P), T = A \cup \overline{A}$, eine indizierte balancierte Grammatik und X und Y zwei Nichtterminalzeichen. Zwei Unterschiedliche Fälle sind zu betrachten:

1. Y steht in Konflikt zu X, wenn:
   $\exists X, Y, Z \in N, \exists z \in A, \exists \overline{z} \in \overline{A}, \exists u, v, w \in N^* :$

   $$(Z \longrightarrow zm\overline{z} \wedge uXwYv \in L(m) \wedge L(FA(X)FA(Y)) \cap D_T \neq \emptyset)$$

   Der Konflikt wird aufgehoben, wenn ein Wort $uCv \in L(m)$ existiert und $L(G') = L(FA(X)FA(w)FA(Y)) \cap D_T, G' = (N, T, C, P)$ gilt.

2. Ein Nichtterminalzeichen X steht auch in Konflikt zu Y, wenn gilt:
   V_Y ist die Menge der Nichtterminalzeichen, deren Produktionen Vorgänger und N_Y die Menge der Nichtterminalzeichen, deren Produktionen Nachfolger von der Produktion $Y \longrightarrow ym_Y\overline{y}$ im Baum der indizierten balancierten Grammatik sind.
   Für alle Produktionen $Z \longrightarrow zm\overline{z}$, wobei $Z \in V_Y$, wird das Inhaltsmodell gelöscht und ein neues Inhaltsmodell m_0 erzeugt, welches durch die disjunkte Vereinigung der Elemente der Menge $M_0 = \{V \in N \mid (\alpha V \beta \in L(m)) \wedge ((V = X) \vee (V \in N_X \wedge V \in V_Y)), \alpha, \beta \in N^*\}$ beschrieben wird.
   Die Produktionen und Nichtterminalzeichen, die durch diese Konstruktion erzeugt werden, werden mit einem $^{\#}$ versehen.
   X steht im Konflikt zu Y, wenn gilt:
   $\exists Y \in N, X \in V_Y :$

   $$((L(FA(Y)FA(X^{\#})) \cap D_T \neq \emptyset) \vee (L(FA(X^{\#})FA(Y)) \cap D_T \neq \emptyset))$$

   Die Konflikte werden aufgehoben, wenn $L(FA(X)) \cap D_T = L(G'), G' = (N, T, Y, P)$.

□

Nun wird definiert, in welchem Fall ein Nichtterminalzeichen mit sich selbst in Konflikt steht.

Definition 3.2.23 Es sei $G = (N, T, S, P)$ eine balancierte Grammatik und X ein selbsteinbettendes Nichtterminalzeichen. X steht in Konflikt mit sich selbst, wenn Ableitungen

$$\begin{array}{lcl} X & \Longrightarrow & uXv \\ X & \Longrightarrow & u'Xv' \end{array}$$

existieren, so dass $uv' \in D_T$ und keine Ableitung

$$X \quad \Longrightarrow \quad uXv'$$

existiert.

□

Definition 3.2.24 Eine RegDyck Grammatik ist eine balancierte Grammatik $G = (N, T, S, P)$ für die gilt:

- Es existieren keine Nichtterminalzeichen, die in Konflikt mit sich selbst stehen.
- Es existieren keine Nichtterminalzeichen, die in Konflikt miteinander stehen.

□

Es folgen nun ein paar Beispiele für RegDyck Grammatiken.

Beispiel 3.2.14 Es sei $G = (N, T, \{S\}, P)$ eine balancierte Grammatik mit:

- $N = \{S, Y, Z\}$
- $T = \{x, \overline{x}, y, \overline{y}, z, \overline{z}\}$
- $P = \{$
  $\begin{array}{lcl} S & \longrightarrow & xY\overline{x} \\ Y & \longrightarrow & y((ZS)|(SZ))?\overline{y} \\ Z & \longrightarrow & z\overline{z}\} \end{array}$

Das Nichtterminalzeichen S steht in Konflikt mit sich selbst, da die Ableitungen $S \Longrightarrow^+ (xyz\overline{z})^n S(\overline{yx})^n$ und $S \Longrightarrow^+ (xy)^m S(z\overline{zyx})^m$ möglich sind, jedoch nicht $S \Longrightarrow^+ (xyz\overline{z})^l S(z\overline{zyx})^l$ mit $l, m, n \in \mathbb{N}$.
Würde die Produktion von Y durch $Y \longrightarrow y((ZS)|(SZ)|(ZSZ))?\overline{y}$ ersetzt, wäre die Grammatik eine RegDyck Grammatik.

□

Beispiel 3.2.15 Es sei $G = (N, T, \{S\}, P)$ eine balancierte Grammatik mit:

- $N = \{S, Y, Z\}$

- $T = \{x, \overline{x}, y, \overline{y}\}$
- $P = \{$
  $S \longrightarrow xYZ\overline{x}$
  $Y \longrightarrow yY?\overline{y}$
  $Z \longrightarrow yZ?\overline{y}\}$

Die balancierte Grammatik G ist keine RegDyck Grammatik, weil in der Startproduktion zwei selbsteinbettende Nichtterminale Y und Z existieren. Beide stehen in Konflikt miteinander, da $FA(Y) = (y)^+(\overline{y})^+$, $FA(Z) = (y)^+(\overline{y})^+$ und $L((y)^+(\overline{y})^+(y)^+(\overline{y})^+) \cap D_T \neq \emptyset$.
Der Konflikt kann aufgehoben werden, wenn die Produktionsmenge folgendermaßen ersetzt wird:

$$\begin{array}{lll} S & \longrightarrow & x((YZ)|A)\overline{x} \\ A & \longrightarrow & y(A|(YZ))\overline{y} \\ Y & \longrightarrow & yY?\overline{y} \\ Z & \longrightarrow & yZ?\overline{y} \end{array}$$

□

Beispiel 3.2.16 Es sei $G = (N, T, \{S\}, P)$ eine balancierte Grammatik mit:

- $N = \{S, X, Y, Z\}$
- $T = \{s, \overline{s}, y, \overline{y}\}$
- $P = \{$
  $S \longrightarrow sXZ\overline{s}$
  $X \longrightarrow yY\overline{y}\}$
  $Y \longrightarrow yY?\overline{y}$
  $Z \longrightarrow yZ?\overline{y}$

Die balancierte Grammatik G ist keine RegDyck Grammatik, da die Nichtterminalzeichen X, Z in Konflikt stehen, da $FA(X) = y(y)^+(\overline{y})^+\overline{y}$ und $FA(Z) = (y)^+(\overline{y})^+$ und $L(y(y)^+(\overline{y})^+\overline{y}(y)^+(\overline{y})^+) \cap D_T \neq \emptyset$. Nach gleichem Muster wie im vorherigen Beispiel könnte die Grammatik zu einer RegDyck Grammatik ergänzt werden.

□

Beispiel 3.2.17 Es sei $G = (N, T, \{S\}, P)$ eine balancierte Grammatik mit:

- $N = \{S, Y, Z\}$
- $T = \{x, \overline{x}, z, \overline{z}\}$
- $P = \{$
  $S \longrightarrow x(S|ZYZ)\overline{x}$
  $Y \longrightarrow xY?\overline{x}\}$
  $Z \longrightarrow z\overline{z}\}$

Die balancierte Grammatik G ist keine RegDyck Grammatik, da die Nichtterminalzeichen S, Y in Konflikt stehen. Es gilt $FA(S^{\#}) = (x)^+(\overline{x})^+$ und $FA(Y) = (x)^+(\overline{x})^+$ und $L((x)^+(\overline{x})^+(x)^+(\overline{x})^+) \cap D_T \neq \emptyset$ gilt. Der Konflikt wird nicht verhindert, da in $FA(S) \cap D_T$ das Wort $w = xxxz\overline{z}x\overline{x}\overline{x}z\overline{zxx}$ enthalten ist, jedoch nicht in $L(G)$.

□

Konstruktion eines RegDyck Automaten aus einer RegDyck Grammatik

Es wird eine beliebige RegDyck Grammatik vorgegeben und für diese ein RegDyck Automat konstruiert.

Es sei $G = (N, T, S, P), T = A \cup \overline{A}$ eine RegDyck Grammatik und $L(G)$ die dazugehörige Sprache.
Alle Produktionen in P haben die Form

$$X \longrightarrow xm\overline{x},$$

wobei $X \in N$, $x \in A$, $\overline{x} \in \overline{A}$, $m \in Reg(N)$.
Die Grammatik G wird indiziert und zu $G' = (N', T', S', P')$.

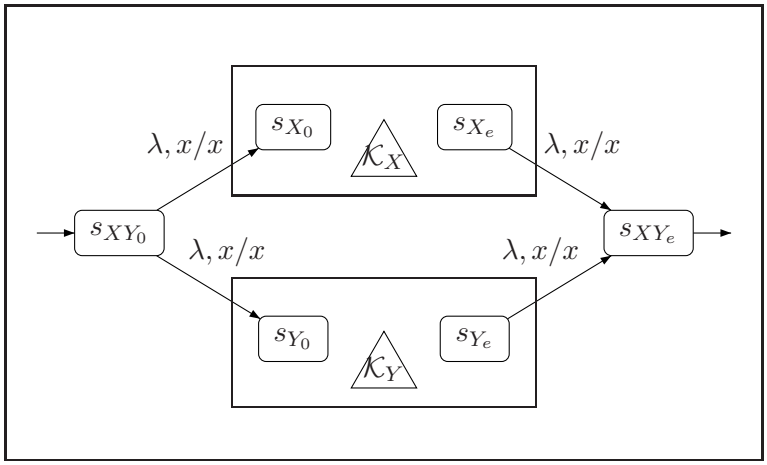
Nun wird ein RegDyck Automat konstruiert, der genau die Sprache dieser indizierten Grammatik erkennt. Die Konstruktion eines erkennenden RegDyck Automaten $\mathcal{K} = (S_K, A_K, G_K, s_{0K}, \delta_K)$ enthält die Konstruktion von RegDyck Automaten $\mathcal{K}_X = (S_X, A_X, G_X, s_{0X}, \delta_X)$. Für jedes Nichtterminalzeichen $X \in N'$ wird ein erkennender Automat $\mathcal{K}_X$ konstruiert. Dieser Automat erkennt die Sprache, die durch das Nichtterminalzeichen X erzeugt wird. Zum Schluss werden die einzelnen erkennenden Automaten gemäß der Grammatik miteinander verbunden, so dass die komplette Sprache der Grammatik G' durch den Automaten $\mathcal{K}$ erkannt wird.
Für alle Nichtterminale $X \in N'$ wird nun ein Teilautomat $\mathcal{K}_X$ von $\mathcal{K}$ konstruiert.
Wenn $m = \lambda$, dann müssen lediglich folgende Überführungen erzeugt werden:

$$\begin{array}{lcl} (s_0, x, \overline{z}) & \longrightarrow & (s_1, \overline{xz})\ \forall \overline{z} \in G_X, \\ (s_1, \overline{x}, \overline{x}) & \longrightarrow & (s_{X_e}, \lambda) \end{array}$$

mit $s_0, s_1, s_{X_e} \in S_X, x \in A_X, \overline{z}, \overline{x} \in G_X$.
Wenn $m \neq \lambda$, dann muss noch beachtet werden, dass die Inhaltsmodelle aus regulären Ausdrücken bestehen und das Zusammensetzen der Teilautomaten erschwert wird.
Die Konstruktion erfolgt induktiv über den Aufbau der regulären Ausdrücke. In den folgenden Abbildungen ist s_{X_0} der Startzustand und s_{X_e} der Endzustand des Automaten $\mathcal{K}_X$. Der Rahmen um $\mathcal{K}_X$ bedeutet, dass hier der Automat $\mathcal{K}_X$ arbeitet (analoges gilt für $\mathcal{K}_Y$).

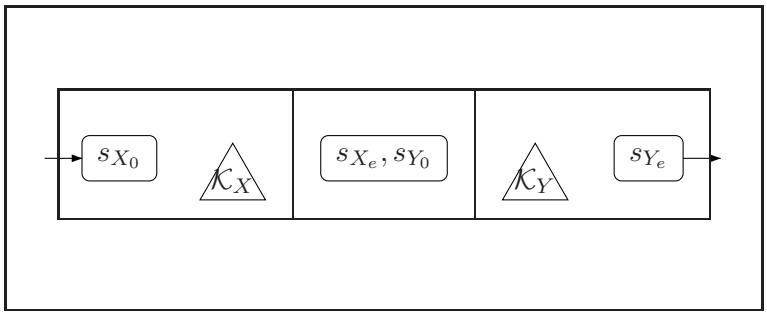
Es wird angenommen, dass $\mathcal{K}_X$ und $\mathcal{K}_Y$ RegDyck Automaten für die Produktionen der Nichtterminale X und Y sind.

1. Für den regulären Ausdruck $X|Y$ wird $\mathcal{K}_{X|Y}$ konstruiert.



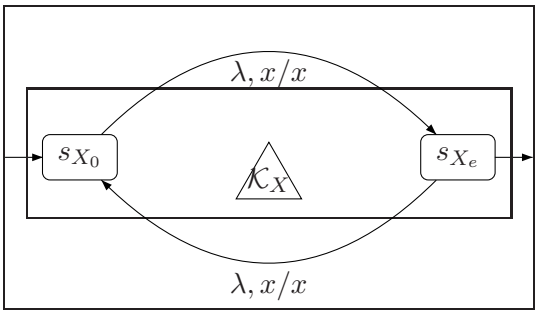
   Hierbei wird über den gemeinsamen Startzustand s_{XY_0} von $\mathcal{K}_X$ und $\mathcal{K}_Y$ durch einen λ-Übergang in die Startzustände von $\mathcal{K}_X$ und $\mathcal{K}_Y$ verwiesen. Es existiert ein gemeinsamer Endzustand s_{XY_e}.

2. Für den regulären Ausdruck XY wird der zusammengesetzte RegDyck Automat $\mathcal{K}_{XY}$ konstruiert.



   Hierbei werden der Endzustand s_{X_e} und der Anfangszustand s_{Y_0} zu einem Zustand vereinigt.

3. Für den regulären Ausdruck X^* wird der RegDyck Automat $\mathcal{K}_{X^*}$ konstruiert.



Hierbei wird ein λ-Übergang vom Startzustand s_{X_0} in den Endzustand s_{X_e} erzeugt und umgekehrt.

Ein Fall muss gesondert betrachtet werden. Es sei X ein Nichtterminalzeichen, für das ein RegDyck Automat $\mathcal{K}_X$ konstruiert werden soll. Während der Konstruktion des RegDyck Automaten $\mathcal{K}_X$ tritt das Nichtterminalzeichen X nochmal auf. Da die balancierte Grammatik indiziert ist, kann es nur innerhalb der Konstruktion des RegDyck Automaten $\mathcal{K}_X$ auftreten und sonst nicht. Das Nichtterminalzeichen X ist somit ein selbsteinbettendes Nichtterminalzeichen. Der Automat, der für X nun ein weiteres mal konstruiert werden müsste, ist also derselbe Automat $\mathcal{K}_X$. Er wird jedoch nicht erneut konstruiert, weil die Konstruktion wegen der Selbsteinbettung wieder und wieder durchgeführt werden müsste. Für alle Konstruktionen (siehe oben Punkt 1 bis 3), die für das Inhaltsmodell durchgeführt werden müssen, muss lediglich der Startzustand und der Endzustand eines RegDyck Automaten bekannt sein. Beide sind von dem RegDyck Automaten $\mathcal{K}_X$ bekannt. Somit kann dieser für die Konstruktion verwendet werden.

Es bleibt zu zeigen, dass alle λ-Übergänge ersetzt werden können, da RegDyck Automaten nur einen einzigen λ-Übergang vom Startzustand s_{0K} zum Initialisieren erlauben und λ-Übergänge zum Entfernen des Kellersymbols $\diamond$ nach dem Abarbeiten eines Dyckprimes.
Es seien $s_k, s_j \in S_X$ zwei Zustände eines Automaten $\mathcal{K}_X$.
Wenn eine Überführung

$$(s_k, \lambda, \overline{z}) \longrightarrow (s_j, \overline{z}), \overline{z} \in G_X$$

existiert, werden für alle Überführungen der Form

$$(s_j, a, \overline{z}) \longrightarrow (s_n, \overline{a}\overline{z}), a \in A_X, \overline{a}, \overline{z} \in G_X, s_n \in S_X$$

und

$$(s_j, \overline{a}, \overline{a}) \longrightarrow (s_n, \lambda)$$

Überführungen

$$(s_k, a, \overline{z}) \longrightarrow (s_n, \overline{az})$$

oder

$$(s_k, \overline{a}, \overline{a}) \longrightarrow (s_n, \lambda)$$

erzeugt.
Dies ist bei den RegDyck Automaten in jedem Fall erlaubt, da sie nichtdeterministische Zustandsüberführungsfunktionen besitzen dürfen.
Die Überführungen des Automaten $\mathcal{K}$, die dafür sorgen, dass die Dyckprimes erkannt werden, die die Grammatik erzeugt, sind durch die Konstruktion erzeugt. Nun müssen noch die Überführungen konstruiert werden, die dafür sorgen, dass nur gültige Dyckprimes erkannt werden.
Es seien

$$\begin{gathered} S \longrightarrow xm_0\overline{x} \\ S \longrightarrow ym_1\overline{y} \\ S \longrightarrow wm_2\overline{w} \\ \dots \end{gathered}$$

die Startproduktionen der Grammatik G'. Dann besitzt der Automat $\mathcal{K}_S$ einen Startzustand s_i, für den es folgende Überführungen gibt:

$$\begin{array}{ccc} (s_{S_0}, x, \overline{z}) & \longrightarrow & (s_j, \overline{xz}) \\ (s_{S_0}, y, \overline{z}) & \longrightarrow & (s_k, \overline{yz}) \\ (s_{S_0}, w, \overline{z}) & \longrightarrow & (s_l, \overline{wz}) \\ & \dots & \end{array}$$

Es wird für $\mathcal{K}$ die Startüberführung

$$(s_{0K}, \lambda, \triangle) \longrightarrow (s_{1K}, \diamond\triangle)$$

und die Überführungen

$$\begin{gathered} (s_{1K}, x, \diamond) \longrightarrow (s_j, \overline{x}\diamond) \\ (s_{1K}, y, \diamond) \longrightarrow (s_k, \overline{y}\diamond) \\ (s_{1K}, w, \diamond) \longrightarrow (s_l, \overline{w}\diamond) \\ \dots \end{gathered}$$

erzeugt.
s_{S_e} sei der Zustand der nach Abarbeitung eines Dyckprimes von $\mathcal{K}_S$ erreicht wurde. Dafür wird in $\mathcal{K}$ eine Überführung zum Leeren des Kellers definiert.

$$(s_{S_e}, \lambda, \diamond) \longrightarrow (s_{eK}, \lambda), s_{S_e} \in S_K \setminus \{s_{0K}, s_{1K}, s_E, s_{eK}\}$$

Für alle anderen Zustände $s_i \in S_K \setminus \{s_{0K}, s_{1K}, s_E, s_{eK}, s_{S_e}\}$ werden Überführungen

$$(s_i, \lambda, \diamond) \longrightarrow (s_E, \diamond)$$

in $\mathcal{K}$ erzeugt.
Damit ist der komplette RegDyck Automat $\mathcal{K}$ konstruiert.

Satz 3.2.3 Jede, von einer RegDyck Grammatik G erzeugte Sprache, kann durch einen RegDyck Automat $\mathcal{K}$ erkannt werden.

Beweis. Es muss nur gezeigt werden, dass der RegDyck Automat genau die Ableitungen der Grammatik G erkennt, die Selbsteinbettung enthalten. Wenn keine Selbsteinbettend vorhanden ist, wird durch die Konstruktion des Automaten mittels der abgewandelten Thompson Methode klar, dass der RegDyck Automat $\mathcal{K}_X$ genau die Sprache, die von der Grammatik G erzeugt wird, erkennt.
Es sei $G = (N, T, S, P)$ eine indizierte RegDyck Grammatik und $\mathcal{K} = (S_K, A_K, G_K, s_{0K}, \delta_K)$ der RegDyck Automat aus der obigen Konstruktion.
Es sei

$$S \Longrightarrow^+ uyw\alpha\overline{y}v$$

eine Ableitung der Grammatik G, wobei $S \in N, u \in T^*, y, \overline{y} \in T, w \in D_T, \alpha \in Reg(N), v \in Reg(N \cup T)$. Nun wird nach einem selbsteinbettenden Nichtterminalzeichen X aus der Menge $M_\alpha = \{Y \mid Yq \in L(\alpha)\}$ abgeleitet, $X \in M_\alpha$.
X erzeugt

$$X \Longrightarrow^* u_0u_1 \dots u_n \; X \; v_n \dots v_1v_0 \text{ mit}$$

$n \in \mathbb{N}, u_iv_i \in D_T, i \in \{1, \dots, n\}$.
Der Automat $\mathcal{K}_X$ erkennt einen selbsteinbettenden Teil mittels einer Schleife. In dem Teilautomat $\mathcal{K}_X$, der die Sprache des selbsteinbettenden Nichtterminalzeichen erkennt, wird, an der Stelle in dem das Nichtterminalzeichen selbsteinbettend auftritt, wieder in den Startzustand von $\mathcal{K}_X$ verwiesen. So erkennt er für X die Sequenzen

$$u_0u_1 \dots u_nw_0v_1v_2 \dots v_j \text{ mit}$$

$j, k \in \mathbb{N}, w_0, u_kv_{n-k} \in D_T$.
Jedoch wird durch den Keller des RegDyck Automaten gesichert, dass nur Dyckprimes erkannt werden. Das bedeutet, wenn der RegDyck Automat eine Schleife von u mehr durchlaufen hat wie von v, dann muss ein anderer Teilautomat existieren, der auch eine Sprache eines selbsteinbettenden Nichtterminalzeichen beschreibt und durch einen zusätzlichen Schleifendurchlauf den Keller leert.
Wir nehmen an, es gebe einen zweiten Teilautomaten $\mathcal{K}_Y$ der dafür sorgt, dass der Keller geleert wird.
Dann muss in der Grammatik G ein selbsteinbettendes Nichtterminalzeichen Y existieren für das gilt:

$$d = u_0u_1 \dots u_n \; X \; v_j \dots v_1v_0 \; p_0p_1 \dots p_l \; Y \; q_m \dots q_1q_0 \text{ mit}$$

$j, l, m \in \mathbb{N}, d \in D_T$.
Es gibt zwei Möglichkeiten, wie die Ableitung auftreten können:

1. $d' = \alpha\beta \; u_0u_1 \dots u_n \; w_0 \; v_1v_2 \dots v_j \; \overline{\beta} \; \gamma \; u'_0u'_1 \dots u'_m \; w'_0 \; v'_1v'_2 \dots v'_i \; \overline{\gamma} \; \overline{\alpha}$
   mit $\alpha, \overline{\alpha}, \beta, \overline{\beta}, \gamma, \overline{\gamma} \in T^*, d' \in D_T, n \neq j, m \neq i$.der

2. $d' = \alpha \; u'_0u'_1 \dots u'_m \; \beta \; u_0u_1 \dots u_n \; w_0 \; v_1v_2 \dots v_j \; \overline{\beta} \; v'_1v'_2 \dots v'_i \; \overline{\alpha}$

Im ersten Fall würde jedoch Y zu X in Konflikt stehen (nach der ersten Variante der Konflikte), weil $FA(X)FA(Y) \cap D_T \neq \emptyset$. Im zweiten Fall würde Y zu X im Konflikt stehen (nach der zweiten Variante der Konflikte), da $FA(X)FA(Y^\#) \cap D_T \neq \emptyset$ oder $FA(Y^\#)FA(X) \cap D_T \neq \emptyset$. Oder X stände im Konflikt mit sich selbst. Es sind jedoch keine Nichtterminalzeichen, die im Konflikt stehen erlaubt und auch keine Nichtterminalzeichen, die mit sich selbst im Konflikt stehen. Dies führt zum Widerspruch der Annahme.

Es zeigt sich, dass der Automat $\mathcal{K}$ genau die Sprache der Grammatik G erkennt.

□

Es wurde nur gezeigt, dass jede RegDyck Grammatik durch einen RegDyck Automaten erkannt wird. Die umgekehrte Richtung bleibt zu zeigen.
Für die Untersuchung der XML-artigen Grammatiken ist es unwichtig, ob aus jedem RegDyck Automaten eine RegDyck Grammatik erzeugt werden kann. Die Vorteile, dass mehrdeutige Inhaltsmodelle benutzt werden können und dass konkurrierende Nichtterminalzeichen in den Grammatiken vorhanden sein dürfen bleiben bestehen. Interessant wäre die Frage, ob für jede, durch einen RegDyck Automaten beschriebene Sprache, eine balancierte Grammatik existiert, die die gleiche Sprache erzeugt. Wenn dies nicht der Fall wäre, könnten durch den RegDyck Automaten noch andere nicht balancierte Mengen von Dyckprimes beschrieben werden.

Kapitel 4

Einordung der Modelle

In dem folgenden Kapitel werden die verschiedenen XML-artigen Sprachfamilien eingeordnet. So wird betrachtet, ob es eine Hierarchie zwischen den einzelnen Sprachfamilien gibt und in welchem Bereich der Chomsky Hierarchie die XML-artigen Sprachfamilien einzuordnen sind. Die Ergebnisse sollen für Validierungsalgorithmen benutzt werden. Es gilt zu überprüfen, wie lange benötigt wird, um einen Parser zu konstruieren und wie lange der Parser benötigt, um XML Daten auf ihre Struktur hin zu überprüfen.

4.1 XML Ausdrücke

In Beispielen werden XML-artige Sprachen oft durch Ausdrücke angegeben, die den regulären Ausdrücken ähneln.

Beispiel 4.1.18 $L(a^n(b\overline{b})^*c\overline{ca}^n) = \{a^n(b\overline{b})^ic\overline{ca}^n \mid i, n \in \mathbb{N}\}$

Sie werden als XML Ausdrücke bezeichnet und in diesem Unterkapitel definiert. Es wird überprüft, ob mittels dieser Ausdrücke alle Sprachen, die durch eine balancierte Grammatik erzeugt werden können, auch durch einen XML Ausdruck dargestellt werden können.
Die entstehenden Ausdrücke werden XML Ausdrücke genannt.

Definition 4.1.25 Es sei $T = A \cup \overline{A}$ ein Alphabet, das keines der Zeichen $V = \{|, ^*, ^+, ?, \emptyset, (,)\}$ enthält. XML Ausdrücke über T werden wie folgt festgelegt:

- $x\overline{x}, x \in A, \overline{x} \in \overline{A}$ und $\emptyset$ sind Unterausdrücke.
- $xu_0u_1 \dots u_i\overline{x}, i \in \mathbb{N}$ ist ein Unterausdruck, wenn $u_0, u_1, \dots, u_i, i \in \mathbb{N}$ Unterausdrücke sind.
- Wenn r und s Unterausdrücke sind, dann ist auch die Konkatenation (rs) ein Unterausdruck.

- Wenn r und s Unterausdrücke sind, dann ist auch $(r|s)$ ein Unterausdruck.
- Wenn r ein Unterausdruck ist, dann ist auch $(r)^*$ ein Unterausdruck.
- $x\overline{x}, x \in A, \overline{x} \in \overline{A}$ und $\emptyset$ sind XML Ausdrücke.
- $xu_0u_1 \dots u_i\overline{x}$ ist ein XML Ausdruck, wobei $u_0u_1 \dots u_i$ Unterausdrücke sind.
- Es gibt eine besondere Art von Ausdrücken, die als rekursive Ausdrücke bezeichnet werden. Sie können sowohl als XML Ausdrücke, als auch als Unterausdrücke benutzt werden.
  Es sei q ein Ausdruck der Form xv, wobei x ein Zeichen aus A und v eine beliebige Sequenz von Unterausdrücken über T und Zeichen $a \in A$ ist. Es sei ψ eine Abbildung, die aus einem Ausdruck q alle Unterausdrücke löscht. q und eine dazugehöhrige Sequenz $\overline{q}$ können mit einer Variablen versehen werden $(q^n, \overline{q}^n, n \in \mathbb{N})$, wenn gilt: $\psi(q)\psi(\overline{q}) \in D_T$, wobei D_T die Menge der Dyckprimes über $T = A \cup \overline{A}$ ist.
  Der Teil w zwischen q und $\overline{q}$ $(q^n w \overline{q}^n)$ ist ein beliebiger Unterausdruck.

□

Die Klammern können weggelassen werden, wenn die Semantik des Ausdrucks dadurch erhalten bleibt.
Es sei r ein Unterausdruck, dann gilt: $r^+ = rr^*$ und $r? = (r|\emptyset^*)$. Für $\emptyset^*$ kann auch λ geschrieben werden.

Jeder XML Ausdruck p über einem Alphabet A legt eine Sprache $L(p)$ folgendermaßen fest:

- Die durch $\emptyset$ bezeichnete Sprache ist die leere Sprache.
- Die durch $a\overline{a}, a, \in A, \overline{a} \in \overline{A}$ bezeichnete Sprache enthält nur das Wort $a\overline{a}$.
- Für Unterausdrücke p und q über A gilt: $L(p|q) = L(p) \cup L(q), L(pq) = L(p)L(q), L(p^*) = (L(p))^*$.
- Es sei u ein Unterausdruck, dann ist die durch den Ausdruck $au\overline{a}$ beschriebene Sprache $aL(u)\overline{a}$.
- Die von $v^n u \overline{v}^n, v, \overline{v} \in (A \cup \overline{A})^+$ erzeugte Sprache ist $L(v)^n L(u) L(\overline{v})^n$ mit $n \in \mathbb{N}$.

Der Ausdruck $(a^m\overline{a}^m)^*$ bedeutet hierbei, dass die Sprache $a^m\overline{a}^m a^i\overline{a}^i a^j\overline{a}^j \dots$ erzeugt werden kann.

Satz 4.1.4 Die Sprache jedes XML Ausdrucks kann durch eine balancierte Grammatik erzeugt werden.

Beweis. Es sei $p = xu\overline{x}, x \in A, \overline{x} \in \overline{A}$ ein XML Ausdruck, wobei u ein Unterausdruck ist. Es wird eine balancierte Grammatik $G = (N, T, S, P)$ konstruiert. Für den XML Ausdruck p wird eine Produktion

$$X \longrightarrow xm_u\overline{x}, X \in N, x \in A, \overline{x} \in \overline{A}, m \in Reg(N)$$

erzeugt. Wenn der Unterausdruck $u = \lambda$ ist, dann ist auch das Inhaltsmodell der Produktion $m_u = \lambda$. Die Konstruktion des Inhaltsmodells m_u erfolgt über den induktiven Aufbau der XML Ausdrücke. Durch ein wiederholtes Anwenden der folgenden Schritte wird die balancierte Grammatik G vervollständigt:

- Es sei $u = (u_0)^*$ der Unterausdruck, dann wird $m_u = (m_{u_0})^*$ erzeugt.
- Es sei $u = (u_0|u_1)$ der Unterausdruck, dann wird $m_u = (m_{u_0}|m_{u_1})$ erzeugt.
- Es sei $u = (u_0u_1)$ der Unterausdruck, dann wird $m_u = (m_{u_0}m_{u_1})$ erzeugt.
- Es sei $u = yu_0\overline{y}, y \in A, \overline{y} \in \overline{A}$ ein Unterausdruck, dann wird $m_u = Y$ und eine Produktion

  $$Y \longrightarrow ym_{u_0}\overline{y}$$

  erzeugt. Wenn der Unterausdruck $u_0 = \lambda$ ist, dann ist auch das Inhaltsmodell der Produktion $m_{u_0} = \lambda$. Wenn $u_0 \neq \lambda$, dann wird aus u_0 das Inhaltsmodell m_{u_0} erzeugt.
- Es sei u ein XML Ausdruck oder ein Unterausdruck der Form $(yvx)^n w(\overline{xvy})^n$, wobei x und y Zeichen aus A sind und v eine Sequenz von Unterausdrücken und Zeichen aus A. So werden wie oben beschrieben für y, w und v Nichtterminalzeichen und Produktionen erzeugt. Es sei Y das Nichtterminalzeichen, das durch die Produktion, $Y \longrightarrow ym\overline{y}$, u erzeugt und α ein regulärer Ausdruck über Nichtterminalzeichen, der w erzeugt, dann wird für x die Produktion

  $$X \longrightarrow xY|\alpha\overline{x}$$

  konstruiert.

□

Satz 4.1.5 Es kann nicht jede Sprache, die durch eine balancierte Grammatik erzeugt wird, durch einen XML Ausdruck erzeugt werden.

Beweis. Es können zum Beispiel keine selbsteinbettenden Nichtterminale behandelt werden, die mit einem Stern * versehen sind. Es sei $G = (N, T, \{S\}, P)$ eine balancierte Grammatik mit der einzigen Produktion:

$$S \longrightarrow xS^*\overline{x}.$$

Angenommen, die Sprache der balancierten Grammatik könnte durch einen XML Ausdruck erzeugt werden. Wird $L(G)$ als Baum betrachtet, dann kann ein Knoten des Baums beliebig viele Nachfolger und beliebig viele nachfolgende Ebenen besitzen.
Ein XML Ausdruck kann beliebig viele Nachfolger durch den Kleene Stern ausdrücken. Jedoch folgt aus der Definition der balancierten Grammatik, dass in der beschriebenen Sprache $L(G)$ auf jeder neuen Ebene wieder beliebig viele Knoten mit beliebig vielen Nachfolgern erzeugt werden können. Dies muss beliebig oft auf beliebig vielen Ebenen möglich sein. Das ist in einem XML Ausdruck nicht möglich, da sich der XML Ausdruck nicht wieder selbst aufrufen kann. Es können nur beliebig viele, aber fest viele Ebenen erzeugt werden, woraus sich ein Widerspruch zur Annahme ergibt. Somit können nicht alle Sprachen, die durch eine balancierte Grammatik erzeugt werden, auch durch einen XML Ausdruck dargestellt werden. □

4.2 Einordnung in die Chomsky Hierarchie

Die in Kapitel 2 definierten XML-artigen Grammatiken werden nun in die Chomsky Hierarchie eingeordnet. Darüber hinaus wird eine Hierarchie der XML-artigen Sprachfamilien nachgewiesen und schließlich wird überprüft, ob die Sprachen eine Teilfamilie der nichtdeterministisch kontextfreien Sprachen oder der deterministisch kontextfreien Sprachen sind. Dies wird eine entscheidende Rolle für die Parser spielen, da deterministisch kontextfreie Sprachen in linearer Zeit geparst werden können.

4.2.1 XML-artige Sprachen und reguläre Sprachen

Es ist intuitiv klar, dass die balancierten Grammatiken kontextfreie Sprachen beschreiben können und somit nicht zu den regulären Sprachen gehören.

Satz 4.2.6 Die balancierten Sprachen sind keine Teilmenge der regulären Sprachen.

Beweis. Es sei $G = (N, T, \{S\}, P)$ eine balancierte Grammatik mit einer einzigen Produktion

$$S \longrightarrow aS?\overline{a}.$$

Intuitiv wird klar, dass die erzeugte Sprache ($L = \{a^m\overline{a}^m \mid m \in \mathbb{N}\}$) nicht durch eine reguläre Grammatik erzeugt werden kann.
Dies wird mittels Pumping Lemma bewiesen. Die Pumping Lemma Konstante sei n. Wir wählen das Wort $a^n\overline{a}^n$ und die Länge $2n$. Aufgrund der 1. Bedingung des Pumping Lemmas darf v nicht leer sein, aufgrund der 2. Bedingung kann es nur aus $a's$ bestehen.
Die 3. Bedingung erlaubt, dass auch die Wörter $a^{n-|v|}\overline{a}^n$ zulässig sind. Dies ist ein Widerspruch zur Definition der Sprache. Der Ausdruck ist nicht regulär. Die balancierten Sprachen sind daher keine Teilmenge der regulären Sprachen. □

Satz 4.2.7 Die balancierten Sprachen sind keine Obermenge der regulären Sprachen.

Beweis. Es sei L eine reguläre Sprache und $a \in L$ ein Terminalzeichen. a kann nicht durch eine balancierte Grammatik erzeugt werden, da zu jedem a ein Gegenstück $\overline{a}$ erzeugt werden muss. Die Sprache L kann somit nicht durch eine balancierte Grammatik erzeugt werden. □

In [BB02a] wird allerdings gezeigt, dass sehr viele balancierte Sprachen zu den regulären Sprachen gehören. Diese Untermenge der balancierten Sprachen wird dort auch untersucht.

Korollar 4.2.1 Die XML Sprachen, die einfachen balancierten Sprachen und die konkurrenzverhindernden Sprachen sind keine Obermengen der regulären Sprachen.

Da die XML Sprachen, die einfachen balancierten Sprachen und die konkurrenzverhindernden balancierten Sprachen die Beispielsprache aus Satz 4.2.7 beinhalten, gehören diese auch nicht zu den regulären Sprachen.

Das nächste Unterkapitel zeigt, dass die Grammatiken zu den kontextfreien Grammatiken gehören.

4.2.2 Auflösung der Struktur von balancierten Grammatiken

In diesem Abschnitt soll gezeigt werden, dass die balancierten Grammatiken eine Unterklasse der kontextfreien Grammatiken darstellen. Dies ist nicht sofort ersichtlich, da die balancierten Grammatiken Produktionen enthalten, deren rechte Seite einen regulären Ausdruck über Nichtterminalzeichen enthält. Es kann somit unendlich viele kontextfreie Produktionen geben.
Nun werden die rechten Seiten der Produktionen in eine endliche Form gebracht. Aus der Zuordnung zu den kontextfreien Grammatiken folgt, dass die balancierten Grammatiken durch Kellerautomaten erkannt werden können.

Satz 4.2.8 Die balancierten Sprachen gehören zu den kontextfreien Sprachen.

Beweis. Gegeben sei eine balancierte Grammatik $G = (N, T, S, P)$. P sind Produktionen der Form $X \longrightarrow xm\overline{x}$. Da m aus einem regulären Ausdruck aus Nichtterminalen besteht, kann das Inhaltsmodell aus einer unbeschränkten Folge von Nichtterminalen bestehen. Diese unbeschränkten Inhaltsmodelle werden aufgelöst und damit gezeigt, dass die definierte Sprache kontextfrei ist.

Die Produktionen werden aufgelöst, so dass keine Symbole der Menge $\{*, (,), |\}$ enthalten sind. Die folgenden Schritte werden wiederholt, bis alle rechten Seiten der Produktionen endlich sind. Es gilt $X \in N, u, v, w, z \in Reg(N), x, \overline{x} \in T$:

1. Es werden alle Gruppierungen aufgelöst. Es sei $X \longrightarrow xu(w)v\overline{x}$ eine Produktion, dann wird eine neue Produktion $A \longrightarrow w, A \in N$ eingeführt und die ursprüngliche Produktion durch $X \longrightarrow xuAv\overline{x}$ ersetzt.

2. Es sei $X \longrightarrow xuw^*v\overline{x}$ eine Produktion, dann werden zwei Produktionen $A \longrightarrow wA, A \in N$ und $A \longrightarrow \lambda$ erzeugt. Zusätzlich wird $X \longrightarrow xuAv\overline{x}$ erzeugt und die ursprüngliche Produktion gelöscht.

3. Es sei $X \longrightarrow xu(v|w)z\overline{x}$ eine Produktion, dann werden die Produktionen $A \longrightarrow v, A \in N$ und $A \longrightarrow w$ erzeugt. Zusätzlich wird $X \longrightarrow xuAz\overline{x}$ eingeführt und die ursprüngliche Produktion gelöscht.

Nach der Umformung existieren nur noch endlich viele kontextfreie Produktionen. Damit ist gezeigt, dass jede balancierte Grammatik eine kontextfreie Sprache erzeugt. □

Zur Anschauung sei folgendes Beispiel verwendet:

Beispiel 4.2.19
Gegeben sei eine balancierte Grammatik:
$G = (\{S, C, D, E\}, \{a, \overline{a}, c, \overline{c}, d, \overline{d}, e, \overline{e}\}, \{S\}, P)$ mit den Produktionen

$$S \rightarrow a(CD)^*E^*\overline{a}$$
$$C \rightarrow c(S|\lambda)\overline{c}$$
$$D \rightarrow d\overline{d}$$
$$E \rightarrow e\overline{e}.$$

Es entsteht folgende Grammatik $G_1 = (\{S, S_1, S_2, A_1, A_2, C, D, E_1, E\}, \{a, \overline{a}, c, \overline{c}, d, \overline{d}, e, \overline{e}\}, \{S_1\}, P_1)$:

1. Schritt	**2. Schritt**	**3. Schritt**
		$S \to aA_2E_1\overline{a}$
	$S \to aA_2E_1\overline{a}$	$A_2 \to A_1A_2$
	$A_2 \to A_1A_2$	$A_2 \to \lambda$
$S \to aA_1^*E^*\overline{a}$	$A_2 \to \lambda$	$A_1 \to CD$
$A_1 \to CD$	$A_1 \to CD$	$E_1 \to EE_1$
$E \to e\overline{e}$	$E_1 \to EE_1$	$E_1 \to \lambda$
$C \to cS_1\overline{c}$	$E_1 \to \lambda$	$E \to e\overline{e}$
$S_1 \to S\|\lambda$	$E \to e\overline{e}$	$C \to cS_1\overline{c}$
$D \to d\overline{d}$	$C \to cS_1\overline{c}$	$S_1 \to S_2$
	$S_1 \to S\|\lambda$	$S_2 \to \lambda$
	$D \to d\overline{d}$	$S_2 \longrightarrow S$
		$D \to d\overline{d}$

Im ersten Schritt werden alle Klammern aufgelöst. Im zweiten Schritt werden die Sterne ersetzt. Im dritten Schritt wird das Zeichen | ersetzt.

Nun muss eine genauere Einordnung vorgenommen werden. So wird überprüft, ob die balancierten Sprachen in den deterministisch kontextfreien Sprachen enthalten sind und welche Rolle die anderen Grammatiktypen der XML-artigen Grammatiken spielen.

kontextfreie Sprachen

.

.

.

balancierte Sprachen $\cdots$ ***reguläre Sprachen***

Bild 4.2.1: Erste Einordnung in die Chomsky Hierarchie.

4.2.3 Balancierte Grammatiken und die Chomsky Hierarchie

Ein erstes Ziel ist es, aus einer beliebigen balancierten Grammatik einen deterministischen Kellerautomaten zu konstruieren.
Ein wichtiger Schritt ist die Konstruktion eines erkennenden Kellerautomaten für jedes Nichtterminalzeichen. Da die Inhaltsmodelle aus regulären Ausdrücken bestehen, muss gezeigt werden, wie die Sprachen zweier Automaten, die eine balancierte Sprache erkennen, vereinigt, konkateniert, iteriert und gruppiert werden. Zuletzt müssen noch selbsteinbettende Nichtterminalzeichen behandelt werden. Diese werden vorerst in der Konstruktion nicht berücksichtigt. Zum Schluss werden die Überführungen, in denen selbsteinbettende Nichtterminalzeichen enthalten sind, ersetzt.
Die Konstruktion verläuft in zwei Schritten. Zuerst wird aus der indizierten

balancierten Grammatik ein nichtdeterministischer Kellerautomat konstruiert. Danach wird der nichtdeterministische Kellerautomat in einen deterministischen Kellerautomaten umgewandelt.
In der Konstruktion wird die Eigenschaft der balancierten Sprachen genutzt, worin zu jedem Zeichen $a \in A$ aus der Eingabe genau ein Zeichen $\overline{a} \in \overline{A}$ existiert. Die Zuordnung von Zeichen $a \in A$ und Zeichen $\overline{a} \in \overline{A}$ wird spezifiziert. Es sei $G' = (N, T, S', P)$ eine indizierte balancierte Grammatik und

$$X \longrightarrow xm\overline{x}, X \in N, x, \overline{x} \in T, m \in Reg(N)$$

eine Produktion aus P. Im erkennenden nichtdeterministischen Automaten wird für das gelesene Zeichen x, das Zeichen $\overline{(\alpha, X, x)}$ in den Keller gelegt. α ist dabei die Menge der Startzustände der Kellerautomaten, welche die Sprachen von Nichtterminalzeichen beschreiben, die das Inhaltsmodell m beinhaltet, plus der Zustände, aus denen das Zeichen $\overline{x}$ dieser Produktion erkannt wird. Diese Menge ist eindeutig. Da die Grammatik indiziert ist, kann ein Startzustand nur dann in zwei verschiedenen Kellerzeichen enthalten sein, wenn ein Nichtterminalzeichen selbsteinbettend auftritt. Dann ist jedoch das Inhaltsmodell gleich und somit ist eindeutig, welcher Zustand zu welchem Kellerzeichen gehört. Somit ist im Kellerzeichen enthalten, welche Produktion noch abzuarbeiten ist und welche Automaten dafür in Frage kommen, das Inhaltsmodell abzuarbeiten.
Im deterministischen Kellerautomaten muss es nicht eindeutig sein, welches Kellerzeichen eingekellert wird (bei Mehrdeutigkeit). In diesem Fall wird ein neues Symbol, welches die Menge aller möglichen Kellersymbole enthält, in den Keller gelegt.

Satz 4.2.9 Balancierte Sprachen werden durch deterministische Kellerautomaten erkannt.

Beweis. Es sei $G' = (N, T, S', P)$ eine indizierte balancierte Grammatik. Die Grammatik muss indiziert sein, da die Nichtterminalzeichen in verschiedenen Produktionen vorkommen und für jedes Nichtterminal ein erkennender Kellerautomat konstruiert wird. Wenn die balancierte Grammatik nicht indiziert würde, würde für ein Nichtterminalzeichen X, welches in zwei verschiedenen Inhaltsmodellen auftritt, nur ein Kellerautomat erzeugt.
Zuerst wird aus der Grammatik G' ein nichtdeterministischer Kellerautomat $\mathcal{D}' = (S', T', G', \triangle, s'_0, \delta', F')$ konstruiert und dieser wird in einen deterministischen Kellerautomaten $\mathcal{D} = (S, T, G, \triangle, s_0, \delta, F)$ umgewandelt. Vorerst wird für jede Produktion ein nichtdeterministischer Kellerautomat, der die Sprache der Produktion erkennt, konstruiert. Es sei

$$X \longrightarrow xm\overline{x}, m \in Reg(N), X \in N, x, \overline{x} \in T$$

eine Produktion aus der Grammatik G'.
Im Folgenden sind $s_{(i,\mathcal{X})}$ Zustände des Automaten $\mathcal{D}'_X$. $a, x, y, \overline{a}, \overline{x}, \overline{y}$ sind

Terminalzeichen. $\overline{(\alpha, X, x)}$ sind Kellerzeichen, die eingekellert werden, wenn eine Überführung für das Zeichen x der Produktion für X benutzt wird. α ist eine Menge von Zuständen, die später definiert wird.
Zuerst sei der Fall betrachtet, dass $m = \lambda$ ist. Dann ist der erkennende Teilautomat $\mathcal{D}'_X$ von $\mathcal{D}'$ durch folgende Überführungen definiert:

$$(s_{(0,\mathcal{X})}, x, \triangle) \longrightarrow (s_{(1,\mathcal{X})}, \overline{(\{s_{(1,\mathcal{X})}\}, X, x)}\triangle)$$
$$(s_{(1,\mathcal{X})}, \overline{x}, \overline{(\{s_{(1,\mathcal{X})}\}, X, x)}) \longrightarrow (s_{(e,(X,x))}, \lambda)$$

Es ist hierbei wichtig zu beachten, dass für ein Zeichen $x \in T$ ein Zeichen in den Keller gelegt wird, welches eindeutig die Produktion identifiziert.

Im Fall, dass eine Produktion $X \longrightarrow xm\overline{x}$ mit dem Inhaltsmodell $m \neq \lambda$ existiert, muss das Inhaltsmodell mitbehandelt werden. Das Inhaltsmodell besteht aus einem regulären Ausdruck über Nichterminalzeichen. Es wird beschrieben, wie die Automaten der Nichtterminalzeichen zusammengesetzt werden und dabei insgesamt ein nichtdeterministischer Kellerautomat entsteht. Die Konstruktion erfolgt induktiv über den Aufbau der regulären Ausdrücke. Dabei genügen allgemein die Konkatenation zweier regulärer Ausdrücke, die Vereinigung, die Gruppierung und Vervielfältigung mit *. Zuletzt müssen noch selbsteinbettende Nichtterminale behandelt werden.

Die Gruppierung besagt, dass der eingeklammmerte Teilausdruck und somit sein erkennender nichtdeterministischer Kellerautomat als ein Ausdruck oder ein Automat anzusehen sind. Hier muss also nur beachtet werden, dass bei Vereinigung und Vervielfältigung der komplette Automat verwendet wird.

Für die Produktion $X \longrightarrow xm\overline{x}, m \neq \lambda$ wird der Automat $\mathcal{D}'_X$ mit folgenden Überführungen erzeugt:

$$(s_{(0,\mathcal{X})}, x, \triangle) \longrightarrow (s_{(1,\mathcal{X})}, \overline{(\{s_{(2,\mathcal{X})}\} \cup \beta, X, x)}\triangle)$$
$$(s_{(1,\mathcal{X})}, m, \overline{(\{s_{(2,\mathcal{X})}\} \cup \beta, X, x)}) \longrightarrow (s_{(2,\mathcal{X})}, \overline{(\{s_{(2,\mathcal{X})}\} \cup \beta, X, x)}\triangle)$$
$$(s_{(2,\mathcal{X})}, \overline{x}, \overline{(\{s_{(2,\mathcal{X})}\} \cup \beta, X, x)}\triangle) \longrightarrow (s_{(e,(X,x))}, \lambda)$$

Es sei V die Menge der Nichtterminalzeichen, die in m enthalten sind. Dann ist β die Menge der Startzustände der Kellerautomaten, die die Sprachen der Nichtterminalzeichen aus V erkennen. Die Startzustände werden im Kellerzeichen kodiert, damit bei der Umwandlung in einen deterministischen Kellerautomaten für jeden Startzustand eines Teilautomaten von $\mathcal{D}'$ klar ist, für welches Kellerzeichen aus der Kombination mehrerer Kellerzeichen, eine Überführung definiert wird.
In dem Kellerautomat $\mathcal{D}'_X$ gibt es eine Überführung, in der das Inhaltsmodell m enthalten ist. Diese Überführung wird noch ersetzt, wenn der erkennende Kellerautomat konstruiert ist.

Selbsteinbettende Nichtterminale werden in den Überführungen erst ganz zum Schluss behandelt.

Wenn der nichtdeterministische Kellerautomat, der die Sprache von m erkennt, eingefügt wird, muss an dem einzufügenden Automaten noch etwas verändert werden.
Es sei $\mathcal{M}$ der Kellerautomat, der die Sprache von m erkennt. Es sei

$$(s_{(0,\mathcal{M})}, y, \triangle) \longrightarrow (s_{(1,\mathcal{M})}, \overline{(\gamma, Y, y)}\triangle)$$

eine Startüberführung des Automaten, welche sich noch auf das Kellerendesymbol bezieht. Jedoch ist der Automat $\mathcal{M}$ jetzt Teil eines „großen“ Automaten, nämlich des Automaten $\mathcal{D}_X$, in den sie eingefügt worden sind.
Die Überführung wird nun zu

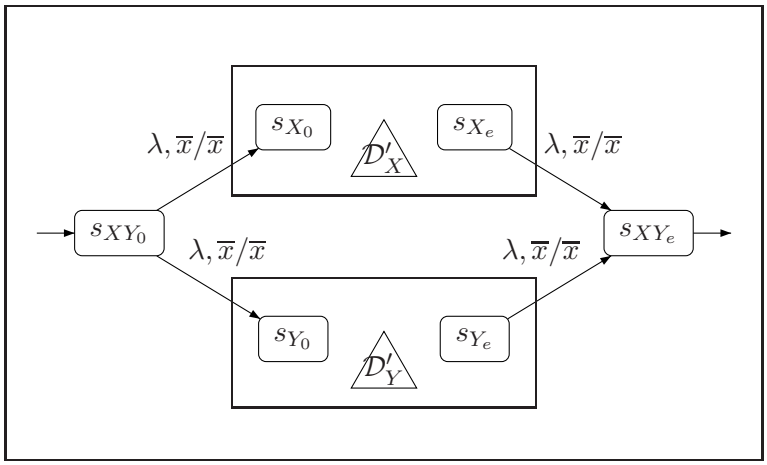
$$(s_{(0,\mathcal{M})}, y, \overline{(\alpha, X, x)}) \longrightarrow (s_{(1,\mathcal{M})}, \overline{(\gamma, Y, y)(\alpha, X, x)})$$

geändert.

Hier ist auch der Unterschied zu der Konstruktion eines RegDyck Automaten aus einer RegDyck Grammatik erkennbar. In einem RegDyck Automaten ist die Überführungsfunktion des Automaten für ein Zeichen $a \in A$ für jedes Kellersymbol definiert. Hinzu kommt, dass bei der Konstruktion aus der balancierten Grammatik die Kellerzeichen eindeutig einer Produktion zuzuordnen sind und die Startzustände der Kellerautomaten des Inhaltsmodells enthalten sind.

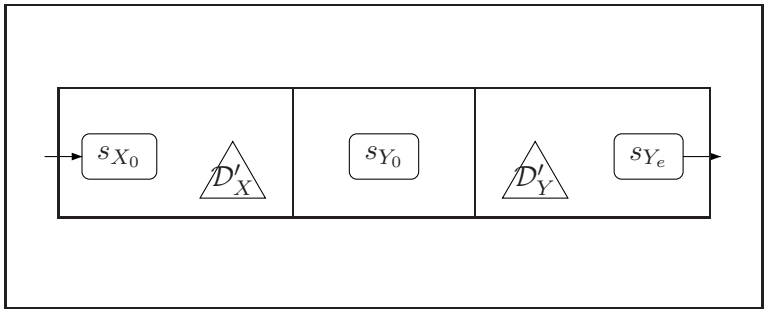
Die folgende Konstruktion ähnelt der Konstruktion aus Kapitel 3.
Hier wird die Konstruktion der nichtdeterministischen Kellerautomaten über den induktiven Aufbau der regulären Ausdrücke für die Inhaltsmodelle gezeigt.
Zuerst wird die Vereinigung beschrieben.

1. Für den regulären Ausdruck $X|Y$ wird $\mathcal{D}'_{X|Y}$ konstruiert.



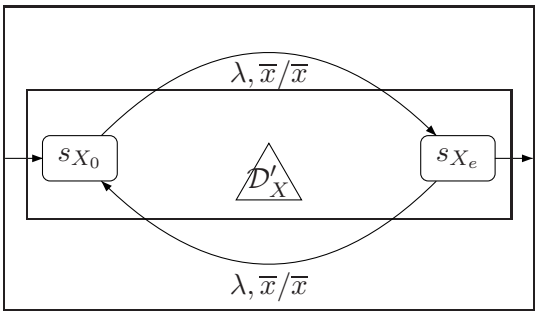
Hierbei wird über den gemeinsamen Startzustand s_{XY_0} von $\mathcal{D}'_{X|Y}$ durch einen λ-Übergang in die Startzustände von $\mathcal{D}'_X$ und $\mathcal{D}'_Y$ verwiesen. Es existiert ein gemeinsamer Endzustand s_{XY_e}. $\overline{x}$ sind dabei Kellerzeichen der Form $\overline{(\alpha, X, x)}$.

2. Für den regulären Ausdruck XY wird der zusammengesetzte Kellerautomat $\mathcal{D}'_{XY}$ konstruiert.



Hierbei wird der Endzustand s_{X_e} des Kellerautomaten $\mathcal{D}'_X$ zum Startzustand s_{Y_0} des Automaten $\mathcal{D}'_Y$.

3. Für den regulären Ausdruck X^* wird der Kellerautomat $\mathcal{D}'_{X^*}$ konstruiert.



Hierbei wird ein λ-Übergang vom Startzustand s_{X_0} in den Endzustand s_{X_e} erzeugt und umgekehrt.

Ein Fall muss gesondert betrachtet werden. Es sei X ein Nichtterminalzeichen, für das ein Kellerautomat $\mathcal{D}'_X$ konstruiert werden soll. Während der Konstruktion des Kellerautomaten $\mathcal{D}'_X$ tritt das Nichtterminalzeichen X nochmal auf. Da die balancierte Grammatik indiziert ist, kann es nur innerhalb der Konstruktion des Kellerautomaten $\mathcal{D}'_X$ auftreten und sonst nicht. Das Nichtterminalzeichen X ist somit ein selbsteinbettendes Nichtterminalzeichen. Der Automat, der für X nun ein weiteres mal konstruiert werden müsste, ist also der selbe Automat $\mathcal{K}_X$. Er wird jedoch nicht erneut konstruiert, da die Konstruktion wegen der Selbsteinbettung wieder und wieder durchgeführt werden müsste. Für alle Konstruktionen (siehe oben Punkt 1 bis 3), die für das Inhaltsmodell durchgeführt werden müssen, muss lediglich der Startzustand und der Endzustand eines Automaten bekannt sein. Beide sind von dem Kellerautomaten $\mathcal{D}'_X$ bekannt. Somit kann dieser für die Konstruktion verwendet werden.

Es bleibt zu zeigen, dass alle λ-Übergänge ersetzt werden können, um den nichtdeterministischen Kellerautomaten einfacher in einem deterministischen Kellerautomaten umzuwandeln.
Es seien $s_k, s_j \in S_X$ zwei Zustände eines Automaten $\mathcal{D}'_X$. Wenn eine Überführung

$$(s_k, \lambda, \overline{z}) \longrightarrow (s_j, \overline{z}), \overline{z} \in G_X,$$

existiert, werden für alle Überführungen der Form

$$(s_j, a, \overline{z}) \longrightarrow (s_n, \overline{az}), a \in A_X, \overline{a}, \overline{z} \in G_X, s_n \in S_X,$$

und

$$(s_j, \overline{a}, \overline{a}) \longrightarrow (s_n, \lambda)$$

Überführungen

$$(s_k, a, \overline{z}) \longrightarrow (s_n, \overline{az})$$

oder

$$(s_k, \overline{a}, \overline{a}) \longrightarrow (s_n, \lambda)$$

erzeugt.

Für jede Produktion der Form

$$X \longrightarrow xm\overline{x}, X \in N, x, \overline{x} \in T, m \in Reg(N)$$

wurde ein erkennender nichttdeterministischer Kellerautomat konstruiert. Das Inhaltsmodell wurde aus den einzelnen Automaten der Nichtterminalzeichen nach einer Abwandlung der Thompson Methode zusammengesetzt. Wenn keine Selbsteinbettung auftritt, ist klar, dass $L(G) = L(\mathcal{D}')$. Bei Selbsteinbettung

$$X \Rightarrow^+ uXv$$

erkennt der Teilautomat $\mathcal{D}'_X$ von Automat $\mathcal{D}'$, die Sequenzen u, bzw. v durch Schleifen, indem er wieder in sich selbst verweist.
Wir nehmen an, es gilt $L(G) \neq L(\mathcal{D})$. Für Grammatiken, die keine selbsteinbettenden Nichtterminalzeichen besitzen, ist aufgrund der Konstruktion klar, dass die Sprachen, die sie erzeugen, gleich sind. Damit $L(G) \neq L(\mathcal{D})$ gilt, müssten selbsteinbettende Nichtterminalzeichen in der Grammatik enthalten sein. Wenn der Kellerautomat $\mathcal{D}'$ gleich viele Sequenzen von u und v durchläuft, erkennt er die Sprache, die das selbsteinbettende Nichtterminalzeichen X erzeugt. Wenn dies nicht der Fall ist, muss ein anderer Teilautomat für ein anderes Nichtterminalzeichen Sequenzen erzeugen, die u oder v zu einem Dyckprime ergänzen, da der Keller geleert werden muss.
Es muss also ein zweites selbsteinbettendes Nichtterminalzeichen Y geben mit

$$Y \Rightarrow^+ u'Yv'$$

und gelten $u'v \in D_T$ oder $uv' \in D_T$. $\mathcal{D}'$ kellert für jedes Zeichen $a \in A$ ein Zeichen ein, dass eindeutig die Produktion, aus der es entstanden ist, identifiziert. Weiter ist nicht mehr jedes Zeichen $a \in A$ für alle Kellerzeichen definiert, sondern nur noch, wenn das Kellerzeichen auf dem Keller liegt, welches in der Produktion steht, in der a enthalten ist.
Der Kellerautomat kann diese Sequenz nicht erkennen, da im Kellerzeichen die Information enthalten ist, dass jedes Zeichen aus u von der Produktion X entstanden ist und jedes Zeichen u' von dem Nichtterminalzeichen Y. Wenn $\mathcal{D}'$ nicht gleich viele Schleifen von u durchläuft wie von v, dann kann $\mathcal{D}'$ seinen Keller nicht mehr leeren. Deshalb müssen gleich viele Schleifen von u und v durchlaufen werden. Daraus folgt ein Widerspruch zu der Annahme.

Der nichtdeterministische Kellerautomat $\mathcal{D}' = (S', T', G', \triangle, s'_0, \delta', F')$ ist nun erzeugt. Es fehlt noch die Umwandlung in einen deterministischen Kellerautomaten. Wichtig zu erkennen ist, dass der Kellerautomat $\mathcal{D}'$ zwar nichtdeterministisch ist, jedoch für ein Eingabezeichen klar ist, wenn ein Zeichen $a \in A$ in der Eingabe steht, wird ein Zeichen eingekellert. Darüber hinaus wird, wenn ein Zeichen $\overline{a} \in \overline{A}$ in der Eingabe steht, ein Zeichen aus dem Keller entfernt.
Nun werden zwei verschiedene Konstruktionen durchgeführt. Es wird die Potenzmenge der Zustände $2^{S'}$ gebildet und ähnlich wie bei endlichen Automaten die Umwandlung der Zustandsübergänge deterministisch gemacht.

Weiter wird, wenn in mehrere Zustände verwiesen wird, eine Kombination aus allen Kellerzeichen, die im nichtdeterministischen Fall in den Keller gelegt würden, in den Keller gelegt. Also wird auch die Potenzmenge der Kellerzeichen gebildet $2^{G'}$. Der deterministische Kellerautomat $\mathcal{D} = (S, T, G, \triangle, s_0, \delta, F)$ wird folgermaßen aus $\mathcal{D}'$ konstruiert:

- $S = 2^{S'} \cup \{m_{S''} \mid S'' \in 2^{S'}\}$ ist die Menge der Zustände.
- $T = T'$ ist die Menge der Eingabesymbole.
- $G = 2^{G'}$ ist die Menge der Kellersymbole.
- $s_0 = s'_0$ ist der Startzustand.
- F ist die Menge der Endzustände. Es sei

  $$X \longrightarrow xm\overline{x}, m \in Reg(N), X \in N, x, \overline{x} \in T$$

  eine Produktion aus der Grammatik G', wobei X ein Startzeichen der Grammatik ist. Dann ist

  $$s_{(e,(X,x))}$$

  ein Endzustand des erkennenden Automaten.
  Alle Zustände $S'', s_{(e,(X,x))} \in S'', S'' \in S$ sind Endzustände.

- δ ist die Überführungsfunktion. Im Folgenden gilt: $R, R' \in S, a, \overline{a} \in T, C, C' \in G$.

  1. Es werden Überführungen

     $$\begin{aligned} \delta(R, a, C) &= (R', C'C) \\ R' &= \{s' \mid \quad s \in R \wedge \overline{(\alpha, X, x)} \in C \wedge s \in \alpha \wedge \\ &\qquad \delta(s, a, \overline{(\alpha, X, x)}) = (s', g'), s' \in S', g' \in G'\} \\ C' &= \{g' \mid \quad s \in R \wedge \overline{(\alpha, X, x)} \in C \wedge s \in \alpha \wedge \\ &\qquad \delta(s, a, \overline{(\alpha, X, x)}) = (s', g'), s' \in S', g' \in G'\} \end{aligned}$$

     erzeugt.
     Die Überführungen definieren das Verhalten des deterministischen Kellerautomaten für Eingabezeichen $a \in A$. Der Kellerautomat $\mathcal{D}$ reagiert darauf, indem er in die Menge der Zustände verweist, in die der nichtdeterministische Kellerautomat $\mathcal{D}'$ für dieses Eingabezeichen verweist. Zusätzlich werden alle möglichen Kellerzeichen, die der Automat $\mathcal{D}'$ in den Keller legt, als Vereinigung in den Keller gelegt.
  2. Es werden Überführungen

$$\begin{aligned}\delta(R,\overline{a},C) &= (m_{R'},\lambda)\\ R' = \{s' \mid\ & s \in R \wedge \overline{(\alpha,X,a)} \in C \wedge s \in \alpha \wedge\\ & \delta(s,\overline{a},\overline{(\alpha,X,a)}) = (s',\lambda), s' \in S'\}\end{aligned}$$

erzeugt.
Die Überführungen definieren das Verhalten des deterministischen Kellerautomaten $\mathcal{D}$ für Eingabezeichen $\overline{a} \in \overline{A}$. Der Kellerautomat $\mathcal{D}$ überprüft dabei, ob ein Kellerzeichen, in dessen Tupel $\overline{(\alpha,X,a)}$ das Zeichen a enthalten ist, an oberster Stelle im Keller liegt. Wenn dies der Fall ist, verweist er in einen Zwischenzustand, in dem alle Zustände, in die der nichtdeterministische Kellerautomat $\mathcal{D}'$ verweist, enthalten sind.

3. Es werden Überführungen

$$\begin{aligned}&\delta(m_{R'},\lambda,C) = (R',C')\\ &C' = \{\overline{(\alpha,X,x)} \mid s \in R' \wedge s \in \alpha \wedge \overline{(\alpha,X,x)} \in C\}\end{aligned}$$

erzeugt.
Diese Überführungen definieren das Verhalten des deterministischen Kellerautomaten $\mathcal{D}$ für die Zwischenzustände $m_{T'}$. In diese Zustände wird verwiesen, wenn der Automat $\mathcal{D}$ ein Zeichen aus dem Keller entfernt. Durch die Zwischenzustände wird das oberste Kellerzeichen aktualisiert. In einem Kellerzeichen des deterministischen Kellerautomaten sind alle Kellerzeichen vereinigt, die der nichtdeterministsiche Kellerautomat für ein einziges Eingabezeichen in den Keller legt. Der deterministische Kellerautomat durchläuft alle Wege des nichtdeterministischen Kellerautomaten gleichzeitig. Wenn der deterministische Kellerautomat erkennt, dass ein Weg falsch war, wird beim deterministischen Kellerautomat $\mathcal{D}$ durch die Zwischenzustände das Kellerzeichen aktualisiert. Somit kodiert der Automat $\mathcal{D}$, dass die Kellerzeichen nur durch die bestimmte Menge aus den ursprünglichen Produktionen entstanden sind.

Es ist zu beachten, dass sowohl die Keller des nichtdeterministischen Kellerautomaten, als auch der Keller des deterministischen Kellerautomaten immer die gleiche Höhe haben, da für ein Zeichen $a \in A$ genau ein Zeichen in den Keller geschrieben wird und für ein Zeichen $\overline{a}$ immer ein Zeichen aus dem Keller entfernt wird.
Es sei $\pi_i, i \in \{1,2,3\}$ die Projektion auf die i-te Komponente einer Konfiguration. Es seien $\mathcal{D} = (S,T,G,\triangle,s_0',\delta,F)$ und $\mathcal{D}' = (S',T',G',\triangle,s_0',\delta',F')$ die Kellerautomaten aus der obigen Konstruktion.
Die Gleichheit $L(\mathcal{D}) = L(\mathcal{D}')$ wird induktiv über die Länge der Eingabe $w \in T^*$ gezeigt. Zunächst wird behauptet, dass

$$\Delta(s_0',w,\triangle) = (s,\lambda,\overline{z}\overline{w''}) \Leftrightarrow \Delta'(s_0',w,\triangle) = (s,\lambda,\overline{z}\overline{w'}), \overline{z} \in \overline{G}, w', w''\overline{G}^*.$$

Es wird die Betrachtungsweise gewählt, dass ein nichtdeterministischer Kellerautomat unter Umständen mehrere Zustandsübergänge gleichzeitig durchläuft. Dabei ist im nichtdeterministischen Kellerautomat der Kellerinhalt $\overline{zw'}$ pro Zeichen $\overline{a} \in \overline{zw'}$ die Vereinigung aus allen Kellerzeichen der möglichen Wege, die der Automat durchläuft.
Die Behauptung ist trivial für $|w|$, da in diesem Fall $w = \lambda$ und $\Delta(s'_0, \lambda, \triangle) = (s'_0, \lambda, \triangle) = \Delta'(s'_0, w, \triangle)$. Es sei nun angenommen, die Behauptung gelte für $|w| \geq i$. Für das Wort wa der Länge $i+1$ und $a \in A$ gilt

$$\Delta(s'_0, wa, \triangle) = \delta(\pi_1(\Delta(s'_0, w, \triangle)), a, \pi_3(\Delta(s'_0, w, \triangle))).$$

Aus der Annahme folgt, dass

$$\Delta(s'_0, w, \triangle) = (s, \lambda, \overline{z}\overline{w''}) \Leftrightarrow \Delta'(s'_0, w, \triangle) = (s, \lambda, \overline{z}\overline{w'}).$$

Aus der Definition folgt weiter, dass

$$\delta(s, a, \overline{z}) = (s', \lambda, \overline{yz}\overline{w''}) \Leftrightarrow \delta'(s, a, \overline{z}) =$$

$$(\bigcup_{s'' \in s} \pi_1(\delta'(s'', a, \overline{z}'')), \lambda, \bigcup_{s'' \in s} \pi_3(\delta'(s'', a, \overline{z}''))) = (s', \lambda, \overline{y}\overline{w'''}), \overline{z}'' \in \overline{z}.$$

Also gilt auch

$$\Delta(s'_0, wa, \triangle) = (s, \lambda, \overline{y}\overline{w''}) \Leftrightarrow \Delta'(s'_0, wa, \triangle) = (s, \lambda, \overline{y}\overline{w'''}).$$

Es sei nun angenommen, die Behauptung gelte für $|w| \geq i$. Für das Wort $w\overline{a}$ der Länge $i+1$ und $\overline{a} \in \overline{A}$ gilt

$$\Delta(s'_0, w\overline{a}, \overline{ay_1}\overline{w'}) = \delta(\pi_1(\Delta(s'_0, w, \triangle)), \overline{a}, \pi_3(\Delta(s'_0, w, \triangle))).$$

Aus der Annahme folgt, dass

$$\Delta(s'_0, w, \triangle) = (s, \lambda, \overline{ay_1}\overline{w''}) \Leftrightarrow \Delta'(s'_0, w, \triangle) = (s, \lambda, \overline{ay_1}'\overline{w'}).$$

Aus der Definition folgt weiter, dass

$$\delta(s, \overline{a}, \overline{a}) = (m_{s'}, \lambda, \overline{y_1 z}\overline{w'}) \longrightarrow \delta(m_{s'}, \lambda, \overline{y_1}) = (s', \lambda, \overline{y}\overline{w'})$$

$$\Leftrightarrow \delta'(s, \overline{a}, \overline{a}) = (\bigcup_{s'' \in s} \pi_1(\delta'(s'', \overline{a}, \overline{a})), \lambda, \bigcup_{s'' \in s} \pi_3(\delta'(s'', \overline{a}, \overline{a}))).$$

Hierbei ist erneut zu beachten, dass in den Kellerzeichen die Zustände eindeutig den Kellerzeichen zugeordnet sind.
Also gilt auch

$$\Delta(s'_0, w\overline{a}, \triangle) = (s, \lambda, \overline{y}\overline{w''}) \Leftrightarrow \Delta'(s'_0, w\overline{a}, \triangle) = (s, \lambda, \overline{y}\overline{w'''}).$$

Somit ist die Gleichheit $L(\mathcal{D}) = L(\mathcal{D}')$ gezeigt.

□

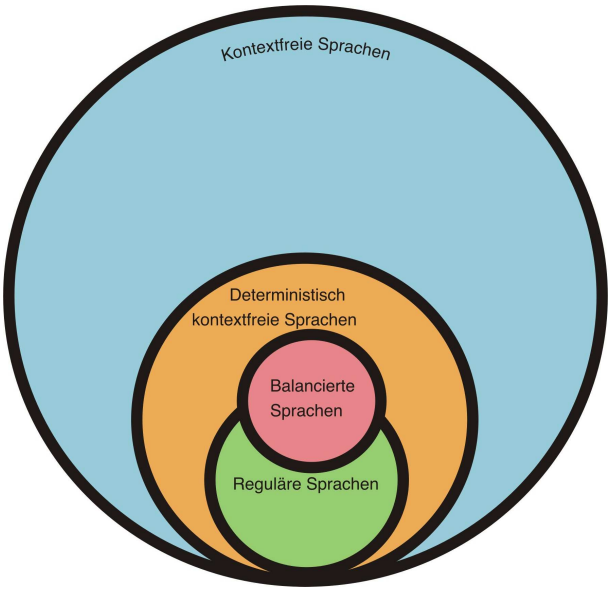


Abbildung 4.1: Genaue Einordnung in die Chomsky Hierarchie.

Satz 4.2.10 XML Grammatiken gehören zu den $LL(k)$ Grammatiken.

Beweis. Es wird gezeigt, dass dieser Grammatiktyp durch einen Kellerautomaten mit einem *1-lookahead* erkannt werden kann.
Gegeben sei eine XML Grammatik $G = (N, T, S, P)$. Für k wird 1 gewählt.

Abgesehen von der Einschränkung, dass jede Produktion in P die Form $X \longrightarrow xm\overline{x}, X \in N, m \in Reg(N), x, \overline{x} \in T$, haben muss, sind drei Eigenschaften der XML Grammatiken entscheidend:

1. Es darf keine zwei Produktionen geben, die mit dem selben Terminalzeichen auf der rechten Seite der Produktion beginnen. Das bedeutet es kann eindeutig, anhand des nächsten Eingabesymbols, entschieden werden, welche Produktion benutzt werden muss.

2. Eine weitere Eigenschaft ist, dass die Inhaltsmodelle aller Produktionen aus einem 1-lookahead deterministischem regulären Ausdruck bestehen müssen.

3. Zusätzlich ist jede Produktion der Grammatik G in doppelter Greibach Normalform.

Die Menge der Terminalzeichen a, die direkt hinter dem Nichtterminalzeichen A in einer Ableitung auftreten können $S \Rightarrow^* \alpha Aa\beta, \alpha, \beta \in (N \cup T)^*, S \in N, a \in T$, werden mit $FOLLOW(A)$ bezeichnet.

Demgegenüber stehen die Eigenschaften von $LL(1)$ Grammatiken [ASU99a] [ASU99b]:
Es seien $A \longrightarrow \alpha|\beta$ mit $\alpha, \beta \in (T \cup N)^*$ zwei unterschiedliche Produktionen in P:

1. Für kein Terminalzeichen a können Zeichenketten aus α und β abgeleitet werden, die beide mit a beginnen.

2. Nur einer der beiden Ausdrücke α oder β kann zur leeren Zeichenkette abgeleitet werden.

3. Wenn $\beta \Rightarrow^* \lambda$, dann darf nicht α in eine Zeichenkette beginnend mit $a \in FOLLOW(A)$ ableiten.

Nach der Umwandlung der Grammatik G nach Satz 4.2.8, wonach alle rechten Seiten der Produktionen aus einer endlichen Menge bestehen, gibt es für ein Nichtterminal $X \in N$ folgende Möglichkeiten von Produktionen:

- $X \longrightarrow xA_1A_2 \ldots A_n\overline{x}$
- $X \longrightarrow A_1A_2 \ldots A_n$
- $X \longrightarrow xA_1A_2 \ldots A_n\overline{x}X_i$ und $X_i \longrightarrow \lambda$
- $X \longrightarrow xA_1A_2 \ldots A_n\overline{x}$ und $X_i \longrightarrow \lambda$.

für $x \in A, \overline{x} \in \overline{A}, A_1, A_2, \ldots, A_n \in N$.
Für den ersten Typ von Produktionen $X \longrightarrow xA_1A_2 \ldots A_n\overline{x}$ ist klar, welche Produktion X gewählt wird, da für dieses Nichtterminal X nur genau diese eine Produktion existiert. Für den zweiten Typ von Produktionen $X \longrightarrow A_1A_2 \ldots A_n$ gilt selbiges.
Der dritte und vierte Typ $X \longrightarrow xA_1A_2 \ldots A_n\overline{x}X$ und $X \longrightarrow \lambda$ bzw. $X \longrightarrow xA_1A_2 \ldots A_n\overline{x}$ und $X \longrightarrow \lambda$ gehorcht der ersten und der zweiten Regel für $LL(1)$ Grammatiken, da die erste Variante von X auf jeden Fall mit dem Terminalzeichen x beginnt und die zweite Variante ein λ ist. Die dritte Regel wird durch die Forderung eingehalten, dass der reguläre Ausdruck 1-deterministisch sein muss und dass es keine zwei Nichtterminale geben darf, deren Produktionen mit einem gleichen Terminalzeichen x beginnen. Innerhalb der Produktion $Y \longrightarrow yX_1X_2 \ldots X_n\overline{y}$ ist eindeutig, dass nach X_{i-1} X_i das einzige Nichtterminal mit einem $x \in A$ auf der rechten Seite der Produktion ist. Somit ergibt sich, dass $x \notin FOLLOW(X_i)$, falls $X_i \Rightarrow^* \lambda$. □

Satz 4.2.11 Die einfachen balancierten Grammatiken gehören zu den $LL(k)$ Grammatiken.

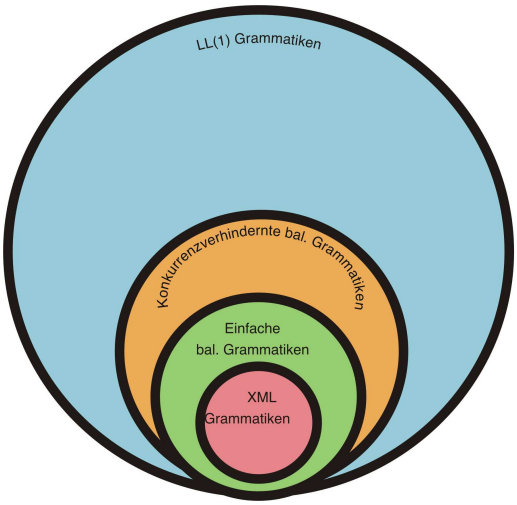


Abbildung 4.2: XML-artige Grammatiken und $LL(1)$ Grammatiken.

Beweis. Da bei den einfachen balancierten Grammatiken keine konkurrierenden Nichtterminale innerhalb einer Produktion auftreten dürfen, ist auch hierbei für ein Terminalzeichen x innerhalb einer Produktion eindeutig klar, von welchem Nichtterminalzeichen dieses erzeugt wird. □

Satz 4.2.12 Die konkurrenzverhindernden balancierten Grammatiken gehören zu den $LL(k)$ Grammatiken.

Beweis. Die Bedingung der konkurrenzverhindernden balancierten Grammatiken war, dass für Nichtterminalsequenzen U, V, W und zwei konkurrierende Nichtterminale $A, B \in N$ nicht UAV und UBW innerhalb einer Produktion erzeugt werden darf.
Es sei x ein Terminalzeichen, welches sowohl von A als auch von B erzeugt wird. Da nach einer Abarbeitung von einer beliebigen Sequenz U nicht A und B folgen dürfen und für den regulären Ausdruck des Inhaltsmodells die 1-lookahead deterministisch Eigenschaft gilt, ist eindeutig klar, welche Produktion für welches Nichtterminalzeichen angewendet werden muss. □

Satz 4.2.13 Jede Sprache L, die von einem RegDyck Automaten erkannt wird, ist deterministisch kontextfrei.

Beweis. Der Durchschnitt einer regulären Sprache und einer deterministisch kontextfreien Sprache ist deterministisch kontextfrei. □

4.3 Hierarchie der XML-artigen Grammatiken

In diesem Abschnitt wird eine Hierarchie der XML-artigen Sprachfamilien gezeigt.
Es ist klar, dass die XML Sprachen, die einfachen balancierten Sprachen, die konkurrenzverhindernden balancierten Sprachen und die RegDyck Sprachen Teilmengen der balancierten Sprachen sind. Die Inhaltsmodelle der Produktionen der Untertypengrammatiken sind mit Einschränkungen in Bezug zu den Produktionen in balancierten Grammatiken belegt. Es wird untersucht, ob die Sprachen der Untertypen echte Teilmengen der balancierten Sprachen sind.

Satz 4.3.14 Es sei $\mathscr{L}$ die Familie der balancierten Sprachen und $\mathscr{L}_1$ sei die Familie der konkurrenzverhindernden Sprachen, dann gilt $\mathscr{L}_1 \subset \mathscr{L}$.

Beweis. Es sei

$$L((x(a(c\overline{c})?\overline{a})^*(a(d\overline{d})\overline{a})^*\overline{x}))$$

die Sprache, die von folgenden Produktionen einer balancierten Grammatik $G = (N, T, \{S\}, P)$ erzeugt wird:

$$\begin{aligned} P = \{ \quad & S \longrightarrow xA^*A_1^*\overline{x} \\ & A \longrightarrow aC?\overline{a} \\ & A_1 \longrightarrow aD\overline{a} \\ & C \longrightarrow c\overline{c} \\ & D \longrightarrow d\overline{d}\} \end{aligned}$$

Es lässt sich keine Grammatik für die Sprache finden, welche in der Produktion für X keine zwei konkurrierenden Nichtterminale enthält, da die beiden Dycksprachen

$$\{(a(c\overline{c})?\overline{a})^*\} \text{ und}$$
$$\{(a(d\overline{d})\overline{a})^*\}$$

zwischen x und $\overline{x}$ erzeugt werden.
Um die beiden Dycksprachen zu erzeugen, werden mindestens zwei unterschiedliche Nichtterminale benötigt, die beide mit einem Stern gekennzeichnet sind.

Angenommen, die Grammatik enthält nur ein Nichtterminalzeichen, dann wird davon die Sprache $\{x(a(c\overline{c})?|(d\overline{d})\overline{a})^*(a(c\overline{c})?|(d\overline{d})\overline{a})^*\overline{x}\}$ erzeugt. Diese Sprache ist jedoch nicht die Ausgangssprache.

Die beiden konkurrierenden Nichtterminale, die die Dycksprachen erzeugen, stehen hintereinander in der Produktion $X \longrightarrow xA^*A_1^*\overline{x}$. Die Nichtterminale können beide auftreten oder komplett weggelassen werden.

Es folgt, dass diese Grammatik keine konkurrenzverhindernde balancierte Grammatik ist, da dadurch Sequenzen UAW und UA_1V, mit $U, V, W \in N^*$, erzeugt werden können. Solche Inhaltsmodelle sind in den konkurrenzverhindernden balancierten Grammatiken nicht erlaubt, somit ist die Sprache keine konkurrenzverhindernde balancierte Sprache. Es folgt, dass die Familie der konkurrenzverhindernden balancierten Sprachen eine echte Untermenge der Familie der balancierten Sprachen ist.

□

Satz 4.3.15 Es sei $\mathscr{L}$ die Familie der konkurrenzverhindernden balancierten Sprachen und $\mathscr{L}_1$ die Familie der einfachen balancierten Sprachen, dann gilt $\mathscr{L}_1 \subset \mathscr{L}$.

Beweis. Es sei

$$L((x(a(c\overline{c})?\overline{a})(a(d\overline{d})\overline{a})^*\overline{x}))$$

die Sprache, die von folgender konkurrenzverhindernden Grammatik $G = (N, T, S, P)$ erzeugt wird:

$$\begin{aligned} P = \{ \quad & S \longrightarrow xAA_1^*\overline{x} \\ & A \longrightarrow aC?\overline{a} \\ & A_1 \longrightarrow aD\overline{a} \\ & C \longrightarrow c\overline{c} \\ & D \longrightarrow d\overline{d}\} \end{aligned}$$

Es ist wieder notwendig zwei konkurrierende Nichtterminale in einer Grammatik zu erzeugen.
Konkurrierende Nichtterminale innerhalb eines Inhaltsmodelles sind in den einfachen balancierten Grammatiken nicht erlaubt. Somit ist die Sprache keine einfache balancierte Sprache und es folgt, dass die Familie der einfachen balancierten Sprachen eine echte Untermenge der Familie der konkurrenzverhindernden balancierten Sprachen ist. □

Satz 4.3.16 Es sei $\mathscr{L}$ die Familie der einfachen balancierten Sprachen und $\mathscr{L}_1$ die Familie der XML Sprachen, dann gilt $\mathscr{L}_1 \subset \mathscr{L}$.

Beweis. Es sei

$$L((x(a(c\overline{c})?\overline{a})(d(a(e\overline{e})\overline{a})\overline{d})^*\overline{x}))$$

die Sprache die von folgender Grammatik $G = (N, T, S, P)$ erzeugt wird:

$$\begin{aligned} P = \{ \quad & S \longrightarrow xAD^*\overline{x} \\ & A \longrightarrow aC?\overline{a} \\ & C \longrightarrow c\overline{c} \\ & D \longrightarrow dA_1\overline{d} \\ & A_1 \longrightarrow aE\overline{a} \\ & E \longrightarrow e\overline{e}\} \end{aligned}$$

Es müssen wieder zwei konkurrierende Nichtterminale erzeugt werden.
Konkurrierende Nichtterminale sind in den XML Grammatiken jedoch nicht erlaubt, daher kann die angegebene Sprache nicht durch eine XML Grammatik erzeugt werden.
Es folgt, dass die Familie der XML Sprachen eine echte Untermenge der Familie der einfachen balancierten Sprachen ist. □

Satz 4.3.17 Es sei $\mathscr{L}$ die Familie der balancierten Sprachen und $\mathscr{L}_1$ die Familie der RegDyck Sprachen, dann gilt $\mathscr{L}_1 \subset \mathscr{L}$.

Beweis. Es sei

$$L((x(a^n(c\overline{c})?\overline{a}^n)^*(a^m(d\overline{d})\overline{a}^m)^*\overline{x})), m, n \in \mathbb{N}$$

die Sprache, die von folgender balancierter Grammatik $G = (N, T, \{S\}, P)$ erzeugt wird:

$$\begin{aligned} P = \{ \quad & S \longrightarrow xA^*A_1^*\overline{x} \\ & A \longrightarrow a(A|C?)\overline{a} \\ & A_1 \longrightarrow a(A_1|D)\overline{a} \\ & C \longrightarrow c\overline{c} \\ & D \longrightarrow d\overline{d}\} \end{aligned}$$

Die Definition der RegDyck Sprachen erlaubt allerdings nicht die Wörter $\{(xa^n(c\overline{c})?\overline{a}^n a^m(d\overline{d})\overline{a}^m\overline{x}) \mid m, n \in \mathbb{N}\}$ in einer Sprache, ohne dass dabei auch die Wörter

$$\{(xa^k a^n(c\overline{c})?\overline{a}^n a^m(d\overline{d})\overline{a}^m\overline{a}^k\overline{x}) \mid m, n, k \in \mathbb{N}\}$$

in der Sprache enthalten sind.
Es folgt somit, dass es balancierte Sprachen gibt, die keine RegDyck Sprachen sind. □

Korollar 4.3.2 Die in Satz 4.3.17 angegebene Sprache ist auch keine konkurrenzverhindernde balancierte Sprache, da die Sprache lediglich eine Abwandlung der Sprache aus Satz 4.3.14 ist. Es gibt somit balancierte Sprachen, die keine konkurrenzverhindernde balancierte Sprachen und keine RegDyck Sprachen sind.

Satz 4.3.18 Es sei $\mathscr{L}_1$ die Familie der konkurrenzverhindernden balancierten Sprachen und $\mathscr{L}$ die Familie der RegDyck Sprachen, dann gilt $\mathscr{L}_1 \not\subset \mathscr{L}$.

Beweis. Es sei

$$L((x(a^n(c\overline{c})?\overline{a}^n)(a^m(d\overline{d})\overline{a}^m)^*\overline{x})), m, n \in \mathbb{N}$$

die Sprache, die von folgender balancierter Grammatik $G = (N, T, S, P)$ erzeugt wird:

$$P = \{ \quad \begin{array}{l} S \longrightarrow xAA_1^*\overline{x} \\ A \longrightarrow a(A|C?)\overline{a} \\ A_1 \longrightarrow a(A_1|D)\overline{a} \\ C \longrightarrow c\overline{c} \\ D \longrightarrow d\overline{d}\} \end{array}$$

Die Definition der RegDyck Sprachen erlauben jedoch nicht die Wörter $\{(xa^n(c\overline{c})?\overline{a}^n a^m(d\overline{d})\overline{a}^m\overline{x}) \mid m,n \in \mathbb{N}\}$ in einer Sprache, ohne dass dabei auch die Wörter

$$\{(xa^k a^n(c\overline{c})?\overline{a}^n a^m(d\overline{d})\overline{a}^m\overline{a}^k\overline{x}) \mid m,n,k \in \mathbb{N}\}$$

in der Sprache enthalten sind.
Es folgt somit, dass es konkurrenzverhindernde balancierte Sprachen gibt, die keine RegDyck Sprachen sind. □

Korollar 4.3.3 Die in Satz 4.3.18 angegebene Sprache ist auch keine einfache balancierte Sprache, da die Sprache lediglich eine Abwandlung der Sprache aus Satz 4.3.15 ist. Es gibt also balancierte Sprachen, die keine einfachen balancierten Sprachen und keine RegDyck Sprachen sind.

Satz 4.3.19 Es sei $\mathscr{L}_1$ die Familie der RegDyck Sprachen und $\mathscr{L}$ die Familie der konkurrenzverhindernden Sprachen, dann gilt $\mathscr{L}_1 \not\subseteq \mathscr{L}$.

Beweis. Es sei $G = (N, T, \{S\}, P)$,

$$P = \{ \quad \begin{array}{l} S \longrightarrow xA^*A_1^*\overline{x} \\ A \longrightarrow a\overline{a} \\ A_1 \longrightarrow ad\overline{d}\overline{a}\}, \end{array}$$

eine RegDyck Grammatik, die die Sprache

$$L((x(a\overline{a})^n(ad\overline{d}\overline{a})^m\overline{x})), m,n \in \mathbb{N}$$

erzeugt.
Es existiert keine konkurrenzverhindernde balancierte Grammatik, die diese Sprache erzeugen kann, da im Inhaltsmodell der zu konstruierenden Startproduktion zwei konkurrierende Nichtterminale erzeugt werden müssten und es für diese sowohl eine Sequenz $UAW, U, W \in N^*$, als auch eine Sequenz $UA_1V, V \in N^*$ geben kann. □

Korollar 4.3.4 Es existieren RegDyck Sprachen, die keine einfachen balancierten Sprachen und keine XML Sprachen sind. Also sind die RegDyck Sprachen keine Teilmenge dieser Sprachen.

Aus dem Beweis von Satz 4.3.19 wird klar, dass es erstens auch einfache balancierte Sprachen gibt, die keine RegDyck Sprachen und XML Sprachen sind und dass es XML Sprachen gibt, die keine RegDyck Sprachen sind.

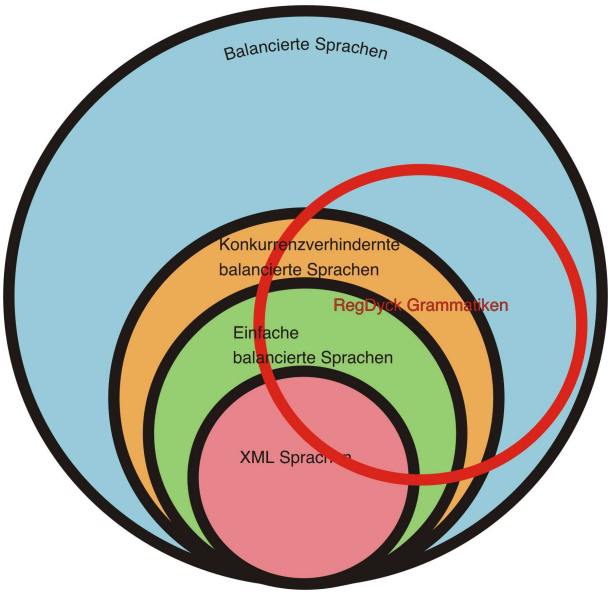


Abbildung 4.3: Hierarchie der XML-artigen Grammatiken.

Zweitens gibt es auch RegDyckSprachen, die keine einfachen balancierten Sprachen und keine XML Sprachen sind.

4.4 Parser Generierung

Dieses Unterkapitel beschäftigt sich mit den Parsergeneratoren und den Parsern von XML-artigen Grammatiken. In der Praxis werden XML Dokumente oft zusammen mit ihren Beschreibungsmodellen verschickt. Zur Validierung muss zuerst das Beschreibungsmodell gelesen und daraus ein geeigneter Parser konstruiert werden.
Es stellt sich die Frage, wie lange es dauert, um einen Parser zu erzeugen und wie lange der Parser braucht, um ein XML Dokument zu validieren.

4.4.1 Sprachen und Methoden

Kapitel 4.2 hat gezeigt, dass die XML Grammatiken, die einfachen balancierten Grammatiken und die konkurrenzverhindernden balancierten Grammatiken zu den $LL(1)$ Grammatiken gehören.
Für diese Art von Grammatiken gibt es Untersuchungen [SSS88] für die Laufzeiten der Parsergeneratoren und der Parser.

Sprache	**max. Laufz. Parsergenerator**	**max. Laufz. Parser**
XML Grammatiken	$O(n)$	$O(n)$
einfache balancierte Grammatiken	$O(n)$	$O(n)$
konk. bal. Grammatiken	$O(n)$	$O(n)$
RegDyck Grammatiken	?	?
balancierte Grammatiken	?	?

Für die balancierten Grammatiken und die RegDyck Grammatiken müssen die Laufzeiten noch untersucht werden. Zusätzlich wird noch eine weitere Variante für Parser von XML Grammatiken, einfachen balancierten Grammatiken und konkurrenzverhindernden balancierten Grammatiken vorgestellt und untersucht. Teile der neuen Variante können in der Praxis für weiterführende Anwendungen benutzt werden, wodurch sich ein Zeitersparnis ergibt.

4.4.2 Konstruktion von codeterministischen Grammatiken aus balancierten Grammatiken

In [BB02a] wird gezeigt, dass aus einer beliebigen balancierten Grammatik eine codeterministische balancierte Grammatik erzeugt werden kann.
Mit Hilfe der codeterministischen balancierten Grammatik soll ein spezieller Parsingalgorithmus verwendet werden.
Eine balancierte Grammatik (N, T, S, P) wird als codeterministisch bezeichnet, falls gilt, wenn $X \longrightarrow m$ und $X' \longrightarrow m$, dann ist $X = X', X, X' \in N, m \in Reg(N \cup T)$.
Im Folgenden wird die Konstruktion in Bezug auf ihre Komplexität analysiert. Die Ergebnisse sollen später zum Vergleich der Grammatikmodelle

genutzt werden.
Die Größe eines endlichen Automaten wird durch die Anzahl der Zustände multipliziert mit der Anzahl der Eingabesymbole $|S| \cdot |A|$ festgelegt. Die Größe einer Grammatik ist durch die Länge aller Produktionen gegeben $\sum_{p \in P} |p|$.

Um eine balancierte Grammatik in eine codeterministische balancierte Grammatik zu transformieren sind die folgenden Schritte notwendig.

- Es sei $G = (N, T, S_0, P)$ eine balancierte Grammatik. Ausgangspunkt sind die rechten Seiten der Produktionen. Für ein Nichtterminalzeichen X werden alle rechten Seiten der Produktionen für X zu einem regulären Ausdruck R_X zusammengefasst. Es sei m die Größe der Grammatik. Es sei n die Länge des regulären Ausdrucks der Produktion für X. $\mathcal{A}_\mathcal{X} = (S, A, s_0, \delta, F)$ sei ein deterministischer endlicher Automat, der R_X erkennt. Die Umwandlung eines regulären Ausdrucks in einen nichtdeterministischen endlichen Automaten nach der Thompson Methode verlangt maximal vier Zustände für das Konstruieren eines erkennenden Automaten pro Zeichen. Bei der Länge n sind das maximal $4 \cdot n$ Zustände. Dadurch, dass die Potenzmenge der Zustände bei der Umwandlung in einen deterministischen endlichen Automaten konstruiert wird, kann die Umwandlung hierbei im schlimmsten Fall $O(2^{4n})$ Zustände zur Folge haben. Die für das Konstruieren notwendige Zeit ist auch $O(2^{4n})$.
- Für jeden Automat $\mathcal{A}_\mathcal{X}$ wird ein Potenzautomat $\mathcal{A}'_\mathcal{X} = (S', A', s'_0, \delta', F')$ gebildet, welcher folgendermaßen definiert ist:
  1. $S' = 2^{|S|}$ ist die Menge der Zustände.
  2. $A' = 2^{|A|} \setminus \emptyset$ ist die Menge der Eingabesymbole.
  3. $s'_0 = \{s_0\}$ ist der Startzustand.
  4. δ', für $s' \in S'$ und $a' \in A'$
  $$\delta(s', a') = \bigcup_{s \in s', a \in a'} \delta(s, a)$$
  5. $F' = \{s' \in S' \mid s' \cap F \neq \emptyset\}$

Für den Potenzautomat ergibt sich die Größe von

$$2^{2^{4n}} \cdot (2^{|T|} - 1) \in O(2^{2^{4n}}).$$

Nun wird für alle Teilmengen von Nichtterminalzeichen $\alpha \in 2^N, \alpha = \{X_1, \ldots, X_i\}$ folgende Konstruktion durchgeführt:

1. $D_\alpha = L(\mathcal{A}'_{\mathcal{X}_1}) \cap L(\mathcal{A}'_{\mathcal{X}_2}) \cap \ldots \cap L(\mathcal{A}'_{\mathcal{X}_i})$

2. Für alle $\beta \in 2^{(N \setminus \alpha)} - \{\emptyset\}, \beta = \{Y_1, \ldots, Y_j\}$ wird die Sprache $P_{\alpha,\beta} = D_\alpha \cap L(\mathcal{A}'_{\mathcal{Y}_1}) \cap L(\mathcal{A}'_{\mathcal{Y}_2}) \cap \ldots \cap L(\mathcal{A}'_{\mathcal{Y}_i})$ gebildet.
3. $G_\alpha = \bigcup_\beta P_{\alpha,\beta}$
4. $S_\alpha = D_\alpha - G_\alpha$

Die Menge D_α beschreibt für eine Menge von Nichtterminalzeichen den Durchschnitt der erzeugten Wörter aus den Produktionen.
Die Produktionen der entstehenden codeterministischen balancierten Grammatik sind:

$$\alpha \longrightarrow S_\alpha, \text{ für alle } \alpha \in 2^N.$$

Nun sei angenommen die Produktionen der balancierten Grammatik G haben alle die Länge n. Dann existieren m/n Nichtterminale. Die Automaten $\mathcal{A}'_{\mathcal{X}_i}, i \in \mathbb{N}$ haben somit eine maximale Größe von $O(2^{2^{4n}})$. Es seien $S_\mathcal{X}$ und $S_\mathcal{Y}$ die Zustandsmengen der Automaten $\mathcal{X}, \mathcal{Y}$. Die Konstruktion des Durchschnitts dieser Automaten benötigt $O(S_\mathcal{X} \cdot S_\mathcal{Y})$ Zeit. Da alle Teilmengen konstruiert werden müssen und für diese jeweils der Durchschnitt berechnet werden muss, ergibt sich folgende Zeit:

$$\sum_{k=1}^{\lceil \frac{m}{n} \rceil} 2^{2^{4n^{(k-1)\left(\lceil \frac{m}{n} \rceil\right)}}} \in \Omega(\frac{m}{n} \cdot 2^{2^{4n^{\lceil \frac{m}{n} \rceil}}}), \lceil \frac{m}{n} \rceil \geq 3$$

Die Laufzeit des Parsergenerators ist sehr groß. Wenn eine balancierte Grammatik aus einer einzigen Produktion besteht, erreicht der Parser eine Laufzeit von $O(2^{2m})$.
Dazu kommt noch die Umwandlung der deterministischen endlichen Automaten in einen regulären Ausdruck. Es sei k die Anzahl der Zustände eines endlichen Automaten, dann ist die Laufzeit zur Konstruktion eines regulären Ausdrucks $O(k^3)$ [HU79]. Dies muss jedoch nicht mehr in die Laufzeitangabe einfließen, da es durch die O-Notation verschwinden würde.
Es wird noch alleine die Größe der codetermistischen balancierten Grammatik betrachtet. Es werden alle Zwischenberechnungen (die Konstruktion eines deterministischen Automaten aus dem regulären Ausdruck der Nichtterminalzeichen und die Potenzautomaten) außer Acht gelassen und nur die Menge der entstehenden Nichtterminalzeichen der codeterministischen balancierten Grammatik mit der Menge der Nichtterminalzeichen in der ursprünglichen Grammatik verglichen. Die codeterministische balancierte Grammatik könnte maximal $2^{|N|}$ Nichtterminalzeichen besitzen, da die Potenzmenge der Nichtterminalzeichen gebildet wird.
Für die Praxis ergibt die Umwandlung einer balancierten Grammatik in eine codeterministische Grammatik keinen Sinn. Jedoch können alle Sprachen, die von einer balancierten Grammatik erzeugt werden, auch durch eine codeterministische balancierte Grammatik erzeugt werden. So können

Beschreibungsmodelle mit codeterministischen balancierten Grammatiken verlangt werden, ohne dabei die Beschreibungsmächtigkeit einzuschränken.

4.4.3 Konstruktion eines deterministischen Kellerautomaten aus einer balancierten Grammatik

Die Konstruktion eines deterministischen Kellerautomaten $\mathcal{D} = (S, A, G, \triangle, s_0, \delta, F)$ aus einer balancierten Grammatik $G = (N, T, S, P)$ wird in zwei Phasen unterteilt. Die erste Phase besteht aus der Konstruktion eines nichtdeterministischen Kellerautomaten $\mathcal{D}' = (S', A', G', \triangle', s_0', \delta', F')$ aus der balancierten Grammatik G. Die zweite Phase besteht aus der Konstruktion des deterministischen Kellerautomaten $\mathcal{D}$ aus dem nichtdeterministischen Kellerautomaten $\mathcal{D}'$.
Die Größe der balancierten Grammatik ist wie im letzten Unterkapitel durch die Länge aller Produktionen gegeben $\sum_{p \in P} |p|$. Die Größe von G sei n. Zur Konstruktion des nichtdeterministischen Kellerautomaten $\mathcal{D}'$ wurde die abgewandelte Thompson Methode benutzt. Wie schon im letzten Unterkapitel beschrieben, erzeugt diese Methode pro Zeichen höchstens 4 Zustände. Bei der abgwandelten Methode werden lediglich die Kelleroperationen hinzugefügt. Dies erzeugt keinen zusätzlichen Aufwand. Es ergibt sich eine Größe von:

$$|S'| = 4 \cdot n$$

Die Menge der Eingabezeichen ermittelt sich durch:

$$|A'| = \frac{|T|}{2}$$

Die maximale Anzahl der Kellersymbole ist:

$$|G'| = \frac{n}{2}$$

In der Konstruktion vom deterministischen Kellerautomaten $\mathcal{D}$ aus dem nichtdeterministischen Kellerautomaten $\mathcal{D}'$ wird die Potenzmenge der Zustände und die Potenzmenge der Kellersymbole gebildet. Für den deterministischen Kellerautomaten ergibt sich dadurch eine Größe von:

$$S \cdot A \cdot G = 2^{4n} \cdot \frac{|T|}{2} \cdot 2^{\frac{n}{2}}$$

In der O-Notation ist die Laufzeit gleich der Größe des Automaten, da nur die Kellerautomaten erzeugt werden müssen und keine zusätzlichen Konstruktionen durchgeführt werden. Es ergibt sich eine Laufzeit von:

$$O(2^{5n} \cdot \frac{|T|}{2}) = O(2^{5n})$$

4.4.4 Konstruktion eines Parsebaums für XML Grammatiken

In [ML02] wurden spezielle Bäume für das Transformieren von XML Dokumenten in relationale Modelle verwendet. Diese Art von Bäumen wird hier erweitert und die Struktur genau definiert. Es wird beschrieben, wie die Bäume für das Parsen verwendet werden können. Dadurch kann Zeit eingespart werden, weil die Konstruktion dieser Bäume, sowohl für die Parsergenerierung, als auch für die Transformation der XML Dokumente genutzt werden kann.
Der folgende Parser kann für XML Grammatiken, einfache balancierte Grammatiken und konkurrenzverhindernde balancierte Grammatiken verwendet werden.
Beim Überprüfen der Gültigkeit eines Dokumentes ergibt sich das Problem, dass die Beschreibungsmodelle von Dokument zu Dokument variieren. Es kann kein allgemeiner Algorithmus verwendet werden, der eine feste Syntax überprüft, wie beispielsweise bei der Syntaxüberprüfung einer Programmiersprache.
Es wird ein Algorithmus benötigt, der ein Beschreibungsmodell, zum Beispiel eine DTD, einliest und ein Programm erzeugt, welches überprüft, ob das Dokument gültig ist.
Folgende DTD soll als Beispiel dienen:

Beispiel 4.4.20 Die Beispiel DTD *Person* ist folgendermaßen definiert:

<!ELEMENT Person (Vorname*, Nachname, Adresse)>
<!ELEMENT Vorname (♯PCDATA)>
<!ELEMENT Nachname (♯PCDATA)>
<!ELEMENT Adresse (Ort, Strasse?)>
<!ELEMENT Ort(♯PCDATA)>
<!ELEMENT Strasse (♯PCDATA)>

□

Eine DTD ist von der Beschreibungsmächtigkeit durch eine XML Grammatik ausdrückbar und allgemein auch ähnlich aufgebaut wie eine kontextfreie Grammatik.

Beispiel 4.4.21 Die Überführung der DTD aus dem letzten Beispiel ergibt eine XML Grammatik mit folgenden Produktionen:

PERSON	$\longrightarrow$	<Person> VORN* NACHN ADR</Person>
VORN	$\longrightarrow$	<Vorname></Vorname>
NACHN	$\longrightarrow$	<Nachname></Nachname>
ADR	$\longrightarrow$	<Adresse> ORT STRASSE? </Adresse>
ORT	$\longrightarrow$	<Ort></Ort>
STRASSE	$\longrightarrow$	<Strasse></Strasse>

□

Es wird ein Parsebaum aufgebaut und das Dokument wird in Bezug auf den Baum überprüft. Der Baum muss variabel für Wiederholungen sein und die Möglichkeit haben optionale Produktionen zu berücksichtigen.
Der Validierungsalgorithmus muss beim Wurzelelement beginnen und dann eine beliebige Anzahl von Unterelementen anhängen. Wiederholungen und Optionen werden durch eine Kennzeichnung des Knotens behandelt.
Folgende Schritte sind durchzuführen:

- Die Nichtterminalzeichen innerhalb der Produktionen werden indiziert.
- Für jedes Nichtterminalzeichen wird ein Knoten erzeugt. In jedem Knoten muss der Name des Elementes eingetragen werden.
- Für die Wiederholungszahl werden zwei Felder benötigt (Feld *Optional* und Feld *Wiederholung* jeweils mit $True$ und $False$ belegbar).
  Wenn das Nichtterminalzeichen im Eltern-Element des Elementes mit einem Fragezeichen „?“ versehen ist, wird das Feld *Optional* mit $True$ und das Feld *Wiederholung* mit $False$ belegt. Für ein Pluszeichen „+“ wird das Feld *Wiederholung* mit $True$ und das Feld *Optional* mit $False$ gekennzeichnet. Bei einem Stern „*“ wird beides mit $True$ belegt.
- Ein Feld *Wert* muss mit Werten belegbar sein (hier wird der Name des Elementes gespeichert).
- Zuletzt muss die Möglichkeit bestehen, auf verschiedene Elemente zu verweisen. Dazu wird eine Liste *next* aus Zeigern erzeugt.

Implementierung des Baums Ein Algorithmus liest zuerst das Beschreibungssystem ein und baut danach den Baum auf. Der Baum, der für die Grammatik aufgebaut werden muss, wird mit Hilfe von Zeigern implementiert.
Die Elemente bestehen aus einem Feld *Wert* vom Typ *String*, einem Feld *Optional* vom Typ *Boolean*, einem Feld $Wiederholung$ vom Typ *Boolean* und einem Feld mit den Verweisen. Die Verweise bestehen aus einer Liste von Zeigern. Diese Liste verweist auf alle Nachfolger des Knotens.

Beispiel 4.4.22

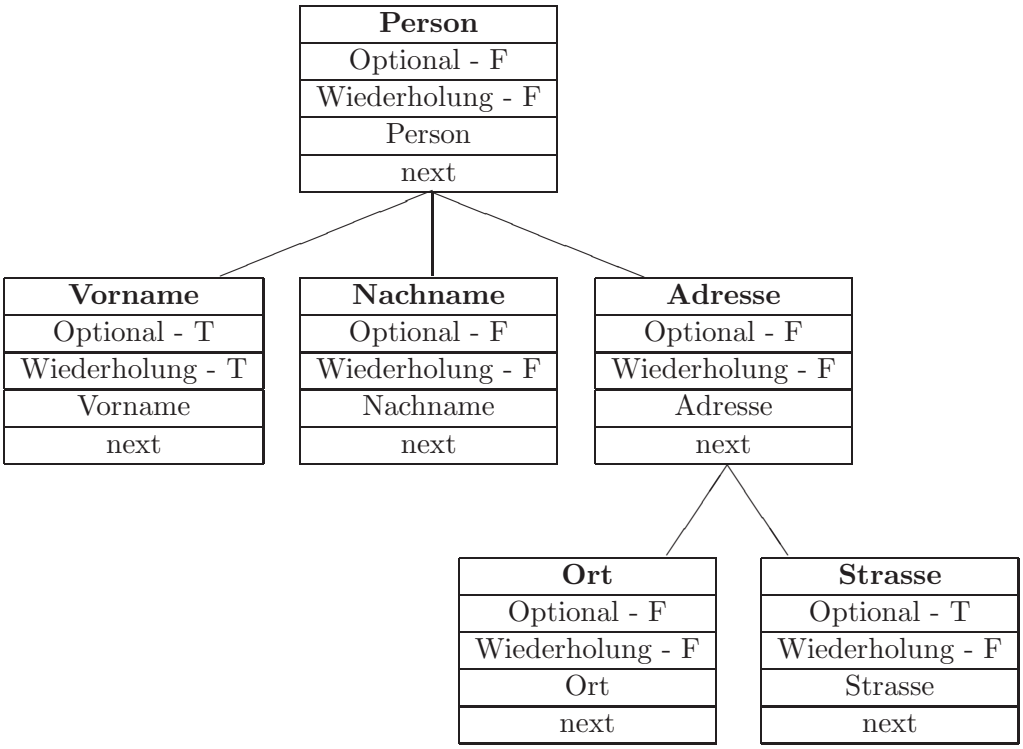


Abbildung 4.4: Beispiel: Parsebaum zur Validierung eines XML Dokuments.

```
Typ der Knoten
TYPE
node = ^nodeelement;
list = ^liste;
liste = Record
    elementlist : node;
    next : list;
end;
nodeelement = Record
    Wert : String;
    Optional : bool;
    Wiederholung : bool;
    nodes : list;
end;
```

Die Inhaltsmodelle der Grammatiken bestehen aus regulären Ausdrücken von Nichtterminalzeichen. Es sei hier nochmal darauf hingewiesen, dass sowohl die XML Grammatiken, als auch die einfachen balancierten Grammatiken und die konkurrenzverhindernden balancierten Grammatiken aus 1-lookahead deterministischen regulären Ausdrücken bestehen. Die Nichtterminalzeichen sind die Wurzeln der Bäume.

Um eine genaue Abfolge der Nachfolgebäume abzubilden, wird eine Liste erzeugt, die nur eine Abarbeitung der Nachfolgebäume in der Reihenfolge des regulären Ausdrucks zulässt.
Die Operatoren des regulären Ausdrucks werden in den Knoten untergebracht. Die Elementdeklaration ermöglicht es, verschiedene Varianten von Unterelementen in der Elementdeklaration anzugeben und diese zu gruppieren:

Beispiel 4.4.23<!ELEMENT((Name | Persnr Lieblingsessen))>

Dieses Element kann wahlweise einen Namen oder eine Personalnummer als Unterelement besitzen. Für jede Gruppe wird ein Knoten erzeugt, dessen *Wert* leer ist. Als Nachfolgebäume werden die Elemente der Gruppe angehängt. Bei Optionen wird den Listenelementen eine Signatur „optional“ gegeben. In der Gültigkeitsanalyse muss entschieden werden, welcher Unterbaum der Prüfung unterzogen werden soll. Auch bei Wiederholungen ganzer Gruppen ist im Knoten der Gruppe das Feld *Wiederholung* mit *true* belegt.

Nun wird für die Generierung des Baums die Komplexität bestimmt. Dabei soll, in Abhängigkeit von der Größe der Produktionen, eine Angabe gemacht werden, wie oft die einzelnen Zeichen durchlaufen werden.
Jedes Nichtterminalzeichen in den Produktionen wird indiziert. Anschließend wird für jedes Nichtterminalzeichen sowie für jede Gruppe ein Knoten erzeugt. Daraus ergibt sich eine Laufzeit von:

$$O(n).$$

Probleme der Überprüfung Bei dieser Methode können zwei verschiedende Arten von Problemen auftreten:

- Mehrdeutigkeiten und 1-lookahead Determinismus
- Ringschlüsse innerhalb einer DTD

Mehrdeutigkeiten und 1-lookahead Determinismus Die Beschreibungsmodelle ermöglichen es im allgemeinen Fall, dass mehrdeutige Grammatiken definiert werden können. Die Mehrdeutigkeiten können bei Alternativen von Unterelementen auftreten.

Beispiel 4.4.24 Ein Element mit einem mehrdeutigen Inhaltsmodell:

ELEMENT((a* b) | b)

□

Das Problem besteht darin, den Baum zu finden, der aufgebaut werden soll. Wenn das Symbol b eingelesen wird, ist nicht klar, ob das a weggelassen worden ist (was durch die Kennzeichnung des Sterns „*“ richtig sein kann) oder ob der Baum mit dem Unterelement b aufgebaut werden soll.
Die Mehrdeutigkeiten müssen in der Definition ausgeschlossen werden, damit es zu keinen Fehlern kommt. Dies ist bei XML Grammatiken, einfachen balancierten Grammatiken und konkurrenzverhindernden balancierten Grammatiken der Fall.
Desweiteren muss die 1-lookahead Determinismus Eigenschaft eingehalten werden. Es existiert zum Beispiel ein Beschreibugsmodell

$$\text{ELEMENT((a}^* \text{ b) | (b c)),}$$

welches zwar nicht mehrdeutig ist, in dem es jedoch trotzdem nicht für jedes Eingabezeichen eindeutig ist, welcher Teil des regulären Ausdrucks dieses Zeichen erzeugt.
Auch dies wird von den oben genannten Grammatiken eingeschränkt.

Ringschlüsse innerhalb einer DTD Bei Ringschlüssen (Selbsteinbettung) innerhalb eines Baums verweisen Knoten auf Vorgängerknoten. Dieses Problem wurde bisher noch nicht behandelt. Eine Möglichkeit besteht darin, sich alle Vorgängerverweise zu merken und im Anschluss zu bearbeiten. Dies kann mit einem Stapel implementiert werden, auf welchen immer die letzte Zeigerposition gelegt wird. Nach der Beendigung eines Baums wird der Stapel überprüft und gegebenenfalls an die Stelle des Zeigers zurückgesprungen.

4.4.5 Konstruktion eines regulären Ausdrucks und einer Dyck Grammatik aus einer RegDyck Grammatik

Es sei $G = (N, T, S, P), T = A \cup \overline{A}$ eine balancierte Grammatik. Die Grammatik G wird indiziert und zu $G' = (N', T', S', P')$.

Die Konstruktion eines erkennenden RegDyck Automaten $\mathcal{K}$ enthält die Konstruktion von Automaten $\mathcal{K}_X, X \in N'$, die einzelne Teile der Grammatik erkennen. Diese Automaten werden miteinander verbunden.
Die Automaten für die Inhaltsmodelle werden durch eine Methode, die der Thompson Methode für reguläre Ausdrücke ähnlich ist, zusammengesetzt. Die Automaten für selbsteinbettende Sprachteile werden über zwei Schleifen konstruiert.
Der entstehende Automat ist ein RegDyck Automat. Aus diesem wird ein endlicher Automat konstruiert, indem nur die Zustandsüberführungen übernommen werden.

Nun wird noch der Dyck Automat erzeugt:

$\mathcal{D} = (S_D, A_D, G_D, \triangle, s_{0D}, \delta_D, F_D)$ ist ein deterministischer Kellerautomat, der die Menge $D_{A \cup \overline{A}}$ von Dyckprimes erkennt:

- $S_D = \{s_{0D}, s_{1D}, s_{eD}\}$ ist die Menge der Zustände des Automaten.
- $A_D = A \cup \overline{A}$ ist das Eingabealphabet.
- $G_D = \overline{A} \cup \{\triangle\}$ ist das Kelleralphabet.
- s_{0D} ist der Startzustand.
- $F_D = \{s_{0D}, s_{eD}\}$ ist die Menge der Endzustände.
- δ_D ist die Überführungsfunktion:

$$\begin{array}{lcl} (s_{0D}, a, \triangle) & \longrightarrow & (s_{1D}, \overline{a}\triangle)\ \forall a \in A \\ (s_{1D}, a, \overline{z}) & \longrightarrow & (s_{1D}, \overline{az})\ \forall a \in A\ \forall \overline{z} \in \overline{A} \\ (s_{1D}, \lambda, \triangle) & \longrightarrow & (s_{eD}, \triangle) \\ (s_{1D}, \overline{a}, \overline{a}) & \longrightarrow & (s_{1D}, \lambda)\ \forall \overline{a} \in \overline{A}. \end{array}$$

Die Konstruktion eines endlichen Automaten aus einer Grammatik mit Hilfe der Erweiterung der Thompson Methode, ist in

$$O(n)$$

Zeit möglich. Im Vergleich zu der originalen Thompson Methode werden hierbei jedoch nur die Kelleroperationen zusätzlich berücksichtigt. Diese stehen aber, wie schon in Kapitel 3 erwähnt, anhand der Eingabesymbole fest und die Konstruktion benötigt keine zusätzliche Zeit. Wichtig hierbei ist die Tatsache, dass nichtdeterministische endliche Automaten konstruiert werden können.
Die Konstruktion des Dyck Automaten benötigt in Bezug auf die Größe der Grammatik konstante Zeit $O(1)$, da die Größe des Dyck Automaten von der Größe des Alphabets abhängig ist. Daraus ergibt sich eine Laufzeit von

$$O(n) + O(1) = O(n).$$

Diese Methode erscheint vor diesem Hintergrund geeignet für den Parsingvorgang.

4.5 Parsing Algorithmen

Im letzten Unterkaptiel wurde die Konstruktion von Parsergeneratoren beschrieben, zusätzlich wurden sie auf ihre Komplexität analysiert. In diesem Kapitel werden zwei Parsing Algorithmen beschrieben. Der erste Parsing Algorithmus überprüft codeterministische balancierte Grammatiken, der zweite nutzt die Parsebäume, die im letzten Unterkapitel vorgestellt worden sind, zur Syntaxüberprüfung. Die anderen Parser, die im letzten Kapitel konstruiert worden sind, müssen nicht mehr analysiert werden, da ihre Laufzeiten bereits bekannt sind.

4.5.1 Parsen mittels codeterministischer Grammatiken

Es sei $G = (N, T, S, P)$ eine codeterministische balancierte Grammatik.
Es wird für jede Produktion $p \in P$ ein nichtdeterministischer endlicher Automat $\mathcal{R}_p$ konstruiert, der die Sprache der rechten Seiten der Produktion erkennt. Aus allen Automaten $\mathcal{R}_p$ wird ein nichtdeterministischer endlicher Automat $\mathcal{A}$ konstruiert, welcher alle Produktionen erkennt.
Aus den Endzuständen ist dabei ersichtlich, welche Produktion der Automat erkannt hat. Die codeterministischen balancierten Grammatiken haben die Eigenschaften, dass jede rechte Seite einer Produktion eindeutig ist.

Aufgrund der speziellen Syntax der Sprache (öffnende und schließende Klammern) können sofort die verschiedenen Ebenen der Produktionen erkannt werden. Dies wird im folgenden Algorithmus ausgenutzt.
Der Algorithmus benutzt eine Tabelle. Die Tabelle besteht aus zwei verschiedenen Spalten (*Ebene* und *Zeichenkette*).
Die eingelesenen Zeichen beim Parsingvorgang werden auf verschiedenen Ebenen aufgeschrieben.
Die Ebenen der weiter unten liegenden Zeichen sind die Nachfolgeebenen und kennzeichnen die Zeichen des Inhaltsmodells.
Für die Überprüfung der Syntax wird folgender Algorithmus verwendet:

- Lese die Zeichen der Eingabe nacheinander von vorne nach hinten ein.
- Bei einem Zeichen $a \in A$ schreibe dieses Zeichen auf die nächste Ebene.
- Bei einem Zeichen $\overline{a} \in \overline{A}$ schreibe das Zeichen auf die aktuelle Ebene und überprüfe die Zeichenkette mit dem Automaten $\mathcal{A}$. Das zurückgegebene Nichtterminal schreibe in die Zeichenkette auf die um eins höhere Ebene. Dann lösche den Inhalt der niedrigsten Ebene.

Wenn die Zeichenkette abgearbeitet ist und das Startsymbol erreicht wurde, war die Überprüfung erfolgreich.
Bei einem der folgenden drei Fälle wird abgebrochen und die Analyse war nicht erfolgreich:

1. $\mathcal{A}$ liefert kein Ergebnis.
2. Die Zeichenkette ist abgearbeitet und das Startsymbol ist nicht erreicht.
3. Es wird ein Zeichen $\overline{a} \in \overline{A}$ eingelesen und es gibt kein passendes Zeichen $a \in A$ auf dieser Ebene.

Jedes Zeichen wird nur einmal eingelesen und verarbeitet. Das bedeutet eine Komplexität in Bezug auf die Eingabe von $O(n)$. Jedoch werden in den Zwischenschritten Zeichenketten erzeugt, die durch den nichtdeterministischen endlichen Automaten $\mathcal{A}$ erkannt werden müssen. Es sei n die Länge

der Eingabe, dann können $n/2$ Nichtterminale entstehen, die der Automat zusätzlich zu der Eingabe verarbeiten muss. Damit ergibt sich eine Komplexität von

$$O(n) + O(n/2) = O(n).$$

Dazu kommt die Zeit, die der Automat $\mathcal{A}$ benötigt, um die Zeichenketten zu verarbeiten. Insgesamt ergibt sich dadurch:

$$2 \cdot (O(n) + O(n/2)) = O(n).$$

Beispiel 4.5.25 Es sei $G = (N, T, \{S\}, P)$ eine codeterministische balancierte Grammatik:

- $N = \{S, B, C, D, E\}$
- $T = A \cup \overline{A}, A = \{a, b, c, d, e\}, \overline{A} = \{\overline{a}, \overline{b}, \overline{c}, \overline{d}, \overline{e}\}$
- $P = \{$

$$\begin{array}{lcl} S & \longrightarrow & aDC^*B^+\overline{a} \\ B & \longrightarrow & b\overline{b} \\ C & \longrightarrow & cE^+\overline{c} \\ D & \longrightarrow & d\overline{d} \\ E & \longrightarrow & e\overline{e}\} \end{array}$$

Der nichtdeterministische endliche Automat für die Grammatik sieht folgendermaßen aus:

- $S' = \{s_0, s_a, s_D, s_C, s_B, s_{\overline{a}}, s_b, s_{\overline{b}}, s_c, s_E, s_{\overline{c}}, s_d, s_{\overline{d}}, s_e, s_{\overline{e}}\}$
- $A' = \{a, b, c, d, e, \overline{a}, \overline{b}, \overline{c}, \overline{d}, \overline{e}, S, B, C, D, E\}$
- $B' = \{S, B, C, D, E\}$
- s_0
- $F = \{s_{\overline{a}}, s_{\overline{b}}, s_{\overline{c}}, s_{\overline{d}}, s_{\overline{e}}\}$
- Die Überführungen des Automaten sind folgende

$$
\begin{array}{lcl}
(s_0, a) & \longrightarrow & (s_a, \lambda) \\
(s_0, b) & \longrightarrow & (s_b, \lambda) \\
(s_0, c) & \longrightarrow & (s_c, \lambda) \\
(s_0, d) & \longrightarrow & (s_d, \lambda) \\
(s_0, e) & \longrightarrow & (s_e, \lambda) \\
(s_a, D) & \longrightarrow & (s_D, \lambda) \\
(s_D, C) & \longrightarrow & (s_C, \lambda) \\
(s_D, B) & \longrightarrow & (s_B, \lambda) \\
(s_C, C) & \longrightarrow & (s_C, \lambda) \\
(s_C, B) & \longrightarrow & (s_B, \lambda) \\
(s_B, B) & \longrightarrow & (s_B, \lambda) \\
(s_B, \overline{a}) & \longrightarrow & (s_{\overline{a}}, S) \\
(s_b, \overline{b}) & \longrightarrow & (s_{\overline{b}}, B) \\
(s_c, E) & \longrightarrow & (s_E, \lambda) \\
(s_E, E) & \longrightarrow & (s_E, \lambda) \\
(s_E, \overline{c}) & \longrightarrow & (s_{\overline{c}}, C) \\
(s_d, \overline{d}) & \longrightarrow & (s_{\overline{d}}, D) \\
(s_e, \overline{e}) & \longrightarrow & (s_{\overline{e}}, E)
\end{array}
$$

Es sei die Zeichenkette $ad\overline{d}ce\overline{e}\overline{c}ce\overline{e}e\overline{e}e\overline{e}\overline{c}b\overline{b}\overline{a}$ gegeben.

Ebene	**Zeichenkette**
0.	

Im ersten Schritt wird das erste Zeichen $a \in A$ eingelesen:

$$\underline{a}d\overline{d}ce\overline{e}\overline{c}ce\overline{e}e\overline{e}e\overline{e}\overline{c}b\overline{b}\overline{a}$$

Ebene	**Zeichenkette**
0.	a

nach dem ersten Schritt ergibt sich folgende Konfiguration:

$$a\underline{d}\overline{d}ce\overline{e}\overline{c}ce\overline{e}e\overline{e}e\overline{e}\overline{c}b\overline{b}\overline{a}$$

nach dem 5. Schritt ergibt sich folgende Konfiguration:

Ebene	**Zeichenkette**
0.	a D
1.	c
2.	e

$$ad\overline{d}ce\underline{\overline{e}}\overline{c}ce\overline{e}e\overline{e}e\overline{e}\overline{c}b\overline{b}\overline{a}$$

Nachdem das Zeichen $\overline{e} \in \overline{A}$ eingelesen wurde, wird die Zeichenkette auf der zweite Stufe von $\mathcal{A}$ ausgewertet und es wird das Nichtterminal E zurückgegeben. Das Nichtterminal wird auf der Ebene eins eingetragen. Die Zeichen auf der Ebene zwei werden gelöscht.

Ebene	**Zeichenkette**
0.	a D
1.	c E

Nach dem 17. Schritt ergibt sich folgende Konfiguration:

Ebene	**Zeichenkette**
0.	a

$$a\overline{d}\overline{d}c\overline{e}\overline{e}\overline{c}c\overline{e}\overline{e}\overline{e}\overline{e}\overline{e}\overline{c}b\overline{b}\underline{\overline{a}}$$

Das Zeichen $\overline{a} \in \overline{A}$ wird eingelesen. Das Nichtterminal S wird ausgegeben. Es ist das Startsymbol.
Die Zeichenkette ist fertig bearbeitet.
Die Analyse war erfolgreich.

4.5.2 Parsen mittels Parsebäumen

Im Folgenden wird der Parsingalgorithmus für die Parsebäume der XML Dokumente beschrieben. Das XML Dokument wird schrittweise durchlaufen. Dabei ist jedes Element ein Zeichen der dazugehörigen kontextfreien Grammatik. Es wird vor dem ersten Knoten des Parsebaums begonnen.

Algorithmus Der Baum wird in Preorder durchlaufen.

1. Das erste Zeichen x ($x \in A$, Starttag) wird eingelesen. Falls $x \in \overline{A}$, dann ist das Dokument fehlerhaft.

2. Konnte kein nächstes Zeichen eingelesen werden, dann wird überprüft, ob im Baum ebenfalls das Ende erreicht ist und alle Knoten behandelt sind oder ob alle weiteren Knoten mit *Optional* mit $True$ belegt sind. Stimmt dies, ist die Gültigkeitsanalyse erfolgreich, andernfalls nicht.

3. Es wird zum nächsten Knoten des Baums gesprungen. (Wird in eine nichtaddressierte Speicherstelle verwiesen, wird ein Fehler gemeldet.)

4. Das Zeichen wird mit dem Wert des Knotens des Baums verglichen. Ist der Vergleich erfolgreich, dann wird mit dem nächsten Punkt des Algorithmus fortgefahren. Falls der Vergleich nicht korrekt ist, wird überprüft, ob *Option* auf $True$ gesetzt ist. Wenn dies der Fall ist, wird zum nächsten Knoten in der Liste gesprungen. Ist dies nicht der Fall, wird ein Fehler ausgelöst.

5. Das nächste Zeichen x wird eingelesen. Falls $x = \overline{z} \in \overline{A}$ ist, wird überprüft, ob der aktuelle Knoten mit dem Wert $z \in A$ belegt ist. Ist dies der Fall, dann ist dieser Knoten abgearbeitet und es wird zu dem Vorgängerknoten gesprungen und mit Punkt 5 fortgefahren. Sonst wird ein Fehler ausgelöst. Falls $x \in A$ ist, wird mit Punkt 6 fortgefahren.

6. Falls $Wiederholen$ im Knoten auf $True$ steht, wird wieder mit dem aktuellen Knoten verglichen. Ist der Vergleich korrekt, wird mit Punkt 5 fortgefahren. Wenn der Vergleich nicht korrekt ist oder $Wiederholen$ auf $False$ steht, wird mit Punkt 2 fortgefahren.

Der Algorithmus liest jedes Zeichen einmal ein. Die Operationen im Baum benötigen konstante Zeit. Es folgt eine Laufzeit von:

$$O(n).$$

Für ein Dokument, welches der DTD aus Beipiel 4.4.23 unterliegt, muss der Parsebaum der Abbildung 4.4, wie im Algorithmus aus Kapitel 4.4.2 beschrieben, durchlaufen werden.

Beispiel 4.5.26

<Person>
<Vorname> Klaus </Vorname>
<Vorname> Ernst </Vorname>
<Nachname> Müller </Nachname>
<Adresse>
<Ort> Wetzlar < /Ort>
</Adresse>
< /Person>

Das erste Zeichen (Elementname, Person) des Dokumentes wird gelesen. Dieser Name wird mit dem Wert des ersten Knotens des Baums verglichen. Sie stimmen überein.
Das nächste Zeichen (Elementname, Vorname) wird eingelesen. Wiederholen steht nicht auf $True$, deswegen wird mit Punkt 2 gearbeitet und zum nächsten Knoten verzweigt. Das Zeichen wird mit dem Knoten Vorname verglichen. Der Vergleich ist erfolgreich.
Das nächste Zeichen (Elementname, Vorname) wird eingelesen. Wiederholen ist im Knoten Vorname mit $True$ markiert, deswegen wird der Elementname wieder mit dem Knoten Vorname verglichen. Der Vergleich ist korrekt, also wird mit Punkt 4 fortgefahren.
Das nächste Zeichen (Elementname, Nachname) wird eingelesen. Es wird wieder mit Vorname verglichen, da Wiederholen auf $True$ steht. Der Vergleich ist nicht erfolgreich, deswegen wird mit Punkt 2 fortgefahren und in den nächsten Knoten verzweigt. Dieser wird mit dem Elementnamen verglichen. Der Vergleich ist erfolgreich.
Das nächste Zeichen (Elementname, Adresse) wird eingelesen. Wiederholen steht für das Element Nachname nicht auf $True$, deswegen wird der nächste Knoten angezeigt. Der Vergleich stimmt.
Das nächste Zeichen (Elementname, Ort) wird eingelesen. Der nächste Knoten wird eingelesen, da Wiederholen in Adresse auf $False$ steht. Der Vergleich stimmt wieder.

Das nächste Zeichen wird eingelesen, allerdings ist das Ende schon erreicht. Es wird nachgeschaut, ob alle noch offenen Knoten im Baum optional sind. Dies ist der Fall, also stimmt die Analyse.

Der Algorithmus wird an die Syntaxanalyse angebunden, damit das Dokument nicht noch mal abgearbeitet werden muss.

Durch die Ergebnisse der letzten Unterkapitel entsteht die folgende Tabelle.

Sprache	**max. Laufz. Parsergenerator**	**max. Laufz. Parser**
XML Grammatiken	$O(n)$	$O(n)$
einfache balancierte Grammatiken	$O(n)$	$O(n)$
konk. bal. Grammatiken	$O(n)$	$O(n)$
RegDyck Grammatiken	$O(n)$	$O(n)$
balancierte Grammatiken durch codet. bal. Grammatiken	$\Omega(\frac{m}{n} \cdot 2^{2^{4n^{\lceil \frac{m}{n} \rceil}}})$	$O(n)$
balancierte Grammatiken det. Kellerautomat	$O(2^{5n})$	$O(n)$

Kapitel 5

Eigenschaften von XML-artigen Sprachen

In dem folgenden Kapitel werden Abschlusseigenschaften der XML-artigen Sprachen untersucht. Darüber hinaus werden Entscheidbarkeitsfragen der balancierten Sprachen geklärt.
Aus praktischer Motivation sind Kenntnisse über Abschlusseigenschaften sehr wichtig. Zum Beispiel können Werkzeuge zum Verbinden oder zum Vereinigen zweier Grammatiken konstruiert werden.
Aus theoretischer Sicht sind die Abschlusseigenschaften und Entscheidbarkeitsfragen Teil der Analyse und geben Aussagen über die Mächtigkeit der Erzeugungsmechanismen.

Die booleschen Operationen Komplement, Vereinigung und Durchschnitt wurden in [BB02a] für die balancierten Sprachen gezeigt. In [BB02b] wurden die booleschen Eigenschaften Durchschnitt, Vereinigung und Komplement der XML Sprachen betrachtet. Das Komplement bezieht sich im Gegensatz zur allgemeinen Definition auf die Menge der Dyckprimes.
Im Folgenden sollen die booleschen Eigenschaften für die einfachen balancierten Sprachen, konkurrenzverhindernden balancierten Sprachen und RegDyck Sprachen untersucht werden.
Konkatenation, Iteration und Substitution werden für alle Sprachtypen in einer erweiterten Form betrachtet, da es in den Wörtern der XML-artigen Sprachen einen festen Start- und Endtag geben muss und z. B. die Konkatenation der Sprachen $L = \{x\overline{x}\}$ und $L' = \{y\overline{y}\}$

$$L^{\#} = \{x\overline{x}y\overline{y}\}$$

ergeben würde. Zusätzlich werden die Operationen Spiegelung und Homomorphismus untersucht.

In dem zweiten Teil des Kapitels werden Entscheidbarkeitsfragen der Spra-

chen geklärt. Für die Praxis ist die Frage interessant, ob eine gegebene balancierte Grammatik eine Grammatik eines Untertyps ist. Damit könnte vorab entschieden werden, welche Art von Parser konstruiert werden muss. So muss nicht für jede balancierte Grammatik eine zeitaufwändige Konstruktion vollzogen werden.
Darüber hinaus wird untersucht, ob entschieden werden kann, ob eine balancierte Grammatik mehrdeutig ist und ob eine balancierte Sprache eine Teilmenge einer anderen ist.

5.1 Abschlusseigenschaften

Die XML-artigen Grammatiken wurden in fünf unterschiedliche Typen unterteilt: balancierte Grammatiken, XML Grammatiken, einfache balancierte Grammatiken, konkurrenzverhindernde balancierte Grammatiken und Reg-Dyck Grammatiken.
In den folgenden Unterkapiteln werden die Sprachen der einzelnen Grammatiktypen auf ihre Abschlusseigenschaften hin untersucht.

5.1.1 Vereinigung

Satz 5.1.20 Die balancierten Sprachen sind unter Vereinigung abgeschlossen [BB02a].

Satz 5.1.21 Die XML Sprachen sind nicht unter Vereinigung abgeschlossen [BB02b].

Satz 5.1.22 Die einfachen balancierten Sprachen sind nicht unter Vereinigung abgeschlossen.

Beweis. L_0 und L_1 sind zwei einfache balancierte Sprachen, die aus den einzigen Wörtern

$$x c d \bar{d} \overline{cx} \text{ und}$$
$$x c \bar{c} e \overline{ex}$$

bestehen.

Angenommen, die einfachen balancierten Sprachen wären unter Vereinigung abgeschlossen, dann müssten die Sprachen L_0 und L_1 vereinigt wieder eine einfache balancierte Sprache ergeben.

Die Grammatik der Vereinigung $G = (N, T, S, P)$ der Sprachen muss in

der Startproduktion $S \longrightarrow xm\overline{x}$ im Inhaltsmodell mindestens zwei Nichtterminale C_0 und C_1, die $cd\overline{d}\overline{c}$ und $c\overline{c}$ erzeugen und ein Zeichen E, welches $e\overline{e}$ erzeugt ($S \longrightarrow x(C_0|(C_1E))\overline{x}$), enthalten.
Wird nur ein Nichtterminal C benutzt ($S \longrightarrow xC|CE\overline{x}$), dann muss es sowohl $cd\overline{d}\overline{c}$ als auch $c\overline{c}$ erzeugen. Dies führt dazu, dass auch das Wort

$$xcdd\overline{d}\overline{c}e\overline{e}\overline{x}$$

erzeugt wird. Dieses konnte von den ursprünglichen Grammatiken nicht erzeugt werden.
Wenn zwei Nichtterminale zur Definition von $cd\overline{d}\overline{c}$ und $c\overline{c}$ verwendet werden, z.B.

$$C_0 \longrightarrow cD\overline{c}$$
$$C_1 \longrightarrow c\overline{c},$$

dann muss das Inhaltsmodell $m = (C_0|(C_1E))$ sein.
Diese Konstruktion ist allerdings in einfachen balancierten Grammatiken nicht erlaubt.
Es dürfen keine zwei Nichtterminale mit dem gleichen Terminal auf der rechten Seite in einer Produktion stehen. Dies führt zum Widerspruch zur Annahme. Die einfachen balancierten Sprachen sind somit nicht unter Vereinigung abgeschlossen. □

Satz 5.1.23 Die konkurrenzverhindernden balancierten Sprachen sind nicht unter Vereinigung abgeschlossen.

Beweis. L_0 und L_1 sind zwei konkurrenzverhindernde balancierte Sprachen, die aus den Wörtern

$$x(cd\overline{d}\overline{c})^*\overline{x} \text{ und}$$
$$x(c\overline{c})^*e\overline{e}\overline{x}$$

bestehen.

Angenommen, die konkurrenzverhindernden balancierten Sprachen wären unter Vereinigung abgeschlossen, dann müssten die Sprachen L_0 und L_1 vereinigt wieder eine konkurrenzverhindernde balancierte Sprache ergeben.

Die Grammatik der Vereinigung $G = (N, T, S, P)$ muss mindestens zwei Nichtterminale C_0 und C_1 und ein Nichtterminalzeichen E in der Startproduktion $S \longrightarrow xm\overline{x}$ im Inhaltsmodell enthalten ($S \longrightarrow x(C_0^*|(C_1^*E))\overline{x}$).
Wird nur ein Nichtterminal C benutzt ($S \longrightarrow x(C^*|(C^*E))\overline{x}$), dann muss es sowohl $cd\overline{d}\overline{c}$ als auch $c\overline{c}$ erzeugen. Dies führt dazu, dass auch das Wort

$$xcdd\overline{d}\overline{c}e\overline{e}\overline{x}$$

erzeugt wird. Dieses konnte von den ursprünglichen Grammatiken nicht erzeugt werden.
Wenn zwei Nichtterminale zur Definition der Wörter $\{(cd\overline{d}\overline{c})^*\}$ und $\{(c\overline{c})^*\}$ verwendet werden, z.B.

$$C_0 \longrightarrow cD\overline{c}$$
$$C_1 \longrightarrow c\overline{c},$$

dann muss das Inhaltsmodell $m = (C_0^*|(C_1^*E))$ sein. Diese Konstruktion ist allerdings in konkurrenzverhindernden balancierten Grammatiken nicht erlaubt, da das Inhaltsmodell m die Sequenzen UC_0W und UC_1V mit $U, V, W \in N^*$ erzeugt. Dies führt zum Widerspruch zur Annahme. Somit sind die konkurrenzverhindernden balancierten Sprachen nicht unter Vereinigung abgeschlossen. □

Satz 5.1.24 Die RegDyck Sprachen sind unter Vereinigung abgeschlossen.

Beweis. L_0 und L_1 sind zwei RegDyck Sprachen, die durch die Grammatiken $G_0 = (N, T, S, P)$ und $G_1 = (N_1, T_1, S_1, P_1)$ erzeugt werden.
In P liegt die Startproduktion

$$S_0 \longrightarrow xm_0\overline{x}$$

und in P_1 liegt die Startproduktion

$$S_1 \longrightarrow xm_1\overline{x}.$$

Dann wird eine Menge von Startzeichen erzeugt, in der beide Startzeichen der ursprünglichen Grammatiken enthalten sind.
Die Vereinigung von zwei RegDyck Sprachen ist folgendermaßen zu konstruieren:
Es seien L_0 und L_1 die beiden Sprachen, die vereinigt werden und G_0 und G_1 die erzeugenden Grammatiken.
Ohne Beschränkung der Allgemeinheit gelte $N_0 \cap N_1 = \emptyset$. (Sonst erfolgt eine Umbenennung in einer der beiden Mengen, so dass diese Forderung erfüllt ist.) Die Vereinigung der beiden Sprachen wird durch eine neue Grammatik $G' = (N', T', S', P')$ erzeugt:

- $N' = N_0 \cup N_1$ sind die Nichtterminalzeichen.
- $T' = T_0 \cup T_1$ sind die Terminalzeichen.
- $S_0 \cup S_1$ sind die Startsymbole.
- $P' = P_0 \cup P_1$ sind die Produktionen.

□

5.1.2 Durchschnitt

Der Abschluss unter Durchschnitt wurde für die balancierten Sprachen und die XML Sprachen schon in [BB02a] gezeigt. Die Beweisidee für die RegDyck Sprachen kann auch auf die einfachen balancierten Sprachen und die konkurrenzverhindernden balancierten Sprachen übertragen werden. In dem Beweis für die XML Sprachen wird jedoch von Grammatiken mit beschänkten Inhaltsmodellen ausgegangen, obwohl die Inhaltsmodelle der XML Grammatiken aus regulären Ausdrücken bestehen dürfen. Der Abschluss der balancierten Sprachen ist eine Folgerung aus den Ergebnissen, dass die balancierten Sprachen unter Vereinigung und Komplement abgeschlossen sind.
Nun wird der Beweis für die RegDyck Grammatiken durchgeführt. Er kann jedoch auf alle anderen XML-artigen Sprachfamilien übertragen werden.

Satz 5.1.25 Die RegDyck Sprachen sind unter Durchschnitt abgeschlossen.

Beweis. L und L_1 sind zwei RegDyck Sprachen, die durch die Grammatiken $G = (N, T, S, P)$ und $G' = (N', T', S', P')$ erzeugt werden.
Die Grammatik $G^{\#} = (N^{\#}, T^{\#}, S^{\#}, P^{\#})$, die den Durchschnitt erzeugt, wird Schritt für Schritt aus den Produktionen der Grammatiken G und G' erzeugt.
Wenn

$$X \longrightarrow x\overline{x}$$

eine Produktion in G ist und

$$X' \longrightarrow x\overline{x}$$

eine Produktion in G' ist, dann wird in $G^{\#}$ eine Produktion

$$(X, X') \longrightarrow x\overline{x}$$

erzeugt. Wenn

$$X \longrightarrow xm\overline{x}$$

eine Produktion in G ist und

$$X' \longrightarrow xm'\overline{x}$$

eine Produktion in G' ist, dann wird in $G^{\#}$ eine Produktion

$$(X, X') \longrightarrow x(m \cap m')\overline{x}$$

erzeugt. Der Durchschnitt des Inhaltsmodells $m \cap m'$ wird im Folgenden behandelt.
Es sei $P_Y, Y \in N$, die Menge aller Nichtterminalzeichen aus der Grammatik G', die mit Y konkurrieren

$$P_Y = \{Y' \mid Y \longrightarrow ym\overline{y} \in P \wedge Y' \longrightarrow ym'\overline{y} \in P'\},$$

mit $Y' \in N', y, \overline{y} \in T', m \in Reg(N), m' \in Reg(N')$. Analog gilt dies für die Menge $P_{Y'}, Y' \in P'$.
Nun werden alle Vorkommen von Nichtterminal Z in m durch die Vereinigung

$$\bigcup_{Z' \in P_Z} (Z, Z')$$

ersetzt und die Vorkommen von Nichtterminalzeichen Z' in m' durch

$$\bigcup_{Z \in P_{Z'}} (Z, Z')$$

ersetzt und es wird der Durchschnitt $m \cap m'$ gebildet. Da m und m' regulär sind, kann dies durchgeführt werden. Der Durchschnitt zweier balancierter Sprachen ist somit effektiv erzeugbar.

□

Satz 5.1.26 Die balancierten Sprachen sind unter Durchschnitt abgeschlossen [BB02a].

Satz 5.1.27 Die XML Sprachen sind unter Durchschnitt abgeschlossen [BB02b].

Satz 5.1.28 Die einfachen balancierten Sprachen sind unter Durchschnitt abgeschlossen.

Beweis. Der Beweis ist analog zu dem der RegDyck Sprachen. □

Satz 5.1.29 Die konkurrenzverhindernden balancierten Sprachen sind unter Durchschnitt abgeschlossen.

Beweis. Der Beweis ist analog zu dem der RegDyck Sprachen. □

5.1.3 Komplement

Die Komplementbildung muss für die balancierten Sprachen in einer erweiterten Form betrachtet werden, da das Komplement jeder balancierten Sprache Wörter enthält, die keine Dyckprimes sind und somit auch nicht mehr durch eine balancierte Grammatik erzeugt werden können.

Definition 5.1.26 Das erweiterte Komplement einer balancierten Sprache L über einem Alphabet T ist die Sprache $D_T \setminus L$, wobei D_T die Menge der Dyckprimes über dem Alphabet T ist.

Satz 5.1.30 Die balancierten Sprachen sind unter erweitertem Komplement

abgeschlossen [BB02a].

Satz 5.1.31 Die XML Sprachen sind nicht unter erweitertem Komplement abgeschlossen [BB02b].

Satz 5.1.32 Die einfachen balancierten Sprachen sind nicht unter erweitertem Komplement abgeschlossen.

Beweis. Angenommen, die einfachen balancierten Sprachen wären unter erweitertem Komplement abgeschlossen, dann müsste das erweiterte Komplement einer Sprache L, die aus dem einzigen Wort

$$xce\overline{ec}\overline{x}$$

besteht, durch eine einfache balancierte Grammatik erzeugbar sein. Das bedeutet, im erweiterten Komplement würden unter anderem alle Wörter $xd_cd_{c0}\overline{x}$ liegen. Dabei ist d_c die Menge aller Dyckprimes, die mit c beginnen und d_{c0} die Menge aller Dyckprimes, die mit c beginnen, ohne das Wort $ce\overline{ec}$. Die Grammatik müsste somit eine Produktion $X \longrightarrow xCC_1\overline{x}$ mit mindestens zwei verschiedenen Nichtterminalzeichen C, C_1 besitzten, da zwei unterschiedliche Sprachen erzeugt werden müssten. Dieses wären zwei konkurrierende Nichtterminalzeichen, da beide Nichtterminalzeichen Sprachen erzeugen, die mit c beginnen. Dies ist in einer einfachen balancierten Sprache im Inhaltsmodell nicht erlaubt, woraus sich der Widerspruch ergibt. □

Satz 5.1.33 Die konkurrenzverhindernden balancierten Sprachen sind nicht unter erweitertem Komplement abgeschlossen.

Beweis. Angenommen, die konkurrenzverhindernden balancierten Sprachen wären unter erweitertem Komplement abgeschlossen, dann müsste das erweiterte Komplement einer Sprache L, die aus dem einzigen Wort

$$xce\overline{ec}\overline{x}$$

besteht, durch eine konkurrenzverhindernde balancierte Grammatik erzeugbar sein. Das bedeutet, im erweiterten Komplement würden alle Wörter $xd_c\overline{x}$ und $xd_{c0}\overline{x}$ liegen. Dabei ist d_c die Menge aller Dyckprimes, die mit c beginnen und d_{c0} die Menge aller Dyckprimes, die mit c beginnen, ohne das Wort $ce\overline{ec}$. Die Grammatik müsste somit eine Produktion $X \longrightarrow x(C|C_1)\overline{x}$ mit mindestens zwei verschiedenen Nichtterminalzeichen C, C_1 besitzen, da zwei unterschiedliche Sprachen erzeugt werden müssten. Dieses wären zwei konkurrierende Nichtterminalzeichen, da beide Nichtterminalzeichen Sprachen erzeugen, die mit c beginnen und aus dem Inhaltsmodell können Wörter der Form UCW und UC_1V abgeleitet werden, wobei U, W, V Folgen von Nichtterminalzeichen sind, die auch leer sein können.

Dies ist in einer konkurrenzverhindernden balancierten Sprache im Inhaltsmodell nicht erlaubt, woraus sich der Widerspruch ergibt.

□

Satz 5.1.34 Die RegDyck Sprachen sind nicht unter erweitertem Komplement abgeschlossen.

Beweis. Es sei $L = \{x^m x^k \overline{x}^k x^n \overline{x}^n \overline{x}^m \mid k, m, n \in \mathbb{N}\}$ eine RegDyck Sprache. Angenommen, die RegDyck Sprachen wären unter Komplement abgeschlossen, dann muss eine RegDyck Grammatik existieren, die das erweiterte Komplement der Sprache L erzeugt.
In dem erweitertem Komplement von L sind auch die Wörter der Form $xx^n\overline{x}^n x^m \overline{x}^m \overline{x}, n, m \in \mathbb{N}$ enthalten. Um diese Wörter zu erzeugen, muss eine Startproduktion der Form $S \longrightarrow xXX\overline{x}$ existieren. Es muss genau zwei Nichtterminalzeichen geben, die selbsteinbettend sind und jeweils $x^i\overline{x}^i$ erzeugen. Dies führt dazu, dass $FA(X) = x^*\overline{x}^*$. Das bedeutet, X steht zu X im Konflikt. Im Durchschnitt $(x^*\overline{x}^*)(x^*\overline{x}^*) \cap D_{(\{x,\overline{x}\})} \neq \emptyset$ liegen auch die Worte $xx^n\overline{x}^n x^m \overline{x}^m \overline{x}, n, m \in \mathbb{N}$. Diese sind jedoch in L enthalten und dürfen somit nicht im Komplement vorkommen. Das bedeutet, der Konflikt kann nicht aufgehoben werden. Es ergibt sich ein Widerspruch zur Annahme, da das erweiterte Komplement der Sprache L nicht durch eine RegDyck Grammatik erzeugt werden kann.
Es folgt, dass die RegDyck Sprachen nicht unter erweitertem Komplement abgeschlosssen sind. □

Die balancierten Sprachen sind unter allen booleschen Eigenschaften abgeschlossen. Für die Praxis erscheint jedoch vor allem wichtig zu sein, ob die Untertypen der balancierten Sprachen abgeschlossen sind. Dabei zeigt sich, dass die RegDyck Sprachen unter den Operationen Vereinigung und Durchschnitt abgeschlossen sind. Die in der Praxis verwendeten Sprachen (XML Sprachen, einfache balancierte Sprachen) und die konkurrenzverhindernden balancierten Sprachen sind lediglich unter Durchschnitt abgeschlossen.

5.1.4 Iteration

Es sei L' eine balancierte Sprache mit den Wörtern $\{x^n\overline{x}^n \mid n \geq 1\}$. Wird diese Sprache iteriert, so entsteht die Sprache

$$L = \{x^{k_1}\overline{x}^{k_1} x^{k_2}\overline{x}^{k_2} \ldots x^{k_p}\overline{x}^{k_p} \mid k_i \geq 1, 1 \leq i \leq p\} \cup \{\lambda\}.$$

Die Sprache ist nach Definition nicht balanciert, da es ein äußeres schließendes Tagpaar geben muss.

Definition 5.1.27 Die erweiterte Iteration einer balancierten Sprache L wird im Folgenden definiert.
Sei $G = (N, T, S, P)$ eine balancierte Grammatik, die L erzeugt. Die iterierte

Sprache L^* wird in ein neues Tagpaar $s', \overline{s}' \notin T$ eingeschlossen. Es wird eine neue Grammatik $G' = (N', T', S', P')$ erzeugt mit

- $N' = N \cup \{S'\}$
- $T' = T \cup \{s', \overline{s}'\}$
- $S' \notin N$
- $P' = P \cup \{S' \longrightarrow s'(\bigcup_{S' \in S} S')^* \overline{s}'\}$.

G' ist offensichtlich wieder eine balancierte Grammatik.

□

Satz 5.1.35 Die balancierten Sprachen sind unter der erweiterten Iteration abgeschlossen.

Beweis. Aus der Definition der erweiterten Iteration folgt die Behauptung. □

Satz 5.1.36 Die XML Sprachen sind unter der erweiterten Iteration abgeschlossen.

Beweis. Der Beweis ist analog zu dem der balancierten Sprachen. □

Satz 5.1.37 Die einfachen balancierten Sprachen sind unter der erweiterten Iteration abgeschlossen.

Beweis. Der Beweis ist analog zu dem der balancierten Sprachen. □

Satz 5.1.38 Die konkurrenzverhindernde balancierte Sprachen sind unter der erweiterten Iteration abgeschlossen.

Beweis. Der Beweis ist analog zu dem der balancierten Sprachen. □

Satz 5.1.39 Die RegDyck Sprachen sind nicht unter der erweiterten Iteration abgeschlossen.

Beweis. Es sei L die folgende RegDyck Sprache

$$L = \{x^n \overline{x}^n \mid n \geq 1\}.$$

In der Iteration sind z.B. die Wörter

$$K = \{s' x^k \overline{x}^k x^l \overline{x}^l \overline{s}' \mid k \geq 1, l \geq 1\}$$

enthalten.
Angenommen, die RegDyck Sprachen wären unter Komplement abgeschlossen, dann müsste es eine Startproduktion

$$S' \longrightarrow s' S^* \overline{s}'$$

geben, wobei es mindestens ein Nichtterminalzeichen S gibt, welches mit einem Stern versehen ist, welches die Sprache L erzeugt. Jedoch lässt sich aus dem Inhaltsmodell von S' ein Wort SS ableiten. Da S selbsteinbettend ist, $FA(S) = x^+\overline{x}^+$ und $FA(S)FA(S) \cap D \neq \emptyset$, steht S zu S im Konflikt. Dies ist ein Widerspruch zur Definition der RegDyck Sprachen.

□

5.1.5 Konkatenation

Es seien L und L_0 zwei balancierte Sprachen, welche konkateniert werden sollen und $G = (N, T, S, P)$ und $G_0 = (N_0, T_0, S_0, P_0)$ die erzeugenden Grammatiken. L erzeuge eine beliebige Zeichenkette der Form $x \ldots \overline{x}$ und L_0 erzeuge eine beliebige Zeichenkette der Form $x_0 \ldots \overline{x}_0$, wobei $x, \overline{x} \in T$ und $x_0, \overline{x}_0 \in T_0$.
Die Konkatenation würde eine Zeichenkette der Form

$$x \ldots \overline{x} x_0 \ldots \overline{x}_0 \text{ oder}$$
$$x_0 \ldots \overline{x}_0 x \ldots \overline{x}$$

ergeben. Diese Form ist nach Definition nicht erlaubt, da jede Produktion von einem Tagpaar umgeben sein muss.
Es sei auch hier eine abgewandelte Form der Konkatenation vorgestellt.

Definition 5.1.28 Die erweiterte Konkatenation von zwei balancierten Sprachen L und L_0 wird wie folgt definiert:

Sei $G = (N, T, S, P)$ eine balancierte Grammatik, die L erzeugt und $G_0 = (N_0, T_0, S_0, P_0)$ eine balancierte Grammatik, die L_0 erzeugt. Die konkatenierte Sprache LL_0 wird in ein neues Tagpaar $s', \overline{s}' \notin (T \cup T_0)$ eingeschlossen. Es wird eine neue Grammatik $G' = (N', T', S', P')$ erzeugt mit

- $N' = N \cup N_0 \cup \{S'\}$
- $T' = T \cup T_0 \cup \{s', \overline{s}'\}$
- $S' \notin (N \cup N_0)$
- $P' = P \cup P_0 \cup \{S' \longrightarrow s'(\bigcup_{S' \in S} S')(\bigcup_{S'' \in S_0} S'')\overline{s}'\}$.

Analog wird eine neue Grammatik $G' = (N', T', S', P')$ für die konkatenierte Sprache $L_0 L$ erzeugt:

- $N' = N \cup N_0 \cup \{S'\}$
- $T' = T \cup T_0 \cup \{s', \overline{s}'\}$
- $S' \notin (N \cup N_0)$

- $P' = P \cup P_0 \cup \{S' \longrightarrow s'(\bigcup_{S'' \in S_0} S'')(\bigcup_{S' \in S} S')\overline{s}'\}$

G' ist in beiden Fällen offensichtlich wieder eine balancierte Grammatik.

□

Satz 5.1.40 Die balancierten Grammatiken sind unter der erweiterten Konkatenation abgeschlossen.

Beweis. Die Behauptung folgt aus der Definition der erweiterten Konkatenation. □

Satz 5.1.41 Die XML Sprachen sind nicht unter der erweiterten Konkatenation abgeschlossen.

Beweis. Wir nehmen an, die XML Sprachen wären unter erweiterter Konkatenation abgeschlossen.
Es seien L und L_0 zwei XML Sprachen, die konkateniert werden sollen. G und G_0 seien die beiden erzeugenden XML Grammatiken.
Die Grammatik G_0 kann konkurrierende Nichtterminale in Bezug zu den Nichtterminalen der Grammatik G enthalten. Für eine XML Grammatik $G' = (N', T', S', P'), T' = (A' \cup \overline{A}')$ muss gelten:
$\forall X, Y \in N' \; \forall m, m_0 \in Reg(N')$

$$\neg \exists x \in A' : (X \longrightarrow xm\overline{x} \in P' \wedge Y \longrightarrow xm_0\overline{x} \in P' \wedge X \neq Y)$$

Dies ist für die Konkatenation zweier XML Sprachen nicht sichergestellt, da es in beiden Grammatiken Produktionen mit den gleichen Terminalzeichen geben kann.
G bestehe aus der einzigen Produktion

$$S \longrightarrow s\overline{s}$$

und G_0 bestehe aus den beiden Produktionen

$$S_0 \longrightarrow sX\overline{s} \text{ und}$$
$$X \longrightarrow x\overline{x}.$$

Die Grammatik G' der Konkatenation muss die Startproduktion

$$S' \longrightarrow s'SS_0\overline{s}'$$

beinhalten. Genau wie bei der Vereinigung ist es nicht möglich, die Sprache, die S erzeugt und die Sprache, die S_0 erzeugt, durch nur ein Nichtterminalzeichen auszudrücken, um die Konkurrenz der Nichtterminalzeichen S und S_0 zu verhindern.
In XML Grammatiken sind keine konkurrierenden Nichtterminalzeichen erlaubt, somit sind die XML Sprachen nicht unter Konkatenation abgeschlossen.

□

Es seien L und L_0 zwei XML Sprachen und G und G_0 die erzeugenden Grammatiken. Es wird angenommen, dass in G und G_0 nur Terminalzeichen existieren, die auch in den Produktionen benutzt werden. Eine Konkatenation der Sprachen L und L_0 kann auf jeden Fall durchgeführt werden, wenn

$$T \cap T_0 = \emptyset$$

gilt.

Satz 5.1.42 Die einfachen balancierten Sprachen sind nicht unter der erweiterten Konkatenation abgeschlossen.

Beweis. Wir nehmen an, die einfachen balancierten Sprachen wären unter Konkatenation abgeschlossen.
Es seien L und L_0 zwei einfache balancierte Sprachen, die konkateniert werden sollen. G und G_0 seien die beiden erzeugenden einfachen balancierten Grammatiken.
Die Grammatik G_0 kann konkurrierende Nichtterminale in Bezug zu den Nichtterminalen der Grammatik G enthalten. Für einfache balancierte Grammatiken $G' = (N', T', S', P'), T' = (A' \cup \overline{A}')$ muss jedoch gelten:
$\forall X, Y, Z \in N' \; \forall z, \overline{z} \in T' \; \forall m, m_0, m_1 \in Reg(N') \; \forall \alpha, \beta, \gamma \in N^*$

$$\neg \exists x \in A' : (X \longrightarrow xm\overline{x} \in P' \wedge Y \longrightarrow xm_0\overline{x} \in P' \wedge X \neq Y \wedge Z \longrightarrow zm_1\overline{z} \in P' \wedge \alpha X \beta Y \gamma \in L(m_1))$$

Dies ist für die Konkatenation zweier einfachen balancierten Sprachen nicht sichergestellt, da eine neue Produktion erzeugt werden muss, in der beide Startsymbole enthalten sind. Im Fall, dass die Startproduktionen miteinander konkurrieren, verstößt die neue Produktion gegen die Definition der einfachen balancierten Grammatiken.
Der Beweis erfolgt über die gleichen Grammatiken wie bei den XML Sprachen.
G bestehe aus der einzigen Produktion

$$S \longrightarrow s\overline{s}$$

und G_0 bestehe aus den beiden Produktionen

$$S_0 \longrightarrow sX\overline{s} \text{ und}$$
$$X \longrightarrow x\overline{x}.$$

Die Grammatik G' der Konkatenation muss die Startproduktion

$$S' \longrightarrow s' S S_0 \overline{s}'$$

beinhalten. Genau wie bei der Vereinigung ist es nicht möglich, die Sprache, die S erzeugt und die Sprache, die S_0 erzeugt, durch nur ein Nichtterminalzeichen auszudrücken, um die Konkurrenz der Nichtterminalzeichen S und S_0 zu verhindern.
In einfachen balancierten Grammatiken sind keine konkurrierenden Nichtterminalzeichen innerhalb einer Produktion erlaubt, somit sind die XML Sprachen nicht unter Konkatenation abgeschlossen.

□

Es seien L und L_0 zwei einfache balancierte Sprachen und G und G_0 die erzeugenden Grammatiken. Eine Konkatenation der Sprachen L und L_0 kann auf jeden Fall durchgeführt werden, wenn

$$\forall m \in Reg(N)\ \forall m_0 \in Reg(N_0)$$
$$\neg\exists x \in A : (S \longrightarrow xm\overline{x} \in P \wedge S_0 \longrightarrow xm_0\overline{x} \in P_0).$$

Satz 5.1.43 Die konkurrenzverhindernden balancierten Sprachen sind unter der erweiterten Konkatenation abgeschlossen.

Beweis. Es seien L und L_0 zwei konkurrenzverhindernde balancierte Sprachen, die konkateniert werden sollen. $G = (N, T, S, P)$ und $G_0 = (N_0, T_0, S_0, P_0)$ seien die beiden erzeugenden konkurrenzverhindernden balancierten Grammatiken.
Die Grammatik G_0 kann konkurrierende Nichtterminale in Bezug zu den Nichtterminalen der Grammatik G enthalten. Für eine konkurrenzverhindernde balancierte Grammatik $G' = (N', T', S', P'), T' = (A' \cup \overline{A}')$, muss jedoch gelten:
$\forall m, m_0, m_1 \in Reg(N')\ \forall X, Y, Z \in N'\ \forall z \in A'\ \forall U, V, W \in N^*$

$$\neg\exists x \in A' : (X \longrightarrow xm\overline{x} \in P' \wedge Y \longrightarrow xm_0\overline{x} \in P' \wedge X \neq Y \wedge$$
$$Z \longrightarrow zm_1\overline{z} \in P' \wedge UXW \in L(m_1) \wedge UYV \in L(m_1))$$

Angenommen, die konkurrenzverhindernden balancierten Sprachen wären nicht unter Konkatenation abgeschlossen, dann müsste es eine Sprache geben, die nur durch eine Grammatik beschrieben werden kann, dessen Inhaltsmodell m_1 mindestens zwei konkurrierende Nichtterminalzeichen X und Y enthält, für die gilt: $UXW \in L(m_1)$ und $UYV \in L(m_1)$. Die Grammatiken G und G_0 enthalten keine Produktionen für die dieses gilt, da sie konkurrenzverhindernd balanciert sind. Es bleibt die neue Startproduktion $S' \longrightarrow s'SS_0\overline{s}'$, bzw. $S' \longrightarrow s'S_0S\overline{s}'$ zu untersuchen.
Da bei der Konkatenation eine eindeutige Reihenfolge LL_0 oder L_0L entsteht, kann in der Startproduktion S' der Sprache der Konkatenation L' nur genau eine Sequenz der Startsymbole entstehen, entweder SS_0 oder S_0S und somit also nicht USW und US_0V. Dies führt zum Widerspruch zur Annahme. Die konkurrenzverhindernden balancierten Grammatiken sind somit unter Konkatenation abgeschlossen. □

Satz 5.1.44 Die RegDyck Sprachen sind nicht unter der erweiterten Konkatenation abgeschlossen.

Beweis. Angenommen, die RegDyck Sprachen wären unter erweiterter Konkatenation abgeschlossen. Es seien L und L_0 zwei RegDyck Sprachen, die durch die RegDyck Grammatiken $G = (N, T, S, P)$ und $G_0 = (N_0, T_0, S_0, P_0)$ erzeugt werden, wobei

$$\begin{aligned} &L = \{x^n\overline{x}^n \mid n \geq 1\} \text{ und} \\ &L_0 = \{x^m\overline{x}^m \mid m \geq 1\} \\ &LL_0 = L' = \{sx^n\overline{x}^n x^m\overline{x}^m\overline{s} \mid n \geq 1, m \geq 1\} \end{aligned}$$

ergeben.
Die Konkatenation muss durch eine balancierte Grammatik mit der Startproduktion $S \longrightarrow sXX\overline{s}$ erzeugt werden, wobei die Bezeichnung der Nichtterminalzeichen unwichtig ist. Die beiden Nichtterminalzeichen müssen selbsteinbettend sein oder in ein selbsteinbettendes Nichtterminalzeichen ableiten. Es muss gelten: $FA(X) = (x)^+(\overline{x})^+$. Daraus folgt, dass $FA(X)FA(X) \cap D_{(T \cup T_0)} \neq \emptyset$. Die balancierte Grammatik, die die Konkatenation der beiden Sprachen L und L_0 beschreibt, kann somit keine RegDyck Grammatik sein. Dies führt zu einem Widerspruch zur Annahme.
Die konkurrenzverhindernden balancierten Sprachen sind nicht unter erweiterter Konkatenation abgeschlossen.

□

Die Abschlusseigenschaften „erweiterte Iteration“ und „erweiterte Konkatenation“ sind in der Praxis sehr wichtig. Die balancierten Sprachen und die konkurrenzverhindernden balancierten Sprachen sind unter beiden Operationen abgeschlossen. Die XML Sprachen und einfachen balancierten Sprachen sind unter Iteration abgeschlossen. Wie in Kapitel 4 gezeigt wurde, benötigt ein Parsergenerator gleich viel Zeit, um einen Parser für XML Sprachen zu konstruieren wie für einen Parser für konkurrenzverhindernde balancierte Sprachen. Jedoch zeigt sich aus den Abschlusseigenschaften, dass es sinnvoll ist, konkurrenzverhindernde balancierte Sprachen in der Praxis zu verwenden. Darüber hinaus zeigte sich in Kapitel 4, dass die konkurrenzverhindernden balancierten Sprachen ausdrucksmächtiger sind, als die XML Sprachen. Die RegDyck Sprachen sind weder unter erweiterter Iteration noch unter erweiterter Konkatenation abgeschlossen, was die Praxisrelevanz nicht mindert, weil sie unter Vereinigung und Durchschnitt abgeschlossen sind.

5.1.6 Spiegelung

Die Spiegelung wird ähnlich, wie in der Konstruktion aus [HU79] für kontextfreie Grammatiken, durchgeführt.

Satz 5.1.45 Die balancierten Sprachen sind unter Spiegelung abgeschlossen.

Beweis. Es sei $G = (N, T, S, P), T = (A \cup \overline{A})$ eine balancierte Grammatik. Hierbei muss erlaubt werden, dass Endtags vor Starttags stehen dürfen, da jede Produktion

$$X \longrightarrow xm\overline{x} \in P$$

umgedreht wird zu

$$X \longrightarrow \overline{x}mx.$$

Die gespiegelte Grammatik sei $G' = (N', T', S', P')$.
Das Vertauschen von Start- und Endtag erfordert zusätzlich

$$A' = \overline{A}, \overline{A}' = A.$$

In einem nächsten Schritt werden alle Inhaltsmodelle gespiegelt. Es sei

$$X \longrightarrow xm\overline{x} \in P$$

eine Produktion in G mit dem Inhaltsmodell m. Dann sei das Inhaltsmodell der Produktion der gespiegelten Sprache m^R und die Produktion

$$X \longrightarrow \overline{x}mx \in P'.$$

Die Spiegelung des Inhaltsmodells ist möglich, da die regulären Sprachen unter Spiegelung abgeschlossen sind. □

Satz 5.1.46 Die XML Sprachen sind unter Spiegelung abgeschlossen.

Beweis. Der Beweis ist analog zu dem der balancierten Sprachen. □

Satz 5.1.47 Die einfachen balancierten Sprachen sind unter Spiegelung abgeschlossen.

Beweis. Der Beweis ist analog zu dem der balancierten Sprachen. □

Satz 5.1.48 Die konkurrenzverhindernden balancierten Sprachen sind unter Spiegelung abgeschlossen.

Beweis. Der Beweis ist analog zu dem der balancierten Sprachen. □

Satz 5.1.49 Die RegDyck Sprachen sind unter Spiegelung abgeschlossen.

Beweis. Der Beweis ist analog zu dem der balancierten Sprachen. □

5.1.7 Substitution

Es sei $G = (N, T, S, P)$ eine balancierte Grammatik, welche eine Sprache L mit den Wörtern $x \dots \overline{x} \in T$ erzeugt, wobei $x, \overline{x} \in T$. Für alle Terminalzeichen $y \in T$ der Sprache L wird eine neue Sprache L_y festgelegt. Vorerst wird ein Wort w der Sprache L abgeleitet. Danach wird für jedes Zeichen $y \in T$ des Wortes w ein Wort der Sprache L_y erzeugt.
Es sei $a_0 \dots \overline{a_0}$ ein Wort der Sprache L_x und $a_1 \dots \overline{a_1}$ ein Wort der Sprache $L_{\overline{x}}$. Dann würde das resultierende Wort die Form

$$a_0 \dots \overline{a_0} \dots a_1 \dots \overline{a_1}$$

haben.
Dies verstößt allerdings gegen die Definition der balancierten Sprachen.
Es wird ein äußeres schließendes Tagpaar benötigt, so dass

$$sa_0 \dots \overline{a_0} \dots a_1 \dots \overline{a_1 s}$$

erzeugt wird.

Es sei auch hier eine abgewandelte Form der Substitution vorgestellt.

Definition 5.1.29 Die erweiterte Substitution einer balancierten Sprache L wird wie folgt definiert:

Sei $G = (N, T, S, P)$ eine balancierte Grammatik, die L erzeugt.
Für jedes Zeichen $a \in T$ wird eine neue balancierte Sprache L_a definiert. Es wird ein Wort aus L erzeugt und jedes Vorkommen von a durch ein Wort aus L_a ersetzt. Es sei w das entstehende Wort aus der Substitution, dann wird das Wort durch ein disjunktes Paar aus Start- und Endtag $s, \overline{s}$ neu eingeklammert, $sw\overline{s}$.

Satz 5.1.50 Die balancierten Sprachen sind nicht unter der erweiterten Substitution abgeschlossen.

Beweis. Es sei $G = (N, T, \{S\}, P)$ eine balancierte Grammatik mit der einzigen Produktion

$$S \longrightarrow xS?\overline{x}.$$

Es kann ein Wort

$$xxxx\overline{xxxx}$$

erzeugt werden.
Im Allgemeinen würde die Sprache $\{x^n\overline{x}^n \mid n \geq 1\}$ erzeugt werden.
Alle Zeichen x werden durch das Wort $b\overline{b}$ substituiert. Alle Zeichen $\overline{x}$ werden durch das Wort $c\overline{c}$ substituiert. Das entstehende Wort

$$b\overline{b}b\overline{b}b\overline{b}b\overline{b}c\overline{c}c\overline{c}c\overline{c}c\overline{c}$$

wird von einem Tagpaar $s, \overline{s}$ umschlossen und es entsteht

$$s\overline{b}\overline{b}\overline{b}\overline{b}\overline{b}\overline{b}c\overline{c}c\overline{c}c\overline{c}c\overline{c}s\overline{s}.$$

Im Allgemeinen ergibt sich die Sprache $\{s(b\overline{b})^n(c\overline{c})^n\overline{s} \mid n \geq 1\}$.
Angenommen, die balancierten Sprachen wären unter Substitution abgeschlossen, dann muss es eine Startproduktion in der substituierten Grammatik geben:

$$S \longrightarrow sm\overline{s}, m \in Reg(N).$$

Die Startproduktion muss folgendes Inhaltsmodell

$$m = B^nC^n$$

enthalten, da beide Dyckprimes $b\overline{b}$ und $c\overline{c}$ durch zwei verschiedene Nichtterminalzeichen erzeugt werden und beide mit n potenziert sind. Das Inhaltsmodell muss eine kontextfreie Sprache beschreiben. Die Inhaltsmodelle dürfen aber nur reguläre Ausdrücke enthalten. Dies ist ein Widerspruch zur Annahme.
Es ergibt sich, dass die balancierten Sprachen nicht unter der erweiterten Substitution abgeschlossen sind. □

Satz 5.1.51 Die XML Sprachen sind nicht unter der erweiterten Substitution abgeschlossen.

Beweis. Der Beweis ist analog zu dem der balancierten Sprachen. □

Satz 5.1.52 Die einfachen balancierten Sprachen sind nicht unter der erweiterten Substitution abgeschlossen.

Beweis. Der Beweis ist analog zu dem der balancierten Sprachen. □

Satz 5.1.53 Die konkurrenzverhindernden balancierten Sprachen sind nicht unter der erweiterten Substitution abgeschlossen.

Beweis. Der Beweis ist analog zu dem der balancierten Sprachen. □

Satz 5.1.54 Die RegDyck Sprachen sind nicht unter der erweiterten Substitution abgeschlossen.

Beweis. Der Beweis ist analog zu dem der balancierten Sprachen. □

5.1.8 Homomorphismus

Satz 5.1.55 Die balancierten Sprachen sind nicht unter Homomorphismus abgeschlossen.

Beweis. Es sei $G = (N, T, \{S\}, P), T = (A \cup \overline{A})$ eine balancierte Grammatik mit der einzigen Produktion

$$S \longrightarrow x\overline{x}.$$

Der Homomorphismus $h : T \longrightarrow \{x\}$ ist definiert durch

$$\begin{array}{rcl} h(x) & \longrightarrow & x \\ h(\overline{x}) & \longrightarrow & x. \end{array}$$

Die Sprache von G ist $L(G) = \{x\overline{x}\}$. Wenn der Homomorphismus angewendet wird, entsteht die Sprache $\{xx\}$. Diese Sprache kann nicht durch eine balancierte Grammatik erzeugt werden, da jede Produktion der balancierten Grammatiken aus mindestens einem Zeichen $y \in A$ und aus einem Zeichen $\overline{y} \in \overline{A}$ bestehen muss. □

Satz 5.1.56 Die XML Sprachen sind nicht unter Homomorphismus abgeschlossen.

Beweis. Der Beweis ist analog zu dem der balancierten Sprachen. □

Satz 5.1.57 Die einfachen balancierten Sprachen sind nicht unter Homomorphismus abgeschlossen.

Beweis. Der Beweis ist analog zu dem der balancierten Sprachen. □

Satz 5.1.58 Die konkurrenzverhindernden balancierten Sprachen sind nicht unter Homomorphismus abgeschlossen.

Beweis. Der Beweis ist analog zu dem der balancierten Sprachen. □

Satz 5.1.59 Die RegDyck Sprachen sind nicht unter Homomorphismus abgeschlossen.

Beweis. Der Beweis ist analog zu dem der balancierten Sprachen. □

Es hat sich gezeigt, dass alle XML-artigen Sprachen unter Spiegelung abgeschlossen sind, wenn es zulässig ist, dass Starttags und Endtags vertauscht werden. Unter Homomorphismus und Substitution ist keine XML-artige Sprache abgeschlossen.

Zusammenfassend kann gesagt werden, dass aus Sicht der Abschlusseigenschaften die balancierten Sprachen in der Praxis bevorzugt werden sollten. Wenn diese aus Gründen großer Laufzeiten bei der Erstellung eines Parsers nicht gewählt werden können, sollten entweder die konkurrenzverhindernden

balancierten Sprachen oder die RegDyck Sprachen gewählt werden. Sollen die XML Dokumente, mit denen gearbeitet wird, vereinigt werden, dann müssen RegDyck Grammatiken verwendet werden. Werden die Dokumente konkateniert oder iteriert, sind die konkurrenzverhindernden balancierten Sprachen auszuwählen.

Zusammenfassung der Abschlusseigenschaften

Abschlusseigenschaften		**reg. Spr.**	**XMLS**	**EBL**	**KVBL**	**RegDyck Spr.**	**BL**	**det. kon. Spr.**	**kontextf. Spr.**
	Vereinigung	+	-	-	-	+	+	-	+
	Durchschnitt	+	+	+	+	+	+	-	-
	erw. Komplement	+	-	-	-	-	+	+	-
	Spiegelung	+	+	+	+	+	+	-	+
	erw. Konkatenation	+	-	-	+	-	+	-	+
	erw. Iteration	+	+	+	+	-	+	-	+
	erw. Substitution	+	-	-	-	-	-	-	+
	Homomorphismus	+	-	-	-	-	-	-	+

reg. Spr. - reguläre Sprachen, **XMLS** - XML Sprachen, **EBL** - einfache balancierte Sprachen, **KVBL** - konkurrenzverhindernde balancierte Sprachen, **RegDyck Spr.** - RegDyck Sprachen, **BL** - balancierte Sprachen, **det. kon. Spr.** - deterministisch kontextfreie Sprachen, **kontextf. Spr.** - kontextfreie Sprachen

5.1.9 Beispiele für Abschlusseigenschaften von XML-artigen Grammatiken

Erweiterte Konkatenation von balancierten Sprachen

$G_1 = (N_1 = \{S_1, A\}, T_1 = \{x, \overline{x}, a, \overline{a}\}, \{S_1\}, P_1 = \{S_1 \longrightarrow xA^*\overline{x}, A \longrightarrow a\overline{a}\})$
$G_2 = (N_2 = \{S_2, B\}, T_2 = \{x, \overline{x}, b, \overline{b}\}, \{S_2\}, P_2 = \{S_2 \longrightarrow xB?\overline{x}, B \longrightarrow b\overline{b}\})$

$$G_{1,2} = (N_1 \cup N_2 \cup \{S\}, T_1 \cup T_2, S, P_1 \cup P_2 \cup \{S \longrightarrow sS_1S_2\overline{s}\})$$

$G_{1,2}$ ist die Grammatik, die die erweiterte Konkatenation von G_1 und G_2 erzeugt. Sie ist wieder eine balancierte Grammatik. Durch sie kann z.B. das Wort

$$sxa\overline{a}a\overline{a}\overline{x}x\overline{x}\overline{s}$$

erzeugt werden.

□

Erweiterte Konkatenation von konkurrenzverhindernden balancierten Sprachen

$G_1 = (N_1 = \{S_1, A, B\}, T_1 = \{x, \overline{x}, a, \overline{a}, b, \overline{b}\}, S_1, P_1 = \{S_1 \longrightarrow$
$xA^*B\overline{x}, A \longrightarrow a\overline{a}, B \longrightarrow b\overline{b}\})$
$G_2 = (N_2 = \{S_2, D\}, T_2 = \{x, \overline{x}, d, \overline{d}\}, S_2, P_2 = \{S_2 \longrightarrow xD?\overline{x}, D \longrightarrow d\overline{d}\}) :$

$$G_{1,2} = (N_1 \cup N_2 \cup \{S\}, T_1 \cup T_2, S, P_1 \cup P_2 \cup \{S \longrightarrow sS_1S_2\overline{s}\})$$

Die erstellte Grammatik $G_{1,2}$ erzeugt die erweiterte Konkatenation von G_1 und G_2. Sie ist wieder eine konkurrenzverhindernde balancierte Grammatik, da im Inhaltsmodell der Produktion S keine konkurrierende Nichtterminale enthalten sind.
Diese Grammatik erzeugt z.B. das Wort

$$sxa\overline{a}a\overline{a}b\overline{b}\overline{x}xd\overline{d}\overline{x}\overline{s}.$$

□

Erweiterte Konkatenation von einfachen balancierten Sprachen

$G_1 = (N_1 = \{S_1, A\}, T_2 = \{x, \overline{x}, a, \overline{a}\}, \{S_1\}, P_1 = \{S_1 \longrightarrow xAA?\overline{x}, A \longrightarrow$
$a\overline{a}\})$
$G_2 = (N_2 = \{S_2, B\}, T_2 = \{x, \overline{x}, a, \overline{a}\}, \{S_2\}, P_2 = \{S_2 \longrightarrow xB^*\overline{x}, B \longrightarrow a\overline{a}\})$

$$G_{1,2} = (N_1 \cup N_2 \cup \{S\}, T_1 \cup T_2, S, P_1 \cup P_2 \cup \{S \longrightarrow sS_1S_2\overline{s}\})$$

$G_{1,2}$ erzeugt die erweiterte Konkatenation der Grammatiken G_1 und G_2. Allerdings ist diese Grammatik keine einfache balancierte Grammatik, da die Variablen S_1 und S_2 konkurrieren und somit die Bedingung für eine einfache balancierte Grammatik nicht erfüllt wird.

$$G_3 = (N_3 = \{S_3, A\}, T_3 = \{x, \overline{x}, a\overline{a}\}, S_3, P_3 = \{S_3 \longrightarrow xA?\overline{x}, A \longrightarrow a\overline{a}) \\ G_4 = (N_4 = \{S_4, B\}, T_4 = \{y, \overline{y}, b\overline{b}\}, S_4, P_4 = \{S_4 \longrightarrow yB^*\overline{y}, B \longrightarrow b\overline{b}\})$$

$$G_{3,4} = (N_3 \cup N_4 \cup \{S\}, T_3 \cup T_4, S, P_3 \cup P_4 \cup S \longrightarrow sS_3S_4\bar{s})$$

$G_{3,4}$ ist die Konkatenation der Grammatiken G_3 und G_4. Sie ist eine einfache balancierte Grammatik, da S_3 und S_4 im Inhaltsmodell nicht konkurrieren. □

Erweiterte Iteration einer balancierten Sprache

$G_1 = (N_1 = \{S_1, A, B\}, T_1 = \{x, \overline{x}, a, \overline{a}, b, \overline{b}\}, \{S_1\}, P_1 = \{S_1 \longrightarrow xA^*B^+\overline{x}, A \longrightarrow a\overline{a}, B \longrightarrow b\overline{b}\})$

$$G = (N_1 \cup \{S\}, T_1, S, P_1 \cup \{S \longrightarrow sS_1^*\overline{s}\})$$

G erzeugt die iterierte Sprache von G_1. Sie ist wieder eine balancierte Gramatik. Sie erzeugt z.B. das Wort

$$sxa\overline{a}a\overline{a}b\overline{b}\overline{x}xa\overline{a}a\overline{a}b\overline{b}\overline{x}\overline{s}.$$

□

Erweiterte Iteration einer XML Sprache

$G_1 = (N_1 = \{S_1, A\}, T_1 = \{x, \overline{x}, a, \overline{a}\}, S_1, P_1 = \{S_1 \longrightarrow xA?\overline{x}, A \longrightarrow a\overline{a}\})$

$$G = (N_1 \cup \{S\}, T_1, S, P_1 \cup \{S \longrightarrow sX^*\overline{s}\})$$

G erzeugt die iterierte Sprache von G_1. Sie ist wieder eine XML Grammatik, da sie keine Variablen enthält, die konkurrieren. Sie erzeugt z.B. das Wort

$$sxa\overline{ax}xa\overline{axs}.$$

□

5.2 Entscheidbarkeitsprobleme

In dem folgenden Unterkapitel soll beantwortet werden, ob es entscheidbar ist, ob eine gegebene balancierte Grammatik eine Grammatik eines Untertyps der balancierten Sprachen ist. In Kapitel 4 wurde diskutiert, dass die Parsergeneratoren unter Umständen sehr lange Zeit benötigen, um einen Parser für balancierte Sprachen zu konstruieren. Deshalb könnte in der Praxis ein Algorithmus vorgeschaltet werden, der entscheidet, ob eine gegebene balancierte Grammatik eine Grammatik eines Untertyps ist.

Des Weiteren wird überprüft, ob entschieden werden kann, dass eine balancierte Grammatik mehrdeutig ist und ob eine balancierte Sprache eine Teilmenge einer anderen balancierten Sprache ist.

5.2.1 Balancierte Grammatiken und ihre Untertypen

Eine XML Grammatik zeichnet aus, dass jedes Nichtterminalzeichen $X \in N$, zu dem eine Produktion

$$X \longrightarrow xm\overline{x}$$

mit $x \in A$ und $m \in Reg(N)$ existiert, eindeutig durch das Terminalzeichen x identifizierbar ist. D.h. es existiert keine Produktion

$$Y \longrightarrow xm'\overline{x}$$

mit $X \neq Y, Y \in N$ und $m' \in Reg(N)$.
Um zu entscheiden, ob eine gegebene balancierte Grammatik eine XML Grammatik ist, muss genau diese Eigenschaft überprüft werden.
Die Idee des Beweises ist, dass alle Produktionen der gegebenen balancierten Grammatik darauf untersucht werden, ob mindestens zwei verschiedene Nichtterminalzeichen durch dasselbe Terminalzeichen identifiziert werden.

Definition 5.2.30 Sei $G = (N, T, S, P), T = A \cup \overline{A}$ eine balancierte Grammatik. Dann liefert die Abbildung

$$\psi : P \longrightarrow N \times A$$

zu jeder Produktion das erzeugende Nichtterminal- und das identifizierende Terminalzeichen als Paar. Die Abbildung ist einfach zu implementieren. Sie liefert für eine Produktion der Grammatik G das Nichtterminalzeichen, was auf der linken Seite der Produktion steht und das erste Terminalzeichen (Starttag) auf der rechten Seite der Produktion.
Die Abbildung

$$\kappa : P \longrightarrow Reg(N)$$

ordnet einer Produktion ihr Inhaltsmodell zu (pro Terminalzeichen darf höchstens ein Inhaltsmodell existieren). Die Abbildung κ ist implementierbar. Sie entfernt von der rechten Seite einer Produktion lediglich den Start- und den Endtag $(x, \overline{x})$ und gibt den Rest zurück.
Sei $p \in P$ eine Produktion. Da G eine balancierte Grammatik ist, hat p die Form $X \longrightarrow xm\overline{x}$ mit $X \in N, x \in A, \overline{x} \in \overline{A}, m \in Reg(N)$. Dann ist $\psi(p) = (X, x)$ und $\kappa(p) = m$. Hierbei ist zu beachten, dass $m = \lambda$ als Inhaltsmodell zugelassen ist.

Satz 5.2.60 Es ist entscheidbar, ob eine balancierte Grammatik eine XML Grammatik ist.

Beweis. Sei $G = (N, T, S, P)$ eine gegebene balancierte Grammatik mit Produktionen der Form

$$X \longrightarrow xm\overline{x}$$

mit $X \in N, x \in A, \overline{x} \in \overline{A}, m \in Reg(N)$.
Sei

$$K_x = \{X \in N \mid \psi(p) = (X, x), p \in P\}$$

die Menge, die jedem $x \in A$ die Nichtterminale zuordnet, die durch x identifiziert werden. Die Menge K_x lässt sich konstruieren, da die Abbildung ψ, wie weiter oben beschrieben, implementierbar ist und die Mengen A, N und P endlich sind. Nun bleibt zu überprüfen, ob jede Menge K_x entweder kein, genau ein oder mehrere Elemente enthält. Falls alle Mengen kein oder genau ein Element enthalten, so ist die gegebene balancierte Grammatik eine XML Grammatik, andernfalls nicht. □

Eine einfache balancierte Grammatik zeichnet aus, dass im Inhaltsmodell einer Produktion nur nicht konkurrierende Nichtterminale auftreten. Um zu entscheiden, ob eine gegebene balancierte Grammatik eine einfache balancierte Grammatik ist, muss diese Eigenschaft nachgewiesen werden. Es wird überprüft, ob die Inhaltsmodelle aller Produktionen der gegebenen balancierten Grammatik jeweils zwei oder mehr konkurrierende Nichtterminale enthalten.

Satz 5.2.61 Es ist entscheidbar, ob eine balancierte Grammatik eine einfache balancierte Grammatik ist.

Beweis. Sei $G = (N, T, S, P), T = A \cup \overline{A}$ eine gegebene balancierte Grammatik. Die Produktionen haben die Form

$$X \longrightarrow xm\overline{x}$$

mit $X \in N, x \in A, \overline{x} \in \overline{A}, m \in Reg(N)$. Sei

$$K_Y = \{X \in N \setminus \{Y\} \mid \psi(p) = (X, x), \psi(p') = (Y, x), p, p' \in P, x \in A\}$$

die Menge der mit $Y \in N$ konkurrierenden Nichtterminale. Die Menge K_Y ist ebenfalls konstruierbar, da die Abbildung ψ implementierbar ist und die Mengen A, N und P endlich sind.
Sei

$$\pi_i : Reg(N) \longrightarrow N$$

die Projektion auf die i-te Komponente des übergebenen Ausdrucks. [Die erste Stelle hat hier den Index 1.] Bei der Indizierung und der Länge eines Ausdrucks werden die Operatoren $+, *, |$ und ? ignoriert. Die Länge n_m eines Ausdrucks m ist also die Anzahl der in m auftretenden Zeichen $X \in N$.
Sei

$$K_{Y,p} = \{X \in N \mid \pi_i(\kappa(p)) = X, \pi_j(\kappa(p)) = Y, X \neq Y, i, j \in \{1, \ldots, n_{\kappa(p)}\}\}$$

die Menge aller Nichtterminale X des Inhaltsmodells der Produktion p, in dem sowohl X als auch das fest gewählte Y vorkommen. Die Abbildung κ und die Projektion π sind implementierbar. Die entstehende Menge $K_{Y,p}$ ist endlich, da κ einen regulären Ausdruck und π eines der Zeichen des regulären Ausdrucks zurückgibt.
Bilde nun den Durchschnitt

$$K_Y \cap K_{Y,p}$$

für alle $Y \in N$ und für alle $p \in P$. Falls alle Durchschnitte leer sind, so ist die gegebene balancierte Grammatik eine einfache balancierte Grammatik, andernfalls nicht. □

Eine konkurrenzverhindernde balancierte Grammatik zeichnet aus, dass für zwei konkurrierende Nichtterminale X und Y, die in demselben Inhaltsmodell einer Produktion auftreten, für alle Kombinationen $U, V, W \in N^*$ gilt: Aus einem Inhaltsmodell m folgt

- nicht sowohl $UXV \in L(m)$
- als auch $UYW \in L(m)$.

Um zu entscheiden, ob eine gegebene balancierte Grammatik eine konkurrenzverhindernde balancierte Grammatik ist, muss genau diese Eigenschaft überprüft werden.
Die Idee des Beweises ist, dass für alle Inhaltsmodelle zunächst geprüft wird, ob zwei konkurrierende Nichtterminale auftreten. Falls dies der Fall ist, wird noch überprüft, ob bei gleicher Anfangssequenz im Inhaltsmodell durch die konkurrierenden Nichtterminale unterschiedliche Worte abgeleitet werden können.

Satz 5.2.62 Es ist entscheidbar, ob eine balancierte Grammatik eine konkurrenzverhindernde balancierte Grammatik ist.

Beweis. Sei $G = (N, T, S, P), T = A \cup \overline{A}$ eine gegebene balancierte Grammatik. Produktionen haben die Form

$$X \longrightarrow xm\overline{x}$$

mit $X \in N, x \in A, \overline{x} \in \overline{A}, m \in Reg(N)$.
Sei

$$K_Y = \{X \in N \setminus \{Y\} \mid \psi(p) = (X, x), \psi(p') = (Y, x), p, p' \in P, x \in A\}$$

die Menge der mit $Y \in N$ konkurrierenden Nichtterminale.

Es sei $W_{p,Y}$ die Menge aller Teilworte aus $L(r), r = \kappa(p), r \in Reg(N)$, die mit $Y \in N$ enden.

$$W_{p,Y} = \{u \mid u = \alpha \wedge \alpha Y \beta \in L(r) \wedge r = \kappa(p), \alpha, \beta \in N^*\}$$

Es werden nun die Durchschnitte gebildet

$$W_{p,Y} \cap W_{p,Z}$$

für alle $Y \in N$ mit $Z \in K_Y$ und für alle Produktionen $p \in P$.

Sind alle Durchschnitte leer (enthalten sie also nicht einmal das leere Wort λ), so ist die gegebene balancierte Grammatik eine konkurrenzverhindernde balancierte Grammatik, andernfalls nicht. □

5.2.2 Inklusion

Satz 5.2.63 Seien L_1 und L_2 zwei balancierte Sprachen. Es ist entscheidbar, ob $L_1 \subseteq L_2$.

Beweis. Seien G_1 und G_2 zwei balancierte Grammatiken und L_1 bzw. L_2 die von ihnen erzeugten Sprachen. Die balancierten Sprachen sind deterministisch kontextfrei und somit ist es entscheidbar, ob zwei balancierte Sprachen äquivalent sind. Zusätzlich sind die balancierten Sprachen unter Durchschnitt abgeschlossen.
Bilde nun den Schnitt L der beiden erzeugten Sprachen

$$L = L_1 \cap L_2.$$

Da die Familie der balancierten Sprachen unter Schnittbildung effektiv abgeschlossen ist, existiert eine balancierte Grammatik G, die L erzeugt.
Ist L äquivalent zu L_1, so gilt $L_1 \subseteq L_2$. Ist L äquivalent zu L_2, so ist $L_2 \subseteq L_1$.
Wenn L zu keiner der beiden Sprachen äquivalent ist, so sind die Sprachen L_1 und L_2 nicht ineinander enthalten. □

5.2.3 Mehrdeutigkeit

Definition 5.2.31 Eine balancierte Grammatik G ist mehrdeutig, wenn es für die indizierte balancierte Grammatik G' ein Wort in $L(G)$ gibt, wofür die Grammatik G' mehr als einen Ableitungsbaum besitzt.

Satz 5.2.64 Mit balancierten Grammatiken können mehrdeutige Grammatiken beschrieben werden.

Mehrdeutigkeiten Es gibt verschiedene Ursachen der Mehrdeutigkeit.

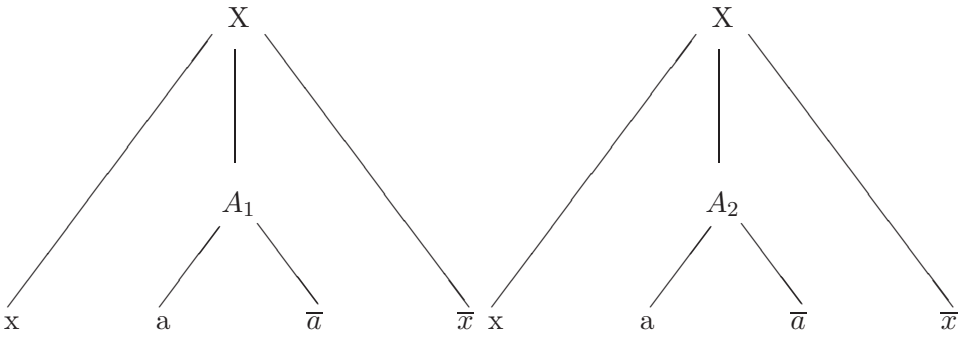
Beispiel 5.2.27 Zum einen kann die Mehrdeutigkeit durch ein Inhaltsmodell, welches aus einem mehrdeutigen regulären Ausdruck besteht, hervorgerufen werden. Es sei $G = (N, T, \{X\}, P)$ eine balancierte Grammatik mit folgenden Produktionen:

$$\begin{aligned} X &\longrightarrow x(A \mid B)^* A^* \overline{x} \\ A &\longrightarrow a\overline{a} \\ B &\longrightarrow b\overline{b}. \end{aligned}$$

Die Produktion von X wird nun indiziert.

$$X \longrightarrow x_1(A_1 \mid B_1)^* A_2^* \overline{x}_1$$

Für das Wort $xa\overline{ax}$ lassen sich zwei verschiedene Ableitungen erzeugen: $x_1a_1\overline{a}_1\overline{x}_1$,$x_1a_2\overline{a}_2\overline{x}_1$.



Beispiel 5.2.28 Die zweite Form von Mehrdeutigkeiten wird durch eine spezielle Eigenschaft der balancierten Grammatiken hervorgerufen. Balancierte Grammatiken erlauben, im Gegensatz zu ihren Untertypen, jegliche Art von Produktionen der Form

$$X \longrightarrow am\overline{a}.$$

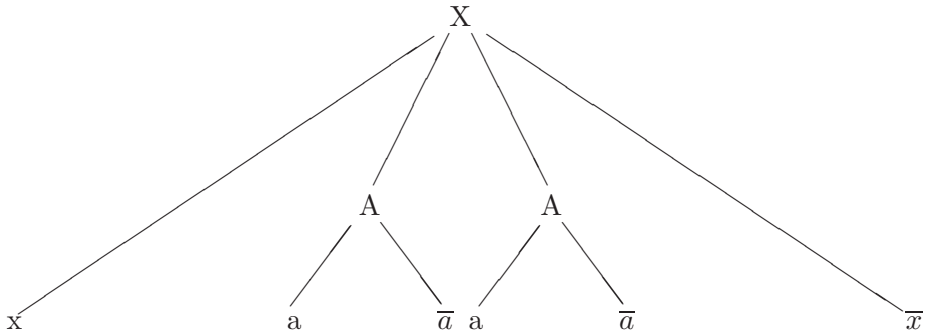
Sie erlauben somit auch eine beliebige Konkatenation von konkurrierenden Nichtterminalen.
Durch den Satz von [BB02a], dass aus einer balancierten Sprache eindeutig eine balancierte Grammatik erzeugt werden kann, wird die Sprache

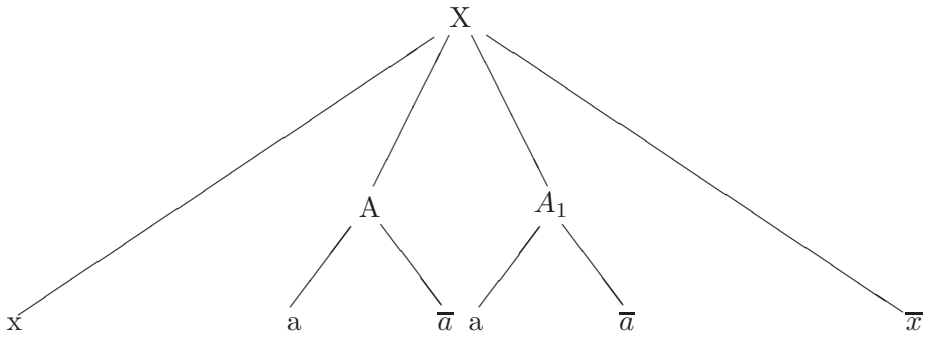
$$(x(a(c\overline{c})?\overline{a})^*(a(d\overline{d})?\overline{a})^*\overline{x})$$

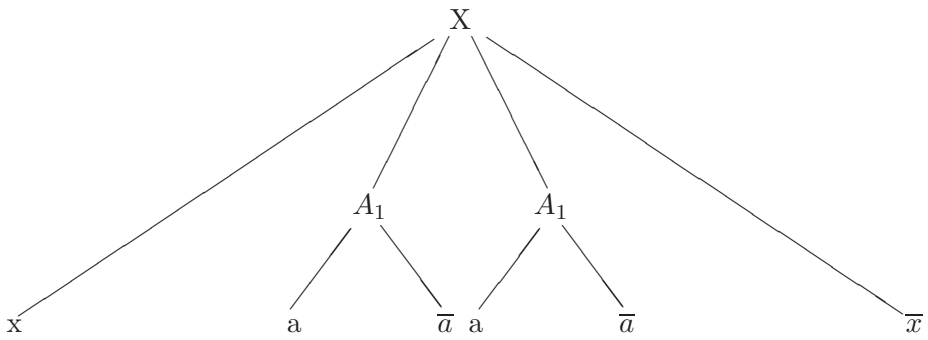
durch die balancierte Grammatik $G = (N, T, \{X\}, P)$

$$\begin{aligned} P = \{ \quad & X \longrightarrow xA^*A_1^*\overline{x} \\ & A \longrightarrow aC?\overline{a} \\ & A_1 \longrightarrow aD?\overline{a} \\ & C \longrightarrow c\overline{c} \\ & D \longrightarrow d\overline{d}\} \end{aligned}$$

beschrieben.
Für ein Wort $w = xa\overline{a}a\overline{a}\overline{x}$ gibt es folgende drei Ableitungsbäume:







Die Grammatik ist mehrdeutig.

Mehrdeutigkeiten führen zu Problemen beim Parsen, da für ein gegebenes Wort nicht klar ist, welcher Parsebaum aufgebaut werden muss. Das kann zu hohen Laufzeiten führen.
Eine wichtige Eigenschaft einer Sprache in Bezug auf Mehrdeutigkeit ist es, ob eine Sprache inhärent mehrdeutig ist. Das bedeutet, dass die Sprache nur durch mehrdeutige Grammatiken erzeugt werden kann. Für die Einschränkung, dass die Grammatik eine balancierte Grammatik sein muss, zeigte das letzte Beispiel, dass es Sprachen gibt, für die nur mehrdeutige balancierte Grammatiken gefunden werden können.

Satz 5.2.65 Es ist entscheidbar, ob eine balancierte Grammatik eindeutig ist.

Beweis. Sei $G = (N, T, S, P)T = A \cup \overline{A}$ eine gegebene balancierte Grammatik. Elemente aus P haben die Form

$$X \longrightarrow xm\overline{x}$$

mit $X \in N, x \in A, \overline{x} \in \overline{A}$ und $m \in Reg(N)$.
Um zu entscheiden, ob eine balancierte Grammatik eindeutig ist, muss geprüft werden, ob zu jedem beliebigen Wort aus der von ihr erzeugten Sprache nur genau ein Ableitungsbaum existiert.
Betrachte hierfür zunächst die Inhaltsmodelle aller Produktionen. Sie sind reguläre Ausdrücke. Für die regulären Ausdrücke ist es entscheidbar, ob sie eindeutige oder mehrdeutige reguläre Ausdrücke sind [BEGO71]. Falls ein Inhaltsmodell mehrdeutig ist, so folgt direkt, dass die Grammatik mehrdeutig ist, wie aus dem Beispiel 5.2.27 trivial zu folgern ist.
Seien also die Inhaltsmodelle alle eindeutige reguläre Ausdrücke.
Betrachte nun die Sprachen, die von jedem einzelnen Nichtterminalzeichen erzeugt werden. Sei $X \in N$. Die von X erzeugte Sprache ist

$$L_X = \{w \in (A \cup \overline{A})^* \mid X \Rightarrow^* w\}.$$

Bilde die Schnitte

$$L_X \cap L_Y$$

für alle $X, Y \in N, X \neq Y$. Wenn für zwei Nichtterminale der Schnitt leer ist, können sie nicht dasselbe Wort erzeugen. Also führen sie in Kombination miteinander im Inhaltsmodell einer Produktion nicht zur Mehrdeutigkeit.
Ist der Schnitt nicht leer, so kann aus diesen zwei Nichtterminalen dasselbe Wort erzeugt werden.
Sei

$$N_X = \{Y \in N \mid L_X \cap L_Y \neq \emptyset, X \neq Y\}$$

die Menge der Nichtterminale, die in Kombination mit X zu Mehrdeutigkeit führen können. Es muss also noch überprüft werden, ob zu zwei Nichtterminalen $X \in N, Y \in N_X$ und einem Inhaltsmodell m Sequenzen $U, U', V, V' \in Reg(N)$ existieren, mit Ableitungen

$$m \Rightarrow^* UXV \Rightarrow^* w$$

und

$$m \Rightarrow^* U'YV' \Rightarrow^* w$$

für ein $w \in (A \cup \overline{A})^*$. Da X und Y dasselbe Wort ableiten können, muss

$$U \longrightarrow u, U' \longrightarrow u, V \longrightarrow v, V' \longrightarrow v$$

gelten für $u, v \in (A \cup \overline{A})^*$. Dann gibt es für das Wort w mehrere Ableitungen und die Grammatik ist mehrdeutig.
Sei

$$A_{m,X} = \{U \in Reg(N) \mid m = UXW, W \in Reg(N)\}$$

die Menge aller Nichtterminalkombinationen, die im Inhaltsmodell m vor einem gegebenen $X \in N$ auftreten können.
Sei

$$E_{m,X} = \{W \in Reg(N) \mid m = UXW, U \in Reg(N)\}$$

die Menge aller Nichtterminalkombinationen, die im Inhaltsmodell m nach einem $X \in N$ auftreten können.
Sei G_U die balancierte Grammatik zu $U \in Reg(N)$ mit $P_U = \{X \longrightarrow xU\overline{x}\}$.
Sei $U \in A_{m,X}$ und $U' \in A_{m,Y}$ und G_U und $G_{U'}$ die balancierten Grammatiken wie oben beschrieben. Bilde nun die Schnitte

$$L(G_U) \cap L(G_{U'}).$$

Sind alle Durchschnitte leer, so existieren keine Nichtterminalkombinationen, die vor X bzw. Y in den Inhaltsmodellen auftreten und gleiche Worte erzeugen können. Damit kann keine Mehrdeutigkeit auftreten.
Sei ein Schnitt nicht leer, es gelte also

$$L(G_U) \cap L(G_{U'}) \neq \emptyset$$

für ein $U \in A_{m,X}$ und ein $U' \in A_{m,Y}$.
Betrachte nun die Durchschnitte

$$L(G_W) \cap L(G_{W'})$$

für alle $W \in E_{m,X}$ und $W' \in E_{m,Y}$ und deren Grammatiken G_W bzw. $G_{W'}$.
Sind diese Schnitte leer, so kann keine Mehrdeutigkeit auftreten.
Ist mindestens einer der Schnitte nicht leer, so ist die Grammatik mehrdeutig, da dann für ein Wort aus $L(G)$ mehrere Ableitungen existieren. □

Auch in den Entscheidbarkeitsfragen zeigt sich, dass die balancierten Sprachen sehr nutzbar für die Praxis sind. Es kann entschieden werden, ob eine balancierte Grammatik eine Grammatik eines Untertyps ist, ob eine balancierte Sprache eine Teilmenge einer anderen Sprache ist und ob eine balancierte Grammatik mehrdeutig ist.
Konkrete Anwendungen könnten sein, dass vor der Generierung eines Parsers für ein Beschreibungsmodell entschieden wird, von welchem Typ die Grammatik ist und danach entschieden wird, welche Art von Parser konstruiert wird.

5.2.4 Beispiele zu Mehrdeutigkeit

Beispiel 1:

Das Inhaltsmodell ist ein mehrdeutiger regulärer Ausdruck.
Seien

$$X \longrightarrow xA^*A^*\overline{x}, A \longrightarrow a\overline{a}$$

Produktionen einer balancierten Grammatik $G = (N, T, \{X\}, P)$.
Das Wort $xa\overline{a}a\overline{a}\overline{x}$ kann durch diese Produktionen erzeugt werden. Die Kombination $a\overline{a}a\overline{a}$ kann sowohl aus A^* als auch aus A^*A^* abgeleitet werden. Sei $xA_1^*A_2^*\overline{x}$ der indizierte Ausdruck der rechten Seite der Startproduktion. Somit existieren die Ableitungen $X \longrightarrow xA_1A_1\overline{x} \longrightarrow xa_1\overline{a_1}a_1\overline{a_1}\overline{x}$ und $X \longrightarrow xA_1A_2\overline{x} \longrightarrow xa_1\overline{a_1}a_2\overline{a_2}\overline{x}$ für das Wort $xa\overline{a}a\overline{a}\overline{x}$.

□

Beispiel 2:

Die Inhaltsmodelle sind eindeutige reguläre Ausdrücke
Seien

$$S \longrightarrow sXY\overline{s}, X \longrightarrow xA^*B^*\overline{x}, Y \longrightarrow xA^*D^*\overline{x},$$
$$A \longrightarrow a\overline{a}, B \longrightarrow b\overline{b}, D \longrightarrow d\overline{d}$$

Produktionen einer balancierten Grammatik $G = (N, T, \{S\}, P)$.
Das Wort $sx\overline{x}s$ kann durch diese Produktionen erzeugt werden. Es existieren die Ableitungen $X \longrightarrow x\overline{x}$ und $Y \longrightarrow x\overline{x}$ für das Teilwort $x\overline{x}$.

□

Zusammenfassung der Entscheidbarkeitsfragen

Entscheidbarkeit		**reguläre Sprachen**	**balancierte Sprachen**	**det. kon. Spr.**	**kontextf. Spr.**
	Wortproblem	+	+	+	+
	Äquivalenz	+	+	+	-
	Leerheit	+	+	+	+
	Unendlichkeit	+	+	+	+
	Inklusion	+	+	-	-
	Mehrdeutigkeit	+	+	+	-
	XML S.		+		
	einfache bal. S.		+		
	konkurrenzverhindernde bal. S.		+		
	RegDyck G.		+		

det. kon. Spr. - deterministisch kontextfreie Sprachen, **kontextf. Spr.** - kontextfreie Sprachen, **einfache bal. S.** - einfache balancierte Sprachen, **konkurrenzverhindernde bal. S.** - konkurrenzverhindernde balancierte Sprachen

Literaturverzeichnis

[ASU99a] Aho, A. ; Sethi, R. ; Ullmann, J.: *Compilerbau Teil 1.* 2. Auflage. München : Oldenbourg Wissenschaftsverlag, 1999

[ASU99b] Aho, A. ; Sethi, R. ; Ullmann, J.: *Compilerbau Teil 2.* 2. Auflage. München : Oldenbourg Wissenschaftsverlag, 1999

[BB00] Berstel, J. ; Boasson, L.: XML grammars. In: *Mathematical Foundations of Computer Science, LNCS* Bd. 1893, Springer, 2000, S. 182–191

[BB02a] Berstel, J. ; Boasson, L.: Balanced Grammars and Their Languages. In: *Formal and Natural Computing, LNCS* Bd. 2300, Springer, 2002, S. 3–25

[BB02b] Berstel, J. ; Boasson, L.: Formal properties of XML grammars and languages. *Acta Informatica* 38 (2002), S. 649–671

[BEGO71] Book, R. ; Even, S. ; Greibach, S. ; Ott, G.: Ambiguity in Graphs and Expressions. *IEEE Transactions on Computers* 66 (1971), S. 451–472

[BK93a] Brüggemann-Klein, A.: *Formal models in document processing*, Faculty of Mathematics at the University of Freiburg, Dissertation, 1993

[BK93b] Brüggemann-Klein, A.: Unambiguity of extended regular expressions in SGML document grammars / Technical University of Munich. 1993. – Forschungsbericht

[BKW92] Brüggemann-Klein, A. ; Wood, D.: Deterministic regular languages. In: *9th Annual Symposium on Theoretical Aspects of Computer Science, LNCS* Bd. 577, Springer, 1992, S. 173–184

[BKW98] Brüggemann-Klein, A. ; Wood, D.: One-Unambiguous Regular Languages. *Information and Computation* 142 (1998), S. 182–206

[Bos05] http://www.javaworld.com/javaworld/javaone00/j1-00-bosak.html

[CDGJ97] Comon, H. ; Dauchet, M. ; Gilleron, R. ; Jacquemard, F.: *Tree Automata Techniques and Apllications*. Forschungsbericht, 1997

[Claa] http://www.thaiopensource.com/trex

[Clab] http://www.oasis.open.org/committees/relax-ng/tutorial.html

[DFS99] Deutsch, A. ; Fernandez, M. ; Suciu, D.: Storing semistructured data with STORED. In: *Proceedings of the 1999 ACM SIGMOD International Conference on Management of Data*, 1999, S. 431–442

[DOM07] http://www.w3.org/DOM/

[DTD07] http://www.w3.org/TR/html401/sgml/dtd.html

[FK99] Florescu, D. ; Kossmann, D.: Storing and Querying XML Data using RDBMS. In: *IEEE Data Bulletin, VLDB Journal* 14 (1999), Nr. 1, S. 30–49

[GH67] Ginsburg, S. ; Harrison, M.: Bracketed context-free Languages. *Journal of Computer and System Sciences* 1 (1967), S. 1–23

[GP00] Goldfarb, C. ; Prescod, P.: *The XML Handbook*. 2. Auflage. London : Prentice Hall, 2000

[Har99] Harold, E.: *XML Bible*. IDG Books Worldwide, 1999

[HU79] Hopcroft, J. ; Ullman, J.: *Introduction to Automata Theory, Language, and Computation*. München : Addion-Wesley, 1979

[HW07] Han, Yo-Sub ; Wood, D.: Generalizations of One-deterministic Regular Languages. In: *Conference on Language and Automata Theory and Applications (LATA'07), Universitat Rovira I Virgili*, 2007, S. 273–285

[Knu67] Knuth, D.: A characterization of parantenthesis languages. *Information Control* 11 (1967), S. 269–289

[LC00] Lee, D. ; Chu, W. W.: Constraints-Preserving Transformation from XML Document Type Definition to Relational Schema. In: *International Conference on Conceptual Modeling / the Entity Relationship Approach, LNCS* Bd. 1920/2000, Springer, 2000, S. 323–338

[LMCC02] Lee, D. ; Mani, M. ; Chiu, F. ; Chu, W. W.: NeT and CoT: Translating Relational Schemas to XML Schemas using Semantic Constraints. In: *ACM International Conference on Information and Knowledge Management*, 2002

[LMM00] Lee, D. ; Mani, M. ; Murata, M.: "Reasoning about XML Schema Languages using Formal Language Theory" / IBM Almaden Research Center. 2000. – Forschungsbericht

[McN67] McNaughton, R.: Paranthesis grammars. *Journal of the ACM* 14 (1967), S. 490–500

[Meg] `http://www.megginson.com/SAX/index.html`

[ML02] Mani, M. ; Lee, D.: XML to Relational Conversion using Theory of Regular Tree Grammars. In: *Proceedings of the VLDB 2002 Workshop EEXTT and CAiSE 2002 Workshop DTWeb on Efficiency and Effectiveness of XML Tools and Techniques and Data Integration over the Web-Revised Papers, LNCS, Springer* Bd. 2590, 2002, S. 81–93

[MLM01a] Mani, M. ; Lee, D. ; Muntz, R. R.: Semantic Data Modeling Using XML Schemas. In: *LNCS,Springer* Bd. 2224,2001, S. 149–163

[MLM01b] Murata, M. ; Lee, D. ; Mani, M.: Taxonomy of XML Schema Languages using Formal Language Theory / IBM Almaden Research Center. 2001. – Forschungsbericht

[SAX07] `http://www.saxproject.org/`

[See03] Seemann, M.: *Native XML Datenbanken im Einsatz*. Paderborn : Software and Support Verlag, 2003

[SSB+01] Shanmugasundaram, J. ; Shekita, E. ; Barr, R. ; Carey, M. ; Lindsay, B. ; Pirahesh, H. ; Reinwald, B.: Efficiently publishing relational data as XML documents. *VLDB Journal: Very Large Data Bases* 10 (2001), S. 133–154

[SSS88] Sippu, S. ; Soisalon-Soininen, E.: *Parsing Theory*. Bd. 1. Berlin : Springer, 1988

[ST04] Sapounakis, A. ; Tsikouras, P.: On k-colored Motzkin words. *Journal of Integer Sequences* 4 (2004)

[STZ+99] Shanmugasundaram, J. ; Tufte, K. ; Zhang, C. ; DeWitt, D. J. ; Naughton, J. F.: Relational Databases for Querying

XML Documents: Limitations and Opportunities. In: *Proceedings of 25th International Conference on Very Large Data Bases, VLDB Journal*, 1999, S. 302–314

[WWW] `http://www.w3.org/TR/REC-xml`

[XML05] `http://www.w3.org/TR/2004/REC-xml-20040204/`

[XML07] `http://www.w3.org/XML/Schema`