Entwurf und Implementierung eines Scanner-Generatorsystems nach der Methode von Gluschkow

DIPLOMARBEIT

zur Erlangung des akademischen Grades

Diplom-Informatiker (FH)

vorgelegt von Sabine Ludvik, Halle a. d. Saale

geboren am 04.07.1969 in Zwickau

Matrikelnummer: 11720

Hochschule Harz

FB Automatisierung und Informatik

Sommersemester 2007

1. Prüfer: Prof. Dr. Bernhard Zimmermann

2. Prüfer: Dipl.-Ing. Ute Kretschmer

Sperrvermerk

Der Inhalt der Arbeit darf Dritten ohne Genehmigung der Ausbildungsstätte nicht zugänglich gemacht werden.

Inhaltsverzeichnis

Vorwort

"Nach mehreren Berufsjahren hast du ein Studium nicht mehr nötig. Such dir doch eine Projektarbeit zur Vertiefung deiner Programmierkenntnisse."

Diesen und ähnliche Ratschläge erhielt ich, als ich im Jahr 2004 vor der Entscheidung pro oder contra Fernstudium Informatik stand. Ich habe mich dafür entschieden – und damit Türen zu Wissensbereichen aufgestoßen, die ich im leistungsorientierten und schnelllebigen Alltagsgeschäft wohl eher nicht entdeckt hätte. Es ist daher nicht zufällig, dass diese Arbeit sich mit einer Thematik beschäftigt, die nicht vordergründig "aktuell" ist, jedoch innerhalb des weiten Bereiches "Compilerbau" einen wichtigen Beitrag für die heutigen Möglichkeiten in der Softwareentwicklung geleistet hat.

Ich möchte Prof. Dr. Zimmermann von ganzem Herzen für dieses Diplomthema danken, welches sich mit fortschreitender Einarbeitung als geradezu maßgeschneidert herausstellte und mir außer dem wohl unvermeidlichen Stress auch viel Freude bereitet hat. Genauso herzlich möchte ich Frau Kretschmer für ihre Unterstützung im Modul "Modelle der Informatik" danken, denn während dieser Tage wurde mein Interesse für das Fach nachhaltig geweckt.

Meine Mitstudenten Ulrich Szagun und Holger Sanio und natürlich mein Mann Jan-Peter Flato halfen mir von Zeit zu Zeit, die nötige Motivation wieder zu finden – auch dafür einen herzlichen Dank.

1 Einleitung

1.1 Aufgabenstellung

Inhalt dieser Diplomarbeit ist die Erstellung eines Software-Systems zur Generierung endlicher deterministischer Automaten nach der von Viktor Michailovitsch Gluschkow 1961 (bzw. 1963 in deutscher Sprache) erstmalig vorgestellten Methode. Dabei sollten von der Analyse über den Entwurf bis zur Implementierung in C# die wesentlichen Schritte der Softwareentwicklung enthalten sein.

Die Software soll folgende grundlegende Funktionalität bieten:

- Die Interaktion mit dem Anwender soll über eine integrierte grafische Oberfläche erfolgen.
- Der deterministische endliche Automat – im folgenden DEA genannt – soll aus einem vom Anwender eingegebenen regulären Ausdruck erzeugt werden und sowohl die Indizierung des regulären Ausdrucks als auch die Zustandsüberführungsfunktion in δ-Notation enthalten.
- Jeder einmal generierte DEA soll beliebig viele Simulationen von Eingabezeichenfolgen ermöglichen, die der Anwender schrittweise mit entsprechender Visualisierung der Zwischenergebnisse durchlaufen kann.
- Das Software-System soll eine Bedienungsanleitung und Dokumentationen enthalten.

1.2 Aufbau der Arbeit

Diese Arbeit beinhaltet einen kurzen theoretischen Einblick zum Thema deterministische endliche Automaten sowie einen größeren Teil, der sich der Erstellung des Softwaresystems widmet.

Diese praktischen Kapitel orientieren sich dabei an den Schritten des Softwareentwicklungsprozesses:

- Anforderungsanalyse
- Entwurf
- Implementierung
- Test

Den Abschluss bilden Erläuterungen zur Dokumentation der Softwarequellen und zur Benutzeranleitung.

2 Theorie des deterministischen endlichen Automaten

Der Begriff des Automaten wurde etwa in der Mitte des vorigen Jahrhunderts als Ergebnis einer sinnvollen Abstraktion der damals real existierenden Rechentechnik zum Forschungsgegenstand erhoben. Die Mehrzahl der Arbeiten, die sich zu dem Zeitpunkt mit der Theorie der Automaten beschäftigten, ging von praktischen Problemstellungen aus. 1961 legte Viktor Michailovitsch Gluschkow eine Systematisierung und Zusammenfassung der abstrakt-algebraischen Seite der Theorie der Automaten vor, die das noch junge Forschungsgebiet beflügelte. Neben den theoretischen Betrachtungen enthält diese Arbeit auch eine sehr praktische Methode der Indizierung regulärer Ausdrücke, die die Erstellung endlicher Automaten ermöglicht.

2.1 Chomsky-Hierarchie der formalen Sprachen

Wenige Jahre vor dem Erscheinen der Arbeit von Gluschkow über die Indizierung regulärer Ausdrücke stellt 1956 Noam Chomsky seine Hierarchie von Klassen formaler Sprachen und Grammatiken vor. Diese beschreibt vier Grammatik-Typen, mittels derer dazugehörige formale Sprachen erzeugt werden können, wie in Tabelle 1 beschrieben.

Grammatik	*Erzeugte Sprache*
Typ-0	rekursiv aufzählbar
Typ-1	kontextsensitiv
Typ-2	kontextfrei
Typ-3	regulär

Tabelle 1: Chomsky-Hierarchie

Dabei repräsentieren Typ-3-Sprachen diejenige Sprachklasse innerhalb der Hierarchie, die die größten Einschränkungen hinsichtlich der sie definierenden Grammatiken aufweist. Typ-3-Grammatiken nennt man auch reguläre Grammatiken, da die Eigenschaft der "Regularität" sie von den weniger restriktiven Grammatiken unterscheidet. Dabei versteht man unter Regularität das Fehlen selbsteinbettender Nichtterminalsymbole.

Eine Grammatik G = (N, T, P, S) mit

N: Menge der Nichtterminalsymbole,

T: Menge der Terminalsymbole,

P: Menge der Produktionsregeln

S: das Startsymbol

heißt Typ-3- oder reguläre Grammatik genau dann, wenn gilt:

- für alle Produktionen $p \rightarrow q \in P$ $(p \in N)$

  entweder $q = t$ oder $q = tn$ (rechtslinear) mit $t \in T^*$ und $n \in N$

oder

- für alle Produktionen $p \rightarrow q \in P$ $(p \in N)$

  entweder $q = t$ oder $q = nt$ (linkslinear) mit $t \in T^*$ und $n \in N$

Formel 1: Definition reguläre Grammatik

(Skript zum Modul Modelle der Informatik, IIN, B. Zimmermann)

2.2 Regulärer Ausdruck

Außer durch eine reguläre Grammatik kann eine reguläre Sprache auch durch einen regulären Ausdruck beschrieben werden. Ein regulärer Ausdruck beschreibt Mengen bzw. Untermengen von Zeichenketten mittels eines geschlossenen Ausdrucks.

Dabei gilt folgende Definition für ein Eingabealphabet T, in dem die Zeichen [,] , « , » , | und ε nicht vorkommen:

(1) Für $t \in T \cup \{\varepsilon\}$ ist t ein regulärer Ausdruck.

(2) Wenn A und B reguläre Ausdrücke sind, dann sind auch

$A \cdot B$, «A», [A] und A | B reguläre Ausdrücke.

(3) Nur Ausdrücke, die mit Hilfe von (1) und (2) gebildet werden, sind regulär.

Formel 2: Definition (Syntax) regulärer Ausdruck

(Skript zum Modul Modelle der Informatik, IIN, B. Zimmermann)

In der folgenden Definition wird induktiv über dem syntaktischen Aufbau der regulären Ausdrücke deren Semantik, nämlich die Menge von Zeichenfolgen, die die Sprache des regulären Ausdrucks darstellen, festgelegt:

(1) | steht für die Mengenvereinigung: $A \mid B := A \cup B$

(2) $\cdot$ steht für die Mengenkonkatenation und wird in den Ausdrücken weggelassen:

- $A \cdot B := ab$ mit $a \in A$ und $b \in B$
- die Konkatenation hat Vorrang vor der Vereinigung

(3) «A» steht für die Kleene'sche Sternoperation, d.h. eine beliebig häufige Konkatenation von Wörtern aus der Sprache, vereinigt mit dem leeren Wort:

- $A^0 = \{\varepsilon\}$
- $A^i = A \cdot A^i$ mit $i > 0$
- die Kleene'sche Sternoperation hat Vorrang vor der Konkatenation und der Vereinigung

(4) Die eckigen Klammern [A] bestimmen die Anwendungsreihenfolge, sie haben höchste Priorität.

Formel 3: Bedeutung (Semantik) regulärer Ausdruck

(Skript zum Modul Modelle der Informatik, IIN, B. Zimmermann)

2.3 Deterministischer Endlicher Automat

Für jede Sprachklasse innerhalb der Chomsky-Hierarchie existiert eine abstrakte Maschine, die genau diese Sprachen erkennt, wie aus Tabelle 2 zu entnehmen ist.

Grammatik	*Erzeugte Sprache*	*Akzeptor-Automat*
Typ-0	rekursiv aufzählbar	Turingmaschine
Typ-1	kontextsensitiv	Linear beschränkter Automat
Typ-2	kontextfrei	Kellerautomat
Typ-3	regulär	Endlicher Automat

Tabelle 2: Chomsky-Hierarchie mit Akzeptoren

Ein endlicher Automat ist also eine abstrakte Maschine, die als Akzeptor für formale Sprachen vom Typ-3 fungiert und damit entscheidet, ob eine bestimmte Eingabezeichenfolge zu der von dem endlichen Automaten definierten regulären Sprache gehört oder nicht.

Dabei nimmt der Automat unter Eingabe eines bestimmten Zeichens einen bestimmten Folgezustand ein. Ist dieser eindeutig festgelegt, handelt es sich um einen deterministischen, ist er es nicht, um einen nichtdeterministischen Automaten.

Im Unterschied zu weiteren Methoden der Erzeugung endlicher Automaten ermöglicht die Vorgehensweise nach Gluschkow das direkte Generieren eines deterministischen endlichen Automaten. Andere Methoden kommen zwar auch zu diesem Ergebnis, gehen jedoch einen Umweg über die Erzeugung eines nichtdeterministischen endlichen Automaten, der anschließend in einen deterministischen Potenzmengenautomat mit sehr vielen Zuständen überführt werden kann. Über entsprechende Minimierungsalgorithmen wird abschlie-

ßend der zugehörige minimale Automat sowohl aus dem Automaten nach Gluschkow als auch aus dem Potenzmengenautomaten erstellt.

Ein deterministischer endlicher Automat besteht aus den fünf Komponenten S, T, δ, S_0, E:

S: endliche Menge von Zuständen

T: endliche Menge der Eingabezeichen (das Eingabealphabet)

δ: die Zustandsüberführungsfunktion $S \times T \rightarrow S$ bestimmt den Folgezustand des Zustands s bezüglich des Eingabezeichens t

S_0: der Anfangszustand mit $S_0 \in S$

E: endliche Menge von Endzuständen mit $E \subseteq S$

Abstrakte Automaten können durch Zustandsdiagramme visualisiert werden. Für Studenten und andere, die sich lernend mit der Materie befassen, kann diese Darstellungsform eine sehr nützliche Hilfestellung sein. Dabei werden Zustände durch Kreise und die Übergänge zwischen ihnen durch Pfeile dargestellt. Jeder Kreis ist mit dem Namen seines Zustandes und jeder Pfeil mit dem Eingabezeichen beschriftet, welches den Übergang ermöglicht. Endzustände werden durch Doppelkreise gekennzeichnet.

Als Beispiel soll folgendes Zustandsdiagramm für den regulären Ausdruck a « b » dienen:

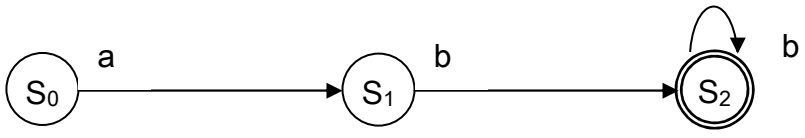


Abbildung 2-1: Beispiel für ein Zustandsdiagramm

2.4 Generierung eines DEA nach Gluschkow

Die Konstruktion eines deterministischen endlichen Automaten nach der Methode von Gluschkow (Gluschkow, 1963) erfolgt in zwei Schritten:

Schritt 1:

Indizierung des regulären Ausdrucks und damit die Beschreibung aller gültigen Wege über den Ausdruck

Schritt 2:

Berechnung der Zustandsüberführungsfunktion

2.4.1 Schritt 1: Indizieren des regulären Ausdrucks

Zur Identifizierung der verschiedenen Stellen des regulären Ausdrucks werden spezifische Termini verwendet, deren Kenntnis für das Verständnis des Algorithmus unverzichtbar ist:

Die Endstelle eines regulären Ausdrucks r mit $r \in T \cup \{\varepsilon\}$ – also eines Terminalsymbols oder des leeren Wortes ε – wird als Grundstelle bezeichnet, die Anfangsstelle als Vorgrundstelle:

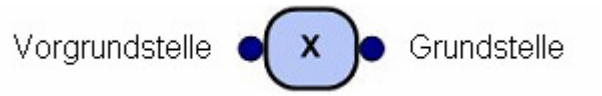


Abbildung 2-2: Vorgrund- und Grundstelle

(Skript zum Modul Modelle der Informatik, IIN, B. Zimmermann)

Ein mehrfach vorkommendes Terminalsymbol besitzt dementsprechend verschiedene Grund- und Vorgrundstellen.

Die Indizierung des regulären Ausdrucks beginnt mit der Zuordnung von so genannten Grundindizes zu allen Grundstellen des Ausdrucks. Dabei erhält die Anfangsstelle des gesamten regulären Aus-

drucks per definitionem den Grundindex Null, die folgenden Grundstellen natürliche Zahlen ungleich Null. Zur besseren Praktikabilität empfiehlt sich die aufsteigende Ordnung der Grundindizes, das ist jedoch keine Bedingung.

Das Ergebnis dieser Grundindex-Zuordnung wird beispielhaft für den regulären Ausdruck [a | a « b »] in Abbildung 2-3 dargestellt:

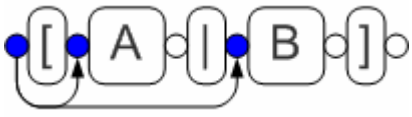
a 1	|	a 2	«	b 3	»	]

Abbildung 2-3: Grundindizes

Diese Indizes gilt es nun im zweiten Arbeitsschritt der Indizierung auf die anderen Stellen des regulären Ausdrucks zu übertragen. Für diese Übertragung existieren fünf Regeln a bis e, die nacheinander solange auf alle Stellen des regulären Ausdrucks angewendet werden, bis keine Indexübertragung mehr erfolgt. Der reguläre Ausdruck ist dann vollständig indiziert.

Die folgenden Erläuterungen über die Anwendungen der Regeln a bis e wurden aus dem Skript zum Modul "Modelle der Informatik" im Studiengang Informatik im Netz entnommen.

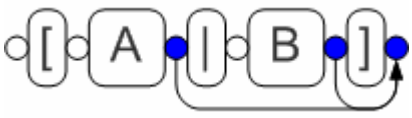
Regel a



Indizes über öffnende Klammern ziehen:

Für r = [A] oder r = [A | B] oder r = «A» oder r = «A | B» werden die Indizes der Anfangsstelle von r auf die Anfangsstelle von A, für r = [A | B] oder r = «A | B», auch auf die Anfangsstelle von B übertragen.

Regel b



Indizes über schließende Klammern ziehen:

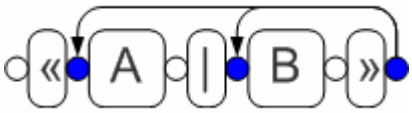
Für r = [A] oder r = [A | B] oder r = «A» oder r = «A | B» werden die Indizes der Endstelle von A (und B) auf die Endstelle von r übertragen.

Regel c

Weglassen des regulären Teilausdrucks:

Für r = «A» oder r = «A | B» werden die Indizes der Anfangsstelle von r auf die Endstelle von r übertragen.

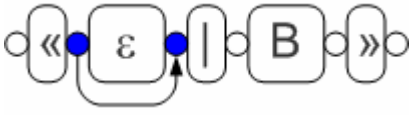
Regel d



Wiederholen des regulären Teilausdrucks:

Für r = «A» oder r = «A | B» werden die Indizes der Endstelle von r auf die Anfangsstelle von A (und B) übertragen.

Regel e



Behandlung des leeren Wortes ε:

Für r = ε werden die Indizes der Anfangsstelle von r auf die Endstelle übertragen.

Für den oben beispielhaft genannten Ausdruck [a | a « b »] ergibt sich die folgende Indizierung:

R 0	[	a 1	|	a 2	«	b 3	»	]
	0a		0a					
								1b
					2ad		2c	2b
					3d		3b	3b

Abbildung 2-4: Grund- und abgeleitete Indizes

Die jeweils angewendete Regel wird den abgeleiteten Indizes nachgestellt. Der Indizierungsvorgang wird damit nachvollziehbarer und das Ergebnis übersichtlicher.

2.4.2 Schritt 2: Berechnen der Zustandsüberführungsfunktion

Die Zustandsüberführungsfunktion besteht aus der Menge der möglichen Zustände und den zwischen ihnen möglichen Übergängen. Ein Zustand S besteht dabei aus den Grundindizes, die den Grundstellen mit demselben Terminalsymbol zugeordnet sind. Der Anfangszustand S_0 enthält damit nur den Index 0. Von diesem Zustand ausgehend werden für jedes vorkommende Terminalsymbol t die Vorgrundstellen mit Indizes aus dem Zustand ermittelt. Die Indizes der zugehörigen Grundstellen bilden zusammen einen Folgezustand von S_0 bezüglich t.

Nach der Berechnung der Folgezustände für alle Terminalsymbole ausgehend von S_0 wird die Generierung mit den so ermittelten Folgezuständen fortgesetzt, für die nun wiederum die Folgezustände ermittelt werden. Die Überführungsfunktion ist vollständig, wenn keine neuen Folgezustände mehr konstruiert werden können. Dabei

muss nicht für jedes Ausgangszustand-Terminalsymbol-Paar ein Folgezustand existieren.

Alle Zustände, die mindestens einen Index enthalten, der an der Endstelle des regulären Ausdrucks vorkommt, bilden zusammen die Menge der Endzustände, in denen ein zu analysierendes Wort vollständig erkannt ist.

Für den Ausdruck [a | a « b »] kann die in Abbildung 2-5 dargestellte Zustandsüberführungsfunktion berechnet werden. Ausgehend vom Zustand S_0 werden mit dem Terminalsymbol a Grundstellen mit den Indizes 1 und 2 erreicht, die den Folgezustand s_1 bilden. Mit dem Terminalsymbol b erreicht man dagegen vom Startzustand aus keine Grundstelle von b – der Folgezustand bleibt leer. Leere Mengen sind auch die Folgezustände von S_1 und S_2 für das Terminalsymbol a.

	Input: a	Input: b
S_0 {0}	$\delta(S_0,a) = \{1,2\} = S_1$	$\delta(S_0,b) = \{\emptyset\} = S(e)$
S_1 {1,2}	$\delta(S_1,a) = \{\emptyset\} = S(e)$	$\delta(S_1,b) = \{3\} = S_2$
S_2 {3}	$\delta(S_2,a) = \{\emptyset\} = S(e)$	$\delta(S_2,b) = \{3\} = S_2$
Endzustände: 1, 2		

Abbildung 2-5: Zustandsüberführungsfunktion

Endzustände für diesen DEA sind S_1 und S_2, da sie beide Indizes enthalten, die an der Endstelle des regulären Ausdrucks vorkommen.

Die Konstruktion des deterministischen endlichen Automaten

$$DEA = (S, T, \delta, S_0, E)$$

umfasst die folgenden Schritte:

Initialisierung:

$$S_0 := \{ 0 \},\ S := (\{ 0 \})$$

Berechnung der Zustandsübergänge:

für $S_n \in S$ und alle $x \in T$ wird

$\delta (S_n, x) = S_n'$ mit

$S_n' = \{ g \mid$ es gibt in dem regulären Ausdruck ein Vorkommen von x mit dem Grundindex g und die zugehörige Vorgrundstelle ist mit mindestens einem Element aus S_n indiziert $\}$

$S := S \cup \{ S_n' \}$

Die Berechnung terminiert genau dann, wenn keine neuen Zustandsübergänge mehr ermittelt werden können.

Berechnung der Endzustände:

$E = \{ S_n \mid S_n \in S$ und S_n enthält mindestens einen Index, der in der Indexierung der Endstelle des regulären Ausdrucks enthalten ist $\}$

3 Anforderungsanalyse

Die Anforderungsanalyse bildet die Grundlage für Entwurf und Implementierung einer Software.

3.1 Zielgruppe(n)

Die zu erstellende Software soll im Wesentlichen im Hochschul-Umfeld eingesetzt werden - ihre Funktion ist also eine vor allem didaktische.

Die Zielgruppe besteht einerseits aus Personen:

- die als Lernende mit (noch) vergleichsweise geringen Kenntnissen der Automatentheorie die Software unterstützend im Lernprozess einsetzen wollen / sollen

und andererseits aus Personen:

- die als Lehrkräfte Anschauungs- und Übungsmaterial erstellen müssen und zu diesem Zweck eine unkomplizierte Software benutzen möchten.

Unter Betrachtung dieser Umstände standen Bedienerfreundlichkeit und Übersichtlichkeit auf Rang 1 der Prioritätenliste.

3.2 Anwendungsfalldiagramm

Die Anforderungsanalyse erfolgte im Wesentlichen auf der Grundlage des in Abbildung 3-1 dargestellten Anwendungsfalldiagramms. Der Anwender agiert ausschließlich über die grafische Benutzerschnittstelle.

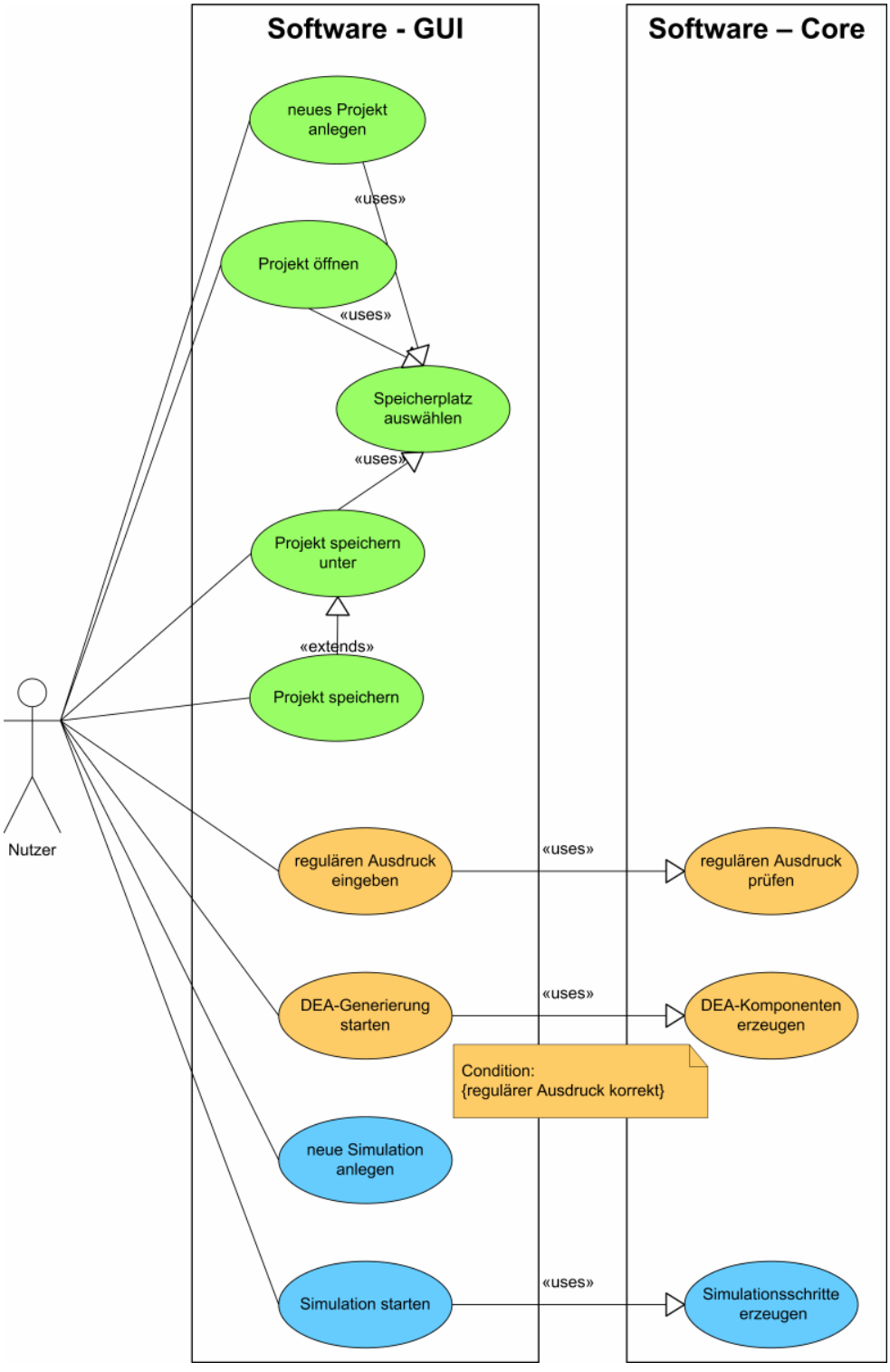


Abbildung 3-1: UML-Anwendungsfalldiagramm

Es können drei Haupt-Aktionsräume unterschieden werden:

- Projekt-Verwaltung
- DEA-Generierung
- Simulation

3.3 Aktivitätsdiagramm

Mit Hilfe von Aktivitätsdiagrammen kann das dynamische Verhalten eines Softwaresystems modelliert werden. Ein solches wurde für den Bereich "Generierung des DEA" erstellt: Abbildung 3-2 zeigt die grundlegenden Schritte, die der Nutzer vom Starten der Software bis zum Vorliegen eines anwendungsbereiten DEA durchlaufen kann bzw. muss. Hierbei kam es vor allem darauf an, die Abläufe der Projektverwaltung zu verdeutlichen.

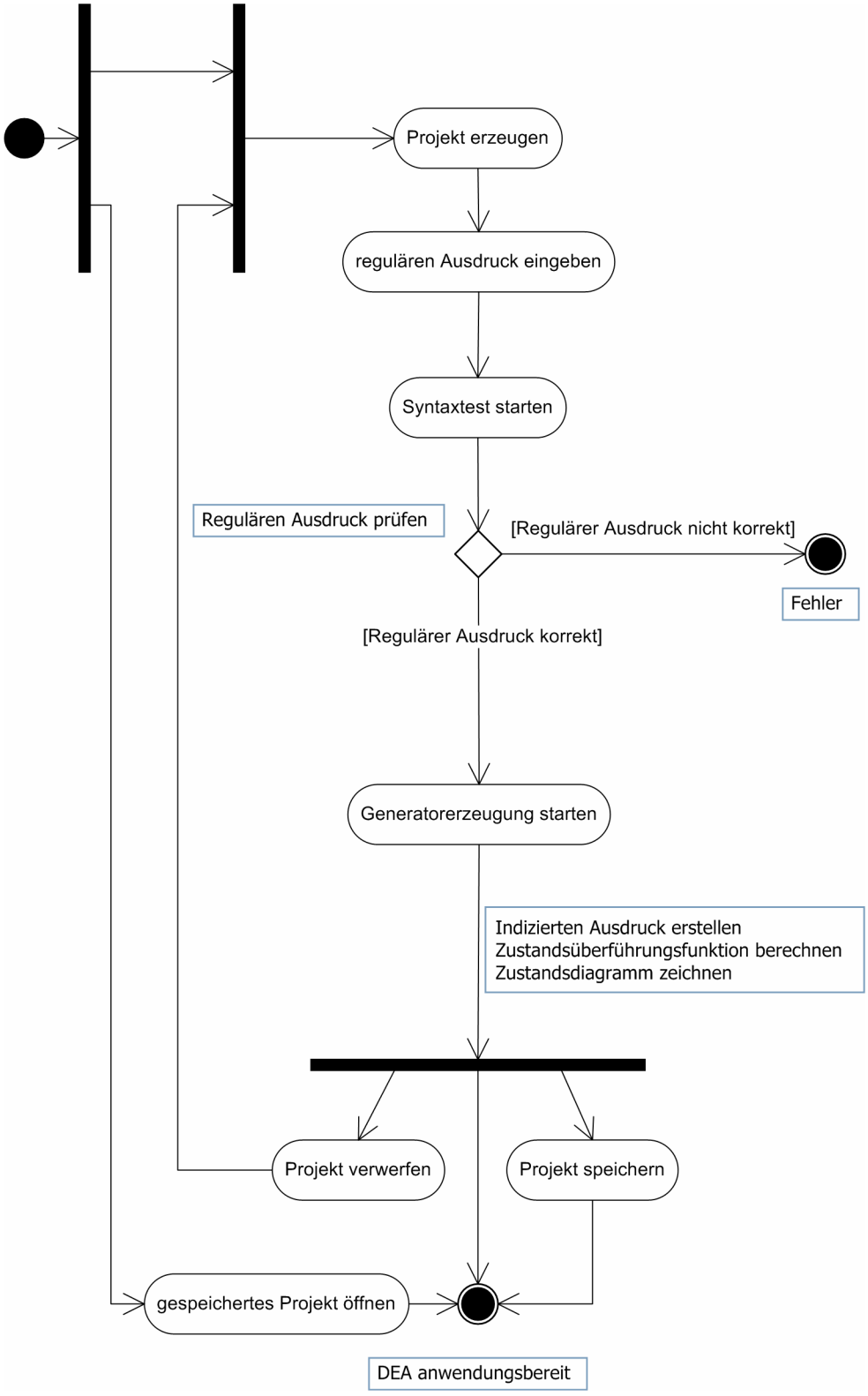


Abbildung 3-2: UML-Generator-Aktivitätsdiagramm

3.4 Anforderungen

3.4.1 Notwendige Anforderungen

Ausgehend von der Aufgabenstellung, den Anwendungsfällen und dem Aktivitätsdiagramm konnten folgende Anforderungen (Requirements: Rx) an die Software herausgearbeitet werden:

- **R1:** Ein DEA bildet gemeinsam mit seinen Simulationen ein Projekt und umfasst dabei folgende Komponenten:
  - **R1.1:** DEA-Generator (genau einer), dargestellt durch:
    - Input: genau ein regulärer Ausdruck
    - Output 1: genau ein indizierter Ausdruck
    - Output 2: genau eine Zustandsüberführungsfunktion
    - Output 3: genau ein Zustandsdiagramm
  - **R1.2:** Simulationen (beliebig viele), dargestellt durch:
    - Input: genau eine Eingabezeichenfolge
    - Output: einer oder mehrere Schritte entsprechend der Zustandsüberführungsfunktion
- **R2:** Projekte sollen in ihrem aktuellen Zustand archiviert und wiederhergestellt werden können.
- **R3:** Der vom Anwender eingegebene reguläre Ausdruck soll auf seine syntaktische Korrektheit geprüft werden.
- **R4:** Die Anwendung soll eine moderne grafische Benutzerschnittstelle im Microsoft-Windows-Stil haben:
  - **R4.1:** Ein Hauptmenü soll Zugang zu allen Aktionen bieten
    - Erstellen von Projekten und Simulationen
    - Speichern und Öffnen von Projekten
    - Einstellungen (z.B. Sprache der GUI)
    - Hilfe / Dokumentation / Kontakt

- **R4.2:** Ein Navigationsframe soll Zugriff auf vorhandene Projekte ermöglichen.
- **R4.3:** Es darf immer nur ein Projekt aktiv (geöffnet) sein.
- **R4.4:** Die einzelnen Komponenten eines Projektes sollen gleichzeitig und gleichberechtigt für den Anwender sichtbar sein und nach dessen Vorstellungen ein- oder ausgeblendet werden können.

3.4.2 Optionale Anforderungen

Für die unmittelbare DEA-Funktionalität nicht notwendig, jedoch unter dem Aspekt der Nutzerfreundlichkeit sehr wünschenswert, sind zwei weitere Anforderungen:

- Die Lokalisierung der Software, d.h. die Änderung der GUI-Sprache, die optimal zur Laufzeit erfolgen kann.
- Eine Druckfunktion, die eine formatierte Ausgabe aller oder ausgewählter DEA-Bestandteile ermöglicht.

4 Entwurf der Softwarekomponenten

4.1 Pakete

Ausgehend von den Anwendungsfällen wurden für die zu erstellende Software die vier Paket-Komponenten `Views`, `Data`, `Core` und `DOM` postuliert, wie in Abbildung 4-1 dargestellt:

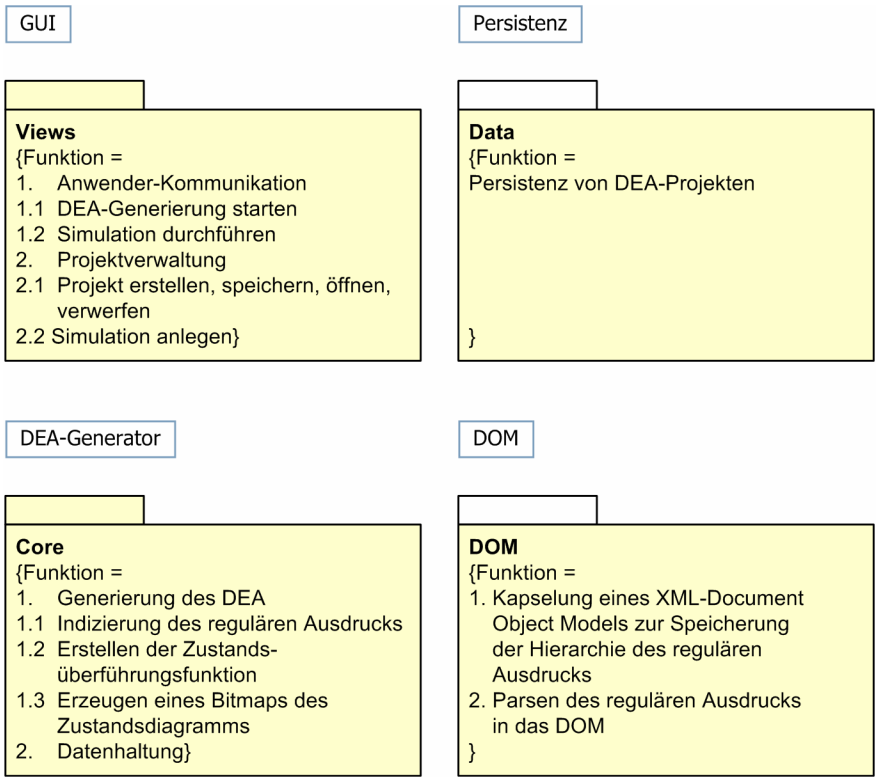


Abbildung 4-1: UML-Pakete

Das Paket `Views` entspricht damit dem "V" (Views) im MVC-Modell, während `Data`, `Core` und `DOM` die Funktionalität des "M" (Model) übernehmen. Die Abgrenzung dieser wesentlichen Funktionsbereiche soll auch in der Implementierung zum Ausdruck kommen.

4.2 Paket Views: Graphical User Interface

Das Paket Views wird die unmittelbar mit der Anwenderkommunikation zusammenhängende Funktionalität enthalten. Abbildung 4-2 verdeutlicht die enthaltenen Bereiche:

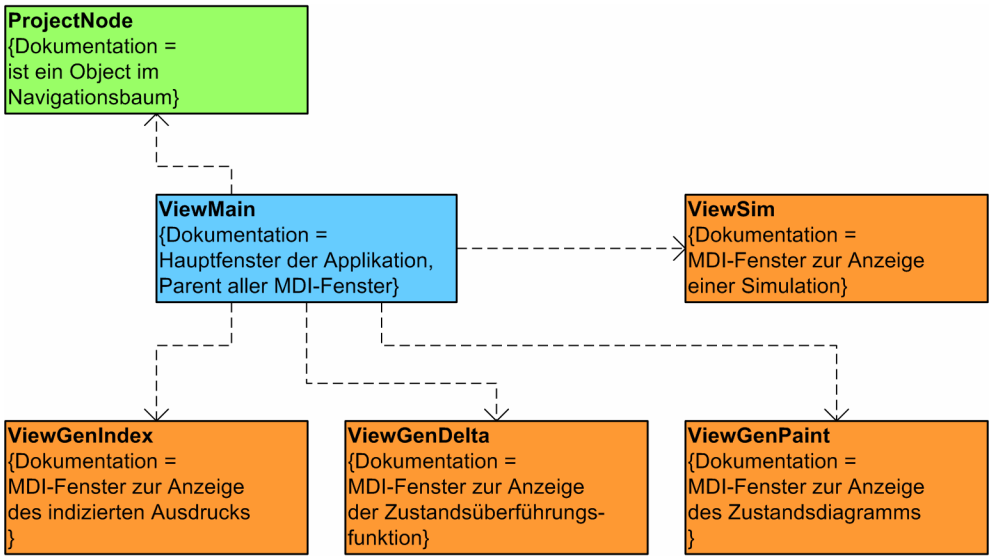


Abbildung 4-2: Klassen des Views-Paketes

4.2.1 ViewMain

Das Objekt `ViewMain` stellt die Hauptklasse der Anwendung dar. Hier werden grundlegende Anforderungen wie die Einbindung von Menüs, die Unterstützung der Lokalisierung oder die Aufteilung der Oberfläche in Navigationsbereich und Arbeitsbereich realisiert.

4.2.2 Visualisierung des DEA Generators

Durch die orange hinterlegten Objekte werden die Entgegennahme der Eingaben und die Darstellung der Ausgaben inkl. der bildschirmgerechten Aufarbeitung der zur Ausgabe bestimmten Daten dargestellt.

- Die MDI-Objekte `ViewGenIndex`, `ViewGenDelta` und `ViewGenPaint` werden den indizierten Ausdruck, die Zustandsüberführungsfunktion und das Zustandsdiagramm visualisieren. Sie exis-

tieren in jedem Projekt genau einmal. Die Daten werden in Tabellenform vom `Core`-Paket zur Verfügung gestellt.

- Das MDI-Objekt `ViewSim` dient zur Visualisierung einer einzelnen Simulation. In jedem Projekt kann es davon beliebig viele geben. Die nötigen Daten werden direkt aus den Daten der Zustandsüberführungsfunktion entnommen.

4.2.3 Projektverwaltung

Das grün hinterlegte Objekt unterstützt die Projektverwaltung, welche im Wesentlichen in der Hauptklasse `ViewMain` erfolgt. Der Navigationsbaum wird verschiedene Objektarten enthalten:

- beliebig verschachtelbare Ordner zur Projekt-Strukturierung
- Projekte in Ordnern oder direkt auf der obersten Ebene
- pro Projekt einen Eintrag für den indizierten Ausdruck, die Zustandsüberführungsfunktion und das Zustandsdiagramm
- pro Projekt einen Ordner für die Simulationen
- Simulationen im Simulationsordner

Die Unterscheidung dieser Objekte ermöglicht ein Aufzählungstyp.

4.2.4 Programmsteuerung

Weiterhin wird in dieses Paket `Views` auch die Programmsteuerung integriert, d. h. für den "Controller" des MVC-Modells wird kein gesondertes Paket erstellt. Angesichts der überschaubaren Gesamtgröße der zu erstellenden Software entspricht dieser Ansatz einer üblichen Vorgehensweise.

4.3 Paket Data: Persistenz

In diesem Paket werden die zur Persistenz der Projekte nötigen Funktionen bereitgestellt. Die Datenhaltung soll auf der Basis von XML-Dokumenten erfolgen. Die zu speichernden Daten umfassen:

- den regulären Ausdruck, aus dem beim Öffnen eines Projektes der Generator in Form von indiziertem Ausdruck, Zustandsüberführungsfunktion und Zustandsdiagramm wiederhergestellt wird,
- die erzeugten Simulationen inkl. Eingabezeichenfolge und aktuellem Schritt, in dem die Simulation sich beim Speichern befand.

4.4 Paket DOM: Document Object Model

In diesem Paket wird die zum Parsen des regulären Ausdrucks nötige Funktionalität zusammengefasst. Dabei werden in einer vorbereitenden Aktion die Eingabezeichen des regulären Ausdrucks durch ein `Lexer`-Objekt in einen Token-Strom umgewandelt. Bei korrekter Syntax des eingegebenen regulären Ausdrucks kann dieser Token-Strom vom `Parser`-Objekt in ein hierarchisch aufgebautes Document Object Model strukturiert werden.

Die Elemente des Paketes DOM sind in Abbildung 4-3 schematisiert:

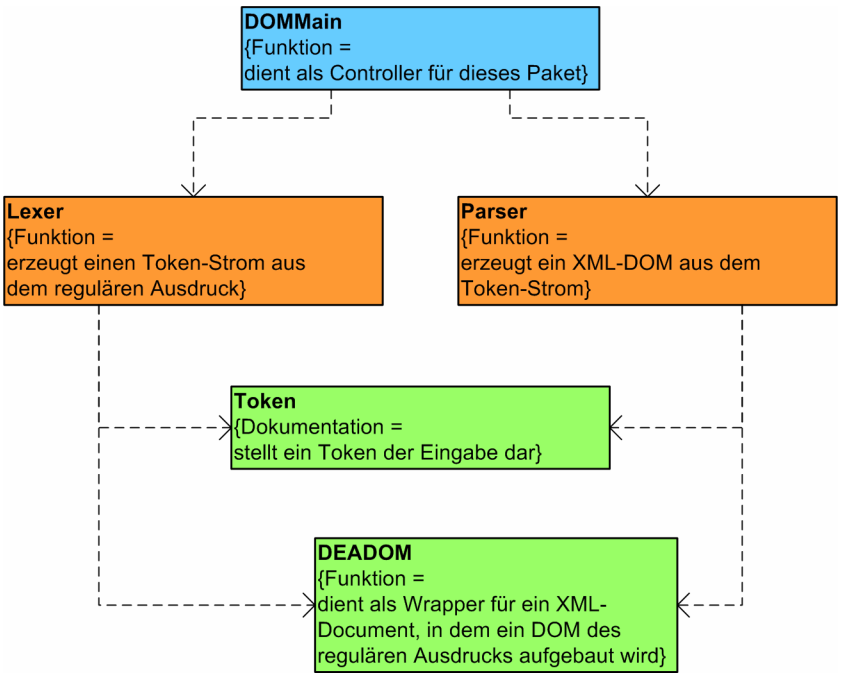


Abbildung 4-3: Klassen des DOM-Paketes

Die erfolgreiche Generierung dieses DOM erfüllt zwei Funktionen:

- Sie dient als Bestätigung der syntaktischen Korrektheit des regulären Ausdrucks und damit als zentrale Kontrollstelle, die die weitere Generierung des DEA ermöglicht oder unterbindet und
- sie liefert in einem XML-Dokument eine strukturierte Datenbasis zur Generierung des DEA.

4.5 Paket Core: DEA-Generator

Das Paket Core wird die gesamte Logik der DEA-Generierung enthalten. Als Schnittstelle zur GUI ist ein zentrales Objekt `CoreMain` vorgesehen, welches auch die Controller-Funktion für diesen Teilbereich übernimmt.

Die Bereiche des Generators werden durch die Objekte `GenIndex` (indizierter Ausdruck), `GenDelta` (Zustandsüberführungsfunktion) und `GenPaint` (Zustandsdiagramm) repräsentiert, wie in Abbildung 4-4 dargestellt.

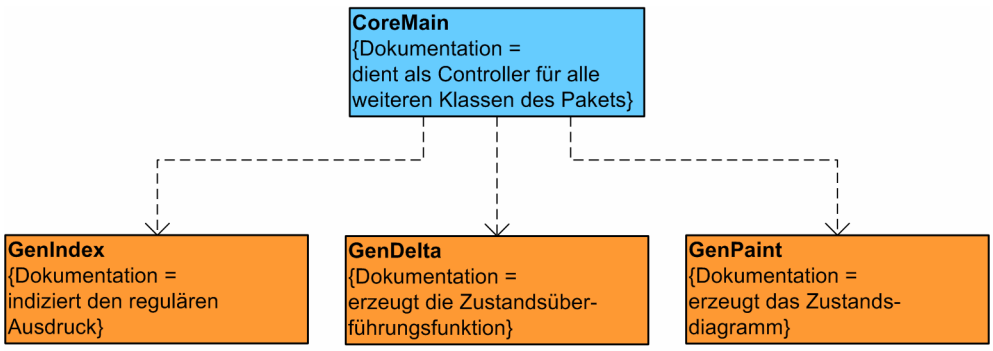


Abbildung 4-4: Komponenten des Core-Paketes

4.5.1 Der Teilbereich "Indizierter Ausdruck"

Zur Implementierung der Logik im `GenIndex`–Objekt werden ein `Indexx`- und ein `IndexList`-Objekt benötigt. `Indexx` –Objekte (das Doppel-X ist nur nötig, weil "Index" ein reservierter Ausdruck ist) sollen folgende Informationen enthalten:

- das Token (Eingabezeichen)
- den Grundindex an dieser Stelle
- eine Aufzählung aller nach den Regeln a bis e abgeleiteten Indizes für diese Stelle
- die Information, ob ein Zeichen des Eingabealphabets oder ein syntaktisches Zeichen des regulären Ausdrucks vorliegt

Ein `IndexTable`-Objekt wird die Daten enthalten, die zur Visualisierung des indizierten Ausdrucks an die GUI zurückgegeben werden.

Die Elemente der `GenIndex`-Komponente des Core-Paketes werden in Abbildung 4-5 dargestellt.

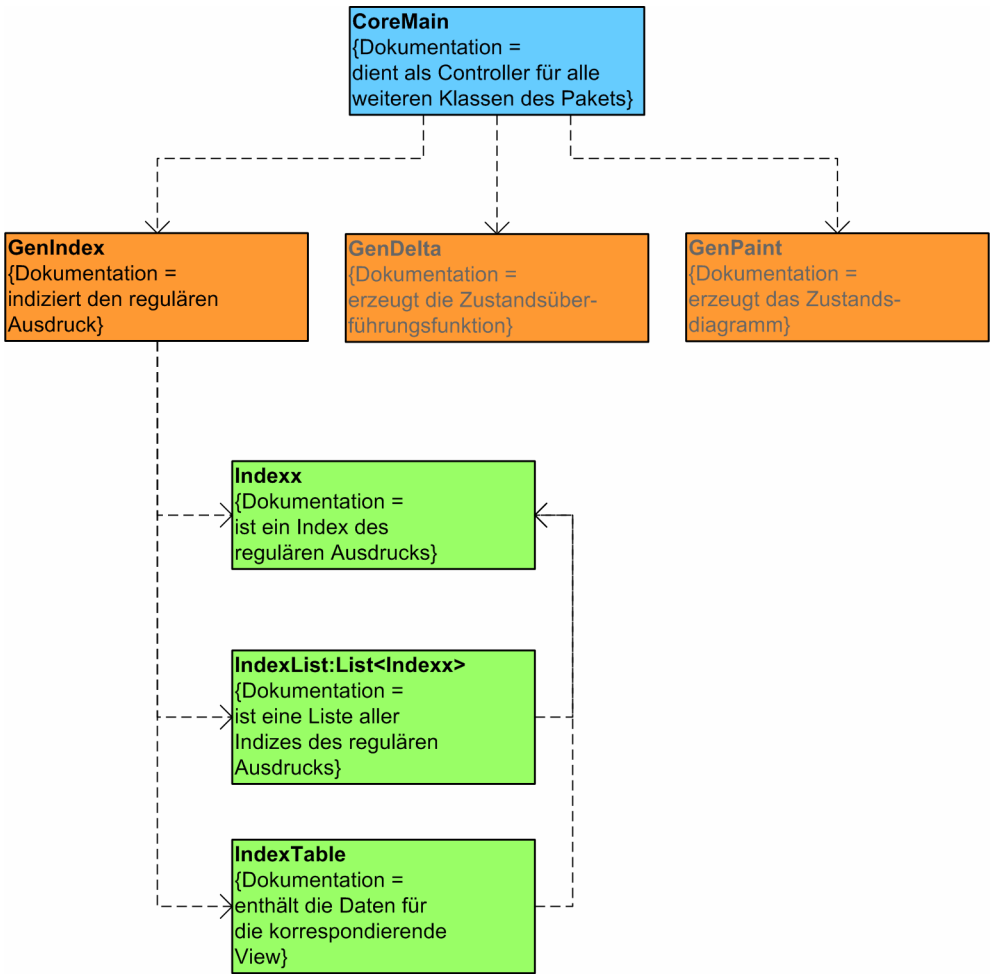


Abbildung 4-5: Klassen der GenIndex-Komponente im Core-Paket

4.5.2 Der Teilbereich "Zustandsüberführungsfunktion"

Das Objekt `GenDelta` wird die Logik dieses Generator-Teilbereichs enthalten, weiterhin sind die Objekte `State` und `StateTransition` zur Repräsentation von Zuständen und deren Übergängen nötig. Eine `StateList` wird alle aus dem indizierten Zustand ermittelten Zustände eines DEA enthalten. Weiterhin wird das Objekt `StateTable` die Daten enthalten, die zur Visualisierung der Zustandsüberführungsfunktion an die GUI zurückgegeben werden. Die Elemente der `GenDelta`-Komponente des Core-Paketes werden in Abbildung 4-6 dargestellt.

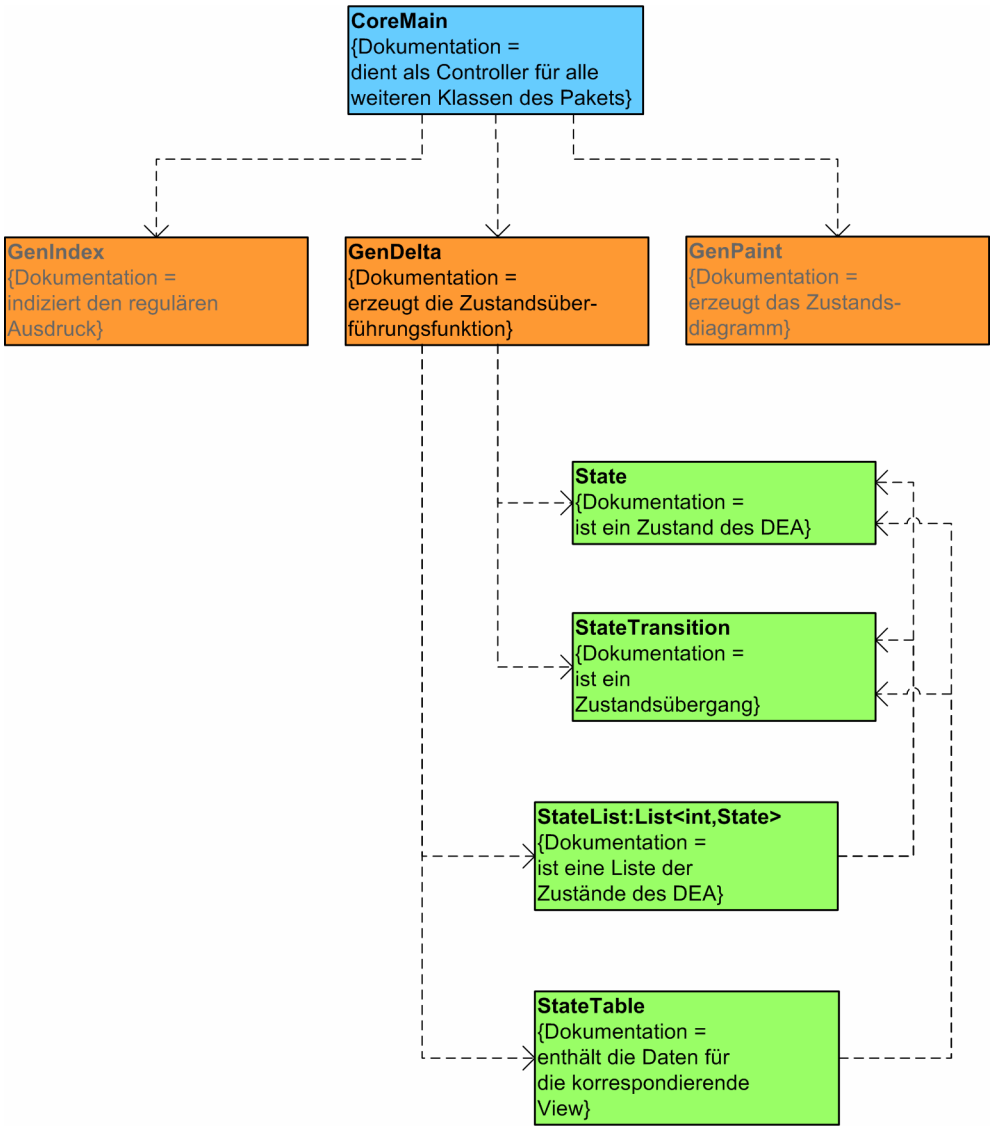


Abbildung 4-6: Klassen der GenDelta-Komponente im Core-Paket

4.5.3 Der Teilbereich "Zustandsdiagramm"

Das Zustandsdiagramm wird als Bitmap erzeugt. Es besteht aus Kreisen, die Zustände darstellen, und Pfeilen, die die Übergänge zwischen den Zuständen markieren. Ein Kreis soll durch ein DiagrammCell-Objekt und ein Pfeil durch ein Arrow-Objekt implementiert werden.

Ein Aufzählungstyp `Sizes` wird alle Größenangaben enthalten, die zur Positionierung und Skalierung der Zeichenobjekte – neben den Kreisen und Pfeilen z.B. auch die Buchstaben der Eingabezeichen - nötig sind.

Die Elemente der `GenPaint`-Komponente des Core-Paketes werden in Abbildung 4-7 dargestellt.

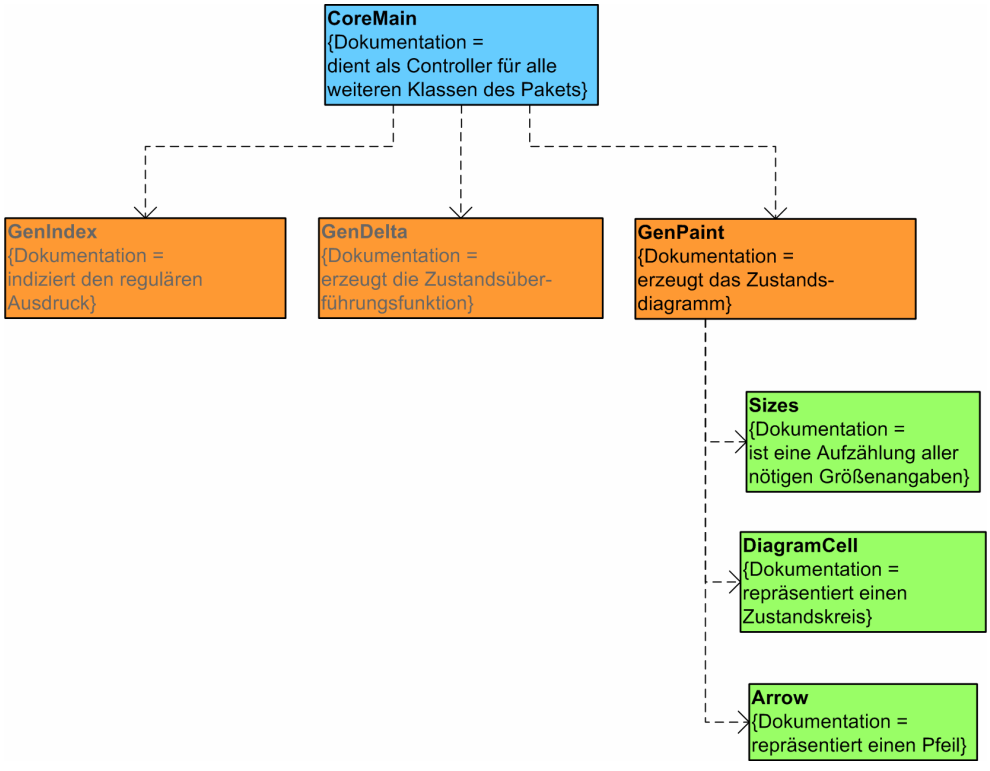


Abbildung 4-7: Klassen der GenPaint-Komponente im Core-Paket

4.6 Lokalisierung

Zur Umsetzung der optionalen Anforderung der Lokalisierung ist es notwendig, alle sprachabhängigen Ressourcen wie Fehlermeldungen oder Beschriftungen von GUI-Elementen aus dem Quelltext auszugliedern und in Ressourcen-Dateien zu verwalten.

Für jede gewünschte Sprache müssen im Weiteren zwei Bedingungen erfüllt werden:

- Es muss eine solche Ressourcen-Datei mit den Texten in dieser Sprache vorliegen.
- Die GUI muss entsprechende Aktionsmöglichkeiten zum Auswählen der Sprache bieten.

5 Algorithmen zur Implementierung der DEA-Logik

5.1 Paket DOM: das Document Object Model

Vor der Generierung des Automaten muss der reguläre Ausdruck in ein passend strukturiertes Format überführt werden. Da ein regulärer Ausdruck in sich hierarchisch aufgebaut ist, bietet sich eine Struktur, die einen Ableitungsbaum darstellt, als Ergebnis des Parsens an.

Diese Struktur soll

- ein Document Object Model mit genau definierten Elementen und Zugriffsmethoden repräsentieren und
- als internes Speicherobjekt für den regulären Ausdruck dienen und damit speicherresident über die Lebensdauer des konkreten DEA existieren.

Beide Anforderungen werden von XML-Dokumenten erfüllt, weshalb ein solches als Basis-Klasse für die spezifische DEA-Baumstruktur-Klasse implementiert wurde. Ein direkter Zugriff auf das XML-Dokument ist dabei nur den privaten Methoden der abgeleiteten DEADOM-Klasse `DEADOM.cs` gestattet, die XML-Funktionalität wird über eine Schnittstellendeklaration (`IDEADOM`) definiert. Die im Folgenden erläuterten Elemente der XML-Struktur werden durch die Schemadatei `DEADOM.xsd` definiert.

5.1.1 Die Elemente des DOM

Die Knoten des XML-DOM repräsentieren die syntaktischen Elemente des regulären Ausdrucks: Teilausdruck, Wiederholung, Alternative und Terminalsymbol.

5.1.1.1 Expression

Ein Teilausdruck in eckigen Klammern wird als `Expression` bezeichnet. Er kann selbst alle Arten von Elementen enthalten, insbesondere müssen mehrere Alternativen von eckigen Klammern ein-

geschlossen werden, wenn sie sich nicht innerhalb einer Wiederholung befinden.

Der gesamte reguläre Ausdruck wird ebenfalls als Expression betrachtet und bildet damit den Ausgangspunkt des rekursiven Parsens.

5.1.1.2 Loop

Ein Teilausdruck in spitzen Klammern stellt eine Wiederholung dar und wird als `Loop` bezeichnet. Er kann selbst alle Arten von Elementen enthalten.

5.1.1.3 Option

Alternative Teilausdrücke werden durch senkrechte Striche getrennt und als `Option` bezeichnet. Eine Option kann alle Arten von Elementen enthalten.

5.1.1.4 Terminal

Die Zeichen des Eingabe-Alphabets bilden die Blätter (Endpunkte) der Baumstruktur. Sie werden als `Terminale` bezeichnet.

5.1.1.5 Epsilon

Das leere Wort ε stellt einen Sonderfall innerhalb der Terminalsymbole dar und wird daher durch einen eigenen Element-Typ verkörpert.

5.1.2 Die Attribute innerhalb des DOM

Die Attribute der XML-Knoten enthalten verschiedene Eigenschaften der Syntax-Elemente, insbesondere die Indizes.

5.1.2.1 Grundindex

Jedes Terminal-Element ebenso wie das ε-Element enthält ein Grundindex-Attribut `INDEX_BASE`.

Als Sonderfall wird auch dem Root-Element des XML-Dokuments ein Grundindex zugeordnet. Dieser hat immer den Wert 0.

5.1.2.2 abgeleiteter Index

Expression- und Loop-Elemente besitzen zwei Attribute für die Speicherung der abgeleiteten Indizes:

- das Attribut `INDEX_OPEN` bezieht sich auf das Zeichen der öffnenden Klammer
- das Attribut `INDEX_CLOSE` steht für das Zeichen der schließenden Klammer

Als Sonderfall besitzt auch das ε-Element ein Attribut `INDEX_OPEN`, da dieses Element nicht nur einen Grundindex, sondern auch einen abgeleiteten Index besitzen kann.

5.1.2.3 Zeichenfolgen-Wert

Jedes Element besitzt ein Attribut `VALUE`, welches den von diesem Element repräsentierten Teilausdruck des regulären Ausdrucks darstellt. Das ε-Element enthält kein `VALUE`-Attribut.

Dieses Attribut hat keinen direkten Nutzen für den Prozess der DEA-Generierung, es dient allein informativen Zwecken für den Fall, dass das DEADOM-Objekt in eine Datei gespeichert wird.

5.1.2.4 HasOptions-Flag

Das Attribut `HAS_OPTIONS` wird für Expressions, Loops und Options gesetzt und zeigt an, ob im entsprechenden Teilausdruck Alternativen enthalten sind. Es wird als strukturelle Information bei der Indizierung des regulären Ausdrucks benutzt.

5.1.2.5 IsFirst-Flag

Das Attribut `IS_FIRST` wird nur für Options vergeben und zeigt an, ob es sich um die erste Alternative eines Teilausdruckes handelt (die im Gegensatz zu allen folgenden nicht von einem senkrechten Strich

eingeleitet wird). Es wird als strukturelle Information bei der Indizierung des regulären Ausdrucks benutzt.

5.1.3 Lexer

Die Funktion der Klasse `Lexer.cs` ist die Umwandlung eines Input-Strings in einen Strom syntaktisch relevanter Token:

- `LPAR_SQUARE` – linke eckige Klammer
- `RPAR_SQUARE` – rechte eckige Klammer
- `LPAR_SPIKY` – linke spitze Klammer
- `RPAR_SPIKY` – rechte spitze Klammer
- `PIPE` – senkrechter Strich

Alle weiteren Zeichen des Eingabe-Alphabets werden als Stringtoken mit ihrem eigenen Wert identifiziert, während Whitespaces herausgefiltert werden.

5.1.4 Parser

Die Aufgabe der Klasse `Parser` ist die Erzeugung des DOM aus dem Tokenstrom. Zu diesem Zweck wird in der Methode `Match_Expression` für jedes Token des Tokenstroms ein entsprechender Knoten im XML-Document des DOM erzeugt. Beim Erreichen einer öffnenden Klammer werden die Token bis zur schließenden Klammer temporär gesammelt und diese in einem rekursiven Aufruf wiederum an `Match_Expression` übergeben, um die innerhalb der Klammern befindlichen Token dem besitzenden Knoten unterzuordnen.

Beim Vorgang des Parsens werden auch die Grundindizes der Terminal- und ε-Elemente geschrieben. In Abbildung 5-1 ist das fertige DOM für den regulären Ausdruck `[a|a«b»c||wxyz]` skizziert.

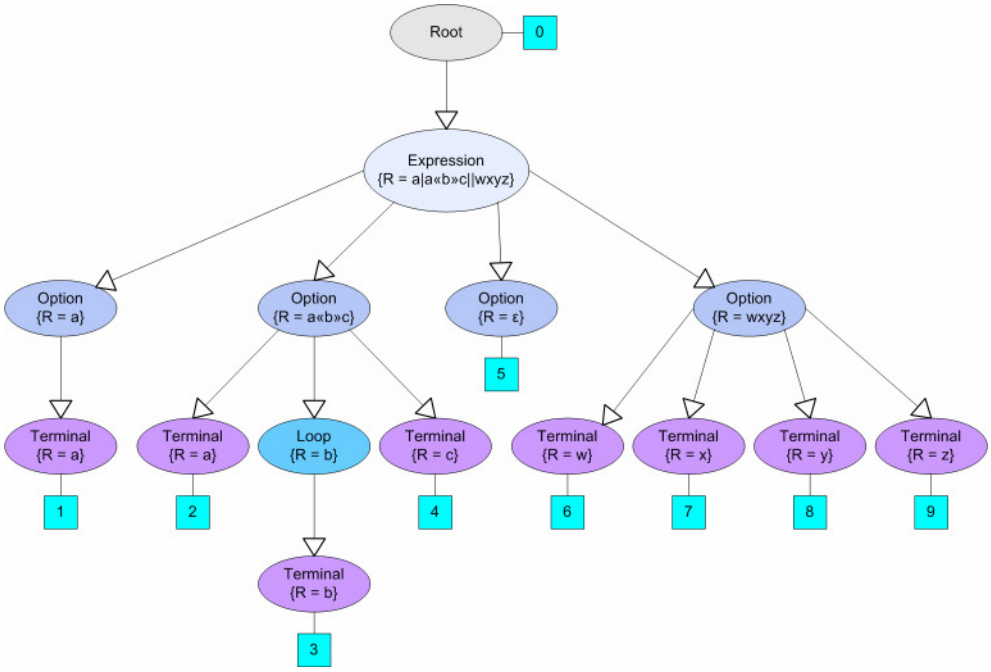


Abbildung 5-1: DOM mit Grundindizes

5.2 Paket Core: die Generierung des DEA

Die Generierung der Komponenten des DEA wird von der Klasse `CoreMain` gesteuert. In dieser werden aufeinander folgend die folgenden Methoden aufgerufen:

- `GenIndex.Create_Indexes` zur Indizierung des regulären Ausdrucks
- `GenDelta.Create_Delta` zur Berechnung der Zustandsüberführungsfunktion aus dem indizierten Ausdruck
- `GenPaint.Get_Bitmap` zur Erzeugung eines Bitmaps zur Visualisierung des Zustandsdiagramms

5.2.1 Die Indizierung des regulären Ausdrucks

5.2.1.1 Der Algorithmus der Indizierung

Grundlage der Indizierung des regulären Ausdrucks ist dessen im DOM gespeicherte Struktur, die der Methode `Create_Indexes` als Input übergeben wird. Die Indizes werden entsprechend ihres Typs

mit den Attributen `INDEX_BASE`, `INDEX_OPEN` oder `INDEX_CLOSE` direkt in den Elementen des DOM gespeichert. Das Ergebnis der Indizierung ist also ein DOM, welches nicht nur die Grundindexes enthält wie in Abbildung 5-1, sondern darüber hinaus auch die abgeleiteten Indizes, wie in Abbildung 5-8 dargestellt.

Innerhalb von `Create_Indexes` wird die DOM-Baumstruktur rekursiv mittels der Methode `Apply_Rules` analysiert. `Apply_Rules` erwartet als Parameter zwei Knoten des DOM – den Quell-Knoten, dessen Indizes übertragen werden sollen, und den Ziel-Knoten, der diese Indizes übernehmen soll. Dabei werden für jedes Quelle-Ziel-Paar nacheinander die fünf Regeln der Indexübertragung nach Gluschkow angewendet.

Da Klammerausdrücke im DOM nur durch einen Knoten repräsentiert werden, ist weiterhin die Information nötig, ob die aktuelle Bearbeitung sich am Anfang oder am Ende eines solchen Teilausdrucks befindet.

Die Rekursion beginnt mit folgenden Parametern:

- Quelle ist der Root-Knoten, er trägt per Festlegung den Grundindex 0 und entspricht damit der Stelle vor dem Beginn des regulären Ausdrucks.
- Ziel ist der einzige Kind-Knoten von Root: die Expression, die dem gesamten regulären Ausdruck entspricht und per Festlegung von eckigen Klammern begrenzt wird.
- Der Start-Indexierungstyp lautet `INDEX_OPEN` für die öffnende Klammer.

Für dieses Knoten-Paar werden nun die Regeln a bis e (Details siehe unten) angewendet, was an dieser Stelle z.B. bedeutet, dass der Grundindex 0 vom Root-Knoten auf die öffnende Klammer des gesamten regulären Ausdrucks übertragen wird.

Nach der Indexübertragung für dieses Quelle-Ziel-Paar werden die im Ziel-Knoten enthaltenen Optionen bzw. die einzige enthaltene Option bearbeitet. Dabei wird:

- der bisherige Ziel-Knoten zur Quelle
- das erste Token der Option zum Ziel

Ist das erste Token der Option wiederum ein Klammerausdruck, startet eine weitere Rekursionsrunde durch Aufruf von `Apply_Rules`.

Danach werden die Indizes durch weitere Aufrufe von `Apply_Rules` von einem Token der Option zum nächsten übertragen:

- Folgen zwei Klammerausdrücke aufeinander, so werden die Indizes der schließenden Klammer des ersten Ausdrucks übertragen: der Indexierungstyp ist `INDEX_CLOSE`.
- Folgt ein Klammerausdruck auf ein Terminalsymbol, ist der Indexierungstyp `INDEX_OPEN`, denn ein Terminal hat keine schließende Klammer.
- Die Aufeinanderfolge von zwei Terminalsymbolen bewirkt keine Indexübertragung.

Dieser Algorithmus muss durchlaufen werden, bis ein vollständiger rekursiver Zyklus keine neuen Indizes mehr überträgt. Eine einzige vollständige Rekursion durch den DOM-Baum ist dabei für die meisten regulären Ausdrücke nicht ausreichend: es können z.B. durch Regel c Indizes übertragen werden, die an den Stellen, zu denen sie übertragen wurden, ihrerseits unter die Regeln a oder b oder beide fallen und damit erst in einem zweiten Durchlauf von diesen beiden Regeln bearbeitet werden können.

Gesteuert wird dieses über eine do-while-Schleife in der aufrufenden Methode `Create_Indexes`. Diese wird erst dann beendet, wenn der interne Zähler, der Indexübertragungen registriert, nach einem Durchlauf auf Null verharrt.

In Abbildung 5-2 wird der Algorithmus der rekursiven Indizierung des DOM zusammenfassend in Pseudocode dargestellt:

```
apply_rules(root, regex, OPEN);
begin

  rule_a();
  rule_b();
  rule_c();
  rule_d();
  rule_e();

  source: regex
  target: first_child_of_regex

  foreach option in source

    if target is bracket_term
      apply_rules(source, target, OPEN);
    end if;

    foreach child_of_target
      source: this_child_of_target
      target: next_child_of_target

      if source is bracket_term and target is bracket_term
        apply_rules(source, target, CLOSE);
      elseif source is terminal and target is bracket_term
        apply_rules(source, target, OPEN);
      end;

    end;
  end;
end;
```

Abbildung 5-2: Pseudocode des Indizierungs-Algorithmus

Im Folgenden werden die fünf Regeln der Indexübertragung nach Gluschkow näher erläutert und die konkreten Stellen ihrer Anwendung am bereits oben verwendeten Beispiel-Ausdruck dargestellt.

5.2.1.2 Regel a

Regel a wird angewendet, wenn das Ziel der Indexübertragung ein Klammerausdruck ist. Die Indizes werden von dem vor der öffnenden Klammer befindlichen Quell-Knoten übernommen und im `IN-DEX_OPEN`-Attribut des Ziel-Knotens gespeichert. Dieses Attribut entspricht der öffnenden Klammer. Diese für die öffnende Klammer eines Klammerausdrucks gespeicherte Information ist ebenfalls für alle innerhalb des Klammerausdrucks befindlichen Optionen gültig. Options-Knoten enthalten per Festlegung keine eigenen Indizes. Sie übernehmen beim Erstellen der Tabelle, die die aufbereiteten Daten für die Visualisierung des indizierten Ausdrucks enthält, die Indizes der öffnenden Klammer ihres umgebenden Klammerausdrucks.

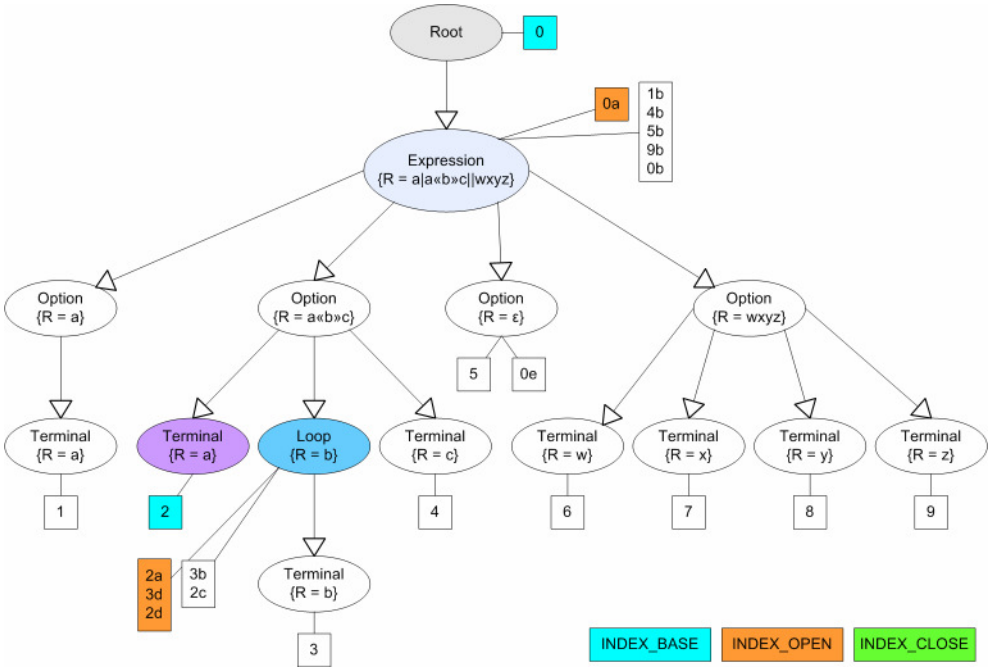


Abbildung 5-3: Anwendung der Regel a im DOM

Abbildung 5-3 zeigt die Knoten-Paare, an denen Indizes nach Regel a übertragen wurden. Dies ist für den gegebenen regulären Ausdruck `[a|a«b»c||wxyz]` einmal von Root auf Expression und in der zweiten Option von Terminal a auf Loop b der Fall.

5.2.1.3 Regel b

Regel b wird ebenfalls angewendet, wenn das Ziel der Indexübertragung ein Klammerausdruck ist. In diesem Fall werden die Indizes jedoch über die schließende Klammer gezogen. Da im Gegensatz zu Regel a die von jedem Ende einer Option zu übertragenden Indizes voneinander verschieden sein können, muss diese Regel für jede Option einzeln angewendet werden.

Die Indizes werden vom letzten Kind-Knoten jeder Option übernommen. Handelt es sich bei diesem Knoten um einen Klammerausdruck, so werden die Indizes des `INDEX_CLOSE`-Attributes verwendet, welche der schließenden Klammer entsprechen. Ist der Knoten kein Klammerausdruck, sondern ein Terminalsymbol oder das leere Wort ε, so werden die Werte des `INDEX_OPEN`-Attributs verwendet. Abbildung 5-4 demonstriert die Übertragung von fünf Indizes vom jeweils letzten Kind einer Option auf die umschließende Expression, die dem vollständigen regulären Ausdruck `[a|a«b»c||wxyz]` entspricht. Die sich aus diesen Indizes ergebenden Zustände definieren die Endzustände des DEA.

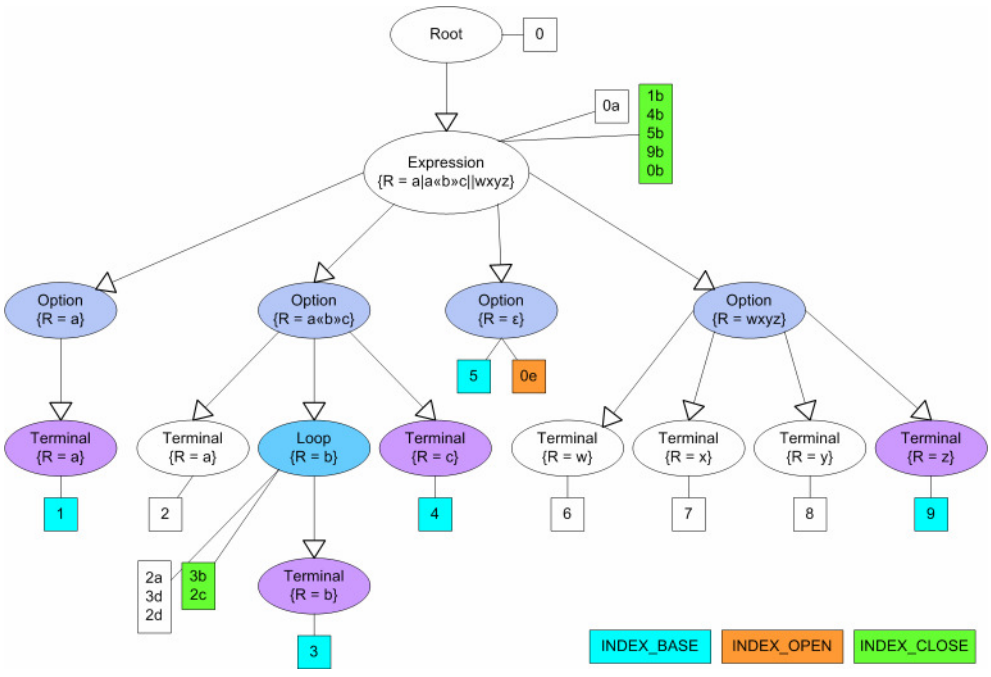


Abbildung 5-4: Anwendung der Regel b im DOM

5.2.1.4 Regel c

Regel c wird angewendet, wenn das Ziel der Indexübertragung eine Wiederholung ist – diese kann übersprungen werden. Die vom Quell-Knoten zu übertragenden Indizes werden in diesem Fall auf das `INDEX_CLOSE`-Attribut, also die schließende Klammer des Ziel-Knotens übertragen.

Abbildung 5-5 zeigt das eine Knoten-Paar, für welches im betrachteten Beispiel `[a|a«b»c||wxyz]` die Indexübertragung nach Regel c erfolgt.

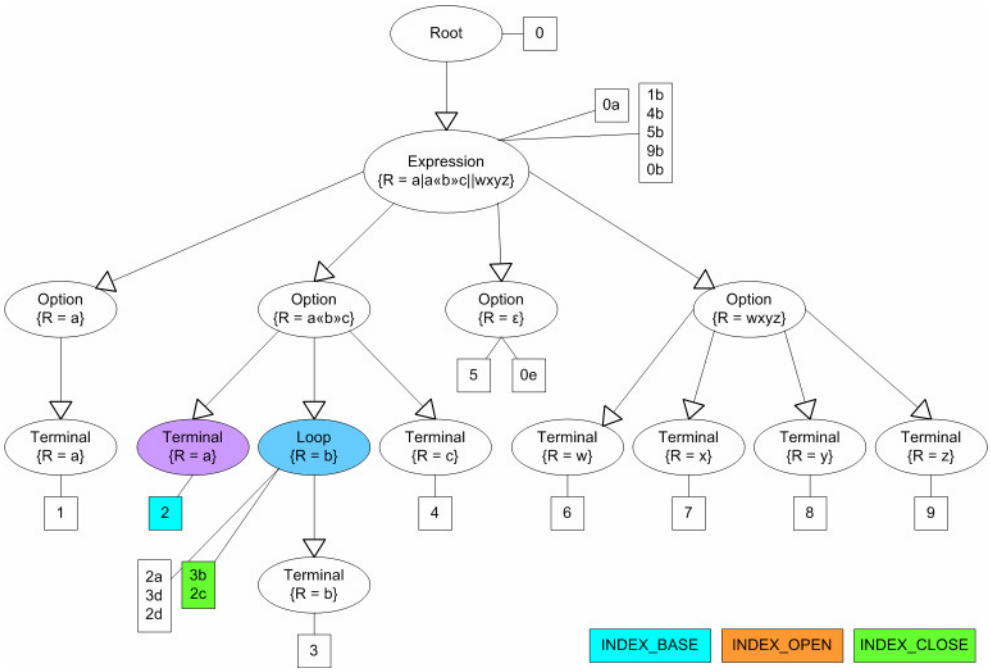


Abbildung 5-5: Anwendung der Regel c im DOM

5.2.1.5 Regel d

Regel d wird wie Regel c angewendet, wenn das Ziel der Indexübertragung eine Wiederholung ist – diese kann wiederholt werden. In diesem Fall werden die `INDEX_CLOSE`-Attributwerte – also die Indizes der schließenden Klammer – auf das `INDEX_OPEN`-Attribut und damit die öffnende Klammer desselben Knotens zurück geschrieben.

In Abbildung 5-6 sind die Knoten hervorgehoben, zwischen denen innerhalb des Ausdrucks `[a|a«b»c||wxyz]` Indexübertragungen nach Regel d erfolgen. Index 2 wird dabei nach Regel c auf die schließende Klammer der Wiederholung getragen, während Index 3 den gleichen Platz nach Regel b erreicht. Beide Indizes wandern anschließend entsprechend Regel d auf die öffnende Klammer der Wiederholung.

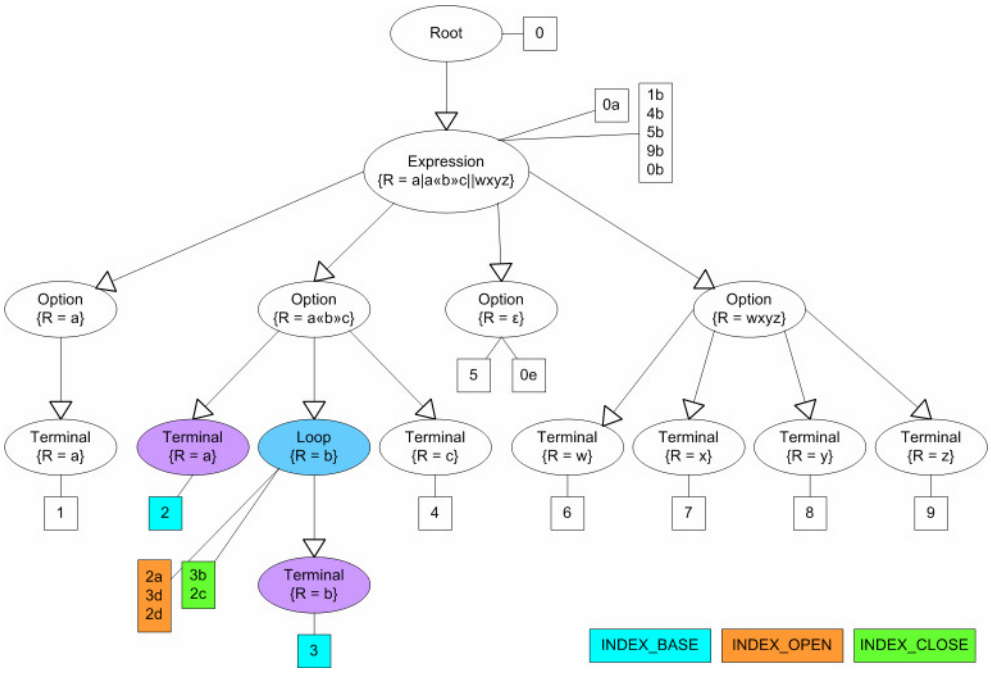


Abbildung 5-6: Anwendung der Regel d im DOM

5.2.1.6 Regel e

Das ε als Sonderfall ist das einzige Terminal, welches sowohl Grund- als auch abgeleitete Indizes besitzt. Die abgeleiteten Indizes werden in das `INDEX_OPEN`-Attribut gespeichert.

Regel e wird innerhalb der Bearbeitung von Regel a und Regel b aufgerufen, wenn das erste Terminalsymbol einer Option ein ε ist. Weiterhin wird diese Regel angewendet, wenn ε auf ein Terminalsymbol folgt.

Die Anwendung von Regel e direkt folgend auf Regel a wird in Abbildung 5-7 exemplarisch für den Ausdruck `[a|a«b»c||wxyz]` aufgezeigt: Index 0 wird vom Root- auf den Expression-Knoten bzw. dessen Optionen übertragen und Regel e wird direkt anschließend auf die leere Option angewendet.

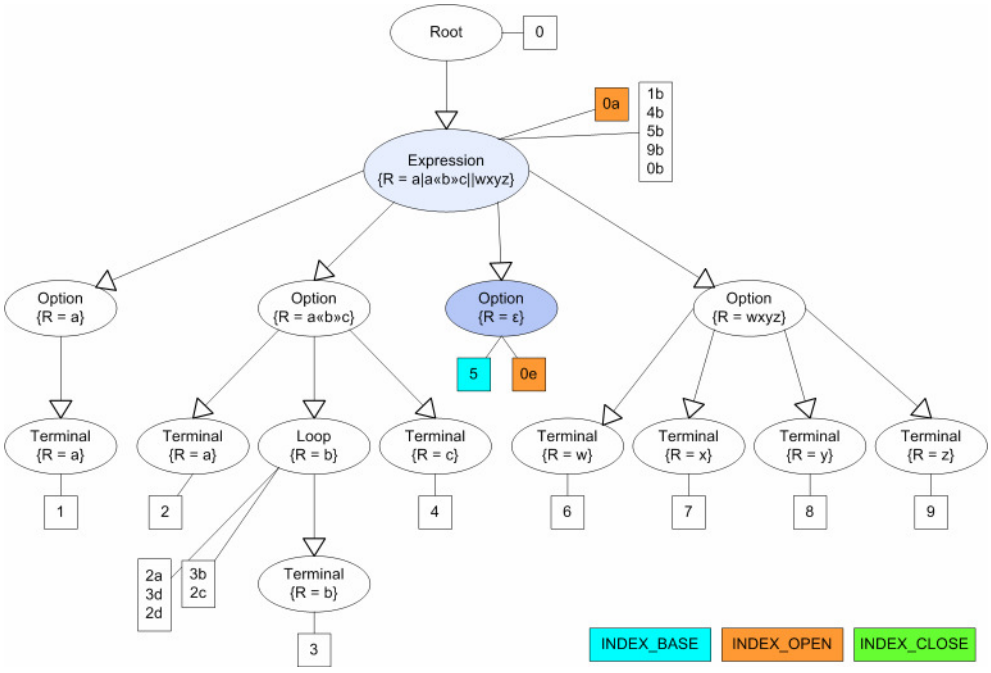


Abbildung 5-7: Anwendung der Regel e im DOM

5.2.1.7 Abschluss der Indizierung

Das erfolgreich indizierte DOM für den Ausdruck `[a|a«b»c||wxyz]` ist in Abbildung 5-8 dargestellt.

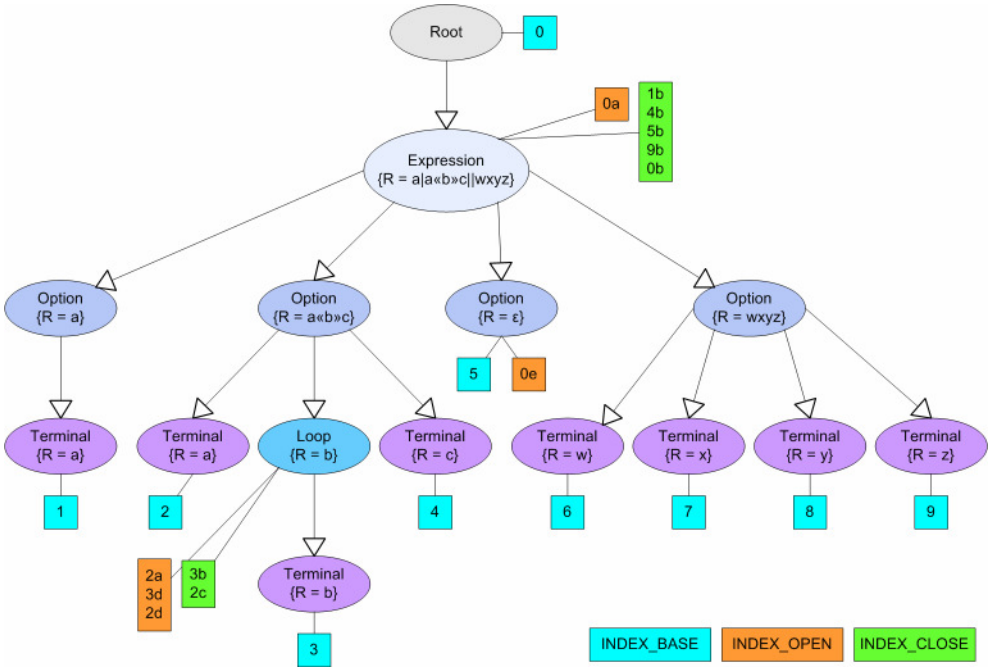


Abbildung 5-8: vollständig indiziertes DOM

Zum Abschluss der Indizierung muss sich nun die Umwandlung der Index-Baumstruktur in ein zur Visualisierung geeignetes Format anschließen. Dieses wird in Form einer Tabelle durch die Klasse `IndexTable.cs` implementiert.

Jedes Token des ursprünglichen regulären Ausdrucks wird mit den passenden generierten Indizes verknüpft. Das resultierende Tabellen-Objekt stellt somit den unmittelbaren Input für die Visualisierungsmethode der Klasse `ViewGenIndex` dar. Abbildung 5-9 zeigt die Tabelle mit dem indizierten Ausdruck, wie sie dem Nutzer der Software präsentiert wird.

R 0	[	a 1	|	a 2	«	b 3	»	c 4	|	ε 5	|	w 6	x 7	y 8	z 9	]
	0a		0a						0a	0e	0a					0b
																1b
					2ad		2c									
					3d		3b									
																4b
																5b
																9b

Abbildung 5-9: Ausgabe – indizierter Ausdruck

Weiterhin werden die Indizes des DOM in eine Indexliste überführt, die als Basis für die nachfolgende Berechnung der Zustandsüberführungsfunktion dient. Diese Liste enthält `Indexx`-Objekte (siehe Abschnitt 4.5.1), die alle Informationen über die Token des regulären Ausdrucks und ihre Indizes zusammenfassen:

- die Token des regulären Ausdrucks
- `den` Grundindex, sofern für dieses Token anwendbar
- alle abgeleiteten Indizes dieses Tokens
- ein Flag, ob es sich bei dem Token um ein Terminalsymbol handelt

5.2.2 Die Zustandsüberführungsfunktion

Die Berechnung der Zustandsüberführungsfunktion wird durch die Klasse `GenDelta` implementiert. Vom Controller CoreMain.cs wird dabei die Methode `Create_Delta()` aufgerufen, deren Input die oben beschriebene `IndexList` ist.

Die Ermittlung aller Zustandsübergänge beginnt mit dem Zustand S_0, dem per definitionem der Index 0 zugeordnet ist. Für jedes mögliche Eingabezeichen, an dessen Vorgrundstelle sich dieser Index 0 befindet, werden nun die erreichbaren Indizes, d.h. die an der Grundstelle des Eingabezeichens befindlichen Indizes ermittelt. Diese bilden zusammengefasst als Menge den Zustand, der bei diesem Zustandsübergang erreicht wird.

Nicht jede Kombination von Ausgangszustand und Eingabezeichen ermöglicht einen Zustandsübergang. Ist die Menge der erreichbaren Indizes leer, so wird dies mit dem Fehlerzustand gekennzeichnet.

Die bereits berechneten Zustände werden in einer Liste gesammelt und sukzessive bearbeitet, d.h. als Ausgangszustand für einen potentiellen Zustandsübergang eingesetzt. Die Berechnung ist beendet, wenn diese Zustandsliste keine unbearbeiteten Zustände mehr enthält.

Abschließend werden noch die Endzustände berechnet, d.h. diejenigen Zustände ermittelt, die mindestens einen der Indizes enthalten, die am Ende des indizierten regulären Ausdrucks stehen.

In Abbildung 5-10 ist ein Ausschnitt der vom Programm erzeugten Zustandsüberführungsfunktion abgebildet:

	Input: a	Input: b	Input: c	
S_0 {0}	$\delta(S_0,a) = \{1,2\} = S_1$	$\delta(S_0,b) = \{\varnothing\} = S(e)$	$\delta(S_0,c) = \{\varnothing\} = S(e)$	$\delta(S_0,$
S_1 {1,2}	$\delta(S_1,a) = \{\varnothing\} = S(e)$	$\delta(S_1,b) = \{3\} = S_3$	$\delta(S_1,c) = \{4\} = S_4$	$\delta(S_1,$
S_2 {6}	$\delta(S_2,a) = \{\varnothing\} = S(e)$	$\delta(S_2,b) = \{\varnothing\} = S(e)$	$\delta(S_2,c) = \{\varnothing\} = S(e)$	$\delta(S_2,$
S_3 {3}	$\delta(S_3,a) = \{\varnothing\} = S(e)$	$\delta(S_3,b) = \{3\} = S_3$	$\delta(S_3,c) = \{4\} = S_4$	$\delta(S_3,$
S_4 {4}	$\delta(S_4,a) = \{\varnothing\} = S(e)$	$\delta(S_4,b) = \{\varnothing\} = S(e)$	$\delta(S_4,c) = \{\varnothing\} = S(e)$	$\delta(S_4,$
S_5 {7}	$\delta(S_5,a) = \{\varnothing\} = S(e)$	$\delta(S_5,b) = \{\varnothing\} = S(e)$	$\delta(S_5,c) = \{\varnothing\} = S(e)$	$\delta(S_5,$
S_6 {8}	$\delta(S_6,a) = \{\varnothing\} = S(e)$	$\delta(S_6,b) = \{\varnothing\} = S(e)$	$\delta(S_6,c) = \{\varnothing\} = S(e)$	$\delta(S_6,$
S_7 {9}	$\delta(S_7,a) = \{\varnothing\} = S(e)$	$\delta(S_7,b) = \{\varnothing\} = S(e)$	$\delta(S_7,c) = \{\varnothing\} = S(e)$	$\delta(S_7,$
Endzustände: 0, 1, 4, 7				

Abbildung 5-10: Ausgabe – Zustandsüberführungsfunktion

5.2.3 Das Zustandsdiagramm

Die Darstellung der Zustandsüberführungsfunktion in einem Zustandsdiagramm gehört im Rahmen dieser Arbeit zu den optionalen Anforderungen. Es wurde daher nach einer unkomplizierten und schnellen Implementierung gesucht, die im Hinblick auf die darzustellenden gerichteten Graphen zwar keinen Anspruch auf Perfektionismus erhebt, jedoch zumindest für kleinere Diagramme den Sachverhalt schlüssig und für den Anwender hilfreich darstellen kann.

Im ersten Schritt werden die darzustellenden Zustände in vier Gruppen eingeteilt, die im Folgenden in vier nebeneinander liegenden Spalten eines Bitmaps dargestellt werden sollen:

1. Spalte: enthält nur den Anfangszustand S_0

2. Spalte: enthält alle Zustände, die direkt von S_0 aus erreicht werden können

3. Spalte: enthält alle Zustände, die nicht in den anderen Spalten enthalten sind

4. Spalte: enthält alle Endzustände, die nicht schon in Spalte 1 und 2 enthalten sind

Im Weiteren wird:

- für jeden Zustand ein Kreis gezeichnet, dessen Positionierung sich aus der Anzahl aller Zustände in den jeweiligen Spalten ergibt,
- für jeden Zustandsübergang ein Pfeil erzeugt, der zu einem anderen Zustand oder auf den Ausgangszustand zurück führt,
- für jedes an einem Zustandsübergang beteiligte Terminal ein Buchstabe an den passenden Pfeil geschrieben.

In Abbildung 5-11 und Abbildung 5-12 wird das Prinzip exemplarisch am Beispiel des Zustandsdiagramms für den regulären Ausdruck "`a|a«b»c||wxyz`" dargestellt.

Die so erzeugten Bitmaps weisen bei zunehmender Anzahl von Zuständen deutliche und störende Überlagerungen vor allem der Pfeile und Buchstaben auf. Für kleinere Diagramme ist der visuelle und didaktische Effekt jedoch ausreichend.

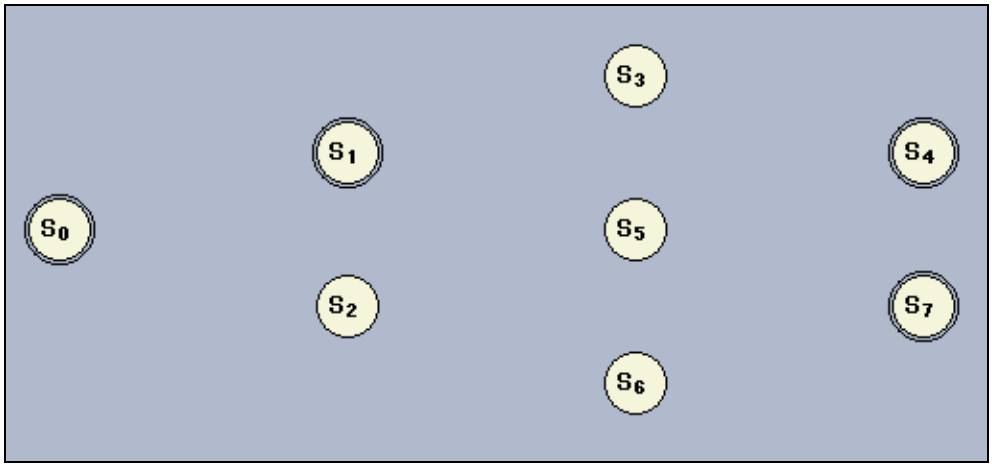


Abbildung 5-11: Verteilung der Zustands-Kreise auf vier Spalten

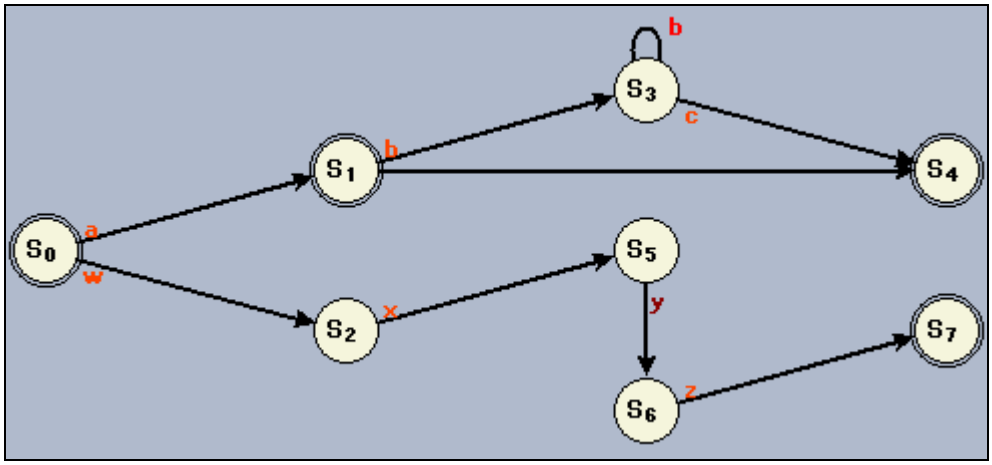


Abbildung 5-12: Ausgabe – Zustandsdiagramm

6 Softwaretest

Das Testen der erstellten Software ist ein sehr wichtiger Bestandteil des Entwicklungsprozesses. Automatisierte Tests dienen nicht nur der Auslieferung eines möglichst fehlerarmen Produktes, sondern sind auch Grundlage für effiziente Wartung und Weiterentwicklung einer bereits produktiven Software.

Für die hier zu testende Software können drei größere Bereiche unterschieden werden:

- Testen der Nutzer-Interaktionen inklusive Persistenz (GUI)
- Testen der DEA-Generierung
- Testen der Simulationen

Testfälle wurden unter Verwendung des Testframeworks NUnit in einem separaten Projekt innerhalb der Solution "DEA" erstellt.

6.1 GUI-Tests

Im begrenzten Rahmen dieser Diplomarbeit wurde in Bezug auf die GUI für manuelles Testen entschieden. Die Anforderungen – Erstellen, Umbenennen, Löschen von Projekten und Simulationen, Betätigung aller Menüelemente, Sprachumstellung etc. pp. – sind noch hinreichend überschaubar, um diese Vorgehensweise zu rechtfertigen. Für automatisierte Testpläne wäre die Erstellung von Skripten nötig, welche standardisierte Nutzeraktionen simulieren. Der Aufwand würde den Rahmen dieser Arbeit jedoch deutlich übersteigen.

6.2 Generator-Tests

Für die Generator-Tests wurden automatisierte Testpläne unter Einsatz des NUnit-Testframeworks erstellt. Es handelt sich um Black-Box-Tests, die nicht einzelne Methoden testen, sondern das Gesamtergebnis bzw. größere Abschnitte der Software.

6.2.1 Positiv-Test

Diese Tests stellen das Kernstück der Testung dar. Es wird ein Vergleich zwischen erwarteten und aktuell durch die Software erzeugten Daten des DEA-Generators unter Eingabe eines syntaktisch korrekten regulären Ausdrucks durchgeführt. Die Übereinstimmung der Werte dient als Nachweis für die Korrektheit der grundlegenden Algorithmen. Damit können diese Tests für die Weiterentwicklung bzw. Refaktorierung der Software als Prüfmittel eingesetzt werden.

Folgende Objekte wurden in den Identitätsvergleich aufgenommen:

- das DEADOM mit allen Grund- und abgeleiteten Indizes
- die Indexliste als Grundlage der Berechnung der Zustandsüberführungsfunktion
- die Indextabelle als Grundlage der Visualisierung des indizierten Ausdrucks
- die Zustandsliste als Ausdruck der Zustandsüberführungsfunktion
- die Zustandstabelle als Datencontainer für die Visualisierung der Zustandsüberführungsfunktion
- das Zustandsdiagramm in Form eines Bitmaps

Als Input für diese Tests wurde auch hier der im Theorieteil dieser Arbeit genutzte reguläre Ausdruck `a|a«b»c||wxyz` verwendet. Die Vorgabedaten, die für den Vergleich zur Verfügung stehen müssen, wurden teils konventionell durch manuelle Bearbeitung des regulären Ausdrucks, teils unter Zuhilfenahme von bereits fertig gestellten Softwareteilen im Debug-Modus erzeugt. Die Dateien, die diese Daten als einfachen Text enthalten, befinden sich im Ordner `test` des Views-Projektes und sind nach dem Benennungsschema `Test_Expected_Diplom_xxx.txt` abgelegt.

Der gleiche reguläre Ausdruck wird auch in den Testmethoden `Test_Diplom1`, `Test_Diplom2` und `Test_Diplom3` verwendet, die sukzessive den (erfolgreichen) Ablauf der DEA-Generierung testen:

- Test_Diplom1: die Indizierung des regulären Ausdrucks und die korrekte Erstellung des DOM, der Indexliste und der Indextabelle
- Test_Diplom2: die Berechnung der Zustandsüberführungsfunktion und die korrekte Erstellung der Zustandsliste und der Zustandstabelle
- Test_Diplom3: die Erzeugung des Zustandsdiagramms in Form eines Bitmaps

Während des Testlaufs werden die aktuellen Daten der DEA-Generierung ebenfalls in Dateien gespeichert, die im gleichen Ordner wie die Vorgabedateien unter dem Schema `Test_Actual_xxx.txt` abgelegt werden.

Über Assert-Methoden, die vom Testframework bereitgestellt werden, wird schließlich die Identität der erwarteten und vorliegenden Daten geprüft. Ist diese nicht gegeben, sind entweder die Algorithmen fehlerhaft oder das Format wurde geändert. Wenn letzteres gewünscht ist, sollte es daher nur im Verbund mit den Testmethoden geschehen.

6.2.2 Negativ-Test

Als Negativ-Test wurde die Eingabe eines relativ kurzen, syntaktisch inkorrekten regulären Ausdrucks gewählt: `a«b|c`. Die schließende spitze Klammer fehlt hier.

Die Indizierung und alle folgenden Schritte können nicht durchgeführt werden, in den Testmethoden werden daher alle Objekte auf Gleichheit mit NULL geprüft. Dies muss ausnahmslos der Fall sein.

6.3 Simulations-Tests

Auch die Simulations-Tests wurden mit Hilfe von NUnit in der gleichen Testklasse wie die Generator-Tests erstellt. Das Verfahren ist prinzipiell das Gleiche: Es wurden einmalig erwartete Daten erstellt, deren identische Erzeugung innerhalb der Testmethoden erfolgt.

Zu Beginn dieser Tests muss jeweils ein passender DEA-Generator erzeugt werden. In den vorliegenden Tests wird der gleiche reguläre Ausdruck verwendet wie in den Testfällen der Generierung.

Mit diesem korrekten DEA-Generator werden exemplarisch die drei verschiedenen Situationen getestet, die während der Simulationen entstehen können:

- ein Wort wird als Bestandteil der durch den regulären Ausdruck definierten Sprache akzeptiert: Beispielwort "`abbbc`"
- ein Wort wird nicht als Bestandteil der durch den regulären Ausdruck definierten Sprache akzeptiert, weil die Simulation in einem Zustand endet, der kein Endzustand ist: Beispielwort "`abbb`"
- ein Wort wird nicht als Bestandteil der durch den regulären Ausdruck definierten Sprache akzeptiert, weil die Simulation in einem Fehlerzustand endet: Beispielwort "`aaa`"

Die erzeugten Simulationsschritte müssen mit den erwarteten übereinstimmen.

7 Dokumentation

7.1 Bedienungs- und Installationsanleitung

7.2 Quellcode-Dokumentation

Eine vollständige Dokumentation des Quellcodes im CHM-Format ist Teil der vorliegenden Arbeit. Diese Hilfedatei wurde mit den Tools *Sandcastle* von Microsoft und *Sandcastle Help File Builder* von Eric Woodruff erstellt. Dabei dient letzteres als grafischer Aufsatz zur bequemen Bedienung von Sandcastle selbst, welches aus den im Quellcode enthaltenen Kommentaren im XML-Stil die Hilfedatei generiert. Diese Kommentare erfassen den hier besprochenen Quelltext vollständig, d.h. sämtliche Namensräume, Klassen und ihre privaten und öffentlichen Member und Methoden.

Sandcastle Help File Builder ermöglicht die Erstellung einer Projekt-Datei mit zahlreichen Einstellungen. Die für die hier besprochene Dokumentation erzeugte Projektdatei ist der Arbeit hinzugefügt.

Eine Benutzeranleitung, die den Einstieg in die Nutzung der Software erleichtern soll, liegt als HTML-Projekt vor. In dieser werden auch Kontaktmöglichkeiten angegeben, die für Fragen oder Anregungen an die Autorin genutzt werden können.

8 Zusammenfassung und Ausblick

8.1 Notwendige Anforderungen

Die im Rahmen dieser Diplomarbeit erstellte Software erfüllt alle in der Aufgabenstellung genannten Punkte:

- Die Interaktion mit dem Anwender erfolgt über eine integrierte grafische Oberfläche im MS-Windows-Stil. Es werden allgemein bekannte und akzeptierte Bedienelemente wie z.B. Menüleisten, Kontextmenüs oder die Trennung in Navigations- und Arbeitsbereich verwendet.
- Der deterministische endliche Automat wird aus einem vom Anwender eingegebenen regulären Ausdruck nach dessen Prüfung auf Korrektheit erzeugt und enthält sowohl die Indizierung des regulären Ausdrucks als auch die Zustandsüberführungsfunktion in der δ-Notation.
- Jeder einmal generierte DEA ermöglicht beliebig viele Simulationen von Eingabezeichenfolgen, die der Anwender schrittweise mit entsprechender Visualisierung der Zwischenergebnisse durchlaufen kann. Damit kann für beliebige Worte die Zugehörigkeit zur durch den regulären Ausdruck definierten Sprache ermittelt werden.
- Das Software-System umfasst eine Dokumentation des Quellcodes und eine Bedienungsanleitung im HTML-Format. Diese ist auch online unter `http://www.sabineludvik.de/dea/` erreichbar.

8.2 Optionale Anforderungen

Darüber hinaus wurde zusätzliche Funktionalität implementiert, die sowohl die Benutzerfreundlichkeit als auch den Kreis der Zielpersonen erhöhen soll:

- Eine einfache Implementierung zur Erzeugung von Zustandsdiagrammen verbessert das didaktische Potential der Software.
- Durch die Mehrsprachigkeit der Software, die das Umschalten zwischen den angebotenen Sprachen zur Laufzeit ermöglicht, kann ein breiterer Personenkreis erreicht werden.
- Es können zu den bereits im Projekt vorhandenen (derzeit Deutsch, Englisch, Russisch und Italienisch) weitere Sprachen hinzugefügt werden. Zu deren Aktivierung sind die Übersetzung der Sprachdatei und das Einfügen eines entsprechenden Menüeintrags im Menü Sprachen nötig. Die Software-Assembly muss anschließend neu erstellt werden, bevor die neue Sprache genutzt werden kann.

8.3 Ausblick

Die Software-Version, die Bestandteil der vorliegenden Arbeit ist, weist an einigen Stellen Unzulänglichkeiten auf, die in weiteren Versionen behoben werden sollten. Dazu gehören:

- die fehlende Druckfunktion,
- neben dem Projekt- und Simulationsmanagement auch eine Ordnerverwaltung und
- das einfache Schließen von Projekten ohne ein neues zu öffnen.

Für diese und andere Erweiterungen steht die Autorin prinzipiell zur Verfügung.

Abkürzungsverzeichnis

CHM	Compiled Help Modules
DOM	Document Object Model
GUI	graphical user interface (grafische Benutzerschnittstelle)
MVC-Modell	Model-View-Controller-Modell

Abbildungsverzeichnis

Tabellenverzeichnis

Formelverzeichnis

Quellenverzeichnis

Literaturverzeichnis

N. Chomsky (1956): *Three models for the description of language.* In: IRE Transactions on Information Theory 2, S. 113–124

V. M. Gluschkow (1963): *Theorie der abstrakten Automaten. Mathematische Forschungsberichte (Hrsg: Prof. Dr. Heinrich Grell)*. Berlin: VEB Deutscher Verlag der Wissenschaften

R. Socher (2005): *Theoretische Grundlagen der Informatik.* Leipzig: Fachbuchverlag Leipzig

Ergänzende Literatur

J. Hopcroft, J. Ullman, R. Motwani: *Einführung in die Automatentheorie, Formale Sprachen und Komplexitätstheorie*. Addison-Wesley

A. Aho, R. Sethi, J. Ullman: *Compilerbau, Teil1*. Oldenbourg-Verlag

G. Goos: *Vorlesungen über Informatik, Band3: Berechenbarkeit, formale Sprachen, Spezifikationen*. Springer Verlag

R. Wilhelm, D. Maurer: *Übersetzerbau*. Springer Verlag

H. Ehrig, B. Mahr, F. Cornelius, M. Große-Rhode, P. Zeitz: *Mathematisch-strukturelle Grundlagen für Informatiker*. Springer Verlag

U. Schöning: *Logik für Informatiker*. Spektrum Akademischer Verlag

Ehrenwörtliche Erklärung

Ich erkläre hiermit an Eides Statt,

dass ich die vorliegende Diplomarbeit selbstständig angefertigt, keine anderen als die angegebenen Quellen benutzt, die wörtlich oder dem Inhalt nach aus fremden Arbeiten entnommenen Stellen, bildlichen Darstellungen und dergleichen als solche genau kenntlich gemacht und keine unerlaubte fremde Hilfe in Anspruch genommen habe.

Halle, 16.9.2007