Data Mining und natürlichsprachliche Verbalmorphologien

Prof. Dr. Alfred Holl

Gordon Zimnik, B. Sc.

Georg-Simon-Ohm-Hochschule Nürnberg
Fakultät Informatik

Abstract

Die Analyse von Daten mit dem Ziel der Mustererkennung ist vor allem in der Betriebswirtschaft weit verbreitet (z.B. für Verkaufsdaten). Aber auch Sprachwissenschaftler führen Datenanalysen durch, wenn sie grammatische Regeln aufstellen. Die Informatik kennt unter dem Namen Data Mining viele verschiedene Vorgehensweisen zur Datenanalyse, darunter Clusteranalyse-Methoden. Ein solcher Algorithmus kann in der Linguistik einzelsprachunabhängig für die Analyse von flexionsmorphologischen Systemen, meist Verbalsystemen, eingesetzt werden. Der jeweils entstehende Regelapparat in Gestalt eines Verbregisters besteht aus morphologisch homogenen Clustern, die rückläufig ähnliche, morphologisch analoge Vertreter der Wortart Verb enthalten.
Diese Regeln liefern sprachdidaktisch verwertbare, quantitative und strukturelle Aussagen über flexionsmorphologische Systeme. Eine wichtige Anwendung ist es, ein beliebiges Verb automatisch seinem Cluster zuzuordnen, d.h. die Konjugationsklasse des Verbs zu ermitteln. Einen dafür geeigneten Algorithmus zu entwickeln, bildet den Gegenstand dieser Untersuchung. Die Entwicklung geschieht linguistisch motiviert, mit englischen Beispielen illustriert und informationstechnisch dokumentiert. Dabei werden Prinzipien des Software Engineerings im Rahmen eines Phasenkonzepts konsequent eingehalten.
Diese Untersuchung ist die Erweiterung einer Bachelorarbeit an der Fakultät Informatik im SoSe 2008. Sie steht in der mehrjährigen Tradition von Forschungsprojekten und Publikationen zum Thema „Data Mining flexionsmorphologischer Systeme“ an der Ohm-Hochschule. Angewendet auf die Verbalsysteme verschiedener Schulsprachen, wurden die Resultate dieser Studie erstmals anlässlich der Technikmeile Ende Juli 2008 einer breiten Öffentlichkeit vorgestellt.

ISSN 1616-0762 Sonderdruck Schriftenreihe der Georg-Simon-Ohm-Hochschule Nürnberg Nr. 43, Nürnberg 2008

Inhaltsverzeichnis

1. Einleitung 5
2. Grundlagen und Zielsetzungen 5
2.1 Historie 5
2.2 Grundidee und Motivation 6
3. Konzept des morphologischen Data Minings 8
3.1 Stand der Forschung 8
3.1.1 Analytischer Teilbereich: Ermittlung morphologisch homogener Cluster 10
3.1.2 Erster Schritt des synthetisch-generativen Teilbereichs: Auffindung passender Regellexeme 10
3.1.3 Zweiter Schritt des synthetisch-generativen Teilbereichs: Erzeugung der Schlüsselformen des Suchlexems 11
3.2 Beschreibung des analytischen Verarbeitungsschritts 11
3.2.1 Struktur der Lexemregister 11
3.2.2 Die drei Phasen des Data-Mining-Prozesses 14
3.3 Konzept des synthetisch-generativen Algorithmus 15
3.3.1 Grundidee des Algorithmus 15
3.3.2 Definition des Inputs 22
3.3.3 Definition des Suchlexems 22
3.3.4 Funktion des Algorithmus 22
3.3.5 Funktion “total length compare“ 24
3.3.6 Funktion “prefix cut“ 25
3.3.7 Funktion “longest match“ 26
3.4 Konsequenzen für die Gestaltung der Register 30
4. Implementierung des synthetisch-generativen Algorithmus 31
4.1 Art der Implementierung 31
4.1.1 Programmaufbau 31
4.1.2 Einsatz und Verwendung von Programmiersprachen 32
4.2 Realisierung der Funktionen 32
4.2.1 Hauptprogramm 33
4.2.2 Realisierung der Funktion “total length compare“ 37
4.2.3 Realisierung der Funktion “prefix cut“ 39
4.2.4 Realisierung der Funktion “longest match“ 42
4.2.5 Zusatzfunktion für den Datenimport 51
5. Anwendungsmöglichkeiten 52

1. Einleitung

In dieser Untersuchung wird Data Mining als Teildisziplin der Informatik mit der linguistischen Analyse flexionsmorphologischer Systeme in natürlichen Sprachen verbunden. Datenanalyse in der Linguistik ist nichts grundsätzlich Neues. Jeder Linguist, der morphologische, syntaktische, phonologische oder ähnliche Regeln aufstellen will, führt eine Datenanalyse durch, wenn er linguistisches Material (Texte, Grammatiken, Wörterbücher) analysiert. Das gilt insbesondere für morphologische Systeme. Jede bestehende Liste unregelmäßiger Verben ist beispielsweise aus einem Datenanalyseprozess hervorgegangen.
Diese Untersuchung nimmt durch Data Mining ermittelte Register flexionsmorphologischer Regellexeme für jeweils eine Wortart einer natürlichen Sprache zum Ausgangspunkt. Ziel ist es, einen Algorithmus zu entwickeln, der zu einem beliebigen Suchlexem dessen Regellexem bzw. Regellexeme liefert. Die hier verwendete Terminologie wird in 3.2 genau definiert.
Kapitel 2 beschreibt die Grundlagen und Zielsetzungen, welche die Basis für die Konzipierung und spätere Implementierung des Algorithmus bilden, und erklärt die dahinter stehende Forschungsintention. In Kapitel 3 und 4 wird ein strukturierter Software-Entwicklungsprozess verfolgt (Abb. 1.01). Kapitel 3 zeigt schrittweise die Konzeption des Algorithmus. In Kapitel 4 wird die technische Realisierung des Algorithmus im Detail erklärt. Kapitel 5 zeigt die Möglichkeiten und Einsatzgebiete dieser Forschungsergebnisse.

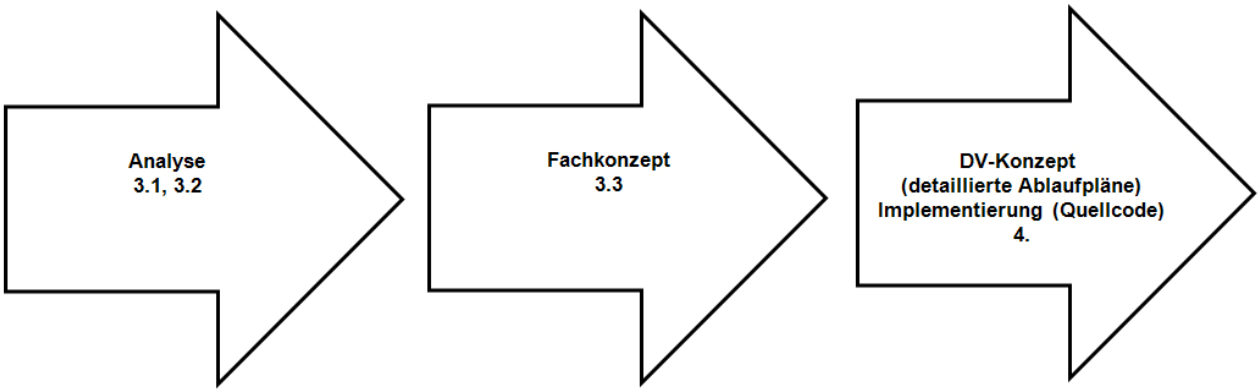

Abbildung 1.01: Phasen des Software-Entwicklungsprozesses
(vgl. Mayr / Hess, 2008, 381, Abb. 4)

2. Grundlagen und Zielsetzungen

In 2.1 wird die Historie dieser Art von Sprachforschung und deren Zielverfolgung erörtert. 2.2 verdeutlicht die über Jahrzehnte reichende Forschungsintention von Alfred Holl.

2.1 Historie

Die ersten Vorarbeiten zu dieser Thematik begannen mit einer Dissertation über Verbalsysteme in der Latein-Romania (Holl 1988), damals noch mit der manuellen Durchführung formalisierbarer Analysealgorithmen. Durch die Finanzierung der bayerischen HighTech-Offensive und der Staedtler-Stiftung Nürnberg war es an der Fakultät Informatik der Georg-Simon-Ohm-Hochschule Nürnberg möglich, diesen Forschungsansatz in mehreren Teilprojekten bis heute weiterzuführen (vgl. 3.1).

2.2 Grundidee und Motivation[1]

Zur Analyse morphologischer Systeme werden Verfahren des Data Mining verwendet. „Data Mining bedeutet buchstäblich Schürfen oder Graben in Daten ... Die Ergebnisse lassen Muster in Daten erkennen, weswegen Data Mining auch als Datenmustererkennung übersetzt wird“ (Alpar / Niedereichholz, 2000, 3). Man sucht nach nicht bekannten, versteckten Zusammenhängen und Ähnlichkeiten, deren Kenntnis einen wirtschaftlichen oder wissenschaftlichen Nutzen verspricht. Für die Disziplin der Datenanalyse finden sich statt des Ausdrucks „Data Mining“ – entsprechend der bekannten Instabilität der Terminologie in der Informatik – weitere, wie „Information Mining“ oder „Knowledge Discovery in Databases (KDD)“, ohne dass eine genaue definitorische Abgrenzung möglich wäre. Der gesamte Bereich umfasst heute eine Vielzahl unterschiedlicher, teils statistischer Methoden, die verbreitet zur Analyse großer Datenbestände eingesetzt werden, etwa im Marketing, um Regelmäßigkeiten im Kundenverhalten festzustellen oder um aus potentiellen Kunden besonders viel versprechende herauszufiltern.

Die im Rahmen dieser Forschungsarbeit zu verwendenden Datenanalyse-Verfahren gehören zu der variantenreichen Gruppe der Clusteranalyse-Algorithmen. Unter Clustern versteht man in diesem Zusammenhang ganz einfach Mengen von Datensätzen. Ein Datensatz ist ein Tupel (eine Aneinanderreihung) zusammengehöriger Einzeldaten, im vorliegenden Zusammenhang die Schlüsselformen eines Lexems[2], etwa die Stammreihe eines Verbs (im Deutschen z.B. *gehen, ging, gegangen*) mit dem Verweis auf ein Musterparadigma, ggf. mit Angabe der Bedeutung, falls der Konjugationstyp von ihr abhängt (z.B. bei *schleifen*, *schliff*, *geschliffen* ‘schärfen‘ vs. *schleifen*, *schleifte*, *geschleift* ‘zerstören‘). Ziel einer Clusteranalyse ist es immer, Cluster mit möglichst ähnlichen Elementen (Datensätzen) zu ermitteln, mit anderen Worten „Daten (semi-)automatisch so in Kategorien, Klassen oder Gruppen (Cluster) einzuteilen, dass Objekte im gleichen Cluster möglichst ähnlich und Objekte aus verschiedenen Clustern möglichst unähnlich zueinander sind“ (Ester / Sander, 2000, 45). Ähnlichkeit ist den jeweiligen Anforderungen entsprechend zu definieren.

Auch bisherige ausführliche Darstellungen von Verbalsystemen sind Produkte „manueller“ Clusteranalyse. Wird etwa zu jedem Musterverb eine Liste aller Verben mit den gleichen Unregelmäßigkeiten angegeben oder in einem Gesamtverzeichnis aller Verben zu jedem das Sigel des zugehörigen Musterverbs, so werden – im Sprachgebrauch des Data Mining – Cluster aus jeweils morphologisch analogen Verben gebildet. Morphologische Analogie bedeutet das Vorhandensein der gleichen morphologischen Eigenschaften, bei Verben also der gleichen Konjugationsmerkmale (Stammalternanzen und Personalendungen). Doch was nützen derartige Cluster unter einem sprachdidaktischen Blickwinkel? Sie sind so umfangreich und unstrukturiert, dass es unmöglich ist, sie auswendig zu lernen.

Sprachstudierende gehen in ihren Lernprozessen daher andere Wege. Sie wollen und müssen ihren Lernaufwand minimieren und würden am liebsten an der lexikalischen Grundform eines Verbs, häufig dem Infinitiv Präsens Aktiv[3], auch dessen Unregelmäßigkeiten ablesen können, so wie man tatsächlich den Konjugationstyp regelmäßiger Verben in romanischen Sprachen an der Infinitivendung erkennt. Ist nun aber mühselig ein Averbo[4] gelernt, so versucht der Lernende in einer ersten Stufe, aus diesem Wissen größtmögliches Kapital zu

[1] Grundidee und Motivation wurden aus (Holl, 2006, 14-18) übernommen.

[2] Wir verwenden den Terminus *Lexem* als Bezeichnung für ein „Wort“ in der Form, wie es in einem Lexikon eingetragen ist. Ein Lexem ist eine abstrakte Basiseinheit des Lexikons, die in verschiedenen Flexionsformen auftreten kann. Es wird durch seine lexikalische Grundform (Lemma) repräsentiert; bei Verben ist dies meist der Infinitiv Präsens Aktiv.

[3] Um die Flexionsformbestimmungen handlicher zu gestalten, wird die Angabe „*Person*“ stets weggelassen. Bei der Diathese wird „*Passiv*“ stets genannt, „*Aktiv*“ nicht immer. Statt „*Infinitiv Präsens Aktiv*“ sagen wir kurz „*Infinitiv*“ oder „*Infinitiv Präsens*“.

[4] Unter dem *Averbo* eines flektierbaren Lexems versteht man die geordnete Menge seiner Flexionsformen.

schlagen und es auf weitere „ähnliche“ Verben anzuwenden, die er dann zu den gleichen morphologischen Eigenschaften „verurteilt“. Als tertium comparationis[5] zur „Feststellung“ der Ähnlichkeit wird die rückläufige Ähnlichkeit gewählt.

Die rückläufige Ähnlichkeit (Ausgangsgleichheit) manifestiert sich in einem gemeinsamen Ausgang in der lexikalischen Grundform, wobei hier der Terminus „Ausgang“ nicht als morphologische Kategorie im Sinne von „Endung“ verstanden wird, sondern ganz einfach als Buchstabenfolge am Ende eines Wortes, deren Länge jeweils pragmatisch festgelegt wird. Ein *n*-stelliger Ausgang sei definiert als Ausgang der Länge *n*, d.h. als die letzten (schließenden) *n* Buchstaben einer lexikalischen Grundform.

Als Beispiele für die genannte Lernstrategie seien die deutschen Verben *gehen*, *flehen* und *drehen* genannt. Bei ihnen besteht Ähnlichkeit in den letzten 4 Buchstaben, dem Ausgang –*ehen*. Würde ein Sprachlernender das Verb *flehen* mit seinen Schlüsselformen (*flehen*, *flehte*, *gefleht*) als erstes dieser Gruppe lernen, würde er diese Regelmäßigkeit auf die beiden anderen übertragen wollen. Für *drehen* (*drehte*, *gedreht*) stimmt dieser Transfer, für *gehen* (*ging*, *gegangen*) allerdings nicht. Letzteres muss als explizite Ausnahme der Verben mit dem Ausgang –*ehen* gelernt werden.

Diese Strategie stimmt bei jedem Simplex und seinen präfigierten Verben (mit wenigen Ausnahmen), bei den regelmäßigen und einem Teil der unregelmäßigen Verben (im Deutschen bei etwa einem Neuntel [Holl, 2002, 159]), aber nicht durchgängig, was eine erhebliche Fehlerquelle bedeutet. Diese Strategie kann also hilfreich oder irreführend sein. Erst wenn sie scheitert, werden in einer zweiten Stufe weitere Verben mit ihren Konjugationsbesonderheiten intensiv gelernt.

Hier treten zwei konkurrierende Formen von Ähnlichkeit auf: Die rückläufige Ähnlichkeit der lexikalischen Grundform und die morphologische Analogie. Diese sind keineswegs deckungsgleich, vor allem kann man bei einer Wortart nicht von der rückläufigen Ähnlichkeit zweier lexikalischer Grundformen auf die morphologische Analogie der zugehörigen Lexeme schließen. Nun ist das der genannten Strategie zugrunde liegende analogische Denken (der Analogieschluss von partieller auf totale Ähnlichkeit) ein allgemeines, wesentliches – häufig unbewusstes – Grundprinzip menschlichen Lernens und Denkens. Es kann also nicht einfach ausgeschaltet werden, sondern man muss bewusst damit umgehen, der Sprachlernende ebenso wie der Sprachlehrende. Es ist am besten, dem Sprachlernenden von vornherein zu zeigen, in welchen Fällen rückläufige Ähnlichkeit morphologische Analogie impliziert und in welchen nicht. Das ist in Holl, 2002, 151-158 ausführlich gezeigt.

Es ist daher der bisherigen Form der Clusteranalyse morphologischer Systeme, die für Nachschlagewerke weiterhin ihren Sinn behält, eine zweite hinzuzufügen, die ganz gezielt den Spezifika analogischen Denkens auf der Basis rückläufiger Ähnlichkeit sprachdidaktisch Rechnung trägt. Ziel ist die automatisierte explizite Ermittlung homogener Cluster ausgangsgleicher Lexeme einer bestimmten Wortart. Ein Cluster heiße morphologisch homogen (im Folgenden kurz homogen), wenn alle seine Lexeme morphologisch analog sind. Das bedeutet, dass z.B. die Verben *rasieren*, *organisieren*, *charakterisieren* usw., d.h. diejenigen mit dem Ausgang –*sieren*, ein homogenes Cluster bilden, da sie die gleichen Konjugationsmerkmale haben.

Für diese Form der Clusteranalyse wird die Menge aller Lexeme einer Wortart – in Gestalt ihrer lexikalischen Grundformen – nicht ungeordnet oder in der üblichen alphabetischen Sortierung von links betrachtet, sondern als rückläufig, also von rechts alphabetisch sortierte Menge. Dadurch werden ausgangsgleiche lexikalische Grundformen benachbart angeordnet. Geordnete Grundmengen liegen in uns aus der Informatik bekannten Clusteranalysen nicht vor, so dass dort übliche Algorithmen nicht verwendet werden können.

[5] Tertium comparationis (lat. „das Dritte des Vergleichs“) ist die Eigenschaft oder Dimension, die zwei zu vergleichende Gegenstände gemeinsam haben und die den Vergleich erst ermöglicht.

Bei der Entwicklung von Clusteranalyse-Verfahren ist generell zwischen zwei Grundtypen zu unterscheiden: „Bei den meisten Varianten wird so verfahren, dass entweder jedes zu gruppierende Objekt als ein Anfangscluster oder alle Objekte als ein Cluster gewählt werden. Danach werden die Anfangscluster zusammengefasst oder das alle Objekte umfassende Cluster aufgespaltet. In beiden Fällen geschieht das so, dass die Abstände zwischen den Elementen eines Clusters möglichst gering werden“ (Alpar / Niedereichholz, 2000, 11). Im vorliegenden Fall ist diese Abstandsbedingung sehr einfach; sie besagt, dass größtmögliche Cluster ermittelt werden, die morphologisch analoge, ausgangsgleiche Lexeme enthalten. Die Strukturierung des allumfassenden Anfangsclusters wird auch als Top-down-Clusteranalyse bezeichnet (divisiv), die Zusammenfassung ähnlicher einelementiger Anfangscluster als Bottom-up-Clusteranalyse (agglomerativ). Bei dem vorliegenden Projekt wird die erste Variante verwendet, da ihre Algorithmen einfacher und besser implementierbar sind.

Als Ergebnis erhält man eine stabile, weitgehend objektivierbare Basis, auf der die Nachbearbeitung aufbauen kann.

3. Konzept des morphologischen Data Minings

3.1 gliedert den betrachteten Forschungsansatz in einen analytischen und einen synthetisch-generativen Teilbereich und zeigt den jeweiligen aktuellen Forschungsstand. 3.2 analysiert anhand des englischen Verbalsystems als Beispiel die Register, welche den Ausgangspunkt für den zu entwickelnden Algorithmus bilden, und gibt eine detaillierte Darstellung der Phasen eines Data-Mining-Prozesses. 3.3 befasst sich mit der Modellierung des synthetischen Algorithmus und erklärt dessen Funktionsweise. In 3.4 werden Konventionen für die Register der Verbalsysteme festgelegt, welche die bereits entwickelten Register optimieren und richtungsweisend für die Erstellung zukünftiger sind. Im Zusammenhang damit steht die Idee, den Algorithmus so zu konzipieren, dass er unabhängig von einem speziellen Verbalsystem und dem damit verbundenen Register Anwendung finden kann.

3.1 Stand der Forschung

Der vorliegende Forschungsgegenstand untergliedert sich in zwei Teilbereiche, einen analytischen und einen synthetisch-generativen. 3.1.1 beschäftigt sich mit dem analytischen Teilbereich, d.h. mit der Entwicklung von Registern (von Regellexemen) für die unterschiedlichsten Verbalsysteme durch morphologisches Data Mining. 3.1.2 und 3.1.3 beziehen sich auf die synthetisch-generativen Teilbereiche. 3.1.2 betrifft das Auffinden von passenden Regellexemen zu einem Suchlexem. 3.1.3 gibt einen Ausblick auf die Erzeugung der Schlüsselformen des Suchlexems.
Diese Untersuchung beschäftigt sich schwerpunktmäßig mit dem ersten Schritt des synthetisch-generativen Teilbereichs.

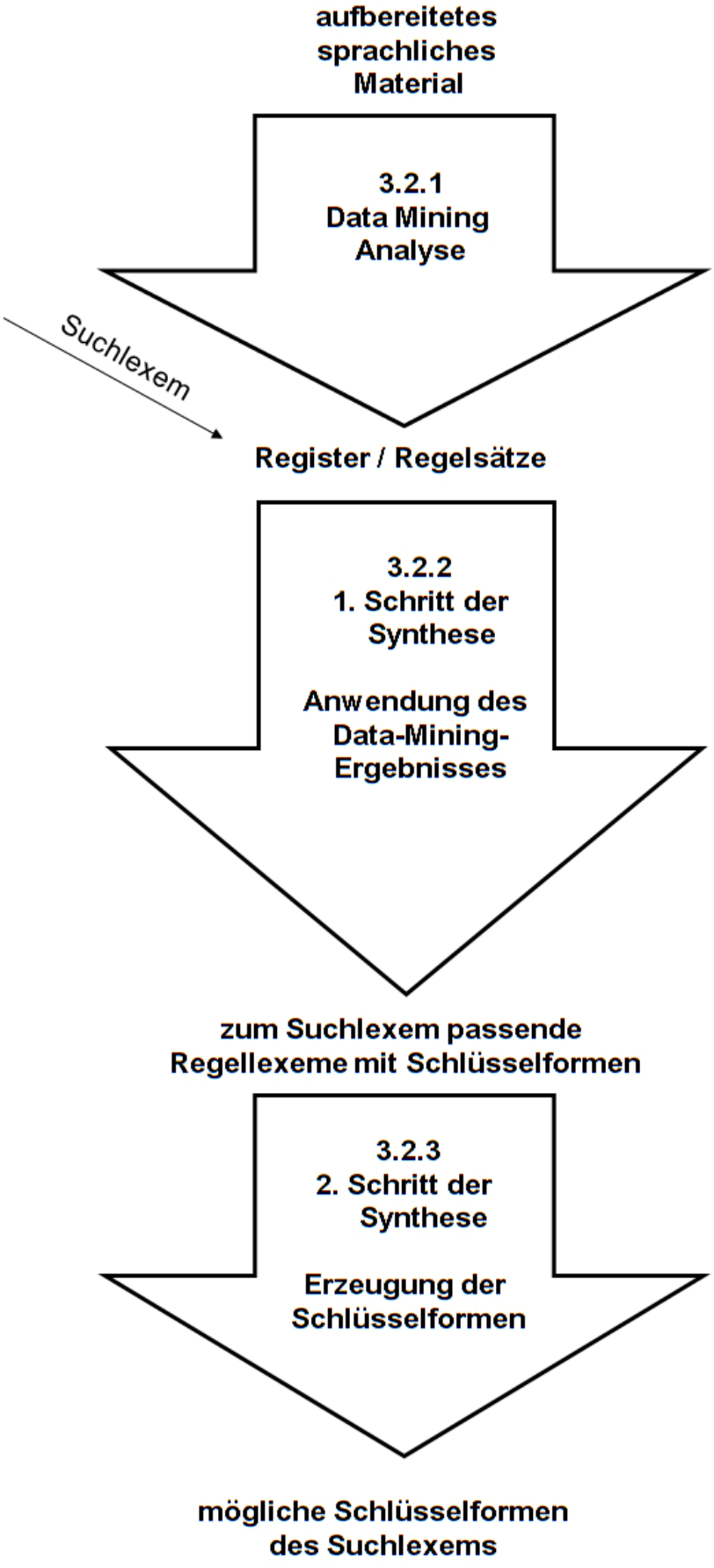

Abbildung 3.01: Verarbeitungsprozess

3.1.1 Analytischer Teilbereich: Ermittlung morphologisch homogener Cluster

In diesem Forschungsbereich werden morphologische Analogieregeln auf der Basis der rückläufigen Ähnlichkeit (Ausgangsgleichheit) von Präsensinfinitiven als den lexikographischen Grundformen konstruiert. Die Datenanalyse erfolgt mittels Clusteranalyse, einer Methode des Data Mining (vgl. 2.2).
Die hierbei entstehenden verdichteten sprachabhängigen Register bestehen aus morphologisch homogenen Clustern, die rückläufig ähnliche, morphologisch analoge Vertreter der untersuchten Wortart enthalten. Diese Register erlauben sprachdidaktisch verwendbare, quantitative und strukturelle Aussagen über flexionsmorphologische Systeme (vgl. Holl, 2003, 107-119).
Die Datenanalyse, Aufbereitung und Auswertung wurden bereits für folgende Verbalsysteme erfolgreich durchgeführt:

- Latein (1988)
- Katalanisch (1988)
- Portugiesisch (1988)
- Rumänisch (1988)
- Italienisch (1988, 2002)
- Spanisch (1988, 2002)
- Französisch (1988, 2002, 2003)
- Deutsch (2002, 2004)
- Russisch (2004)
- Neugriechisch (2006)
- Altgriechisch (2006)
- Englisch (2002, 2007)
- Schwedisch (2002, 2007)
- Kroatisch (2008)
- das schwedische Substantiv (2007)

Die so gewonnenen Register bilden für die in 3.1.2 und 3.1.3 dargelegten synthetisch-generativen Teilbereiche die Ausgangsbasis.

3.1.2 Erster Schritt des synthetisch-generativen Teilbereichs: Auffindung passender Regellexeme

Auf der Basis einheitlicher Regel-Konventionen (vgl. 3.4) für die Register wird mittels der in der Informatik zur Verfügung stehenden Funktionselemente ein Algorithmus modelliert (erste Gedanken zu dieser Art der Formalisierung finden sich in Holl, 1988, 184), welcher zu einem Suchlexem die entsprechenden Regellexeme eines Registers findet.
Um Korrektheit und Sprachunabhängigkeit zu überprüfen, wird dieser Algorithmus in einer ersten Version ("SMIRT Ver. 1.0")[6] realisiert.

[6] Der Name "SMIRT" ist abgeleitet von der englischen Wortkreation "Smirting", welche aus den Worten "Smoking and Flirting" gebildet wurde. Smirting bezeichnet das Flirten beim Tabakkonsum vor Gebäuden wie Büros oder Restaurants, in denen ein Rauchverbot gilt. Der Algorithmus wie auch die englische Wortkreation sind zu einer Zeit entstanden, als zunehmend in Europa ein generelles Rauchverbot eingeführt wurde. Die gleiche Temporalität und die Tatsache, dass auch das Wort "Smirting" zu den natürlichsprachlichen Verbalformen zählt, mit welchen der Algorithmus arbeitet, haben zu dieser Namensgebung geführt. Zur Behandlung von Neologismen siehe auch 5.

3.1.3 Zweiter Schritt des synthetisch-generativen Teilbereichs: Erzeugung der Schlüsselformen des Suchlexems

In diesem Forschungsbereich sollen zukünftig die morphologischen Eigenschaften (d.h. hier das Bildeprinzip der Schlüsselformen) der durch den Algorithmus ermittelten Regellexeme auf das Suchlexem übertragen werden. Im Falle des deutschen Suchlexems *schreiben* liefert der Algorithmus aus 3.1.2 das Regellexem *reiben* mit dessen Schlüsselformen *rieb* und *gerieben* als Ergebnismenge. Eine analogische Übertragung auf das Suchlexem *schreiben* liefert die Wortformen *schrieb* und *geschrieben*.

3.2 Beschreibung des analytischen Verarbeitungsschritts

Vor der Entwicklung eines Algorithmus ist es erforderlich, den Aufbau des Inputs zu analysieren, d.h. sich den vorgelagerten analytischen Verarbeitungsprozess sowie dessen Teilergebnisse näher anzuschauen. Hierzu wird die Struktur der Lexemregister analysiert (3.2.1) und das Phasenmodell des Data-Mining-Prozesses vorgestellt (3.2.2).

3.2.1 Struktur der Lexemregister

Im analytischen Verarbeitungsschritt bilden aufbereitete Daten eines Verbalsystems den Input der Datenanalyse. Der Output besteht aus Registern morphologischer Cluster, welche die nachfolgende Tabelle zusammenfassend illustriert.

Homogene Cluster - homC	**Inhomogene Cluster (basic cluster - basC)**
1) homC aus verschiedenen Verben (Simplicia), von denen nur **ein** Repräsentant in der Kommentarspalte des Lexemregisters genannt ist (keine Detaillierung): Dies ist der Fall bei regelmäßig flektierenden Lexemen mit regelhaften Besonderheiten. Beispiel: homC *~sh, ~shed, ~shed* *(fish, fished, fished)* wegen 3. Person Singular auf *-es*.	basC aus verschiedenen Verben (Simplicia), von denen nur ein Repräsentant in der Kommentarspalte des Lexemregisters genannt ist (keine Detaillierung): Es gibt Ausnahmen, aber die Mehrheit sind regelmäßig flektierende Lexeme. Beispiel: *basC ~C, ~Ced, ~Ced* *(look, looked, looked)* Ausnahme: z.B. *send, sent, sent*
2) homC aus verschiedenen Verben (Simplicia), von denen **alle** Repräsentanten in der Kommentarspalte des Lexemregisters genannt sind (Detaillierung): Dies ist der Fall bei unregelmäßig flektierenden Lexemen. Beispiel: homC *~ling, ~lung, ~lung* *(cling, sling, fling)*	basC2: Ein Cluster, das im Bereich eines basic clusters enthalten ist, aber anders flektiert als das übergeordnete basic cluster. Von den verschiedenen Verben (Simplicia) ist nur ein Repräsentant in der Kommentarspalte des Lexemregisters genannt (keine Detaillierung): Es bestehen Ausnahmen, aber die Mehrheit sind regelmäßig flektierende Lexeme mit regelhaften Besonderheiten. Beispiel: basC2 *#CCVd, #CCVdded, #CCVdded* *(kid, kidded, kidded)* # markiert die Wortgrenze Ausnahme: z.B. *shed, shed, shed*
3) homC aus einem Verb (Simplex): Einelementiges Cluster, welches nicht als solches markiert wird, da trivial (keine Detaillierung): Beispiel: *send, sent, sent*	Weitere basC(3,…,n)-Cluster, welche sich im Bereich eines übergeordneten Clusters befinden, sind möglich.

Tabelle 3.01: Typisierung der homogenen und inhomogenen Cluster (vgl. Holl, 2007, 108-109,115)

Diese Art der Clusterbildung erfolgt im Allgemeinen nach einer hierarchischen Struktur, welche der nachfolgende selbstähnliche Strukturbaum illustriert.

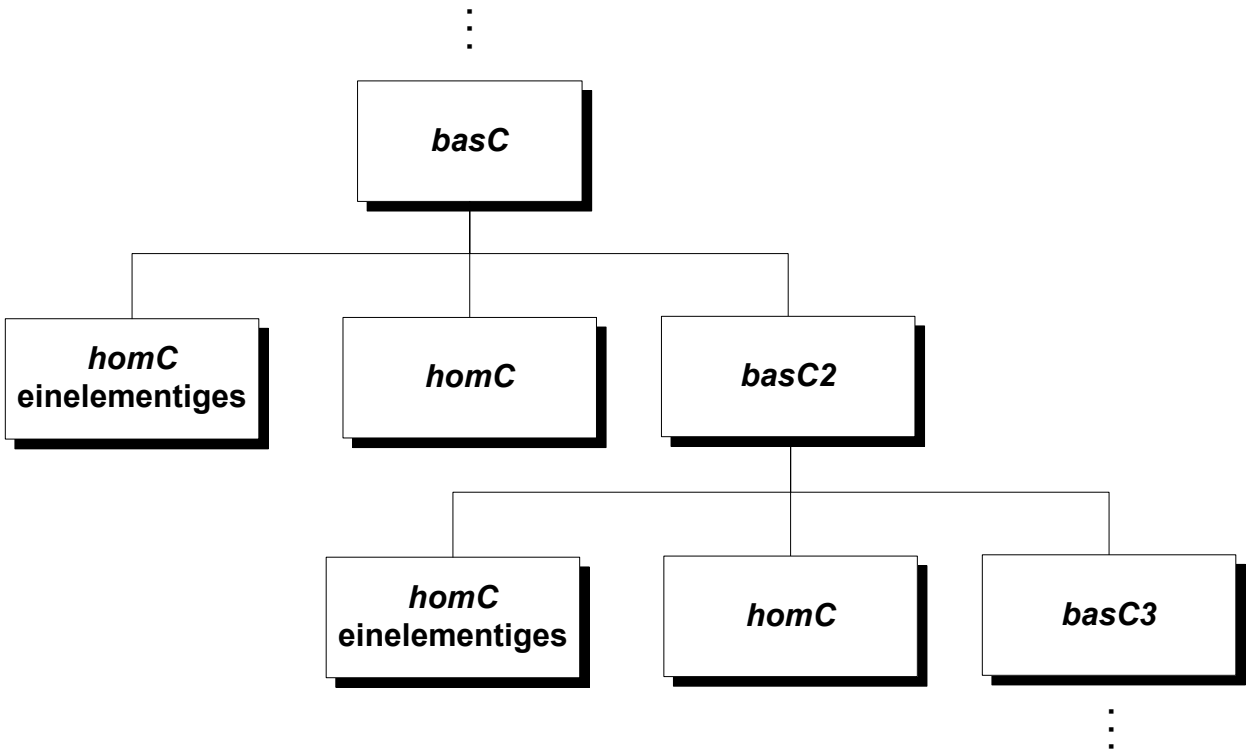

Abbildung 3.02: Allgemeiner Strukturbaum der Clusterbildung

Der in Abb. 3.02 beschriebene Baum findet sich beispielsweise in einem Ausschnitt der Struktur des englischen Verbalsystems wieder.

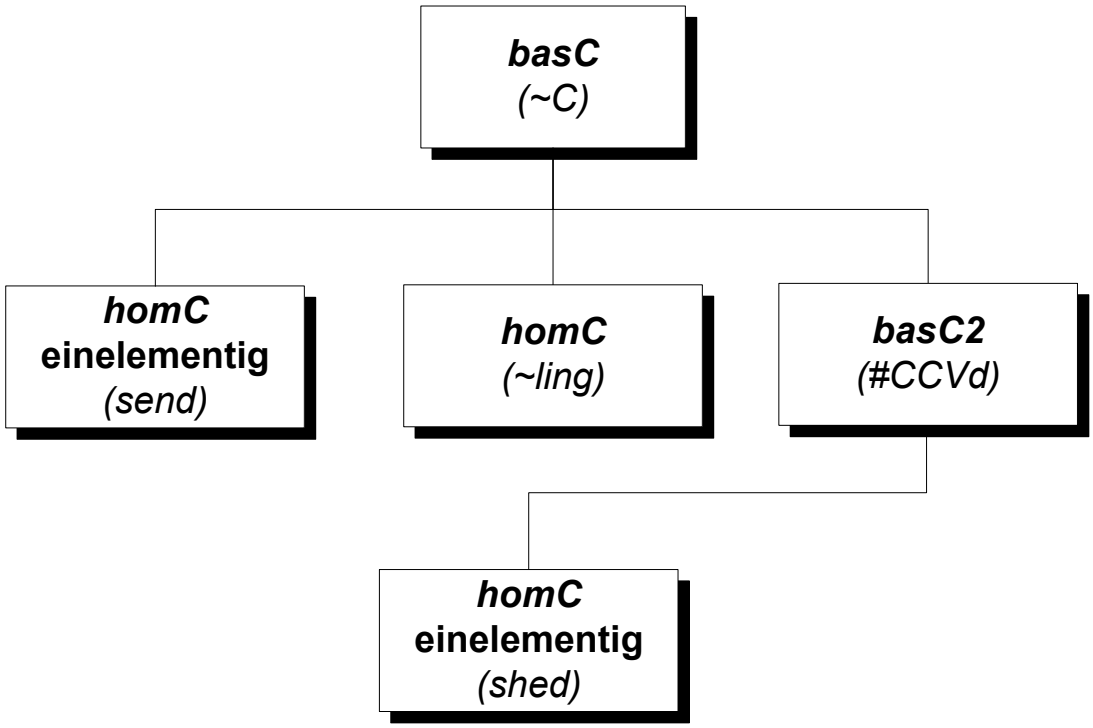

Abbildung 3.03: Beispiel eines Strukturbaums für englische Verben

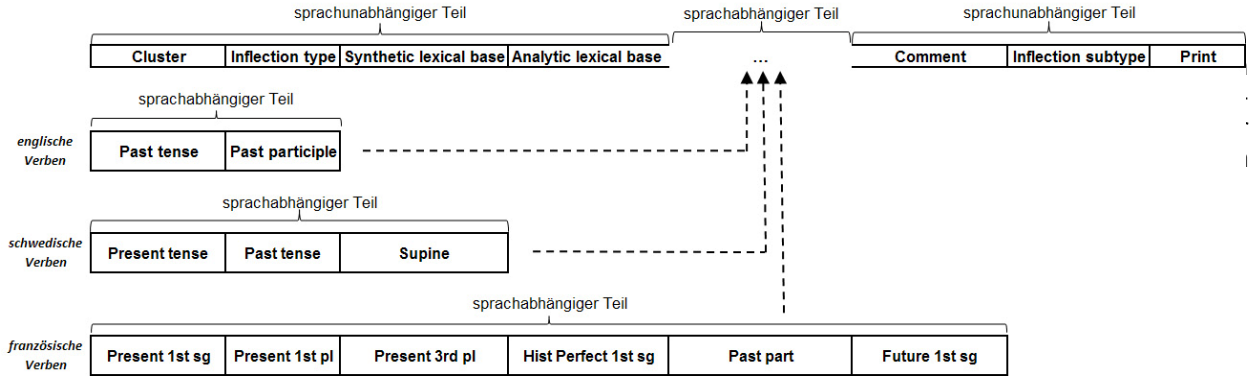

Abbildung 3.04: Datenstrukturen für verschiedene Verbalsysteme

Definition der Registerspalten:

- Sprachunabhängige Spalten
  - Cluster
    In dieser Spalte erfolgt die Kennzeichnung eines Clusters (siehe Tabelle 3.01). Hierbei kann es sich um homC oder basC handeln.
  - Inflection type
    Der Flexionstyp beschreibt die Beugung eines Lexems, im Fall von Verben den Konjugationstyp und im Fall von Substantiven den Deklinationstyp.
  - Synthetic lexical base (ähnlich dem Lexikoneintrag)
    Die lexikalische Grundform ist bei Substantiven der Nominativ Singular und bei Verben meist der Infinitiv Präsens Aktiv.
    Die synthetische lexikalische Grundform dient dem synthetisch-generativen Algorithmus als Vergleichsform mit dem Suchlexem. Sie entspricht den Einträgen der Spalte "analytic lexical base" reduziert um deren Kennzeichnungen (außer dem #-Zeichen für die Wortgrenze), da vom Benutzer eine derartige Kennzeichnung des Suchlexems nicht erwartet werden kann.
  - Analytic lexical base (aufbereiteter Lexikoneintrag)
    Die Einträge in konventionellen Lexika sind ggf. um folgende Kennzeichnungen ergänzt:
    - Unterstrich zur Kennzeichnung von Präfixen
      Beispiel (engl.): *de_pend, depended, depended*
    - Ziffern zur Kennzeichnung morphologischer Varianten mit Bedeutungsunterschied
      Beispiel (engl.):
      1shed, shedded, shedded ('ein Fahrzeug in einem Depot parken')
      2shed, shed, shed ('Blätter oder Früchte fallen zu Boden')
    - Griechische Buchstaben zur Kennzeichnung morphologischer Varianten ohne Bedeutungsunterschied
      Beispiel (engl.): *α+learn, learned, learned* ('lernen')
      β+learn, learnt, learnt ('lernen')
  - Comment
    In der Kommentarspalte werden die Lexeme eines Clusters mit ihren lexikalischen Grundformen in Abhängigkeit vom Clustertyp (wie in Tabelle 3.01 dargestellt) aufgezählt oder die Bedeutung von Flexionsvarianten angegeben.
  - Inflection subtype
    Beugungsfeinklassifizierung, die nicht zur Clusterbildung herangezogen wird.
  - Print
    In dieser Spalte werden alle ausgaberelevanten Zeilen mit "P" gekennzeichnet. Der synthetisch-generative Algorithmus wird später nur diese Zeilen zur Verarbeitung heranziehen.

- Sprachabhängige Spalten
  - Die Spalten in diesem Teil sind bestimmt durch die Schlüsselformen der jeweiligen Sprache-Wortart-Kombination (siehe Abbildung 3.04).

3.2.2 Die drei Phasen des Data-Mining-Prozesses

Pre-processing oder auch Datenvorverarbeitung ist die manuelle Aufbereitung und Strukturierung der Ausgangsdaten. Hierbei ist zu beachten, dass die Aufbereitung nicht zu einem Verlust informationsrelevanter Parameter führen darf. Die Datenvorverarbeitung ist der Schlüssel zu einer effizienten Muster- / Clustererkennung. In dieser Phase werden von einem Linguisten die Lexeme der zu untersuchenden Sprache-Wortart-Kombination in einer Excel-Tabelle erfasst und kategorisiert (vgl. Abbildung. 3.05).

	sprachunabhängiger Teil			sprachabhängiger Teil		sprachunabhängiger Teil		
Cluster	**Inflection type**	**Synthetic lexical base**	**Analytic lexical base**	**Past tense**	**Past participle**	**Comment**	**Inflection subtype**	**Print**
	i-u-u / 0	cling	cling	clung	clung			
	i-u-u / 0	fling	fling	flung	flung			
	i-u-u / 0	sling	sling	slung	slung			

Abbildung 3.05: Beispielergebnis einer Pre-processing-Phase (vgl. Holl, Maroldo, Urban, 2007, 77)

Processing
Der Data-Mining-Algorithmus analysiert die im Pre-processing aufbereiteten Daten auf der Suche nach bestimmten Mustern / Clustern, d.h. es werden morphologisch analoge Cluster ermittelt und in das jeweilige Register eingetragen. Bezogen auf das Beispiel wird das homogene Cluster *~ling* mit den lexikalischen Grundformen (lexical bases) *cling*, *fling* und *sling* gefunden.

	sprachunabhängiger Teil			sprachabhängiger Teil		sprachunabhängiger Teil		
Cluster	**Inflection type**	**Synthetic lexical base**	**Analytic lexical base**	**Past tense**	**Past participle**	**Comment**	**Inflection subtype**	**Print**
ling	i-u-u / 0	cling	cling	clung	clung			
ling	i-u-u / 0	fling	fling	flung	flung			
ling	i-u-u / 0	sling	sling	slung	slung			

Abbildung 3.06: Beispielergebnis einer Processing-Phase (vgl. Holl, Maroldo, Urban, 2007, 115)

Post-processing meint die manuelle Nachbearbeitung und Aufbereitung der durch Data Mining gewonnenen Register. Für das untersuchte Verbalsystem bedeutet dies:

<u>Erster Schritt der manuellen Nachbearbeitung:</u>

Erzeugung einer übersichtlichen Druckaufbereitung

- Einführung einer neuen Zeile für das Cluster mit entsprechender Benennung (im Beispiel homC)
- Zusammenfassung der sich im Cluster befindlichen lexikalischen Grundformen (lexical bases) im Kommentarfeld (Comment):
  vollständige Aufzählung bei unregelmäßigen Lexemen, Beispiele bei regelmäßigen Lexemen (ggf. mit regelhaften Besonderheiten)
- Kennzeichnung von unter verschiedenen linguistischen Perspektiven relevanten Regellexemen mit “P“ in der Printspalte

	sprachunabhängiger Teil			sprachabhängiger Teil		sprachunabhängiger Teil		
Cluster	**Inflection type**	**Synthetic lexical base**	**Analytic lexical base**	**Past tense**	**Past participle**	**Comment**	**Inflection subtype**	**Print**
homC	i-u-u / 0	ling	~ling	~lung	~lung	cling,sling,fling		P
ling	i-u-u / 0	cling	cling	clung	clung			
ling	i-u-u / 0	fling	fling	flung	flung			
ling	i-u-u / 0	sling	sling	slung	slung			

Abbildung 3.07: Beispielergebnis einer Post-processing-Phase, erster Schritt (vgl. Holl, Maroldo, Urban, 2007, 106)

Zweiter Schritt der manuellen Nachbearbeitung:

Aufbereitung für den synthetisch-generativen Algorithmus

- Elimination nicht mit “P“ gekennzeichneter Regellexeme

	sprachunabhängiger Teil			sprachabhängiger Teil		sprachunabhängiger Teil		
Cluster	**Inflection type**	**Synthetic lexical base**	**Analytic lexical base**	**Past tense**	**Past participle**	**Comment**	**Inflection subtype**	**Print**
homC	i-u-u / 0	ling	~ling	~lung	~lung	cling,sling,fling		P

Abbildung 3.08: Beispielergebnis einer Post-processing-Phase, zweiter Schritt (vgl. Holl, Maroldo, Urban, 2007, 106)

Das Ergebnis des Post-processing bildet einen Teil des Inputs für den ersten Schritt der Synthese. Einen weiteren Teil bildet ein zufällig gewähltes Suchlexem.

3.3 Konzept des synthetisch-generativen Algorithmus

3.3.1 erklärt die Grundidee des Algorithmus anhand einer Suchbaumhierarchie mit verschiedenen Clustertypen. 3.3.2 spezifiziert die Eingangsgrößen (Input) für den Algorithmus. 3.3.3 spezifiziert das Suchlexem. 3.3.4 erläutert die Funktionsweise des Algorithmus. 3.3.5, 3.3.6 und 3.3.7 erklären die Funktionen “total length compare“, “prefix cut“ und “longest match“.

3.3.1 Grundidee des Algorithmus

Der Algorithmus soll zu einem beliebigen Suchlexem das ihm am besten entsprechende Regellexem (oder mehrere solche) aus dem Register liefern unter Beachtung von Flexionstyp und Lexemausgang. Dazu stellt 3.3.1.1 den verwendeten Suchbaum vor. 3.3.1.2 und 3.3.1.3 erklären die Bedeutung von Clustern mit und ohne Wortbegrenzung bzw. mit und ohne Jokerzeichen. 3.3.1.4 kommentiert den in 3.3.1.1 beschriebenen Suchbaum. 3.3.1.5 illustriert Testfälle des Algorithmus.

3.3.1.1 Suchbaum

Der Start des Algorithmus wird durch die Eingabe eines zufälligen Suchlexems initiiert. Der Endpunkt des Algorithmus ist erreicht, wenn zum eingegebenen Suchlexem einelementige bzw. mehrelementige Cluster mit rückläufig gleichem Ausgang gefunden sind oder die Suche erfolglos bleibt und die “leere Menge“ zurückgegeben wird.
Die Suchlexeme werden ausschließlich mit den Einträgen in der Tabellenspalte “Synthetic lexical base“ verglichen. Da dort keine Präfixmarkierungen vorhanden sind, muss der Algorithmus, um die Korrektheit des Ergebnisses sicherzustellen, befähigt sein, ein morphologisch präfigiertes Suchlexem mit Präfixmarkierung um dieses Präfix zu reduzieren (3.3.6 prefix cut). Ziel des Algorithmus ist es, die Cluster zu finden, welche mit dem Suchlexem die längste rückläufige Übereinstimmung besitzen (3.3.7 longest matching von rechts) (vgl. Holl, 2006, 51). Dies kann, wie die nachfolgende Abbildung deutlich macht, in unterschiedlichen Fällen erreicht werden.[7]

[7] Hier und im Folgenden wird die Zuordnung eines Suchlexems zu Regellexemen mit einem einheitlichen Muster beschrieben:

Suchlexem → Spalte “Synthetic lexical base“ : Spalte “Analytic lexical base“, Schlüsselformen

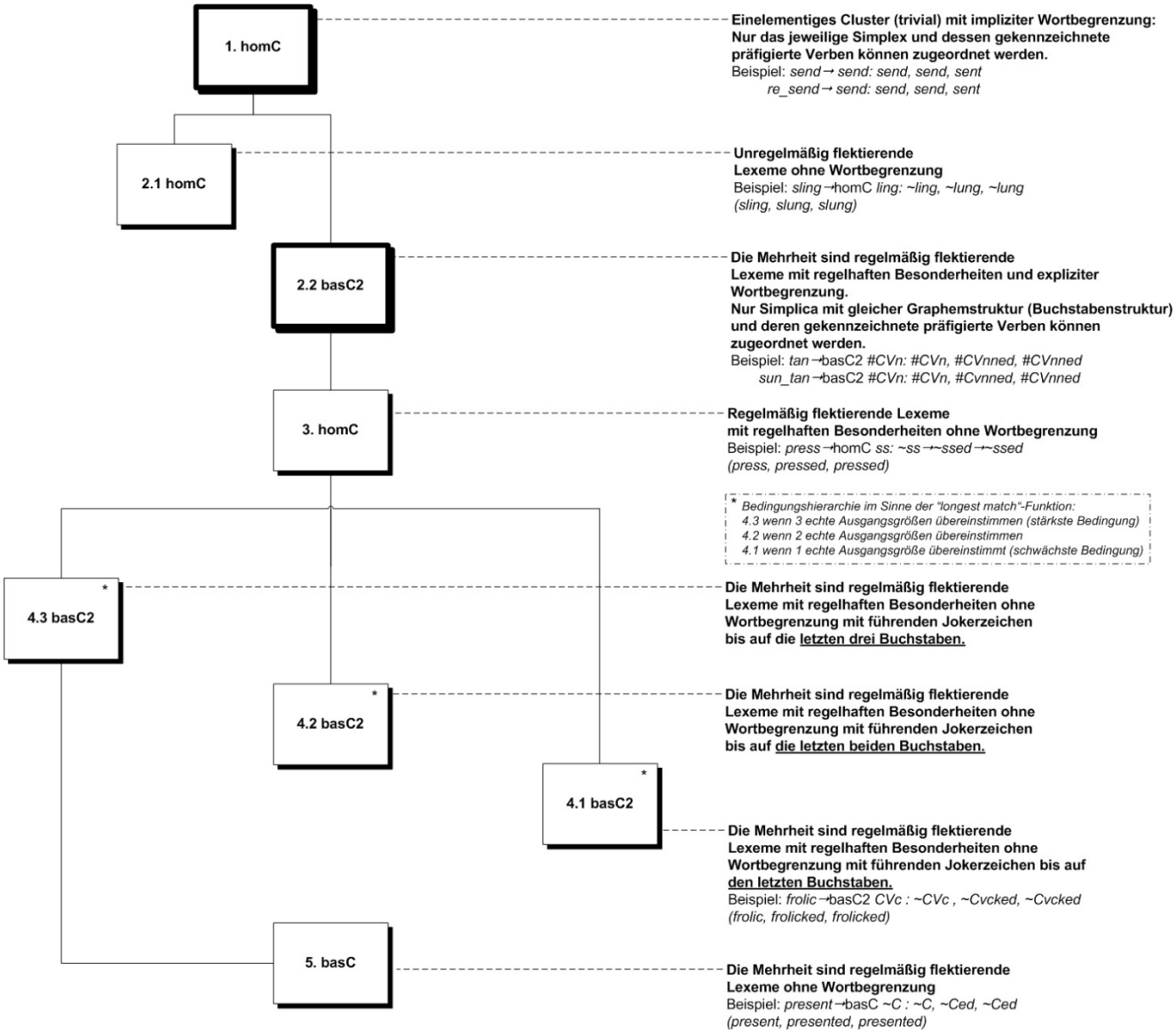

Abbildung 3.09: Suchbaumhierarchie für den Algorithmus

Ausgangspunkt für die Entwicklung einer Regel- / Suchbaumhierarchie ist der Strukturbaum der Clusterbildung (Abb. 3.03). Der Strukturbaum wird in umgekehrter Reihenfolge durchlaufen, d.h. während sich der Strukturbaum von allgemeinen zu speziellen Clustern orientiert, orientiert sich die Suchbaumhierarchie von speziellen zu allgemeinen Clustern, um die längste rückläufige Übereinstimmung zwischen Suchlexem und Regellexemen zu finden. Die Eigenschaften der verschiedenen Cluster der Suchbaumhierarchie werden in Tabelle 3.02 tabellarisch dargestellt.

Fälle nach Abb. 3.09	Wortbegrenzung (3.3.1.2)	Jokerzeichen (3.3.1.3)	Elemente im Cluster	Vollstring (ohne ~) oder Teilstring (mit ~)
1 homC	implizit	-	ein Simplex	Voll
2.1 homC	-	-	>1	Teil
2.2 basC2	explizit	ja	>1	Voll
3 homC	-	-	>1	Teil
4.3 basC2	-	ja	>1	Teil
4.2 basC2	-	ja	>1	Teil
4.1 basC2	-	ja	>1	Teil
5 basC	-	ja	>1	Teil

Tabelle 3.02: Suchbaumhierarchie für den Algorithmus

3.3.1.2 Erste Fallunterscheidung: Cluster mit und ohne Wortbegrenzung

Hierbei sind zwei grundsätzliche Fälle zu unterscheiden: Cluster mit Wortbegrenzung und Cluster ohne Wortbegrenzung. Eine Wortbegrenzung entspricht der exakten zulässigen Länge (Anzahl Buchstaben), die ein Wort besitzen muss (kürzere Wortlängen sind nicht zulässig). Die Nummerierungen in den Beispielen beziehen sich auf Tab. 3.02.
Cluster mit einer Wortbegrenzung sind entweder durch ein voranstehendes #-Zeichen oder gar nicht gekennzeichnet.

Beispiel zu 1 (implizite Wortbegrenzung):
send→send: send, send, sent
(Wortlänge des Simplex: 4 Buchstaben)
re_send→send: send, send, sent
(Wortlänge des Simplex: 4 Buchstaben)
also *re_send, re_sent, re_sent*
Derartige einelementige Cluster (bestehend aus einem Simplex und seinen präfigierten Verben) werden in den Registern nicht eigens als “homC“ markiert.

Beispiel zu 2.2 (explizite Wortbegrenzung):
tan→basC2 *#CVn: #CVn, #CVnned, #CVnned*
(Wortlänge des Simplex: 3 Buchstaben)
also *tan, tanned, tanned*
sun_tan→basC2 *#CVn: #CVn, #CVnned, #CVnned*
(Wortlänge des Simplex: 3 Buchstaben)
also *sun_tan, sun_tanned, sun_tanned*

Cluster ohne Wortbegrenzung sind mit einer voranstehenden Tilde (~) gekennzeichnet.

Beispiel zu 3: *press*→homC *ss: ~ss, ~ssed, ~ssed*
also *press, pressed, pressed*
un_dress / *undress*→homC *ss: ~ss, ~ssed, ~ssed*
also *undress, undressed, undressed*

Beispiel zu 5: *present*→basC *C: ~C, ~Ced, ~Ced*
also *present, presented, presented*
look→basC *C: ~C, ~Ced, ~Ced*
also *look, looked, looked*

3.3.1.3 Zweite Fallunterscheidung: Cluster mit und ohne Jokerzeichen

Um den Vergleich eines Suchlexems mit “Synthetic lexical base“-Einträgen zu ermöglichen, die die Jokerzeichen “C“ und “V“ enthalten, muss die Buchstabenfolge des Suchlexems teilweise in Jokerzeichen umgesetzt werden. Das Jokerzeichen “C“ repräsentiert die Konsonanten, “V“ die Vokale der jeweiligen Sprache.
Hierzu wird das Suchlexem auf dessen Buchstabenzusammensetzung in Leserichtung (europäisch) folgendermaßen überprüft. Handelt es sich bei einem Buchstaben um einen Vokal, wird in den Suchalternativen stellvertretend ein “V“ eingetragen, und wenn es sich um einen Konsonanten handelt, ein “C“. Diese Suchalternativen werden in Abhängigkeit von einer Alphabet-Tabelle des untersuchten Verbalsystems gebildet.

Das ergibt die nachfolgenden unterschiedlichen Suchalternativen (Nummerierung erfolgt nach Abb. 3.09).

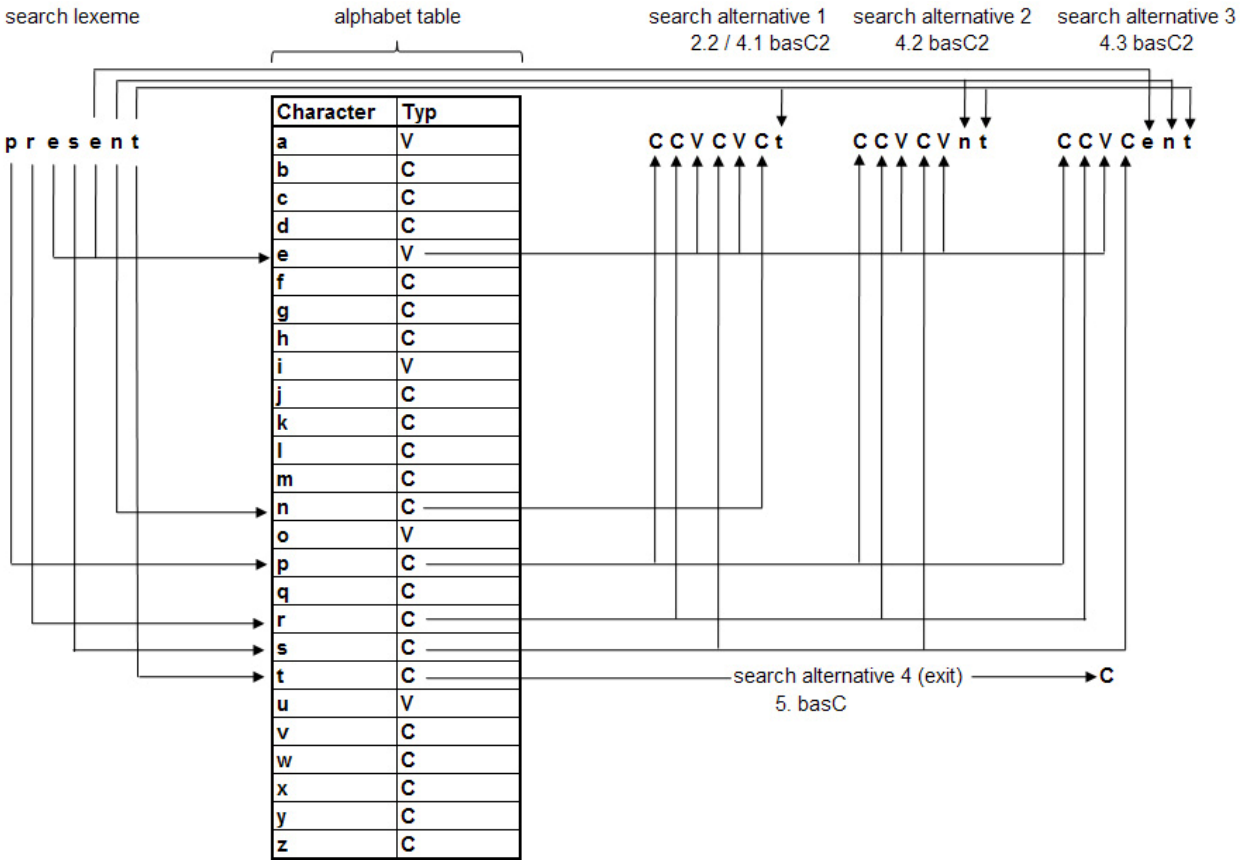

Abbildung 3.10: Generierung der Suchalternativen

Die Suchalterativen in Abb. 3.10 werden nur beim Aufruf der “longest match“-Funktion generiert. In Abhängigkeit von der Länge des an die Funktion übergebenen Suchlexems werden drei Fälle unterschieden.

1. Die Suchlexemlänge ist größer eins.
   → Generierung der Suchalternative 1

2. Die Suchlexemlänge ist größer zwei.
   → Generierung der Suchalternativen 1 und 2

3. Die Suchlexemlänge ist größer drei.
   → Generierung der Suchalternativen 1, 2 und 3

Hierbei wird in Leserichtung (europäisch) Buchstabe um Buchstabe mittels der Alphabet-Tabelle durch das jeweilige Jokerzeichen ersetzt. Die oben genannten Fallunterscheidungen stellen sicher, dass bei der Suchalternative 1 alle Buchstaben bis auf den letzten, bei Suchalternative 2 bis auf die letzten beiden und bei Suchalternative 3 bis auf die letzten drei Buchstaben ersetzt werden. Im Anschluss werden die Suchalternativen durch die echten Buchstaben des Suchlexems nach rechts aufgefüllt.

Die “longest match“-Funktion benötigt jedoch für den weiteren Funktionsverlauf zwei weitere Alternativen. Die Suchalternative (# search alternative 1) entspricht exakt der Suchalternative 1 mit vorangestelltem Hashzeichen, während die Suchalternative (search alternative 4 exit) der Jokerzeichendarstellung des letzten Buchstabens des Suchlexems entspricht.

3.3.1.4 Kommentar zum Suchbaum

1. Der Buchstabenfolge des Suchlexems entspricht ein genau gleich langes Cluster (total length compare) mit impliziter Wortbegrenzung.
   Beispiel: *send→send: send, send, sent*

2. – 5. Es gibt ein Cluster, das einem Ausgang des Suchlexems entspricht (longest match). Die Generierung der Suchalternativen der Fälle 2.2, 4.1, 4.2, 4.3 und 5 erfolgt, wie in Abbildung 3.10 illustriert.

2.1 Ein Ausgang des Suchlexems wird von einem unregelmäßig flektierenden homC repräsentiert. Dieses Cluster besitzt keine Wortbegrenzung. Beispiel: *sling→*homC *ling: ~ling, ~lung, ~lung*

2.2 Das Suchlexem gehört zu einem Cluster, in welchem alle Buchstaben bis auf den letzten durch Jokerzeichen (C / V) ersetzt wurden. Eine Wortbegrenzung ist hier durch das voranstehende #-Zeichen gekennzeichnet (# search alternative 1).
Beispiel: *tan→*basC2 *#CVn: #CVn, #CVnned, #CVnned*

3. Ein Ausgang des Suchlexems entspricht einem Cluster ohne Wortbegrenzung und ohne Jokerzeichen.
Beispiel: *press→*homC *ss: ~ss, ~ssed, ~ssed*

4.3 Ein Ausgang des Suchlexems, in welchem alle Buchstaben bis auf die letzten drei durch Jokerzeichen (C / V) ersetzt wurden, entspricht einem Cluster ohne Wortbegrenzung (search alternative 3).

4.2 Ein Ausgang des Suchlexems, in welchem alle Buchstaben bis auf die letzten beiden durch Jokerzeichen (C / V) ersetzt wurden, entspricht einem Cluster ohne Wortbegrenzung (search alternative 2).
Die Suchalternativen 2 und 3 werden nur sicherheitshalber eingeführt, obwohl dazu noch keine Beispiele vorliegen.

4.1 Ein Ausgang des Suchlexems, in welchem alle Buchstaben bis auf den letzten durch Jokerzeichen C / V) ersetzt wurden, entspricht einem Cluster ohne Wortbegrenzung (search alternative 1).
Beispiel: *frolic→*basC2 *CVc : ~CVc , ~CVcked, ~CVcked*

5. Wenn das Suchlexem weder einem homC noch einem basC2 entspricht, wird eine weitere Alternative generiert. Hierbei wird der letzte Buchstabe des Suchlexems durch ein Jokerzeichen (C / V) ersetzt. Dieser allgemeine Ausgang entspricht im Fall der englischen Verben einem basC ohne Wortbegrenzung (search alternative 4).
Beispiel: *present→*basC *C : ~C, ~Ced, ~Ced*

Diese Art der Suchlexem-Konvertierung in Jokerzeichen-Kombinationen ist derzeitig lediglich für die englischen und deutschen Verbregister nötig. Es besteht jedoch die Möglichkeit, diese Form der Cluster-Benennung auch bei anderen Verbalsystemen einzusetzen. Daher müssen auch diese Fälle in vollem Umfang durch den Algorithmus abgebildet werden. Die nachfolgenden Testfälle sollen dies verdeutlichen und einen kurzen Überblick über die Vielzahl möglicher Suchresultate illustrieren.

3.3.1.5 Testfälle

		1	2.1 / 3	2.2	4.1-4.3 / 5
	Clustertypen	homC	homC / basC / basC2	basC / basC2	homC / basC / basC2
	Regellexeme / Suchlexeme	Vollstring	Teil- / Vollstring	Vollstring	Teilstring
		einelementige Cluster	mehrelementige Cluster	mehrelementige Cluster	mehrelementige Cluster
		ohne Jokerzeichen	ohne Jokerzeichen	mit Jokerzeichen	mit Jokerzeichen
		Wortbegrenzung	ohne Wortbegrenzung	Wortbegrenzung	ohne Wortbegrenzung
1	*send* (senden)	*send*	-	-	-
	resend (wieder senden)	-	-	-	*~C* (falsch)*
	re_send (wieder senden)	*send*	-	-	-
	grave (gravieren)	*1grave / 2grave*	-	-	-
	engrave (eingravieren)	*en_2grave*	-	-	-
	en_grave (eingravieren)	*en_2grave*	-	-	-
2.1					
	sling (schleudern)	-	*~ling*	-	-
2.2					
	tan (bräunen)	-	-	*#CVn*	-
	suntan (sonnenbräunen)	-	-	-	*~C* (falsch)*
	sun_tan (sonnenbräunen)	-	-	*#CVn*	-
3					
	press (pressen)	-	*~ss*	-	-
	compress (komprimieren)	-	*~ss*	-	-
	com_press (komprimieren)	-	*~ss*	-	-
4.1					
	frolic (scherzen)	-	-	-	*~CVc*
5					
	present (präsentieren)	-	-	-	*~C*
	represent (repräsentieren)	-	-	-	*~C*
	re_present (repräsentieren)	-	-	-	*~C*

* **nicht vom Algorithmus her falsch, sondern linguistisch (siehe 3.3.4)**

Tabelle 3.03: Testfälle für den ersten Schritt der Synthese (Ausgabe: Regellexeme)

In der linken Spalte stehen die Suchlexeme, rechts davon die gefundenen Regellexeme, klassifiziert nach Tabelle 3.02.

Varianten:
Die Ausgabe von mehreren Regellexemen als Suchresultat ist möglich, wie der Fall *grave* zeigt. Wie in 3.2.1 beschrieben, gibt es Lexeme unterschiedlicher Flexion mit gleicher lexikalischer Grundform sowohl mit als auch ohne Bedeutungsunterschied. Diese Varianten sind jeweils als eigenständige Regellexeme in den Registern vorhanden (unterschieden durch Ziffern oder kleine griechische Buchstaben) und werden daher in ihrer Gesamtheit als Resultat angezeigt, so dass der Benutzer die Möglichkeit besitzt, diese ggf. in ihrer Semantik zu unterscheiden.
Desweiteren können sich präfigierte Verben als Regellexeme in den Registern finden. Bei präfigierten Suchlexemen ist die Eingabe mit Präfixmarker empfohlen, da es sonst zu falschen Suchresultaten, wie in den Fällen *resend* und *suntan*, kommen kann. Die Präfixkennzeichnung für den Algorithmus wird in Form eines Unterstrichs nach dem Präfix realisiert.

Präfigierte Lexeme:
Erster Fall: als Regellexeme im Register nicht vorhanden.
In den beiden Beispielen (*resend / suntan)* befindet sich das präfigierte Verb selbst nicht als Regellexem im Register, sondern lediglich das Simplex in orthographischer Form für *send* bzw. Jokerzeichenform für *tan,* welches hier das richtige Suchresultat wäre. Ohne eine Kennzeichnung der Präfixe findet der Algorithmus kein homC mit Wortbegrenzung und wird daher versuchen, die Suchlexeme soweit zu reduzieren, bis er ein homC, basC2 oder basC mit rückläufiger Übereinstimmung findet. In den beiden Beispielen findet der Algorithmus fälschlicherweise das basC *~C* für regelmäßig flektierende Lexeme. Erfolgt eine Kennzeichnung des Präfixes durch einen nachgestellten Unterstrich, dann findet der Algorithmus (wie in Tabelle 3.03 ersichtlich) richtigerweise jeweils das dazugehörige homogene Cluster.

Präfigierte Lexeme:
Zweiter Fall: als Regellexeme im Register vorhanden.
Solche Lexeme werden im Register immer richtig gefunden, egal ob das Suchlexem mit oder ohne Präfixmarkierung eingegeben wird.

Die nachfolgende Abbildung ist eine kompakte Darstellung der wesentlichen Testfälle aus Tabelle 3.03 und kennzeichnet die Bereiche, welche bei morphologisch präfigierten Lexemen ohne Präfixmarkierung zu Falschresultaten führen können.

Clustertypen	homC	homC / basC / basC2	basC / basC2	homC / basC / basC2
Regellexeme / Suchlexeme	Vollstring	Teil- / Vollstring	Vollstring	Teilstring
	einelementige Cluster	mehrelementige Cluster	mehrelementige Cluster	mehrelementige Cluster
	ohne Jokerzeichen	ohne Jokerzeichen	mit Jokerzeichen	mit Jokerzeichen
	Wortbegrenzung	ohne Wortbegrenzung	Wortbegrenzung	ohne Wortbegrenzung
Suchwort morphologisch nicht präfigiert	*send*	*press*	*tan*	*present*
Suchwort morphologisch präfigiert, formal nicht präfigiert (ohne Präfixmarker)	*engrave*	*compress*	-	*represent resend, suntan /* "mögliche Falschresultate"
Suchwort morphologisch präfigiert, formal präfigiert (mit Präfixmarker)	*re_send en_grave*	*com_press*	*sun_tan*	*re_present*

Tabelle 3.04: Kompakte Darstellung der Testfälle (Eingabe: Suchlexeme)

Diese Testfälle motivieren die nachfolgende formale Darstellung (Tabelle 3.05). In dieser Tabelle werden sämtliche Suchlexemvarianten den möglichen Resultaten bestehend aus Regellexemen gegenübergestellt. Dabei können mehrere Suchlexeme nicht gleichzeitig, sondern nur nacheinander eingegeben werden. Desweiteren sind (wie beschrieben) Falschresultate möglich.
Die Suchbaumhierarchie (Abb. 3.09) und diese formale Darstellung sind der Ausgangspunkt zur Entwicklung des Algorithmus und dienen später zu dessen Korrektheitsprüfung.

Clustertypen	homC	homC / basC / basC2	basC / basC2	homC / basC / basC2
Regellexeme / Suchlexeme	Vollstring	Teil- / Vollstring	Vollstring	Teilstring
	einelementige Cluster	mehrelementige Cluster	mehrelementige Cluster	mehrelementige Cluster
	ohne Jokerzeichen	ohne Jokerzeichen	mit Jokerzeichen	mit Jokerzeichen
	Wortbegrenzung	ohne Wortbegrenzung	Wortbegrenzung	ohne Wortbegrenzung
Suchwort morphologisch nicht präfigiert	☑	☑	☑	☑
Suchwort morphologisch präfigiert, formal nicht präfigiert (ohne Präfixmarker)	☑	☑	-	☑
Suchwort morphologisch präfigiert, formal präfigiert (mit Präfixmarker)	☑	☑	☑	☑

Tabelle 3.05: Formale Darstellung der vorkommenden Testfälle ohne Bewertung des Resultats

3.3.2 Definition des Inputs

Der Input des Algorithmus setzt sich (siehe Abb. 3.01) aus zwei Teilen zusammen, dem Register und einem zufällig gewählten Suchlexem (und ggf. einer Alphabettabelle).

3.3.3 Definition des Suchlexems

Für das Suchlexem[8] sind nachfolgende Varianten zulässig. Suchlexeme sind mit und ohne Präfixe erlaubt. Präfixe müssen, wie in 3.3.1.5 beschrieben, bei der Eingabe durch einen Unterstrich nach dem Präfix gekennzeichnet werden. Die Markierung von mehreren Präfixen bei einem Suchlexem ist möglich (zur Behandlung von präfigierten Suchlexemen siehe 3.3.6 und 3.3.7).

3.3.4 Funktion des Algorithmus

Das akzeptierte Suchlexem wird an eine Do-While-Schleife übergeben. Diese übergibt im ersten Schritt das komplette Suchlexem, beim wiederholten Aufruf das veränderte (um ein Präfix reduzierte) Suchlexem, an die "total length compare"-Funktion (3.3.5). Diese überprüft, ob in der Spalte "Synthetic lexical base" des Registers ein dem Suchlexem identisches Regellexem existiert. Wenn dies der Fall ist, terminiert die Schleife. Andernfalls wird überprüft, ob das Suchlexem einen weiteren Präfixmarker besitzt.
Besitzt das Suchlexem keinen Präfixmarker, terminiert die Schleife ebenfalls. Andernfalls wird das Suchlexem an die Funktion "prefix cut" (3.3.6) übergeben, welche das Suchlexem um ihr erstes Präfix reduziert, und die Schleife beginnt von vorn.

Wenn die Schleife terminiert, bedeutet dies entweder:

- Es befindet sich das ganze oder "reduzierte" Suchlexem im Register
- Das "reduzierte" Suchlexem befindet sich nicht im Register und besitzt auch keinen Präfixmarker.

Wenn sich das Suchlexem im Register befindet, werden die zugehörigen Regellexeme ausgegeben, andernfalls wird das "reduzierte" Suchlexem an die "longest match"-Funktion (3.3.7) übergeben.
Diese generiert die Suchalternativen aus 3.3.1.3 und überprüft, ob sich eine davon im Register findet. Die Suchalternativen und das Suchlexem werden von links buchstabenweise verkürzt, bis eine Übereinstimmung gefunden wird.

Werden beispielsweise die Regellexeme für das Verb *reengrave* gesucht, so sind vier Eingabevarianten zulässig und liefern folgende Regellexeme mit ihren Schlüsselformen als Ergebnismenge:

- *reengrave* → basC2 *Ce* : *~Ce, ~Ced, ~Ced*
- *re_engrave → engrave* : *en_2grave, engraved, engraved*
- *re_en_grave → engrave* : *en_2grave, engraved, engraved*
- *reen_grave → grave* : *1grave, graved, graved*
  α +2grave, graved, graved
  β +2grave, graved, graven

Nur das zweite und dritte Resultat ist linguistisch richtig. Das erste Ergebnis wird von "longest match" gefunden, die restlichen von "total length compare" nach "prefix cut".

[8] In dieser Untersuchung steht das Suchlexem (search lexeme) entweder für ein Eingabewort in seiner ursprünglichen Form, d.h. wie es durch einen Benutzer eingegeben wird, oder für ein Eingabewort, welches durch den Algorithmus um sein Präfix reduziert wurde.

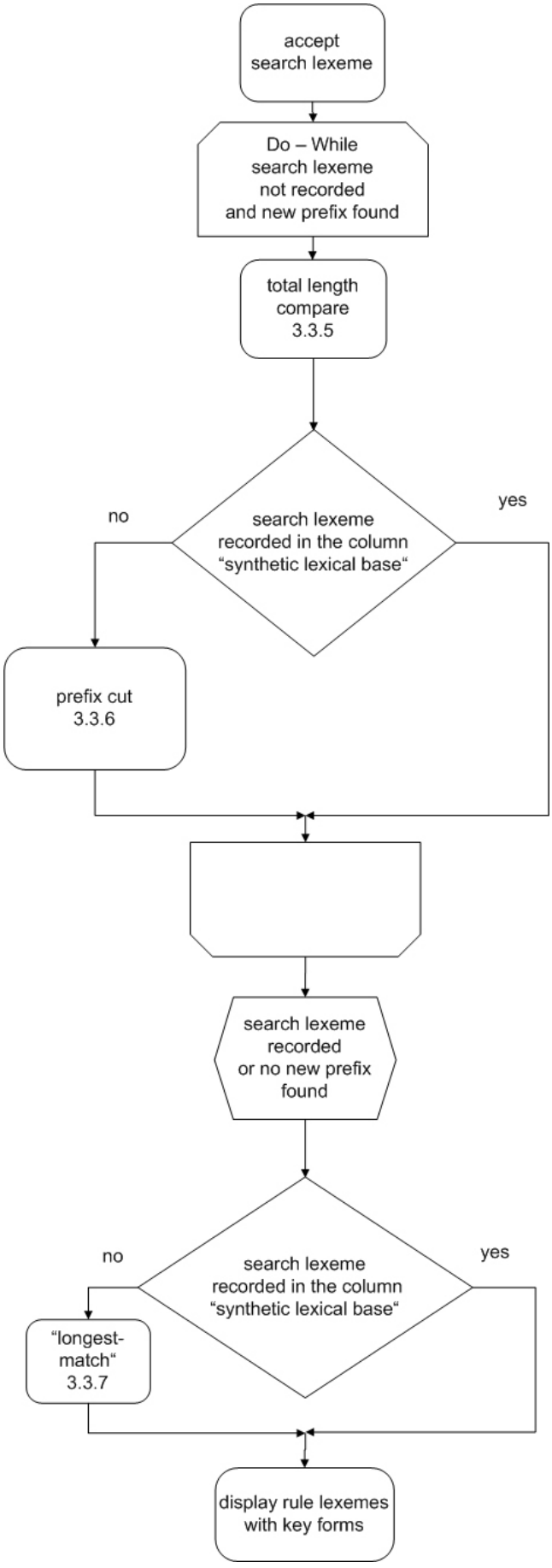

Abbildung 3.11: Ablaufplan des SMIRT-Algorithmus

3.3.5 Funktion “total length compare“

In der Vergleichsspalte “Synthetic lexical base“ existieren keine Präfixmarkierungen. Daher kann ein formal präfigiertes Suchlexem nur dann in voller Länge im Register gefunden werden, wenn im Suchlexem die Präfixmarkierungen entfernt werden. Das geschieht in dieser Funktion.
Das (ggf. um Präfixe reduzierte) Suchlexem ohne Präfixmarkierung bildet den Input der “total length compare“-Funktion. Diese Funktion sucht im Register in der Tabellenspalte “Synthetic lexical base“, ob sich das Suchlexem dort in voller Länge findet (auch mehrere gefundene Regellexeme sind möglich).

Beispiel für mehrere gefundene Regellexeme:

Wie in 3.2.1 beschrieben, handelt es sich im Fall *grave* um morphologische Varianten mit Bedeutungsunterschied.

grave → grave: 1grave, graved, graved (‘clean a ship's bottom‘)
α +2grave, graved, graved
β +2grave, graved, graven (‘engrave an inscription on a surface‘)

Wird ein Regellexem gefunden, ist es gleich dem (ggf. um Präfixe reduzierten) Suchlexem. Es handelt sich um den Fall 1 in Abb. 3.09 und Tab. 3.03:

- **Vollstring, einelementiges Cluster, ohne Jokerzeichen, Wortbegrenzung**
  Beispiel *send* mit seinen Schlüsselformen *send, sent, sent*

Andernfalls passiert nichts. In beiden Fällen bleibt das Suchlexem unverändert.

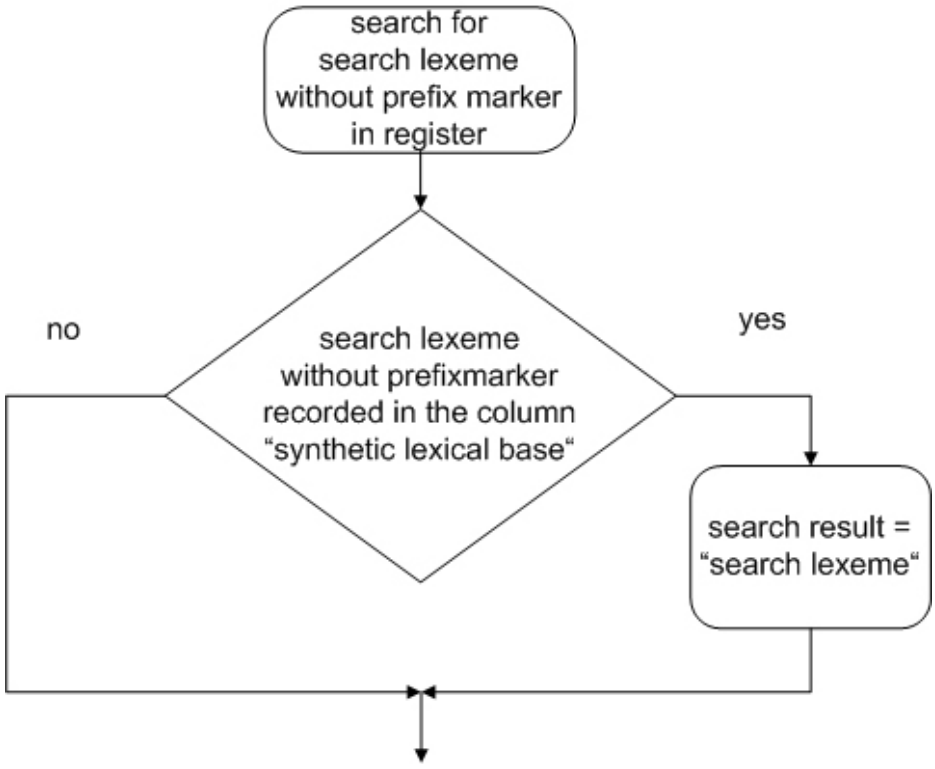

Abbildung 3.12: Ablaufplan der “total length compare“-Funktion

3.3.6 Funktion “prefix cut“

In der Funktion “prefix cut“ bildet beim ersten Aufruf das akzeptierte Suchlexem den Input, bei einem wiederholten Aufruf ein um ein Präfix reduziertes Suchlexem. Der Input wird an eine Do-While-Schleife übergeben. Diese übergibt den Input an eine Suchfunktion, welche Buchstabe für Buchstabe in Leserichtung (europäisch) nach einem Unterstrich im Suchlexem sucht. Wird ein Unterstrich gefunden, wird ein Flag gesetzt, das in der Bedingung “search lexeme not recorded and new prefix found“ (Abb. 3.11) abgefragt wird. Alle sich rechts vom Unter-strich befindlichen Buchstaben bilden das neue Suchlexem, und die Suche wird fortgesetzt.
Durch diese Vorgehensweise werden auch Mehrfach-Präfixe erkannt und reduziert, wie das nachfolgende Beispiel verdeutlicht:

- *re_send* reduziert auf *send* bildet das neue Suchlexem
- *re_en_grave*
  - nach dem ersten Durchlauf:
    reduziert auf *en_grave* bildet das neue Suchlexem
  - *en_grave* nach dem zweiten Durchlauf:
    reduziert auf *grave* bildet das neue Suchlexem

Solange die Suchfunktion keinen Unterstrich findet, überprüft sie Buchstabe für Buchstabe, bis sie ans Wortende gelangt. Dann terminiert die Schleife.

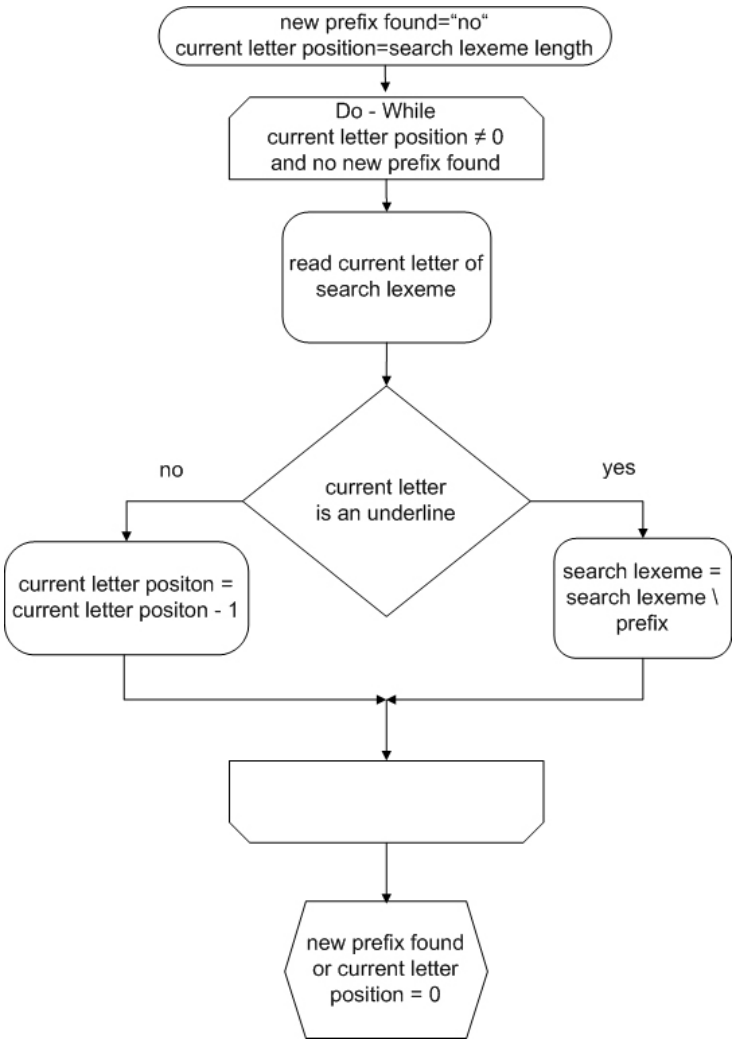

Abbildung 3.13: Ablaufplan der “prefix cut“-Funktion

3.3.7 Funktion “longest match“

Den Input der “longest match“-Funktion bildet das akzeptierte oder reduzierte Suchlexem.

3.3.7.1 Generierung der Suchalternativen (Abb. 3.14)

Im ersten Schritt werden zusätzlich zum Suchlexem weitere Suchalternativen generiert zum Vergleich mit den in den Registern vorzufindenden Vokal-Konsonant-Konstellationen, welche in 3.3.1.3 beschrieben wurden. Hierzu wird das Suchlexem auf dessen Buchstabenzusammensetzung folgendermaßen überprüft. Handelt es sich bei einem Buchstaben um einen Vokal, wird in der Suchalternative stellvertretend ein “V“ eingetragen, und wenn es sich um einen Konsonanten handelt, ein “C“. Diese Suchalternativen werden in Abhängigkeit von einer Alphabet-Tabelle des untersuchten Verbalsystems gebildet, welche dem Algorithmus zu Beginn zur Verfügung gestellt (importiert) werden muss (siehe Abbildung 3.10). Als Erstes wird die Länge des Suchlexems ermittelt. Diese legt die maximale Anzahl der Schleifendurchläufe fest und bestimmt die Anzahl der zu generierenden Suchalternativen. Suchalternativen werden erst ab einer Suchlexemlänge größer eins generiert. Andernfalls ist eine Suchalternative nicht nötig, da einbuchstabige Suchlexeme entweder durch das reduzierte Suchlexem selbst oder durch eine Bedingung am Ende der “longest match“-Funktion abgefangen werden (Suchalternative 4 exit). Ab einer Länge des Suchlexems von größer eins wird nur die Suchalternative 1, ab einer Länge größer zwei die Suchalternativen 1-2 und ab einer Länge von größer drei die Suchalternativen 1-3 generiert (vgl. Abb. 3.10).

Beispiel:
Suchlexem: *tan*
#-Suchalternative 1: #*CVn*
Suchalternative 1: *CVn*
Suchalternative 2: *Can*
Suchalternative 3: wird nicht generiert, da sie gleich dem Suchlexem wäre (=*tan*)
Suchalternative 4 (exit): *C*

Bei jedem Schleifendurchlauf wird in Leserichtung Buchstabe für Buchstabe durch ein Jokerzeichen ersetzt und die Buchstabenposition (beginnend mit der Suchlexemlänge) um eins dekrementiert. Fällt der Wert der Buchstabenposition unter vier, endet die Generierung der Alternative 3, ab einem Wert von kleiner drei die Generierung der Alternative 2, unter einem Wert von zwei, d.h. Buchstabenposition = 1, die Generierung der Alternative 1, und die Schleife terminiert. Danach werden die Suchalternativen mit den letzten Buchstaben des Suchlexems aufgefüllt.

3.3.7.2 Überprüfung der Suchalternativen (Abb. 3.15)

Im ersten Match-Schritt erfolgt eine Überprüfung, ob sich die #-Suchalternative 1 im Register befindet. Ist dies der Fall, wird die Funktion beendet, und die dazugehörigen Regellexeme werden ausgegeben.
Ansonsten werden das reduzierte Suchlexem und die Suchalternativen 4.3, 4.2 und 4.1 (sofern diese erzeugt wurden, andernfalls eine leere Alternative) an eine Do-While-Schleife[9] übergeben. Diese reduziert in Leserichtung (europäisch) sowohl das Suchlexem als auch dessen Suchalternativen Buchstabe um Buchstabe und überprüft in jedem Schleifendurchlauf, ob sich die verbleibende Buchstabenfolge in der Spalte “Synthetic lexical base“ im Register findet. Diese Überprüfung erfolgt nach der Reihenfolge der Bedingungshierarchie, d.h. reduziertes Suchlexem, reduzierte Alternative 4.3, dann 4.2 und endlich 4.1. Ergibt sich eine Suchüberein-

[9] Im ersten Durchlauf der Schleife wird das Suchlexem (hier noch in voller Länge) nicht überprüft, da dies schon die Funktion “total length compare“ getan hat (length of search lexeme ≠ original length of search lexeme).

stimmung, wird nur jenes Regellexem, welches eine Kennzeichnung (homC, basC2, basC u.a.) in der Clusterspalte besitzt, ausgegeben, und der Algorithmus terminiert.

Folgende Strukturen sind hierbei möglich:

#-Suchalternative 1 (2.2 aus Abb. 3.09 und Tab. 3.03)

- **Vollstring, mehrelementiges Cluster, mit Jokerzeichen, Wortbegrenzung**
  Beispiel *tan* → basC2 *#CVn: #CVn, #CVnned, #CVnned*
  also *tan, tanned, tanned*

Reduziertes Suchlexem (3)

- **Teil- / Vollstring, mehrelementiges Cluster, ohne Jokerzeichen, ohne Wortbegrenzung**
  Beispiel *search* → homC *ch: ~ch, ~ched, ~ched*
  also *search, searched, searched*

Reduzierte Suchalternative 3 (4.3)

- **Teilstring, mehrelementiges Cluster, mit Jokerzeichen, ohne Wortbegrenzung**
  Im Fall des Suchlexems *present* ergibt sich im ersten Schleifendurchlauf die Suchvariante *CCVCent*, welche auch durch Reduktion zu keiner Übereinstimmung mit Regellexemen führt.

Reduzierte Suchalternative 2 (4.2)

- **Teilstring, mehrelementiges Cluster, mit Jokerzeichen, ohne Wortbegrenzung**
  Im Fall des Suchlexems *present* ergibt sich im ersten Schleifendurchlauf die Suchvariante *CCVCVnt*, welche auch durch Reduktion zu keiner Übereinstimmung mit Regellexemen führt.

Reduzierte Suchalternative 1 (4.1)

- **Teilstring, mehrelementiges Cluster, mit Jokerzeichen, ohne Wortbegrenzung**
  Beispiel *frolic* → basC2 *CVc: ~CVc, ~CVcked, ~CVcked*

Wurde auch nach Überprüfen der dritten "Suchalternative" keine Übereinstimmung in den Registern gefunden, so werden das Suchlexem wie auch die Suchalternativen in Leserichtung (europäisch) um einen Buchstaben verkürzt, die Suchlexemlänge um eins dekrementiert und die Schleife beginnt von vorn.
Bei erfolgloser Suche, d.h. wenn die Suchlexemlänge den Wert 0 erreicht hat, terminiert die Schleife endgültig. Nach dem Schleifenende wird eine letzte Suche initiiert, bestehend aus dem durch Jokerzeichen ersetzten letzten Buchstaben des Suchlexems (Suchalternative 4 exit).

Suchalternative 4 exit (5)

- **Teilstring, Cluster, mit Jokerzeichen, ohne Wortbegrenzung**
  Beispiel *present* → basC *C: ~C, ~Ced, ~Ced*

Führt auch diese Überprüfung zu keinem Resultat, d.h. befindet sich weder das Suchlexem noch eine rückläufige Übereinstimmung als Regellexem im Register des Verbalsystems, terminiert der Algorithmus, und es erscheint die Meldung, dass sich kein passender Eintrag zum eingegebenen Suchlexem im Register dieser Sprache findet.

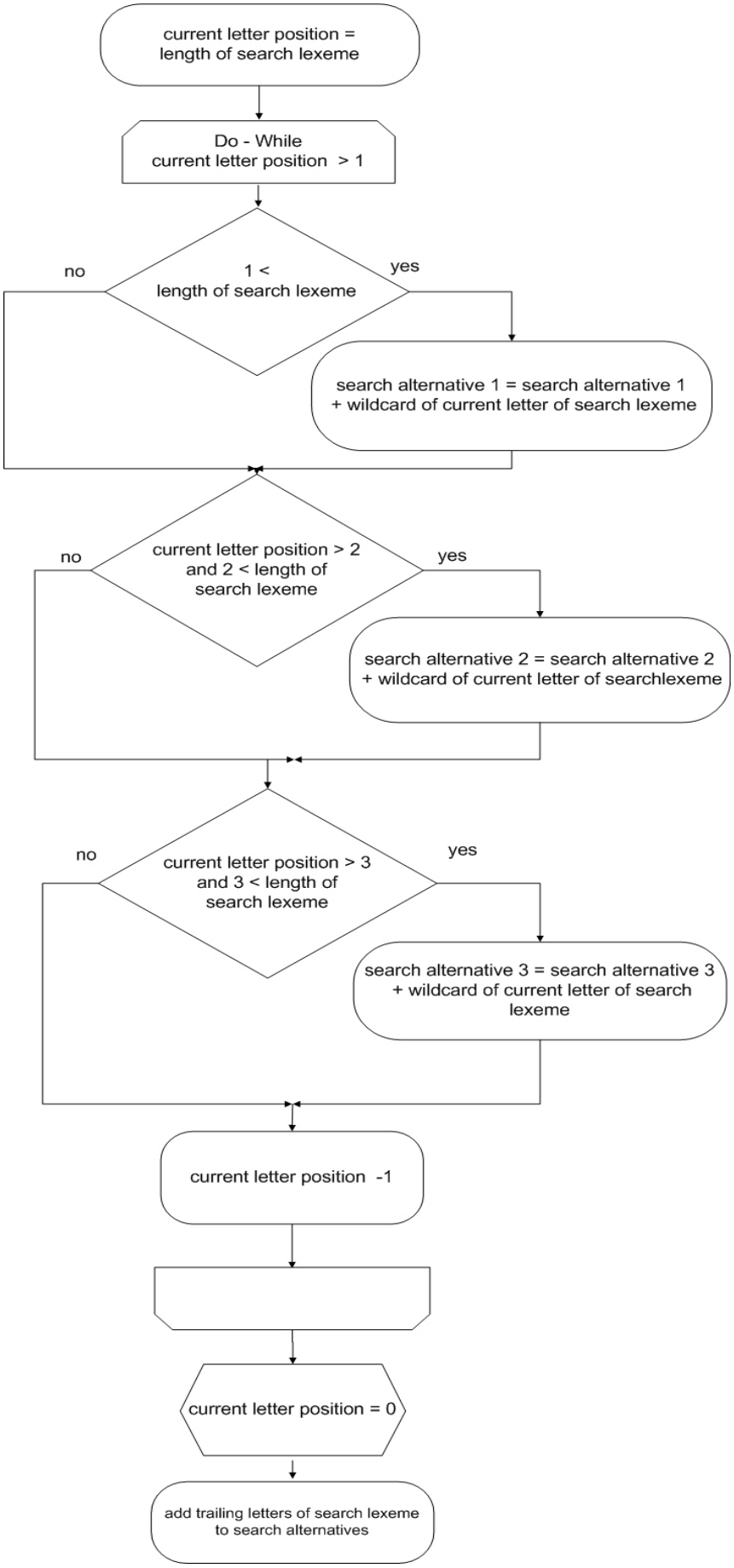

Abbildung 3.14: Ablaufplan der “longest match“-Funktion – Generierung der Suchalternativen

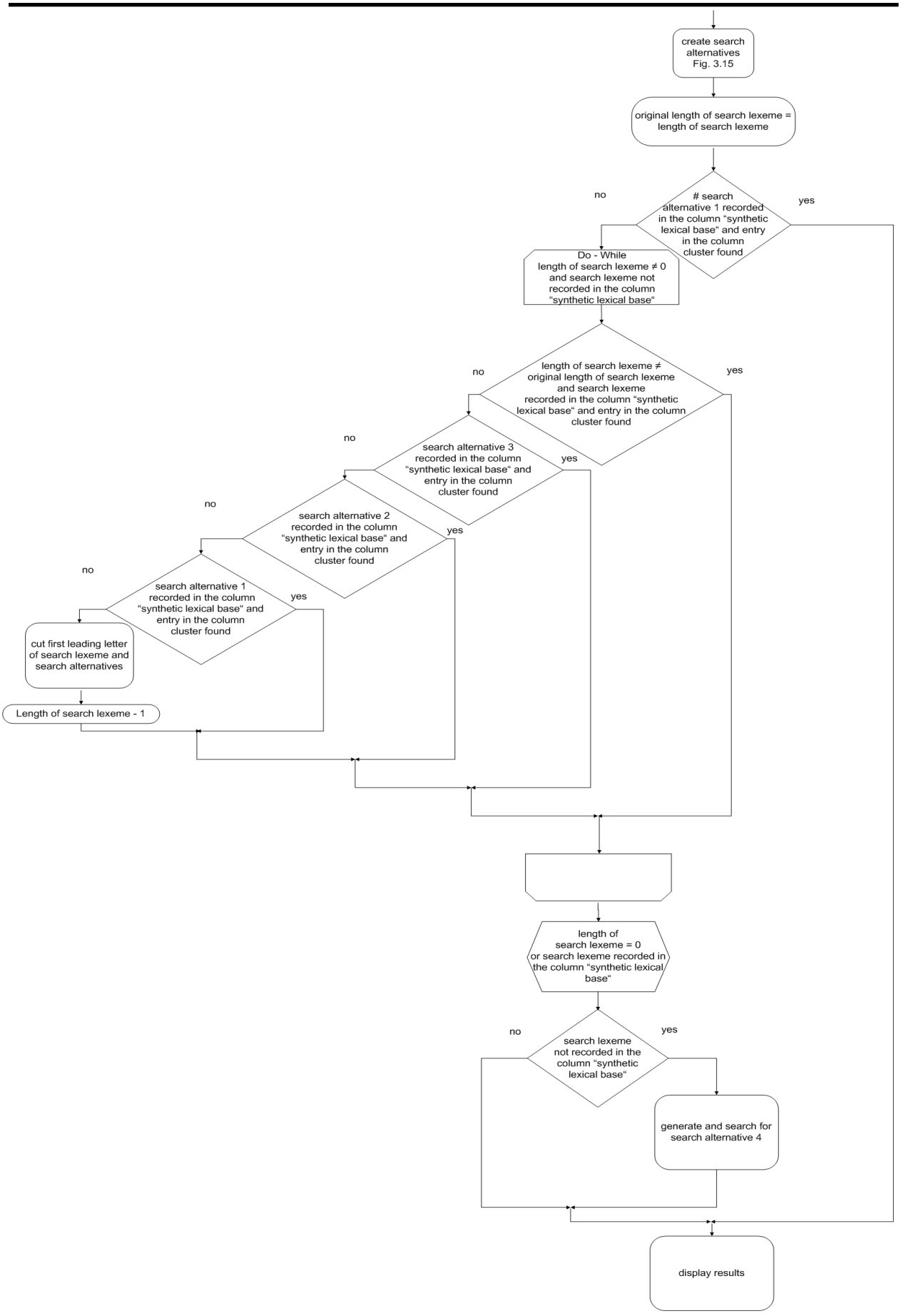

Abbildung 3.15: Ablaufplan der "longest match"-Funktion – Überprüfung der Suchalternativen

3.4 Konsequenzen für die Gestaltung der Register

Der Input des Algorithmus besteht aus drei Teilen, einem zufälligen Suchlexem, einer sprachabhängigen Alphabettabelle und dem entsprechenden Register aus dem Data-Mining-Prozess. Sowohl der erste als auch der zweite Schritt des synthetisch-generativen Teilbereichs aus 3.1.2 und 3.1.3 verdeutlichen, dass im Register einige Konventionen erfüllt sein müssen, um die korrekte Funktionsweise des SMIRT-Algorithmus gewährleisten zu können. In der nachfolgenden Tabelle 3.06 sind diese Konventionen aufgelistet und um einige weitere ergänzt, welche für die spätere technische Umsetzung ebenfalls von Bedeutung sein werden. Durch die Festlegung der Konventionen sowie eine exakte Definition der Eingangsgrößen ist es gelungen, den Algorithmus so zu formalisieren, dass dieser für die Register unterschiedlicher Verbalsysteme einwandfrei funktioniert und zu einem zufälligen Suchlexem die dazugehörigen Regellexeme findet.

Nr.	Konvention	Beschreibung
01	Jokerzeichen für Vokale und Konsonanten	Damit für ein zu untersuchendes Verbalsystem die entsprechenden Kombinationen aus Konsonant- / Vokal-Jokerzeichen durch den Algorithmus identifiziert werden können, benötigt dieser eine Zuordnungstabelle. In dieser sprachabhängigen Alphabet-Tabelle ist jedem Buchstaben des Alphabets ein "V" für Vokal oder ein "C" für Konsonant zugeordnet.
02	Anzahl sprachabhängiger Spalten	Der sprachabhängige Teil eines Registers kann in der Anzahl der Spalten variieren. Zur Vereinheitlichung und Übersichtlichkeit des Outputs wird der Import eines Registers auf die ersten 17 Spalten (inkl. sprachabhängigem Teil) begrenzt.
03	Bereinigung der Synthetic Lexical Base	Die "Synthetic Lexical Base"-Spalte ist die vom Algorithmus zu untersuchende Spalte. Suchlexem bzw. Suchalternativen werden mit den Registereinträgen dieser Spalte verglichen. Daher muss diese Spalte von numerischen, Sonderzeichen (ausgenommen dem #-Zeichen) und Blanks bereinigt werden.
04	Sprachabhängige Benennung der Registerspalten berücksichtigen	Die Benennung der Registerspalten erfolgt entsprechend den Schlüsselformen des zu untersuchenden Verbalsystems. Die Ausgabe der Regellexeme des jeweiligen Verbalsystems soll dies mitberücksichtigen und die Ausgabespalten entsprechend benennen.
05	Reihenfolge der Registerspalten	Die Suche erfolgt in der Spalte "Synthetic Lexical Base", während die Cluster-Spalte zur Verifizierung in der "longest match"-Funktion benötigt wird. Es ist erforderlich, dass diese beiden Spalten immer an derselben Position im Register zu finden sind. Die Einträge für die Cluster werden in der ersten, der Flexionstyp in der zweiten und die Einträge für Synthetic Lexical Base in der dritten Spalte des Registers erwartet.
06	Anzahl Jokerzeichen	Die Suchalternativen setzen sich aus Konsonant- / Vokalkombinationen plus den letzten Buchstaben des Suchwortes zusammen. Die Anzahl dieser letzten Buchstaben wird auf maximal drei begrenzt. Einträge in "Synthetic Lexical Base" müssen diese Struktur berücksichtigen.
07	Flexionsvarianten	Wie am Beispiel *grave* deutlich wird, sind Mehrfachresultate zulässig, was bei einem doppelten Eintrag der Regellexeme in den Regelsätzen vorkommt.
08	Klammerung	Keine Klammerung in der Spalte "Synthetic Lexical Base". Jeder Eintrag muss in einer eigenen Zeile eingetragen werden. Beispiel: *#C(C)Vn → #CCCVn, #CCVn und #CVn*

Tabelle 3.06: Konventionen für den Input

4. Implementierung des synthetisch-generativen Algorithmus

4.1 stellt das verwendete Implementierungskonzept vor. 4.2 erklärt die technische Umsetzung der in 3.3 entworfenen Funktionen.

4.1 Art der Implementierung

4.1.1 erklärt allgemein das verwendete Implementierungskonzept, dessen Aufbau und Struktur und den sich daraus ergebenden Nutzen und die Vorteile. 4.1.2 erläutert die technische Umsetzung dieses Konzepts in einem ersten Prototyp, die ausgewählten Komponenten und verwendeten Programmiersprachen.

4.1.1 Programmaufbau

Die Umsetzung des SMIRT-Algorithmus in einem ersten Prototyp (SMIRT Ver. 1.0) erfolgt auf Basis des MVC-Konzeptes (Model View Controller)[10]

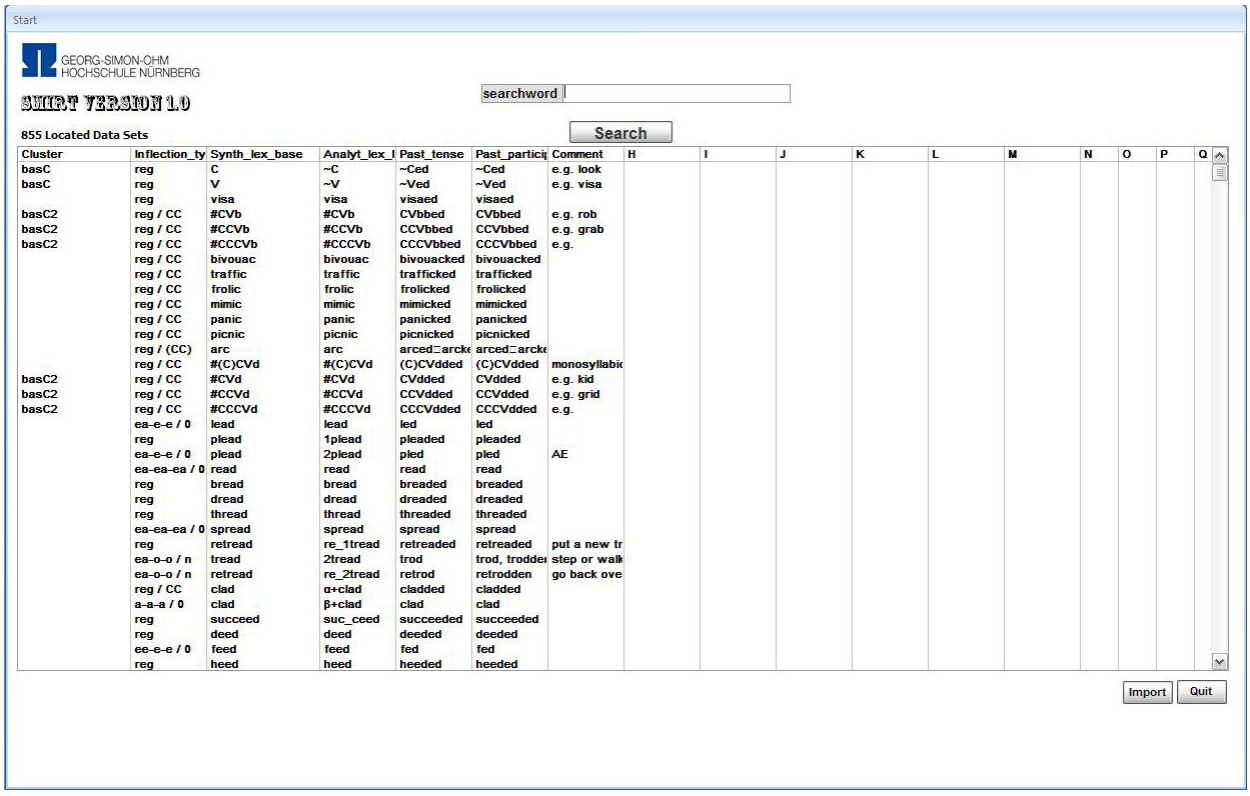

Abbildung 4.01: Benutzeroberfläche (View) Ver.1.0

[10] MVC-Konzept (vgl. Middendorf, Singer, Heid 2002)

Das Model-View-Controller-Konzept wird in vielen Bereichen moderner Softwareentwicklung eingesetzt und bedeutet die strikte Aufgabenverteilung bei einer Anwendung.
So wird als **Model** die Datenquelle bezeichnet, die die Daten unabhängig vom Design des Erscheinungsbildes der Benutzeroberfläche liefert (also beispielsweise aus einer relationalen Datenbank).
Das **View** zeigt diese Daten dann in passender Art und Weise an – bestimmt dadurch den »Look«. Wie das View die Daten anzeigt, wird nicht vom Model beeinflusst. Der **Controller** kümmert sich um die Interaktion mit dem Benutzer. Der Controller ist also die Logik der Anwendung. Der Vorteil dieser Aufgabenverteilung ist einerseits die Möglichkeit der Aufteilung in logische, unabhängige Bereiche, andererseits die Möglichkeit, jeden der drei Teile jederzeit auszutauschen.

4.1.2 Einsatz und Verwendung von Programmiersprachen

Die technische Umsetzung dieses Konzeptes erfolgt durch Microsoft Access. Für die Darstellung (Look) der Benutzeroberfläche (**View**) wird die integrierte Designumgebung von Access verwendet. Die Funktionalität der Buttons und Auswahlmenüs der Oberfläche ist durch die Programmiersprache VBA (Visual Basic for Application) realisiert. Die Kommunikation und Manipulation (**Controller**) der Daten erfährt ihre Umsetzung durch VBA als auch durch SQL (Structured Query Language). Die Tabellen der Datenbank (**Model**) stehen in keinerlei Relation zu einander, sie dienen in diesem Prototyp lediglich als Datenspeicher (Speicherung der Regellexeme, des Sprachalphabets und Speicherung der spachabhängigen Spaltenüberschriften). Das Suchlexem sowie dessen Suchalternativen (wie in 3.3.1 beschrieben) wird in VBA in String-Variablen gespeichert und den entsprechenden Funktionen übergeben. Die zur Realisierung verwendeten Do-While-Schleifen von Microsoft Access sind kopfgesteuert und somit abweisende Schleifen.

4.2 Realisierung der Funktionen

4.2.1 erläutert das Hauptprogramm. 4.2.2, 4.2.3 und 4.2.4 gehen auf die Umsetzung der Funktionen “total length compare“, “prefix cut“ und “longest match“ ein. 4.2.5 erklärt in kurzen Zügen die Zusatzfunktion für den Datenimport.

Die Realisierung der Funktionen erfolgt in Form einer eigenständigen Kapselung, d.h. sie sind in sich geschlossen und vom aufrufenden Programmteil getrennt. Dies ermöglicht einen gezielten Aufruf (Instanziierung), eine modulare Anpassung sowie die Wiederverwendbarkeit. In den Funktionen werden sowohl lokale als auch globale Variablen verwendet. Die lokalen Variablen sind nur innerhalb der Funktionen existent, während die globalen Variablen funktionsübergreifend verwendet werden.

4.2.1 Hauptprogramm

Name	Typ	Ort	Verwendung
F1 bis F17	String	global	In diesen Variablen werden während des Programmablaufs die Spaltenüberschriften des jeweiligen Verbregisters gespeichert, welche auch bei der Ausgabe der Regellexeme als Spaltenüberschriften verwendet und durch die Einträge aus der Tabelle “Head“ befüllt werden (s. u. in dieser Tabelle).
Flag_Match_Found	Integer	global	Zeigt an, ob eine Übereinstimmung des (reduzierten) Suchlexems mit einem Regellexem gefunden wurde. 0: keine Übereinstimmung gefunden 1: Übereinstimmung gefunden
Flag_Prefix_Found	Integer	global	Zeigt an, ob im Suchlexem ein Präfix gefunden wurde. 0: kein Präfix gefunden 1: Präfix gefunden
Searchlexeme	String	global	Speichert das durch den Benutzer eingegebene und ggf. im Programmverlauf bei einer Präfixmarkierung von “prefix cut“ verkürzte Suchlexem.
Que_Rules	SQL Abfrage	global	Que_Rules ist eine SQL-Text-Variable in ACCESS. Diese Variable wird im Laufe des Programms so lange verändert, bis sie den Text (SQL-String) der endgültigen, für die Ausgabe des Suchresultats relevanten SQL-Abfrage enthält. Diese SQL-Abfrage wird im Laufe des Programms nur an einer einzigen Stelle ausgeführt, nämlich am Programmende durch die Zuweisung (Me.Rule_Lexemes.RowSource="Que_Rules") an das Listenfeld. Diese Abfrage ist verantwortlich für die Ausgabe der Regellexeme und ist eine Selektion über alle 17 Spalten der Tabelle “Rules“.
User_Input_Of_ Searchlexeme	Eingabefeld	global	Name für das Eingabefeld des Suchlexems, definiert in der Designumgebung von ACCESS.
Rule_Lexemes	Listenfeld	global	Name für das Ausgabelistenfeld der Regellexeme, definiert in der Designumgebung von ACCESS.
Alphabet	Tabelle	global	Verwaltet das Alphabet des untersuchten Verbalsystems mit der Kennzeichnung, ob es sich bei einem Buchstaben um einen Vokal oder einen Konsonanten handelt.
Head	Tabelle	global	Die Spaltenüberschriften des untersuchten Verbalsystems sind hier hinterlegt.
Rules	Tabelle	global	Verwaltet die Regellexeme des untersuchten Verbalsystems, welche mittels Datenimport hinterlegt wurden; siehe 4.2.5.

Tabelle 4.01: Datenlexikon des SMIRT-Algorithmus

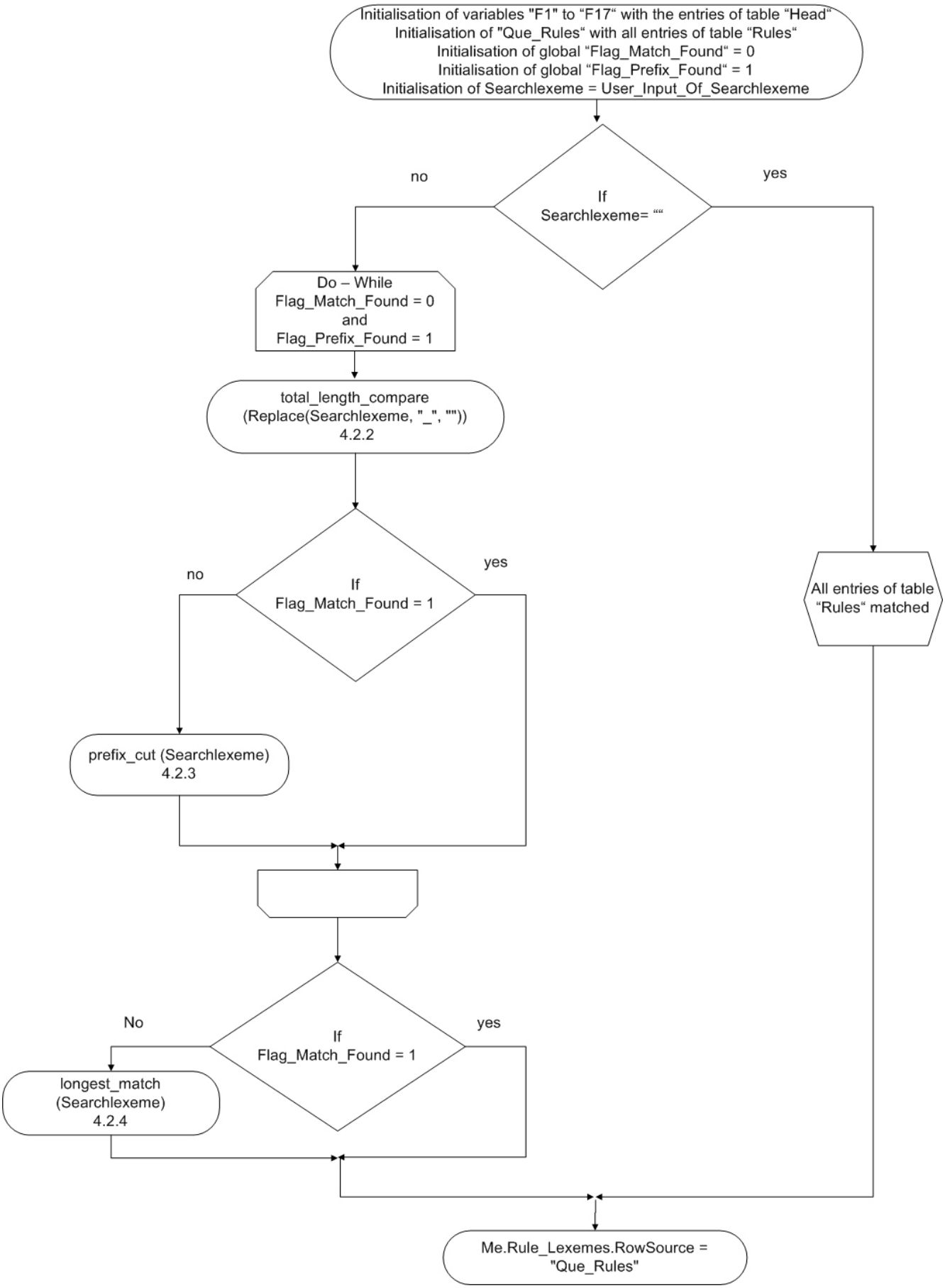

Abbildung 4.02: Ablaufplan der Implementierung des SMIRT-Algorithmus

Die Main-Funktion, welche durch das Betätigen des “Search-Buttons“ der Benutzeroberfläche aufgerufen wird, beginnt mit der Initialisierung der Variablen “F1“ bis “F17“ und weist diesen die entsprechenden Einträge aus der Tabelle “Head“ zu. Beim Programmstart sowie bei einer Leereingabe durch den Benutzer sollen alle sich in der Tabelle “Rules“ befindlichen Regellexeme als Resultat ausgegeben werden; daher wird die SQL-Abfrage “Que_Rules“ mit der Abfrage initialisiert, die alle Einträge der Tabelle “Rules“ liefert. Dabei werden die zuvor belegten Spaltenüberschriften in den Variablen “F1“ bis “F17“ verwendet, welche als Aliasnamen der Ausgabespalten benutzt werden. Die Variable “Flag_Match_Found“ erhält eine Vorbelegung mit dem Wert “0“ und “Flag_Prefix_Found“ mit “1“, die Variable “Searchlexeme“ wird mit dem durch den Benutzer eingegebenen Suchlexem initialisiert.

Eine If-Anweisung überprüft, ob die Variable “Searchlexeme“ einen Wert erhalten hat oder ob der “Search-Button“ ohne Eingabe ausgeführt wurde. Ist dies der Fall, werden (wie beschreiben) alle Regellexeme ausgegeben (dies entspricht der vorangegangenen Initialisierung der “Que_Rules“).

Besitzt die Variable “Searchlexeme“ einen Wert, wird eine Do-While-Schleife ausgeführt, die als Erstes die “total_length_compare“-Funktion aufruft und dieser das um sämtliche Unterstriche bereinigte Suchlexem übergibt[11]. Wird ein entsprechendes Regellexem gefunden, so erhält die Variable “Flag_Match_Found“ den Wert “1“. Andernfalls bleibt der Wert dieser Variablen unverändert.

Im weiteren Verlauf der Schleife überprüft eine If-Anweisung, ob die Variable “Flag_Match_Found“ den Wert “1“ erhalten hat. Trifft dies zu, terminiert die Schleife.
Wenn diese Bedingung nicht erfüllt ist, wird die “prefix_cut“-Funktion aufgerufen, welcher ebenfalls der Wert der Variablen “Searchlexeme“ übergeben wird. Diese überprüft, ob sich ein Unterstrich in dem übergebenen Suchlexem befindet. In diesem Fall wird das “Flag_Prefix_Found“ auf den Wert “1“ gesetzt, und dem “Searchlexeme“ wird der präfixreduzierte Wert (alle Buchstaben rechts vom Unterstrich) zugewiesen. Bleibt die Suche nach einem Unterstrich erfolglos, wird der Wert von “Processing_Lexeme“ nicht verändert, und das “Flag_Prefix_Found“ wird auf “0“ gesetzt, und die Schleife terminiert.

Nach der Schleife überprüft eine If-Anweisung, ob die Variable “Flag_Match_Found“ den Wert “1“ besitzt. Ist dies zutreffend, wird das Listenfeld der Ausgabe (Rule_Lexemes) mit der durch die “total_length_compare“-Funktion veränderten SQL-Abfrage “Que_Rules“ belegt. Ansonsten wird die “longest_match“-Funktion aufgerufen, die ihrerseits die SQL-Abfrage “Que_Rules“ verändert. Die Belegung des Listenfelds erfolgt erst nach deren Abarbeitung.

Hinweis:
In den Funktionen “total length compare“ und “longest match“ wird die Suche nach dem ggf. durch die “prefix cut“-Funktion veränderten Suchlexem durch eine Selektion über die ersten drei Spalten (F1-F3) der Tabelle “Rules“ durchgeführt. Das Ergebnis wird einer Variable vom Typ DAO.recordset übergeben, um mit Hilfe der VBA-Funktion “RecordCount“ festzustellen, ob zu diesem Suchlexem Regellexeme vorhanden sind. Die Ausgabe der tatsächlich gefundenen Regellexeme erfolgt durch eine andere Selektionsanweisung über alle 17 Spalten der Tabelle “Rules“, welche einer SQL-Text-Variablen namens “Que_Rules“ übergeben wird. Das Ergebnis der Abfrage dient als Datenbasis für die Ausgabe des Listenfeldes.

[11] Dies wird über die accessinterne Replace-Funktion erreicht, welche eine ausgewählte Zeichenfolge sucht und durch eine andere ersetzt, in diesem Fall den Unterstrich durch eine leere Zeichenfolge.

```
Private Sub Search_Button_Click()

F1 = DLookup("[F1]", "[Head]", "")
F2 = DLookup("[F2]", "[Head]", "")
F3 = DLookup("[F3]", "[Head]", "")
F4 = DLookup("[F4]", "[Head]", "")
F5 = DLookup("[F5]", "[Head]", "")
F6 = DLookup("[F6]", "[Head]", "")
F7 = DLookup("[F7]", "[Head]", "")
F8 = DLookup("[F8]", "[Head]", "")
F9 = DLookup("[F9]", "[Head]", "")
F10 = DLookup("[F10]", "[Head]", "")
F11 = DLookup("[F11]", "[Head]", "")
F12 = DLookup("[F12]", "[Head]", "")
F13 = DLookup("[F13]", "[Head]", "")
F14 = DLookup("[F14]", "[Head]", "")
F15 = DLookup("[F15]", "[Head]", "")
F16 = DLookup("[F16]", "[Head]", "")
F17 = DLookup("[F17]", "[Head]", "")

CurrentDb.QueryDefs("Que_Rules").sql = " SELECT Rules.F1 as [" & F1 & "], Rules.[F2] as [" & F2 & "], Rules.F3 as
[" & F3 & "], Rules.F4 as [" & F4 & "], Rules.F5 as [" & F5 & "], Rules.F6 as [" & F6 & "], Rules.F7 as [" & F7 & "], Ru-
les.F8 as [" & F8 & "], Rules.F9 as [" & F9 & "], Rules.F10 as [" & F10 & "], Rules.F11 as [" & F11 & "], Rules.F12 as
[" & F12 & "], Rules.F13 as [" & F13 & "], Rules.F14 as [" & F14 & "], Rules.F15 as [" & F15 & "],Rules.F16 as [" &
F16 & "], Rules.F17 as [" & F17 & "]" & _
" FROM Rules "
Flag_Match_Found = 0
Flag_Prefix_Found = 1
Searchlexeme = Nz(Me.User_Input_Of_Searchlexeme.Value)

If (Searchlexeme) = "" Then

Else

    Do While (Flag_Match_Found = 0 and Flag_Prefix_Found = 1)

      total_length_compare (Replace(Searchlexeme, "_", ""))

      If (Flag_Match_Found = 1) Then

         Else
           prefix_cut (Searchlexeme)
      End If

    Loop

    If Flag_Match_Found = 1 Then

    Else
        longest_match (Searchlexeme)

    End If
End If

Me.Rule_Lexemes.RowSource = "Que_Rules"

End Sub
```

Quellcode 4.01: Main-Funktion (Search_Button_Click)

4.2.2 Realisierung der Funktion “total length compare“

Name	Typ	Ort	Verwendung
rs	DAO.recordset	lokal	Speichert das Ergebnis einer SQL-SELECT-Anweisung. Mithilfe der in VBA für DAO.recordset-Variablen bereitgestellten Funktion “Recordcount“ lässt sich einfach feststellen, wie viele Einträge in der Variable gespeichert sind, d.h. ob der an die Variable übergebene SQL-String mögliche Ausgaberesultate erzielte.
Flag_Match_Found	Integer	global	Zeigt an, ob eine Übereinstimmung des (reduzierten) Suchlexems mit einem Regellexem gefunden wurde. 0: keine Übereinstimmung gefunden 1: Übereinstimmung gefunden
Processing_Lexeme	String	lokal	Speichert das an die Funktion übergebene Suchlexem.
Que_Rules	SQL	global	Que_Rules ist eine SQL-Text-Variable in ACCESS. Diese Variable wird im Laufe des Programms so lange verändert, bis sie den Text (SQL-String) der endgültigen, für die Ausgabe des Suchresultats relevanten SQL-Abfrage enthält. Diese SQL-Abfrage wird im Laufe des Programms nur an einer einzigen Stelle ausgeführt, nämlich am Programmende durch die Zuweisung (Me.Rule_Lexemes.RowSource="Que_Rules") an das Listenfeld. Diese Abfrage ist verantwortlich für die Ausgabe der Regellexeme und ist eine Selektion über alle 17 Spalten der Tabelle “Rules“.

Tabelle 4.02: Datenlexikon der “total length compare“-Funktion

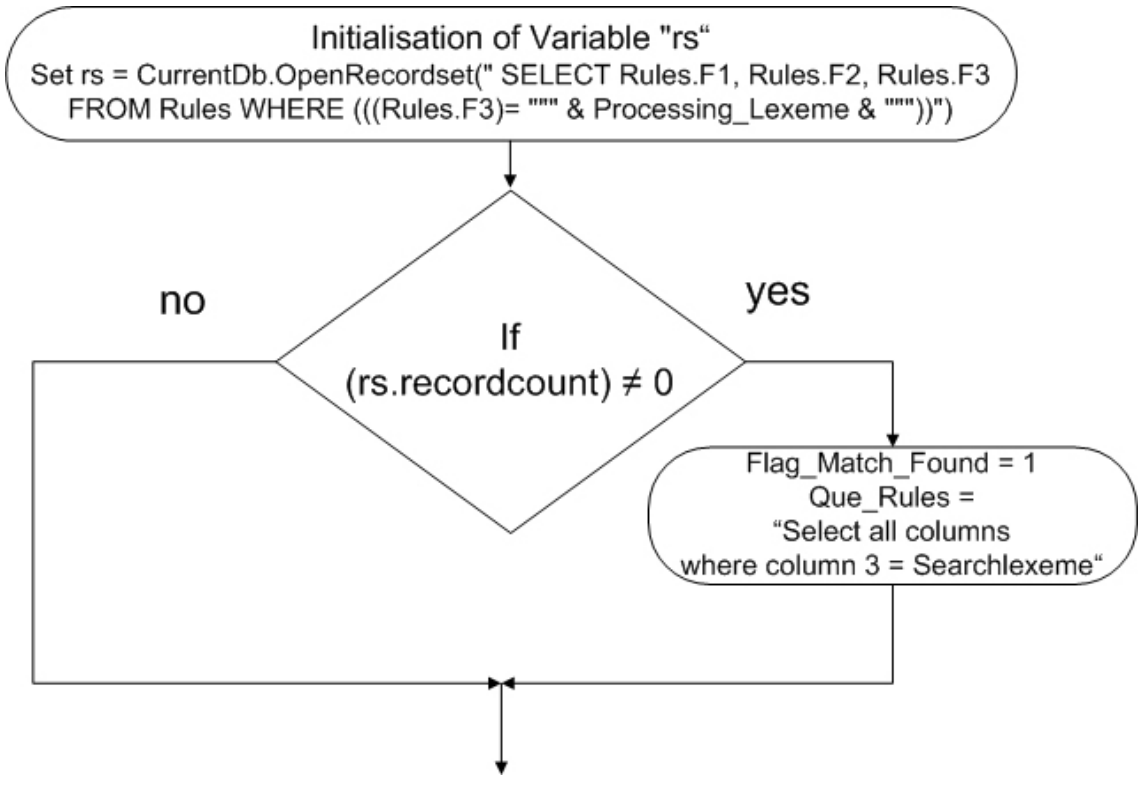

Abbildung 4.02: Ablaufplan der Implementierung der “total length compare“-Funktion

Durch den Aufruf durch die “Main“-Funktion wird der “total length compare“-Funktion eine Variable vom Typ String übergeben. Diese entspricht dem in 3.3.4 beschriebenen akzeptierten Suchlexem ohne Präfixmarker (Unterstriche).
Es wird eine lokale Variable “rs“ vom Typ Recordset[12] deklariert. Nach deren Instanziierung wird diesem Objekt das Ergebnis einer SQL-SELECT-Anweisung übergeben. Die SELECT-Anweisung selektiert die Tabellenspalten von F1 bis F3 aus der Tabelle Rules und ermittelt jene, bei denen sich in der Spalte F3 (Synthetic lexical base) das Suchlexem befindet.

Eine If-Anweisung überprüft, ob der Recordcount[13] des Objektes “rs“ ungleich “0“ ist. Wenn dies der Fall ist, existiert zu dem eingegebenen Suchlexem mindestens ein Eintrag in der Tabelle der Regellexeme, und die Variable “Flag_Match_Found“ wird auf “1“ gesetzt.
Desweiteren wird die im DBMS von ACCESS hinterlegte SQL-Abfrage namens “Que_Rules“ mit einer aktualisierten SQL-Anweisung überschrieben. Diese Anweisung selektiert alle Tabellenspalten von “F1“ bis “F17“ aus der Tabelle “Rules“ und weist ihnen als Alias-Namen die Inhalte der globalen Variablen “F1“ bis “F17“ zu. Es werden nur jene Zeilen ausgewählt, bei denen der Wert der Spalte F3 gleich der an die Funktion übergebenen Variable “Searchlexeme“ (entspricht dem Wert der Variable “Processing_Lexeme“) ist.

Danach endet diese Funktion. Das geschieht ebenfalls, wenn “RecordCount“ das Ergebnis “0“ enthält und der leere Else-Zweig der If-Anweisung ausgeführt wird.

```
Function total_length_compare(Processing_Lexeme As String) As String

Dim rs As DAO.Recordset

Set rs = CurrentDb.OpenRecordset(" SELECT Rules.F1, Rules.F2, Rules.F3 FROM Rules WHERE (((Rules.F3)= """ & Processing_Lexeme & """))")

If rs.RecordCount <> 0 Then

Flag_Match_Found = 1

CurrentDb.QueryDefs("Que_Rules").sql = " SELECT Rules.F1 as [" & F1 & "], Rules.[F2] as [" & F2 & "], Rules.F3 as [" & F3 & "], Rules.F4 as [" & F4 & "], Rules.F5 as [" & F5 & "], Rules.F6 as [" & F6 & "], Rules.F7 as [" & F7 & "], Rules.F8 as [" & F8 & "], Rules.F9 as [" & F9 & "], Rules.F10 as [" & F10 & "], Rules.F11 as [" & F11 & "], Rules.F12 as [" & F12 & "], Rules.F13 as [" & F13 & "], Rules.F14 as [" & F14 & "], Rules.F15 as [" & F15 & "],Rules.F16 as [" & F16 & "], Rules.F17 as [" & F17 & "]" & _
" FROM Rules WHERE (((Rules.F3) = """ & Processing_Lexeme & """))"
Else

End If

End Function
```

Quellcode 4.02: “total length compare“-Funktion

[12] Durch die Konzeption des Recordsets als Objekt mit Eigenschaften und Methoden hat man einen einfachen und komfortablen Zugriff auf die Daten einer Datenbank. Dieser bietet jedem Recordset-Objekt die Möglichkeit, nach bestimmten Werten zu suchen, Werte zu sortieren, Datensätze hinzuzufügen, wieder zu löschen oder zu zählen (vgl. Zimmer, 2006).

[13] Die RecordCount-Eigenschaft ist eine DAO (Data Access Object)-Eigenschaft, die die Anzahl der Datensätze in einer Tabelle einer Access-Datenbank anzeigt (vgl. Microsoft Hilfe und Support, 2004).

4.2.3 Realisierung der Funktion “prefix cut“

Name	Typ	Ort	Verwendung
Search_Letter	String	lokal	Speichert einen Buchstaben des Suchlexems.
Processing_ Lexeme_Length	Integer	lokal	Speichert die Anzahl Buchstaben des Processing_Lexeme
Flag_Prefix_Found	Integer	global	Zeigt an, ob im Suchlexem ein Präfix gefunden wurde. 0: kein Präfix gefunden 1: Präfix gefunden
Searchlexeme	String	global	Speichert das bei Vorhandensein einer Präfixmarkierung von “prefix cut“ verkürzte Suchlexem.
Processing_Lexeme	String	lokal	Speichert das an die Funktion übergebene Suchlexem sowie das durch die Funktion “prefix cut“ reduzierte Suchlexem.

Tabelle 4.03: Datenlexikon der “prefix cut“-Funktion

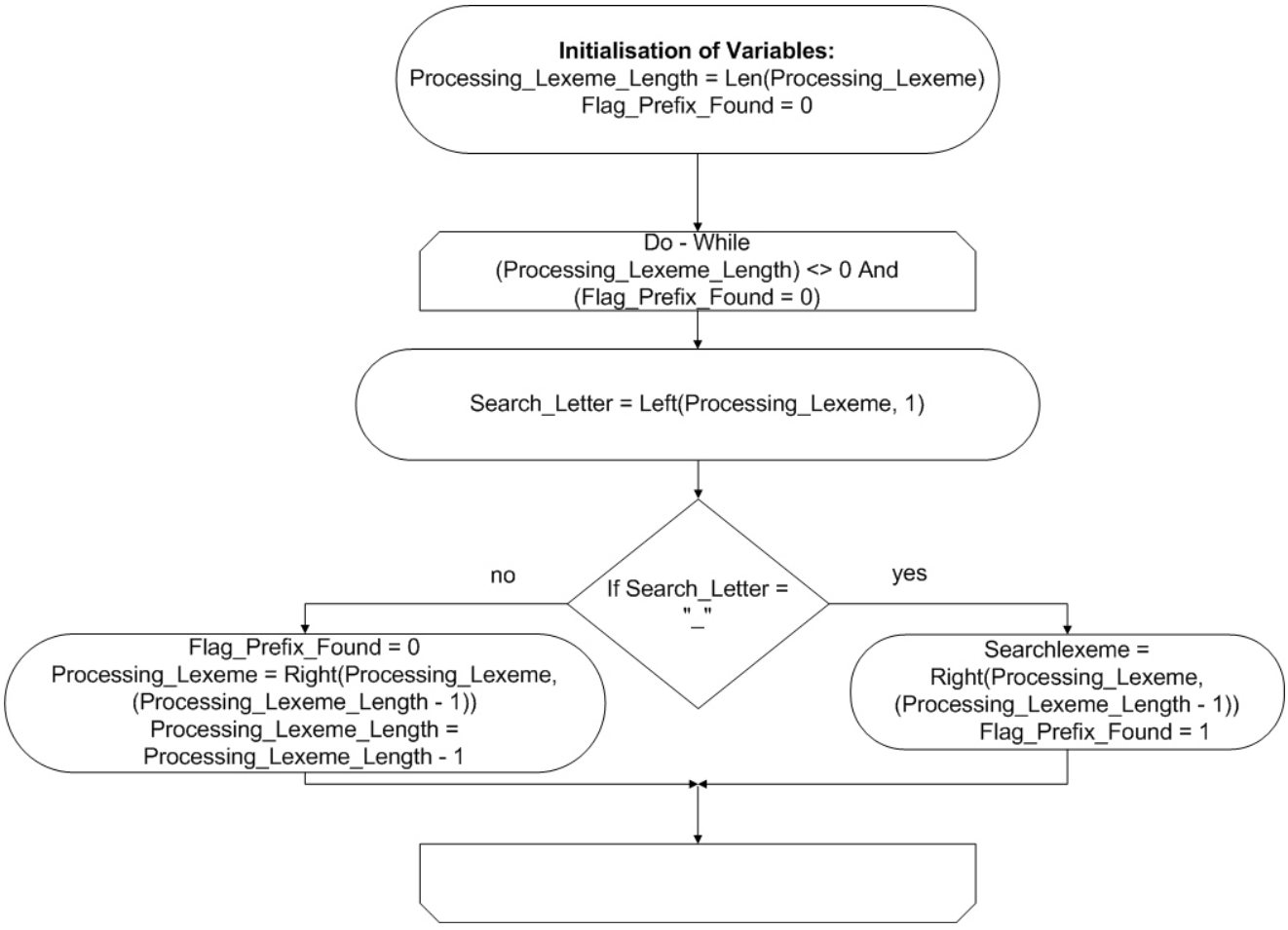

Abbildung 4.04: Ablaufplan der Implementierung der “prefix cut“-Funktion

Beim Aufruf durch die “Main“-Funktion wird der “prefix cut“-Funktion eine Variable vom Typ String übergeben. Diese entspricht dem in 3.3.4 beschriebenen akzeptierten Suchlexem.
Nach der Deklaration gemäß Datenlexikon (Tabelle 4.03) wird die Variable “Processing_Lexeme_Length“ mittels der Funktion “len“[14] mit der Suchlexemlänge initialisiert, und das “Flag_Prefix_Found“ erhält den Wert “0“, d.h. keine Präfixmarkierung.
Nachfolgend wird eine Do-While-Schleife gestartet, in deren Schleifenkopf die Einstiegsbedingung “(Processing_Lexeme_Length) <> 0 And (Flag_Prefix_Found = 0)“ geprüft wird. Diese Schleife wird nun solange wiederholt, bis diese Einstiegsbedingung nicht mehr erfüllt ist. Der Schleifenrumpf beginnt mit einer Initialisierung der Variable “Search_Letter = Left(Processing_Lexeme, 1)“, welche mittels der “left“-Funktion[15] realisiert wird.

Beispiel:
Processing_Lexeme = “en_grave“ (implizit durch den Funktionsaufruf)
Processing_Lexeme_Length = “8“
Flag_Prefix_Found = 0

Erster Schleifendurchlauf
Search_Letter = Left(“en_grave“), 1) →“e“
If-Anweisung→“False“ → Else
Flag_Prefix_Found = 0
Processing_Lexeme = Right(“en_grave“), (8-1)) → “n_grave“
Processing_Lexeme_Length = 8 - 1 →“7“

Zweiter Schleifendurchlauf
Search_Letter = Left(“n_grave“), 1) →“n“
If-Anweisung→“False“ → Else
Flag_Prefix_Found = 0
Processing_Lexeme = Right(“en_grave“), (7-1)) → “_grave“
Processing_Lexeme_Length = 7 - 1 →“6“

Dritter Schleifendurchlauf
Search_Letter = Left(“_grave“), 1) →“_“
If-Anweisung→“True“
Searchlexeme = Right(Processing_Lexeme, (6 - 1)) → “grave“
Flag_Prefix_Found = 1

Durch dieses Verfahren wird das gesamte Suchlexem (Processing_Lexeme) Buchstabe für Buchstabe an die Variable “Search_Letter“ übergeben.
Eine If-Anweisung überprüft, ob sich in dieser Variable ein Unterstrich befindet. Wird dieser identifiziert, wird der globalen “Searchlexeme“-Variable mittels der “right“-Funktion[16] der verbleibende Wert der Variablen “Processing_Lexeme“ (alle Buchstaben rechts vom Unterstrich) zugewiesen und das “Flag_Prefix_Found“ auf den

[14] Die len-Funktion behandelt den ihr übergebenen Inhalt einer Variable wie eine Zeichenfolge. Wenn eine leere Variable übergeben wird, wird der Wert Null zurückgegeben, ansonsten wird die Anzahl der sich in der Variable befindlichen Zeichen (numerische Daten, Zeichenfolgendaten, Datumsangaben) zurückgegeben (vgl. Microsoft Office, 2008).

[15] Gibt einen Wert zurück, der ausgehend von der linken Seite einer Zeichenfolge eine angegebene Anzahl von Zeichen enthält (vgl. Microsoft Office, 2008).

[16] Gibt einen Wert zurück, der ausgehend von der rechten Seite einer Zeichenfolge eine angegebene Anzahl von Zeichen enthält (vgl. Microsoft Office, 2008).

Wert “1“ gesetzt, worauf die Schleife terminiert. Im anderen Fall wird das “Flag_Prefix_Found“ auf “0“ gesetzt, die “Processing_Lexeme_Length“ dekrementiert und die String-Variable “Processing_Lexeme“ (welche das reduzierte Suchlexem enthält) mittels der “right“-Funktion mit dem verbleibenden Wert der Variablen “Processing_Lexeme“ (alle Buchstaben ohne den ersten) belegt. Danach beginnt die Schleife erneut, bis die Eintrittsbedingung im Schleifenkopf nicht mehr erfüllt ist.

```
Function prefix_cut(Processing_Lexeme As String) As String

Dim Search_Letter As String
Dim Processing_Lexeme_Length As Integer

Processing_Lexeme_Length = Len(Processing_Lexeme)
Flag_Prefix_Found = 0

Do While ((Processing_Lexeme_Length) <> 0 And (Flag_Prefix_Found = 0))

   Search_Letter = Left(Processing_Lexeme, 1)

      If Search_Letter = "_" Then

      Searchlexeme = Right(Processing_Lexeme, (Processing_Lexeme_Length - 1))
      Flag_Prefix_Found = 1

      Else

        Flag_Prefix_Found = 0
        Processing_Lexeme = Right(Processing_Lexeme, (Processing_Lexeme_Length - 1))
        Processing_Lexeme_Length = Processing_Lexeme_Length - 1

      End If

Loop

End Function
```

Quellcode 4.03: “prefix cut“-Funktion

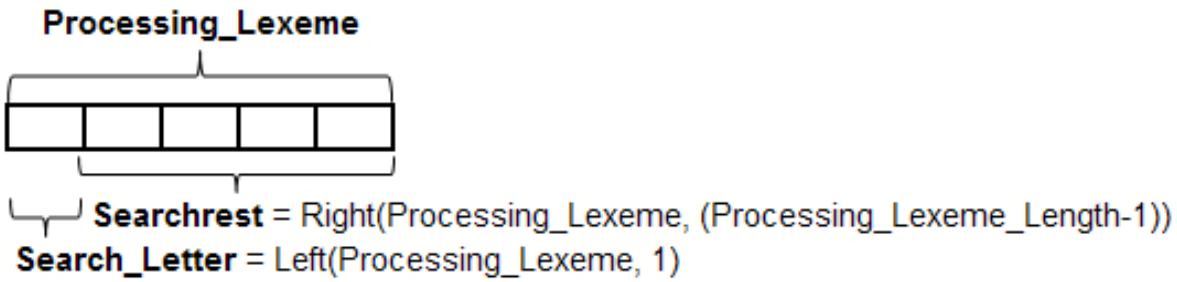

Abbildung 4.04: Verarbeitung der “prefix cut“-Funktion

4.2.4 Realisierung der Funktion “longest match“

Name	Typ	Ort	Verwendung
rs	DAO. recordset	lokal	Speichert das Ergebnis einer SQL-SELECT-Anweisung. Mithilfe der in VBA für DAO.recordset-Variablen bereitgestellten Funktion “Recordcount“ lässt sich einfach feststellen, wie viele Einträge in der Variable gespeichert sind, d.h. ob der an die Variable übergebene SQL-String mögliche Ausgaberesultate erzielte.
Searchalternative _1	String	lokal	
Searchalternative _2	String	lokal	
Searchalternative _3	String	lokal	
Processing _Lexeme_Length	Integer	lokal	Speichert die Anzahl Buchstaben des Processing_Lexeme
Flag_Match_Found	Integer	global	Zeigt an, ob eine Übereinstimmung des (reduzierten) Suchlexems mit einem Regellexem gefunden wurde. 0: keine Übereinstimmung gefunden 1: Übereinstimmung gefunden
Searchlexeme	String	global	Speichert das bei Vorhandensein einer Präfixmarkierung von “prefix cut“ verkürzte Suchlexem.
Match_Lexeme	String	lokal	Speichert das mit einem Registereintrag übereinstimmende veränderte Suchlexem.
Processing _Lexeme	String	lokal	Speichert das an die Funktion übergebene Suchlexem sowie das durch die “longest match“-Funktion reduzierte Suchlexem.
Que_Rules	SQL Abfrage	global	Que_Rules ist eine SQL-Text-Variable in ACCESS. Diese Variable wird im Laufe des Programms so lange verändert, bis sie den Text (SQL-String) der endgültigen, für die Ausgabe des Suchresultats relevanten SQL-Abfrage enthält. Diese SQL-Abfrage wird im Laufe des Programms nur an einer einzigen Stelle ausgeführt, nämlich am Programmende durch die Zuweisung (Me.Rule_Lexemes.RowSource="Que_Rules") an das Listenfeld. Diese Abfrage ist verantwortlich für die Ausgabe der Regellexeme und ist eine Selektion über alle 17 Spalten der Tabelle “Rules“.

Tabelle 4.04: Datenlexikon der “longest match“-Funktion

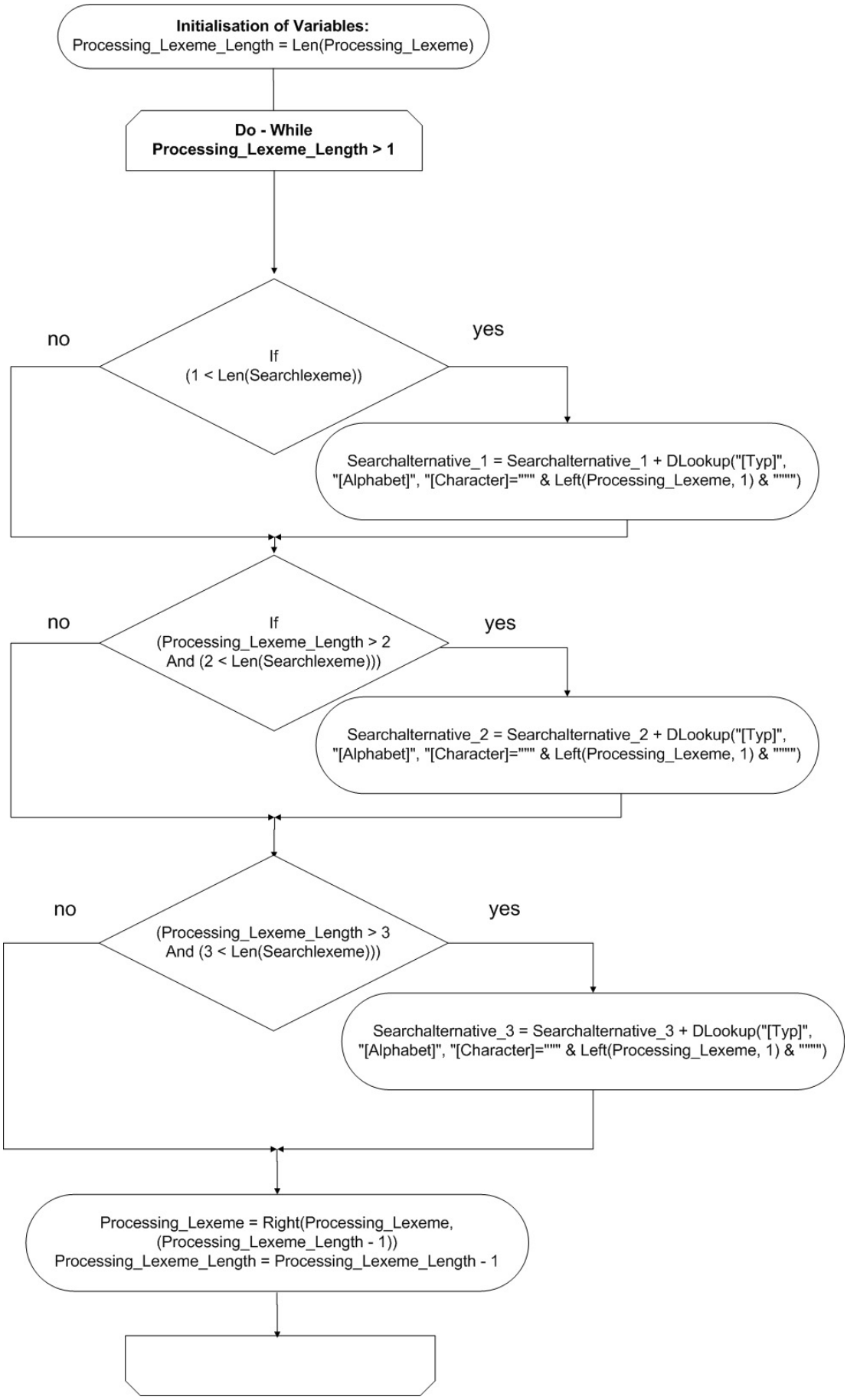

Abbildung 4.05: Ablaufplan der Implementierung der “longest match“-Funktion: Generierung von Jokerzeichen

Abbildung 4.06: Ablaufplan der Implementierung der “longest match“-Funktion: rechtsbündiges Auffüllen mit Buchstaben

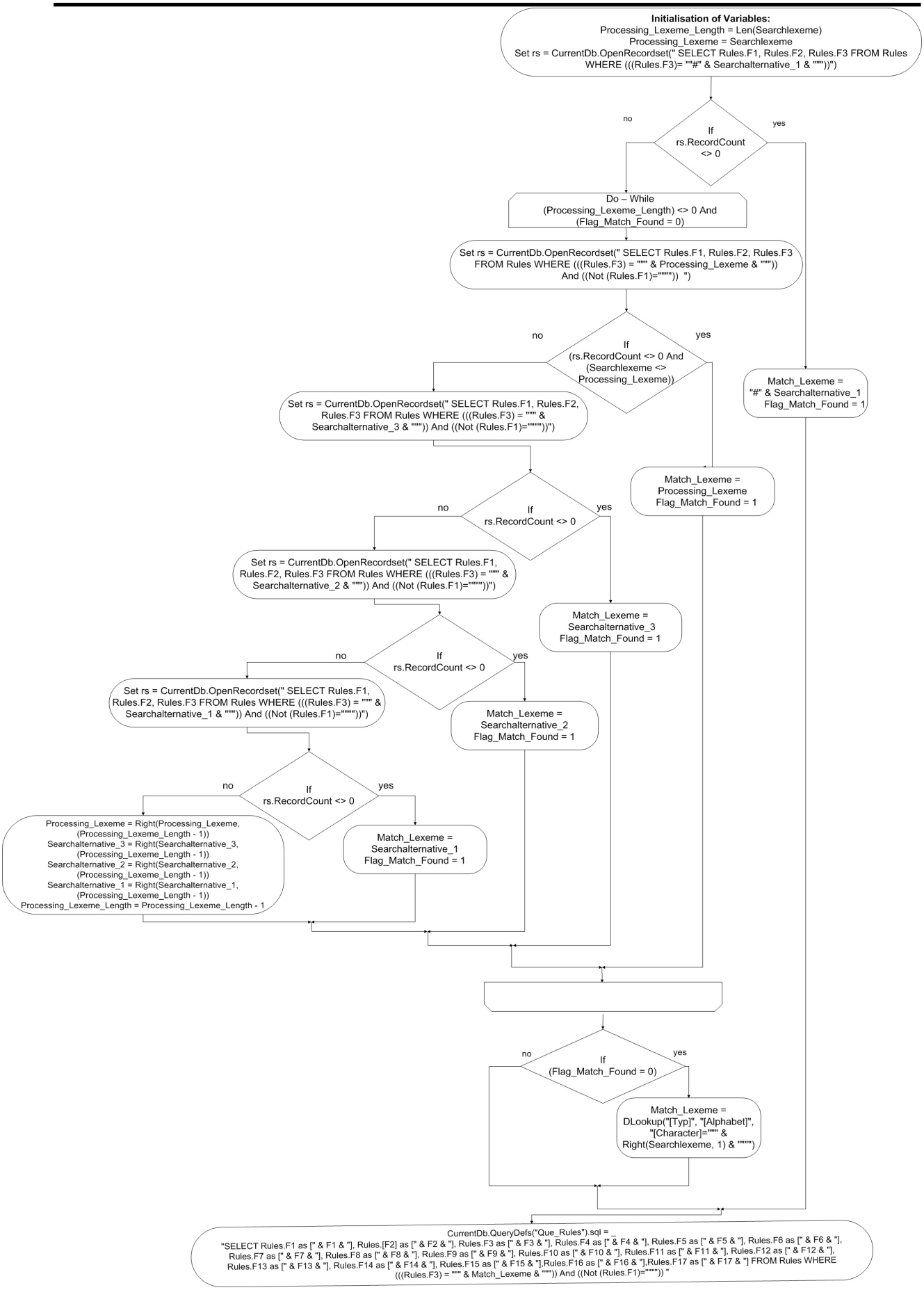

Abbildung 4.07: Ablaufplan der Implementierung der “longest match“-Funktion: Prüfen der Suchalternativen

Beim Aufruf durch die “Main“-Funktion wird der “longest match“-Funktion eine Variable vom Typ String übergeben. Diese entspricht dem in 3.3.4 beschriebenen akzeptierten Suchlexem. Nach der Deklaration gemäß Datenlexikon (Tabelle 4.04) wird die Variable “Processing_Lexeme_Length“ mit der Länge des an die Funktion übergebenen Suchlexems initialisiert.

4.2.4.1 Generierung der Suchalternativen

Nun beginnt das Generieren der Suchalternativen mit Jokerzeichen (Abb. 4.05 und 4.06). Dazu wird eine Do-While-Schleife gestartet, in deren Schleifenkopf die Einstiegsbedingung “Processing_Lexeme_Length > 1“ geprüft wird.
Es folgen drei If-Anweisungen, welche in Abhängigkeit der an die Funktion übergebenen Suchlexemlänge und der Länge des Verarbeitungslexems überprüfen, ob eine entsprechende Suchalternative generiert bzw. wie viele Jokerzeichen angefügt werden. Ist die jeweilige Bedingung erfüllt, wird den Suchalternativen (“Searchalternative_1“, “Searchalternative_2“, “Searchalternative_3“) ihr eigener Wert, d.h. der Wert aus dem vorangegangenen Schleifendurchlauf, konkateniert mit dem Jokerzeichen (C / V), welches mittels der “DLookup“-Funktion“[17] ermittelt wird, übergeben. Die “DLookup“-Funktion“ sucht einen Eintrag in der Tabellenspalte “Typ“ aus der Tabelle “Alphabet“, bei dem der Wert in der Spalte “Character“ gleich dem linken Buchstaben der Variablen “Processing_Lexeme“ ist.

Beispiel für “Searchalternative“ 2:
Processing_Lexeme = “present“
Processing_Lexeme_Length = Len(“present“) →“7“

Erster Schleifendurchlauf
If (7 > 2 And (2 < Len(“present“)))) →“True“

Searchalternative_2 = NULL + DLookup("[Typ]", "[Alphabet]", "[Character]=""" & Left(“present“, 1) & """")
d.h. Searchalternative_2 = NULL + DLookup("[Typ]", "[Alphabet]", "[Character]="p“)
d.h. Searchalternative_2 = NULL + “C“

…
Processing_Lexeme = Right(Processing_Lexeme, (7 - 1)) →“resent“
Processing_Lexeme_Length = 7 - 1 →“6“

Zweiter Schleifendurchlauf
If (6 > 2 And (2 < Len(“Present“)))) →“True“
Searchalternative_2 = “C“ + DLookup("[Typ]", "[Alphabet]", "[Character]=""" & Left(“resent“, 1) & """")
d.h. Searchalternative_2 = “C“ + DLookup("[Typ]", "[Alphabet]", "[Character]="r“)
d.h. Searchalternative_2 = “C“ + “C“
…
Processing_Lexeme = Right(Processing_Lexeme, (6 - 1)) →“esent“
Processing_Lexeme_Length = 6 - 1 →“5“
usw.

Mit diesem Verfahren werden je Schleifendurchlauf die Suchalternativen nach und nach mit Jokerzeichen gefüllt, wobei das Anfügen der Jokerzeichen je nach Suchalternative an unterschiedlichen Positionen beendet

[17] Die DLookup-Funktion wird verwendet, um einen Wert eines bestimmten Feldes aus einer angegebenen Datensatzgruppe, definiert durch eine Tabelle, eine Abfrage oder einen SQL-Ausdruck, zu entnehmen (vgl. Microsoft Office, 2008).

wird (Suchalternative_1: Anzahl Buchstaben des Suchlexems - 1, Suchalternative_2: Anzahl Buchstaben des Suchlexems - 2, Suchalternative_3: Anzahl Buchstaben des Suchlexems - 3).
Nach der Schleife werden in drei If-Anweisungen die Suchalternativen mit der “right“-Funktion um die “fehlenden“ echten Buchstaben des Suchlexems ergänzt (Abb. 4.06), sofern eine Suchalternative erstellt wurde. Ansonsten wird diese mit “NULL“ belegt.

4.2.4.2 Überprüfung der Suchalternativen

Nach der Generierung der Suchalternativen wird der Wert der “Processing_Lexeme_Length“ auf die Länge des “Searchlexeme“ und das “Processing_Lexeme“ auf “Searchlexeme“ zurückgesetzt.
Zunächst wird der Variable “rs“ das Ergebnis einer SQL-SELECT-Anweisung übergeben. Diese SQL-Anweisung selektiert die ersten drei Spalten der Tabelle “Rules“. Es werden nur jene Datensätze in der Variablen “rs“ gespeichert, welche in der Spalte “F3“ (Synthetic lexical base) den Eintrag der Suchalternative_1 mit vorange-stelltem #-Zeichen enthalten.
Eine If-Anweisung überprüft, ob die Anzahl (“RecordCount“) der in der Variablen gespeicherten Einträge ungleich “0“ ist. Wenn diese Bedingung erfüllt ist, wird der Wert der Variable “Match_Lexeme“ auf den Wert der Suchalternative_1 mit vorangestellten #-Zeichen gesetzt, und das “Flag_Match_Found“ erhält den Wert “1“. Die nachfolgende Do-While-Schleife wird in diesem Fall komplett übersprungen.

Ist diese Bedingung nicht erfüllt, d.h. wurden keine Einträge gespeichert, dann wird eine Do-While-Schleife gestartet, in deren Schleifenkopf als Einstiegsbedingung ((Processing_Lexeme_Length) <> 0 And (Flag_Match_Found = 0)) geprüft wird. In dieser Schleife werden gemäß der Reihenfolge der Bedingungshierarchie zuerst Verarbeitungslexem (Processing_Lexeme), dann Suchalternative_3, Suchalternative_2 und Suchalternative_1 abgearbeitet. Die Vorgehensweise ist immer die gleiche. Der Variable “rs“ wird das Ergebnis einer SQL-SELECT-Anweisung übergeben, welche über die ersten drei Spalten der Tabelle “Rules“ selektiert und nur jene Datensätze speichert, die in der Spalte “F3“ (Synthetic lexical base) den Wert des reduzierten Verarbeitungslexems (“Processing_Lexeme“) bzw. der reduzierten Suchalternativen und zusätzlich einen Eintrag in der Spalte “F1“ (Cluster) enthält.
Nachfolgend wird mittels einer If-Anweisung überprüft, ob die Anzahl der gespeicherten Datensätze ungleich “0“ ist.
In diesem Fall, d.h. befinden sich Datensätze in der Variable “rs“, erhält die Variable “Match_Lexeme“ den Wert der jeweiligen reduzierten Suchalternative bzw. des reduzierten Verarbeitungslexems, und die Variable “Flag_Match_Found“ erhält den Wert “1“, welches dazu führt, dass kein weiterer Schleifeneintritt erfolgt. Im anderen Fall passiert nichts, und der Programmcode des Schleifenrumpfs wird fortgesetzt. Wurde keine der If-Bedingungen erfüllt, erfolgen folgende Wertzuweisungen im Else-Zweig der Suchalternative_1, und solange die Schleifeneintrittsbedingung erfüllt ist, beginnt die Do-While-Schleife erneut.

Processing_Lexeme = Right(Processing_Lexeme, (Processing_Lexeme_Length - 1))
Searchalternative_3 = Right(Searchalternative_3, (Processing_Lexeme_Length - 1))
Searchalternative_2 = Right(Searchalternative_2, (Processing_Lexeme_Length - 1))
Searchalternative_1 = Right(Searchalternative_1, (Processing_Lexeme_Length - 1))
Processing_Lexeme_Length = Processing_Lexeme_Length - 1

Beispiel:
Processing_Lexeme_Length = Len(“present“) →“7“
Processing_Lexeme = “present“
Set rs = CurrentDb.OpenRecordset(" SELECT Rules.F1, Rules.F2, Rules.F3 FROM Rules WHERE (((Rules.F3)= ""#" & “CCVCVCt“ & """))")→“0“
If rs.RecordCount <> 0 →“False“ → Else

Erster Schleifendurchlauf
Set rs = CurrentDb.OpenRecordset(" SELECT Rules.F1, Rules.F2, Rules.F3 FROM Rules WHERE (((Rules.F3) = """ & “Present“ & """)) And ((Not (Rules.F1)="""")) ")→“0“
If (rs.RecordCount <> 0 And (“Present“ <> “Present“)) →“False“ → Else

Set rs = CurrentDb.OpenRecordset(" SELECT Rules.F1, Rules.F2, Rules.F3 FROM Rules WHERE (((Rules.F3) = """ & “CCVCent“ & """)) And ((Not (Rules.F1)=""""))")
If rs.RecordCount <> 0 →“False“ → Else

Set rs = CurrentDb.OpenRecordset(" SELECT Rules.F1, Rules.F2, Rules.F3 FROM Rules WHERE (((Rules.F3) = """ & “CCVCVnt“ & """)) And ((Not (Rules.F1)=""""))")
If rs.RecordCount <> 0 →“False“ → Else

Set rs = CurrentDb.OpenRecordset(" SELECT Rules.F1, Rules.F2, Rules.F3 FROM Rules WHERE (((Rules.F3) = """ & “CCVCVCt“ & """)) And ((Not (Rules.F1)=""""))")
If rs.RecordCount <> 0 →“False“ → Else

Processing_Lexeme = Right(Processing_Lexeme, (7 - 1)) →“resent“
Searchalternative_3 = Right(Searchalternative_3, (7 - 1)) →“CVCent“
Searchalternative_2 = Right(Searchalternative_2, (7 - 1)) →“CVCVnt“
Searchalternative_1 = Right(Searchalternative_1, (7 - 1)) →“CVCVCt“
Processing_Lexeme_Length = 7 - 1

Wenn die Do-While-Schleife durchlaufen ist, ohne vorzeitig abzubrechen, und somit weder das bis auf den letzten Buchstaben reduzierte Verarbeitungslexem noch die reduzierten Suchalternativen eine Übereinstimmung mit den Registereinträgen ergaben (Flag_Match_Found ungleich 1), wird der letzte Buchstabe des Suchlexems mittels der “DLookup“-Funktion in ein Jokerzeichen umgewandelt und der Variablen “Match_Lexeme“ zugewiesen (Suchalternative 4 exit).

Beispiel nach dem letzten erfolglosen Schleifendurchlauf:

If (Flag_Match_Found = 0) →“True“
Match_Lexeme = DLookup("[Typ]", "[Alphabet]", "[Character]=""" & Right(“Present“, 1) & """)→“C“

Im letzten Schritt wird die für die Ausgabe verantwortliche Abfrage “Que_Rules“ mit einer SQL-Anweisung überschrieben. Diese SQL-Anweisung selektiert alle Spalten der Tabelle “Rules“ von “F1“ bis “F17“ und wählt jene Zeilen aus, bei denen der Eintrag in der Spalte “F3“ (Synthetic lexical base) dem Wert der Variablen “Match_Lexeme“ entspricht und ein Eintrag in der Spalte “F1“ (Cluster) existiert. Desweiteren wird jeder selektierten Spalte als Überschrift der Wert aus der gleichnamigen globalen Variable zugewiesen (siehe 4.2).

```
‘ Suchalternativen generieren

Function longest_match(Processing_Lexeme As String) As String

Dim rs As DAO.Recordset
Dim Searchalternative_1 As String
Dim Searchalternative_2 As String
Dim Searchalternative_3 As String
Dim Processing_Lexeme_Length As Integer
Dim Match_Lexeme as String
Processing_Lexeme_Length = Len(Processing_Lexeme)

Do While Processing_Lexeme_Length >  1

If (1 < Len(Searchlexeme)) Then
Searchalternative_1 = Searchalternative_1 + DLookup("[Typ]", "[Alphabet]", "[Character]=""" &
Left(Processing_Lexeme, 1) & """)
Else
End If

If (Processing_Lexeme_Length > 2 And (2 < Len(Searchlexeme))) Then
Searchalternative_2 = Searchalternative_2 + DLookup("[Typ]", "[Alphabet]", "[Character]=""" &
Left(Processing_Lexeme, 1) & """)
Else
End If
```

```
If (Processing_Lexeme_Length > 3 And (3 < Len(Searchlexeme))) Then
Searchalternative_3 = Searchalternative_3 + DLookup("[Typ]", "[Alphabet]", "[Character]=""" &
Left(Processing_Lexeme, 1) & """")
Else
End If
Processing_Lexeme = Right(Processing_Lexeme, (Processing_Lexeme_Length - 1))
Processing_Lexeme_Length = Processing_Lexeme_Length - 1

Loop

'Auffüllen mit Buchstaben

If (1 < Len(Searchlexeme)) Then
Searchalternative_1 = Searchalternative_1 + Right(Searchlexeme, (1))
Else
Searchalternative_1 = ""
End If

If (2 < Len(Searchlexeme)) Then
Searchalternative_2 = Searchalternative_2 + Right(Searchlexeme, (2))
Else
Searchalternative_2 = ""
End If

If (3 < Len(Searchlexeme)) Then
Searchalternative_3 = Searchalternative_3 + Right(Searchlexeme, (3))
Else
Searchalternative_3 = ""
End If

'Prüfen der Suchalternativen

Processing_Lexeme_Length = Len(Searchlexeme)
Processing_Lexeme = Searchlexeme

Set rs = CurrentDb.OpenRecordset(" SELECT Rules.F1, Rules.F2, Rules.F3 FROM Rules
          WHERE (((Rules.F3)= ""#" & Searchalternative_1 & """))")

If rs.RecordCount <> 0 Then
Match_Lexeme = "#" & Searchalternative_1
Flag_Match_Found = 1
Else

     Do While ((Processing_Lexeme_Length) <> 0 And (Flag_Match_Found = 0))

        Set rs = CurrentDb.OpenRecordset(" SELECT Rules.F1, Rules.F2, Rules.F3 FROM Rules
                  WHERE (((Rules.F3) = """ & Processing_Lexeme & """)) And ((Not (Rules.F1)=""""))  ")

        If (rs.RecordCount <> 0 And (Searchlexeme <> Processing_Lexeme)) Then
        Match_Lexeme = Processing_Lexeme
        Flag_Match_Found = 1
        Else

        Set rs = CurrentDb.OpenRecordset(" SELECT Rules.F1, Rules.F2, Rules.F3 FROM Rules
                  WHERE (((Rules.F3) = """ & Searchalternative_3 & """)) And ((Not (Rules.F1)=""""))")

           If rs.RecordCount <> 0 Then
```

```
        Match_Lexeme = Searchalternative_3
        Flag_Match_Found = 1
        Else

        Set rs = CurrentDb.OpenRecordset(" SELECT Rules.F1, Rules.F2, Rules.F3 FROM Rules
                 WHERE (((Rules.F3) = """ & Searchalternative_2 & """)) And ((Not (Rules.F1)=""""))")

           If rs.RecordCount <> 0 Then
           Match_Lexeme = Searchalternative_2
           Flag_Match_Found = 1
           Else

           Set rs = CurrentDb.OpenRecordset(" SELECT Rules.F1, Rules.F2, Rules.F3 FROM Rules
                    WHERE (((Rules.F3) = """ & Searchalternative_1 & """)) And ((Not (Rules.F1)=""""))")

              If rs.RecordCount <> 0 Then
              Match_Lexeme = Searchalternative_1
              Flag_Match_Found = 1
              Else

              Processing_Lexeme = Right(Processing_Lexeme, (Processing_Lexeme_Length - 1))
              Searchalternative_3 = Right(Searchalternative_3, (Processing_Lexeme_Length - 1))
              Searchalternative_2 = Right(Searchalternative_2, (Processing_Lexeme_Length - 1))
              Searchalternative_1 = Right(Searchalternative_1, (Processing_Lexeme_Length - 1))
              Processing_Lexeme_Length = Processing_Lexeme_Length - 1

              End If         '(Searchalternative_1)
           End If          '(Searchalternative_2)
        End If           '(Searchalternative_3)
     End If            '(Processing_Lexeme)

    Loop

     If (Flag_Match_Found = 0) Then
     Match_Lexeme = DLookup("[Typ]", "[Alphabet]", "[Character]=""" & Right(Searchlexeme, 1) & """")
     Else
     End If     '(Searchalternative_4 exit)
 End If  '(#Searchalternative_1)

CurrentDb.QueryDefs("Que_Rules").sql = _
" SELECT Rules.F1 as [" & F1 & "], Rules.[F2] as [" & F2 & "], Rules.F3 as [" & F3 & "], Rules.F4 as [" & F4 & "],
Rules.F5 as [" & F5 & "], Rules.F6 as [" & F6 & "], Rules.F7 as [" & F7 & "], Rules.F8 as [" & F8 & "], Rules.F9 as ["
& F9 & "], Rules.F10 as [" & F10 & "], Rules.F11 as [" & F11 & "], Rules.F12 as [" & F12 & "], Rules.F13 as [" & F13
& "], Rules.F14 as [" & F14 & "], Rules.F15 as [" & F15 & "],Rules.F16 as [" & F16 & "],Rules.F17 as [" & F17 & "]
FROM Rules WHERE (((Rules.F3) = """ & Match_Lexeme & """)) And ((Not (Rules.F1)=""""))  "

End Function
```

Quellcode 4.04: “longest match“-Funktion

4.2.5 Zusatzfunktion für den Datenimport

Nach dem Betätigen des Importbuttons auf der Benutzeroberfläche (siehe 4.1.1, Abb. 4.01) wird mittels der Accessfunktion “DoCmd.Close acForm“ das Startformular geschlossen. Dies ist erforderlich, damit die dahinterliegende SQL-Abfrage und die darunterliegende Tabelle problemlos verändert werden können.
Eine Variable “Path“ vom Typ String wird definiert, in welcher der durch den Benutzer über den Microsoft Programmbrowser ausgewählte Dateipfad gespeichert wird. Mittels der Accessfunktion “DateiOeffnen“ wird der Microsoft Programmbrowser geöffnet, welcher standardmäßig den Pfad zu den “Eigenen Dateien“ aktiviert.

Nun kann der Benutzer einen beliebigen Pfad auswählen, in dem sich entweder seine excelbasierte Alphabettabelle oder sein excelbasiertes Register mit den Regellexemen der jeweiligen Sprache befindet. Diese Entscheidung trifft der Benutzer im Formularfeld “Select“. Nun wird dessen Wert mit dem Befehl “Me.Select“ abgefragt. Eine If-Anweisung überprüft, ob es sich bei dem Wert um “Alphabet“ handelt.

Wenn dies der Fall ist, wird die alte Alphabet-Tabelle entfernt (DoCmd.RunSQL "DROP TABLE Alphabet") und die neue mittels der Funktion “DoCmd.TransferSpreadsheet acImport“ importiert. In den Parametern dieser Funktion steht der neue Name der Tabelle “Alphabet“, der Pfad, wo die zu importierende Datei zu finden ist, und “True“ oder “False“ dafür, ob die Spalten-Namen der Exceldatei als Tabellenspaltennamen übernommen werden sollen.

Andernfalls (nicht Alphabet, sondern Register) wird die alte Tabelle “Head“ (in dieser Tabelle werden die Spaltenbeschriftungen der Lexemregister gespeichert) entfernt. Durch die Funktion “DoCmd.TransferSpreadsheet acImport“ wird nun die neue “Head“-Tabelle nach der oben beschriebenen Struktur importiert. Der zusätzliche Parameter “A1:Q1“ beschränkt den Import auf die erste Zeile der 17 Spalten “A“ bis “Q“.
Desweiteren wird die alte Tabelle “Rules“ aus der Datenbank entfernt und anschließend die neue mit der gleichen Prozedur beginnend ab Zeile zwei importiert.

Nun wird das “Start“-Formular, welches der Benutzeroberfläche entspricht, erneut geöffnet und das “Import“-Formular geschlossen. Nach Abschluss dieser Funktion befinden sich die neuen Datenbestände in der Datenbank.

```
Private Sub Import_Click()
DoCmd.Close acForm, "Start"
Dim Path As String
Path = DateiOeffnen("C:Eigene Dateien", "Datei öffnen")

If Me.Select = "Alphabet" Then

'Alphabet
DoCmd.RunSQL "DROP TABLE Alphabet"
DoCmd.TransferSpreadsheet acImport, acSpreadsheetTypeExcel97, "Alphabet", Path, True

Else

'Head
DoCmd.RunSQL "DROP TABLE Head"
DoCmd.TransferSpreadsheet acImport, acSpreadsheetTypeExcel97, "Head", Path, False, "A1:Q1"

'Rules
DoCmd.RunSQL "DROP TABLE Rules"
DoCmd.TransferSpreadsheet acImport, acSpreadsheetTypeExcel97, "Rules", Path, False, "A2:Q"

End If

DoCmd.OpenForm "Start", acNormal
DoCmd.Close acForm, "Import"
End Sub
```

Quellcode 4.05: Importfunktion

5. Anwendungsmöglichkeiten

Das Ergebnis dieser Forschungsarbeit, d.h. die technische Umsetzung des SMIRT-Algorithmus in einer benutzerfreundlichen Programmversion, kann in folgenden Bereichen hilfreiche Verwendung finden. Dem Sprachwissenschaftler liefert es auf Knopfdruck strukturelle Informationen über das Beugungsverhalten eines bestimmten Verbs in einem untersuchten Verbalsystem. Desweiteren erhält er statistische Informationen über das abweichende Verhalten (es gibt Ausnahmen, aber die Mehrheit sind regelmäßig flektierende Lexeme) von Verben im jeweiligen Verbalsystem. Dieser schnelle Informationszugang kann einem Sprachlernenden ebenfalls das Lernen einer Sprache bzw. das Lernen von Verben einer Sprache erleichtern. Der Unterricht für Sprachlernende könnte hierdurch anschaulicher gemacht werden.
Während herkömmliche Übersetzungsprogramme lediglich ihre Listeneinträge durchgehen und nur, sofern sich das gesuchte Lexem darin wiederfinden lässt, ein Resultat liefern, macht sich dieser Algorithmus, mit Hilfe der durch Data Mining generierten inhomogenen Cluster, die statistische Wahrscheinlichkeit von Klassenzuordnungen zu Nutze.
Eine Sprache wie auch deren Verben unterliegt einem Veränderungsprozess. Daher kommen neue Verben in das Verbalsystem einer Sprache, für die der Algorithmus ebenfalls in der Lage ist, richtige Aussagen zu treffen.

Beispiele (eingedeutschte Verben):
googeln → homC *eln: ~eln, ~elte, ge~elt* (*googeln, googelte, gegoogelt*)
downloaden → basC2 *den: ~den, ~det, ~dete, ge~det* (*downloaden, downloadet, downloadete, gedownloadet*)

Weitere Vorteile ergeben sich aus dem in 4.1.1 beschriebenen Programmkonzept. Da die drei Bereiche (MVC) entkoppelt sind, lassen sich diese problemlos durch andere ersetzen. Beispielsweise könnte eine webbasierte Realisierung wie folgt aussehen:
View: HTML-Seite im Webbrowser
Controller: Umsetzung der Programmlogik in HTML, PHP oder Java
Model: beliebige SQL-Datenbank (mySQL, Oracle, Microsoft SQL Server)

Literaturverzeichnis

Alpar, Paul; Niedereichholz, Joachim: Data Mining im praktischen Einsatz. Braunschweig u.a.: Vieweg u.a. 2000.

Ester, Martin; Sander, Jörg: Knowledge discovery in databases. Techniken und Anwendungen. Berlin: Springer 2000.

Hesse, Wolfgang; Mayr, Heinrich C.: Modellierung in der Softwaretechnik. Informatik Spektrum 31 (2008) 377-393.

Holl, Alfred: Romanische Verbalmorphologie und relationentheoretische mathematische Linguistik. Axiomatisierung und algorithmische Anwendung des klassischen Wort-und-Paradigma-Modells.
Tübingen: Niemeyer 1988.

Holl, Alfred: The inflectional morphology of the Swedish verb with respect to reverse order: analogy, pattern verbs and their key forms.
Arkiv för nordisk filologi 116 (2001) 193-220.

Holl, Alfred: Nutzen und Tücken von Analogieschlüssen in der Verbalmorphologie: Rückläufige Ähnlichkeit als tertium comparationis in ausgewählten romanischen und germanischen Sprachen.
In: Heinemann, S.; Bernhard, G.; Kattenbusch, D. (ed.): Roma et Romania. Festschrift für Gerhard Ernst zum 65. Geburtstag.
Tübingen: Niemeyer 2002, 151-167.

Holl, Alfred: Datenanalyseverfahren der Informatik (Data Mining) als Grundlage einer didaktischen Darstellung der französischen Verbalmorphologie.
In: Bernhard, G.; Kattenbusch, D.; Stein, P. (ed.): Namen und Wörter. Festschrift für Josef Felixberger zum 65. Geburtstag.
Regensburg: Haus des Buches 2003, 107-119.

Holl, Alfred; Behrschmidt, André; Kühn, Alexander: Rückläufige Register zur russischen und deutschen Verbalmorphologie. Aufbereitung mit Datenanalyseverfahren der Informatik (Data Mining).
Regensburg: Roderer 2004
[= Studia et exempla linguistica et philologica, Series V: Lexica, Tom. 4].

Holl, Alfred; Pavlidis, Stilianos; Urban, Reinhard: Rückläufiges Wörterbuch zur alt- und neugriechischen Verbalmorphologie. Aufbereitung mit Datenanalyseverfahren der Informatik (Data Mining).
Regensburg: Roderer 2006
[= Studia et exempla linguistica et philologica, Series V: Lexica, Tom. 5].

Holl, Alfred; Maroldo, Sara; Urban, Reinhard: The inflectional morphologies of the Swedish noun, the Swedish verb and the English verb. Reverse dictionaries based upon data mining methods.
Växjö: Växjö University Press 2007
[= Mathematical modelling in physics, engineering and cognitive sciences, vol. 12].

Holl, Alfred; Suljic, Ivan: Rückläufiges Wörterbuch zur kroatischen Verbalmorphologie. Aufbereitung mit Datenanalyseverfahren der Informatik (Data Mining).
Regensburg: Roderer 2009, Forthcoming
[= Studia et exempla linguistica et philologica, Series V: Lexica, Tom. 6].

Internet-Links

Middendorf, Stefan; Singer, Reiner; Heid, Jörn: Programmierhandbuch und Referenz für die JavaTM-2-Plattform, Standard Edition, 3. Auflage 2002.
http://www.dpunkt.de/java/Programmieren_mit_Java/Oberflaechenprogrammierung/40.html

Zimmer, Wolfgang: Datenbanklösung Access Datenbanksysteme, 2006.
http://www.access-hilfe.de/

Microsoft Hilfe und Support, 2004.
http://support.microsoft.com/kb/875287/de

Microsoft Office, 2008.
http://office.microsoft.com/de-at/access/HA012288741031.aspx