Deutsches
Forschungszentrum
für Künstliche
Intelligenz GmbH

Document
D-95-07

Morphic - Plus
Ein morphologisches Analyseprogramm für die deutsche Flexionsmorphologie und Komposita-Analyse

Ottmar Lutzy

July 1995

Deutsches Forschungszentrum für Künstliche Intelligenz GmbH

Postfach 20 80
67608 Kaiserslautern, FRG
Tel.: + 49 (631) 205-3211
Fax: + 49 (631) 205-3210

Stuhlsatzenhausweg 3
66123 Saarbrücken, FRG
Tel.: + 49 (681) 302-5252
Fax: + 49 (681) 302-5341

Deutsches Forschungszentrum für Künstliche Intelligenz

The German Research Center for Artificial Intelligence (Deutsches Forschungszentrum für Künstliche Intelligenz, DFKI) with sites in Kaiserslautern and Saarbrücken is a non-profit organization which was founded in 1988. The shareholder companies are Atlas Elektronik, Daimler-Benz, Fraunhofer Gesellschaft, GMD, IBM, Insiders, Mannesmann-Kienzle, Sema Group, Siemens and Siemens-Nixdorf. Research projects conducted at the DFKI are funded by the German Ministry for Research and Technology, by the shareholder companies, or by other industrial contracts.

The DFKI conducts application-oriented basic research in the field of artificial intelligence and other related subfields of computer science. The overall goal is to construct systems with technical knowledge and common sense which - by using AI methods - implement a problem solution for a selected application area. Currently, there are the following research areas at the DFKI:

- ☐ Intelligent Engineering Systems
- ☐ Intelligent User Interfaces
- ☐ Computer Linguistics
- ☐ Programming Systems
- ☐ Deduction and Multiagent Systems
- ☐ Document Analysis and Office Automation.

The DFKI strives at making its research results available to the scientific community. There exist many contacts to domestic and foreign research institutions, both in academy and industry. The DFKI hosts technology transfer workshops for shareholders and other interested groups in order to inform about the current state of research.

From its beginning, the DFKI has provided an attractive working environment for AI researchers from Germany and from all over the world. The goal is to have a staff of about 100 researchers at the end of the building-up phase.

Dr. Dr. D. Ruland
Director

Morphic - Plus
Ein morphologisches Analyseprogramm für die deutsche Flexionsmorphologie und Komposita-Analyse

Ottmar Lutzy

DFKI-D-95-07

This work has been supported by a grant from The Federal Ministry of Education, Science, Research and Technology (FKZ ITWM-94-01).

ISSN 0946-0098

Morphic - Plus
Ein morphologisches Analyseprogramm für die deutsche Flexionsmorphologie und Komposita-Analyse

Ottmar Lutzy

Juni 1995

Morphic - Plus

Ein morphologisches Analyseprogramm
für die deutsche Flexionsmorphologie
und Komposita-Analyse

Ottmar Lutzy

Juni 1995

Zusammenfassung

Die morphologische Analyse nimmt bei der Verarbeitung geschriebener natürlicher Sprache eine überaus wichtige Position ein. Neben der Grundform- und Wortartbestimmung mit Hilfe einer morphologischen Analyse wird vor allem die Ermittlung von Flexionsinformationen verstanden. Die Wichtigkeit dieses Teilprozesses für die Textanalyse ist von der zu bearbeitetenden Sprache abhängig. Die deutsche Sprache gehört zu einer Sprachklasse mit freier Wortstellung, d. h. die grammatikalische Information für ein Wort wird fast ausschließlich durch die Analyse der Flexionsform des Wortes und nicht durch dessen Stellung im Satz gewonnen.

Mit *Morphic-Plus* steht ein Lemmatisierungsmodul zur Verfügung, mit dessen Hilfe flektierte Wortformen auf deren kanonische Wortstämme zurückgeführt werden kann. In der deutschen Sprache sind häufig zusammengesetzte Wörter, sogenannte Komposita, zu finden. Insbesondere in der wissenschaftlichen Literatur ist die Bildung neuer Worte aus bekannten Teilworten gängig. *Morphic-Plus* bietet daher neben einer Flexionsmorphologie auch eine Kompositaanalyse für zusammengesetzte Wörter. Bei der Analyse der Komposita durch *Morphic-Plus* wird die Wortbildung dahingehend eingeschränkt, daß ein Kompositum nur aus *Nomen*, *Verben* und *Adjektiven* gebildet werden kann.

In dieser Arbeit wird sowohl auf die Organisation des zugrundeliegenden Lexikons und dessen Aufbau als auch auf die Algorithmen von *Morphic-Plus* und deren Implementierung eingegangen. Das Lexikon und die darin verschlüsselten Information ist für die Analyse von zentraler Bedeutung. Im Lexikon, *Morphic-Lex* genannt, ist das Wissen über die Sprache kodiert.

Inhaltsverzeichnis

Abbildungsverzeichnis

1 Motivation

Der Anstoß zur Entwicklung von *Morphic-Plus* kam einerseits aus dem Projekt **ALV** (Automatisches Lesen und Verstehen) und dessen Nachfolgeprojekt **OMEGA** (Office Mail Expert for Goal-Directed Analysis) des Deutschen Forschungszentrum für Künstliche Intelligenz (DFKI), das bereits für die morphologische Analyse von Geschäftsbriefen das Analyseprogramm *MORPHIX* einsetzt und andererseits durch das Kooperationsprojekt **INCA** (Indexierung, Klassifikation und Archivierung von strukturierten Dokumenten) zwischen dem DFKI und der Daimler-Benz AG. *Morphic-Plus* versteht sich als eigenständige Weiterentwicklung des morphologischen Analyseprogramms *MORPHIX*, das von Wolfgang Finkler und Günter Neumann [FN86] [FN91] am Lehrstuhl Künstliche Intelligenz - Wissensbasierte Systeme der Universität des Saarlandes in Saarbrücken entwickelt wurde.
Morphic-Plus wurde wegen der Anforderung einer leichten Portierbarkeit auf unterschiedliche Rechnersysteme in *C* (*ANSI C*) implementiert. Bisher wird *Morphic-Plus* auf *SUN-SPARC*-Workstations und *NEXT*-Rechnern eingesetzt. Der Einsatz auf PC's im 32-Bit Modus ist möglich. Da *MORPHIX* in *Common-Lisp* implementiert ist, verspricht man sich durch die *C*-Implementierung von *Morphic-Plus* einen Zeitvorteil bei der Analyse.

2 Allgemeine Beschreibung von *Morphic-Plus*

Bei *Morphic-Plus* handelt es sich um ein System zur Flexionsmorphologie, es wird keine Derivationsanalyse durchgeführt. Das bedeutet, daß eine Wortform, die durch Derivation aus einer anderen Form abgeleitet werden kann, als Stamm im Lexikon eingetragen sein muß. So kann beispielsweise das Wort *Schreiber* nicht analysiert werden, da nur *schreib (schreiben)* als Verb im Lexikon eingetragen ist und die Endung *er* keine gültige Verwendung bei Verben darstellt. Dieses Vorgehen erzwingt zwar mehr Lexikoneinträge, erhöht aber auch die Sicherheit eines korrekten Analyseergebnisses, da 'unmögliche' Zusammensetzungen vermieden werden.
Die Lemmatisierung wird in *Morphic-Plus* in zwei Schritten durchgeführt. Im ersten Schritt, der *Segmentierung*, wird das zu analysierende Wort in mögliche Tripel aus Präfix, Wortstamm und Suffix aufgespalten. In einem zweiten Schritt, der *Verifikation*, wird überprüft, welche dieser Abspaltungen zu einem 'korrekten' Analyseergebnis führt. Eine solche Segmentierung führt im Falle von unregelmäßigen Stämmen nicht zu einem kanonischen Stamm. In *Morphic-Plus* werden zwei Wege verfolgt, zum einen wird mit einer Umlautreduktion gearbeitet und zum anderen sind irreguläre Stämme im Lexikon gespeichert und mit den kanonischen Wortstämmen in Beziehung gesetzt (z. B.: das unregelmäßige Verb 'ging' und 'geh'). Dieses Vorgehen ist für das Deutsche eine effiziente Methode, da im Deutschen nur etwa 200 unregelmäßige Verben vorkommen.
Morphic-Plus wurde um eine Komponente zur Bearbeitung von zusammengesetzten Wörtern, die sogenannten *Komposita* erweitert. Die Zerlegung in einzelne Teilworte erfolgt erst, wenn die bisherige morphologische Analyse zu keinem Ergebnis gelangt ist. Ein Kompositum wird nur dann als analysiert angesehen, wenn alle Teilwörter von *Morphic-Plus* analysiert werden konnten.

3 Erläuterungen zum Lexikon

Das verwendete Lexikon enthält die kanonischen Wortstämme mit einer Kodierung des Wortes in einer 6 bzw. 7 stelligen Zahl. Sowie weitere Zusatzinformationen, die für die morphologische Analyse von Bedeutung sind. Aus Effizienzgründen wird mit Hilfe einer schnellen Hash-Funktion [KH93] auf die Lexikoneinträge zugegriffen (siehe Kapitel C). Im folgenden wird auf die verschiedenen Wortklassen und die verwendeten Kodierungen näher eingegangen. In der ersten bzw. den ersten zwei Stellen (bei 6 bzw. 7 stelliger Kodierung) wird die Wortart verschlüsselt. Dies sind im einzelnen:

Schlüssel	Wortart	Länge der Kodierung
1	für Nomen	6-stellig
2	für Verben (regelmäßige und unregelmäßige)	6-stellig
3	für Adjektive	6-stellig
4	für Possessivpronomen	6-stellig
5	für Verbpräfixe	6-stellig
6	für Adverbien	6-stellig
7	für Partikel	6-stellig
8	für Konjunktionen	6-stellig
9	für Subjunktionen	6-stellig
10	für Präpositionen	7-stellig
11	für Relativpronomen	7-stellig
12	für bestimmte Artikel	7-stellig
13	für Reflexivpronomen	7-stellig
14	für Personalpronomen	7-stellig
15	für Interrogationspronomen	7-stellig
16	für Sonderzeichen	7-stellig
17	für Frageadverbien	7-stellig
18	für unbestimmte Artikel	7-stellig
19	für Kardinalzahlen	7-stellig
20	für Stopwörter	7-stellig
21	für Namen (Firmen-, Produkt- oder Eigen-)	7-stellig
22	für Abkürzungen	7-stellig

Abkürzungen, die mit einem Punkt enden, sind in einer gesonderten Datei abgespeichert. Auf diese Abkürzungen kann nicht mit Hilfe der sonst verwendeten Hash-Funktionen zugegriffen werden. Der Zugriff erfolgt über eine sequentielle Suche (Einzelheiten zum Lexikonzugriff ist in Kapitel C zu finden). Schlüsselworte und der Aufbau des Lexikons werden in einem späteren Unterpunkt beschrieben.

3.1 Die Kodierung der Nomeneinträge

Für die Kodierung von Nomen reicht eine 6-stellige Zahl aus.
Die 1. Stelle hat den Wert '1' und bezeichnet das Wort als Nomen.
Die 2. Stelle kann die Werte '1', '2' oder '3' annehmen und kodiert den Genus des Nomens; '1' = maskulin, '2' = feminin und '3' = neutrum.
Die 3. Stelle nimmt die Werte '0' oder '1' an. Sie gibt an ob bei der Pluralbildung eine Umlautung stattfindet ('1' = Umlautung).
Die 4. Stelle nimmt die Werte '0', '1', ..., '9' an. Sie bestimmt die Singularklasse des Nomens.
Die 5. und 6. Stelle kann die Werte '00', '01', ..., '13' annehmen. Mit diesen Werten wird die Pluralklasse bestimmt.

Die Klassifizierung ist dabei an [Sch72] *angelehnt.* Die Werte '0' und '00' bei der 4. und 5.-6. Stelle werden bei Nomen ohne kanonischen Stamm bzw. bei unregelmäßiger Pluralbildung verwendet wie z. B.: *Atlas* und *Atlanten.* Die Klasseneinteilung stützt sich für den Singular auf die Genitiv- und Dativendungen des Wortes und für den Plural auf die Nominativ- und Dativendungen. In den beiden folgenden Tabellen werden die Singular- und Pluralklassen mit den charakteristischen Endungen aufgelistet.

Singular - Klassen

Klasse	Endung im Genitiv	Endung im Dativ	Beispiele
0	-	-	kein Singular
1	-	-	Decke
2	s	-	Engel
3	es	-/e	Gesetz
4	s/es	-/e	Zweig
5	ses	-/se	Ergebnis
6	ens	-/en	Herz
7	ns	n	Wille
8	n	n	Waise
9	en	en	Graf

Plural - Klassen

Klasse	Endung im Nominativ	Endung im Dativ	Beispiel
00	-	-	kein Plural
01	-	-	Wagen, Faden
02	s	s	Auto
03	n	n	Einbusse
04	en	en	Herz, Werkstatt
05	nen	nen	Herzogin
06	-	n	Kapitel, Hammer
07	e	en	Keim, Raum
08	se	sen	Kenntnis
09	er	ern	Licht, Mann
10	ien	ien	Material
11	sen	sen	Kirmes
12	ten	ten	Bau
13	te	ten	Klima, -

Hier nun einige Beispiele für die Nomenkodierungen:

haus 131309
ansatz 111307
atlas 110500 plural atlanten
atlanten 110001 stamm atlas

3.2 Die Kodierung der Verbeinträge

Die Kodierung von Verben ist in einer 6-stelligen Zahl realisiert.
Die 1. Stelle hat den Wert '2' und kennzeichnet das Wort als Verb.
Die 2. und 3. Stelle ist die Kodierung für den Verbtyp und die Unterklasse. Der Verbtyp wird mit 'modulo 12' und die Unterklasse mit 'div 12' berechnet. Der Wert '00' ist bei Verben mit einem festen abtrennbaren Präfix eingetragen. Daraus resultiert der Verbtyp '0'.
Die 4. Stelle gibt das Hilfsverb zur Bildung der zusammengesetzten Zeiten und die Präfixart an. Zahl gerade = Hilfsverb 'sein', Zahl ungerade = Hilfsverb 'haben', Zahl ≥ 2 Verb mit festem Präfix und Zahl < 2 abtrennbarer Präfix.
Die 5. Stelle hat den Wert '0'.
Die 6. Stelle kodiert ob ein Präfix abtrennbar ist und dessen Länge. z. B.: '7' ein Präfix der Länge 7 kann abgetrennt werden.

Die Unterteilung der Verben in Klassen erfolgt wie in *SUTRA* ([Bus83] Seite 48) beschrieben. Dabei werden 11 Verbtypen verwendet. Die Verben werden zusätzlich noch nach Verbgrundformen und deren Unterklassen unterteilt. Diese sind im einzelnen: Die Verbgrundformen *VGFA*, *VGFB*, *VGFC* und *VGFD* sowie *PPRF* für Partizip Perfekt und *SOUND1..4* für das Hilfsverb 'sein'. Es gibt 8 Unterklassen für *VGFA* (*VGFA1..7* und *VGFA+*), 6 für *VGFB* (*VGFB1..6*) und 4 für *VGFC* (*VGFC1..4*). Die Charakteristika der einzelnen Unterklassen ist den folgenden Tabellen zu entnehmen.

Die Verbtypen

Verbtyp	1.P.Sg. Präsens	2./3.P.Sg. Präsens	Imperativ Sg.	Imperfekt	Konjunktiv 2	Beispiel
0						abtret
1	VGFA	VGFA	VGFA	VGFA	VGFA	hol
2	VGFA	VGFA	VGFA	VGFC	VGFA	kenn
3	VGFA	VGFA	VGFA	VGFC	VGFC	bleib
4	VGFA	VGFA	VGFA	VGFC	VGFD	beginn
5	VGFA	VGFB	VGFA	VGFC	VGFC	blas
6	VGFA	VGFB	VGFA	VGFC	VGFD	fahr
7	VGFA	VGFB	VGFB	VGFC	VGFD	geb
8	VGFB	VGFB	VGFA	VGFA	VGFA	woll
9	VGFB	VGFB	VGFA	VGFC	VGFA	könn
10	VGFB	VGFB	VGFA	VGFC	VGFD	wiss
11	SOND1	SOND2 SOND3	VGFA	VGFA	VGFA	sei

Unterklassen von VGFA

Unterklasse	Beschreibung	Beispiel
A1	sonst	rasier
A2	Stammende 'd' oder 't'	arbeit
A2	Stammende 'm' oder 'n' wenn kein 'l' oder 'r' vorausgeht	atm
A3	Modalverb 'sollen'	soll
A4	Stammende 's', 'z' oder 'x'	ras
A5	Stammende 'el'	sammel
A6	Stammende 'er'	erneuer
A7	'sein' und 'tun'	tu
A+	Stamm mit elidiertem 'e' auf 'ern' oder 'eln'	handl

Unterklassen von VGFB

Unterklasse	Flex.Endung 2. P.Sg.	Flex.Endung 3. P.Sg.	Beispiel
B1	st	t	gib
B2	-	-	birst
B3	st	-	brät
B4	t	t	iß
B5	st	d	wir
B6	t	-	muß

Unterklassen von VGFC

Unterklasse	Beschreibung	Beispiel
C1	sonst	fuhr
C2	Stammende 's', 'z' oder 'x'	wuchs
C3	Stammende 'd' oder 't'	trat
C4	Stammende 'e'	mochte

3.3 Die Kodierung der Adjektiveinträge

Für die Kodierung der Adjektive wird auch eine 6-stellige Zahl verwendet. Die Einteilung in Klassen erfolgt nach dem Schema von [HB81].
Die 1. Stelle hat den Wert '3' und bezeichnet das Wort als Adjektiv.
Die 2. Stelle kann die Werte '0', '1', ..., '4' annehmen. '0' nicht graduierbar, '1' Superlativendung 'st', '2' Superlativendung 'est', '3' anderer Stamm in Komperativ oder Superlativ, '4' anderer Stamm auch im Positiv.
Die 3. Stelle nimmt die Werte '0', '1' oder '2' an. '0' keine Umlautung bei Steigerung, '1' Umlautung muß erfolgen, '2' Umlautung kann erfolgen.
Die 4. Stelle nimmt die Werte '0', '1' oder '2' an. '0' keine Elision, '1' Elision muß erfolgen, '2' Elision kann erfolgen.
Die 5. Stelle kann die Werte '0' oder '1' annehmen. '0' endet nicht auf 'e' in prädikativer Stellung, '1' endet mit 'e' in prädikativer Stellung.
Die 6. Stelle nimmt die Werte '1', '2' ..., '7' an. Die Ziffern bedeuten folgende Untergruppen: '1' = A1, '2' = A2, '3' = A3, '4' = B1, '5' = B2, '6' = B3 und '7' = C1.

Die Einteilung der Adjektive erfolgt in 3 Hauptklassen 'A', 'B' und 'C' mit folgenden Bedeutungen:

Klasse 'A' : Attributiv und prädikativ verwendbar
Unterklasse 'A1' : flektierbar und graduierbar (z.B.: wichtig)
Unterklasse 'A2' : flektierbar nicht graduierbar (z.B.: ledig)
Unterklasse 'A3' : nicht flektierbar, nicht graduierbar (z.B.: allerlei)
Klasse 'B' : nur attributiv gebraucht
Unterklasse 'B1' : flektierbar und graduierbar (z.B.: vorder)
Unterklasse 'B2' : flektierbar nicht graduierbar (z.B.: jetzig)
Unterklasse 'B3' : nicht flektierbar, nicht graduierbar (z.B.: keinerlei)
Klasse 'C' : nur prädikativ gebraucht
Unterklasse 'C1' : nicht flektierbar, nicht graduierbar (z.B.: schuld)

3.4 Die Kodierung der Einträge für Possessivpronomen

Für die Kodierung der Possessivpronomen wird eine 6-stellige Zahl verwendet.
Die 1. Stelle hat den Wert '4' und kennzeichnet somit das Wort als Possessivpronomen.
Die 2. Stelle kann die Werte '0' oder '2' annehmen. '0' keine Elision, '2' Elision kann erfolgen.
Die Stellen 3 bis 6 haben den Wert '0' und sind ohne weitere Bedeutung.

3.5 Die Kodierung von Verbpräfixen

Die Verbpräfixe werden in einer 6 stelligen Zahl kodiert.
Die 1. Stelle hat den Wert '5' und ist die Kodierung für Verbpräfixe.
Die 2. Stelle kann die Werte '0' und '1' annehmen.
Die Stellen 3 bis 6 haben den Wert '0' und sind ohne weitere Bedeutung.

3.6 Die Kodierung von Adverbien

Für die Kodierung der Adverbien wird mit einer 6-stelligen Zahl realisiert.
Die 1. Stelle hat den Wert '6' und bezeichnet das Wort als Adverb.
Die 2. Stelle nimmt die Werte '0' oder '1' an.
Die Stellen 3 bis 6 haben den Wert '0' und sind ohne weitere Bedeutung.

3.7 Die Kodierung von Partikeln (Konjunktoren)

Die Partikel, Konjunktoren werden mit einer 6-stelligen Zahl kodiert.
Die 1. Stelle hat den Wert '7' und repräsentiert die Wortart Partikel / Konjunktor.
Die 2. Stelle kann die Werte '0' oder '1' annehmen.
Die Stellen 3 bis 6 haben den Wert '0' und sind ohne weitere Bedeutung.

3.8 Die Kodierung von Partikeln (Koordinations Konjunktoren)

Die Partikel werden mit einer 6-stelligen Zahl kodiert.
Die 1. Stelle hat den Wert '8' und repräsentiert die Wortart Partikel.
Die 2. Stelle kann die Werte '0' oder '1' annehmen.
Die Stellen 3 bis 6 haben den Wert '0' und sind ohne weitere Bedeutung.

3.9 Die Kodierung von Partikeln (Subjunktoren)

Die Partikel, Subjunktoren werden mit einer 6-stelligen Zahl kodiert.
Die 1. Stelle hat den Wert '9' und repräsentiert die Wortart Partikel / Subjunktor.
Die 2. Stelle kann die Werte '0' oder '1' annehmen.
Die Stellen 3 bis 6 haben den Wert '0' und sind ohne weitere Bedeutung.

3.10 Die Kodierung von Partikeln (Präposition)

Die Partikel, Präpositionen werden mit einer 7-stelligen Zahl kodiert.
Die 1. und 2. Stelle haben den Wert '10' und repräsentieren die Wortart Partikel / Präposition.
Die 3. Stelle kann die Werte '0' oder '1' annehmen.
Die 4. Stelle kann die Werte '1', '2', ..., '6' annehmen; kodiert den Folgekasus und wird in der folgenden Tabelle erörtert.
Die Stellen 5 bis 7 haben den Wert '0' und sind ohne weitere Bedeutung.

Kodierung der 4. Stelle

Kodierung	Folgekasus	Beispiel
1001000	Genitiv	anfangs
1002000	Dativ	aus
1003000	Akkusativ	bei
1004000	Genitiv, Dativ, Akkusativ	außer
1005000	Genitiv, Dativ	inklusive
1006000	Dativ, Akkusativ	an

3.11 Die Kodierung von Relativpronomen

Die Relativpronomen werden mit einer 7-stelligen Zahl kodiert.
Die 1. und 2. Stelle nehmen den Wert '11' an und kennzeichnen damit das Wort als Relativpronomen.
Die Stellen 3 bis 7 haben den Wert '0' und sind ohne weitere Bedeutung.

3.12 Die Kodierung bestimmter Artikel

Bestimmte Artikel werden mit einer 7-stelligen Zahl kodiert.
Die 1. und 2. Stelle haben den Wert '12' und bezeichnen das Wort als bestimmter Artikel.
Die Stellen 3 bis 6 haben den Wert '0' und sind ohne weitere Bedeutung.
Die 7. Stelle kann die Werte '0', '1' oder '2' annehmen. '0' besagt, daß der Artikel sowohl im Singular wie im Plural verwendet wird. '1' bedeutet eine Verwendung im Singular und '2' im Plural.

3.13 Die Kodierung von Reflexivpronomen

Die Reflexivpronomen werden mit einer 7-stelligen Zahl kodiert.
Die 1. und 2. Stelle nehmen den Wert '13' an und kennzeichnen damit das Wort als Reflexivpronomen.
Die Stellen 3 bis 7 haben den Wert '0' und sind ohne weitere Bedeutung.

3.14 Die Kodierung von Personalpronomen

Die Personalpronomen werden mit einer 7-stelligen Zahl kodiert.
Die 1. und 2. Stelle haben den Wert '14' und repräsentieren die Wortart Personalpronomen.
Die 3. Stelle kann die Werte '0' oder '1' annehmen.
Die Stellen 4 bis 7 haben den Wert '0' und sind ohne weitere Bedeutung.

3.15 Die Kodierung von Interrogativpronomen

Die Interrogativpronomen werden mit einer 7-stelligen Zahl kodiert.
Die 1. und 2. Stelle nehmen den Wert '15' an und kennzeichnen damit das Wort als Interrogativpronomen.
Die Stellen 3 bis 7 haben den Wert '0' und sind ohne weitere Bedeutung.

3.16 Die Kodierung von Sonderzeichen

Die Sonderzeichen oder Satzzeichen werden mit einer 7-stelligen Zahl kodiert.
Die 1. und 2. Stelle nehmen den Wert '16' an und kennzeichnen damit das Wort als Sonderzeichen oder Satzzeichen.
Die Stellen 3 bis 7 haben den Wert '0' und sind ohne weitere Bedeutung.

3.17 Die Kodierung von Partikeln(Frageadverbien)

Die Partikel, Frageadverbien werden mit einer 7-stelligen Zahl kodiert.
Die 1. und 2. Stelle haben den Wert '17' und repräsentieren die Wortart Partikel / Frageadverb.
Die Stellen 3 bis 7 haben den Wert '0' und sind ohne weitere Bedeutung.

3.18 Die Kodierung unbestimmter Artikel

Die unbestimmten Artikel werden mit einer 7-stelligen Zahl kodiert.
Die 1. und 2. Stelle haben den Wert '18' und bezeichnen das Wort als unbestimmter Artikel.

Die Stellen 3 bis 6 haben den Wert '0' und sind ohne weitere Bedeutung.
Die 7. Stelle kann die Werte '0' oder '1' annehmen.

3.19 Die Kodierung von Kardinalzahlen

Die Kardinalzahlen werden mit einer 7-stelligen Zahl kodiert.
Die 1. und 2. Stelle nehmen den Wert '19' an und kennzeichnen das Wort als Kardinalzahl.
Die Stellen 3 bis 6 haben den Wert '0' und sind ohne weitere Bedeutung.
Die 7. Stelle kann die Werte '0', '1', ..., '3' annehmen. '0' bedeutet, daß die Ordinalzahl unregelmäßig gebildet wird. '1' besagt eine Ordinalzahlbildung durch anhängen des Buchstaben 't', '2' durch anhängen von 'st' gebildet. Bei '3' wird kein Zeichen angehängt.

3.20 Die Kodierung von Stop-Wörtern

Stop-Wörter werden ebenfalls mit einer 7-stelligen Zahl kodiert.
Die 1. und 2. Stelle nehmen den Wert '20' an.
Die restlichen Stellen haben den Wert '0' und sind ohne weitere Bedeutung.
Bei den Stop-Wörtern handelt es sich um eine künstliche Wortart, die für eine Analyse ohne Bedeutung sind.

3.21 Die Kodierung von Namen

Namen werden ebenfalls mit einer 7-stelligen Zahl kodiert.
Die 1. und 2. Stelle nehmen den Wert '21' an.
Die restlichen Stellen haben den Wert '0' und sind ohne weiter Bedeutung.
Unter Namen werden hier zum Beispiel Eigen-, Produkt- oder Firmennamen verstanden.

3.22 Die Kodierung von Abkürzungen

Abkürzungen werden ebenfalls mit einer 7-stelligen Zahl kodiert.
Die 1. und 2. Stelle nehmen den Wert '22' an.

Die restlichen Stellen haben den Wert '0' und sind ohne weiter Bedeutung.
Es wird zwischen Abkürzungen mit und ohne Punkt am Wortende unterschieden. Die Abkürzungen ohne Punkt sind im *MORPHIC-Lexikon* eingetragen und werden mit Hilfe von Hash-Tabellen verwaltet. Die mit Punkt sind in einer sequentiellen ASCII-Datei, z. B. *abkuerzung* organisiert.

3.23 Schlüsselworte und Kodierungen des Lexikons

Im folgenden wird näher auf die verwendeten Schlüsselworte und die Zusammmensetzungen der Kodierung der Lexikoneinträge eingegangen. Es wird auf eine Reihe von Schlüsselworte zurückgegriffen um das morphologische Wissen neben der Kodierung in den 6-/7-stelligen Zahlen in den Lexikoneinträgen zu speichern. Dabei werden folgende Schlüsselworte verwendet:
POS, KOM, SUP, ELISION, STAMM, PLURAL, MODALVERB, HILFSVERB, ORDINAL, ORDINAL-B, FLEXION, NOM, AKK, DAT, GEN, FEM, MAS, NTR, PL, SG, SOND1, SOND2, SOND3, SOND4, A5, A6, VGFA+, VGFAA1, VGFAA3, VGFAA4, VGFAA7, VGFA+A5, VGFA+A6, VGFBB1, VGFBB2, VGFBB3, VGFBB4, VGFBB5, VGFBB6, VGFCC1, VGFCC2, VGFCC3, VGFCC4, VGFD, VTYP1, VTYP6, VTYP8, VTYP9, VTYP10, VTYP11, 1, 2, 2A, 3, PPRF, PAUX, MULTIPLE.

Mit Hilfe der Schlüsselworte: SOND1, SOND2, SOND3, SOND4, A5, A6, VGFA+, VGFAA1, VGFAA3, VGFAA4, VGFAA7, VGFA+A5, VGFA+A6, VGFBB1, VGFBB2, VGFBB3, VGFBB4, VGFBB5, VGFBB6, VGFCC1, VGFCC2, VGFCC3, VGFCC4, VGFD, VTYP1, VTYP6, VTYP8, VTYP9, VTYP10, VTYP11 werden Verben, wie in den Tabellen des Abschnitts 3.2 zusehen, kodiert, falls die 6-stellige Ziffernkodierung nicht ausreicht.

Die Worte POS, KOM und SUP finden bei Adjektiven Verwendung und kennzeichnen das Wort als Possitiv, Komparativ oder Superlativ.

Mit Hilfe des Wortes ELISION wird das Weglassen des Buchstaben 'e', die sogenannte Elision markiert.

Das Wort STAMM kennzeichnet unregelmäßige Verben, Nomen oder Adjektive. Nach diesem Schlüsselwort steht der zugrundeliegende Wortstamm.

Besitzt ein Nomen eine unregelmäßige Pluralbildung wird dies mit dem Schlüsselwort PLURAL gekennzeichnet. Es gibt 2 Verwendungen des Schlüsselwortes, die später erklärt werden.

Die Schlüsselworte NOM, AKK, DAT, GEN, FEM, MAS, NTR, PL und SG sind Abkürzungen für den Kasus: Nominativ, Akkusativ, Dativ oder Genitiv, den Genus: Feminin, Maskulin oder Neutrum sowie für Plural oder Singular.

PPRF und PAUX bei Verben kennzeichnen das Wort als Partizip Perfekt oder bestimmen das Hilfsverb bei der Partizipbildung.

Die Worte ORDINAL, ORDINAL-B finden bei der Kodierung von Ordinal- bzw. Kardinalzahlen Anwendung.

Das Schlüsselwort MULTIPLE kennzeichnet eine weitere Kodierungsvariante des Lexikoneintrages.

Im folgenden werden einige Beispiele von Lexikoneinträgen angegeben und näher erklärt:

```
modus 110100 PLURAL modi
modi 110001 STAMM modus PLURAL
```

Das Wort 'MODUS' ist ein Nomen im Singular mit unregelmäßiger Pluralbildung, der Plural ist 'MODI'. Das Wort 'MODI' ist ein Nomen im Plural und der Stamm bzw. der Singular ist 'MODUS'.

```
meld 225100 PPRF gemeldet
```

Das Wort 'MELD' (Infinitiv 'MELDEN') besitzt 'GEMELDET' als Partizip Perfekt.

```
mich l-f ich 1400000 FLEXION 1 SG AKK MULTIPLE 1300000 FLEXION 1 SG AKK
```

Bei dem Wort 'MICH' handelt es sich zum einen um ein Personalpronomen mit der Flexion 1. Person Singular Akkusativ und im zweiten Fall (durch 'MULTIPLE' verbunden) um ein Reflexivpronomen mit der Flexion 1. Person Singular Akkusativ.

```
gelt 231100 VGFBB3 gilt VGFCC3 galt VGFD gaelte PPRF gegolten
gilt STAMM gelt VGFB
```

Das Wort 'GELT' (Infinitiv 'GELTEN') ist ein unregelmäßiges Verb. Die verschiedenen Zeiten und Konjugationen stehen nach den Schlüsselworten 'VGFBB3' oder 'VGFCC3' etc. Dies sind Kodierungen wie sie in den Tabellen des Kapitels 3.2 beschrieben sind. Der Eintrag 'GILT' verweißt mit Hilfe des Schlüsselwortes 'STAMM' auf den Verbstamm 'GELT'.

```
best STAMM gut SUP
```

Bei dem Wort 'BEST' handelt es sich um den Superlativ des Adjektivs 'GUT'.

```
besser 273100 PPRF gebessert VGFA+A6 bessr MULTIPLE STAMM gut KOM
```

Das Wort 'BESSER' kann ein Verb mit dem Partizip Perfekt 'GEBESSERT' sein, das durch Elision zu 'BESSR' wird oder als weitere alternative Wortart kann es sich um den Komparativ des Adjektivs 'GUT' handeln.

```
kg 2200000 Kilogramm, Kommanditgesellschaft
```

Bei 'KG' handelt es sich um eine Abkürzung. Hinter der 7-stelligen Kodierung steht als Text die Erklärung. Alternativen sind durch Kommata getrennt.

```
sei HILFSVERB VTYP11 VGFAA7 SOND1 bin SOND2 bist SOND3 ist SOND4 sind
VGFCC1 war VGFD waere PPRF gewesen PAUX sei
```

Das Hilfsverb 'SEIN' ist vom Verbtyp 11 und gehört zur Unterklasse VGFAA7. Die Konjugationsformen im Präsens werden mit 'SOND1'...'SOND4' bezeichnet. Das Partizip Perfekt ist 'GEWESEN' und das Hilfverb zur Partizipbildung ist 'SEI' (durch das Schlüsselwort 'PAUX' gekennzeichnet).

```
muess MODALVERB VTYP9 VGFAA4 VGFBB6 muss VGFCC4 musste PPRF gemusst PAUX hab
```

Modalverben werden durch das Schlüsselwort 'MODALVERB' markiert. Die restliche Information ist analog der Beschreibung der Hilfsverben.

In den Lexikoneinträgen ist noch die Zeichenfolge 'l-f' zu finden. Sie kennzeichnet den Eintrag als Vollform Eintrag. Diese Information wird nur von *Morphix* ausgewertet. Da sowohl *Morphic-Plus* als auch —em Morphix mit den selben lexikalischen Quellen arbeiten ist dieses Schlüsselwort erforderlich.

4 *Morphic-Plus*-Analyse

Der verwendete Analysealgorithmus von *Morphic-Plus* lehnt sich an den zugrundeliegenden Algorithmus von *MORPHIX* [FN86] an. Es wurden einige Modifikationen eingebaut wie z. B. Ansätze zur Derivationsanalyse bei Adjektiven oder die Vorgehensweise zur Bestimmung der Wortstämme. Andererseits wurde auf eine 'Vollformanalyse' wie sie von *MORPHIX* durchgeführt wird verzichtet. Nachfolgend wird der von *Morphic-Plus* verwendete Analysealgorithmus beschrieben.

- Im ersten Schritt wird überprüft, ob das Eingabewort mit einem Punkt endet und somit eine Abkürzung sein könnte. Dieses Wort wird dann sequentiell in einer Datei mit Abkürzungen z. B. *abkuerzung.dat* gesucht. Ist diese Suche erfolgreich, findet die morphologische Analyse, wie sie im folgenden beschrieben wird, nicht statt. Das Ergebnis wird aufbereitet und das nächte Wort kann analysiert werden.

- Im 'ersten' Schritt der morphologischen Analyse wird das Eingabewort in Kleinbuchstaben transformiert. Dadurch gehen jedoch Informationen, wie z.B. die Großschreibung von Namen verloren. Aufgrund des verwendeten Lexikons ist diese Vorgehensweise erforderlich.

- Besteht das Eingabewort aus Ziffern, so wird die Eingabe direkt als Kardinalzahl klassifiziert.

Im folgenden wird nun die Behandlung von Worten (Buchstabenfolgen) in der morphologischen Analyse durch *Morphic-Plus* behandelt.

- Im nächsten Schritt wird das Wort auf Umlaute (ä, ö, ü) und 'Scharfes-S' (ß) hin untersucht und diese Buchstaben in (ae, oe, ue und ss) umgewandelt. Dies ist erforderlich, da zum einen im verwendeten Lexikon keine Umlaute oder 'Scharfes-S' vorhanden sind und zum anderen können mit einer 7-bit ASCII-Kodierung keine 'Sonderbuchstaben' dargestellt werden.

- In einem weiteren Schritt der Umlautbehandlung wird das Umlaut 'e' entfernt und in der Morphic-DS (Morphic-Datenstruktur) in einem 'Umlautflag' vermerkt. Dazu wird ein neues Listenelement mit diesen Daten erzeugt. Um z. B. unregelmäßige Nomen wie *Haus - Häuser* zu analysieren.

- Im nächsten Verarbeitungsschritt werden mögliche Präfixe vom Eingabewort abgespalten. Durch das Einfügen eines neuen Elementes (das veränderte Wort) hinter dem aktuell bearbeiteten Element in der Liste und dem Abarbeiten der gesamten Liste wird eine Rekursion bei der Präfixabspaltung erreicht.

- Analog zur Präfixabspaltung wird eine Suffixabspaltung vorgenommen. Damit auch mehrfach Präfixe abgearbeitet werden, wird die Präfixabspaltung wiederholt durchgeführt.

- Aufgrund der Rekursion bei der Präfix- und Suffixabspaltung können die zu analysierenden Wortalternativen mehrfach erzeugt werden. Diese Mehrfacheinträge werden in einem weiteren Verarbeitungsschritt aus der Liste entfernt.

- Danach wird für jedes Wort (Wortalternative) der Liste im Morphic-Lexikon nachgesehen, ob das Wort enthalten ist. Der Eintrag, falls vorhanden, wird in die Morphic-DS übernommen. Dazu werden diese Daten aufbereitet und in den entsprechenden Variablen der Morphic-DS gespeichert.

- Die Elemente der Liste, für die kein Lexikoneintrag gefunden wurde, werden anschließend aus der Liste entfernt.

- Im nächsten Verarbeitungsschritt wird der Lexikoneintrag weiter analysiert, d.h. es werden im Lexikon vermerkte Wortalternativen als neue Morphic-Einträge in der Liste erzeugt. Bei unregelmäßigen Verben wird für den Verbstamm (Imperativ) ein neuer Morphic-Eintrag generiert und mit den zugehörigen Informationen aus dem Lexikon initialisiert.

- Die gesamte Liste wird nochmals auf doppelte Elemente hin untersucht und solche daraus entfernt.

- Ein weiterer Schritt bildet die Ausgabe der Ergebnisse der morphologischen Analyse. In diesem Schritt werden die Ergebnisse aus dem Lexikon mit denen der Verarbeitungsschritte 'Umlautung', 'Präfix-' und 'Suffixabspaltung' kombiniert und ausgegeben.

- *Konnte das Eingabewort bisher mit den oben beschriebenen Mitteln nicht analysiert werden, wird eine* Komposita-Analyse *durchgeführt. Diese wird im Kapitel 5 näher beschrieben.*

Eine graphische Darstellung der oben dargestellten Verarbeitungsschritte ist in der Abbildung 1 zusehen.

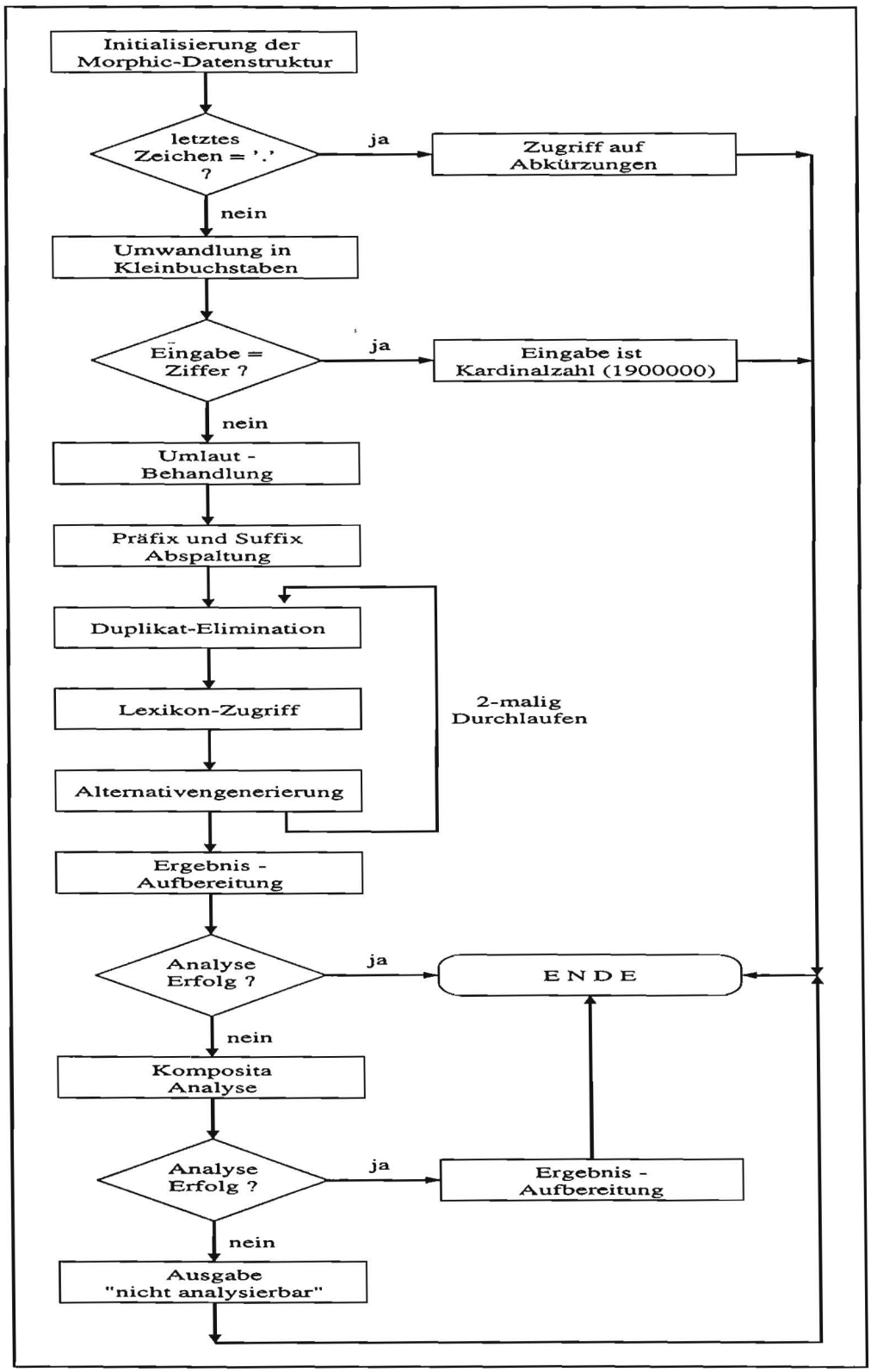

Abbildung 1: Kontrollfluß von Morphic-Plus

5 Komposita-Analyse

In der deutschen Sprache gibt es sehr viele Worte, die aus mehreren Teilwörtern zusammengesetzt werden. Diese sogenannten *Komposita* können aus Teilwörtern der unterschiedlichsten Wortarten (wie Nomen, Verben, Adjektiven etc.) gebildet werden. Wegen der Häufigkeit dieser Wörter ist es wünschenswert, auch diese morphologisch analysieren zu können. Für eine bestimmte Domäne wäre es denkbar, die wichtigsten Komposita in das zugrundeliegende Lexikon einzutragen. Damit kann man jedoch nicht die ganze Breite der Bildungsmöglichkeiten abdecken. Insbesondere sind in technischen Dokumenten Wortbildungen aus Komposita stark vertreten.
Eine Komposita-Behandlung, die auf der Aufspaltung des Wortes in einzelne Komponenten arbeitet gewährleistet die Beibehaltung eines kleinen (kompakten) Lexikons (das *Morphic-Plus* Lexikon hat ca. 11.000 Einträge). Bei der Teilkomponente der Analyse eines Textes in *Morphic-Plus* können sogenannte *Bindestrich-Komposita* (z. B. Hoch- und Tiefbau) nur bedingt analysiert werden. Bisher können nur solche Bindestrich-Komposita behandelt werden, bei denen im zweiten Wortteil ein Präfix abgespaltet werden kann, wie im vorherigen Kapitel beschrieben. Mit einer Komposita-Behandlung liessen sich auch Bindestrich-Komposita wie z. B. *Haus- und Hofhund* in die Textkomponenten *Haushund und Hofhund* überführen [1]. Für diesen Zweck ist eine Zerlegung des Wortes von links vorteilhaft, falls man keine komplette Komposita-Zerlegung des zweiten Wortes durchführen möchte [2].

5.1 Grundlegende Bemerkungen zu den Komposita

Der segmentierbare erste Bestandteil eines Kompositum wird als *Bestimmungswort* und der zweite mit *Grundwort* bezeichnet. Bei mehrteiligen Komposita verschiebt sich die Grenze zwischen beiden Teilen je nach Satzkonstruktion oder Bedeutung. Im allgemeinen bestimmt das Grundwort die Wortart der ganzen Zusammensetzung. Eine Ausnahme bilden die *Kopulativkomposita* wie sie im Duden [Dro84] beschrieben sind und deren Bedeutung auf dem ersten Teilwort liegt. Beispiele für 'normale' Komposita (*Determinativkomposita*) sind die *Autovermietung* oder die *Wohnungsbauförderung* mit den Bedeutungen: *Vermietung von Autos* bzw. *Förderung des Wohnungsbaus* bzw. *Förderung des Baus von*

[1] Zur Zeit noch nicht realisiert

[2] Bei der Textvorverarbeitung wäre in diesem Falle die Kompositaanalyse für dieses Wort anzustoßen und das längste Grundwort it dem ersten Wort z. B. Haus zu konkatenieren

Wohnungen. Ein Beispiel für Kopulativkomposita ist die *Himmelskuppel* mit der Bedeutung: *Himmel, der sich wie eine Kuppel wölbt.*
Das Bestimmungswort ist im Deutschen häufig mehrgliedrig, d. h. es läßt sich in weitere Teilwörter aufspalten. In der Fachliteratur spricht man dann von einer *Linksverzweigung.* Diese Komposita finden sich häufig in Verwaltungs- und Wissenschaftstexten sowie in der technischen Literatur. Als Beispiel hierfür möchte ich das Kompositum *Wohnungsbauförderungsgesetz* näher betrachten. Es läßt sich in

- das Bestimmungswort: *Wohnungsbauförderung-s-* und
- das Grundwort: *-gesetz* zerlegen.

Das Bestimmungswort läßt sich, wie in Abbildung 2 zu sehen ist, weiter aufspalten bis schließlich folgende Wortteilung erreicht wird:
Wohn- -ung-s -bau förder- -ung-s-
Wegen der Komplexität bei der Aufspaltung ist es wünschenswert, daß Wörter wie z. B. *Wohnung* oder *Förderung* im Lexikon eingetragen sind und die Endung *ung* nicht abgespalten wird. Bei der Komposita-Analyse von *Morphic-Plus* wird der letztere Ansatz verfolgt. Die Verbindungselemente *-s-* zwischen Woh-

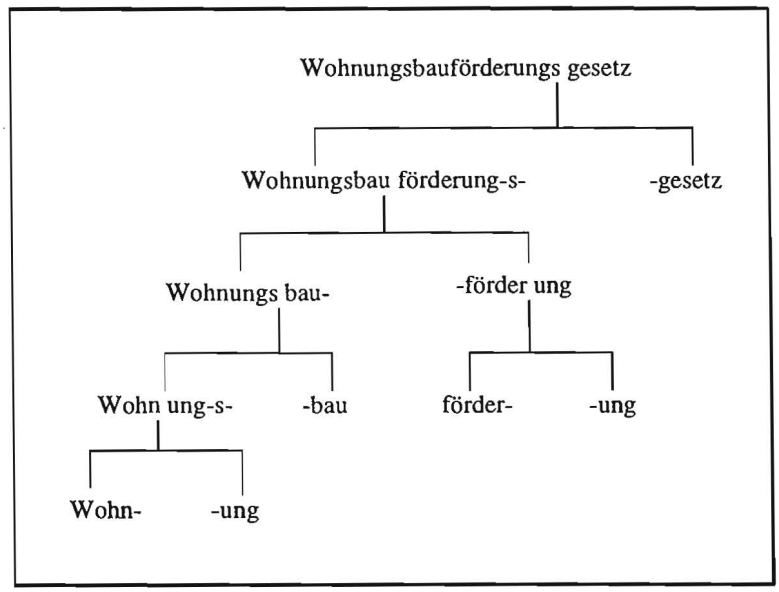

Abbildung 2: Linksverzweigung

nungsbauförderung- und -gesetz oder zwischen Wohnung- und -bau sind keine Flexionsendungen sondern Verbindungselemente, die sogenannten *Fugenelemete.* Weitere *Fugenelemete* sind beispielsweise *-e, -s, -n, -es, -en, -er etc..*

Die Fugenelemente werden in *Morphic-Plus* wie Endungen (Flexionsendungen) behandelt, da durch die gewählte Datenstruktur keine Unterscheidungsmöglichkeit gegeben ist.
Mehrgliedrige Grundwörter findet man häufig, wenn das Grundwort selbst ein häufig verwendetes Kompositum ist. Dies wird mit *Rechtsverzweigung* (siehe Abb. 3) bezeichnet. Zum Beispiel wird das Wort *Reiseschreibmaschine* in das Bestimmungswort *Reise-* und das Grundwort *-schreibmaschine* gespalten. Wobei sich Schreibmaschine wieder in Schreib- und -maschine auftrennen läßt.
Man findet auch Kombinationen dieser beiden Aufspaltungsvarianten, die

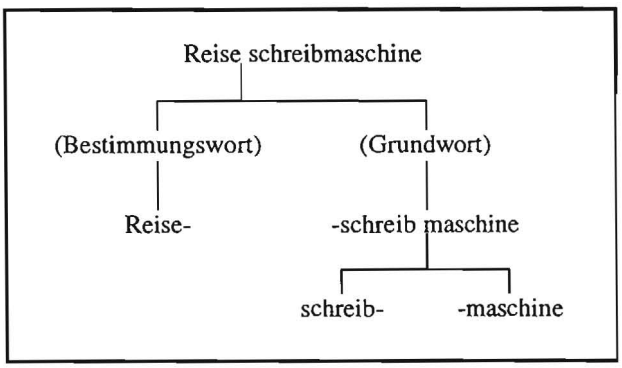

Abbildung 3: Rechtsverzweigung

Links-Rechts-Verzweigung, wie sie in Abbildung 4 für das Beispiel *Druckluftbremszylinder* dargestellt ist.
Wie die Aufspaltung letztendlich richtig ist oder auf welche Konstruktion die

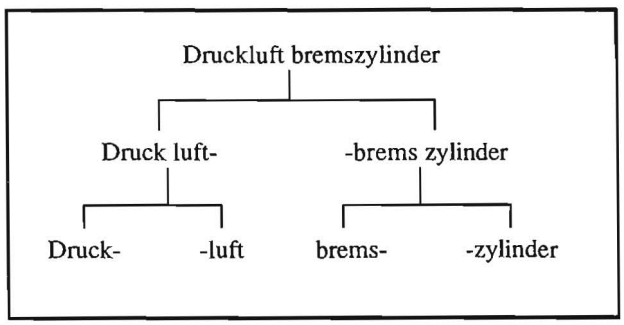

Abbildung 4: Links - Rechts - Verzweigung

komplexe Zusammensetzung zurückgeführt werden kann, ist selbst in der Fachliteratur nicht eindeutig geklärt. So haben Komposita im allgemeinen Sprachgebrauch Inhaltskomponenten angenommen, die sich mit einer Kompositа-Analyse nicht erfassen lassen. Beispielsweise ist ein *Lesebuch* ein Buch, in dem man liest, aber nicht jedes Buch, in dem man liest, ist ein Lesebuch.

Im verwendeten Lexikon zur morphologischen Analyse sind keine semantischen Informationen enthalten, deshalb ist es unmöglich Aussagen über semantische Analyseergebnisse zu machen. Eine syntaktische und grammatikaleische Analyse ist nicht möglich, da mit *Morphic-Plus* nur einzelne isolierte Worte ohne jeden Kontext analysiert werden.

5.2 Vorgehensweise bei der Komposita-Analyse

Für die Aufspaltung eines Kompositums kann man sich zwei Richtungen vorstellen. Zum einen versucht man von rechts beginnend das Kompositum in dessen Teilworte zu zerlegen. Die andere Möglichkeit ist von links startend das Wort aufzuspalten. Beide Vorgehensweisen werden in der morphologischen Analyse angewendet. Auf der *1. Morpholympics* im März 1994 [Ges94] [Len94] in Erlangen hatten die teilnehmenden Systeme beide Bearbeitungsrichtungen zur Komposita-Analyse verwendet. Beide Vorgehensweisen können als gleichwertig angesehen werden.[Ges94]
Das Grundwort des Kompositums bestimmt die Zugehörigkeit zu einer Wortart. Nur das Grundwort kann im Deutschen flektieren. In *Morphic-Plus* wird die Aufspaltung der Komposita von rechts durchgeführt. Bei *Morphic-Plus* handelt es sich um eine Flexionsmorphologie, d. h. es wird mit einer Suffixabspaltung begonnen, so daß sich auch bei der Komposita-Analyse die Vorgehensweise zur Abspaltung von rechts anbietet. Die Speicherung des Grundwortes bei der Zerlegung in der Wurzel des Analysebaumes hat die Bearbeitungsrichtung ebenso beeinflußt. In diesem Fall kann die Bestimmung der Wortart des Kompositums schnell durchgeführt werden.
Bei der Aufspaltung des Kompositums in Teilwörter kann man mehrere Strategien verfolgen. Zum einen wird versucht, das längstmögliche Wort abzuspalten (Strategie 1). Im anderen Ansatz kann man, sobald das abgespaltete Teilwort analysierbar ist, eine weitere Aufspaltung des 'Rest-Kompositums' durchführen (Strategie 2). In *Morphic-Plus* werden beide Strategien verfolgt um 'alle' Zerlegungsalternativen zu erzeugen.
Damit bekäme man mit der Eingabe *Reiseschreibmaschine* beim ersten Ansatz die beiden Teilwörter *Reise* und *Schreibmaschine*, vorausgesetzt beide Teilwörter sind im Lexikon vorhanden. Im zweiten Fall erhielte man drei Teilwörter *Reise*, *Schreib* und *Maschine*. Bei einer Kombination beider Ansätze können für ein Kompositum verschiedene Lesarten, wie in Abbildung 5 zu sehen ist, erzeugt werden. Zum Beispiel erhält man bei der Analyse des Eingabewortes *Bohrersatz* zwei Lesarten. Die erste Alternative *Bohr* und *Ersatz* bei der

längsten Abspaltung und als zweite Alternative *Bohrer* und *Satz*, wenn das abgespaltete Grundwort analysiert werden konnte und dannach noch versucht wird das Bestimmungswort weiter aufzutrennen.

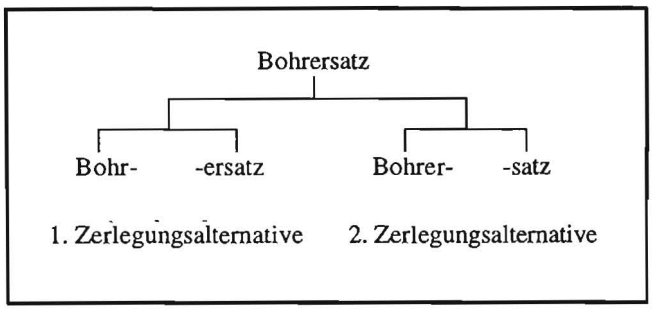

Abbildung 5: Verzweigungsalternativen

5.3 Besonderheiten bei der Komposita-Analyse

Da im Deutschen nur das Grundwort flektiert wird, bietet sich die Bearbeitungsrichtung der Komposita von rechts nach links an. In der Projektarbeit von Michael Stein [Ste94] wird *MORPHIX* unter anderem um eine Komposita-Analyse-Komponente erweitert. Hier wird die Zerlegung der Eingabewortes von links begonnen. Durch die Verwendung der Morphix-Analyseergebnisse zur Kompositabehandlung kommt es bei diesem Ansatz zu dem folgenden Problem: Im Lexikon sind nur die Grundformen der Wörter eingetragen. Wird nun ein Wort in seiner Grundform eingegeben, kann dieses Wort durch *MORPHIX* nicht analysiert werden, wenn durch die Kodierung des Wortes im Lexikon eine Flexionsendung vorgeschrieben ist. Ist zum Beispiel das Wort *Schreibmaschine* nicht im Lexikon eingetragen, kommt es bei der Abarbeitung zu der folgenden Situation: *Schreibmaschine* wird in die Teilworte *Schreib* und *Maschine* zerlegt. *Schreib* wird als Grundform des Verbs *schreiben* im Lexikon gefunden. Da jedoch keine Flexionsendung wie 'e', 'st' oder 'en' etc. abgespalten wurde, liefert *MORPHIX* das Ergebnis: `schreib konnte nicht analysiert werden.` Unter der Voraussetzung, daß alle Teilwörter analysiert werden müssen, würde die Komposita-Analyse nach dieser Vorgehensweise sehr oft scheitern und kein Ergebnis liefern.

Im allgemeinen wird auch bei der Pluralbildung durch Umlautung von Nomen-Komposita das Grundwort umgelautet. Ein Beispiel wäre *Schneeball - Schneebälle*. Es gibt natürlich auch Ausnahmen von Komposita, bei denen eine Umlautung im Bestimmungswort vorhanden ist und die Pluralform des Wortes

darstellt, das Kompositum als Ganzes aber im Singular ist. Die Pluralform des Wortes wird jedoch wie üblich durch die Pluralbildung im Grundwort erzeugt. Als Beispiel hierfür läßt sich *Häusermeer - Häusermeere* angeben.
Eine weitere Besonderheit bilden Komposita mit zwei gleichen aufeinanderfolgenden Konsonanten, die bei der Aufspaltung des Wortes in Teilworte zu zwei Konsonanten am Ende des ersten Wortteils und einem dritten Konsonanten am Anfang des zweiten Wortteils überführt werden (z. B. *Schiffahrt* $\Longrightarrow$ *Schiff* und *Fahrt*). Es gibt jedoch auch Komposita die mit drei gleichen aufeinanderfolgenden Konsonanten geschrieben werden (z. B. Sauerstoffflasche), die Regeln sind im Duden *Die deutsche Rechtschreibung* [Dro91] und im Duden *Die Grammatik* [Dro84] nachzulesen. Die Kompositaanalyse von *Morphic-Plus* ist in der Lage die zuvor beschriebenen Besonderheiten der deutschen Sprache bei der Zerlegung zu berücksichtigen.

5.4 Entwurf eines Komposita - Behandlungs - Algorithmus'

Hier wird die Vorgehensweise zur Abarbeitung von Komposita in *Morphic-Plus* beschrieben. Die Komposita-Behandlung wird angestoßen, wenn die morphologische Analyse für das eingegebene Wort erfolglos verlaufen ist.
Für die Komposita-Behandlung kann die oben angegebene *Morphic-Plus*-Datenstruktur verwendet werden. Bei der Abarbeitung des Eingabewortes entsteht ein Analysebaum, dessen Knoten Zerlegungsaltenativen enthalten, die durch die Aufspaltung des Wortes nach den beiden oben erwähnten Strategien (längstes bzw. kürzestes Teilwort) erzeugt werden.
Aufbauend auf dem ursprünglichen Eingabewort (Wort in 'Groß-Klein-Schreibung', mit Umlauten) werden folgende Verarbeitungsschritte durchgeführt:

- In einem ersten Schritt wird die für *Morphic-Plus* nötige Transformation in Kleinschreibung vorgenommen. Für das transformierte Eingabewort wird ein neues Listenelement erzeugt und über den *komp_word*-Pointer mit dem Top-Element (Eingabe-Element) verbunden und somit als Kompositum gekennzeichnet.

- Im nächsten Schritt findet eine Suffixabspaltung statt. Hiermit können die ersten Alternativen für die Komposita-Zerlegung erzeugt werden, da das Eingabewort mehrfach Endungen haben kann (z. B. -lichstere $\Longrightarrow$ -e, -ere, -stere und -lichstere; also 4 Alternativen). Diese Alternativen werden

ebenfalls als jeweils neues Element über den *komp_word*-Pointer in den zubildenden Analysebaum eingefügt.

Danach beginnt die eigentliche Zerlegung des Kompositums in Teilwörter.

- Im folgenden Schritt werden die beiden letzten Buchstaben des Wortes abgespalten [3].
- Mit dem so entstandenen Teilwort wird ein Lexikonzugriff durchgeführt um festzustellen, ob dieses Wort (als STAMM) im Lexikon eingetragen ist.
- Wird kein Eintrag gefunden wird ein weiterer Buchstaben am *linken Teilwort* (*LTW*) abgespalten und vorne an das *rechten Teilwort* (*RTW*) angehängt.
- Danach wird abermals ein Lexikonzugriff durchgeführt.

Die beiden letzten Schritte werden solange wiederholt, bis ein Eintrag für ein Teilwort gefunden wurde oder kein weiterer Buchstaben mehr an das Teilwort angehängt werden kann.
Nachfolgend wird das weitere Vorgehen beschrieben, wenn das *RTW* analysiert werden konnte. Ab diesem Punkt können verschiedene Alternativen verfolgt werden. Das *LTW* kann seinerseits weiter in *RTW* und *LTW* aufgespalten werden.

- Zunächst untersucht man, ob ein Fugenelemete abgespalten werden kann. Fugenzeichen nach [Dro84] sind z. B.: *-e, -s, -n, -es, -en, -er etc.*

Danach werden die obigen Schritte, Zeichen abspalten und Lexikonzugriff, wiederholt durchgeführt. Die gefundenen Teilworte werden mit dem *alt_word*-Pointer in einer linearen Liste als Ast des Analysebaumes abgespeichert.
Eine zweite Alternative stellt die Abspaltung des längstmöglichen *RTW*'s dar. Bei dieser Alternativen werden weiter Zeichen vom *LTW* abgespalten und an das *RTW* angehängt und Lexikonzugriffe durchgeführt. In diesem Falle findet keine Abspaltung eines Fugenelementes statt.
Das Verfahren endet, wenn das *LTW* (Eingabewort) vollständig abgearbeitet ist (Länge = 0). Durch die Rückwärtsverkettung kann die ganze verkettete Struktur (Baum) rekursiv abgearbeitet werden. Das gewählte Verfahren geht erst in

[3] im Deutschen ist die Mindestwortlänge 2 Buchstaben

die Breite, danach in die Tiefe. Um den Baum rekursiv durchlaufen zu können wird die Variable *ana_flag* der *Morphic-Plus* Datenstruktur mit Werten initialisiert, die nur für die Komposita-Analyse von Bedeutung sind. Diese Werte steuern das Durchlaufen durch den bereits erzeugten Baum und die Generierung aller möglichen Zerlegungsalternativen. Am Ende der Komposita-Behandlung liegt ein Baum vor, dessen Zweige die möglichen Aufspaltungsalternativen des Eingabewortes enthalten mit dem *Bestimmungswort* im 'Blatt' und dem *Grundwort* in der 'Wurzel' oder einem Verzweigungsknoten.
Zur besseren Veranschaulichung wird der oben beschriebene Kontrollfluß der Kompositaanalyse in Bild 6 graphisch dargestellt.
In Abbildung 7 wird der entstehende Analysebaum für das Eingabewort *Kindergarten* dargestellt, wie er nach der Zerlegung der Eingabe in mögliche Teilworte vorliegt. Das Wort *Kindergarten* ist im Lexikon eingetragen, deshalb gibt es auch einen Knoten mit dem *RTW Kindergarten* [4].
Die Behandlung von Umlauten und die bereits früher erwähnten Doppel- und Trippelkonsonanten erfordern eine aufwendige Unterroutine bei der Abspaltung eines Zeichens vom *LTW* und Anfügen an das *RTW*. So muß beispielsweise bei der Abspaltung eines Umlautes dieser zum einen in seine Zwei-Vokal-Umschreibung überführt werden und andererseits eine Umlautreduktion mit einer Alternativengenerierung vorgenommen werden. Bei Doppel- und Trippelkonsonanten müssen die letzten Zeichen des *LTW*'s und das erste Zeichen des *RTW*'s bei der Generierung eines neuen Teilwortes überprüft werden und gemäß den Rechtschreibregeln [Dro84] und [Dro91] ein weiterer Konsonant an das noch zu bearbeitende Wort angehängt werden.
Zur Generierung der möglichen Zerlegungsalternativen für die Ausgabe wird dieser Baum mehrmals durchlaufen. Je Durchlauf wird eine mögliche Lesart des Kompositums erzeugt. Der Weg beim Einsinken in den Baum bis zu den Blättern wird über die Werte der Variablen *ana_flag* gesteuert. Beim Aufstieg zur Wurzel des Baumes werden die relevanten Teilergebnisse der Zerlegung in eine *Ausgabeliste* übernommen und der Wert der Variablen *ana_flag* in den Knoten nach der folgenden Konvention verändert[5]:

- mit 5, falls der Knoten einen Nachfolger über *komp_word* und einen Nachfolger über *alt_word* mit einem *komp_word* Nachfolger besitzt
- mit 6, falls der Knoten einen Nachfolger über *komp_word* ~~besitzt und~~ dieser Zweig des Baumes noch nicht abgearbeitet wurde

[4] Dieses Beispiel wurde beim Entwurf der Komposita-Analyse in einer *Standalone-Version* berechnet

[5] Andere Werte werden in Kapitel 9 beschrieben

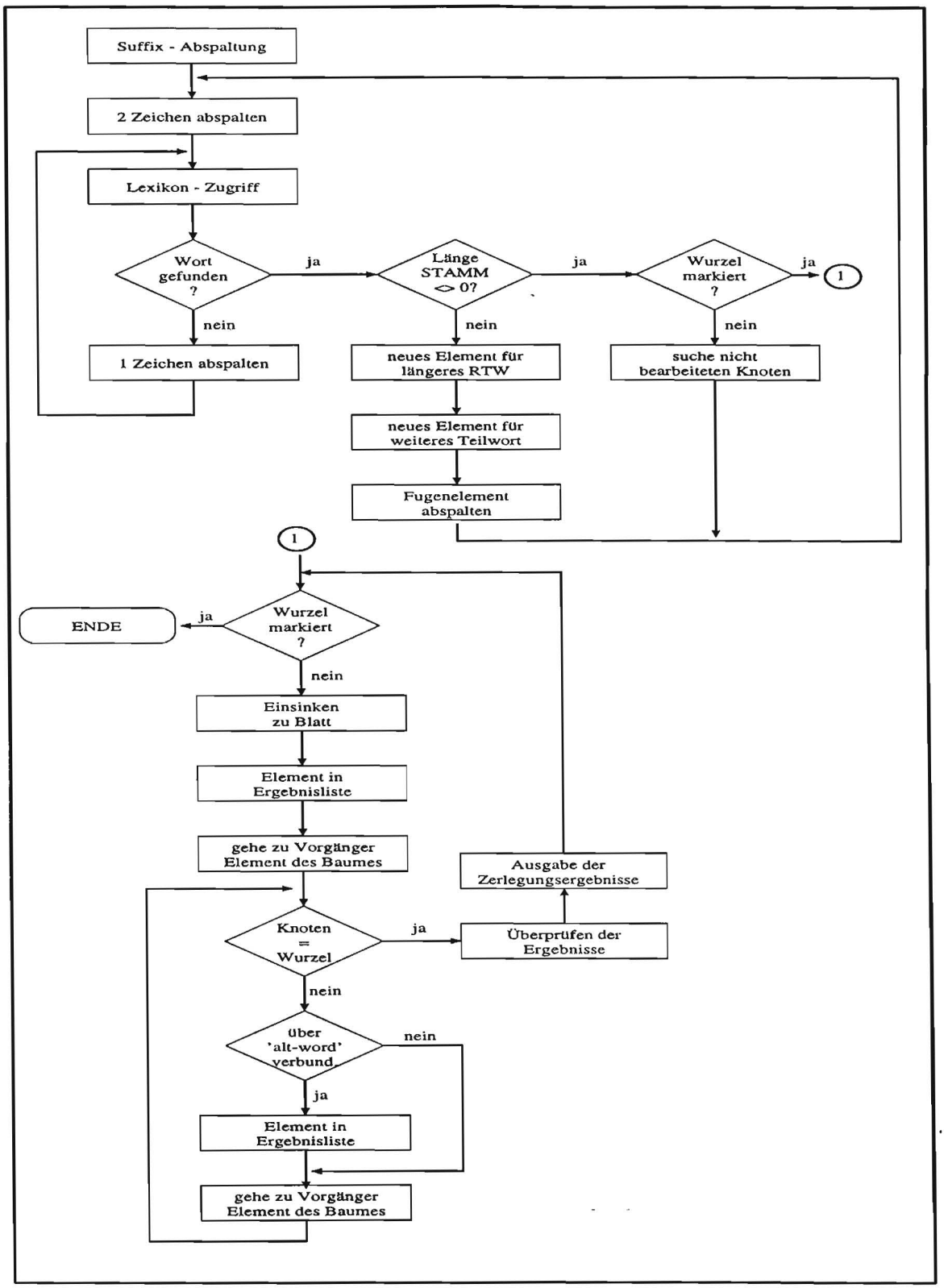

Abbildung 6: Kontrollfluß der Kompositazerlegung

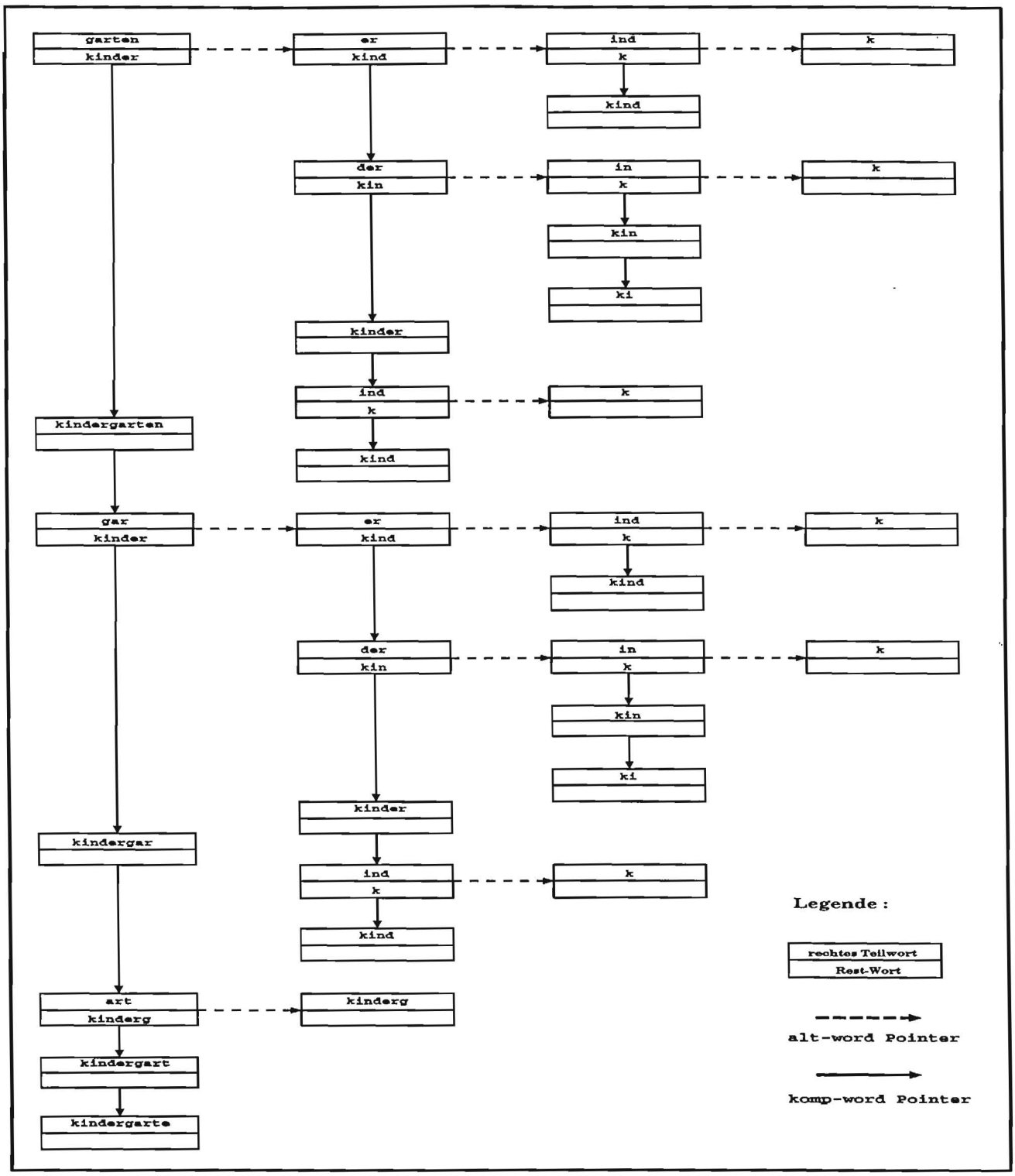

Abbildung 7: Komposita-Analysebaum nach der Zerlegung des Eingabewortes Kindergarten

- mit 7 wird der Knoten markiert, wenn die Daten in die *Ausgabeliste* übernommen wurden und die Variablen *ana_flag* der Nachfolger des Knoten mit dem Wert 7 initialisiert sind

Beim Aufstieg zur Wurzel werden nur die Knoten in die *Ausgabeliste* aufgenommen, die über den *alt_word* - Zeiger mit ihrem Vorgänger verbunden sind. Ist der Vorgänger des aktuellen Knotens über den *komp_word* - Zeiger verbunden, so wird der Vorgänger nicht in die Ergebnisliste aufgenommen. Als Ergebnis erhält man die folgenden Zerlegungsvarianten für das Beispiel *Kindergarten* aus Abbildung 7:

Alternative	1.	2.	3.	4. Teilwort
1.	k	ind	er	garten
2.	kind	er	garten	
3.	k	in	der	garten
4.	kin*	der	garten	
5.	ki	der	garten	
6.	kinder*	garten		
7.	k	ind	garten	
8.	kind	garten		
9.	kindergarten			
10.	k	ind	er	gar
11.	kind	er	gar	
12.	k	in	der	gar
13.	kin*	der	gar	
14.	ki	der	gar	
15.	kinder*	gar		
16.	k	ind	gar	
17.	kind	gar		
18.	kindergar*			
19.	kinderg*	art		
20.	kindergart*			
21.	kindergarte*			

Aufbauend auf diesen Lösungsalternativen sind alle Varianten zu entfernen, deren Teilworte nicht in die Wortklassen `Nomen`, `Verben` oder `Adjektive` fallen. Durch eine Funktion, die der Ausgabe vorgeschaltet ist, werden alle erzeugten Alternativen verworfen, die den obigen Anforderungen nicht entsprechen. Falls eine abgespaltene Endung nicht zu der ermittelten Wortart/-klasse paßt wird diese Wortart als Hypothese ausgegeben.
Als erstes können die Alternativen verworfen werden, die Teilworte enthalten,

die nicht im Lexikon enthalten sind. Das sind die Alternativen **4**, **6**, **13**, **15**, **18**, **19**, **20** und **21**. Die nicht analysierten Teilworte sind mit '*' markiert. Danach können die Alternativen deren Teilworte als *Stopwort* im *MORPHIX*-Lexikon eingetragen sind, entfernt werden. Das sind im obigen Beispiel die Alternativen **1**, **3**, **7**, **10**, **12** und **16** sie enthalten die Stopworte *K* und *KI*. Die Alternativen **11**, **14** und **17** enthalten das Wort *GAR*. Sie können entfernt werden, da das Wort als Koordinations-Konjunktion im Lexikon klassifiziert ist. Ebenso können die Alternativen **2** und **5** entfallen, da 'ungültige' Wortarten enthalten sind (*ER* ist ein Personalpronomen und *IND* ist eine Abkürzung). Die Alternative **9** kann in diesem Beispiel auch entfallen, da sie nur aus einem Teilwort besteht. Durch ein Vorschalten von *Morphic-Plus* wäre für das Eingabewort *Kindergarten* die Komposita-Analyse nicht gestartet worden (Hier wurde die Komposita-Analyse isoliert gestartet!). Somit bleibt bei diesem Beispiel als einzige Lesart die Variante **8** übrig. Sie liefert das folgende Ergebnis:

```
Nomen : garten
GENUS : MASKULIN
PLURALBILDUNG : mit UMLAUTUNG
Singular-Klasse 2 : Nominativ, Dativ, Akkusativ : keine Endung

Nomen : kind
GENUS : NEUTRUM
PLURALBILDUNG : ohne UMLAUTUNG
Plural-Klasse 9 : Nominativ, Genitiv, Akkusativ : 'er' Endung
```

Die obige Ausgabe ist nun folgendermaßen zu verstehen: Das *Grundwort* des Kompositums ist *GARTEN* ein Nomen im Singular. Das *Bestimmungswort* ist *KIND* mit der Endung *ER* und ist ein Nomen im Plural. Damit ist das Kompositum *Kindergarten* ein Nomen im Singular mit dem Genus Maskulin und dem Kasus Nominativ, Dativ oder Akkusativ.
Bei der Entfernung von Lösungsalternativen ist zu beachten, daß für Teilworte mehrere Wortart im Lexikon-Eintrag vermerkt sein können. Es wird jedoch nur die erste eingetragene Wortart betrachtet[6]. Als ein weiteres Beispiel mit mehreren möglichen Lesarten ist in Abbildung 8 das Wort *Bohrersatz* zerlegt[7].

Aus dem Analysebaum kann man nun zwei verschiedene Lesarten der Komposita-Analyse erzeugen:

1. LTW `bohrer` Nomen Singular, Plural
   RTW `satz` Nomen Singular
2. LTW `bohr` ~~Verb (Imperativ~~ Singular~~)~~
   RTW `ersatz` Nomen Singular

[6]Dies ist möglich, da die Kodierungen im Lexikon aufsteigend sortiert sind

[7]Kompakte graphische Darstellung der 'gültigen' Zerlegungsergebnisse

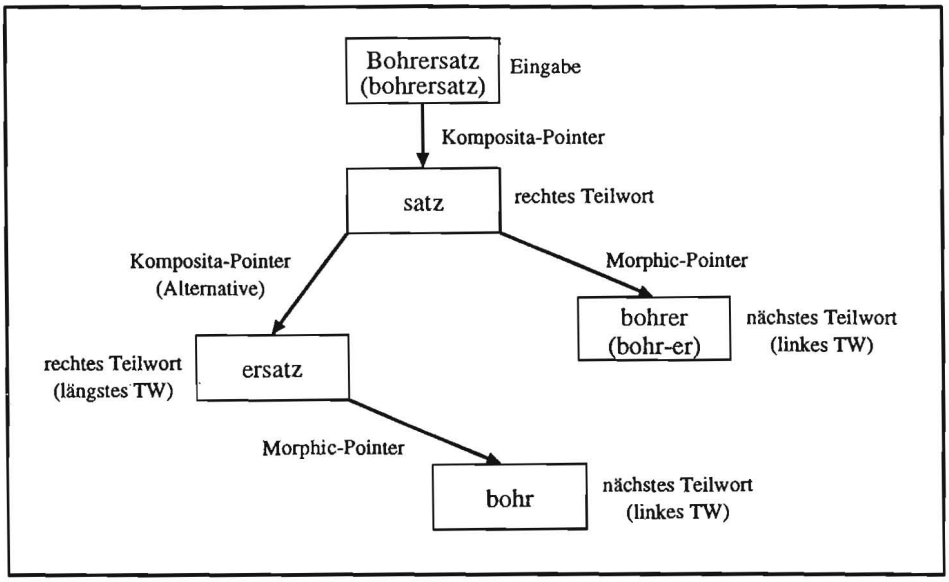

Abbildung 8: Komposita-Analysebaum(Ergebnis)

Die Ausgabe der Ergebnisse erfolgt mit Hilfe der bekannten *Morphic-Plus* - Funktionen. Diese überprüfen bei der Ausgabe der Analyseergebnisse ob die abgespaltenen Suffixe mit der Wortart aus dem Lexikon zu einem gültigen Ergebnis im Sinne von *Morphic-Plus* führen. Im Falle der *Komposita-Analyse* wird für Ergebnisse, bei denen z. B. Wortart und abgespaltener Suffix nicht zusammenpassen, eine Prognose der Wortart auf Grund der *Code*-Eintragung für das Teilwort abgegeben. Dies ist insbesondere bei unregelmäßigen Verben und Adjektiven notwendig, wenn im Lexikoneintrag keine Kodierung des Wortes zu finden ist, sondern nur auf den Stamm des Wortes verwiesen wird. In diesem Falle enthält die Komponente *R_Lex_Entry* der *Morphic*-Datenstruktur den Teilstring *STAMM*.

Da für Nomina nicht der Wortstamm, sondern der Nominativ im Lexikon eingetragen ist, wird in bestimmten Fällen für das Bestimmungswort kein Lexikoneintrag gefunden. Bei der Ausgabe der Ergebnisse wird in diesem Falle ein **e** an das Teilwort angehängt und ein weiterer Lexikonzugriff durchgeführt. Damit können auch Komposita wie *Farbfleck* mit der Zerlegung in *Farb* (*Farbe*) und *Fleck* analysiert werden.

Ausgehend von der Wurzel wird der Analysebaum solange durchlaufen, bis die Variable *ana_flag* der Wurzel den Wert **7** angenommen hat. In diesem Falle ist die Kompositaanalyse beendet. Wurde keine gültige Zerlegung für das Eingabewort gefunden, so wird die Eingabe als nicht analysierbar klassifiziert.

5.5 Einbindung der Komposita-Analyse in die morphologische Analyse

Wie man aus der obigen Beschreibung der Komposita-Analyse leicht ersehen kann ist diese Analysekomponente rechenzeit- und speicherplatzintensiv. Aus diesem Grund sollte sie nur selten genutzt werden um die gute Performanz von *Morphic-Plus* zu erhalten. Die Komposita-Analyse wird nur gestartet, wenn *Morphic-Plus* für eine Eingabe kein Ergebnis liefern kann. Zur Steigerung der Verarbeitungsgeschwindigkeit wird eine Mindestwortlänge für die Komposita-Analyse eingeführt (zur Zeit 7 Zeichen lang). Für die Verarbeitung von *Bindestrich-Komposita*, wie sie bei der Text-Analyse vorkommen, könnte man sich eine einfachere Variante vorstellen, die versucht, ein *Linkes Teilwort*, das *Bestimmungswort* abzuspalten. Somit müßte keine vollständige Komposita-Zerlegung der Eingabe durchführt werden.
Nach der Ausgabe der Zerlegungsalternativen wird eine 'Substruktur' über den kompos-Zeiger in die bestehende Analysestruktur eingehängt. Für jede 'gültige' Zerlegung wird eine solche 'Substruktur' erzeugt und aufgenommen. Eine graphische Darstellung ist in Kapitel B Abbildung 9 zu sehen.

6 Komponente zur Verarbeitung ganzer Sätze und Texte

Hierbei handelt es sich um ein Werkzeug, das einen Text (ASCII-Text) aus einer Datei einliest und in einzelne Strings (Worte) aufspaltet. Die verwendeten Trennsymbole sind Leerzeichen (blank) und Steuerzeichen wie Tabulatoren, Zeilenschaltung etc. Die einzelnen Wörter werden als Elemente einer linear verketteten Liste abgespeichert, die als Eingabe für *Morphic-Plus* dient.
Die beim Aufspalten des Textes entstandenen Strings müssen nochmals zeichenweise abgearbeitet werden, da sie Ziffern, Buchstaben und Sonderzeichen in beliebiger Reihenfolge enthalten können. Eine wichtige Komponente stellt dabei die Behandlung von 'Trennzeichen' [Sri93] dar. Bei der Aufspaltung des Textes werden folgende Zeichenfolgen erzeugt:

- In **Ziffernfolgen** sind Punkt und Kommata erlaubt (z.B.: 788.452.686,097 oder 654,787,451.056)
- **Sonderzeichen** oder **Satzzeichen** stehen als Einzelzeichen (1 Zeichen = 1 Wort)
- **Buchstabenfolgen** enthalten:
  - nur Buchstaben
  - einen Punkt am Wortende (z.B. für Abkürzungen)
  - Trennungszeichen bzw. Bindestriche im Wort (z.B. Sachsen-Anhalt), das Wort wird zusätzlich in Einzelworte aufgetrennt. Für das Beispiel werden schließlich 4 Listenelemente (1. **Sachsen-Anhalt**, 2. **Sachsen**, 3. -, 4. **Anhalt**) erzeugt
  - Trennungszeichen am Wortende werden eliminiert und das Folgewort, falls kleingeschrieben angehängt (Ausnahme bilden z. B. *oder*, *und*, etc. bei der Kombination wie: *ab- und zugestellt*)
  - Trennungszeichen und Schrägstrich sollen aufgelöst werden (z.B. aus *Ein-/Ausgabe* wird *Eingabe und Ausgabe* falls das zweite Wort einen abtrennbaren Präfix besitzt)

7 Beispiele der Analyseergebnisse

Angegeben wird jeweils das eingegebene Wort und die Analyseergebnisse, die von *Morphic-Plus* erzielt wurden.
Eingabe: 'den' liefert:

```
Relativ-Pronomen : den
FLEXION :  maskulin :  Singular :  Akkusativ,

Determinativ / Relativpronomen : den
FLEXION :  maskulin :  Singular :  Akkusativ,  Plural :  Dativ,
           feminin :  Plural :  Dativ,
           neutrum :  Plural :  Dativ,
Numerus-Klasse : sg + pl
```

Interpretation:

- 'den' ist ein Relativpronomen, im Genus: Maskulin, Numerus: Singular und Kasus: Akkusativ
- 'den' ist ein Determinativ (Relativpronomen) im Genus: Maskulin, Numerus: Singular, Kasus Akkusativ oder Numerus: Plural, Kasus: Dativ oder im Genus: Feminin, Numerus: Plural, Kasus: Dativ oder im Genus: Neutrum, Numerus: Plural und Kasus Dativ. Determinativ kann in Numerus Singular (sg) und Plural (pl) verwendet werden.

Eingabe: 'Ländern' liefert:

```
Nomen : land
Plural
GENUS : NEUTRUM
PLURALBILDUNG : mit UMLAUTUNG
Plural-Klasse 9 : Dativ : 'ern' Endung
```

Interpretation:
'ländern' (Stamm 'land') ist ein Nomen, im Genus: Neutrum, Numerus: Plural, Kasus: Dativ mit Endung 'ern'; die Pluralbildung erfolgt durch Umlautung; die Klasse '9' resultiert aus der Klassifikation von [Sch72].
Eingabe: 'werden' liefert:

```
Verb-Stamm : werden
Klasse 6
Praesens : Singular : Anrede : Plural : 1. und 3. Person, Anrede /
Konjunktiv-1 : Singular : Anrede : Plural : 1. und 3. Person, Anrede /
Infinitiv /
```

```
Imperativ : Anrede
Hilfsverb (0) : sein
abgetrennter Praefix :
Laenge des Praefix : 0
```

Interpretation:
'werden' (Stamm 'werd') ist ein Verb, Klasse 6 (nach SUTRA [Bus83]); Tempus: Praesens, Numerus: Singular, Anredeform, oder Numerus: Plural, 1. und 3. Person und Anredeform; oder Tempus: Konjunktiv-1, Numerus: Singular, Anredeform, oder Numerus: Plural, 1. und 3. Person und Anredeform; oder Tempus: Infinitiv; oder Tempus: Imperativ, Numerus: Anredeform. Bildung des Tempus Partizip-Perfekt verwendet das Hilfsverb 'sein'. Es wurde kein Präfix abgetrennt.
Eingabe: 'wesentlichen' liefert:

```
Adjektiv : wesentlich
Positiv : ohne : maskulin : Singular : Genitiv, Akkusativ /
                            Plural : Dativ
                feminin  : Plural : Dativ
                neutrum  : Singular : Genitiv /
                            Plural : Dativ
          best.: maskulin : Singular : Genitiv, Dativ, Akkusativ /
                            Plural : Nominativ, Genitiv, Dativ, Akkusativ
                feminin  : Singular : Genitiv, Dativ /
                            Plural : Nominativ, Genitiv, Dativ, Akkusativ
                neutrum  : Singular : Genitiv, Dativ /
                            Plural : Nominativ, Genitiv, Dativ, Akkusativ
          unbe.: maskulin : Singular : Genitiv, Dativ, Akkusativ /
                            Plural : Nominativ, Genitiv, Dativ, Akkusativ
                feminin  : Singular : Genitiv, Dativ /
                            Plural : Nominativ, Genitiv, Dativ, Akkusativ
                neutrum  : Singular : Genitiv, Dativ /
                            Plural : Nominativ, Genitiv, Dativ, Akkusativ
keine Umlautung
keine Elision
attributiv und praedikativ verwendbar: flektierbar und graduierbar
```

Interpretation:
'wesentlich' ist ein Adjektiv, Positiv, ohne Artikel; Genus: Maskulin, Numerus: Singular, Kasus: Genitiv, Akkusativ oder Numerus: Plural, Kasus: Dativ oder Genus: Feminin, Numerus: Plural, Kasus: Dativ oder Genus: Neutrum, Numerus: Singular, Kasus: Genitiv, Dativ oder Numerus: Plural, Kasus: Dativ
analog für die Fälle mit bestimmtem und unbestimmtem Artikel.
Es findet keine Umlautung und keine Elision statt, das Adjektiv kann attributiv oder prädikativ verwendet werden und ist flektier- und graduierbar.
Eingabe: 'in' liefert:

```
Praeposition : in
Folgekasus Dativ, Akkusativ
```

Eingabe: 'fünf' liefert:

```
Kardinalzahl : fuenf
Ordinal-Endung : t
Ordinalzahl : Praedikativ - gebraucht
```

Interpretation:
'fuenf' ist eine Kardinalzahl; Ordinalzahl hat die Endung 't', Ordinalzahl wird pädikativ verwendet.
Eingabe: 'berührt' liefert:

```
Verb : beruehren
Tempus : Partizip Perfekt
Hilfsverb: haben
abgetrennter Praefix :
Laenge des Praefix : 0

Verb-Stamm : ruehren
klasse 1
Praesens : Singular : 3. Person : Plural : 2. Person / Imperativ : Plural
Hilfsverb: haben
abgetrennter Praefix : be
Laenge des Praefix : 2
```

Interpretation:

- 'berührt' (Stamm 'beruehren') ist ein Verb, Tempus: Partizip Perfekt.
- 'berührt' (Stamm 'ruehren') ist ein Verb, Klasse 1 (nach SUTRA [Bus83]); Tempus: Präsens, Numerus: 3. Person Singular, oder 2. Person Plural; oder Tempus: Imperativ, Numerus: Plural; Bildung des Tempus Partizip-Perfekt verwendet das Hilfsverb 'haben'. Es wurde kein Präfix abgetrennt.

Eingabe: 'nicht' liefert:

```
Konjunktor : nicht
Klasse-0 : Partikel
```

Interpretation:
'nicht' ist ein Konjunktor, Klasse 0 als Partikel verwendbar.
Eingabe: 'mit' liefert:

```
Praeposition : mit

Verb-Praefix : mit
Klasse-0 : Verbzusatz
```

Interpretation:

- 'mit' ist eine Präposition.
- 'mit' ist ein Verb-Präfix, also ein Verbzusatz.

Eingabe: 'Dokumentanalysesysteme' liefert die folgenden Zerlegungsalternativen:

```
 Neue Lesart der Kompositazerlegung:
**************************************
 Nomen : system
 GENUS : NEUTRUM
 PLURALBILDUNG : ohne UMLAUTUNG
 Singular-Klasse 4 : Dativ : 'e' Endung
 Nomen : system
 GENUS : NEUTRUM
 PLURALBILDUNG : ohne UMLAUTUNG
 Plural-Klasse 7 : Nominativ, Genitiv, Akkusativ : 'e' Endung

 Nomen : analyse
 GENUS : FEMININ
 PLURALBILDUNG : ohne UMLAUTUNG
 Singular-Klasse 1 : Nominativ, Genitiv, Dativ, Akkusativ : keine Endung


 Nomen : dokument
 GENUS : NEUTRUM
 PLURALBILDUNG : ohne UMLAUTUNG
 Singular-Klasse 4 : Nominativ, Dativ, Akkusativ : keine Endung

 Neue Lesart der Kompositazerlegung:
**************************************
 Nomen : system
 GENUS : NEUTRUM
 PLURALBILDUNG : ohne UMLAUTUNG
 Singular-Klasse 4 : Dativ : 'e' Endung
 Nomen : system
 GENUS : NEUTRUM
 PLURALBILDUNG : ohne UMLAUTUNG
 Plural-Klasse 7 : Nominativ, Genitiv, Akkusativ : 'e' Endung

 Nomen : dokumentanalyse
 GENUS : FEMININ
 PLURALBILDUNG : ohne UMLAUTUNG
 Singular-Klasse 1 : Nominativ, Genitiv, Dativ, Akkusativ : keine Endung
```

```
 Neue Lesart der Kompositazerlegung:
**************************************
 Nomen : analysesystem
 GENUS : NEUTRUM
 PLURALBILDUNG : ohne UMLAUTUNG
 Singular-Klasse 4 : Dativ : 'e' Endung
 Nomen : analysesystem
 GENUS : NEUTRUM
 PLURALBILDUNG : ohne UMLAUTUNG
 Plural-Klasse 7 : Nominativ, Genitiv, Akkusativ : 'e' Endung

 Nomen : dokument
 GENUS : NEUTRUM
 PLURALBILDUNG : ohne UMLAUTUNG
 Singular-Klasse 4 : Nominativ, Dativ, Akkusativ : keine Endung
```

Interpretation:
Die Komposita-Analyse liefert 3 Zerlegungsalternativen für das Wort 'Dokumentanalysesysteme'.

- eine Zerlegung in 3 Teilworte 'system' als Grundwort und die Bestimmungsworte 'analyse' und 'dokument'
- eine Aufspaltung des Wortes in 2 Teilworte mit dem Grundwort 'system' und dem Bestimmungswort 'dokumentanalyse'
- ebenfalls eine Zerlegung in 2 Teilworte mit 'analysesystem' als Grundwort und 'dokument' als Bestimmungswort

Beim Grundwort handelt es sich um ein Nomen im Genus: Neutrum, Numerus: Singular oder Plural und dem Kasus: Dativ / Nominativ, Genitiv, Akkusativ.
Die Bestimmungsworte sind ebenfals Nomen.
Die übrigen Angaben sind wie bei den früheren Beispielen zu lesen.
Eingabe: 'Zeitungsschreiber' liefert die folgenden Zerlegungsalternativen:

```
 Neue Lesart der Kompositazerlegung:
**************************************
 Das Teilwort: schreiber ist keine korrekte Wortform nach MORPHIC-PLUS
 Das Teilwort koennte ein Verb sein.

 Nomen : zeitung
 GENUS : FEMININ
 PLURALBILDUNG : ohne UMLAUTUNG
 Singular-Klasse 1 : Nominativ, Genitiv, Dativ, Akkusativ : keine Endung
```

Interpretation:
Die Eingabe wurde von der Komposita-Analyse in die beiden Teilworte 'schreiber' (schreib) und 'zeitung' zerlegt.
Das Teilwort 'schreiber' ist das Grundwort. Der Stamm 'schreib' ist ein Verb. Bei der Aufbereitung des Analyseergebnisses wird aber festgestellt, daß 'er' keine Verbendung ist. In diesem Falle wird auf grund des Wortstammes die Prognose abgegeben es könnte sich um ein Verb handeln.
Das Bestimmungswort 'zeitung' ist ein Nomen im Genus: Feminin, Numerus: Singular und Kasus: Nominativ, Genitiv, Dativ oder Akkusativ.

8 Vergleich der Vorgehensweise bei der Morphologischen Analyse zwischen *MORPHIX* und *Morphic-Plus*

Abweichend zu *MORPHIX* greift *Morphic-Plus* einzig auf ein Gesamtlexikon zu und führt keine Unterscheidung zwischen Vollform- und Stammformanalyse durch. *MORPHIX* arbeitet auf verschiedenen wortartspezifischen Lexika [FN86] [FN91]. Die Anzahl der unterscheidbaren Wortarten wurde in *Morphic-Plus* um die Klassen der *Namen* und *Abkürzungen* erweitert. *MORPHIX* verwendet bei der Analyse eine Property-Liste, als einen Analysechache in der alle bereits analysierten Worte eingetragen werden, um keine redundanten Wortanalysen durchzuführen. In der derzeitigen Implementierung von *Morphic-Plus* ist keine Liste der bereits analysierten Wörter vorhanden, in der nachgesehen werden kann, ob ein Wort bereits analysiert wurde. Der damit verbundene Mehraufwands durch wiederholte Lexikonzugriffe und wiederholtes abarbeiten der Analyseschritte für das gleiche Wort hat keine signifikante Auswirkung auf die Analysezeiten bei einem langen Text mit mehreren tausend Wörtern [8]. Es wurde eine einfache Funktion zur Vermeidung der wiederholten Analyse des selben Wortes in einem Text implementiert.

Morphic-Plus ist nur zur morphologischen Analyse Deutscher Sprache konzipiert. Die Flexionsinformation der Sprache ist zum größten Teil in den Ausgabefunktionen kodiert. Die Generierung und ein sogenannter Klärungsdialog, zum Eintragen neuer Worte in das Lexikon, wie in *MORPHIX* enthalten, sind in *Morphic-Plus* nicht implementiert. Für die Lexikonzugriffe werden Funktionen aus der Arbeit von Thomas Kieninger [KH93] bereitgestellt, die den Zugriff auf die Lexikondaten über Hashtabellen erlauben. Die Hashtabellen werdenin einem separaten Lauf generiert und sind auf Sekundärspeicher (Festplatte) gespeichert. Es ist auch möglich Funktionen zur Generierung und Zugriff auf Hashtabellen im Hauptspeicher zu verwenden. Diese Funktionen werden im Rahmen der Diplomarbeit von Stefan Agne [Agn95] zur Verfügung gestellt.

Die Analyse in *Morphic-Plus* wurde um eine Komponente zur Bearbeitung von zusammengesetzten Wörtern, sogenannte Komposita erweitert.

[8] siehe hierzu die Messungen in Kapitel D

9 Bemerkung zur verwendeten Datenstruktur von *Morphic-Plus.*

Die Grundlage der Analyseausgabe bildet eine verkettete Liste, die alle möglichen Analyse-Ergebnisse und Alternativen für ein Wort enthält. In der Struktur eines Listenelementes wird das zu analysierende Wort, die Morphic-Kodierung, die abgespalteten Präfixe und Suffixe, sonstige Informationen aus dem zugrundeliegenden Lexikon und andere analyserelevanten Daten zur Verfügung gestellt.
Die Datenstruktur für die morphologische Analyse sieht wie folgt aus:

```
typedef struct  morphic_info
 {
   char         *word;                 /* zuanalysierendes Wort              */
   char         *stamm;                /* unregelmaessiges Verb(Eingabewort) */
   char         code[8];               /* Codierung des Lexikon-Eintrags     */
   int          uml_flag;              /* Codierung bei Umlautreduktion      */
   int          w_flag;                /* Codierung Praef.-Suff.Abspaltung   */
   int          ana_flag;              /* Codierung fuer analysierte Worte   */
   char         *praefix;              /* abgespaltener Praefix              */
   char         *suffix;               /* abgespaltener Suffix               */
   char         *R_Lex_Entry;          /* restlicher Lexikon-Eintrag         */
   struct morphic_info  *alt_word;     /* Pointer auf Wort-Alternative       */
   struct morphic_info  *previous_word; /* Pointer auf vorherige Alternative */
   struct morphic_info  *komp_word;    /* Pointer auf Komposit-Teil          */
 }M_INFO;
```

In WORD ist das Eingabewort, bzw. der Stamm des Eingabewortes (Prä- und Suffixe sind abgespalten) oder der Stamm von unregelmäßigen Verben gespeichert. STAMM enthält das 'unregelmäßige Verb' bzw. das 'unregelmäßige Adjektiv'. Bei der Komposita-Analyse enthält WORD das aktuelle rechte Teilwort und STAMM das noch zubearbeitende linke Teilwort der Eingabe. Die 6- oder 7-stellige *MORPHIX*-Kodierung [FN86] [FN91] wird in CODE abgelegt.
Das UML_FLAG hat den Wert '1', wenn eine Umlautreduktion durchgeführt wurde. W_FLAG enthält eine Hilfskodierung aus der Präfixabspaltung um Partizip Präsens oder Partizip Perfekt Formen von Verben zubestimmen. Die Variable ANA_FLAG wird durch ein korrektes Analyseergebnis ('Plausibilitäts-Kontrolle' durch die Ausgabefunktionen) auf den Wert 1 gesetzt. Diese Variable wird in der Komposita-Analyse für andere Zwecke verwendet. Die Variable wird mit den Werten 2, 3, oder 4 belegt um bei der Zerlegung der Komposita den entstehenden Baum zu durchlaufen oder bei der Ausgabe mögliche Hypothesen zu spezifizieren. In PRAEFIX und SUFFIX sind die abgespalteten (konkatenierten) Prä- und Suffixe gespeichert. R_LEX_ENTRY enthält weitere Informationen

zu dem analysierten Wort, die im Lexikon steht.
ALT_WORD ist der Zeiger auf das nächste Element der Analyse-Liste und kennzeichnet dieses als weitere Alternative. Der PREVIOUS_WORD Zeiger verweist auf das Vorgängerelement und wird von der Komposita-Analyse benötigt. KOMP_WORD ist der Zeiger auf Elemente, die bei der Kompositabehandlung gebildet werden. Über diesen Zeiger werden alternative Zerlegungen des Wortes dargestellt.

Für die Analyse eines ganzen Textes oder Satzes wird eine weitere verkettete Liste benötigt, die mit der folgenden Datenstruktur aufgebaut wird:

```
typedef struct  eingabe_info
 {
  char                  *word;          /* zuanalysierendes. Wort              */
  int                   connect;        /* 'Analysecache'-Varibale             */
  struct eingabe_info   *next_word;     /* Pointer auf naechstes Eingabe-Wort  */
  struct eingabe_info   *prev_word;     /* Pointer auf vorheriges Eingabe-Wort */
  struct eingabe_info   *kompos;        /* Pointer auf Kompositzerlegung       */
  struct morphic_info   *morphic_word;  /* Pointer auf Morphic-Analyse DS      */
 }MC_INFO;
```

In WORD steht ein einzelnes Eingabewort, das aus dem Text extrahiert wurde. Mit Hilfe der Variablen CONNECT wird das Eingabewort als wiederholt vorkommend gekennzeichnet. NEXT_WORD zeigt auf das nächste Listenelement mit dem nächsten Wort des Textes. PREV_WORD zeigt bei einem wiederholt auftretendem Wort auf dessen erstes Vorkommen in der Liste. KOMP_WORD zeigt auf eine Komposit-Zerlegung. Der Zeiger MORPHIC_WORD weist auf ein erstes Element der zuvor beschriebenen Datenstruktur für die morphologische Analyse.

10 Diskussion und Ausblick

Durch die strikte algorithmische Abarbeitung und die lokale Sicht nur auf ein einzelnes Wort können auch falsche und unsinnige Analyseergebnisse erzeugt werden. Bei einer Kombination mit einer Sprachverarbeitungskomponenten oberhalb der Morphologieebene und der damit gewonnenen Kontextinformation könnten diese 'Ausreißer' wahrscheinlich vermieden werden. Dies wurde bisher wegen einer einfachen und schnellen Verarbeitung in Kauf gegenommen. In der derzeitigen Implementierung wird in Kauf genommen, daß nicht alle redundanten Ergebnisse vor der Ausgabe eliminiert werden. Wenn sich die Einträge in der Morphic-Datenstruktur, die für das Analyseergebnis irrelevant sind, unterscheiden, kommt es zu einer doppelten Ausgabe. Beispielsweise wird für das Eingabewort 'abbestellen' die beiden Ergebnisse 'Verb bestellen' und 'Verb stellen' geliefert (aufgrund der Suffix-Abspaltung).
Einen weiteren Fall für redundante Analyseergebnisse bilden Adjektive mit der Endung *lich*. Diese werden zum einen wegen der Endung (Suffix) 'lich' als Adjektiv klassifiziert (Ansatz einer Derivationsmorphologie) und andererseits kann das Wort im Lexikon gefunden werden und ebenfalls das Ergebnis Adjektiv liefern. Hier wurde versucht, für Adjektive mit der Endung *lich* eine *Derivations-Analyse* durchzuführen. Enthält das Eingabewort noch einen Umlaut, tritt eine weitere redundante Ausgabe auf.

Für den Einsatz von *Morphic-Plus* im Project **INCA** wären einige der nachfolgenden Punkte eine Erleichterung bei der Klassifikation und dem Retrieval von Dokumenten.

- Einbau einer Komponenten zum manuellen Eintragen von neuen Worten in das Morphic-Lexikon, die nicht erkannt wurden (analog zum Klärungsdialog von *MORPHIX*), jedoch für alle Wortarten von *Morphic-Plus*.
- Erweiterte Worterkennung aufgrund von abspaltbaren Suffixen; wie bereits bei Adjektiven mit der Endung 'lich' könnte man z.B. für Nomen die Endungen 'heit', 'keit' oder 'ung' verwenden. Weitere Prä- und Suffixe, die abgespalten werden können und bei einer Hypothese des zu analysierenden Wortes in Anwendung gebracht werden, findet man z. B. im Kapitel 'Die Wortbildung' des Duden [Dro84].
- Einbau von Derivationsanalyse-Komponenten zur Verbesserung der Analyseergebnisse, insbesondere bei der Komposita-Analyse. Beispielsweise

kann das Wort *Schreiber* nicht analysiert werden, *schreib* steht als Verb im Lexikon, aber die Endung *er* ist kein gültiger Verb-Suffix.

- Berücksichtigung von phonetischem und grammatikalischem Wissen bei der Abspaltung von Präfixen und Suffixen.

- Einfließen von Kontextinformationen und Arbeiten mit 'Lokalen Sichten' auf Wörtern. D.h. Einschränkung von Fehlern, die durch große Wörterbücher entstehen würden. Es besteht die Möglichkeit der Einbindung des Sichtenkonzeptes, wie es mit Hilfe von Lexikas in **ALV/OMEGA** durchgeführt wird.

- Erkennen von Datums-, Zeit- und Telefonnummernangaben sowie Namen. Einführung neuer 'Wortarten' in der morphologischen Analyse.

- Behandlung von Römischen Ziffern.

- Analyse unvollständiger oder inkorrekter Eingabe. Typische Fehler, die aufgrund von schlechten Scan- und OCR-Ergebnissen herrühren.

A User Interface

Zur Zeit wird für die Benutzung von *Morphic-Plus* nur eine einfache ASCII-Schnittstelle bereitgestellt. Der Benutzer hat daneben die Möglichkeit drei Varianten von *Morphic-Plus* zu verwenden. Diese sind im einzelnen:

- *Morphic-Plus* **ohne** Komposita-Analyse
- *Morphic-Plus* **mit** Komposita-Analyse
- Komposita-Analyse als Stand-Alone-Version

A.1 *Morphic-Plus* mit/ohne Komposita-Analyse

Für den Benutzer besteht kein Unterschied in der Bedienung der beiden verschiedenen Versionen. Sie unterscheiden sich nur in der Mächtigkeit der morphologischen Analyse. Der Aufruf des Programms erfolgt für

- *Morphic-Plus* **mit** Komposita-Analyse mit dem Befehl: `$ k_morphic`
- *Morphic-Plus* **ohne** Komposita-Analyse mit dem Befehl: `$ morphic`

In einer ersten Anzeige wird der Benutzer mit der folgenden Ausgabe

```
M O R P H I C - P L U S

  Version 2.01

Ein morphologisches Analyseprogramm fuer das Deutsche


            hit ANYKEY (return-key) to continue
```

über die Version von *Morphic-Plus* informiert.
In den nächsten Schritten werden die Namen der Dateien mit dem Lexikon und den Abkürzungen erwartet:

```
 Geben Sie den File-Namen der Lexikon-Datei ein:
Geben Sie den File-Namen der Abkuerzungs-Datei ein:
```

Dem Programm wird z. B. **abkuerz** als Namen der Abkürzungsdatei und **morphic.dict** als Lexikondatei übergeben.
Als nächstes hat der Benutzer die Auswahl zwischen verschiedenen 'Analyse-Verfahren':

```
Morphic-Plus Analyse Programm
Sie haben die folgende Auswahl bei der Analyse:
- Einen ganzen Text aus einer Datei zu analysieren. (Eingabe: t)
- Einen Text bzw. einen Satz in Online-Eingabe zu analysieren. (Eingabe: s)
- Ein Wort zu analysieren. (Eingabe: w)
- Jede andere Eingabe beendet das Programm
```

A.1.1 Wortanalyse

Interaktiv wird der Benutzer von *Morphic-Plus* aufgefordert das zu analysierende Wort einzugeben.

```
Bitte geben Sie das zu analysierende Wort ein (Ende: '\#ende'):
```

Als Ergebnis erhält man im Erfolgsfall alle möglichen Analyseergebnisse zurück. Ansonsten wird

```
Das Wort: ... konnte nicht analysiert werden.
```

ausgegeben.

A.1.2 Satz- / Text-Analyse

Analog zur Wortanalyse wird der Benutzer aufgefordert, den zu bearbeitenden Text einzugeben. Die Eingabe wird in einzelne Wörter aufgespalten und mit jedem Wort die Wortanalyse angestoßen. Die Ausgabe der morphologischen Ergebnisse erfolgt für jedes Wort wie bereits oben beschrieben.

A.1.3 Text-Analyse (File-Input)

Zur Bearbeitung großer Texte, die in ASCII-Format vorliegen, steht eine Funktion bereit, die vom Benutzer die Eingabe des Filenamens erwartet, in dem der Text gespeichert vorliegt.

```
Geben Sie bitte den Namen des Eingabe-Files mit den Daten
zur Mophologischen-Analyse ein:
```

Die Aufspaltung des Textes erfolgt wie in Kapitel 6 beschrieben. Für jedes Wort wird dann die Wortanalyse durchgeführt. Bei größeren Texten ist es ratsam, die Ausgabe der Analyseergebnisse in ein File umzulenken, da für die Ergebnisdarstellung ein ausführliches Format gewählt wurde.

A.2 Komposita-Analyse

Mit diesem Programm wird dem Benutzer ein Werkzeug zur Komposita-Analyse in die Hand gegeben. Das eingegebene Wort wird in Teilworte zerlegt, obwohl z. B. die morphologische Analyse von *Morphic-Plus* das Wort hätte analysieren können.
In einer ersten Anzeige wird der Benutzer mit der folgenden Ausgabe

```
M O R P H I C  -  P L U S

   Komposita - Analyse

  Version 1.0


            hit ANYKEY (return-key) to continue
```

über die Version der *Komposita - Analyse* informiert.
Danach wird die Eingabe des zu zerlegenden (analysierenden Wortes erwartet.

```
Bitte geben sie ein Wort fuer die Komposita-Analyse ein :
```

B Programmer's Interface

Im folgenden werden die Hauptfunktionen vorgestellt, mit denen die morphologische Analyse durchgeführt werden kann, bzw. die die Analyse steuern.
Mit den Funktionen *analyse_text*, *analyse_sentence*, *analyse_word* wird ein Dialog angestoßen um einzelne Worte, Sätze oder Textfragmente und ganze Texte zu bearbeiten. Nachfolgend wird auf die einzelnen Funktionen kurz eingegangen. Eine detailierte Beschreibung aller *Morphic-Plus*-Funktionen ist in Kapitel C zu finden.
Die Analyse eines ASCII-Textes, der in einer ASCII-Datei gespeichert vorliegt, wird durch die Funktion *analyse_text* gesteuert:

```
void analyse_text ()

{
. . .
        printf(" Geben Sie bitte den Namen des Eingabe-Files mit den Daten
                 zur Mophologischen-Analyse ein: ");
        gets(input_file_name);
. . .
        compute_parse_input(kopf->next_word);
        compute_analyse (kopf->next_word);
        free_memory(kopf);
}
```

Wobei die Funktion *compute_parse_input()* eine Vorverarbeitung darstellt, um z. B. Worttrennungen und Bindestrichkomposita, die in einem Text auftreten, zu bearbeiten. Die Funktion *compute_analyse()* steuert dann die morphologische Analyse. *Freememory* wird verwendet um nach Beendigung der Analyse den reservierten Speicherplatz freizugeben.
Die Funktion *analyse_sentence* erwartet die Eingabe des zu analysierenden Textes über Tastatur. Die Aufspaltung des Textes und die morphologische Analyse erfolgen in gleiche Weise wie in der obigen Funktion *analyse_text* beschrieben.

```
void analyse_sentence ()

{
. . .
        printf(" Geben Sie bitte den zu analysierenden Text ein: ");
        gets(inputbuffer);
. . .
        compute_parse_input(kopf->next_word);
        compute_analyse (kopf->next_word);
        free_memory(kopf);
}
```

analyse_word ist die Grundfunktion zur morphologischen Analyse eines Wortes.

```
void analyse_word ()

{
. . .
      do
         {
           printf(" Bitte geben Sie das zu analysierende Wort ein (Ende: '#ende'): ");
           gets(in_word);
           if (strcmp(in_word, "#ende") != 0)
             {
                 help = (MC_INFO *) malloc (sizeof *help);
                 help->word = in_word;
                 morphic_analyse (help);
                 free_memory (help);
              }
           if (strcmp(in_word, "#ende") == 0)
                 b = 1;
       } while (b == 0);
```

Mit der Funktion *morphic_analyse* wird die morphologische Analyse für das Wort in *help* $\rightarrow$ *word* durchgeführt. Als Funktionsparameter wir der Zeiger auf das zu bearbeitende Listenelement übergeben.

```
void morphic_analyse (MC_INFO *help)
{
. . .
     init_morphic_ds ();
     strcpy(h_word, help->word);
     help->morphic_word = head;
     l = strlen(h_word);
     printf("\n Eingabewort : %s \n", h_word);
     if (l > 1 && h_word[l-1] == '.')
       {
          strcpy(head->word, h_word);
          lex_access_abkuerz(head);    /* Lexikonzugriff fuer Abkuerzungen */
       }
     else
       {
          strcpy(head->word, lower_input(h_word));
          if (isdigit(head->word[0]))
             strcpy(head->code, "1900000");
          else
            {
              compute_red_umlaut(head);    /* Umlaut Reduktion   */
              compute_elim_umlaut(head);   /* Umlaut Elimination */
              compute_stem_praef(head);    /* Praefix Abspaltung */
```

```
            compute_stem_suff(head);      /* Suffix Abspaltung  */
            compute_stem_praef(head);     /* Praefix Abspaltung */
            compute_duplicate(head);      /* Elimination von Duplikaten */
            compute_lex_search(head);     /* Lexikonzugriff     */
            compute_non_lex_entry(head); /* Elimination von nicht gefundenen 'Worten' */
            compute_alternate(head);      /* Alternativen Generierung */
            compute_lex_search(head);     /* Lexikonzugriff     */
            compute_non_lex_entry(head); /* Elimination von nicht gefundenen 'Worten' */
            compute_alternate(head);      /* Alternativen Generierung */
            compute_lex_search(head);     /* Lexikonzugriff     */
            compute_duplicate(head);      /* Elimination von Duplikaten */
          }
      }
    analysiert = 0;
    compute_output(head);              /* Ausgabe der Ergebnisse */
    compute_non_ana_entry(head);       /* Entfernen falscher Ergebniselemente */
    if (analysiert == 0 && (h_word[l-1] != '.' || l >= 7))
      {
         komp_analysis(head->word);    /* Aufruf der Komposita-Analyse */
         if (komp_ana == 0)
            printf(" Das Wort: %s konnte nicht analysiert werden. \n",   head->word);
      }
}
```

In der obigen Funktion ist auch der Aufruf der Komposita-Analyse mit *komp_analysis()* realisiert. Dieser Befehl fehlt bei der 'reinen' morphologischen Analyse.
Die Funktion *komp_analysis* steuert die Zerlegung des Eingabewortes in die Komposita-Bestandteile. Es wird eine globale Variable für den Analysebaum generiert. In einer weiteren globalen Variable werden dann die Ergebnisse der Zerlegung zwischengespeichert, bevor sie mit Hilfe der *Morphic-Plus* Ausgabefunktion aufbereitet werden.

```
void komp_analysis (char komposit[MAXWORD])
{
. . .
    komp_init_morphic_ds ();
    komp_ana = 0;
/*  Aufspaltung der Eingabe und Aufbau des Analysebaumes        */
    strcpy (start->stamm, lower_input(komposit));
    komp_suffix_separation (start);
    anker = start;
    help = komp_generate_wordsection (start);
    help = komp_go_back (help);
    while (help != start)
         {
            help = komp_generate_wordsection (help);
            help = komp_go_back (help);
```

```
        }
/*  Zusammenbau des Analyseergebnisses der Kompositazerlegung   */
    while (start->ana_flag < 7)
        {
           komp_init_ergebnis_ds ();
           komp_prepare_result (start);
        }
}
```

Zum besseren Verständnis der verwendeten Datenstruktur und der möglichen Verkettungen der einzelnen Elemente ist die Abbildung 9 gedacht. An zwei Beispielen, einer einfachen morphologischen Analyse und einer Komposita-Analyse sind die Zusammenhänge graphisch dargestellt.

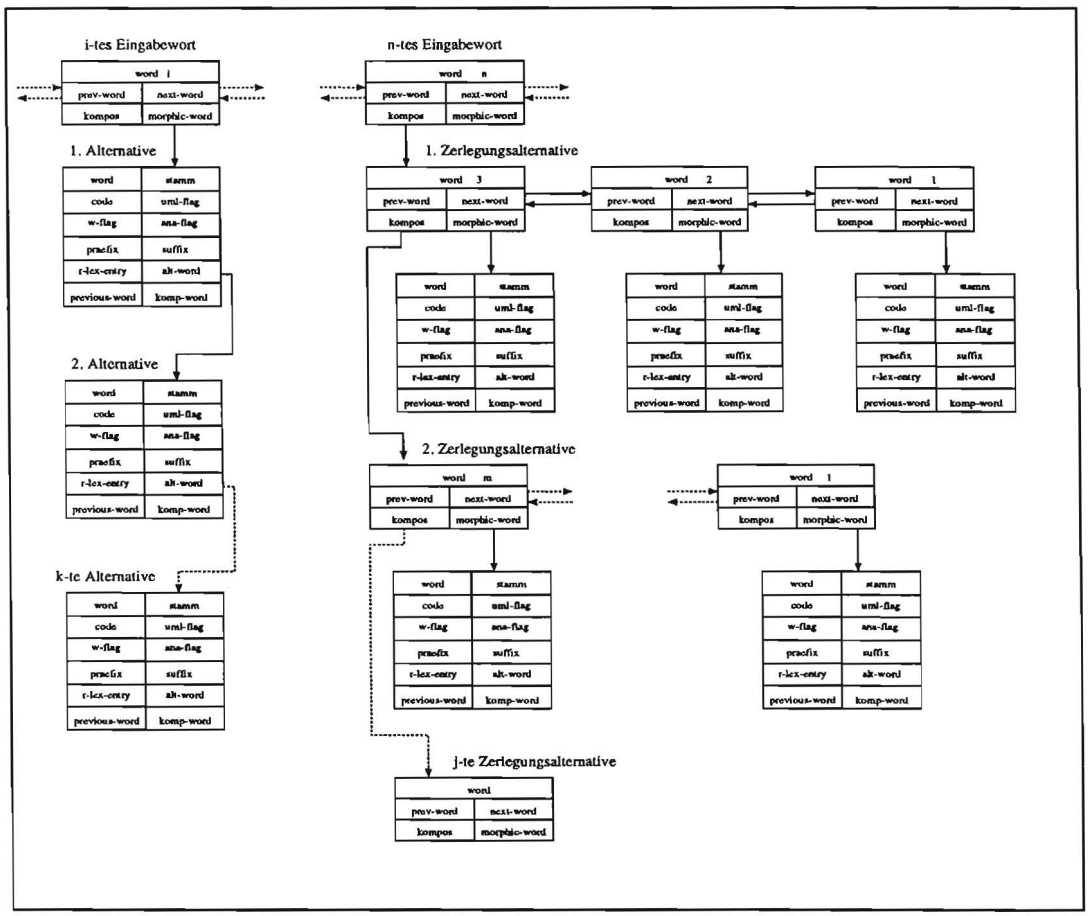

Abbildung 9: Beispiel der Analyse-Struktur

Bei der morphologischen Analyse sind alternative Wortarten für ein Eingabewort mit Hilfe der `alt_word`-Zeiger verkettet. Für jede Wortart gibt es ein Element der *Morphic*-Datenstruktur. Bei der Komposita-Analyse gibt es für die

Zerlegung in die einzelnen Teilworte je ein Element der 'Top'-Datenstruktur, an dem jeweils ein Element der *Morphic*-Datenstruktur hängt. Im ersten Listenelement ist das Grundwort und im letzten das Bestimmungswort gespeichert. Die verschiedenen Zerlegungsmöglichkeiten für ein Kompositum sind über den `kompos`-Zeiger der 'Top'-Datenstruktur verbunden.

Um eine einfache Weiterverarbeitung der Komposita-Analyseergebnisse zu ermöglichen, wurde der Wertebereich der Variablen `ana_flag` erweitert. In der nachfolgenden Tabelle sind die Werte näher spezifiziert.

Wert	Bedeutung
0	nicht analysiert
1	analysiert
2	Hypothese (Nomen, Verb oder Adjektiv)
3	Hypothese (unregelmäßige Verben oder Adjektive)

C Programm-Module und Funktionen von *Morphic-Plus*

Zur besseren Lesbarkeit wurde der Source-Code für *Morphic-Plus* in einzelne Module unterteilt. In diesen Modulen sind thematisch zusammenhängende Bearbeitungsschritte zusammengefaßt. Die Daten der morphologischen Analyse werden in globalen Variablen der in Kapitel 9 beschriebenen Datenstrukturen gespeichert. Die Definition aller globalen Variablen ist im File *global.h* zu finden.
Das Hauptsteuerprogramm von *Morphic-Plus* ist **text_ana.c** mit den Funktionen:

```
void init_help_ds ()
void init_input_ds ()
void init_s_ds ()
void new_word (MC_INFO *help, char *h_word)
void compute_alpha (MC_INFO *help)
void compute_digit (MC_INFO *help)
void compute_p_char (MC_INFO *help)
void compute_parse_input (MC_INFO *help)
void red_analyse (MC_INFO *help)
void compute_analyse (MC_INFO *help)
void analyse_text ()
void analyse_sentence ()
void analyse_word ()
void test_out (MC_INFO *help)
void free_memory (MC_INFO *help)
void intro ()
```

Mit Hilfe der Funktion *init_help_ds* wird eine Hilfsvariable der *Morphic-Datenstruktur* initialisiert, die bei der Vorverarbeitung der Eingabestrings in *compute_alpha* benötigt wird. Es werden die beiden Listenelemente *first* und *last* erzeugt und miteinander verkettet.

Die Initialisierung der Top-Datenstruktur wird in der Funktion *init_input_ds* durchgeführt. Es werden die Elemente *kopf* und *ende* erzeugt und miteinander verkettet.

Die Funktion *init_s_ds* wird zur Initialisierung der globalen Struktur S_DS verwendet.

Mit der Funktion *new_word* wird hinter dem aktuellen Element ein neues Listenelement mit dem Inhalt von *h_word* in die Liste eingefügt.

In der Funktion *compute_alpha* wird der Eintrag des Elementes zeichenweise abgearbeitet um Satzzeichen vom Wort zu trennen, Trennzeichen zubehandeln

oder beim Auftreten anderer Zeichen als Buchstaben den Rest des Wortes als neues Element in die Liste aufzunehmen. Bei der Behandlung des Satzzeichens *Punkt* bleibt zum einen der Punkt am Wort, es könnte eine Abkürzung sein; zum anderen werden zwei neue Elemente, das Wort und der Punkt, neugeneriert. Beim Trennstrich kann es sich um eine echte Worttrennung handeln, bei der das aktuelle Wort mit dem nächsten Wort konkateniert wird oder es handelt sich um ein Auslassungszeichen, bei dem dann versucht wird vom übernächsten Wort ein Präfix abzuspalten und den Rest mit dem aktuellen Wort zu konkatenieren. Bei Bindestrichworten werden zusätzlich zu dem Wort für jedes Teilwort ein neues Listenelement erzeugt. *k -k* oder *Doppel- bzw. Trippelkonsonanten* erfordern bei der Konkatenation von Teilworten besondere Behandlung [Dro91].

Die Funktion *compute_digit* parst das Wort auf Ziffern, Punkte und Kommata. Treten andere Zeichen auf, wird mit dem Rest ein neues Element generiert.

Beginnt das Wort mit einem Zeichen außer Buchstaben oder Ziffer, wird in der Funktion *compute_p_char* das erste Zeichen abgetrennt und mit dem Rest ein neues Element erzeugt.

Die Funktion *compute_parse_input* steuert den Aufruf der Funktionen *compute_alpha*, *compute_digit* und *compute_p_char*. Dabei wird die Liste Element für Element abgearbeitet.

Mit Hilfe der Funktion *red_analyse* wird die gesamte Liste der Eingabeworte auf mehrfaches Vorkommen einzelner Wörter untersucht und über den *prev_word*-Pointer eine Verkettung aller gleichen Worte des Textes durchgeführt. Daneben wird mit der Initialisierung der Variablen *connect*, mit dem Wert '1', dieses Wort als nicht mehr zu analysieren gekennzeichnet.

Die Liste der Eingabeworte wird in der Funktion *compute_analyse* durchlaufen und mit dem jeweils aktuellen Element die morphologische Analyse aufgerufen. Wurde die Liste der Eingabeworte mit der obigen Funktion *red_analyse* vorverarbeitet, so erfolgt bei wiederholt vorkommenden Wörtern nur die Ausgabe des Analyseergebnisses.

Die Funktion *analyse_text* liest einen ASCII-Text aus einem File ein und generiert mit den einzelnen Wörtern, die durch Leerzeichen, Zeilentrennern, Tabulatoren etc. getrennt sind, Einträge für die verkettete Liste. Diese Liste wird als Parameter an die Funktionen *compute_parse_input* und *compute_analyse* übergeben. Ist die Analyse des Textes abgeschlossen wird der Speicherplatz der Struktur wieder freigegeben.

Mit Hilfe der Funktion *analyse_sentence* wird die Dialog-Eingabe eines Textes in einzelne Wörter aufgespalten. Die Verarbeitungsschritte zum Aufbau der

'Analyse-Liste' ist analog zur Funktion *analyse_text.*

Die Funktion *analyse_word* ruft mit dem eingegebenen Wort die morphologische Analyse auf.

Zur Überprüfung der Einträge in der verketteten Liste während der Testphase steht die Ausgabefunktion *test_out* zur Verfügung. Alle Listenelemente mit ihrem Inhalt werden ausgegeben.

Mit der Funktion *free_memory* wird am Ende der Analyse die erzeugte Liste gelöscht und der Speicherplatz freigegeben.

Für die morphologische Analyse ist das Modul **morphic.c** zuständig.

```
char *lower_input (char *s)
char *stringclean (char *s)
char *split_line (char *s1)
void spalte_rest ()
void init_morphic_ds ()
void delete_entry (M_INFO *help)
void compute_duplicate (M_INFO *help)
void compute_non_lex_entry (M_INFO *help)
void compute_non_ana_entry (M_INFO *help)
void compute_output (M_INFO *help)
void make_alternate (M_INFO *help, char code[7], char rest[MAXLINE])
void make_stamm (M_INFO *help, char word[MAXWORD], char rest[MAXLINE])
void compute_alternate (M_INFO *help)
void test_output (M_INFO *help, int step)
```

Die Funktion **lower_input* wandelt den Eingabestring in Kleinbuchstaben um.

Leerzeichen am Anfang und Ende des Eingabestrings werden mit der Funktion **stringclean* entfernt.

Mit der Funktion **split_line* wird ein String aufgespalten. Leerzeichen sind als Worttrenner definiert. Das erste Wort des Strings wird zurückgegeben.

Die Funktion *spalte_rest* spaltet den Eingabestring in das erste Wort des Strings und den Rest des Strings. Sie greift auf die Funktion *split_line* zu. Als Übergabevariable dient die globale Variable *S_DS* mit den beiden Komponenten *S_DS.substr* für das erste 'Wort' und *S_DS.rest* für den Rest des zuzerlegenden Strings.

Die Funktion *delete_entry* wird verwendet um ein Element aus der Liste zu löschen. Der Übergabeparameter kennzeichnet das Element vor dem zu löschenden Element.

Die Initialisierung der Morphic-Datenstruktur wird in der Funktion *init_morphic_ds* durchgeführt. Es werden die globalen Elemente *head* und *tail* erzeugt und miteinander verkettet.

Die Funktion *compute_duplicate* dient dem Entfernen doppelter Elemente in der Liste. Diese doppelten Elemente entstanden bei der Präfix- und Suffixabspaltung.

Mit der Funktion *compute_non_lex_entry* werden alle Elemente der Liste entfernt, für die kein Lexikoneintrag gefunden worden ist.

Die Funktion *compute_output* ruft für jedes Element der Liste die Ausgabefunktion der morphologischen Ergebnisse auf.

Wird festgestellt, daß der Lexikonzugriff für ein Wort mehrere Analyseergebnisse für das Wort anbietet, erzeugt die Funktion *make_alternate* hinter dem aktuellen Element ein neues Element mit dem gleichen 'Worteintrag', der übergebenen Kodierung *code* und der restlichen Lexikoninformation *R_Lex_Entry*.

Bei unregelmäßigen Verben ist im Lexikoneintrag das Schlüsselwort *STAMM* und die Grundform des Verbs vorhanden. Die Funktion *make_stamm* fügt hinter der aktuellen Listenposition ein neues Element mit der Grundform des Verbs als 'Worteintrag' ein. Die restliche Lexikoninformation wird ebenfalls übernommen. Danach wird ein Lexikonzugriff mit der Grundform durchgeführt. Analoges gilt für unregelmäßige Adjektive.

Die Funktion *compute_alternate* steuert die Erzeugung vom Wortalternativen und die Behandlung der Stammform von unregelmäßigen Verben aufgrund der Lexikoninformation.

Zur Anzeige von Zwischenergebnissen der morphologischen Analyse bzw. Testausgaben (Ausgabe des Inhalts der Listenelemente) wird die Funktion *test_output* verwendet.

Mit Hilfe der Funktion *compute_non_ana_entry* werden alle Elemente aus der Liste entfernt, die kein Analyseergebnis bzw. kein gültiges Analyseergebnis für das Eingabewort liefern. Die Ausgabefunktionen der einzelnen Wortarten überprüfen die Korrektheit des Analyseergebnisses.

Für die morphologische Analyse Steuerung sind die Module **morph_st.c** bzw. **k_morph_st.c** zuständig.

```
void morphic_analyse (MC_INFO *help)
```

Die Funktion *morphic_analyse* ist die Rahmen- bzw. Steuerungsfunktion der morphologischen Analyse und steuert die oben beschriebenen Funktionen.

Je nach Modul ist die Steuerung mit oder ohne Komposita-Analyse.

Die Behandlung der Umlaute und ß für die morphologische Analyse wird im Modul **umlaut.c** durchgeführt. Dies ist in *Morphic-Plus* notwendig, da im Lexikon keine Umlaute und kein ß eingetragen sind.

```
void insert_new_element (M_INFO *help, char r_word [MAXWORD], int flag)
void compute_elim_umlaut (M_INFO *help)
void compute_red_umlaut (M_INFO *help)
```

Mit Hilfe der Funktion *insert_new_element* wird ein neues Listenelement in die Struktur eingefügt, wenn ein Umlaut 'e' entfernt werden konnte. Die Funktion *compute_elim_umlaut* entfernt das Umlaut 'e' aus dem Wort und setzt in einem neu erzeugten Element der Liste das 'uml_flag'. Der Worteintrag enthält kein Umlaut 'e'. Eine Ausnahme, die beachtet werden muß ist das 'ue', dem kein 'a' oder 'e' vorausgehen darf. Die Funktion *compute_red_umlaut* wandelt die Umlaute 'ä', 'ö' und 'ü' in Doppelvokale 'ae', 'oe' und 'ue' um. Diese Funktion transformiert auch das 'ß' in 'ss'.

Die Abspaltung von Präfixen und Suffixen wird von dem Modul **w_stamm.c** gesteuert. Die abspaltbaren Präfixe und Suffixe sind zu Gruppen mit gleichem ersten bzw. letzten Buchstaben zusammengefaßt. Jede Präfix- und Suffixgruppe ist in einer eigenen Funktion (Source-File) beschrieben. Die Datenübergabe für die Präfix- und Suffixabspaltung geschieht durch die globale Variable *S_DS*. In der Komponente *S_DS.substr* ist der Präfix bzw. der Suffix und in *S_DS.rest* ist der Wortstamm gespeichert.

```
void ins_mo_suff (M_INFO *help, char suff[MAXAFFIX],
                  char stem[MAXWORD], M_INFO *akt_MDS)
void ins_mo_praef(M_INFO *help, char praef[MAXAFFIX],
                  char stem[MAXWORD], M_INFO *akt_MDS, int verb_flag)
void compute_praefix (M_INFO *new_elem)
void compute_text_praefix (M_INFO *new_elem)
void compute_suffix (M_INFO *new_elem)
void compute_komp_suffix (M_INFO *new_elem)
void compute_stem_praef (M_INFO *help)
void compute_text_praef (M_INFO *help)
void compute_stem_suff (M_INFO *help)
void compute_komp_suff (M_INFO *help)
void a_praefix (M_INFO *anker, M_INFO *help)
void b_praefix (M_INFO *anker, M_INFO *help)
void d_praefix (M_INFO *anker, M_INFO *help)
void e_praefix (M_INFO *anker, M_INFO *help)
void f_praefix (M_INFO *anker, M_INFO *help)
void g_praefix (M_INFO *anker, M_INFO *help)
void h_praefix (M_INFO *anker, M_INFO *help)
```

```
void i_praefix (M_INFO *anker, M_INFO *help)
void k_praefix (M_INFO *anker, M_INFO *help)
void l_praefix (M_INFO *anker, M_INFO *help)
void m_praefix (M_INFO *anker, M_INFO *help)
void n_praefix (M_INFO *anker, M_INFO *help)
void r_praefix (M_INFO *anker, M_INFO *help)
void s_praefix (M_INFO *anker, M_INFO *help)
void u_praefix (M_INFO *anker, M_INFO *help)
void v_praefix (M_INFO *anker, M_INFO *help)
void w_praefix (M_INFO *anker, M_INFO *help)
void z_praefix (M_INFO *anker, M_INFO *help)
void e_suffix (M_INFO *anker, M_INFO *help)
void em_suffix (M_INFO *anker, M_INFO *help)
void er_suffix (M_INFO *anker, M_INFO *help)
void h_suffix (M_INFO *anker, M_INFO *help)
void n_suffix (M_INFO *anker, M_INFO *help)
void nd_suffix (M_INFO *anker, M_INFO *help)
void s_suffix (M_INFO *anker, M_INFO *help)
void t_suffix (M_INFO *anker, M_INFO *help)
```

Die Funktion *ins_mo_suff* wird aufgerufen, wenn ein abspaltbarer Suffix gefunden wurde. Das neue Listenelement wird hinter dem aktuellen Element eingefügt. Der abgetrennte Suffix wird mit einem bereits abgetrennten konkateniert. Der übergebene *stem* ist der neue 'Worteintrag'. Die restlichen Einträge des alten Listenelementes werden in das neue Listenelement übernommen.

Die Funktion *ins_mo_praef* ist analog zur obigen 'Suffix-Funktion'. Abgetrennte Präfixe werden an vorherige angehängt. Die Variable *verb_flag* enthält für bestimmte Präfixe einen Wert ungleich Null. Durch diesen Wert wird z. B. bei Verben die Zeiten *Partizip Präsens* und *Partizip Perfekt* gesteuert.

Die Ansteuerung der einzelnen Präfixabspaltfunktionen geschieht mit Hilfe der Funktion *compute_praefix*. Der erste Buchstaben des Wortes ist für die Auswahl der entsprechenden Funktion entscheidend.

Bei der Vorverarbeitung von Bindestichwörtern in der Analyse von Texten oder Sätzen wird bisher eine Präfixabspaltung durchgeführt. Die Funktion *compute_text_praefix* ist die zugehörige Ansteuerfunktion.

Von der Funktion *compute_suffix* werden in Abhängigkeit des letzten Buchstaben des Wortes eine entsprechende Funktion zur Abspaltung des Suffixes aufgerufen.

Für die Komposita Analyse wird die Funktion *compute_komp_suffix* zur Abspaltung möglicher Suffixe verwendet.

Die Funktionen *compute_stem_praef*, *compute_text_praef* und *compute_stem_suff compute_komp_suff* sind als Steuerungsfunktion zu verstehen und rufen für jedes Element der verketteten Liste die Funktion zur Präfix- oder Suffixabspaltung auf. Da das neue Listenelement immer hinter dem aktuellen Element eingefügt wird, erreicht man eine rekursive Abarbeitung der Liste.

Mit den **_praefix* und **_suffix* Funktionen werden die folgenden Präfixe und Suffixe von den Eingabeworten abgespalten.
Die Präfixe sind:

ab, an, auf, aus, abbe, aufrecht, auseinander.

be, bei, bevor, bereit, bestehen.

da, dar, dabei, daran, davon, davor, durch, dafuer, danach, darauf, dagegen, daneben, darueber, darunter.

er, ein, emp, ent, empor, ernst, entgegen.

fest, fort.

ge, gegen, gleich, gewaehr, getrennt, gegenueber.

her, hin, herab, heran, herum, hinzu, herauf, heraus, herbei, herein, hervor, hinter, herunter.

inne.

kurz.

los.

mit.

nach, nieder.

rad.

statt, still, sicher.

um, ueber, unter, ueberein.

ver, vor, voll, voran, voraus, vorbei, vorher, vorweg, vorueber, vorwaerts.

weg, weiter, wieder.

zu, zuvor, zurueck, zugrunde, zusammen.

Die Suffixe sind:
e, se, te, nde, ere, ene, ste, ete, ende, tere, tste, este, ndere, etere, enere,
ndste, etste, enste, endere, endste.
em, tem, ndem, erem, enem, stem, etem, endem, terem, tstem, estem, nderem,
eterem, enerem, ndstem, etstem, enstem, enderem, endstem.
er, ter, nder, erer, ener, ster, eter, ender, terer, tster, ester, nderer, eterer,
enerer, ndster, etster, enster, enderer, endster.
lich.
n, en, in, ien, nen, sen, ten, ern, nden, eren, enen, sten, eten, enden, teren, tsten,
esten, innen nderen, eteren, eneren, ndsten, etsten, ensten, enderen, endsten.
nd, end.
s, es, ns, ses, tes, ens, ndes, eres, enes, stes, etes, endes, teres, tstes, estes,
nderes, eteres, eneres, ndstes, etstes, enstes, enderes, endstes.
t, et, st, tet, est, etet, test, etest.

Die Suchzugriffe auf die Lexika bzw. Wortlisten werden in dem Programm-Modul **lexikon.c** mit den folgenden Funktionen durchgeführt.

```
void lex_access (M_INFO *new_elem)
void lex_access_abkuerz (M_INFO *new_elem)
void compute_lex_search (M_INFO *help)
```

Die Funktion *lex_access* ist für den Lexikonzugriff verantwortlich. Das zusuchende Wort ist in der Variablen 'word' des übergebenen Listenelementes gespeichert. Ist der Lexikonzugriff erfolgreich, wird die Komponente 'code' mit der *MORPHIX*-Kodierung belegt und die restliche Information zu dem Wort in der Variablen 'R_Lex_Entry' gespeichert. Handelt es sich bei dem zusuchenden Wort um ein unregelmäßiges Verb, wird die Information *STAMM* 'Verb-Stamm' ebenfalls in der Variablen 'R_Lex_Entry' gespeichert.

Der Zugriff auf Abkürzungen, die mit einem Punkt enden geschieht mit Hilfe der Funktion *lex_access_abkuerz*. Eine Datei mit Abkürzungen wird zu diesem Zweck sequentiell nach dem Eingabewort durchsucht.

Die Funktion *compute_lex_search* steuert die Lexikazugriffe für alle Elemente der Liste. Für jedes Listenelement wird die obige Funktion *lex_access* aufgerufen.

Die Ausgabe der morphologischen Analyseergebnisse wird durch das Modul **modecode.c** gesteuert. Für jede Wortart von *MORPHIX* bzw. *Morphic-Plus* (auf 22 Wortarten erweitert) gibt es eine eigene Ausgabefunktion.

```
void compute_last_line (char line[MAXLINE])
void compute_code (M_INFO *help)
void comp_adjektiv (M_INFO *help)
```

```
void comp_adverb (M_INFO *help)
void comp_best_artikel (M_INFO *help)
void comp_frage_adverb (M_INFO *help)
void comp_int_rog_pronomen (M_INFO *help)
void comp_kardinal (M_INFO *help)
void comp_konjunktiv (M_INFO *help)
void comp_nomia (M_INFO *help)
void comp_partikel (M_INFO *help)
void comp_pers_pronomen (M_INFO *help)
void comp_poss_pronomen (M_INFO *help)
void comp_praeposition (M_INFO *help)
void comp_reflex_pronomen (M_INFO *help)
void comp_rel_pronomen (M_INFO *help)
void comp_sonderzeichen (M_INFO *help)
void comp_stopwort (M_INFO *help)
void comp_subjunktiv (M_INFO *help)
void comp_unbest_artikel (M_INFO *help)
void comp_verb (M_INFO *help)
void comp_verb_praefix (M_INFO *help)
void comp_namen (M_INFO *help)
void comp_abkuerz (M_INFO *help)
```

Für die Ausgabe der 'restlichen Lexikoninformation' die in der Variablen 'R_Lex_Entry' gespeichert ist wird für verschiedene Wortarten die Funktion *compute_last_line* aufgerufen, die die darin enthaltene Information expandiert.

In der Funktion *compute_code* findet die Ansteuerung der Dekodierungsfunktionen für die einzelnen Wortarten statt. Dies sind im einzelnen:

Wortart	Funktion
Nomina	*comp_nomia*
Verben	*comp_verb*
Adjektive	*comp_adjektiv*
Possessivpronomen	*comp_poss_pronomen*
Verbpräfix	*comp_verb_praefix*
Adverbien	*comp_adverb*
Konjunktive	*comp_konjunktiv*
Parikel	*comp_partikel*
Subjunktive	*comp_subjunktiv*
Präpossitionen	*comp_praeposition*
Relativpronomen	*comp_rel_pronomen*
bestimmte Artikel	*comp_best_artikel*
Reflexivpronomen	*comp_reflex_pronomen*
Personalpronomen	*comp_pers_pronomen*
Interrogationspronomen	*comp_int_rog_pronomen*
Sonderzeichen	*comp_sonderzeichen*
Frageadverbien	*comp_frage_adverb*
unbestimmte Artikel	*comp_unbest_artikel*
Kardinalzahlen	*comp_kardinal*
Stopwörter	*comp_stopwort*
Namen	*comp_namen*
Abkürzungen	*comp_abkuerz*

In diesen Funktionen wird die Ausgabe für das einzelne Analyseergebnis des betreffenden Wortes aufbereitet. Dabei werden die *MORPHIX-Kodierung* und die abgespaltenen Präfixe und Suffixe des Eingabewortes berücksichtigt. Dadurch wird eine implizite Korrektheitsprüfung durchgeführt. In den Funktionen *comp_...*, den Ausgabefunktionen der morphologischen Analyseergebnissen, wird für ein korrektes Ergebnis ein Flag (*ana_flag* der Morphic-Datenstruktur) gesetzt. Alle Elemente der Liste, bei denen dieses Flag nicht gesetzt ist werden mit Hilfe der Funktion *compute_non_ana_entry* anschließend gelöscht.

In einem Programm-Modul **text_tok.c** werden Funktion zur Aufspaltung von Zeichenketten bereitgestellt.

```
int MC_set_tokendelimiter (char *delimiter)
int MC_tokenize (char *instring, MC_Token_type *token_stream)
```

Die Funktion *MC_set_tokendelimiter* ist für die Definition der verwendeten Trennungszeichen, wie Leerzeichen, Tabulatoren, Zeilenschaltung etc., erforderlich.

Die Funktion *MC_tokenmize* zerlegt die Eingabe in einzelne Zeichenketten. Dabei sind die zuvor festgelegten Trennungszeichen für das Aufspalten der Eingabe maßgebend.

Die einzelnen Funktionen für die Komposita-Analyse sind in den Programm-Modulen **komposita.c** und **komp_erg.c** zu finden. Im einzelnen sind dies in **komposita.c**:

```
void komp_init_morphic_ds ()
void komp_new_elem (M_INFO *help);
void komp_init_stamm (M_INFO *help, char *stamm);
M_INFO *komp_dummy (M_INFO *help);
void komp_new_alter (M_INFO *help);
void komp_init_alter_word (M_INFO *help, char *word);
void komp_init_alter_stamm (M_INFO *help, char *stamm);
void komp_init_alter_uml_flag (M_INFO *help);
void komp_init_alter_suffix (M_INFO *help, char *suffix);
int komp_is_umlaut (char ch);
void komp_handle_umlaut (M_INFO *help);
void komp_fugenelement (M_INFO *help);
M_INFO *komp_go_back (M_INFO *help);
void komp_compute_leaf (M_INFO *help);
M_INFO *komp_generate_wordsection (M_INFO *help);
void komp_suffix_separation (M_INFO *help);
void komp_test_output (M_INFO *help, double fk_nr, int step);
void komp_test_out1 (M_INFO *help, double fk_nr);
void komp_test_out2 (M_INFO *help, double fk_nr);
void komp_free_memory (MC_INFO *help);
void komp_analysis (char komposit[MAXWORD]);
```

und in **komp_erg.c** die folgenden Funktionen:

```
int komp_node_leaf (M_INFO *help);
int komp_komp_next (M_INFO *help);
int komp_komp_alt (M_INFO *help);
void komp_go_one_back (M_INFO *help);
int komp_word_analysed (M_INFO *help)
void komp_init_ergebnis_ds ();
void komp_add_result_item (M_INFO *help);
void komp_new_word (MC_INFO *help, char *h_word);
int komp_wordart (char code[8]);
void komp_prepare_result (M_INFO *help);
void komp_generate_result (M_INFO *help);
int komp_proof_result (MC_INFO *help);
void komp_output_res (MC_INFO *help);
```

Die Initialisierung der globalen Variablen *start* der *Morphic-Plus* Datenstruktur für die Kompositaanalyse wird mit der Funktion *komp_init_morphic_ds* durchgeführt.

Die Funktion *komp_new_elem* erzeugt ein neues Listenelement für ein weiteres linkes Teilwort, nachdem ein rechtes Teilwort im Lexikon gefunden wurde. Das neue Element wird über der *alt_word*-Pointer mit dem aktuellen Listenelement verbunden.

Mit der Funktion *komp_init_stamm* wird die Datenstruktur-Komponente *stamm* mit dem noch zuanalysierenden linken Wortteil initialisiert.

komp_dummy erzeugt ein letztes leeres Element der *Morphic-Plus* Datenstruktur, das an jedes Element des Analysebaumes, das keinen Nachfolger mit 'Inhalt' besitzt, angehängt wird. An einem Blatt der Struktur sind zwei solcher Dummyelemente über den *alt_word*- und *komp_word*-Pointer angehängt.

Zum Erzeugen neuer Elemente der Morphic-Datenstruktur dient die Funktion *komp_new_alter*, in dem eine weitere Zerlegungsalternative ihren Ausgangspunkt hat. Das neue Element wird üben dem *komp_word*-Pointer an das aktuelle Element angehängt.

Die Funktionen *komp_init_alter_word*, *komp_init_alter_stamm*, *komp_init_alter_uml_flag* und *komp_init_alter_suffix* werden zur Initialisierung eines neuen Elementes verwendet, das mit Hilfe der Funktion *komp_new_alter* erzeugt wurde (man könnte auch von einer partiellen Duplikation des Elementes sprechen).

Die Funktion *komp_is_umlaut* ist ein Prädikat zur Entscheidung, ob es sich bei dem aktuellen Buchstaben um einen Umlaut (Ä, ä, Ö, ö, Ü oder ü) oder ein 'Scharfes-S' (ß) handelt.

Mit der Funktion *komp_handle_umlaut* werden die Umlaute *ä*, *ö*, *ü* und das Scharfe-S *ß* in ihre Doppelvokal-Schreibweisen *ae*, *oe ue* oder Doppelkonsonant-Schreibweise *ss* überführt. Es findet auch gleichzeitig eine Umlautreduktion statt, bei der das Umlaut *e* entfernt und mit dem so entstandenen Wort eine neue Alternative erzeugt wird, die über den *komp_word*-Pointer an das aktuellen Element angehängt wird.

Die Behandlung der Fugenzeichen, *s*, *e*, *n*, *en*, *er*, *es*, *ns* und *ens* (Aufzählung nach [Dro84]) wird für das linke Teilwort angestossen, wenn das rechte im *MORPHIX*-Lexikon eingetragen ist. Die Abspaltung eines Fugenzeichens mit der Funktion *komp_fugenelement* erzeugt ein neues Element, das als Kompositaalternative (*komp_word*-Pointer) am aktuellen Element angehängt wird. Das Fugenzeichen wird als Suffix des Wortes in der Datenstruktur des neuen Elementes gespeichert.

Die Funktion *komp_go_back* wird benötigt um den erzeugten Baum bei der Zerlegung des Kompositums 'rekursiv' zu durchlaufen und alle möglichen

Aufspaltungen zu erzeugen. Dabei wird über den *alt_word*-Pointer erst in die 'Breite' und auf dem Rückweg über den *komp_word*-Pointer in die 'Tiefe' gegangen.

Wurde das Eingabe-Wort soweit zerlegt, daß kein weiterer Buchstaben zum Anhängen an das Teilwort (Bestimmungswort) vorhanden ist und wurde kein Lexikoneintrag für das Wort gefunden, wird nun versucht durch Abspaltung von Präfixen das Wort zu analysieren. Dies wird mit Hilfe der Funktion *komp_compute_leaf* durchgeführt.

Die Hauptfunktion für die Zerlegung des Eingabewortes ist die Funktion *komp_generate_wordsection*. Sie spaltet die Buchstaben zeichenweise von der Eingabe (linkes Teilwort) ab und fügt sie an das aktuelle rechte Teilwort vorne an. Nach jedem Anhängen eines Zeichens wird überprüft, ob das gebildete Teilwort im *MORPHIX*-Lexikon enthalten ist. Wird ein Eintrag gefunden so werden zwei neue Elemente generiert und an das aktuelle Element angehängt. Ein Duplikat des aktuellen Elementes wird über den *komp_word*-Pointer angehängt und ein neues Element mit dem linken Teilwort (Restwort) wird über den *alt_word*-Pointer mit dem aktuellen Element verkettet. Danach wird mit dem letzteren die Zerlegung fortgeführt. Dies wird solange wiederholt bis kein Zeichen mehr im linken Teilwort vorhanden ist. Ist zu diesem Zeitpunkt noch kein Ergebnis (d.h. kein Lexikon-Eintrag für das Wort vorhanden), wird für dieses Teilwort die Funktion *komp_compute_leaf* aufgerufen. Ansonsten wird mit Hilfe der oben beschriebenen Funktion *komp_go_back* in noch nicht bearbeitete Alternativen verzweigt und die weitere Zerlegung dieser durchgeführt.

Zu Beginn der Komposita-Analyse wird eine Suffix-Abspaltung wie in *Morphic-Plus* durchgeführt. Die Steuerung übernimmt die Funktion *komp_suffix_separation*. Die generierten Alternativen werden als Kompositaalternative (*komp_word*-Pointer) am aktuellen Element (der Wurzel) angehängt.

Zur Anzeige von Zwischenergebnissen und Debug-Unterstützung der Kompositaanalyse (Ausgabe des Inhalts der Listenelemente) werden die Funktionen *komp_test_output*, *komp_test_out1* und *komp_test_out2* verwendet.

Zur Freigabe des Speicherplatzes, der bei der Wortzerlegung reserviert wurde dient die Funktion *komp_free_memory*.

Die Steuerung der Kompositaanalyse wird von der Funktion *komp_analysis* durchgeführt. In der Standalone-Version der Kompositaanalyse ist dies die *main*-Funktion.

Für die Generierung der Ergebnisse der Wortzerlegung stehen die nachfolgenden Funktionen zur Verfügung:

Die Funktion *komp_node_leaf* überprüft, ob ein Element ein Blatt ist. Ein Blatt hat in diesem Falle zwei *Dummy*-Elemente als Nachfolger.

Mit Hilfe der Funktionen *komp_komp_next* und *komp_komp_alt* wird geprüft, ob das Element einen Nachfolger hat, der über den *komp_word*- oder *alt_word*-Pointer mit diesem verbunden ist.

komp_go_one_back wird verwendet um festzustellen über welchen Pointer das aktuelle Element mit seinem Vorgänger verbunden ist. Dies entscheidet, welches Element in die Ergebnisliste für eine Zerlegungsalternative aufgenommen wird.

Mit der Funktion *komp_init_ergebnis_ds* wird eine *Morphic-Plus* Datenstruktur für die Aufnahme einer Zerlegungsalternativen generiert.

Die Funktionen *komp_add_result_item* und *komp_new_word* erzeugen eine Datenstruktur, wie sie bei der morphologischen Analyse von Sätzen oder Texten benötigt wird. Sie fügen für jedes neue Teilwort ein weiteres Element mit den Analyseergebnissen am Anfang der Liste ein. Damit wird erreicht, daß das Grundwort das erste Element in der Liste ist und das Bestimmungswort des Kompositums das letzte.

Zur Bestimmung der Wortart eines Wortes werden in der Funktion *komp_wordart* die erste bzw. die zwei ersten Stellen der Variable *code* betrachtet. Sie bildet die Entscheidungsgrundlage ob ein Teilwort einer gültigen Wortklasse (Nomen, Verben oder Adjektive) angehört.

Die Funktion *komp_prepare_result* durchläuft den Zerlegungsbaum von der Wurzel bis zu einem 'Blatt', ab dem auf dem Rückweg das Analyseergebnis zusammengesetzt wird. Der Weg beim Einsinken in den Baum wird durch die Variable *ana_flag* der einzelnen Elemente bestimmt. Dabei bedeutet der Wert **5** der nächste Knoten ist über den *alt_word*-Pointer zu erreichen und der Wert **6** legt den Weg über den *komp_word*-Pointer fest.

komp_generate_result erzeugt beim Aufstieg im Analysebaum eine Zerlegungsalternative des Eingabewortes. Dabei werden alle Elemente in eine Ergebnisliste aufgenommen, die über einen *alt_word*-Pointer mit dem Vorgänger verbunden sind.

Die generierte Analyseliste wird mit Hilfe der Funktion *komp_proof_result* auf die korrekte Wortart der Einzelelemente geprüft. Listen die andere Teilkomponenten als *Nomen*, *Verben* oder *Adjektive* besitzen werden als Lösung verworfen.

Das Analyseergebnis der Kompositazerlegung wird mit Hilfe der Funktion *komp_output_res* aufbereitet und angezeigt. Die Ausgabe wird von den früher beschriebenen Funktionen *comp_nomina*, *comp_verb* oder *comp_adjectiv* erzeugt.

D Daten zur Performanz von *Morphic-Plus*

Die Laufzeiten wurden auf einer *SPARCstation-20* unter dem Betriebsystem *Solaris 2.4* ermittelt.

Programm	Anz. Worte des Textes	Lexikongröße (Anz. Einträge)	Anzahl Lexikonzugriffe	Zugriffe/Wort	Worte/sec. analysiert
k_morphic	38.856	52.460	618.115	15,91	80,0
k_morphic	27.294	-"-	378.698	13,87	88,9
morphic	38.856	-"-	158.895	4,09	150,6
morphic	27.294	-"-	112.095	4,11	153,7
k_morphic	38.856	11.473	546.953	14,08	123,7
k_morphic	27.294	-"-	334.496	12,26	133,0
morphic	38.856	-"-	162.204	4,17	177,1
morphic	27.294	-"-	114.509	4,2	176,8

Wie bereits in der Beschreibung der Komposita-Analyse (Kapitel 5) angeführt wurde, ist die Komposita-Zerlegung rechenintensiv. Es werden bei der Analyse eines Wortes im Durchschnitt zwischen 12 und 16 Zugriffe auf die Hashtabelle durchgeführt. Betrachtet man nur die Anzahl der Zugriffe bei Wörtern für die eine Komposita-Zerlegung durgeführt wird so erhält man leicht rund 88 Hashtabellen-Zugriffe je Wort (Text mit 38.856 Wörttern, 5.187 Wörter mit Kompositabehandlung und 457.497 Lexikonzugriffe).
Im Zuge einer Weiterentwicklung von *Morphic-Plus* ist hier ein Ansatzpunkt für Laufzeit- und Performanzoptimierungen. Man könnte sich die folgenden Ansätze für Verbesserungen vorstellen:

- Einschränkung der Worte für die die Komposita-Analyse gestartet wird.
- Einbau eines 'Analyse-Caches', der die wiederholte Analyse eines Wortes verhindert.
- Arbeiten mit einer anderen Hash-Funktion, die die Tabellen komplett im Hauptspeicher hält. (Zur Zeit auf Sekundärspeicher/Platte)

Literatur

[Agn95] Stefan Agne. Regelbasierte Dokumentklassifikation, 1995. Diplomarbeit im Fachbereich Informatik, Universität Kaiserslautern.

[Bus83] S. Busemann. Oberflächentransformationen bei der automatischen Generierung geschriebener deutscher Sprache — Entwurf und Implementierung des modularen und anpaßbaren Systems SUTRA. Master's thesis, Universität Hamburg, Fachbereich Informatik, 1983.

[Dro84] Günther Drosdowski. *Duden, Die Grammatik.* Enzyklopädie. Dudenverlag Mannheim, 4 edition, 1984.

[Dro91] Günther Drosdowski. *Duden, Die deutsche Rechtschreibung.* Enzyklopädie. Dudenverlag Mannheim, 20 edition, 1991.

[FN86] Wolfgang Finkler and Günter Neumann. MORPHIX Ein hochportabler Lemmatisierungsmodul für das Deutsche. Memo 8, Universität des Saarlandes, KI — Labor am Lehrstuhl für Informatik IV, 1986.

[FN91] Wolfgang Finkler and Günter Neumann. *MORPHIX 3.0 Online Manual.* Universität des Saarlandes, KI — Labor am Lehrstuhl für Informatik IV, 1991.

[Ges94] Gesellschaft für Linguistische Datenverarbeitung GLDV. *1. Morpholympics für Wortformerkennung*, 1994. Noch nicht erschienen.

[HB81] G. Helbig and J. Buscha. *Deutsche Grammatik.* Enzyklopädie. VEB Verlag Leipzig, 1981.

[KH93] Thomas Kieninger and Rainer Hoch. Ein Generator mit Anfragesystem für strukturierte Wörterbücher zur Unterstützung von Texterkennung und Textanalyse. Document D-93-08, DFKI Deutsches Forschungszentrum für Künstliche Intelligenz GMBH, Kaiserslautern, 1993.

[Len94] Winfried Lenders. MORPHOLYMPICS - ein Unternehmen der GLDV. *LDV - Forum*, 11(1):5–64, 6 1994.

[Sch72] G. Schott. Automatic Analysis of Inflectional Morphems in German Nouns. *Acta Informatica*, 1:360–374, 1972.

[Sri93] V. Srinivasan. Punctuation and Parsing of Real-World Texts. In A. Nijholt K. Sikkel, editor, *TWLT 6, Parsing Natural Language*, pages 163–167, 1993.

[Ste94] Michael Stein. Weiterentwicklung der morphologischen Komponente MORPHIX, 1994. Projektarbeit im Fachbereich Informatik, Universität Kaiserslautern.

Deutsches
Forschungszentrum
für Künstliche
Intelligenz GmbH

-Bibliothek, Information
und Dokumentation (BID)-
PF 2080
67608 Kaiserslautern
FRG

Telefon (0631) 205-3506
Telefax (0631) 205-3210

e-mail
dfkibib@dfki.uni-kl.de
WWW
http://www.dfki.uni-sb.de/dfkibib

Veröffentlichungen des DFKI

Die folgenden DFKI Veröffentlichungen sowie die aktuelle Liste von allen bisher erschienenen Publikationen können von der oben angegebenen Adresse oder (so sie als per ftp erhaeltlich angemerkt sind) per anonymous ftp von ftp.dfki.uni-kl.de (131.246.241.100) im Verzeichnis pub/Publications bezogen werden. Die Berichte werden, wenn nicht anders gekennzeichnet, kostenlos abgegeben.

DFKI Publications

The following DFKI publications or the list of all published papers so far are obtainable from the above address or (if they are marked as obtainable by ftp) by anonymous ftp from ftp.dfki.uni-kl.de (131.246.241.100) in the directory pub/Publications.
The reports are distributed free of charge except where otherwise noted.

DFKI Research Reports

1995

RR-95-07
Francesco M. Donini, Maurizio Lenzerini. Daniele Nardi, Werner Nutt
The Complexity of Concept Languages
57 pages

RR-95-04
M. Buchheit, H.-J. Bürckert, B. Hollunder, A. Laux, W. Nutt, M. Wojcik
Task Acquisition with a Description Logic Reasoner
17 pages

RR-95-03
Stephan Baumann, Michael Malburg, Hans-Guenther Hein, Rainer Hoch, Thomas Kieninger, Norbert Kuhn
Document Analysis at DFKI
Part 2: Information Extraction
40 pages

RR-95-02
Majdi Ben Hadj Ali, Frank Fein, Frank Hoenes, Thorsten Jaeger, Achim Weigel
Document Analysis at DFKI
Part 1: Image Analysis and Text Recognition
69 pages

1994

RR-94-39
Hans-Ulrich Krieger
Typed Feature Formalisms as a Common Basis for Linguistic Specification.
21 pages

RR-94-38
Hans Uszkoreit, Rolf Backofen, Stephan Busemann, Abdel Kader Diagne, Elizabeth A. Hinkelman, Walter Kasper, Bernd Kiefer, Hans-Ulrich Krieger, Klaus Netter, Günter Neumann, Stephan Oepen, Stephen P. Spackman.
DISCO–An HPSG-based NLP System and its Application for Appointment Scheduling.
13 pages

RR-94-37
Hans-Ulrich Krieger, Ulrich Schaefer
TDL - A Type Description Language for HPSG, Part 1: Overview.
54 pages

RR-94-36
Manfred Meyer
Issues in Concurrent Knowledge Engineering. Knowledge Base and Knowledge Share Evolution.
17 pages

RR-94-35
Rolf Backofen
A Complete Axiomatization of a Theory with Feature and Arity Constraints
49 pages

RR-94-34
Stephan Busemann, Stephan Oepen, Elizabeth A. Hinkelman, Günter Neumann, Hans Uszkoreit
COSMA - Multi-Participant NL Interaction for Appointment Scheduling
80 pages

RR-94-33
Franz Baader, Armin Laux
Terminological Logics with Modal Operators
29 pages

RR-94-31
Otto Kühn, Volker Becker, Georg Lohse, Philipp Neumann
Integrated Knowledge Utilization and Evolution for the Conservation of Corporate Know-How
17 pages

RR-94-23
Gert Smolka
The Definition of Kernel Oz
53 pages

RR-94-20
Christian Schulte, Gert Smolka, Jörg Würtz
Encapsulated Search and Constraint Programming in Oz
21 pages

RR-94-18
Rolf Backofen, Ralf Treinen
How to Win a Game with Features
18 pages

RR-94-17
Georg Struth
Philosophical Logics—A Survey and a Bibliography
58 pages

RR-94-16
Gert Smolka
A Foundation for Higher-order Concurrent Constraint Programming
26 pages

RR-94-15
Winfried H. Graf, Stefan Neurohr
Using Graphical Style and Visibility Constraints for a Meaningful Layout in Visual Programming Interfaces
20 pages

RR-94-14
Harold Boley, Ulrich Buhrmann, Christof Kremer
Towards a Sharable Knowledge Base on Recyclable Plastics
14 pages

RR-94-13
Jana Koehler
Planning from Second Principles—A Logic-based Approach
49 pages

RR-94-12
Hubert Comon, Ralf Treinen
Ordering Constraints on Trees
34 pages

RR-94-11
Knut Hinkelmann
A Consequence Finding Approach for Feature Recognition in CAPP
18 pages

RR-94-10
Knut Hinkelmann, Helge Hintze
Computing Cost Estimates for Proof Strategies
22 pages

RR-94-08
Otto Kühn, Björn Höfling
Conserving Corporate Knowledge for Crankshaft Design
17 pages

RR-94-07
Harold Boley
Finite Domains and Exclusions as First-Class Citizens
25 pages

RR-94-06
Dietmar Dengler
An Adaptive Deductive Planning System
17 pages

RR-94-05
Franz Schmalhofer, J. Stuart Aitken, Lyle E. Bourne jr.
Beyond the Knowledge Level: Descriptions of Rational Behavior for Sharing and Reuse
81 pages

RR-94-03
Gert Smolka
A Calculus for Higher-Order Concurrent Constraint Programming with Deep Guards
34 pages

RR-94-02
Elisabeth André, Thomas Rist
Von Textgeneratoren zu Intellimedia-Präsentationssystemen
22 Seiten

RR-94-01
Elisabeth André, Thomas Rist
Multimedia Presentations: The Support of Passive and Active Viewing
15 pages

1993

RR-93-48
Franz Baader, Martin Buchheit, Bernhard Hollunder
Cardinality Restrictions on Concepts
20 pages

RR-93-46
Philipp Hanschke
A Declarative Integration of Terminological, Constraint-based, Data-driven, and Goal-directed Reasoning
81 pages

RR-93-45
Rainer Hoch
On Virtual Partitioning of Large Dictionaries for Contextual Post-Processing to Improve Character Recognition
21 pages

RR-93-44
Martin Buchheit, Manfred A. Jeusfeld, Werner Nutt, Martin Staudt
Subsumption between Queries to Object-Oriented Databases
36 pages

RR-93-43
M. Bauer, G. Paul
Logic-based Plan Recognition for Intelligent Help Systems
15 pages

RR-93-42
Hubert Comon, Ralf Treinen
The First-Order Theory of Lexicographic Path Orderings is Undecidable
9 pages

RR-93-41
Winfried H. Graf
LAYLAB: A Constraint-Based Layout Manager for Multimedia Presentations
9 pages

RR-93-40
Francesco M. Donini, Maurizio Lenzerini, Daniele Nardi, Werner Nutt, Andrea Schaerf
Queries, Rules and Definitions as Epistemic Statements in Concept Languages
23 pages

RR-93-38
Stephan Baumann
Document Recognition of Printed Scores and Transformation into MIDI
24 pages

RR-93-36
Michael M. Richter, Bernd Bachmann, Ansgar Bernardi, Christoph Klauck, Ralf Legleitner, Gabriele Schmidt
Von IDA bis IMCOD: Expertensysteme im CIM-Umfeld
13 Seiten

RR-93-35
Harold Boley, François Bry, Ulrich Geske (Eds.)
Neuere Entwicklungen der deklarativen KI-Programmierung — *Proceedings*
150 Seiten

Note: This document is available for a nominal charge of 25 DM (or 15 US-$).

RR-93-34
Wolfgang Wahlster
Verbmobil Translation of Face-To-Face Dialogs
10 pages

RR-93-33
Bernhard Nebel, Jana Koehler
Plan Reuse versus Plan Generation: A Theoretical and Empirical Analysis
33 pages

RR-93-32
David R. Traum, Elizabeth A. Hinkelman
Conversation Acts in Task-Oriented Spoken Dialogue
28 pages

RR-93-31
Elizabeth A. Hinkelman, Stephen P. Spackman
Abductive Speech Act Recognition, Corporate Agents and the COSMA System
34 pages

RR-93-30
Stephen P. Spackman, Elizabeth A. Hinkelman
Corporate Agents
14 pages

RR-93-29
Armin Laux
Representing Belief in Multi-Agent Worlds via Terminological Logics
35 pages

RR-93-28
Hans-Ulrich Krieger, John Nerbonne, Hannes Pirker
Feature-Based Allomorphy
8 pages

RR-93-27
Hans-Ulrich Krieger
Derivation Without Lexical Rules
33 pages

RR-93-26
Jörg P. Müller, Markus Pischel
The Agent Architecture InteRRaP: Concept and Application
99 pages

RR-93-25
Klaus Fischer, Norbert Kuhn
A DAI Approach to Modeling the Transportation Domain
93 pages

RR-93-24
Rainer Hoch, Andreas Dengel
Document Highlighting — Message Classification in Printed Business Letters
17 pages

RR-93-23
Andreas Dengel, Ottmar Lutzy
Comparative Study of Connectionist Simulators
20 pages

RR-93-22
Manfred Meyer, Jörg Müller
Weak Looking-Ahead and its Application in Computer-Aided Process Planning
17 pages

RR-93-20
Franz Baader, Bernhard Hollunder
Embedding Defaults into Terminological Knowledge Representation Formalisms
34 pages

RR-93-18
Klaus Schild
Terminological Cycles and the Propositional μ-Calculus
32 pages

RR-93-17
Rolf Backofen
Regular Path Expressions in Feature Logic
37 pages

RR-93-16
Gert Smolka, Martin Henz, Jörg Würtz
Object-Oriented Concurrent Constraint Programming in Oz
17 pages

RR-93-15
Frank Berger, Thomas Fehrle, Kristof Klöckner, Volker Schölles, Markus A. Thies, Wolfgang Wahlster
PLUS - Plan-based User Support Final Project Report
33 pages

RR-93-14
Joachim Niehren, Andreas Podelski, Ralf Treinen
Equational and Membership Constraints for Infinite Trees
33 pages

RR-93-13
Franz Baader, Karl Schlechta
A Semantics for Open Normal Defaults via a Modified Preferential Approach
25 pages

RR-93-12
Pierre Sablayrolles
A Two-Level Semantics for French Expressions of Motion
51 pages

RR-93-11
Bernhard Nebel, Hans-Jürgen Bürckert
Reasoning about Temporal Relations: A Maximal Tractable Subclass of Allen's Interval Algebra
28 pages

RR-93-10
Martin Buchheit, Francesco M. Donini, Andrea Schaerf
Decidable Reasoning in Terminological Knowledge Representation Systems
35 pages

RR-93-09
Philipp Hanschke, Jörg Würtz
Satisfiability of the Smallest Binary Program
8 pages

RR-93-08
Harold Boley, Philipp Hanschke, Knut Hinkelmann, Manfred Meyer
CoLab: A Hybrid Knowledge Representation and Compilation Laboratory
64 pages

RR-93-07
Hans-Jürgen Bürckert, Bernhard Hollunder, Armin Laux
Concept Logics with Function Symbols
36 pages

RR-93-06
Hans-Jürgen Bürckert, Bernhard Hollunder, Armin Laux
On Skolemization in Constrained Logics
40 pages

RR-93-05
Franz Baader, Klaus Schulz
Combination Techniques and Decision Problems for Disunification
29 pages

RR-93-04
Christoph Klauck, Johannes Schwagereit
GGD: Graph Grammar Developer for features in CAD/CAM
13 pages

RR-93-03
Franz Baader, Berhard Hollunder, Bernhard Nebel, Hans-Jürgen Profitlich, Enrico Franconi
An Empirical Analysis of Optimization Techniques for Terminological Representation Systems
28 pages

RR-93-02
Wolfgang Wahlster, Elisabeth André, Wolfgang Finkler, Hans-Jürgen Profitlich, Thomas Rist
Plan-based Integration of Natural Language and Graphics Generation
50 pages

RR-93-01
Bernhard Hollunder
An Alternative Proof Method for Possibilistic Logic and its Application to Terminological Logics
25 pages

DFKI Technical Memos

1995

TM-95-01
Martin Buchheit, Rüdiger Klein, Werner Nutt
Constructive Problem Solving: A Model Construction Approach towards Configuration
34 pages

1994

TM-94-04
Cornelia Fischer
PAntUDE – An Anti-Unification Algorithm for Expressing Refined Generalizations
22 pages

TM-94-03
Victoria Hall
Uncertainty-Valued Horn Clauses
31 pages

TM-94-02
Rainer Bleisinger, Berthold Kröll
Representation of Non-Convex Time Intervals and Propagation of Non-Convex Relations
11 pages

TM-94-01
Rainer Bleisinger, Klaus-Peter Gores
Text Skimming as a Part in Paper Document Understanding
14 pages

1993

TM-93-05
Michael Sintek
Indexing PROLOG Procedures into DAGs by Heuristic Classification
64 pages

TM-93-04
Hans-Günther Hein
Propagation Techniques in WAM-based Architectures — The FIDO-III Approach
105 pages

TM-93-03
Harold Boley, Ulrich Buhrmann, Christof Kremer
Konzeption einer deklarativen Wissensbasis über recyclingrelevante Materialien
11 pages

TM-93-02
Pierre Sablayrolles, Achim Schupeta
Conflict Resolving Negotiation for COoperative Schedule Management Agents (COSMA)
21 pages

TM-93-01
Otto Kühn, Andreas Birk
Reconstructive Integrated Explanation of Lathe Production Plans
20 pages

DFKI Documents

1995

D-95-03
Christoph Endres, Lars Klein, Markus Meyer
Implementierung und Erweiterung der Sprache $\mathcal{ALCP}$
110 Seiten

D-95-02
Andreas Butz
BETTY
Ein System zur Planung und Generierung informativer Animationssequenzen
95 Seiten

RR-93-25
Klaus Fischer, Norbert Kuhn
A DAI Approach to Modeling the Transportation Domain
93 pages

RR-93-24
Rainer Hoch, Andreas Dengel
Document Highlighting — Message Classification in Printed Business Letters
17 pages

RR-93-23
Andreas Dengel, Ottmar Lutzy
Comparative Study of Connectionist Simulators
20 pages

RR-93-22
Manfred Meyer, Jörg Müller
Weak Looking-Ahead and its Application in Computer-Aided Process Planning
17 pages

RR-93-20
Franz Baader, Bernhard Hollunder
Embedding Defaults into Terminological Knowledge Representation Formalisms
34 pages

RR-93-18
Klaus Schild
Terminological Cycles and the Propositional μ-Calculus
32 pages

RR-93-17
Rolf Backofen
Regular Path Expressions in Feature Logic
37 pages

RR-93-16
Gert Smolka, Martin Henz, Jörg Würtz
Object-Oriented Concurrent Constraint Programming in Oz
17 pages

RR-93-15
Frank Berger, Thomas Fehrle, Kristof Klöckner, Volker Schölles, Markus A. Thies, Wolfgang Wahlster
PLUS - Plan-based User Support Final Project Report
33 pages

RR-93-14
Joachim Niehren, Andreas Podelski, Ralf Treinen
Equational and Membership Constraints for Infinite Trees
33 pages

RR-93-13
Franz Baader, Karl Schlechta
A Semantics for Open Normal Defaults via a Modified Preferential Approach
25 pages

RR-93-12
Pierre Sablayrolles
A Two-Level Semantics for French Expressions of Motion
51 pages

RR-93-11
Bernhard Nebel, Hans-Jürgen Bürckert
Reasoning about Temporal Relations: A Maximal Tractable Subclass of Allen's Interval Algebra
28 pages

RR-93-10
Martin Buchheit, Francesco M. Donini, Andrea Schaerf
Decidable Reasoning in Terminological Knowledge Representation Systems
35 pages

RR-93-09
Philipp Hanschke, Jörg Würtz
Satisfiability of the Smallest Binary Program
8 pages

RR-93-08
Harold Boley, Philipp Hanschke, Knut Hinkelmann, Manfred Meyer
CoLab: A Hybrid Knowledge Representation and Compilation Laboratory
64 pages

RR-93-07
Hans-Jürgen Bürckert, Bernhard Hollunder, Armin Laux
Concept Logics with Function Symbols
36 pages

RR-93-06
Hans-Jürgen Bürckert, Bernhard Hollunder, Armin Laux
On Skolemization in Constrained Logics
40 pages

RR-93-05
Franz Baader, Klaus Schulz
Combination Techniques and Decision Problems for Disunification
29 pages

RR-93-04
Christoph Klauck, Johannes Schwagereit
GGD: Graph Grammar Developer for features in CAD/CAM
13 pages

RR-93-03
Franz Baader, Berhard Hollunder, Bernhard Nebel, Hans-Jürgen Profitlich, Enrico Franconi
An Empirical Analysis of Optimization Techniques for Terminological Representation Systems
28 pages

RR-93-02
Wolfgang Wahlster, Elisabeth André, Wolfgang Finkler, Hans-Jürgen Profitlich, Thomas Rist
Plan-based Integration of Natural Language and Graphics Generation
50 pages

RR-93-01
Bernhard Hollunder
An Alternative Proof Method for Possibilistic Logic and its Application to Terminological Logics
25 pages

DFKI Technical Memos

1995

TM-95-01
Martin Buchheit, Rüdiger Klein, Werner Nutt
Constructive Problem Solving: A Model Construction Approach towards Configuration
34 pages

1994

TM-94-04
Cornelia Fischer
PAntUDE – An Anti-Unification Algorithm for Expressing Refined Generalizations
22 pages

TM-94-03
Victoria Hall
Uncertainty-Valued Horn Clauses
31 pages

TM-94-02
Rainer Bleisinger, Berthold Kröll
Representation of Non-Convex Time Intervals and Propagation of Non-Convex Relations
11 pages

TM-94-01
Rainer Bleisinger, Klaus-Peter Gores
Text Skimming as a Part in Paper Document Understanding
14 pages

1993

TM-93-05
Michael Sintek
Indexing PROLOG Procedures into DAGs by Heuristic Classification
64 pages

TM-93-04
Hans-Günther Hein
Propagation Techniques in WAM-based Architectures — The FIDO-III Approach
105 pages

TM-93-03
Harold Boley, Ulrich Buhrmann, Christof Kremer
Konzeption einer deklarativen Wissensbasis über recyclingrelevante Materialien
11 pages

TM-93-02
Pierre Sablayrolles, Achim Schupeta
Conflict Resolving Negotiation for COoperative Schedule Management Agents (COSMA)
21 pages

TM-93-01
Otto Kühn, Andreas Birk
Reconstructive Integrated Explanation of Lathe Production Plans
20 pages

DFKI Documents

1995

D-95-03
Christoph Endres, Lars Klein, Markus Meyer
Implementierung und Erweiterung der Sprache $\mathcal{ALCP}$
110 Seiten

D-95-02
Andreas Butz
BETTY
Ein System zur Planung und Generierung informativer Animationssequenzen
95 Seiten

D-95-01
Susanne Biundo, Wolfgang Tank (Hrsg.)
Beiträge zum Workshop „Planen und Konfigurieren", Februar 1995
169 Seiten

Note: This document is available for a nominal charge of 25 DM (or 15 US-$).

1994

D-94-15
Stephan Oepen
German Nominal Syntax in HPSG
— On Syntactic Categories and Syntagmatic Relations —
80 pages

D-94-14
Hans-Ulrich Krieger, Ulrich Schaefer.
TDL - A Type Description Language for HPSG, Part 2: User Guide.
72 pages

D-94-12
Arthur Sehn, Serge Autexier (Hrsg.)
Proceedings des Studentenprogramms der 18. Deutschen Jahrestagung für Künstliche Intelligenz KI-94
69 Seiten

D-94-11
F. Baader, M. Buchheit, M. A. Jeusfeld, W. Nutt (Eds.)
Working Notes of the KI'94 Workshop: KRDB'94 - Reasoning about Structured Objects: Knowledge Representation Meets Databases
65 pages

Note: This document is no longer available in printed form.

D-94-10
F. Baader, M. Lenzerini, W. Nutt, P. F. Patel-Schneider (Eds.)
Working Notes of the 1994 International Workshop on Description Logics
118 pages

Note: This document is available for a nominal charge of 25 DM (or 15 US-$).

D-94-09
Technical Staff
DFKI Wissenschaftlich-Technischer Jahresbericht 1993
145 Seiten

D-94-08
Harald Feibel
IGLOO 1.0 - Eine grafikunterstützte Beweisentwicklungsumgebung
58 Seiten

D-94-07
Claudia Wenzel, Rainer Hoch
Eine Übersicht über Information Retrieval (IR) und NLP-Verfahren zur Klassifikation von Texten
25 Seiten

D-94-06
Ulrich Buhrmann
Erstellung einer deklarativen Wissensbasis über recyclingrelevante Materialien
117 Seiten

D-94-04
Franz Schmalhofer, Ludger van Elst
Entwicklung von Expertensystemen: Prototypen, Tiefenmodellierung und kooperative Wissensevolution
22 Seiten

D-94-03
Franz Schmalhofer
Maschinelles Lernen: Eine kognitionswissenschaftliche Betrachtung
54 Seiten

Note: This document is no longer available in printed form.

D-94-02
Markus Steffens
Wissenserhebung und Analyse zum Entwicklungsprozeß eines Druckbehälters aus Faserverbundstoff
90 pages

D-94-01
Josua Boon (Ed.)
DFKI-Publications: The First Four Years
1990 - 1993
75 pages

1993

D-93-27
Rolf Backofen, Hans-Ulrich Krieger, Stephen P. Spackman, Hans Uszkoreit (Eds.)
Report of the EAGLES Workshop on Implemented Formalisms at DFKI, Saarbrücken
110 pages

D-93-26
Frank Peters
Unterstützung des Experten bei der Formalisierung von Textwissen INFOCOM - Eine interaktive Formalisierungskomponente
58 Seiten

D-93-25
Hans-Jürgen Bürckert, Werner Nutt (Eds.)
Modeling Epistemic Propositions
118 pages

Note: This document is available for a nominal charge of 25 DM (or 15 US-$).

D-93-24
Brigitte Krenn, Martin Volk
DiTo-Datenbank: Datendokumentation zu Funktionsverbgefügen und Relativsätzen
66 Seiten

D-93-22
Andreas Abecker
Implementierung graphischer Benutzungsoberflächen mit Tcl/Tk und Common Lisp
44 Seiten

Note: This document is no longer available in printed form.

D-93-21
Dennis Drollinger
Intelligentes Backtracking in Inferenzsystemen am Beispiel Terminologischer Logiken
53 Seiten

D-93-20
Bernhard Herbig
Eine homogene Implementierungsebene für einen hybriden Wissensrepräsentationsformalismus
97 Seiten

D-93-16
Bernd Bachmann, Ansgar Bernardi, Christoph Klauck, Gabriele Schmidt
Design & KI
74 Seiten

D-93-15
Robert Laux
Untersuchung maschineller Lernverfahren und heuristischer Methoden im Hinblick auf deren Kombination zur Unterstützung eines Chart-Parsers
86 Seiten

D-93-14
Manfred Meyer (Ed.)
Constraint Processing - Proceedings of the International Workshop at CSAM'93, St.Petersburg, July 20-21, 1993
264 pages

Note: This document is available for a nominal charge of 25 DM (or 15 US-$).

D-93-12
Harold Boley, Klaus Elsbernd, Michael Herfert, Michael Sintek, Werner Stein
RELFUN Guide: Programming with Relations and Functions Made Easy
86 pages

D-93-11
Knut Hinkelmann, Armin Laux (Eds.)
DFKI Workshop on Knowledge Representation Techniques — Proceedings
88 pages

Note: This document is no longer available in printed form.

D-93-10
Elizabeth Hinkelman, Markus Vonerden, Christoph Jung
Natural Language Software Registry (Second Edition)
174 pages

D-93-09
Hans-Ulrich Krieger, Ulrich Schäfer
TDLExtraLight User's Guide
35 pages

D-93-08
Thomas Kieninger, Rainer Hoch
Ein Generator mit Anfragesystem für strukturierte Wörterbücher zur Unterstützung von Texterkennung und Textanalyse
125 Seiten

D-93-07
Klaus-Peter Gores, Rainer Bleisinger
Ein erwartungsgesteuerter Koordinator zur partiellen Textanalyse
53 Seiten

D-93-06
Jürgen Müller (Hrsg.)
Beiträge zum Gründungsworkshop der Fachgruppe Verteilte Künstliche Intelligenz, Saarbrücken, 29. - 30. April 1993
235 Seiten

Note: This document is available for a nominal charge of 25 DM (or 15 US-$).

D-93-05
Elisabeth André, Winfried Graf, Jochen Heinsohn, Bernhard Nebel, Hans-Jürgen Profitlich, Thomas Rist, Wolfgang Wahlster
PPP: Personalized Plan-Based Presenter
70 pages

D-93-04
Technical Staff
DFKI Wissenschaftlich-Technischer Jahresbericht 1992
194 Seiten

D-93-03
Stephan Busemann, Karin Harbusch(Eds.)
DFKI Workshop on Natural Language Systems: Reusability and Modularity - Proceedings
74 pages

D-93-02
Gabriele Schmidt, Frank Peters, Gernod Laufkötter
User Manual of COKAM+
23 pages

D-93-01
Philipp Hanschke, Thom Frühwirth
Terminological Reasoning with Constraint Handling Rules
12 pages

Morphic - Plus
Ein morphologisches Analyseprogramm für die deutsche Flexionsmorphologie und Komposita-Analyse

Ottmar Lutzy

D-95-07
Document