Aufbau und Regelformat von DeKo

Bettina Säuberlich,
Institut für Maschinelle Sprachverarbeitung
Azenbergstr. 12, D-70174 Stuttgart

7. Juli 2001

1 Gesamtaufbau des DeKo-Tools

Das DeKo-Tool besteht aus mehreren Modulen. Zunächst wird das Eingabewort in einzelne Morpheme zerlegt (markiert durch [++]). Auf diesen Morphemstring wird versucht, durch Wortbildungsregeln, eine hierarchische Struktur zu legen (gekennzeichnet durch geschweifte Klammern). Die morphologische Analyse und die Strukturierung sind jeweils Finite State Automaten. Im strukturierten String wird dann durch eine C-Funktion der Hauptakzent auf das erste Element der am tiefsten eingebetteten Komponente zugewiesen. Zum Schluß werden die Aussprache und die Silbenstruktur, wieder durch Automaten realisiert, zugewiesen (dieser Teil wurde komplett von B. Möbius übernommen).

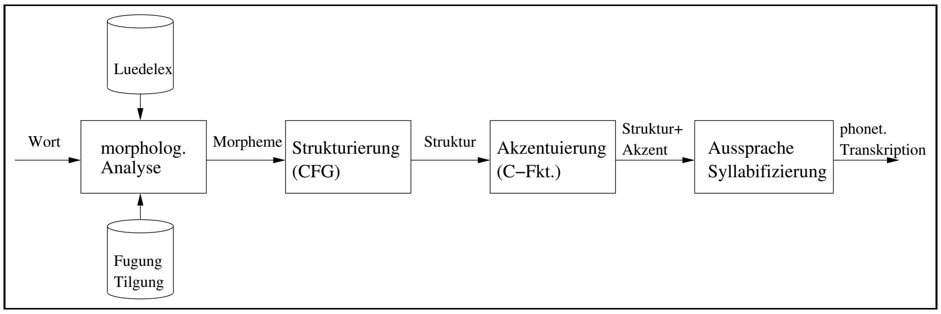

Abb. 1

2 Die morphologische Analyse

2.1 Aufbau

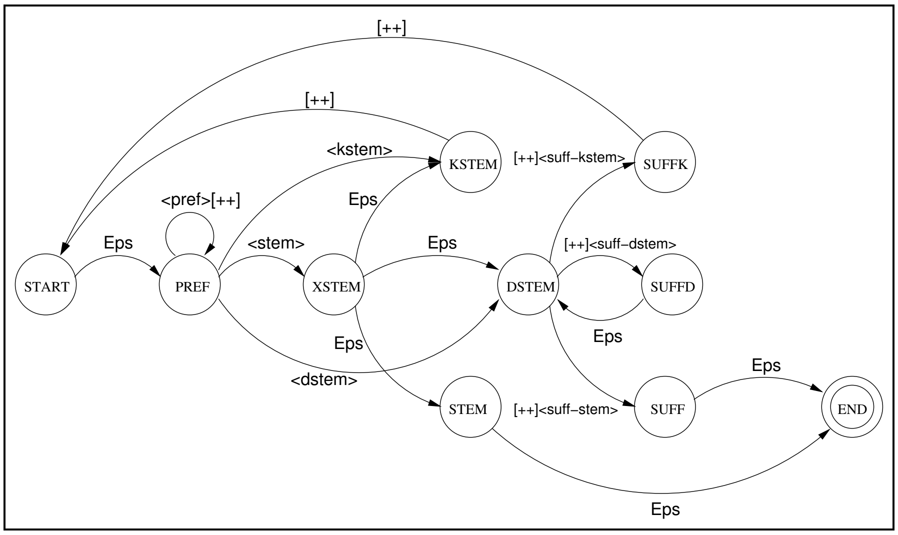

Abb. 2

In Abbildung 2 ist der Zustands-Übergangsgraph des Morphemgrenzenerkenners zu sehen. Zunächst werden beliebig viele (oder keine) Präfixe aneinandergehängt. Darauf folgt genau ein Stamm: entweder ein Kompositions, Derivationsstamm oder ein 'normaler' Stamm. Der Umweg über den Zustand XSTEM ist nötig, um aus einem normalen Stamm einen Derivations- oder Kompositionsstamm zu machen (evtl. durch zusätzliche Fugungs-, Tilgungs- und Umlautungsregeln), da bei weitem noch nicht alle Derivations- und Kompositionsstämme im Lexikon eingetragen sind. Nach einem normalen Stamm ist das Wort beendet. Auf einen Kompositionsstamm muß wieder ein Element mit einem eigenen Stamm folgen. An einen Derivationsstamm dürfen beliebig viele (aber mindestens ein) Suffixe gehängt werden. Für die Suffixe gilt im Prinzip das gleiche wie für die Stämme: auf einen Kompositionsstamm muss eine Morphemfolge mit einem eigenen Wurzelelement folgen, auf einen Derivationsstamm muss noch mindestens ein Suffix folgen und nach einem 'normalen' Suffix darf nichts mehr kommen. Allerdings besteht hier nicht die Möglichkeit durch Tilgung/Fugung einen D- oder K-Stamm zu erzeugen, da davon ausgegangen wird, dass die Lexikoneinträge für Suffixe vollständig sind. Das bedeutet, dass wenn ein Suffix keinen Derivationsstamm besitzt, dann kann es auch keine Derivation eingehen, das gleich gilt für die Komposition.

Das Regelformat für den oben abgebildeten Automaten ist in Listing 1 zu sehen. Hierbei steht in der ersten Spalte immer der Ausgangszustand, in der zweiten Spalte der Folgezustand und in der dritten Spalte das Übergangslabel. Labels der Form [X_lex_] bezeichnen selber wieder Automaten, die hier die verschiedenen Lexika kodieren.

```
START   PREF    [<bos>]
PREF    PREF    [praef_lex_][++]<10.0>

PREF    XSTEM   [stem_lex_][stem][root]
PREF    KSTEM   [kstem_lex_][komp_stem][root]
PREF    DSTEM   [dstem_lex_][deriv_stem][root]

XSTEM   STEM    [<epsilon>]
XSTEM   KSTEM   [<epsilon>]<20.0>
XSTEM   DSTEM   [<epsilon>]<20.0>

KSTEM   PREF    [++][komp]<30.0>
KSTEM   DSTEM   [<epsilon>]

DSTEM   SUFFD   [++][suff_dstem_lex_][deriv_stem]<10.0>
DSTEM   SUFFD   [konversion][verb][pres][egal]<10.0>
DSTEM   SUFFD   [konversion][nomen][sg][uk]<10.0>
DSTEM   KSTEM   [++][suff_kstem_lex_][komp_stem]<10.0>
DSTEM   SUFF    [++][suff_stem_lex_][stem]<10.0>

SUFFD   DSTEM   [<epsilon>]
STEM    END     [<eos>]
SUFF    END     [<eos>]
END
```

Listing 1 (Arclist)

2.2 Lexikon

Das neue IMS-Lex (in XML kodiert) wird per Sylesheets in sogenannte Wordlists und Mappinglists ausgelesen. Zur Wortbildung werden haupsächlich die Stammformen und, wo vorhanden, die Kompositions- und Derivationsstämme verwendet. Diese Stämme werden am Ende der Analyse auf ihre Zitierform abgebildet. In Listing 2 ist ein beispielhafter Eintrag für alle Stämme des Lexems 'Bischof' und dem Mapping auf dessen Zitierform zu sehen. Ist ein Eintrag für einen Derivations- oder Kompositionsstamm vorhanden, so wird der Stamm mit dem Label [hab_deriv] bzw. [hab_kompos] markiert. Dies soll einer geplante Ausfilterung der Analysen dienen, die mit dem normalen Stamm gemacht werden, obwohl auch ein K- bzw. D-Stamm vorhanden ist (noch nicht implementiert).

Bischof[nomen][sg][masc][hab_kompos][hab_deriv]		
Bischofs[nomen][sg][masc]		*Kompositionsstamm*
Bischöf[nomen][sg][masc]		*Derivationsstamm*

Bischof[nomen]	:	Bischof[nomen]
Bischofs[nomen]	:	Bischof[nomen]
Bischöf[nomen]	:	Bischof[nomen]

Listing 2 (Word- und Mappinglist)

Die Derivations- und Kompositionsstämme werden in der Wortbildung mit dem Label [deriv_stem] bzw. [komp_stem] versehen (siehe auch Listing 1). Die normalen Stämme erhalten das Label [stem].

2.3 Umlautung, Fugungs- und Tilgungsregeln

Zur Simulation der noch fehlenden Derivations- und Kompositionsstämme dienen Regeln für Umlautung, Fugung und Tilgung. Die Umlautung ist allgemein definiert für alle Suffixe, die Umlautung auslösen können (markiert durch das Label [an_umlt] (siehe auch Listing 3).

```
a -> ä / __ (u)? ([nonumvowel]|[cons])* ([grammatical])* [stem] [root]
        [++] ([Letters])+ [suff] ([grammatical])* [an_umlt]
o -> ö / __ ([nonumvowel]|[cons])* ([grammatical])* [stem] [root]
        [++] ([Letters])+ [suff] ([grammatical])* [an_umlt]
u -> ü / __ ([nonumvowel]|[cons])* ([grammatical])* [stem] [root]
        [++] ([Letters])+ [suff] ([grammatical])* [an_umlt]
```

Listing 3 (2-Level-Regeln für Umlautung)

Die Fugungs- und Tilgungsregeln werden speziell für jedes Suffix einzeln definiert. Für die Komposition existieren ein paar allgemeine Regeln.

In Listing 4 sind je ein Beispiel für Fugung und Tilgung zu finden: in Regel (1) wird am Ende eines Nomens ein 'e' getilgt, wenn darauf das Suffix 'lich' folgt. In Regel (2) wird am Ende eines Nomens ein 's' gefugt, wenn darauf das Suffix 'mäßig' folgt.

```
(1) e -> [<epsilon>] / __  [nomen] [sg] ([grammatical])* [stem] [root]
                       [++] lich [suff]
(2) [<epsilon>] -> s / __ [nomen] [sg] ([grammatical])* [stem] [root]
                       [++] mäßig [suff]
```

Listing 4 (2-Level-Regeln für Tilgung und Fugung)

Um sicher zu stellen, dass diese Regeln nicht auf Derivations-, Kompositionsstämme oder Suffixe angewendet werden dienen die Labels [stem] (nur Stämme) und [root] (nur freie Stämme).

2.4 Restriktionen

Die Restriktionsregeln dienen zur Einschränkung der Kombinationen der Stämme mit Suffixen und der Suffixe untereinander. Jedes Suffix besitzt eine Menge von Selektionseigenschaften (z.B. für Wortart). In den Restriktionsregeln wird dafür gesorgt, dass nur zulässige Kombinationen akzeptiert werden (z.B. dass ein Suffix, das Nomen selektiert nicht mit einer adjektivischen Basis kombiniert werden kann). Dies wird über 2-Ebenen-Regeln realisiert, da sich der relevante Kontext auf zwei aufeinanderfolgende Morpheme einschränken lässt: die Basis und das Suffix. Dies ist deshalb möglich, da der Kopf der Basis (also genau das dem Suffix vorhergehende Element) die Eigenschaften der gesamten Basis bestimmt.
In Listing 5 ist ein Beipiel für eine Restriktionsregel zu sehen: Alle Vorkommen der Wortart Nomen werden zuerst als falsch markiert und nur in dem Kontext, in dem auf ein Nomen auch ein Suffix folgt, das ein Nomen selektiert, wird diese Markierung wieder gelöscht. Zum Schluss werden über einen Filter alle Analysen mit der Markierung falsch 'aussortiert', d.h. nicht akzeptiert.

```
[<epsilon>] -> [wrong] / [nomen] __ ([grammatical])*
                         [++]([Letters])+ [suff]
[wrong] -> [<epsilon>] / [nomen] __ ([grammatical])* [++]
                         ([Letters])+ [suff] ([grammatical])* [an_nom]
```

Listing 5

Aufgrund der lückenhaften Information des Lexikons erfolgt die Restriktion bisher fast ausschliesslich über die Wortart. Geplant sind aber z.B. auch noch Restriktionen über Herkunft und Komplexität.

Für Präfixe ist dieses Verfahren leider nicht möglich, da sich der relevante Kontext über mehrere Morphemgrenzen erstrecken kann. Daher kann die

Restriktion der Präfixselektion nur über die hierarchischen Strukturregeln erfolgen.
Wünschenswert wäre es gewesen, auch die Suffixselektion über die Strukturierung zu restringieren. Dies war aus Performanzgründen jedoch leider nicht möglich, da die resultierenden Automaten zu gross und daher nicht mehr handhabbar wurden.

3 Strukturierung

Für die Strukturzuweisung wurde das Regelformat kontextfreie Grammatik gewählt. Allerdings ist es in einem Finite-State-Formalismus nicht möglich sowohl Links- als auch Rechtsrekursivität zu haben. Da Wortbildung aber gemischt links- und rechtsrekursiv sein kann, wurde die Rekursion durch einen iterativen Ansatz ersetzt. Wortbildung kann zwar theoretisch endlos rekursiv sein, praktisch aber werden kaum Wörter mit mehr als 5 Ebenen gebildet (z.B. Donaudampfschiffahrtskapitänswitwenkaffeekränzchenkuchenwettessen, siehe Abb. 3).

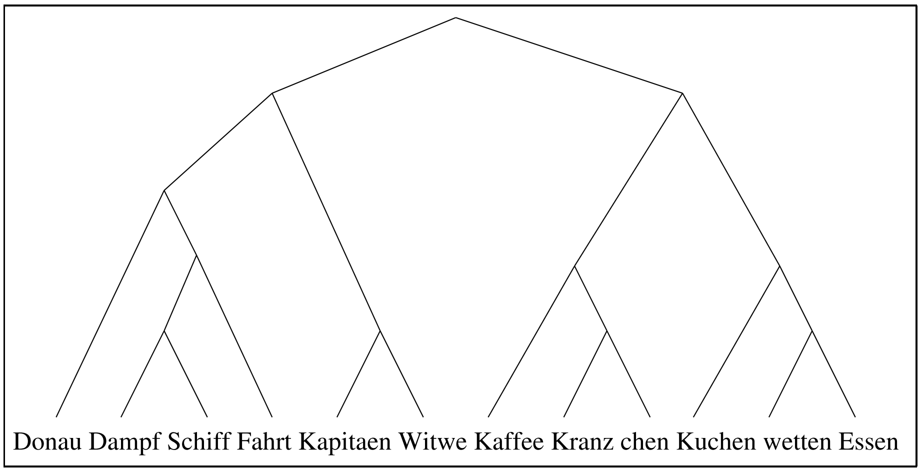

Abb. 3

Ein Nomen z.B. kann gebildet werden, entweder durch Kombination eines Nomenpräfixes mit einem Nomenstamm, eines Stammes mit einem Nomensuffix oder eines beliebigen Stammes mit einem Nomenstamm. Dies entspricht der folgenden rekursiven Regel (NN kann ein beliebig komplexer Nomenstamm sein):

```
NN -> Npref + NN | NN + Nsuff | NN + NN
```

Im iterativen Ansatz würde diese Regel wie folgt aussehen (NN steht dabei für einen simplexen Nomenstamm):

```
NN1 -> Npref NN | NN + Nsuff | NN + NN | NN
NN2 -> Npref NN1 | NN1 + Nsuff | NN1 + NN1 | NN1
NN3 -> Npref NN2 | NN2 + Nsuff | NN2 + NN2 | NN2
usw.
```

In Listing 6 ist eine entsprechende Grammatik abgebildet. Die einzelnen Stufen wurden dabei auf einzelne Automaten verteilt. Je einer pro Stufe und Wortart. Die Labels der Form [hierarchie_stufeX_wortart_] bezeichnen dabei die Automaten der jeweils niedrigeren Stufe.

```
[STEM_NEU] -> [VERB] | [ADJ] | [NN] | [STEM]

[VERB] -> { [VPREF] [VSTEM] } [pref_derivat]
[VERB] -> { [STRING] [vpref] [partikel] [++] [VSTEM] } [trennbar] [pref_derivat]
[VERB] -> { [STEM] [VSUF] } [suff_derivat]

[ADJ] -> { [APREF] [ASTEM] } [pref_derivat]
[ADJ] -> { [STEM] [ASUF] } [suff_derivat]
[ADJ] -> { [ASTEM] [KOMP] [ASTEM] } [kompositum]
[ADJ] -> { [ASTEM] [KOMP] [ASTEM] [KOMP] [ASTEM] } [kompositum]
[ADJ] -> { [NSTEM] [KOMP] [ASTEM] } [kompositum]
[ADJ] -> { [VSTEM] [KOMP] [ASTEM] } [kompositum]

[NN] -> { [NPREF] [NSTEM] } [pref_derivat]
[NN] -> { [STEM] [NSUF] } [suff_derivat]
[NN] -> { [NSTEM] [KOMP] [NSTEM] } [kompositum]
[NN] -> { [ASTEM] [KOMP] [NSTEM] } [kompositum]
[NN] -> { [VSTEM] [KOMP] [NSTEM] } [kompositum]

[STEM] -> ( [VSTEM] | [ASTEM] | [NSTEM])
[VSTEM] -> [hierarchie_stufe3_verb_]
[ASTEM] -> [hierarchie_stufe3_adj_]
[NSTEM] -> [hierarchie_stufe3_nom_]

[VSUF] -> [++] [STRING] [suff] [verb] [time] [stem_types]
[ASUF] -> [++] [STRING] [suff] [adjektiv] [steigerung] [stem_types]
[NSUF] -> [++] [STRING] [suff] [nomen] [number] [gender] [stem_types]
```

```
[VPREF] -> [STRING] [vpref] [++]
[APREF] -> [STRING] [apref] [++]
[NPREF] -> [STRING] [npref] [++]

[STRING] -> [Letters] ([STRING])?

[KOMP] -> [++] [komp]
```

Listing 6

4 Akzentuierung

Die Akzentzuweisung wurde über eine C-Funktion realisiert, da es nicht möglich schien, dies durch einen Finite State Mechanismus zu lösen.

Aus dem strukturierten Morphemstring wird eine Baumstruktur erzeugt, nach dem folgenden Schema:

- für jede öffnende Klammer gehe eine Ebene tiefer
- für jede schliessende Klammer gehe eine Ebene höher
- bei jeder Morphemgrenze ([++]) ohne vorhergehende oder nachfolgende Klammer gehe zum benachbarten Knoten

In dieser Baumstruktur wird dann rekursiv die am tiefsten eingebettete Komponente ermittelt, und der Hauptakzent auf deren erstes Element zugewiesen. Dabei wird nach dem folgenden Schema vorgegangen (siehe auch Abb. 4):

- für jeden Knoten gilt:
  - linker Sohn erhält den Akzentwert+1
  - rechter Sohn erhält den Akzentwert

Begonnen wird am Wurzelknoten mit dem Akzentwert 0. Der Knoten mit dem höchsten Akzentwert erhält den Hauptakzent.

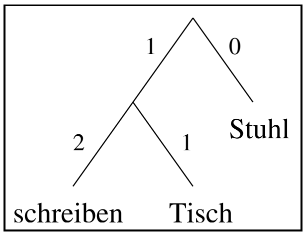

Abb. 4

5 Ausspracheregeln und Syllabifizierung

Dieser Teil wurde komplett von Bernd Möbius übernommen. Die Syllabifizierung besteht wieder aus einer deklarativen Grammatik (siehe Listing 7).

<table>
<tr><td>START</td><td>FIRST</td><td>([<epsilon>]:[++])?[ww]?[Boundary]*[ww]</td></tr>
<tr><td>FIRST</td><td>ONSET</td><td>[<epsilon>]<12.0></td></tr>
<tr><td>ONSET</td><td>CODA</td><td>(([PCons]<100.0>)|([PVowel]<100.0>))+</td></tr>
<tr><td>FIRST</td><td>ONSET</td><td>[PCons]<9.0></td></tr>
<tr><td>FIRST</td><td>ONSET</td><td>((pf<9.0>)|(sf<29.0>))</td></tr>
<tr><td>FIRST</td><td>ONSET</td><td>((ps<12.0>)|(ts<9.0>)|(ks<29.0>))</td></tr>
<tr><td>FIRST</td><td>ONSET</td><td>((pS<29.0>)|(tS<9.0>))</td></tr>
<tr><td>ONSET</td><td>NUCLEUS</td><td>["]?((I<12.0>)|(W<12.0>)|([<]<12.0>))</td></tr>
<tr><td>ONSET</td><td>NUCLEUS</td><td>(([&]<12.0>)|([^]<12.0>))</td></tr>
<tr><td>NUCLEUS</td><td>CODA</td><td>p<10.0>((t<10.0>)|(f<10.0>)|(s<10.0>)|(S<20.0>))</td></tr>
<tr><td>NUCLEUS</td><td>CODA</td><td>t<10.0>((s<10.0>)|(S<10.0>))</td></tr>
<tr><td>NUCLEUS</td><td>CODA</td><td>k<10.0>((t<10.0>)|(s<10.0>)|(S<20.0>))</td></tr>
<tr><td>CODA</td><td>END</td><td>([<epsilon>]:[++])?[ww]?[Boundary]*[ww]</td></tr>
<tr><td>END</td><td></td><td></td></tr>
</table>

Listing 7 (Syllabifizierungsregeln, Auszug)

Die Ausspracheregeln bestehen hauptsächlich aus 2-Ebenen-Regeln. Die Umwandlung erfolgt 2-stufig: zunächst findet eine Überführung in ein Zwischenformat statt, pro Phonem exakt ein spezielles Symbol, danach folgt die Überführung in Sampa-Transkription. Dieser Zwischenschritt ist notwendig um unerwünschte Nebeneffekte von sich gegenseitig beeinflussenden Regeln (bleeding/feeding-rules) zu vermeiden, da sowohl auf dem Eingabe- als auch auf dem Ausgabeband zum Teil das gleiche Alphabet verwendet wird. In Listing 8 findet sich ein Beispiel für eine sog. letter-to-sound-rule.

```
aa -> [aA]
a -> [oo] / ___ x
a -> [oo] / k ["]? ___ p ([<sigma>] & !(er))
a -> [oo] / p ["]? ___ l ([<sigma>] & !([ww]|[++]|[Boundary]))
a -> [oo] / (p|k) ["]? ___ p ["]? (a|r)
a -> [aA] / ___ [kk][vv]
a -> [aA] / ___ ((r? [Grammatical]* ([ww]|[++]|[Boundary]))|h)
```

Listing 8 (letter to sound rules)