

Christian Simon

Finite-State-basierte Morphologie-Tools und ihre Stärken und Schwächen bei der maschinellen Wortbildungsanalyse

1. Einleitung

Was ist ein Morphologie-Tool? Ein Morphologie-Tool ist in der maschinellen Sprachverarbeitung ein Programm, das seine Eingabe anhand einer natürlich-sprachlichen Morphologie verarbeitet. Diese Verarbeitung kann sowohl die *morphologische Analyse* umfassen als auch die *Generierung* von Wortformen. Bei der morphologischen Analyse bekommt das Tool eine natürlichsprachliche Wortform als Eingabe und liefert als Ausgabe eine Analyse, die ihr zugrunde liegendes Lexem, ihre morphosyntaktischen Flexionsmerkmale sowie Angaben ihrer Gebildetheit enthält. Die Generierung dagegen funktioniert genau umgekehrt: Hier soll aus den entsprechenden Angaben eine flektierte, in der natürlichen Sprache gültige Wortform produziert werden. Dabei ist es ein wesentliches Merkmal eines Morphologie-Tools, dass diese morphologische Verarbeitung des Eingabestrings anhand eines Lexikons und von Regeln bzw. anhand eines statistischen Modells in Echtzeit vorgenommen wird. Das triviale Abrufen von fertigen Ergebnissen aus einer Datenbank oder einer anderen Wissensbasis soll gerade nicht stattfinden. Das Lexikon und das Regelwerk sollen ausdrucksstark genug sein, um mit beliebigen Eingaben umgehen zu können.

Zum Einsatz kommen Morphologie-Tools insbesondere als Komponente in komplexeren computerlinguistischen Anwendungen wie der maschinellen Übersetzung oder dem Information-Retrieval. Da sie in solchen Kontexten lediglich eine Analysevorstufe für eine aufwendige Verarbeitung natürlicher Sprache darstellen, müssen sie darüber hinaus sehr schnell sein. In diesem Sinne eignen sie sich auch als Werkzeug, um große Menge von linguistischen Daten um morphologische Informationen anzureichern.

Den Anspruch auf eine hundertprozentige Korrektheit der produzierten Ausgaben oder gar eine vollständige Abdeckung der jeweiligen unterstützten Sprache können sie allerdings nicht erfüllen. Ihrem Können sind Grenzen gesetzt, die sich zum einen aus ihrer grundlegenden Funktionsweise heraus ergeben, aber auch aus der Unendlichkeit natürlicher Sprachen sowie der alltäglichen Praxis.

In diesem Sinne will dieser Aufsatz eine kurze Einführung in die grundlegende Funktionsweise von Morphologie-Tools geben, die auf der so genannten Finite-State-Technologie basieren. Dabei soll exemplarisch anhand des Mor-

phologie-Tools *Morphisto* (Zielinski/Simon 2008) gezeigt werden, wo gerade bei der Wortbildungsanalyse ihre Stärken und Schwächen liegen. Der Aufsatz schließt mit einem Ausblick, in dem auch alternative Ansätze für Morphologie-Tools angesprochen werden.

2. Grundlegende Funktionsweise von Morphologie-Tools

2.1 Der Finite-State-Transducer

Die Finite-State-Technologie hat sich zur etabliertesten Technologie für Morphologie-Tools entwickelt. Im Kern begründet sie sich auf ein Konzept aus der theoretischen Informatik, dem 'Finite-State-Transducer' (FST) bzw. 'endlichen Transduktor'. Den Finite-State-Transducer muss man sich als eine simple, abstrakte gedachte Maschine vorstellen, die eine beliebige Folge von Symbolen, den 'String', abarbeitet und ein Netzwerk von Knoten von einem Startknoten zu einem Ziel- bzw. Endknoten durchläuft. Während der Übergänge von einem Knoten zum nächsten werden Zeichen ausgegeben, die aneinandergereiht den Ausgabestring bzw. das Ergebnis darstellen. Ein Regelwerk bestimmt das Zusammenspiel von gelesenen Eingabezeichen, Ausgabezeichen, Start- und Zielknoten.

Zur Veranschaulichung sei auf die grafische Darstellung eines FST als gerichteter Graph in Abbildung 1 verwiesen: Knoten **S** ist der Startknoten, **E** der Endknoten. In den Kantenbeschriftungen stehen die gelesenen Eingabezeichen links vom Doppelpunkt, die zu produzierenden Ausgabezeichen rechts vom Doppelpunkt.

Formal lässt sich ein FST als ein 6-fach-Tupel $A = \langle \Phi, \Sigma_1, \Sigma_2, \delta, \lambda, S, F \rangle$ wie folgt definieren (basierend auf Klavunde 1998, Definition 3.9; sowie Mohri 1997, Definition S. 271):

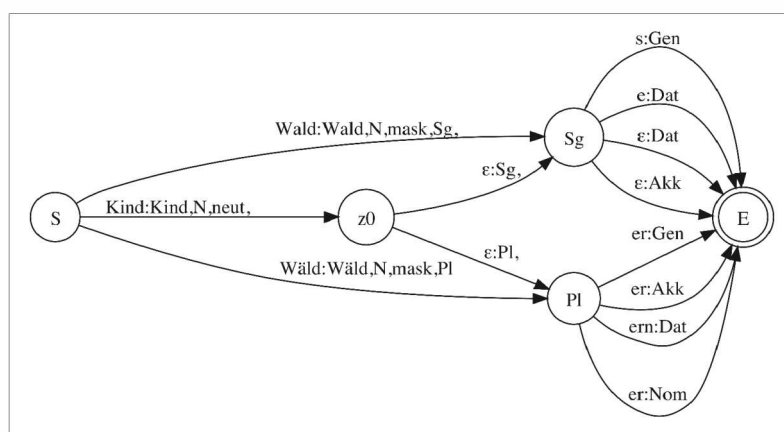


Abb. 1: Finite-State-Transducer für *Wald*, *Kind*

- Φ ist die Menge der Zustände, die der Automat annehmen kann. Für den Automaten in Abbildung 1 ist dies $\Phi = \{S, z0, Sg, Pl, E\}$.
- Σ_1 ist die Menge der möglichen Eingabesymbole. Bei einem FST für deutsche Wortformen entspricht das der Menge der lateinischen Buchstaben plus den deutschen Sondergraphemen \ddot{i} , \ddot{a} , \ddot{o} , β . Dazu kommt das so genannte ‘leere Wort’ ε , das es dem Automaten erlaubt, einen Übergang vorzunehmen, ohne ein Zeichen lesen zu müssen:
- $\Sigma_1 = \{A-Z, a-z, \ddot{A}, \ddot{a}, \ddot{O}, \ddot{o}, \ddot{U}, \ddot{u}, \beta, \varepsilon\}$
- Σ_2 ist die Menge der möglichen Ausgabesymbole, in diesem Fall gilt $\Sigma_2 = \Sigma_1$.
- Die Funktion $\delta: \Phi \times \Sigma_1 \rightarrow \Phi$ definiert die Übergänge von einem Zustand in den nächsten in Abhängigkeit vom gelesenen Symbol.
- Die Funktion $\lambda: \Phi \times \Sigma_1 \rightarrow \Sigma_2^*$ definiert die Ausgabe in Abhängigkeit vom derzeitigen Zustand und der gelesenen Eingabe.
- Es gibt einen Startzustand $S \in \Phi$, in unserem Fall S , von dem aus der FST die Verarbeitung des Eingabestrings beginnt.
- $F \subseteq \Phi$ bezeichnet die Menge der möglichen Endzustände, an denen die Verarbeitung beendet werden kann; in diesem Fall $F = \{E\}$.

2.2 FST-basierte Wortformenanalyse

Wie geht der FST vor, wenn eine deutsche Wortform analysiert werden soll? Der in Abbildung 1 dargestellte FST ist dafür konzipiert, die Flexionsparadigmata von *Wald* und *Kind* analysieren zu können.¹ Nehmen wir an, dass der Automat auf die Wortform *Wälder* angewandt wird: Zunächst befindet sich der Automat in seinem Startzustand, also S . Für die Eingabezeichenkette *Wäld* existiert im Zustand S ein Übergang, nämlich in den Zustand Pl . Bei diesem Übergang wird der Ausgabestring *N,mask,Pl* ausgegeben. Im Zustand Pl gibt es für den verbleibenden Eingabestring drei mögliche Übergänge: Verfolgen wir den ersten, gelangen wir in den Endzustand E und an die Ausgabe wird *Gen* angehängt. Da der Automat für diesen Eingabestring einen Endzustand erreicht hat, wird die Eingabe als gültig akzeptiert und die Analyse

¹ Der Vollständigkeit halber sollte hier erwähnt werden, dass dieser Transducer lediglich dem Zweck dient, dem Leser die grundlegende Funktionsweise eines FST näherzubringen. Er ist darüber hinaus *nichtdeterministisch*, d.h., dass es für jedes Eingabesymbol mehr als einen Zielzustand gibt. In der Praxis würde man einen solchen Transducer zunächst in einen *deterministischen* umwandeln und danach hinsichtlich der Anzahl seiner Zustände *minimieren*.

N,mask,Pl,Gen ausgegeben. Da es aber im Zustand *PI* noch zwei weitere Übergänge gibt, erlaubt das so genannte ‘Backtracking’, zur letzten Verzweigungsmöglichkeit zurückzukehren und die übrigen Übergänge zu verfolgen. Damit ergeben sich zusätzlich die Ausgaben *N,mask,Pl,Akk* sowie *N,mask,Pl,Nom*.

Was passiert nun bei einer falschen Eingabe wie **Wäld*? Der Automat würde bis in den Zustand *PI* kommen und dann abbrechen, weil der Eingabestring bereits abgearbeitet worden ist, obwohl noch kein Endzustand erreicht wurde. Für die falsche Eingabe **Wälde* gäbe es im Zustand *PI* keinen Übergang für die Eingabe *e*, sodass der Automat ebenfalls stehen bliebe und die Wortform als nicht gültig zurückgewiesen würde.

Bei der Eingabe *Kind* gelangt der Automat zunächst in den Zustand *z0* und produziert die Ausgabe *N,neut*. Von dort erlaubt es das leere Wort ϵ , sowohl in den Zustand *Sg* als auch in den Zustand *PI* zu springen, ohne dass eine Eingabe gelesen werden muss. Der erste Sprung führt in den Zustand *Sg* und hängt *Sg* an die Ausgabe. Danach erlauben wiederum das leere Wort und das Backtracking, dass die Analysen *N,neut,Sg,Dat* sowie *N,neut,Sg,Akk* ausgegeben werden. Ein erneutes Backtracking führt dazu, dass der Automat in den Zustand *PI* wechselt. Da aber dort schon die Eingabe zu Ende gelesen worden ist, passt kein weiterer Übergang, sodass keine weiteren Analysen mehr ausgegeben werden.

2.3 FST-basierte Wortformengenerierung

Eine weitere Eigenschaft, die FSTs für die Computermorphologie so attraktiv gemacht haben, ist die Möglichkeit, die Ein- mit der Ausgabeseite zu vertauschen und somit den Automaten für die Wortformengenerierung zu nutzen: Nehmen wir an, die Ein- und die Ausgabeseite im FST von Abbildung 1 wären miteinander vertauscht worden. De facto bedeutet dies, dass der String, der rechts vor dem Doppelpunkt stand, nun links davon steht und dass der String links nun rechts steht. Wenden wir den FST mit vertauschter Ein- und Ausgabeseite auf den Eingabestring *Wald,N,mask,Sg,Dat* an: Für die Teileingabe *Wald,N,mask,Sg*, springt der Automat zunächst in den Zustand *Sg* und produziert die Ausgabe *Wald*. Danach gibt es zwei mögliche Übergänge für die Eingabe *Dat*: Beide führen in den Endzustand, der eine jedoch produziert die Ausgabe *Walde*, der andere hängt lediglich das leere Wort an, sodass die Ausgabe *Wald* entsteht.

2.4 Erstellung von Finite-State-Transducern

In der Praxis kommen für die Computermorphologie FSTs zum Einsatz, die Millionen von Zuständen und Übergängen haben, sodass eine manuelle Erstellung unmöglich ist. Stattdessen hat man das Problem wie in der Informatik üblich auf eine höhere Abstraktionsebene verlagert, indem man formale Sprachen definiert hat, mit denen sich die Erstellung von FSTs automatisieren lassen. Sie bedienen sich formal bewiesener Eigenschaften von FSTs, wie z.B. ihrer Abgeschlossenheit über der Vereinigung.

Mit der Eigenschaft der Vereinigung lassen sich zwei beliebige FSTs zu einem einzigen FST vereinigen, der das Erkennungs- und Ausgabeverhalten von den beiden Ursprungs-FSTs hat. Folgendes Code-Beispiel ist in der Beschreibungssprache des *Stuttgart Finite State Toolkits (SFST)* (Schmid 2006) geschrieben:

```
$A$ = {Kind,N,neut,Pl,Nom}:{Kinder}
$B$ = {Kind,N,neut,Pl,Gen}:{Kinder}
$C$ = $A$ | $B$
$C$
```

Mit **\$A\$** wird ein FST definiert, der *Kinder* erkennt und dafür die Ausgabe *Kind,N,neut,Pl,Nom* produziert. **\$B\$** erkennt ebenfalls Kinder, produziert aber die Ausgabe *Kind,N,neut,Pl,Gen*. In der letzten Zeile werden diese beiden FSTs zu einem FST **\$C\$** vereinigt (Operator |), der für die Eingabe *Kinder* die zwei bekannten Ausgaben produziert.

Dieses Vorgehen ist allerdings trivial und zudem unelegant, weil es einer Wortform eine fertige Analyse zuordnet. Erheblich besser ist es dagegen, generischer zu arbeiten und zwar mit der von Kimmo Koskeniemi entwickelten *Two-Level-Morphology* (Koskeniemi 1983). Zunächst definieren wir einen FST, der Grundformen einer Morphologie abdeckt, indem die Basisstämme über die Eigenschaft der Vereinigung miteinander verknüpft werden:

```
$LEXIKON$ = Kind | Wald
```

Die Reihe ließe sich mit beliebig vielen weiteren Basisstämmen fortsetzen, der Einfachheit halber bleiben wir bei *Kind* und *Wald*.

In einem zweiten Schritt konkatenieren wir diesen FST mit einem FST, der die morphosyntaktische Angabe *N,neut,Sg,Gen* auf den String *\$s* abbildet:

```
$A$ = $LEXIKON$ {\,N,mask,Sg,Gen}:{$s}
```

Für die Eingabe *Wald,N,neut,Sg,Gen* produziert dieser FST die Ausgabe *Wald\$s*. Das *\$* ist hier ein beliebiges Sonderzeichen, das vorzugsweise nicht in der natürlichen Sprache, deren Morphologie modelliert werden soll, vorkommt. Als Nächstes modellieren wir einen FST, der jedes Vorkommen von *\$* durch *e* ersetzt:

ALPHABET = [A-Za-z] \$:e
\$GENITIV1\$ = .*

Analog dazu brauchen wir noch einen FST, der jedes Vorkommen von *\$* durch den leeren String (hier durch *<>* ausgedrückt) ersetzt:

ALPHABET = [A-Za-z] \$:<>
\$GENITIV2\$ = .*

Abschließend werden diese beiden FSTs zu einem großen FST vereinigt.

\$GENITIVREGELN\$ = \$GENITIV1\$ | \$GENITIV2\$

Schließlich nutzen wir die Eigenschaft von FSTs aus, dass sie über Komposition abgeschlossen sind. Abgeschlossenheit über Komposition bedeutet, dass man zwei beliebige FSTs, die kaskadiert hintereinander arbeiten, d.h., dass die Ausgabe des ersten zur Eingabe für den zweiten wird, in einen einzigen äquivalenten FST verwandeln kann. Und das geschieht hier:

\$A\$ || \$GENITIVREGELN\$

Am Ende steht ein FST, der sich genauso verhält, als würde man die Eingabe zunächst vom FST *\$A\$* verarbeiten lassen und dann dessen Ausgabe später auf den FST in *\$GENITIVREGELN\$* anwenden. Damit verwandelt sich eine Eingabe *Wald,N,neut,Sg,Gen* zunächst in *Wald+s* und danach kann sie entweder in *Waldes* umgewandelt werden (FST *\$GENITIV1\$*) oder aber in *Walds* (FST *\$GENITIV2\$*). Und da FSTs auch ihre Seiten vertauschen können, lässt sich der Prozess so umkehren, dass aus *Walds* die bekannte Analyse *Wald,N,neut,Gen,Sg* produziert wird.

De facto wird das Problem auf zwei Ebenen verlagert. Auf der ersten wird aus der Eingabe eine morphologisch-allophonische Zwischenrepräsentation erstellt (*Wald+s*), auf der zweiten wird dann daraus das Endergebnis (*Walds*, *Waldes*). Und dabei ließe sich die Menge der Basisstämme, auf die man diese Regel anwenden kann, noch mühelos erweitern. Diese Möglichkeit, einerseits phonologische Gesetzmäßigkeiten zu modellieren und gleichzeitig das Lexikon offen zu lassen, hat dem FST-basierten Ansatz für Morphologie-Systeme zum Durchbruch verholfen.

3. Morphisto – ein freies Morphologie-Tool für das Deutsche

3.1 Vorstellung von Morphisto

Morphisto ist ein auf der FST-Technologie basierendes Morphologie-Tool für das Deutsche.² Es stellt eine Weiterentwicklung der Computermorphologie *SMOR* der Universität Stuttgart (Schmid/Fitschen/Heid 2004)³ dar, die am IDS Mannheim um ein freies Lexikon erweitert wurde (Zielinski/Simon 2008). *SMOR* steht unter der Open-Source-Softwarelizenz *GPLv2*,⁴ das Lexikon, das *Morphisto* zu *SMOR* beisteuert, unter der *CC BY-SA 2.0*-Lizenz für nichtkommerzielle Zwecke.⁵

Für deutsche Wortformen produziert *Morphisto* eine Flexionsanalyse sowie eine „flache“ Wortbildungsanalyse; flach in dem Sinne, dass eine Wortform in ihre lexikalischen Komponenten zerlegt wird, aber ohne dass dabei Aussagen über die Hierarchie dieser Zerlegung getroffen werden. Folgendes Beispiel zeigt *Morphistos* Analyse für die ambige Wortform *Staubecken*, die sowohl als *Stau-Becken* bzw. *Staub-Ecken* bzw. *Staub-Eck* interpretiert werden kann:

```

analyze> Staubecken
Staub<NN>Ecke<+NN><Fem><Akk><PI>
Staub<NN>Ecke<+NN><Fem><Gen><PI>
Staub<NN>Ecke<+NN><Fem><Nom><PI>
Staub<NN>Ecke<+NN><Fem><Dat><PI>
Staub<NN>Eck<+NN><Neut><Dat><PI>
Stau<NN>Becken<+NN><Neut><Akk><PI>
Stau<NN>Becken<+NN><Neut><Akk><Sg>
Stau<NN>Becken<+NN><Neut><Gen><PI>
Stau<NN>Becken<+NN><Neut><Nom><PI>
Stau<NN>Becken<+NN><Neut><Nom><Sg>
Stau<NN>Becken<+NN><Neut><Dat><PI>

```

² Morphisto-Projektseite: <http://code.google.com/p/morphisto/> (Stand: 09/2011). Zum Einsatz von *Morphisto* im Projekt *ellexiko* vgl. Kapitel 3 im Beitrag „Chancen und Probleme bei der automatischen Ermittlung von Wortbildungsprodukten für *ellexiko* und bei ihrer Präsentation“ von Sabina Ulsamer in diesen Band.

³ *SMOR*-Projektseite: <http://www.ims.uni-stuttgart.de/projekte/gramotron/SOFTWARE/SFST.html> (Stand: 09/2011).

⁴ <http://www.gnu.org/licenses/gpl-2.0.html> (Stand: 09/2011).

⁵ <http://creativecommons.org/licenses/by-sa/2.0/de/> (Stand: 09/2011).

Stau<NN>*Becken*<+NN><Neut><Dat><Sg>
stauben<V>*Ecke*<+NN><Fem><Akk><PI>
stauben<V>*Ecke*<+NN><Fem><Gen><PI>
stauben<V>*Ecke*<+NN><Fem><Nom><PI>
stauben<V>*Ecke*<+NN><Fem><Dat><PI>
stauben<V>*Eck*<+NN><Neut><Dat><PI>
stauen<V>*Becken*<+NN><Neut><Akk><PI>
stauen<V>*Becken*<+NN><Neut><Akk><Sg>
stauen<V>*Becken*<+NN><Neut><Gen><PI>
stauen<V>*Becken*<+NN><Neut><Nom><PI>
stauen<V>*Becken*<+NN><Neut><Nom><Sg>
stauen<V>*Becken*<+NN><Neut><Dat><PI>
stauen<V>*Becken*<+NN><Neut><Dat><Sg>

Alle Angaben in spitzen Klammern sind als morphologische Metainformationen zu lesen. Zuerst werden die lexikalischen Komponenten mit ihren jeweiligen Wortarten ausgegeben (<V> für Verb, <NN> für Nomen, <+NN> für das letzte Glied). Danach stehen die morphosyntaktischen Angaben.

Wie in Kapitel 1.2 beschrieben, kann Morphisto auch Wortformen generieren:

> *Staub*<NN>*Eck*<+NN><Neut><Dat><PI>
Staubecken

Morphisto versucht beliebige Wortformen des Deutschen zu analysieren, auch fiktive wie *hinwissen* oder *Donaudampfschiffahrtskapitänsmütze*. Das setzt natürlich voraus, dass im FST die jeweiligen Übergänge für die entsprechenden Lexeme, Affixe und Wortbildungsregeln vorhanden sind. Hergeleitet werden diese Übergänge durch ein entsprechendes Lexikon, das separat gepflegt wird und bei der Erstellung des Morphisto-FSTs integriert werden muss.

3.2 Das Lexikon

Der Erfolg eines regelbasierten Morphologie-Tools steht und fällt mit dem Lexikon: Ein zu kleines Lexikon führt zu einer geringen Trefferquote, ein zu großes kann für einen unkontrollierten Anstieg der Größe des FSTs sorgen: Die technischen Probleme, die sich daraus ergeben, sind eine längere Erstellungszeit des FSTs sowie eine schlechtere Performanz beim Einsatz des FSTs. Die linguistische Gefahr, die sich daraus ergibt, ist die *Übergenerierung*.

Bei der Entwicklung von Morphisto wurde daher Redundanzfreiheit als Ziel verfolgt: Es sollten möglichst nur Simplizia ins Lexikon aufgenommen werden, aus denen die Bildung komplexerer Wortformen hergeleitet werden kann. In diesem Sinne wurde beispielsweise zugunsten der Simplizia *Bahn* und *Hof* auf das Lexem *Bahnhof* als Lexikoneintrag verzichtet:

```
<Base_Stems>Bahn<NN><base><nativ><NFem_0_en>
```

```
<Base_Stems>Hof<NN><base><nativ><NMasc_es_$e>
```

Der erste Eintrag ist so zu interpretieren, dass der erkannte Stamm *Bahn* in der Ausgabe wiederum auf *Bahn* abgebildet wird. Dabei handelt es sich um einen Basisstamm, der wie ein nativer, deutscher Stamm behandelt wird. Das dazugehörige Nomen ist feminin, flektiert im Singular ohne Suffix (0 für ‘Null-Morphem’) und im Plural mit dem Suffix *-en*. Bei *Hof* dagegen handelt es sich um ein Maskulinum, dessen Genitiv auf *-es* gebildet wird (wobei der *-e*-Einschub optional ist), während für das Plural-Paradigma ein *-e* angehängt wird und Umlautung (§) stattfindet. Bei der Codierung von morphologischen Zusammenhängen für ein Morphologie-Tool muss man sich von den gedanklichen Vorstellungen der traditionellen Linguistik lösen und sich vor Augen führen, dass die Flexionsparadigmata nicht für ein klassisches Wörterbuch codiert werden, sondern für einen FST. Beim FST in Abbildung 1 brauchte es zwei separate Übergänge für den Stamm *Wald* und den Stamm *Wäld*. In diesem Sinne würde ein Lexem wie *Cello* wie folgt codiert werden:

```
<Base_Stems>Cello<NN><base><nativ><NNeut/Sg_s>
```

```
<Base_Stems>Cello:Celli<NN><base><nativ><NNeut/Pl>
```

Der erste Eintrag beschreibt den Stamm *Cello*, der auf *Cello* abgebildet wird, und zwar als reines Singular-Paradigma, das seinen Genitiv auf *-s* ohne *e*-Einschub bildet. Der zweite Eintrag lässt den Plural-Stamm *Celli* auf die Grundform *Cello* abbilden, wobei keine Suffigierung in diesem Paradigma stattfinden darf.

Hinsichtlich seiner Abdeckung sollte das redundanzarme Lexikon von Morphisto den Morphisto-FST in die Lage versetzen, die 30 000 frequentesten Lemmata des Deutschen, wie sie in der DeReWo-Lemmaliste veröffentlicht wurden (IDS Mannheim 2007), analysieren zu können. Das derzeitige Morphisto-Lexikon umfasst 17 759 so genannter Basisstämme, die im Wesentlichen Lexemen entsprechen, von denen 9 718 Substantiv- und Eigennamenstämme sind, 4 655 Verbstämme und 3 421 Adjektivstämme. Dazu kommen noch 432 Präfixe und 231 sonstige Einträge, sodass das finale Lexikon in Summe 18 653 Einträge hat.

3.3 Morphistos Probleme bei der Wortformenanalyse

3.3.1 Das Problem der unzulänglichen Lexikonabdeckung

Unzulänglichkeiten in der Lexikonabdeckung führen dazu, dass Wortformen insgesamt nicht analysiert werden können, selbst wenn die übrigen Morphe, aus denen sich die Wortform konstituiert, bekannt sind. Das erklärt sich damit, dass der FST in seinem Übergangnetzwerk in einen Zustand gerät, von dem aus keine weiteren Übergänge für dieses Morph existieren. Da es im Deutschen sehr viel motivierte Wortbildung gibt, kommt man mit dem Lexikon von Morphisto recht weit. So kann Morphisto für die 100 000 Einträge umfassende korpusbasierte Wortformenliste DeReWo (IDS Mannheim 2009) lediglich zu 15 114 Einträgen überhaupt keine Analyse ermitteln. Das Hauptproblem stellen dabei Simplizia dar, die aus einer Fremdsprache entlehnt wurden (*Fakir*), darunter vor allem Anglizismen (*Entry*), sowie Eigennamen (*Hartz*) und eher niedrigfrequente Simplizia (*filzen*).

Es bieten sich nicht sehr viele Lösungsmöglichkeiten für dieses Problem: Das eine ist die manuelle, von Menschen zu leistende Lexikonpflege, die bei Bekanntwerden nichtanalysierbarer Wortformen nötig wird und das Lexikon entsprechend anpasst. Diese Methode wird momentan beim Morphisto-Projekt gepflegt: Auf der Projekt-Homepage gibt es einen so genannten ‘Bug-’ oder ‘Issue-Tracker’, eine Software, die es jedermann erlaubt, Probleme, die mit Morphisto auftauchen, zu protokollieren. Es wird gefragt, mit welcher Version von Morphisto das Problem auftritt, bei welcher Eingabe und was die zu erwartende Ausgabe gewesen wäre. Die Projektverantwortlichen haben dann die Möglichkeit, auf das Problem zu reagieren, es auszudiskutieren, zu beheben oder für nichtig oder gelöst zu erklären. Jeder Schritt, und sei es nur, dass jemand für die Lösung des Problems verantwortlich erklärt worden ist, kann dadurch protokolliert werden.

Eine alternative Lösungsstrategie könnte in dem Ansatz von Lindén/Tuovila (2009) bestehen: Das Lexikon wird automatisch durch das so genannte ‘Morphological Guessing’ um weitere Einträge angereichert. Dabei werden für Lexeme die fehlenden Angaben (insbesondere die Flexionsklasse) erraten. Die aus den erratenen Angaben generierten Wortformen werden dann mit einem Korpus abgeglichen.

3.3.2 Übergenerierungen

Einer der offensichtlichsten Mängel bei regelbasierten Morphologie-Tools sind die bereits angesprochenen Übergenerierungen. Übergenerierungen zeich-

nen sich dadurch aus, dass das gewünschte Ergebnis (bzw. bei Ambiguitäten die gewünschten Ergebnisse) zwar in der Ergebnismenge enthalten ist, doch nicht alleine: Dabei steht eine beliebig große Anzahl von Analysen, die entweder theoretisch möglich sind oder aber gänzlich falsch. Folgendes Beispiel demonstriert dieses Phänomen für die Wortform *lebenslänglich*, wobei aus Gründen der Platzökonomie auf die morphosyntaktischen Angaben verzichtet wurde und Duplikate entfernt wurden:

> *lebenslänglich*
Leben<NN>*Länge*<NN>*lich*<SUFF><+ADJ>
Leben<NN>*länglich*<+ADJ>
Leben<NN>*lang*<ADJ>*lich*<SUFF><+ADJ>
Leben<NN>*langen*<V>*lich*<SUFF><+ADJ>
leben<V><NN><SUFF>*Länge*<NN>*lich*<SUFF><+ADJ>
leben<V><NN><SUFF>*länglich*<+ADJ>
leben<V><NN><SUFF>*lang*<ADJ>*lich*<SUFF><+ADJ>
leben<V><NN><SUFF>*langen*<V>*lich*<SUFF><+ADJ>

Berücksichtigt man die Semantik von *lebenslänglich*, so erscheint die Analyse *Leben+lang+lich* am plausibelsten. Eine Analyse wie *leben+lang+lich* wäre vielleicht noch vertretbar, während man wohl als Muttersprachler die Analyse *leben+langen+lich* als inkorrekt entschieden zurückweisen würde.

Ein Spezialfall der Übergenerierungen sind Interferenzen, die durch einsilbige Lexeme entstehen, deren Basisstämme in vielen anderen Stämmen sehr frequent vorkommen. Dazu gehören Lexeme wie *Ei*, *Ion* oder *Uni*. Da es sich dabei um Simplicia handelt, hat ihre Existenz im Lexikon seine Berechtigung. Im folgenden Beispiel ist die Analyse von *Union* als *Un-Ion* theoretisch denkbar, aber doch sehr unwahrscheinlich:

Union<+NN>
Un<OTHER>*Ion*<+NN>

Es liegt im Ermessen des Lexikographen, wie mit Einträgen, die Übergenerierungen verursachen, umzugehen ist: Es gibt die Strategie, solche Stämme gar nicht erst ins Lexikon aufzunehmen, nicht zuletzt weil sie bisweilen eher niedrigfrequent sind. Ein weiterer Trick ist ihre Markierung als so genannte ‘Initialstämme’, die nur am Wortanfang als Wortbildungseinheit genutzt werden dürfen. Damit könnte man im folgenden Beispiel die unschöne Analyse *Buch+Ei* ausschließen:

> *Büchereiausleihe*

buchen<V>*er*<NN><SUFF>*ei*<NN><SUFF>*Ausleihe*<+NN>

buchen<V>*er*<NN><SUFF>*Ei*<NN>*Ausleihe*<+NN>

Bücherei<NN>*Ausleihe*<+NN>

Buch<NN>*Ei*<NN>*Ausleihe*<+NN>

Das würde aber zu keinem korrekten Ergebnis bei der Analyse von *Weichei* führen. Eine weitere Strategie, die allerdings die Lexikonerstellung erheblich aufwendiger macht, ist, nur solche Wortstämme aufeinanderfolgen zu lassen, die ein gemeinsames bestimmtes Merkmal aufweisen. Ansatzweise wird das in SMOR bzw. Morphisto bei Basisstämmen und Präfixen gemacht, die beide als *klassisch* markiert worden sind. Sie sollen verhindern, dass ein als *klassisch* markiertes Präfix wie *a-* mit einem nichtklassischen Basisstamm zusammengeführt wird. Will man dieses Konzept allerdings im großen Maßstab praktizieren, muss man zunächst eine solche Feature-Menge aufstellen und sicherstellen, dass neben den falschen, unterbundenen Analysen nicht auch fälschlicherweise korrekte, gewünschte Analysen unterschlagen werden.

Praktikabler ist dagegen die nachträgliche Filterung der Ergebnismenge. Dabei können verschiedene Strategien verfolgt werden. Die einfachste Heuristik ist die Filterung nach Zerlegungen: Es werden bevorzugt nur die Ergebnisse ausgegeben, die die wenigsten Zerlegungen vorweisen. In diesem Sinne würden in den obigen Beispielen nur die Analyse *Leben+länglich*, *leben+länglich*, *Union* und *Bücherei+Ausleihe* ausgegeben. In bestimmten Fällen kann das aber immer noch nicht helfen. In Zielinski/Simon/Wittl (2009) wird ein Verfahren vorgestellt, mit dem die Ergebnisse mithilfe eines statistischen Modells nachgefiltert wurden. Ebenfalls mit statistischen Verfahren, aber nicht erst nach der Ausgabe der Analyseergebnisse arbeitet man mit den so genannten 'gewichteten (weighted) Finite-State-Transducern (wFST)'. Sie enthalten bereits in ihren Übergängen von einem Zustand in den nächsten Wahrscheinlichkeiten. Am Ende hat dadurch jede Analyse eine Gesamtwahrscheinlichkeit, sodass lediglich jene Analysen mit der größten Wahrscheinlichkeit ausgegeben werden müssen. Ein Morphologie-Tool fürs Deutsche, das sich dieser Methodik bedient, ist TAGH (Geyken/Hanneforth 2006).

3.3.3 Das Problem der fehlenden Hierarchien

Finite-State-Transducer sind formal als endliche Automaten einzustufen und sind damit darauf beschränkt, nur Strings erkennen zu können, die die formalen Eigenschaften einer regulären Sprache haben (vgl. Klabunde 1998, S. 142).

Für flektierte Simplizia ist das ausreichend, für komplexe Wortbildungsmuster aber, wie sie besonders im Deutschen vorkommen, nicht. Allein die Bildung des deutschen Partizips II, das man morphologisch als Zirkumfix (*ge*-Verbstamm-*t* bzw. *ge*-Verbstamm-*en*) auffassen kann, ist von einem endlichen Automaten wie einem FST nicht auf eine elegante Art zu handhaben. Da es keinen Speicher gibt, kann kein Rückschluss auf das bereits erkannte erste Teil *ge*- geführt werden. Abhilfe schafft da der Trick, die Vollform des Partizips II eines Verbs als separaten Wortstamm ins Lexikon aufzunehmen.

Verwandt ist dieses Problem mit der Auflösung der sich öffnenden und schließenden Klammern in einem mathematischen Ausdruck, wie er beispielsweise in Programmiersprachen vorkommen kann. Ihre jeweilige Anzahl muss identisch sein. Abhilfe verschafft man sich dort damit, dass man das Problem auf eine komplexere Ebene verlagert, indem man diese Ausdrücke durch eine kontextfreie Grammatik beschreibt und die Erkennung durch einen so genannten ‘Parser’ vornehmen lässt, der anhand der Grammatik eine hierarchische bzw. baumartige Analyse liefern kann. Für den Einsatz als Morphologie-Tool im Sinne von Morphisto wären kontextfreie Grammatiken und Parser aus mehreren Gründen unpraktikabel: Eine entsprechend mächtige kontextfreie Grammatik, die dasselbe leistet wie ein FST mit Millionen von Zuständen und Übergängen, müsste immens groß sein und von Hand erstellt werden. Dazu kommt, dass Parser in der Regel ein erheblich ineffizienteres Laufzeitverhalten haben. Das Reizvolle an FSTs ist dagegen die in Kapitel 1.3 erwähnte Möglichkeit, den FST durch eine abstrakte Beschreibungssprache automatisch erzeugen zu lassen unter Ausnutzung der dort ansatzweise vorgestellten Two-Level-Morphology.

Dessen ungeachtet wäre ein Ansatz denkbar, bei dem die flache Analyse, die FST-basierte Tools produzieren, als Vorstufe zu einer hierarchischen Analyse verwendet wird. Dann könnte eine kontextfreie Grammatik für die Wortbildung so aussehen:

ComplexNoun = <NN> <NN> | <NN> <NN+>

ComplexNoun = <V> <NN> | <V> <NN+>

ComplexNoun = ComplexNoun <NN> | ComplexNoun <NN+>

Für die Wortform *Bahnhofshalle* liefert Morphisto die folgenden drei Analysen ohne Flexionsangaben:

analyze> *Bahnhofshalle*

bahnen<V>*Hof*<NN>*Halle*<+NN>

Bahn<NN><*Hof*<NN><*Halle*<+NN>

Bahn<NN><*Hof*<NN><*Hall*<+NN>

Mithilfe dieser Grammatik ließen sich für alle drei Zerlegungen diese zwei hierarchischen Darstellungen ermitteln:

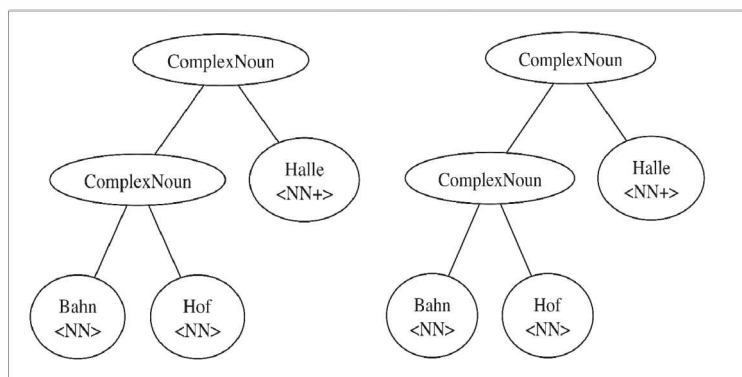


Abb. 2: Hierarchische Analysen auf Morphisto-Ausgabe für *Bahnhofshalle*

Ein Ansatz, der mithilfe probabilistischer kontextfreier Grammatiken in diese Richtung geht, wird in Schmid (2005) beschrieben. Ein eher einfacheres Verfahren, das durch Hypothesenbildung und Verifikation anhand einer Lemmaliste funktioniert, wurde am IDS Mannheim für eine morphologische Baumdatenbank für das *ellexiko*-Projekt entwickelt.⁶ In Zielinski/Simon/Wittl (2009, S. 71) wird es kurz umrissen. Ein statistisches Verfahren, das auf einem Ansatz mit maschinellem Lernen beruht, wird in Marek (2006) vorgestellt.

4. Schlussbetrachtung

Shuly Wintner fasst die Vorteile von FST-Morphologie-Systemen dahingehend zusammen, dass sie eine linguistisch motivierte Beschreibung (‘true representation’) von phonologischen und morphologischen Gesetzmäßigkeiten bieten, somit *modular* sind, dass sie unter den verschiedenen Operationen abgeschlossen und so miteinander kombinierbar sind, hinsichtlich ihrer Größe von den Toolkits *kompakt* gemacht werden können, *effizient* sind sowie *umkehrbar*, d.h., dass sie sowohl analysieren als auch generieren können (Wintner 2007, S. 458). Zur Effizienz von Morphisto lässt sich hinzufügen, dass die in Kapitel 2.3.1 vorgenommene Prozessierung der DeReWo-Wortformenliste

⁶ Zur Gewinnung von Angaben zu Wortbildungsprodukten in *ellexiko* auf der Grundlage dieser Datenbank vgl. den Beitrag „Chancen und Probleme bei der automatischen Ermittlung von Wortbildungsprodukten für *ellexiko* und bei ihrer Präsentation“ von Sabina Ulsamer in diesem Band.

auf einem 2009 handelsüblichen Laptop ca. 7 Sekunden dauerte. In Zielinski/Simon/Wittl (2009) wird dargestellt, wie diese Effizienz genutzt wird. Im Rahmen des TextGrid-Projektes ist Morphisto die Kernkomponente eines Webservice, der die morphologische Analyse und Lemmatisierung von ganzen Texten über das Internet erlaubt.

Trotz aller Vorzüge sind die FST-basierten Ansätze nicht unumstritten und auch nicht als *Ultima Ratio* anzusehen. Wie die Probleme der mangelnden Hierarchien und der Übergenerierungen beweisen, sind die Ausgaben dieser Morphologie-Tools in der Praxis erst nach einer Nachbearbeitung wirklich brauchbar. Und hinsichtlich der Effizienz haben Wintners Experimente sogar ergeben, dass ein Vollformenlexikon mit flektierten und morphosyntaktisch bestimmten hebräischen Wortformen, das in einer Datenbank oder in einer Hash-Tabelle innerhalb eines Java-Programms abrufbereit zur Verfügung steht, ab mehr als 1 000 zu analysierenden Wortformen effizienter ist als eine FST-basierte Analyse (Wintner 2007, S. 466f.).

Wären FST-basierte Morphologie-Systeme der Königsweg, dann wären auch nicht die diversen anderen Ansätze, die im Laufe der Jahre entwickelt und vorgestellt worden sind, zu erklären. So haben Cohen-Sygal/Wintner (2006) das Konzept der ‘Finite State Registered Automaton’ vorgestellt, bei dem FSTs um einen Speicher fester Länge angereichert werden. Mit ihnen wollte man die Morphologie semitischer Sprachen, die sich vor allem durch Zirkumfigierungs- und Infigierungsphänomene auszeichnen, beschreibbar und maschinell verarbeitbar machen. Andere Systeme wie das Erlanger ‘JSlim-System’ (Handl et al. 2009) oder das Heidelberger ‘PLAIN-System’ (Visser/Koch 1996) wiederum setzen auf einer bestimmten Grammatiktheorie auf und stehen der breiten Öffentlichkeit nicht zur Verfügung.

Den Anspruch auf eine hundertprozentige Korrektheit der produzierten Ausgaben oder gar eine vollständige Abdeckung der jeweiligen unterstützten Sprache kann kein Morphologie-Tool erfüllen – egal auf welcher Technologie es basiert. Morphologie-Tools sollte man eher als „best-effort approach“ verstehen; man versucht es, so gut es eben geht. In dem jeweiligen Kontext, in dem sie zum Einsatz kommen, haben sie jedoch trotz ihrer Defizite einen berechtigten Platz und praktischen Nutzen. Und die FST-basierten Systeme haben sich dabei bislang am besten etabliert: Gerade für das Deutsche haben sie mit Morphisto einen Vertreter, der offen vorliegt und frei zur Verfügung steht, eine hohe Abdeckung vorweisen kann, stetig weiterentwickelt und verbessert wird und praxistauglich ist.

5. Literatur

5.1 Forschungsliteratur

- Cohen-Sygal, Yael/Wintner, Shuly (2006): Finite-state registered automata for non-concatenative morphology. In: *Computational Linguistics* 32, 1, S. 49-82.
- Geyken, Alexander/Hanneforth, Thomas (2006): TAGH: A complete morphology for German based on weighted finite state automata. In: Yli-Jyrä/Karttunen/Karhumäki (Hg.), S. 55-66.
- Handl, Johannes et al. (2009): JSLIM – Computational morphology in the framework of the SLIM theory of language. In: Mahlow/Piotrowski (Hg.), S. 10-27.
- IDS Mannheim (2007): Institut für Deutsche Sprache, Programmbereich Korpuslinguistik: Korpusbasierte Wortgrundformenliste DeReWo, v-30000g- 2007-12-31-0.1, mit Benutzerdokumentation. Internet: <http://www.ids-mannheim.de/kl/derewo/> (Stand: 11/2011). Mannheim.
- IDS Mannheim (2009): Institut für Deutsche Sprache, Programmbereich Korpuslinguistik: Korpusbasierte Wortformenliste DeReWo, v-100000t-2009- 04-30-0.1, mit Benutzerdokumentation. Internet: <http://www.ids-mannheim.de/kl/derewo/> (Stand: 11/2011). Mannheim.
- Klabunde, Ralf (1998): *Formale Grundlagen der Linguistik. Ein Arbeitsbuch.* Tübingen.
- Koskenniemi, Kimmo (1983): *Two-level morphology: A general computational model for word-form recognition and production.* Diss. Univ. Helsinki.
- Lindén, Krister/Tuovila, Jussi (2009): Corpus-based lexeme ranking for morphological guesser In: Mahlow/Piotrowski (Hg.), S. 118-135.
- Mahlow, Cerstin/Piotrowski, Michael (Hg.) (2009): *State of the art in computational morphology. Workshop on systems and frameworks for computational morphology, SFCM 2009. Zurich, Switzerland, September 2009, Proceedings.* Heidelberg/Berlin.
- Marek, Torsten (2006): *Analysis of German compounds using weighted finite state transducers.* Bachelorarb. Universität Tübingen.
- Mohri, Mehryar (1997): Finite-state transducers in language and speech processing. In: *Computational Linguistics* 23, 2, S. 269-311.
- Schmid, Helmut (2005): Disambiguation of morphological structure using a PCFG. In: *The Association for Computational Linguistics (Hg.): Proceedings of Human Language Technology Conference and Conference on Empirical Methods in Natural Language Processing (HLT/EMNLP).* Vancouver, S. 515-522.
- Schmid, Helmut (2006): A programming language for finite state transducers. In: Yli-Jyrä/Karttunen/Karhumäki (Hg.), S. 308-309.

- Schmid, Helmut/Fitschen, Arne/Heid, Ulrich (2004): SMOR: A German computational morphology covering derivation, composition, and inflection. In: Proceedings of the IVth International Conference on Language Resources and Evaluation (LREC 2004). Lisbon, Portugal, S. 1263-1266. Internet: www.lrec-conf.org/proceedings/lrec2004/pdf/468.pdf (Stand: 11/2011).
- Visser, Henriette/Koch, Heinz-Detlev (1996): PLAIN. In: Hausser, Roland (Hg.): Linguistische Verifikation. Dokumentation zur Ersten Morpholympics 1994. Tübingen, S. 89-102.
- Wintner, Shuly (2007): Strengths and weaknesses of finite-state technology: A case study in morphological grammar development. In: Natural Language Engineering 14, 4, S. 457-469.
- Yli-Jyrä, Anssi/Karttunen, Lauri/Karhumäki, Juhani (Hg.) (2006): Finite state methods and natural language processing. (= Lecture Notes in Computer Science 4002). Berlin.
- Zielinski, Andrea/Simon, Christian (2008): Morphisto – An open source morphological analyzer for German. In: Piskorski, Jakub S./Watson, Bruce W./Yli-Jyrä, Anssi (Hg.): Finite-state methods and natural language processing, 7th International Workshop, FSMNLP 2008, Ispra, Italy, September 11-12, 2008. (= Frontiers in Artificial Intelligence and Applications 191). Amsterdam u.a., S. 224-231.
- Zielinski, Andrea/Simon, Christian/Wittl, Tilman (2009): Morphisto: Service-oriented open source morphology for German. In: Mahlow/Piotrowski (Hg.), S. 64-75.

5.2 Internetressourcen

- CC BY-SA 2.0-Lizenz für nichtkommerzielle Zwecke. Internet: <http://creativecommons.org/licenses/by-sa/2.0/de/> (Stand: 11/2011).
- Morphisto-Projektseite. Internet: <http://code.google.com/p/morphisto/> (Stand: 11/2011).
- Open-Source-Softwarelizenz GPLv2. Internet: <http://www.gnu.org/licenses/gpl-2.0.html> (Stand: 11/2011).
- SMOR-Projektseite. Internet: <http://www.ims.uni-stuttgart.de/projekte/gramotron/SOFTWARE/SFST.html> (Stand: 11/2011).