Zwei-Ebenen-Morphologie

Morphologieanalyse und Lexikonaufbau (5. Vorlesung)

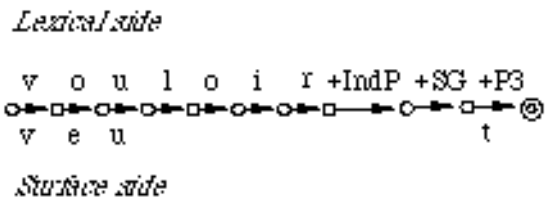
Dozent: Gerold Schneider

Übersicht

Einleitung

Zwei-Ebenen-Morphologie ist eine Theorie zur Beschreibung morphologischer Phänomene von K. Koskenniemi, insbesondere zur Beschreibung der Gesetzmässigkeiten an Morphemgrenzen. Seine Grundannahmen:

- Man benötigt nur zwei Ebenen (Ausgangsform und Oberflächenform) zur morphologischen Beschreibung, da aufeinanderfolgende Regeln aus Zwischenebenen kombiniert werden können.



- Die meisten Regeln sind unabhängig voneinander und können parallel (d.h. simultan) angewendet werden.
- Jede Regel kann in einen deterministischen endlichen Transducer übersetzt werden, der eine effiziente Abarbeitung garantiert.
- Die Verarbeitung ist richtungsneutral (kann für Analyse und Synthese eingesetzt werden).

Literatur:

- [Karttunen91] (online hier, ausführlich, verständlich),
- [Karttunen94] (online hier, spätere Erweiterungen),
- [Ritchie et al. 93] pp. 19-30 (etwas theoretisch),
- [Sproat 92] (m.A. nicht sehr verständlich)
- [Koskenniemi 84]

Beispiel: Bildung der Form 2. Sg Präsens vom Verb `rasen`

```
Ausgangsform:    r a s + s t
                 | | | | | |
Oberflächenform: r a s 0 0 t
```

Benötigt wird also ein Formalismus, der eine Ausgangsform in eine Oberflächenform (und umgekehrt) übersetzen kann. Dazu benutzt man einen sog. Transducer. Dies ist eine Sonderform eines endlichen Automaten, der gleichzeitig durch zwei Zeichenketten läuft.

Endliche Automaten (Finite-State-Machines)

[Dieser Abschnitt ist entnommen aus: *Informatik-eine grundlegende Einführung in vier Teilen (Teil IV), Manfred Broy*: 1.3.2 Endliche Automaten]

********** *Beginn des übernommenen Abschnitts*

Endliche Automaten sind ein sehr einfaches Modell für informationsverarbeitende Maschinen. Ein *endlicher Automat* $A = (S, T, s_0, S_z, \delta)$ ist gegeben durch:

- eine endliche Menge S von Zuständen,
- eine endliche Menge T von Eingangszeichen,
- ein Anfangszustand $s_0 \in S$,
- eine Menge $S_z \subseteq S$ von Endzuständen,
- eine Übergangsfunktion (bzw. -relation) $\delta : S \times (T \cup \{\varepsilon\}) \rightarrow \wp(S)$.

Sind die Werte der Übergangsfunktion höchstens einelementig und gilt

$$\delta(s, \varepsilon) \subseteq \{s\}$$

für alle $s \in S$, so heißt der Automat *deterministisch*, sonst *nichtdeterministisch*.

Ist $\delta(s, a)$ (für $a \in T$) stets verschieden von der leeren Menge, so heißt der Automat *total*, sonst *partiell*.

Ein endlicher Automat läßt sich einfach durch einen durch Teilmengen aus T kantenmarkierten Graphen über der Menge der Zustände darstellen. Es existiert eine Kante vom Zustand s_1 zum Zustand s_2, falls gilt

$$\exists a \in T \cup \{\varepsilon\}: s_2 \in \delta(s_1, a) .$$

Dann wird die Kante mit der Menge

$$\{a \in T \cup \{\varepsilon\}: s_2 \in \delta(s_1, a)\}$$

markiert. Diese Graphen nennen wir *Transitionsgraphen* oder auch *Zustandsübergangsgraphen*.

Beispiel(Endlicher Automat). Wir führen einen endlichen deterministischen Automaten durch Angabe seiner Bestandteile ein:

Zustandsmenge	$S = \{s_0, s_1, s_f\}$,
Eingangszeichen	$T = \{a, b\}$,
Anfangszustand	s_0,
Endzustände	$S_z = \{s_1\}$.

Die Übergangsfunktion wird durch die Tabelle 1.2 beschrieben.

Tabelle 1.2. Übergangstabelle für einen Zustandsautomaten

	a	b
s_0	s_1	s_f
s_1	s_f	s_0
s_f	s_f	s_f

Diese Information können wir auch durch die graphische Darstellung des Automaten in Abb. 1.18 wiedergeben.

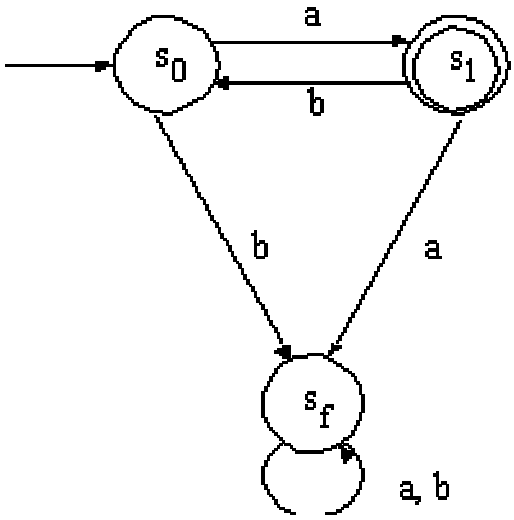


Abb. 1.18. Übergangsgraph eines endlichen Automaten

Aus diesem Transitionsgraph können wir die Bestandteile des Automaten ablesen. Wir kennzeichnen den Anfangszustand mit einem unmarkierten Pfeil ohne Quelle und Endzustände durch doppelte Kreise.

Gilt für einen Zustand $s_f \in S$:

$$\forall t \in T \cup \{\varepsilon\}: \delta(s_f, t) \subseteq \{s_f\},$$

so heißt s_f *Fangzustand.*

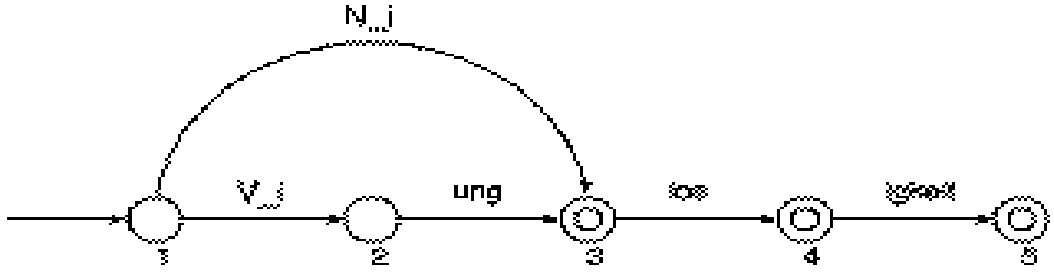
********** *Ende des übernommenen Abschnitts*

Ein endlicher Automat ist ein Formalismus, der beschreibt, wie durch die Abarbeitung von Symbolen von einem Zustand in einen anderen Zustand gewechselt wird. Man kann einen endlichen Automaten nutzen, um einfache Morphemfolgen zu beschreiben. Also z.B. die Abfolge von Verbstamm plus Endung.

`{schwimm,zeig}+{e,st,t,en}`

Zur Beschreibung der Besonderheiten bestimmter Stämme (z.B. Stamm endet auf `-s` wie bei `rasen`), benötigt man entweder eine Klasseneinteilung (mit entsprechenden Endungslisten) oder die Zwei-Ebenen-Morphologie.

Der Nutzen eines endlichen Automaten wird noch deutlicher bei der Beschreibung der Abfolge mehrerer Suffixe. Z.B. bei Adjektiven die Abfolge von Komparations- und Flexionssuffix (Beispiel: `schön+er+em`) oder, wie im folgenden Automaten, bei mehreren Derviationssuffixen. Dabei steht `N_i` für eine Unterklasse aller Substantive, die das Derivationssuffix `-los` nehmen (z.B. `Kopf, Hilf`) und `V_i` für eine Unterklasse aller Verben (genauer: Verbstämme), die eine Substantivierung mit dem Suffix `-ung` bilden können (z.B. `bedeuten, wirken`).



Reguläre Ausdrücke

Die von endlichen Automaten akzeptierten Sprachen lassen sich durch einfache Ausdrücke beschreiben, die als reguläre Ausdrücke bezeichnet werden. Sei T eine endliche Menge von Eingangszeichen (auch Alphabet genannt). Dann werden die regulären Ausdrücke über T wie folgt definiert:

1. Die leere Menge ist ein regulärer Ausdruck.
2. ε ist ein regulärer Ausdruck.
3. Jedes $a \in T$ ist für sich ein regulärer Ausdruck.
4. Wenn *r* und *s* reguläre Ausdrücke sind, so sind auch *(r | s)* [d.h. r oder s], *(rs)* [d.h. r gefolgt von s] und *(r*)* [d.h. r null oder mehrmals] reguläre Ausdrücke.

Merke: $(r+) = (r(r^*))$, $(r?) = (r \mid \varepsilon)$

[Der folgende Abschnitt ist entnommen aus: *Informatik - eine grundlegende Einführung in vier Teilen (Teil IV), Manfred Broy*: 1.3.4 Reguläre Ausdrücke, endliche Automaten und Chomsky-3-Sprachen]

********** *Beginn des übernommenen Abschnitts*

Satz: Zu jedem regulärem Ausdruck können wir einen nichtdeterministischen endlichen Automaten angeben, der die durch den regulären Ausdruck angegebene Sprache akzeptiert.

Beweis: Wir realisieren den regulären Ausdruck durch endliche Automaten mit genau einem Anfangszustand, einem Endzustand und ohne Kanten, die in den Anfangszustand oder aus dem Endzustand führen. Wir geben für jede Form, die ein regulärer Ausdruck haben kann, induktiv einen endlichen Automaten an. Wir benennen zur Vereinfachung der Darstellung die Zustände nicht.

(1) Für den regulären Ausdruck x mit x $\in$ T verwenden wir den Automaten aus Abb. 1.24.

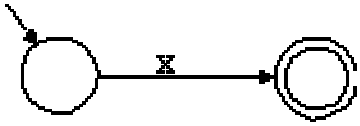


Abb. 1.24. Automat, der die Sprache {x} akzeptiert

Für den regulären Ausdruck ε verwenden wir den Automaten aus Abb. 1.25.

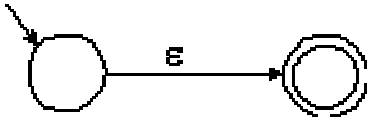


Abb. 1.25. Automat, der die Sprache { ε } akzeptiert

Für den regulären Ausdruck {} verwenden wir den Automaten aus Abb. 1.26.

Abb. 1.26. Automat für die leere Sprache

(2) Seien X und Y reguläre Ausdrücke mit den Automaten in Abb. 1.27.

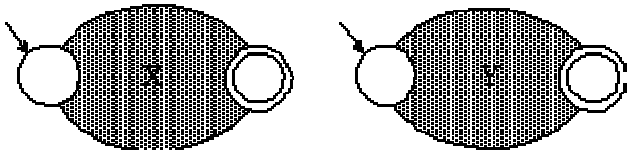


Abb. 1.27. Gegebene Automaten für die Sprache der regulären Ausdrücke X und Y

Für den regulären Ausdruck [X|Y] verwenden wir den Automaten in Abb. 1.28.

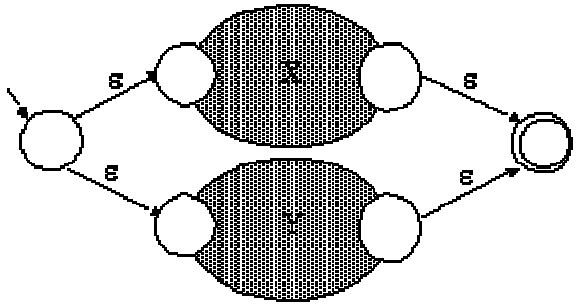


Abb. 1.28. Automat für Sprache [X|Y]

Für den regulären Ausdruck [XY] verwenden wir den Automaten in Abb. 1.29.

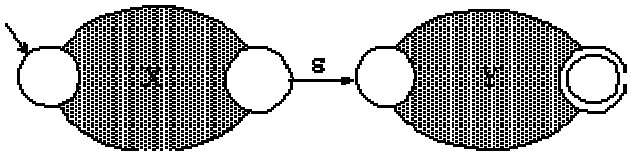


Abb. 1.29. Automat für die Sprache [XY]

Für den regulären Ausdruck X^* verwenden wir den in Abb. 1.30 angegebenen Automaten.

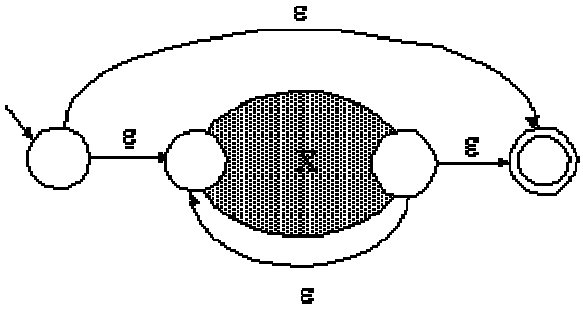


Abb. 1.30. Automat für die Sprache X^*

********** *Ende des übernommenen Abschnitts*

Wichtig: Es gelten folgende Äquivalenzen:

1. Jeder nichtdeterministische endliche Automat (N-FSA) kann in einen äquivalenten deterministischen endlichen Automaten (D-FSA) überführt werden.
2. Jeder deterministische endliche Automat (D-FSA) kann in einen äquivalenten regulären Ausdruck überführt werden.
3. Jeder reguläre Ausdruck kann in einen äquivalenten nichtdeterministischen endlichen Automaten mit ε-Transitionen (Nε-FSA) überführt werden (wie oben gesehen).
4. Jeder nichtdeterministische endliche Automat mit ε-Transitionen (Nε-FSA) kann in einen äquivalenten nichtdeterministischen endlichen Automaten (N-FSA) überführt werden.

Transducer (Finite-State Transducer)

Während ein endlicher Automat eine Sprache über einem Alphabet mit einfachen Symbolen (z.B. `T = {a,b,c}`) akzeptiert, akzeptiert ein Transducer eine Sprache über Paaren von Symbolen (z.B. `T = {a:a, b:b, c:c, a:b, a:0, 0:c}`).

Bsp.:

```
T = {a:0, a:a, b:b}
RegExpr = "a:a*  a:0  b:b*"

Akzeptiert z.B.: <aaabbb,aabbb>, <a,0>, <abb,bb>
Akzeptiert nicht: <aaabbb,aaabbb>, <bb,bb>

* steht für null oder mehr
+ steht für 1 oder mehr
0 steht für ein leeres Symbol
```

Ein Transducer (wie auch ein FSM) kann durch eine Übergangstabelle repräsentiert werden.

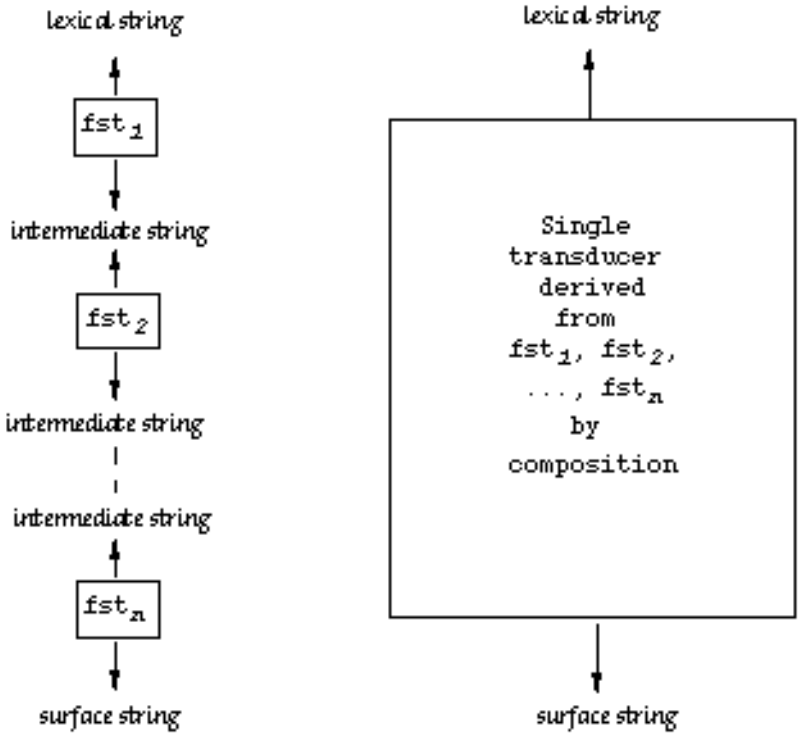
Tabelle: (in der linken Spalte stehen durchnumeriert die Zustände; ein Doppelpunkt deutet an, dass es sich um einen Endzustand handelt; der Zustand '1' ist der Anfangszustand; Ø steht für einen Fehlerzustand)

	a a	a 0	b b
1.	1	2	Ø
2:	Ø	Ø	2

Die Paare in einer Transducer-Tabelle können so interpretiert werden, dass jeweils der erste Buchstabe zu einer (hypothetischen) Ausgangsform (engl. *lexical form* oder *underlying form*) gehört und der zweite Buchstabe jeweils ein Bestandteil der Oberflächenform (engl. *surface form*) ist. Man spricht auch von einem Eingabe- und einem Ausgabeband. Da jedoch die Richtung unbestimmt ist, kann die Interpretation wechseln.

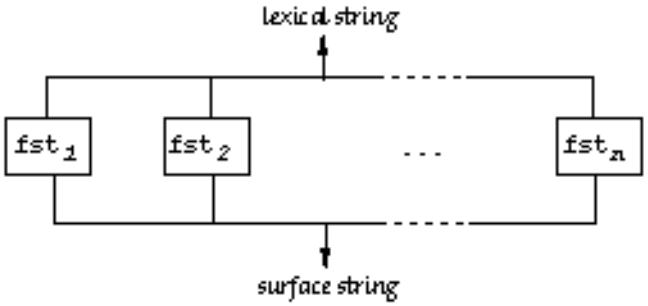
Problem: Wie geht man vor, wenn die Anwendung eines FST die Anwendung eines anderen FST auslöst?

- 1. Lösung: man schaltet die FST in Serie hintereinander. Das Ausgabeband von FST1 ist dann das Eingabeband von FST2.



Allenfalls lässt sich ein Gesamttransducer aus den einzelnen zusammensetzen. Es hat sich jedoch herausgestellt, dass der herzustellende Gesamttransducer unüberschaubar komplex und extrem nicht-deterministisch werden kann, so dass seine Herstellung sehr aufwendig und nicht-automatisierbar und seine Anwendung sehr ineffizient wird.

- 2. Lösung (von Koskenniemi): man führt die FSTs parallel aus und zwar lediglich auf einem Eingabe- und einem Ausgabeband (zwei-Ebenen-Morphologie). Dazu muss man festlegen, wie die parallelen FSTs koordiniert werden. Man nimmt immer den FST mit den spezifischeren (detaillierteren) Bedingungen.



Dazu muss man festlegen, wie die parallelen FSTs koordiniert werden. Man nimmt immer den FST mit den spezifischeren (detaillierteren) Bedingungen (disjunctive ordering). Mit wenigen Ausnahmen, bei denen man nicht-linguistische Hacks programmieren muss, funktioniert das richtig. (siehe [Karttunen91] für eine Diskussion)

Beispiele aus der Morphologie

1. Die Regel: Ein Anfangs-`s` einer Flexionsendung wird eliminiert, wenn der Verbstamm auf `s, x` oder `z` endet. (Bsp.: `ras+st -> rast`, `mix+st -> mixt`). `pi` steht für beliebige Paare.

   ```
   T = {s:s, s:0, x:x, z:z, +:0}
   RegExpr  = "pi+ (s:s | x:x | z:z) +:0 s:0 pi+"
   ```

   Ein Beispielprogramm in Prolog.

2. Die Regel: Ein Anfangs-`t` einer Flexionsendung wird eliminiert, wenn der Verbstamm auf `t` endet. (Bsp.: `tritt+t -> tritt`)

   ```
   T = {t:t, t:0, +:0}
   RegExpr = "pi+ t:t +:0  t:0  pi*"
   ```

Merke: Transducer sind in ihrer Funktion als Analysator (oder Generator) nicht immer deterministisch (d.h. ihre Implementation erfordert Backtracking)

Bsp.:

```
T = {x:a, x:b, a:a, b:b}
RegExpr = "(x:a* a:a) | (x:b* b:b)"

Akzeptiert:  <xxxa, aaaa>,  <xxxxb, bbbb>

        x       x       a       b
        a       b       a       b
-------------------------------------
1.      2       3       Ø       Ø
2.      2       Ø       4       Ø
3.      Ø       3       Ø       4
4:      Ø       Ø       Ø       Ø
```

Bei Eingabe von `xxxxa` kann der Transducer erst bei `a` entscheiden, ob er den richtigen Pfad gewählt hat.

Ein höherer Abstraktionsgrad: Koskenniemis Regelformalismus

(nach [Sproat 92] S. 145): Um morphologische Phänomene eleganter beschreiben zu können, als das mit regulären Ausdrücken möglich ist, führt Koskenniemi einen Regelformalismus auf einer höheren Abstraktionsebene ein. Diese Regeln nennt er *Two-Level*-Regeln, kurz TWOL-Regeln. Diese können automatisch in Transducer übersetzt werden.

TWOL-Regeln haben die Syntax:

`CP` **op** `LC _ RC,` wobei

- `CP` = Übereinstimmungsteil (correspondence part); ein regulärer Ausdruck über dem Alphabet der möglichen Paare.
- `LC, RC` = linker und rechter Kontext; jeweils ein regulärer Ausdruck über dem Alphabet der möglichen Paare.
- **op** = Operator mit 4 Möglichkeiten:
  1. Exclusion rule: `a` darf **nicht** als `b` realisiert werden im Kontext `LC _ RC`
     `a:b /<= LC _ RC`
  2. Context restriction rule: `a` darf als `b` realisiert werden **nur** im Kontext `LC _ RC`
     `a:b => LC _ RC`
  3. Surface coercion rule: `a` muss als `b` realisiert werden im Kontext `LC _ RC`
     `a:b <= LC _ RC`
  4. Composite rule: `a` muss als `b` realisiert werden im Kontext `LC _ RC` und nur in diesem Kontext (Kombination aus 2. und 3.)
     `a:b <=> LC _ RC`

Umsetzung von Koskenniemis Regelformat in Transducer

Hier beispielhaft gezeigt für eine Surface Coercion Regel (nach [Sproat 92] S.154-156). Seien `b` und `v` Buchstaben und `V` die Menge aller Vokale. Dann heisst:

```
b:v  <= V:V _ V:V
```

`b` muss durch `v` ersetzt werden, wenn davor und dahinter ein Vokal auftritt. Die Vokale werden durch die Regelanwendung nicht geändert.

1. Berechnung der Ausschlussmenge (rejection set). Dazu:
   - Bestimmung der möglichen Paare mit `b`. Nehmen wir an, das sei `{b:b, b:v, b:p}`. Dann besagt unsere Regel, dass `b:b` und `b:p` zwischen zwei Vokalen ausgeschlossen sind.
   - Erstellung eines regulären Ausdrucks, der die Ausschlussmenge beschreibt:
     `pi* V:V (b:b|b:p) V:V pi*`
     wobei `pi*` die Menge der möglichen Paare symbolisiert.
2. Erstellung eines FST *X* für die Ausschlussmenge. Dazu:
   - Erstellung von FSTs für jeden Bestandteil des regulären Ausdrucks zur Ausschlussmenge. D.h. je einen FST für
     `pi*  V:V  (b:b|b:p)`
     und
   - Konkatenation dieser FSTs in den Gesamt-FST *X*

3. Erstellung eines FST *Y*, der die Komplementmenge zu FST *X* akzeptiert. Dazu:
   - Erstellung eines deterministischen FST *Z* aus FST *X*
   - Erstellung des FST *Y* aus FST *Z* durch Umkehrung der Zustände. D.h. jeder Endzustand wird zu einem Nicht-Endzustand und umgekehrt.

Beispiele zu Koskenniemis Regelformalismus

1. **Bsp.** (aus [Trost 91] S.428) ein Anfangs-`s` in einer Flexionsendung wird eliminiert, wenn der Stamm auf `s`, `x` oder `z` endet. (`ras+st -> rast`, `mix+st -> mixt`)
   ```
   s:0 <=> {s:s x:x z:z} +:0 _
   ```
2. Bsp. (aus [Trost 91] S.439) ein Schwa-`e` muss eingefügt werden zwischen Stamm, der auf `d` oder `t` endet und Flektionsendung, die mit `s` oder `t` beginnt. (`rast+st -> rastest`, `red+st -> redest`)
   ```
   +:e <=> {d:d  t:t} _ {s:s  t:t}
   ```

3. Komplexe TWOL-Regeln für Umlautung von deutschen Substantiven. (Gertwol-Beschreibung im LDV-Forum 1 1994 S.19)
   - Umlaut-Trigger ist nur in den betreffenden Substantiven vorhanden.
   - Der Umlaut-Trigger (@U) erzwingt die Umlautung von a nach ä genau dann, wenn es zwischen a und dem Trigger keine Wortgrenze (%#) gibt und kein Vorkommen von a, o oder u. Es darf jedoch ein u genau hinter dem a auftreten.

     ```
     "Uml a~ä" a:ä <=>
     _\[%#: | a: | o: | u:]* @U:;
     _u: \[%#: | a: | o: | u:]* @U:;
     ```

   - u wird zu ü nach dem selben Schema wie oben, ausser wenn es unmittelbar hinter a steht.

     ```
     "Uml u~ü"   u:ü <=>
     \a: _\[%#: | a: | o: | u:]* @U:;
     ```

4. Weitere Einsatzmöglichkeiten für TWOL-Regeln: siehe Gertwol: 2.6. DIE WICHTIGSTEN TWOL-REGELN.

Beispiele aus PC-Kimmo

Hintergrund-Informationen zu PC-Kimmo

```
PC-KIMMO>rec hospitalization
`hospital+ize+ation     `hospital+VR6+NR23

1:
             Word
               |
             Stem
           _____|______
        Stem       SUFFIX
      ____|____    +ation
    Stem    SUFFIX   +NR23
      |      +ize
    ROOT     +VR6
  `hospital
  `hospital

Word:
[ cat:   Word
  clitic:-
  head:    [ number:SG
             pos:   N ]
  lemma: `hospital
  lemma_pos:N
  nonreg:- ]

1 parse found

PC-KIMMO>
```

Beispieldatei (Auszug) mit Zwei-Ebenen-Regeln für Englisch:

```
;english.rul
;Rules file for Englex, annotiert GS
…
;y:i-spelling e.g. berrIes, trIes, carrIer
RULE " y:i => @:C (+:0)_+:0" 4 5

      y  @ e  +  @
      i  C 0  0  @
 1:   0  2 1  1  1
 2:   3  2 4  4  1
 3.   0  0 0  1  0
 4:   3  2 1  4  1

; but y remains in e.g. trYIng, MarY's
RULE " y:i /<= @:C (+:0)_+:0 [i|']" 5 7
      @  y  e  +  '  i  @
      C  i  0  0  '  i  @
 1:   2  1  1  1  1  1  1
 2:   2  3  5  5  1  1  1
 3:   2  1  1  4  1  1  1
 4:   2  1  1  1  0  0  1
 5:   2  3  1  1  1  1  1

;Epenthesis e.g. churchEs, heroEs, berriEs
RULE " 0:e <= [Csib|ch|sh|y:i] +:0_s [+:0|#]" 6 8
      Csib  +  s  #  c  h  y  @
      Csib  0  s  #  c  h  i  @
 1:      2  1  6  1  5  1  2  1
 2:      2  3  6  1  5  1  2  1
 3:      2  1  4  1  5  1  2  1
 4:      2  0  6  0  5  2  2  1
 5:      2  1  6  1  5  2  2  1
 6:      2  3  6  1  5  2  2  1

RULE " 0:e => [Csib|ch|sh|y:i|o] +:0_s [+:0|#]" 7 10
      0  Csib  +  s  #  c  h  y  o  @
      e  Csib  0  s  #  c  h  i  o  @
 1:   0     2  1  7  1  6  1  2  2  1
 2:   0     2  3  7  1  6  1  2  2  1
 3:   4     2  1  7  1  6  1  2  2  1
 4.   0     0  0  5  0  0  0  0  0  0
 5.   0     0  3  0  1  0  0  0  0  0
 6:   0     2  1  7  1  6  2  2  2  1
 7:   0     2  3  7  1  6  2  2  2  1
```

Zusammenfassung

1. Zwei-Ebenen-Morphologie ist eine Theorie, die beschreibt, wie Morpheme kombiniert werden können.
2. Zwei-Ebenen-Morphologie kann mit einem Transducer (ein endlicher Automat mit einem Eingabe- und einem Ausgabeband) effizient implementiert werden.
3. Koskenniemis Regelformalismus erlaubt eine abstrakte Formulierung der morphologischen Gesetzmässigkeiten. Die Regeln können automatisch in einen Transducer übersetzt werden.

Gerold Schneider, Martin Volk