

# Computerlinguistik I

Vorlesung im WS 2009/10

Prof. Dr. Udo Hahn

Lehrstuhl für Computerlinguistik  
Institut für Germanistische Sprachwissenschaft  
Friedrich-Schiller-Universität Jena

## Morphologische Prozesse: Flexion - Deflexion

- Kombination von **Grundformen** mit **Flexionsaffixen** (Kasus, Numerus, Tempus usw.)
  - Deklination
    - **Land**: Land, Land**es**, Land**e**, L**ä**nder, L**ä**ndern
  - Konjugation
    - **landen**: land**e**, land**est**, land**et**, land**eten**, **gelandet**
- primär syntaktische, nur minimale semantische Information, kein Wortartwechsel

2

## Morphologische Prozesse: Derivation - Dederivation

- Kombination von **Grundformen** mit **Derivationsaffixen**
  - **Land**: land**en**, ver**land**en, an**land**en,
  - **Land**: Land**ung**, Ver**land**ung, An**land**ung
  - **Land**: l**änd**lich, ver**l**änd**lich**en, Ver**l**änd**lich**ung
- modifizierende semantische Information, häufig mit Wortartwechsel verbunden

3

## Morphologische Prozesse: Komposition - Dekomposition

- Kombination von **Grundformen** mit **Grundformen** (mittels **Fugeninfixen**)
  - **Land**: Land**na**hme, Land**fl**ucht, Land**g**ang
  - **Land**: Heim**at**land, Aus**land**, Bau**land**
  - **Land**: Land**es**rekord, Land**es**verrat, Land**sm**ann
  - **Land**: In**land**sflug, Land**es**ratspr**ä**sident**en**gatt**in**
- starke semantische Modifikation, kein Wortartwechsel

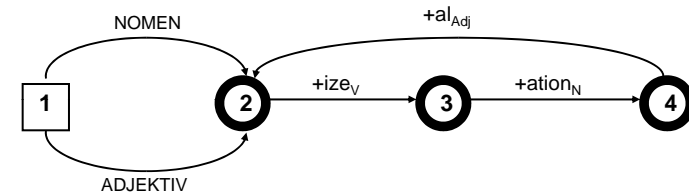
4

# Lemmatisierung vs. Wort-Parsing

Eingabe	Lemma	Wort-Parse
Töchtern	Tochter	Tochter
Hauses	Haus	Haus
sagte	sagen	sagen
Spiegelungen	Spiegelung	[Spiegel] <sub>N</sub> [ung] <sub>ds</sub>
leichter	leicht	leicht
verlängerte	verlängert	[ver] <sub>dp</sub> [[lang] <sub>Adj</sub> [er] <sub>ds</sub> ] <sub>Adj</sub> [t] <sub>ds</sub>
	verlängern	[ver] <sub>dp</sub> [[lang] <sub>Adj</sub> [er] <sub>ds</sub> ] <sub>Adj</sub> [n] <sub>ds</sub>

5

# Automat für Dederivation



NOMEN: hospital, motor, category, ...

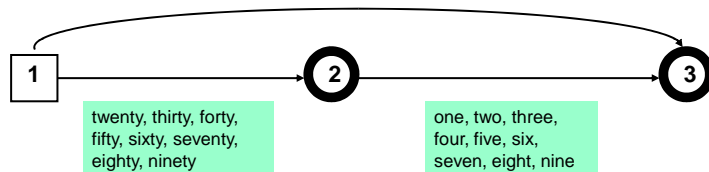
ADJEKTIV: moral, concrete, tender, ...

n Anfangszustand    n möglicher Endzustand

6

# Automat für englische Zahlen von 1 bis 99

one, two, three, four, five, six, seven, eight, nine, ten,  
eleven, twelve, thirteen, fourteen, fifteen, sixteen, seventeen, eighteen, nineteen



n Anfangszustand    n möglicher Endzustand

7

# Formale Grundlagen von Automaten: Alphabet

- Sei  $\Sigma$  ein beliebiges **Alphabet**, d.i. eine Menge von Symbolen oder Zeichen

– Beispiele für verbreitete Alphabete:

- {A,B,C, ..., X,Y,Z} lateinisches Alphabet
- {1,2,3, ..., 7,8,9, 0} indisch-arabisches Zahlensystem
- {0,1} Binärzahlen
- { ●, ●, ● } internat. Ampelalphabet
- {A[denin], G[uanin], T[hymin], C[ytosin]} Basen-Alphabet der DNA

8

## Formale Grundlagen von Automaten: Ketten

- Seien **Ketten** über einem Alphabet  $\Sigma$  in der folgenden Weise definiert:
  1.  $\epsilon$  ist eine Kette über  $\Sigma$  ( $\epsilon$  ist die leere Kette, die keine Symbole hat)
  2. falls  $\chi$  eine Kette über  $\Sigma$  und  $\alpha \in \Sigma$  ist, dann ist  $\chi \alpha$  eine Kette über  $\Sigma$
  3.  $\gamma$  ist eine Kette über  $\Sigma$  genau dann, wenn ihre Bildung aus (1) oder (2) folgt

9

## Formale Grundlagen von Automaten: Konkatination

- Die Kette  $\omega \circ \tau$  heißt **Konkatination** von  $\omega$  und  $\tau$ , falls  $\omega$  und  $\tau$  Ketten sind;  
„ $\circ$ “ (sprich: „Kringel“) ist der Konkatinationsoperator.  
 $\omega \circ \tau$  wird auch als  $\omega\tau$  geschrieben.
  - Für alle Ketten  $\omega$  gilt:  $\omega \circ \epsilon = \epsilon \circ \omega = \omega$
  - **Beispiele für Konkatinationen:**
    - $ABC \circ X = ABCX$
    - $24 \circ 24 = 2424$
    - $101 \circ 00 = 10100$

10

## Formale Grundlagen von Automaten: formale Sprache

- Eine (**formale**) **Sprache**  $\mathcal{L}$  über einem Alphabet  $\Sigma$  ist eine Menge von Ketten über  $\Sigma$ .
- Sei ferner  $\Sigma^*$  (bzw.  $\Sigma^+$ ) die Menge *aller* Ketten über  $\Sigma$  unter Einschluss (bzw. Ausschluss) von  $\epsilon$ .
- Dann gilt für jede Sprache  $\mathcal{L}$  über  $\Sigma$ :

$$\mathcal{L} \subseteq \Sigma^*$$

11

## Beispiele für formale Sprachen

- $\Sigma = \{A, B, C, \dots, X, Y, Z, \_ \}$ 
  - $\mathcal{L}_1 = \{\text{GUTEN\_TAG}, \text{GUTEN}, \text{TAG}\}$   
 $= \{\text{GUTEN}, \text{GUTEN\_TAG}, \text{TAG}\}$
  - $\mathcal{L}_2 = \{\text{GTNTG}, \text{GTN}, \text{TG}\}$
  - $\mathcal{L}_3 = \{\text{TNT}, \text{TN}, \text{T}\}$
  - $\mathcal{L}_4 = \{\text{GG}, \text{GTTG}, \text{GGGG}, \text{GGTTGG}, \dots\}$
  - $\mathcal{L}_5 = \{\text{A}, \text{C}, \text{G}, \text{T}, \text{ACCGTG}, \dots\}$

12

## Beispiele für formale Sprachen

- $\Sigma = \{1,2,3, \dots, 7,8,9,+,-,=\}$ 
  - $\mathcal{L}_1 = \{5+7=20-8, 5+7=5+8, 5759=57-59\}$
  - $\mathcal{L}_2 = \{5+=7-2=, +++, 1-11782---3\}$
  - $\mathcal{L}_3 = \{3, 33, 333, 3333, 33333, \dots\}$

13

## Beispiele für formale Sprachen

- $\Sigma = \{\text{NOMEN, ADJEKTIV, +ize}_V, +ation_N, +al_{Adj}\}$ 
  - $\mathcal{L}_1 = \{\text{hospital, hospital+ize}_V, \text{hospital +ize}_V+\text{ation}_N, \dots\}$
  - $\mathcal{L}_2 = \{\text{hospital, moral, hospital+ize}_V, \text{moral+ize}_V, \dots\}$
  - $\mathcal{L}_3 = \{+ize_V, +ize_V+ize_V, +ize_V+ize_V\text{hospital}, \dots\}$

NOMEN: hospital, motor, category, ...

ADJEKTIV: moral, concrete, tender, ...

14

## Endlicher Automat

- Ein endlicher Automat (finite-state auto-maton, FSA) ist ein formales System zur **Erkennung von** (formalen) **Sprachen**.
- Ein FSA beginnt den Erkennungsprozess in seinem ausgezeichneten **Startzustand**. Falls der FSA die Eingabe komplett gelesen hat und er sich in einem der ausgezeichneten **Endzustände** befindet, hat er die Eingabe **akzeptiert**, sonst nicht.

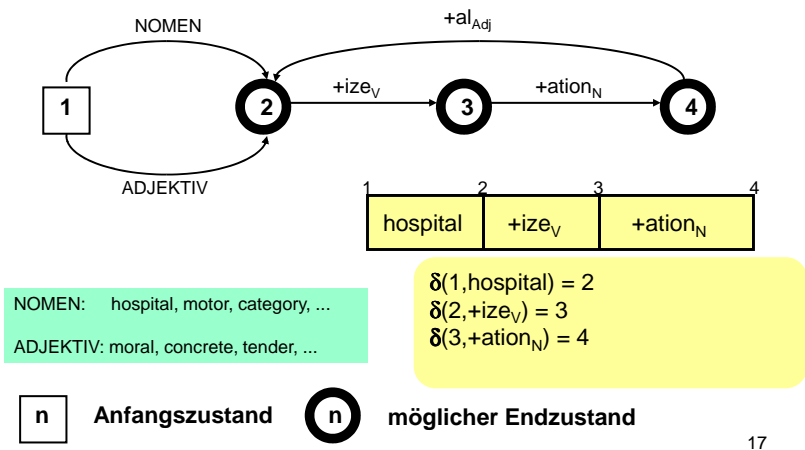
15

## Endlicher Automat

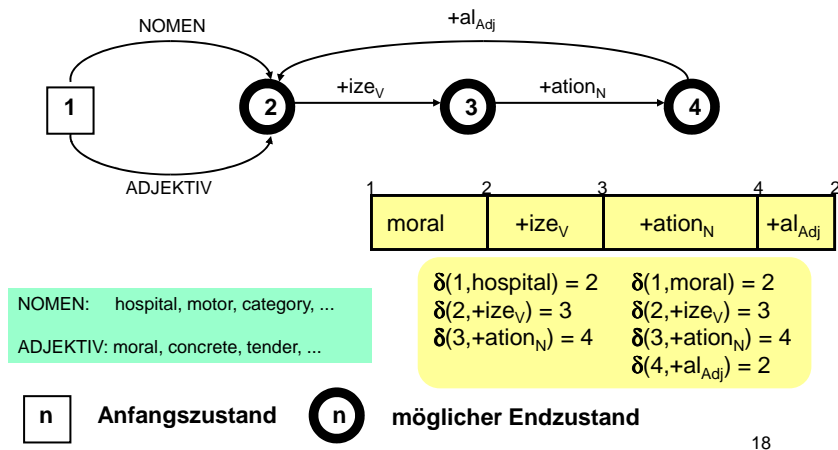
- Ein FSA besteht i.A. aus einem **Eingabeband**, auf dem der zu akzeptierende Ausdruck steht, und einem endlichen **Steuerungs-mechanismus**, der die Form der erlaubten Zustandsänderungen (Bewegungen) determiniert.
- Die **Zustandsänderungsfunktion** gibt an, welche möglichen Folgezustände aus dem aktuellen Zustand und dem aktuellen Symbol auf dem Eingabeband erreichbar sind.

16

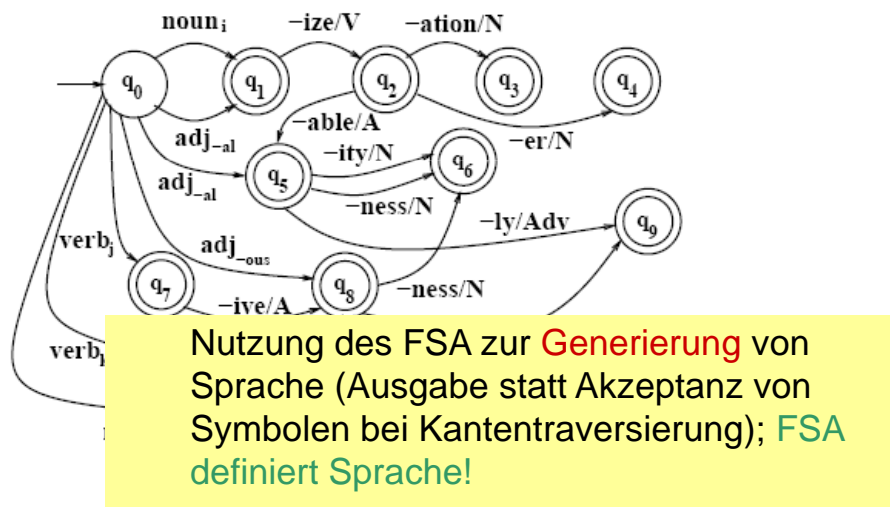
# Automat für Dederivation



# Automat für Dederivation



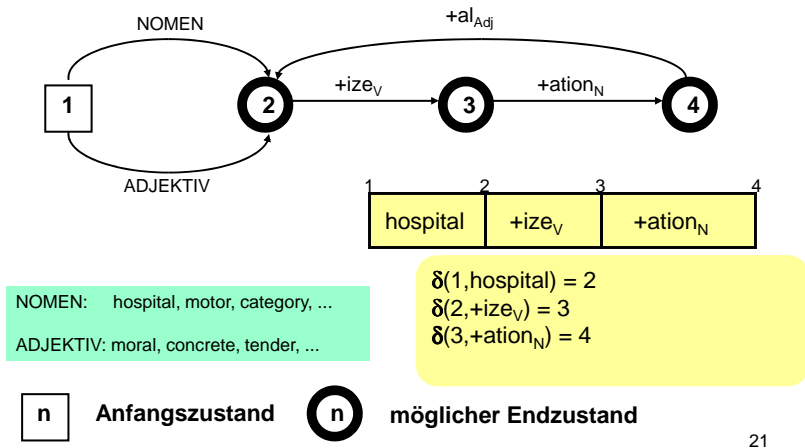
# Komplexerer Dederivationsautomat



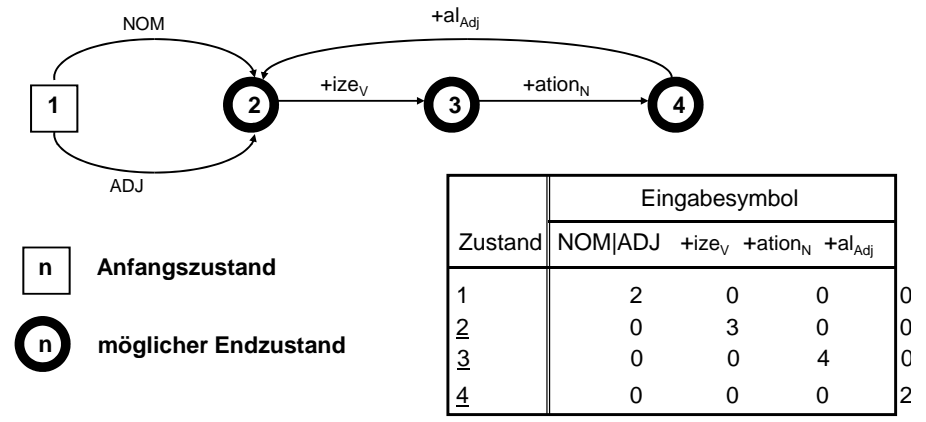
# Formale Grundlagen von Automaten: endlicher Automat

- Ein (nicht-deterministischer) **endlicher Automat** ist ein 5-Tupel  $(Q, \Sigma, \delta, q_0, F)$  mit
  - $Q$ : endliche Menge von (Steuerungs-)Zuständen  
 $q_0, q_1, q_2, \dots, q_n$
  - $\Sigma$ : endliches Eingabealphabet
  - $\delta$ : Zustandsübergangsfunktion, die das Steuerungsverhalten des FSA determiniert  
 $\delta: Q \times \Sigma \mapsto \wp(Q)$
  - $q_0 \in Q$ : der ausgezeichnete Startzustand
  - $F \subseteq Q$ : Menge der ausgezeichneten Endzustände

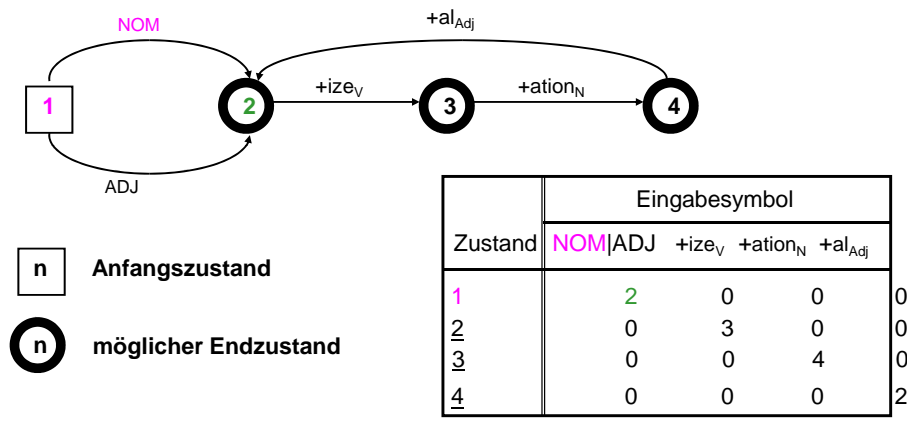
# Graph- und Funktionsnotation für Dederivationsautomaten



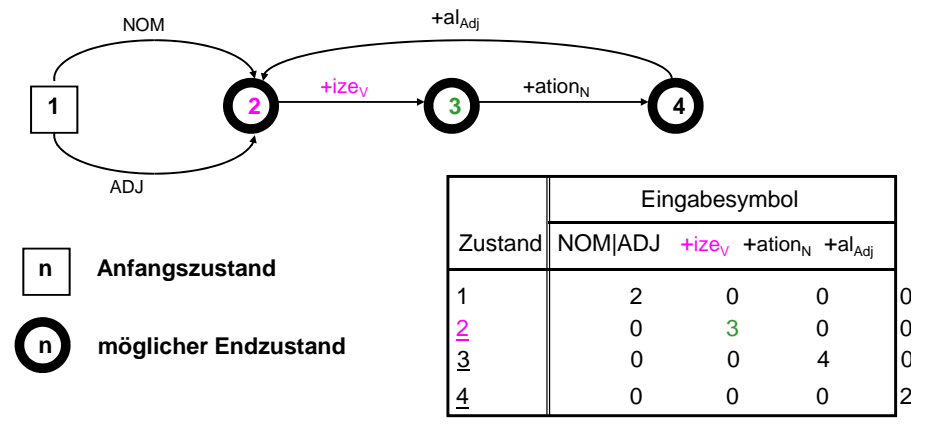
# Graph- und Tabellennotation für Dederivationsautomaten



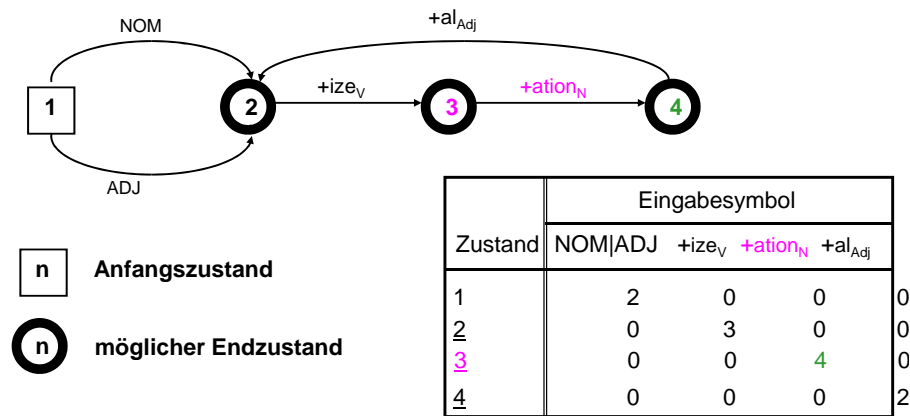
# Graph- und Tabellennotation für Dederivationsautomaten



# Graph- und Tabellennotation für Dederivationsautomaten

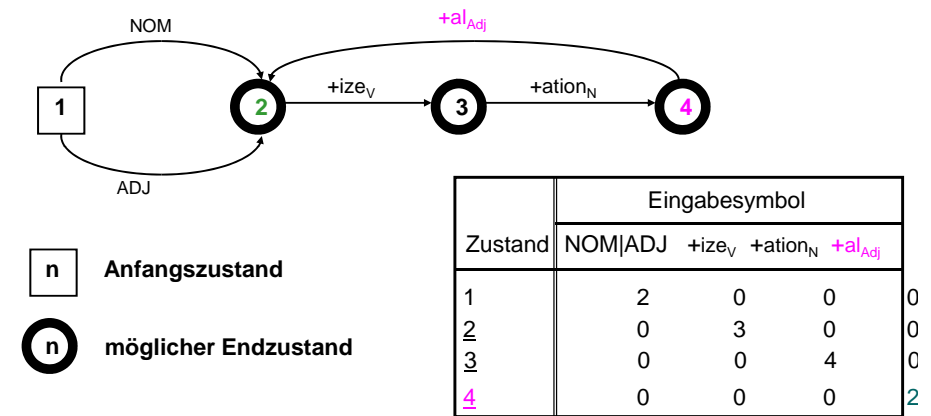


## Graph- und Tabellennotation für Dederivationsautomaten



25

## Graph- und Tabellennotation für Dederivationsautomaten



26

## Endlicher Automat

- Das Verhalten eines FSAs hängt von zwei Arten von Informationen ab:
  - dem aktuellen Zustand des endlichen Steuerungssystems,
  - der Symbolkette auf dem Eingabeband, die aus dem aktuellen Symbol unter dem Lesekopf und allen folgenden Symbolen rechts vom aktuellen Symbol besteht

27

## Formale Grundlagen von Automaten: Konfiguration

- Sei  $FSA = (Q, \Sigma, \delta, q_0, F)$  ein (nicht-deterministischer) endlicher Automat.  
 Dann heie ein Paar  $(q, \omega) \in Q \times \Sigma^*$  eine **Konfiguration** von FSA.  
 Eine Konfiguration der Form  $(q_0, \omega)$  heie **initiale Konfiguration**, eine der Form  $(q, \epsilon)$ , wobei  $q \in F$ , heie **akzeptierende** bzw. **Endkonfiguration**.

28

## Formale Grundlagen von Automaten: Bewegung

- Sei  $FSA = (Q, \Sigma, \delta, q_0, F)$  ein (nicht-deterministischer) endlicher Automat.

Eine **Bewegung** in FSA wird durch eine binäre Relation  $\vdash_{FSA}$  („geht unter FSA nach“) über Konfigurationen repräsentiert.

Falls  $q' \in \delta(q, \tau)$  ( $q'$  ist also ein möglicher Folgezustand von  $q$ ,  $\tau \in \Sigma$ ), dann gilt:

$$(q, \tau\gamma) \vdash_{FSA} (q', \gamma) \text{ für alle } \gamma \in \Sigma^*$$

29

## Formale Grundlagen von Automaten: Akzeptanz

- Sei  $FSA = (Q, \Sigma, \delta, q_0, F)$  ein (nicht-deterministischer) endlicher Automat und  $\vdash_{FSA}^*$  die reflexive und transitive Hülle von  $\vdash_{FSA}$ .

– **reflexive Hülle:**

- $(q, \tau) \vdash_{FSA} (q, \tau)$  für alle  $(q, \tau) \in Q \times \Sigma^*$

– **transitive Hülle:**

- $(q', \tau_1) \vdash_{FSA} (q'', \tau_2)$  und  $(q'', \tau_2) \vdash_{FSA} (q''', \tau_3)$   
 $\Rightarrow (q', \tau_1) \vdash_{FSA} (q''', \tau_3)$  für alle  $(q, \tau) \in Q \times \Sigma^*$

30

## Formale Grundlagen von Automaten: Akzeptanz

- Sei  $FSA = (Q, \Sigma, \delta, q_0, F)$  ein (nicht-deterministischer) endlicher Automat und  $\vdash_{FSA}^*$  die reflexive und transitive Hülle von  $\vdash_{FSA}$ .

Eine Eingabekette  $\omega$  wird durch den FSA **akzeptiert**, wenn

$$(q_0, \omega) \vdash_{FSA}^* (q, \epsilon) \text{ für ein } q \in F.$$

31

## Formale Grundlagen von Automaten: akzeptierte Sprache

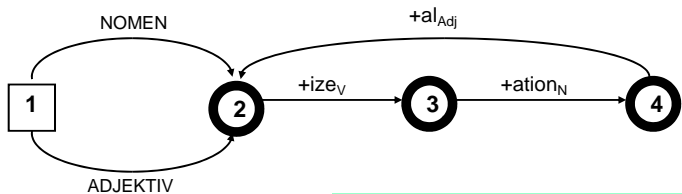
- Sei  $FSA = (Q, \Sigma, \delta, q_0, F)$  ein (nicht-deterministischer) endlicher Automat und  $\vdash_{FSA}^*$  die reflexive und transitive Hülle von  $\vdash_{FSA}$ .

Die (**formale**) **Sprache**  $\mathcal{L}_{FSA}$ , die durch FSA definiert wird, ist die Menge von Eingabeketten, die vom FSA **akzeptiert** wird:

$$\mathcal{L}_{FSA} := \{ \omega \mid \omega \in \Sigma^*, (q_0, \omega) \vdash_{FSA}^* (q, \epsilon) \text{ für ein } q \in F \}$$

32

# Automat für Dederivation

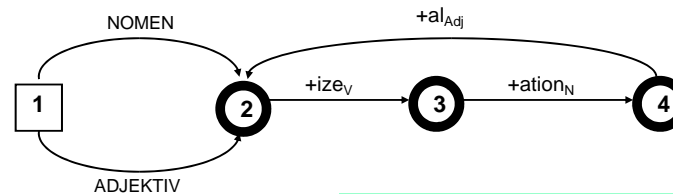


NOMEN: hospital, motor, category, ...  
 ADJEKTIV: moral, concrete, tender, ...

n Anfangszustand    n möglicher Endzustand

hospital+ize<sub>V</sub>+ation<sub>N</sub> ∈  $\mathcal{L}_{FSA}$  ?

# Automat für Dederivation



NOMEN: hospital, motor, category, ...  
 ADJEKTIV: moral, concrete, tender, ...

n Anfangszustand    n möglicher Endzustand

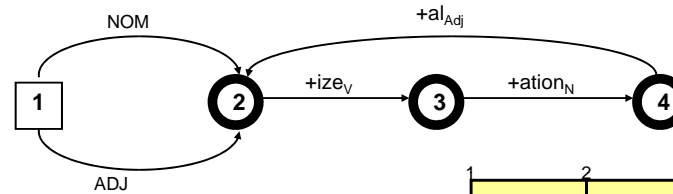
hospital+ize<sub>V</sub>+ation<sub>N</sub> ∈  $\mathcal{L}_{FSA}$  !

$\{ (1, \text{hospital+ize}_V\text{+ation}_N), (2, \text{+ize}_V\text{+ation}_N), (3, \text{+ation}_N), (4, \epsilon) \} \subseteq \vdash^*_{FSA}$

# Deterministischer FSA-Erkennungsalgorithmus

- **Band** (für die zu testende Kette)
  - in n Zellen aufgeteilt
  - jede Zelle hält ein Symbol der Kette
- **Zustandstranstabelle** (2-D Matrix)
  - Zeilen: Zustandsmarken des FSA
  - Spalten: Symbole des Alphabets
  - Zelle: Folgezustand

# Automat für Dederivation



NOM: hospital, motor, category, ...  
 ADJ: moral, concrete, tender, ...

n Anfangszustand    n möglicher Endzustand

1	hospital	+ize <sub>V</sub>	+ation <sub>N</sub>	ε
---	----------	-------------------	---------------------	---

Zustand	Eingabe			
	NOM ADJ	+ize <sub>V</sub>	+ation <sub>N</sub>	+al <sub>Adj</sub>
1	2	0	0	0
<u>2</u>	0	3	0	0
<u>3</u>	0	0	4	0
<u>4</u>	0	0	0	2

# Deterministischer FSA-Erkennungsalgorithmus

```

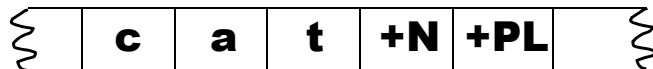
Funktion D-Erkennen(↓Band,↓FSA) = „accept“ oder „reject“
Index ← Bandanfang
AktualZustand ← Anfangszustand des FSA
LOOP
IF Ende der Eingabekette ist erreicht THEN
  IF AktualZustand ist ein Endzustand THEN return „accept“
  ELSE return „reject“
ELSE-IF Zustandstransitionstabelle[ AktualZustand, Band(Index) ] = 0
THEN
  return „reject“
ELSE AktualZustand ← Zustandstransitionstabelle[ AktualZustand,
Band(Index) ]
  Index ← Index + 1
LOOPEND
    
```

# Morphologisches Parsing

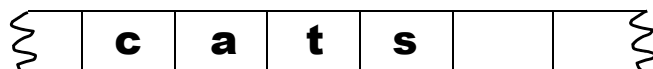
- **Lexikon**
  - Liste von Stämmen und Affixen sowie geeigneten morphologischen Merkmalen
- **Morphotaktik**
  - Modell der Ordnung von Morphemklassen in flektierten Wortformen
    - Beispiel: Pluralsuffix nach Stamm
- **Orthographieregeln**
  - Regeln zur Beschreibung von Kombinations-effekten bei Morphemen
    - Beispiel aus dem Englischen: y → ie

# Zwei-Ebenen-Morphologie

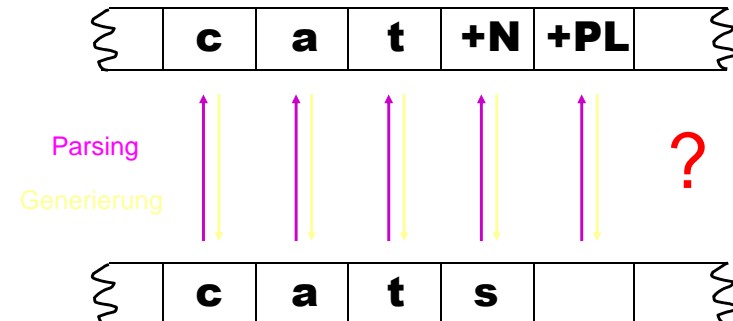
- **Lexikalische Ebene**
  - Wortrepräsentation als Konkatination von Morphemen



- **Oberflächenebene**
  - Wortrepräsentation als Abfolge von Buchstaben



# Zwei-Ebenen-Morphologie



## Formale Grundlagen von Automaten: endlicher Transduktor

- Ein (nicht-deterministischer) **endlicher Trans-duktor** ist ein 5-Tupel  $(Q, \Sigma, \delta, q_0, F)$  mit
  - $Q$ : endliche Menge von (Steuerungs-)Zuständen
  - $\Sigma \subseteq I \times O$ : endliches Alphabet mit komplexen Symbolen, die aus Eingabe-Ausgabe-Paaren  $i:o$  bestehen,  $i \in I$  (Eingabealphabet),  $o \in O$  (Ausgabealphabet), beide inkl.  $\epsilon$
  - $\delta$ : Zustandsübergangsfunktion  $\delta(q, i:o)$ , die das Steuerungsverhalten des FST determiniert
$$\delta : Q \times \Sigma \mapsto \wp(Q)$$
  - $q_0 \in Q$  : der ausgezeichnete Startzustand
  - $F \subseteq Q$  : Menge der ausgezeichneten Endzustände <sup>41</sup>

## Formale Grundlagen von Automaten: endlicher Transduktor

- Ein (nicht-deterministischer) **endlicher Trans-duktor** ist ein 6-Tupel  $(Q, \Sigma, \Delta, \delta, q_0, F)$  mit
  - $Q$ : endliche Menge von (Steuerungs-)Zuständen
  - $\Sigma$ : endliches Eingabealphabet (inkl.  $\epsilon$ )
  - $\Delta$ : **endliches Ausgabealphabet** (inkl.  $\epsilon$ )
  - $\delta$ : Zustandsübergangsfunktion, die das Steuerungsverhalten des FST determiniert
$$\delta : Q \times \Sigma \mapsto \wp(Q \times \Delta^*)$$
  - $q_0 \in Q$  : der ausgezeichnete Startzustand
  - $F \subseteq Q$  : Menge der ausgezeichneten Endzustände <sup>42</sup>

## Akzeptierte Sprache: FSA vs. FST

- Ein FSA akzeptiert eine Sprache, die über einem Alphabet *einzelner* Symbole spezifiziert ist
- Ein FST akzeptiert eine Sprache, die über *Paaren* von Symbolen spezifiziert ist. Diese werden auch *zulässige Paare* (*feasible pairs*) genannt.

## Konventionen zur Zwei-Ebenen-Morphologie

- Zeichenpaare für FSTs ( $\alpha:\gamma$ ) werden so kodiert, dass  $\alpha$  das Zeichen auf dem Lexikalischen Band und  $\gamma$  das auf dem Oberflächenband bezeichnet.
- Identische Zeichenpaare für FSTs ( $\alpha:\alpha$ ) heißen Default-Paare und werden auch kurz als  $\alpha$  notiert.
- @ (oder =) steht für ein beliebiges Symbol
- ^ ist das Morphembegrenzungssymbol
- # ist das Wortbegrenzungssymbol

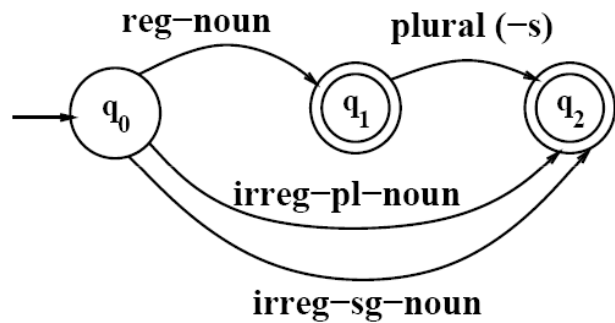
# Logische Sichten auf FSTs

- **FST als Erkenner bzw. Parser**
  - FST nimmt Paare von Zeichen als Eingabe und akzeptiert sie, wenn sie in der Sprache enthalten sind. Sonst: Zurückweisung
- **FST als Generator**
  - FST erzeugt Paare von Zeichen als Ausgabe, wenn sie in der Sprache enthalten sind. Sonst: keine Ausgabe

# Logische Sichten auf FSTs

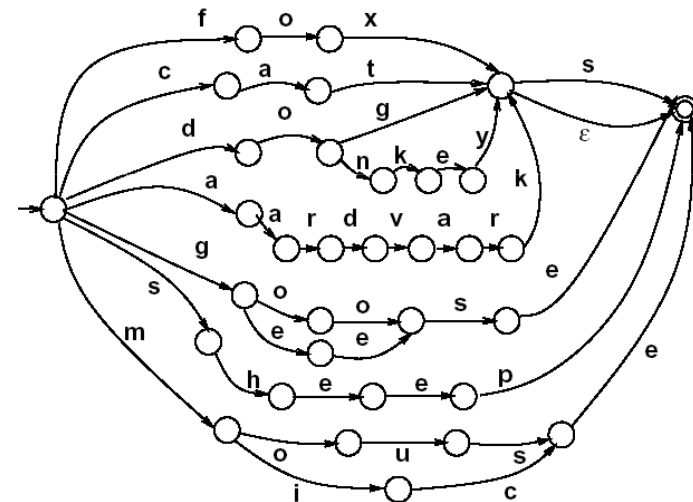
- **FST als Übersetzer**
  - FST liest Zeichenketten und gibt Zeichenketten aus
- **FST als Relator**
  - FST berechnet eine Relation zwischen Mengen von Zeichenketten

## FSA für Flexionsmorphologie: Englische Nomen

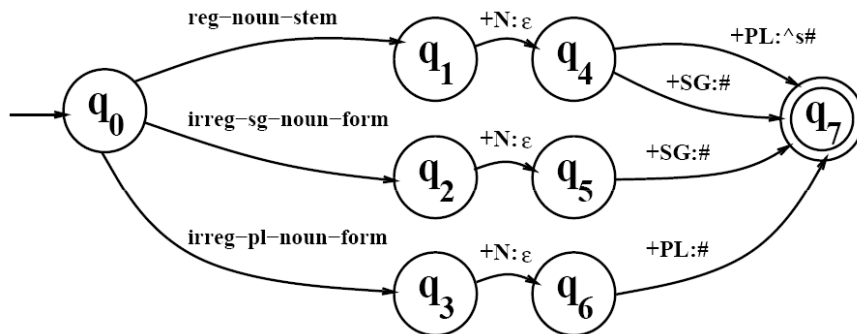


reg-noun	irreg-pl-noun	irreg-sg-noun	plural
aardvark	geese	goose	-s
cat	sheep	sheep	
dog	mice	mouse	
fox			

## FSAs als Erkenner mit integrierten Sublexika als TRIE



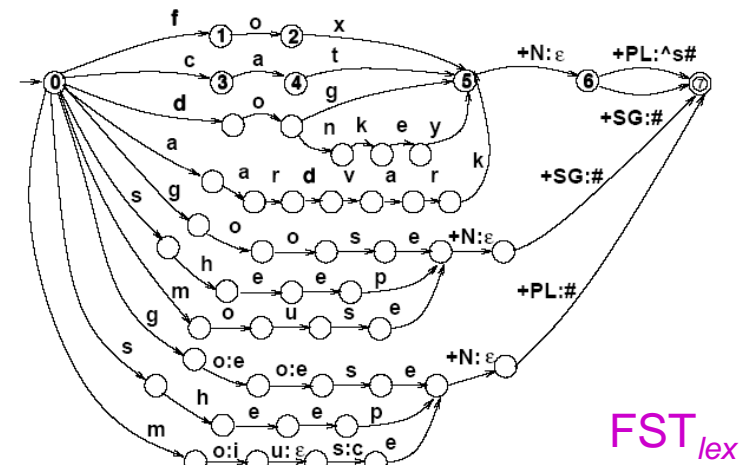
# FST für Flexionsmorphologie: Englische Nomen



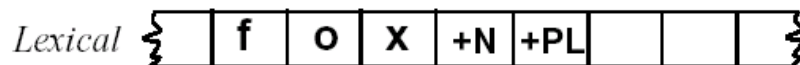
reg-noun	irreg-pl-noun	irreg-sg-noun
aardvark	g o:e o:e s e	goose
cat	sheep	sheep
dog	m o:i u:ε s:c e	mouse
fox		

49

# FSTs als Wort-Parser mit Fortsetzungslexikon als TRIE



# Lexikalische und Intermediäre Bänder



Problem:

Orthographische Regeln:

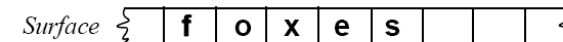
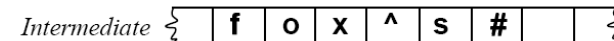
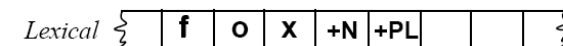
FST würde korrekte Form **foxes** zurückweisen, aber inkorrekte Form **foxs** akzeptieren

51

# Einige englische Orthographieregeln

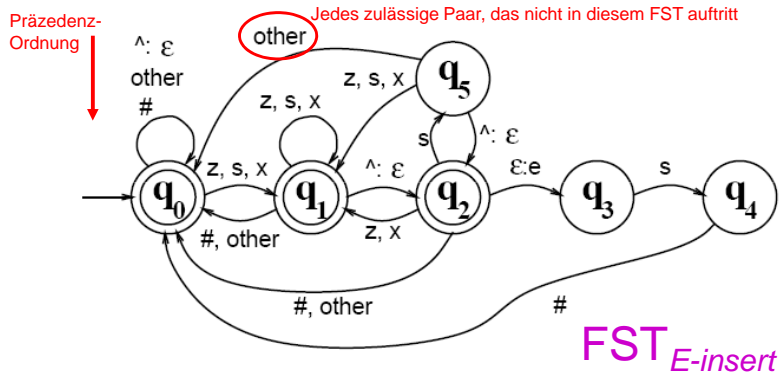
Für jede Regel  
einen Transduktor!

Name	Regelbeschreibung	Beispiel
Consonant Doubling	1-letter consonant doubled before <i>-ing/-ed</i>	beg/begging
E-deletion	Silent e dropped before <i>-ing</i> and <i>-ed</i>	make/making
E-insertion	e added after <i>s,z,x,ch,sh</i> before <i>s</i>	watch/watches
Y-replacement	-y changes to <i>-ie</i> before <i>-s</i> , to <i>-i</i> before <i>-ed</i>	try/tries, try/tried
K-insertion	verbs ending with vowel + <i>-c</i> add <i>-k</i>	panic/panicked



52

# Intermediär-Oberflächenband-Transduktor (E-Insertion)



„Füge ein „e“ auf dem Oberflächenband ein, wenn auf dem Lexikalischen Band ein Morphem auf „x“, „s“ oder „z“ endet und das nächste Morphem ein „s“ ist.“

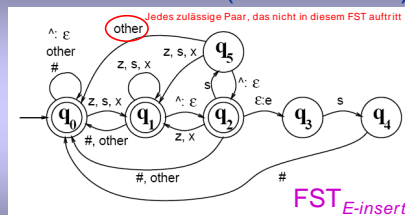
# Zustandstransitionstabelle für E-Insertion

State \ Input	s : s	x : x	z : z	^ : ε	ε : e	#	other
q0:	1	1	1	0	-	0	0
q1:	1	1	1	2	-	0	0
q2:	5	1	1	0	3	0	0
q3:	4	-	-	-	-	-	-
q4:	-	-	-	-	-	0	-
q5:	1	1	1	2	-	-	0

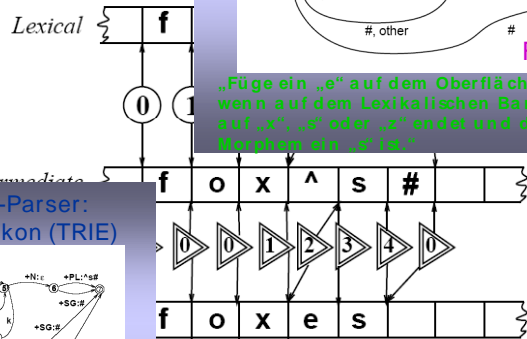
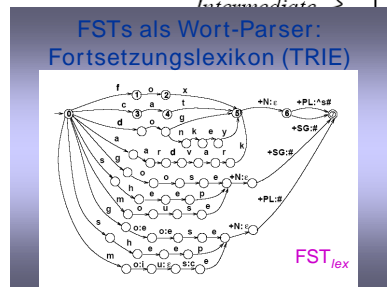
Explizite Markierung illegaler Transitionen

## FST-Akzeptanz

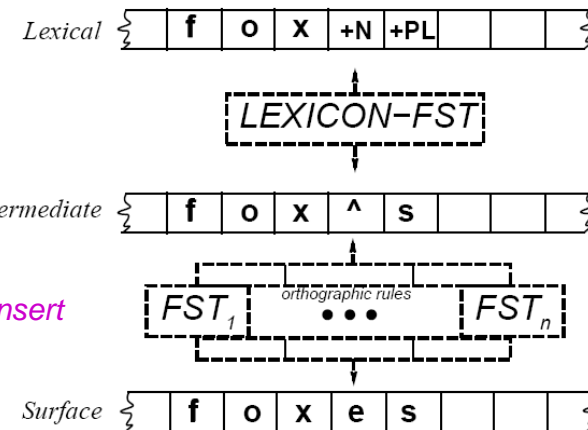
### Intermediär-Oberflächenband-Transduktor (E-Insertion)



„Füge ein „e“ auf dem Oberflächenband ein, wenn auf dem Lexikalischen Band ein Morphem auf „x“, „s“ oder „z“ endet und das nächste Morphem ein „s“ ist.“



# Zwei-Ebenen-Morphologie: Parser/Generator



## Fazit

- Zwei-Ebenen-Morphologie (ZEM) ist das dominierende Modell der automatischen morphologischen Analyse (falls eine umfassende, „tiefe“ morphologische Analyse nötig ist)
- ZEM-Systeme gibt es für eine Fülle von Sprachen, aber für das Deutsche ist die-ses Modell problematisch (nicht-konka-tenative Morphologie, z.B. Umlautung)
- ZEM-Systeme sind überschaubar groß