Words and transducers

Bognár Ivett

2020. Március 6.

Tartalom

Bevezetés

Mit szeretnénk?

Meg szeretnénk találni az összeset az összes állatból:

Nem csak annyit szeretnénk tudni, hogy a *woodchuck* többesszáma *woodchucks*, hanem felismerni az olyan többesszámokat is, mint:

fox, *foxes; peccary*, *peccaries; goose*, *geese*

Ehhez szükségünk van:

- Helyesírási szabályok (*peccary*, *peccaries*)
- Morfológiai szabályok

Voltaképpen morfolgóiai elemzőt szeretnénk.

Miért szeretnénk?

Mire jó a morfolgóiai elemzés?

- Kap egy felszíni/bemeneti formát *going*
  $\Rightarrow$ *VERB* $-$ *go* $+$ *GERUND* $-$ *ing*

- Automatikus keresés egy adott szó minden alakjára.
- POS tagging
- Helyesírás ellenőrzés
- Gépi fordítás
- A **produktivitás** miatt muszáj, hogy legyen morfológiai elemzőnk.

Véges állapotú morfológiai elemzés

Véges állapotú morfológiai elemzés

English		Spanish		
Input	**Morphological Parse**	**Input**	**Morphological Parse**	**Gloss**
cats	cat +N +PL	pavos	pavo +N +Masc +Pl	'ducks'
cat	cat +N +SG	pavo	pavo +N +Masc +Sg	'duck'
cities	city +N +Pl	bebo	beber +V +PInd +1P +Sg	'I drink'
geese	goose +N +Pl	canto	cantar +V +PInd +1P +Sg	'I sing'
goose	goose +N +Sg	canto	canto +N +Masc +Sg	'song'
goose	goose +V	puse	poner +V +Perf +1P +Sg	'I was able'
gooses	goose +V +1P +Sg	vino	venir +V +Perf +3P +Sg	'he/she came'
merging	merge +V +PresPart	vino	vino +N +Masc +Sg	'wine'
caught	catch +V +PastPart	lugar	lugar +N +Masc +Sg	'place'
caught	catch +V +Past			

Figure 3.2 Output of a morphological parse for some English and Spanish words. Spanish output modified from the Xerox XRCE finite-state language tools.

Véges állapotú morfológiai elemzés

Kell hozzá:

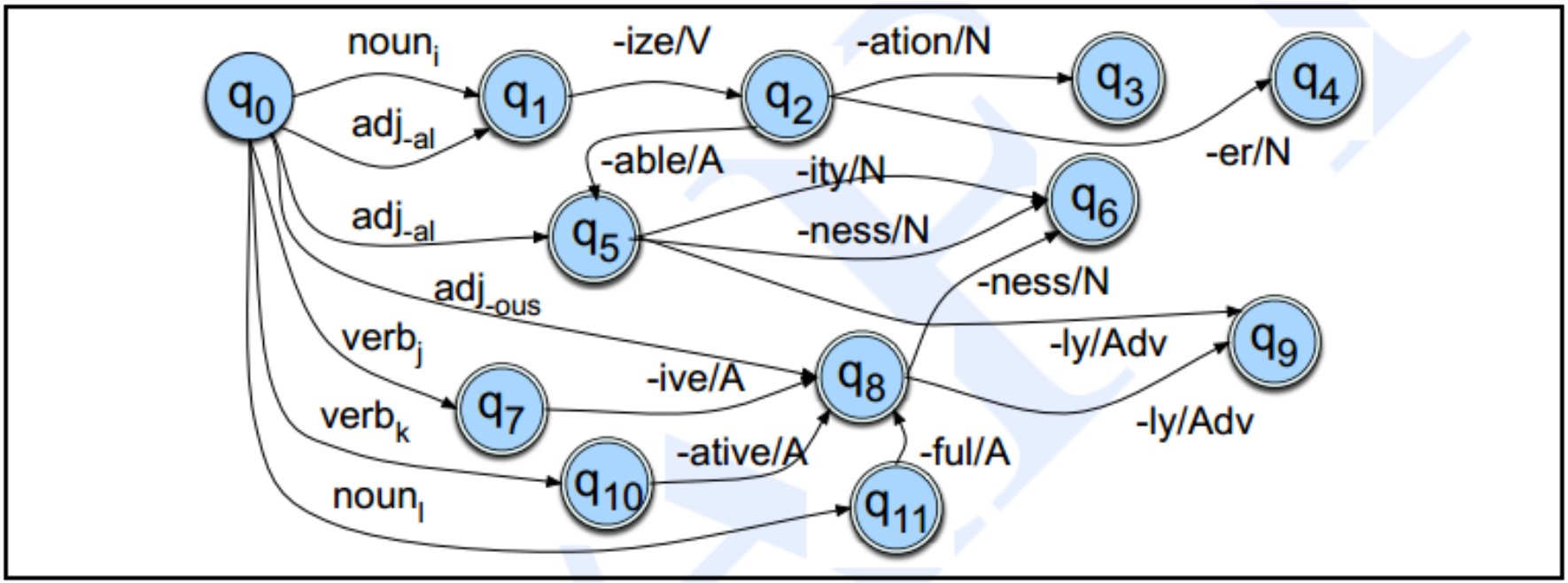
1. **lexikon**: tövek és affixumok listája, minimális információval (igei vagy főnévi tő, stb.)
2. **morfotaktika**
3. **helyesírás**

Véges állapotú lexikon

Véges állapotú lexikon

- Egy lexikonba nem tudunk minden szót felvenni.
- A lexikonok általában tövek és affixumok listája + morfotaktika, ami elmondja, hogyan kombinálhatjuk őket.
- Morfotaktika modellezésére: **véges állapotú automaták.**

Angol melléknevek



Véges állapotú transzducer

Véges állapotú transzducer

- "Finite-state transducer or FST is a type of finite automation which maps between two sets of symbols."
- Az FST relációt definiál két string-halmaz között.
- Vagyis: Olvas egy stringet és generál egy másikat.
  1. **Felismerőként**: vesz egy string párt inputként és outputként, **elfogadja**, ha a pár része a nyelvnek, és **elutasítja**, ha nem.
  2. **Generátor**: a nyelv string párjait adja vissza.
  3. **Fordító**: olvas egy stringet, és visszaad egy msikat.
  4. **Set relater**

FST

Q	a finite set of N states $q_0, q_1, \ldots, q_{N-1}$
Σ	a finite set corresponding to the input alphabet
Δ	a finite set corresponding to the output alphabet
$q_0 \in Q$	the start state
$F \subseteq Q$	the set of final states
$\delta(q, w)$	the transition function or transition matrix between states; Given a state $q \in Q$ and a string $w \in \Sigma^*$, $\delta(q, w)$ returns a set of new states $Q' \in Q$. δ is thus a function from $Q \times \Sigma^*$ to 2^Q (because there are 2^Q possible subsets of Q). δ returns a set of states rather than a single state because a given input may be ambiguous in which state it maps to.
$\sigma(q, w)$	the output function giving the set of possible output strings for each state and input. Given a state $q \in Q$ and a string $w \in \Sigma^*$, $\sigma(q, w)$ gives a set of output strings, each a string $o \in \Delta^*$. σ is thus a function from $Q \times \Sigma^*$ to 2^{Δ^*}

FST

Ahogy az FSA-k izomorfok a regurális nyelvekre, úgy az FST-k izomorfok a regurális relációkra.

A **reguláris relációk** olyan kifejezések, melyek két reguláris kifejezésből állnak, melyeket : választ el. Egy ilyen azt jelenti, hogy a kettőspont előtti kifejezésnek (a reláció felső tagjának) megfelelő nyelv minden eleme relációban áll a kettőspont után álló kifejezésnek (a reláció alsó tagjának) megfelelő nyelv minden elemével.

FST-k két fontos művelete:

- **inverzió**: megfordítja az input és az output címkéket.
- **kompozíció**

FST determinisztikusság

Míg az FSA-król elmondhattuk, hogy mindegyik átalakítható determinisztikussá, az FST-kről ez már nem mondható el.

- **Sequential transducers**: determinisztikusak az input tekintetében.
- **Subsequential transducers**: a sequential transducer általánosítása, generál további output stringet a végállapotnál, hozzákonkatenálva az eddig produkált outputhoz.

A szavakat úgy vesszük, hogy van egy lexikális szintjük (**lexical level**) és egy felszíni szintjük (**surface level**).

FST esetén ez azt jelenti, hogy van kettő tape:

- **lexical tape** from Σ
- **surface tape** from Δ

$\Sigma + \Delta = \Sigma'$: *komplexszimbólumokvégesábécéje*.

A szimbólumpárokat ebben megvalósítható pároknak (**feasible pairs**), illetve találhatunk benne párokat, mint pl. *a:a*, amit **default párnak** hívunk.

Angol többesszám

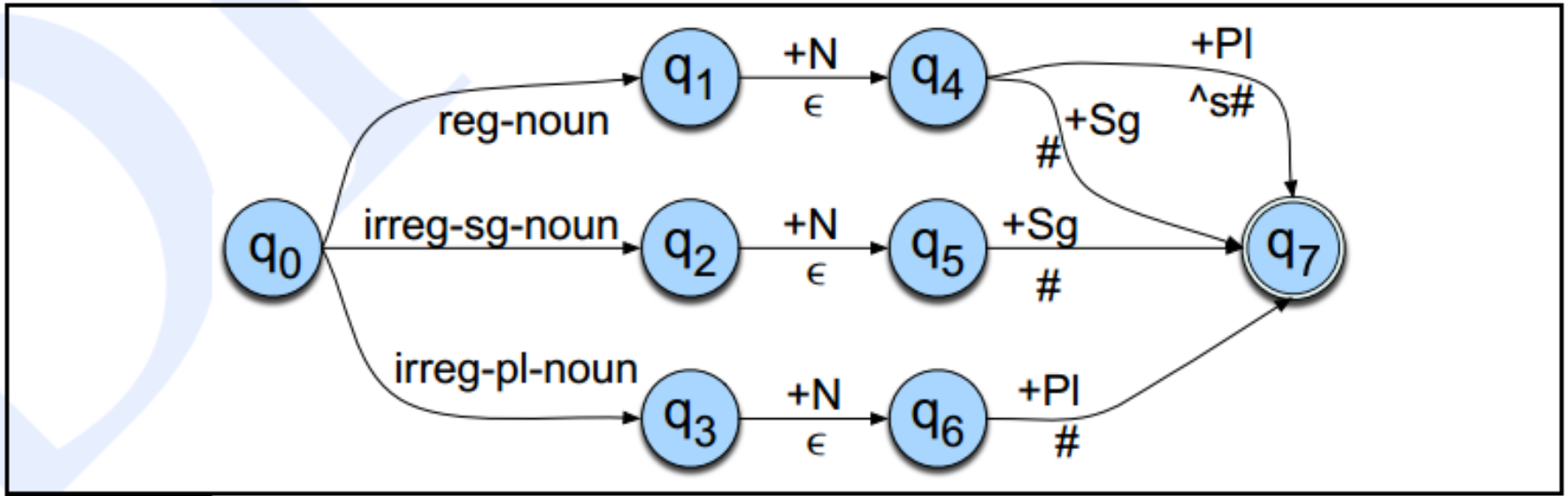
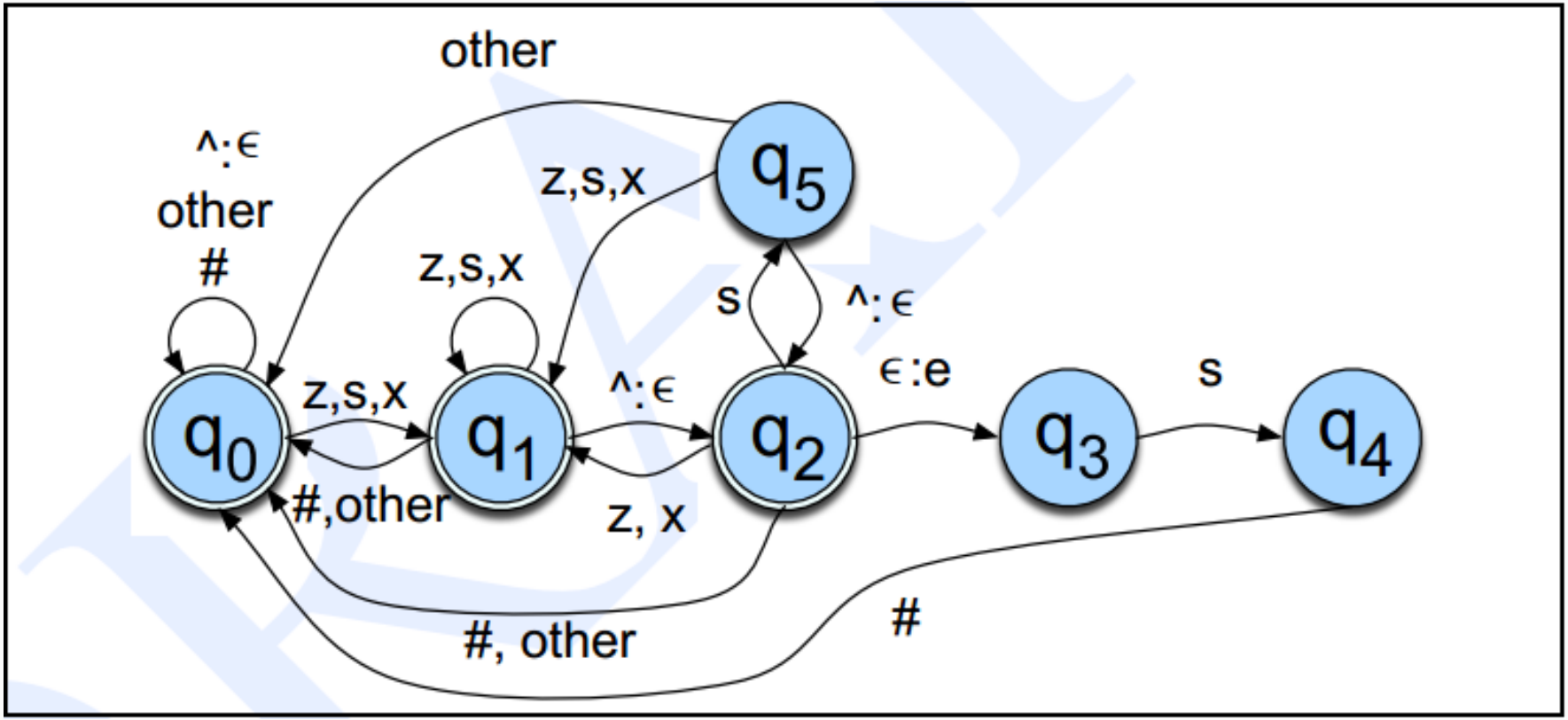


Figure 3.13 A schematic transducer for English nominal number inflection T_{num}. The sym-

Angol többesszám - jobban



Lexikon + szabályok

Rókák generálása

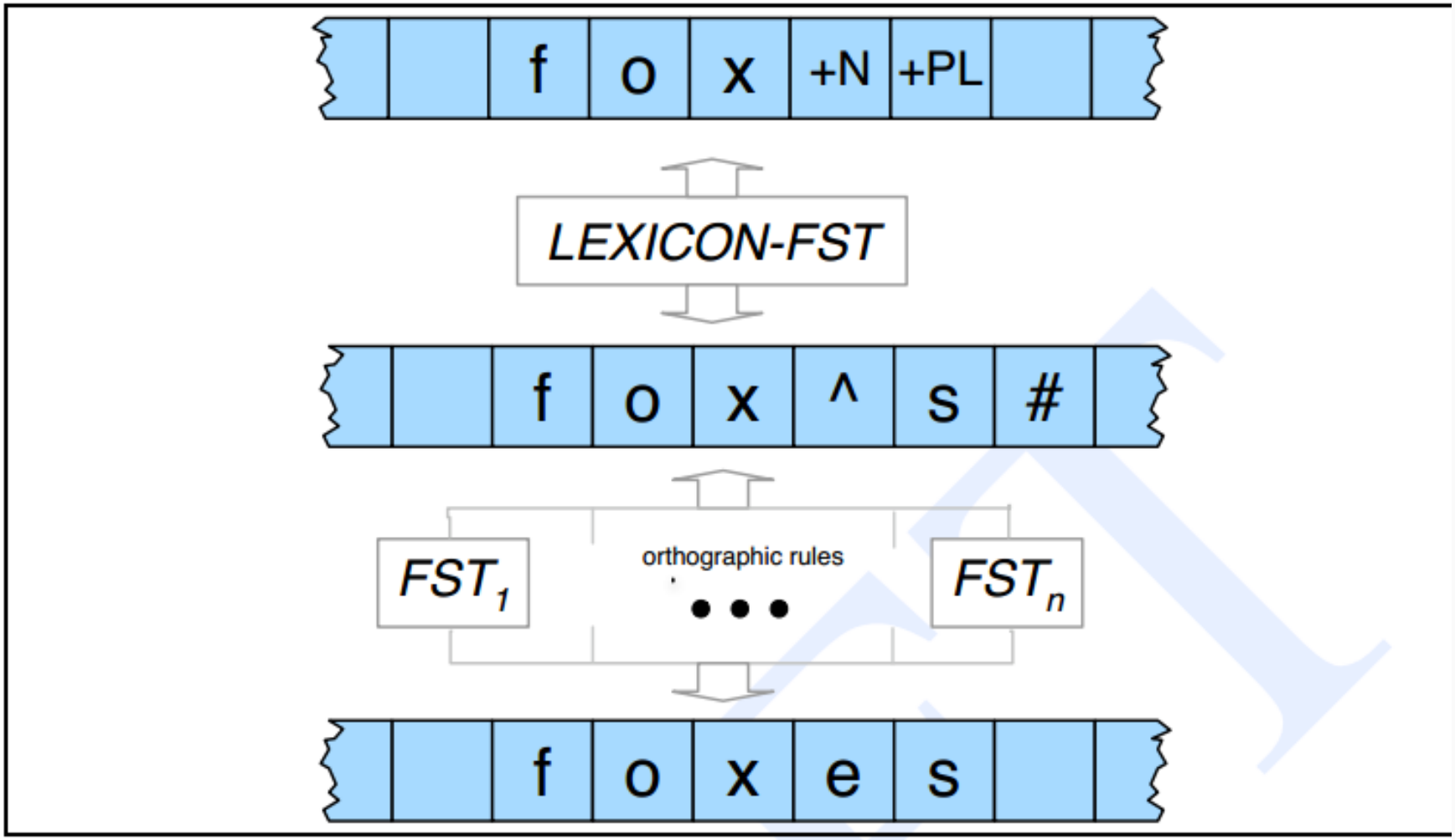


Figure 3.19 Generating or parsing with FST lexicon and rules

Rókák elfogadása

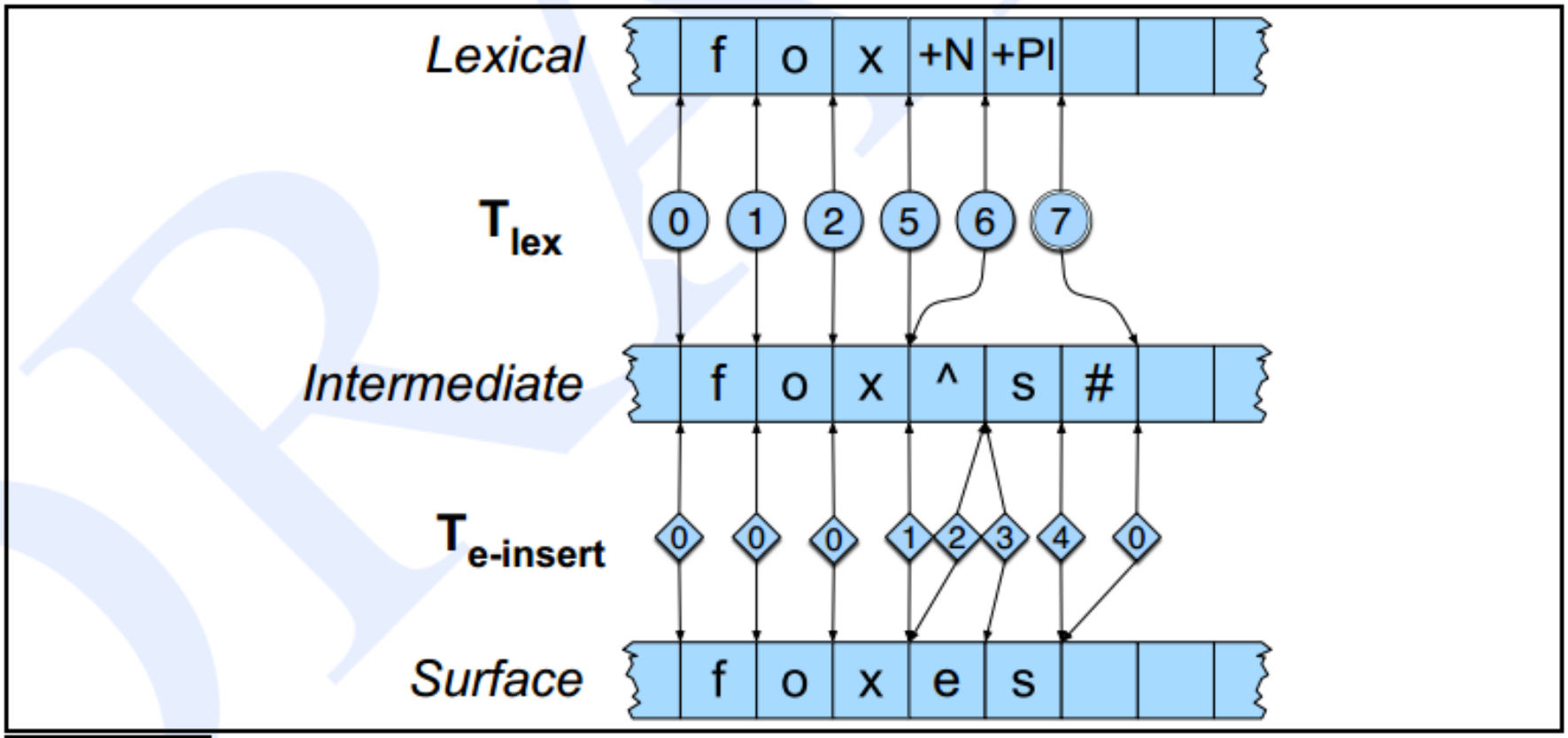


Figure 3.20 Accepting *foxes*: The lexicon transducer T_{lex} from Fig. 3.14 cascaded with the E-insertion transducer in Fig. 3.17.

Homályos rókák

- A *foxes* lehet ige is.
- A transducer ilyen esetekben nem képes dönteni, szükség van a szövegkörnyezet ismeretére is. (*That trickster foxes me every time!*)
  A legjobb, amit tehetünk, hogy felsoroljuk az összes lehetőséget.
  Amit viszont kezelnünk kell: local ambiguity.
  "Transducers in parallel can be combined by **automaton intersection**."

Porter Stemmer

Porter Stemmer

- lemmatizálás
  stemming / (szó)tövesítés vs. lemmatizálás A **lemma** mindig értelmes szóalak, a **szótő** pedig olykor nem.
- tokenizáció – példányosítás

Porter Stemmer (1980) Egy egyszerű és viszonylag hatékony algoritmus tövesítésre.

Porter szabályai és hibái

Szabályok:

ATIONAL → ATE (e.g., relational → relate)

ING → ϵ if stem contains vowel (e.g., motoring → motor)

SSES → SS (e.g., grasses → grass)

Hibák:

Errors of Commission		Errors of Omission	
organization	organ	European	Europe
doing	doe	analysis	analyzes
numerical	numerous	noise	noisy
policy	police	sparse	sparsity

Tokenizáció

Tokenizáció

Azt hinnénk, hogy csak levágjuk a whitespace-t és megvagyunk, de azért ennyire nem egyszerű.

- A tokenizálók általában külön szedik a többszavas kifejezéseket. (*New York*)
  A kérdéskör kicsit a névfelismerésé is.
- Az írásjelek nem mindig egyértelműek: *Mr.*, *Inc.*, stb.
- A mondat szegmentállása lesz az első lépés!
  Szabály alapon, vagy machine learninggel.
- Egyszerű, de nagyszerű tokenizálót lehet csinálni reguláris kifejezésekkel.

Helyesírási hibák

Helyesírási hibák

Miért szeretjük felismerni őket?

- Keresésekhez
- Kézírás felismerőkhoz

Milyen problémákat kell kezelni?

- **non-word error detection**
  Olyan helyesírási hibák felismerése, amik nem létező szóhoz vezetnek, pl. *graffe, giraffe* helyett
- **isolated-word error correction**
  Kijavítjuk a zsiráfot, de csak önmagában nézzük.
- **context-dependent error detection and correction**
  A kontextust is használjuk, hogy felismerjünk egy hibát, még akkor is, ha az véletlenül éppen egy valódi szó. (**real-word error**) Pl. *desert*, *dessert* helyett.

Minimum Edit Distance

Két string távolsága

Két string távolsága az a szám, ami kifejezi, hogy mennyire hasonló két string egymáshoz.

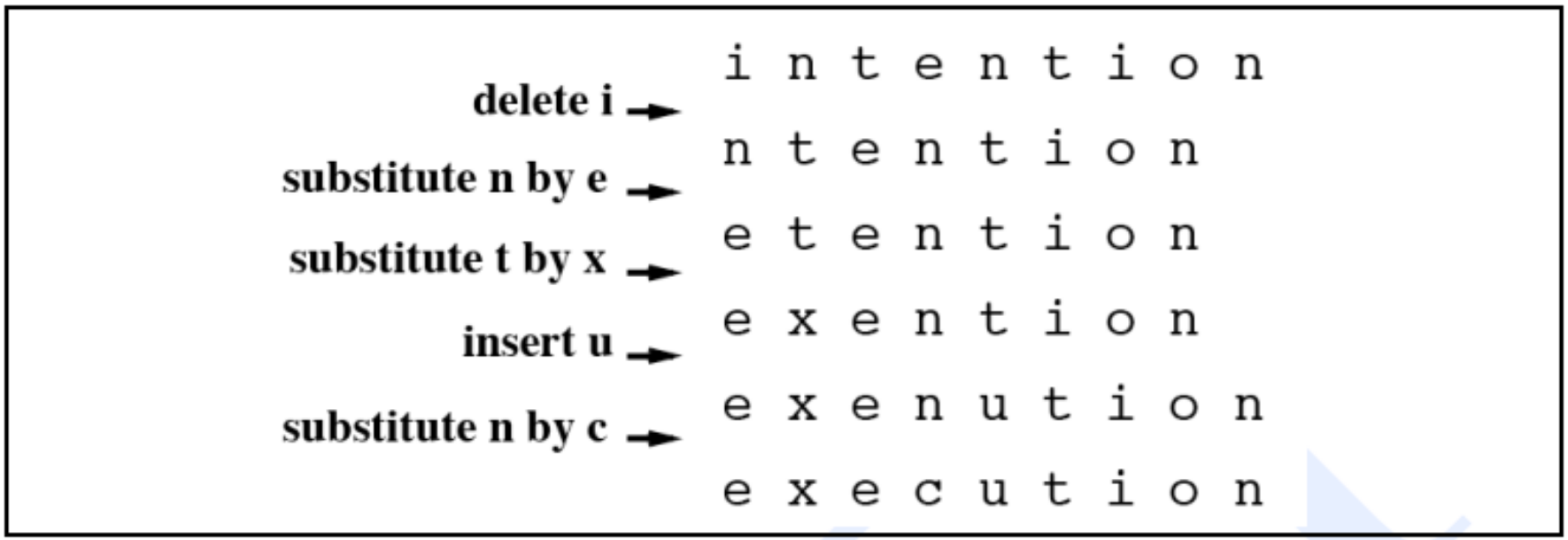


Figure 3.24 Operation list transforming *intention* to *execution* (after Kruskal 1983)

$$distance[i,j] = \min \begin{cases} distance[i-1,j] + \text{ins-cost}(target_{i-1}) \\ distance[i-1,j-1] + \text{sub-cost}(source_{j-1}, target_{i-1}) \\ distance[i,j-1] + \text{del-cost}(source_{j-1})) \end{cases} \tag{3.5}$$

Levenshtein távolság

http://www.let.rug.nl/ kleiweg/lev/

n	*9*	8	9	10	11	12	11	10	9	8
o	*8*	7	8	9	10	11	10	9	8	9
i	*7*	6	7	8	9	10	9	8	9	10
t	*6*	5	6	7	8	9	8	9	10	11
n	*5*	4	5	6	7	8	9	10	11	10
e	*4*	3	4	5	6	7	8	9	10	9
t	*3*	4	5	6	7	8	7	8	9	8
n	*2*	3	4	5	6	7	8	7	8	7
i	*1*	2	3	4	5	6	7	6	7	8
#	*0*	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*
	#	**e**	**x**	**e**	**c**	**u**	**t**	**i**	**o**	**n**

Figure 3.26 Computation of minimum edit distance between *intention* and *execution* via algorithm of Fig. 3.25, using Levenshtein distance with cost of 1 for insertions or deletions, 2 for substitutions. In italics are the initial values representing the distance from the empty string.

Köszönöm a figyelmet!