

# Language Technology

<http://cs.lth.se/edan20/>

## Chapter 6: Words, Parts of Speech, and Morphology

Pierre Nugues

Pierre.Nugues@cs.lth.se  
[http://cs.lth.se/pierre\\_nugues/](http://cs.lth.se/pierre_nugues/)

September 13, 2021



# The Parts of Speech

The parts of speech (POS) are classes that correspond to the lexical – or word – categories

Plato made a distinction between the verb and the noun.

After him, the word categories further evolved and grew in number until Dionysus Thrax formulated and fixed them.

Aelius Donatus popularized the list of the eight parts of speech: noun, pronoun, verb, participle, conjunction, adverb, preposition, and interjection.

Grammarians have adopted these POS for most European languages although they are somewhat arbitrary

POS divide between two main classes: the open class and the closed class



# Parts of Speech: Open Class Words

POS	English	French	German
Nouns	<i>name, Frank</i>	<i>nom, François</i>	<i>Name, Franz</i>
Adjectives	<i>big, good</i>	<i>grand, bon</i>	<i>groß, gut</i>
Verbs	<i>to swim</i>	<i>nager</i>	<i>schwimmen</i>
Adverbs	<i>rather, very, only</i>	<i>plutôt, très, uniquement</i>	<i>fast, nur, sehr, endlich</i>



# Parts of Speech: Closed Class Words

POS	English	French	German
Determiners	<i>the, several, my</i>	<i>le, plusieurs, mon</i>	<i>der, mehrere, mein</i>
Pronouns	<i>he, she, it</i>	<i>il, elle, lui</i>	<i>er, sie, ihm</i>
Prepositions	<i>to, of</i>	<i>vers, de</i>	<i>nach, von</i>
Conjunctions	<i>and, or</i>	<i>et, ou</i>	<i>und, oder</i>
Auxiliaries and modals	<i>be, have, will, would</i>	<i>être, avoir, pouvoir</i>	<i>sein, haben, können</i>



# Part-of-Speech Annotation (CoNLL 2000)

Annotation of: *He reckons the current account deficit will narrow to only # 1.8 billion in September.* We set aside the last column for now.

He	PRP	B-NP
reckons	VBZ	B-VP
the	DT	B-NP
current	JJ	I-NP
account	NN	I-NP
deficit	NN	I-NP
will	MD	B-VP
narrow	VB	I-VP
to	TO	B-PP
only	RB	B-NP
#	#	I-NP
1.8	CD	I-NP
billion	CD	I-NP
in	IN	B-PP
September	NNP	B-NP
.	.	O



# The CoNLL Format (2006)

Annotation of: *La reestructuración de los otros bancos checos se está acompañando por la reducción del personal* 'The restructuring of Czech banks is accompanied by the reduction of personnel'.

ID	FORM	LEMMA	CPOS	POS	FEATS
1	La	el	d	da	num=s gen=f
2	reestructuración	reestructuración	n	nc	num=s gen=f
3	de	de	s	sp	for=s
4	los	el	d	da	gen=m num=p
5	otros	otro	d	di	gen=m num=p
6	bancos	banco	n	nc	gen=m num=p
7	checos	checo	a	aq	gen=m num=p
8	se	se	p	p0	—
9	está	estar	v	vm	num=s per=3 mod=i tmp=p
10	acompañando	acompañar	v	vm	mod=g
11	por	por	s	sp	for=s
12	la	el	d	da	num=s gen=f
13	reducción	reducción	n	nc	num=s gen=f
14	del	del	s	sp	gen=m num=s for=s
15	personal	personal	n	nc	gen=m num=s
16	.	.	F	Fp	—



# Part-of-Speech Annotation (Universal Dependencies)

Annotation of: *Do museum labels have an impact on how people look at artworks?*

ID	FORM	LEMMA	UPOS	XPOS	FEATS
1	Do	do	AUX	VBP	Mood=Ind Number=Plur Person=3 Tense=Pres VerbForm=Fin
2	museum	museum	NOUN	NN	Number=Sing
3	labels	label	NOUN	NNS	Number=Plur
4	have	have	VERB	VB	VerbForm=Inf
5	an	a	DET	DT	Definite=Ind PronType=Art
6	impact	impact	NOUN	NN	Number=Sing
7	on	on	ADP	IN	
8	how	how	SCONJ	WRB	PronType=Rel
9	people	person	NOUN	NNS	Number=Plur
10	look	look	VERB	VBP	Mood=Ind Number=Plur Person=3 Tense=Pres VerbForm=Fin
11	at	at	ADP	IN	
12	artworks	artwork	NOUN	NNS	Number=Plur
13	?	?	PUNCT	.	



# Part-of-Speech Annotation (Universal Dependencies)

Annotation of: *Genom skattereformen införs individuell beskattning (särbeskattning) av arbetsinkomster.*

ID	FORM	LEMMA	UPOS	XPOS	FEATS
1	Genom	genom	ADP	PP	
2	skattereformen	skattereform	NOUN	NN UTR SIN DEF NOM	Case=Nom Definite=Def Gender=Com Number=Sing
3	införs	införa	VERB	VB PRS SFO	Mood=Ind Tense=Pres VerbForm=Fin Voice=Pass
4	individuell	individuell	ADJ	JJ POS UTR SIN IND NOM	Case=Nom Definite=Ind Degree=Pos Gender=Com Number=Sing
5	beskattning	beskattning	NOUN	NN UTR SIN IND NOM	Case=Nom Definite=Ind Gender=Com Number=Sing
6	(	(	PUNCT	PAD	
7	särbeskattning	särbeskattning	NOUN	NN UTR SIN IND NOM	Case=Nom Definite=Ind Gender=Com Number=Sing
8	)	)	PUNCT	PAD	
9	av	av	ADP	PP	
10	arbetsinkomster	arbetsinkomst	NOUN	NN UTR PLU IND NOM	Case=Nom Definite=Ind Gender=Com Number=Plur
11	.	.	PUNCT	MAD	





# Features

Main parts of speech	Features (subcategories)
Adjective, noun, pronoun	Regular base comparative superlative interrogative person number case
Adverb	Regular base comparative superlative interrogative
Article, determiner, preposition	Person case number
Verb	Tense voice mood person number case

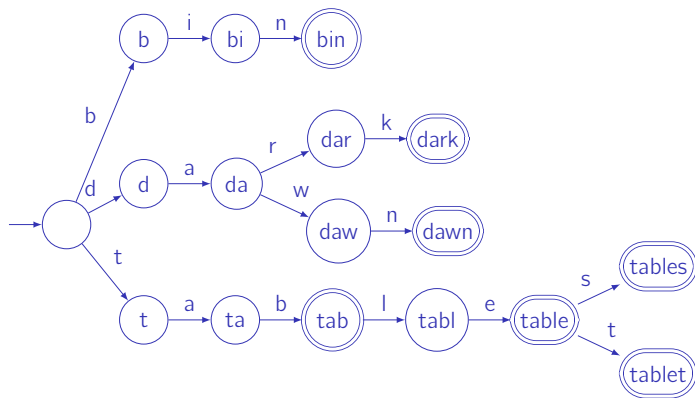


# Lexicons: An Excerpt from the Oxford Advanced Learner's Dictionary

Word	Pronunciation	Syntactic tag	Syllable count or verb pattern (for verbs)
a	@	S-*	1
a	EI	Ki\$	1
a fortiori	el ,fOtI'Oral	Pu\$	5
a posteriori	el ,p0sterI'Oral	OA\$,Pu\$	6
a priori	el ,pral'Oral	OA\$, Pu\$	4
a's	Eiz	Kj\$	1
ab initio	&b I'nISl@U	Pu\$	5
abaci	'&b@sal	Kj\$	3
aback	@'b&k	Pu%	2
abacus	'&b@k@s	K7%	3
abacuses	'&b@k@slz	Kj%	4
abaft	@'bAft	Pu\$,T-\$	2
abandon	@'b&nd@n	H0%,L@%	36A,14
abandoned	@'b&nd@nd	Hc% Hd% OA% □	36A,14



# Letter Trees



# Letter Trees in Prolog

```
[  
  [b, [i, [n, bin]]]  
  [d, [a, [r, [k, dark]],  
        [w, [n, dawn]]]]  
  [t, [a, [b, tab,  
            [l, [e, table,  
                  [s, tables],  
                  [t, tablet]]]]]]]]  
]
```



# Finding a Word in a Trie

```
% Checks if a word is in a trie
% is_word_in_trie(+WordChars, +Trie, -Lex)
is_word_in_trie([H | T], Trie, Lex) :-
    member([H | Branches], Trie),
    is_word_in_trie(T, Branches, Lex).
is_word_in_trie([], Trie, LexList) :-
    findall(Lex, (member(Lex, Trie), atom(Lex)), LexList),
    LexList \= [].
% We assume that the word lexical entry is an atom
```



# Morphemes

	Word	Morpheme decomposition
English	<i>disentangling</i>	<u>dis</u> + <u>en</u> + <b>tangle</b> + <u>ing</u>
	<i>rewritten</i>	<u>re</u> + <b>write</b> + <u>en</u>
French	<i>désembrouillé</i>	<u>dé</u> + <u>em</u> + <b>brouiller</b> + <u>é</u>
	<i>récite</i>	<u>re</u> + <b>écrire</b> + <u>te</u>
German	<i>entwirrend</i>	<u>ent</u> + <b>wirren</b> + <u>end</u>
	<i>wiedergeschrieben</i>	<u>wieder</u> + <u>ge</u> + <b>schreiben</b> + <u>en</u>



# Inflection

	<b>Plural of nouns</b>	<b>Morpheme decomposition</b>
English	<i>hedgehogs</i>	<i>hedgehog+s</i>
	<i>churches</i>	<i>church+es</i>
	<i>sheep</i>	<i>sheep+∅</i>
French	<i>hérissons</i>	<i>hérisson+s</i>
	<i>chevaux</i>	<i>cheval+ux</i>
German	<i>Gründe</i>	<i>Grund+(¨)e</i>
	<i>Hände</i>	<i>Hand+(¨)e</i>
	<i>Igel</i>	<i>Igel+∅</i>



# Derivation

## Creation of a new word

	English	French	German
Prefixes	<b>fore</b> see, <b>un</b> pleasant	<b>pré</b> voir, <b>dé</b> plaisant	<b>vor</b> hersehen, <b>un</b> angenehm
Suffixes	manage <b>able</b> , rigor <b>ous</b>	gér <b>able</b> , rigour <b>eux</b>	vorsicht <b>ich</b> , streit <b>bar</b>





# Morphological Processing

Generation →

English		French		German	
<i>dog+s</i>	<i>dogs</i>	<i>chien+s</i>	<i>chiens</i>	<i>Hund+e</i>	<i>Hunde</i>
<i>work+ing</i>	<i>working</i>	<i>travailler+ant</i>	<i>travaillant</i>	<i>arbeiten+end</i>	<i>arbeitend</i>
<i>un+do</i>	<i>undo</i>	<i>dé+faire</i>	<i>défaire</i>		

← Parsing



# Language Differences (Source: Xerox)

Language	# stems	# inflected forms	Lex. size (kb)
English	55,000	240,000	200–300
French	50,000	5,700,000	200–300
German	50,000	350,000 or infinite (compounding)	450
Japanese	130,000	200 suffixes	500
		20,000,000 word forms	500
Spanish	40,000	3,000,000	200–300



# Ambiguities

	Words	Words in context	Lemmatization
E	<i>Run</i>	<ol style="list-style-type: none"> <li>1 A <b>run</b> in the forest</li> <li>2 Sportsmen <b>run</b> everyday</li> </ol>	<ol style="list-style-type: none"> <li>1 <b>run</b>: noun sing.</li> <li>2 <b>run</b>: verb present third pers. pl.</li> </ol>
F	<i>Marche</i>	<ol style="list-style-type: none"> <li>1 Une <b>marche</b> dans la forêt</li> <li>2 Il <b>marche</b> dans la cour</li> </ol>	<ol style="list-style-type: none"> <li>1 <b>marche</b>: noun sing. fem.</li> <li>2 <b>marcher</b>: verb present third pers. sing.</li> </ol>
G	<i>Lauf</i>	<ol style="list-style-type: none"> <li>1 Der <b>Lauf</b> der Zeit</li> <li>2 <b>Lauf</b> schnell!</li> </ol>	<ol style="list-style-type: none"> <li>1 <b>Der Lauf</b>: noun sing, masc</li> <li>2 <b>laufen</b>: verb, imp. sing.</li> </ol>



# Two-Level Morphology

Current morphological parsers are based on the two-level model of Kimmo Koskenniemi (1983).

It links the surface form of a word – the word as it is in a text – to its lexical or underlying form – its sequence of morphemes

<b>Surface:</b>	disentangled
<b>Lexical (or underlying):</b>	dis+en+tangle+ed



# Examples

---

## Generation: Lexical to surface form →

---

English	<i>dis+en+tangle+ed</i>	<i>disentangled</i>
	<i>happy+er</i>	<i>happier</i>
	<i>move+ed</i>	<i>moved</i>
French	<i>dés+em+brouiller+é</i>	<i>désembrouillé</i>
	<i>dé+chanter+erons</i>	<i>déchanterons</i>
German	<i>ent+wirren+end</i>	<i>entwirrend</i>
	<i>wieder+ge+schreiben+en</i>	<i>wiedergeschrieben</i>

---

## Parsing: ← Surface to lexical form

---



# Aligning the Two Forms

English	dis+en+tangle+ed ⇕ ... dis0en0tangl00ed	happy+er ⇕ ... happi0er	move+ed ⇕ ... mov00ed
French	dé+chanter+erons ⇕ ... dé0chant000erons	cheval+ux ⇕ ... cheva00ux	cheviller+é ⇕ ... chevill000é
German	singen+st ⇕ ... sing00st	Grund+`e ⇕ ... Gründ00e	Igel+Ø ⇕ ... Igel00



# Interpreting the Morphemes

Suffixes have a grammatical interpretation: *erons* in a French verb corresponds to verb + future + 1st person + plural

Morphological parsers can represent the lexical form as a concatenation of the stem and its features instead of the stem and the suffix.

The Xerox parser output for *disentangled* and *happier* is:

disentangle+Verb+PastBoth+123SP

happy+Adj+Comp

where +Verb denotes a verb, +PastBoth, either past tense or past participle, and +123SP any person, singular or plural; +Adj denotes an adjective and +Comp, a comparative.



# Aligning Morphemes and Features

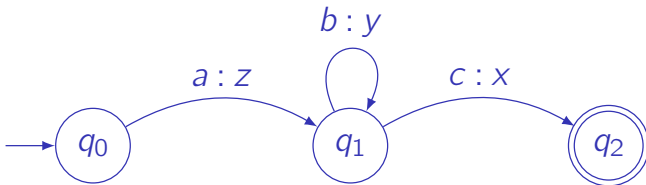
Lexical:	d	i	s	e	n	t	a	n	g	l	e	+Verb	+PastBoth	+123sp
	↕	↕	↕	↕	↕	↕	↕	↕	↕	↕	↕	↕	↕	↕
Surface:	d	i	s	e	n	t	a	n	g	l	0	0	e	d

Lexical:	h	a	p	p	y	+Adj	+Comp
	↕	↕	↕	↕	↕	↕	↕
Surface:	h	a	p	p	i	e	r





# Transducers



The string *abbbc* is transduced into *zyyyx*



# Mathematical Definition of a FST

- ①  $Q$  is a finite set of states.
- ②  $\Sigma$  is a finite set of symbol or character pairs  $i : o$ , where  $i$  is a symbol of the input alphabet and  $o$  of the output alphabet. As we saw, both alphabets may include epsilon transitions.
- ③  $q_0$  is the start state,  $q_0 \in Q$ .
- ④  $F$  is the set of final states,  $F \subseteq Q$ .
- ⑤  $\delta$  is the transition function  $Q \times \Sigma \rightarrow Q$ , where  $\delta(q, i, o)$  returns the state where the automaton moves when it is in state  $q$  and consumes the input symbol pair  $i : o$ .

The quintuple defining automaton is  $Q = \{q_0, q_1, q_2\}$ ,

$\Sigma = \{a : z, b : y, c : x\}$ ,

$\delta = \{\delta(q_0, a : z) = q_1, \delta(q_1, b : y) = q_1, \delta(q_1, c : x) = q_2\}$ , and  $F = \{q_0\}$ .



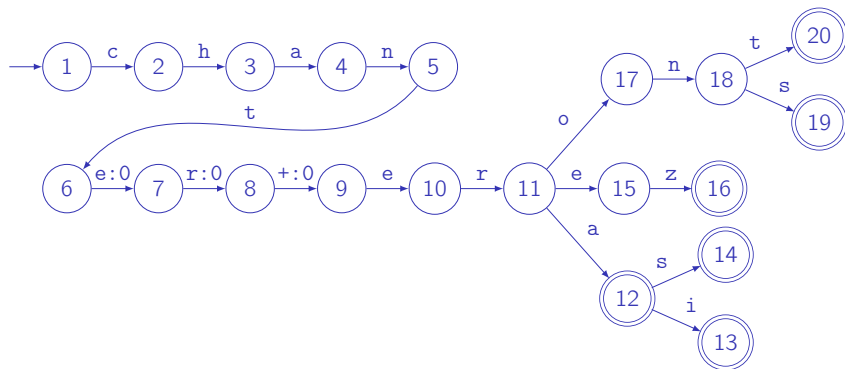
# French Verb Transducers for *chanter*

Number\Person	First	Second	Third
singular	<i>chanterai</i>	<i>chanteras</i>	<i>chantera</i>
plural	<i>chanterons</i>	<i>chanterez</i>	<i>chanteront</i>

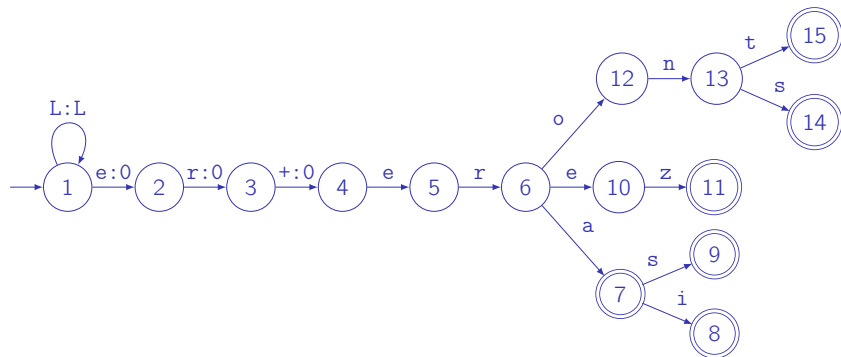
Number\Pers.	First	Second	Third
singular	chanter+erai	chanter+eras	chanter+era
	chant000erai	chant000eras	chant000era
plural	chanter+erons	chanter+erez	chanter+eront
	chant000erons	chant000erez	chant000eront



# Transducer for *chanter*



# French Verb Transducers: Future, 1st Group




# Transducers in Prolog

```

arc(1,1,C,C) :- letter(C).
arc(1,2,e,0).    arc(6,7,a,a).    arc(6,12,o,o).
arc(2,3,r,0).    arc(7,8,i,i).    arc(12,13,n,n).
arc(3,4,+,0).    arc(7,9,s,s).    arc(13,14,s,s).
arc(4,5,e,e).    arc(6,10,e,e).    arc(13,15, t, t).
arc(5,6,r,r).    arc(10,11,z,z).

final_state(7).    final_state(9).    final_state(14).
final_state(8).    final_state(11).    final_state(15).

% letter(+L) describes the French lower-case letters
letter(L) :- atom_codes(L, [Code]), 97 =< Code, Code =< 122, !
letter(L) :-
    member(L, [à, â, ä, ç, é, è, ê, ë, î, ï, ô, ö, ù,
                
                u, i, o]).

```

# Running the Transducer

```
transduce(+Start, ?Final, ?Underlying, ?Surface).  
% arc(Start, End, UnderlyingChar, SurfaceChar) describes the automa  
  
% transduce(+Start, ?Final, ?UnderlyingString, ?SurfaceString)  
transduce(Start, Final, [U | UnderlyingString], SurfaceString) :-  
    arc(Start, Next, U, 0),  
    transduce(Next, Final, UnderlyingString, SurfaceString).  
transduce(Start, Final, UnderlyingString, [S | SurfaceString]) :-  
    arc(Start, Next, 0, S),  
    transduce(Next, Final, UnderlyingString, SurfaceString).  
transduce(Start, Final, [U | UnderlyingString],  
    [S | SurfaceString]) :-  
    arc(Start, Next, U, S),  
    U \== 0, S \== 0,  
    transduce(Next, Final, UnderlyingString, SurfaceString).  
transduce(Final, Final, [], []) :- final_state(Final).
```



# Transducers with OpenFst

OpenFst is a library to create and process transducers. We encode the lexical and surface forms of the conjugation as:

1 1 a a	10 11 z z
1 1 b b	6 12 o o
1 1 c c	12 13 n n
...	13 14 s s
1 2 e <epsilon>	13 15 t t
2 3 r <epsilon>	7
3 4 + <epsilon>	8
4 5 e e	9
5 6 r r	11
6 7 a a	14
7 8 i i	15
7 9 s s	
6 10 e e	

that we store the `first_group_future.fst` file.





# Transducers with OpenFst (II)

We encode rêver+era as a single chain automaton:

```
0 1 r
1 2 ê
2 3 v
3 4 e
4 5 r
5 6 +
6 7 e
7 8 r
8 9 a
9
```

```
$ fstcompile --isymbols=symbols.txt --osymbols=symbols.txt \
  first_group_future.fst first_group_future.bin
$ fstcompile --isymbols=symbols.txt --acceptor \
  rêver+era.fst rêver+era.bin
```



where the `symbols.txt` file contains all the letters of the alphabet and

# Transducers with OpenFst (III)

We generate the surface form by composing the input with the transducer:

```
$ fstcompose rêver+era.bin first_group_future.bin | \  
  fstprint --isymbols=symbols.txt --osymbols=symbols.txt  
0 1 r r  
1 2 ê ê  
2 3 v v  
3 4 e <epsilon>  
4 5 r <epsilon>  
5 6 + <epsilon>  
6 7 e e  
7 8 r r  
8 9 a a  
9
```



# Transducers with OpenFst (IV)

To remove the  $\epsilon$ , we need to project the results using the `fstproject` command that restricts a transducer to an acceptor with only the output and we and apply the `fstrmepsilon` command:

```
$ fstcompose rêver+era.bin first_group_future.bin | \  
  fstproject --project_output | fstrmepsilon | \  
  fstprint --isymbols=symbols.txt --osymbols=symbols.txt
```

```
0 1 r r  
1 2 ê ê  
2 3 v v  
3 4 e e  
4 5 r r  
5 6 a a  
6
```



# Romance Languages

Language	Number\Person	First	Second	Third
Italian				
	singular	<i>canterò</i>	<i>canterai</i>	<i>canterà</i>
	plural	<i>canteremo</i>	<i>canterete</i>	<i>canteranno</i>
Spanish				
	singular	<i>cantaré</i>	<i>cantarás</i>	<i>cantará</i>
	plural	<i>cantaremos</i>	<i>cantaréis</i>	<i>cantarán</i>
Portuguese				
	singular	<i>cantarei</i>	<i>cantarás</i>	<i>cantará</i>
	plural	<i>cantaremos</i>	<i>cantareis</i>	<i>cantarão</i>



# Ambiguity

In the transducer for future tense, there is no ambiguity: A surface form has only one lexical form with a unique final state.

This is not the case with the present tense

(je) *chante* 'I sing'

(il) *chante* 'he sings'

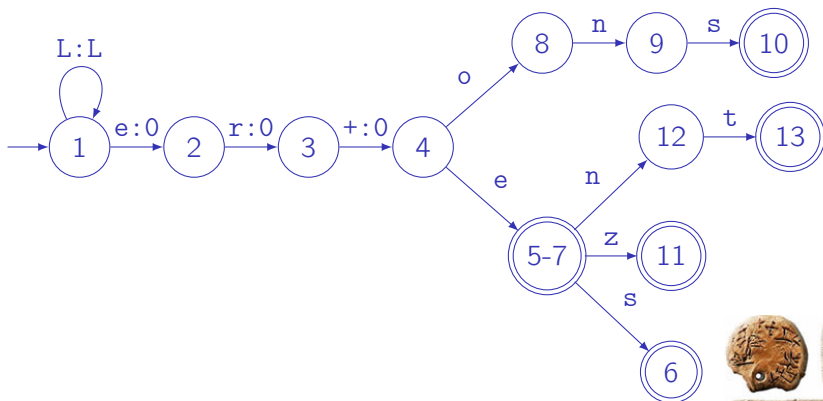
Number\Person	First	Second	Third
singular	<i>chante</i>	<i>chantes</i>	<i>chante</i>
plural	<i>chantons</i>	<i>chantez</i>	<i>chantent</i>



# Transducer Ambiguity

Final states 5 and 7 are the same.

The implementation in Prolog is similar to that of the future tense. Using backtracking, the transducer can produce all the final states reflecting the morphological ambiguity.



# Koskenniemi's Rules

Koskenniemi described morphology with declarative rules.

They use the left and right context and the  $\Rightarrow$ ,  $\Leftarrow$ ,  $\Leftrightarrow$ , or  $/\Leftarrow$  operators

In English, a lexical *y* can correspond to a surface *i* as in *happier*.

It occurs when *y* is preceded by a consonant and followed by *-er*, *-ed*, or *-s*.

①  $y:i \Leftarrow C:C \_ \_ +:0 \ e:e \ r:r$

②  $y:i \Leftarrow C:C \_ \_ +:e \ s:s$

③  $y:i \Leftarrow C:C \_ \_ +:0 \ e:e \ d:d$



# Two-level Rules

Lexical:surface transduction is described by rules.

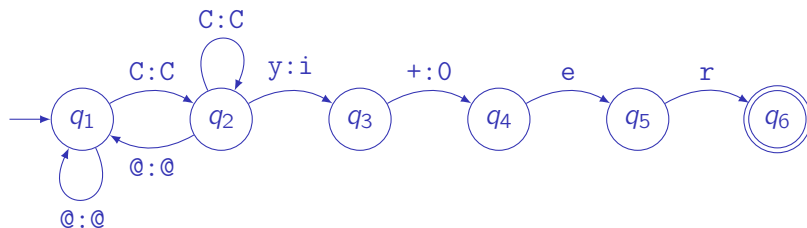
Rules	Description
$a:b \Rightarrow lc \_ rc$	a is transduced as b <b>only</b> when it has lc to the left and rc to the right
$a:b \Leftarrow lc \_ rc$	a is <b>always</b> transduced as b when it has lc to the left and rc to the right
$a:b \Leftrightarrow lc \_ rc$	a is transduced as b <b>always and only</b> when it has lc to the left and rc to the right
$a:b / \Leftarrow lc \_ rc$	a is <b>never</b> transduced as b when it has lc to the left and rc to the right





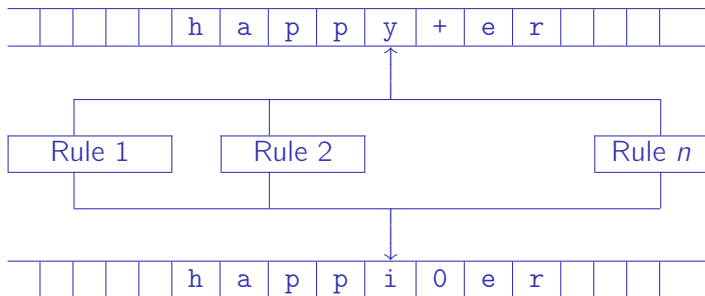
# Parallel Rules

All the rules are applied in parallel (provided that their context match)



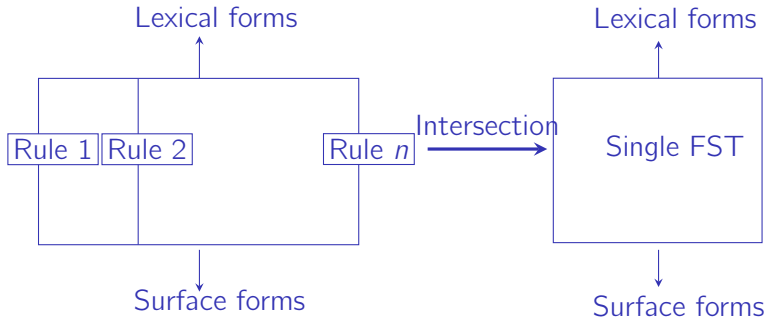
# Rules and Transducers

Rules can be compiled as an equivalent transducer



# Rule Intersection

The parallel transducers are then combined into a single one using the transducer intersection.



# Problems with Intersection

The intersection of two finite automata defines a finite-state automaton. It is not always the case for finite-state transducers.

Kaplan and Kay (1994) demonstrated that when surface and lexical pairs have the same length – without  $\epsilon$  –, the intersection is a transducer.

This property is sufficient to intersect the rules in practical applications.

In fact, transducers obtained from two-level rules are intersected by treating the  $\epsilon$  symbol as an ordinary symbol (Beesley and Karttunen 2003, p. 55).



# Xerox

Originally, rules were compiled by hand.

However, it can quickly become intractable especially when it comes to managing conflicting rules or when rule contexts interfere with transduced symbols.

To solve it, we can use a compiler that creates transducers automatically from two-level rules.

The Xerox's XFST is an example of it. It is a publicly available tool and to date the only serious implementation of a morphological rule compiler.



# Morphology of French Verbs

We used the stem and a set of suffixes for French regular verbs.

French irregular verbs are notoriously more complex.

Chanod (1994) gives an example of decomposition into simple rules.

<b>Infinitive</b>	<b>courir</b>	<b>dormir</b>	<b>battre</b>	<b>peindre</b>	<b>écrire</b>
First person sing.	cours <u>s</u>	<b>dors</b>	<b>bats</b>	<b>peins</b>	écris
Second person sing.	cours <u>s</u>	<b>dors</b>	<b>bats</b>	<b>peins</b>	écris
Third person sing.	court <u>t</u>	<b>dort</b>	<b>bat</b>	<b>peint</b>	écrit
First person pl.	cour <u>ons</u>	dormons	battons	peignons	<b>écrivons</b>
Second person pl.	cour <u>ez</u>	dormez	battez	peignez	<b>écrivez</b>
Third person pl.	cour <u>ent</u>	dorment	battent	peignent	<b>écrivent</b>



# French Morphology

<b>Lexical form:</b> stem	dormir	+IndP	+SG	+P1
	↕		↕	
<b>Intermediate form:</b> inflection	dorm	+IndP	+SG	+P1
	↕		↕	
<b>Intermediate form:</b> deletion of <i>m</i> followed by <i>s</i>	dorm		s	
	↕		↕	
<b>Surface form:</b>	dor		s	

From *peindre* to *peins*

n:0 ⇔ g \_\_ [s|t]



# Composition and Intersection

