Morphological Analysis

Daniel Zeman

March 31, 2021

Charles University
Faculty of Mathematics and Physics
Institute of Formal and Applied Linguistics

Morphological Annotation

ID	FORM	LEMMA	POS	FEATS
1	They	they	PRON	Case=Nom|Number=Plur
2	buy	buy	VERB	Mood=Ind|Number=Plur|Person=3|Tense=Pres
3	and	and	CCONJ	_
4	sell	sell	VERB	Mood=Ind|Number=Plur|Person=3|Tense=Pres
5	books	book	NOUN	Number=Plur
6	.	.	PUNCT	_

Morphological Annotation

ID	FORM	LEMMA	POS	FEATS
1	Kupují	kupovat	VERB	Mood=Ind|Number=Plur|Person=3|Tense=Pres
2	a	a	CCONJ	_
3	prodávají	prodávat	VERB	Mood=Ind|Number=Plur|Person=3|Tense=Pres
4	knihy	kniha	NOUN	Case=Acc|Gender=Fem|Number=Plur
5	.	.	PUNCT	_

Morphological Annotation

ID	FORM	LEMMA	POS	FEATS
1	Kupují	kupovat	VERB	Mood=Ind|Number=Plur|Person=3|Tense=Pres
2	a	a	CCONJ	_
3	prodávají	prodávat	VERB	Mood=Ind|Number=Plur|Person=3|Tense=Pres
4	knihy	kniha	NOUN	Case=Acc|Gender=Fem|Number=Plur
5	.	.	PUNCT	_

ID	FORM	LEMMA	XPOS
1	Kupují	kupovat	VB-P---3P-AA---
2	a	a	J^----------
3	prodávají	prodávat	VB-P---3P-AA---
4	knihy	kniha	NNFP4-----A----
5	.	.	Z:----------

Tagsets

- Tag as a set of feature (category) values ... $(k_1, k_2, ..., k_n)$
- Simple list of tags

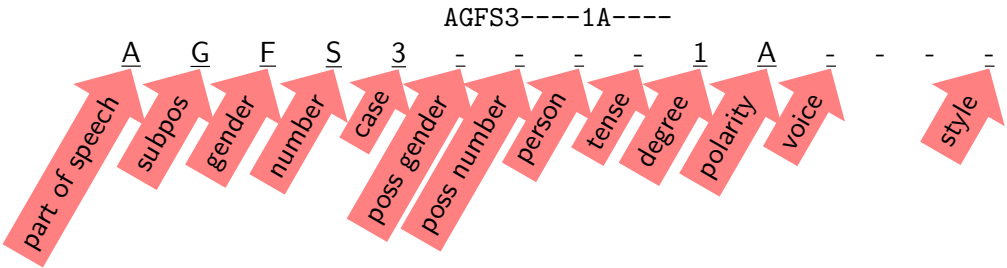
$$T = \{t_i\}_{i=1..n}$$

- 1-1 mapping between tags and feature-value space

$$T \leftrightarrow (K_1, K_2, ..., K_n)$$

- English
  - Penn Treebank (45 tags), Brown Corpus (87), Claws c5 (62), London-Lund (197)
- Czech
  - Prague Dependency Treebank (4294; positional), Multext-East (1458; Orwell 1984 parallel corpus), Majka / Desam (MU Brno), Prague Spoken Corpus (over 10000!)
- Universal Dependencies (UD)
  - 17 universal POS tags (UPOS)
  - 24 universal features, each 1 – 34 possible values

Czech Positional Tags of PDT



Parts of Speech in PDT

- N noun *(podstatné jméno)*
- A adjective *(přídavné jméno)*
- P pronoun *(zájmeno)*
- C numeral *(číslovka)*
- V verb *(sloveso)*
- D adverb *(příslovce)*
- R preposition *(předložka)*
- J conjunction *(spojka)*
- T particle *(částice)*
- I interjection *(citoslovce)*
- Z special (e.g. punctuation) *(zvláštní, např. interpunkce)*
- X unknown word *(neznámé slovo)*

Gender in PDT

M	masculine animate *(mužský životný)*	**Y**	M or I
I	masculine inanimate *(mužský neživotný)*	**T**	I or F
F	feminine *(ženský)*	**W**	I or N
N	neuter *(střední)*	**H, Q**	F or N
X	unknown *(neznámý)*	**Z**	M, I or N

Number in PDT

S	singular *(jednotné)*
D	dual *(dvojné)*
P	plural *(množné)*
X	unknown *(neznámé)*

Case in PDT

1	nominative *(první pád)*
2	genitive *(druhý pád)*
3	dative *(třetí pád)*
4	accusative *(čtvrtý pád)*
5	vocative *(pátý pád)*
6	locative *(šestý pád)*
7	instrumental *(sedmý pád)*
X	unknown *(neznámý)*

Degree, Polarity, and Person

- Degree of comparison of adjectives and adverbs:
  - 1 (positive), 2 (comparative), 3 (superlative)
- Polarity of verbs, adjectives, adverbs, and nouns:
  - A (affirmative), N (negative)
- Person of pronouns and verbs:
  - 1, 2, 3

Mood, Tense, and Voice

- Changes relevance of other categories (such as person and number) $\Rightarrow$ in a sense, these are subparts of speech
- Tense:
  - P (present – *přítomný*), M (past – *minulý*), F (future – *budoucí*)
- Voice:
  - A (active – *činný*), P (passive – *trpný*)
- Mood:
  - N (indicative – *oznamovací*), R (imperative – *rozkazovací*), C (conditional – *podmiňovací*)

Style and/or Variant

1	other variant, less frequent
2	other variant, very rare, archaic or literary
3	very archaic or colloquial variant
5	colloquial, tolerated both in spoken and in written discourse
6	colloquial, inappropriate in written discourse
7	colloquial like 6 but less preferred by speakers
9	special usage (e.g. after some prepositions)

The Penn Treebank Tagset

1. **CC** coordinating conjunction
2. **CD** cardinal number
3. **DT** determiner
4. **EX** existential *there*
5. **FW** foreign word
6. **IN** preposition or subordinating conjunction
7. **JJ** adjective
8. **JJR** adjective, comparative
9. **JJS** adjective, superlative
10. **LS** list item marker
11. **MD** modal
12. **NN** noun, singular/mass
13. **NNS** noun, plural
14. **NNP** proper noun, singular
15. **NNPS** proper noun, plural
16. **PDT** predeterminer
17. **POS** possessive ending
18. **PRP** personal pronoun
19. **PRP$** possessive pronoun

The Penn Treebank Tagset

20. **RB** adverb
21. **RBR** adverb, comparative
22. **RBS** adverb, superlative
23. **RP** particle
24. **SYM** symbol
25. **TO** *to*
26. **UH** interjection
27. **VB** verb, base *(do)*
28. **VBD** verb, past *(did)*
29. **VBG** verb, gerund or present participle *(doing)*
30. **VBN** verb, past participle *(done)*
31. **VBP** verb, non-3rd person singular present *(do)*
32. **VBZ** verb, 3rd person singular present *(does)*
33. **WDT** wh-determiner *(which)*
34. **WP** wh-pronoun *(who)*
35. **WP$** possessive wh-pronoun *(whose)*
36. **WRB** wh-adverb *(where)*
37. **.** period...

Universal POS Tags

http://universaldependencies.org/u/pos/index.html

- **NOUN**
- **PROPN** (proper noun)
- **VERB**
- **ADJ** (adjective)
- **ADV** (adverb)
- **INTJ** (interjection)
- **PRON** (pronoun)
- **DET** (determiner)
- **AUX** (auxiliary)
- **NUM** (numeral)
- **ADP** (adposition)
- **SCONJ** (subordinating conjunction)
- **CCONJ** (coordinating conjunction)
- **PART** (particle)
- **PUNCT** (punctuation)
- **SYM** (symbol)
- **X** (unknown)

Universal Features

http://universaldependencies.org/u/feat/index.html

- **PronType** *(druh zájmena)*
- **NumType** *(druh číslovky)*
- **Poss** *(přivlastňovací)*
- **Reflex** *(zvratné)*
- **Foreign** *(cizí slovo)*
- **Abbr** *(zkratka)*
- **Typo** *(překlep)*
- **Gender** *(rod)*
- **Animacy** *(životnost)*
- **NounClass** *(jmenná třída)*
- **Number** *(číslo)*
- **Case** *(pád)*
- **Definite(ness)** *(určitost)*
- **Degree** *(stupeň)*
- **VerbForm** *(slovesný tvar)*
- **Mood** *(způsob)*
- **Tense** *(čas)*
- **Aspect** *(vid)*
- **Voice** *(slovesný rod)*
- **Evident(iality)** *(zjevnost)*
- **Polarity** *(zápor)*
- **Person** *(osoba)*
- **Polite(ness)** *(zdvořilost)*
- **Clusivity** *(kluzivita)*

Part of Speech

- Vague definitions, criteria or mixed nature
- Looong tradition... (difficult to change)
  - Traditional linguistics:
    - Classification differs cross-linguistically!
    - Even among established classes, not just endemic minor parts of speech.
  - Computational linguistics:
    - Dozens of classes and subclasses
    - Significant differences even within one language

History

- 4th century BC: Sanskrit
- European tradition (prevailing in modern linguistics): Ancient Greek
  - Plato (4th century BC): sentence consists of nouns and verbs
  - Aristotle added "conjunctions" (included conjunctions, pronouns and articles)
  - End of 2nd century BC: classification stabilized at 8 categories (Διονύσιος ὁ Θρᾷξ: *Τέχνη Γραμματική* / Dionysios o Thrax: *Art of Grammar*)

Ancient Greek Word Classes

- Noun (ὄνομα *onoma*)
  - inflected for case, signifying a concrete or abstract entity
- Verb (ῥῆμα *rēma*)
  - without case inflection, but inflected for tense, person and number, signifying an activity or process performed or undergone
- Participle (μετοχή *metochē*)
  - sharing the features of the verb and the noun
- Interjection (ἄρθρον *arthron*)
  - expressing emotion alone
- Pronoun (ἀντωνυμία *antōnymia*)
  - substitutable for a noun and marked for person
- Preposition (πρόθεσις *prothesis*)
  - placed before other words in composition and in syntax
- Adverb (ἐπίρρημα *epirrēma*)
  - without inflection, in modification or in addition to a verb
- Conjunction (σύνδεσμος *syndesmos*)
  - binding together the discourse and filling gaps in its interpretation

Where Are Adjectives?

- The best matching Ancient Greek definition is that of nouns, and perhaps participles.
- Adjectives are a relatively new (1767) invention from France:
  - Nicolas Beauzée: *Grammaire générale, ou exposition raisonnée des éléments nécessaires du langage.* Paris, 1767

Traditional English Parts of Speech

1. Noun
2. Verb
3. Adjective
4. Adverb
5. Pronoun
6. Preposition
7. Conjunction
8. Interjection

"Traditional" means: taught in elementary schools, marked in dictionaries.

Linguists (and especially computational linguists) may see other categories, e.g., determiners.

Traditional Czech Parts of Speech

1. Noun *(podstatné jméno, substantivum)*
2. Adjective *(přídavné jméno, adjektivum)*
3. Pronoun *(zájmeno)*
4. Numeral *(číslovka)*
5. Verb *(sloveso)*
6. Adverb *(příslovce, adverbium)*
7. Preposition *(předložka)*
8. Conjunction *(spojka)*
9. Particle *(částice)*
10. Interjection *(citoslovce)*

Openness vs. Closeness
Content vs. Function Words

- Open classes (take new words)
  - verbs (non-auxiliary), nouns, adjectives, adjectival adverbs, interjections
  - word formation (derivation) across classes
- Closed classes (words can be enumerated)
  - pronouns / determiners, adpositions, conjunctions, particles
  - pronominal adverbs
  - auxiliary and modal verbs / particles
  - numerals (mathematically infinite, linguistically closed)
  - typically they are not base for derivation
- Even closed classes evolve but over longer period of time
  - *Vuestra Merced* "Your Mercy, Your Grace" $\Rightarrow$ *usted* (new singular second person pronoun in formal/honorific register)
  - $\Rightarrow$ new plural *ustedes*

Finite-State Morphology

Finite-State Automaton/Machine (FSA)

- Five-tuple (A, Q, P, q_0, F).
  - A ... finite alphabet of input symbols
  - Q ... finite set of states
  - P ... transition function (set of rules) $A \times Q \to Q$
  - $q_0 \in Q$ ... initial state
  - $F \subseteq Q$ ... set of terminal states

- A word is accepted as correct if we read it as input and we end up in a terminal state.
- An additional action can be bound to the terminal state (output info).

Example of Finite-State Machine

- Checks correct spelling of Czech: *dě*, *tě*, *ně*...
- Czech orthographical rules:
  - *di*, *ti*, *ni* is pronounced *[ďi, ťi, ňi]*
  - *dě*, *tě*, *ně* is pronounced *[ďe, ťe, ňe]*

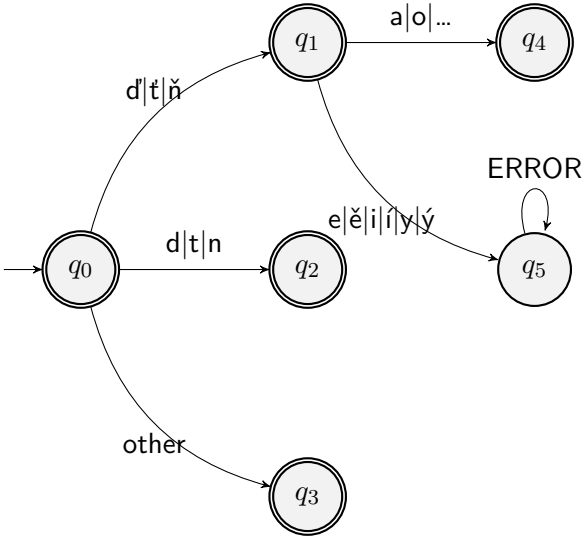
Example of Finite-State Machine

- Checks correct spelling of Czech: *dě*, *tě*, *ně*...
- Czech orthographical rules:
  - *di*, *ti*, *ni* is pronounced *[ďi, ťi, ňi]*
  - *dě*, *tě*, *ně* is pronounced *[ďe, ťe, ňe]*
  - Orthography prohibits strings *ďi*, *ťi*, *ňi*, *ďy*, *ťy*, *ňy*, *ďe*, *ťe*, *ňe*, *ďě*, *ťě*, *ňě*

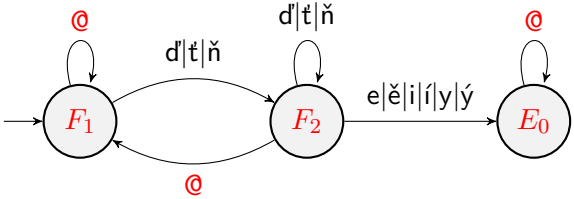
Example of Finite-State Machine

- Checks correct spelling of Czech: *dě, tě, ně...*
- Czech orthographical rules:
  - *di, ti, ni* is pronounced *[ďi, ťi, ňi]*
  - *dě, tě, ně* is pronounced *[ďe, ťe, ňe]*
  - Orthography prohibits strings *ďi, ťi, ňi, ďy, ťy, ňy, ďe, ťe, ňe, ďě, ťě, ňě*
  - Note however that long *ďé, ťé* is permitted: these are the names of the letters *Ď, Ť*. (And *ě* cannot be used for them because it is short.)
- Exception: Czech system of transcription of Mandarin Chinese (used for Chinese names in news and encyclopedias):
  - *ťin* ... pinyin equivalent is *jin*

Example of Finite-State Machine



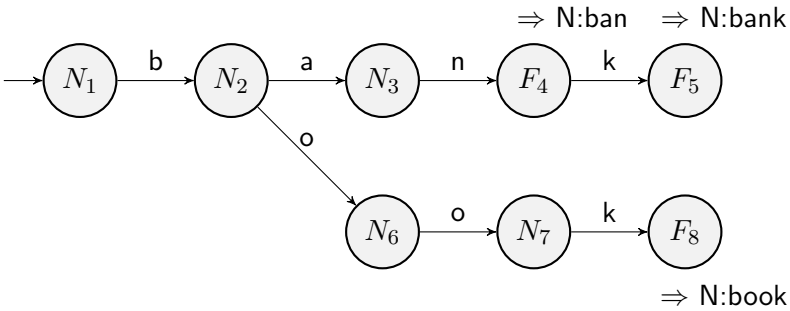
Example of Finite-State Machine (polished, new notation)



- Initial state indexed 1, not 0 (here F_1).
- Index 0 reserved for the error state.
- Terminal states denoted by the letter F.
- The *at* sign ("@") means "other", i.e., characters not found on other transitions from the same state.

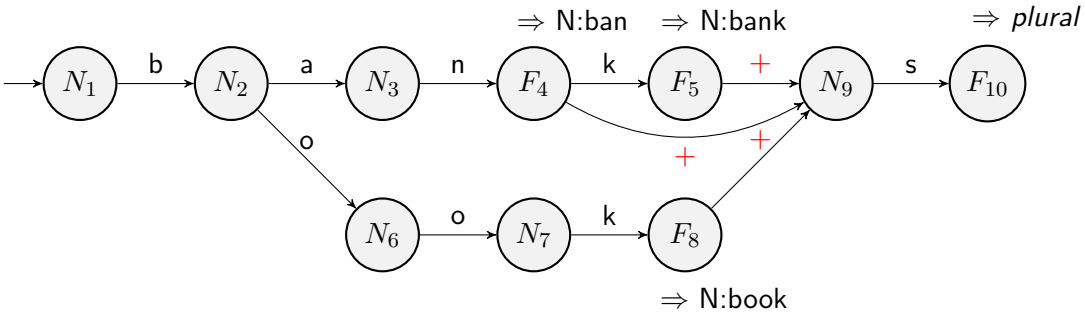
Lexicon

- Implemented as a FSA (trie) *[tri:]*.



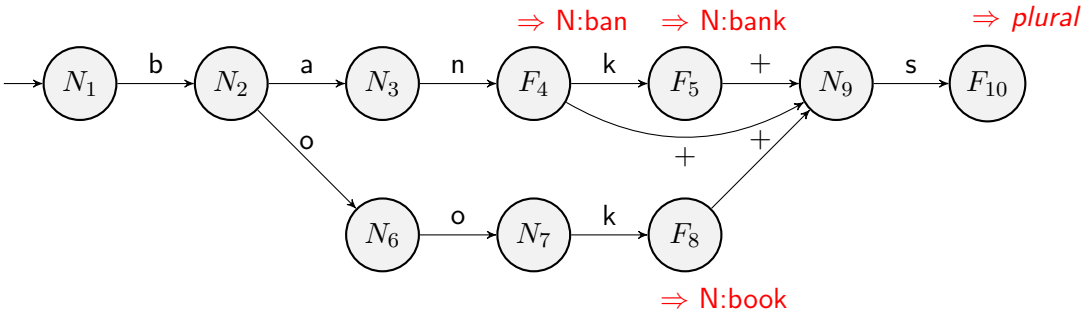
Lexicon

- Implemented as a FSA (trie) *[tri:]*.
- Composed of multiple sublexicons (prefixes, stems, suffixes).



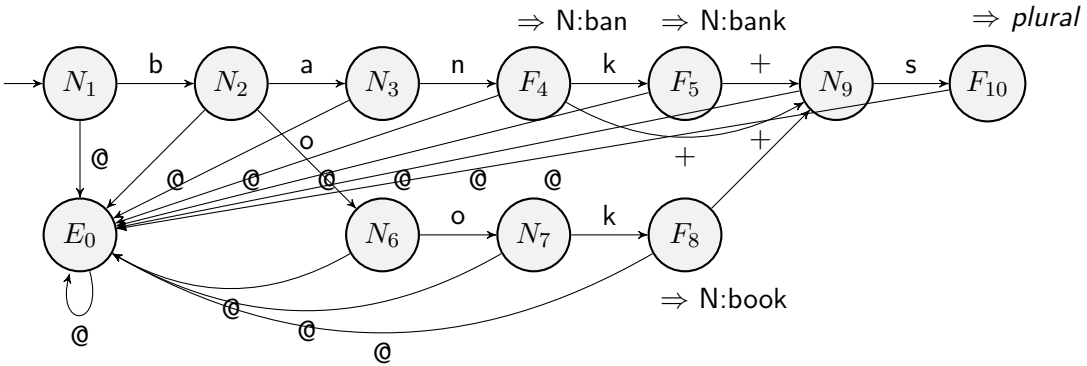
Lexicon

- Implemented as a FSA (trie) *[tri:]*.
- Composed of multiple sublexicons (prefixes, stems, suffixes).
- Notes (glosses) at the end of every sublexicon.
- Compiled from a list of strings and sublexicon references.



Lexicon

- Implemented as a FSA (trie) *[tri:]*.
- Composed of multiple sublexicons (prefixes, stems, suffixes).
- Notes (glosses) at the end of every sublexicon.
- Compiled from a list of strings and sublexicon references.



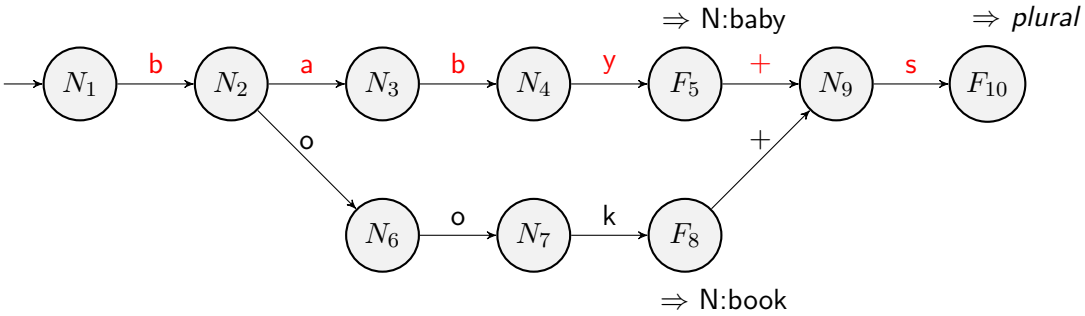
Interlinking Sublexicons

- Unlike trie the lexicon is not a tree but a DAG (directed acyclic graph)
- Each sublexicon entry knows the set of sublexicons we can jump to in the next step ⇒ continuation class or alternation

Sublexicon	Entry	Gloss	Continuation Class
INIT			NounStem AdjStem VerbStem ...
NounStem	muž	N:muž(man)	NMmanSuff
	učitel	N:učitel(teacher)	NMmanSuff
	žen	N:žena(woman)	NFwomSuff
	růž	N:růže(rose)	NFrosSuff
NMmanSuff	+e	Sing:Gen	
	+i	Sing:Dat	
	+e	Sing:Acc	

A Problem Called Phonology

- Sometimes attaching a suffix causes phoneme or grapheme (spelling) changes!
  - For simplicity I will call both *phonology*.
- Plural of *baby* is not **babys* but *babies*!



Two-Level Morphology

- Integration of morphology and phonology is possible and easy.
- Upper (lexical) language
- Lower (surface) language
- Two-level rules:

```
b a b y + 0 s
b a b i 0 e s
```

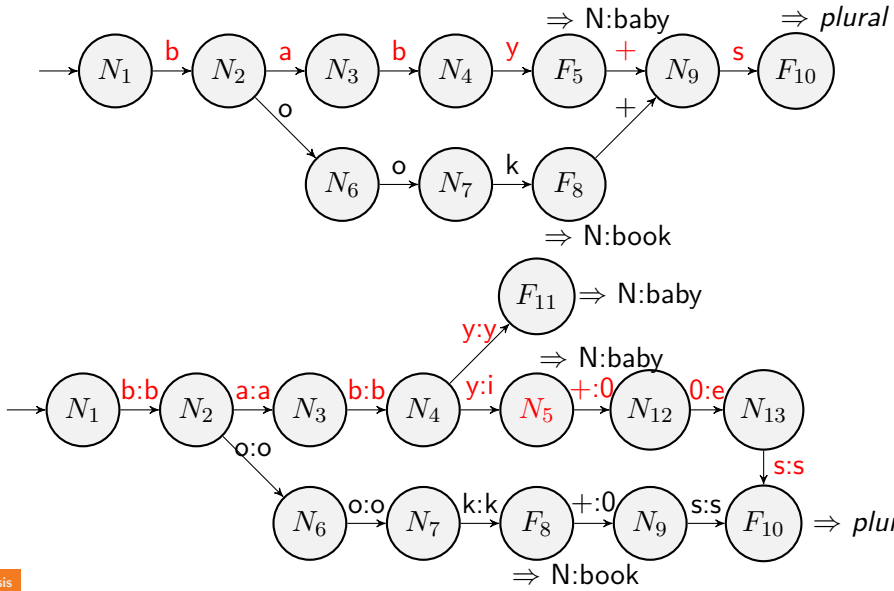
- Alternative notation with colons:

```
b:b a:a b:b y:i +:0 0:e s:s
```

Finite-State Transducer *(převodník)*

- Transducer is a special case of automaton
  - Symbols are pairs (r:s) from finite alphabets R and S.
- Checking (finite-state automaton)
  - Input: sequence of characters
  - Output: yes / no (accept / reject) + state id / gloss
- Analysis (finite-state transducer)
  - Input: sequence $s \in S$ (surface string)
  - Output: sequence $r \in R$ (lexical string) + state id / gloss
    - So how do we obtain it?
- Generation (finite-state transducer)
  - Same as analysis but swapped roles $S \leftrightarrow R$

Automaton vs. Transducer



Another Way of Rule Notation: Two-Level Grammar

- If lexical *y* is followed by *+s*, then on surface the *y* must be replaced by *i* (generation).
- If surface *i* is followed by *+s*, then in lexicon the *i* must be replaced by *y* (analysis).

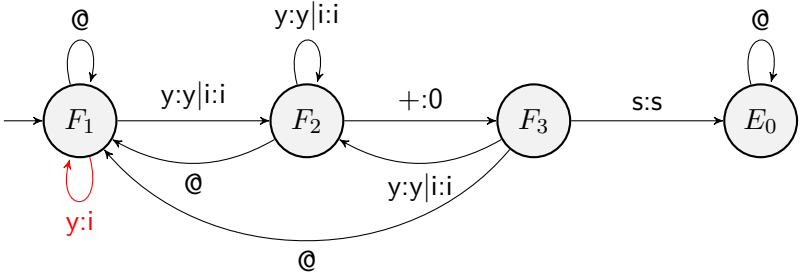
```
y:i <= _ +:0 s:s
```

- We don't require the reverse implication this time. It is possible that *y* corresponds to *i* elsewhere for other reasons.
- In the same context we also require that an *e* is inserted before *s*:

```
0:e <= y:i +:0 _ s:s
```

- Create a transducer (FST) that converts between the surface and lexical layers.
  - More precisely: FST is an automaton that only checks that we are converting the layers correctly.

FST Example: `y:i <= _ +:0 s:s`



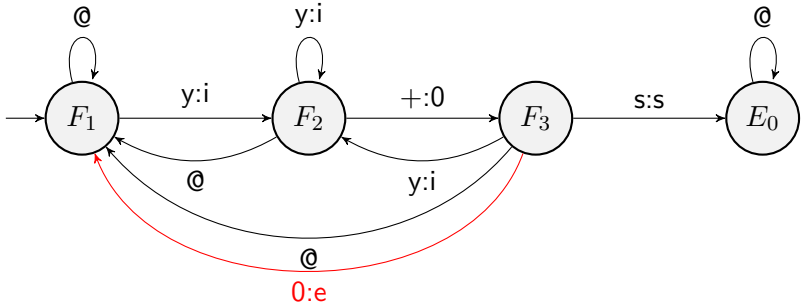
How to Get the FST Input

- FSA simply checked the input.
- With FST we only read half of the input.
- Where do we get the other half?

How to Get the FST Input

- FSA simply checked the input.
- With FST we only read half of the input.
- Where do we get the other half?
- We know it in advance!
  - Typical letter corresponds to itself: i:i, y:y
  - Some letters arise phonologically: y:i
  - We thus know in advance that a surface *i* can correspond either to lexical *i* or *y*.
  - We will check both possibilities. If both are accepted, the analyzed word is ambiguous.

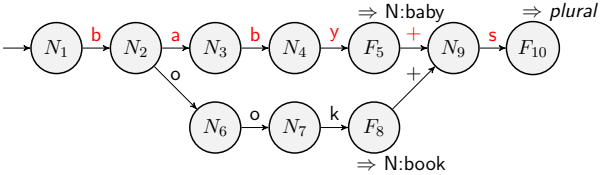
FST Example: 0:e <= y:i +:0 _ s:s

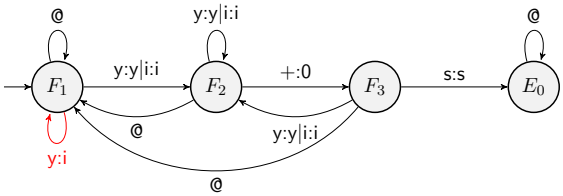


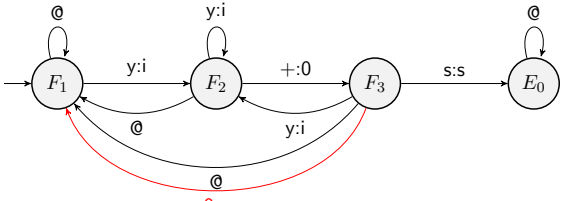
How Does It Work Together

- Parallel FST (including lexicon FSA) can be compiled to one gigantic FST.
- The transducer itself in fact does not convert, it only checks.
- Nevertheless the transducer is a source of information what can be converted to what (i.e. what we can try and have checked by the FST).
  - Besides explicit conversion rules we can also assume for all x the default conversion rule `x:x`.

Lexicon and Rules Together







Two-Level Morphological Analysis

1. Initialize set of paths $P = \{\}$.
2. Read input symbols one-by-one.
3. For each input symbol x generate all lexical symbols y that may correspond to the empty symbol (`y:0`).
4. Extend all paths in P by all corresponding pairs (`y:0`).
5. Check all new extensions against the phonological transducers and the lexical automaton. Remove disallowed paths (partial solutions) from P.
6. Repeat 4–5 until the maximum possible number of subsequent zeros is reached.
7. Generate all possible lexical symbols z for the current input symbol x. Create pairs.
8. Extend each path in P by all such pairs.
9. Check all paths in P (the next transition in FST/FSA). Remove impossible paths.
10. Repeat since step 3 until input finishes.
11. Collect glosses from the lexicon from all paths that survived.

Algorithm Example

- Every letter corresponds to itself
- In addition: `y:i`, `+:0`, `0:e`
- Input: *babies*
- Try inserting lexical + (`+:0`) … blocked by lexicon (no word starts like that)

Algorithm Example

- Every letter corresponds to itself
- In addition: `y:i`, `+:0`, `0:e`
- Input: *babies*
- Try inserting lexical + (`+:0`) ... blocked by lexicon (no word starts like that)
- Try `b:b` ... OK (neither lexicon nor the transducers object)

Algorithm Example

- Every letter corresponds to itself
- In addition: `y:i`, `+:0`, `0:e`
- Input: *babies*
- Try inserting lexical + (`+:0`) ... blocked by lexicon (no word starts like that)
- Try `b:b` ... OK (neither lexicon nor the transducers object)
- `b:b +:0` ... lexicon error

Algorithm Example

- Every letter corresponds to itself
- In addition: `y:i`, `+:0`, `0:e`
- Input: *babies*
- Try inserting lexical + (`+:0`) … blocked by lexicon (no word starts like that)
- Try `b:b` … OK (neither lexicon nor the transducers object)
- `b:b +:0` … lexicon error
- `b:b a:a` … OK

Algorithm Example

- Every letter corresponds to itself
- In addition: `y:i`, `+:0`, `0:e`
- Input: *babies*
- Try inserting lexical + (`+:0`) … blocked by lexicon (no word starts like that)
- Try `b:b` … OK (neither lexicon nor the transducers object)
- `b:b +:0` … lexicon error
- `b:b a:a` … OK
- `b:b a:a +:0` … lexicon error

Algorithm Example

- Every letter corresponds to itself
- In addition: `y:i`, `+:0`, `0:e`
- Input: *babies*
- Try inserting lexical + (`+:0`) ... blocked by lexicon (no word starts like that)
- Try `b:b` ... OK (neither lexicon nor the transducers object)
- `b:b +:0` ... lexicon error
- `b:b a:a` ... OK
- `b:b a:a +:0` ... lexicon error
- `b:b a:a b:b` ... OK

Algorithm Example

- Every letter corresponds to itself
- In addition: `y:i`, `+:0`, `0:e`
- Input: *babies*
- Try inserting lexical + (`+:0`) … blocked by lexicon (no word starts like that)
- Try `b:b` … OK (neither lexicon nor the transducers object)
- `b:b +:0` … lexicon error
- `b:b a:a` … OK
- `b:b a:a +:0` … lexicon error
- `b:b a:a b:b` … OK
- `b:b a:a b:b +:0` … l. error

Algorithm Example

- Every letter corresponds to itself
- In addition: `y:i`, `+:0`, `0:e`
- Input: *babies*
- Try inserting lexical + (`+:0`) … blocked by lexicon (no word starts like that)
- Try `b:b` … OK (neither lexicon nor the transducers object)
- `b:b +:0` … lexicon error
- `b:b a:a` … OK
- `b:b a:a +:0` … lexicon error
- `b:b a:a b:b` … OK
- `b:b a:a b:b +:0` … l. error
- `b:b a:a b:b i:i` … l. error

Algorithm Example

- Every letter corresponds to itself
- In addition: `y:i`, `+:0`, `0:e`
- Input: *babies*
- Try inserting lexical + (`+:0`) … blocked by lexicon (no word starts like that)
- Try `b:b` … OK (neither lexicon nor the transducers object)
- `b:b +:0` … lexicon error
- `b:b a:a` … OK
- `b:b a:a +:0` … lexicon error
- `b:b a:a b:b` … OK
- `b:b a:a b:b +:0` … l. error
- `b:b a:a b:b i:i` … l. error
- `b:b a:a b:b y:i` … OK

Algorithm Example

- Every letter corresponds to itself
- In addition: `y:i`, `+:0`, `0:e`
- Input: *babies*
- Try inserting lexical + (`+:0`) … blocked by lexicon (no word starts like that)
- Try `b:b` … OK (neither lexicon nor the transducers object)
- `b:b +:0` … lexicon error
- `b:b a:a` … OK
- `b:b a:a +:0` … lexicon error
- `b:b a:a b:b` … OK
- `b:b a:a b:b +:0` … l. error
- `b:b a:a b:b i:i` … l. error
- `b:b a:a b:b y:i` … OK $[p_1]$
- … `b:b y:i +:0` … OK $[p_1 \rightarrow p_2]$

Algorithm Example

- Every letter corresponds to itself
- In addition: `y:i`, `+:0`, `0:e`
- Input: *babies*
- Try inserting lexical + (`+:0`) … blocked by lexicon (no word starts like that)
- Try `b:b` … OK (neither lexicon nor the transducers object)
- `b:b +:0` … lexicon error
- `b:b a:a` … OK
- `b:b a:a +:0` … lexicon error
- `b:b a:a b:b` … OK
- `b:b a:a b:b +:0` … l. error
- `b:b a:a b:b i:i` … l. error
- `b:b a:a b:b y:i` … OK $[p_1]$
- … `b:b y:i +:0` … OK $[p_1 \rightarrow p_2]$
- … `b:b y:i +:0 +:0` … error $[p_2 \rightarrow ?]$

Algorithm Example

- Every letter corresponds to itself
- In addition: `y:i`, `+:0`, `0:e`
- Input: *babies*
- Try inserting lexical + (`+:0`) … blocked by lexicon (no word starts like that)
- Try `b:b` … OK (neither lexicon nor the transducers object)
- `b:b +:0` … lexicon error
- `b:b a:a` … OK
- `b:b a:a +:0` … lexicon error
- `b:b a:a b:b` … OK
- `b:b a:a b:b +:0` … l. error
- `b:b a:a b:b i:i` … l. error

- `b:b a:a b:b y:i` … OK $[p_1]$
- … `b:b y:i +:0` … OK $[p_1 \rightarrow p_2]$
- … `b:b y:i +:0 +:0` … error
- … `y:i e:e` … error $[p_1 \rightarrow ?]$
- … `y:i 0:e` … OK $[p_1 \rightarrow p_3]$
- … `y:i +:0 e:e` … error $[p_2 \rightarrow ?]$
- … `y:i +:0 0:e` … OK $[p_2 \rightarrow p_4]$

Algorithm Example

- Every letter corresponds to itself
- In addition: `y:i`, `+:0`, `0:e`
- Input: *babies*
- Try inserting lexical + (`+:0`) … blocked by lexicon (no word starts like that)
- Try `b:b` … OK (neither lexicon nor the transducers object)
- `b:b +:0` … lexicon error
- `b:b a:a` … OK
- `b:b a:a +:0` … lexicon error
- `b:b a:a b:b` … OK
- `b:b a:a b:b +:0` … l. error
- `b:b a:a b:b i:i` … l. error

- `b:b a:a b:b y:i` … OK
- … `b:b y:i +:0` … OK
- … `b:b y:i +:0 +:0` … error
- … `y:i e:e` … error
- … `y:i 0:e` … OK $[p_1 \rightarrow p_3]$
- … `y:i +:0 e:e` … error
- … `y:i +:0 0:e` … OK $[p_2 \rightarrow p_4]$
- … `y:i 0:e +:0` … OK $[p_3 \rightarrow p_5]$
- … `y:i 0:e +:0 +:0` … error $[p_5 \rightarrow ?]$
- … `+:0 0:e +:0` … error $[p_4 \rightarrow ?]$

Algorithm Example

- Every letter corresponds to itself
- In addition: `y:i`, `+:0`, `0:e`
- Input: *babies*
- Try inserting lexical + (`+:0`) … blocked by lexicon (no word starts like that)
- Try `b:b` … OK (neither lexicon nor the transducers object)
- `b:b +:0` … lexicon error
- `b:b a:a` … OK
- `b:b a:a +:0` … lexicon error
- `b:b a:a b:b` … OK
- `b:b a:a b:b +:0` … l. error
- `b:b a:a b:b i:i` … l. error

- `b:b a:a b:b y:i` … OK
- … `b:b y:i +:0` … OK
- … `b:b y:i +:0 +:0` … error
- … `y:i e:e` … error
- … `y:i 0:e` … OK $[p_1 \rightarrow p_3]$
- … `y:i +:0 e:e` … error
- … `y:i +:0 0:e` … OK $[p_2 \rightarrow p_4]$
- … `y:i 0:e +:0` … OK $[p_3 \rightarrow p_5]$
- … `y:i 0:e +:0 +:0` … error
- … `+:0 0:e +:0` … error
- … `y:i 0:e s:s` … error $[p_3 \rightarrow ?]$
- … `+:0 0:e s:s` … OK $[p_4 \rightarrow p_6]$
- … `0:e +:0 s:s` … OK $[p_5 \rightarrow p_7]$

Algorithm Example

- Every letter corresponds to itself
- In addition: `y:i`, `+:0`, `0:e`
- Input: *babies*
- Try inserting lexical + (`+:0`) … blocked by lexicon (no word starts like that)
- Try `b:b` … OK (neither lexicon nor the transducers object)
- `b:b +:0` … lexicon error
- `b:b a:a` … OK
- `b:b a:a +:0` … lexicon error
- `b:b a:a b:b` … OK
- `b:b a:a b:b +:0` … l. error
- `b:b a:a b:b i:i` … l. error

- `b:b a:a b:b y:i` … OK
- … `b:b y:i +:0` … OK
- … `b:b y:i +:0 +:0` … error
- … `y:i e:e` … error
- … `y:i 0:e` … OK
- … `y:i +:0 e:e` … error
- … `y:i +:0 0:e` … OK
- … `y:i 0:e +:0` … OK
- … `y:i 0:e +:0 +:0` … error
- … `+:0 0:e +:0` … error
- … `y:i 0:e s:s` … error
- … `+:0 0:e s:s` … OK $[p_4 \rightarrow p_6]$
- … `0:e +:0 s:s` … OK $[p_5 \rightarrow p_7]$
- … `+:0 0:e s:s +:0` … error
- … `0:e +:0 s:s +:0` … error

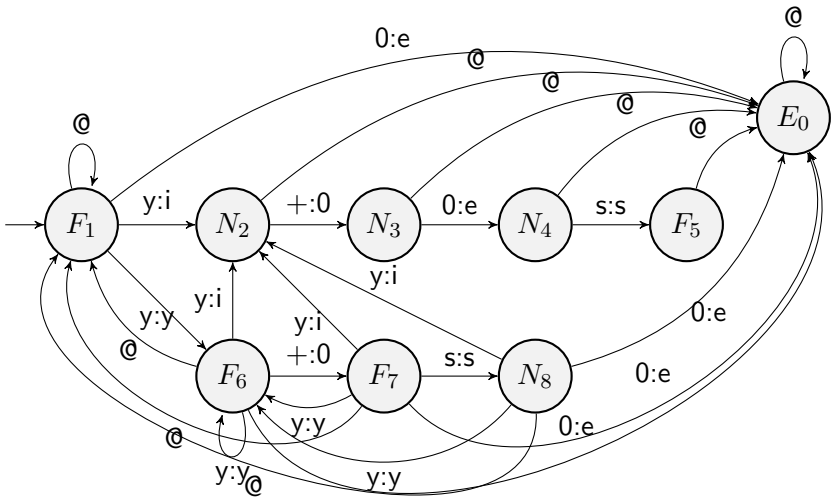
Algorithm Example

- Every letter corresponds to itself
- In addition: `y:i`, `+:0`, `0:e`
- Input: *babies*
- Try inserting lexical + (`+:0`) … blocked by lexicon (no word starts like that)
- Try `b:b` … OK (neither lexicon nor the transducers object)
- `b:b +:0` … lexicon error
- `b:b a:a` … OK
- `b:b a:a +:0` … lexicon error
- `b:b a:a b:b` … OK
- `b:b a:a b:b +:0` … l. error
- `b:b a:a b:b i:i` … l. error

- `b:b a:a b:b y:i` … OK
- … `b:b y:i +:0` … OK
- … `b:b y:i +:0 +:0` … error
- … `y:i e:e` … error
- … `y:i 0:e` … O
- … `y:i +:0 e`
- … `y:i +:0`
- … `y:i`
- … `y:i`
- … `+:0`
- … `y:i`
- … `+:0 0:e`
- … `0:e +:0 s`

One of the hypotheses could be blocked by our FSTs if we designed them better (⇔)

Fixed and Merged FST



Czech Examples

- *skrýš* "hideaway" — genitive *skrýš+e* → *skrýše*
- *káď* "tun" — genitive *káď+e* → *kádě*

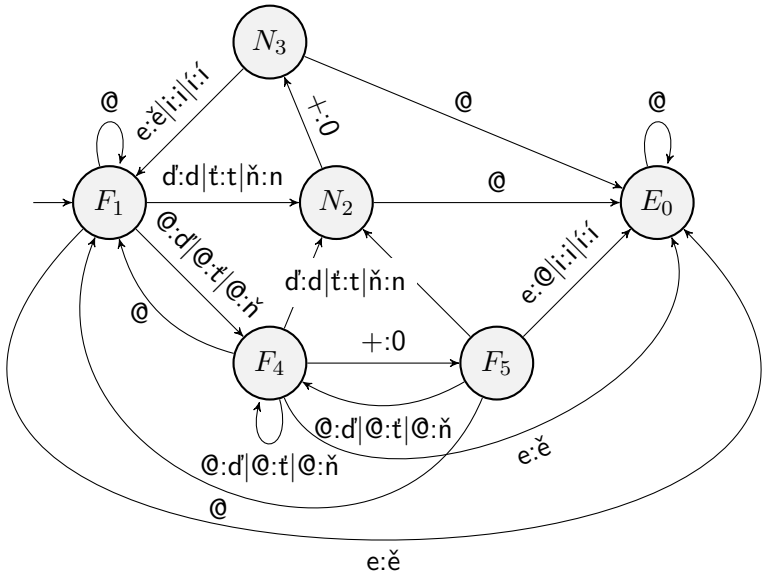
- *ď* and *e* normally cannot occur together...
- ... unless they come from separate morphemes (stem + suffix)!
- We need a rule that will ensure the correct conversion ďe → dě.

```
k á ď + e
k á d 0 ě
```

Example of Transducer: ď, ť, ň on morpheme boundary

- `ď:d +:0 e:ě` is correct, other possibilities are not.
- Assumption: *ďe*, *ďi* could only occur on morpheme boundary (otherwise it is in the lexicon $\Rightarrow$ it should be correct).
- We don't cover *ďě*. If the character *ě* occurs in a suffix, it must be because of phonology:
  - *brzda brzďe (brzdě), žena žeňe (ženě), máta máťe (mátě), máma mámňe (mámě), bába bábje (bábě), lípa lípje (lípě), chůva chůvje (chůvě), matka matce, váha váze, sprcha sprše, kůra kůře, mula mule, vosa vose, lůza lůze*
- We don't cover *ďy* here (which could arise when inflecting a noun ending in *-ďa*; it is incorrect and should be changed to *-di*).

Example of Transducer: ď, ť, ň on morpheme boundary



Czech Feminine Noun Consonant Changes

The pairs illustrate various stem-final changes in the paradigm *žena* of Czech feminine nouns. All words are **surface** strings—nominative singular on the left, dative singular on the right.

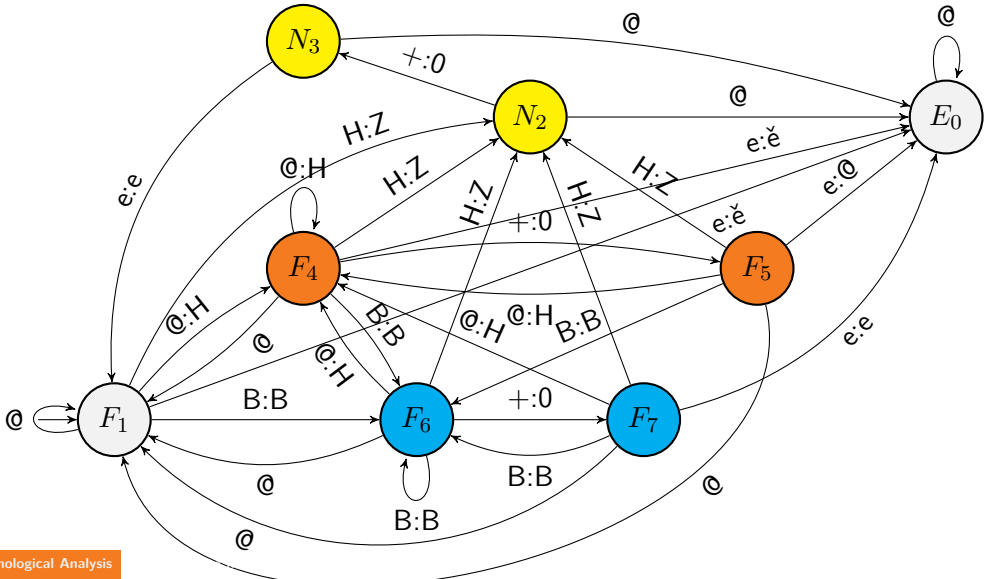
- *váha – váze* “weight”
- *sprcha – sprše* “shower”
- *matka – matce* “mother”
- *kůra – kůře* “bark”
- *Olga – Olze* “Olga”

- *vláda – vládě* “government”
- *máta – mátě* “mint”
- *žena – ženě* “woman”

- *bába – bábě* “old woman”
- *karafa – karafě* “carafe”
- *máma – mámě* “mom”
- *chrpa – chrpě* “cornflower”
- *jíva – jívě* “goat willow”

- *Naďa – Naďě* “Naďa”
- *Jíťa – Jítě* “Jíťa”
- *Áňa – Áně* “Áňa”

Czech Feminine Noun Consonant Changes



H:Z =
g:z | h:z | ch:š | k:c | r:ř

B:B =
b:b | f:f | m:m | p:p | v:v | w:w | q:q | d:d | t:t | n:n | ď:d | ť:t | ň:n

Long-Distance Dependencies

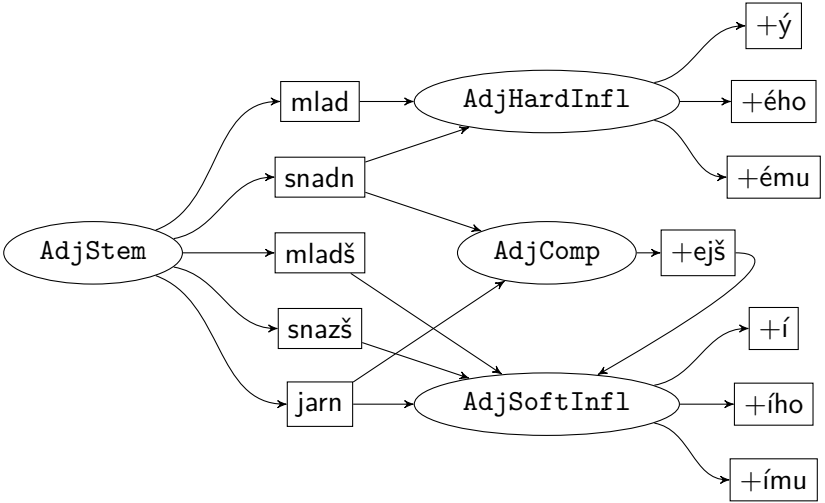
Disadvantage of finite-state morphology:

- Capturing of long-distance dependencies is clumsy!

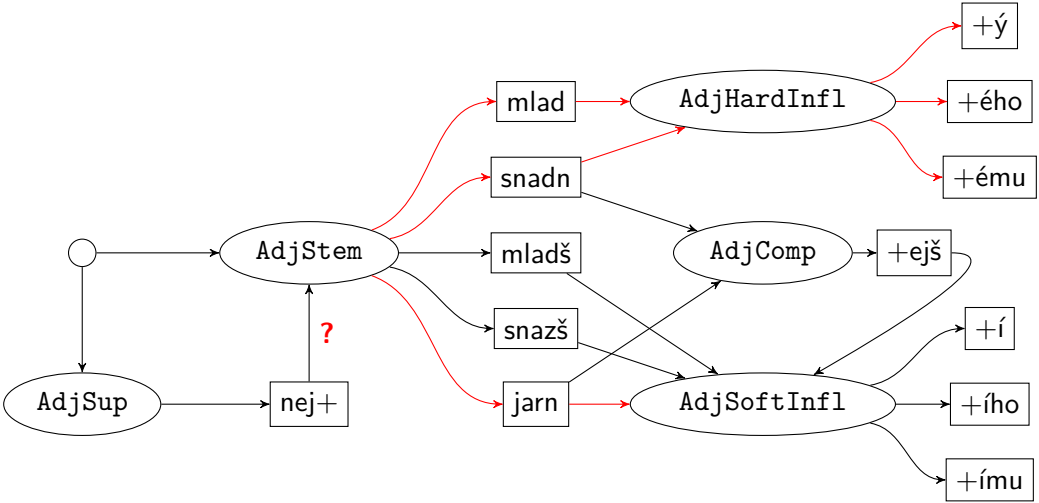
Long-Distance Dependencies: Czech Adjectives

- Two inflection classes:
  - Hard: *černý* "black", *černého*, *černému*, ..., *černá* [Fem], *černé*...
  - Soft: *jarní* "spring", *jarního*, *jarnímu*, ..., *jarní* [Fem], *jarní*...
- Regular comparative:
  - Suffix *-ejš*
  - Comparative is always soft regardless the original class:
    černější, *černějšího*, *černějšímu*, ..., *jarnější*, *jarnějšího*, *jarnějšímu*...
- Irregular comparatives:
  - *mladý* "young" ⇒ *mladší*
  - *snadný* "easy" ⇒ *snadnější* | *snazší*
- Superlative = *nej-* + comparative:
  - *nejmladší* "youngest"
  - **We must remember the prefix until, indefinitely later, we see the suffix!**

Czech Adjectives without Superlative



Czech Adjectives including Superlative



Czech Adjectives including Superlative