Computational morphology. Day 3. Real-world morphology.

Alexey Sorokin[1,2]

[1]Moscow State University, [2]Moscow Institute of Science and Technology

European Summer School
in Logic, Language and Information,
Toulouse, 24-28 July, 2017

Day 3 outline

- Real-world linguistic phenomena in FOMA.

Day 3 outline

- Real-world linguistic phenomena in FOMA.
- Morphological tagging: problem setting.

Day 3 outline

- Real-world linguistic phenomena in FOMA.
- Morphological tagging: problem setting.
- N-gram language models.

Two-level morphology

- Finite-state morphology deals well with concatenative morphology.
- Ideally: agglutinative languages (Turkish, Finnish, etc.).

Two-level morphology

- Finite-state morphology deals well with concatenative morphology.
- Ideally: agglutinative languages (Turkish, Finnish, etc.).
- General two-level scheme:
  - Create the slots for prototypical morphemes.

Two-level morphology

- Finite-state morphology deals well with concatenative morphology.
- Ideally: agglutinative languages (Turkish, Finnish, etc.).
- General two-level scheme:
  - Create the slots for prototypical morphemes.
  - Fill these slots with appropriate morphemes according to morphotactics and phonology.

Two-level morphology

- Finite-state morphology deals well with concatenative morphology.
- Ideally: agglutinative languages (Turkish, Finnish, etc.).
- General two-level scheme:
    - Create the slots for prototypical morphemes.
    - Fill these slots with appropriate morphemes according to morphotactics and phonology.
- Case study: Turkish verb inflection.

Two-level morphology

- Finite-state morphology deals well with concatenative morphology.
- Ideally: agglutinative languages (Turkish, Finnish, etc.).
- General two-level scheme:
  - Create the slots for prototypical morphemes.
  - Fill these slots with appropriate morphemes according to morphotactics and phonology.
- Case study: Turkish verb inflection.
- Categories to model:
  - Voice: passive, active.
  - Tense: aorist, continuous.

Two-level morphology

- Finite-state morphology deals well with concatenative morphology.
- Ideally: agglutinative languages (Turkish, Finnish, etc.).
- General two-level scheme:
  - Create the slots for prototypical morphemes.
  - Fill these slots with appropriate morphemes according to morphotactics and phonology.
- Case study: Turkish verb inflection.
- Categories to model:
  - Voice: passive, active.
  - Tense: aorist, continuous.
  - Number: singular, plural.

Two-level morphology

- Finite-state morphology deals well with concatenative morphology.
- Ideally: agglutinative languages (Turkish, Finnish, etc.).
- General two-level scheme:
  - Create the slots for prototypical morphemes.
  - Fill these slots with appropriate morphemes according to morphotactics and phonology.
- Case study: Turkish verb inflection.
- Categories to model:
  - Voice: passive, active.
  - Tense: aorist, continuous.
  - Number: singular, plural.
  - Person: 1, 2, 3.

Turkish verb conjugation

- Input format: infinitive+Voice+Tense+Person+Number.

Turkish verb conjugation

- Input format: infinitive+Voice+Tense+Person+Number.
- +Voice: +Pass/+Act.
- +Tense: +Aor/+Cont.
- +Person: +1/+2/+3.
- +Number: +Sg/+Pl.

Turkish verb conjugation

- Input format: infinitive+Voice+Tense+Person+Number.
- +Voice: +Pass/+Act.
- +Tense: +Aor/+Cont.
- +Person: +1/+2/+3.
- +Number: +Sg/+Pl.
- Verb form structure:

$$\langle \text{stem} \rangle \langle \text{VoiceSuf} \rangle \langle \text{Tense} \rangle \langle \text{PersNumSuf} \rangle$$

Turkish verb conjugation

- Verb form structure: $\langle\text{stem}\rangle\langle\text{VoiceSuf}\rangle\langle\text{Tense}\rangle\langle\text{PersNumSuf}\rangle$.

Turkish verb conjugation

- Verb form structure: $\langle\text{stem}\rangle\langle\text{VoiceSuf}\rangle\langle\text{Tense}\rangle\langle\text{PersNumSuf}\rangle$.
- Passive voice suffix: *-n* after vowels, *-In* after *l*, *-Il* otherwise.

Turkish verb conjugation

- Verb form structure: $\langle \mathrm{stem} \rangle \langle \mathrm{VoiceSuf} \rangle \langle \mathrm{Tense} \rangle \langle \mathrm{PersNumSuf} \rangle$.
- Passive voice suffix: *-n* after vowels, *-In* after *l*, *-Il* otherwise.
- Aorist suffix:
  - *-r* after vowels.
  - *-Ir* after consonants in polysyllabic stems.
  - *-Ar* after consonants in monosyllabic stems.
  - *-Ir* after 13 monosyllabic exceptions.

Turkish verb conjugation

- Verb form structure: $\langle\text{stem}\rangle\langle\text{VoiceSuf}\rangle\langle\text{Tense}\rangle\langle\text{PersNumSuf}\rangle$.
- Passive voice suffix: *-n* after vowels, *-In* after *l*, *-Il* otherwise.
- Aorist suffix:
  - *-r* after vowels.
  - *-Ir* after consonants in polysyllabic stems.
  - *-Ar* after consonants in monosyllabic stems.
  - *-Ir* after 13 monosyllabic exceptions.
- Progressive suffix:
  - *-Iyor* after consonants.
  - *-yor* after u, ü, i, ı.
  - *-Iyor* after vowels, the vowel is removed.
  - *-iyor* after roots de-/ye-, the vowel is removed.

Turkish verb conjugation

- Verb form structure: ⟨stem⟩⟨VoiceSuf⟩⟨Tense⟩⟨PersNumSuf⟩.
- Passive voice suffix: -*n* after vowels, -*In* after *l*, -*Il* otherwise.
- Aorist suffix:
  - -*r* after vowels.
  - -*Ir* after consonants in polysyllabic stems.
  - -*Ar* after consonants in monosyllabic stems.
  - -*Ir* after 13 monosyllabic exceptions.
- Progressive suffix:
  - -*Iyor* after consonants.
  - -*yor* after u, ü, i, ı.
  - -*Iyor* after vowels, the vowel is removed.
  - -*iyor* after roots de-/ye-, the vowel is removed.
- Verb ending (⟨PersNumSuf⟩):

Number \ Person	1	2	3
Singular	-Im	-sIn	-∅
Plural	-Iz	-sInIz	-lAr

Turkish verb conjugation

1 step: defining slots

```
define Voice "+Act" | "+Pass" ;
define Tense "+Aor" | "+Prog";
define Number "+Sg" | "+Pl" ;
define Person "+1" | "+2" | "+3" ;
define Input Infinitive Voice Tense Person Number;
# deleting -mAk and defining slots
define MarkerInsertion [..] -> "!" || _ m [a | e] k Voice ;
define InfinitiveDeletion m [ a | e ] k -> "" || "!" _ ;
define TensePattern [ [..] -> "!AorSuffix!" || "!" _ ?+ "+Aor" ] .o. [ [..] -> "!ProgSuffix!"
    || "!" _ ?+ "+Prog" ];
define PassivePattern [..] -> "!PassSuffix!" || "!" _ ?+ "+Pass" ;
define Cleanup [ Voice | Tense | "!" ] -> "" ;
```

```
$ flookup -i -w "" turkish_diacr.bin < test.in
okumak+Pass+Prog+1+Pl    oku!PassSuffix!!ProgSuffix!+1+Pl
gelmek+Pass+Aor+2+Sg     gel!PassSuffix!!AorSuffix!+2+Sg
uyumak+Act+Prog+3+Pl     uyu!ProgSuffix!+3+Pl
izlemek+Act+Prog+3+Pl    izle!ProgSuffix!+3+Pl
bilmek+Act+Aor+2+Pl      bil!AorSuffix!+2+Pl
görmek+Act+Aor+2+Pl      gör!AorSuffix!+2+Pl
```

Turkish verb conjugation

2 step: filling voice

```
## passive suffix filling
define Passive1 "!PassSuffix!" -> I l || [ Consonant - l ] _ ;
define Passive2 "!PassSuffix!" -> I n || l _ ;
define Passive3 "!PassSuffix!" -> n || Vowel _ ;
define PassiveSuffix Passive1 .o. Passive2 .o. Passive3 ;
```

```
$ flookup -i -w "" turkish_diacr.bin < test.in
okumak+Pass+Prog+1+Pl    okun!ProgSuffix!+1+Pl
gelmek+Pass+Aor+2+Sg     gelIn!AorSuffix!+2+Sg
uyumak+Act+Prog+3+Pl     uyu!ProgSuffix!+3+Pl
izlemek+Act+Prog+3+Pl    izle!ProgSuffix!+3+Pl
bilmek+Act+Aor+2+Pl      bil!AorSuffix!+2+Pl
görmek+Act+Aor+2+Pl      gör!AorSuffix!+2+Pl
```

Turkish verb conjugation

3 step: filling aorist

```
## aorist suffix filling
define PseudoVowel Vowel | I | A ;
read lexc aor_exception.lexc
define AorException;
define Monosyllable Consonant* Vowel Consonant* ;
define AorSuffix0 "!AorSuffix!" -> I r || .#. AorException _ ;
define AorSuffix1 "!AorSuffix!" -> r || PseudoVowel _ ;
define AorSuffix2 "!AorSuffix!" -> A r || .#. Monosyllable _ ;
define AorSuffix3 "!AorSuffix!" -> I r || _ ;
define AorSuffix AorSuffix0 .o. AorSuffix1 .o. AorSuffix2 .o. AorSuffix3 ;
```

```
$ flookup -i -w "" turkish_diacr.bin < test.in
okumak+Pass+Prog+1+Pl    okun!ProgSuffix!+1+Pl
gelmek+Pass+Aor+2+Sg     gelInIr+2+Sg
uyumak+Act+Prog+3+Pl     uyu!ProgSuffix!+3+Pl
izlemek+Act+Prog+3+Pl    izle!ProgSuffix!+3+Pl
bilmek+Act+Aor+2+Pl      bilIr+2+Pl
görmek+Act+Aor+2+Pl      görAr+2+Pl
```

Turkish verb conjugation

4 step: filling progressive

```
## progressive suffix filling
define ProgSuffix0 "!ProgSuffix!" -> "!" I y o r || _ ;
## after i, ı, u, ü
define ProgSuffixVowel0 "!" I -> "" || [ u | ü| i | ı] _ ;
## other vowels
define ProgSuffixVowel1 [ a | o | e | ö] "!" -> "" || _ ;
## demek, yemek
define ProgDemek e "!" I -> i || .#. [ d | y ] _ ;
## sonorization
define ProgSonor t -> d || .#. [ g i | e | t a ] _ "!" ;
define ProgCleanup "!" -> "" || _ ;
define ProgSuffix ProgSuffix0 .o. ProgSuffixVowel0 .o. ProgSuffixVowel1 .o. ProgDemek .
    o. ProgSonor .o. ProgCleanup ;
```

```
$ flookup -i -w "" turkish_diacr.bin < test.in
okumak+Pass+Prog+1+Pl    okunIyor+1+Pl
gelmek+Pass+Aor+2+Sg     gelInIr+2+Sg
uyumak+Act+Prog+3+Pl     uyuyor+3+Pl
izlemek+Act+Prog+3+Pl    izlIyor+3+Pl
bilmek+Act+Aor+2+Pl      bilIr+2+Pl
görmek+Act+Aor+2+Pl      görAr+2+Pl
```

Turkish verb conjugation

5 step: verbal endings

```
## ending filling
define Ending1s "+1" "+Sg" -> I m || _ ;
define Ending2s "+2" "+Sg" -> s I n || _ ;
define Ending3s "+3" "+Sg" -> "" || _ ;
define Ending1p "+1" "+Pl" -> I z || _ ;
define Ending2p "+2" "+Pl" -> s I n I z || _ ;
define Ending3p "+3" "+Pl" -> l A r || _ ;
define Ending Ending1s .o. Ending2s .o. Ending3s .o. Ending1p .o. Ending2p .o.
      Ending3p ;
```

```
$ flookup -i -w "" turkish_diacr.bin < test.in
okumak+Pass+Prog+1+Pl    okunIyorIz
gelmek+Pass+Aor+2+Sg     gelInIrsIn
uyumak+Act+Prog+3+Pl     uyuyorlAr
izlemek+Act+Prog+3+Pl    izlIyorlAr
bilmek+Act+Aor+2+Pl      bilIrsInIz
görmek+Act+Aor+2+Pl      görArsInIz
```

Turkish verb conjugation

6 step: vowel harmony

```
## Vowel Harmony (left context on output size)
define VowelHarmony [A -> a // LastVowelHard _ ,, A -> e // LastVowelSoft _ ,,
                    I -> ı// LastVowelHardStraight _ ,, I -> i //
                        LastVowelSoftStraight _ ,,
                    I -> u // LastVowelHardRound _ ,, I -> ü//
                        LastVowelSoftRound _] ;

define Fill PassiveSuffix .o. AorSuffix .o. ProgSuffix .o. Ending .o. VowelHarmony ;
define Grammar Input .o. Pattern .o. Fill ;
```

```
$ flookup -i -w "" turkish_diacr.bin < test.in
okumak+Pass+Prog+1+Pl    okunuyoruz
gelmek+Pass+Aor+2+Sg     gelinirsin
uyumak+Act+Prog+3+Pl     uyuyorlar
izlemek+Act+Prog+3+Pl    izliyorlar
bilmek+Act+Aor+2+Pl      bilirsiniz
görmek+Act+Aor+2+Pl      görersiniz
```

General model

- Spanish verb conjugation is rather simple:

Number	Person	*-ar* (tomar)	*-er* (comer)	*-ir* (escribir)
Singular	1	tomo	como	escribo
	2	tomas	comes	escribes
	3	toma	come	escribe
Plural	1	tomamos	comemos	escribimos
	2	tomáis	coméis	escribís
	3	toman	comen	escriben

General model

- Spanish verb conjugation is rather simple:

Number	Person	*-ar* (tomar)	*-er* (comer)	*-ir* (escribir)
Singular	1	tomo	como	escribo
	2	tomas	comes	escribes
	3	toma	come	escribe
Plural	1	tomamos	comemos	escribimos
	2	tomáis	coméis	escribís
	3	toman	comen	escriben

- There are several morphonetic alterations:
  - In +1+Sg *g* becomes *j* before *-er*: emerger → emerjo.

General model

- Spanish verb conjugation is rather simple:

Number	Person	*-ar* (tomar)	*-er* (comer)	*-ir* (escribir)
Singular	1	tomo	como	escribo
	2	tomas	comes	escribes
	3	toma	come	escribe
Plural	1	tomamos	comemos	escribimos
	2	tomáis	coméis	escribís
	3	toman	comen	escriben

- There are several morphonetic alterations:
  - In +1+Sg *g* becomes *j* before *-er*: emerger → emerjo.
  - In +1+Sg *c* turns to *zc* before *-er*/*-ir* and after vowel: *conducir* → *conduzco*, *agradecer* → *agradezco* (though *mecer* → *mezo*).

Model alterations

- Spanish verb conjugation is rather simple.
- But model vowel alterations exist:

Number	Person	*-o-/-ue-* *contar*	*-e-/-ie-* *sentir*	*-e-/-i-* *servir*
Singular	1	cuento	siento	sirvo
	2	cuentas	sientes	sirves
	3	cuenta	siente	sirve
Plural	1	contamos	sentimos	servimos
	2	contáis	sentís	servís
	3	cuentan	sienten	sirven

Model alterations

- Spanish verb conjugation is rather simple.
- But model vowel alterations exist:

Number	Person	*-o-/-ue-* *contar*	*-e-/-ie-* *sentir*	*-e-/-i-* *servir*
Singular	1	cuento	siento	sirvo
	2	cuentas	sientes	sirves
	3	cuenta	siente	sirve
Plural	1	contamos	sentimos	servimos
	2	contáis	sentís	servís
	3	cuentan	sienten	sirven

- These classes include much more verbs:
  - *-o-/-ue-*: *morir*, *dormir*, *soler*, *soñar*, ...
  - *-e-/-ie-*: *pensar*, *entender*, *perder*, *preferir*, ...

Model alterations

- Spanish verb conjugation is rather simple.
- But model vowel alterations exist:

Number	Person	*-o-/-ue-* *contar*	*-e-/-ie-* *sentir*	*-e-/-i-* *servir*
Singular	1	cuento	siento	sirvo
	2	cuentas	sientes	sirves
	3	cuenta	siente	sirve
Plural	1	contamos	sentimos	servimos
	2	contáis	sentís	servís
	3	cuentan	sienten	sirven

- These classes include much more verbs:
  - *-o-/-ue-*: *morir*, *dormir*, *soler*, *soñar*, ...
  - *-e-/-ie-*: *pensar*, *entender*, *perder*, *preferir*, ...
  - *-e-/-i-*: *pedir*, *vestir*, *elegir*, *expedir*, ...

Model alterations

- Also Spanish has some irregular verbs:

Number	Person	*estar*	*ser*	*haber*
Singular	1	estoy	soy	he
	2	estás	eres	has
	3	está	es	ha
Plural	1	estamos	somos	hemos
	2	estáis	sois	habéis
	3	están	son	han

Model alterations

- Also Spanish has some irregular verbs:

Number	Person	*estar*	*ser*	*haber*
Singular	1	estoy	soy	he
	2	estás	eres	has
	3	está	es	ha
Plural	1	estamos	somos	hemos
	2	estáis	sois	habéis
	3	están	son	han

- There are some more irregular verbs: *decir*, *dar*, *ver*, . . .
- Some verbs just have irregular +1+Sg forms:
  - *traer* → *traigo* (also *caer*).
  - *valer* → *valgo* (also *salir*, *poner*).
  - *saber* → *sé*, *caber* → *quepo*.

Model alterations

- Also Spanish has some irregular verbs:

Number	Person	*estar*	*ser*	*haber*
Singular	1	estoy	soy	he
	2	estás	eres	has
	3	está	es	ha
Plural	1	estamos	somos	hemos
	2	estáis	sois	habéis
	3	están	son	han

- There are some more irregular verbs: *decir*, *dar*, *ver*, ...
- Some verbs just have irregular +1+Sg forms:
  - *traer* → *traigo* (also *caer*).
  - *valer* → *valgo* (also *salir*, *poner*).
  - *saber* → *sé*, *caber* → *quepo*.
- How to model that all properly?

Regular model

First, model regular verbs (with regular phonetic alterations):

```
define Vowel e | i | é| í| a | u | o | á| ú| ó;
define Cons b | c | d | f | g | h | j | k | l | m | n | ñ| p | q | r | s | t | v | x | y | z ;
define Letter Cons | Vowel ;
define Stem Letter* Vowel Letter* ;
define InfSuffix [ a | i | e ] r ;
define Infinitive Stem InfSuffix ;
define Number "+Sg" | "+Pl" ;
define Person "+1" | "+2" | "+3" ;
define Input Infinitive Number Person;
## phonetic alterations
define ChangeEndCons1 c -> z c || Vowel _ [ e | i ] r "+Sg" "+1";
define ChangeEndCons2 c -> z || [ Cons - z ] _ [ e | i ] r "+Sg" "+1" ;
define ChangeEndCons3 g -> j, g u -> g, q u -> c || _ [ e | i ] r "+Sg" "+1" ;
define UIR [..] -> y || [ Letter - q ] u _ i r [ "+Sg" | "+Pl" "+3" ] ;
define ChangeEnd ChangeEndCons1 .o. ChangeEndCons2 .o. ChangeEndCons3 .o. UIR ;
## endings
define ieInfSuffix [ i | e ] r ;
define PresEnding1s InfSuffix -> o || _ "+Sg" "+1" ;
define PresEnding2s a r -> a s, ieInfSuffix -> e s || _ "+Sg" "+2" ;
define PresEnding3s a r -> a, ieInfSuffix -> e || _ "+Sg" "+3" ;
define PresEnding1p r -> m o s || _ "+Pl" "+1" ;
define PresEnding2p a r -> ái s, e r -> éi s, i r -> ís || _ "+Pl" "+2" ;
define PresEnding3p a r -> a n, ieInfSuffix -> e n || _ "+Pl" "+3" ;
define PresEnding PresEnding1s .o. PresEnding2s .o. PresEnding3s .o. PresEnding1p .o. PresEnding2p .o. PresEnding3p ;
## combining all
define CleanUp [ Person | Number ] -> "" || _ ;
define Regular [ Input .o. ChangeEnd .o. PresEnding ] ;
define Grammar [ IrregularForm .P. Regular ] .o. CleanUp ;
```

Lexicon file

Exceptions are listed in the lexicon file:

```
Multichar_Symbols +Sg +Pl +1 +2 +3

LEXICON Root

Verb ; Sg1Verb ;

LEXICON Verb

estar+Sg+1:estoy #;
estar+Sg+2:estás #;
estar+Sg+3:está #;
estar+Pl+3:están #;

ser+Sg+1:soy #;
ser+Sg+2:eres #;
ser+Sg+3:es #;
ser+Pl+1:somos #;
ser+Pl+2:sois #;
ser+Pl+3:son #;

haber+Sg+1:he #;
haber+Sg+2:has #;
haber+Sg+3:has #;
haber+Pl+3:han #;

LEXICON Sg1Verb

saber+Sg+1:sé #;
traer+Sg+1:traigo #;
caer+Sg+1:caigo #;
caber+Sg+1:quepo #;
poner+Sg+1:pongo #;
valer+Sg+1:valgo #;
salir+Sg+1:salgo #;
```

Spanish: stem alterations

Regular model: application

```
$ flookup -i -w "" spanish.bin < spanish_test.in
```

caer+Sg+1	caigo	comer+Sg+3	come
ser+Pl+1	somos	correr+Pl+2	corréis
ser+Pl+2	sois	vender+Pl+3	venden
ser+Sg+1	soy	escribir+Sg+2	escribes
estar+Pl+3	están	surgir+Pl+1	surgimos
estar+Sg+2	estás	destruir+Pl+3	destruyen
estar+Sg+3	está	instruir+Sg+2	instruyes
hablar+Sg+1	hablo	cojer+Sg+1	cojo
hablar+Sg+2	hablas	distinguir+Sg+1	distingo
cantar+Pl+1	cantamos	conducir+Sg+1	conduzco

Spanish: stem alterations

- Stem alterations occur simultaneously in several forms (all singular and +Pl+3).
- It is inconvenient to write in the lexicon all alterations.

Spanish: stem alterations

- Stem alterations occur simultaneously in several forms (all singular and +Pl+3).
- It is inconvenient to write in the lexicon all alterations.
- Moreover, after stem alterations stems are subject to usual phonological rules:
  - *elegir*+Sg+1 → *elijo*
  - *seguir*+Sg+1 → *sigo* (not **siguo*).

Spanish: stem alterations

- Stem alterations occur simultaneously in several forms (all singular and +Pl+3).
- It is inconvenient to write in the lexicon all alterations.
- Moreover, after stem alterations stems are subject to usual phonological rules:
  - *elegir*+Sg+1 → *elijo*
  - *seguir*+Sg+1 → *sigo* (not **siguo*).
- In stem alteration branch we compose stem alteration with phonological changes.
- In regular branch only phonological changes are applied.

Spanish: stem alterations

- Stem alterations occur simultaneously in several forms (all singular and +Pl+3).
- It is inconvenient to write in the lexicon all alterations.
- Moreover, after stem alterations stems are subject to usual phonological rules:
    - *elegir*+Sg+1 → *elijo*
    - *seguir*+Sg+1 → *sigo* (not **siguo*).
- In stem alteration branch we compose stem alteration with phonological changes.
- In regular branch only phonological changes are applied.
- This is **lenient composition**:

$$X.O.Y = (X.o.Y).P.Y$$

Spanish: stem alterations

- Stem alterations occur simultaneously in several forms (all singular and +Pl+3).
- It is inconvenient to write in the lexicon all alterations.
- Moreover, after stem alterations stems are subject to usual phonological rules:
  - *elegir*+Sg+1 → *elijo*
  - *seguir*+Sg+1 → *sigo* (not **siguo*).
- In stem alteration branch we compose stem alteration with phonological changes.
- In regular branch only phonological changes are applied.
- This is **lenient composition**:

$$X.O.Y = (X.o.Y).P.Y$$

- But we use priority union instead.

Spanish: stem alterations

- We have two alteration branches:
    - First inserts -*(i)g*- before ending of exceptional +Sg+1 forms: (*caer*+Sg+1 → *caigo*, *salir*+Sg+1 → *salgo*).
    - Second deals with stem vowel change (-*o*-/-*ue*-, -*e*-/-*ie*-, -*e*-/-*i*-).
- First branch has higher priority: (*tener+Sg+1* → *tengo*, but *tener+Sg+2* → *tienes*, *tener+Sg+3* → *tiene*).

Spanish: stem alterations

- We have two alteration branches:
  - First inserts -*(i)g*- before ending of exceptional +Sg+1 forms: (*caer*+Sg+1 → *caigo*, *salir*+Sg+1 → *salgo*).
  - Second deals with stem vowel change (-*o*-/-*ue*-, -*e*-/-*ie*-, -*e*-/-*i*-).
- First branch has higher priority: (*tener+Sg+1* → *tengo*, but *tener+Sg+2* → *tienes*, *tener+Sg+3* → *tiene*).
- Not to deal with pseudoforms as **traiger* we replace ending with special symbol:

```
!!!first_stem.lexc!!!
LEXICON Root
traer:traiG%!Ending2%! #;
salir:salG%!Ending3%! #;
```

Spanish: stem alterations

- We have two alteration branches:
  - First inserts *-(i)g-* before ending of exceptional +Sg+1 forms: (*caer*+Sg+1 → *caigo*, *salir*+Sg+1 → *salgo*).
  - Second deals with stem vowel change (*-o-/-ue-*, *-e-/-ie-*, *-e-/-i-*).
- First branch has higher priority: (*tener+Sg+1* → *tengo*, but *tener+Sg+2* → *tienes*, *tener+Sg+3* → *tiene*).
- Not to deal with pseudoforms as **traiger* we replace ending with special symbol:

```
!!!first_stem.lexc!!!
LEXICON Root
traer:traiG%!Ending2%! #;
salir:salG%!Ending3%! #;
```

- Analogously for second branch (*dorm-* → *duerm-*):

```
!!!second_stem.lexc!!!
LEXICON Root
tener:tien%!Ending2%! #;
pedir:pid%!Ending2%! #;
```

Spanish: stem alterations

- Verb endings are replaced by markers (rules are changed accordingly):

```
define Marker [ a r ] -> "!Ending1!" , [ e r ] -> "!Ending2!" ,
    [ i r ] -> "!Ending3!" || _ Number ;
```

- Stem transformations are read from lexicons:

```
## lexicon for stem changes
read lexc first_stem.lexc
define FirstStem ;
define FirstStemChange FirstStem "+Sg" "+1" ;
read lexc second_stem.lexc
define SecondStem ;
define SecondStemChange SecondStem ["+Sg" ? | "+Pl" "+3" ] ;
define IrregularStemChange FirstStemChange .P. SecondStemChange ;
```

Spanish: stem alterations

- Verb endings are replaced by markers (rules are changed accordingly):

```
define Marker [ a r ] −> "!Ending1!" , [ e r ] −> "!Ending2!" ,
    [ i r ] −> "!Ending3!" || _ Number ;
```

- Stem transformations are read from lexicons:

```
## lexicon for stem changes
read lexc first_stem.lexc
define FirstStem ;
define FirstStemChange FirstStem "+Sg" "+1" ;
read lexc second_stem.lexc
define SecondStem ;
define SecondStemChange SecondStem ["+Sg" ? | "+Pl" "+3" ] ;
define IrregularStemChange FirstStemChange .P. SecondStemChange ;
```

- In the end everything is combined by priority union:

```
define Regular [ Input .o. [IrregularStemChange .P. Marker ] .o. ChangeEnd .o.
    PresEnding ] ;
```

Spanish: stem alterations

- Stem alterations work indeed:

```
$ flookup -i -w "" spanish_full.bin < spanish_stem.in

detraer+Sg+1     detraigo           pensar+Pl+1      pensamos
tener+Pl+1       tenemos            morir+Sg+3       muere
tener+Pl+2       tenéis             morir+Pl+2       morís
tener+Sg+1       tengo              pedir+Pl+3       piden
dormir+Pl+3      duermen            pedir+Sg+2       pides
dormir+Sg+2      duermes            preferir+Pl+1    preferimos
hacer+Sg+1       hago               preferir+Pl+3    prefieren
hacer+Sg+3       hace               preferir+Sg+1    prefiero
pensar+Sg+1      pienso             decir+Sg+3       dice
pensar+Sg+2      piensas            preferir+Sg+1    prefiero
```

Spanish: stem alterations

- Stem alterations work indeed:

```
$ flookup -i -w "" spanish_full.bin < spanish_stem.in

detraer+Sg+1    detraigo        pensar+Pl+1     pensamos
tener+Pl+1      tenemos         morir+Sg+3      muere
tener+Pl+2      tenéis          morir+Pl+2      morís
tener+Sg+1      tengo           pedir+Pl+3      piden
dormir+Pl+3     duermen         pedir+Sg+2      pides
dormir+Sg+2     duermes         preferir+Pl+1   preferimos
hacer+Sg+1      hago            preferir+Pl+3   prefieren
hacer+Sg+3      hace            preferir+Sg+1   prefiero
pensar+Sg+1     pienso          decir+Sg+3      dice
pensar+Sg+2     piensas         preferir+Sg+1   prefiero
```

- Should be added: derivatonal prefixes.
  - *tener* → *contener*, *mantener*, *detener*, ...
  - *hacer* → *rehacer*, *deshacer*, ...

Spanish: fusion

- +1+Sg form once more:

Infinitive	+1+Sg	gerund
partir	parto	partiendo
imbuir	imbuyo	imbuyendo
destruir	destruyo	destruyendo
delinquir	delinco	delinquiendo
distinguir	distingo	distinguiendo
coger	cojo	cogiendo
agradecer	agradezco	agradeciendo
mecer	mezo	meciendo

Spanish: fusion

- +1+Sg form once more:

Infinitive	+1+Sg	gerund
partir	parto	partiendo
imbuir	imbuyo	imbuyendo
destruir	destruyo	destruyendo
delinquir	delinco	delinquiendo
distinguir	distingo	distinguiendo
coger	cojo	cogiendo
agradecer	agradezco	agradeciendo
mecer	mezo	meciendo

- Personal ending fuses with the stem on morpheme boundary.
- That could be carefully modeled with context “phonetic” rules.

Arabic: root-and-pattern morphology

- So far morpheme structure was linear.

Arabic: root-and-pattern morphology

- So far morpheme structure was linear.
- That is not true for Semitic languages (e.g. Arabic):

kataba	"(he) wrote+Perf"
kattabat	"(she intensively) wrote+Perf"
yaktubu	"(he) was written+Imp"
takattibu	"(she) was (intensively) written+Imp"

Arabic: root-and-pattern morphology

- So far morpheme structure was linear.
- That is not true for Semitic languages (e.g. Arabic):

kataba	"(he) wrote+Perf"
kattabat	"(she intensively) wrote+Perf"
yaktubu	"(he) was written+Imp"
takattibu	"(she) was (intensively) written+Imp"

- Root *k-t-b* consists of consonants (usually 3).
- Vowels reflect grammatical information.

Arabic: root-and-pattern morphology

- So far morpheme structure was linear.
- That is not true for Semitic languages (e.g. Arabic):

kataba	"(he) wrote+Perf"
kattabat	"(she intensively) wrote+Perf"
yaktubu	"(he) was written+Imp"
takattibu	"(she) was (intensively) written+Imp"

- Root *k-t-b* consists of consonants (usually 3).
- Vowels reflect grammatical information.
- Different verb classes have different vowel patterns:

marida	"(he became) ill+Perf"
marradat	"(she intensively became) ill+Perf"
yamradu	"(he) was made ill+Imp"
tamarridu	"(she) was (intensively) made ill+Imp"

Arabic: simple example

- We want to model something like:

$$\langle\text{stem}\rangle\langle\text{Type}\rangle\langle\text{Voice}\rangle\langle\text{Aspect}\rangle\langle\text{Person}\rangle\langle\text{Gender}\rangle \mapsto \langle\text{wordForm}\rangle$$

Arabic: simple example

- We want to model something like:

  $$\langle \text{stem} \rangle \langle \text{Type} \rangle \langle \text{Voice} \rangle \langle \text{Aspect} \rangle \langle \text{Person} \rangle \langle \text{Gender} \rangle \mapsto \langle \text{wordForm} \rangle$$

- Possible values:
  - $\langle \text{Type} \rangle \in \{I, II\}$,
  - $\langle \text{Voice} \rangle \in \{\mathsf{Act}, \mathsf{Pass}\}$,
  - $\langle \text{Aspect} \rangle \in \{\mathsf{Perf}, \mathsf{Imperf}\}$,
  - $\langle \text{Person} \rangle \in \{3\}$,
  - $\langle \text{Gender} \rangle \in \{\mathsf{M}, \mathsf{F}\}$.
- 16 variants.

Arabic: simple example

- We want to model something like:

  $$\langle\text{stem}\rangle\langle\text{Type}\rangle\langle\text{Voice}\rangle\langle\text{Aspect}\rangle\langle\text{Person}\rangle\langle\text{Gender}\rangle \mapsto \langle\text{wordForm}\rangle$$

- Possible values:
  - $\langle\text{Type}\rangle \in \{I, II\}$,
  - $\langle\text{Voice}\rangle \in \{\mathsf{Act}, \mathsf{Pass}\}$,
  - $\langle\text{Aspect}\rangle \in \{\mathsf{Perf}, \mathsf{Imperf}\}$,
  - $\langle\text{Person}\rangle \in \{3\}$,
  - $\langle\text{Gender}\rangle \in \{\mathsf{M}, \mathsf{F}\}$.
- 16 variants.
- We model only one class (of the verb *KTB* "to write").

Arabic: word formation

- Word formation in Arabic (A. A. Zalizniak's handout):
- Stem variants:

Type	Pattern	Example
I (basic)	K-T-B	kataba "to write"
II (intensive)	K-TT-B	kattaba "to write a lot"

- Prefix/suffix variants:

Person+Gender	Perf. suffix	Imp. prefix-suffix
+3+Masc	-a	ya- -u
+3+Fem	-at	ta- -u

- Vowel filler variants:

Aspect	Voice	Prefix	Filler I	Filler II
Perfect	Active		a-a	a-a
Perfect	Passive		u-i	u-i
Imperfect	Active	ya-	∅-u	a-i
Imperfect	Passive	yu-	∅-a	a-a

Arabic conjugation in FOMA: input

- Input format:

```
define Vowel [ a | i | u ];
define Consonant [ k | t | b | z | h | r | s | f | m | d | n | y];
define Letter [ Vowel | Consonant ];
define Stem Consonant Consonant Consonant;
define Type [ "+I" | "+II" ];
define Voice ["+Act" | "+Pass"];
define Aspect ["+Perf" | "+Imperf"];
define Person "+3";
define Gender ["+M" | "+F"];
define Input Stem Type Voice Aspect Person Gender;
```

Arabic conjugation in FOMA: input

- Input format:

```
define Vowel [ a | i | u ];
define Consonant [ k | t | b | z | h | r | s | f | m | d | n | y];
define Letter [ Vowel | Consonant ];
define Stem Consonant Consonant Consonant;
define Type [ "+I" | "+II" ];
define Voice ["+Act" | "+Pass"];
define Aspect ["+Perf" | "+Imperf"];
define Person "+3";
define Gender ["+M" | "+F"];
define Input Stem Type Voice Aspect Person Gender;
```

- Vowel positions are marked with digits:

```
define 0Insertion [..] -> "0" || .#. _ ;
define 1Insertion [..] -> "1" || "0" Consonant _ ;
define 2Insertion [..] -> "2" || "1" Consonant _ ;
define 3Insertion [..] -> "3" || "2" Consonant _ ;
define PosInsertion 0Insertion .o. 1Insertion .o. 2Insertion .o. 3Insertion;
```

Arabic conjugation in FOMA: fillers

- Doubling second consonant of intensive:

```
define CheckTypeI ?* "+I" ?*;
define CheckTypeII ?* "+II" ?*;
define TypeIIDuplication k -> [k k], b -> [b b], t -> [t t], z -> [z z], h
    -> [h h], r -> [r r], s -> [s s], f -> [f f], m -> [m m], d -> [d d], n
    -> [n n] || _ "2";
define StemProcessing [ CheckTypeI ] | [ CheckTypeII .o. TypeIIDuplication ];
```

Arabic conjugation in FOMA: fillers

- Doubling second consonant of intensive:

```
define CheckTypeI ?* "+I" ?*;
define CheckTypeII ?* "+II" ?*;
define TypeIIDuplication k -> [k k], b -> [b b], t -> [t t], z -> [z z], h
    -> [h h], r -> [r r], s -> [s s], f -> [f f], m -> [m m], d -> [d d], n
    -> [n n] || _ "2";
define StemProcessing [ CheckTypeI ] | [ CheckTypeII .o. TypeIIDuplication ];
```

- Defining fillers:

```
define aaFill "1" -> a, "2" -> a;
define aiFill "1" -> a, "2" -> i;
define uiFill "1" -> u, "2" -> i;
define 0aFill "1" -> [], "2"-> a;
define 0uFill "1" -> [], "2"-> u;
```

Arabic conjugation in FOMA: selecting the rule

- Exhaustive search for appropriate rule:

```
define PerfectActiveFill aaFill;
define ImperfectActiveFill [ CheckTypeI .o. 0uFill ] | [ CheckTypeII .o. aiFill ];
define ActiveFill [CheckPerf .o. PerfectActiveFill] | [CheckImperf .o.
    ImperfectActiveFill];
define PerfectPassiveFill uiFill;
define ImperfectPassiveFill [ CheckTypeI .o. 0aFill ] | [ CheckTypeII .o. aaFill ];
define PassiveFill [CheckPerf .o. PerfectPassiveFill] | [CheckImperf .o.
    ImperfectPassiveFill];
define Fill [CheckPass .o. PassiveFill] | [CheckAct .o. ActiveFill] ;
```

Arabic conjugation in FOMA: selecting the rule

- Exhaustive search for appropriate rule:

```
define PerfectActiveFill aaFill;
define ImperfectActiveFill [ CheckTypeI .o. 0uFill ] | [ CheckTypeII .o. aiFill ];
define ActiveFill [CheckPerf .o. PerfectActiveFill] | [CheckImperf .o.
    ImperfectActiveFill];
define PerfectPassiveFill uiFill;
define ImperfectPassiveFill [ CheckTypeI .o. 0aFill ] | [ CheckTypeII .o. aaFill ];
define PassiveFill [CheckPerf .o. PerfectPassiveFill] | [CheckImperf .o.
    ImperfectPassiveFill];
define Fill [CheckPass .o. PassiveFill] | [CheckAct .o. ActiveFill] ;
```

- The same for prefixes (0 marker):

```
define 0Prefix "0" -> [];
define yaPrefix "0" -> y a;
define yuPrefix "0" -> y u;
define taPrefix "0" -> t a;
define tuPrefix "0" -> t u;
define PerfectPrefix 0Prefix;
define ImperfectActivePrefix [CheckMasc .o. yaPrefix] | [CheckFem .o. taPrefix] ;
define ImperfectPassivePrefix [CheckMasc .o. yuPrefix] | [CheckFem .o. tuPrefix] ;
define ImperfectPrefix [CheckAct .o. ImperfectActivePrefix] | [CheckPass .o.
    ImperfectPassivePrefix] ;
define Prefix [CheckPerf .o. PerfectPrefix] | [CheckImperf .o. ImperfectPrefix] ;
```

Arabic conjugation in FOMA: selecting the rule

- Processing the suffixes (3 marker):

```
define ImperfectSuffix "3" -> u || _ Type;
define PerfectMascSuffix "3" -> a || _ Type;
define PerfectFemSuffix "3" -> a t || _ Type;
define PerfectSuffix [ CheckMasc .o. PerfectMascSuffix ] | [CheckFem .o.
    PerfectFemSuffix ] ;
define Suffix [ CheckPerf .o. PerfectSuffix ] | [ CheckImperf .o. ImperfectSuffix ];
```

Arabic conjugation in FOMA: selecting the rule

- Processing the suffixes (3 marker):

```
define ImperfectSuffix "3" -> u || _ Type;
define PerfectMascSuffix "3" -> a || _ Type;
define PerfectFemSuffix "3" -> a t || _ Type;
define PerfectSuffix [ CheckMasc .o. PerfectMascSuffix ] | [CheckFem .o.
      PerfectFemSuffix ] ;
define Suffix [ CheckPerf .o. PerfectSuffix ] | [ CheckImperf .o. ImperfectSuffix ];
```

- Combining all stages together:

```
define Cleanup Type | Voice | Aspect | Person | Gender -> [] ;
define Grammar Input .o. PosInsertion .o. StemProcessing .o. Fill .o. Prefix .o.
      Suffix .o. Cleanup;
```

Arabic conjugation in FOMA: selecting the rule

- Processing the suffixes (3 marker):

```
define ImperfectSuffix "3" -> u || _ Type;
define PerfectMascSuffix "3" -> a || _ Type;
define PerfectFemSuffix "3" -> a t || _ Type;
define PerfectSuffix [ CheckMasc .o. PerfectMascSuffix ] | [CheckFem .o.
    PerfectFemSuffix ] ;
define Suffix [ CheckPerf .o. PerfectSuffix ] | [ CheckImperf .o. ImperfectSuffix ];
```

- Combining all stages together:

```
define Cleanup Type | Voice | Aspect | Person | Gender -> [] ;
define Grammar Input .o. PosInsertion .o. StemProcessing .o. Fill .o. Prefix .o.
    Suffix .o. Cleanup;
```

- Real Arabic morphology is much more complex.

Arabic conjugation in FOMA: selecting the rule

- Processing the suffixes (3 marker):

```
define ImperfectSuffix "3" -> u || _ Type;
define PerfectMascSuffix "3" -> a || _ Type;
define PerfectFemSuffix "3" -> a t || _ Type;
define PerfectSuffix [ CheckMasc .o. PerfectMascSuffix ] | [CheckFem .o.
    PerfectFemSuffix ] ;
define Suffix [ CheckPerf .o. PerfectSuffix ] | [ CheckImperf .o. ImperfectSuffix ];
```

- Combining all stages together:

```
define Cleanup Type | Voice | Aspect | Person | Gender -> [] ;
define Grammar Input .o. PosInsertion .o. StemProcessing .o. Fill .o. Prefix .o.
    Suffix .o. Cleanup;
```

- Real Arabic morphology is much more complex.
- But it was one of the first languages to obtain a transducer grammar (Beesley, 1990).

Morphological tagging: example

- The main task of computational morphology: morphological tagging.
- Tagging assigns morphological labels to words.

DT	JJ	NN	VBD	DT	JJ	NN
The	*baseball*	*player*	*made*	*a*	*home*	*run*

Morphological tagging: example

- The main task of computational morphology: morphological tagging.
- Tagging assigns morphological labels to words.

DT	JJ	NN	VBD	DT	JJ	NN
The	*baseball*	*player*	*made*	*a*	*home*	*run*

- The most difficult problem: homonymy.

PRP	VB	RB	TO	VB	NN
I	*run*	*home*	*to*	*play*	*baseball*

Morphological tagging: example

- The main task of computational morphology: morphological tagging.
- Tagging assigns morphological labels to words.

DT	JJ	NN	VBD	DT	JJ	NN
The	*baseball*	*player*	*made*	*a*	*home*	*run*

- The most difficult problem: homonymy.

PRP	VB	RB	TO	VB	NN
I	*run*	*home*	*to*	*play*	*baseball*

- Some words have several tags:
  - *baseball*: NN, JJ
  - *run*: VB, VBN, NN
  - *home*: NN, JJ, RB

Morphological tagging: example

- The main task of computational morphology: morphological tagging.
- Tagging assigns morphological labels to words.

DT	JJ	NN	VBD	DT	JJ	NN
The	*baseball*	*player*	*made*	*a*	*home*	*run*

- The most difficult problem: homonymy.

PRP	VB	RB	TO	VB	NN
I	*run*	*home*	*to*	*play*	*baseball*

- Some words have several tags:
  - *baseball*: NN, JJ
  - *run*: VB, VBN, NN
  - *home*: NN, JJ, RB
- How to discriminate between possible variants?

Morphological tagging: example

- The main task of computational morphology: morphological tagging.
- Tagging assigns morphological labels to words.

DT	JJ	NN	VBD	DT	JJ	NN
The	*baseball*	*player*	*made*	*a*	*home*	*run*

- The most difficult problem: homonymy.

PRP	VB	RB	TO	VB	NN
I	*run*	*home*	*to*	*play*	*baseball*

- Some words have several tags:
  - *baseball*: NN, JJ
  - *run*: VB, VBN, NN
  - *home*: NN, JJ, RB
- How to discriminate between possible variants?
- Other problem: tagging of unknown words.

Morphological tagging: variants

- Two variants of morphological tagging.
- Coarse (POS-tagging): only part-of-speech labels (about 10—15 labels).

baseball	NN

Morphological tagging: variants

- Two variants of morphological tagging.
- Coarse (POS-tagging): only part-of-speech labels (about 10—15 labels).

baseball NN

- Fine-grained: full morphological description.
- Feature-based description:

kupila "(she) bought" VERB Mood=Ind, Tense=Past, Aspect=Perf, Voice=Active, Number=Sing, Gender=Fem

Morphological tagging: variants

- Two variants of morphological tagging.
- Coarse (POS-tagging): only part-of-speech labels (about 10–15 labels).

  baseball NN

- Fine-grained: full morphological description.
- Feature-based description:

  kupila "(she) bought" VERB Mood=Ind, Tense=Past, Aspect=Perf, Voice=Active, Number=Sing, Gender=Fem

- Positional description:

  kupila Vmis-sfa-e-

Morphological tagging: variants

- Two variants of morphological tagging.
- Coarse (POS-tagging): only part-of-speech labels (about 10—15 labels).

  baseball NN

- Fine-grained: full morphological description.
- Feature-based description:

  kupila "(she) bought" VERB Mood=Ind, Tense=Past, Aspect=Perf, Voice=Active, Number=Sing, Gender=Fem

- Positional description:

  kupila Vmis-sfa-e-
- For English: no coarse tags, extended set of POS-tags.

Morphological tagging: variants

- Two variants of morphological tagging.
- Coarse (POS-tagging): only part-of-speech labels (about 10—15 labels).

baseball NN

- Fine-grained: full morphological description.
- Feature-based description:

kupila "(she) bought" VERB Mood=Ind, Tense=Past, Aspect=Perf, Voice=Active, Number=Sing, Gender=Fem

- Positional description:

kupila Vmis-sfa-e-

- For English: no coarse tags, extended set of POS-tags.
- For inflectional languages: large number of complex tags (up to 1000 for Russian or Czech).

Morphological tagging standards

- Oldest standard — Penn treebank (Marcus et al., 1993). 36 POS-tags for English with no inner structure (https://www.ling.upenn.edu/courses/Fall_2003/ling001/penn_treebank_pos.html):

12.	NN	Noun, singular or mass
13.	NNS	Noun, plural
14.	NNP	Proper noun, singular
15.	NNPS	Proper noun, plural

Morphological tagging standards

- Oldest standard — Penn treebank (Marcus et al., 1993). 36 POS-tags for English with no inner structure (https://www.ling.upenn.edu/courses/Fall_2003/ling001/penn_treebank_pos.html):

12.	NN	Noun, singular or mass
13.	NNS	Noun, plural
14.	NNP	Proper noun, singular
15.	NNPS	Proper noun, plural

- For inflectional languages, two basic approaches:
  - Positional tagset (Multext-East project for Slavic languages).

Morphological tagging standards

- Oldest standard — Penn treebank (Marcus et al., 1993). 36 POS-tags for English with no inner structure (`https://www.ling.upenn.edu/courses/Fall_2003/ling001/penn_treebank_pos.html`):

12.	NN	Noun, singular or mass
13.	NNS	Noun, plural
14.	NNP	Proper noun, singular
15.	NNPS	Proper noun, plural

- For inflectional languages, two basic approaches:
  - Positional tagset (Multext-East project for Slavic languages).
  - Feature-based tagset (Universal Dependencies project).

Positional tagsets

- Used in Multext-East project for Slavic languages (`http://nl.ijs.si/ME/`).
- Each tag is a sequence of letters.

Positional tagsets

- Used in Multext-East project for Slavic languages (`http://nl.ijs.si/ME/`).
- Each tag is a sequence of letters.
- First capital letter stands for part-of-speech (N — noun, V — verb, etc.).
- For most Slavic languages there are 13 basic POS-tags.

Positional tagsets

- Used in Multext-East project for Slavic languages (`http://nl.ijs.si/ME/`).
- Each tag is a sequence of letters.
- First capital letter stands for part-of-speech (N — noun, V — verb, etc.).
- For most Slavic languages there are 13 basic POS-tags.
- Other smallcase letters reflect features:

Ncmsny	common noun, **m**asculine, **s**ingular, **n**euter, animate (**y**es).
Vmis-sfa-e-	**m**ain verb, **i**ndicative, past(**s**), **s**ingular, **f**eminine, **a**ctive voice, perfect (**e**)

Positional tagsets

- Used in Multext-East project for Slavic languages (`http://nl.ijs.si/ME/`).
- Each tag is a sequence of letters.
- First capital letter stands for part-of-speech (N — noun, V — verb, etc.).
- For most Slavic languages there are 13 basic POS-tags.
- Other smallcase letters reflect features:

Ncmsny	common noun, **m**asculine, **s**ingular, **n**euter, animate (**y**es).
Vmis-sfa-e-	**m**ain verb, **i**ndicative, past(**s**), **s**ingular, **f**eminine, **a**ctive voice, perfect (**e**)

- Disadvantage: tags are language- and specification-dependent.

Feature-based tagsets

- Tags are specified accoriding to CONLL-U format http://universaldependencies.org/format.html.
- Each tag has two parts: universal POS-tag (UPOSTAG) and feature-value description (FEATS).

Feature-based tagsets

- Tags are specified accoriding to CONLL-U format `http://universaldependencies.org/format.html`.
- Each tag has two parts: universal POS-tag (UPOSTAG) and feature-value description (FEATS).
- 17 universal POS labels:

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

Feature-based tagsets

- Tags are specified accoriding to CONLL-U format `http://universaldependencies.org/format.html`.
- Each tag has two parts: universal POS-tag (UPOSTAG) and feature-value description (FEATS).
- 17 universal POS labels:

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

- 21 features: 6 lexical and 15 inflectional (Gender, Number, etc.).

Feature-based tagsets

- Tags are specified accoriding to CONLL-U format `http://universaldependencies.org/format.html`.
- Each tag has two parts: universal POS-tag (UPOSTAG) and feature-value description (FEATS).
- 17 universal POS labels:

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

- 21 features: 6 lexical and 15 inflectional (Gender, Number, etc.).
- Is a general standard for corpora in different languages (50 languages in version 2.0, March, 2017).

N-gram models: motivation

- Morphological tagging seeks for most probable sequence of tags for given sequence of words.

N-gram models: motivation

- Morphological tagging seeks for most probable sequence of tags for given sequence of words.
- Formally, for given words $\mathbf{w}_{1,N} = w_1 \ldots w_N$ we search for sequence of tags $\widehat{\mathbf{t}}_{1,N} = t_1 \ldots t_N$ with highest probability $p(\mathbf{t}|\mathbf{w})$.

$$\widehat{t} = \operatorname{argmax}_{\mathbf{t}} p(\mathbf{t}|\mathbf{w})$$

N-gram models: motivation

- Morphological tagging seeks for most probable sequence of tags for given sequence of words.
- Formally, for given words $\mathbf{w}_{1,N} = w_1 \dots w_N$ we search for sequence of tags $\widehat{\mathbf{t}}_{1,N} = t_1 \dots t_N$ with highest probability $p(\mathbf{t}|\mathbf{w})$.

$$\widehat{t} = \text{argmax}_{\mathbf{t}}\, p(\mathbf{t}|\mathbf{w})$$

- But how to calculate the probability $p(\mathbf{t}|\mathbf{w})$?

N-gram models: motivation

- Morphological tagging seeks for most probable sequence of tags for given sequence of words.
- Formally, for given words $\mathbf{w}_{1,N} = w_1 \dots w_N$ we search for sequence of tags $\widehat{\mathbf{t}}_{1,N} = t_1 \dots t_N$ with highest probability $p(\mathbf{t}|\mathbf{w})$.

$$\widehat{t} = \text{argmax}_{\mathbf{t}}\, p(\mathbf{t}|\mathbf{w})$$

- But how to calculate the probability $p(\mathbf{t}|\mathbf{w})$?
- For now we cannot estimate even $p(\mathbf{t})$.

Probability of sequence

- By chain rule, $p(t_1 \dots t_N)$ is

$$p(t_1 \dots t_N) = p(t_1)p(t_2|t_1)p(t_3|t_1 t_2) \dots p(t_N|t_1 \dots t_{N-1})$$

Probability of sequence

- By chain rule, $p(t_1 \dots t_N)$ is

$$p(t_1 \dots t_N) = p(t_1)p(t_2|t_1)p(t_3|t_1 t_2) \dots p(t_N|t_1 \dots t_{N-1})$$

- There is no way to estimate $p(t_{1000}|t_1 \dots t_{999})$.

Probability of sequence

- By chain rule, $p(t_1 \dots t_N)$ is

$$p(t_1 \dots t_N) = p(t_1)p(t_2|t_1)p(t_3|t_1 t_2) \dots p(t_N|t_1 \dots t_{N-1})$$

- There is no way to estimate $p(t_{1000}|t_1 \dots t_{999})$.
- N-gram model assumption: each word depends only on $n-1$ preceding words (in our case, tags).
- Formally, $p(t_N|t_1 \dots t_{N-1}) = p(t_N|t_{N-n+1} \dots t_{N-1})$.

Probability of sequence

- By chain rule, $p(t_1 \dots t_N)$ is

$$p(t_1 \dots t_N) = p(t_1)p(t_2|t_1)p(t_3|t_1 t_2) \dots p(t_N|t_1 \dots t_{N-1})$$

- There is no way to estimate $p(t_{1000}|t_1 \dots t_{999})$.
- N-gram model assumption: each word depends only on $n-1$ preceding words (in our case, tags).
- Formally, $p(t_N|t_1 \dots t_{N-1}) = p(t_N|t_{N-n+1} \dots t_{N-1})$.
- For example, for trigram model ($n = 3$):

$$p(t_1 \dots t_N) = p(t_1)p(t_2|t_1)p(t_3|t_1 t_2)p(t_4|t_2 t_3) \dots p(t_N|t_{N-2} t_{N-1})$$

Probability of sequence

- By chain rule, $p(t_1 \dots t_N)$ is

$$p(t_1 \dots t_N) = p(t_1)p(t_2|t_1)p(t_3|t_1 t_2) \dots p(t_N|t_1 \dots t_{N-1})$$

- There is no way to estimate $p(t_{1000}|t_1 \dots t_{999})$.
- N-gram model assumption: each word depends only on $n-1$ preceding words (in our case, tags).
- Formally, $p(t_N|t_1 \dots t_{N-1}) = p(t_N|t_{N-n+1} \dots t_{N-1})$.
- For example, for trigram model ($n = 3$):

$$p(t_1 \dots t_N) = p(t_1)p(t_2|t_1)p(t_3|t_1 t_2)p(t_4|t_2 t_3) \dots p(t_N|t_{N-2} t_{N-1})$$

- But how to estimate $p(t_N|t_{N-2} t_{N-1})$?

Estimating n-gram probabilities

- $p(t_3|t_1\,t_2)$ is the fraction of time we expect t_3 to occur after $t_1\,t_2$.

Estimating n-gram probabilities

- $p(t_3|t_1\,t_2)$ is the fraction of time we expect t_3 to occur after $t_1\,t_2$.
- Let us calculate this fraction:

$$
\begin{aligned}
p(t_3|t_1\,t_2) &= \frac{c(t_1\,t_2\,t_3)}{c(t_1\,t_2\odot)} = \frac{c(t_1\,t_2\,t_3)}{\sum_t c(t_1\,t_2\,t)}\\
c(t_1\,t_2\,t_3) &\text{ — number of } t_1\,t_2\,t_3 \text{ occurrences,}\\
c(t_1\,t_2\odot) &\text{ — number of times something occurs after } t_1\,t_2.
\end{aligned}
$$

Estimating *n*-gram probabilities

- $p(t_3|t_1\,t_2)$ is the fraction of time we expect t_3 to occur after $t_1\,t_2$.
- Let us calculate this fraction:

$$
\begin{aligned}
p(t_3|t_1\,t_2) &= \frac{c(t_1\,t_2\,t_3)}{c(t_1\,t_2\odot)} = \frac{c(t_1\,t_2\,t_3)}{\sum_t c(t_1\,t_2\,t)} \\
c(t_1\,t_2\,t_3) &\text{ — number of } t_1\,t_2\,t_3 \text{ occurrences,} \\
c(t_1\,t_2\odot) &\text{ — number of times something occurs after } t_1\,t_2.
\end{aligned}
$$

- Problem: everything containing a trigram that never occurred in training corpus ($c(t_1\,t_2\,t_3) = 0$) has count 0.

Estimating n-gram probabilities

- $p(t_3|t_1\, t_2)$ is the fraction of time we expect t_3 to occur after $t_1\, t_2$.
- Let us calculate this fraction:

$$
\begin{aligned}
p(t_3|t_1\, t_2) &= \frac{c(t_1\, t_2\, t_3)}{c(t_1\, t_2 \odot)} = \frac{c(t_1\, t_2\, t_3)}{\sum_t c(t_1\, t_2\, t)} \\
c(t_1\, t_2\, t_3) &\text{ — number of } t_1\, t_2\, t_3 \text{ occurrences,} \\
c(t_1\, t_2 \odot) &\text{ — number of times something occurs after } t_1\, t_2.
\end{aligned}
$$

- Problem: everything containing a trigram that never occurred in training corpus ($c(t_1\, t_2\, t_3) = 0$) has count 0.
- Solution: every n-gram additionally occurs α times.

$$
p(t_3|t_1\, t_2) = \frac{c(t_1\, t_2\, t_3) + \alpha}{c(t_1\, t_2 \odot) + \alpha|D|}, \; |D| \text{ — size of dictionary.}
$$

Estimating n-gram probabilities

- additive (Laplace) smoothing — add α to all the counts:

$$p(t_3|t_1 t_2) = \frac{c(t_1 t_2 t_3) + \alpha}{c(t_1 t_2 \odot) + \alpha|D|}, |D| \text{ — size of dictionary.}$$

Estimating *n*-gram probabilities

- additive (Laplace) smoothing — add α to all the counts:

$$p(t_3|t_1\,t_2) = \frac{c(t_1\,t_2\,t_3) + \alpha}{c(t_1\,t_2\odot) + \alpha|D|},\ |D| \text{ — size of dictionary.}$$

- How to choose α? It should depend on n-gram order, size of dictionary, corpus size...

Estimating n-gram probabilities

- additive (Laplace) smoothing — add α to all the counts:

$$p(t_3|t_1 t_2) = \frac{c(t_1 t_2 t_3) + \alpha}{c(t_1 t_2 \odot) + \alpha|D|}, |D| \text{ — size of dictionary.}$$

- How to choose α? It should depend on n-gram order, size of dictionary, corpus size...
- With improper α: inadequate.
- Selection of proper α: too complicated (used only for unigram models).

Backoff smoothing

- Sometimes trigram counts are too sparse (data from Europarl corpus):

new scientific fact	0
scientific fact	12
new scientific do	0
scientific do	0

Backoff smoothing

- Sometimes trigram counts are too sparse (data from Europarl corpus):

new scientific fact	0
scientific fact	12
new scientific do	0
scientific do	0

- By trigram model $p(fact|new\,scientific) = p(do|new\,scientific)$.

Backoff smoothing

- Sometimes trigram counts are too sparse (data from Europarl corpus):

new scientific fact	0
scientific fact	12
new scientific do	0
scientific do	0

- By trigram model $p(fact|new\, scientific) = p(do|new\, scientific)$.
- We should “descend” to lower order for more reliable estimates.

Backoff smoothing

- Sometimes trigram counts are too sparse (data from Europarl corpus):

new scientific fact	0
scientific fact	12
new scientific do	0
scientific do	0

- By trigram model $p(fact|new\,scientific) = p(do|new\,scientific)$.
- We should "descend" to lower order for more reliable estimates.
- General scheme (interpolation):

$$
\begin{aligned}
p_I(t_n|\mathbf{t}_{1,n-1}) &= \lambda p_c(t_n|\mathbf{t}_{1,n-1}) + (1-\lambda) p_I(t_n|\mathbf{t}_{2,n-1}) \\
p_c(t_n|\mathbf{t}_{1,n-1}) &= \frac{c(t_1 \ldots t_n)}{c(t_1 \ldots t_{n-1} \odot)} (\text{“honest” counts})
\end{aligned}
$$

Backoff smoothing

- General scheme (interpolation):

$$\begin{aligned} p_I(t_n|\mathbf{t}_{1,n-1}) &= \lambda p_c(t_n|\mathbf{t}_{1,n-1}) + (1-\lambda)p_I(t_n|\mathbf{t}_{2,n-1}) \\ p_c(t_n|\mathbf{t}_{1,n-1}) &= \frac{c(t_1 \ldots t_n)}{c(t_1 \ldots t_{n-1} \odot)} (\text{“honest” counts}) \end{aligned}$$

- General scheme (backoff):

$$p_{BO}(t_n|\mathbf{t}_{1,n-1}) = \begin{cases} \lambda p_c(t_n|\mathbf{t}_{1,n-1}), & c(t_1 \ldots t_n) > 0, \\ (1-\lambda)p_{BO}(t_n|\mathbf{t}_{2,n-1}), & c(t_1 \ldots t_n) = 0 \end{cases}$$

Backoff smoothing

- General scheme (interpolation):

$$\begin{aligned} p_I(t_n|\mathbf{t}_{1,n-1}) &= \lambda p_c(t_n|\mathbf{t}_{1,n-1}) + (1-\lambda)p_I(t_n|\mathbf{t}_{2,n-1}) \\ p_c(t_n|\mathbf{t}_{1,n-1}) &= \frac{c(t_1 \ldots t_n)}{c(t_1 \ldots t_{n-1}\odot)} (\text{“honest” counts}) \end{aligned}$$

- General scheme (backoff):

$$p_{BO}(t_n|\mathbf{t}_{1,n-1}) = \begin{cases} \lambda p_c(t_n|\mathbf{t}_{1,n-1}), & c(t_1 \ldots t_n) > 0, \\ (1-\lambda)p_{BO}(t_n|\mathbf{t}_{2,n-1}), & c(t_1 \ldots t_n) = 0 \end{cases}$$

- How to calculate λ?

Backoff smoothing

- General scheme (interpolation):

$$\begin{aligned} p_I(t_n|\mathbf{t}_{1,n-1}) &= \lambda p_c(t_n|\mathbf{t}_{1,n-1}) + (1-\lambda)p_I(t_n|\mathbf{t}_{2,n-1}) \\ p_c(t_n|\mathbf{t}_{1,n-1}) &= \frac{c(t_1 \ldots t_n)}{c(t_1 \ldots t_{n-1}\odot)} (\text{“honest” counts}) \end{aligned}$$

- General scheme (backoff):

$$p_{BO}(t_n|\mathbf{t}_{1,n-1}) = \begin{cases} \lambda p_c(t_n|\mathbf{t}_{1,n-1}), & c(t_1 \ldots t_n) > 0, \\ (1-\lambda)p_{BO}(t_n|\mathbf{t}_{2,n-1}), & c(t_1 \ldots t_n) = 0 \end{cases}$$

- How to calculate λ?
- The greater is λ for history $t_1 \ldots t_{n-1}$, the more we “trust” the counts and the less expect new words.

Backoff smoothing

- General scheme (interpolation):

$$
\begin{aligned}
p_I(t_n|\mathbf{t}_{1,n-1}) &= \lambda p_c(t_n|\mathbf{t}_{1,n-1}) + (1-\lambda)p_I(t_n|\mathbf{t}_{2,n-1}) \\
p_c(t_n|\mathbf{t}_{1,n-1}) &= \frac{c(t_1\ldots t_n)}{c(t_1\ldots t_{n-1}\odot)} (\text{“honest” counts})
\end{aligned}
$$

- General scheme (backoff):

$$
p_{BO}(t_n|\mathbf{t}_{1,n-1}) = \begin{cases} \lambda p_c(t_n|\mathbf{t}_{1,n-1}), & c(t_1\ldots t_n) > 0, \\ (1-\lambda)p_{BO}(t_n|\mathbf{t}_{2,n-1}), & c(t_1\ldots t_n) = 0 \end{cases}
$$

- How to calculate λ?
- The greater is λ for history $t_1 \ldots t_{n-1}$, the more we “trust” the counts and the less expect new words.
- We do it when:
  - $t_1 \ldots t_{n-1}$ occurs enough times.

Backoff smoothing

- General scheme (interpolation):

$$\begin{aligned} p_I(t_n|\mathbf{t}_{1,n-1}) &= \lambda p_c(t_n|\mathbf{t}_{1,n-1}) + (1-\lambda)p_I(t_n|\mathbf{t}_{2,n-1}) \\ p_c(t_n|\mathbf{t}_{1,n-1}) &= \frac{c(t_1 \dots t_n)}{c(t_1 \dots t_{n-1}\odot)} (\text{“honest” counts}) \end{aligned}$$

- General scheme (backoff):

$$p_{BO}(t_n|\mathbf{t}_{1,n-1}) = \begin{cases} \lambda p_c(t_n|\mathbf{t}_{1,n-1}), & c(t_1 \dots t_n) > 0, \\ (1-\lambda)p_{BO}(t_n|\mathbf{t}_{2,n-1}), & c(t_1 \dots t_n) = 0 \end{cases}$$

- How to calculate λ?
- The greater is λ for history $t_1 \dots t_{n-1}$, the more we “trust” the counts and the less expect new words.
- We do it when:
  - $t_1 \dots t_{n-1}$ occurs enough times.
  - $t_1 \dots t_{n-1}$ has not much continuations.

Witten-Bell smoothing

- Witten-Bell smoothing:

$$
\begin{aligned}
p_I(t_n|\mathbf{t}_{1,n-1}) &= \lambda p_c(t_n|\mathbf{t}_{1,n-1}) + (1-\lambda)p_I(t_n|\mathbf{t}_{2,n-1}) \\
\lambda &= c(t_1 \ldots t_{n-1}\odot)c(t_1 \ldots t_{n-1}\odot) + N_{1+}(t_1 \ldots t_{n-1}) \\
N_{1+}(t_1 \ldots t_{n-1}) &= |\{t|c(t_1 \ldots t_{n-1}t) > 0\} \\
N_{1+}(t_1 \ldots t_{n-1}) &- \text{“number of continuations”}
\end{aligned}
$$

Witten-Bell smoothing

- Witten-Bell smoothing:

$$
\begin{aligned}
p_I(t_n|\mathbf{t}_{1,n-1}) &= \lambda p_c(t_n|\mathbf{t}_{1,n-1}) + (1-\lambda)p_I(t_n|\mathbf{t}_{2,n-1}) \\
\lambda &= c(t_1 \ldots t_{n-1}\odot)c(t_1 \ldots t_{n-1}\odot) + N_{1+}(t_1 \ldots t_{n-1}) \\
N_{1+}(t_1 \ldots t_{n-1}) &= |\{t|c(t_1 \ldots t_{n-1}t) > 0\} \\
N_{1+}(t_1 \ldots t_{n-1}) &- \text{“number of continuations”}
\end{aligned}
$$

- Example (BNC corpus):

w_1	$c(w_1\odot)$	$N_{1+}(w_1)$	$N_{3+}(w_1)$	$\lambda(w_1)$	$1-\lambda(w_1)$
spite	2899	59	15	$\frac{2899}{2899+59} = 0.980$	0.02
stupid	2898	602	117	$\frac{2898}{2898+602} = 0.828$	0.172

Witten-Bell smoothing

- Witten-Bell smoothing:

$$
\begin{aligned}
p_I(t_n|\mathbf{t}_{1,n-1}) &= \lambda p_c(t_n|\mathbf{t}_{1,n-1}) + (1-\lambda)p_I(t_n|\mathbf{t}_{2,n-1}) \\
\lambda &= c(t_1 \dots t_{n-1}\odot)c(t_1 \dots t_{n-1}\odot) + N_{1+}(t_1 \dots t_{n-1}) \\
N_{1+}(t_1 \dots t_{n-1}) &= |\{t|c(t_1 \dots t_{n-1}t) > 0\} \\
N_{1+}(t_1 \dots t_{n-1}) &— \text{“number of continuations”}
\end{aligned}
$$

- Example (BNC corpus):

w_1	$c(w_1\odot)$	$N_{1+}(w_1)$	$N_{3+}(w_1)$	$\lambda(w_1)$	$1-\lambda(w_1)$
spite	2899	59	15	$\frac{2899}{2899+59} = 0.980$	0.02
stupid	2898	602	117	$\frac{2898}{2898+602} = 0.828$	0.172

- Unigram counts for *stupid* are 86 times more valuable than for *spite*.
- The more continuations we have, the less is λ.

Witten-Bell smoothing

- In the worst case (even bigram $t_{n-1}t_n$ is unseen) we backoff to unigram probability.

Witten-Bell smoothing

- In the worst case (even bigram $t_{n-1}t_n$ is unseen) we backoff to unigram probability.
- But that's not the unigram probability that should be used.

Witten-Bell smoothing

- In the worst case (even bigram $t_{n-1}t_n$ is unseen) we backoff to unigram probability.
- But that's not the unigram probability that should be used.
- Example: $c(Angeles)$ is rather high, but it occurs only after *Los*.
- It is strange to assume this word after others.

Witten-Bell smoothing

- In the worst case (even bigram $t_{n-1}t_n$ is unseen) we backoff to unigram probability.
- But that's not the unigram probability that should be used.
- Example: $c(Angeles)$ is rather high, but it occurs only after *Los*.
- It is strange to assume this word after others.
- Instead of unigram probability of t_n we use

$$
\begin{aligned}
p_{BO}(t_n) &= \frac{N_{+1}(t_n)}{\sum_t N_{+1}(t)} \\
N_{+1}(t_n) &= |\{t | c(t\ t_n) > 0\}| \\
N_{+1}(t_n) &- \text{(number of left continuations)}
\end{aligned}
$$

Witten-Bell smoothing

- In the worst case (even bigram $t_{n-1}t_n$ is unseen) we backoff to unigram probability.
- But that's not the unigram probability that should be used.
- Example: $c(\mathit{Angeles})$ is rather high, but it occurs only after *Los*.
- It is strange to assume this word after others.
- Instead of unigram probability of t_n we use

$$
\begin{aligned}
p_{BO}(t_n) &= \frac{N_{+1}(t_n)}{\sum_t N_{+1}(t)} \\
N_{+1}(t_n) &= |\{t|c(t\ t_n) > 0\} \\
N_{+1}(t_n) &- \text{(number of left continuations)}
\end{aligned}
$$

- Witten-Bell smoothing is not the best, but enough for our purposes.

Witten-Bell smoothing

- In the worst case (even bigram $t_{n-1}t_n$ is unseen) we backoff to unigram probability.
- But that's not the unigram probability that should be used.
- Example: $c(Angeles)$ is rather high, but it occurs only after *Los*.
- It is strange to assume this word after others.
- Instead of unigram probability of t_n we use

$$
\begin{aligned}
p_{BO}(t_n) &= \frac{N_{+1}(t_n)}{\sum_t N_{+1}(t)} \\
N_{+1}(t_n) &= |\{t|c(t\ t_n) > 0\}| \\
N_{+1}(t_n) &- \text{(number of left continuations)}
\end{aligned}
$$

- Witten-Bell smoothing is not the best, but enough for our purposes.
- More powerful methods:
  - Deleted interpolation.
  - Kneser-Ney smoothing (and its modified version).

Witten-Bell smoothing

- In the worst case (even bigram $t_{n-1}t_n$ is unseen) we backoff to unigram probability.
- But that's not the unigram probability that should be used.
- Example: $c(Angeles)$ is rather high, but it occurs only after *Los*.
- It is strange to assume this word after others.
- Instead of unigram probability of t_n we use

$$
\begin{aligned}
p_{BO}(t_n) &= \frac{N_{+1}(t_n)}{\sum_t N_{+1}(t)} \\
N_{+1}(t_n) &= |\{t|c(t\ t_n) > 0\}| \\
N_{+1}(t_n) &\ - \ (\text{number of left continuations})
\end{aligned}
$$

- Witten-Bell smoothing is not the best, but enough for our purposes.
- More powerful methods:
  - Deleted interpolation.
  - Kneser-Ney smoothing (and its modified version).
  - Also non-ngram language model (factored models, neural net-based, etc.).