MORPHEME-LEVEL PROCESSING

Lecture on 9 March 2021 at Aalto University (Zoom)

Slides by Mathias Creutz and Sami Virpioja

INTRODUCTION

LINGDIG (LINGUISTIC DIVERSITY AND DIGITAL HUMANITIES) MASTER'S PROGRAMME:

LANGUAGE TECHNOLOGY COURSES OFFERED AT THE UNIVERSITY OF HELSINKI

- Computational Morphology (fall 2021, 5 cr: *Oct – Dec*)
- Computational Syntax (spring 2021, 5 cr: *Mar – May*)
- Computational Semantics (spring 2022, 5 cr: *Jan – Mar*)
- Models and Algorithms in NLP applications (fall 2021, 5 cr: *Sep – Oct*)
- Approaches to Natural Language Understanding (spring 2022, 5 cr: *Mar – May*)
- Introduction to Deep Learning (spring 2022, 5 cr)
- A practical intro to modern Neural Machine Translation (fall 2021?, 5 cr: *Oct – Dec*)
- *plus* courses in General Linguistics, Phonetics, Cognitive Science and Digihum

- More info: http://blogs.helsinki.fi/language-technology/

CONTENTS

CONTENTS

CONTENTS

- Linguistic theory
- Automatic morphological processing
  - Approach 1: Normalization or “Canonical forms”
    - Stemming
    - Lemmatization
  - **Approach 2: Analysis and generation**
    - **Finite-state methods**
    - **Supervised machine learning: Morphological reinflection**
  - Approach 3: Segmentation
    - Unsupervised learning, method 1: Harris’s method
    - Unsupervised learning, method 2: Morfessor
    - Unsupervised learning, method 3: Byte pair encoding (BPE) and SentencePiece
  - Approach 4: Implicit modeling
    - Feature extraction in word embeddings (word2vec): FastText
    - Character-based models

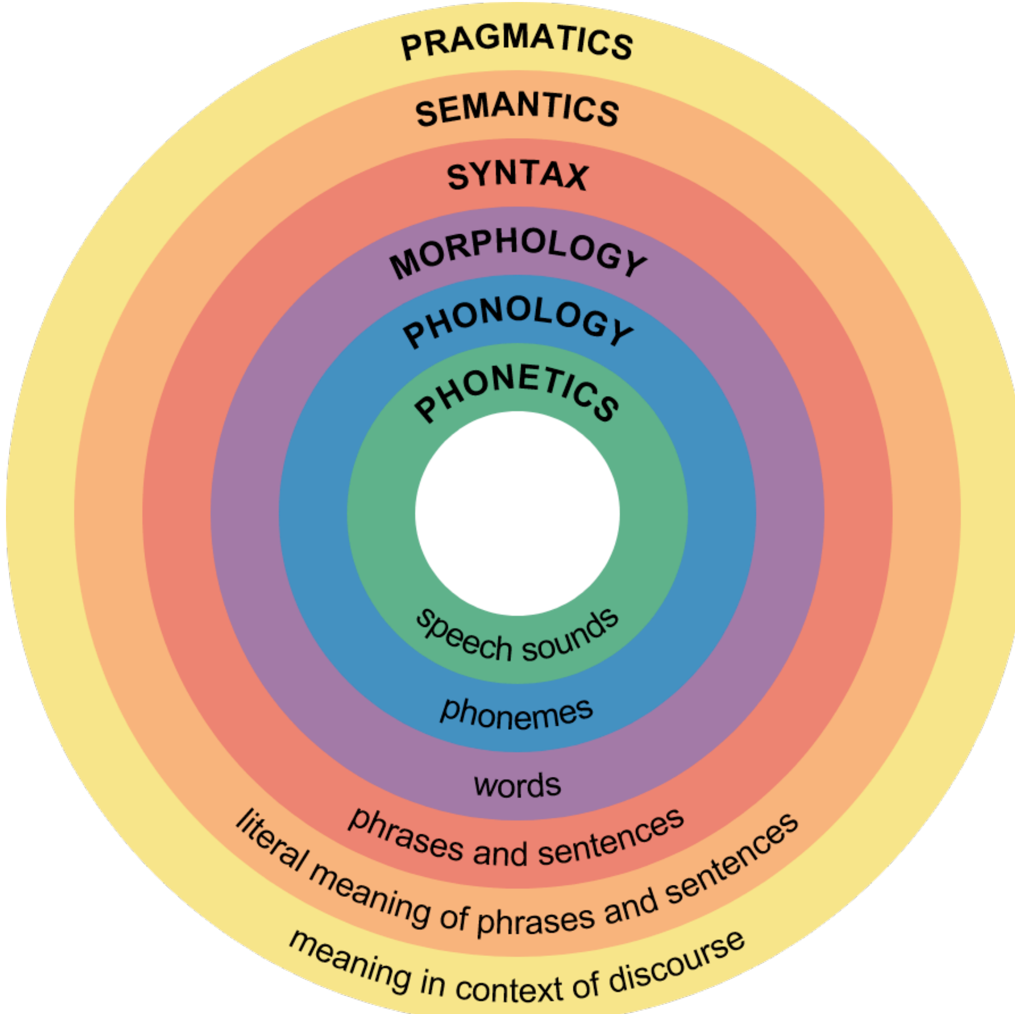
CONTENTS

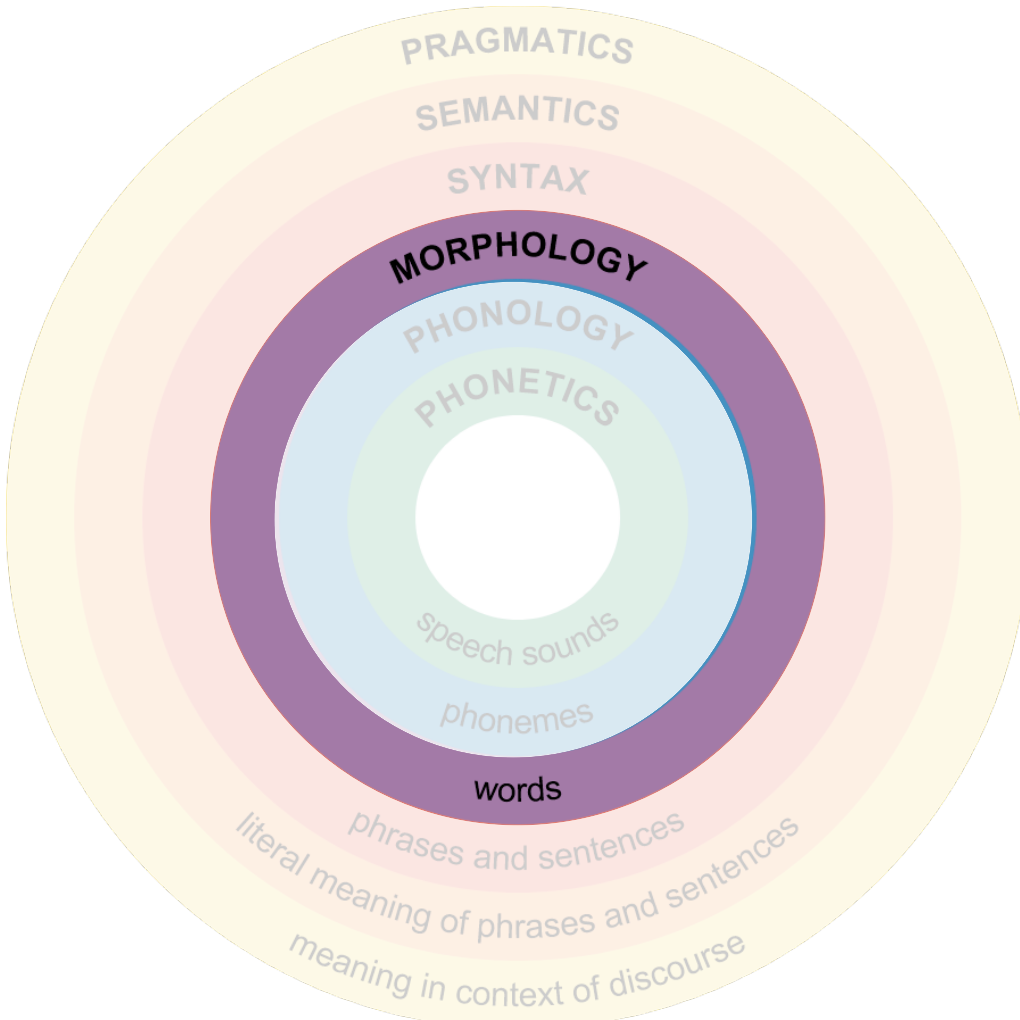
- Linguistic theory
- Automatic morphological processing
  - Approach 1: Normalization or “Canonical forms”
    – Stemming
    – Lemmatization
  - Approach 2: Analysis and generation
    – Finite-state methods
    – Supervised machine learning: Morphological reinflection
  - **Approach 3: Segmentation**
    – Unsupervised learning, method 1: Harris’s method
    – Unsupervised learning, method 2: Morfessor
    – Unsupervised learning, method 3: Byte pair encoding (BPE) and SentencePiece
  - Approach 4: Implicit modeling
    – Feature extraction in word embeddings (word2vec): FastText
    – Character-based models

CONTENTS

- Linguistic theory
- Automatic morphological processing
  - Approach 1: Normalization or “Canonical forms”
    - Stemming
    - Lemmatization
  - Approach 2: Analysis and generation
    - Finite-state methods
    - Supervised machine learning: Morphological reinflection
  - Approach 3: Segmentation
    - Unsupervised learning, method 1: Harris’s method
    - Unsupervised learning, method 2: Morfessor
    - Unsupervised learning, method 3: Byte pair encoding (BPE) and SentencePiece
  - **Approach 4: Implicit modeling**
    - **Feature extraction in word embeddings (word2vec): fastText**
    - **Character-based models**

LINGUISTIC THEORY


PRAGMATICS
SEMANTICS
SYNTAX
MORPHOLOGY
PHONOLOGY
PHONETICS
speech sounds
phonemes
words
phrases and sentences
literal meaning of phrases and sentences
meaning in context of discourse


PRAGMATICS
SEMANTICS
SYNTAX
MORPHOLOGY
PHONOLOGY
PHONETICS
speech sounds
phonemes
words
phrases and sentences
literal meaning of phrases and sentences
meaning in context of discourse

LINGUISTIC MORPHOLOGY

- **Morphology:** Study (-*logy*) of shape and form (*morpho*)
- In linguistics:
  - Identification, analysis and description of the structure of words

LINGUISTIC MORPHOLOGY

- **Morphology:** Study (*-logy*) of shape and form (*morpho*)
- In linguistics:
  - Identification, analysis and description of the structure of words
- **Word form** vs. word **lexeme**:

  Are “cat” and “cats” the same word or not?
  - The same lexeme
  - Different forms

LINGUISTIC MORPHOLOGY

- **Morphology:** Study (*-logy*) of shape and form (*morpho*)
- In linguistics:
  - Identification, analysis and description of the structure of words
- **Word form** vs. word **lexeme**:

  Are “cat” and “cats” the same word or not?
  - The same lexeme
  - Different forms
- Traditional view: Grammar = morphology + syntax

LINGUISTIC MORPHOLOGY

- **Morphology:** Study (-*logy*) of shape and form (*morpho*)
- In linguistics:
  - Identification, analysis and description of the structure of words
- **Word form** vs. word **lexeme**:

  Are “cat” and “cats” the same word or not?
  - The same lexeme
  - Different forms
- Traditional view: Grammar = morphology + syntax
- The **morphological complexity** of languages vary:
  - “punaviinipullossa" (Finnish) vs. “in the bottle of red wine”
  - “itsega" (Cherokee) vs. “you are all going”

TERMINOLOGY

Morphemes are

- “the smallest individually meaningful elements in the utterances of a language” (Charles F. Hockett, A Course in Modern Linguistics, 1958)
- “the primitive units of syntax, the smallest units that can bear meaning” (Peter H. Matthews, Morphology, 1991)
- “minimal meaningful form-units” (Robert de Beaugrande, A New Introduction to the Study of Text and Discourse, 2004)

TERMINOLOGY

Morphemes are

- "the smallest individually meaningful elements in the utterances of a language" (Charles F. Hockett, A Course in Modern Linguistics, 1958)
- "the primitive units of syntax, the smallest units that can bear meaning" (Peter H. Matthews, Morphology, 1991)
- "minimal meaningful form-units" (Robert de Beaugrande, A New Introduction to the Study of Text and Discourse, 2004)

Meaning elements (cats = CAT + PLURAL) or **form elements** (cats = *cat* + *-s*)?

TYPES OF MORPHEMES

- **Root:** a portion of word without any affixes; carries the principal portion of meaning (buildings ➔ build)

TYPES OF MORPHEMES

- **Root:** a portion of word without any affixes; carries the principal portion of meaning (buildings ➔ build)
- **Stem:** a root, or compound of roots together with derivational affixes (buildings ➔ building)

TYPES OF MORPHEMES

- **Root:** a portion of word without any affixes; carries the principal portion of meaning (buildings ➔ build)
- **Stem:** a root, or compound of roots together with derivational affixes (buildings ➔ building)
- **Affix:** a bound morpheme (does not occur by itself) that is attached before, after, or inside a root or stem
  - **Prefix** (**un**-happy)
  - **Suffix** (build-**ing**, happi-**er**)
  - **Infix** (abso-**bloody**-lutely)
  - **Circumfix** (**ge**-sproch-**en**)
  - **Transfix** (e.g., vowel patterns for consonant roots in Semitic languages: k-**i**-t-**aa**-b – k-**u**-t-**u**-b)

TYPES OF MORPHEMES

- **Root:** a portion of word without any affixes; carries the principal portion of meaning (buildings ➔ build)
- **Stem:** a root, or compound of roots together with derivational affixes (buildings ➔ building)
- **Affix:** a bound morpheme (does not occur by itself) that is attached before, after, or inside a root or stem
  - **Prefix** (**un**-happy)
  - **Suffix** (build-**ing**, happi-**er**)
  - **Infix** (abso-**bloody**-lutely)
  - **Circumfix** (**ge**-sproch-**en**)
  - **Transfix** (e.g., vowel patterns for consonant roots in Semitic languages: k-**i**-t-**aa**-b – k-**u**-t-**u**-b)
- **Clitic:** a bound (but more "independent") morpheme that has syntactic characteristics of a word (that**'s**, hän**kin**)

MORPHOLOGICAL TYPOLOGY

Isolating or **analytic** (little or no morphology)

vs.

synthetic (many morphemes per word)

漢語 汉语

* Correct Latin: *Romani ite domum*

MORPHOLOGICAL TYPOLOGY

漢 汉
語 语

Isolating or **analytic** (little or no morphology)

vs.

synthetic (many morphemes per word)

Agglutinative (morphemes joined together to form words)

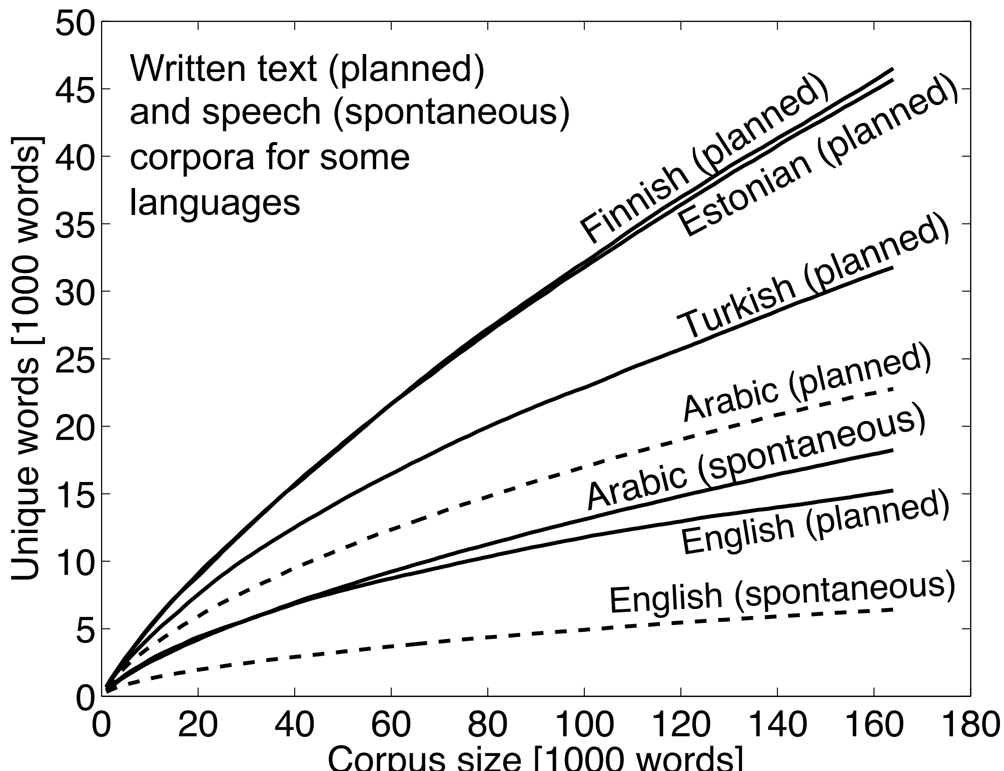
vs.

fusional (overlaying of morphemes; difficult to segment)

* Correct Latin: *Romani ite domum*

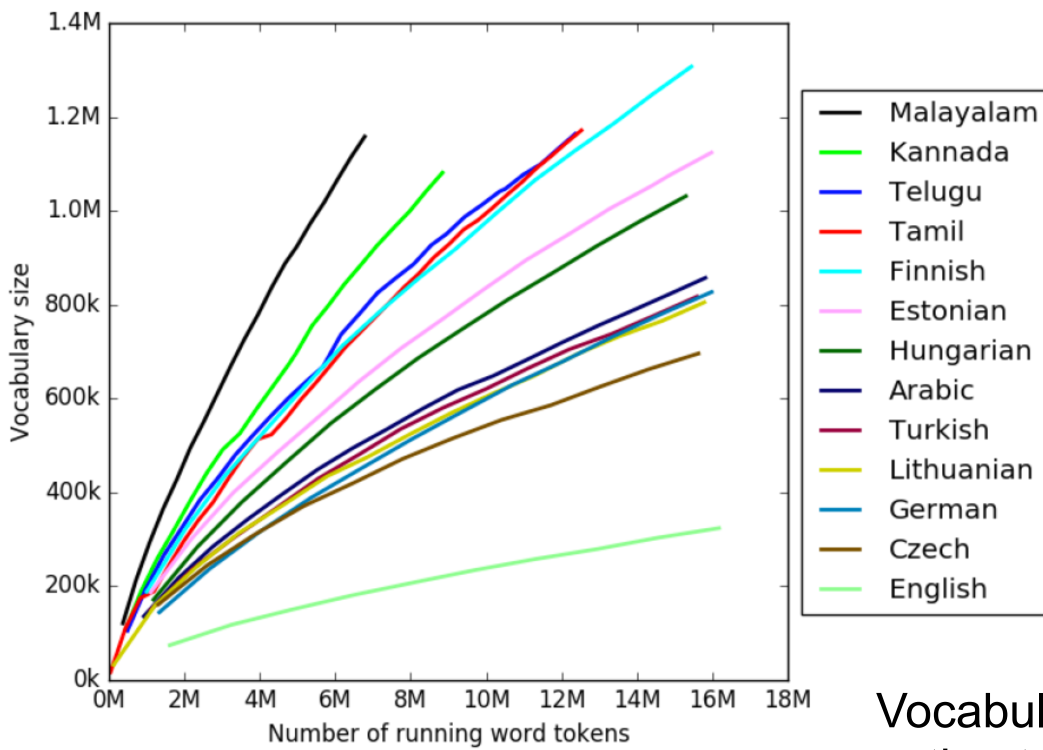
Different types of morphology in different languages:
EFFECT ON VOCABULARY SIZE (1)

Vocabulary growth estimated from text and speech



Different types of morphology in different languages:

EFFECT ON VOCABULARY SIZE (2)



Varjokallio, Kurimo, Virpioja (2016)

Vocabulary growth estimated from Wikipedia

MORPHOLOGICAL PROCESSES

Inflection:

- cat – cats
- slow – slower
- find – found

Derivation:

- build (V) – building (N)
- do (V) – doable (ADJ)
- short (ADJ) – shorten (V)
- write – rewrite
- do – undo

Compounding:

- fireman (fire + man)
- hardware (hard + ware)

HOCKETT'S MODELS OF MORPHOLOGY

Three general approaches to the modeling of morphology (Charles F. Hockett, 1954):

1. **Word-and-Paradigm** (word-based morphology)
2. **Item-and-Arrangement** (morpheme-based morphology)
3. **Item-and-Process** (lexeme-based morphology)

WORD AND PARADIGM (W&P)

	Paradigms				
Grammatical form	I	II	III	IV	V
Infinitive	wait	invite	split	sell	take
Present tense, 3rd person	waits	invites	splits	sells	takes
Present participle	waiting	inviting	splitting	selling	taking
Past tense	waited	invited	split	sold	took
Past participle	waited	invited	split	sold	taken

WORD AND PARADIGM (W&P)

	Paradigms				
Grammatical form	I	II	III	IV	V
Infinitive	wait	invite	split	sell	take
Present tense, 3rd person	waits	invites	splits	sells	takes
Present participle	waiting	inviting	splitting	selling	taking
Past tense	waited	invited	split	sold	took
Past participle	waited	invited	split	sold	taken

New word forms by analogy:

shout → I   like → II   cut → III

tell → IV   shake → V

The W&P model does not describe derivation or compounding.

ITEM & ARRANGEMENT (I&A)

I	II	III	IV	V
WAIT	INVITE	SPLIT	SELL	TAKE
WAIT + -S	INVITE + -S	SPLIT + -S	SELL + -S	TAKE + -S
WAIT + -ING	INVITE + -ING	SPLIT + -ING	SELL + -ING	TAKE + -ING
WAIT + -ED	INVITE + -ED	SPLIT + -ED	SELL + -ED	TAKE + -ED
WAIT + -EN	INVITE + -EN	SPLIT + -EN	SELL + -EN	TAKE + -EN

ITEM & ARRANGEMENT (I&A)

I	II	III	IV	V
WAIT	INVITE	SPLIT	SELL	TAKE
WAIT + -S	INVITE + -S	SPLIT + -S	SELL + -S	TAKE + -S
WAIT + -ING	INVITE + -ING	SPLIT + -ING	SELL + -ING	TAKE + -ING
WAIT + -ED	INVITE + -ED	SPLIT + -ED	SELL + -ED	TAKE + -ED
WAIT + -EN	INVITE + -EN	SPLIT + -EN	SELL + -EN	TAKE + -EN

Morphemes and allomorphs:

$\text{WAIT} = \{\text{wait}\}$, $\text{INVITE} = \{\text{invite, invit}\}$, $\text{SPLIT} = \{\text{split, splitt}\}$, $\text{SELL} = \{\text{sell, sol}\}$, $\text{TAKE} = \{\text{take, tak, took}\}$, $\text{-S} = \{\text{s}\}$, $\text{-ING} = \{\text{ing}\}$, $\text{-ED} = \{\text{ed, d}, \varnothing\}$, and $\text{-EN} = \{\text{ed, d}, \varnothing, \text{en}\}$

Morph (e.g., “splitt”):

- surface realization of a morpheme

Allomorphs (e.g., “split”, “splitt”):

- different surface realizations of the *same morpheme*

ITEM & ARRANGEMENT (I&A)

I	II	III	IV	V
WAIT	INVITE	SPLIT	SELL	TAKE
WAIT + -S	INVITE + -S	SPLIT + -S	SELL + -S	TAKE + -S
WAIT + -ING	INVITE + -ING	SPLIT + -ING	SELL + -ING	TAKE + -ING
WAIT + -ED	INVITE + -ED	SPLIT + -ED	SELL + -ED	TAKE + -ED
WAIT + -EN	INVITE + -EN	SPLIT + -EN	SELL + -EN	TAKE + -EN

Morphemes and allomorphs:

WAIT = {wait}, INVITE = {invite, invit}, SPLIT = {split, splitt}, SELL = {sell, sol}, TAKE = {take, tak, took}, -S = {s}, -ING = {ing}, -ED = {ed, d, ∅}, and -EN = {ed, d, ∅, en}

Rules:

INVITE + -ING → invit + ing = inviting

SPLIT + -EN → split + ∅ = split

SELL + -EN → sol + d = sold

TAKE + -ED → took + ∅ = took.

Morph (e.g., "splitt"):

- surface realization of a morpheme

Allomorphs (e.g., "split", "splitt"):

- different surface realizations of the *same morpheme*

ITEM & PROCESS (I&P)

- **Items:** Word forms, free morphemes (wait, invite, split, sell, take) and bound morphemes (-s, -ing, -ed, -en), all represented as lists of features (phonemic/orthographic form and grammatical categories).
- **Processes:** Operations that take one or more items and return a new item. Output and one of the inputs is always a free morpheme or word.

ITEM & PROCESS (I&P)

- **Items:** Word forms, free morphemes (wait, invite, split, sell, take) and bound morphemes (-s, -ing, -ed, -en), all represented as lists of features (phonemic/orthographic form and grammatical categories).
- **Processes:** Operations that take one or more items and return a new item. Output and one of the inputs is always a free morpheme or word.
  - Present_participle($[\text{stem}]_V$)
    - add suffix -ing to stem
    - drop final "e" from stem, if present: tak(e)+ing
    - double final stem consonant if short syllable: split+t+ing

ITEM & PROCESS (I&P)

- **Items:** Word forms, free morphemes (wait, invite, split, sell, take) and bound morphemes (-s, -ing, -ed, -en), all represented as lists of features (phonemic/orthographic form and grammatical categories).
- **Processes:** Operations that take one or more items and return a new item. Output and one of the inputs is always a free morpheme or word.
  - Present_participle([stem]$_{\mathrm{V}}$)
    - add suffix -ing to stem
    - drop final "e" from stem, if present: tak(e)+ing
    - double final stem consonant if short syllable: split+t+ing
  - Derivation$_{\mathrm{ADJ\text{-}N}}$([stem]$_{\mathrm{ADJ}}$, -ness) $\rightarrow$ [stem-ness]$_{\mathrm{N}}$
    - e.g. [black]$_{\mathrm{ADJ}}$ $\rightarrow$ [blackness]$_{\mathrm{N}}$
  - Compound([stem1]$_{\mathrm{ADJ}}$, [stem2]$_{\mathrm{N}}$) $\rightarrow$ [stem1+stem2]$_{\mathrm{N}}$
    - e.g. [black]$_{\mathrm{ADJ}}$ + [bird]$_{\mathrm{N}}$ $\rightarrow$ [blackbird]$_{\mathrm{N}}$

AUTOMATIC MORPHOLOGICAL PROCESSING

APPROACHES IN MORPHOLOGICAL PROCESSING

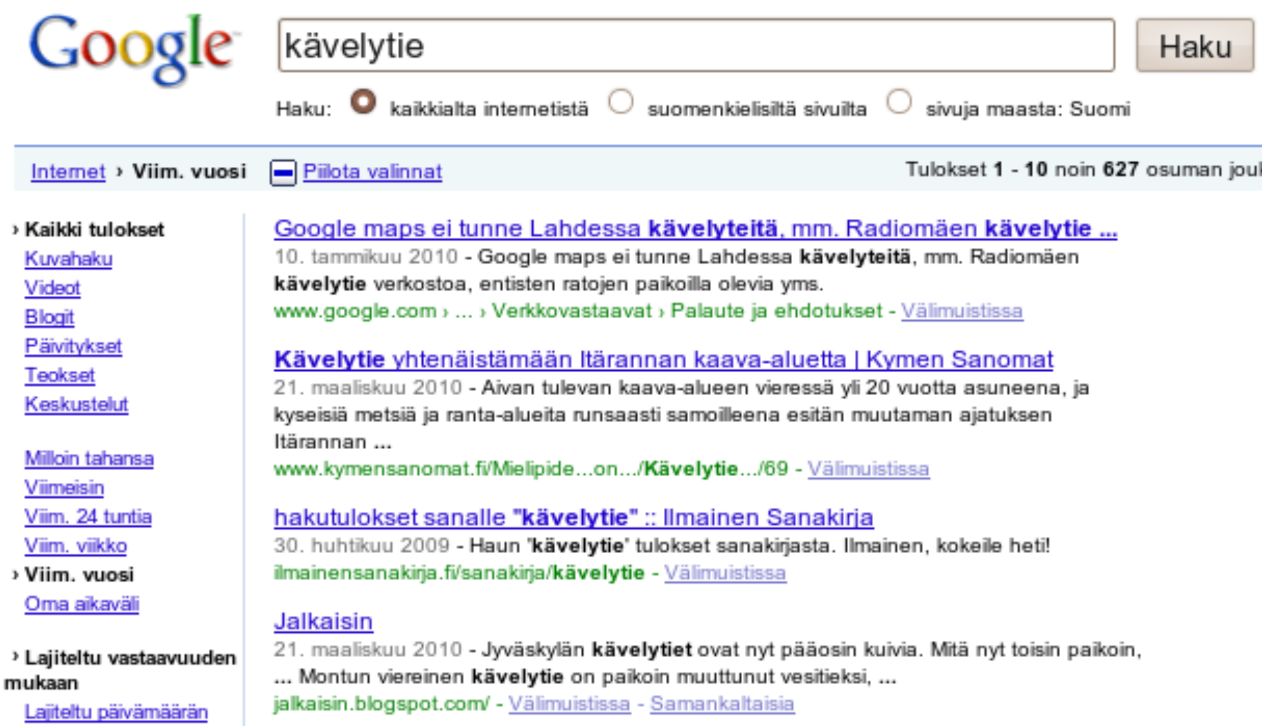
1. **Normalization** or **“Canonical forms”:** identification of morphologically related word forms
   - Stemming
   - Lemmatization
2. **Analysis and generation:** full-blown morphological lexicons
3. **Segmentation:** splitting of words into *morphs*
4. **Implicit modeling:** no explicit selection of morphs or morphemes at input level

Different applications (e.g., information retrieval, speech recognition, machine translation) have different needs.

APPROACH 1: NORMALIZATION OR “CANONICAL FORMS”

MORPHOLOGICAL "CANONICAL FORMS"

- Works both for agglutinative and fusional languages
- Applications that need to identify which word forms "are the same", without having to produce any correct word forms
- Useful in **information retrieval** ➔



Canonical form 1:
STEMMING

- Reduce inflected word forms to their stem; usually also derived forms to roots.
- Happens through suffix-stripping and reduction rules
- Stemmers for English: e.g., Porter (1980), Snowball: http://snowball.tartarus.org

STEMMING EXAMPLES

Sample text: Such an analysis can reveal features that are not easily visible from the variations in the individual genes and can lead to a picture of expression that is more biologically transparent and accessible to interpretation

Lovins stemmer: such an analys can reve featur that ar not eas vis from th vari in th individu gen and can lead to a pictur of expres that is mor biolog transpar and acces to interpres

Porter stemmer: such an analysi can reveal featur that ar not easili visibl from the variat in the individu gene and can lead to a pictur of express that is more biolog transpar and access to interpret

Paice stemmer: such an analys can rev feat that are not easy vis from the vary in the individ gen and can lead to a pict of express that is mor biolog transp and access to interpret

LIMITATIONS OF STEMMING

- Stemming is typically a much too simplified *approximation*
- Stemming fails to see connections between irregular forms or more complex phenomena
  - *bring – brought*
  - *swim – swam – swum*
  - *yksi – yhden*
  - *tähti – tähden*
- Stemming finds connections between similar, but unrelated forms
  - *sing – singed*
  - *tähtien – tähteiden*

Canonical form 2:
LEMMATIZATION

- Reduce inflected word forms to **lemmas**
- Lemma = canonical form of the lexeme = dictionary form = **base form**
  - cat's ➔ cat
  - swum ➔ swim
  - tähtien ➔ tähti
- More accurate than stemming
- Can be used in the same applications as stemming
- Often implemented as a by-product of *full morphological analysis* (= our "Approach 2" to be looked at next)

FULL MORPHOLOGICAL ANALYSIS

Examples:

cat's	cat+N+GEN
swum	swim+V+PPART
tähtien	tähti N Gen Pl
tähteiden	tähde N Gen Pl
epäjärjestyksessä	epä#järjestys N Ine Sg
epäjärjestyksessäkö	epä#järjestys N Ine Sg Foc_kO

LIMITATIONS OF MORPHOLOGICAL ANALYSIS

- Out-of-vocabulary words
  - epäjärjestelmällistyttämättömyydelläänsäkäänköhän ➔ epäjärjestelmällistyttämättömyydelläänsäkäänköhän+?

LIMITATIONS OF MORPHOLOGICAL ANALYSIS

- Out-of-vocabulary words
  - epäjärjestelmällistyttämättömyydelläänsäkäänköhän ➔ epäjärjestelmällistyttämättömyydelläänsäkäänköhän+?
- Ambiguous forms
  - saw see+V+PAST *or* saw+N *or* saw+V+INF ?

    "I **saw** her yesterday." ➔ SEE (verb)

    "The **saw** was blunt." ➔ SAW (noun)

    "Don't **saw** off the branch you are sitting on." ➔ SAW (verb)

LIMITATIONS OF MORPHOLOGICAL ANALYSIS

- Out-of-vocabulary words
  - epäjärjestelmällistyttämättömyydelläänsäkäänköhän ➔ epäjärjestelmällistyttämättömyydelläänsäkäänköhän+?

- Ambiguous forms
  - saw see+V+PAST *or* saw+N *or* saw+V+INF ?

    "I **saw** her yesterday." ➔ SEE (verb)

    "The **saw** was blunt." ➔ SAW (noun)

    "Don't **saw** off the branch you are sitting on." ➔ SAW (verb)
  - meeting meet+V+PROG *or* meeting+N ?

    "We are **meeting** tomorrow." ➔ MEET (verb)

    "In our **meeting**, we decided not to meet again." ➔ MEETING (noun)

- Solutions?

APPROACH 2: ANALYSIS AND GENERATION

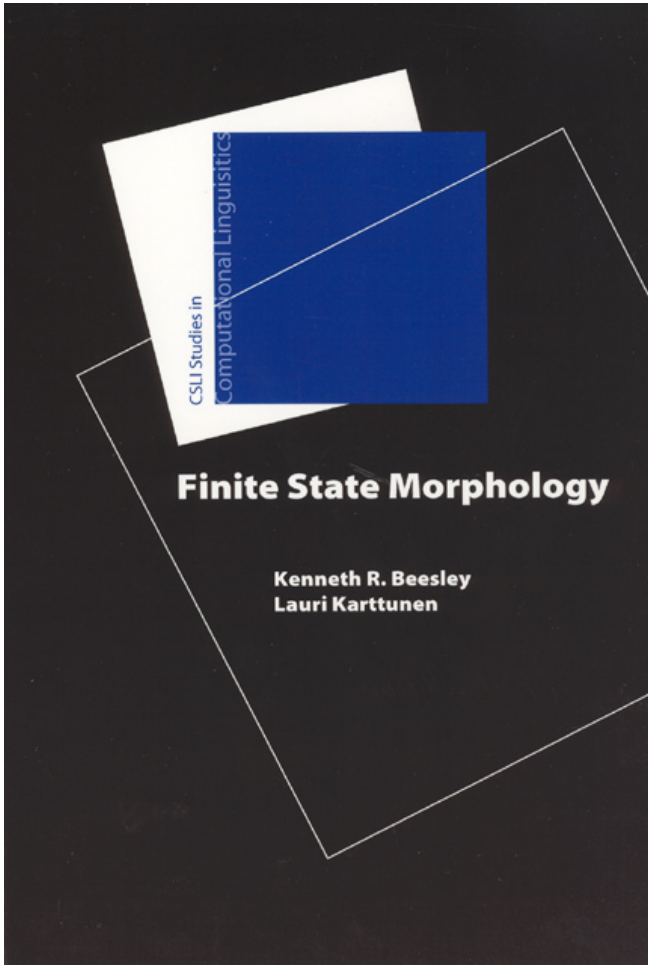
FINITE-STATE MORPHOLOGY

Book:

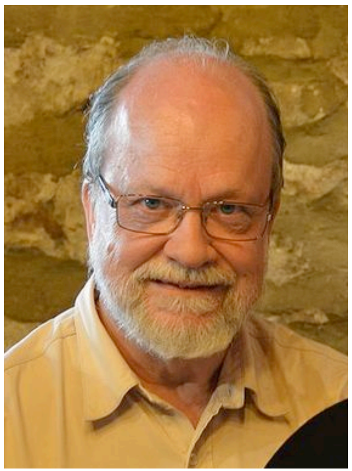
Kenneth R. Beesley and **Lauri Karttunen**, *Finite State Morphology*, CSLI Publications, 2003

http://press.uchicago.edu/ucp/books/book/distributed/F/bo3613750.html

These are **rule-based systems**, i.e., computer programs written by linguists that model morphological lexicons of different languages.



FINITE-STATE MORPHOLOGY CONTRIBUTORS FROM FINLAND



Professor emeritus
Kimmo Koskenniemi

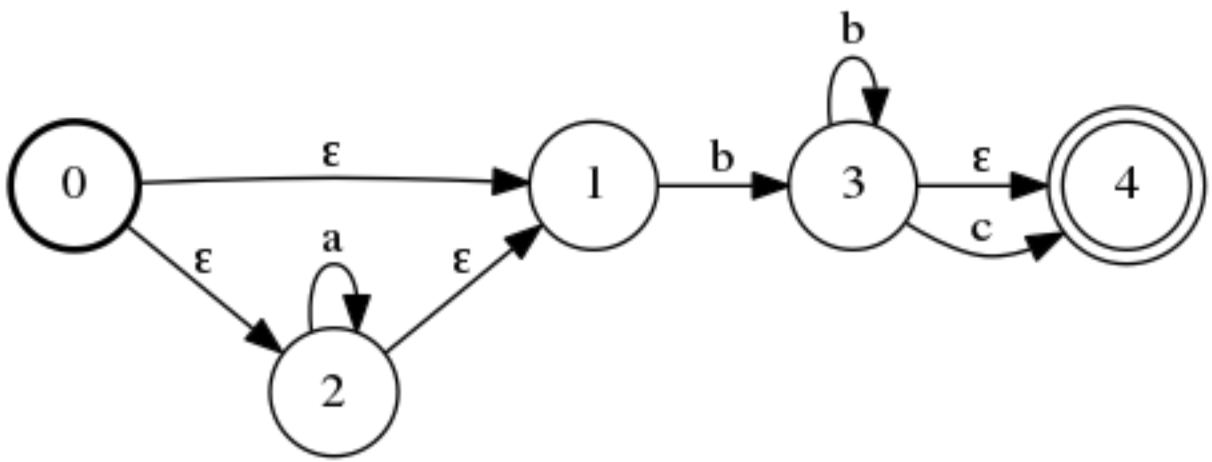
Lauri Karttunen
(Stanford university, Xerox Research etc.)

FINITE-STATE MORPHOLOGY SOFTWARE

- **HFST – Helsinki Finite-State Transducer Technology**
  - Open source software and demos
  - Python interface also available
  - https://www.kielipankki.fi/tools/demo/cgi-bin/omor/omordemo.bash
- **Lingsoft**
  - Commercial licenses?

FINITE-STATE AUTOMATON

A finite-state automaton (FSA) – or finite automaton – is a network consisting of nodes, which represent states, and directed arcs connecting the states, which represent transitions between states. Every arc is labeled with a symbol that is consumed from input. State transitions can also take place without consuming any input; these transitions are called epsilon transitions.

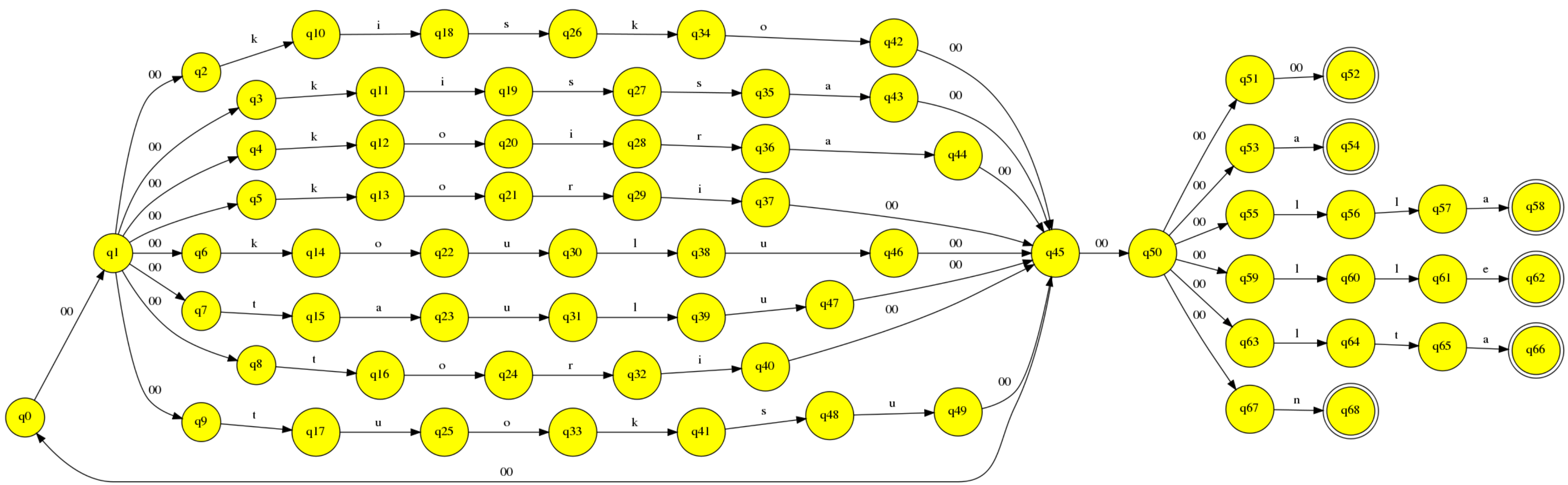


From: http://www.tylerpalsulich.com/blog/2015/05/12/introduction-to-finite-state-automata/

FINITE STATE AUTOMATON FOR SOME FINNISH NOUNS WITH CASE ENDINGS

STEMS

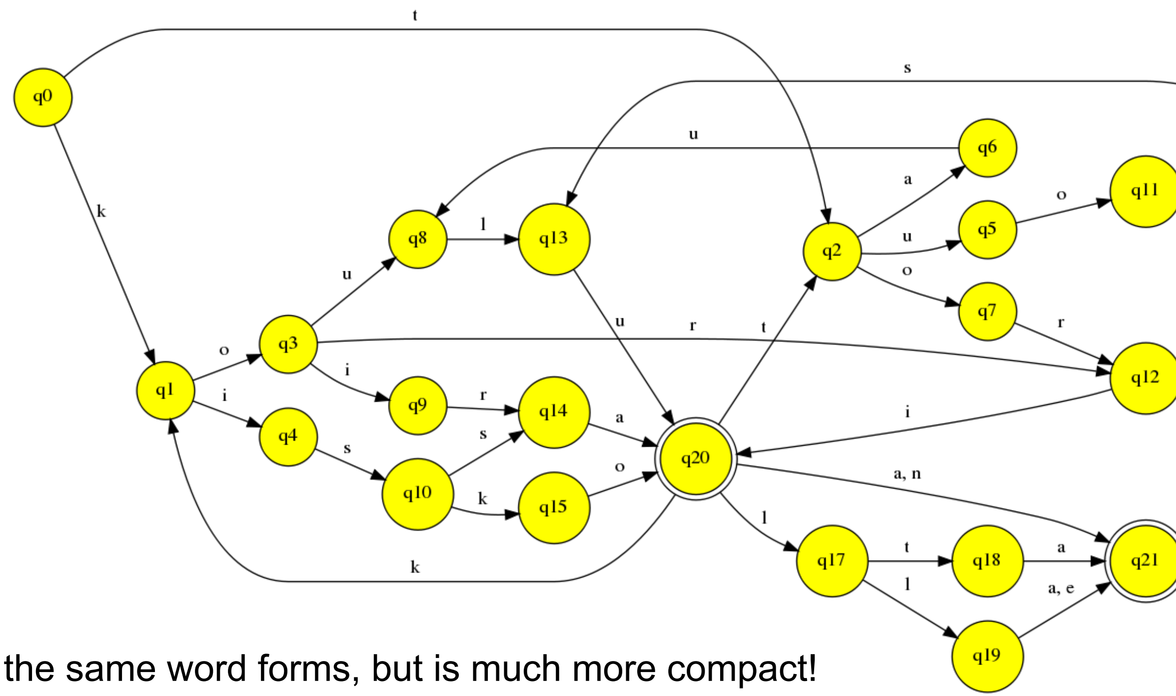
ENDINGS



Accepts input strings such as: kisko, kiskoa, kiskolla, kiskolle, kissa, kissaa, kissakoulu, …

The epsilon transition is written as ”00” and does not consume any input.

OPTIMIZED FINITE STATE AUTOMATON OF FINNISH NOUNS WITH CASE ENDINGS

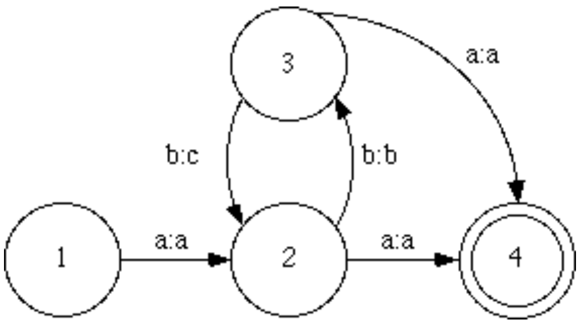


Accepts exactly the same word forms, but is much more compact!

Produced using algorithms for **epsilon removal**, **determinization** and **minimization** of finite state networks.

FINITE-STATE TRANSDUCER

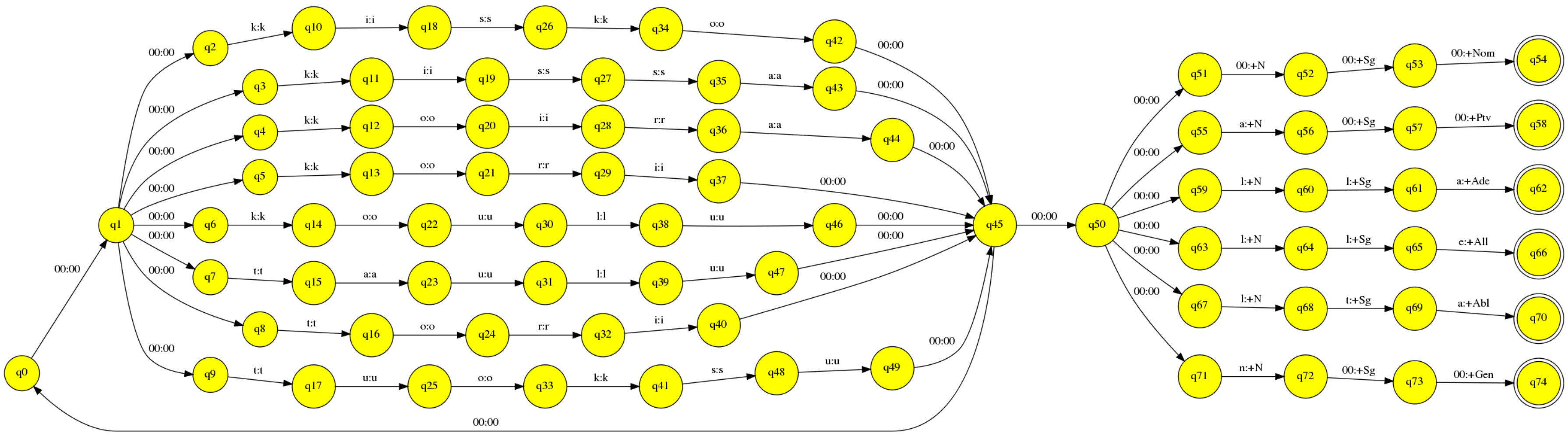
A finite-state transducer (FST) is a finite automaton for which each transition has an input label and an output label.



It recognizes whether the two strings are valid correspondences (or translations) of each other.

From: http://www-01.sil.org/pckimmo/v2/doc/Rules_2.html

FINITE STATE TRANSDUCER FOR SOME FINNISH NOUNS WITH CASE ENDINGS

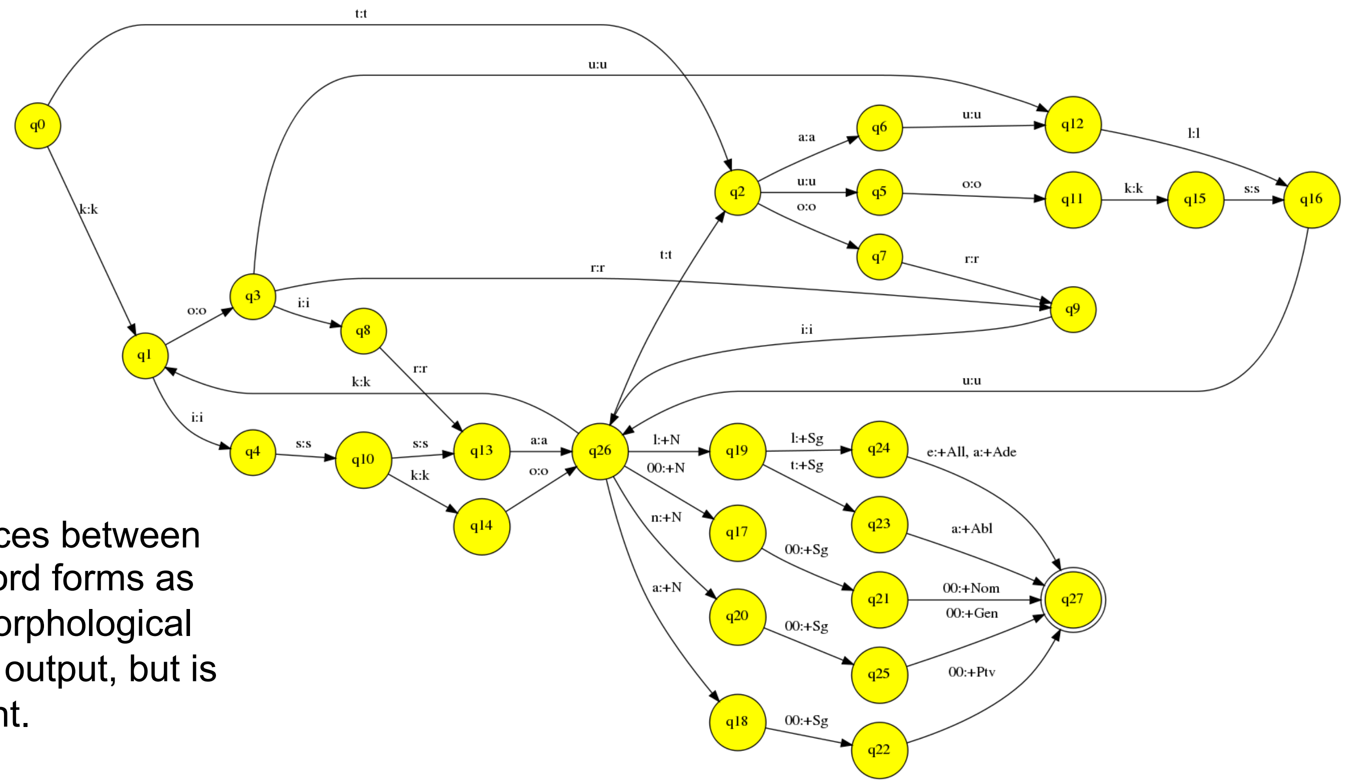


Transduces (translates) between word forms as input and morphological analyses as output:

Input: kisko ➔ Output: kisko+N+Sg+Nom
Input: kiskoa ➔ Output: kisko+N+Sg+Ptv
Input: kiskolla ➔ Output: kisko+N+Sg+Ade

Input: koululle ➔ Output: koulu+N+Sg+All
Input: kissakoulua ➔ Output: kissakoulu+N+Sg+Ptv
…

OPTIMIZED FINITE STATE TRANSDUCER FOR FINNISH NOUNS WITH CASE ENDINGS



Still transduces between the same word forms as input and morphological analyses as output, but is more efficient.

MORPHOLOGICAL ANALYSIS VS. GENERATION

- You have seen how a finite state transducer can be used as a **morphological analyzer**:

  Input: kisko ➔ Output: kisko+N+Sg+Nom
  Input: kiskoa ➔ Output: kisko+N+Sg+Ptv
  Input: kiskolla ➔ Output: kisko+N+Sg+Ade
  Input: koululle ➔ Output: koulu+N+Sg+All
  Input: kissakoulua ➔ Output: kissakoulu+N+Sg+Ptv
  …

- A **morphological generator** is simple to produce by inverting the transducer, such that input becomes output and vice versa:

  Input: kisko+N+Sg+Ade ➔ Output: kiskolla
  Input: koulu+N+Sg+Ptv ➔ Output: koulua
  …

EXAMPLE OF SUPERVISED MACHINE LEARNING: MORPHOLOGICAL REINFLECTION

- Learn morphological inflection patterns from tagged, incomplete data.

Cases \ Numbers	Singular	Plural
Nominative	**susi**	**sudet**
Genitive	**suden**	**?**
Partitive	**sutta**	**susia**
Inessive	**sudessa**	**?**
Elative	**?**	**susista**
Illative	**suteen**	**susiin**
Adessive	**?**	**?**

Cases \ Numbers	Singular	Plural
Nominative	**käsi**	**?**
Genitive	**käden**	**käsien, kätten**
Partitive	**?**	**?**
Inessive	**kädessä**	**käsissä**
Elative	**kädestä**	**?**
Illative	**?**	**käsiin**
Adessive	**kädellä**	**?**

- Check out the SIGMORPHON shared tasks: https://sigmorphon.github.io/sharedtasks/2019/

APPROACH 3: SEGMENTATION

MORPHOLOGICAL SEGMENTATION

- Suitable for agglutinative languages; problems with fusional languages.
- Applications that need only the surface forms:
  - speech recognition, text prediction, language identification, etc.
- Can be considered as a labeling problem:

Binary labels for boundaries:

	1	0	1	0	0	0	0	0	1	1	0	0	0	1
#	u	n	r	e	l	a	t	e	d	n	e	s	s	#

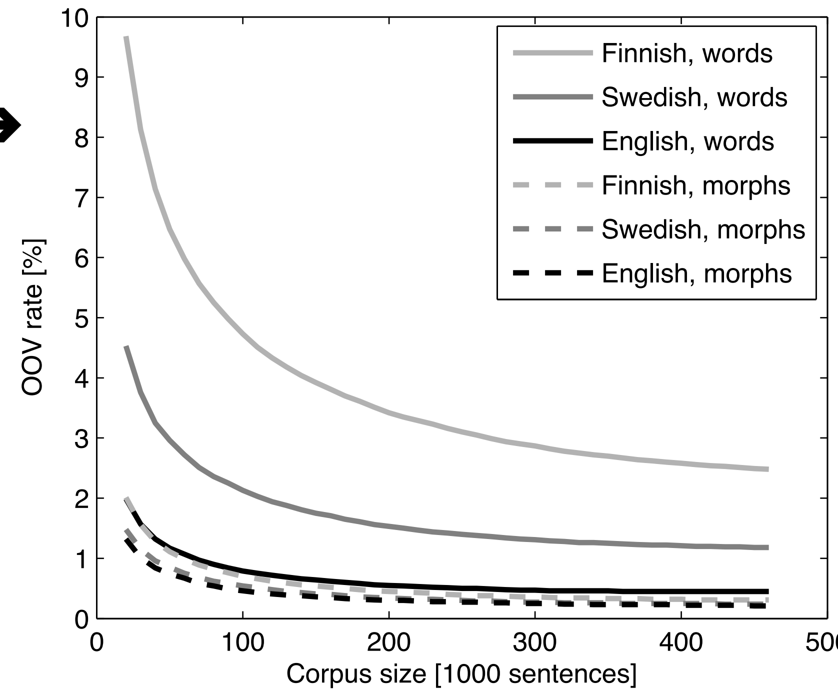
BIES label set:

	B	E	B	I	I	I	I	E	S	B	I	I	E	
#	u	n	r	e	l	a	t	e	d	n	e	s	s	#

- A related task is word segmentation for languages written without spaces between words; e.g., Chinese word segmentation.

EFFECT OF MORPH-LEVEL MODELING

- Proportion of **out-of-vocabulary (OOV)** units in different languages as a function of the training corpus size, estimated form the Europarl corpus ➔
- By using morphs instead of words as basic units in the NLP system, the OOV rate is reduced.



Morphological segmentation:

SUPERVISED LEARNING

- Train a model that predicts the label y_i of the current character x_i given the characters and the previous labels: $P(y_i \mid (x_0, \ldots, x_n); (y_0, \ldots, y_{i-1}))$
- E.g., Hidden Markov Models, Conditional Random Fields

Morphological segmentation:

UNSUPERVISED LEARNING, METHOD 1

- **Zellig Harris** proposed the first(?) unsupervised morpheme segmentation algorithm (1955)
- Computer experiment carried out in 1967
  - Test data consisted of 48 words…
- Principle:
  - Morpheme boundaries are proposed at intra-word locations with a **peak** in **successor** and **predecessor variety**.
  - Demonstrated on the next slides.

Zellig Harris's morpheme segmentation model:

SUCCESSOR VARIETY

Test word:
readable

Corpus:
able
ape
beatable
fixable
read
readable
reading
reads
red
rope
ripe

Prefix	Successor variety

From: Hafer & Weiss: Word segmentation by letter successor varieties (1974)

Zellig Harris's morpheme segmentation model:

SUCCESSOR VARIETY

Test word:

readable

Corpus:

able
ape
beatable
fixable
read
readable
reading
reads
red
rope
ripe

Prefix	Successor variety	
r	3	e, o, i

From: Hafer & Weiss: Word segmentation by letter successor varieties (1974)

Zellig Harris's morpheme segmentation model:

SUCCESSOR VARIETY

Test word:
readable

Corpus:
able
ape
beatable
fixable
read
readable
reading
reads
red
rope
ripe

Prefix	Successor variety	
r	3	e, o, i
re	2	a, d

From: Hafer & Weiss: Word segmentation by letter successor varieties (1974)

Zellig Harris's morpheme segmentation model:
SUCCESSOR VARIETY

Test word:
readable

Corpus:
able
ape
beatable
fixable
read
readable
reading
reads
red
rope
ripe

Prefix	Successor variety	
r	3	e, o, i
re	2	a, d
rea	1	d

From: Hafer & Weiss: Word segmentation by letter successor varieties (1974)

Zellig Harris's morpheme segmentation model:

SUCCESSOR VARIETY

Test word:
readable

Corpus:
able
ape
beatable
fixable
read_
readable
reading
reads
red
rope
ripe

Prefix	Successor variety	
r	3	e, o, i
re	2	a, d
rea	1	d
read	3*	a, i, s

From: Hafer & Weiss: Word segmentation by letter successor varieties (1974)

Zellig Harris's morpheme segmentation model:

SUCCESSOR VARIETY

Test word:
readable

Corpus:
able
ape
beatable
fixable
read
readable
reading
reads
red
rope
ripe

Prefix	Successor variety	
r	3	e, o, i
re	2	a, d
rea	1	d
read	3*	a, i, s
reada	1	b

← peak here
successor variety higher than before and after

From: Hafer & Weiss: Word segmentation by letter successor varieties (1974)

Zellig Harris's morpheme segmentation model:
SUCCESSOR VARIETY

Test word:
readable

Corpus:
able
ape
beatable
fixable
read
readable
reading
reads
red
rope
ripe

Prefix	Successor variety	
r	3	e, o, i
re	2	a, d
rea	1	d
read	3*	a, i, s
reada	1	b
readab	1	l

From: Hafer & Weiss: Word segmentation by letter successor varieties (1974)

Zelig Harris's morpheme segmentation model:
SUCCESSOR VARIETY

Test word:
readable

Corpus:
able
ape
beatable
fixable
read
readable
reading
reads
red
rope
ripe

Prefix	Successor variety	
r	3	e, o, i
re	2	a, d
rea	1	d
read	3*	a, i, s
reada	1	b
readab	1	l
readabl	1	e

From: Hafer & Weiss: Word segmentation by letter successor varieties (1974)

Zellig Harris's morpheme segmentation model:

SUCCESSOR VARIETY

Test word:
readable

Corpus:
able
ape
beatable
fixable
read
readable_
reading
reads
red
rope
ripe

Prefix	Successor variety	
r	3	e, o, i
re	2	a, d
rea	1	d
read	3*	a, i, s
reada	1	b
readab	1	l
readabl	1	e
readable	1*	-

From: Hafer & Weiss: Word segmentation by letter successor varieties (1974)

Zellig Harris's morpheme segmentation model:

PREDECESSOR VARIETY

Test word:
readable

Corpus:
able
ape
beatable
fixable
read
readable
reading
reads
red
rope
ripe

Suffix	Predecessor variety

From: Hafer & Weiss: Word segmentation by letter successor varieties (1974)

Zellig Harris's morpheme segmentation model:

PREDECESSOR VARIETY

Test word:
readable

Corpus:
able
ape
beatable
fixable
read
readable
reading
reads
red
rope
ripe

Suffix	Predecessor variety	
e	2	l, p

From: Hafer & Weiss: Word segmentation by letter successor varieties (1974)

Zellig Harris's morpheme segmentation model:

PREDECESSOR VARIETY

Test word:
readable

Corpus:
able
ape
beatable
fixable
read
readable
reading
reads
red
rope
ripe

Suffix	Predecessor variety	
e	2	l, p
le	1	b

From: Hafer & Weiss: Word segmentation by letter successor varieties (1974)

Zellig Harris's morpheme segmentation model:

PREDECESSOR VARIETY

Test word:
readable

Corpus:
able
ape
beatable
fixable
read
readable
reading
reads
red
rope
ripe

Suffix	Predecessor variety	
e	2	l, p
le	1	b
ble	1	a

From: Hafer & Weiss: Word segmentation by letter successor varieties (1974)

Zellig Harris's morpheme segmentation model:
PREDECESSOR VARIETY

Test word:
readable

Corpus:
_able
ape
beatable
fixable
read
readable
reading
reads
red
rope
ripe

Suffix	Predecessor variety	
e	2	l, p
le	1	b
ble	1	a
able	3*	d, t, x

From: Hafer & Weiss: Word segmentation by letter successor varieties (1974)

Zellig Harris's morpheme segmentation model:

PREDECESSOR VARIETY

Test word:
readable

Corpus:
able
ape
beatable
fixable
read
re<u>a</u>dable
reading
reads
red
rope
ripe

Suffix	Predecessor variety	
e	2	l, p
le	1	b
ble	1	a
able	3*	d, t, x
dable	1	a

← peak here
predecessor variety higher than before and after

From: Hafer & Weiss: Word segmentation by letter successor varieties (1974)

Zellig Harris's morpheme segmentation model:

PREDECESSOR VARIETY

Test word:
readable

Corpus:
able
ape
beatable
fixable
read
r<u>e</u>adable
reading
reads
red
rope
ripe

Suffix	Predecessor variety	
e	2	l, p
le	1	b
ble	1	a
able	3*	d, t, x
dable	1	a
adable	1	e

From: Hafer & Weiss: Word segmentation by letter successor varieties (1974)

Zellig Harris's morpheme segmentation model:

PREDECESSOR VARIETY

Test word:
readable

Corpus:
able
ape
beatable
fixable
read
readable
reading
reads
red
rope
ripe

Suffix	Predecessor variety	
e	2	l, p
le	1	b
ble	1	a
able	3*	d, t, x
dable	1	a
adable	1	e
eadable	1	r

From: Hafer & Weiss: Word segmentation by letter successor varieties (1974)

Zellig Harris's morpheme segmentation model:

PREDECESSOR VARIETY

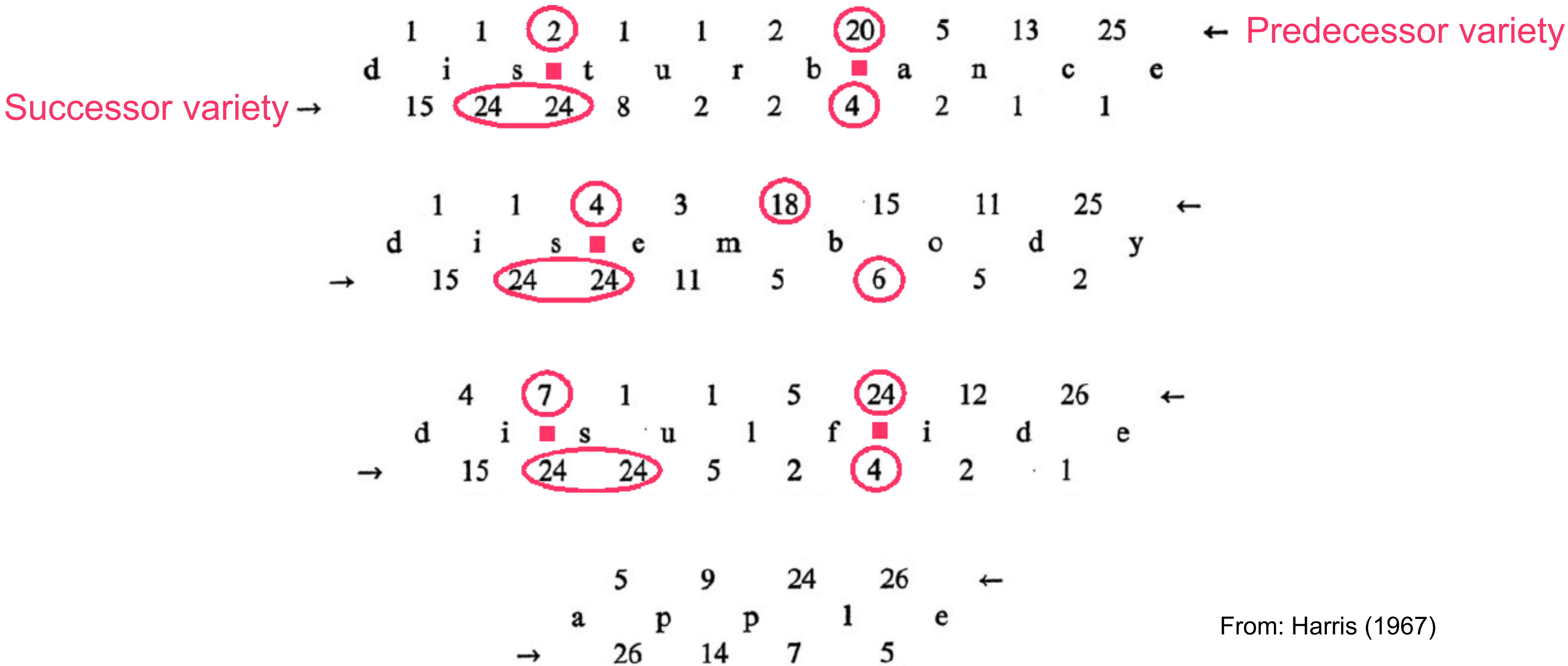
Test word:
readable

Corpus:
able
ape
beatable
fixable
read
_readable
reading
reads
red
rope
ripe

Suffix	Predecessor variety	
e	2	l, p
le	1	b
ble	1	a
able	3*	d, t, x
dable	1	a
adable	1	e
eadable	1	r
readable	1*	-

From: Hafer & Weiss: Word segmentation by letter successor varieties (1974)

Zellig Harris's morpheme segmentation model:
INSERT A BOUNDARY WHERE THE PEAKS "MEET"



From: Harris (1967)

Morphological segmentation:

UNSUPERVISED LEARNING, METHOD 2

- We want to send a vocabulary (= word list) of some language over a channel with limited band-width.
- We want to compress the vocabulary.
- What regularities can we exploit?
- What about morphemes, the smallest meaning-bearing units of language?
- The method is called *Morfessor* (Creutz & Lagus, 2002)

	? →	
...		...
aamu		aamu
aamua		aamu a
aamuaurinko		aamu aurinko
aamukahvi		aamu kahvi
aamuksi		aamu ksi
aamulehti		aamu lehti
aamulla		aamu lla
aamun		aamu n
aamunaamasi		aamu naama si
aamupalalla		aamu pala lla
aamupalan		aamu pala n
aamupostia		aamu posti a
aamupäivä		aamu päivä
aamupäivällä		aamu päivä llä
aamuyö		aamu yö
aamuyöllä		aamu yö llä
aamuyöstä		aamu yö stä
...		...

Morfessor:

TWO-PART CODE

- Instead of sending over the vocabulary as it is, we split it into two parts:
  1. a fairly compact lexicon of morphs: “aamu”, “aurinko”, “ksi”, “lla”, …
  2. the word vocabulary expressed as sequences of morphs

Morfessor:

TWO-PART CODE

- Instead of sending over the vocabulary as it is, we split it into two parts:
  1. a fairly compact lexicon of morphs: “aamu”, “aurinko”, “ksi”, “lla”, …
  2. the word vocabulary expressed as sequences of morphs

- Since we are doing unsupervised learning, we do not know the correct answer.

- Our target is to **minimize the combined code length** of:
  1. the code length of the morph lexicon
  2. plus the code length of the word vocabulary expressed using the morph lexicon.

Morfessor:

TWO-PART CODE

- Instead of sending over the vocabulary as it is, we split it into two parts:
  1. a fairly compact lexicon of morphs: “aamu”, “aurinko”, “ksi”, “lla”, …
  2. the word vocabulary expressed as sequences of morphs

- Since we are doing unsupervised learning, we do not know the correct answer.

- Our target is to **minimize the combined code length** of:
  1. the code length of the morph lexicon
  2. plus the code length of the word vocabulary expressed using the morph lexicon.

- There are two theories that operate on two-part codes like this:
  - (Two-part code version of) **Minimum Description Length** (MDL)
  - **Minimum Message Length** (MML)

Morfessor:

CODE LENGTH OF THE MORPH LEXICON

- Let us assume, for simplicity, that there are 32 different letters in our alphabet.
- This means we need 5 bits to encode one letter, because $2^5 = 32$:
  - The letter ‘a’ could have the code 00000.
  - The letter ‘b’ could have the code 00001.
  - The letter ‘c’ could have the code 00010.
  - The letter ‘d’ could have the code 00011, etc.

Morfessor:

CODE LENGTH OF THE MORPH LEXICON

- Let us assume, for simplicity, that there are 32 different letters in our alphabet.
- This means we need 5 bits to encode one letter, because $2^5 = 32$:
  - The letter ‘a’ could have the code 00000.
  - The letter ‘b’ could have the code 00001.
  - The letter ‘c’ could have the code 00010.
  - The letter ‘d’ could have the code 00011, etc.
- We could send over a four-morph lexicon as the following string: **`aamu#aurinko#ksi#lla##`** (binary: 000000000001000 …)
- Here we use the hash tag ‘#’ as a morph separator and use two hash tags ‘##’ to indicate that the lexicon ends.
- The lexicon string contains 22 characters.
- Thus, the code length of this lexicon is 22 * 5 bits = 110 bits.

Morfessor:

CODE LENGTH OF THE CORPUS (1)

- Each word in our word vocabulary (or hereafter called **corpus**) is expressed as a concatenation of morphs:
  - **`aamu`** is expressed as *Morph1* + *EoW* (= End of Word)
  - **`aamuksi`** is expressed as *Morph1* + *Morph3* + *EoW*
  - **`aamulla`** is expressed as *Morph1* + *Morph4* + *EoW*
  - **`aamuaurinko`** is expressed as *Morph1* + *Morph2* + *EoW*
- How are the symbols (or ”variables”) *Morph1*, *Morph2*, etc encoded?

Morfessor:

CODE LENGTH OF THE CORPUS (2)

- For instance, if there were 64 different morphs, and all morphs were as frequently used, we could use a fixed 6-bit code for every morph (because $2^6 = 64$).
  - The first morph would have the code 000000.
  - The second morph would have the code 000001.
  - The third morph would have the code 000010.
  - The fourth morph would have the code 000011, etc.

Morfessor:

CODE LENGTH OF THE CORPUS (2)

- For instance, if there were 64 different morphs, and all morphs were as frequently used, we could use a fixed 6-bit code for every morph (because $2^6 = 64$).
  - The first morph would have the code 000000.
  - The second morph would have the code 000001.
  - The third morph would have the code 000010.
  - The fourth morph would have the code 000011, etc.

- However, the morph distribution of a natural language is not uniform at all:
  - Some morphs are very frequent, such as ‘ksi’ and ‘lla’.
  - Other morphs are infrequent, such as ‘aurinko’.

Morfessor:

CODE LENGTH OF THE CORPUS (3)

- Suppose that our morph-segmented "corpus" (= word vocabulary) consists of 8 words and looks like this.
  - The underscore '_' represents the end-of-word morph.

```
aamu aurinko a _
aamu ksi ko _
aamu lla kin han _
aamu pala lla _
pala a _
pala ksi _
posti n kulje t us _
suu pala _
```

- In this segmentation there are 32 morph **tokens**, representing 16 different morph **types**.
  - The morph frequencies are as follows:

8 _	1 kulje
2 a	2 lla
4 aamu	1 n
1 aurinko	4 pala
1 han	1 posti
1 kin	1 suu
1 ko	1 t
2 ksi	1 us

Morfessor:

CODE LENGTH OF THE CORPUS (4)

- It turns out that the optimal code length of a symbol is the **negative logprob** (with base 2) of the symbol in the data.
  - The probability of a symbol is the frequency of the symbol in the data divided by the total frequency of all symbols in the data.
    - For instance, *Prob*("aamu") = 4/32 = 1/8 = 0.125.
  - The negative logprob of a symbol is: $-\log_2$ *Prob*(symbol)
    - For instance, neglogprob("aamu") = $-\log_2 1/8 = \log_2 8 = 3$ (because $2^3 = 8$)
  - Frequent morphs will have shorter codes than rare morphs.

Morfessor:

CODE LENGTH OF THE CORPUS (4)

- It turns out that the optimal code length of a symbol is the **negative logprob** (with base 2) of the symbol in the data.
  - The probability of a symbol is the frequency of the symbol in the data divided by the total frequency of all symbols in the data.
    - For instance, *Prob*("aamu") = 4/32 = 1/8 = 0.125.
  - The negative logprob of a symbol is: $-\log_2$ *Prob*(symbol)
    - For instance, neglogprob("aamu") = $-\log_2 1/8 = \log_2 8 = 3$ (because $2^3 = 8$)
  - Frequent morphs will have shorter codes than rare morphs.

- The code needs to be a so-called **prefix code** in order to be unambiguous:
  - When symbols have different code lengths, it must be clear to the decoder at every time how many bits to expect for that symbol.
  - For instance, if there is one symbol that has code length = 2, then it could have the code '00'.
  - This means that no other symbol is allowed to have a code that starts with '00', because then this prefix would be ambiguous, and the system would not know when the whole symbol has been read.

- Let's do the maths for our morph set…

Morfessor:

CODE LENGTH OF THE CORPUS (5)

Morph	Frequency	Probability	Neglogprob	Binary prefix code	Morph	Frequency	Probability	Neglogprob	Binary prefix code
_	8	0.25	2	00	**kin**	1	0.03125	5	11000
aamu	4	0.125	3	010	**ko**	1	0.03125	5	11001
pala	4	0.125	3	011	**kulje**	1	0.03125	5	11010
a	2	0.0625	4	1000	**n**	1	0.03125	5	11011
ksi	2	0.0625	4	1001	**posti**	1	0.03125	5	11100
lla	2	0.0625	4	1010	**suu**	1	0.03125	5	11101
aurinko	1	0.03125	5	10110	**t**	1	0.03125	5	11110
han	1	0.03125	5	10111	**us**	1	0.03125	5	11111

In the “Binary prefix code” columns above I have underlined the part of the code, after which the decoder knows how long the code for that symbol is.

Morfessor:

CODE LENGTH OF THE CORPUS (6)

- The code for our corpus is thus: 0101011010000001010011100100 …

- The total code length of the corpus is:

  8 * 2 bits + (4 + 4) * 3 bits

  + (2 + 2 + 2) * 4 bits + 10 * 5 bits

  = 114 bits

```
aamu aurinko a _
aamu ksi ko _
aamu lla kin han _
aamu pala lla _
pala a _
pala ksi _
posti n kulje t us _
suu pala _
```

8 _	1 kulje
2 a	2 lla
4 aamu	1 n
1 aurinko	4 pala
1 han	1 posti
1 kin	1 suu
1 ko	1 t
2 ksi	1 us

Morfessor:

TO CONSIDER

- In real situations, we don't get tidy integer-number code lengths, such as 2, 3, 4 in the example above.
- Instead, we can get any real-valued number of bits, such as 5.37 or 1.111.
  - There is a proof by Jorma Rissanen (the inventor of MDL) that this does not matter.
- Also, the base of the logarithm does not matter either: we don't have to calculate in bits (with base 2), but can use **nats** (with base *e* for the natural logarithm).
- Furthermore, we are not really interested in the actual codes of our symbols, because we are not building an encoder/decoder.
  - We use this encode-decode methodology as a "metaphor" to learn a morph segmentation in an unsupervised way.
  - Maximum A Posteriori (MAP) optimization is a fully equivalent method that does not deal with code lengths at all, just plain probabilities.
- Also on the lexicon side, we could have used variable-length codes instead of fixed-length codes for the letters of the alphabet.
- There are other parts of the mathematical formulation that I have been left out, for simplicity.

Morfessor:
HOW TO FIND THE BEST SEGMENTATION

- We use a **search algorithm** that tests different morph segmentations and calculates the two-part code length: code length of lexicon plus code length of corpus.

- The algorithm stops when it has reached a minimum, the shortest code length it can find.

Morfessor:

DIFFERENT MORPH SPLITTING SCENARIOS

1. The algorithm splits every word into individual letters, such as: **`a a m u p a l a`**
   - The code length of the lexicon will be very small, because it only contains 32 morphs: every letter of the alphabet is its own morph.
   - The code length of the corpus will be large, because it consists of a very high number of morph symbols.
   - As a consequence, the combined code length will be fairly large.

Morfessor:

DIFFERENT MORPH SPLITTING SCENARIOS

1. The algorithm splits every word into individual letters, such as: `a a m u p a l a`
   - The code length of the lexicon will be very small, because it only contains 32 morphs: every letter of the alphabet is its own morph.
   - The code length of the corpus will be large, because it consists of a very high number of morph symbols.
   - As a consequence, the combined code length will be fairly large.

2. The algorithm does not split any word at all; each word is its own morph, such as **`aamupala`**.
   - The code length of the corpus will be fairly small, because it contains the smallest number of morph symbols possible.
   - The code length of the lexicon will be large, because every word form is there as its own morph.
   - As a consequence, the combined code length will be fairly large.

Morfessor:

DIFFERENT MORPH SPLITTING SCENARIOS

1. The algorithm splits every word into individual letters, such as: **`a a m u p a l a`**
   - The code length of the lexicon will be very small, because it only contains 32 morphs: every letter of the alphabet is its own morph.
   - The code length of the corpus will be large, because it consists of a very high number of morph symbols.
   - As a consequence, the combined code length will be fairly large.
2. The algorithm does not split any word at all; each word is its own morph, such as **`aamupala`**.
   - The code length of the corpus will be fairly small, because it contains the smallest number of morph symbols possible.
   - The code length of the lexicon will be large, because every word form is there as its own morph.
   - As a consequence, the combined code length will be fairly large.
3. Balanced morph splitting, such as: **`aamu pala`**.
   - The shortest combined code length is achieved by an optimal balance (a "compromise"): not the shortest possible lexicon, nor the shortest possible representation of the corpus.

Morfessor:

DOES THIS WORK?

English example output from (the earliest context-insensitive version of) *Morfessor*, which corresponds fairly closely to the model described above:

abandon ed	absolute	differ	present ed
abandon ing	absolute ly	differ ence	present ing
abb	absorb	differ ence s	present ly
abb y	absorb ing	differ ent	present s
ab del	absurd	differ ent ial	pre serve
able	absurd ity	differ ent ly	pre serve s
ab normal	ab t	differ ing	provide s
a board	a bu	difficult	pro vi d ing
ab out	abuse	difficult ies	pull ed
a broad	abuse d	difficult y	pull ers
ab rupt ly	abuse r s	dig	pull ing
ab s ence	abuse s	dig est	pump
ab s ent	ab y s s	dig it al	pump ed
ab s ent ing	ac cent	dig li pur	pump ing

Morfessor: **ERROR ANALYSIS**

Morphs that make sense in some context appear in contexts where they don’t really belong. There are also instances of over- and under-segmentation.

abandon ed	absolute	differ	present ed
abandon ing	absolute ly	differ ence	present ing
abb	absorb	differ ence s	present ly
abb y	absorb ing	differ ent	present s
ab del	absurd	differ ent ial	pre serve
able	absurd ity	differ ent ly	pre serve s
ab normal	ab t	differ ing	provide s
a board	a bu	difficult	pro vi d ing
ab out	abuse	difficult ies	pull ed
a broad	abuse d	difficult y	pull ers
ab rupt ly	abuse r s	dig	pull ing
ab s ence	abuse s	dig est	pump
ab s ent	ab y s s	dig it al	pump ed
ab s ent ing	ac cent	dig li pur	pump ing

Morfessor:
IMPROVED MODEL

Software available at:
http://www.cis.hut.fi/projects/morpho/

- A later context-sensitive version of Morfessor introduces three categories: stem (STM), prefix (PRE) and suffix (SUF) that each morph must belong to.
- A word form must have the structure of the following regular expression: (PRE* STM SUF*)+
- From the updated examples below, you can see that many issues have been fixed, but the model is still fairly crude; for instance, it suggests two consecutive s-suffixes in the word “abyss”: aby s s.

abandon/STM ed/SUF	absolute/STM	differ/STM	present/STM ed/SUF
abandon/STM ing/SUF	absolute/STM ly/SUF	differ/STM ence/STM	present/STM ing/SUF
abb/STM	absorb/STM	differ/STM ence/STM s/SUF	present/STM ly/SUF
abby/STM	absorb/STM ing/SUF	different/STM	present/STM s/SUF
abdel/STM	absurd/STM	differential/STM	preserv/STM e/SUF
able/STM	absurd/STM ity/SUF	different/STM ly/SUF	preserv/STM e/SUF s/SUF
ab/STM normal/STM	abt/STM	differ/STM ing/SUF	provide/STM s/SUF
aboard/STM	abu/STM	difficult/STM	provi/STM ding/STM
about/STM	abuse/STM	difficult/STM i/SUF es/SUF	pull/STM ed/SUF
abroad/STM	abuse/STM d/SUF	difficult/STM y/SUF	pull/STM er/SUF s/SUF
abrupt/STM ly/SUF	ab/STM users/STM	dig/STM	pull/STM ing/SUF
absence/STM	abuse/STM s/SUF	digest/STM	pump/STM
absent/STM	aby/STM s/SUF s/SUF	digital/STM	pump/STM ed/SUF
absent/STM ing/SUF	accent/STM	diglipur/STM	pump/STM ing/SUF

Pragmatic segmentation approach:

METHOD 3: BYTE PAIR ENCODING (BPE)

- Simple data compression algorithm (like Morfessor)

- Repeat in multiple steps: The *most common* pair of consecutive bytes (characters) of data is replaced with a byte (character) that does not occur within that data:

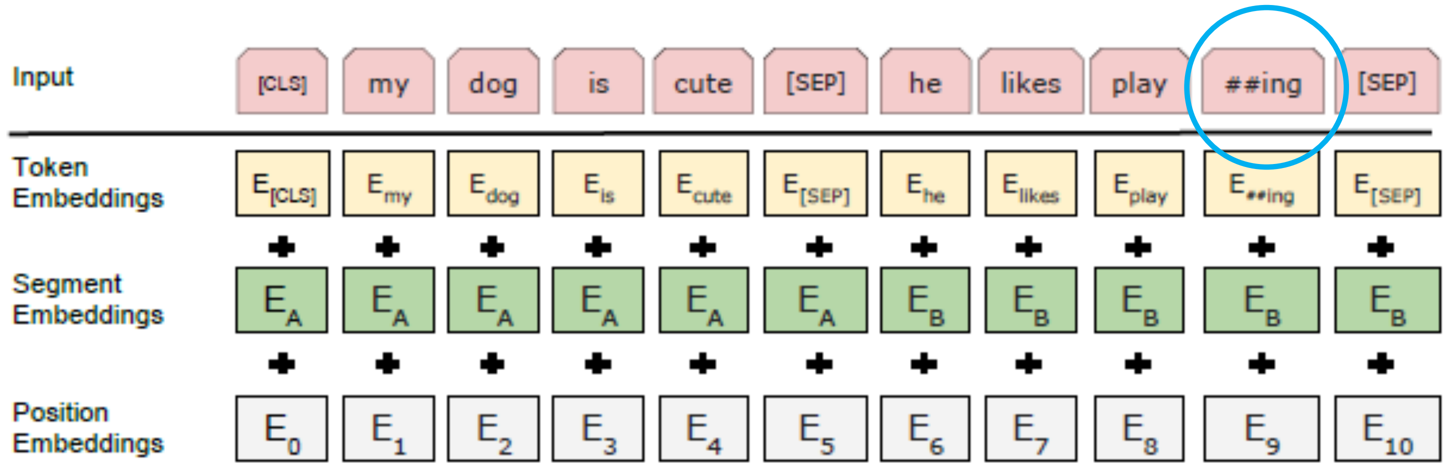
```
1. aaabdaaabac
2. Z = aa -> ZabdZabac
3. Y = ab, Z = aa -> ZYdZYac
4. X=ZY, Y = ab, Z = aa -> XdXac
```

- Stop when you have reached the number of **subword units** you want or when there is no byte pair that occurs more than once.

For more info, see Wikipedia, Philip Gage (1994) or Sennrich, Haddow, and Birch (2016).

SUBWORD UNITS OBTAINED USING BPE OFTEN USED AS INPUT VECTORS TO NEURAL NETWORKS

- For instance, the widely used neural language model BERT creates input embeddings based on a BPE segmentation, even for English input:



Extension of BPE:

METHOD 3++: SENTENCE PIECE

- Supports two segmentation algorithms: BPE and a unigram language model
- **Whitespace is treated as a basic symbol**
  - *Raw text:* Hello_world.

  ➢ *Tokenized:* [Hello] [_wor] [ld] [.]

  - *Raw text:* こんにちは世界。(Hello world.)

  ➢ *Tokenized:* [こんにちは] [世界] [。]

For more info, see https://github.com/google/sentencepiece

Extension of BPE:

METHOD 3++: SENTENCE PIECE

- Sampling of multiple alternatives

```
>>> import sentencepiece as spm
>>> s = spm.SentencePieceProcessor(model_file='spm.model')
>>> for n in range(5):
... s.encode('New York', out_type=str, enable_sampling=True, alpha=0.1, nbest=-1)
...
['▁', 'N', 'e', 'w', '▁York']
['▁', 'New', '▁York']
['▁', 'New', '▁Y', 'o', 'r', 'k']
['▁', 'New', '▁York']
['▁', 'New', '▁York']
```

For more info, see https://github.com/google/sentencepiece

APPROACH 4: IMPLICIT MODELING

FASTTEXT: OVERLAPPING SUB-WORD SEGMENTS

- The *fastText* model is based on the *skipgram* model of the *word2vec* package.
- In *fastText*, word embeddings are created by summing overlapping subword vectors together.
- Also a vector for the whole word is included, if available (not possible for OOV words).

Piotr Bojanowski, Edouard Grave, Armand Joulin and Tomas Mikolov: [Enriching Word Vectors with Subword Information.] Transactions of the Association for Computational Linguistics, Vol 5, 2017.

FASTTEXT: OVERLAPPING SUB-WORD SEGMENTS

Each word w is represented as a bag of character n-gram. We add special boundary symbols $<$ and $>$ at the beginning and end of words, allowing to distinguish prefixes and suffixes from other character sequences. We also include the word w itself in the set of its n-grams, to learn a representation for each word (in addition to character n-grams). Taking the word *where* and $n = 3$ as an example, it will be represented by the character n-grams:

`<wh, whe, her, ere, re>`

and the special sequence

`<where>`.

Note that the sequence `<her>`, corresponding to the word *her* is different from the tri-gram `her` from the word *where*. In practice, we extract all the n-grams for n greater or equal to 3 and smaller or equal to 6.

- The *fastText* model is based on the *skipgram* model of the *word2vec* package.
- In *fastText*, word embeddings are created by summing overlapping subword vectors together.
- Also a vector for the whole word is included, if available (not possible for OOV words).

Piotr Bojanowski, Edouard Grave, Armand Joulin and Tomas Mikolov: Enriching Word Vectors with Subword Information. Transactions of the Association for Computational Linguistics, Vol 5, 2017.

FASTTEXT: OVERLAPPING SUB-WORD SEGMENTS

	word	n-grams		
DE	autofahrer	fahr	fahrer	auto
	freundeskreis	kreis	kreis>	<freun
	grundwort	wort	wort>	grund
	sprachschule	schul	hschul	sprach
	tageslicht	licht	gesl	tages
EN	anarchy	chy	<anar	narchy
	monarchy	monarc	chy	<monar
	kindness	ness>	ness	kind
	politeness	polite	ness>	eness>
	unlucky	<un	cky>	nlucky
	lifetime	life	<life	time
	starfish	fish	fish>	star
	submarine	marine	sub	marin
	transform	trans	<trans	form
FR	finirais	ais>	nir	fini
	finissent	ent>	finiss	<finis
	finissions	ions>	finiss	sions>

Table 6: Illustration of most important character n-grams for selected words in three languages. For each word, we show the n-grams that, when removed, result in the most different representation.

Piotr Bojanowski, Edouard Grave, Armand Joulin and Tomas Mikolov: Enriching Word Vectors with Subword Information. Transactions of the Association for Computational Linguistics, Vol 5, 2017.

FASTTEXT: OVERLAPPING SUB-WORD SEGMENTS

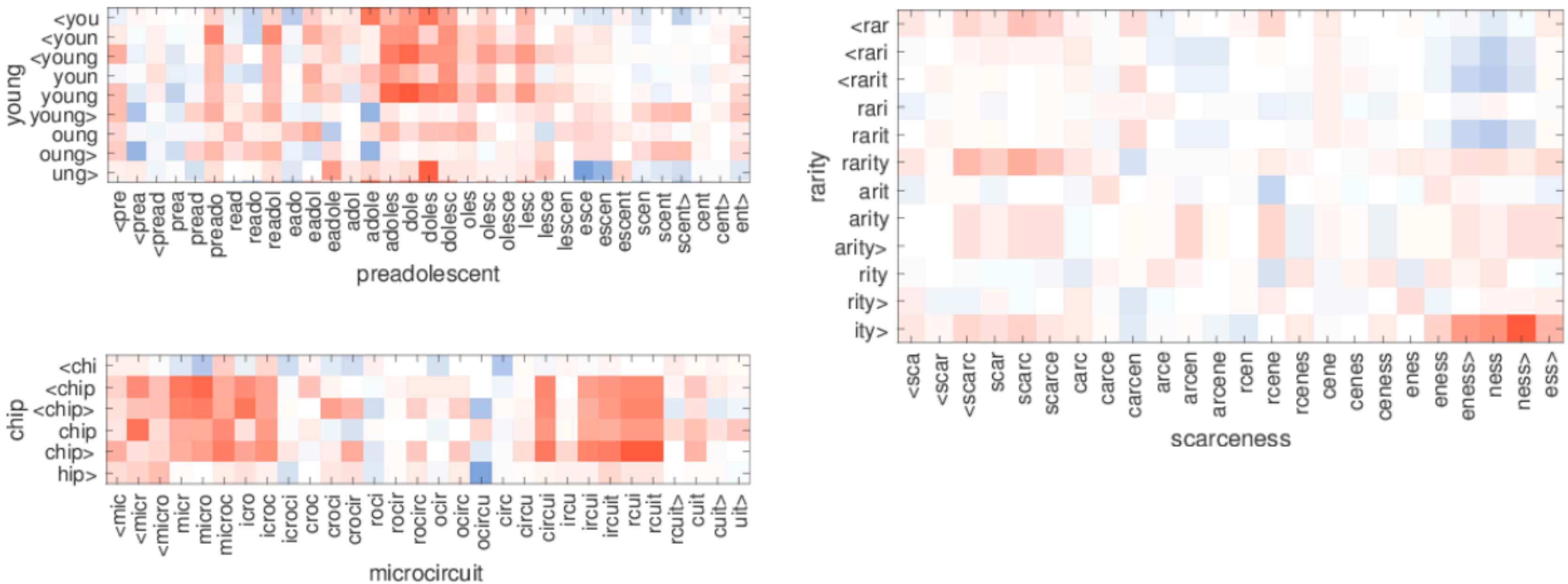
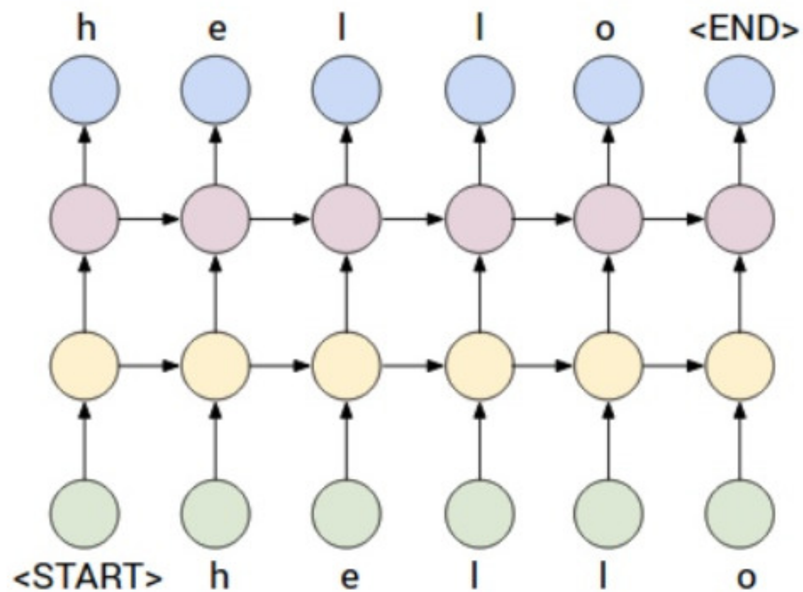


Figure 2: Illustration of the similarity between character n-grams in out-of-vocabulary words. For each pair, only one word is OOV, and is shown on the x axis. Red indicates positive cosine, while blue negative.

CHARACTER-LEVEL EMBEDDINGS

- No morphology used!
- The neural network learns what it needs (hopefully…) about the internal structure of words.
- Each character (letter) is treated as its own ”word” vector.
- Computationally heavy but some people believe this will be the standard approach in the future.



THE END

THANK YOU!