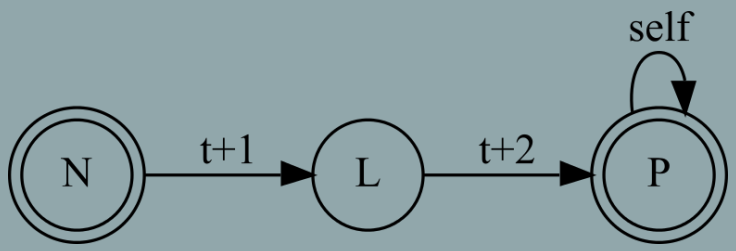
LING-362

Introduction to Natural Language Processing

From HMM to Sequence Labeling



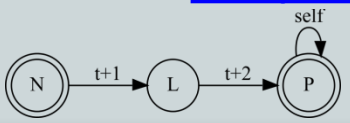
Talk

- Center for Language and Speech Processing Seminar, Friday, November 12 at 12:00pm ET

- **Andrew Piper,** McGill University

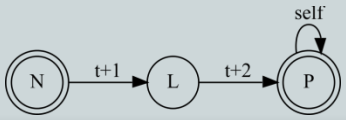
  - "How Can We Use Machine Learning to Understand Narration?"

  - **Zoom Link: https://wse.zoom.us/j/93327212503**
  - Sign In Sheet: https://docs.google.com/spreadsheets/d/1-2sEgxADpmp2jlCvlAb8m-8VqccUneHtYK1zFGUqWAs/edit?usp=sharing



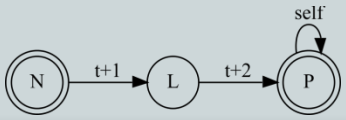
Notes about homework

- Create same dictionaries for start/trans/emit
- Create the state list from seen tags
- Read a file line by line…
- Check if lines have a tab…
  - Split by tab
  - Get POS and word columns
  - Remember tag as prev_tag for transition…
  - +=1 to relevant frequencies
- Divide all dictionary values by sum of dictionary so you get **probabilities**

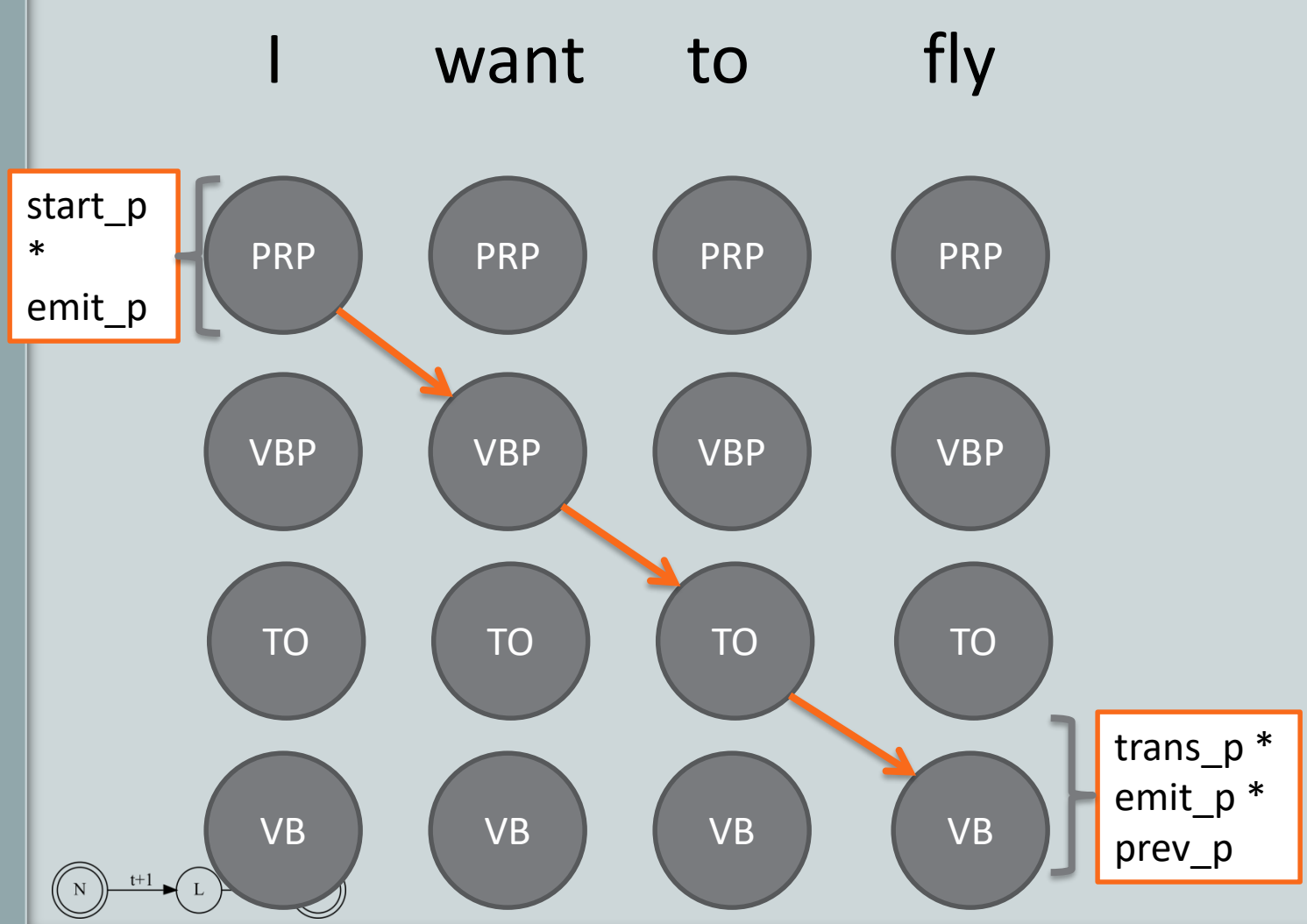


Notes about homework

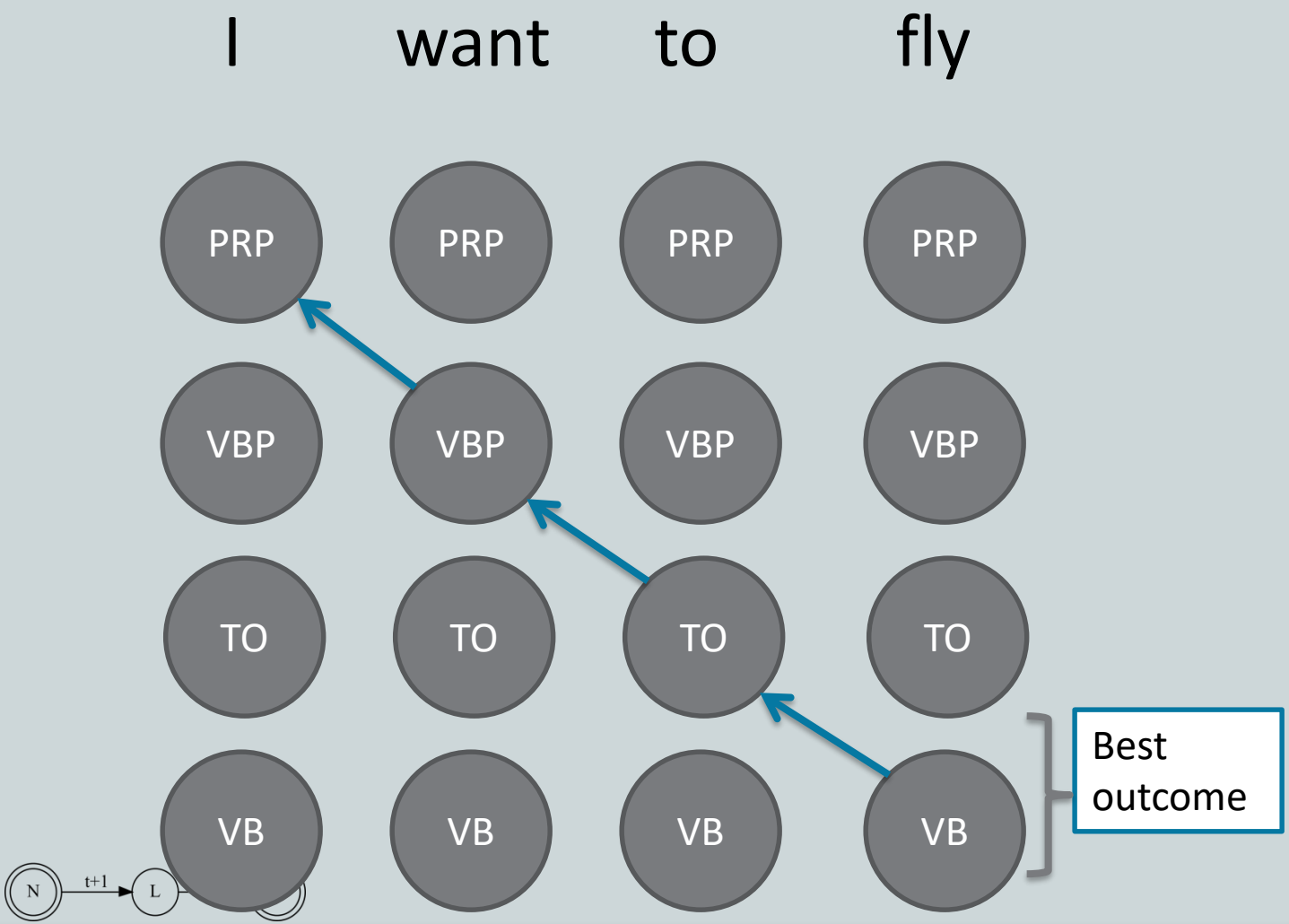
```
# Suppose each key in mydict points to a dict of counts
# Compute probabilities from sums and store in 'probs'
probs = defaultdict(lambda: defaultdict(lambda: 0.0001))

for pos1 in mydict:
    total = sum(mydict[pos1].values())  # sum values for this pos
    for pos2 in mydict[pos1]:
        freq = mydict[pos1][pos2]
        probs[pos1][pos2] = freq/total

# And similarly for emit_p!
```



Brief review: Viterbi algorithm

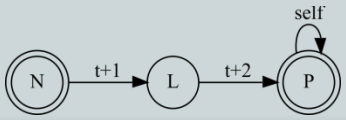


Backtrace


I
want
to
fly
PRP
PRP
PRP
PRP
VBP
VBP
VBP
VBP
TO
TO
TO
TO
VB
VB
VB
VB
Best outcome
N
t+1
L

From tagging to sequences

- Part of speech labeling is a classic example of token-wise tagging:
  - Input is a sequence of words (tokens)
  - Each word receives exactly one category
  - There are usually no other features except words to decide the correct tag
- But not all labeling tasks are like this!
- We could tag more complex sequences and with more input features!



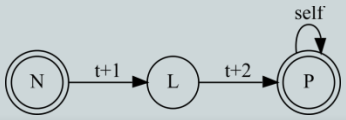
Sequence labeling – NER

- A typical example is **Named Entity Recognition**
- Not every token is labeled:

PER	**--**	***ORG***	**--**
• Kim	visited	Intel	.

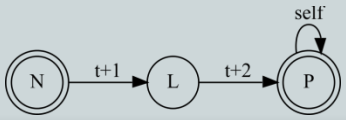
- Labels come in **spans**:

PER	***PER***	**--**	***ORG***	***ORG***
• Kim	Jung	visited	Intel	Corp.



How many spans?

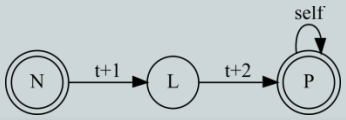
- If we only use labels like PER and ORG, we can treat this as an HMM/Viterbi problem
  - Tags: **PER, ORG, .., --** ('--' is a tag)
  - Input: **Kim, Jung, visited …**
  - Emission probabilities: P(Kim|PER), P(Intel|ORG)
  - Transition probabilities: PER → -- → ORG → ORG
- But how can we tell how many ORGs we have?
  - Intel Corp. ORG ORG → 1 org., 2 tokens
  - IBM Google lawsuit ORG ORG -- → 2 orgs!!



Solution: BIO encoding

- We add more label types to indicate **B**eginning and **I**nside of entities:
  - IBM *B-ORG*
  - Corp. *I-ORG*
  - hired *O*
  - Kim *B-PER*
  - Jung *I-PER*

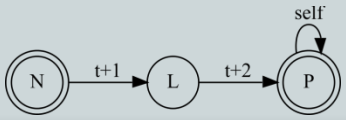
- The label **O** is like our '--': **Outside** any entity



Solution: BIO encoding

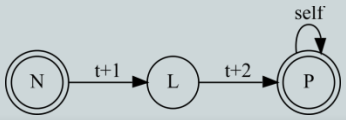
- Labels impose restrictions on transitions:
  - P(B-PER → I-PER) > p(I-PER → B-PER)
  - P(O → I-PER) = 0 (why?)

- We can still use HMM/Viterbi…
- But is just one emission probability enough?
  - P(PER|Kim) …
  - What about other features?



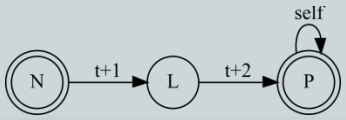
Just one emission?

- Many things influence the probability that a word is a person/company name:
  - Capitalization (very good at finding 'O')
  - All caps? (ORG)
  - Word length
  - Knowledge bases (is this in a list of company names? Place names?)
  - …
- Viterbi can only multiply feature probabilities
- Not good for learning arbitrary feature conjunctions (weighted features)



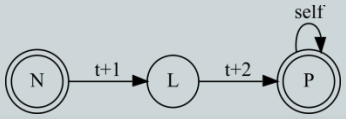
Using multiple features

- Ideally our input should look like this:
  - IBM NNP allcaps … ***B-ORG***
  - Corp. NNP title … ***I-ORG***
  - hired VBD lower … ***O***
  - Kim NNP title … ***B-PER***
  - Jung NNP title … ***I-PER***



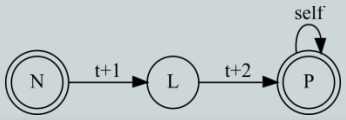
Decoding - CRF

- Efficient decoding over **multiple** weighted features can be done using **Conditional Random Fields** (CRF)
- We do not have time to implement CRF in this course
- For our purposes, a Linear Chain CRF is
  - a sequence label decoder equivalent to a Viterbi decoder
  - using **multiple input features**
  - and **arbitrary functions** for features over the sequence
- Advanced reading: Sutton & McCallum (2006) in Canvas (optional!)



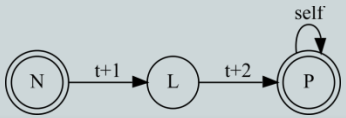
Decoding - CRF

- For smaller datasets, CRF taggers can learn joint discrete feature value distributions:
  - Python library:
    - pip install python-crfsuite (Okazaki 2007)
  - Good off the shelf CRF tagger:
    - Marmot (Müller et al. 2013; Java), http://cistern.cis.lmu.de/marmot/
  - CRF NER tagging example in Canvas:
    - ner/crf_entities.py



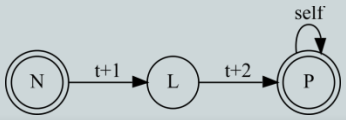
Neural sequence labeling

- Since features can be anything…
- For larger datasets, we can use neural networks
- Word embeddings as features
- What algorithm to use?
  - Definitely not Viterbi: we don't want the product of probabilities that dimension 1 is…
  - -> use CRF on neural network outputs!



Popular libraries

- Flair (Akbik et al. 2019)
- AllenNLP (Gardner et al. 2018)
- NCRF++ (Yang & Zhang 2018)

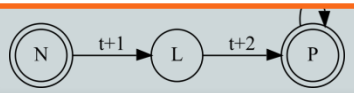


Example – Flair (Akbik et al. 2019)

```
from flair.models import SequenceTagger

# pretrained NER tagger
tagger = SequenceTagger.load('ner')

sentence = Sentence('George Washington went to Washington .')

# predict NER tags
tagger.predict(sentence)

# print sentence with predicted tags
print(sentence.to_tagged_string())
```

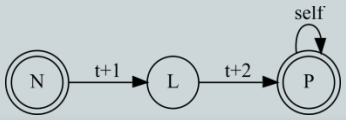
George <B-PER> Washington <E-PER> went to Washington <S-LOC> .

For training see:
https://github.com/flairNLP/flair/blob/master/resources/docs/TUTORIAL_7_TRAINING_A_MODEL.md



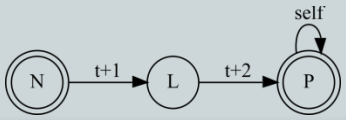
From HMMs to syntax

- We've already seen some simple ways of dealing with syntax:
  - Markov models capture surface properties of syntax
    - N-grams (VMM): ***A lot of …***
    - HMM: **DT JJ NN**



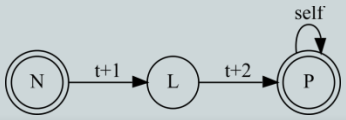
From HMMs to syntax

- We've already seen some simple ways of dealing with syntax:
  - Markov models capture surface properties of syntax
    - N-grams (VMM): ***A lot of …***
    - HMM: **DT JJ NN**
  - Finite-state methods build possible sequences
    - Coptic:
      - PREP -> ART -> NOUN
      - AUX -> SUBJ -> V -> OBJ



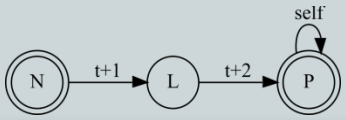
Modelling syntax

- Why isn't it enough?
  - FSAs and n-grams (weighted FSAs) have **no memory**
  - No way to manage long distance dependencies:
    - Pick up
      the one we saw yesterday…
  - Distance > n (for n-order Markov model)
  - Unlimited embedding depth can exceed properties of regular languages
  - Sparse attestation can exceed learnability with realistic (finite) unconstrained neural network



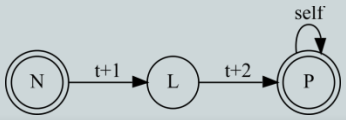
Languages and complexity

- Regular languages are the simplest grammars we can build:
  - Include all **finite** languages (where we can enumerate all expressions)
  - Potential for infinite generation (a+)
  - Optional or empty elements (ab?, ab*)
  - (Regular languages without the latter are also called 'star-free')



Beyond regular languages

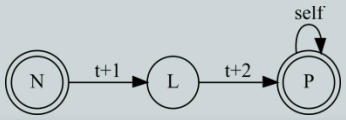
- What if we want to name a+b something else?
  - We could do things like: (DT+JJ+N)=NP: NP+…
  - This is **still** a regular language (can use FSA)
  - Even some recursion is OK:
    - x -> x
    - un + adj -> adj
- Are there constructions that can't be expressed using regular grammars?

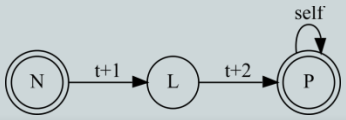


Example: center-embedding

- In English we can center-embed relative clauses:
  - *The boy laughed*
  - *The boy the cat bit laughed*

- Structure:
  - S > NP VP
  - S > NP S VP → NP NP VP VP
- We can potentially continue to center-embed…
  - Result:
    utterances of the type NP^n VP^n (or generally $a^n b^n$)





Morten Christiansen
@MH_Christiansen

Replying to @MH_Christiansen

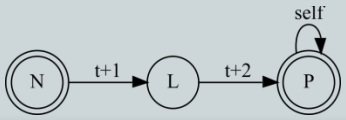
The winner meme was about center-embedded sentences. The caption was: Let's speak properly shall we.
2/6

1:32 PM · May 6, 2021 · Twitter Web App

Another example

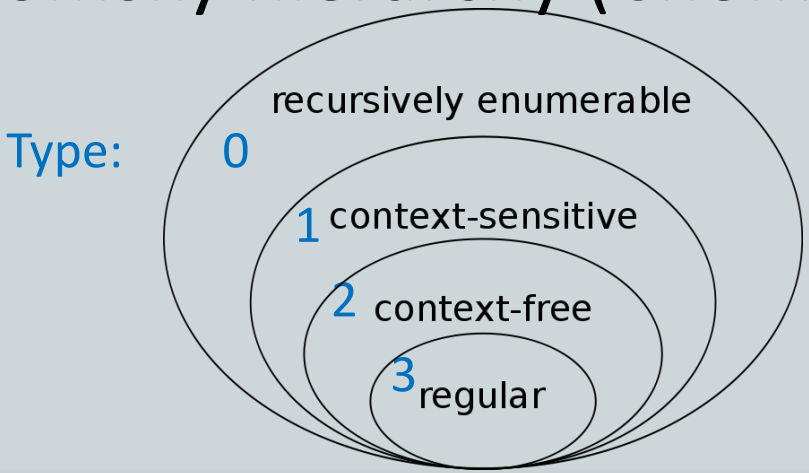
- Less famous – Semitic embedded compound modifiers:
  - [bat [melex ʻašir] yafa]
    daughter king rich.M beautiful.F
    Beautiful daughter of a rich king
  - [bat [melex [ʻam gadol] ʻašir] yafa]
    daughter king people great.M rich.M beautiful.F
    Beautiful daughter of a rich king of a great people

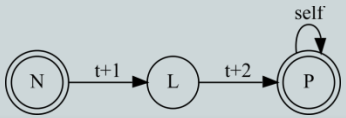
- Note that agreement information must match
- Memory: N^n A^n with matching gender/number



The Chomsky Hierarchy

- "self-embedding" categories:
  - Are a feature of **context free** languages
  - Allow us a sort of 'memory'
  - Long thought to cover human grammars
- Context free grammars (CFGs) occupy Type-2 of the Chomsky hierarchy (Chomsky 1956)



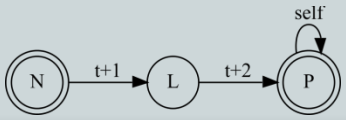


(Image: Wikimedia)

The limits of CFGs

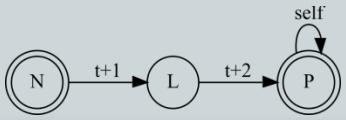
- Context free grammars allow rules of the form:
  - α > β
  - α is a non terminal symbol (hidden node: NP, VP, S …)
  - β is any sequence of terminal or non-terminal symbols (tokens or higher nodes)

- Examples:
  - S > NP VP (could still be regular)
  - S > NP S VP (context free)



Are human languages more complex?

- Some conceivable rules are not covered:
  - We cannot 'peek' to limit application of a rule:
    - S > NP S VP (OK)
    - PP S PP > PP NP S VP PP (check for surrounding PPs: not OK)
- Rules of this type are **context-sensitive**
  - αAβ > αγβ
  - We can prove that patterns of the type $\mathbf{a^n b^n c^n}$ are **context-sensitive**
  - So are patterns like $(abc \ldots)^n$



Are human languages more complex?

- There are few examples of context sensitive structures in natural language
- Famous example: Swiss German crossing dependencies (Shieber 1985)

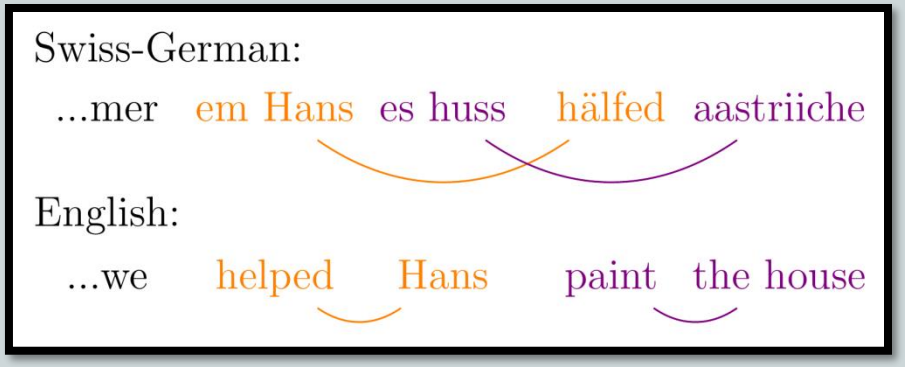
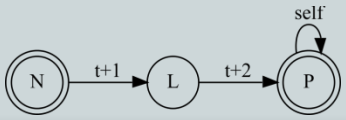


Image: wikimedia



Are human languages more complex?

- There are few examples of context sensitive structures in natural language
- Famous example: Swiss German crossing dependencies (Shieber 1985)

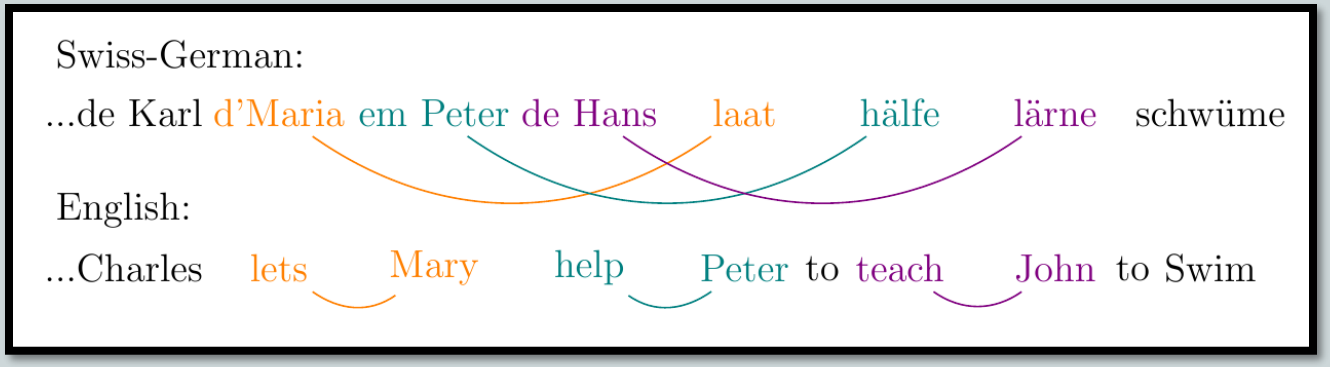
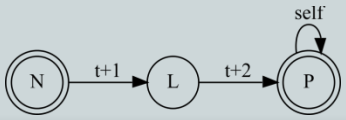
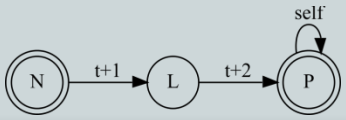


Image: wikimedia



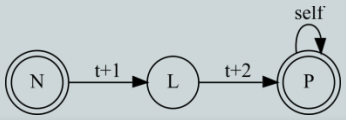
Context Free Grammars

- CFGs are nevertheless enough for most structures and much more efficient to compute
  - A context free grammar is a set of (de)composition rules over a set of symbols:
    - NP > DT NN
    - NP > NNP
    - DT > the
    - NN > house
    - NN > mouse
    - …
  - Symbols which do not decompose are called **terminals** (often =tokens)



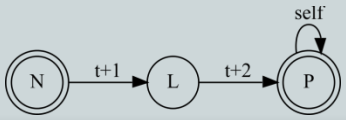
Context Free Grammars

- The set of decomposition combinations generates all utterances in the language **L** modelled by the grammar
- A **starting symbol** must be selected to generate from; usually S



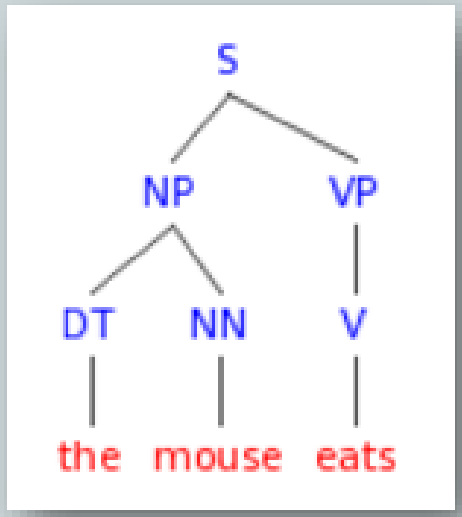
Context Free Grammars

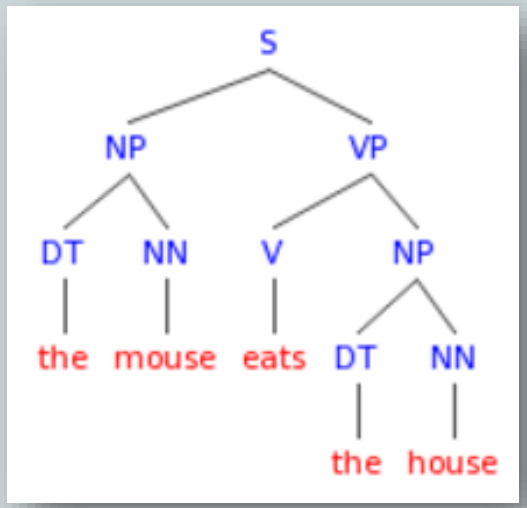
- Some example rules:
  - S > NP VP
  - VP > V NP
  - VP > V
  - V > eats
  - NP > DT NN
  - NN > mouse
  - NN > house
  - DT > the
  - …

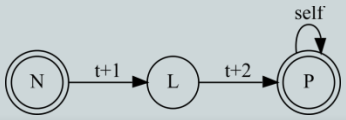


Now we can generate…

- (never minding meaning – à la 'colorless green ideas…')

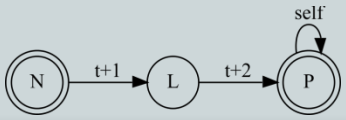






Exercise

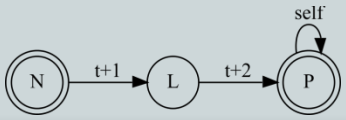
- Let's try to extract **context free rules** from sentences:
  - Every sentence has **S** at the top
  - Breaks down into phrases
  - Phrases decompose into our POS tags/other phrases
  - POS tags lead to tokens



Exercise

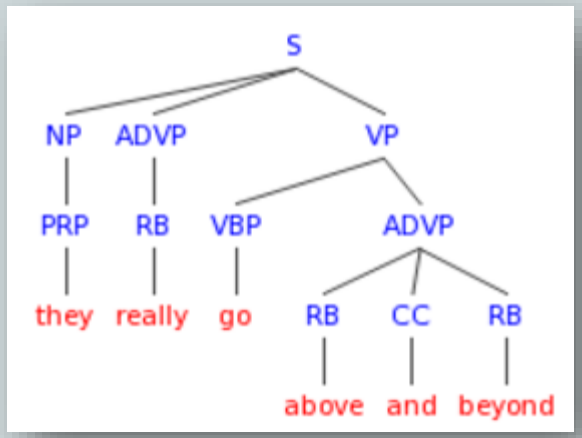
- Example:
  - They really go above and beyond!
- Tag it first:
  - PRP RB VBP RB CC RB .

  So we have:
  - RB > really
  - VBP > go
  …
- What are the phrase structure rules?

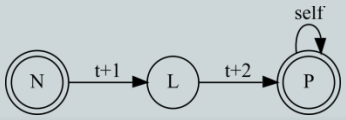


Exercise

- A possible analysis (English Web Treebank; other analyses are possible!)



- How can we write the rules?



Exercise

- Break down the transitions:
  - S > NP ADVP VP
  - NP > PRP
  - ADVP > RB
  - VP > VBP ADVP
  - ADVP > RB CC RB