Introduction to Natural Language Processing II [Statistické metody zpracování přirozených jazyků] (NPFL068)

prof. RNDr. Jan Hajič, Dr.
Mgr. Jindřich Helcl, Ph.D.
ÚFAL MFF UK
{hajic,helcl}@ufal.mff.cuni.cz

http://ufal.mff.cuni.cz or SIS

Mon, S9, 09:00-10:30

Tagging, Tagsets, and Morphology

→ the context is having the major voice in the decision of the final tag

The task of (Morphological) Tagging

→ we usually say there can be just one tag for a word, given all required context

- Formally: $A^+ \rightarrow T$
  - A is the alphabet of phonemes (A^+ denotes any non-empty sequence of phonemes)
    - often: phonemes ~ letters
  - T is the set of tags (the "tagset")
- Recall: 6 levels of language description:
  - phonetics ... phonology ... morphology ... syntax ... meaning ...
- Recall: $A^+ \rightarrow 2^{(L,C_1,C_2,...,Cn)} \rightarrow T$

- a step aside: tagging

morphology

tagging: ***disambiguation*** (~ "select")

Tagging Examples

- Word form: $A^+ \rightarrow 2^{(L,C_1,C_2,...,Cn)} \rightarrow T$
  - He always books the violin concert tickets early.
    - MA: books → {(book-1,Noun,Pl,-,-),(book-2,Verb,Sg,Pres,3)}
    - tagging (disambiguation): ... → (Verb,Sg,Pres,3)
  - ...was pretty good. However, she did not realize...
    - MA: However → {(however-1,Conj/coord,-,-,-),(however-2,Adv,-,-,-)}
    - tagging: ... → (Conj/coord,-,-,-)
  - [æ n d] [g i v] [i t] [t u:] [j u:] ("and give it to you")
    - MA: [t u:] → {(to-1,Prep),(two,Num),(to-2,Part/inf),(too,Adv)}
    - tagging: ... → (Prep) —> it helps me to determine the correct form

Tagsets

- General definition:
  - tag ~ $(c_1,c_2,...,c_n)$ → tag can be a set of categories
  - often thought of as a flat list

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

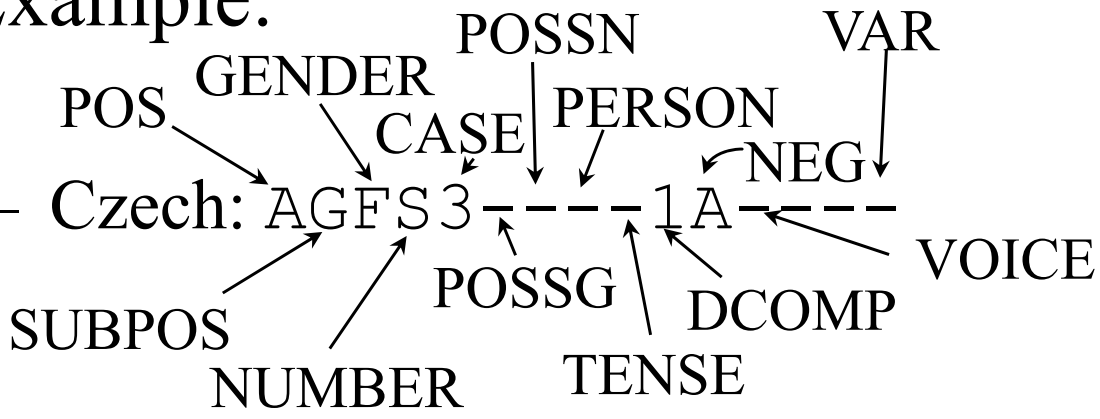
    with some assumed 1:1 mapping

    $T \leftrightarrow (C_1,C_2,...,C_n)$
- English tagsets (see MS): → has morphological features as well
  - Penn treebank (45) (VBZ: Verb,Pres,3,sg, JJR: Adj. Comp.)
  - Brown Corpus (87), Claws c5 (62), London-Lund (197)

Other Language Tagsets

one Czech corpus has ~6000 distinct tag

- Differences:
  - size (10..10k)
  - categories covered (POS, Number, Case, Negation,...)
  - level of detail
  - presentation (short names vs. structured ("positional"))
- Example:
  - Czech: AGFS3----1A----



→ each letter is one tagging category

Morphology only gives all possible choices.
Tagging decides which is correct.

Tagging Inside Morphology

- Do tagging first, then morphology:
- Formally: $A^+ \rightarrow T \rightarrow (L,C_1,C_2,...,C_n)$
- Rationale:
  - have $|T| < |(L,C_1,C_2,...,C_n)|$ (thus, less work for the tagger) and keep the mapping $A^+ \times T \rightarrow (L,C_1,C_2,...,C_n)$ unique.
- Possible for some languages only ("English-like")
- Same effect within "regular" $A^+ \rightarrow 2^{(L,C1,C2,...,Cn)} \rightarrow T$:
  - mapping $R : (C_1,C_2,...,C_n) \rightarrow T_{reduced}$,
    then (new) unique mapping U: $A^+ \times T_{reduced} \rightarrow (L,T)$

books -> book, are -> be, were -> be

-> lemmatization

Lemmatization

helps to understand the text or correctly modify the text or helps in search

- Full morphological analysis:

  MA: $A^+ \rightarrow 2^{(L,C1,C2,...,Cn)}$

  (recall: a lemma $l \in L$ is a lexical unit (~ dictionary entry ref)

you don't even know all used forms in the database...

- Lemmatization: reduced MA:
  - $L: A^+ \rightarrow 2^L: w \rightarrow \{l; (l,t_1,t_2,...,t_n) \in MA(w)\}$
  - again, need to disambiguate (want: $A^+ \rightarrow L$)

    (special case of word sense disambiguation, WSD)
  - "classic" tagging does not deal with lemmatization

    (assumes lemmatization done afterwards somehow)

Using class modelling is something like tagging, as the alg. is hiddenly creating „tag groups", but the alg. is not able to name those groups.

Morphological Analysis: Methods

CZ: 25M forms, EN: 1M, FINN: 100M, GER: 8M forms ! Annotators must do it by hand.

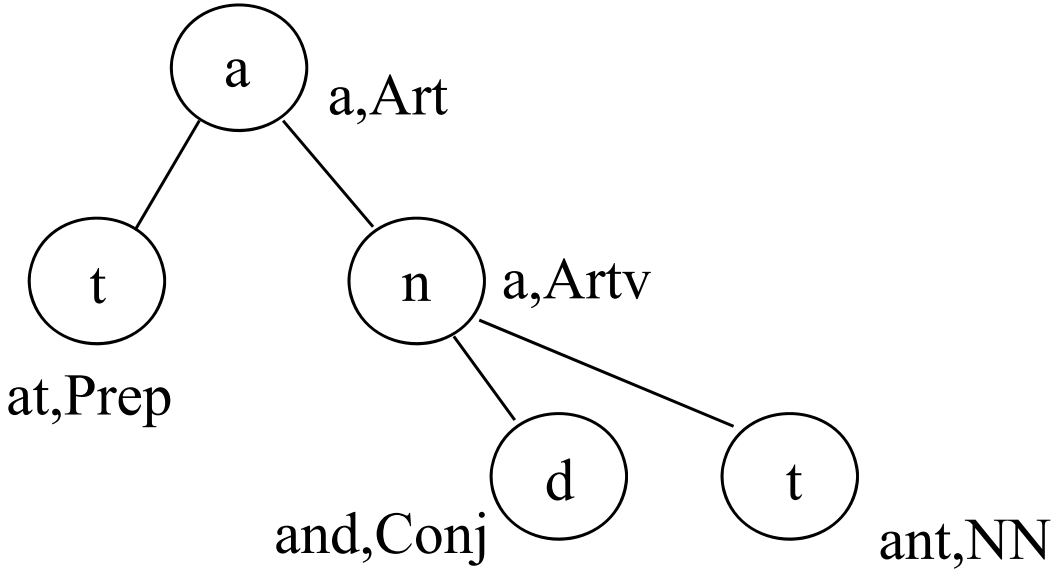
- Word form list
  - books: book-2/VBZ, book-1/NNS

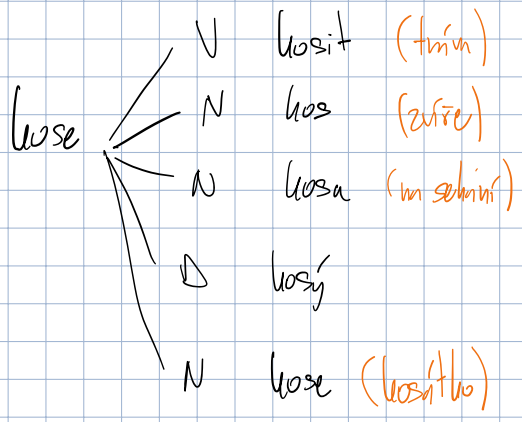
→ computationally heavy → too much space needed

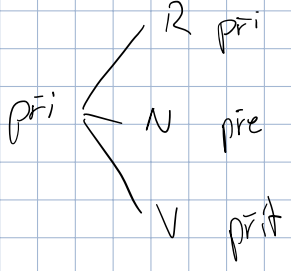
- Direct coding
  - endings: verbreg:s/VBZ, nounreg:s/NNS, adje:er/JJR, ...
  - (main) dictionary: book/verbreg, book/nounreg,nic/adje:nice
- Finite state machinery (FSM)
  - many "lexicons", with continuation links: reg-root-lex → reg-end-lex
  - phonology included but (often) clearly separated
- CFG, DATR, Unification, ...
  - address linguistic rather than computational phenomena
  - in fact, better suited for morphological synthesis (generation)

Word Lists

- Works for English
  - "input" problem: repetitive hand coding
- Implementation issues:
  - search trees
  - hash tables (Perl!)
  - (letter) trie:
- Minimization?







Sometimes even lemmas are not unique and we need to add indexes.

stát - 1 na zemi
stát - 2 pekař
stát - 3 stálo se
stát - 4 ČR

→ tady to prostě musím nadělit

Word-internal[1] Segmentation (Direct)

- Strip prefixes: (un-, dis-, ...)
- Repeat for all plausible endings:
  - Split rest: root + ending (for every possible ending)
  - Find root in a dictionary, keep dictionary information
    - in particular, keep inflection class (such as reg, noun-irreg-e, ...)
  - Find ending, check inflection+prefix class match
  - If match found:
    - Output root-related info (typically, the lemma(s))
    - Output ending-related information (typically, the tag(s)).

[1]Word segmentation is a different problem (Japanese, speech in general)

fonologie: např. že mám „tomuto" -> „tomutoes" tedy že něco obecně přidávám/ubírám

Finite State Machinery

- Two-level Morphology
  - phonology + “morphotactics” (= morphology)
- Both components use finite-state automata:
  - phonology: “two-level rules”, converted to FSA
    - e:0 ⇔ _ +:0 e:e r:r
  - morphology: linked lexicons
    - root-dic: book/”book” ⇒ end-noun-reg-dic
    - end-noun-reg-dic: +s/”NNS”
- Integration of the two possible (and simple)

– dobrý to bylo pro Finštinu, jelikož vytváří význam slov skládáním mnoha suffixů, které ještě mají různé tvary podle kontextu.

Finite State Transducer

– umíš doplnit externí constraints (morpho. dict), a tyto nemusíš neustále hlídat.

FSA can be minimized a lot and then the trie is very efficient

- FST is a FSA where – but they are exponential for $x \in L$
  - symbols are pairs (r:s) from a finite alphabets R and S.
- "Checking" run:
  - input data: sequence of pairs, output: Yes/No (accept/do not)
  - use as a FSA
- Analysis run:
  - input data: sequence of only $s \in S$ (TLM: surface);
  - output: seq. of $r \in R$ (TLM: lexical), + lexicon "glosses"
- Synthesis (generation) run:
  - same as analysis except roles are switched: $S \leftrightarrow R$, no gloss

this is one symbol for FSA

→ it says { yes → then complete the second half of the sequence. no → over

→ when we want just one output

FST Example

- German umlaut (greatly simplified!):

  u ↔ ü if (but not only if) c h e r follows (Buch → Bücher)

  rule: u:ü ⇐ _ c:c h:h e:e r:r

  FST:

  Buch/Buch:

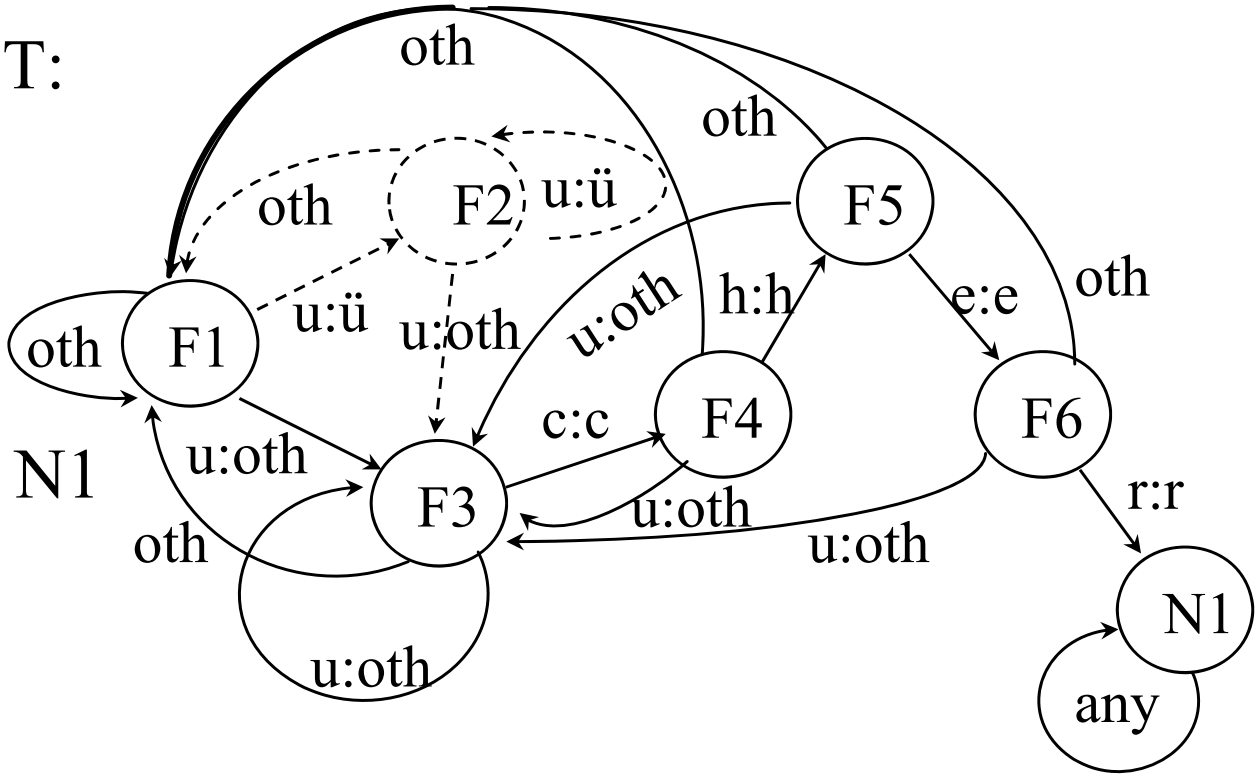
  F1 F3 F4 F5

  Bucher/Bucher:

  F1 F3 F4 F5 F6 N1

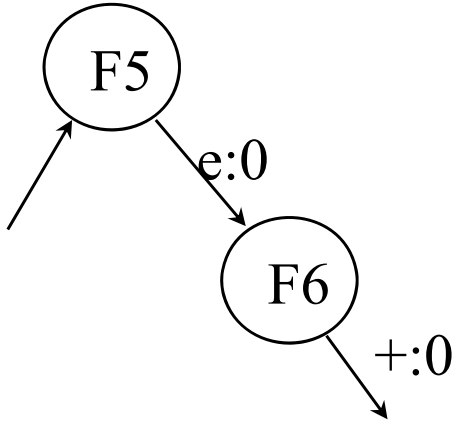
  Buch/Buck:

  F1 F3 F4 F1



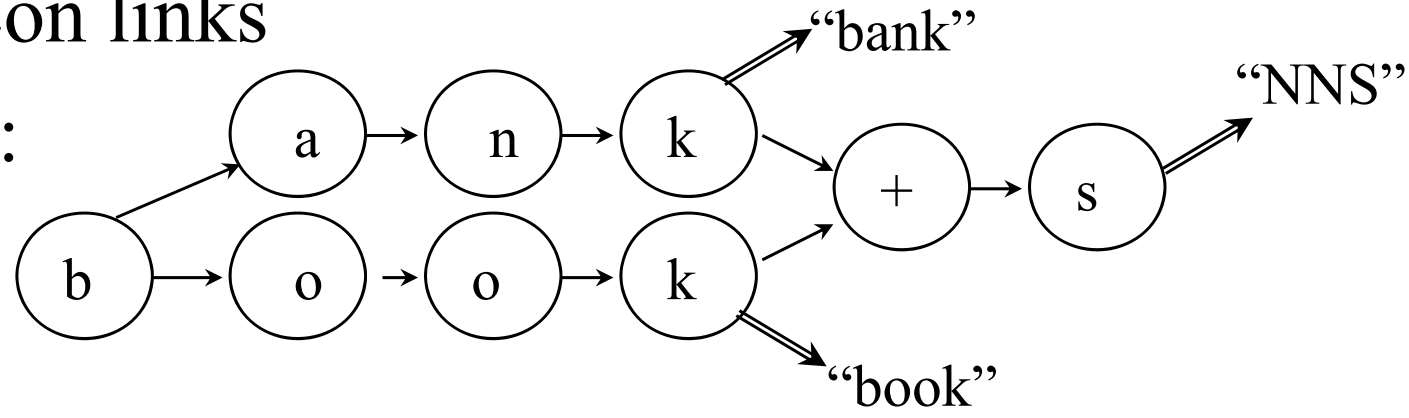
Parallel Rules, Zero Symbols

- Parallel Rules:
  - Each rule ~ one FST
  - Run in parallel
  - Any of them fails ⇒ path fails
- Zero symbols (one side only, even though 0:0 o.k.)
  - behave like any other



The Lexicon

- Ordinary FSA ("lexical" alphabet only)
- Used for analysis only (NB: disadvantage of TLM):
  - additional constraint:
    - lexical string must pass the linked lexicon list
- Implemented as a FSA; compiled from lists of strings and lexicon links
- Example:



TLM: Analysis

1. Initialize set of paths to P = {}.
2. Read input symbols, one at a time.
3. At each symbol, generate all lexical symbols possibly corresponding to the 0 symbol (voilà!).
4. Prolong all paths in P by all such possible (x:0) pairs.
5. Check each new path extension against the phonological FST and lexical FSA (lexical symbols only); delete impossible paths prefixes.
6. Repeat 4-5 until max. # of consecutive 0 reached.

TLM: Analysis (Cont.)

7. Generate all possible lexical symbols (get from all FSTs) for the current input symbol, form pairs.
8. Extend all paths from P using all such pairs.
9. Check all paths from P (next step in FST/FSA). Delete all outright impossible paths.
10. Repeat from 3 until end of input.
11. Collect lexical “glosses” from all surviving paths.

TLM Analysis Example

- Bücher:
  - suppose each surface letter corresponds to the same symbol at the lexical level, just ü might be ü as well as u lexically; plus zeroes (+:0), (0:0)
- Use the FST as before.
- Use lexicons:

  root: Buch “book” ⇒ end-reg-uml

  Bündni “union” ⇒ end-reg-s

  end-reg-uml: +0 “NNomSg”

  +er “NNomPl”

B:B $\overset{u}{\Rightarrow}$ Bu:Bü ⇒ Buc:Büc ⇒ Buch:Büch ⇒ Buch+e:Büch0e ⇒ Buch+er:Büch0er

$\overset{ü}{\Rightarrow}$ ~~Bü:Bü ⇒ Büc:Büc~~

TLM: Generation

- Do not use the lexicon (well you have to put the "right" lexical strings together somehow!)
- Start with a lexical string L.
- Generate all possible pairs l:s for every symbol in L.
- Find all (hopefully only 1!) traversals through the FST which end in a final state.
- From all such traversals, print out the sequence of surface letters.

TLM: Some Remarks

- Parallel FST (incl. final lexicon FSA)
  - can be compiled into a (gigantic) FST
  - maybe not so gigantic (XLT - Xerox Language Tools)
- “Double-leveling” the lexicon:
  - allows for generation from lemma, tag
  - needs: rules with strings of unequal length
- Rule Compiler
  - Karttunen, Kay
- PC-KIMMO: free version from www.sil.org (Unix,too)

Tagging: An Overview

~ in C2E 1/2 words are ambiguous

Rule-based Disambiguation

- Example after-morphology data (using Penn tagset):

I	watch	a	fly	.
NN	NN	DT	NN	.
PRP	VB	NN	VB	
	VBP		VBP	

- Rules using
  - word forms, from context & current position
  - tags, from context and current position
  - tag sets, from context and current position
  - combinations thereof

Example Rules

I	watch	a	fly
NN	NN	DT	NN
PRP	VB	NN	VB
	VBP		VBP

- If-then style:
  - $DT_{eq,-1,Tag} \Rightarrow NN$

    (implies $NN_{in,0,Set}$ as a condition)
  - $PRP_{eq,-1,Tag}$ *and* $DT_{eq,+1,Tag} \Rightarrow VBP$
  - $\{DT,NN\}_{sub,0,Set} \Rightarrow DT$
  - $\{VB,VBZ,VBP,VBD,VBG\}_{inc,+1,Tag} \Rightarrow$ *not* DT
- Regular expressions:
  - *not*(<*,*,DT> <*,*,*not*NN>))
  - *not*(<*,*,PRP>,<*,*,*not*VBP>,<*,*,DT>)
  - *not*(<*,{DT,NN}$_{sub,}$*not*DT>)
  - *not*(<*,*,DT>,<*,*,{VB,VBZ,VBP,VBD,VBG}>)

the biggest problem is not time, nor space, but that the rules are created by hand...

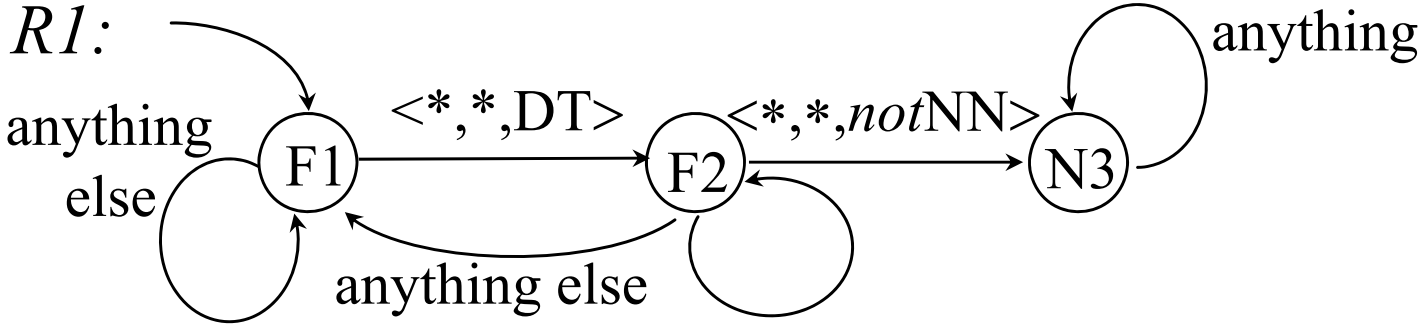
Implementation

- Finite State Automata
  - parallel (each rule ~ automaton);
    - algorithm: keep all paths which cause all automata say *yes*
  - compile into single FSA (intersection)
- Algorithm:
  - a version of Viterbi search, but:
    - no probabilities ("categorical" rules)
    - multiple input:
      - keep track of all possible paths

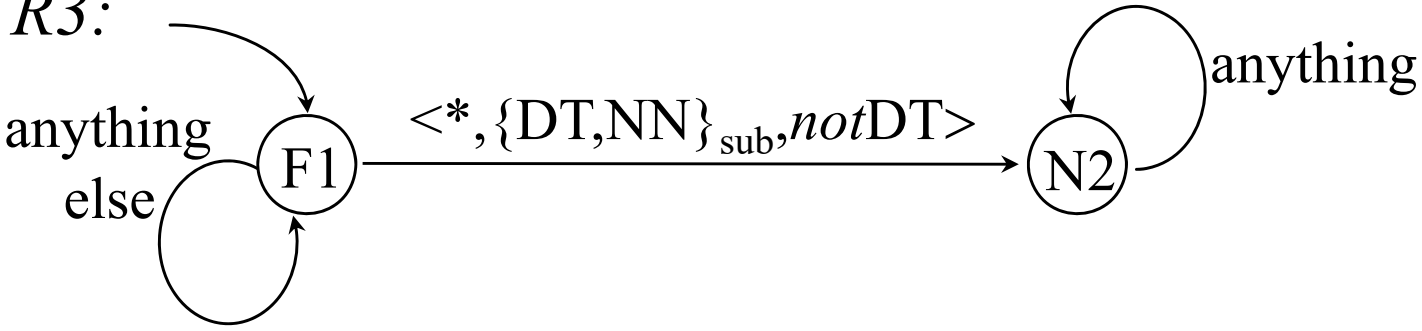
Example: the FSA

- *R1: not*(<*,*,DT> <*,*,*not*NN>))
- *R2: not*(<*,*,PRP>,<*,*,*not*VBP>,<*,*,DT>)
- *R3: not*(<*,{DT,NN}$_{sub,}$DT>)
- *R4: not*(<*,*,DT>,<*,*,{VB,VBZ,VBP,VBD,VBG}>)

- *R1:*



- *R3:*



Applying the FSA

I	watch	a	fly
NN	NN	DT	NN
PRP	VB	NN	VB
	VBP		VBP

- *R1: not*(<*,*,DT> <*,*,*not*NN>))
- *R2: not*(<*,*,PRP>,<*,*,*not*VBP>,<*,*,DT>)
- *R3: not*(<*,{DT,NN}$_{sub}$,DT>)
- *R4: not*(<*,*,DT>,<*,*,{VB,VBZ,VBP,VBD,VBG}>)

- R1 blocks:

a	fly
DT	
	VB
	VBP

remains:

a	fly
DT	NN

or

a	fly
	NN
NN	VB
	VBP

- R2 blocks:

I	watch	a
	NN	DT
PRP	VB	

remains e.g.:

I	watch	a
		DT
PRP		
	VBP	

and more

- R3 blocks:

a
NN

remains only:

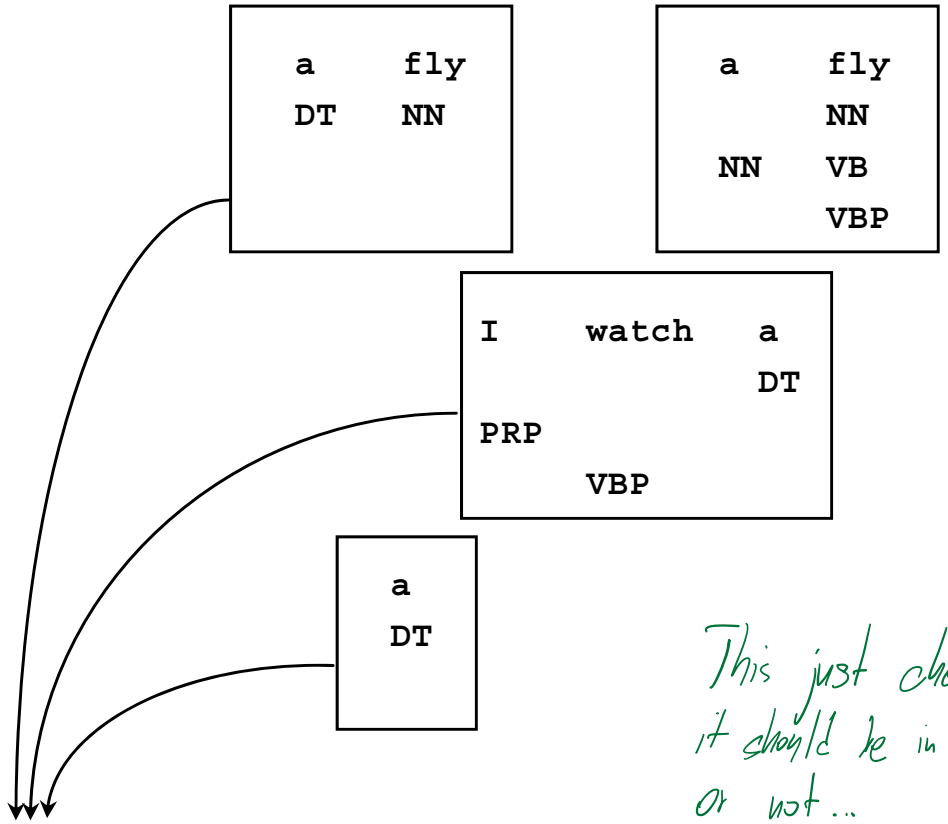
a
DT

- R4 $\subset$ R1!

Applying the FSA (Cont.)

I	watch	a	fly
NN	NN	DT	NN
PRP	VB	NN	VB
	VBP		VBP

- Combine:



Why is it usefull in age of LLMs?
- for checking annotating errors with already very simple rules
- deep-learning is not always having 100% accuracy, so sometimes you can find errors easily

This just checks whether it should be in the language or not...

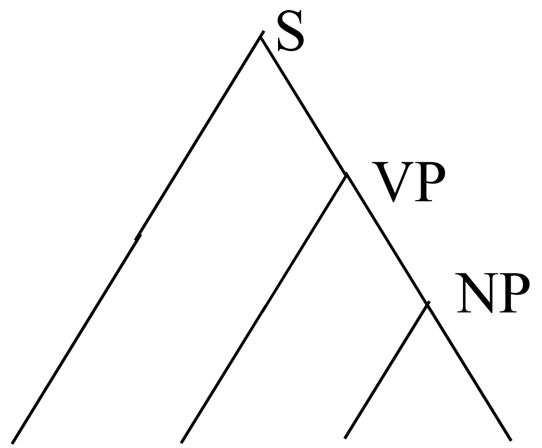
- Result:

I	watch	a	fly	.
PRP	VBP	DT	NN	.

Lineární procházení ucestnění, protože lidi taky parsují jazyk sofistikovaněji.

Tagging by Parsing

- Build a parse tree from the multiple input:



Lingvisticky daleko víc přívětivé, informaticky daleko těžší.

I	watch	a	fly
NN	NN	DT	NN
PRP	VB	NN	VB
	VBP	VBP	

- Track down rules: e.g., NP → DT NN: extract (a/DT fly/NN)
- More difficult than tagging itself; results mixed

Statistical Methods (Overview)

- "Probabilistic":
  - HMM
    - Merialdo and many more (XLT)
  - Maximum Entropy
    - DellaPietra et al., Ratnaparkhi, and others
- Rule-based:
  - TBEDL (Transformation Based, Error Driven Learning)
    - Brill's tagger
  - Example-based
    - Daelemans, Zavrel, others → "best neighbour"
- Feature-based (inflective languages)
- Classifier Combination (Brill's ideas)

in exam as well

HMM Tagging

cf. NPFL067 slides 170-190

HMM in general: NPFL067 slides 155-169

Transformation-Based Tagging

The Task, Again

- Recall:
  - tagging ~ morphological disambiguation
  - tagset $V_T \subset (C_1,C_2,...C_n)$
    - C_i - morphological categories, such as POS, NUMBER, CASE, PERSON, TENSE, GENDER, ...
  - mapping $w \rightarrow \{t \in V_T\}$ exists
    - restriction of Morphological Analysis: $A^+ \rightarrow 2^{(L,C1,C2,...,Cn)}$

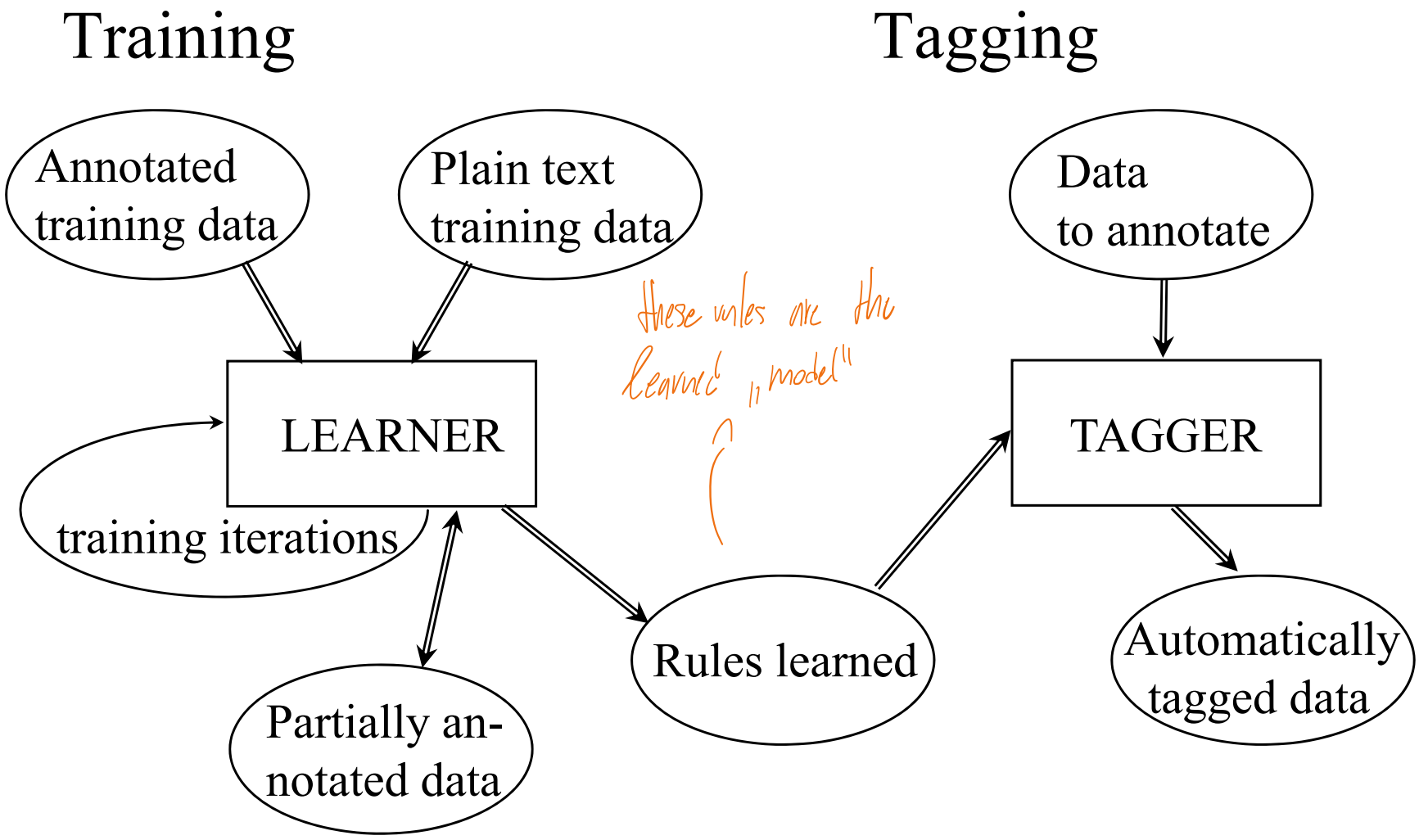
      where A is the language alphabet, L is the set of lemmas
  - extension to punctuation, sentence boundaries (treated as words)

Setting

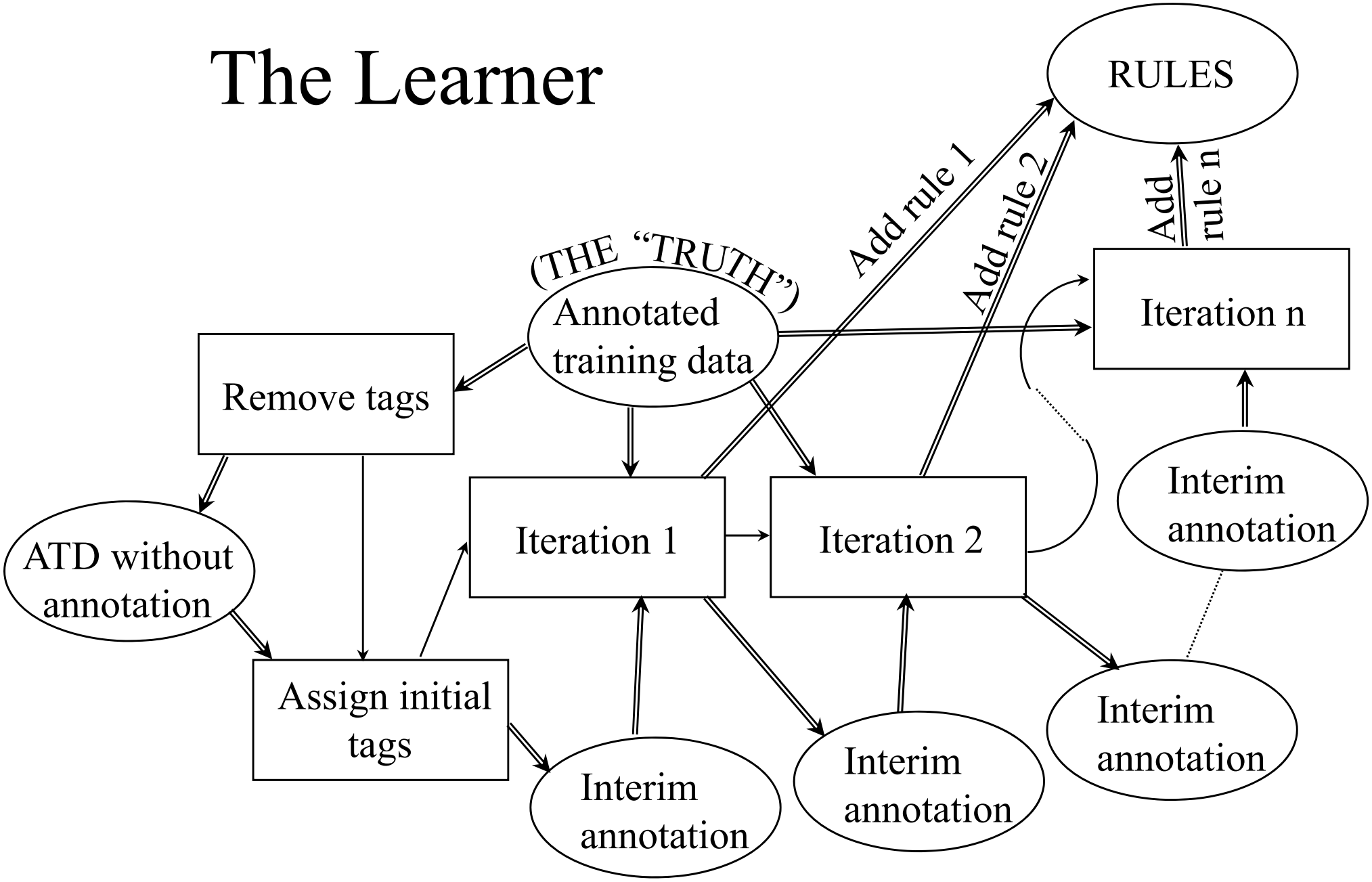
- ***Not*** a source channel view
- ***Not*** even a probabilistic model (no "numbers" used when tagging a text after a model is developed)
- Statistical, yes:
  - uses training data (combination of supervised [manually annotated data available] and unsupervised [plain text, large volume] training)
  - learning [rules]
  - criterion: accuracy (that's what we are interested in in the end, after all!)

- Sequentially selecting new and new rules until the text is correct.
  - always trying to take the best rule.

The General Scheme



The Learner


RULES
Add rule 1
Add rule 2
Add rule n
(THE "TRUTH")
Annotated training data
Iteration n
Remove tags
Interim annotation
ATD without annotation
Iteration 1
Iteration 2
Assign initial tags
Interim annotation
Interim annotation
Interim annotation

For high-freq texts, 90% tags are automatically correct when only assigning the most common tag for the word in the text

The I/O of an Iteration

- In (iteration i):
  - Intermediate data (initial or the result of previous iteration)
  - The TRUTH (the annotated training data)
  - [pool of possible rules]
- Out:
  - One rule $r_{selected(i)}$ to enhance the set of rules learned so far
  - Intermediate data (input data transformed by the rule learned in this iteration, $r_{selected(i)}$)

The Initial Assignment of Tags

- One possibility:
  - NN —> stupid, yet simple and working as beg. of training
- Another:
  - the most frequent tag for a given word form
- Even:
  - use an HMM tagger for the initial assignment
- Not particularly sensitive

The Criterion

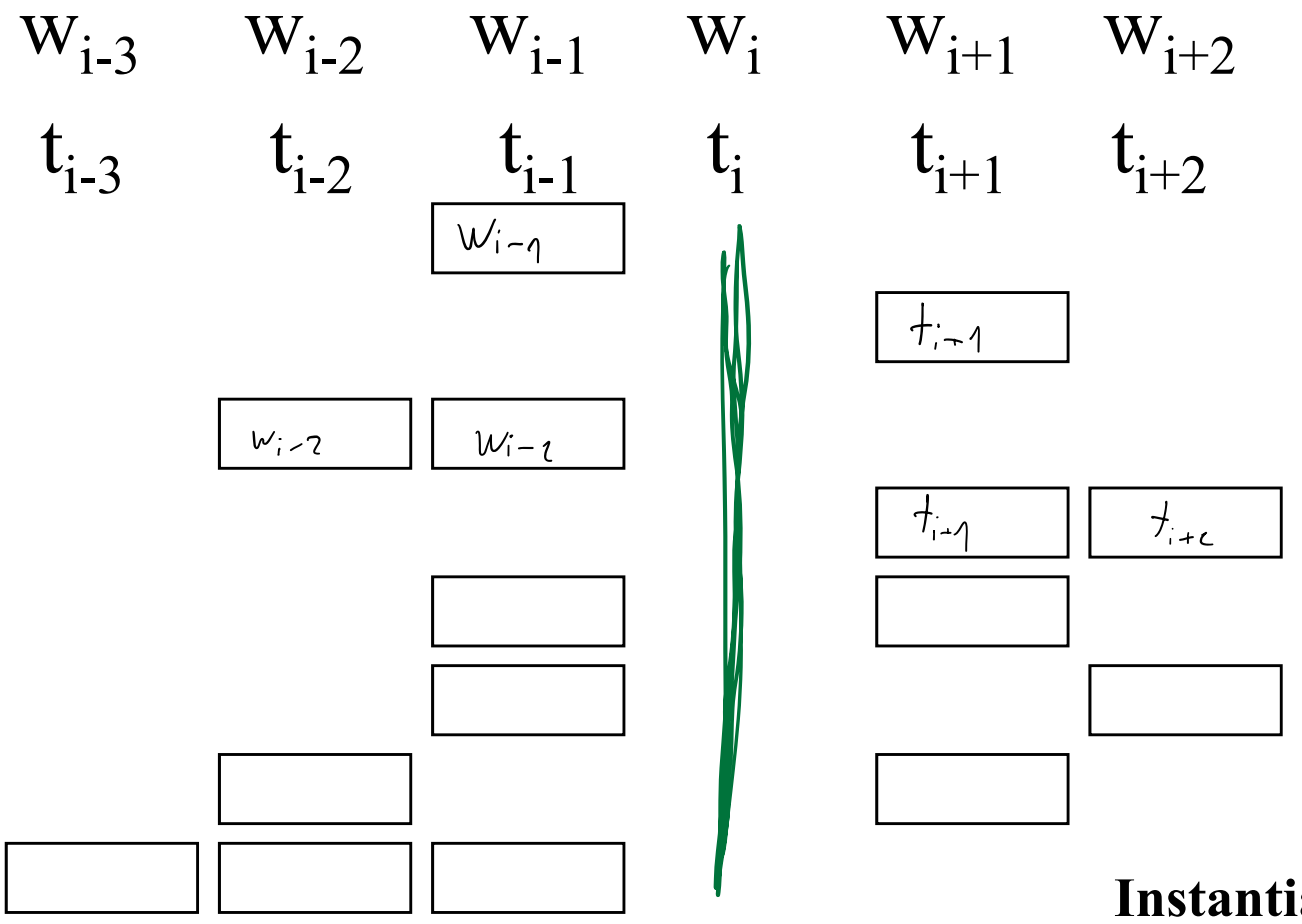
- Error rate (or Accuracy):
  - beginning of an iteration: some error rate E_{in}
  - each possible rule r, when applied at every data position:
    - makes an improvement somewhere in the data ($c_{improved}(r)$)
    - makes it worse at some places ($c_{worsened}(r)$)
    - and, of course, does not touch the remaining data
- Rule contribution to the improvement of the error rate:
  - $contrib(r) = c_{improved}(r) - c_{worsened}(r)$
- Rule selection at iteration i:
  - $r_{selected(i)} = argmax_r\ contrib(r)$
- New error rate: $E_{out} = E_{in} - contrib(r_{selected(i)})$

The Stopping Criterion

- Obvious:
  - no improvement can be made
    - contrib(r) ≤ 0
  - or improvement too small
    - contrib(r) ≤ Threshold
- NB: prone to overtraining!
  - therefore, setting a reasonable threshold advisable
- Heldout?
  - maybe: remove rules which degrade performance on H

The Pool of Rules (Templates)

- Format: *change tag at position i from a to b / condition*
- Context rules (condition definition - "template"):



We can look in the future as well.
– that is different from HMM

Instantiation: w, t permitted

Lexical Rules

- Other type: lexical rules

w_{i-3}	w_{i-2}	w_{i-1}	w_i	w_{i+1}	w_{i+2}	w_{i+3}
t_{i-3}	t_{i-2}	t_{i-1}	t_i	t_{i+1}	t_{i+2}	t_{i+3}

"look inside the word"

- Example:
  - w_i has suffix -ied
  - w_i has prefix ge-

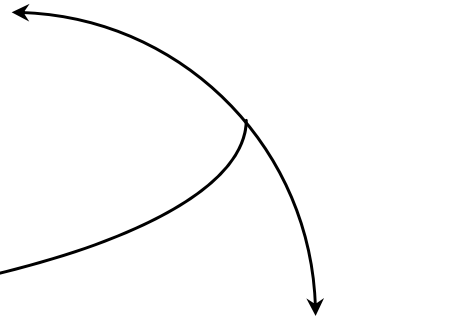
having complete word in the rule is not really convenient -> too many words, not general at all

Rule Application

- Two possibilities:
  - immediate consequences (left-to-right):
    - data: DT NN VBP NN VBP NN...
    - rule: NN → NNS / preceded by NN VBP
    - apply rule at position 4:

      DT NN VBP NN VBP NN...

      DT NN VBP NNS VBP NN...
    - ...then rule cannot apply at position 6 (context not NN VBP).
  - delayed ("fixed input"):
    - use original input for context
    - the above rule then applies twice.



the order of the final rules is important!

In Other Words...

- 1. Strip the tags off the truth, keep the original truth
- 2. Initialize the stripped data by some simple method
- 3. Start with an empty set of selected rules S.
- 4. Repeat until the stopping criterion applies:
  - compute the contribution of the rule r, for each r:

    $\text{contrib}(r) = c_{\text{improved}}(r) - c_{\text{worsened}}(r)$
  - select r which has the biggest contribution contrib(r), add it to the final set of selected rules S.
- 5. Output the set S.

Můžeme mít víc pravidel se stejným IF-partem, s jiným ELSE partem. Jenže ten algo. zpravidla nebude mít víc „duplicitních" pravidel, protože takové druhé pravidlo bude mít hrozně malou contribution, takže se asi nevybere. -> Vždy zpravidla chytí jiného pravidla a pak tam bude i s velkou contribution.

The Tagger

Musíme použít stejný postup výběru jako při trénování

- Input:
  - untagged data
  - rules (S) learned by the learner
- Tagging:
  - use the same initialization as the learner did
  - for i = 1..n (n - the number of rules learnt)
    - apply the rule i to the whole intermediate data, changing (some) tags
  - the last intermediate data is the output.

N-best & Unsupervised Modifications

- N-best modification
  - allow adding tags by rules
  - criterion: optimal combination of accuracy and the number of tags per word (we want: close to ↓1)
- Unsupervised modification
  - use only unambiguous words for evaluation criterion
  - work extremely well for English
  - does not work for languages with few unambiguous words

Maximum Entropy

Proč max. místo min?

- Uvažujeme mít více modelů, které splňují constraints, tak (filosoficky) ten model s max. entropy má největší šanci nebýt biased s nějakou konkrétní featurou, která může být významná pouze při trénování.

- „do max. entropy mi pak nejméně dávám dvéře" u příkladů, které jsem ještě neviděl.

uniform. ——> má zde největší entropy

1	2	3	4	5	6
0.1	0.1	0.1	0.4	0.1	0.2

1	2	3	4	5	6
0.25	0.05	0.05	0.4	0.05	0.2

znám

doplním

Maximum?? Entropy

- Why ***maximum*** entropy?
- Recall: so far, we always "liked"
  - minimum entropy...
    = minimum uncertainty
    = maximum predictive power
      .... distributions
  - always: relative to some "real world" data
  - always: clear relation between the data, model and parameters: e.g., n-gram language model
- This is still the case! But...

The Maximum Entropy Principle

- Given some set of constraints ("relations", "facts"), which ***must*** hold (i.e., we believe they correspond to the real world we model):

  What is the best distribution among those available?

- Answer: the one with ***maximum entropy***

  (of such distributions satisfying the constraints)

- Why? ...philosophical answer:
  - Occam's razor; Jaynes, ...:
    - make things as simple as possible, but not simpler;
    - do not pretend you know something you don't

Example

- Throwing the "unknown" die
  - do not know anything → we should assume a fair die
    (uniform distribution ~ max. entropy distribution)
- Throwing unfair die
  - we know: p(4) = 0.4, p(6) = 0.2, nothing else
  - best distribution?
  - do not assume anything about the rest:

1	2	3	4	5	6
0.1	0.1	0.1	0.4	0.1	0.2

- What if we use instead:

1	2	3	4	5	6
0.25	0.05	0.05	0.4	0.05	0.2

?

Using Non-Maximum Entropy Distribution

- ME distribution: p:

1	2	3	4	5	6
0.1	0.1	0.1	0.4	0.1	0.2

- Using instead: q:

1	2	3	4	5	6
0.25	0.05	0.05	0.4	0.05	0.2

- Result depends on the real world:
  - real world ~ our constraints (p(4) = 0.4, p(6) = 0.2), everything else no specific constraints:
    - our average error: D(q||p) [recall: Kullback-Leibler distance]
  - real world ~ orig. constraints + p(1) = 0.25:
    - q is best (but hey, then we should have started with all 3 constraints!)

Things in Perspective: n-gram LM

- Is an n-gram model a ME model?
  - yes if we believe that trigrams are the all and only constraints
    - trigram model constraints: $p(z|x,y) = c(x,y,z)/c(x,y)$
  - no room for any "adjustments"
    - like if we say $p(2) = 0.7$, $p(6) = 0.3$ for throwing a die
- Accounting for the apparent inadequacy:
  - smoothing
  - ME solution: (sort of) smoothing "built in"
    - constraints from training, maximize entropy on training + heldout

- generally these features can model anything we choose. They can look left / right to the word etc.
- they can also accept inputs that are formed by another preprocessing component

Features and Constraints

<=> ToLower(w_i) + w_i

- each feature must return a prob., value predicted is part of the input

- Introducing...
  - binary valued selector functions ("features"):
    - $f_i(y,x) \in \{0,1\}$, where
      - $y \in Y$ (sample space of the event being predicted: words, tags, ...),
      - $x \in X$ (space of contexts, e.g. word/tag bigrams, unigrams, weather conditions, of - in general - unspecified nature/length/size)

unusual!

  - constraints:
    - $E_p(f_i(y,x)) = E'(f_i(y,x))$ (= empirical expectation)
    - recall: expectation relative to distribution p: $E_p(f_i) = \sum_{y,x} p(x,y) f_i(y,x)$
    - empirical expectation: $E'(f_i) = \sum_{y,x} p'(x,y) f_i(y,x) = 1/|T| \sum_{t=1..T} f_i(y_t,x_t)$
    - notation: $E'(f_i(y,x)) = d_i$: constraints of the form $E_p(f_i(y,x)) = d_i$

Additional Constraint (Ensuring Probability Distribution)

- The model's p(y|x) should be probability distribution:
  - add an "omnipresent" feature $f_0(y,x) = 1$ for all y,x
  - constraint: $E_p(f_0(y,x)) = 1$
- Now, assume:
  - We know the set $S = \{f_i(y,x), i=0..N\}$ ($|S| = N+1$)
  - We know all the constraints
    - i.e. a vector d_i, one for each feature, i=0..N
- Where are the parameters?
  - ...we do not even know the form of the model yet

The Model

- Given the constraints, what is the form of the model which maximizes the entropy of p?
- Use Lagrangian Multipliers:
  - minimizing some function $\phi(z)$ in the presence of N constraints $g_i(z) = d_i$ means to minimize

$$\phi(x) - \Sigma_{i=1..N}\lambda_i(g_i(x) - d_i) \qquad \text{(w.r.t. all } \lambda_i \text{ and x)}$$

  - our case, minimize

$$A(p) = -H(p) - \Sigma_{i=1..N}\lambda_i(E_p(f_i(y,x)) - d_i) \quad \text{(w.r.t. all } \lambda_i \text{ and p!)}$$

  - i.e. $\phi(z) = -H(p)$, $g_i(z) = E_p(f_i(y,x))$ (variable z ~ distribution p)

Loglinear (Exponential) Model

- Maximize: for p, derive (partial derivation) and solve A'(p) = 0:

$$\delta[-H(p) - \Sigma_{i=0..N}\lambda_i(E_p(f_i(y,x)) - d_i)]/\delta p = 0$$

$$\delta[\ \Sigma\ p \log(p) - \Sigma_{i=0..N}\lambda_i((\Sigma\ p\ f_i) - d_i)]/\delta p = 0$$

...

$$1 + \log(p) - \Sigma_{i=0..N}\lambda_i\ f_i = 0$$

$$1 + \log(p) = \Sigma_{i=1..N}\lambda_i\ f_i + \lambda_0$$

$$p = e^{\Sigma_{i=1..N}\lambda_i\ f_i + \lambda_0 - 1}$$

- $p(y,x) = (1/Z)\ e^{\Sigma_{i=1..N}\lambda_i f_i(y,x)}$ ($Z = e^{1-\lambda_0}$, the normalization factor)

The form of the constraints for the Maximum Entropy model is defined as

Vyberte jednu nebo více možností:

- a. $1/|T| \sum_{t=1..T} \sum_{y \in Y} p(y|x_t) f_i(y,x_t) - d_i = 0$, where T is the training data (and |T| its size), p(y|x) the conditional model probability distribution, y the predicted variable and x the context (x_t is the concrete context at t-th data item), and f_i the i-th feature. d_i is the true feature count as extracted from the training data.
  ✓ Yes, this is the approximation formula using the data-oriented expected value computation due to the complexity of summing over all possible xs (which is often impossible to enumerate).
- b. $\sum_{y,x} p(y|x) f_i(y,x) - d_i = 0$, where p is the conditional model distribution, f_i are the features, and d_i is the true count as extracted from the training data.
  ✗ No. The weight in the expected value formula computation must always be the joint distribution, not the conditional one.
- c. $E_p(f_i(y,x)) - d_i = 0$, where E is the expected value of feature count expressed in terms of the probability distribution p as $E_p(f_i) = \Sigma_{y,x} p(x,y) f_i(y,x)$, with p being the model joint distribution, and d_i is the true count as extracted from the training data.
  ✓ Yes. It simply says that the (joint) distribution p must be such that it models the feature count in such a way that it equals to the true count as found in the data, by using the standart optimization technique known as Lagrange multipliers (where the "multipliers" then serve as the feature weights).

this is what you want to use..

↳ this is how the constraints work

?

there can be potentially unlimited number of features, therefore it is not good idea to enumerate over all x

Getting the Lambdas: Setup

- Model: $p(y,x) = (1/Z)\, e^{\Sigma_{i=1..N}\lambda_i f_i(y,x)}$
- Generalized Iterative Scaling (G.I.S.)
  - obeys form of model & constraints:
    - $E_p(f_i(y,x)) = d_i$
  - G.I.S. needs, in order to work, $\forall y,x\ \Sigma_{i=1..N} f_i(y,x) = C$
    - to fulfill, define additional constraint:
    - $f_{N+1}(y,x) = C_{max} - \Sigma_{i=1..N} f_i(y,x)$, where $C_{max} = \max_{x,y} \Sigma_{i=1..N} f_i(y,x)$
  - also, approximate (because $\Sigma_{x \in \text{All contexts}}$ is not (<u>never</u>) feasible)
    - $E_p(f_i) = \Sigma_{y,x} p(x,y) f_i(y,x) \cong 1/|T| \Sigma_{t=1..T} \Sigma_{y \in Y} p(y|x_t) f_i(y,x_t)$
      (use $p(y,x)=p(y|x)p'(x)$, where $p'(x)$ is empirical i.e. from data T)

Generalized Iterative Scaling

- 1. Initialize $\lambda_i^{(1)}$ (any values, e.g. 0), compute d_i, i=1..N+1
- 2. Set iteration number n to 1.
- 3. Compute current model distribution expected values of all the constraint expectations

  $E_{p^{(n)}}(f_i)$ (based on $p^{(n)}(y|x_t)$)
  - [pass through data, see previous slide; at each data position t, compute $p^{(n)}(y,x_t)$, normalize]
- 4. Update $\lambda_i^{(n+1)} = \lambda_i^{(n)} + (1/C) \log(d_i/E_{p^{(n)}}(f_i))$
- 5. Repeat 3.,4. until convergence.

Comments on Features

- Advantage of "variable" (~ not fixed) context in f(y,x):
  - any feature o.k. (examples mostly for tagging):
    - previous word's part of speech is VBZ or VB or VBP, y is DT
    - next word: capitalized, current: ".", and y is a sentence break (SB detect)
    - y is MD, and the current sentence is a question (last w: question mark)
    - tag assigned by a different tagger is VBP, and y is VB
    - it is before Thanksgiving and y is "turkey" (Language modeling)
    - even manually written „rules," e.g. y is VBZ and there is ...
  - remember, the predicted event plays a role in a feature:
    - also, a set of events: f(y,x) is true if y is NNS or NN, and x is ...
    - x can be ignored as well ("unigram" features)

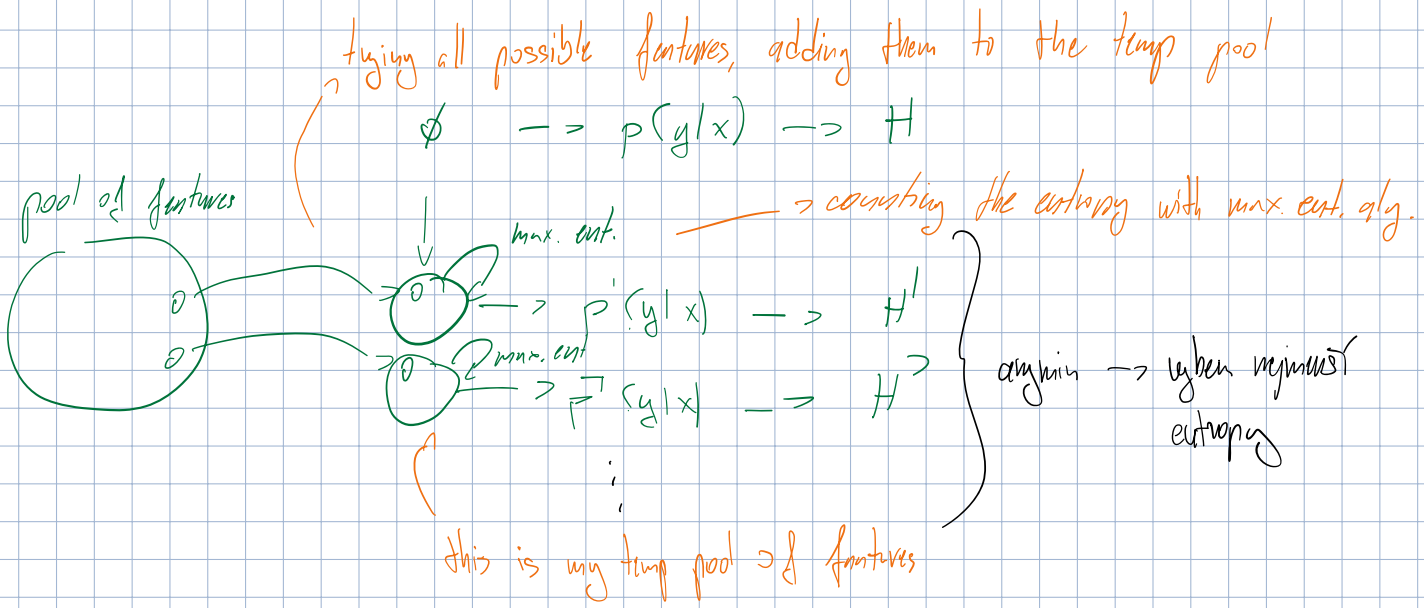
Feature Selection

- Advantage:
  - throw in many features
    - typical case: specify templates manually (pool of features P), fill in from data, possibly add some specific manually written features
  - let the machine select
    - Maximum Likelihood ~ Minimum Entropy on training data
    - after, of course, computing the λ_i's using the MaxEnt algorithm
- Naive (greedy of course) algorithm:
  - start with empty S, add feature at a time (MLE after ME)
  - too costly for full computation (|S| x |P| x |ME-time|)
  - Solution: see Berger & DellaPietras

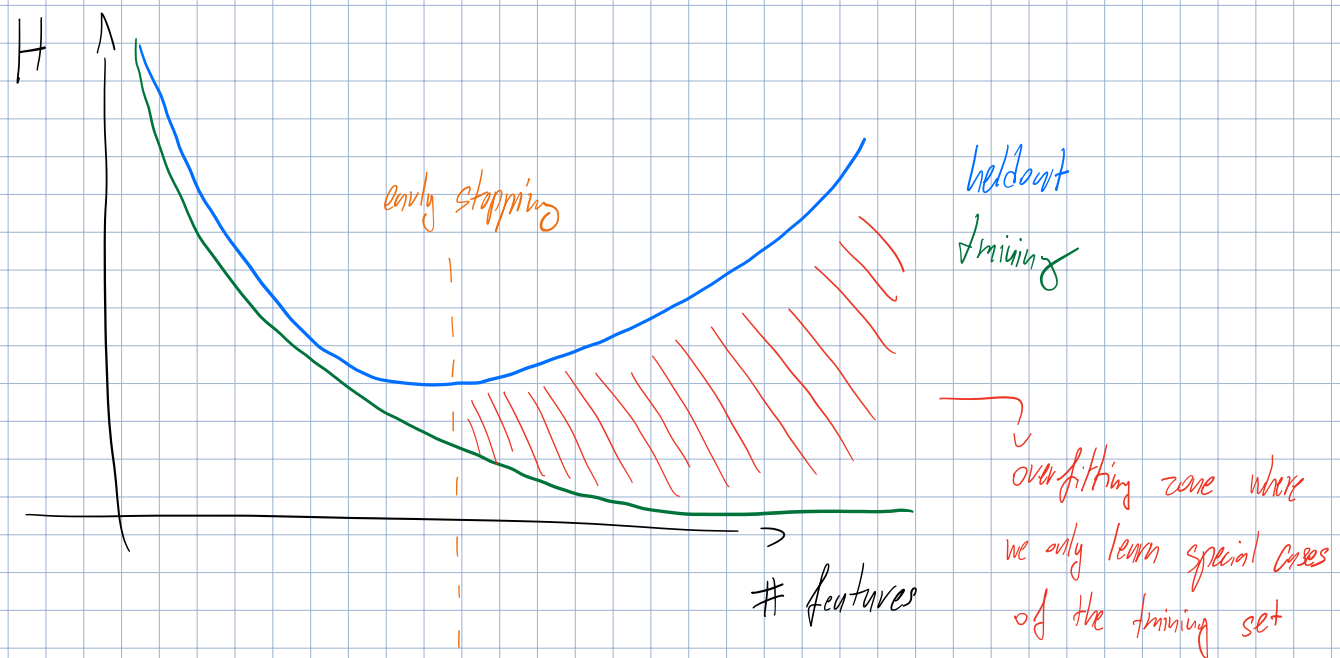
References

- Manning-Schuetze:
  - Section 16.2
- Jelinek:
  - Chapter 13 (includes application to LM)
  - Chapter 14 (other applications)
- Berger & DellaPietras in CL, 1996, 1997
  - Improved Iterative Scaling (does not need $\Sigma_{i=1..N} f_i(y,x) = C$)
  - “Fast” Feature Selection!
- Hildebrand, F.B.: Methods of Applied Math., 1952

With max. entropy alg. I obtain the feature set and its weight.
And then I look for such combination of features and weights
such that it gives the lowest entropy. → this deals with overtraining



At the end, the entropy is decreasing too slowly and we only add
too specialised rules...
↳ and therefore they would only little decrease
the entropy and very likely only on the training set



↳ overfitting zone where
we only learn special cases
of the training set

Maximum Entropy Tagging

The Task, Again

- Recall:
  - tagging ~ morphological disambiguation
  - tagset $V_T \subset (C_1,C_2,...C_n)$
    - C_i - morphological categories, such as POS, NUMBER, CASE, PERSON, TENSE, GENDER, ...
  - mapping $w \rightarrow \{t \in V_T\}$ exists
    - restriction of Morphological Analysis: $A^+ \rightarrow 2^{(L,C1,C2,...,Cn)}$

      where A is the language alphabet, L is the set of lemmas
  - extension to punctuation, sentence boundaries (treated as words)

Maximum Entropy Tagging Model

- General

  $p(y,x) = (1/Z)\, e^{\Sigma_{i=1..N}\lambda_i f_i(y,x)}$

  → upon what to predict
  → context

  Task: find λ_i satisfying the model and constraints

  - $E_p(f_i(y,x)) = d_i$

  where

  - $d_i = E'(f_i(y,x))$ (empirical expectation i.e. feature frequency)
- Tagging

  $p(t,x) = (1/Z)\, e^{\Sigma_{i=1..N}\lambda_i f_i(t,x)}$ (λ_0 might be extra: cf. μ in AR)

  - $t \in$ Tagset,
  - $x \sim$ context (words and tags alike; say, up to three positions R/L)

Features for Tagging

– we need only reasonable # of features

- Context definition
  - two words back and ahead, two tags back, current word:
    - $x_i = (w_{i-2},t_{i-2},w_{i-1},t_{i-1},w_i,w_{i+1},w_{i+2})$
  - features may ask any information from this window
    - e.g.:
      - previous tag is DT
      - previous two tags are PRP$ and MD, and the following word is “be”
      - current word is “an”
      - suffix of current word is “ing”
    - do not forget: feature also contains t_i, the current tag:
      - feature #45: suffix of current word is “ing” & the tag is VBG $\Leftrightarrow$ $f_{45} = 1$

Feature Selection

- The ritght data-based way:
  - (try to) test all possible feature combinations
    - features may overlap, or be redundant; also, general or specific - impossible to select manually
  - greedy selection:
    - add one feature at a time, test if (good) improvement:
      - keep if yes, return to the pool of features if not
  - even this is costly, unless some shortcuts are made
    - see Berger & DPs for details
- The other way:
  - use some heuristic to limit the number of features

Limiting the Number of Features

- Always
  - use contexts which appear in the training data (lossless selection)
- Some heuristics
  - use features appearing only L-times in the data (L ~ 10)
  - use w_i-derived features which appear with rare words only
  - do not use all combinations of context
  - but then, use all of them, and compute the λ_i only once using the Generalized Iterative Scaling algorithm

Feature Examples (Context)

- From A. Ratnaparkhi (EMNLP, 1996, UPenn)
  - $t_i = T$, $w_i = X$ (frequency $c > 4$):
    - t_i = VBG, w_i = selling
  - $t_i = T$, w_i contains uppercase char (rare):
    - t_i = NNP, tolower(w_i) $\neq w_i$
  - $t_i = T$, $t_{i-1} = Y$, $t_{i-2} = X$:
    - t_i = VBP, t_{i-2} = PRP, t_{i-1} = RB
- Other examples of possible features:
  - $t_i = T$, t_j is X, where j is the closest left position where Y
    - t_i = VBZ, t_j = NN, Y $\Leftrightarrow$ $t_j \in$ {NNP, NNS, NN}

Feature Examples (Lexical/Unknown)

- From AR:
  - t_i = T, suffix(w_i)= X (length X < 5):
    - t_i = JJ, suffix(w_i) = eled (traveled, leveled, ...)
  - t_i = T, prefix(w_i)= X (length X < 5):
    - t_i = JJ, prefix(w_i) = well- (well-done, well-received,...)
  - t_i = T, w_i contains hyphen:
    - t_i = JJ, '-' in w_i (open-minded, short-sighted,...)
- Other possibility, for example:
  - t_i = T, w_i contains X:
    - t_i = NounPl, w_i contains umlaut (ä,ö,ü) (Wörter, Länge,...)

“Specialized” Word-based Features

- List of words with most errors (WSJ, Penn Treebank):
  - about, that, more, up, ...
- Add “specialized”, detailed features:
  - $t_i = T$, $w_i = X$, $t_{i-1} = Y$, $t_{i-2} = Z$:
    - t_i = IN, w_i = about, t_{i-1} = NNS, t_{i-2} = DT
  - possible only for relatively high-frequency words
- Slightly better results (also, problems with inconsistent [test] data)

Maximum Entropy Tagging: Results

- Base experiment (133k words, < 3% unknown):
  - 96.31% word accuracy
- Specialized features added:
  - 96.49% word accuracy
- Consistent subset (training + test)
  - 97.04% word accuracy (97.13% w/specialized features)
    - Best in 2000; for details, see the AR paper
- [Now: perceptron 97%; Deep neural networks: 98%
  - Collins 2002, Raab 2009, Straka 2018 (Czech)]

Feature-Based Tagging

The Task, Again

- Recall:
  - tagging ~ morphological disambiguation
  - tagset $V_T \subset (C_1,C_2,...C_n)$
    - C_i - morphological categories, such as POS, NUMBER, CASE, PERSON, TENSE, GENDER, ...
  - mapping $w \rightarrow \{t \in V_T\}$ exists
    - restriction of Morphological Analysis: $A^+ \rightarrow 2^{(L,C1,C2,...,Cn)}$

      where A is the language alphabet, L is the set of lemmas
  - extension to punctuation, sentence boundaries (treated as words)

Feature Selection Problems

- Main problem with Maximum Entropy [tagging]:
  - Feature Selection (if number of possible features is in the hundreds of thousands or millions)
  - No good way
    - best so far: Berger & DP's greedy algorithm
    - heuristics (cutoff based: ignore low-count features)
- Goal:
  - few but "good" features ("good" ~ high predictive power ~ leading to low final cross entropy)

Pro češtinu je těžké slovosled odlišit 1. a 4. pád ~ free word order...

Feature-based Tagging

- Idea:
  - save on computing the weights (λ_i)
    - are they really so important?
  - concentrate on feature selection
- Criterion (training):
  - error rate (~ accuracy; borrows from Brill's tagger)
- Model form (probabilistic - same as for Maximum Entropy):

$$p(y|x) = (1/Z(x))\ e^{\Sigma_{i=1..N}\lambda_i f_i(y,x)}$$

→ Exponential (or Loglinear) Model

Feature Weight (Lambda) Approximation

- Let Y be the sample space from which we predict (tags in our case), and $f_i(y,x)$ a b.v. feature
- Define a "batch of features" and a "context feature":

  $B(x) = \{f_i$; all f_i's share the same context $x\}$ $= \{f(y,x) \mid \forall y\}$

  $f_{B(x)}(x') = 1 \Leftrightarrow_{df}$ $x \subset x'$ (x is part of x')
  - in other words, holds wherever a context x is found
- Example:

  $f_1(y,x) = 1 \Leftrightarrow_{df}$ y=JJ, left tag = JJ

  $f_2(y,x) = 1 \Leftrightarrow_{df}$ y=NN, left tag = JJ

  B(left tag = JJ) = $\{f_1, f_2\}$ (but not, say, [y=JJ, left tag = DT])

? může pouze nabýt jedné hodnoty v kontextu

aby tedy model byl $f_{B(x)}(x') \leq 1$.

Estimation

- Compute:

counting it for each single tag

$p(y|B(x)) = (1/Z(B(x))) \Sigma_{d=1..|T|} \delta(y_d,y) f_{B(x)}(x_d)$

  - frequency of y relative to all places where any of B(x) features holds for some y; Z(B(x)) is the natural normalization factor
  - $Z(B(x)) = \Sigma_{d=1..|T|} f_{B(x)}(x_d)$ → here the form of the context is given by the batch - it grouped all the same contexts

"compare" to uniform distribution:

  - $\alpha(y,B(x)) = p(y|B(X)) / (1 / |Y|)$ → almost like PMI

    $\alpha(y,B(x)) > 1$ for $p(y|B(x))$ better than uniform; and vice versa

- If $f_i(y,x)$ holds for exactly one y (in a given context x),

  then we have 1:1 relation between $\alpha(y,B(x))$ and $f_i(y,x)$ from B(x)

  and $\lambda_i = \log(\alpha(y,B(x)))$

NB: works in constant time independent of λ_j, $j \neq i$

In fact this approximation is very good and helps testing many features

What we got

We can have only one feature that is true in the completely same context (position, words before/after)

- Substitute:

$$p(y|x) = (1/Z(x))\, e^{\sum_{i=1..N} \lambda_i f_i(y,x)} =$$

$$= (1/Z(x)) \prod_{i=1..N} \alpha(y,B(x))^{f_i(y,x)}$$

$$= (1/Z(x)) \prod_{i=1..N} (|Y|\, p(y|B(x)))^{f_i(y,x)}$$

$$= (1/Z'(x)) \prod_{i=1..N} (p(y|B(x)))^{f_i(y,x)}$$

$$= (1/Z'(x)) \prod_{B(x');\, x' \subset x} p(y|B(x'))$$

$1/|Y|$

$f_i(y,x) \in \{0,1\}$ ... only 1:1 mapping

... Naive Bayes (independence assumption)

– it must be only approx, since independence assumption does not generally holds in between features.

The features are then treated as independent, so we can even manipulate the weights and try to boost it.

The Reality

- take advantage of the exponential form of the model (do ***not*** reduce it completely to naive Bayes):
  - vary $\alpha(y,B(x))$ up and down a bit (quickly)
    - captures dependence among features
  - recompute using "true" Maximum Entropy
    - the ultimate solution
  - combine feature batches into one, with new $\alpha(y,B(x'))$
    - getting very specific features

Search for Features

- Essentially, a way to get rid of unimportant features:
  - start with a pool of features extracted from full data
  - remove infrequent features (small threshold, < 2)
  - organize the pool into batches of features
- Selection from the pool P:
  - start with empty S (set of selected features)
  - try all features from the pool, compute $\alpha(y,B(x))$, compute error rate over training data.
  - add the best feature batch permanently; stop when no correction made [complexity: |P| x |S| x |T|]

↳ we suppose these "first ten" are independent – even though in reality they are not

Adding Features in Blocks, Avoiding the Search for the Best

- Still slow; solution: add ***ten (5,20)*** best features at a time, assuming they are independent (i.e., the next best feature would change the error rate the same way as if no intervening addition of a feature is made).
- Still slow [(|P| x |S| x |T|)/10, or 5, or 20]; solution:
- Add ***all*** features improving the error rate by a certain threshold; then gradually lower the threshold down to the desired value; complexity [|P| x log|S| x |T|] if

$$\text{threshold}^{(n+1)} = \text{threshold}^{(n)} / k,\ k > 1 \text{ (e.g. } k = 2)$$

↳ we first get the very strong features, efficiently lowering the number of evaluations

Types of Features

- Position:
  - current
  - previous, next
  - defined by the closest word with certain major POS
- Content:
  - word (w), tag(t) - left only, "Ambiguity Class" (AC) of a subtag (POS, NUMBER, GENDER, CASE, ...)
- Any combination of position and content
- Up to three combinations of (position,content)

this can indeed # of options A LOT!

there can be lot information stored

Ambiguity Classes (AC)

- Also called “pseudowords” (MS, for word sense disambiguationi task), here: “pseudotags”
- AC (for tagging) is a set of tags (used as an indivisible token).
  - Typically, these are the tags assigned by a morphology to a given word:
    - MA(books) [restricted to tags] = { NNS, VBZ }:

      AC = NNS_VBZ
- Advantage: deterministic

  → looking at the ACs (and words, as before) to the ***right*** allowed

Subtags

- Inflective languages: too many tags → data sparseness
- Make use of separate categories (remember morphology):
  - tagset $V_T \subset (C_1, C_2, ... C_n)$
    - C_i - morphological categories, such as POS, NUMBER, CASE, PERSON, TENSE, GENDER, ...
- Predict (and use for context) the individual categories
- Example feature:
  - previous word is a noun, and current CASE subtag is genitive
- Use separate ACs for subtags, too (AC_{POS} = N_V)

Combining Subtags

- Apply the separate prediction (POS, NUMBER) to
  - MA(books) = { (Noun, Pl), (VerbPres, Sg)}
- Now what if the best subtags are
  - Noun for POS
  - Sg for NUMBER
    - (Noun, Sg) is ***not*** possible for books

- Allow only possible combinations (based on MA)
- Use independence assumption (Tag = (C_1, C_2, ..., C_n)):

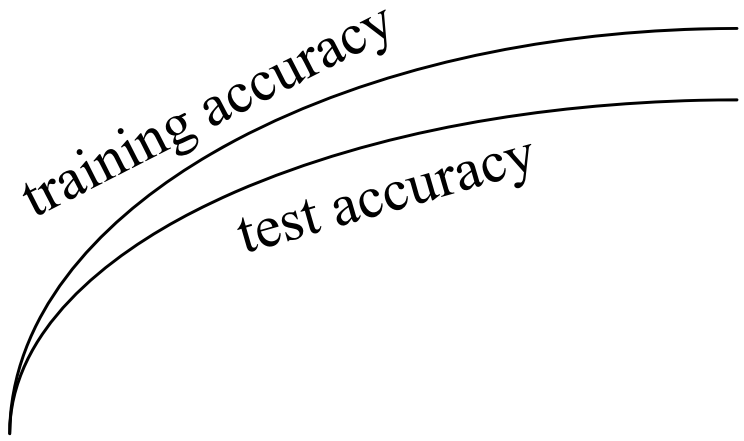
$$\text{(best) Tag} = \text{argmax}_{\text{Tag} \in \text{MA(w)}} \prod_{i=1..|\text{Categories}|} p(C_i|w,x)$$

Smoothing

- Not needed in general (as usual for exponential models)
  - however, some basic smoothing has an advantage of not learning unnecessary features at the beginning
  - very coarse: based on ambiguity classes
    - assign the most probable tag for each AC, using MLE
    - e.g. NNS for AC = NNS_VBZ
  - last resort smoothing: unigram tag probability
  - can be even parametrized from the outside
  - also, needed during training

Overtraining

- Does not appear in general
  - usual for exponential models
  - does appear in relation to the training curve:



  - but does not go down until very late in the training (singletons do cause overtraining)

Parsing: Introduction

Context-free Grammars

- Chomsky hierarchy
  - Type 0 Grammars/Languages
    - rewrite rules $\alpha \rightarrow \beta$; α,β are any string of terminals and nonterminals
  - Context-sensitive Grammars/Languages
    - rewrite rules: $\alpha X \beta \rightarrow \alpha\gamma\beta$, where X is nonterminal, α,β,γ any string of terminals and nonterminals (γ must not be empty)
  - **Context-free Grammars/Lanuages**
    - rewrite rules: $X \rightarrow \gamma$, where X is nonterminal, γ any string of terminals and nonterminals
  - Regular Grammars/Languages
    - rewrite rules: $X \rightarrow \alpha$ Y where X,Y are nonterminals, α string of terminal symbols; Y might be missing

Parsing Regular Grammars

- Finite state automata
  - Grammar $\leftrightarrow$ regular expression $\leftrightarrow$ finite state automaton
- Space needed:
  - constant
- Time needed to parse:
  - linear (~ length of input string)
- Cannot do e.g. $a^n b^n$, embedded recursion (context-free grammars can)

Parsing Context Free Grammars

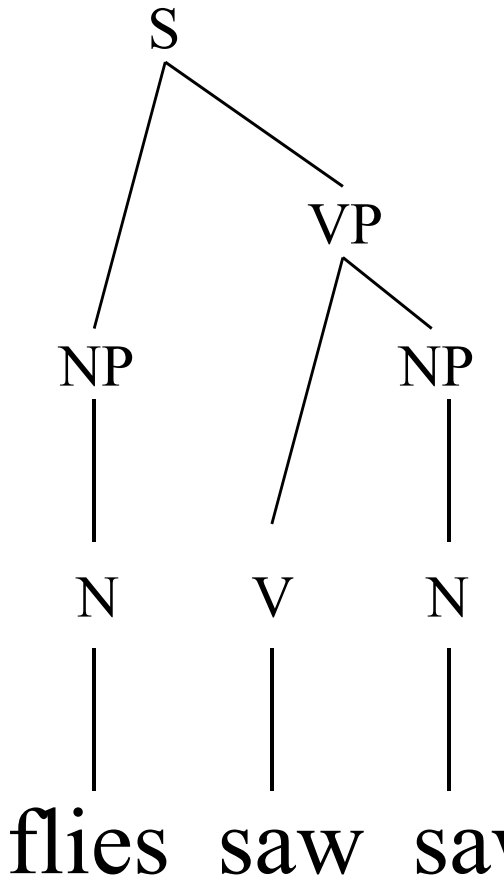
- Widely used for surface syntax description (or better to say, for correct word-order specification) of natural languages
- Space needed:
  - stack (sometimes stack of stacks)
    - in general: items ~ levels of actual (i.e. in data) recursions
- Time: in general, $O(n^3)$
- Cannot do: e.g. $a^n b^n c^n$ (Context-sensitive grammars can)

Example Toy NL Grammar

- #1 S → NP
- #2 S → NP VP
- #3 VP → V NP
- #4 NP → N
- #5 N → flies
- #6 N → saw
- #7 V → flies
- #8 V → saw

2 věty

8 vět



Celkem existuje 10 vět, patřících do této gramatiky.

Shift-Reduce Parsing in Detail

Grammar Requirements

- Context Free Grammar with
  - no empty rules ($N \rightarrow \varepsilon$)
    - can always be made from a general CFG, except there might remain one rule $S \rightarrow \varepsilon$ (easy to handle separately)
  - recursion OK
- Idea:
  - go bottom-up (otherwise: problems with recursion)
  - construct a Push-down Automaton (non-deterministic in general, PNA)
  - delay rule acceptance until all of a (possible) rule parsed

PNA Construction - Elementary Procedures

Dot serves as a place holder to mark where the grammar currently is.

- Initialize-Rule-In-State(q,$A \rightarrow \alpha$) procedure:
  - Add the rule ($A \rightarrow \alpha$) into a state q.
  - Insert a dot in front of the R[ight]H[and]S[ide]: $A \rightarrow . \alpha$
- Initialize-Nonterminal-In-State(q,A) procedure:
  - Do "Initialize-Rule-In-State(q,$A \rightarrow \alpha$)" for all rules having the nonterminal A on the L[eft]H[and]S[ide]
- Move-Dot-In-Rule(q,$A \rightarrow \alpha . Z\beta$) procedure:
  - Create a new rule in state q: $A \rightarrow \alpha Z . \beta$, Z term. or not

PNA Construction

- Put 0 into the (FIFO/LIFO) list of incomplete states, and do Initialize-Nonterminal-In-State(0,S)
- Until the list of incomplete states is not empty, do:
  1. Get one state, i from the list of incomplete states.
  2. Expand the state:
     - Do recursively Initialize-Nonterminal-In-State(i,A) for all nonterminals A right after the dot in any of the rules in state i.
  3. If the state matches exactly some other state already in the list of complete states, renumber all shift-references to it to the old state and discard the current state.

C → remove redundancy

PNA Construction (Cont.)

4. Create a set T of Shift-References (or, transition/continuation links) for the current state i {(Z,x)}:
   - Suppose the highest number of a state in the incomplete state list is n.
   - For each symbol Z (regardless if terminal or nonterminal) which appears after the dot in any rule in the current state q, do:
     - increase n to n+1
     - add (Z,n) to T
       - *NB: each symbol gets only one Shift-Reference, regardless of how many times (i.e. in how many rules) it appears to the right of a dot.*
     - Add n to the list of incomplete states
     - Do Move-Dot-In-Rule(n,A → α . Zβ)

tedy znamená, že gramatika dovoluje přejít na terminály.

5. Create Reduce-References for each rule in the current state i:
   - For each rule of the form (A → α .) (i.e. dot at the end) in the current state, attach to it the rule number r of the rule A → α from the grammar.

Using the PNA (Initialize)

- Maintain two stacks, the input stack I and the state stack Q.
- Maintain a stack B[acktracking] of the two stacks.
- Initialize the I stack to the input string (of terminal symbols), so that the first symbol is on top of it.
- Initialize the stack Q to contain state 0.
- Initialize the stack B to empty.

Using the PNA (Parse)

- Do until you are not stuck and/or B is empty:
  - Take the top of stack Q state ("current" state i).
  - Put all possible reductions in state i on stack B, including the contents of the current stacks I and Q.
  - Get the symbol from the top of the stack I (symbol Z).
  - If (Z,x) exists in the set T associated with the current state i, push state x onto the stack Q and remove Z from I. Continue from beginning.
  - Else pop the first possibility from B, remove n symbols from the stack Q, and push A to I, where $A \rightarrow Z_1 ... Z_n$ is the rule according which you are reducing.

trying another pass.

Small Example

#1 S → NP VP
#2 NP → N
#3 VP → V NP
#4 N → a_cat
#5 N → a_dog
#6 V → saw

Grammar

no ambiguity,
no recursion

Tables: <symbol> <state>: shift
#<rule>: reduction

0 S → . NP VP	NP 1
NP → . N	N 2
N → . a_cat	a_cat 3
N → . a_dog	a_dog 4

1 S → NP . VP	VP 5
VP → . V NP	V 6
V → . saw	saw 7
2 NP → N .	#2
3 N → a_cat .	#4
4 N → a_dog .	#5
5 S → NP VP .	#1
6 VP → V . NP	NP 8
NP → . N	N 2
N → . a_cat	a_cat 3
N → . a_dog	a_dog 4
7 V → saw .	#6
8 VP → V NP .	#3

NB: dotted rules in states need not be kept

Parsing ≈ determines the structure of the string

↳ task „$x \in L$" is only a subtask of parsing

left-recursion / right-recursion CFG determines how well can we parse them ↳ and if to use bottom-up / top-down

Shift-reduce parser will be part of the exam...

LR(k) ~ I need at most k-steps look ahead/back, to deterministically choose the rewrite rule.

↳ proto pouze pro LR(0) nepotřebuji žádného

Next week, we will try construct one table by ourselves...

Small Example: Parsing(1)

- To parse: **a_dog saw a_cat**

Input stack (top on the left)	Rule	State stack (top on the left)	Comment(s)
• a_dog saw a_cat		0	
• saw a_cat		4 0	shift to 4 over a_dog
• N saw a_cat	#5	0	reduce #5: N → a_dog
• saw a_cat		2 0	shift to 2 over N
• NP saw a_cat	#2	0	reduce #2: NP → N
• saw a_cat		1 0	shift to 1 over NP
• a_cat		7 1 0	shift to 7 over saw
• V a_cat	#6	1 0	reduce #6: V → saw

Small Example: Parsing (2)

- ...still parsing: **a_dog saw a_cat**

• [V a_cat	#6	1 0] ← Previous parser configuration	
• a_cat		6 1 0	shift to 6 over V
•		3 6 1 0	empty input stack (not finished though!)
• N	#4	6 1 0	N inserted back
•		2 6 1 0	...again empty input stack
• NP	#2	6 1 0	
•		8 6 1 0	...and again
• VP	#3	1 0	two states removed (|RHS(#3)|=2)
•		5 1 0	
• S	#1	0	again, two items removed (RHS: NP VP)

Success: S/0 alone in input/state stack; reverse right derivation: 1,3,2,4,6,2,5

Big Example: Ambiguous and Recursive Grammar

- #1 S → NP VP
- #2 NP → NP REL VP
- #3 NP → N
- #4 NP → N PP
- #5 VP → V NP
- #6 VP → V NP PP
- #7 VP → V PP
- #8 PP → PREP NP

#9 N → a_cat

#10 N → a_dog

#11 N → a_hat

#12 PREP → in

#13 REL → that

#14 V → saw

#15 V → heard

Big Example: Tables (1)

0 S → . NP VP	NP	1
NP → . NP REL VP		
NP → . N	N	2
NP → . N PP		
N → . a_cat	a_cat	3
N → . a_dog	a_dog	4
N → . a_mirror	a_hat	5

1 S → NP . VP	VP	6
NP → NP . REL VP	REL	7
VP → . V NP	V	8
VP → . V NP PP		
VP → . V PP		
REL → . that	that	9
V → . saw	saw	10
V → . heard	heard	11

2 NP → N .	#3	
NP → N . PP	PP	12
PP → . PREP NP	PREP	13
PREP → . in	in	14

3 N → a_cat .	#9

4 N → a_dog .	#10

5 N → a_hat .	#11

6 S → NP VP .	#1

Big Example: Tables (2)

7 NP → NP REL . VP	VP	15
VP → . V NP	V	8
VP → . V NP PP		
VP → . V PP		
V → . saw	saw	10
V → . heard	heard	11

8 VP → V . NP	NP	16
VP → V . NP PP		
VP → V . PP	PP	17
NP → . NP REL VP		
NP → . N	N	2
NP → . N PP		
N → . a_cat	a_cat	3
N → . a_dog	a_dog	4
N → . a_hat	a_hat	5
PP → . PREP NP	PREP	13
PREP → . in	in	14

9 REL → that .	#13

10 V → saw .	#14

11 V → heard .	#15

12 NP → NP PP .	#4

13 PP → PREP . NP	NP	18
NP → . NP REL VP		
NP → . N	N	2
NP → . N PP		
N → . a_cat	a_cat	3
N → . a_dog	a_dog	4
N → . a_hat	a_hat	5

Big Example: Tables (3)

14 PREP → in .	#12	

15 NP → NP REL VP .	#2	

16 VP → V NP .	#5	
VP → V NP . PP	PP	19
NP → NP . REL VP	REL	7
PP → . PREP NP	PREP	13
PREP → . in	in	14
REL → . that	that	9

17 VP → V PP .	#7	

18 PP → PREP NP .	#8	
NP → NP . REL VP	REL	7
REL → . that	that 9	

19 VP → V NP PP .	#6	

Comments:
- states 2, 16, 18 have shift-reduce conflict
- no states with reduce-reduce conflict
- also, again there is no need to store the dotted rules in the states for parsing. Simply store the pair input/goto-state, or the rule number.

Big Example: Parsing (1)

- To parse: **a_dog heard a_cat in a_hat**

Input stack (top on the left)	Rule	State stack (top on the left)	Backtrack	Comment(s)
a_dog heard a_cat in a_hat		0		shifted to 4 over a_dog
heard a_cat in a_hat		4 0		shift to 4 over a_dog
N heard a_cat in a_hat	#10	0		reduce #10: N → a_dog
heard a_cat in a_hat		2 0		shift to 2 over N[1]
NP heard a_cat in a_hat	#3	0		reduce #3: NP → N
heard a_cat in a_hat		1 0		shift to 1 over NP
a_cat in a_hat		11 1 0		shift to 11 over heard
V a_cat in a_hat	#15	1 0		reduce #15: V → heard
a_cat in a_hat		8 1 0		shift to 8 over V

[1] see also next slide, last comment

Big Example: Parsing (2)

- ...still parsing: **a_dog heard a_cat in a_hat**

Input stack (top on the left)	Rule	State stack (top on the left) Backtrack	Comment(s)
• [a_cat in a_hat		8 1 0] ← [previous parser configuration]	
• in a_hat		3 8 1 0	shift to 3 over a_cat
• N in a_hat	#9	8 1 0	reduce #9: N → a_cat
• in a_hat		2 8 1 0 ⊗	shift to 2 over N; see why we need the state stack? we are in 2 again, but after we return, we will be in 8 not 0; also save for backtrack[1]!

[1] the whole input stack, state stack, and [reversed] list of rules used for reductions so far must be saved on the backtrack stack

Big Example: Parsing (3)

- ...still parsing: **a_dog heard a_cat in a_hat**

Input stack (top on the left)	Rule	State stack (top on the left) Backtrack	Comment(s)
• [in a_hat		2 8 1 0 ⊗] ← [previous parser configuration]	
• a_hat		14 2 8 1 0	shift to 14 over in
• PREP a_hat	#12	2 8 1 0	reduce #12: PREP → in[1]
• a_hat		13 2 8 1 0	shift to 13 over PREP
•		5 13 2 8 1 0	shift to 5 over a_hat
• N	#11	13 2 8 1 0	reduce #11: N → a_hat
•		2 13 2 8 1 0	shift to 2 over N
• NP	#3	13 2 8 1 0	shift not possible; reduce #3: NP → N[1 on s.19]
•		18 13 2 8 1 0	shift to 18 over NP

[1] when coming back to an ambiguous state [here: state 2] (after some reduction), reduction(s) are not considered; nothing put on backtrk stack

Big Example: Parsing (4)

- ...still parsing: **a_dog heard a_cat in a_hat**

Input stack (top on the left)	Rule	State stack (top on the left) Backtrack	Comment(s)
• [		18 13 2 8 1 0] ←	[previous parser config.]
• PP	#8	2 8 1 0	shift not possible; reduce #8[1 on s.19]: PP → PREP NP[1,prev.slide]
•		12 2 8 1 0	shift to 12 over PP
• NP	#4	8 1 0	reduce #4: NP → N PP
•		16 8 1 0	shift to 16 over NP
• VP	#5	1 0	shift not possible, reduce #5[1]: VP → V NP

[1] no need to keep the item on the backtrack stack; no shift possible now and there is only one reduction (#5) in state 16

Big Example: Parsing (5)

- ...still parsing: **a_dog heard a_cat in a_hat**

Input stack (top on the left)	Rule	State stack (top on the left) Backtrack	Comment(s)
• [VP	#5	1 0] ← [previous parser configuration]	
•		6 1 0	shift to 6 over VP
• S	#1	0	reduce #1: S → NP VP
			first solution found:
			1,5,4,8,3,11,12,9,15,3,10
			backtrack to previous ⊗ :
• in a_hat		2 8 1 0	was: shift over in, now[1]:
• NP in a_hat	#3	8 1 0	reduce #3: NP → N
• in a_hat		16 8 1 0 ⊗	shift to 16 over NP
• a_hat		14 16 8 1 0	shift, but put on backtrk

[1] no need to keep the item on the backtrack stack; no shift possible now and there is only one reduction (#3) in state 2

Big Example: Parsing (6)

- ...still parsing: **a_dog heard a_cat in a_hat**

Input stack (top on the left)	Rule	State stack (top on the left) Backtrack	Comment(s)
[a_hat		14 16 8 1 0 ⊗] ←	[previous parser config.]
PREP a_hat	#12	16 8 1 0	reduce #12: PREP → in
a_hat		13 16 8 1 0	shift over PREP[1 on s.17]
		5 13 16 8 1 0	shift over a_hat to 5
N	#11	13 16 8 1 0	reduce #11: N → a_hat
		2 13 16 1 0	shift to 2 over N
NP	#3	13 16 1 0	shift not possible[1 on s.19]
		18 13 16 1 0	shift to 18
PP	#8	16 1 0	shift not possible[1], red.#8
		19 16 1 0	shift to 19[1 on s.17]

[1] no need to keep the item on the backtrack stack; no shift possible now and there is only one reduction (#8) in state 18

Big Example: Parsing (7)

- ...still parsing: **a_dog heard a_cat in a_hat**

Input stack (top on the left)	Rule	State stack (top on the left) Backtrack	Comment(s)
[		19 16 8 1 0] ←	[previous parser config.]
VP	#6	1 0	red. #6: VP → V NP PP
		6 1 0	shift to 6 over VP
S	#1	0	next (2nd) solution: 1,6,8,3,11,12,3,[1]9,15,3,10 backtrack to previous ⊗ :
in a_hat		16 8 1 0	was: shift over in[1 on s.19],
VP in a_hat	#5	1 0	now red. #5: VP → V NP
in a_hat		6 1 0	shift to 6 over VP
S in a_hat	#1	0	error[2]; backtrack empty: stop

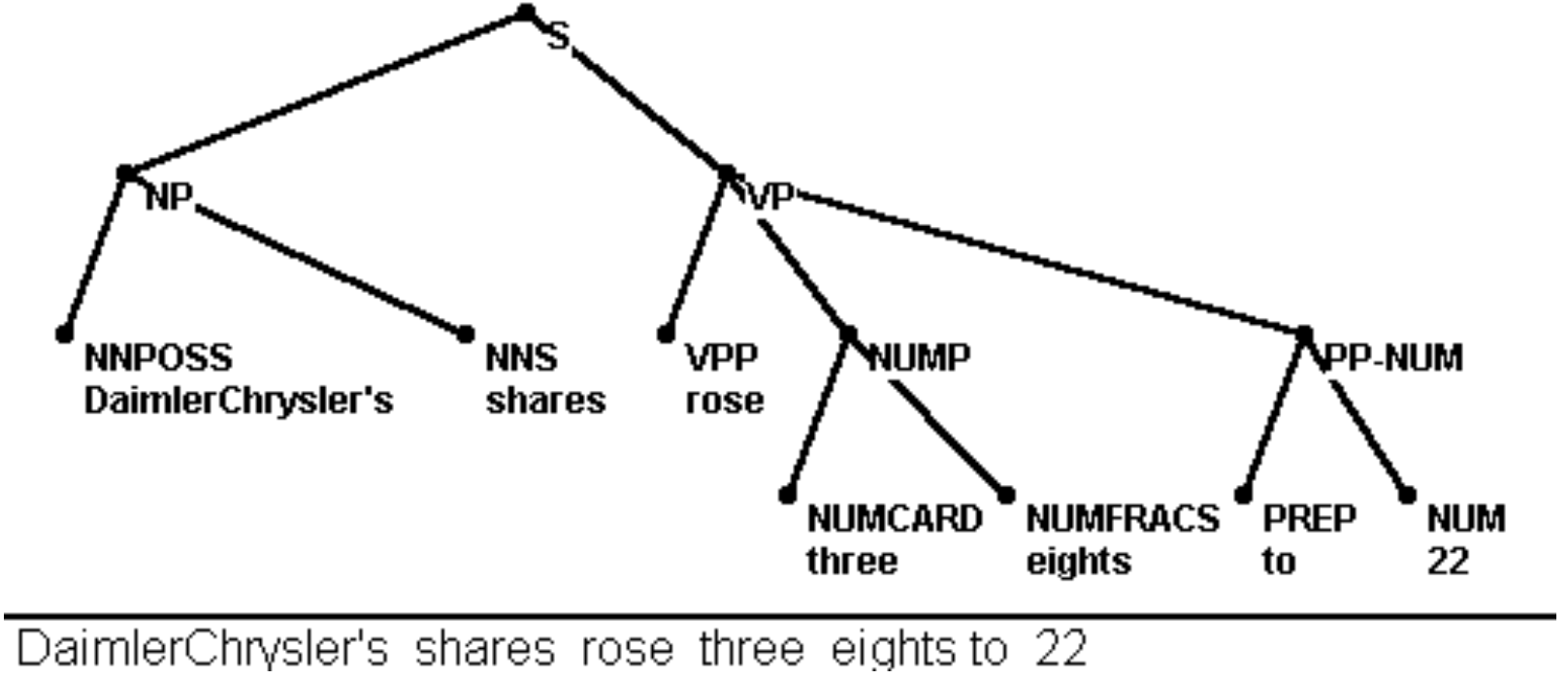
[1] continue list of rules at the orig. backtrack mark (s.16,line 3) [2] S (the start symbol) not alone in input stack when state stack = (0)

Treebanks, Treebanking and Evaluation

Phrase Structure Tree

- Example:

– bracketing is always possible, it is projective tree

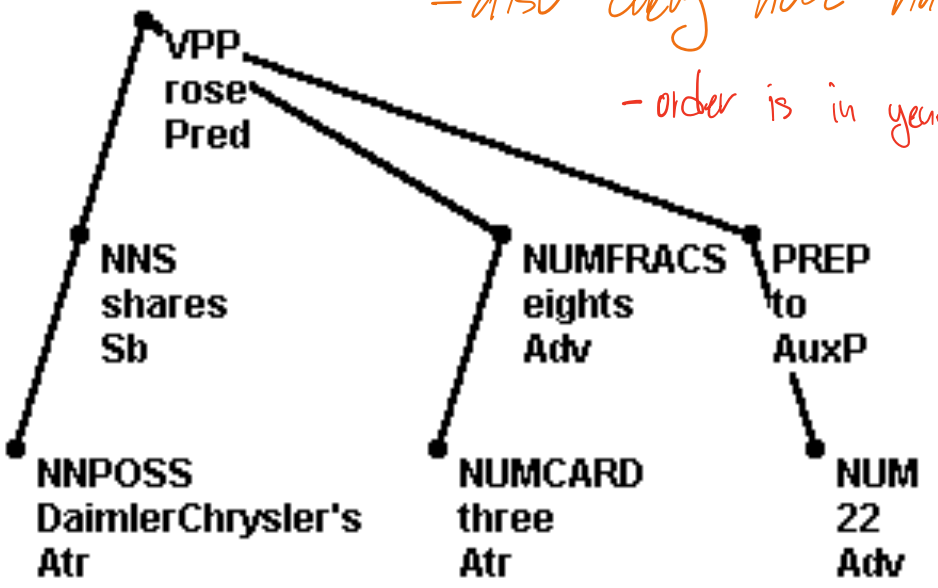


((DaimlerChrysler's shares)$_{NP}$ (rose (three eights)$_{NUMP}$ (to 22)$_{PP\text{-}NUM}$)$_{VP}$)$_{S}$

Dependency Tree

- Example:

every node has its word
- also every node has its parent (except root)
- order is in general not preserved
  ↳ ordering of sons is added sometimes



DaimlerChrysler's shares rose three eights to 22

$$rose_{Pred}(shares_{Sb}(DaimlerChrysler's_{Atr}),eights_{Adv}(three_{Atr}),to_{AuxP}(22_{Adv}))$$

Parser Development

- Use training data for learning phase
  - segment as needed (e.g., for heldout)
  - use all for
    - manually written rules (seldom today)
    - automatically learned rules/statistics
- Occasionally, test progress on Development Test Set
  - (simulates real-world data)
- When done, test on Evaluation Test Set
- ***Unbreakable Rule #1: Never look at Evaluation Test Data (not even indirectly, e.g. performance numbers)***

Evaluation

- Evaluation of parsers (regardless of whether manual-rule-based or automatically learned)
- Repeat: Test against Evaluation Test Data
- Measures:
  - Dependency trees:
    - Dependency Accuracy, Precision, Recall
  - Parse trees:
    - Crossing brackets
    - Labeled precision, recall [F-measure]

Dependency Parser Evaluation

- Dependency Recall:
  - R_D = Correct(D) / |S|
    - Correct(D): number of correct dependencies
      - correct: word attached to its true head
      - Tree root is correct if marked as root
    - |S| - size of test data in words (since |dependencies| = |words|)
- Dependency precision (if output not a tree, partial):
  - P_D = Correct(D) / Generated(D)
    - Generated(D) is the number of dependencies output
      - some words without a link to their head
      - some words with several links to (several different) heads

- F1 score not applicable, as there can be random tree above the words

Phrase Structure (Parse Tree) Evaluation

(in (New York))

golden

wrong

- Crossing Brackets measure
  - Example "truth" (evaluation test set):
    - ((the ((New York) - based company)) (announced (yesterday)))
  - Parser output - 0 crossing brackets:
    - ((the New York - based company) (announced yesterday))
  - Parser output - 2 crossing brackets:
    - (((the New York) - based) (company (announced (yesterday))))

2 jakého nedorozumění by! dosah vytvořen

- Labeled Precision/Recall:
  - Usual computation using bracket labels (phrase markers)

    T: $((\text{Computers})_{NP}\ (\text{are down})_{VP})_{S}$ ↔ P: $((\text{Computers})_{NP}\ (\text{are}\ (\text{down})_{NP})_{VP})_{S}$

    - Recall = 100%, Precision = 75%

this wasn't in gold

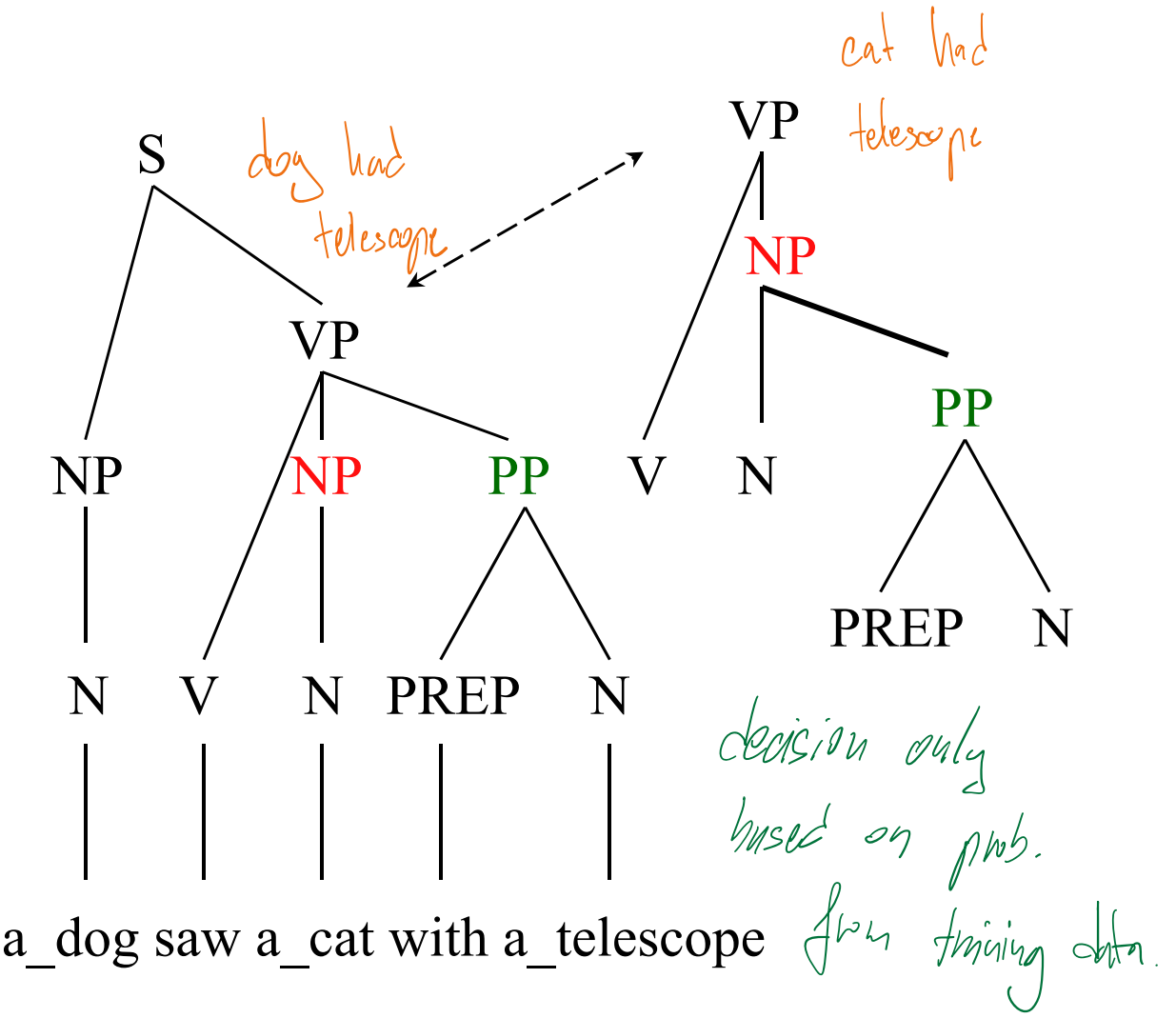
Probabilistic CFG (Introduction)

Context-free Grammars

- Chomsky hierarchy
  - Type 0 Grammars/Languages
    - rewrite rules $\alpha \rightarrow \beta$; α,β are any string of terminals and nonterminals
  - Context-sensitive Grammars/Languages
    - rewrite rules: $\alpha X \beta \rightarrow \alpha \gamma \beta$, where X is nonterminal, α,β,γ any string of terminals and nonterminals (γ must not be empty)
  - **Context-free Grammars/Lanuages**
    - rewrite rules: $X \rightarrow \gamma$, where X is nonterminal, γ any string of terminals and nonterminals
  - Regular Grammars/Languages
    - rewrite rules: $X \rightarrow \alpha$ Y where X,Y are nonterminals, α string of terminal symbols; Y might be missing

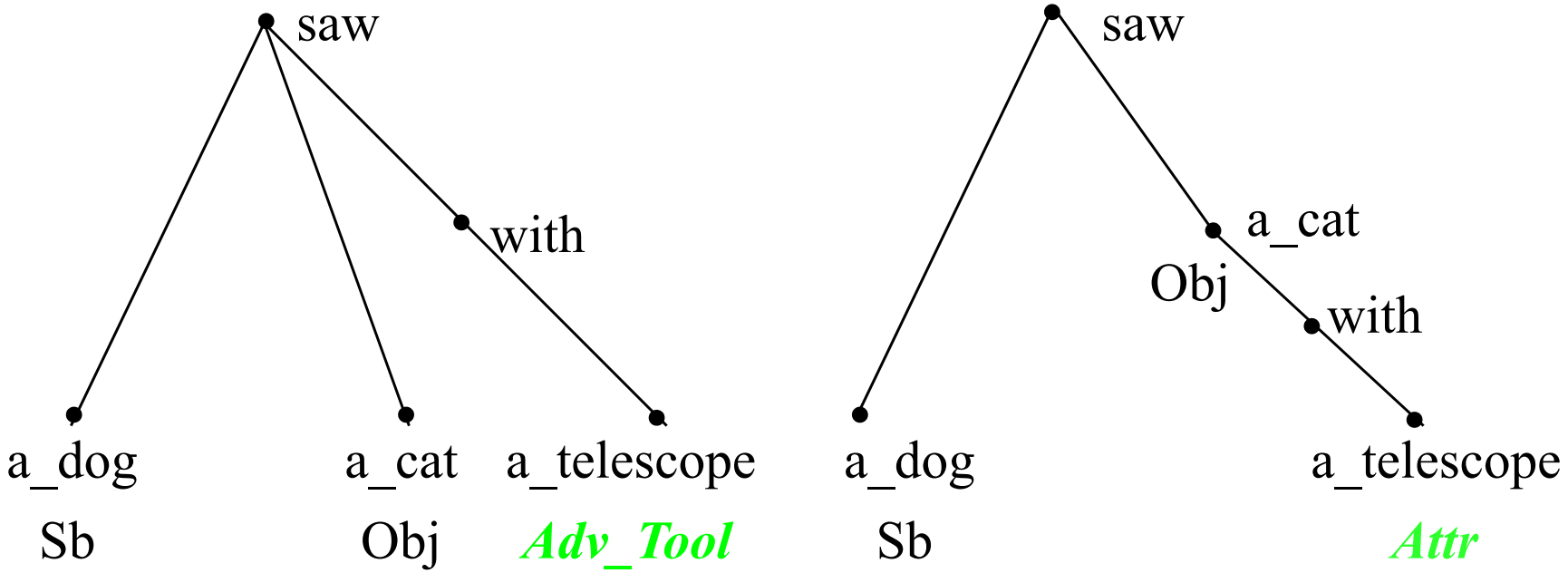
Another NLP Example

- #1 S → NP VP
- #2 VP → V NP PP
- #3 VP → V NP
- #4 NP → N
- #5 NP → N PP
- #6 PP → PREP N
- #7 N → a_dog
- #8 N → a_cat
- #9 N → a_telescope
- #10 V → saw
- #11 PREP → with



Dependency Style Example

- Same example, dependency representation



Probability of a Derivation Tree

- Both phrase/parse/derivational "grammatical"
- Different meaning: which is better [in context]?
- "Internal context": relations among phrases, words
- Probabilistic CFG:
  - relations among a mother node & daughter nodes
  - in terms of expansion [rewrite,derivation] probability
  - define probability of a derivation (i.e. parse) tree:

$$P(T) = \prod_{i=1..n} p(r(i))$$

r(i) are all rules of the CFG used to generate the sentence W of which T is a parse

Assumptions

- Independence assumptions (very strong!)
- Independence of context (neighboring subtrees)
- Independence of ancestors (upper levels)
- Place-independence (regardless where in tree it appears) ~ time invariance in HMM

Probability of a Rule

- Rule r(i): $A \rightarrow \alpha$;
- Let R_A be the set of all rules r(j), which have nonterminal A at the left-hand side;
- Then define probability distribution on R_A:

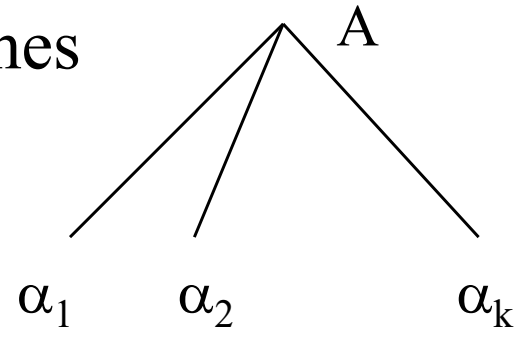
$$\Sigma_{r \in R_A} p(r) = 1, 0 \leq p(r) \leq 1$$

- Another point of view:

$$p(\alpha|A) = p(r), \text{ where } r = A \rightarrow \alpha, \alpha \in (N \cup T)^+$$

Estimating Probability of a Rule

- MLE from a treebank ***following a CFG grammar***
- Let's $r = A \rightarrow \alpha_1 \alpha_2 ... \alpha_k$:
  - $p(r) = c(r) / c(A)$
  - Counting rules (c(r)): how many times



appears in the treebank.

  - Counting nonterminals c(A):

just count'em (in the treebank)

Using Probabilistic CFG

Probability of a Derivation Tree

- Probabilistic CFG:
  - relations among a mother node & daughter nodes
  - in terms of expansion [rewrite,derivation] probability
  - define probability of a derivation (i.e. parse) tree:

$$P(T) = \prod_{i=1..n} p(r(i))$$

r(i) are all rules of the CFG used to generate the sentence W of which T is a parse

- Probability of a string $W = (w_1, w_2, ..., w_n)$?
- Non-trivial, because there may be many trees T_j as a result of parsing W.

Probability of a String

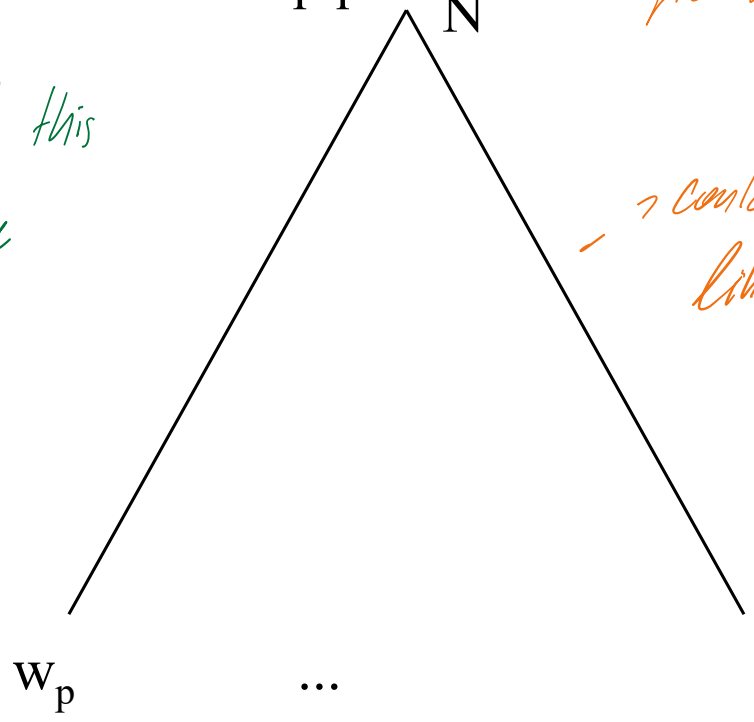
- Input string: W
- Parses: $\{T_j\}_{j=1..n} = \text{Parse}(W)$.

$$P(W) = \Sigma_{j=1..n} P(T_j) \; !$$

- Impossible to use the naive method.

Inside Probability

- $\beta_N(p,q) = P(N \Rightarrow^* w_{pq})$



Formula for Inside Probability

- $\beta_N(p,q) =$ — based on independence

splitting anywhere in between

$$\Sigma_{A,B} \Sigma_{d=p..q-1} P(N \rightarrow A,B)\beta_A(p,d)\beta_B(d+1,q)$$

assuming the grammar G has rules of the form

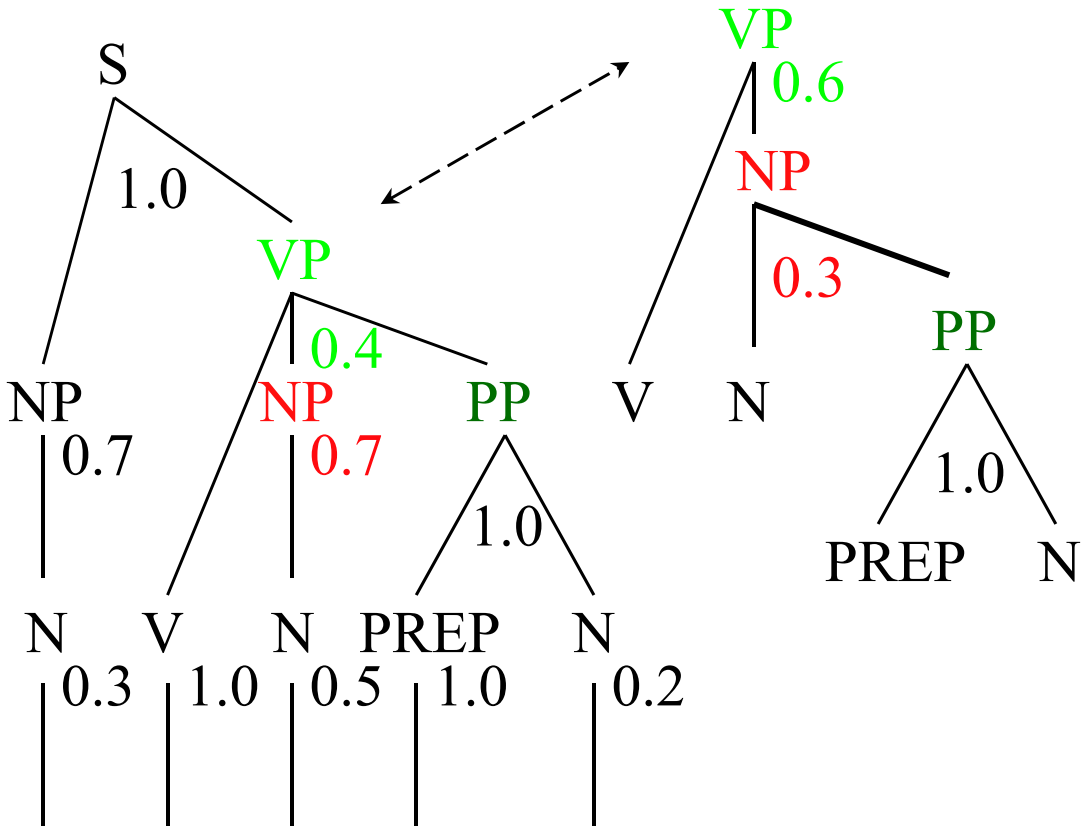
$N \rightarrow \omega$ (terminal string only)

$N \rightarrow A\ B$ (two nonterminals) this is important why we are doing it

only (Chomsky Normal Form).

Example PCFG

- #1 S → NP VP 1.0
- #2 VP → V NP PP 0.4
- #3 VP → V NP 0.6
- #4 NP → N 0.7
- #5 NP → N PP 0.3
- #6 PP → PREP N 1.0
- #7 N → a_dog 0.3
- #8 N → a_cat 0.5
- #9 N → a_telescope 0.2
- #10 V → saw 1.0
- #11 PREP → with 1.0



P(a_dog saw a_cat with a_telescope) =

$$1 \cdot .7 \cdot .4 \cdot .3 \cdot .7 \cdot 1 \cdot .5 \cdot 1 \cdot 1 \cdot .2 + \ldots \cdot .6 \ldots \cdot .3 \ldots = .00588 + .00378 = .00966$$

Tohle může být taky v testu

Computing String Probabilty

- a_dog saw a_cat with a_telescope

  1 2 3 4 5

from\to	1	2	3	4	5
1	NP .21 N .3		S .0441		S .00966
2		V 1	VP .21		VP .046
3			NP .35 N .5		NP .03
4				PREP 1	PP .2
5					N .2

sem se chceme ve finále dostat a mít tam pust. tobo stromu

- Create table n x n (n = length of string). Cells might have more “lines”.
- Initialize on diagonal, using N → α rules.
- Recursively compute along the diagonal towards the upper right corner.

Statistical Parsing

Language Model vs. Parsing Model

- Language model:
  - interested in string probability:

    P(W) = probability definition using a formula such as

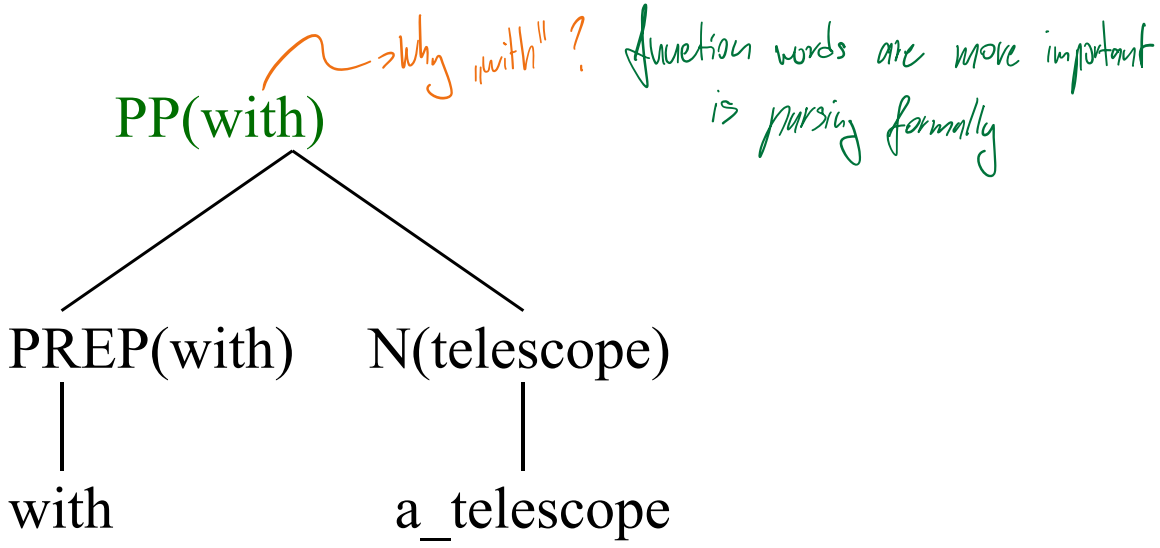
    $= \prod_{i=1..n} p(w_i|w_{i-2},w_{i-1})$ trigram language model

    $= \sum_{s \in S} p(W,s) = \sum_{s \in S} \prod_{r \in s} r$ PCFG; r ~ rule used in parse tree
- Parsing model
  - conditional probability of tree given string:

    P(s|W) = P(W,s) / P(W) = P(s) / P(W) !! P(W,s) = P(s) !!
  - for argmax, just use P(s) (P(W) is constant)

Once again, Lexicalization

- Lexicalized parse tree (~ dependency tree+phrase labels)
- Ex. subtree:



  - Pre-terminals (above leaves): assign the word below
  - Recursive step (step up one level): (a) select node, (b) copy word up.

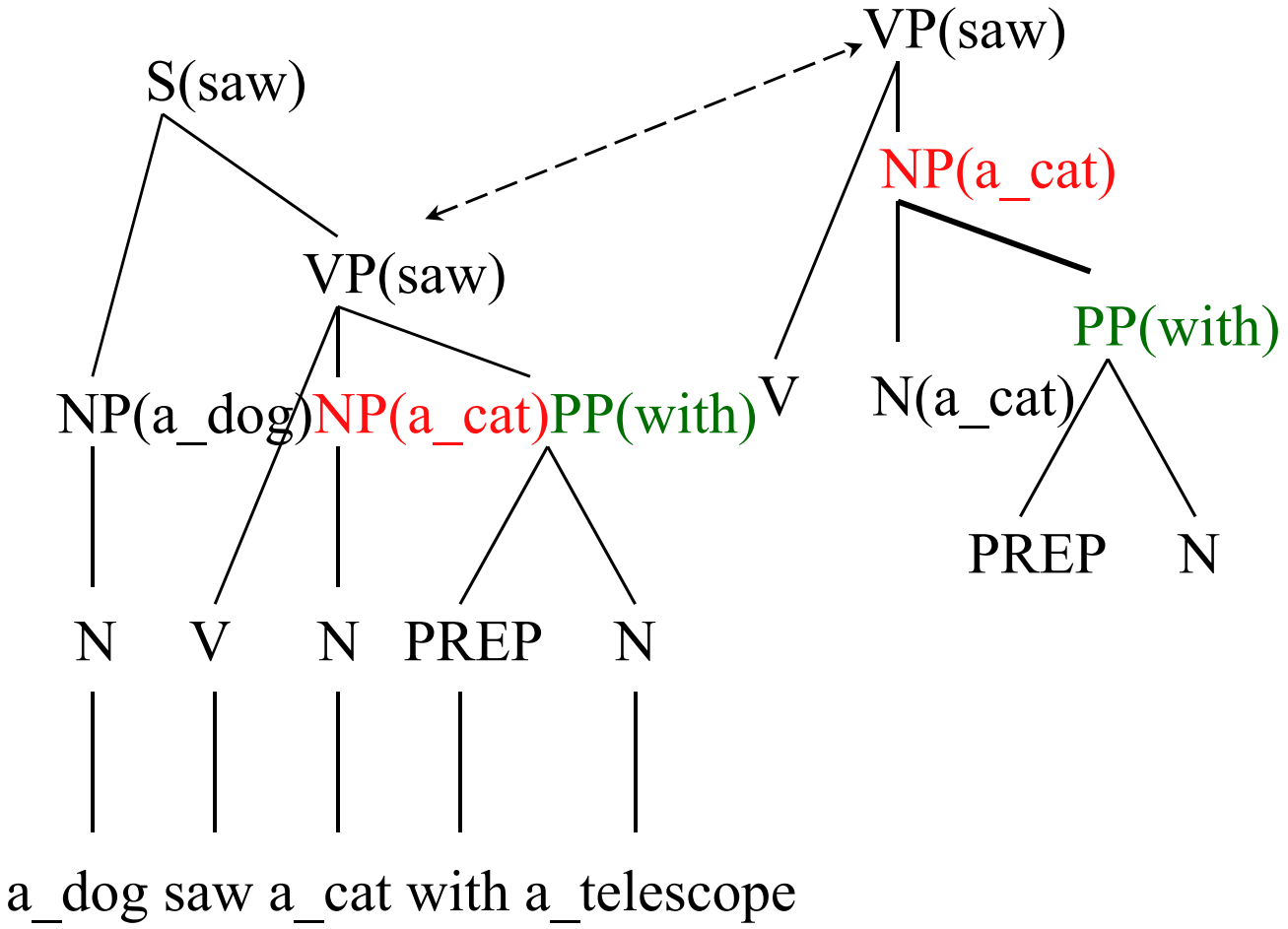
– we generate many more symbols

+ we can assign different probs to S(saw) than to S(catch)

Lexicalized Tree Example

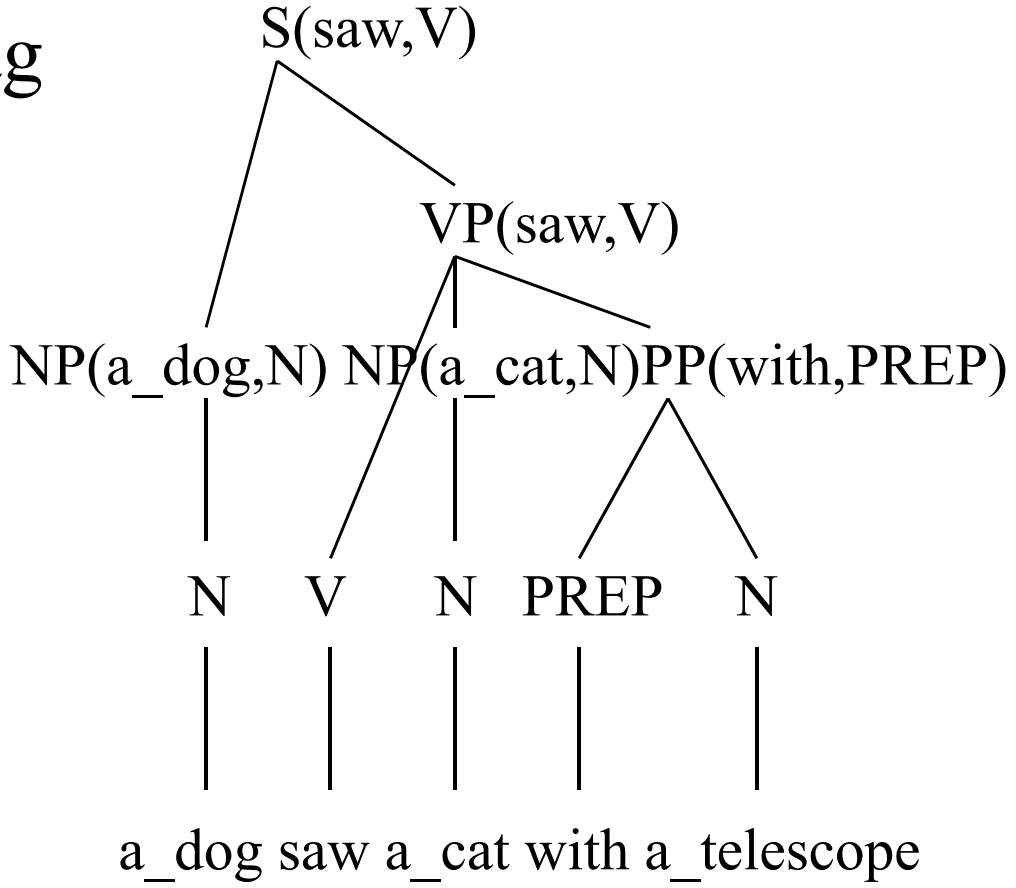
↳ can help with finding the correct parse with more "language-driven" states

- #1 S → NP VP
- #2 VP → V NP PP
- #3 VP → V NP
- #4 NP → N
- #5 NP → N PP
- #6 PP → PREP N
- #7 N → a_dog
- #8 N → a_cat
- #9 N → a_telescope
- #10 V → saw
- #11 PREP → with



Using POS Tags

- Head ~ word,tag



Conditioning

- Original PCFG: $P(\alpha B \gamma D \varepsilon ... | A)$
  - No "lexical" units (words)
- Introducing words:

$$P(\alpha\ B(head_B)\ \gamma\ D(head_D)\ \varepsilon\ ...\ |A(head_A))$$

where $head_A$ is one of the heads on the left

E.g. rule $VP(saw) \rightarrow V(saw)\ NP(a\_cat)$:

$$P(V(saw)\ NP(a\_cat) \mid VP(saw))$$

Without the lexicalization, we generalized the structure perfectly, but have hard times predicting words.
With -lt, we are very focused, but the structure is not generalizing at all.

2 = sparse data problem

Independence Assumptions

- Too many rules
- Decompose:

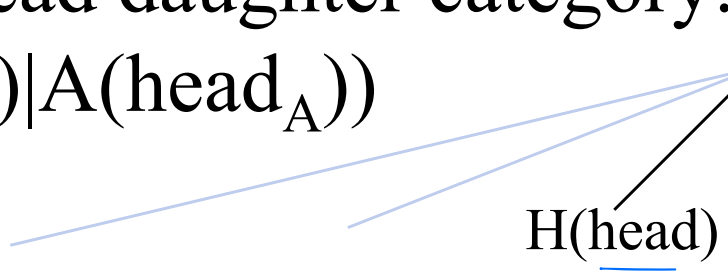
  $P(\alpha\ B(head_B)\ \gamma\ D(head_D)\ \varepsilon\ \ldots\ |A(head_A)) =$
- In general (total independence):

  $P(\alpha|A(head_A)) \times P(B(head_B)|A(head_A)) \times$

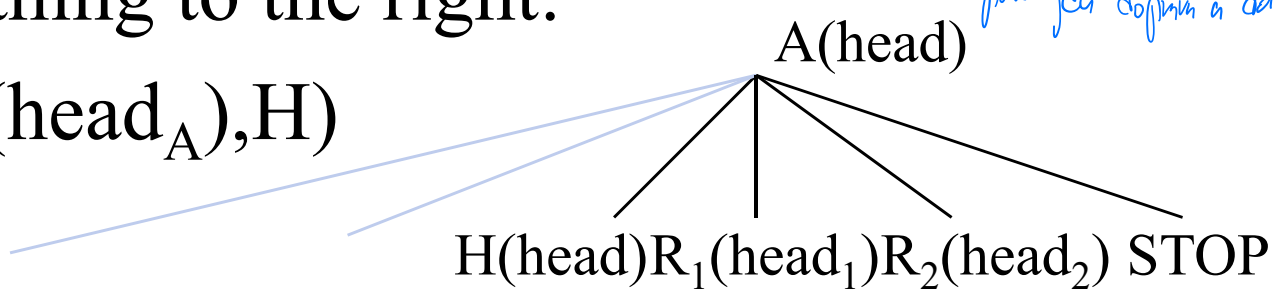
  $\ldots \times P(\varepsilon|A(head_A))$
- Too much independent: need a compromise.

The Decomposition

- Order does not matter; let's use intuition ("linguistics"):
- Select the head daughter category:
  $P_H(H(head_A)|A(head_A))$



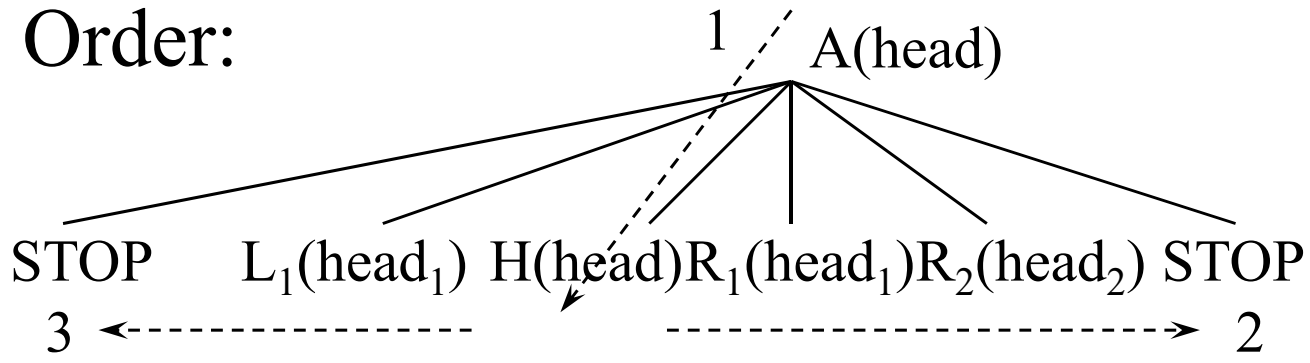
- Select everything to the right:
  $P_R(R_i(r_i) \mid A(head_A),H)$



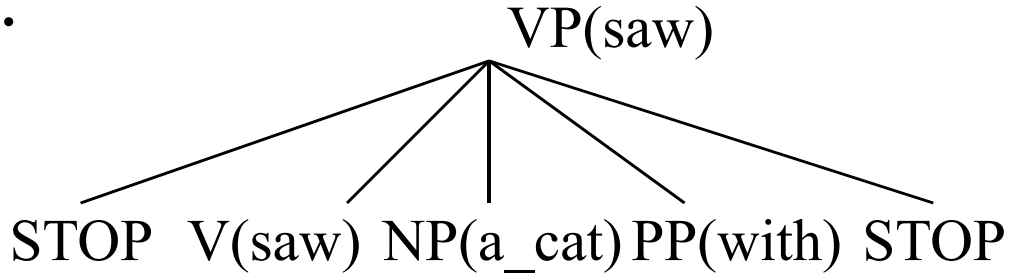
- Also, choose when to finish: $R_{m+1}(r_{m+1}) = STOP$
- Similarly, for the left direction: $P_L(L_i(l_i) \mid A(head_A),H)$

Example Decomposition

- Order:



- Example:

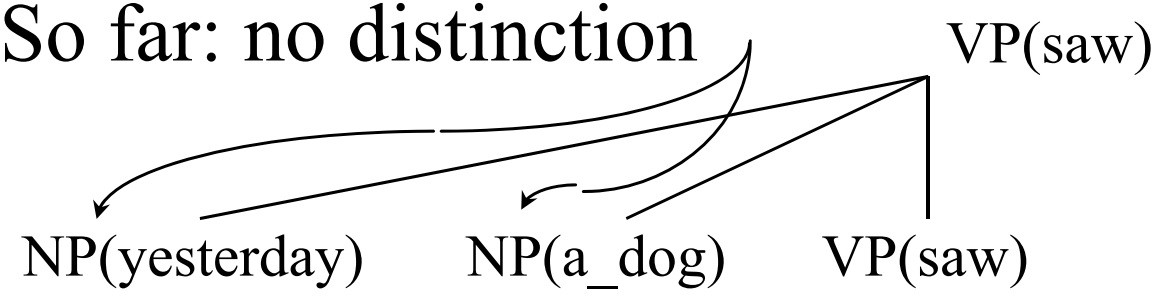


More Conditioning: Distance

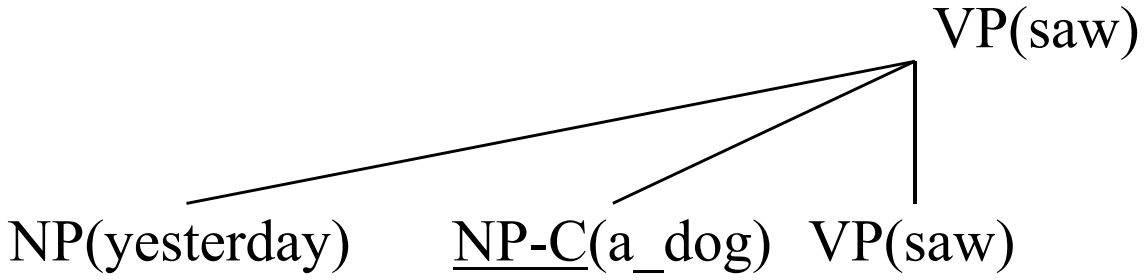
- Motivation:
  - close words tend to be dependents (or phrases) more likely
  - ex.: walking on a sidewalk on a sunny day without looking on...
- Words: too detailed distribution, though:
  - use more sophisticated (yet more robust) distance measure $d_{r/l}$:
    - distinguish 0 and non-zero distance (2)
    - distinguish if verb is in-between the head and the constituent in question (2)
    - distinguish if there are commas in-between: 0, 1, 2, >2 commas (4).
    - ...total: 16 possibilities added to the condition: $P_R(R_i(r_i) \mid A(head_A),H,d_r)$
    - same to the left: $P_L(L_i(l_i) \mid A(head_A),H,d_l)$

More Conditioning: Complement/Adjunct

- So far: no distinction



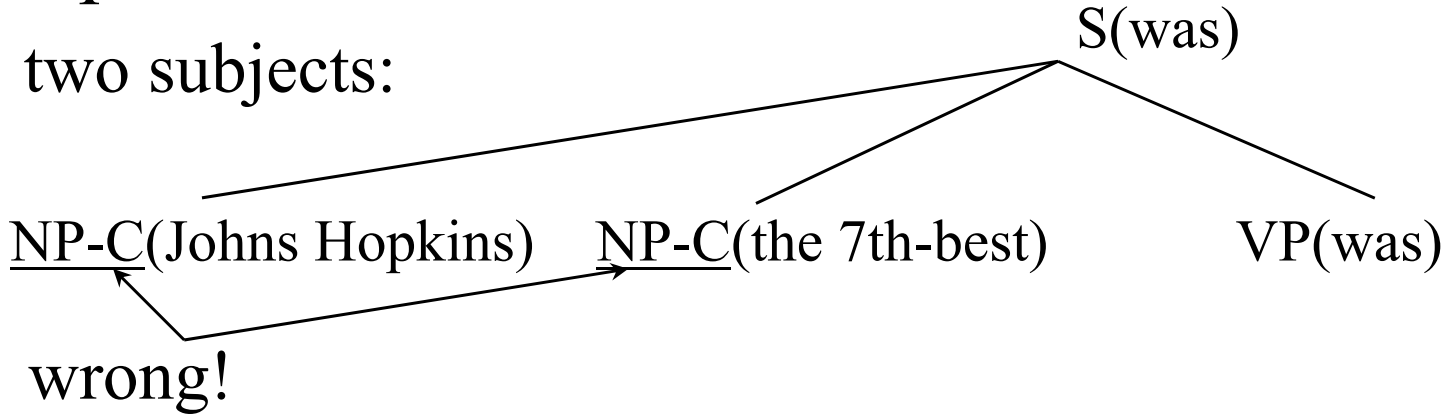
- ...but: time NP ≠ subject NP
- also, Subject NP cannot repeat... useful <u>during</u> parsing



[Must be added in training data]

More Conditioning: Subcategorization

- The problem still not solved:
  - two subjects:



- Need: relation among complements.
  - [linguistic observation: adjuncts can repeat freely.]
- Introduce:
  - Left & Right Subcategorization Frames (multisets)

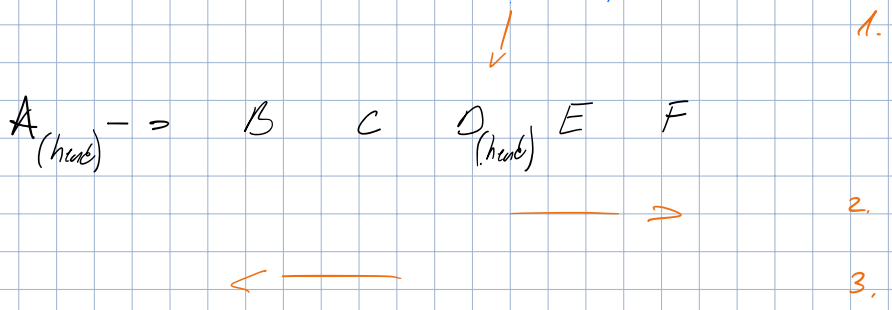
Inserting Subcategorization

- Use head probability as before:

  $P_H(H(head_A)|A(head_A))$
- Then, add left & right subcat frame:

  $P_{lc}(LC|\ A(head_A),H),\ P_{rc}(RC|\ A(head_A),H)$
  - LC, RC: list (multiset) of phrase labels (not words)
- Add them to context condition:

  (left) $P_L(L_i(l_i)\ |\ A(head_A),H,d_l,LC)$ [right: similar]
- LC/RC: "dynamic": remove labels when generated
  - P(STOP|.....,LC) = 0 if LC non-empty



computed serially with Bayes rule

2.: $p(D_{(head)} \mid A_{(head)}) \cdot p(E_{(...)} \mid D_{(head)}) \cdot p(F_{(...)} \mid E_{(...)}) \cdot p(\text{STOP} \mid F_{(...)})$

3.: obdobně

Pak bych to spojil a dostal bych celkovou pravděpodobnost toho pravidla.

Smoothing

- Adding conditions... ~ adding parameters
- Sparse data problem as usual (head ~ <word,tag>!)
- Smooth (step-wise):
  - $P_{smooth\text{-}H}(H(head_A)|A(head_A)) =$
    $= \lambda_1 P_H(H(head_A)|A(head_A)) + (1\text{-}\lambda_1)P_{smooth\text{-}H}(H(head_A)|A(tag_A))$
  - $P_{smooth\text{-}H}(H(head_A)|A(tag_A)) =$
    $= \lambda_2 P_H(H(head_A)|A(tag_A)) + (1\text{-}\lambda_2)P_H(H(head_A)|A)$
- Similarly, for P_R and P_L.

The Parsing Algorithm for a Lexicalized PCFG

- Bottom-up Chart parsing
  - Elements of a chart: a pair
    - <(from-position,to-position,label,head,distance), probability>
      span score
  - Total probability = multiplying elementary probabilities
    $\Rightarrow$ enables dynamic programming:
    - discard chart element with the same span but lower score.
- "Score" computation:
  - joining chart elements: [for 2]: $<e_1, p_1>$, $<e_2, p_2>$, $<e_n,p_n>$:

$$P(e_{new}) = p_1 \times p_2 \times ... \times p_n \times P_H(...) \times \prod P_R(...) \times \prod P_L(...);$$

Results (PCFG)

- English, WSJ, Penn Treebank, 40k sentences

	< 40Words	< 100 Words
– Labeled Recall:	88.1%	87.5%
– Labeled Precision:	88.6%	88.1%
– Crossing Brackets (avg):	0.91	1.07
– Sentences With 0 CBs:	66.4%	63.9%

- Czech, Prague Dependency Treebank, 13k sentences:
  - Dependency Accuracy overall: 80.0% (MST'05: 85%)
    (~ unlabelled precision/recall)

Dependency Parsing

- Graph-based
  - Maximum Spanning Tree method (see the following slides)
    - McDonald et al., 2005, 2006
- Transition-based
  - See (non-deterministic) Shift-reduce parsing + probablistic model
    - Nivre et al., MALT Parser since 2003

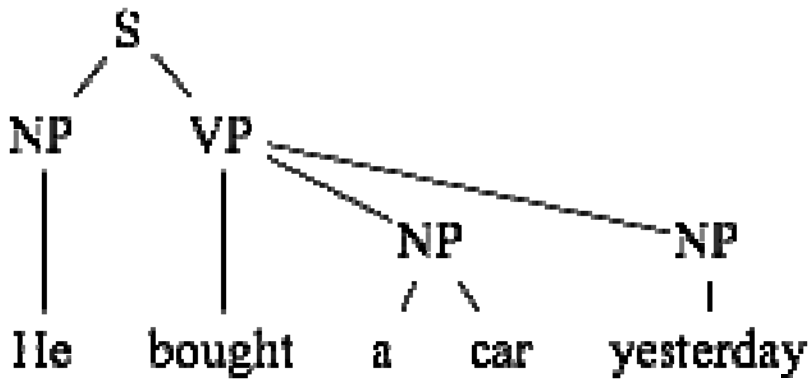
Some slides in the next section are from J. Choi, Emory University

Dependency Structure

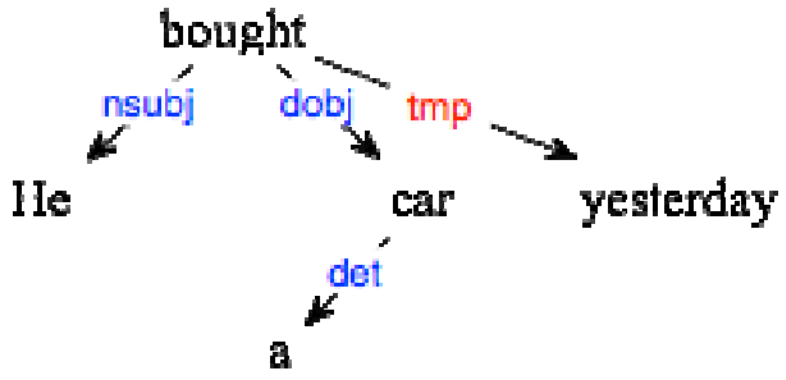
- What is a dependency? There is no reason between syntactic or semantic for the only.
  - A syntactic or semantic (or other) relation between a pair of tokens.

- Constituent vs. Dependency Structure

"phrase-based" tree



dependency tree

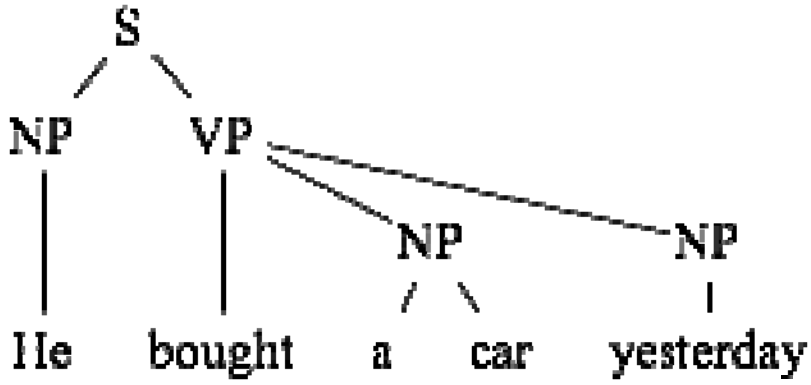


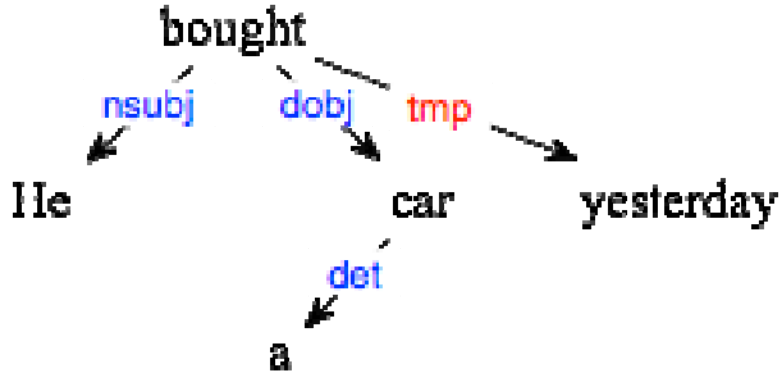
↳ number of nodes = number of tokens

Dependency Structure

- Constituent structure
  - Starts with the bottom level constituents (tokens).
  - Group smaller constituents into bigger constituents (phrases).
- Dependency structure
  - Starts with vertices (tokens).
  - Build a graph by adding edges between vertices (arcs).

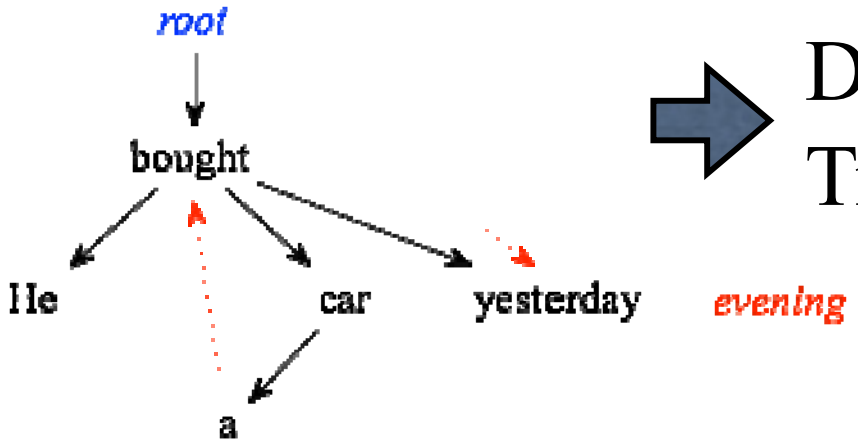
we are trying to correctly connect tokens





Dependency Graph

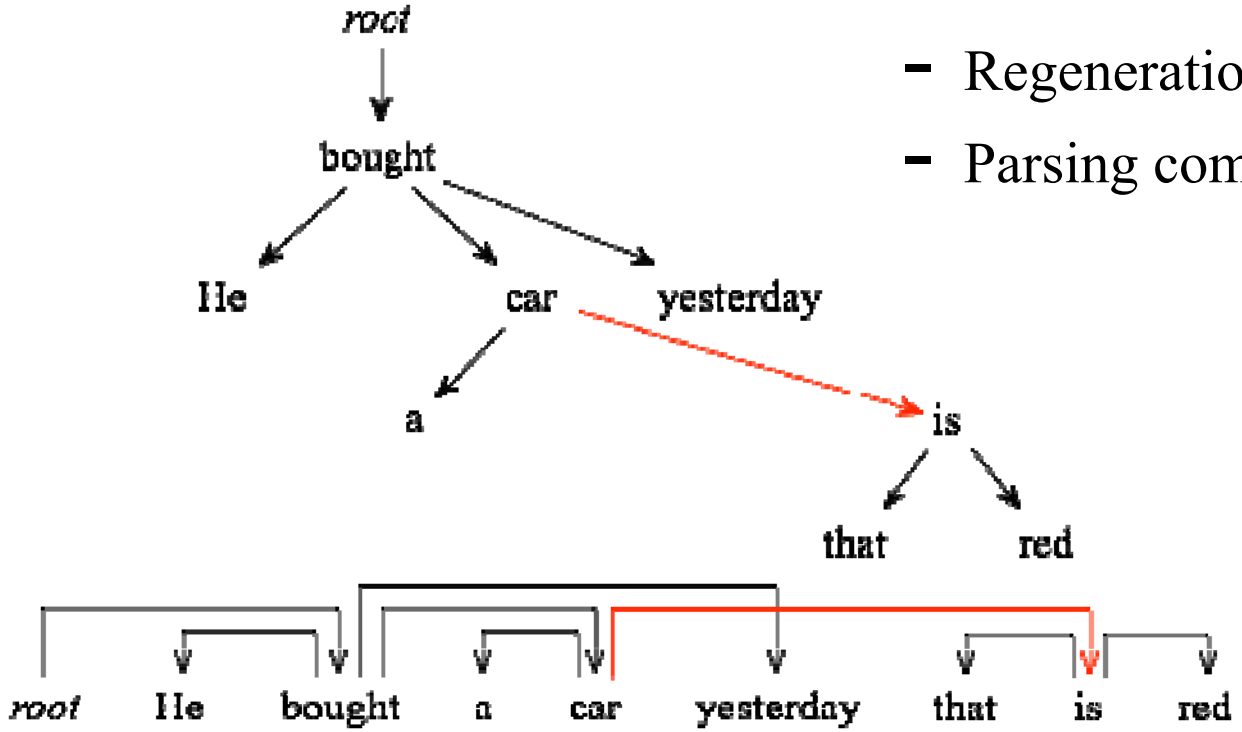
- For a sentence $s = w_1 \ldots w_n$, a dependency graph $G_s = (V_s, A_s)$
  - $V_s = \{w_0 = root, w_1, \ldots, w_n\}$.
  - $A_s = \{(w_i, r, w_j) : i \neq j, w_i \in V_s, w_j \in V_s - \{w_0\}, r \in R_s\}$.
    - R_s = a subset of dependency relations in s.
- A well-formed dependency graph
  - Root
  - Single head
  - Connected
  - Acyclic



Dependency Tree

Dependency Graph

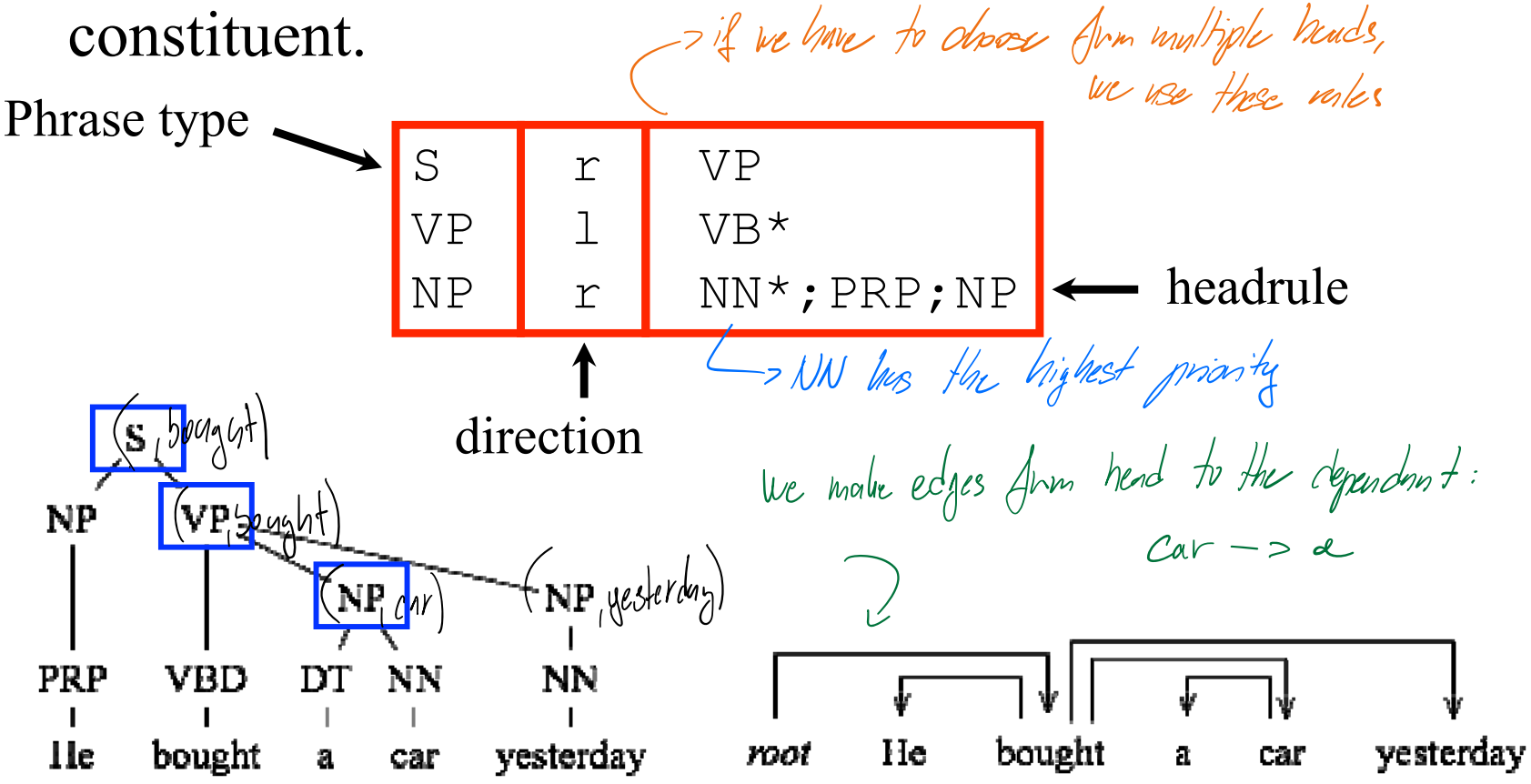
- Projectivity
  - A projective dependency tree has no crossing arc when all vertices are lined up in linear order and arcs are drawn above.
  - e.g., *He bought a car yesterday that is red*.

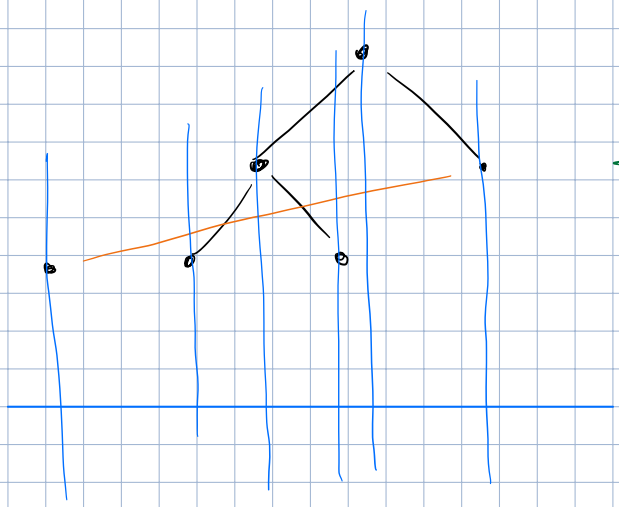


- Regeneration of the original sentence.
- Parsing complexity: O(n) vs. O(n^2).

Constituent To Dependency

- Head-finding rules (i.e., head-percolation rules, headrules)
  - Constituent trees can be converted into dependency trees.
  - Apply headrules recursively to find the head of each constituent.





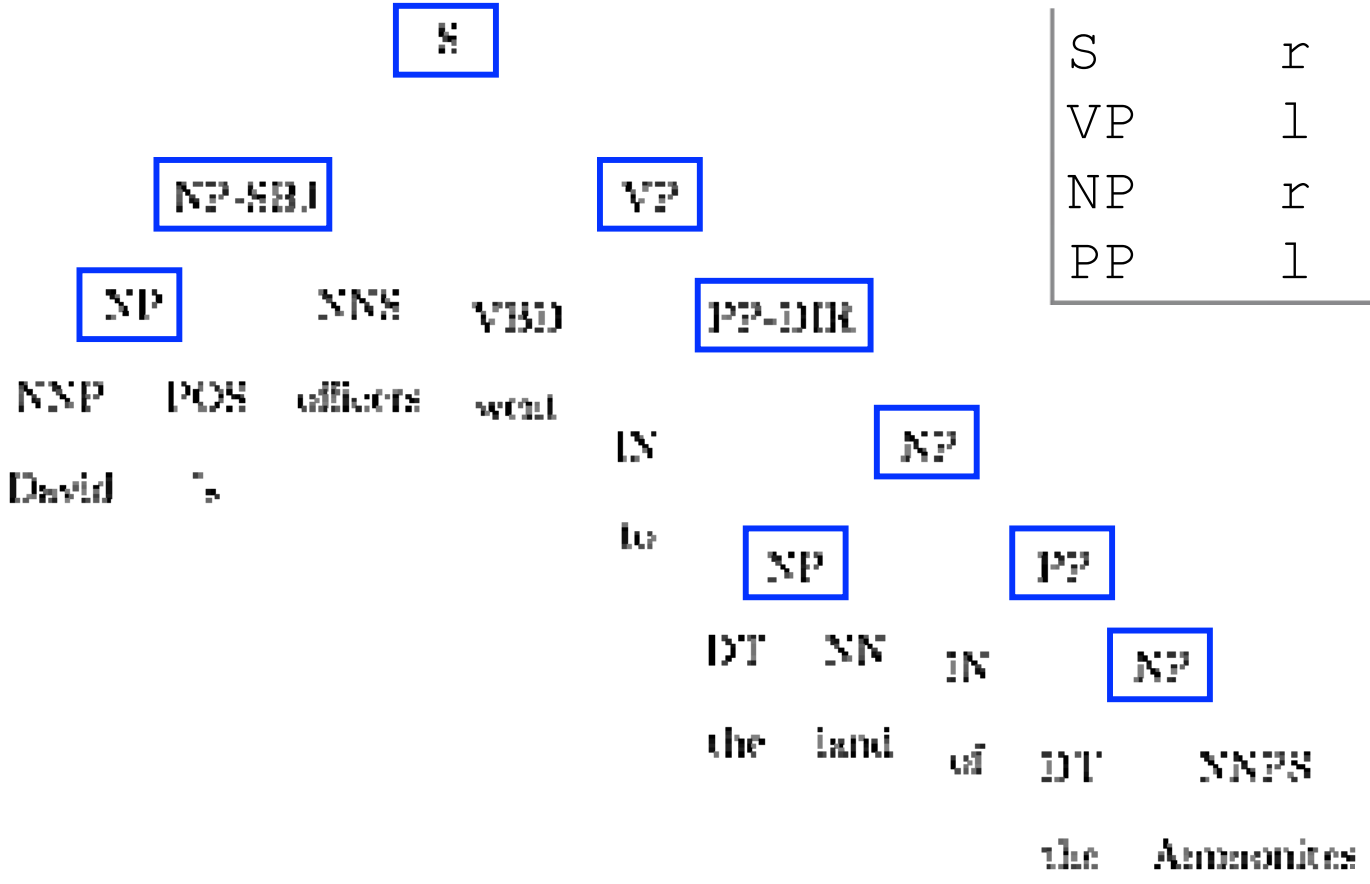
-> not always must we draw tree and look for crossings

-> it is better to look how many projections the ——— is crossing. If any, it is not projective tree.

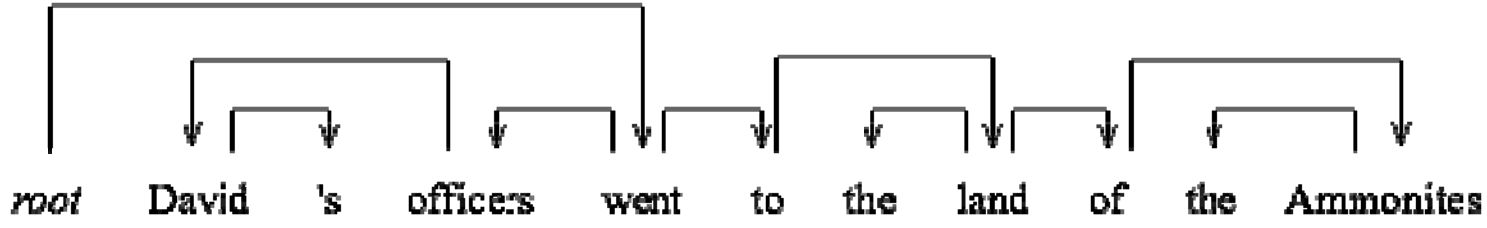
Google Search was actually using dependency parsing for finding common parts between query and responses

Graph-based parsing can never go below $O(n^2)$, using Bomchai's algorithm

Constituent To Dependency



S	r	VP
VP	l	VB*
NP	r	NN*;PRP;NP
PP	l	IN



Dependency Parsing

- Why dependency parsing?
  - Provides useful information for many NLP tasks
    : information extraction, machine translation, question-answering, sentiment analysis, etc.
  - Faster than most parsing approaches (esp. Transition-based)
    : about 1 milliseconds per sentence.
  - More language independent
    : CoNLL shared tasks 2006, 2007, and 2009
    : Universal Dep tasks 2017, 2018; MRP tasks 2019, 2020
- Approaches
  - Transition-based vs. graph-based.
    - Important feature: whether handling projective vs. non-projective.

Dependency Parsing

- Transition-based parsing
  - Transition: an operation searching for a dependency relation between each pair of tokens (e.g., Shift, Reduce).
  - Greedy search for local optima (locally optimized transitions)
    - does better for local dependencies.
  - Projective: O(n), non-projective: O(n^2).
- Graph-based parsing – starting with complete graph and sequentially remove edges
  - Build a complete graph with directed/weighted edges and find a tree with the highest score (sum of all weighted edges).
  - Exhaustive search that finds for the global optimum (maximum spanning tree) → do better for long-distance dependencies.
  - Projective: O(n^3), non-projective: O(n^2).

Transition-based Parsing

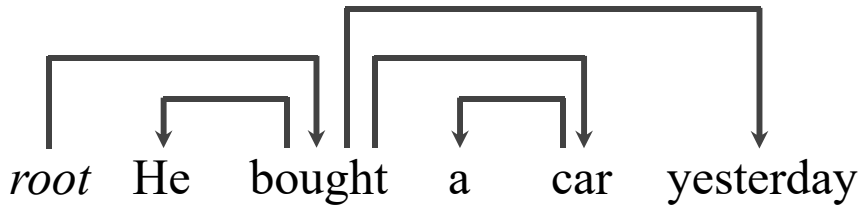
- Projective parsing: ~O(n) Shift-reduce parsing
  - Bottom-up: Yamada & Matsumoto, 2003.
  - Top-down, bottom-up: Nivre, 2003.
  - Beam search: Zhang & Clark, 2008.
  - Dynamic programming: Huang & Sagae, 2010.
  - Selectional branching: Choi & McCallum, 2013.
- Non-projective parsing: O(n^2)
  - Exhaustive search: Covington, 2001.
  - Reordering tokens: Nivre, 2009 (linear-time in practice).
  - Selective search: Choi & Palmer, 2011 (linear-time in practice)
  - Search-based "oracle": Straka et al., 2015, TLT, Warsaw

Transition-based Parsing

- Nivre's arc-eager algorithm
  - Projective parsing algorithm with a worst-case complexity of O(*n*).
  - S = stack, I = list of input tokens, A = set of arcs.

Initialization	$(\text{nil}, W, \emptyset)$	
Termination	(S, nil, A)	
Left-Reduce	$(w_i\|S, I, A) \Rightarrow (S, I, A \cup \{(w_j, w_i)\})$	$\exists w_k(w_k, w_i) \in A$
Right-Reduce	$(w_i\|S, I, A) \Rightarrow (w_i\|S, I, A \cup \{(w_i, w_j)\})$	$\exists w_k(w_k, w_i) \in A$
Shift	$(S, w_i\|I, A) \Rightarrow (w_i\|S, I, A)$	

Transition-based Parsing



We have transition probs. that tell us what to choose from actions.

root

S

He
bought
a
car
yesterday

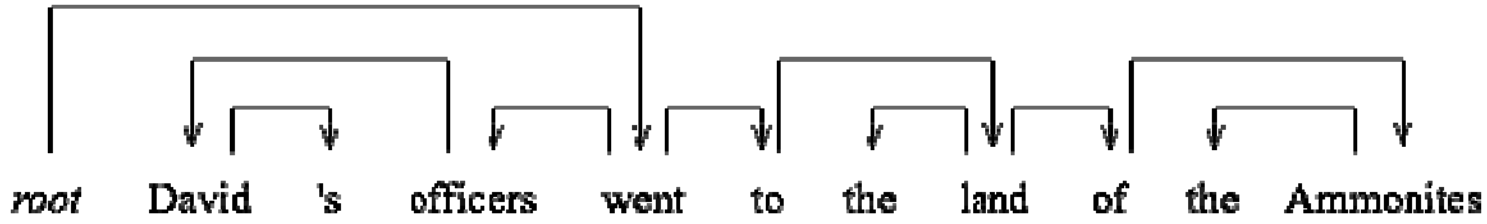
I

bought → yes..
bought → car
a ← car
root → bought
He ← bought

A

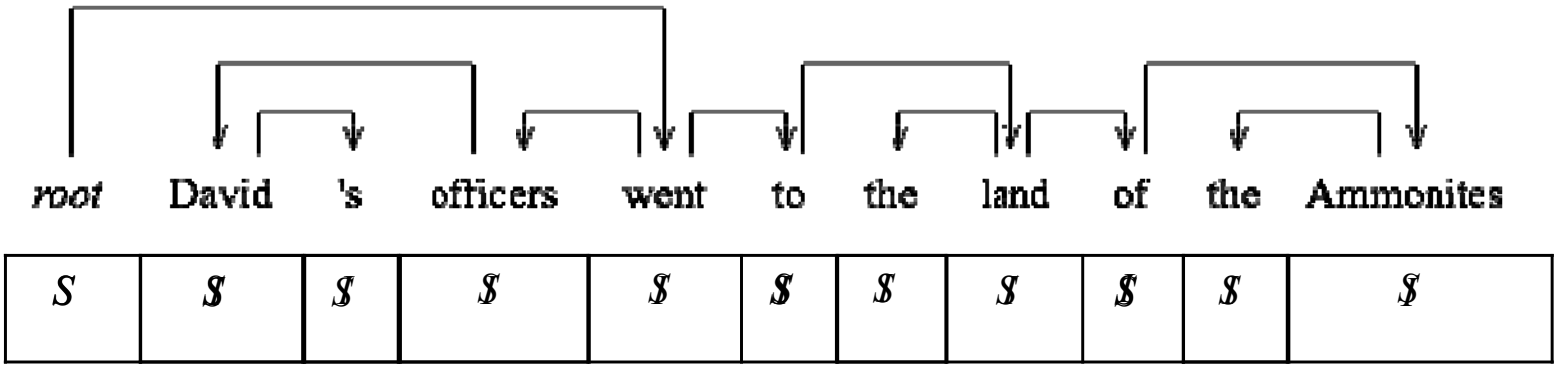
- Shift : ‘He’
- LeftArc : ‘He’ ← ‘bought’
- RightArc: *root* → ‘bought’
- Shift : ‘a’
- LeftArc : ‘a’ ← ‘car’
- RightArc: ‘bought’ → ‘car’
- Reduce: ‘car’
- RightArc: ‘bought’ → ‘yesterday’

Nivre's Arc-eager Algorithm



Initialization	$(\text{nil}, W, \emptyset)$	
Termination	(S, nil, A)	
Left-Reduce	$(w_i\|S, I, A) \Rightarrow (w_i\|S, I, A \cup \{(w_j, w_i)\})$	$\exists w_k(w_k, w_i) \in A$
Right-Reduce	$(w_i\|S, I, A) \Rightarrow (w_i\|S, I, A \cup \{(w_i, w_j)\})$	$\exists w_k(w_k, w_i) \in A$
Shift	$(S, w_i\|I, A) \Rightarrow (w_i\|S, I, A)$	

Nivre's Arc-eager Algorithm



- Initialize
- Shift: 'David'
- Right-Arc: David → 's
- Reduce: 's
- Left-Arc: David ← 'officers'
- Shift: officers
- Left-Arc: officers ← went
- Right-Arc: *root* → went
- Right-Arc: went → to
- Shift: the
- Left-Arc: the ← land
- Right-Arc: to → land
- Right-Arc: land → of
- Shift: 'the'
- Left-Arc: the ← Ammonites
- Right-Arc: of → Ammonites
- Terminate

Graph-based Parsing

- using MERT on training data to get scores, with several hand-made feature templates

- Inspired by maximum spanning tree algorithms.
- Projective parsing: O(n^3)
  - CKY parsing: Eisner, 2000.
- Non-projective parsing
  - Chu-Liu-Edmonds' algorithm: McDonald et al, 2005 (O(n^2)).
  - 2nd-order parsing: McDonald & Pereira, 2006 (O(n^3)).
  - 3rd-order parsing: Koo & Collins, 2010 (O(n^4)).
- Advance parsing
  - Vine pruning: Rush and Petrov, 2012.

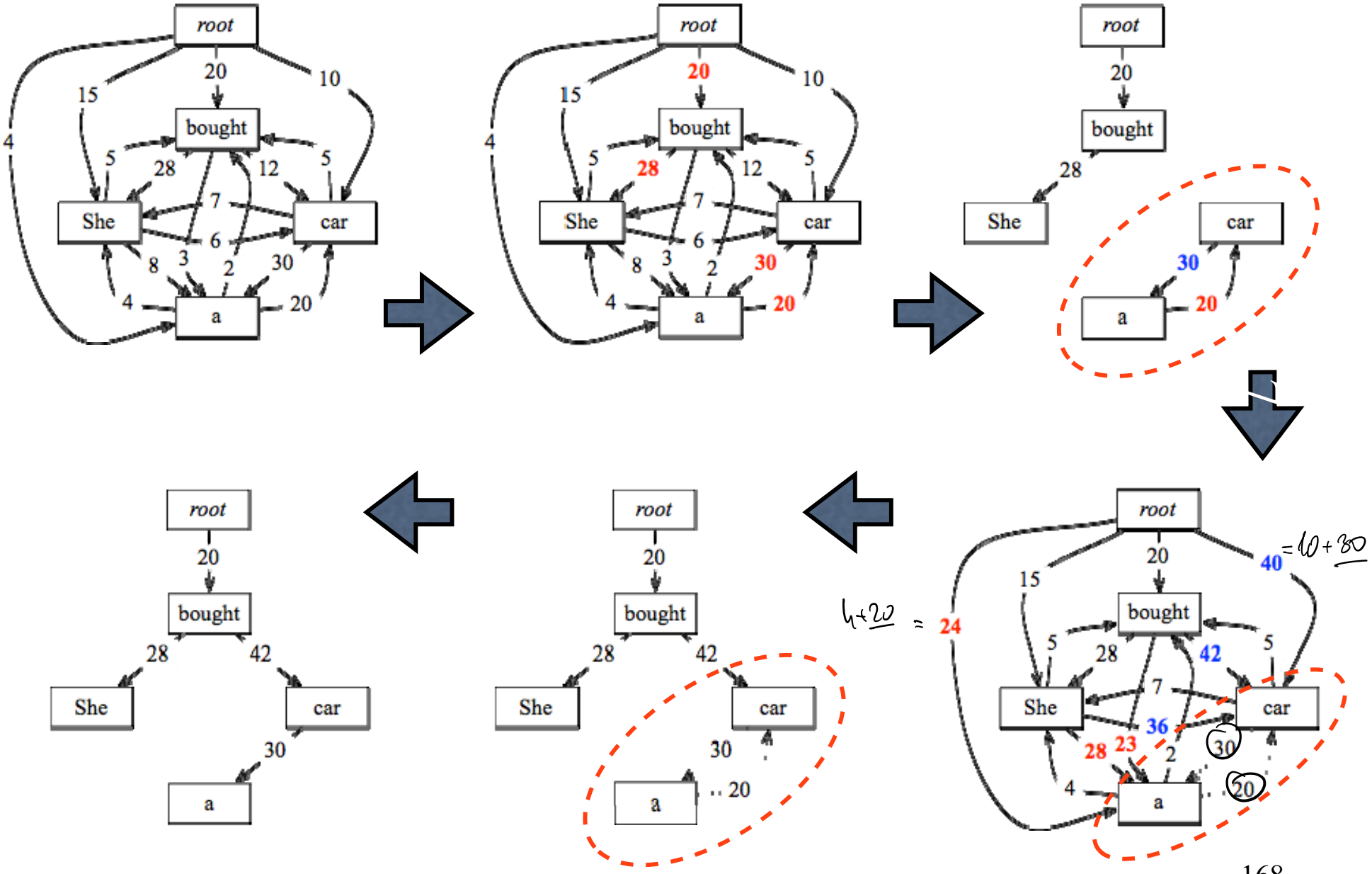
you never know when the word can be connected and therefore need to assume all combination

Chu-Liu-Edmonds' Algorithm

To s cyklem je jako když Borůvka spojuje komponenty souvislosti

- Based on a maximum spanning tree algorithm
  1. Build a complete graph with directed and weighted edges.
  2. Keep only incoming edges with the maximum scores.
  3. If there is no cycle, go to #5.
  4. If there is a cycle, pretend vertices in the cycle as one vertex and update scores for all incoming edges to the cycle; goto #2.
  5. Break all cycles by removing inappropriate edges in the cycle. —> if more options, remove the least prob. weight edge

Chu-Liu-Edmonds' Algorithm



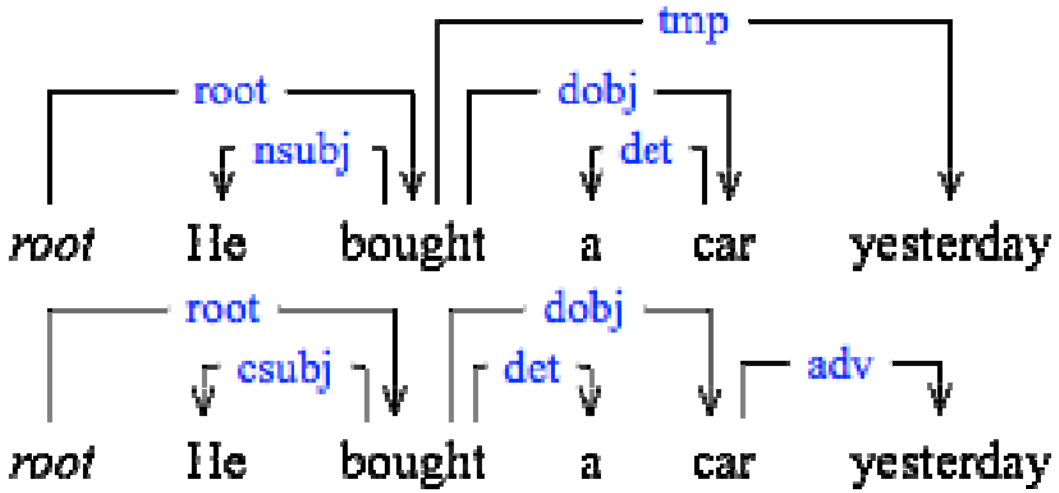
Feature Extraction

- Part-of-speech tagging
  - Word-forms, POS tags, ambiguity classes.
  - Given w_i, extract features from $w_{i \pm n}$ (usually $n \in [0, 3]$).
- Dependency parsing
  - Word forms, lemmas, POS tags, dependency labels.
  - Given w_i and w_j, extract features from
    - $w_{i \pm n}$, $w_{j \pm n}$.
    - The ancestors of w_i and w_j.
    - The dependents of w_i and w_j.
    - The siblings of w_i and w_j.

Evaluation

- Assume each node has exactly one head except for the *root*.
- For each tree, count
  - how many nodes found correct heads
    : Unlabeled attachment score (UAS).
  - how many nodes found correct labels
    : Label accuracy (LS).
  - how many nodes found both correct heads and labels
    : Labeled attachment score (LAS).

Evaluation

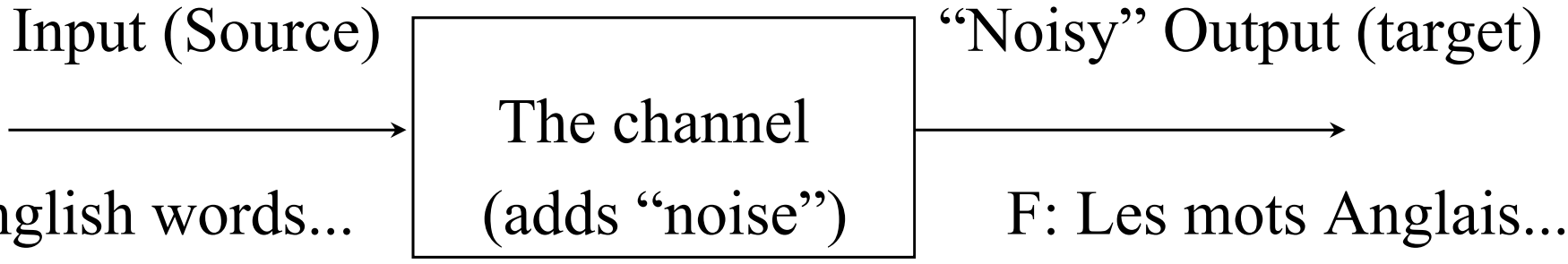


- Unlabeled attachment score
  - Mismatches: bought → a, car → yesterday (3/5 = 60%)
- Label accuracy
  - Mismatches: He - csubj, yesterday - adv (3/5 = 60%)
- Labeled attachment score
  - He - csubj, bought → a, car → yesterday - adv (2/5 = 40%)

Statistical Machine Translation

The Main Idea

- Treat translation as a noisy channel problem:



- The Model: $P(E|F) = P(F|E)\,P(E) / P(F)$
- Interested in rediscovering E given F:

  After the usual simplification (P(F) fixed):

$$\text{argmax}_E\, P(E|F) = \text{argmax}_E\, P(F|E)\,P(E)$$!

We treat the foreign language as a code that is decoded to english.
Theoretically having very laargre datasets, it would train reasonably well.

The Necessities

- Language Model (LM)

  P(E)
- Translation Model (TM): Target given source

  P(F|E)
- Search procedure
  - Given F, find best E using the LM and TM distributions.
- Usual problem: sparse data
  - We cannot create a "sentence dictionary" E ↔ F
  - Typically, we do not see a sentence even twice!

The Language Model

- Any LM will do:
  - 3-gram LM
  - 3-gram class-based LM (cf. HW #2!)
  - decision tree LM with hierarchical classes
- Does not necessarily operates on word forms:
  - cf. later the “analysis” and “generation” procedures
  - for simplicity, imagine now it ***does*** operate on word forms

The Translation Models

- Do not care about correct strings of English words (that's the task of the LM)
- Therefore, we can make more independence assumptions:
  - for start, use the "tagging" approach:
    - 1 English word ("tag") ~ 1 French word ("word")
  - not realistic: rarely even the number of words is the same in both sentences (let alone there is 1:1 correspondence!)
- $\Rightarrow$ use "Alignment".

The Alignment

0 1 2 3 4 5 6

- e_0 And the program has been implemented

- f_0 Le programme a été mis en application

0 1 2 3 4 5 6 7

- Linear notation:
  - f_0(1) Le(2) programme(3) a(4) été(5) mis(6) en(6) application(6)
  - e_0 And(0) the(1) program(2) has(3) been(4) implemented(5,6,7)

Alignment Mapping

- In general:
  - |F| = m, |E| = l (length of sent.):
    - lm connections (each French word to any English word),
    - 2^{lm} different alignments for any pair (E,F) (any subset)
- In practice:
  - From English to French
    - each English word 1-n connections (n - empirical max.)
    - each French word ***exactly*** 1 connection
  - therefore, "only" $(l+1)^m$ alignments ($<< 2^{lm}$)
    - $a_j = i$ (link from j-th French word goes to i-th English word)

Elements of Translation Model(s)

- Basic distribution:
- P(F,A,E) - the joint distribution of the English sentence, the Alignment, and the French sentence (length m)
- Interested also in marginal distributions:

$$P(F,E) = \Sigma_A P(F,A,E)$$

$$P(F|E) = P(F,E) / P(E) = \Sigma_A P(F,A,E) / \Sigma_{A,F} P(F,A,E) = \Sigma_A P(F,A|E)$$

- Useful decomposition [one of possible decompositions]:

$$P(F,A|E) = P(m \mid E) \prod_{j=1..m} P(a_j|a_1^{j-1},f_1^{j-1},m,E)\, P(f_j|a_1^{j},f_1^{j-1},m,E)$$

Decomposition

- Decomposition formula again:

$$P(F,A|E) = P(m \mid E) \prod_{j=1..m} P(a_j|a_1^{j-1},f_1^{j-1},m,E)\, P(f_j|a_1^{j},f_1^{j-1},m,E)$$

m - length of French sentence

a_j - the alignment (single connection) going from j-th French w.

f_j - the j-th French word from F

a_1^{j-1} - sequence of alignments a_i up to the word preceding f_j

a_1^{j} - sequence of alignments a_i up to and including the word f_j

f_1^{j-1} - sequence of French words up to the word preceding f_j

Decomposition and the Generative Model

- ...and again:

$$P(F,A|E) = P(m \mid E) \prod_{j=1..m} P(a_j|a_1^{j-1},f_1^{j-1},m,E)\, P(f_j|a_1^{j},f_1^{j-1},m,E)$$

- Generate:
  - first, the length of the French given the English words E;
  - then, the link from the first position in F (not knowing the actual word yet) $\Rightarrow$ <u>now</u> we know the English word
  - then, given the link (and thus the English word), generate the French word at the current position
  - then, move to the next position in F until m position filled.

Approximations

- Still too many parameters
  - similar situation as in n-gram model with “unlimited” n
  - impossible to estimate reliably.
- Use 5 models, from the simplest to the most complex (i.e. from heavy independence assumptions to light)
- Parameter estimation:

  Estimate parameters of Model 1; use as an initial estimate for estimating Model 2 parameters; etc.

Model 1

Then could be, however, too many alignments for each word without simplification

- Approximations:
  - French length P(m | E) is constant (small ε)
  - Alignment link distribution $P(a_j|a_1^{j-1},f_1^{j-1},m,E)$ depends on English length l only (= 1/(l+1))
  - French word distribution depends only on the English and French word connected with link a_j.
- ⇒ Model 1 distribution:

$$P(F,A|E) = \varepsilon / (l+1)^m \prod_{j=1..m} p(f_j|e_{a_j})$$

trying to align the foreign word to the original for each possible alignment

However no-one gives us alignment, therefore we need to model (statistically) the alignments for each word as well

Models 2-5

- Model 2
  - adds more detail into $P(a_j|...)$: more "vertical" links preferred
- Model 3
  - adds "fertility" (number of links for a given English word is explicitly modeled: $P(n|e_i)$
  - "distortion" replaces alignment probabilities from Model 2
- Model 4
  - the notion of "distortion" extended to chunks of words
- Model 5 is Model 4, but not deficient (does not waste probability to non-strings)

It is trying to find the „original" sentence for the encoded („foreign") language.

The Search Procedure

- "Decoder":
  - given "output" (French), discover "input" (English)
- Translation model goes in the opposite direction:

  $p(f|e) = ...$
- Naive methods do not work.
- Possible solution (roughly):
  - generate English words one-by-one, keep only n-best (variable n) list; also, account for different lengths of the English sentence candidates!

Same as looking into dictionary: first I come up with lemma for my word, I secondly find translation for the lemma and finally I correctly use the target lemma.

Analysis - Translation - Generation (A-T-G)

Idea is that the translation is the hardest, therefore we try to simplify by translating only „base forms"

- Word forms: too sparse
- Use four basic analysis, generation steps:
  - tagging
  - lemmatization
  - word-sense disambiguation
  - noun-phrase "chunks" (non-compositional translations)
- Translation proper:
  - use chunks as "words"

Training vs. Test with A-T-G

- Training:
  - analyze both languages using all four analysis steps
  - train TM(s) on the result (i.e. on chunks, tags, etc.)
  - train LM on analyzed source (English)
- Runtime/Test:
  - analyze given language sentence (French) using identical tools as in training
  - translate using the trained Translation/Language model(s)
  - generate source (English), reversing the analysis process

Analysis: Tagging and Morphology

- Replace word forms by morphologically processed text:
  - lemmas
  - tags
    - original approach: mix them into the text, call them "words"
    - e.g. She bought two books. ⇒ she buy VBP two book NNS.
- Tagging: yes
  - but reversed order:
    - tag first, then lemmatize [NB: does not work for inflective languages]
    - technically easy
- Hand-written deterministic rules for tag+form ⇒ lemma

Word Sense Disambiguation, Word Chunking

- Sets of senses for each E, F word:
  - e.g. book-1, book-2, ..., book-n
  - prepositions (de-1, de-2, de-3,...), many others
- Senses derived automatically using the TM
  - translation probabilities measured on senses: p(de-3|from-5)
- Result:
  - statistical model for assigning senses monolingually based on context (also MaxEnt model used here for each word)
- Chunks: group words for non-compositional translation

Generation

- Inverse of analysis
- Much simpler:
  - Chunks ⇒ words (lemmas) with senses (trivial)
  - Words (lemmas) with senses ⇒ words (lemmas) (trivial)
  - Words (lemmas) + tags ⇒ word forms
- Additional step:
  - Source-language ambiguity:
    - electric vs. electrical, hath vs. has, you vs. thou: treated as a single unit in translation proper, but must be disambiguated at the end of generation phase; using additional pure LM on word forms.