Language and Computers (Ling 384)

Topic 4: Writer's aids (Spelling and Grammar Correction)

Detmar Meurers*

Dept. of Linguistics, OSU
Winter 2005

---

* The course was created together with Markus Dickinson and Chris Brew.

Outline

Introduction

Error causes

Difficult issues

Non-word error detection

Isolated-word error correction

Grammar correction

Caveat emptor

Who cares about spelling?

> *Aoccdrnig to a rscheearch at Cmabrigde Uinervtisy, it deosn't mttaer in waht oredr the ltteers in a wrod are, the olny iprmoetnt tihng is taht the frist and lsat ltteer be at the rghit pclae. The rset can be a toatl mses and you can sitll raed it wouthit porbelm. Tihs is bcuseae the huamn mnid deos not raed ervey lteter by istlef, but the wrod as a wlohe.*

(See http://www.mrc-cbu.cam.ac.uk/personal/matt.davis/Cmabrigde/ for the story behind this supposed research report.)

> *A dootcr has aimttded the magltheuansr of a tageene ceacnr pintaet who deid aetfr a hatospil durg blendur.*

Why people care about spelling

- Misspellings can cause misunderstandings and real-life problems:
  - For example:
    - Did you see her god yesterday? It's a big golden retriever.
    - This will be a fee [free] concert.
  - 1991 Bell Atlantic & Pacific Bell telephone network outages were partly caused by a typographical error: A *6* in a line of computer code was supposed to be a *D*. "That one error caused the equipment and software to fail under an avalanche of computer-generated messages." (Wall Street Journal, Nov. 25, 1991)

Why people care about spelling (cont.)

- Standard spelling makes it easy to organize words and text:
  - e.g., Without standard spelling, how would you look up things in a lexicon or thesaurus?
  - e.g., Optical character recognition software can use knowledge about standard spelling to recognize scanned words even for hardly legible input.
- Standard spelling makes it possible to provide a single text, which is accessible to a wide range of readers (different backgrounds, speaking different dialects, etc.).
- Using standard spelling is associated with being well-educated, i.e., is used to make a good impression in social interaction.

How are spell checkers used?

- **interactive spelling checkers** = spell checker detects errors as you type.
  - It may or may not make suggestions for correction.
  - Requires a "real-time" response (i.e., must be fast)
  - It is up to the human to decide if the spell checker is right or wrong.
  - If there are a list of choices, we may not require 100% accuracy in the corrected word
- **automatic spelling correctors** = spell checker runs on a whole document, finds errors, and corrects them
  - A much more difficult task.
  - A human may or may not proofread the results later.

Detection vs. Correction

- There are two distinct tasks:
  - **error detection** = simply find the misspelled words
  - **error correction** = correct the misspelled words
- e.g., It might be easy to tell that *ater* is a misspelled word, but what is the correct word? *water*? *later*? *after*?

⇒ Depends on what we want to do with our results as to what we want to do.
Note, though, that detection is a prerequisite for correction.

What causes errors?

- Keyboard mistypings
- Phonetic errors
- Knowledge problems

Keyboard mistypings

Space bar issues

- **run-on** errors = two separate words become one
  - e.g., *the fuzz* becomes *thefuzz*
- **split** errors = one word becomes two separate words
  - e.g., *equalization* becomes *equali zation*

Note that the resulting items might still be words!

- e.g., *a tollway* becomes *atoll way*

Keyboard mistypings (cont.)

Keyboard proximity

- e.g., *Jack* becomes *Hack* since *h* and *j* are next to each other on a typical American keyboard

Physical similarity

- similarity of shape, e.g., mistaking two physically similar letters when typing up something handwritten
  - e.g., *tight* for *fight*

Phonetic errors

phonetic errors = errors based on the sounds of a language (not necessarily on the letters)

- **homophones** = two words which sound the same
  - e.g., *red*/*read* (past tense), *cite*/*site*/*sight*, *they're*/*their*/*there*
- **Spoonerisms** = switching two letters/sounds around
  - e.g., *It's a tavy grain with biscuit wheels.*

Phonetic errors (cont.)

- letter substitution: replacing a letter (or sequence of letters) with a similar-sounding one
  - e.g., *John kracked his nuckles.* instead of *John cracked his knuckles.*
  - e.g., *I study sikologee.*
- word replacement: replacing one word with some similar-sounding word
  - e.g., *John battled me on the back.* instead of *John patted me on the back.*

More examples for phonetic errors

(1) a. death in Venice
b. deaf in Venice
(2) a. give them an ice bucket
b. give them a nice bucket
(3) a. the stuffy nose
b. the stuff he knows
(4) a. the biggest hurdle
b. the biggest turtle
(5) a. some others
b. some mothers

Knowledge problems

- not knowing a word and guessing its spelling (can be phonetic)
  - e.g., *sientist*
- not knowing a rule and guessing it
  - e.g., Do we double a consonant for *ing* words? *jog* → *joging*

What makes spelling correction difficult?

- **Tokenization**: What is a word?
- **Inflection**: How are some words related?
- **Productivity of language**: How many words are there?

How we handle these issues determines how we build a dictionary.

Tokenization

Intuitively a “word” is simply whatever is between two spaces, but this is not always so clear.

- contractions = two words combined into one
  - e.g., *can't*, *he's*, *John's [car]* (vs. *his car*)
- multi-token words = (arguably) a single word with a space in it
  - e.g., *New York*, *in spite of*, *deja vu*
- hyphens (note: can be ambiguous if a hyphen ends a line)
  - Some are always a single word: *e-mail*, *co-operate*
  - Others are two words combined into one: *Columbus-based*, *sound-change*
- Abbreviations: may stand for multiple words
  - e.g., *etc.* = *et cetera*, *ATM* = *Automated Teller Machine*

Inflection

- A word in English may appear in various guises due to word **inflections** = word endings which are fairly systematic for a given part of speech
  - plural noun ending: *the boy* + *s* → *the boys*
  - past tense verb ending: *walk* + *ed* → *walked*
- This can make spell-checking hard:
  - There are exceptions to the rules: *mans*, *runned*
  - There are words which look like they have a given ending, but they don't: *Hans*, *deed*

Productivity

- part of speech change: nouns can be verbified
  - *emailed* is a common new verb coined after the noun *email*
- morphological productivity: prefixes and suffixes can be added
  - e.g., I can speak of *un-email-able* for someone who you can't reach by email.
- words entering and exiting the lexicon, e.g.:
  - *thou*, or *spleet* 'split' (*Hamlet III.2.10*) are on their way out
  - *d'oh* seems to be entering

Techniques used for spell checking

- Non-word error detection
- Isolated-word error correction
- Context-dependent word error detection and correction → grammar correction.

Non-word error detection

- **non-word error detection** is essentially the same thing as **word recognition** = splitting up “words” into true words and non-words.
- How is non-word error detection done?
  - using a dictionary (construction and lookup)
  - n-gram analysis

Dictionaries

Intuition:

- Have a complete list of words and check the input words against this list.
- If it’s not in the dictionary, it’s not a word.

Two aspects:

- **Dictionary construction** = build the dictionary (what do you put in it?)
- **Dictionary lookup** = lookup a potential word in the dictionary (how do you do this quickly?)

Dictionary construction

- Do we include inflected words? i.e., words with prefixes and suffixes already attached.
  - Pro: lookup can be faster
  - Con: takes much more space, doesn’t account for new formations
- Want the dictionary to have only the word relevant for the user → **domain-specificity**
  - e.g., For most people *memoize* is a misspelled word, but in computer science this is a technical term and spelled correctly.
- Foreign words, hyphenations, derived words, proper nouns, and new words will always be problems for dictionaries since we cannot predict these words until humans have made them words.
- Dictionary should probably be dialectally consistent.
  - e.g., include only *color* or *colour* but not both

Dictionary lookup

Several issues arise when trying to look up a word:

- Have to make lookup fast by using efficient lookup techniques, such as a hash table (cf. the indices we discussed under the searching topic)
- Have to strip off prefixes and suffixes if the word isn’t an entry by itself.

N-gram analysis

- An **n-gram** here is a string of *n* letters.

a	1-gram (unigram)
at	2-gram (bigram)
ate	3-gram (trigram)
late	4-gram
⋮	⋮

- We can use this n-gram information to define what the possible strings in a language are.
  - e.g., *po* is a *po*ssible English string, whereas *kvt* is not.

How do we store and use n-gram information?

- We could have a list of possible and impossible n-grams (1 = possible, 0 = impossible):

po	1
kvt	0
police	1
asdf	0

- Any word which has a 0 for any substring is a misspelled word.
- Problems with such an approach:
  - Information is repeated (*po* is in *police*)
  - Requires a lot of computer storage space
  - Inefficient (slow) when looking up every string

Bigram array

- Instead, we can define a **bigram array** = information stored in a tabular fashion.
- An example, for the letters *k*, *l*, *m*, with examples in parentheses

	. . .	k	l	m	. . .
⋮					
k		0	1 (*tac**kl**e*)	1 (*Hac**km**an*)	
l		1 (*e**lk***)	1 (*he**ll**o*)	1 (*a**lm**s*)	
m		0	0	1 (*ha**mm**er*)	
⋮					

- The first letter of the bigram is given by the vertical letters (i.e., down the side), the second by the horziontal ones (i.e., across the top).
- This is a **non-positional bigram array** = the array 1’s and 0’s apply for a string found anywhere within a word (beginning, 4th character, ending, etc.).

Positional bigram array

- To store information specific to the beginning, the end, or some other position in a word, we can use a **positional bigram array** = the array only applies for a given position in a word.
- Here’s the same array as before, but now only applied to word endings:

	. . .	k	l	m	. . .
⋮					
k		0	0	0	
l		1 (*e**lk***)	1 (*ha**ll***)	1 (*e**lm***)	
m		0	0	0	
⋮					

Isolated-word error correction

- Having discussed how errors can be detected, we want to know how to correct these misspelled words:
  - The most common method is **isolated-word error correction** = correcting words without taking context into account.
  - Note: This technique can only handle errors that result in non-words.
- Knowledge about what is a typical error helps in finding correct word.

Knowledge about typical errors

- word length effects: most misspellings are within two characters in length of original
  - → When searching for the correct spelling, we do not usually need to look at words with greater length differences.
- first-position error effects: the first letter of a word is rarely erroneous
  - → When searching for the correct spelling, the process is sped up by being able to look only at words with the same first letter.

Isolated-word error correction methods

- Many different methods are used; we will briefly look at four methods:
  - rule-based methods
  - similarity key techniques
  - minimum edit distance
  - probabilistic methods
- The methods play a role in one of the three basic steps:
  1. Detection of an error (discussed above)
  2. Generation of candidate corrections
     - rule-based methods
     - similarity key techniques
  3. Ranking of candidate corrections
     - probabilistic methods
     - minimum edit distance

Rule-based methods

One can generate correct spellings by writing rules:

- Common misspelling rewritten as correct word:
  - e.g., *hte* → *the*
- Rules
  - based on inflections:
    - e.g., *V+C+ing* → *V+CC+ing* (where V = vowel and C = consonant)
  - based on other common spelling errors (such as keyboard effects or common transpositions):
    - e.g., *CsC* → *CaC*
    - e.g., *Cie* → *Cei*

Similarity key techniques

- Problem: How can we find a list of possible corrections?
- Solution: Store words in different boxes in a way that puts the similar words together.
- Example:
  1. Start by storing words by their first letter (first letter effect),
     - e.g., *punc* starts with the code P.
  2. Then assign numbers to each letter
     - e.g., 0 for vowels, 1 for *b, p, f, v* (all bilabials), and so forth, e.g., *punc* → P052
  3. Then throw out all zeros and repeated letters,
     - e.g., P052 → P52.
  4. Look for real words within the same box,
     - e.g., *punk* is also in the P52 box.

Probabilistic methods

Two main probabilities are taken into account:

- **transition probabilities** = probability (chance) of going from one letter to the next.
  - e.g., What is the chance that *a* will follow *p* in English? That *u* will follow *q*?
- **confusion probabilities** = probability of one letter being mistaken (substituted) for another (can be derived from a confusion matrix)
  - e.g., What is the chance that *q* is confused with *p*?

Useful to combine probabilistic techniques with dictionary methods

---

Confusion probabilities

- For the various reasons discussed above (keyboard layout, phonetic similarity, etc.) people type other letters than the ones they intended.
- It is impossible to fully investigate all possible error causes and how they interact, but we can learn from watching how often people make errors and where.
- One way of doing so is to build a **confusion matrix** = a table indicating how often one letter is mistyped for another

		correct				
		…	r	s	t	…
	⋮					
	r		n/a	12	22	
typed	s		14	n/a	15	
	t		11	37	n/a	
	⋮					

(cf. Kernighan et al 1999)

How is a mistyped word related to the intended?

Types of errors

- **insertion** = a letter is added to a word
- **deletion** = a letter is deleted from a word
- **substitution** = a letter is put in place of another one
- **transposition** = two adjacent letters are switched

Note that the first two alter the length of the word, whereas the second two maintain the same length.

General properties

- **single-error misspellings** = only one instance of an error
- **multi-error misspellings** = multiple instances of errors (harder to identify)

Minimum edit distance

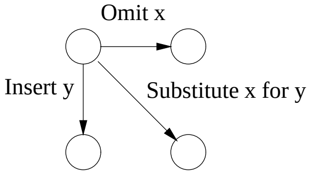
- In order to rank possible spelling corrections, it can be useful to calculate the **minimum edit distance** = minimum number of operations it would take to convert one word into another.
- For example, we can take the following five steps to convert *junk* to *haiku*:
  1. *ju**n**k* → *juk* (deletion)
  2. ***j**uk* → ***h**uk* (substitution)
  3. ***huk*** → *h**ku*** (transposition)
  4. *hku* → *h**i**ku* (insertion)
  5. *hiku* → *h**a**iku* (insertion)
- But is this the minimal number of steps needed?

Computing edit distances
Figuring out the worst case

- To be able to compute the edit distance of two words at all, we need to ensure there is a finite number of steps.
- This can be accomplished by
  - requiring that letters cannot be changed back and forth a potentially infinite number of times, i.e., we
  - limit the number of changes to the size of the material we are presented with, the two words.
- Idea: Never deal with a character in either word more than once.
- Result:
  - In the worst case, we delete each character in the first word and then insert each character of the second word.
  - The worst case edit distance for two words is *length*(*word*1) + *length*(*word*2)

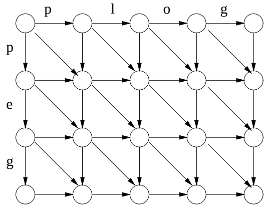
Computing edit distances
Using a graph to map out the options

- To calculate minimum edit distance, we set up a **directed, acyclic graph**, a set of nodes (circles) and arcs (arrows).
- Horizontal arcs correspond to deletions, vertical arcs correspond to insertions, and diagonal arcs correspond to substitutions (and a letter can be "substituted" for itself).



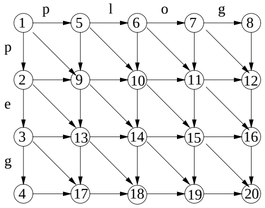
Computing edit distances
An example graph

- Say, the user types in *plog*.
- We want to calculate how far away *peg* is (one of the possible corrections). In other words, we want to calculate the minimum edit distance (or minimum edit cost) from *plog* to *peg.*
- As the first step, we draw the following directed graph:



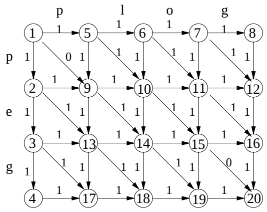
Computing edit distances
Adding numbers to the example graph

- The graph is **acyclic** = for any given node, it is impossible to return to that node by following the arcs.
- We can add identifiers to the states, which allows us to define a **topological order**:



Computing edit distances
Adding costs to the arcs of the example graph

- We need to add the costs involved to the arcs.
- In the simplest case, the cost of deletion, insertion, and substitution is 1 each (and substitution with the same character is free).



- Instead of assuming the same cost for all operations, in reality one will use different costs, e.g., for the first character or based on the confusion probability.

Computing edit distances
How to compute the path with the least cost

We want to find the path from the start (1) to the end (2) with the least cost.

- The simple but dumb way of doing it:
  - Follow every path from start (1) to finish (20) and see how many changes we have to make.
  - But this is very inefficient! There are 131 different paths to check.

Computing edit distances
The smart way to compute the least cost

- The smart way to compute the least cost uses **dynamic programming** = a program designed to make use of results computed earlier
  - We follow the topological ordering.
  - As we go in order, we calculate the least cost for that node:
    - We add the cost of an arc to the cost of reaching the node this arc originates from.
    - We take the minimum of the costs calculated for all arcs pointing to a node and store it for that node.
  - The key point is that we are storing partial results along the way, instead of recalculating everything, every time we compute a new path.

Context-dependent word correction

Context-dependent word correction = correcting words based on the surrounding context.

- This will handle errors which are real words, just not the right one or not in the right form.
- Essentially a fancier name for a **grammar checker** = a mechanism which tells a user if their grammar is wrong.

Grammar correction—what does it correct?

- Syntactic errors = errors in how words are put together in a sentence: the order or form of words is incorrect, i.e., ungrammatical.
- **Local** syntactic errors: 1-2 words away
  - e.g., *The study was conducted mainly* ***be*** *John Black.*
  - A verb is where a preposition should be.
- **Long-distance** syntactic errors: (roughly) 3 or more words away
  - e.g., *The* ***kids*** *who are most upset by the little totem* ***is*** *going home early.*
  - Agreement error between subject *kids* and verb *is*

More on grammar correction

- Semantic errors = errors where the sentence structure sounds okay, but it doesn't really mean anything.
    - e.g., *They are leaving in about fifteen* ***minuets*** *to go to her house.*

  ⇒ *minuets* and *minutes* are both plural nouns, but only one makes sense here

There are many different ways in which grammar correctors work, two of which we'll focus on:

- Bigram model (bigrams of words)
- Rule-based model

Bigram grammar correctors

We could also look at **bigrams**: now we are talking about bigrams of words, i.e., two words appearing next to each other.

- **Question:** Given the previous word, what is the probability of the current word?
    - e.g., given *these*, we have a 5% chance of seeing *reports* and a 0.001% chance of seeing *report* (*these report cards*).
    - Thus, we will change *report* to *reports*
- But there's a major problem: we may hardly ever see *these reports*, so we won't know the probability of that bigram.
- **(Partial) Solution:** use bigrams of **parts of speech**.
    - e.g., What is the probability of a noun given that the previous word was an adjective?

Rule-based grammar correctors

We can write regular expressions to target specific error patterns. For example:

- *To a certain extend, we have achieved our goal.*
    - Match the pattern *some* or *certain* followed by *extend*, which can be done using the regular expression `some|certain extend`
    - Change the occurrence of *extend* in the pattern to *extent*.
- Naber (2003) uses 56 such rules to build a grammar corrector which works nearly as well as that in commercial products.

Beyond regular expressions

- But what about correcting the following:
    - *A baseball teams were successful.*
- We should change *A* to *The*, but a simple regular expression doesn't work because we don't know where the word *teams* might show up.
    - ***A*** *wildly overpaid, horrendous baseball* ***teams*** *were successful.* (Five words later; change needed.)
    - ***A*** *player on both my* ***teams*** *was successful.* (Five words later; no change needed.)
- We need to look at how the sentence is constructed in order to build a better rule.

Syntax

- **Syntax** = the study of the way that sentences are constructed from smaller units.
- There cannot be a "dictionary" for sentences since there is an infinite number of possible sentences:

(6) The house is large.
(7) John believes that the house is large.
(8) Mary says that John believes that the house is large.

There are two basic principles of sentence organization:

- Linear order
- Hierarchial structure (Constituency)

Linear order

- **Linear order** = the order of words in a sentence.
- A sentence has different meanings based on its linear order.

(9) John loves Mary.
(10) Mary loves John.

- Languages vary as to what extent this is true, but linear order in general is used as a guiding principle for organizing words into meaningful sentences.
- Simple linear order as such is not sufficient to determine sentence organization though. For example, we can't simply say "The verb is the second word in the sentence."

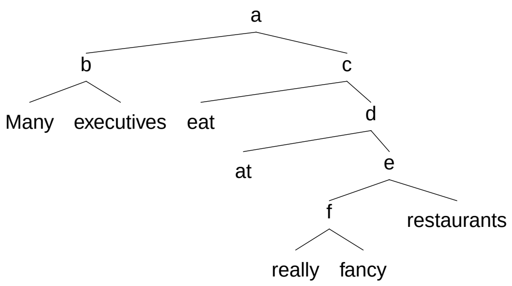
(11) I **eat** at really fancy restaurants.
(12) Many executives **eat** at really fancy restaurants.

Constituency

- What are the "meaningful units" of a sentence like *Many executives eat at really fancy restaurants*?
    - Many executives
    - really fancy
    - really fancy restaurants
    - at really fancy restaurants
    - eat at really fancy restaurants
- We refer to these meaningful groupings as **constituents** of a sentence.
- There are many "tests" to determine what a constituent is, but we will not concern ourselves with them here.

Hierarchical structure

- Constituents can appear within other constituents. We can represent this in a bracket form or in a **syntactic tree**
- Constituents shown through brackets:
  [[Many executives] [eat [at [[really fancy] restaurants]]]]
- Constituents displayed as a tree:



Categories

- We would also like some way to say that *Many executives* and *really fancy restaurants* are the same type of grouping, or constituent, whereas *at really fancy restaurants* seems to be something else.
- For this, we will talk about different **categories**
    - Lexical
    - Phrasal

Lexical categories

Lexical categories are simply word classes, or what you may have heard as **parts of speech**. The main ones are:

- verbs: *eat, drink, sleep, ...*
- nouns: *gas, food, lodging, ...*
- adjectives: *quick, happy, brown, ...*
- adverbs: *quickly, happily, well, westward*
- prepositions: *on, in, at, to, into, of, ...*
- determiners/articles: *a, an, the, this, these, some, much, ...*

Determining lexical categories

How do we determine which category a word belongs to?

- Distribution: Where can these kinds of words appear in a sentence?
  - e.g., Nouns like *mouse* can appear after articles ("determiners") like *the*, while a verb like *eat* cannot.
- **Morphology**: What kinds of word prefixes/suffixes can a word take?
  - e.g., Verbs like *walk* can take a *ed* ending to mark them as past tense. A noun like *mouse* cannot.

Closed & Open classes

We can add words to some classes, but not to others. This also seems to correlate with whether a word is "meaningful" or just a **function word** = only meaning comes from its usage in a sentence.
Open classes: new words can be easily added:

- verbs
- nouns
- adjectives
- adverbs

Closed classes: new words cannot be easily added:

- prepositions
- determiners

Phrasal categories

What about phrases? Can we assign them categories? We can also look at their distribution and see which ones behave in the same way.

- The joggers ran through the park.

What other phrases can we put in place of *The joggers*?

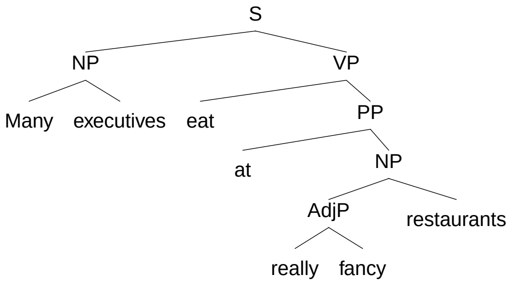
Phrasal categories (cont.)

- Susan
- students
- you
- most dogs
- some children
- a huge, lovable bear
- my friends from Brazil
- the people that we interviewed

Since all of these contain nouns, we consider these to be *noun phrases*, abbreviated with NP.

Building a tree

Other phrases work similarly (S = sentence, VP = verb phrase, PP = prepositional phrase, AdjP = adjective phrase):



Phrase Structure Rules

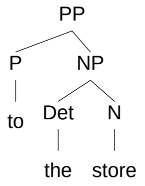
- We can give rules for building these phrases. That is, we want a way to say that a determiner and a noun make up a noun phrase, but a verb and an adverb do not.
- **Phrase structure rules** are a way to build larger constituents from smaller ones.
  - e.g., S → NP VP
    This says:
    - A sentence (S) constituent is composed of a noun phrase (NP) constituent and a verb phrase (VP) constituent. (hierarchy)
    - The NP must precede the VP. (linear order)

Some other English rules

- NP → Det N (*the cat, a house, this computer*)
- NP → Det AdjP N (*the happy cat, a really happy house*)
  - For phrase structure rules, as shorthand parentheses are used to express that a category is optional.
  - We thus can compactly express the two rules above as one rule:
  - NP → Det (AdjP) N
  - Note that this is different and has nothing to do with the use of parentheses in regular expressions.
- AdjP → (Adv) Adj (*really happy*)
- VP → V (*laugh, run, eat*)
- VP → V NP (*love John, hit the wall, eat cake*)
- VP → V NP NP (*give John the ball*)
- PP → P NP (*to the store, at John, in a New York minute*)
- NP → NP PP (*the cat on the stairs*)

Phrase Structure Rules and Trees

With every phrase structure rule, you can draw a tree for it.



Phrase Structure Rules in Practice

Try analyzing these sentences and drawing trees for them, based on the phrase structure rules given above.

- The man in the kitchen drives a truck.
- That dang cat squeezed some fresh orange juice.
- The mouse in the corner by the stairs ate the cheese.

Properties of Phrase Structure Rules

- **generative** = a schematic strategy that describes a set of sentences completely.
- potentially **(structurally) ambiguous** = have more than one analysis

  (13) We need more intelligent leaders.

  (14) Paraphrases:
    a. We need leaders who are more intelligent.
    b. Intelligent leaders? We need more of them!

- **hierarchical** = categories have internal structure; they aren't just linearly ordered.
- **recursive** = property allowing for a rule to be reapplied (within its hierarchical structure).
  e.g., **NP** → NP PP
  PP → P **NP**
  The property of recursion means that the set of potential sentences in a language is **infinite**.

Context-free grammars

A **context-free grammar** (CFG) is essentially a collection of phrase structure rules.

- It specifies that each rule must have:
  - a left-hand side (LHS): a single **non-terminal** element = (phrasal and lexical) categories
  - a right-hand side (RHS): a mixture of non-terminal and terminal elements
    terminal elements = actual words
- A CFG tries to capture a natural language completely.

Why "context-free"? Because these rules make no reference to any context surrounding them. i.e. you can't say "PP → P NP" *when* there is a verb phrase (VP) to the left.

Pushdown automata

Pushdown automaton = the computational implementation of a context-free grammar.
It uses a **stack** (its memory device) and has two operations:

- **push** = put an element onto the top of a stack.
- **pop** = take the topmost element from the stack.

This has the property of being **Last In First Out (LIFO)**.
So, when you have a rule like "PP → P NP", you push NP onto the stack and then push P onto it. If you find a preposition (e.g., *on*), you pop P off of the stack and now you know that the next thing you need is an NP.

Parsing

So, using these phrase structure (context-free) rules and using something like a pushdown automaton, we can get a computer to **parse** a sentence = assign a structure to a sentence.
Do you parse top-down or bottom-up (or a mixture)?

- **top-down**: build a tree by starting at the top (i.e. S → NP VP) and working down the tree.
- **bottom-up**: build a tree by starting with the words at the bottom and working up to the top.

There are many, many parsing techniques out there.

Writing grammar correction rules

So, with context-free grammars, we can now write some correction rules, which we will just sketch here.

- *A baseball teams were successful.*

A followed by PLURAL NP: change *A* → *The*

- *John at the taco.*

The structure of this sentence is NP PP, but that doesn't make up a whole sentence. We need a verb somewhere.

Is this really how spell checkers work?

As far as we know, yes, but:

- Many spell checkers are proprietary and the way they work is kept secret; we don't know how they work exactly, which hampers research and thereby progress.
- Others, such as aspell and ispell, are **open source** spell checkers, meaning that anyone can
  - contribute to their further development, and
  - see how they work, which makes it possible to understand exactly what they will and what they won't catch.

  (cf. http://aspell.sourceforge.net/ and http://fmg-www.cs.ucla.edu/fmg-members/geoff/ispell.html)

Dangers of spelling and grammar correction

- The more we depend on spelling correctors, the less we try to correct things on our own. But spell checkers are not 100%
- A study at the University of Pittsburgh found that students made **more** errors when using a spell checker!

	high SAT scores	low SAT scores
use checker	16 errors	17 errors
no checker	5 errors	12.3 errors

(cf., http://www.wired.com/news/business/0,1367,58058,00.html)

A Poem on the Dangers of Spell Checkers

Michael Livingston
(http://www.courses.rochester.edu/livingston/guide/phonix.html)

Eye halve a spelling chequer
It came with my pea sea.
It plainly marques four my revue
Miss steaks eye kin knot sea.
Eye strike a key and type a word
And weight four it two say
Weather eye am wrong oar write
It shows me strait a weigh.
As soon as a mist ache is maid
It nose bee fore two long
And eye can put the error rite
Its rare lea ever wrong.
Eye have run this poem threw it
I am shore your pleased two no
Its letter perfect awl the weigh
My chequer tolled me sew.

References

- The discussion is based on Markus Dickinson (to appear). Writer's Aids. In Keith Brown (ed.): *Encyclopedia of Language and Linguistics. Second Edition.*. Elsevier.
- A major inspiration for that article and our discussion is Karen Kukich (1992): *Techniques for Automatically Correcting Words in Text.* ACM Computing Surveys, pages 377–439.
- For a discussion of the confusion matrix, cf. Mark D. Kernighan, Kenneth W. Church and William A. Gale (1990). A spelling Correction Program Based on a Noisy Channel Model. In *Proceedings of COLING-90*. pp. 205–210.
- An open-source style/grammar checker is described in Daniel Naber (2003). *A Rule-Based Style and Grammar Checker*. Diploma Thesis, Universität Bielefeld. http://www.danielnaber.de/languagetool/