Tagging, Tagsets, and Morphology
The task of (Morphological) Tagging

• 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 ...
  - a step aside:

• Recall: $A^+ \rightarrow 2^{(L,C_1,C_2,...,C_n)} \rightarrow T$
  • morphology
  • tagging: disambiguation ( ~ “select”)
Tagging Examples

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

• General definition:
  - tag ~ (c₁,c₂,...,cₙ)
  - 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):
  - Penn treebank (45) (VBZ: Verb,Pres,3,s,sg, JJR: Adj. Comp.)
  - Brown Corpus (87), Claws c5 (62), London-Lund (197)
Other Language Tagsets

• Differences:
  – size (10..10k)
  – categories covered (POS, Number, Case, Negation,...)
  – level of detail
  – presentation (short names vs. structured ("positional"))

• Example:
  – Czech: AGFS3----1A----
    POS  GENDER  SUBPOS
    CASE  PERSON  NUMBER
    POSSN  DCOMP  TENSE
    VAR  VOICE
    NEG
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” $A^+ \rightarrow 2^{(L,C_1,C_2,...,C_n)} \rightarrow T$:
  – mapping $R : (C_1,C_2,...,C_n) \rightarrow T_{\text{reduced}}$,
    then (new) unique mapping $U : A^+ \times T_{\text{reduced}} \rightarrow (L,T)$
Lemmatization

• Full morphological analysis:
  MA: $A^+ \rightarrow 2^{(L,C_1,C_2,...,C_n)}$
  (recall: a lemma $l \in L$ is a **lexical unit** (~ dictionary entry ref)

• 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)
Morphological Analysis: Methods

- **Word form list**

- **Direct coding**
  - endings: verbreg:s/VBZ, nounreg:s/NNS, adje:er/JJR, ...

- **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?
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)
Finite State Machinery

• Two-level Morphology
  – phonology + “morphotactics” (= morphology)

• Both components use finite-state automata:
  – phonology: “two-level rules”, converted to FSA
    • $e:0 \Leftrightarrow \_ +:0 e:e r:r$
  – morphology: linked lexicons
    • root-dic: book/”book” $\Rightarrow$ end-noun-reg-dic
    • end-noun-reg-dic: $+s/”NNS”$

• Integration of the two possible (and simple)
Finite State Transducer

- FST is a FSA where
  - 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
FST Example

- German umlaut (greatly simplified!):
  \[ u \leftrightarrow ü \text{ if (but not only if) } \text{c h e r follows (Buch } \rightarrow \text{ Bücher)} \]
  rule: \[ u:ü \leftrightarrow _- \text{c:h:e:r:} \]

  
  FST:
  
  Buch/Buch:
  \[ \text{F1 F3 F4 F5} \]
  
  Bucher/Bucher:
  \[ \text{F1 F3 F4 F5 F6 N1} \]
  
  Buch/Buck:
  \[ \text{F1 F3 F4 F1} \]
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:

```
  b  o  o  k  +  s
   a  n  k
```

“book”

“bank”

“NNS”
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 ü ⇒ Bu:Bü ⇒ Buc:Büc ⇒ Buch:Büch ⇒ Buch+e:Büch0e ⇒ Buch+er:Büch0er
  ⇒ 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)
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