Ling 182

Morphology

and Word Recognition

Martin Kay

Stanford University

Trial and Error

1. **Look word up in the lexicon. If we find it, we are done!**
2. **Rewrite the word using one of the morphological rules, and go back to step 1.**

running

ing# $\Rightarrow$ #

runn

nning $\Rightarrow$ n#

run

Morphological Rules

```
suffix([X], [X, 0'$], 0).
suffix("ies", "ie", s).
suffix("ies", "y", s).
suffix("ches", "ch", s).
suffix("shes", "sh", s).
suffix("ches", "", 0).
suffix("shes", "", 0).
suffix([X, 0'h, 0's], "", 0) :- member(X, "cs").
suffix([S, 0'e, 0's], [S], s) :-
  member(S, "jsxz").
suffix([X, 0's], [X], s) :- \+ member(X, "jsxz").
suffix("s", [0'$], s).
```

Word Recognition

A word recognizer takes a string of characters as input and returns “yes” or “no” according as the word is or is not in a given set.

Solves the *membership* problem.

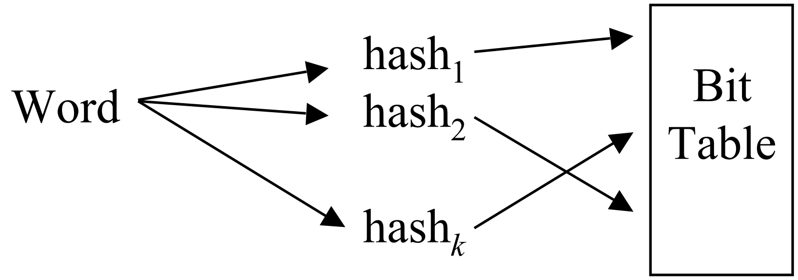
e.g. Spell Checking, Scrabble

Approximate methods

Has right set of letters (any order).

Has right sounds (Soundex).

Random (suprimposed) coding (Unix Spell)



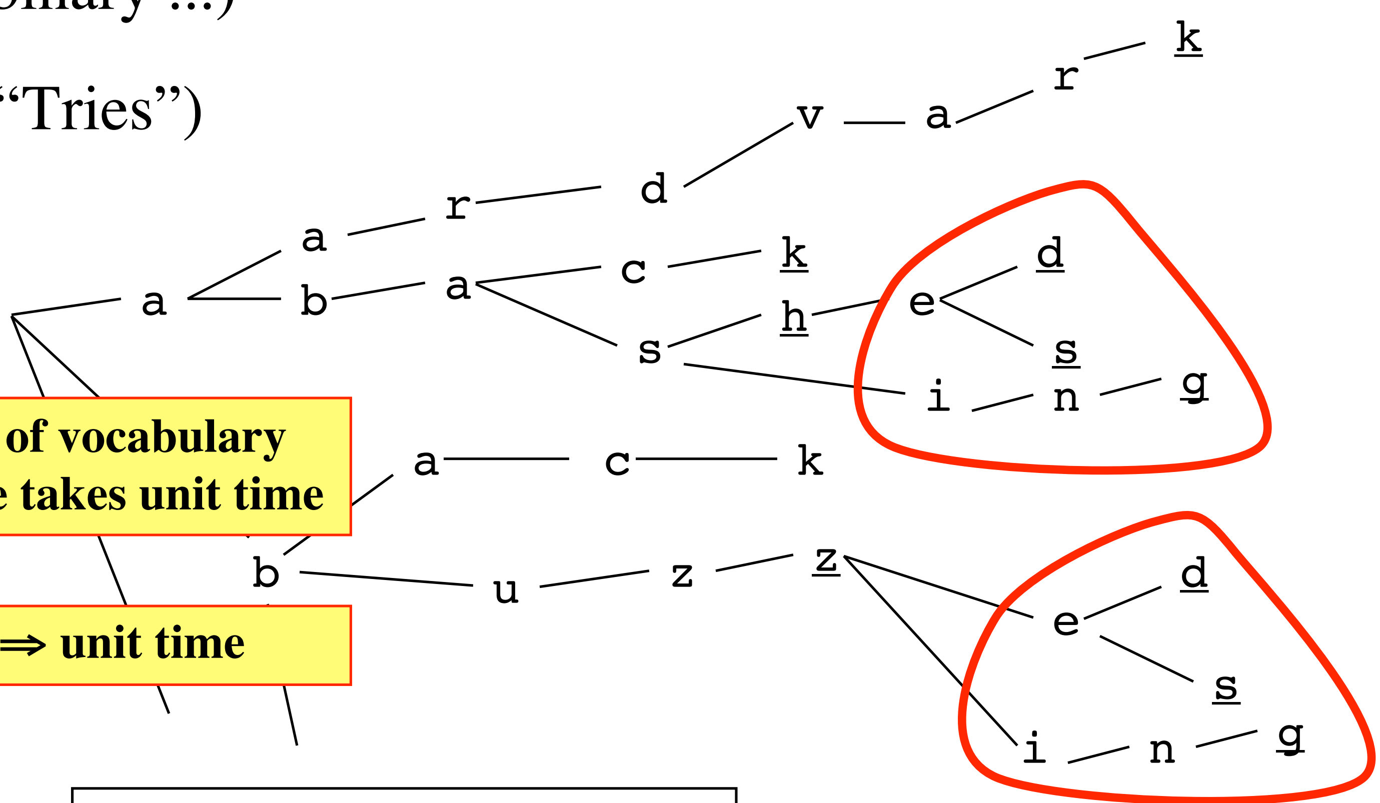
Exact Methods

Hashing

Search (linear, binary ...)

Digital search ("Tries")

Redundancy



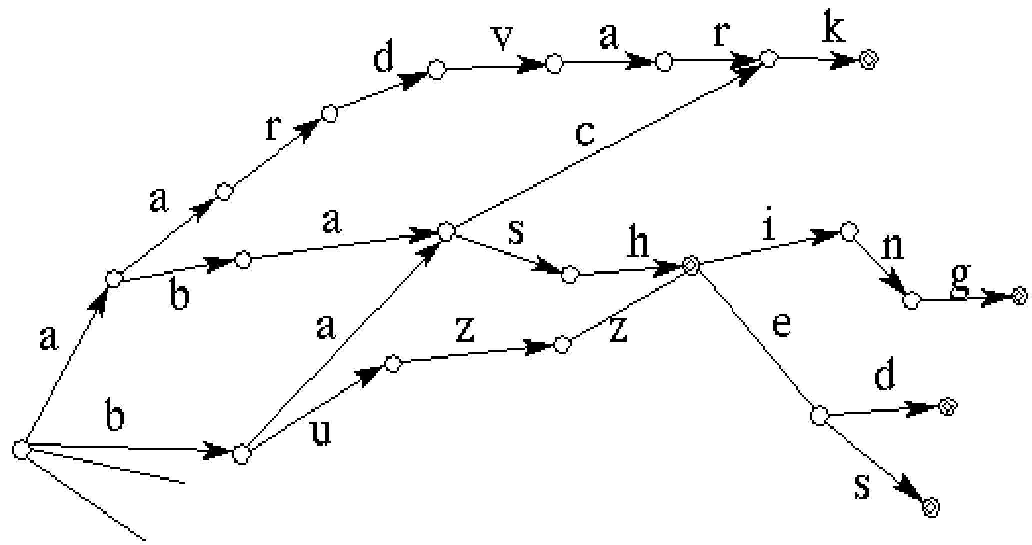
Lookup independent of vocabulary size (if visiting a node takes unit time

Binary alphabet ⇒ unit time

Folds together common <u>prefixes</u>

Exact Methods (continued)

- **Digital Acyclic Word Graph**
- **Finite-state automata**



Folds together common prefixes and suffixes

Enumeration vs. Description

- **Enumeration**

  Representation includes an item for each object.

  Size = f(Items)

Description

Representation provides a characterization of the set of all items.

Size = g(Common properties, Exceptions)

Adding item can decrease size.

Classification

	Exact	Approximate
Ennumeration	**Hash table** **Binary Search**	**Soundex**
Description	**Trie** **FSM**	**Unix Spell**

Dictionary Lookup

Dictionary lookup takes a string of characters as input and returns "yes" or "no" according as the word is or is not in a given set *and returns information about the word*.

Lookup Methods

Approximate — guess the information

If it ends in "ed", it's a past-tense verb.

Exact — store the information for finitely many words

Table Lookup

- **Hash**
- **Search**
- **Trie —store at word-endings.**

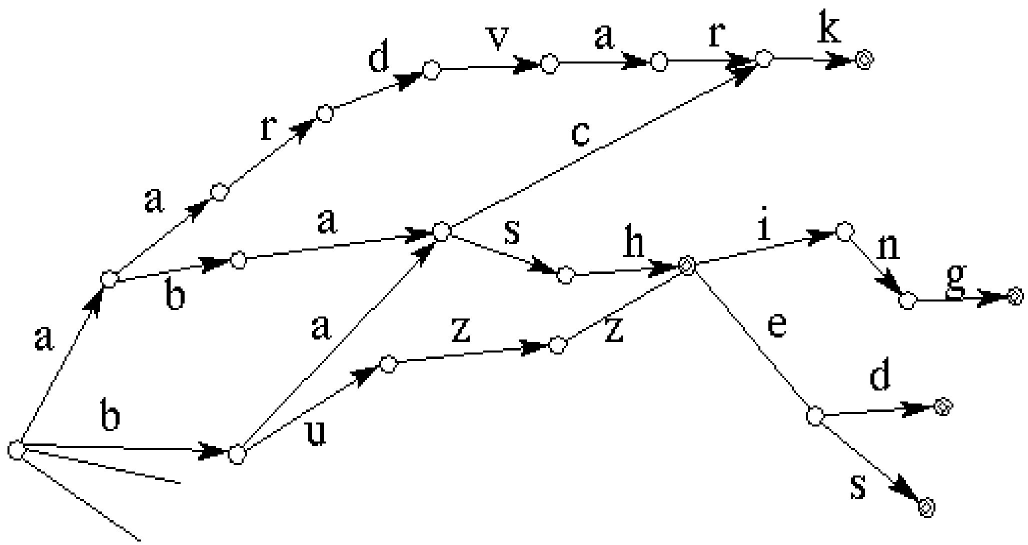
FSM

- **Store at final states?**

  No suffix collapse — reverts to Trie.

Exact Methods (continued)

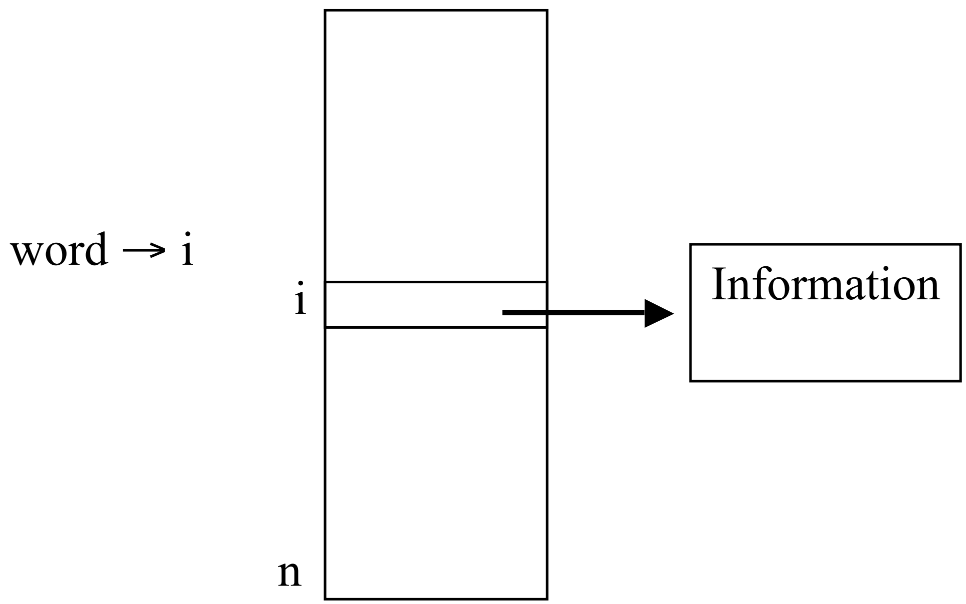
- **Digital Acyclic Word Graph**
- **Finite-state automata**



Folds together common prefixes and suffixes

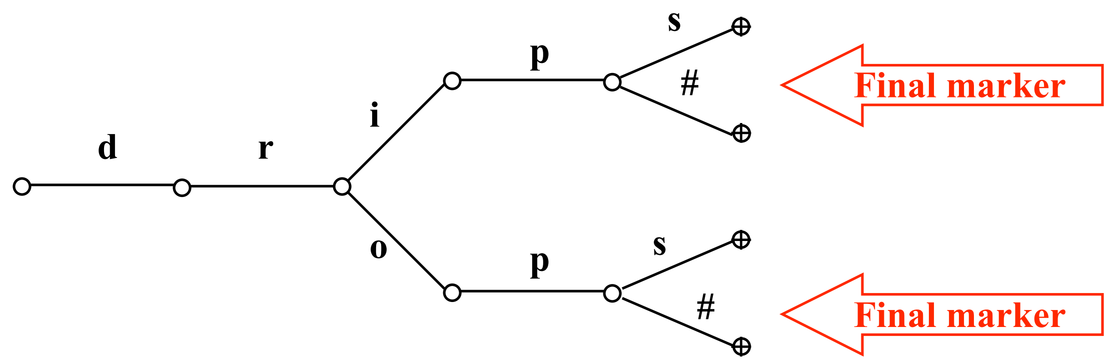
Word Identifiers

Associate a unique, useful, identifier with each of *n* words, e.g. an integer from 1 to *n*. This can be used to index a vector of dictionary information.



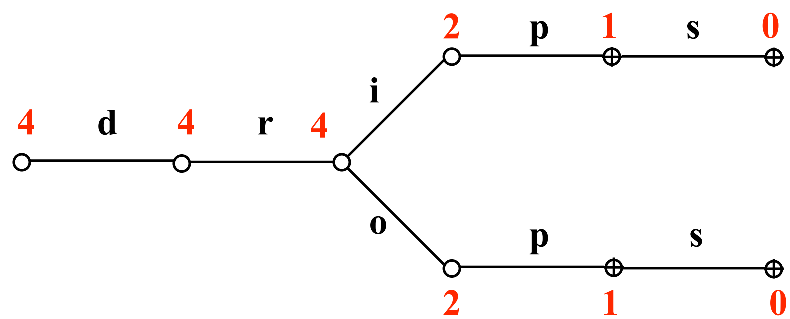
Digital Search Trees

Ordered Read-out



Digital Search Trees

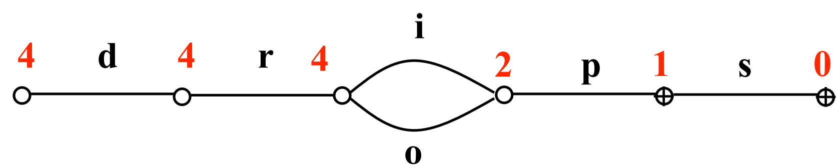
Eliminate final marker



Suffix counts

Digital Search Tree

Collapse common tails



- **Minimal Perfect Hash (Lucchesi and Kowaltowski)**
- **Word-number mapping (Kaplan and Kay, 1985)**

Minimal Meaningful Units

Walk / walks / walked / walking

amo / amas / amat / amamus ... (Latin)

Unredecontaminatability

Untieable / undoable

Word can be decomposable

Paradigms

- **Declensions**
- **Conjugations**
- **...**

Productivity

true

truer

truest

truly

truth

truthful

truthfully

untruthfullness

...

Morphological “Processes”

Affixation ~ Concatenation

- walking = walk+ing
- undoable = un+do+able

Infixation

- sulat (write) / sumulat (to write) (Tagalog)
- bili (buy) / bumili (to buy)

Internal changes (Umlaut~Ablaut)

- swim / swam / swum
- mener / je mène (Frecnh)
- man / men

Intercelation

- kutib = ktb+ui (Arabic)

Reduplication

- begibegi (Malay)
- rumah / rumahrumah (house) (Indonesian)
- tango /tetigi / tactum

Suppletion

- go /went
- fero / tuli / latum

Compounding

Types of Morpholgy

- **Inflexion**
  - —**Does not change basic grammatical category (part of speech)**
    - ***Infinitives / Gerunds***
    - ***Zero derivation***
    - ***Adjective-noun alternation***
  - —**Obligatory**
  - —**Applies after derivation**
  - —**Some laguages have (almost) none**
- **Derivation**
  - —**Changes basic grammatical category (in general)**
- **Compounding**
  - —***Lebensversicherungsgeselschaftangestellter***
  - —**Airport courtesy shuttle pick-up point**

Morphophonology
(Spelling rules)

- **English stress and vowel quality**
  - **porous/porosity telephone/telephony**
  - **parasite/parasitic**
- **French *schwa***
  - **lever / je lève jeter / je jetter**
- **English consonant doubling**
  - **fat / fatter**
  - **set / setting**
- **English -*e* deletion**
  - **love / loving**

Computational Approaches

Analysis by synthesis

Lookup-modify loop

Finite-state technology

Finite-state + context-free technology

Analysis by synthesis

- Make a word list from the text
- Read the whole lexicon
  - generated all forms of each word
  - when a form is in the word list, add information from the lexicon
- Look up the text in the word list
- Pro
  - Parallelism of generation and synthesis
  - Simplicity
- Con
  - Expensive
  - Assumes a finite set of forms for each lexical entry

Look-up Modify Loop

- **For ever:**
  - —Look up the word in the lexicon. Quit if you find it
  - —Apply the next modification rule
- **Examples**

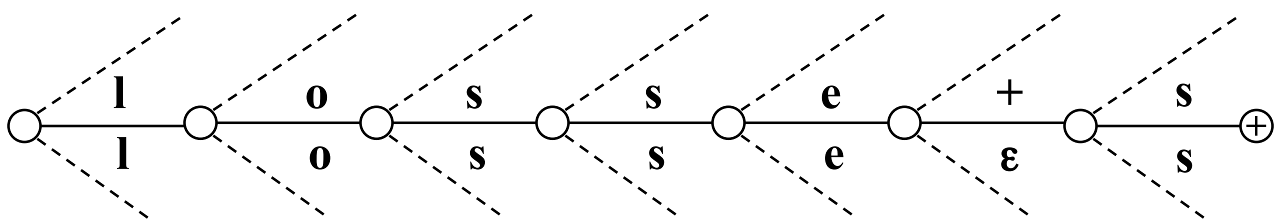
```
loves
s# ⇒ #: love Bingo!
```

```
wishes
s # ⇒ #: wishe
shes# ⇒ sh#: wish Bingo!
```

- **Pros**
  - —Simple(?) and Intuitive
- **Cons**
  - —Generation unrelated to analysis

Finite-state

- **Examine the word from left to right making corresponding transitions in a finite-state transducer. When the transducer enters a final state, the transduced string is the analysis of the word.**



Pros

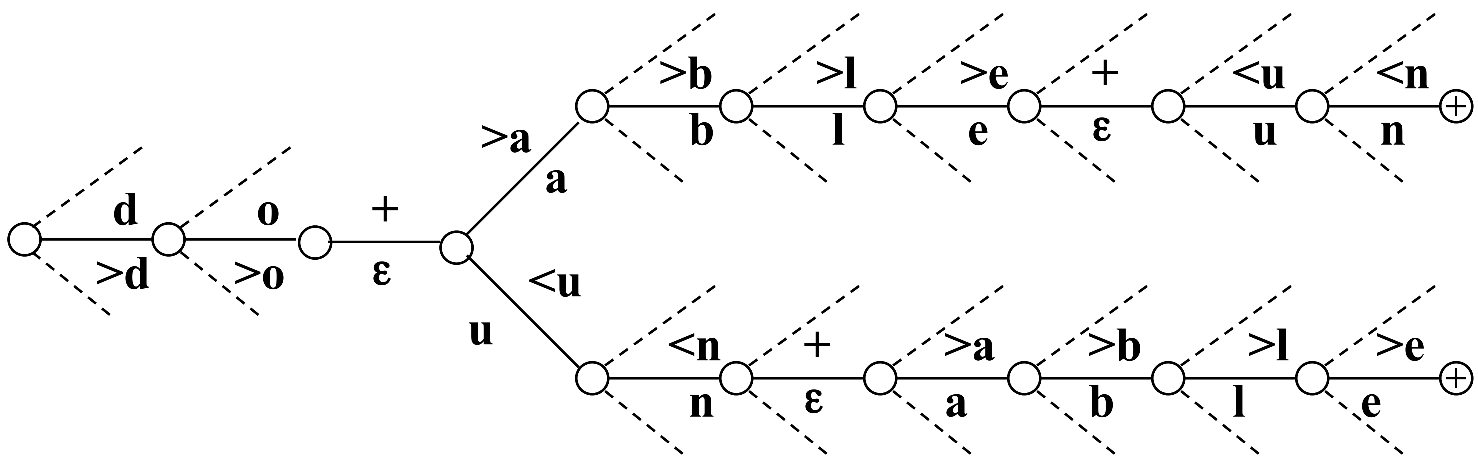
- —**Superfast**
- —**Generation and analysis are (almost) the same**

Cons

- —**Complex compilation**
- —**No structure**

Modified Finite State

- **Each transition specifies on which end of the word (left or right) it operates.**
- **Mildly context free!**



Finite-state = Context-free

- **Everything is syntax except phonology (spelling rules)**

English Derivational Morphology

```
[Verb(ag,str)]:
 > ing#+[Adjective(UN)] s2# er#+[Noun]

[Noun]:
 > s1#

[Verb(ag)]:
 > ed1# ed2# ing#+[Adjective(UN)] s2# er#+[Noun]

[Adjective]:
 > ly# ness#+[Noun]

[Verb(ag,tr)]:
 > ed1# ed2# ing#+[Adjective(UN)] s2# er#+[Noun]
  able#+[Adjective(UN,ITY)]
```

English Derivational Morphology

```
absent:
 + [Verb(ag)]
 + [CE_Adjective]
 + ia# ee#+[Noun]

absinthe:
 + [Noun]

absolute:
 + [Noun]
 + [Adjective]
 + [ISM_IST-2] --{ ist#+[Noun] ist#+ic# }
 + ist#+[Noun] ist#+[IC_Adjective] ion#+[ION_Noun]

absolve:
 + [Verb(ag,tr,pp)]
```

The Problem Set

```
consonant='(b|c|d|f|g|h|j|k|l|m|n|p|q|r|s|t|v|w|x|y|z)'
vowel='(a|e|i|o|u)'


class rule
   __init__(self, lhs, rhs, l_context, r_context)
   __str__(self)
   apply(self, word)                                  -- yield
class
   __init__(self, rules=None)
   compile_rule(self, rule_string)
   compile_rules(self, rules)
   generate_word0(self, word, rules=None)      -- yield
   generate_word(self, word, rules=None)       -- yield
```

Rule Application

```
def apply(self, word):
    i=0
    while i<len(word):
      lc_len=0
      lc_match=re.search(self.compiled_l_context, word[i:])
      if lc_match:
        lc_len=len(lc_match.group(0))
        lhs_match=re.match(self.compiled_lhs, word[i+lc_len:])
        if lhs_match:
          lhs_len=len(lhs_match.group(0))
          rc_match=re.match(self.compiled_r_context,
                   word[i+lc_len+lhs_len:])
          if rc_match:
            if monitor: print '   ', word,
            word=word[:i+lc_len]+self.rhs+word[i+lc_len+lhs_len:]
            if monitor: print '=> %s [%s]' % (word, self)
      i+=max(1, lc_len)
    yield word
```

Generate a word (version 1)

```
def generate_word0(self, word, rules=None):
    if not rules: rules=self.rules
    for rule in rules:
      word=rule.apply(word)
    yield word
```

Why not ‘`return`’

Generate a word (version 2)

```
def generate_word(self, word, rules=None):
    if rules==None: rules=self.rules
    if len(rules)>0:
      for i in rules[0].apply(word):
        for j in self.generate_word(i, rules[1:]):
          yield j
    else:
      yield word
```

What about optional rules?