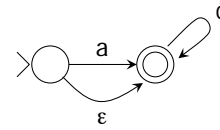


Finite-State Methods

Finite state acceptors (FSAs)



- Regexps
- Union, Kleene *, concat, intersect, complement, reversal
- Determinization, minimization
- Pumping, Myhill-Nerode

slide courtesy of L. Karttunen (modified)

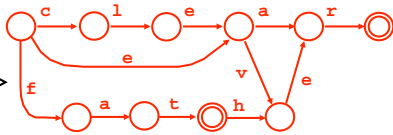
Example: Unweighted acceptor

Wordlist

```
clear
clever
ear
ever
fat
father
```

compile

Network



```
/usr/dict/words 0.6 sec
25K words
206K chars
```

compile

```
FSM
17728 states,
37100 arcs
```

slide courtesy of L. Karttunen (modified)

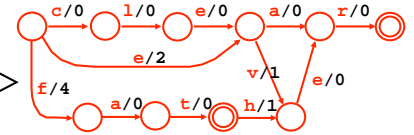
Example: Weighted acceptor

Wordlist

```
clear 0
clever 1
ear 2
ever 3
fat 4
father 5
```

compile

Network



Computes a perfect hash!

slide courtesy of L. Karttunen (modified)

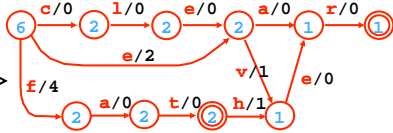
Example: Weighted acceptor

Wordlist

```
clear 0
clever 1
ear 2
ever 3
fat 4
father 5
```

compile

Network



- Compute number of paths from each state (Q: how?)
- Successor states partition the path set
- Use offsets of successor states as arc weights
- Q: Would this work for an arbitrary numbering of the words?

Example: Unweighted transducer

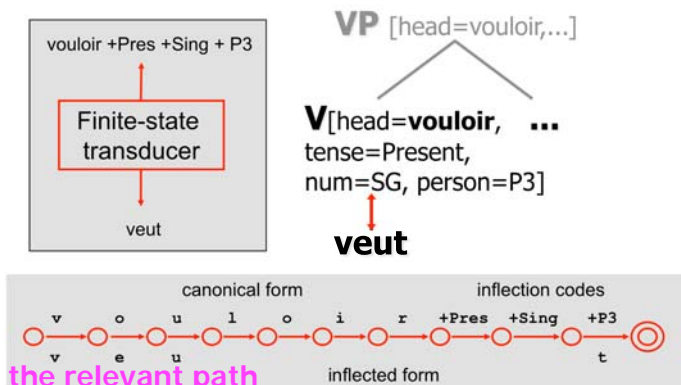
VP [head=vouloir,...]

V[head=vouloir, ...
tense=Present,
num=SG, person=P3]

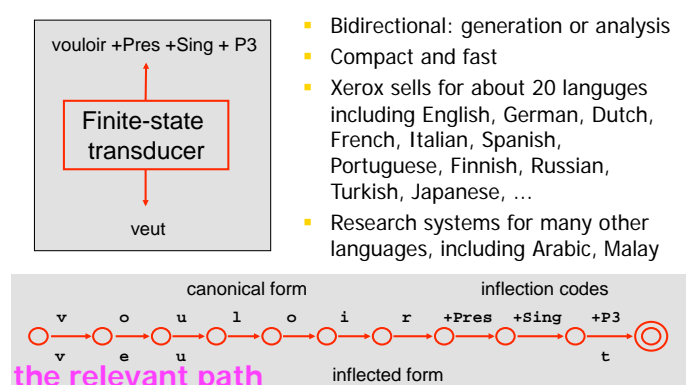
veut

the problem of morphology ("word shape") - an area of linguistics

Example: Unweighted transducer

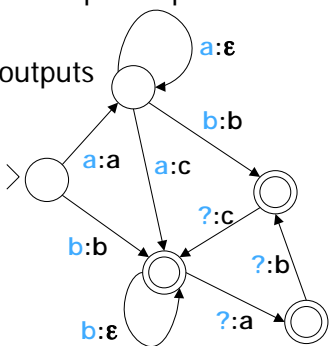


Example: Unweighted transducer



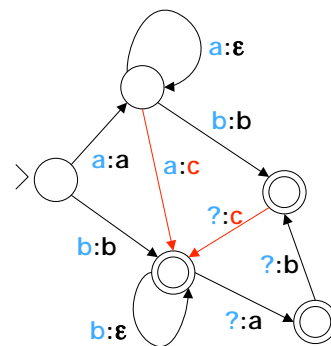
Regular Relation (of strings)

- Relation: like a function, but multiple outputs ok
- Regular: finite-state
- Transducer: automaton w/ outputs
- $b \rightarrow \{b\}$ $a \rightarrow \{\}$
- $aaaaa \rightarrow \{ac, aca, acab, acabc\}$
- Invertible?
- Closed under composition?

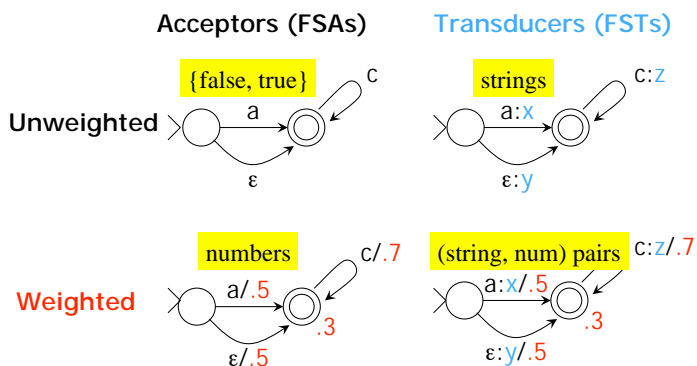


Regular Relation (of strings)

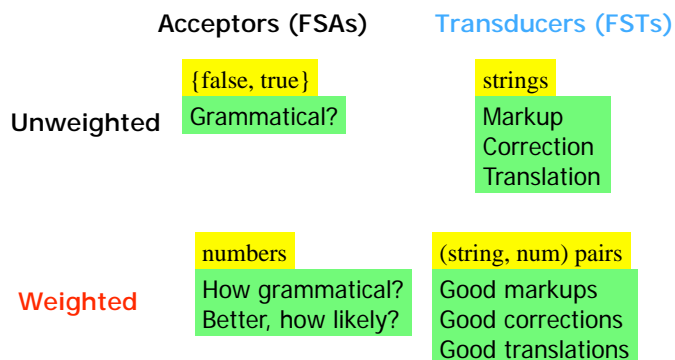
- Can weight the arcs: \rightarrow vs. \rightarrow
- $b \rightarrow \{b\}$ $a \rightarrow \{\}$
- $aaaaa \rightarrow \{ac, aca, acab, acabc\}$
- How to find best outputs?
 - For $aaaaa$?
 - For all inputs at once?



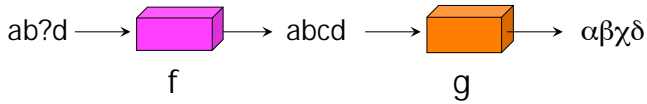
Function from strings to ...



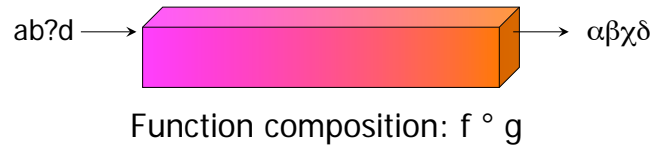
Sample functions



Functions

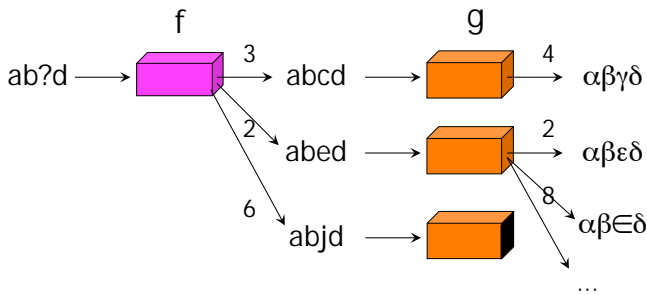


Functions

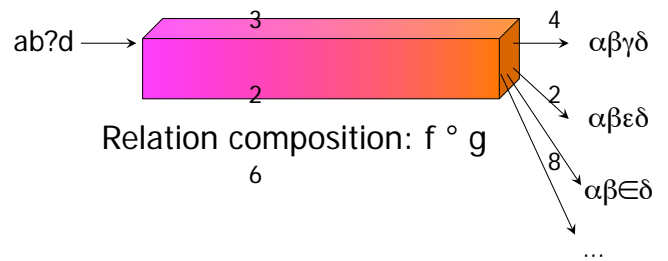


[first f, then g – intuitive notation, but opposite of the traditional math notation]

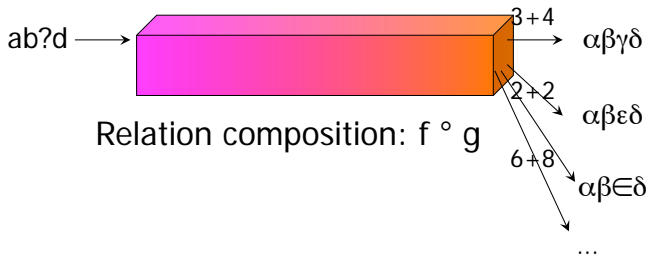
From Functions to Relations



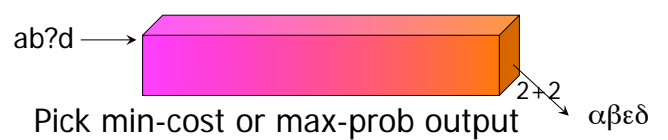
From Functions to Relations



From Functions to Relations

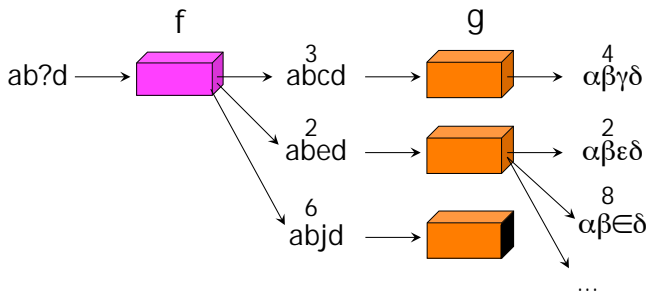


From Functions to Relations

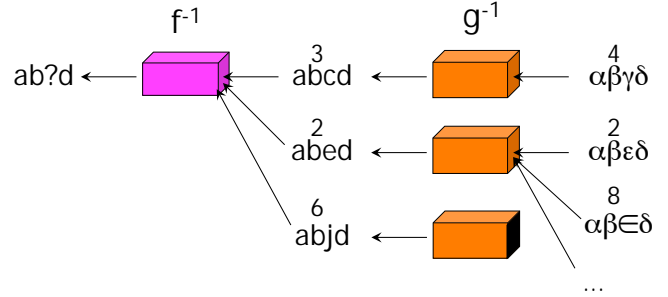


Often in NLP, all of the functions or relations involved can be described as finite-state machines, and manipulated using standard algorithms.

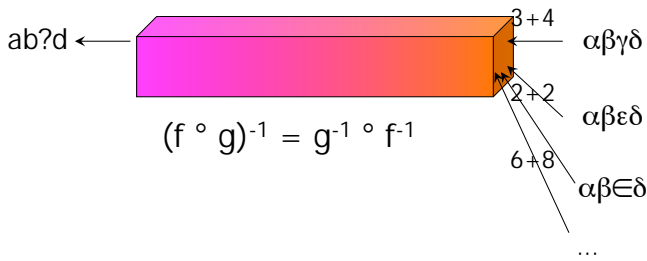
Inverting Relations



Inverting Relations

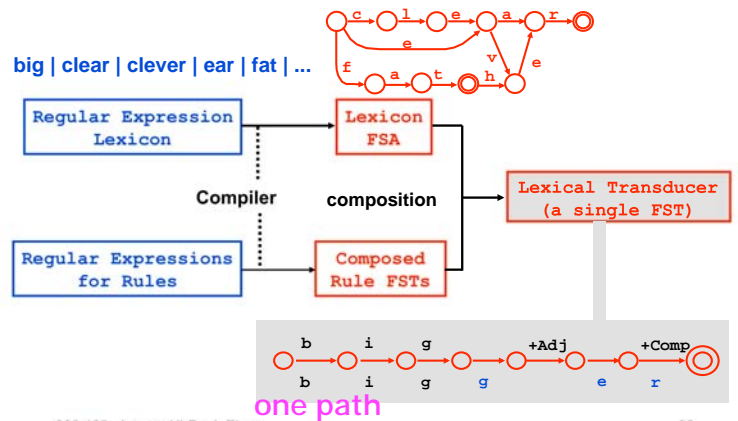


Inverting Relations



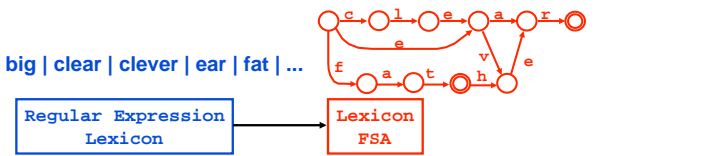
slide courtesy of L. Karttunen (modified)

Building a lexical transducer



slide courtesy of L. Karttunen (modified)

Building a lexical transducer



- Actually, the lexicon must contain elements like **big +Adj +Comp**
- So write it as a more complicated expression: **(big | clear | clever | fat | ...) +Adj (ε | +Comp | +Sup)** ← *adjectives*
(ear | father | ...) +Noun (+Sing | +Pl) ← *nouns*
 | ... ← ...
- Q: Why do we need a lexicon at all?

slide courtesy of L. Karttunen (modified)

Weighted version of transducer: Assigns a weight to each string pair

