Example Applications of Finite State Machines

Data structures and algorithms for Computational Linguistics III

Çağrı Çöltekin
ccoltekin@sfs.uni-tuebingen.de

University of Tübingen
Seminar für Sprachwissenschaft

Winter Semester 2018–2019

Applications of finite-state methods

- Finite state methods are attractive for formal and computational reasons
- They are applied in a vast diversity of fields
  - Electronic circuit design
  - Workflow management
  - Games
  - Pattern matching
  - Tokenization, stemming
  - Morphological analysis
  - Chunking
  - …
- This lecture
  - FSA for pattern matching
  - FSA for storing a lexicon
  - Finite-state morphology

Finite state automata

a refresher

- An FSA recognizes and generates a regular language, also equivalent to regular expressions
- FSA are closed under
  - Concatenation
  - Kleene star
  - Union
  - Intersection
  - Complement
  - Reversal
- Two types:
  - DFA single transition from each state on each input symbol
  - NFA transitions to possibly multiple states on a single input symbol, or without consuming an input symbol (ϵ-NFA)
- Every FSA has a unique minimal DFA
  - For every NFA there is a DFA that accepts the same regular language (determinization)
  - A DFA can be minimized to equivalent DFA with minimum nodes (minimization)

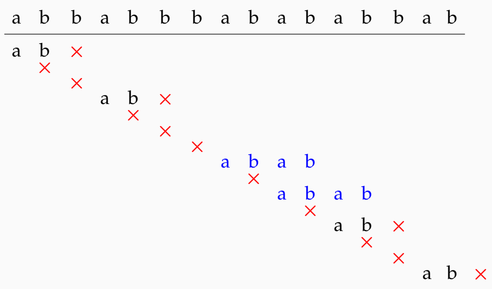
Finite state transducers

a refresher

- FST transitions are defined on a pair of input–output symbols
- An FST moves between the states on the input symbol, while outputting the output symbol
- FSTs define a regular relation
- FSTs are closed under
  - Concatenation
  - Kleene star
  - Union
  - Reversal
  - Inversion
  - Composition
- Not all FSTs can be determinized

Naive string match

Example: searching 'abab' in 'abbabbbababbab'

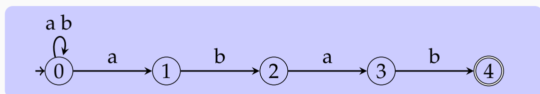


Note the wasted effort after a partial match.

String matching with an NFA

Another solution

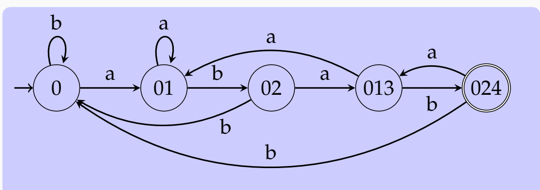
Consider running the following NFA over the string.



- The NFA will be in the accepting state when last four letters processed matches `abab` (including overlapping matches)
- Is this faster than the naive algorithm?

DFA version

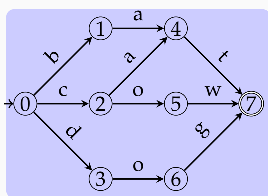
Knuth-Morris-Pratt (KMP) algorithm



- DFA processes every input symbol only once
- The resulting DFA has the same number of states (generally, not much larger than the NFA)
- Approach generalizes to arbitrary regular expressions without additional computational cost

Finite state lexicons

- FSA are an efficient way to store lexicons
- One can start from NFA for individual words, and minimize/determinize the union of them
- Or there are algorithms for constructing finite-state lexicons incrementally



Morphology
some definitions

- **Morpheme** is an abstract linguistic unit, often defined as smallest meaningful or grammatical unit. Morphemes make up words
- **Root** of a word is a *free* morpheme, often carrying the semantic information
- **Derivational** morphemes change the meaning of a word, sometimes changing the POS
- **Inflectional** morphemes change the syntactic properties of words
- **Lemma** of a word is its 'citation' form, what you look up in a lexicon
- **Stem** of a (possibly derived) word is the common string shared by all morphologically related forms

Morphological typology

Languages of the world behave differently with respect to how words are formed.

- *Isolating* languages have little or no morphology, all words are simple (e.g., Vietnamese, Chinese)
- *Analytic* languages have little or no inflectional morphology (e.g., English)
- *Synthetic* languages have rich morphological system
  - In *agglutinative* languages each morpheme has a single function (e.g., Finnish, Turkish)
  - In *inflecting/fusional* a single morpheme indicates multiple functions (e.g., Latin, Russian)
  - *Polysynthetic* languages may pack multiple 'words' in a single word (e.g., Ainu, Chukchi)

Note that these are tendencies.

Where do morphemes go

- Affixation:
  attach → ***un**-attach-**ed***
- Infixes:
  aussteigen → *aus**zu**steigen*
- Circumfixation:
  spiel → ***ge**spiel**t***
- Root-pattern morphology:
  ktb → *kitāb* 'book'
  ktb → *kātib* 'writer'
  (Arabic)
- Reduplication:
  orang 'person' → *orang-orang* 'people'
  (some Austronesian languages)

Interaction of morphology and phonology
or morphology and orthography

Morphology and phonology/orthography interact. A few examples:

- *dog-**s**, but fox-**es***
- *cit**y*** →*cit**i**-es*
- *stop* →*stop**p**ing*
- *panic* →*panic**k**-ed*
- *goose* →*geese*
- Vowel harmony
  ev 'house' → *ev-l**e**r* 'houses'
  oda 'room' → *oda-l**a**r* 'rooms'
  (Turkish)

Two-level morphology

- We assume that there are two 'levels' of representation
  - A *surface* representation which is what we hear or see
  - An *underlying*, an abstract representation for the word

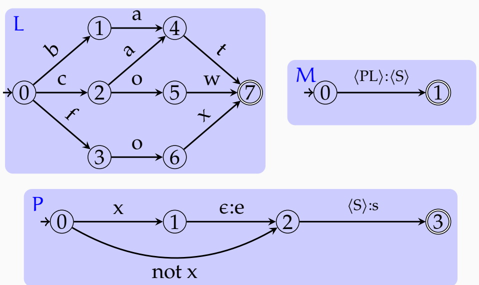
Surface:	cat	s
Underlying:	cat	⟨PL⟩

- An FST is used to map the underlying representation to the surface representation (generation)
- If we run the FST in the inverse direction, we get an analysis
- Often the FST is a complex combination of many small FSA or FSTs

Two-level morphology
a typical architecture

- Typically, lexicon is converted to FSA
- Concatenated (or composed) with morphological rules (affixation, applying templates, …)
- The result is composed with phonological/orthographic alternations
- The phonological/orthographic rules can be designed as cascades (composition), or can be applied in parallel

Two-level morphology
a (simplified) example



Generator: LM ∘ P   Analyzer: $(\mathrm{LM} \circ \mathrm{P})^{-1}$

How to specify morphological analyzers

- Lexicons are easiest to specify as lists of (root) words
  cat
  dog
  fox
  …
- For affixation, regular expressions (or regular rewrite rules)
  $\mathrm{N}_{\mathtt{plu}}$ → N ⟨PL⟩:⟨S⟩
- For phonological/orthographic alternations context sensitive rules
  ⟨S⟩ →es / x _
- There are a few standard languages for specifying morphological analyzers
  - SFST
  - Xerox languages: XFST, Twolc, lexc
  - OpenFST OpenGRM (more general purpose)

XFST

A quick reference some common notation/operations

`?`	any symbol
`0`	empty string (ϵ)
`(a)`	optional a
`[a|b]`	grouping
`a*`	Kleene start
`a+`	Kleene plus
`a b`	concatenation
`a&b`	intersection
`a|b`	union
`~b`	complement
`a-b`	difference
`{cat}`	concatenation of `c a t`
`a:b`	FST rule with input ‘a’ and output ‘b’
`a .o. b`	compose a with b
`a -> b`	unconditionally replace a to b

XFST (cont.)

A quick reference some common notation/operations

`a (->)b`	optionally replace a to b
`a -> b || c _`	replace a to b only after c
`a -> b || c _ d`	replace a to b only after c and before d

- There are (at least) two free implementations of xfst
  - Foma
  - hfst-xfst (part of HFST)
- You will receive a separate ‘tutorial’ (and an exercise) on working with xfst and lexc

Tools of the trade

Some of the practical, feely-available, tools (with an emphasis on ones targeted for CL) include:

- Gertjan van Noord’s FSA tools
- OpenFST: a general purpose finite state library
- Helsinki finite-state technology (HFST): library tools from University of Helsinki
- Foma: a re-implementation of Xerox’s xfst, a language/toolbox for defining/manipulating FST
- SFST another language/toolbox for defining/manipulating FSTs

Wrapping up

- Finite-state tools are commonly used in a number of CL task
- There are off-the-shelf free tools

Next:

- Dependency grammars and dependency parsing
- Constituency parsing

References / additional reading material

- Jurafsky and Martin (2009, Ch. 3)
- Roche and Schabes (1997) includes more examples of FSTs used for NLP
- The Xerox languages and tools are described in Beesley and Karttunen (2003)
- HFST and Foma web pages include some documentation and (links to) tutorials

References / additional reading material (cont.)

Beesley, Kenneth R. and Lauri Karttunen (2003). “Finite-state morphology: Xerox tools and techniques”. In: *CSLI, Stanford*.

Hulden, Mans (2009). “Foma: a finite-state compiler and library”. In: *Proceedings of the 12th Conference of the European Chapter of the Association for Computational Linguistics*. Association for Computational Linguistics, pp. 29–32.

Jurafsky, Daniel and James H. Martin (2009). *Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition*. second. Pearson Prentice Hall. ISBN: 978-0-13-504196-3.

Lindén, Krister, Erik Axelson, Senka Drobac, Sam Hardwick, Juha Kuokkala, Jyrki Niemi, Tommi A. Pirinen, and Miikka Silfverberg (2013). “HFST — A System for Creating NLP Tools”. In: *Systems and Frameworks for Computational Morphology*. Ed. by Cerstin Mahlow and Michael Piotrowski. Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 53–71.

Roche, Emmanuel and Yves Schabes (1997). *Finite-state Language Processing*. A Bradford book. MIT Press. ISBN: 9780262181822.