

# Finite state transducers

Data Structures and Algorithms for Computational Linguistics III  
(ISCL-BA-07)

Çağrı Çöltekin

`ccoltekin@sfs.uni-tuebingen.de`

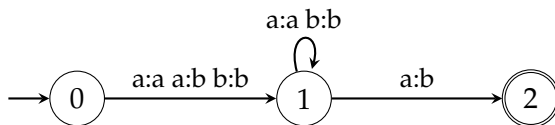
University of Tübingen  
Seminar für Sprachwissenschaft

Winter Semester 2021/22

# Finite state transducers

## A quick introduction

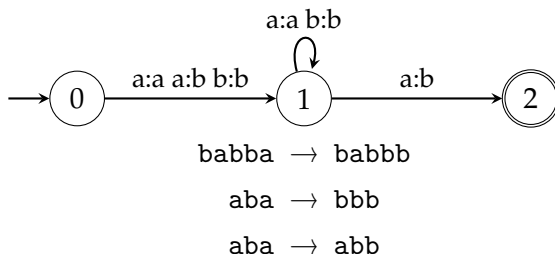
- A *finite state transducer* (FST) is a finite state machine where transitions are conditioned on pairs of symbols
- The machine moves between the states based on an *input* symbol, while it outputs the corresponding *output* symbol
- An FST encodes a *relation*, a mapping from a set to another
- The relation defined by an FST is called a *regular* (or *rational*) relation



# Finite state transducers

## A quick introduction

- A *finite state transducer* (FST) is a finite state machine where transitions are conditioned on pairs of symbols
- The machine moves between the states based on an *input* symbol, while it outputs the corresponding *output* symbol
- An FST encodes a *relation*, a mapping from a set to another
- The relation defined by an FST is called a *regular* (or *rational*) relation



# Formal definition

A finite state transducer is a tuple  $(\Sigma_i, \Sigma_o, Q, q_0, F, \Delta)$

$\Sigma_i$  is the *input* alphabet

$\Sigma_o$  is the *output* alphabet

$Q$  a finite set of states

$q_0$  is the start state,  $q_0 \in Q$

$F$  is the set of accepting states,  $F \subseteq Q$

$\Delta$  is a relation  $(\Delta : Q \times \Sigma_i \rightarrow Q \times \Sigma_o)$

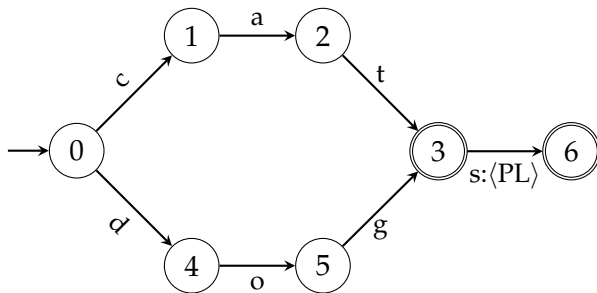
# Where do we use FSTs?

## Uses in NLP/CL

- Morphological analysis
- Spelling correction
- Transliteration
- Speech recognition
- Grapheme-to-phoneme mapping
- Normalization
- Tokenization
- POS tagging (not typical, but done)
- partial parsing / chunking
- ...

# Where do we use FSTs?

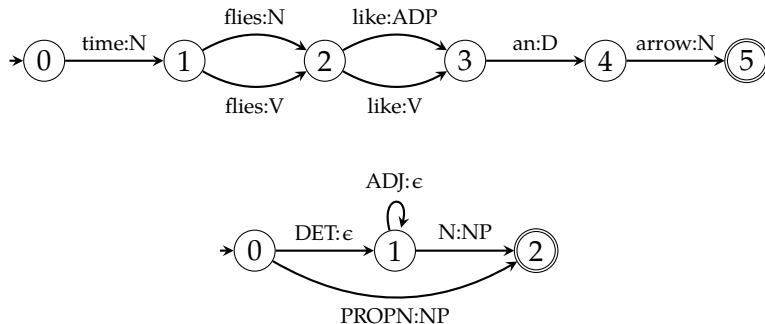
example 1: morphological analysis



In this lecture, we treat an FSA as a simple FST that outputs its input:  
the edge label 'a' is a shorthand for 'a:a'.

# Where do we use FSTs?

example 2: POS tagging / shallow parsing



Note: (1) It is important to express the ambiguity. (2) This gets interesting if we can 'compose' these automata.

# Closure properties of FSTs

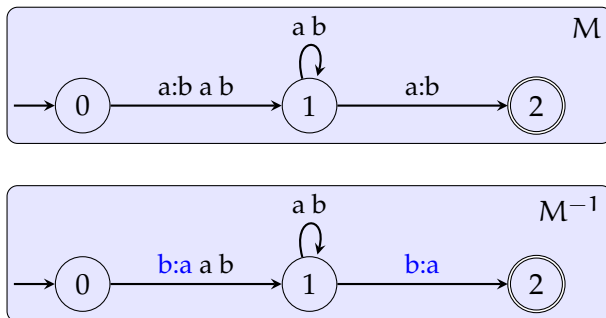
Like FSA, FSTs are closed under some operations.

- Concatenation
- Kleene star
- Complement
- Reversal
- Union
- Intersection
- *Inversion*
- *Composition*



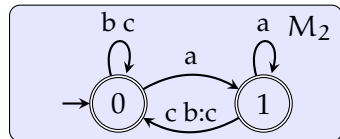
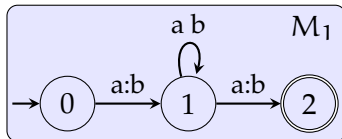
# FST inversion

- Since an FST encodes a relation, it can be reversed
- Inverse of an FST swaps the input symbols with output symbols
- We indicate inverse of an FST  $M$  with  $M^{-1}$



# FST composition

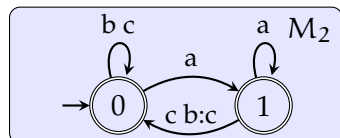
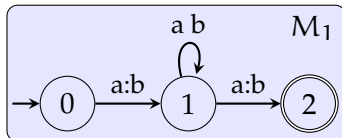
sequential application



$aa \xrightarrow{M_1} \rightarrow$   
 $bb \xrightarrow{M_1} \rightarrow$   
 $aaaa \xrightarrow{M_1} \rightarrow$   
 $abaa \xrightarrow{M_1} \rightarrow$

# FST composition

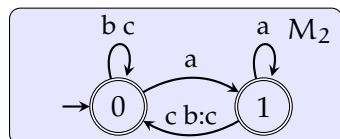
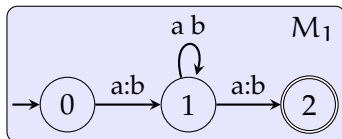
## sequential application



$aa$	$\xrightarrow{M_1}$	$bb$	$\xrightarrow{M_2}$
$bb$	$\xrightarrow{M_1}$	$\emptyset$	$\xrightarrow{M_2}$
$aaaa$	$\xrightarrow{M_1}$	$baab$	$\xrightarrow{M_2}$
$abaa$	$\xrightarrow{M_1}$	$bbab$	$\xrightarrow{M_2}$

# FST composition

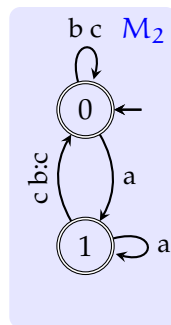
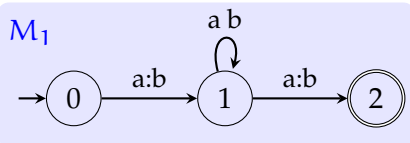
## sequential application



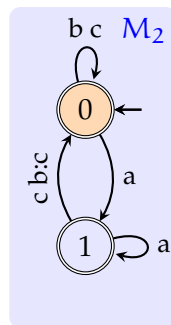
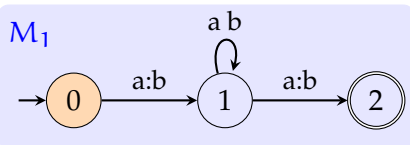
$M_1 \circ M_2$				
aa	$\xrightarrow{M_1}$	bb	$\xrightarrow{M_2}$	bb
bb	$\xrightarrow{M_1}$	$\emptyset$	$\xrightarrow{M_2}$	$\emptyset$
aaaa	$\xrightarrow{M_1}$	baab	$\xrightarrow{M_2}$	baac
abaa	$\xrightarrow{M_1}$	bbab	$\xrightarrow{M_2}$	bbac

- Can we compose two FSTs without running them sequentially?

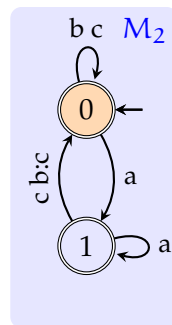
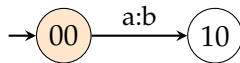
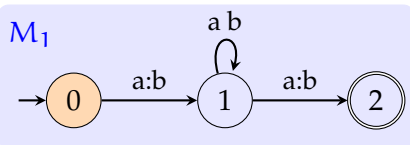
# FST composition



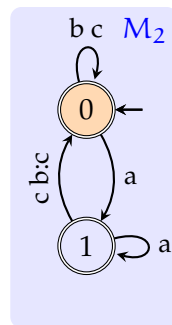
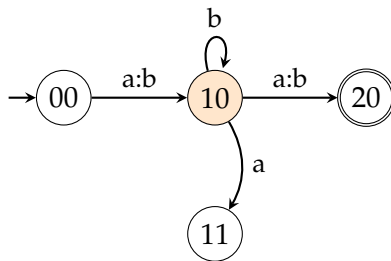
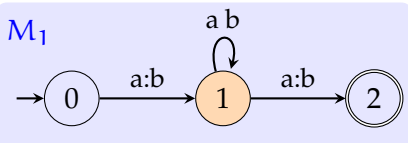
# FST composition



# FST composition

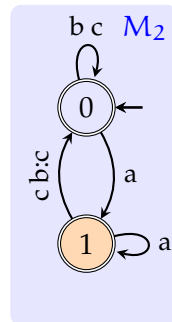
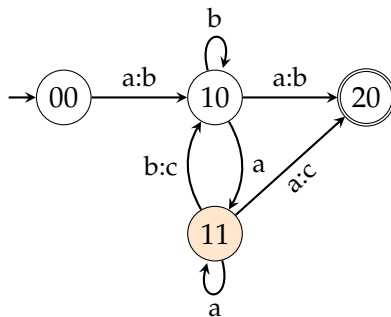
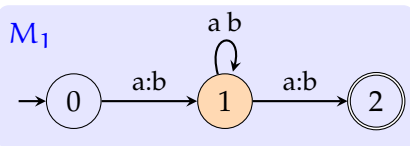


# FST composition



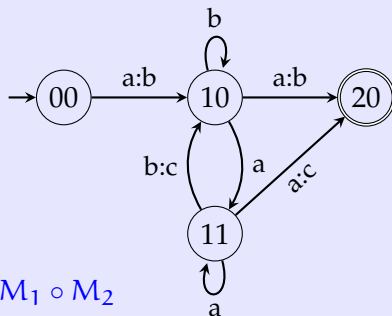
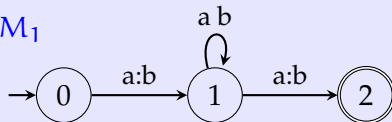


# FST composition

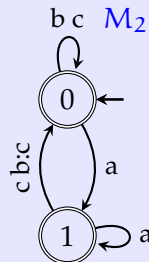


# FST composition

$M_1$



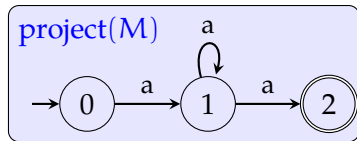
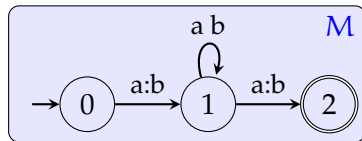
$M_1 \circ M_2$



$M_2$

# Projection

- Projection* turns an FST into a FSA, accepting either the input language or the output language



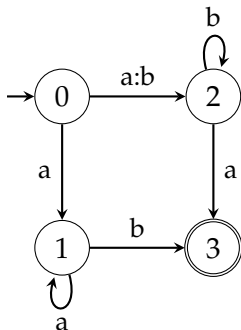
# FST determinization

- A deterministic FST has unambiguous transitions from every state on any *input* symbol
- We can extend the subset construction to FSTs
- Determinization of FSTs means converting to a *subsequential* FST
- However, not all FSTs can be determinized

# FST determinization

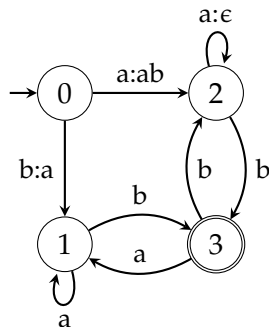
- A deterministic FST has unambiguous transitions from every state on any *input* symbol
- We can extend the subset construction to FSTs
- Determinization of FSTs means converting to a *subsequential* FST
- However, not all FSTs can be determinized

Is this FST deterministic?



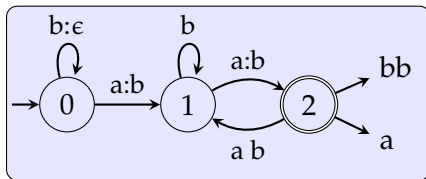
# Sequential FSTs

- A sequential FST has a single transition from each state on every *input* symbol
- Output symbols can be strings, as well as  $\epsilon$
- The recognition is linear in the length of input
- However, sequential FSTs do not allow ambiguity



## Subsequential FSTs

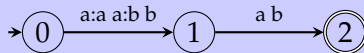
- A *k*-subsequential FST is a sequential FST which can output up to *k* strings at an accepting state
- Subsequential transducers allow limited ambiguity
- Recognition time is still linear



- The 2-subsequential FST above maps every string it accepts to two strings, e.g.,
  - $baa \rightarrow bba$
  - $baa \rightarrow bbbb$

# An exercise

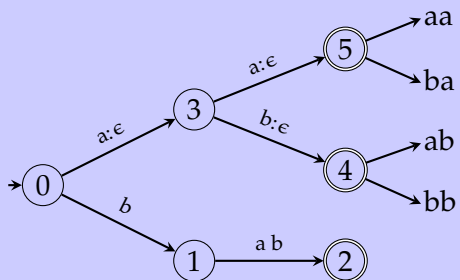
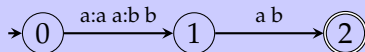
Convert the following FST to a subsequential FST





# An exercise

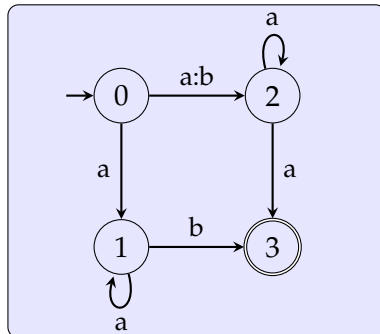
Convert the following FST to a subsequential FST



# Determinizing FSTs

## Another example

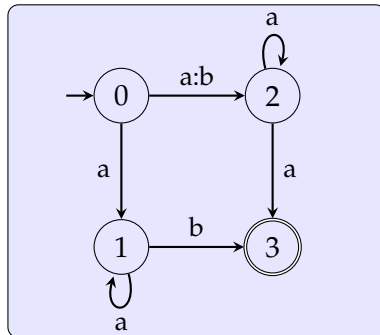
Can you convert the following FST to a subsequential FST?



# Determinizing FSTs

## Another example

Can you convert the following FST to a subsequential FST?



Note that we cannot ‘determine’ the output on first input, until reaching the final input.





# FSA vs FST

- FSA are *acceptors*, FSTs are *transducers*
- FSA accept or reject their input, FSTs produce output(s) for the inputs they accept
- FSA define sets, FSTs define relations between sets
- FSTs share many properties of FSAs. However,
  - FSTs are not closed under intersection and complement
  - We can compose (and invert) the FSTs
  - Determinizing FSTs is not always possible
- Both FSA and FSTs can be weighted (not covered in this course)

# References / additional reading material

- Jurafsky and Martin (2009, Ch. 3)
- Additional references include:
  - Roche and Schabes (1996) and Roche and Schabes (1997): FSTs and their use in NLP
  - Mohri (2009): weighted FSTs

## References / additional reading material (cont.)

-  Jurafsky, Daniel and James H. Martin (2009). *Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition*. second edition. Pearson Prentice Hall. ISBN: 978-0-13-504196-3.
-  Mohri, Mehryar (2009). “Weighted automata algorithms”. In: *Handbook of Weighted Automata*. Monographs in Theoretical Computer Science. Springer, pp. 213–254.
-  Roche, Emmanuel and Yves Schabes (1996). *Introduction to Finite-State Devices in Natural Language Processing Technical Report*. Tech. rep. TR96-13. Mitsubishi Electric Research Laboratories. URL:  
<http://www.merl.com/publications/docs/TR96-13.pdf>.
-  — (1997). *Finite-state Language Processing*. A Bradford book. MIT Press. ISBN: 9780262181822.









