Ling/CSE 472: Introduction to Computational Linguistics

4/6/17: Morphology & FST 2

Overview

- Review: FSAs & FSTs
- XFST
- xfst demo
- Examples of FSTs for spelling change rules
- Reading questions

Review: FSAs and FSTs

- FSAs define sets of strings (regular languages).
- FSTs define sets of ordered pairs of strings (regular relations).
- Formally interesting because not all languages/relations can be defined by FSAs/FSTs.
- Are all finite languages and relations regular?
- Linguistically interesting because:
    - FSAs have enough power for morphotactics.
    - FSTs have (almost) enough power for morphophonology.
    - Both are very efficient.

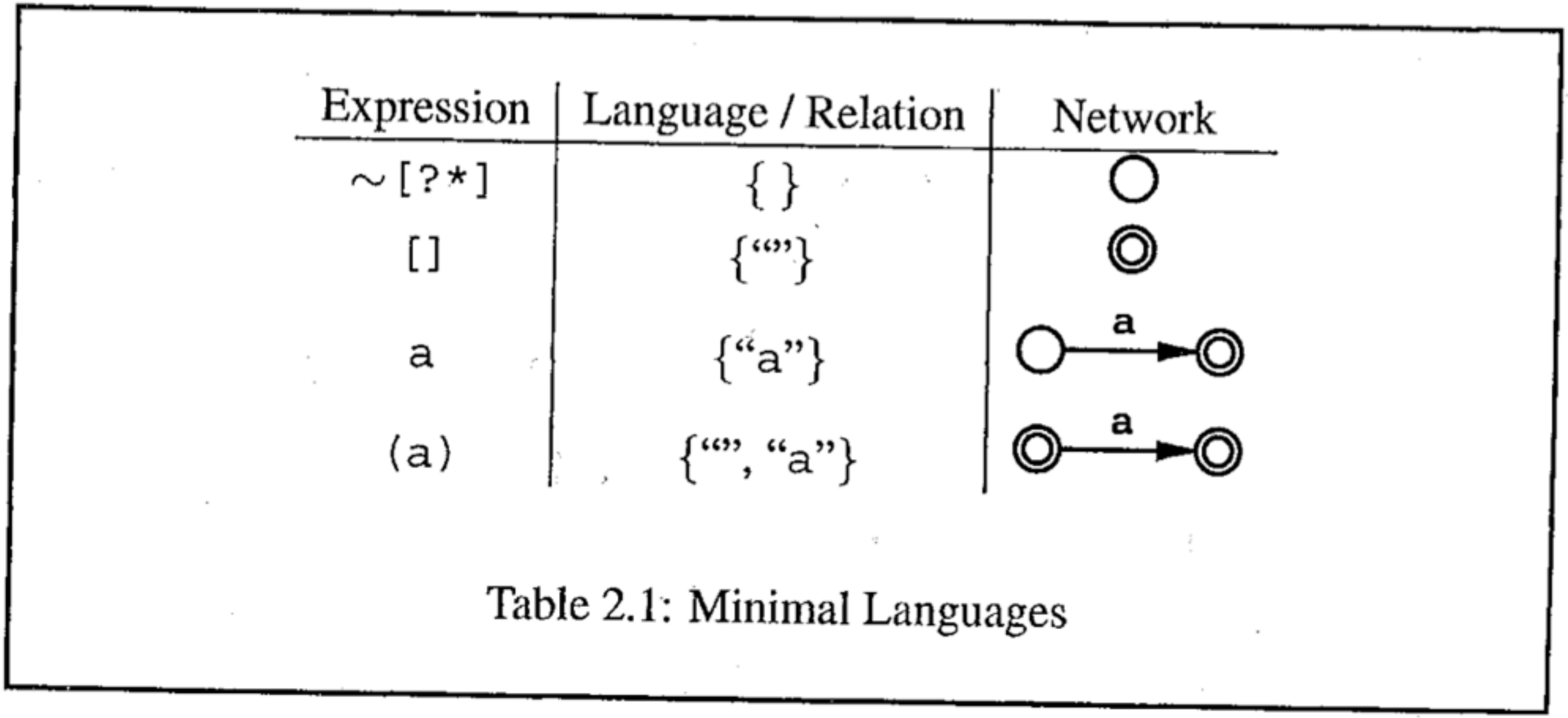
FSTs: “Quiz”

- Why do FSTs have complex symbols labeling the arcs?
- What happens if you give an FST an input on only one “tape”?
- What happens if the input has symbols outside the FST’s alphabet?
- Do the upper and lower tape strings always have the same length?

xfst regex syntax

- Why is it so different from what we see in J&M and elsewhere?
- Why are there so many operators?

B&K minimal languages



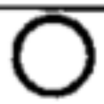
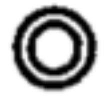
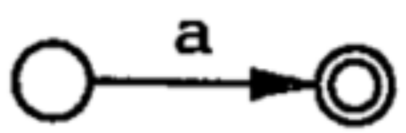
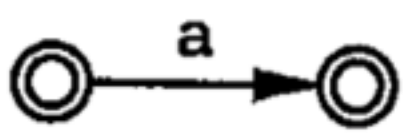
Expression	Language / Relation	Network
`~[?*]`	{ }	
`[]`	{“”}	
`a`	{“a”}	
`(a)`	{“”, “a”}	

Table 2.1: Minimal Languages

B&K: Iteration

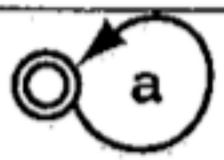
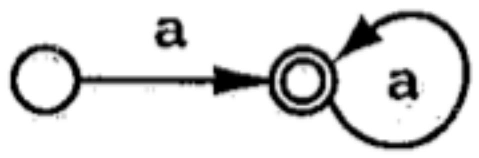
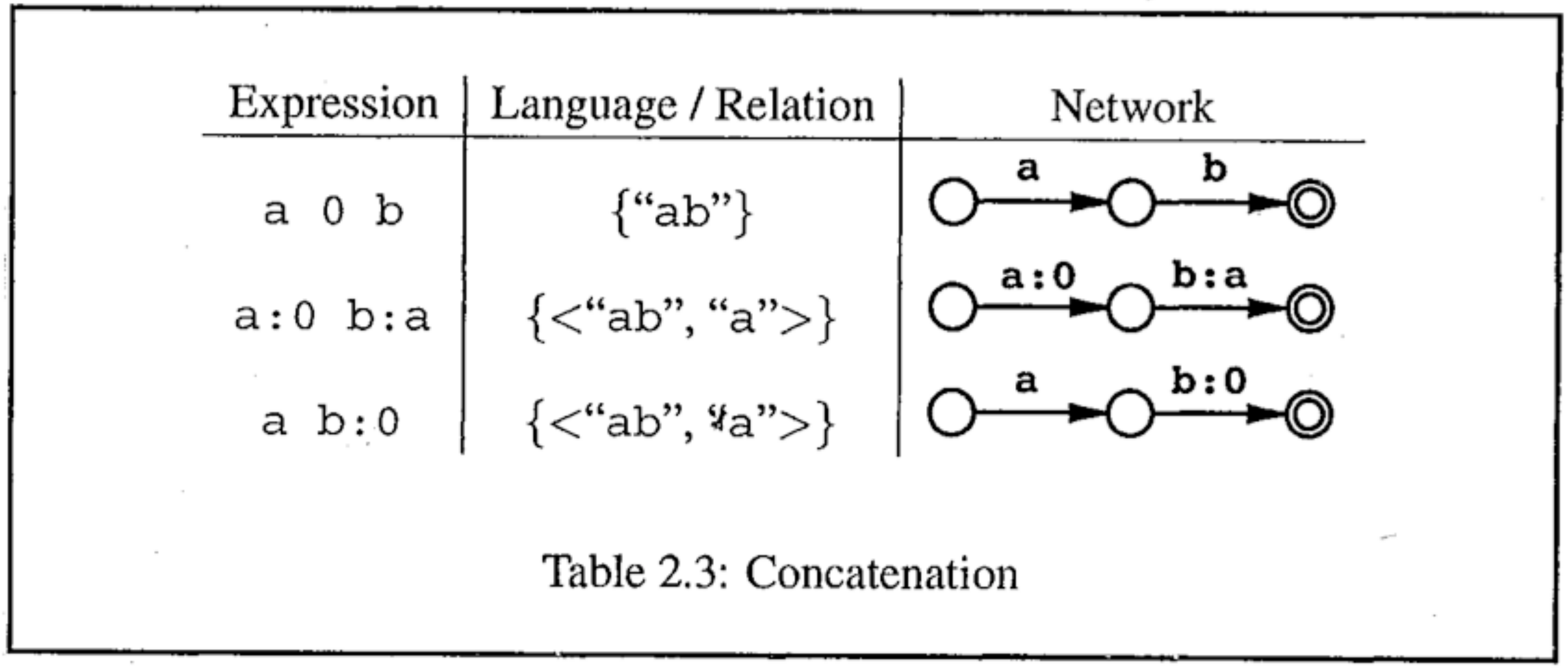
Expression	Language / Relation	Network
a*	{“”, “a”, “aa”, . . . }	
a+	{“a”, “aa”, . . . }	

Table 2.2: Iteration

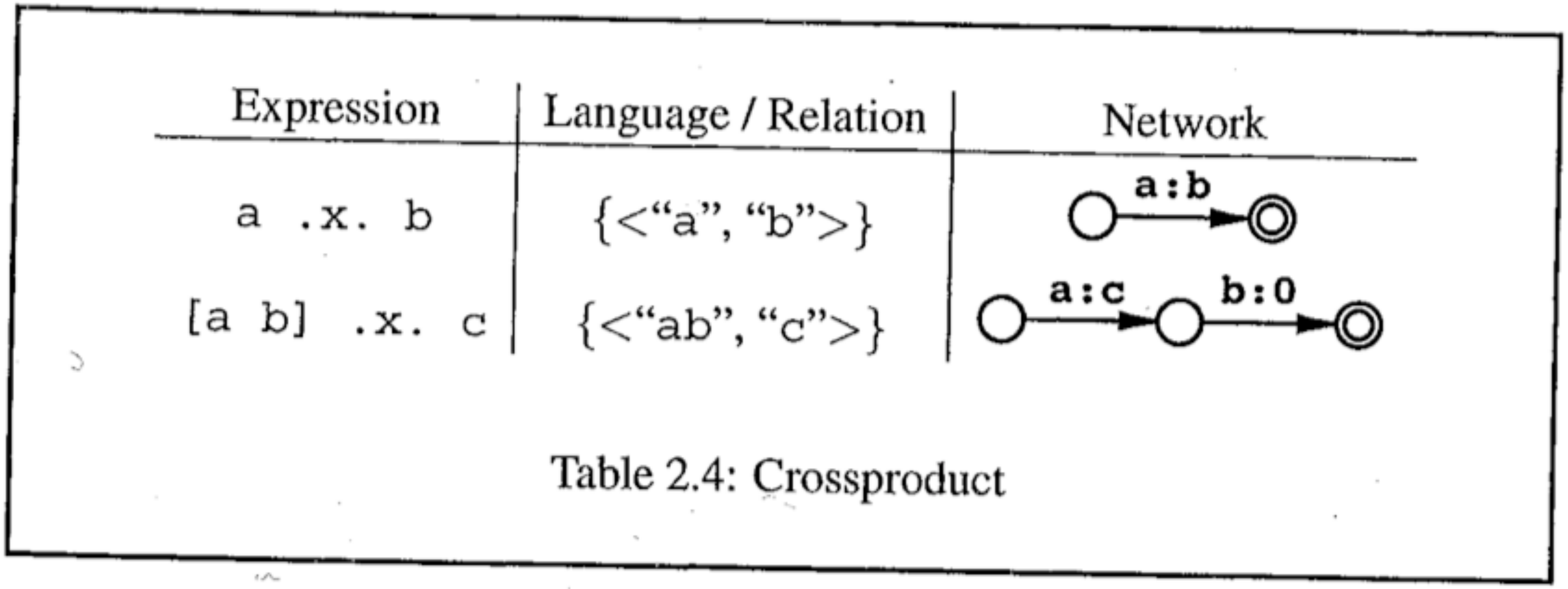
B&K: Concatenation



Expression	Language / Relation	Network
`a 0 b`	{“ab”}	$\begin{tikzcd} \bigcirc \arrow[r, "\mathbf{a}"] & \bigcirc \arrow[r, "\mathbf{b}"] & \circledcirc \end{tikzcd}$
`a:0 b:a`	{<“ab”, “a”>}	$\begin{tikzcd} \bigcirc \arrow[r, "\mathbf{a:0}"] & \bigcirc \arrow[r, "\mathbf{b:a}"] & \circledcirc \end{tikzcd}$
`a b:0`	{<“ab”, “a”>}	$\begin{tikzcd} \bigcirc \arrow[r, "\mathbf{a}"] & \bigcirc \arrow[r, "\mathbf{b:0}"] & \circledcirc \end{tikzcd}$

Table 2.3: Concatenation

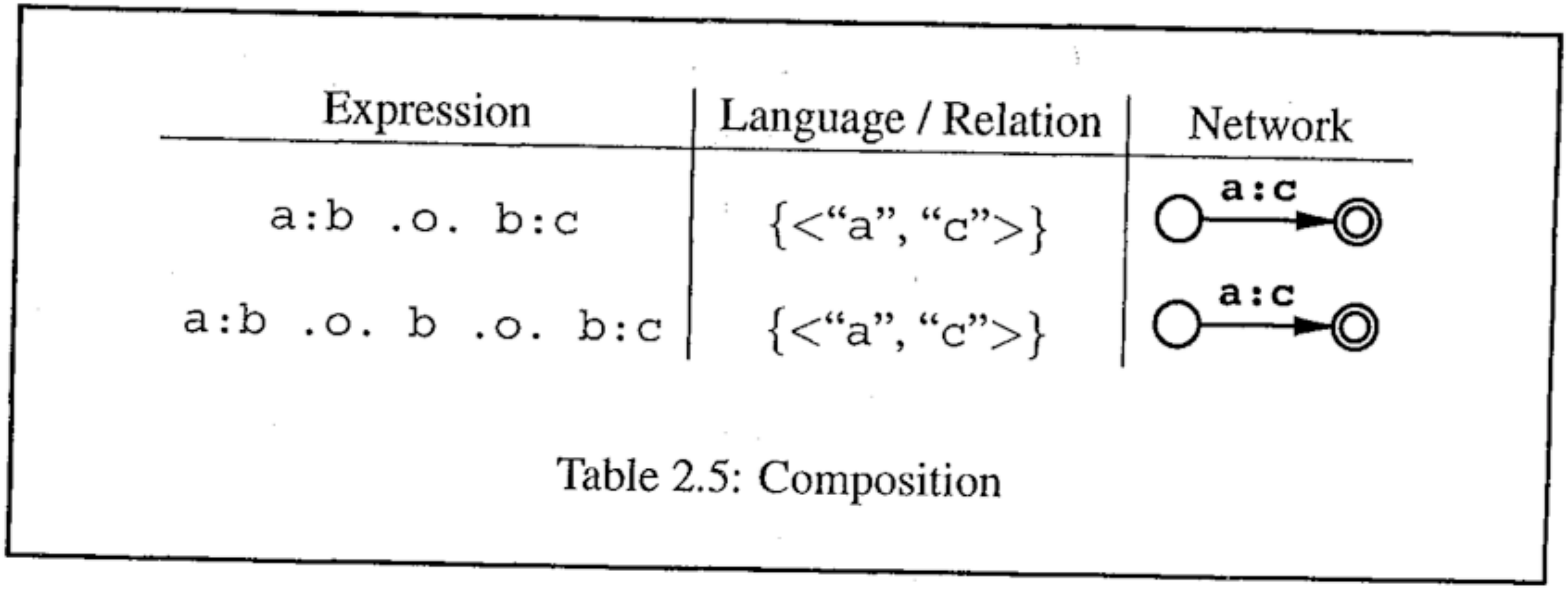
B&K: Cross-product



Expression	Language / Relation	Network
`a .x. b`	{<"a", "b">}	$\bigcirc \xrightarrow{a:b} \circledcirc$
`[a b] .x. c`	{<"ab", "c">}	$\bigcirc \xrightarrow{a:c} \bigcirc \xrightarrow{b:0} \circledcirc$

Table 2.4: Crossproduct

B&K: Composition



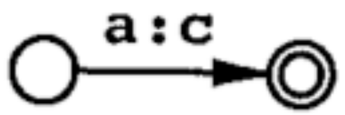
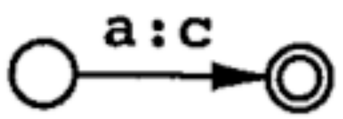
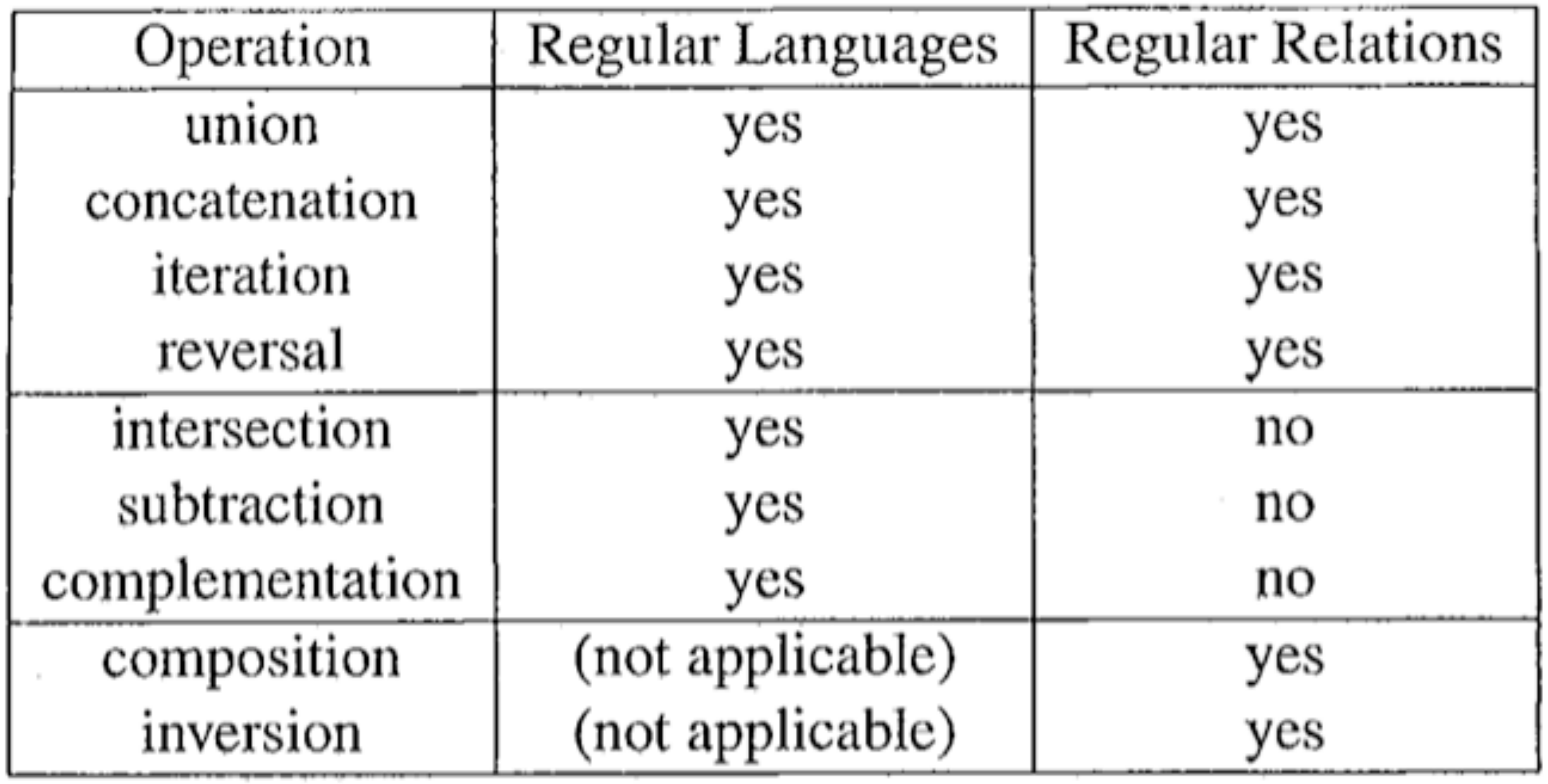
Expression	Language / Relation	Network
`a:b .o. b:c`	{<"a", "c">}	
`a:b .o. b .o. b:c`	{<"a", "c">}	

Table 2.5: Composition

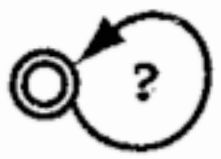
B&K: Closure



Operation	Regular Languages	Regular Relations
union	yes	yes
concatenation	yes	yes
iteration	yes	yes
reversal	yes	yes
intersection	yes	no
subtraction	yes	no
complementation	yes	no
composition	(not applicable)	yes
inversion	(not applicable)	yes

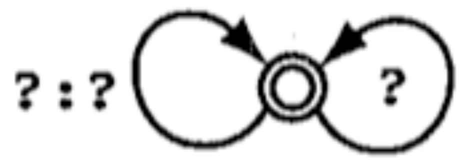
Table 2.7: Closure Properties

B&K: Universal relations



Sigma: {?}

Figure 2.6: The Universal Language/Identity Relation ?*



Sigma: {?}

Figure 2.7: The Universal Equal-Length Relation [?:?]*

B&K: Universal relations

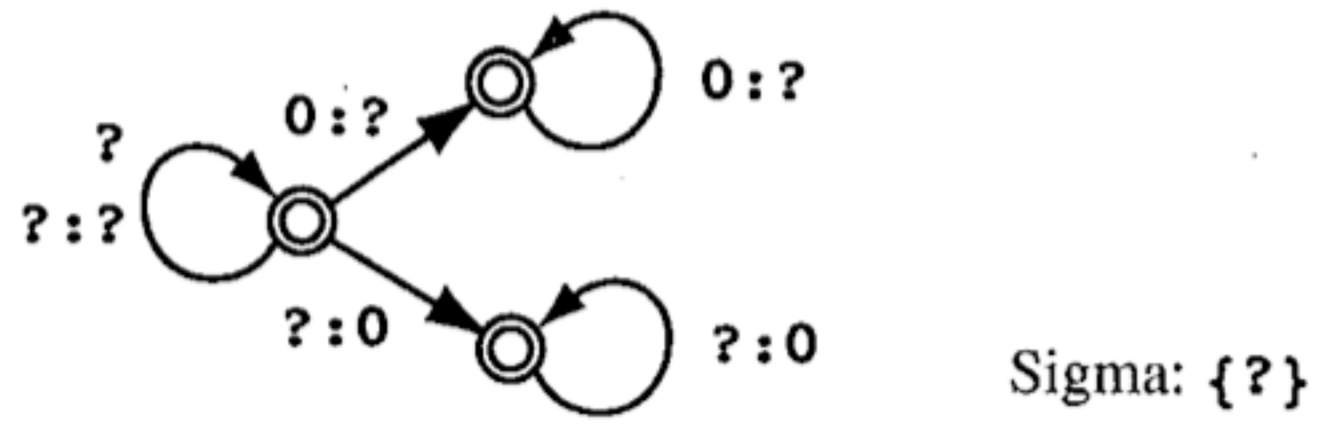


Figure 2.8: The Universal Relation `[?* .x. ?*]`

- Why are these useful?

B&K: Replacement

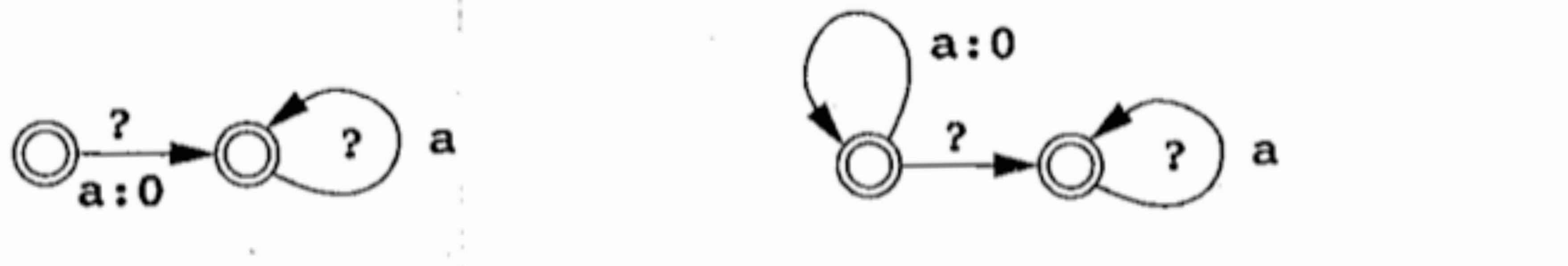
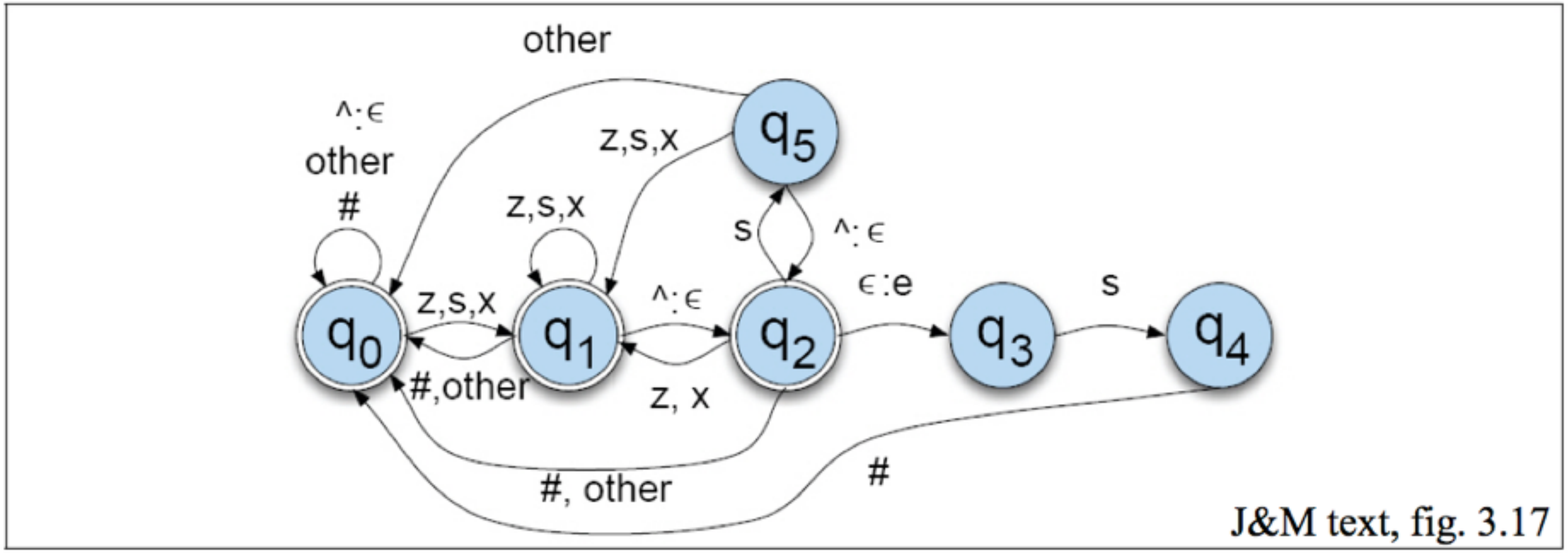


Figure 2.12: [a -> 0 || .#. _] vs. [a -> 0 // .#. _]

Overview

- Review: FSAs & FSTs
- XFST
- xfst demo
- Examples of FSTs for spelling change rules
- Reading questions

Sample e-insertion FST



- The idea is to add e only in the proper environments while letting all other sequences pass through.

Spelling change rule FST 1

```
define Rule1 [ ?* e:0 %+:0 [e|i] ?*];
```

- Draw an FST corresponding to Rule1.
- What are the upper and lower languages of Rule1?
- What linguistic work is this rule supposed to do?
- If the upper tape has expect+ed, what goes on the lower tape?

Spelling change rule FST 2

```
define Rule2 [[?* e:0 %+:0 [e|i] ?*] |
              [?* e %+:0 (\[e|i])] |
              [?* \e %+ ?*] |
              [\[%+]*]]
```

- What are the upper and lower languages of Rule2?
- What linguistic work is each part of this rule supposed to do?
- If the upper tape has expect+ed, what goes on the lower tape?
- If the upper tape has write+ing, what goes on the lower tape?

Overview

- Review: FSAs & FSTs
- XFST
- xfst demo
- Examples of FSTs for spelling change rules
- Reading questions

Reading questions

- They keep referencing "Xerox". However, I'm not sure I'm clear on what Xerox is. Do we need to know more than the fact that it is some compiler software that people use? Why is it particularly good for us to use? (Why is it what Beesley & Karttunen are using to explain with?)

- These Xerox regular expressions are quite different from the ones we previously saw in Perl. What are they special for? Is it because they operate on a set(=language) in stead of a string?

Reading questions

- What does the Sigma set mean in 2.3.4? What is the relationship between "Sigma" as used here and Σ as used to denote an alphabet in our previous description of regular languages?

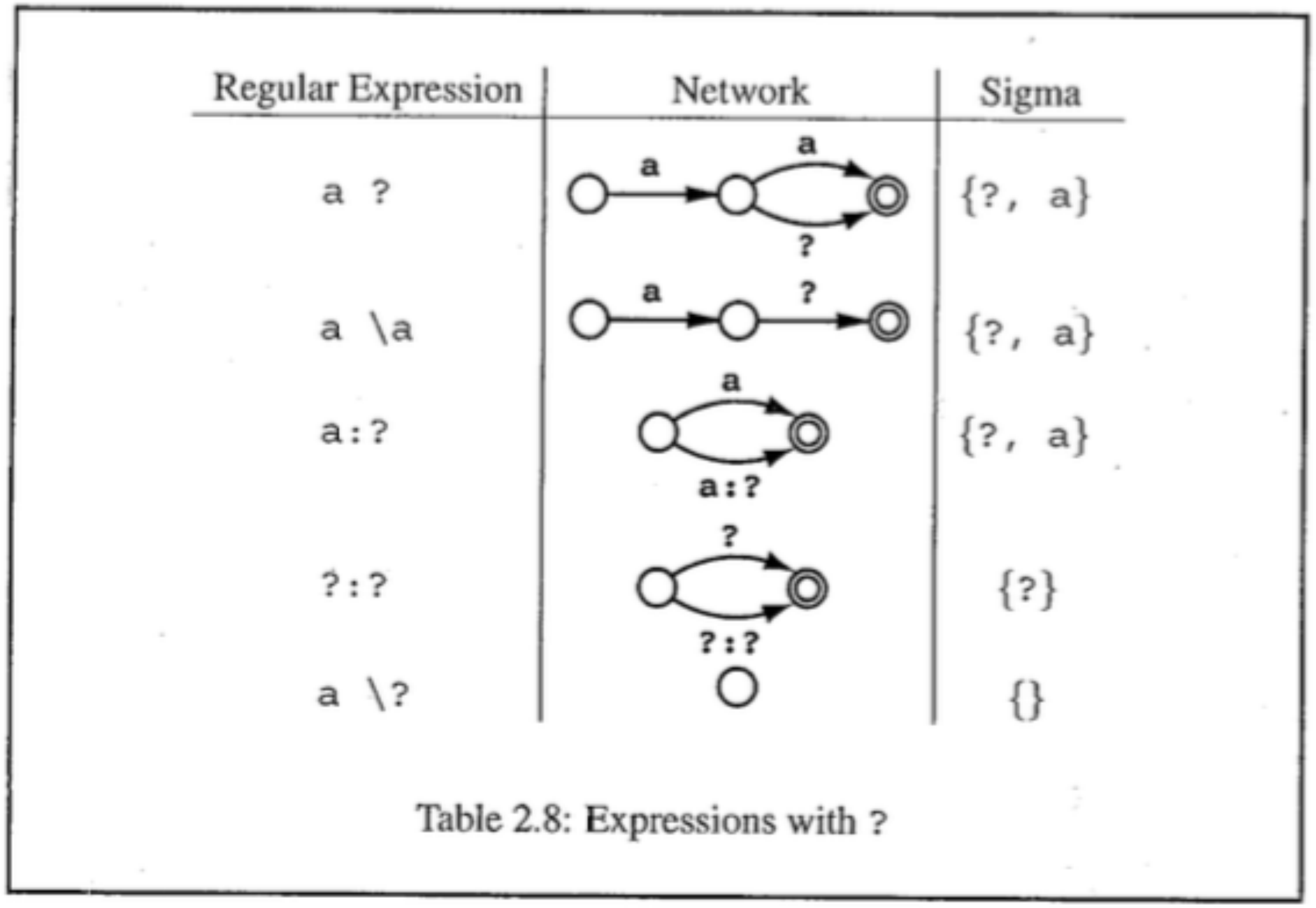
- Why does the ? mean different from what we are using in the regular expression as 0 or 1 occurrence of the character before it?

- Also, is the "." wildcard expression from the textbook the same as the "?" ANY symbol?

- What is the distinction between the ANY and UNKNOWN symbols described on page 57, and why is it a problem?

Reading questions

- In table 2.8, I'm not clear why 'a' wouldn't be included in '?' in the first line. From the text, "the compiler adds a redundant 'a' arc to the [?] to make explicit the fact that the string 'a' is included in the language". Does that mean that 'a b ?' would have three arcs between the last two nodes, and 'a b c d e f ?' would have seven?



Regular Expression	Network	Sigma
`a ?`	$\begin{tikzcd} \circ \arrow[r, "a"] & \circ \arrow[r, bend left, "a"] \arrow[r, bend right, "?"'] & \circledcirc \end{tikzcd}$	{?, a}
`a \a`	$\begin{tikzcd} \circ \arrow[r, "a"] & \circ \arrow[r, "?"] & \circledcirc \end{tikzcd}$	{?, a}
`a:?`	$\begin{tikzcd} \circ \arrow[r, bend left, "a"] \arrow[r, bend right, "a:?"'] & \circledcirc \end{tikzcd}$	{?, a}
`?:?`	$\begin{tikzcd} \circ \arrow[r, bend left, "?"] \arrow[r, bend right, "?:?"'] & \circledcirc \end{tikzcd}$	{?}
`a \?`	$\circ$	{}

Table 2.8: Expressions with ?

Reading questions

- Does [A:A]* mean and [A:A] and [A:A][A:A] and ... ? What is [A*:A*]?
- The reading states that A is a variable over regular expression. In the Brackets section, it writes that [A] denotes the same language or relation as A. Is A a language or a relation? What is a variable over a regular expression? I interpret A as a set of strings or ordered pairs of strings over the alphabet of a regular expression.
- How to interpret the statement that a syntactic ambiguity is semantically irrelevant for associative operation?
- What is the importance of explaining/specifying a language that consists of only an empty string?
- When would you not want to include an empty string?

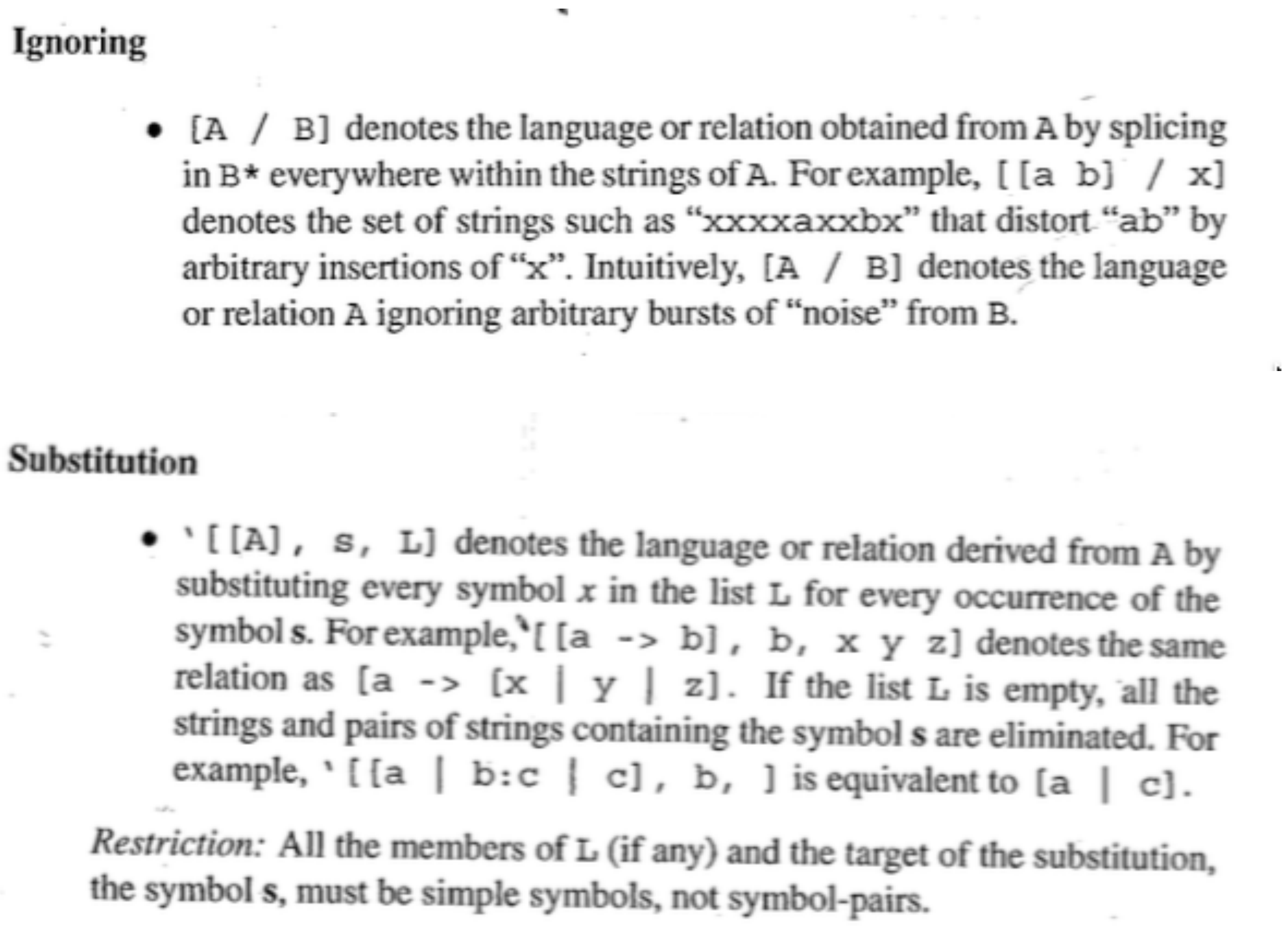
Reading questions

- I am curious about the idea that multiple regular expressions can produce the same set of strings. For this to happen, is it a requirement that the two regular expressions be using a different alphabet? Or at least that it is the case that there is a multicharacter symbol that can be misconstrued as a number of single character symbols.

- The concept of multiple regular expressions producing the same output is clear, and maybe one way could be better/is more concise/is a more elegant method, but as long as the output is always the same, is the specific construction of the regular expression ever significantly important?

Reading questions

- I was highly confused by the set operations Ignoring and Substitution. Would it be possible to go over what it is exactly that each of them does and where we might see them used in Computational Linguistics?



Ignoring

- `[A / B]` denotes the language or relation obtained from A by splicing in `B*` everywhere within the strings of A. For example, `[[a b] / x]` denotes the set of strings such as "xxxxaxxbx" that distort "ab" by arbitrary insertions of "x". Intuitively, `[A / B]` denotes the language or relation A ignoring arbitrary bursts of "noise" from B.

Substitution

- `` `[[A], s, L]`` denotes the language or relation derived from A by substituting every symbol *x* in the list L for every occurrence of the symbol **s**. For example, `` `[[a -> b], b, x y z]`` denotes the same relation as `[a -> [x | y | z]`. If the list L is empty, all the strings and pairs of strings containing the symbol **s** are eliminated. For example, `` `[[a | b:c | c], b, ]`` is equivalent to `[a | c]`.

Restriction: All the members of L (if any) and the target of the substitution, the symbol **s**, must be simple symbols, not symbol-pairs.