

Compiler Construction

Theoretical Exercise 2: Lexical analysis

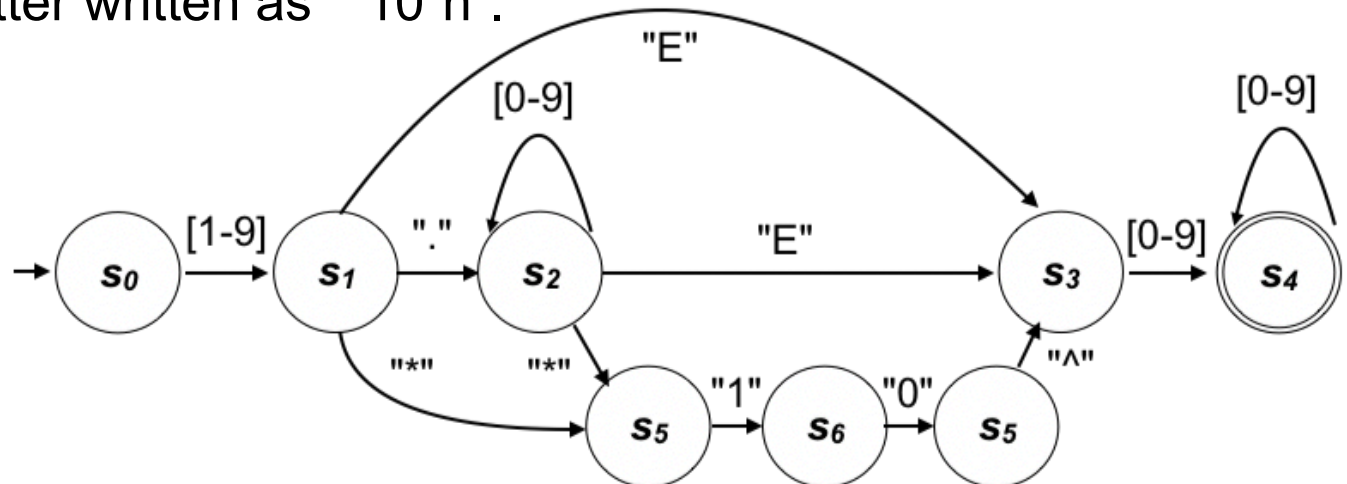
Michael Engel

2.1 Regular languages

- a. A palindrome is a word (or set of words, ignoring whitespace, punctuation characters and capitalization) that reads the same when read from left to right and from right to left.
Simple examples are “rotor” and “madam”, but you can also come up with sophisticated palindromes such as “saippuakuppinippukauppias”
Are palindromes regular languages?
- Detecting palindromes requires some sort of memory mechanism – you have to remember the first n characters and replay them in reverse order, e.g. using a stack.
 - For example, to detect "rotor" as a palindrome, the scanner (using a DFA, so a regular language) would have to be able to remember that there was already an "r". For "rorotor", you would need to be able to *nest* the "r" and "o", similar to problems detecting correctly used brackets in mathematical expressions.
 - This is only possible using recursion, so regular languages are not sufficient to detect palindromes.

2.1 Regular languages

- b. In scientific notation, all numbers are written in the form $m \times 10^n$ or, as shorthand, mEn . The integer n is called the order of magnitude and the real number m is called the significand or mantissa.
- In normalized notation, the exponent is chosen so that the absolute value (modulus) of the significand m is at least 1 but less than 10.
- Examples for normalized notation are Avogadro's constant $6.022E23$ or the speed of light $c = 2.99792458E8$ m/s.
- Draw a deterministic finite automaton (DFA) which accepts numbers in normalized notation (no units required) using either the "E" or the "10ⁿ" notation, the latter written as " $*10^n$ ".



2.1 Regular languages

- c. Create a regular expression that generates the language over the alphabet $\{a, b\}$ of all strings containing an even number of b 's.

The idea here is to always create two b 's with an arbitrary (including zero) number of a 's before, after or in between the b 's:

$(a^*ba^*ba^*)^*$

We need the first a^* to generate strings such as "a" or "abbbb"

We need the middle a^* to generate e.g. "abaaaaaba" or "bab"

And we need the final a^* to generate e.g. "abbaa"

2.1 Regular languages

- c. Create a regular expression that generates the language over the alphabet $\{a, b\}$ of all strings in which $n \bmod m = 2$ where n is the length of the string. In other words, all strings in the language have a length of 2, or 5, or 8, or 11, etc.

Hint: generate two initial letters, and expand the string by appending chunks of length 3.

$[a|b][a|b] ([a|b][a|b][a|b])^*$

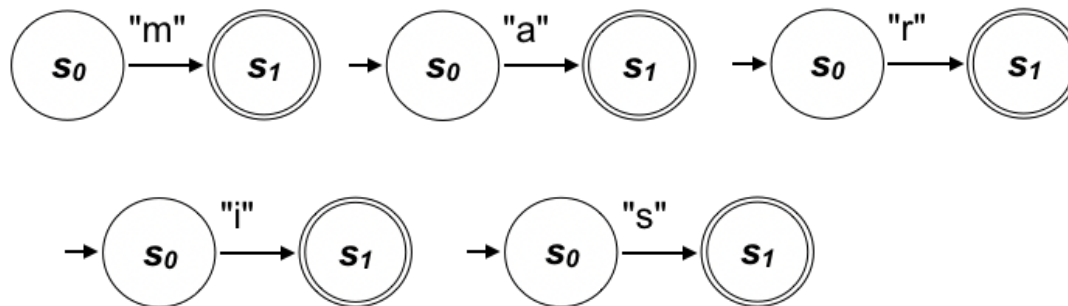
The first part $[a|b][a|b]$ generates exactly two characters that can have all of the valid combinations aa, ab, ba and bb.

Similarly, the second part always generates three characters at the same time and this partial regexp can repeat an arbitrary number of times, including zero, due to the Kleene star, so the length of the string are 2, 5, 8, 11, ...

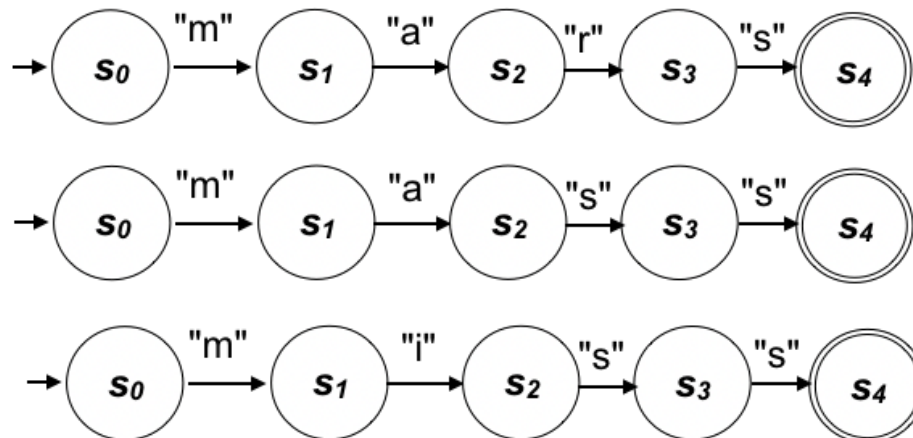
2.2 NFAs and DFAs

- a. Construct an NFA that accepts the following regular expression:
(mars | mass | miss).

First, we construct simple scanners for each of the included letters:



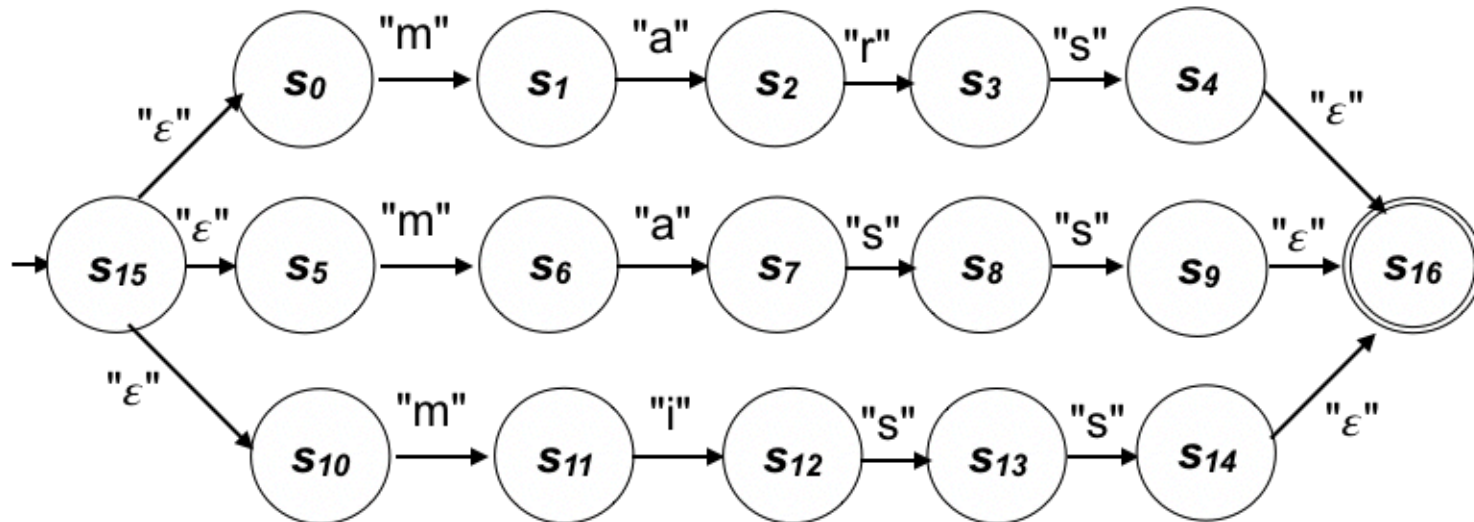
Then, we combine the simple scanners into ones that detect each word separately:



2.2 NFAs and DFAs

- a. Construct an NFA that accepts the following regular expression:
(mars | mass | miss).

Finally, we combine these using ϵ -transitions:

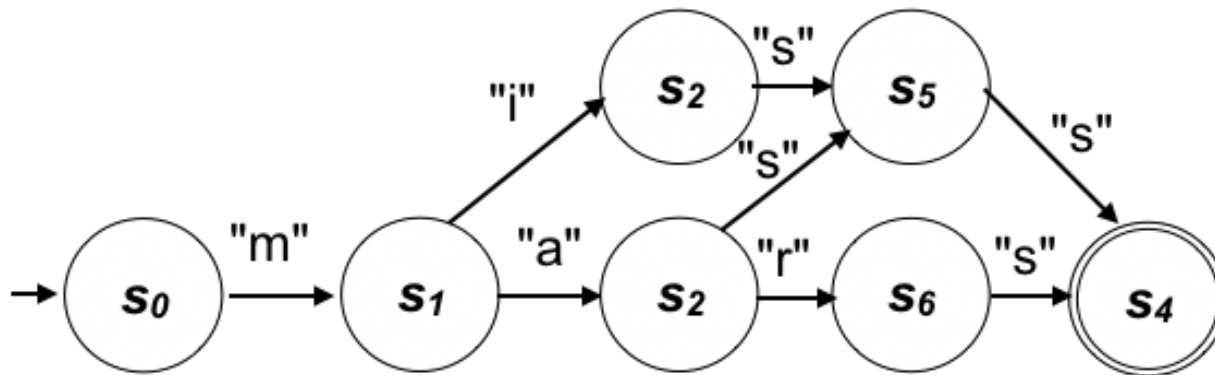


2.2 NFAs and DFAs

b. Convert your NFA to a DFA using the subset construction algorithm.

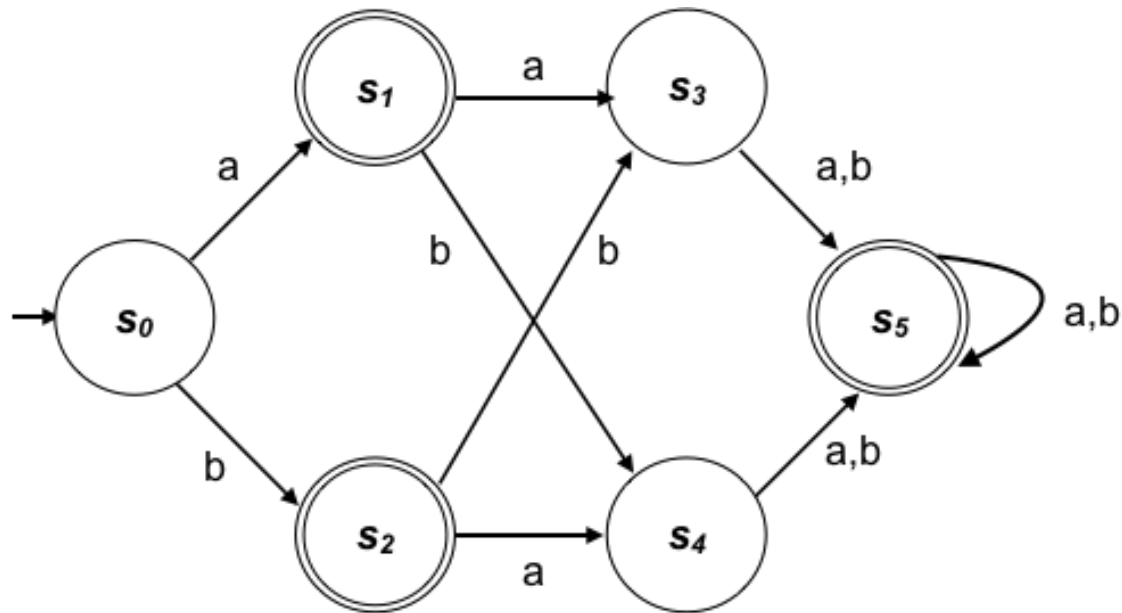
Initialization:

$q_0 \leftarrow \varepsilon\text{-closure}(\{s_{15}\}) = \{s_{15}, s_0, s_5, s_{10}\}; Q_0 \leftarrow q_0 = \{s_{15}, s_0, s_5, s_{10}\};$



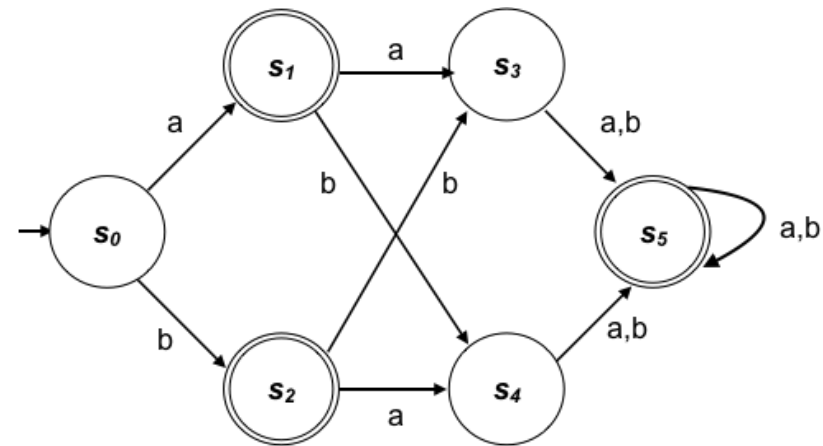
2.3 DFA minimization

Use the table method (Myhill-Nerode) to minimize the DFA given in Fig. 1. Show the steps performed, the related changes to the table and the final minimized DFA.



2.3 DFA minimization

Use the table method (Myhill-Nerode) to minimize the DFA.



Step 1:

Construct a table with all pairs of states of the DFA. Mark all pairs (p, q) of states where $p \in F$ and $q \notin F$ or vice versa.

Here: s_1 , s_2 and s_5 are accepting states, the others are not.

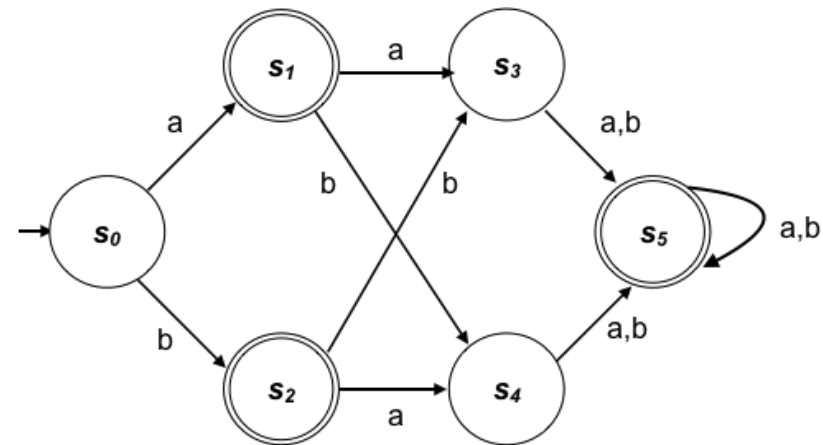
	s0	s1	s2	s3	s4	s5
s0						
s1	X					
s2	X					
s3		X	X			
s4		X	X			
s5	X			X	X	

2.3 DFA minimization

Use the table method (Myhill-Nerode) to minimize the DFA.

Step 2:

If there are any unmarked pairs (p, q) such that $[\delta(p, x), \delta(q, x)]$ is marked, then mark $[p, q]$ (here 'x' is an arbitrary input symbol) – new markings shown in red:



	s0	s1	s2	s3	s4	s5
s0						
s1	X					
s2	X					
s3		X	X			
s4		X	X			
s5	X			X	X	

(s3, s0), x=a

(s3, a) = s5
(s0, a) = s1

(s3, s0), x=b

(s3, b) = s5
(s0, b) = s2

(s4, s0), x=a

(s4, a) = s5
(s0, a) = s1

(s4, s0), x=b

(s4, b) = s5
(s0, b) = s2

(s2, s1), x=a

(s2, a) = s4
(s1, a) = s3

(s2, s1), x=b

(s2, b) = s3
(s1, b) = s4

(s1, s5), x=a

(s1, a) = s3
(s5, a) = s5

(s1, s5), x=b

(s1, b) = s4
(s5, b) = s5

(s5, s2), x=a

(s5, a) = s5
(s2, a) = s4

(s5, s2), x=b

(s5, b) = s5
(s2, b) = s3

(s4, s3), x=a

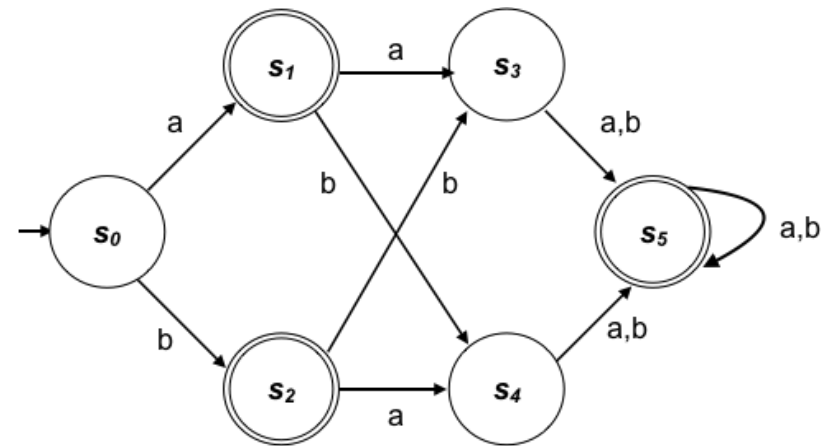
(s4, a) = s5
(s3, a) = s5

(s4, s3), x=b

(s4, b) = s5
(s3, b) = s5

2.3 DFA minimization

Use the table method (Myhill-Nerode) to minimize the DFA.



Step 2:

This results in new markings at positions (s1, s5), (s2, s5):

(s3, s0), x=a

(s3, a) = s5

(s0, a) = s1

(s3, s0), x=b

(s3, b) = s5

(s0, b) = s2

(s4, s0), x=a

(s4, a) = s5

(s0, a) = s1

(s4, s0), x=b

(s4, b) = s5

(s0, b) = s2

(s2, s1), x=a

(s2, a) = s4

(s1, a) = s3

(s2, s1), x=b

(s2, b) = s3

(s1, b) = s4

(s1, s5), x=a

(s1, a) = s3

(s5, a) = s5

(s1, s5), x=b

(s1, b) = s4

(s5, b) = s5

(s5, s2), x=a

(s5, a) = s5

(s2, a) = s4

(s5, s2), x=b

(s5, b) = s5

(s2, b) = s3

(s4, s3), x=a

(s4, a) = s5

(s3, a) = s5

(s4, s3), x=b

(s4, b) = s5

(s3, b) = s5

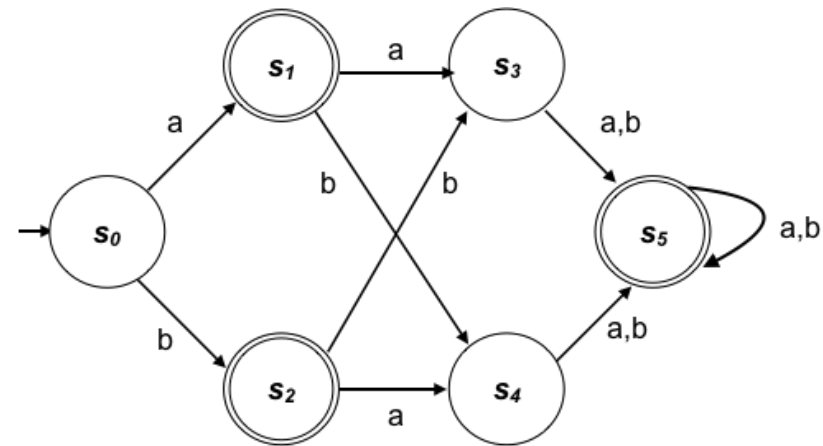
	s0	s1	s2	s3	s4	s5
s0						
s1	X					
s2	X					
s3		X	X			
s4		X	X			
s5	X	X	X	X	X	

2.3 DFA minimization

Use the table method (Myhill-Nerode) to minimize the DFA.

Step 3:

Repeat until no more markings can be made:



(s3, s0), x=a

(s3, a) = s5

(s0, a) = s1

(s3, s0), x=b

(s3, b) = s5

(s0, b) = s2

(s4, s0), x=a

(s4, a) = s5

(s0, a) = s1

(s4, s0), x=b

(s4, b) = s5

(s0, b) = s2

(s2, s1), x=a

(s2, a) = s4

(s1, a) = s3

(s2, s1), x=b

(s2, b) = s3

(s1, b) = s4

(s4, s3), x=a

(s4, a) = s5

(s3, a) = s5

(s4, s3), x=b

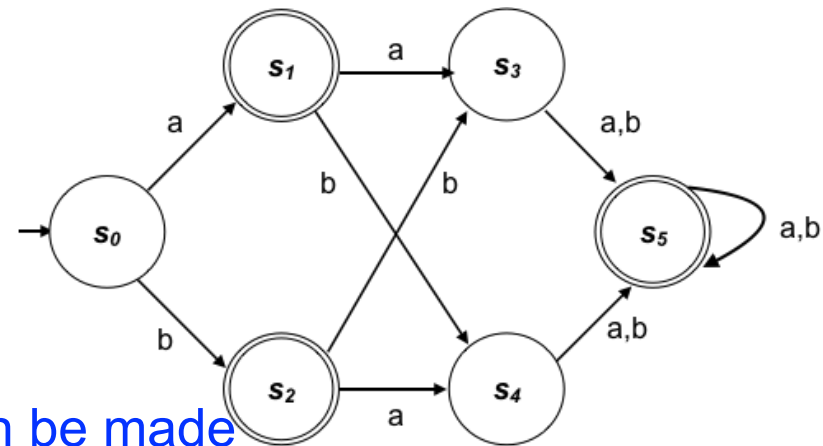
(s4, b) = s5

(s3, b) = s5

	s0	s1	s2	s3	s4	s5
s0						
s1	X					
s2	X					
s3	X	X	X			
s4	X	X	X			
s5	X	X	X	X	X	

2.3 DFA minimization

Use the table method (Myhill-Nerode) to minimize the DFA.



Step 4: Repeat until no more markings can be made

(s2, s1), x=a

(s2, a) = s4

(s1, a) = s3

(s2, s1), x=b

(s2, b) = s3

(s1, b) = s4

(s4, s3), x=a

(s4, a) = s5

(s3, a) = s5

(s4, s3), x=b

(s4, b) = s5

(s3, b) = s5

No more markings could be made in this iteration \Rightarrow **terminate**

The unmarked elements imply that state $s1 \approx s2$ and state $s3 \approx s4$.

The resulting automaton looks like this:

	s0	s1	s2	s3	s4	s5
s0						
s1	X					
s2	X					
s3	X	X	X			
s4	X	X	X			
s5	X	X	X	X	X	

