Regular Expression to Finite Automaton Conversion - Theory

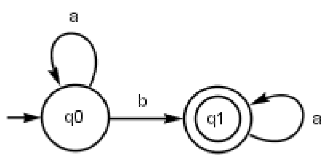
Regular Languages

Definition: A regular language L over an alphabet A is a formal language that satisfies the following equivalent properties:

- L is accepted by some (deterministic) finite state automaton with alphabet A.
- L is described/denoted by some regular expression over A.
- L is generated by some regular/right linear grammar with terminal alphabet A.

Example:

Acceptor



Descriptor

a*ba*

Les symboles utilisés

a|b	: a OU b
a*	: lettre a au moins une fois
(ab)*	: séquence ab au moins une fois

Exercices possibles (application « Exorciser »)

Regular Languages

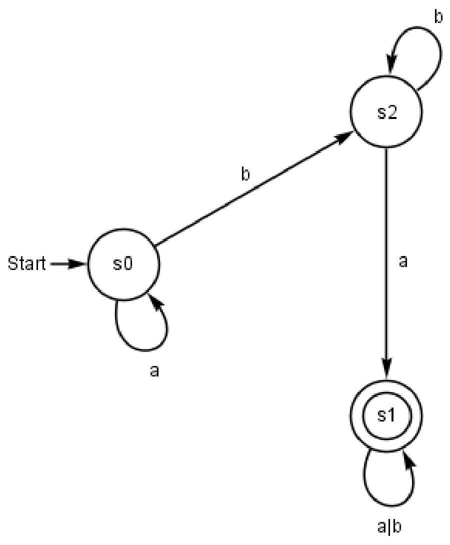
- Constructing Finite Automata
- Regular Expression to Finite Automata Conversion

Constructing Finite Automata

Give the state diagram of a deterministic finite automaton over the alphabet {a, b} recognizing language L.

L = { w | w contains the infix "ba" }

En voici la solution
(Ici : transition entre des états s0,s1,s2 avec état initial s0 et état acceptant s1)



Regular Expression to Finite Automata Conversion

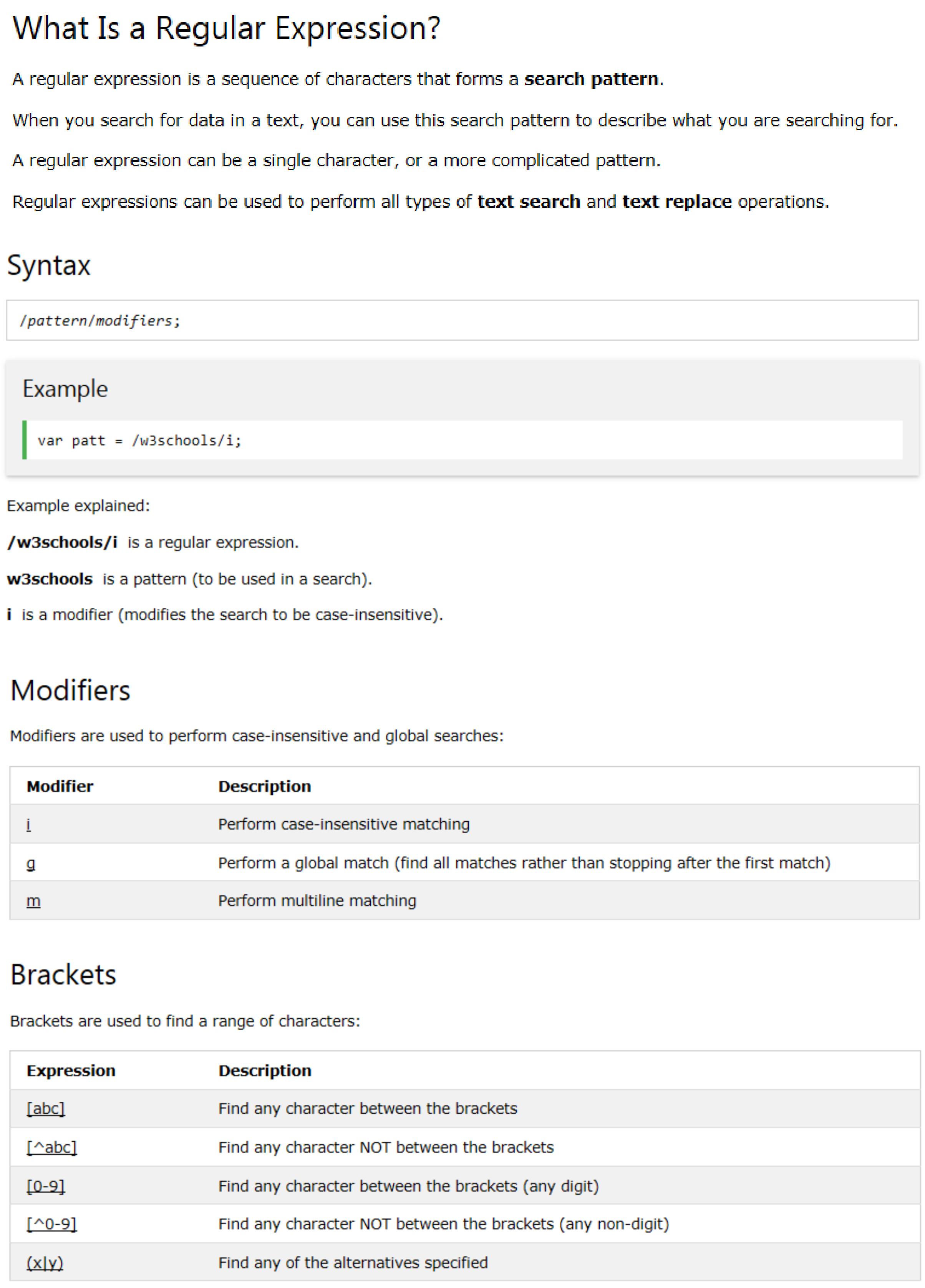
Construct a finite automaton describing the language *L=L(R)*.

R = (bb)*a*

L'expression régulière R est la règle d'acceptation de certains mots générés à partir d'un alphabète.

Sur l'alphabète considéré, ici {a,b}, l'expression R est génératrice de mots et crée le langage L.

De *W3Schools*



What Is a Regular Expression?

A regular expression is a sequence of characters that forms a **search pattern**.

When you search for data in a text, you can use this search pattern to describe what you are searching for.

A regular expression can be a single character, or a more complicated pattern.

Regular expressions can be used to perform all types of **text search** and **text replace** operations.

Syntax

```
/pattern/modifiers;
```

Example

```
var patt = /w3schools/i;
```

Example explained:

/w3schools/i is a regular expression.

w3schools is a pattern (to be used in a search).

i is a modifier (modifies the search to be case-insensitive).

Modifiers

Modifiers are used to perform case-insensitive and global searches:

Modifier	**Description**
i	Perform case-insensitive matching
g	Perform a global match (find all matches rather than stopping after the first match)
m	Perform multiline matching

Brackets

Brackets are used to find a range of characters:

Expression	**Description**
[abc]	Find any character between the brackets
[^abc]	Find any character NOT between the brackets
[0-9]	Find any character between the brackets (any digit)
[^0-9]	Find any character NOT between the brackets (any non-digit)
(x|y)	Find any of the alternatives specified

Metacharacters

Metacharacters are characters with a special meaning:

Metacharacter	**Description**
.	Find a single character, except newline or line terminator
\w	Find a word character
\W	Find a non-word character
\d	Find a digit
\D	Find a non-digit character
\s	Find a whitespace character
\S	Find a non-whitespace character
\b	Find a match at the beginning/end of a word
\B	Find a match not at the beginning/end of a word
\0	Find a NUL character
\n	Find a new line character
\f	Find a form feed character
\r	Find a carriage return character
\t	Find a tab character
\v	Find a vertical tab character
\xxx	Find the character specified by an octal number xxx
\xdd	Find the character specified by a hexadecimal number dd
\uxxxx	Find the Unicode character specified by a hexadecimal number xxxx

Quantifiers

Quantifier	**Description**
n+	Matches any string that contains at least one *n*
n*	Matches any string that contains zero or more occurrences of *n*
n?	Matches any string that contains zero or one occurrences of *n*
n{X}	Matches any string that contains a sequence of *X* *n*'s
n{X,Y}	Matches any string that contains a sequence of X to Y *n*'s
n{X,}	Matches any string that contains a sequence of at least X *n*'s
n$	Matches any string with *n* at the end of it
^n	Matches any string with *n* at the beginning of it
?=n	Matches any string that is followed by a specific string *n*
?!n	Matches any string that is not followed by a specific string *n*

Using String Methods

In JavaScript, regular expressions are often used with the two **string methods**: search() and replace().

The search() method uses an expression to search for a match, and returns the position of the match.

The replace() method returns a modified string where the pattern is replaced.

Use a regular expression to do a case-insensitive search for "w3schools" in a string:

```
var str = "Visit W3Schools";
var n = str.search(/w3schools/i);
```

The result in n will be:

```
6
```

Use a case insensitive regular expression to replace Microsoft with W3Schools in a string:

```
var str = "Visit Microsoft!";
var res = str.replace(/microsoft/i, "W3Schools");
```

The result in res will be:

```
Visit W3Schools!
```

Using the RegExp Object

Using test()

The test() method is a RegExp expression method.

It searches a string for a pattern, and returns true or false, depending on the result.

```
var patt = /e/;
patt.test("The best things in life are free!");
```

Since there is an "e" in the string, the output of the code above will be:

```
true
```

Using exec()

The exec() method is a RegExp expression method.

It searches a string for a specified pattern, and returns the found text.

If no match is found, it returns *null*.

```
/e/.exec("The best things in life are free!");
```

Since there is an "e" in the string, the output of the code above will be:

```
e
```