Grammars, Regular Expressions, and Finite-State Automata

Vincent J. Matsko

WFNMC 2010, Riga, Latvia

27 July 2010

The First Problem

Consider the following method for creating strings of 1's. Begin with the symbol x, and apply the following replacement rules as many times as desired and in any order.

$$x \mapsto 111x \quad (1)$$
$$x \mapsto 11111x \quad (2)$$
$$x \mapsto 111 \quad (3)$$

How many strings of one or more 1's CANNOT be obtained by applying these rules?

Solution to the First Problem

Key idea: this is the Postage Stamp problem.

The maximum number of ones which cannot be produced by (1) and (2) only is $(3-1)(5-1)-1=7$.

Quick enumeration: only strings of 1, 2, 4, or 7 ones cannot be obtained by using (1) and (2). Since the process ends with an application of (3), strings of 4, 5, 7, or 10 ones cannot be obtained.

Thus, only strings of 1, 2, 4, 5, 7, or 10 cannot be produced by using the above rules.

Representation of the Solution to the First Problem

We may choose 3 or 5 ones as many times as we like (perhaps zero times), followed by three ones:

$$(111 \vee 11111)^{*}111.$$

This expression is called a *regular expression*.

Here, the disjunction $\vee$ means there is a choice (exclusive or), the $*$ means the expression may be repeated zero or more times, and juxtaposition means concatenation.

Λ is the empty string.

The Second Problem

Consider the following method for creating strings of 0's and 1's. Begin with the symbol x, and apply the following replacement rules as many times as desired and in any order.

$$
\begin{aligned}
x &\mapsto 1 \quad (1) & y &\mapsto 1x \quad (5) \\
x &\mapsto 0x \quad (2) & z &\mapsto 0 \quad (6) \\
x &\mapsto 1y \quad (3) & z &\mapsto 0y \quad (7) \\
y &\mapsto 0z \quad (4) & z &\mapsto 1z \quad (8)
\end{aligned}
$$

The process ends when only 0's and 1's remain. For example,

$$x \overset{3}{\mapsto} 1y \overset{4}{\mapsto} 10z \overset{8}{\mapsto} 101z \overset{6}{\mapsto} 1010.$$

Describe all binary strings which can be produced by these rules.

Solution to the Second Problem (1)

$$
\begin{array}{llll}
x \mapsto 1 & (1) & y \mapsto 1x & (5) \\
x \mapsto 0x & (2) & z \mapsto 0 & (6) \\
x \mapsto 1y & (3) & z \mapsto 0y & (7) \\
y \mapsto 0z & (4) & z \mapsto 1z & (8)
\end{array}
$$

A consistent interpretation of these rules is:

- "x" means that we have read in a string which is 0 mod 3 so far,
- "y" means that we have read in a string which is 1 mod 3 so far, and
- "z" means that we have read in a string which is 2 mod 3 so far.

Solution to the Second Problem (2)

$$
\begin{array}{llll}
x \mapsto 1 & (1) & y \mapsto 1x & (5) \\
x \mapsto 0x & (2) & z \mapsto 0 & (6) \\
x \mapsto 1y & (3) & z \mapsto 0y & (7) \\
y \mapsto 0z & (4) & z \mapsto 1z & (8)
\end{array}
$$

Example:

x is replaced (Rule 3) by $1y$ (now 1 mod 3).

$1y$ becomes (Rule 4) $10z$ (now 2 mod 3).

Then, we get (Rule 8) $101z$ (still 2 mod 3).

Finally, we have (Rule 6) 1010 (now 1 mod 3).

Solution to the Second Problem (3)

$$
\begin{array}{llll}
x \mapsto 1 & (1) & y \mapsto 1x & (5) \\
x \mapsto 0x & (2) & z \mapsto 0 & (6) \\
x \mapsto 1y & (3) & z \mapsto 0y & (7) \\
y \mapsto 0z & (4) & z \mapsto 1z & (8)
\end{array}
$$

Thus, this grammar describes all strings which represent binary numbers which are 1 mod 3.

How might we come up with this interpretation? We'll see later!

Representing the Solution to the Second Problem (1)

So how can we write a regular expression for those binary strings congruent to 1 mod 3? Let's begin with a 1 (preceded, perhaps, by leading zeroes):

$$0^*1$$

Representing the Solution to the Second Problem (2)

So how can we write a regular expression for those binary strings congruent to 1 mod 3? Let's begin with a 1 (preceded, perhaps, by leading zeroes):

$$0^*1(0$$

The next digit can be either a 0 or a 1. Let's look at the 0 first.

Representing the Solution to the Second Problem (3)

So how can we write a regular expression for those binary strings congruent to 1 mod 3? Let's begin with a 1 (preceded, perhaps, by leading zeroes):

$$0^*1(0 \cdots 0$$

We are currently at 2 mod 3, so we must continue. Note that a 1 leaves us at 2 mod 3, so we must eventually add another 0.

Representing the Solution to the Second Problem (4)

So how can we write a regular expression for those binary strings congruent to 1 mod 3? Let's begin with a 1 (preceded, perhaps, by leading zeroes):

$$0^*1(01^*0$$

But even though we need another 0, we may include as many 1's as we wish as they leave the string at 2 mod 3.

Representing the Solution to the Second Problem (5)

So how can we write a regular expression for those binary strings congruent to 1 mod 3? Let's begin with a 1 (preceded, perhaps, by leading zeroes):

$$0^*1(01^*0 \vee 1 \cdots 1$$

Now what if the next digit is a 1? Then we are at 0 mod 3, and adding 0's doesn't change that. So we definitely need another 1.

Representing the Solution to the Second Problem (6)

So how can we write a regular expression for those binary strings congruent to 1 mod 3? Let's begin with a 1 (preceded, perhaps, by leading zeroes):

$$0^*1(01^*0 \vee 10^*1)$$

But although we need another 1, we may include as many 0's as we like since they leave the string at 0 mod 3.

Representing the Solution to the Second Problem (7)

So how can we write a regular expression for those binary strings congruent to 1 mod 3? Let's begin with a 1 (preceded, perhaps, by leading zeroes):

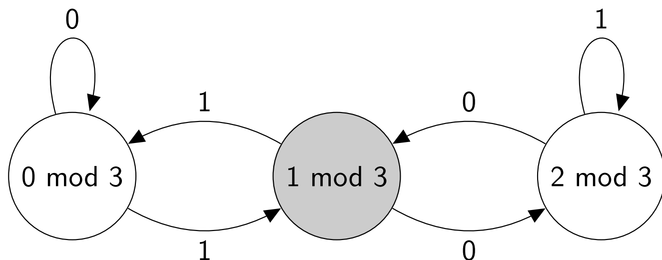
$$0^*1(01^*0 \vee 10^*1)^*$$

This leaves us at 1 mod 3, so this process may be repeated as many times as desired. It may even not be used at all, since 0^*1 produces strings which are 1 mod 3.

Alternate Representation of the Solution (1)

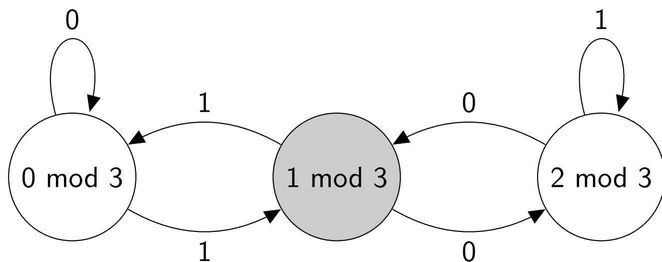
Can we make a finite-state machine which describes the strings which are 1 mod 3?

Start state: 0 mod 3; accepting state(s): 1 mod 3.

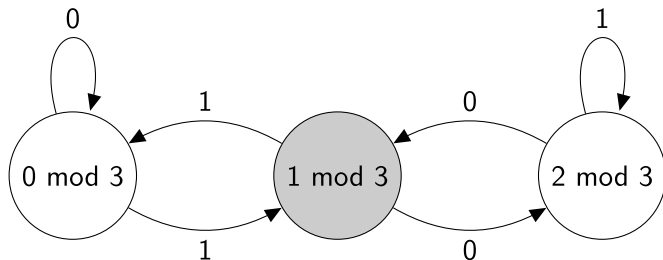


Alternate Representation of the Solution (2)

See that the string 10010111 (which is 151_{10}) is accepted, while 11001111 (which is 209_{10}) is not.



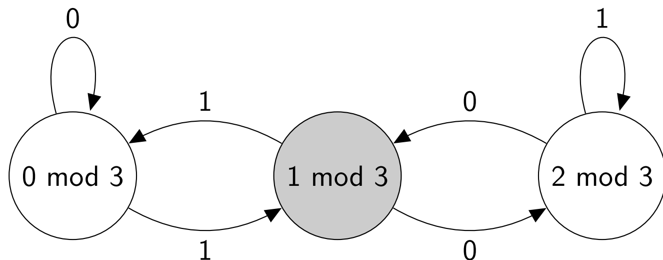
Alternate Representation of the Solution (3)



To get to the state 1 mod 3, we must read in any number of 0's followed by a 1:

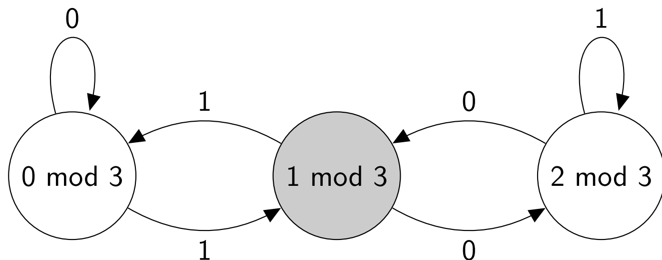
$$0^*1.$$

Alternate Representation of the Solution (4)



Now, how can we leave the state 1 mod 3 and return to it?

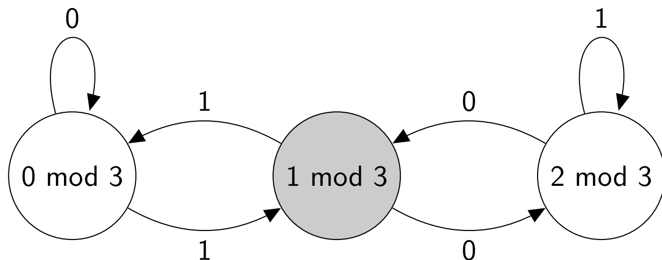
Alternate Representation of the Solution (4)



Now, how can we leave the state 1 mod 3 and return to it?

Either by going right: 01*0,

Alternate Representation of the Solution (4)

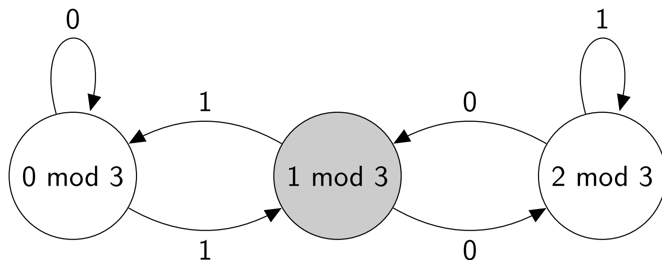


Now, how can we leave the state 1 mod 3 and return to it?

Either by going right: 01^*0,

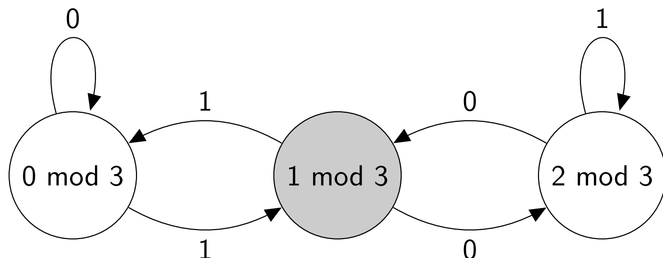
or by going left: 10^*1.

Alternate Representation of the Solution (5)



Thus, we leave and return by $01^*0 \vee 10^*1$.

Alternate Representation of the Solution (6)



This gives us the regular expression we want:

$$0^{*}1(01^{*}0 \vee 10^{*}1)^{*}.$$

Getting a Perspective

Note that we have already seen an interesting relationship among

- grammars,
- regular expressions, and
- finite-state machines (automata).

Type 3 Grammars

Each rule consists of:

- One non-terminal on the left-hand side, and
- A string with one non-terminal on the right-hand side, occurring right-most.

$$x \mapsto 111x \qquad (1)$$
$$x \mapsto 11111x \qquad (2)$$
$$x \mapsto 111 \qquad (3)$$

Type 2 Grammars

Each rule consists of:

- One non-terminal on the left-hand side, and
- A string with one non-terminal on the right-hand side, occurring anywhere.

$$x \mapsto 11x1 \qquad (1)$$
$$x \mapsto 11x111 \qquad (2)$$
$$x \mapsto 111 \qquad (3)$$

Note: The grammar is of a different type, but the language accepted is the same as the previous grammar.

More Type 2 Grammars

Each rule consists of:

- One non-terminal on the left-hand side, and
- A string with one non-terminal on the right-hand side, occurring anywhere.

$$x \mapsto 0x1 \quad (1)$$
$$x \mapsto 01 \quad (2)$$

What language is accepted by this grammar?

More Type 2 Grammars

Each rule consists of:

- One non-terminal on the left-hand side, and
- A string with one non-terminal on the right-hand side, occurring anywhere.

$$x \mapsto 0x1 \quad (1)$$
$$x \mapsto 01 \quad (2)$$

What language is accepted by this grammar?

Answer:

$$\{0^n 1^n \mid n \in \mathbb{N}^\times\}.$$

0^n1^n is not a regular expression, however! (More later.)

Type 1 Grammars

Rules cannot shrink strings:

- The length of the left-hand string in a rule is no greater than the length of the right-hand string.

$$
\begin{aligned}
x &\mapsto 11x1 && (1)\\
x &\mapsto 11x111 && (2)\\
x &\mapsto 111 && (3)\\
1x &\mapsto 1111x && (4)
\end{aligned}
$$

Note: The grammar is of a different type, but the language accepted is the same as the previous Type 3 grammar.

Type 0 Grammars

Type 0 grammars may shrink strings.

$$x \mapsto 111x \tag{1}$$
$$x \mapsto 11111x \tag{2}$$
$$x \mapsto 111 \tag{3}$$
$$111x \mapsto x \tag{4}$$

Type 0 Grammars

Type 0 grammars may shrink strings.

$$\begin{aligned} x &\mapsto 111x && (1) \\ x &\mapsto 11111x && (2) \\ x &\mapsto 111 && (3) \\ 111x &\mapsto x && (4) \end{aligned}$$

Careful!

$$x \overset{2[2]}{\mapsto} 1111111111x \overset{3[4]}{\mapsto} 1x \overset{3}{\mapsto} 1111.$$

Because we can get 4 ones, we can get 7 or 10 ones, also.

$$x \overset{2}{\mapsto} 11111x \overset{4}{\mapsto} 11x \overset{3}{\mapsto} 11111.$$

Basic Syntax

The main elements of regular expressions are:

- Juxtaposition:

$$001 = \{001\}.$$

- Disjunction:

$$01 \vee 000 \vee 1101 = \{01, 000, 1101\}.$$

- Repetition:

$$(01)^* = \{\Lambda, 01, 0101, 010101, 01010101, ...\}$$

Some Examples

- An even number of ones (perhaps none):

$$(11)^*$$

- An odd number of ones:

$$1(11)^*$$

- The number of ones is 3 mod 5:

$$(11111)^*111$$

- All strings of zeroes and ones:

$$(0 \vee 1)^*$$

More Examples

- Strings with zeroes and an even number of ones:

More Examples

- Strings with zeroes and an even number of ones:

$$(0^*10^*1)^*0^*$$

More Examples

- Strings with zeroes and an even number of ones:

$$(0^*10^*1)^*0^*$$

- Strings of zeroes and ones with no zeroes adjacent:

More Examples

- Strings with zeroes and an even number of ones:

$$(0^*10^*1)^*0^*$$

- Strings of zeroes and ones with no zeroes adjacent:

$$1^*(011^*)^*(\Lambda \vee 0)$$

More Examples

- Strings with zeroes and an even number of ones:

$$(0^*10^*1)^*0^*$$

- Strings of zeroes and ones with no zeroes adjacent:

$$1^*(011^*)^*(\Lambda \vee 0)$$

- Strings with an even number of zeroes and an even number of ones:

More Examples

- Strings with zeroes and an even number of ones:

$$(0^*10^*1)^*0^*$$

- Strings of zeroes and ones with no zeroes adjacent:

$$1^*(011^*)^*(\Lambda \vee 0)$$

- Strings with an even number of zeroes and an even number of ones:

$$((00 \vee 11)^*(01 \vee 10)(00 \vee 11)^*(01 \vee 10))^*\,(00 \vee 11)^*$$

The Theorem

The following sets of languages are the *same:*

- the set of languages described by Type 3 grammars,
- the set of languages described by regular expressions, and
- the set of languages accepted by finite-state machines.

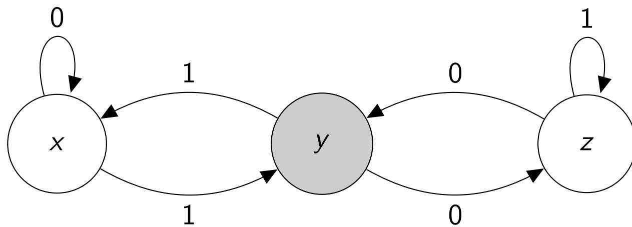
A Problem Revisited

$$
\begin{aligned}
x &\mapsto 1 && (1) & y &\mapsto 1x && (5)\\
x &\mapsto 0x && (2) & z &\mapsto 0 && (6)\\
x &\mapsto 1y && (3) & z &\mapsto 0y && (7)\\
y &\mapsto 0z && (4) & z &\mapsto 1z && (8)
\end{aligned}
$$

A Problem Revisited

$$
\begin{array}{llll}
x \mapsto 1 & (1) & y \mapsto 1x & (5) \\
x \mapsto 0x & (2) & z \mapsto 0 & (6) \\
x \mapsto 1y & (3) & z \mapsto 0y & (7) \\
y \mapsto 0z & (4) & z \mapsto 1z & (8)
\end{array}
$$

Rules 1 and 6 always take you to the accepting state.



A Non-Regular Expression

0^n1^n is *not* a regular expression. Therefore, no Type 3 grammar can describe this set of strings, nor can a finite-state machine be built which accepts only those strings. Why?

A Non-Regular Expression

0^n1^n is *not* a regular expression. Therefore, no Type 3 grammar can describe this set of strings, nor can a finite-state machine be built which accepts only those strings. Why?

Essentially, finite-state machines only have a finite capacity to remember.

To accept only the strings 0^n1^n, an FSM would need to be able to remember that it read *arbitrarily long* strings of zeroes so far. This is not possible with a *finite*-state machine.

Hint for Problem Three

Sometimes it is useful to derive other rules from those given.

In this problem, it is possible to derive the rule

$$a \mapsto a0$$

and the rule

$$a \mapsto a1.$$

Use these additional rules to come up with a complete solution.