DFA to Regular Expressions

To convert a machine into a regular expression we first characterize it by its corresponding **Set Equations**.

Consider the machine in the following figure.

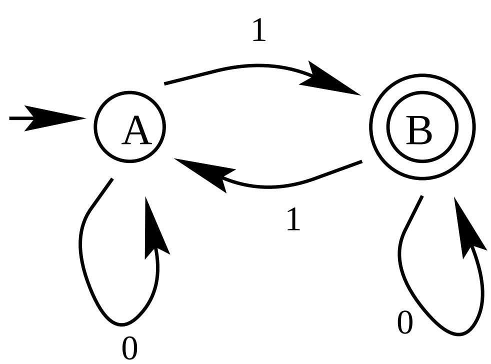


Figure 1: A DFA for strings with an odd number of 1's

The **Set Equations** describe ways to reach a final state.

From state A, one can scan (generate) a 0 and go to state A, or one can scan (generate) a 1 and go to state B.

A = 0A + 1B

From state B, one can scan (generate) a 1 and go to state A, or one can scan (generate) a 0 and go to state B.

Since B is a final state, one can do nothing (i.e., λ keeps one in a final state).

B = 1A + 0B + λ.

Slide Lecture 3 –59

Because A is the initial state, we construct the regular expression with respect to A (i.e., ways of getting from the initial state A to a final state.)

We select a state that is not the initial state to remove. In this case, we remove state B.

Before we remove B, we use Arden's rule to remove recursive references.

B = 1A + 0B + λ.

B = 0B + (1A + λ).

B = 0*(1A + λ).

B = 0*(1A) + 0*(λ).

B = 0*1A + 0*.

Slide Lecture 3 –60

Now we can rewrite A

A = 0A + 1B

A = 0A + 1(0*1A + 0*)

A = 0A + 10*1A + 10*

A = (0 + 10*1)A + 10*

Applying Arden's rule

A = (0 + 10*1)*(10*)

Slide Lecture 3 –61

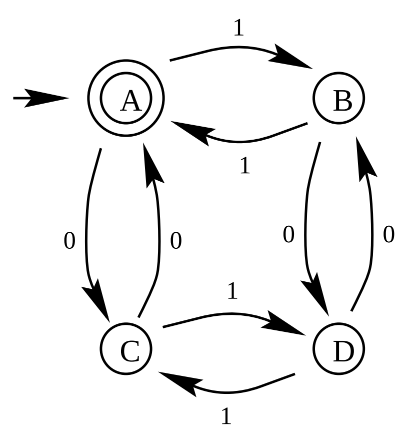


Figure 2: Use Set Equations to generate the regular expression for this machine.

Slide Lecture 3 –62

$$A = 0C + 1B + \lambda$$
$$B = 0D + 1A$$
$$C = 0A + 1D$$
$$D = 0B + 1C$$

Remove state B

$$A = 0C + 1(0D + 1A) + \lambda$$
$$C = 0A + 1D$$
$$D = 0(0D + 1A) + 1C$$

Slide Lecture 3 –63

Simplify

$$A = 0C + 10D + 11A + \lambda$$
$$C = 0A + 1D$$
$$D = 00D + 01A + 1C$$

Remove state C.

$$A = 0(0A + 1D) + 10D + 11A + \lambda$$
$$D = 00D + 01A + 1(0A + 1D)$$

Slide Lecture 3 –64

Simplify A

$$A = 00A + 01D + 10D + 11A + \lambda$$
$$A = 00A + 11A + 01D + 10D + \lambda$$
$$A = (00 + 11)A + (01 + 10)D + \lambda$$

Simplify D

$$D = 00D + 01A + 10A + 11D)$$
$$D = (00 + 11)D + (01 + 10)A$$

Slide Lecture 3 –65

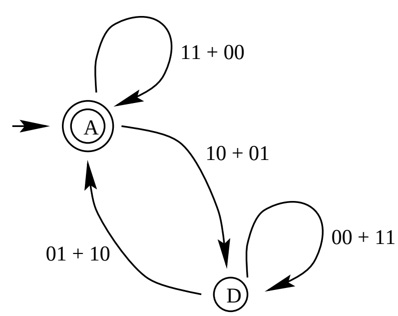


Figure 3: The Corresponding Machine

Slide Lecture 3 –66

$$A = (00 + 11)A + (01 + 10)D + \lambda$$
$$D = (00 + 11)D + (01 + 10)A$$

Apply Arden's Rule to D

$$D = (00 + 11)^*(01 + 10)A$$

Remove D

$$A = (00 + 11)A + (01 + 10)(00 + 11)^*(01 + 10)A + \lambda$$
$$A = [00 + 11 + (01 + 10)(00 + 11)^*(01 + 10)]A + \lambda$$

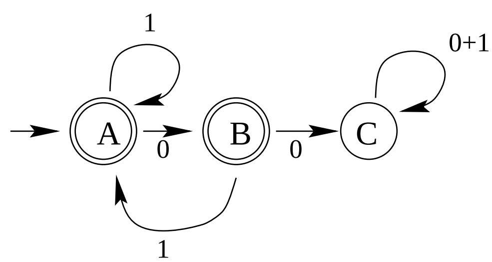
Apply Arden's Rule to A

$$A = [00 + 11 + (01 + 10)(00 + 11)^*(01 + 10)]^*$$

Slide Lecture 3 –67

TRAP STATES

Build a DFA that recognizes all strings that do not contain 2 consecutive 0's.



Slide Lecture 3 –68

Construct the Regular Expression.

$$A = 1A + 0B + \lambda$$

$$B = 1A + 0C + \lambda$$

$$C = 1C + 0C$$

Note that C can only go to C and so can never reach a final state.

If we view state C as a generator, it represents an infinite recursion that never reaches a final state.

Since a final state cannot be reached from C, drop state C from the set equations as well as all references to C.

First remove state C

$$A = 1A + 0B + \lambda$$

$$B = 1A + 0C + \lambda$$

Slide Lecture 3 –69

Next drop references to state C

$$B = 1A + \lambda$$

Now construct the regular expression by substituting B in the expressions for A.

$$A = 1A + 0(1A + \lambda) + \lambda$$

$$A = 1A + 01A + 0 + \lambda$$

$$A = (1 + 01)A + (0 + \lambda)$$

$$A = (1 + 01)^*(0 + \lambda)$$

$$A = (1 + 01)^*0 + (1 + 01)^*$$

Slide Lecture 3 –70

SOME SAMPLE PROBLEMS

Introduction to Languages and the Theory of Computation, (Martin, 1991)

Give the machine and regular expression for languages over (0+1) such that each string

1) contains exactly 2 0's.
2) contains at least 2 0's.
3) does not end with 01.
4) begins or ends with 00 or 11.
5) has every 0 followed by 11.
6) does not contain 110.
7) contains both 11 and 010 as substrings.

ANOTHER SAMPLE PROBLEM

Construct the DFA that recognizes the language composed of all bits strings that begin with a 1 and which is congruent to zero modulo 5 when interpreted as the binary representation of an integer (Hopcroft and Ullman, 1979).

In other words, the language is the set of bit strings representing the integer sequence
5, 10, 15, 20, 25, 30, 35, ...

HINT: how many states does it take to compute and track mod 5?

NONDETERMINISTIC MACHINES

Nondeterministic finite automata (**NDFA**) allow more freedom in the description of machines.

The transition function of a **NDFA** allows **zero or more moves** to be associated with **each** current state/input pair.

It is often easier to build a NDFA.

For example, build the machine that recognizes the set of all strings such that the tenth symbol from the right end is a 1.

Slide Lecture 3 –73

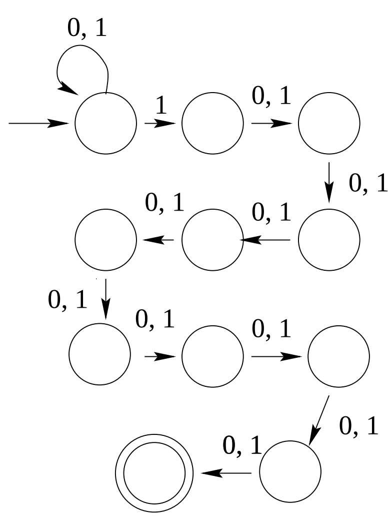


Figure 4: An NDFA.

Slide Lecture 3 –74

For a DFA, $\omega \in L$ implies exactly one accepting computation for ω on a DFA.

There may be many accepting computations on an equivalent NDFA.

A string ω is accepted by a NDFA, M, if there exists **some** (any!) accepting configuration.

Slide Lecture 3 –75

Regular expressions can also be nondeterministic.

Examples:

$(1 + 0)^*\ 00\ (1 + 0)^*$

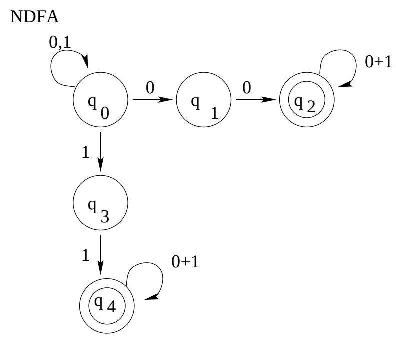
$(1 + 0)^*\ (000 + 11111)$

The problem with an NDFA is that computation can be more complex to track. (Does it accept exactly what you want and nothing more?)

The machine and regular expression however, can be less complex than the equivalent DFA.

Slide Lecture 3 –76

Let L = { ω = (0, 1)* | ω contains 2 consecutive 0s or 2 consecutive 1s}



Slide Lecture 3 –77

Current	0	1
q_0	$\{q_0,q_1\}$	$\{q_0,q_3\}$
q_1	$\{q_2\}$	$\{\ \phi\ \}$
q_2	$\{q_2\}$	$\{q_2\}$
q_3	$\{\ \phi\ \}$	$\{q_4\}$
q_4	$\{q_4\}$	$\{q_4\}$

Note: a transition can map to any subset of K (i.e., a transition maps to an element in the power set of K).

Slide Lecture 3 –78

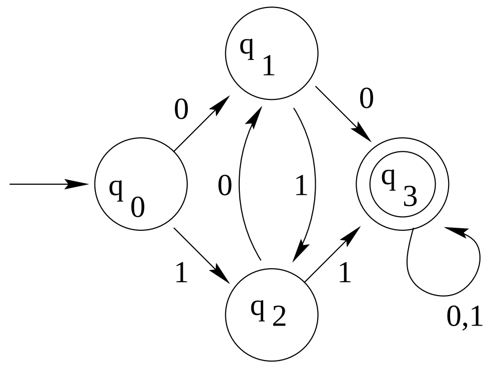


Figure 5: An equivalent DFA