Worked Solutions 1 - enumerability

Richard M^{c}Kinley

March 13, 2005

1 Introduction

These are selected model answers to exercises from lecture handout 2. Each section corresponds to a slide from that handout.

2 Page 12

Exercise 2.1. *If A, B and C are enumerable sets, the set of triples*

$$A \times B \times C = \{(a, b, c) : a \in A, b \in Bc \in C\}$$

is enumerable.

Proof. The product of two enumerable sets is enumerable. Writing $A \times B \times C$ as $(A \times B) \times C$, we see that it is the cartesian product of two enumerable sets: $A \times B$ and C. Since A and B are enumerable, $A \times B$ is enumerable So we see that $A \times B \times C$ is the product of two enumerable sets, and hence enumerable. □

Exercise 2.2. *If $A_1, \ldots, A_k$ are enumerable sets, then $A_1 \times \ldots \times A_k$ is enumerable.*

Proof. By induction. The base case ($k = 1$) is trivial.

Now suppose that, for any enumerable sets $A_1, \ldots, A_n$, the set $A_1 \times \ldots \times A_n$ is enumerable. Then, given another enumerable set A_{n+1}, $A_1 \times \ldots \times A_n \times A_{n+1} = B \times A$, where $B = A_1 \times \ldots \times A_n$ is enumerable. $B \times A$ is the cartesian product of two enumerable sets, and so is enumerable. □

3 Page 13

Exercise 3.2. *If B is enumerable and there is a total injective function $A \to B$, then A is enumerable.*

Proof. If B is enumerable, there is a coding of B - an total injective function f from B to $\mathbb{N}$. We know we have a total injective function $g : A \to B$, and their composition $f \circ g$ is a function $A \to \mathbb{N}$. This function is total and injective (you should check this, and in an exam we would expect you to prove it) and so A is enumerable. □

4 Page 14

Exercise 4.1. *The set $\mathbb{Q}^+$ is enumerable.*

Proof. Let the function $f : \mathbb{N} \times \mathbb{N} \to \mathbb{Q}^+$ take (a, b) to $\frac{a}{b}$. This function is surjective,(for example, each rational number r has a form $\frac{m}{n}$ where m and n share no factors, and then (m, n) is a pre image for r) and we know that $\mathbb{N} \times \mathbb{N}$ is enumerable, (via Cantor's zig-zag or an encoding $c((m, n)) = p^n \cdot q^m$ where p and q are prime) so $\mathbb{Q}^+$ is enumerable. □

Exercise 4.4. *The set A^* of strings over an enumerable alphabet A is enumerable.*

Proof. A is enumerable, and so we have an encoding $c_A : A \to \mathbb{N}$. We will give a total injective function $A^* \to \mathbb{N} \times \mathbb{N}$. This will be enough to show that A^* is enumerable (Why?).

A is enumerable, and so we have an encoding $c_A : A \to \mathbb{N}$. From a previous exercise, we know that for each $n \in \mathbb{N}$ there is an encoding $c_n : A^n \to \mathbb{N}$.

$$c_n((a_1, \ldots, a_n)) = c_{\mathbb{N}\times\cdots\times\mathbb{N}}(c_A(a_1), \ldots c_A(a_n))$$

Regarding each string s of length n as an n-tuple, we have a function

$$c_{A^*} : A^* \to \mathbb{N} \times \mathbb{N},$$

$$c_{A^*}(a_1 a_2 a_3 \ldots a_m) = (m, c_m((a_1, \ldots, a_m))$$

This is injective and total, so we are done. (You should verify yourself that this function is injective and total, and in an exam you should prove these claims). □

5 Page 25

Exercise 5.2. *Suppose that we have a programming language, such that every program describes a function $N \to N$. Show that there must be functions $N \to N$ that are described by no program.*

Proof. Any program can be thought of as a finite string from a finite alphabet of letters. We can therefore infer that the set of programs is enumerable. By contrast, the set of all functions $N \to N$ is non-enumerable (see lectures).

So there is no surjective map from programs to functions (if there were, then the set of functions would be enumerable). In particular, the map that takes programs to the function they compute is not surjective, and hence there are functions for which there is no program. □

Worked Solutions 2 - finite automata

Richard M^{c}Kinley

March 16, 2005

1 Introduction

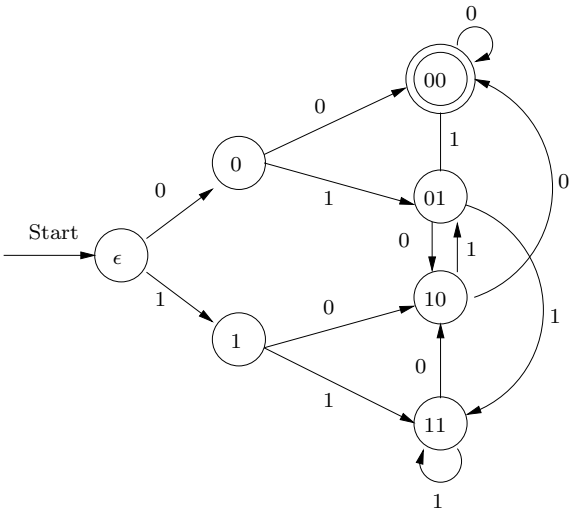
These are selected model answers to exercises from lecture handout 3. Each section corresponds to a slide from that handout.

Exercise 1.1. *Give DFA's accepting the following languages over the alphabet* $\{0, 1\}$.

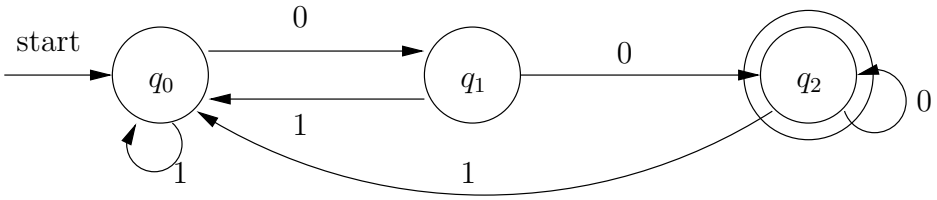
1. *The set of all strings ending in* 00.
2. *The set of all strings with two consecutive* 0*'s (not necessarily at the end).*
3. *The set of strings with* 011 *as a substring.*

Proof. For each part, we give a transition diagram which is a solution to the problem (and for the first, we give two alternative solutions). Following each diagram is a discussion of the meaning of the states of the machine; you should try to work this out for yourselves before reading.

1. The following is a "conceptually clean" solution, in the sense that each state corresponds to different ending of the string:

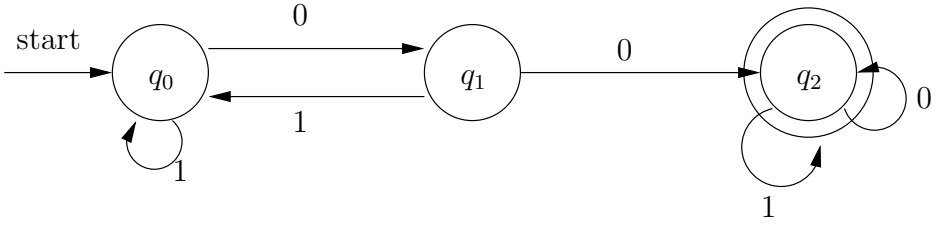


An alternative solution is given below:



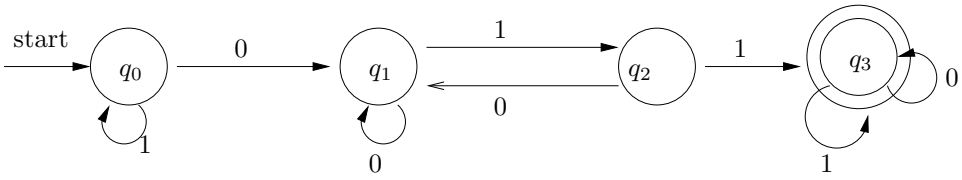
q_0 : String has no trailing zeroes.
q_1 : String has one trailing zero.
q_2 : String has two trailing zeroes.

2. Here is a machine which accepts the given language.



q_0 : String contains no 00, and the last symbol was a one
q_1 : String contains no 00, and the last symbol was a zero
q_2 : String contains a 00

3.



q_0 : String contains no 011, and the last symbol was a one
q_1 : String contains no 011, and the last symbol was a zero
q_2 : String contains no 011, and the string so far ends 01
q_3 : String contains 011

□

Exercise 1.2. *Give a DFA accepting the following language over the alphabet* $\{0,1\}$*: The set of all strings whose tenth symbol from the right is a* 1*.*

Proof. **Main Idea**: The states of the machine track the last 10 symbols read by the machine.

$$Q = \{x_1 x_2 \dots x_n \,|\, 0 \leq n \leq 10, \text{ and } x_i \in \{0,1\} \text{ for all } i \in \{1, \dots n\}\}$$

$$q_0 = \varepsilon \text{ (the empty string)}$$

$$\Sigma = \{0, 1\}$$

$$F(\text{the accepting states}) = \{1x_2 \dots x_{10} \mid x_i \in \{0, 1\} \text{ for all } i \in \{2, \dots 10\}\}$$

$$\delta(x_1 x_2 \dots x_n, y) = \begin{cases} x_1 x_2 \dots x_n y & \text{if } n < 10 \\ x_2 \dots x_n y & \text{if } n = 10 \end{cases}$$

□

Exercise 1.3. *Consider the DFA with the following transition table:*

	0	1
$\rightarrow A$	A	B
$*B$	B	A

(1) Informally describe the language accepted by this DFA; (2)prove by induction on the length of an input string that your description is correct.

Proof. You should draw the transition graph before reading this answer; it will help with your understanding.

We claim that the language of the DFA is $\{w \mid w \text{ contains an odd number of 1's}\}$. (Now you know about regular expressions, write this as a regular expression).

The proof is by induction, but the induction hypothesis is stronger than you might expect:

Claim:

$$\hat{\delta}(A, w) = \begin{cases} A & \text{if } w \text{ contains an even number of 1's} \\ B & \text{if } w \text{ contains an odd number of 1's} \end{cases}$$

Base case $w = \varepsilon$

$\hat{\delta}(A, \varepsilon) = A$, as ε has an even number of 1's

Inductive Step: We show the case for a string which ends in 1, and leave the case of a string ending in 0 to the reader as an exercise.

$$\hat{\delta}(A, w) = \hat{\delta}(A, v1) = \delta(\hat{\delta}(A, v1), 1) = \begin{cases} \delta(A, 1) & \text{if } v \text{ contains an even number of 1's} \\ \delta(B, 1) & \text{if } v \text{ contains an odd number of 1's} \end{cases}$$

□

Worked Solutions 3: Non deterministic finite automata

Emma Jones

April 13, 2005

These are selected worked solutions to the exercises from lecture handout 4. Each section corresponds to a slide from that handout.

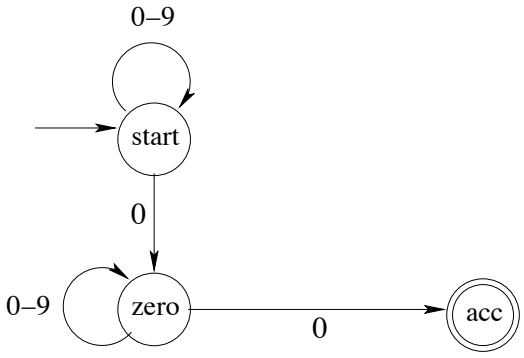
Exercise set 1

Give NFA to accept the following languages.

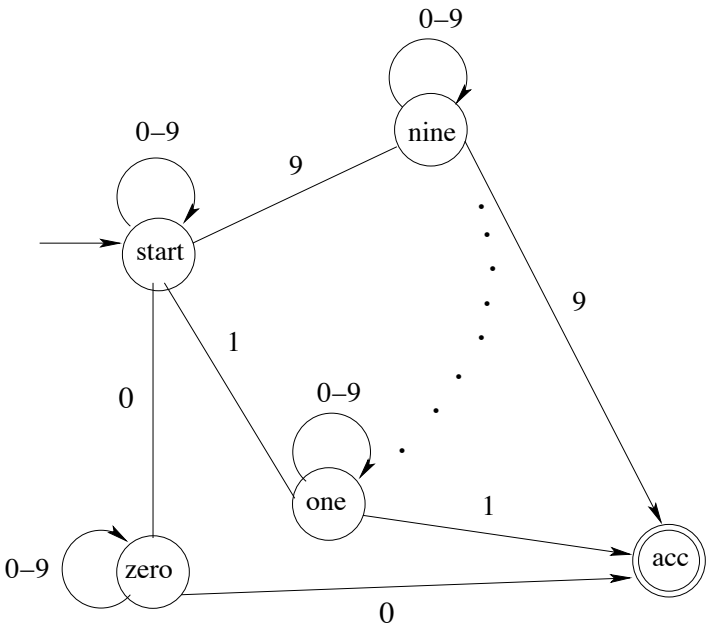
1. *The set of strings over an alphabet* $\{0, 1, \ldots, 9\}$ *such that the final digit has appeared before.*
2. *The set of strings over an alphabet* $\{0, 1, \ldots, 9\}$ *such that the final digit has not appeared before.*
3. *The set of strings of* 0*'s and* 1*'s such that there are two* 0*'s separated by a number of positions that is a multiple of* 4*.*

Exercise 1.1

In the exam I would recall and give the definition of a non deterministic finite automaton (NFA). Further to that I would note that an NFA, given some string of input symbols, will be in a set of states. The input symbols of our machine will be the set $\{0, 1, \ldots, 9\}$. If we assume that the last symbol of the input string is a 0, then to accept the string we must previously have had the input 0. The smallest such input string is 00 but there could be any string of symbols $0 - 9$ before the first 0 or after the first 0. A machine that accepts such strings would look like:

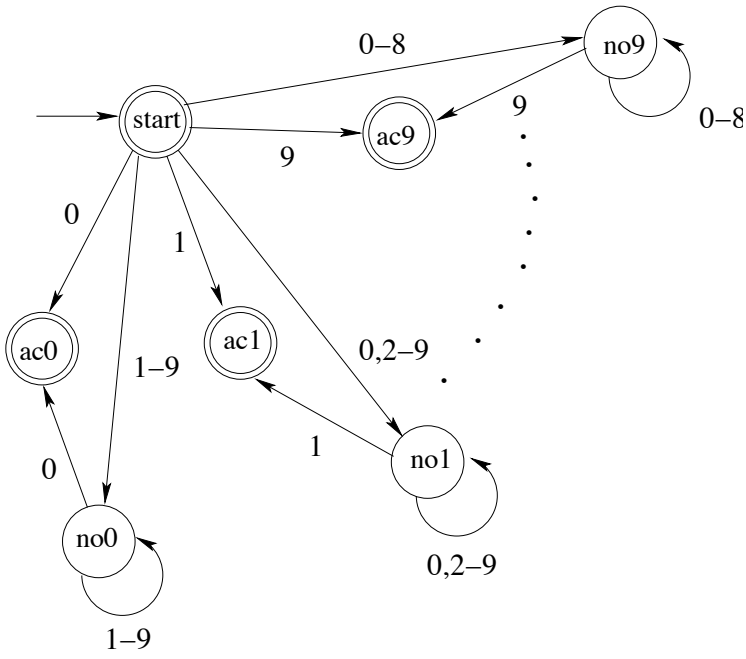


We can only accept the input string if it consists of a string of any length of symbols from the alphabet, then a 0, then a string of any length of symbols from the alphabet then another 0 (What set of states would the machine be in?). We want to do the same for the numbers $1, 2, \ldots, 9$ and we end up with the following machine (the "..." stands for other pieces of machine with the same structure to those given):



Exercise 1.2

This example is similar to the last one. It helps to consider, for instance, a machine that will accept strings consisting of any number of the digits $1-9$ and then a 0. Once you have done this the whole machine is made by duplicating the same structure for the digits $1-9$.



If you consider the triangle of states $\{$*start,no0,ac0*$\}$ we can see that *start* will accept the empty string, *ac0* will accept either the string 0 or a string of the symbols $1-9$ (during which we stay in state *no0*) followed by a single 0.

Exercise 1.3

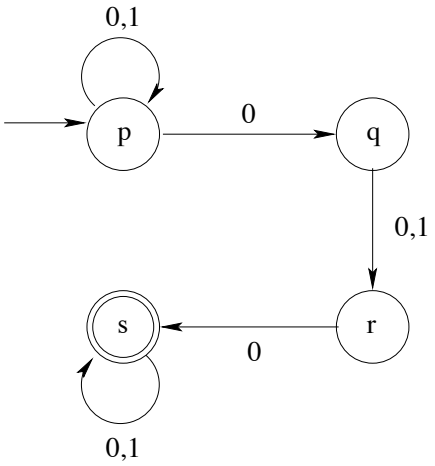
The final exercise in the set is a little different from the others but no more difficult. The key is to consider the smallest strings that should be accepted and then to generalise the idea.

Exercise set 2

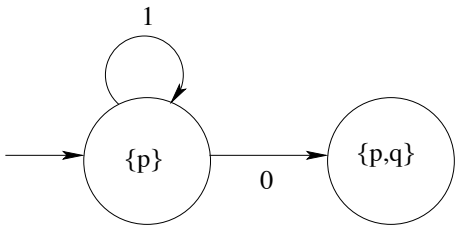
Convert the following NFA to a DFA:

	0	1
$\rightarrow p$	$\{p, q\}$	$\{p\}$
q	$\{r\}$	$\{r\}$
r	$\{s\}$	$\{\}$
$*s$	$\{s\}$	$\{s\}$.

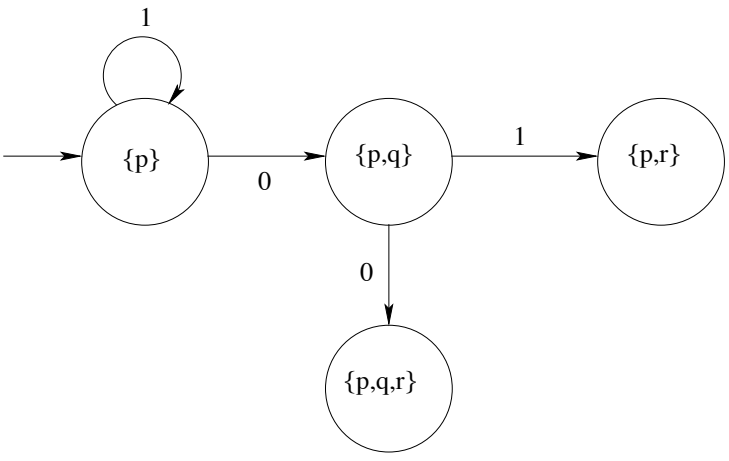
It can help to draw the transition graph of the NFA and this is good practise:



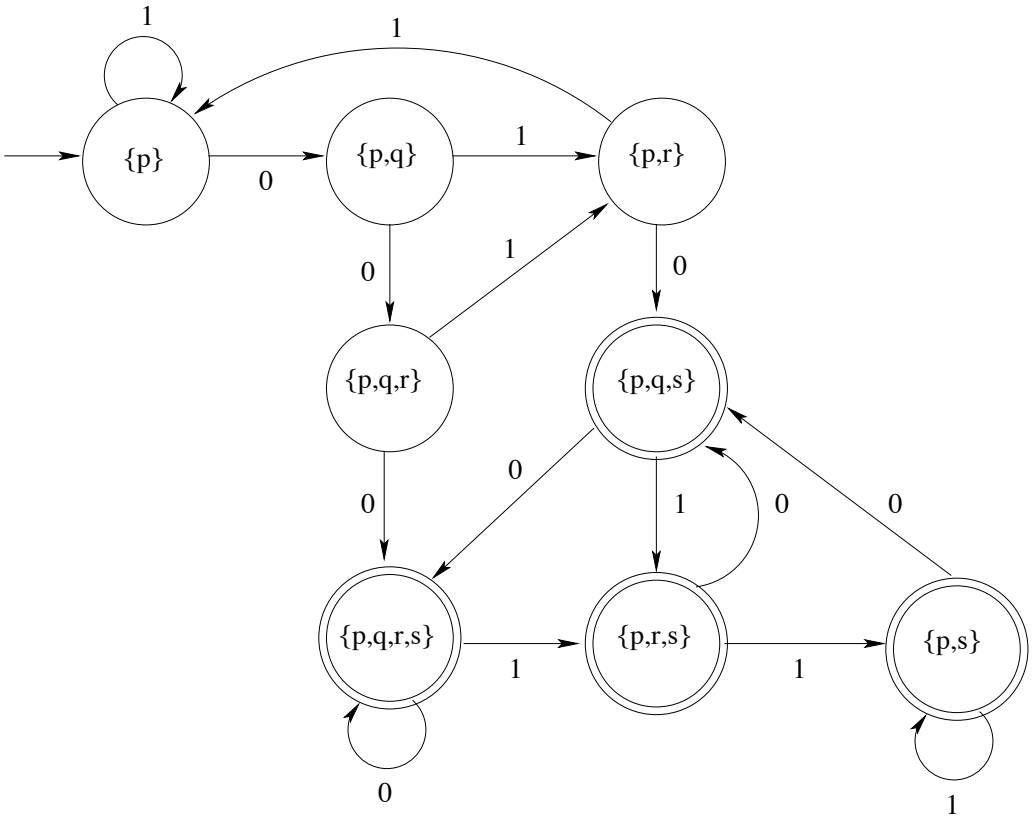
To convert the NFA to a DFA we use the powerset construction demonstrated in lectures and in an exam I would give a short explanation of this method. In what follows we are ignoring the unreachable states. We note that the start state of the DFA is defined to be the set containing just the start state of the NFA: $\{p\}$. To draw the transition graph of the DFA we start with this state and consider what set of states the NFA will enter from state p given input 0 or 1. We get the following:



We now have a new state $\{p, q\}$ and we continue by assuming that the NFA is in the set of states $\{p, q\}$ and considering what set of states it will enter on the inputs 0 and 1. This leads to:



Each time we add a new state (like $\{p, r\}$ and $\{p, q, r\}$ in the above) we must work out what the DFA should do when in that new state and given input 0 or 1. This process ends when no new states are added and every state in the DFA has one transition leaving it labelled 0 and one labelled 1. The final step is to mark the accepting states, any state labelled with a set containing an accepting state of the NFA, in this case s. The complete machine is:



Exercise set 3

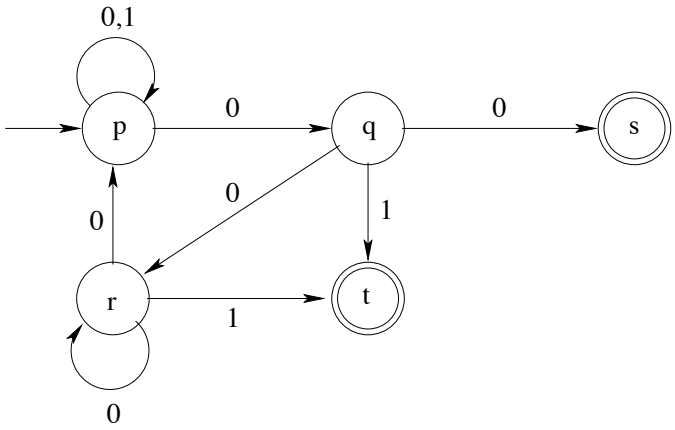
This is very similar to the previous exercise.

Exercise set 4

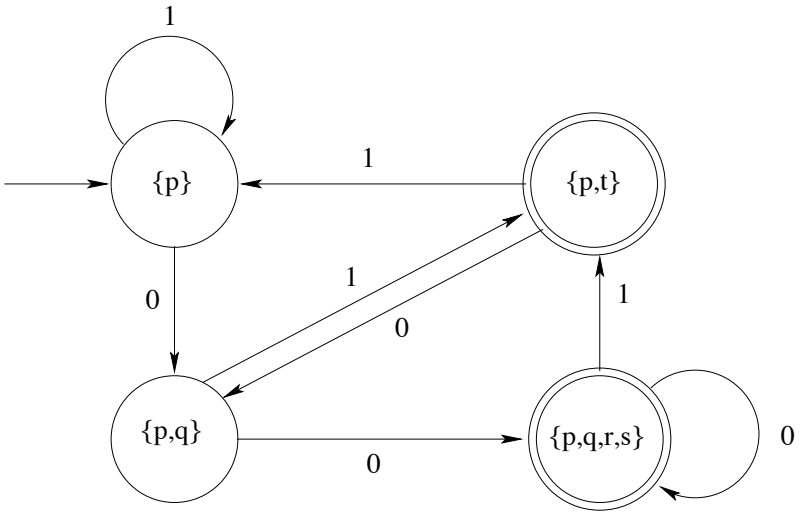
Convert the following NFA to a DFA:

	0	1
$\rightarrow p$	$\{p, q\}$	$\{p\}$
q	$\{r, s\}$	$\{t\}$
r	$\{p, r\}$	$\{t\}$
$*s$	$\{\}$	$\{\}$
$*t$	$\{\}$	$\{\}$

Again, it can help to draw the transtion graph of the NFA:



Again the DFA is found by the powerset construction. The machine is:



Describe informally the language accepted by this NFA.

Firstly, the machines will only accept strings that end in 00 or in 01. One way of working this out is by considering what input strings can lead to each of the states in the DFA:

- $\{p\}$: The string is the empty string, 1 or ends in 11.
- $\{p, q\}$: The string ends in 10.
- $\{p, q, r, s\}$: The string ends in 00.
- $\{p, t\}$: The string ends in 01.

Only the final two states are accepting states. We now need to show that the machines accept **all** strings that end in 00 or 01. We can see this by looking at the NFA, it is clear that we will be in a set of states that contains the state p if we input any string of 0's and 1's. From the state p on input 00 or 01 we would enter a set of states containing an accepting state.

Worked Solutions 4: Regular expressions

Emma Jones

April 24, 2005

These are selected worked solutions to the exercises from lecture handout 5. Each section corresponds to a slide from that handout.

Exercise set 1

Write regular expressions for the following languages:

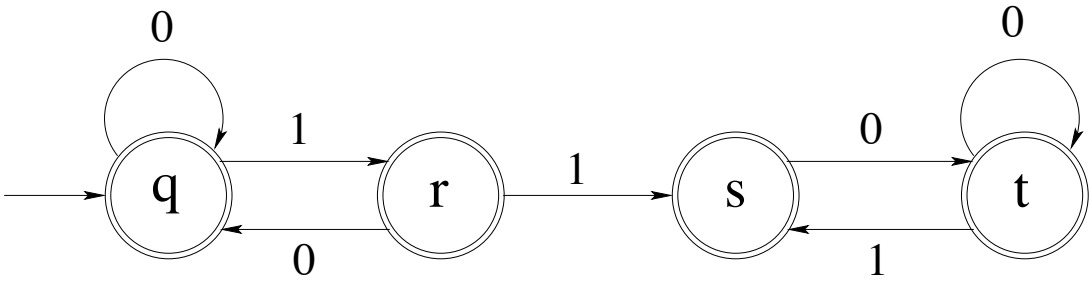
1. *The set of strings over alphabet* $\{a, b, c\}$ *with at least one* a *and at least one* b.
2. *The set of strings of 0's and 1's whose tenth symbol from the right end is 1.*
3. *The set of strings of 0's and 1's with at most on pair of consecutive 1's.*

The first two are quite easy and you should be able to write them staight down. The third exercise is harder:

Exercise 1.3

It can be far easier to write down a regular expression for a language if you first design either an NFA or DFA that accepts it. In this case we will use an NFA (why?). In the NFA there are four states corresponding to:

- q: There have been no pairs of consecutive 1's and the last symbol was a 0.
- r: There have been no pairs of consecutive 1's and the last symbol was a 1.
- s: There have been one pair of consecutive 1's and the last symbol was a 1.
- t: There have been one pair of consecutive 1's and the last symbol was a 0.



Note that all of the states are accepting states. Check that you are sure the machine accepts the correct language.

Now we consider the regular expression describing the strings accepted by each of the 4 states:

- q: $(0+10)^*$
- r: $(0+10)^*1$
- s: $(0+10)^*11(00^*1)^*$
- t: $(0+10)^*11(00^*1)^*00^*$

To see the first of these, for instance, we note that the state q will accept any strings made up of any number of 0's (from the loop on state q) or the string 10 (from state q to state r and then back to state q). This is written as the regular expression $(0+10)^*$, that is "any number of 0 or 10".

Since all of the states are accepting states we can see that all strings accepted are:

$$(0+10)^* + (0+10)^*1 + (0+10)^*11(00^*1)^* + (0+10)^*11(00^*1)^*00^*.$$

This expression can (and should) be simlified by extracting the common expression $(0+10)^*$ we get:

$$(0+10)^*(\epsilon + 1 + 11(00^*1)^* + 11(00^*1)^*00^*),$$

then inside the second set of brackets we can extract the common expression 1 to get:

$$(0+10)^*(\epsilon + 1(\epsilon + 1(00^*1)^* + 1(00^*1)^*00^*)),$$

this process continues until we have:

$$(0+10)^*(\epsilon + 1(\epsilon + 1(00^*1)^*(\epsilon + 00^*))).$$

Note that once you have worked through the lectures after this one you should be able to use a more systematic approach to produce a regular expression that denotes the same language as is accepted by an NFA (via regular grammars). Try this exercise again once you are happy with that process.

Exercise set 2

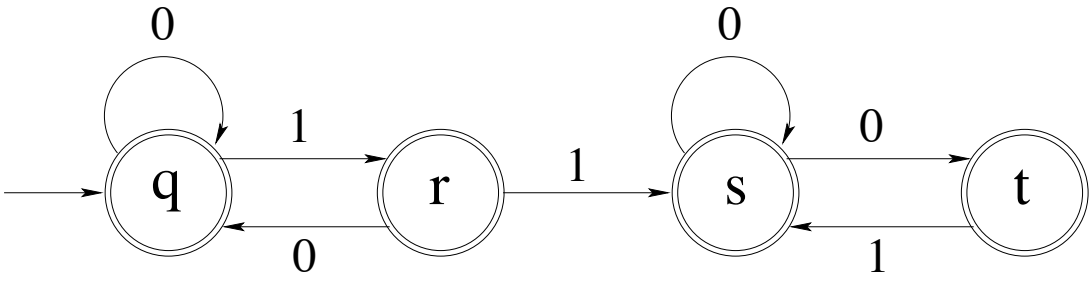
Write regular expressions for the following languages:

1. *The set of all strings of 0's and 1's such that every pair of adjacent 0's appears before any pair of adjacent 1's.*
2. *The set of strings of 0's and 1's whose number of 0's is divisible by five.*

You can either do these questions in the ad hoc manner used above or you can use the systematic method of designing an NFA or DFA that accepts the language, finding the equivalent regular grammar and then using that to find the regular expression. The latter method is not introduced until the section on regular grammars so we will not use it here but it is good extra practise to do these examples both ways.

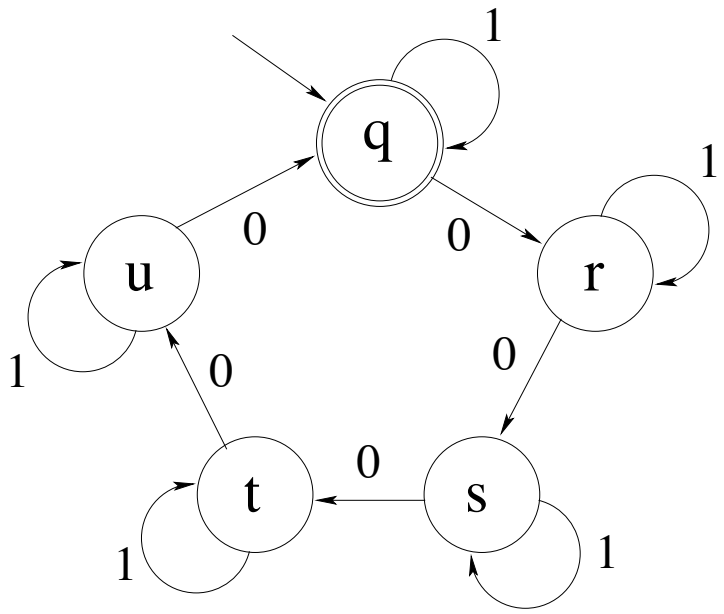
Exercise 2.1

Once you have designed the machine to accept this language the process is exactly the same as above. The NFA is:



Exercise 2.2

This question is a little harder. It is easier if we work with the DFA this time:



This time we have only one accepting state, state q. We need to consider the regular expression denoting what is accepted after each loop:

- After 0 loops: 1^*
- After 1 loop: $1^*01^*01^*01^*01^*$
- After 2 loops: $1^*(01^*01^*01^*01^*)^2$
- $\cdots$

- After arbitarily many loops: $1^*(01^*01^*01^*01^*)^*$

To see the first for instance, the only strings accepted by state q after no loops are strings of any number of 1's (from the loop on state q). For the others we consider what input strings will take us from state q back to state q in a loop via the other states.

Finally we note that we can loop round an arbitary number of times and hence get the regular expression $1^*(01^*01^*01^*01^*)^*$.

Exercise set 3

For any alphabet Σ, which are the subsets S of Σ^* such that the set S^* is finite?

Solution: $\{\}, \{\epsilon\}$

Reasoning: In an exam you should note the definition of the Kleene Star. From the definition, if S contains a non empty word w then S^* contains arbitarily long words $w, w^2, w^3, w^4, \ldots$. Hence S cannot contain a non empty word. This leaves only two possible sets: $\{\}$ and $\{\epsilon\}$. We must then show that in both cases the set S^* is finite, you can do this by finding the set S^* using the definition of the Kleene star.

Worked Solutions 5: ϵ-NFAs

Emma Jones

April 24, 2005

These are selected worked solutions to the exercises from lecture handout 6. Each section corresponds to a slide from that handout.

Exercise set 1

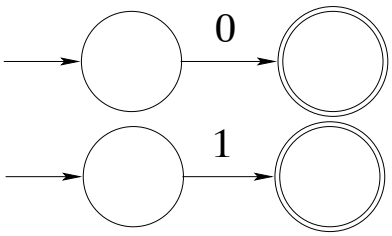
Convert each of the following regular expressions to an ϵ-NFA:

1. 01^*
2. $(0 + 1)01$
3. $00(0 + 1)^*$

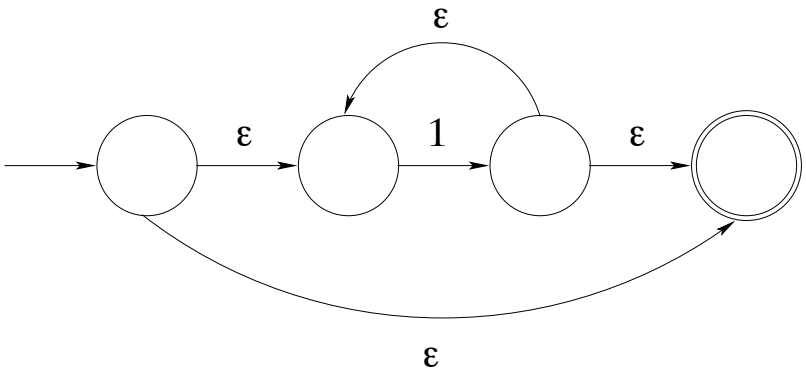
A method was given in lecture 6 by which to achieve this. In an exam I might give a short introduction to the method defining an ϵ-NFA if I had not done so already, then it is a case of following the steps.

Exercise 1.1

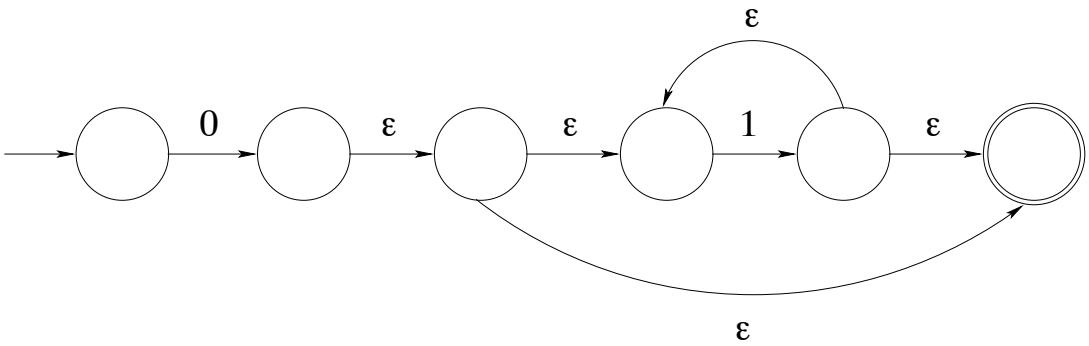
First we need to build the base parts of the machine. In this case an ϵ-NFA that accepts 0 and an ϵ-NFA that accepts 1:



We now need a machine that accepts 1^*, so we use the base machine for 1 and the method for building a machine for E^* (where E is some expression) on slide 17 of lecture notes 6. We get the following:

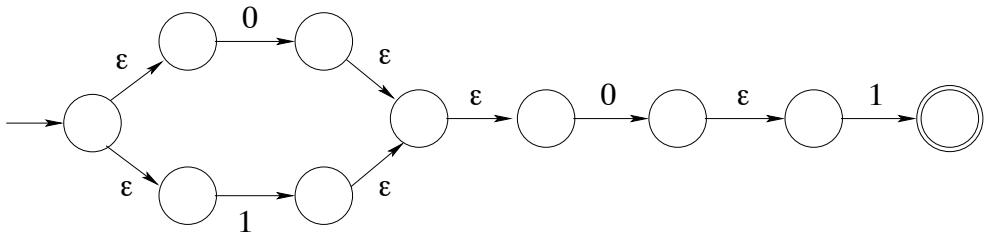


Finally we put the machine accepting 0 and the machine accepting 1^* together using the method of contructing a machine to accept $E \cdot E'$ (where E and E' are expressions) given on page 16 of lecture notes 6:



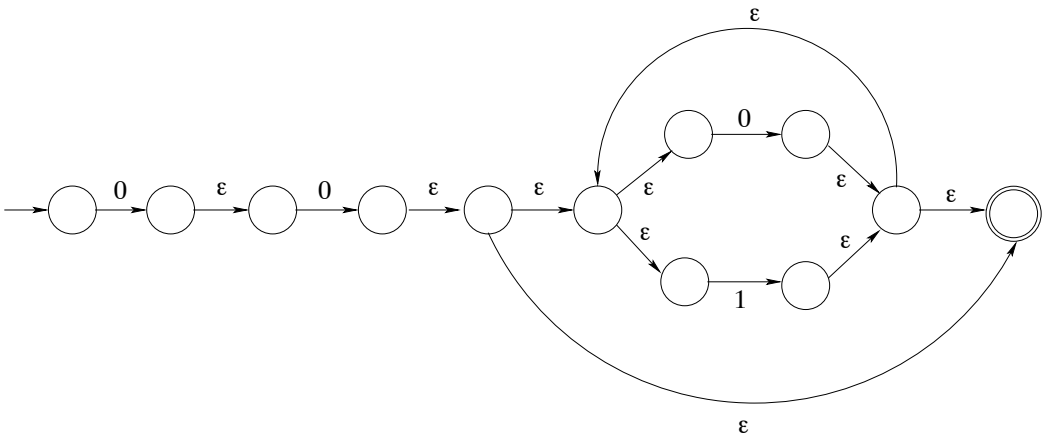
Exercise 1.2

The machine is found by a similar method to above. You may need to use other methods from slides 12-18 from lecture notes 6. The machine is:



Exercise 1.3

The machine is:



Exercise set 2

Consider the following ϵ-NFA.

	ϵ	a	b	c
$\rightarrow p$	$\emptyset$	$\{p\}$	$\{q\}$	$\{r\}$
q	$\{p\}$	$\{q\}$	$\{r\}$	$\emptyset$
$*r$	$\{q\}$	$\{r\}$	$\emptyset$	$\{p\}$

1. *Compute the ϵ-closure of each state.*
2. *Give all the strings of length three or less accepted by this automaton.*
3. *Convert the automaton to a DFA.*

It might help you to draw the ϵ-NFA. To calculate the ϵ-closure of a state we consider what states we can get to from it following any number of ϵ transitions (including no ϵ transitions):

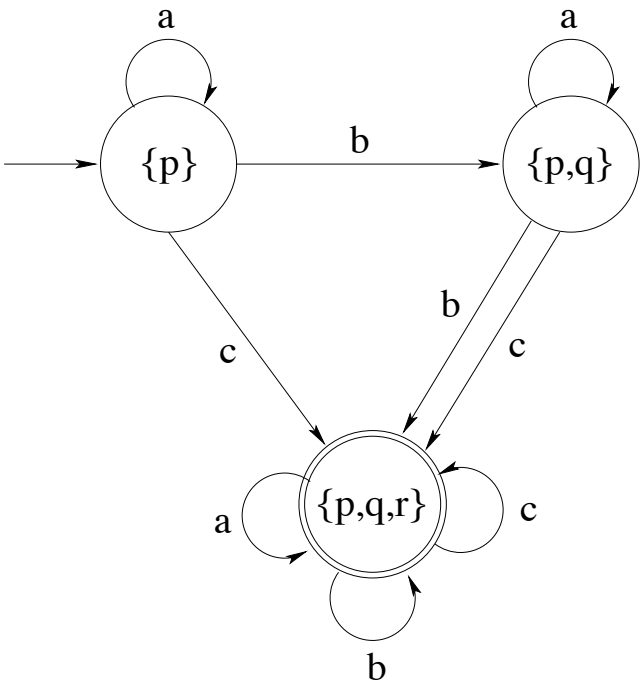
- $\mathrm{cl}(p) = \{p\}$
- $\mathrm{cl}(q) = \{p, q\}$
- $\mathrm{cl}(r) = \{p, q, r\}$

The easiest way to ensure that you do not miss any strings of length three is to list all of them and then test each one. Whichever way you tackle producing this list you should keep in mind strings such as cba, this is accepted since input c will take us to a set containing state r, from state r we make an ϵ transition to a set containing state q, then input b takes us back to a set containing state r and finally on input a we remain in a set containing state r the accepting state. You should also recall that we may allow a sequence of any number of ϵ transitions so strings such as bbc are also accepted.

To convert the ϵ-NFA to a DFA we use the modified powerset construction. We ignore unreachable states. This process is very similar to the powerset construction however we need

to take the ϵ-closure of each state into account. In an exam you might give a short introduction to the modified powerset construction (e.g. summarise slides 24 and 25 from lecture notes 6).

To build our DFA we start with the initial state, which by definition is the ϵ-closure of the set containing the start state of the ϵ-NFA, in this case $\mathrm{cl}(\{p\}) = \{p\}$. We then proceed as before - considering which state we move to on input a, b or c when in each state that we encounter. The only difference is that we must take into account possible ϵ transitions e.g. when in state $\{p\}$ given input b we end up not in state $\{q\}$ but in $\mathrm{cl}(\{q\}) = \{p, q\}$:



Exercise set 3

Repeat the previous exercise for the following ϵ-NFA...

This exercise requires exactly the same steps as the last one.

Exercise set 4

In an earlier exercise, we converted the regular expressions 01^*, $(0 + 1)01$ and $00(0 + 1)^*$ into ϵ-NFA's. Convert each of those ϵ-NFA's into a DFA.

Again, you need to use exactly the same steps as above. This is good practise.

Worked Solutions 6: Context-free grammars

Emma Jones

May 3, 2005

These are selected worked solutions to the exercises from lecture handout 7. Each section corresponds to a slide from that handout.

Exercise set 1

Consider the grammar

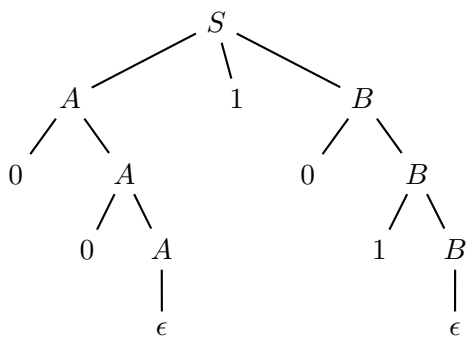
$$S \to A1B$$
$$A \to 0A \,|\, \epsilon$$
$$B \to 0B \,|\, 1B \,|\, \epsilon.$$

Give parse trees for the strings

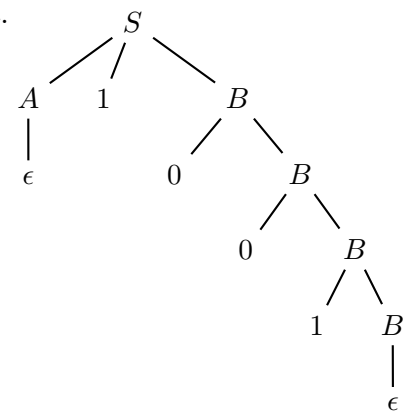
1. 00101
2. 1001
3. 00011

In an exam I would briefly explain what a parse tree is e.g. slide 18. Then I would note that A can produce only 0's. Since we must start with the start symbol S the first production used will always be $S \to A1B$. For each of the strings we need to decide which 1 in the string is the 1 in $A1B$. Since A can only produce 0's we can see that it must always be the left most 1. Once we know this it should be a simple matter to produce the trees. They are:

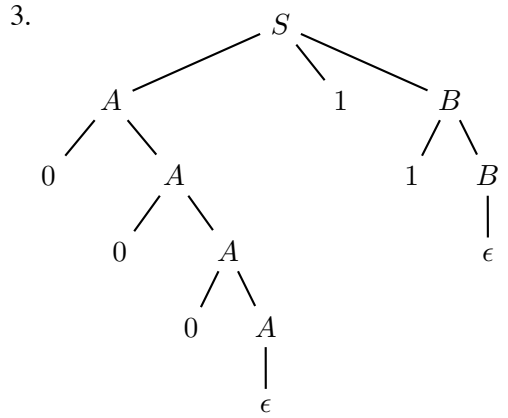
1.



2.



3.



Exercise set 2

For the CFG G defined by the productions

$$S \to aS \,|\, Sb \,|\, a \,|\, b,$$

prove by induction on the size of the parse tree that the no string in language $L(G)$ has ba as a substring.
We argue by induction on the size of the parse tree. The base cases should be all of the smallest possible parse trees.
Proof:

- **Base cases:**

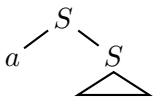
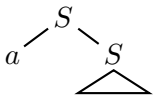
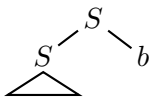
  The two smallest parse trees are:

  Neither the string a nor the string b has the string ba as a substring.

- **Inductive step:**

  There are two cases:

  1. The parse tree is of the form  where  is a parse tree for $w \in \Sigma^*$. Since the parse tree for w is smaller than  the induction hypothesis implies that w does not contain ba hence it is not possible for aw to contain the substring ba.
  2. The argument is similar but we are considering parse trees of the form 

  If you complete the last part of the inductive step you should see that we have completed the proof since no other forms of parse trees are possible in this grammar.

Exercise set 3

Give a context-free grammar for the language of all palindromes over the alphabet $\{a, b, c\}$. *(A palindrome is a word that is equal to the reversed version of itself, e.g. abba, bab.)*
In an exam I would give the definition of a context-free grammar, this is also useful as it involves recalling what types of production rules you are allowed to use.
We then consider the smallest strings that should be in the language: $\{\epsilon, a, b, c\}$. This will immediately give us four production rules. We then note that the key idea behind forming palindromes is that of symmetry. If you have a play with possible production rules you should quickly notice that the following context-free grammar fits our requirements:

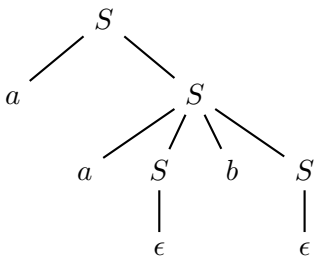
$$S \to \epsilon \,|\, a \,|\, b \,|\, c \,|\, aSa \,|\, bSb \,|\, cSc.$$

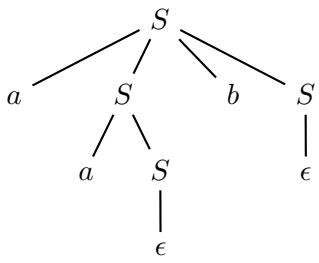
Exercise set 4

Consider the grammar

$$S \to aS \,|\, aSbS \,|\, \epsilon.$$

Show that this grammar is ambiguous.
In an exam I would give a definition of ambiguous - from notes. We then need to find a string that has two different parse trees. The first thing we notice about the above grammar is that aS is part of two of the production rules. In light of this we look for a string containing aa that can be produced by using the two production rules in either order. For instance:



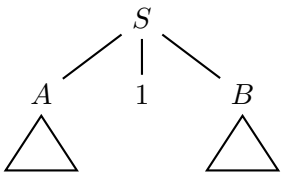


Exercise set 5

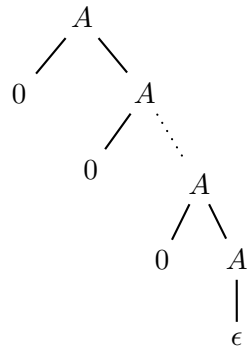
Consider the grammar

$$S \to A1B$$
$$A \to 0A \,|\, \epsilon$$
$$B \to 0B \,|\, 1B \,|\, \epsilon.$$

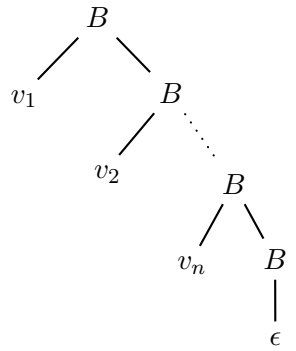
Show that this grammar is unambiguous.
This question is more difficult than the others.
In an exam I would start off by giving the definition from notes of what it means for a grammar to be ambiguous. From this we can see that to show that the above grammar is unambiguous we must prove that it is not possible to have two different parse trees for a string produced by the grammar.
Proof: Since S is the start symbol and $S \to A1B$ the only rule containing S any parse tree for a string w produced by the grammar must have the form



A can produce only strings of 0's and clearly the parse tree below A will always have the structure



Thus if there are two parse trees for a string w they must differ below B. So far we can see that the string w must be of the form $w = 00 \dots 01v$, let us assume that v is of length n and $v = v_1 v_2 \dots v_n$ where $v_i \in \{0, 1\}$ for $i \in \{1, 2, \dots, n\}$. However since $B \to 0B \mid 1B \mid \epsilon$ with both the 0 and the 1 on the left we will always have a parse tree of the form



It is clear that two different trees of this structure cannot produce the same string hence we are unable to produce two differing parse trees for a string w.

Worked Solutions 7: Regular grammars

Emma Jones

May 9, 2005

These are selected worked solutions to the exercises from lecture handout 8. Each section corresponds to a slide from that handout.

Exercise set 1

Turn some of the NFA's in this lecture into regular grammars.
To do this for any NFA in the course you need only follow the method on slide 9 of lecture handout 8. You should practise doing this and make sure that you consider whether the regular grammar you end up with defines the same language as the NFA accepts.

Exercise set 2

Consider the NFA give by the transition table below.

	0	1
$\rightarrow X$	$\{X\}$	$\{X, Y\}$
$*Y$	$\{Y\}$	$\{Y\}$

1. *Give a regular grammar for it.*
2. *Calculate a regular expression that describes the language.*

You may find it helpful to draw the transition graph for the NFA. To find a regular grammar we follow the method on slide 9 of handout 8. The alphabet will be the same as that of the machine $\{0, 1\}$.

1. The non-terminal symbols are the states of the NFA: $\{X, Y\}$.
2. The start symbol is the intial state of the NFA: X.
3. For every transition $q \stackrel{a}{\rightarrow} q'$ in the NFA, introduce a production $q \rightarrow aq'$: e.g The transition $X \stackrel{0}{\rightarrow} X$ gives production $X \rightarrow 0X$, $X \stackrel{1}{\rightarrow} X$ gives $X \rightarrow 1X$ etc.
4. For every final state q add production $q \rightarrow \epsilon$. The only final state is Y so we add $Y \rightarrow \epsilon$.

In compact form the grammar is:

$$\begin{aligned} X &\rightarrow 0X|1X|1Y \\ Y &\rightarrow 0Y|1Y|\epsilon \end{aligned}$$

To calculate a regular expression that describes the language we use the method "From reg. grammar to reg. expression" on slides 13 to 21 of handout 8.

We already have the grammar in compact notation so we proceed to the second step and replace $\rightarrow$ by $=$ and $|$ by $+$:

$$X = 0X + 1X + 1Y \quad (1)$$
$$Y = 0Y + 1Y + \epsilon \quad (2)$$

Our start symbol is X so we need to get a solution for X that does not rely on Y. We first note from slide 17 of lecture notes 8:
Fact: Generally, the solution of any equation of the form $A = cA + B$ is $A = c^*B$.

We reduce the two equations to one equation:

- We can rewrite (2) as $Y = (0 + 1)Y + \epsilon$.
- We use the fact above to rewrite this as $Y = (0 + 1)^*\epsilon = (0 + 1)^*$.
- We can now substitute this into equation (1) to get

$$\begin{aligned} X &= 0X + 1X + 1(0 + 1)^* \\ &= (0 + 1)X + 1(0 + 1)^* \end{aligned}$$

Using fact above this can then be rewritten as $X = (0 + 1)^*1(0 + 1)^*$. Finally you should take a moment to consider whether this regular expression does describe the language accepted by the NFA.

Exercise set 3

Repeat the exercise for the NFA below.

	0	1
$\rightarrow *X$	$\{X\}$	$\{Y\}$
Y	$\{Y\}$	$\{Z\}$
Z	$\{Z\}$	$\{X\}$

Also, describe the accepted language in English.
You would use the same method as above to find the grammar and the regular expression. You will need to reduce three equations to two and then two to one following the method in lectures.

Worked Solutions 8: Turing machines

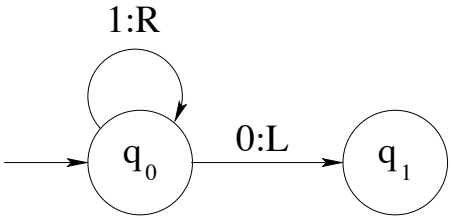
Emma Jones

May 13, 2005

These are selected worked solutions to the exercises from lecture handout 10. Each section corresponds to a slide from that handout.

Exercise set 1

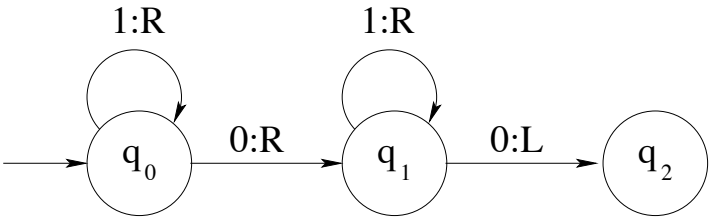
Design a TM that will do the following. Given a tape containing a block of 1*'s and otherwise blank, if the machine is started the leftmost* 1 *on the tape, it will eventually halt scanning the rightmost stroke on the tape, having neither printed nor erased anything.*
Start: One block of 1's, otherwise blank, scanning leftmost 1. **Halt:** Scanning rightmost 1, no printing, no erasing.



n.b. This machine does not halt in a standard configuration but is a useful example of a subroutine.

Exercise set 2

Design a TM that will do the following. Given a tape containing a block of 1*'s, followed by a 0, followed by another block of* 1*'s, and otherwise blank, if the machine is started the leftmost* 1 *on the tape, it will eventually halt scanning the rightmost stroke on the tape, having neither printed nor erased anything.*
Start: Two blocks of 1's, otherwise blank, scanning leftmost 1. **Halt:** Scanning rightmost 1, no printing, no erasing.

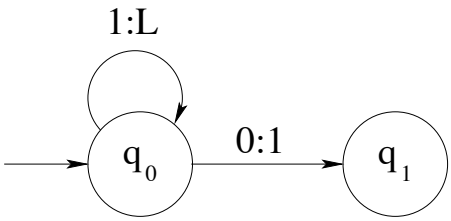


n.b. This machine does not halt in a standar configuration but is a useful example of a subroutine.

Exercise set 3

Design a TM that, started scanning the leftmost 1 *of an unbroken block of* 1*'s on an otherwise blank tape, adds another* 1 *to the block and halts, scanning the leftmost* 1.

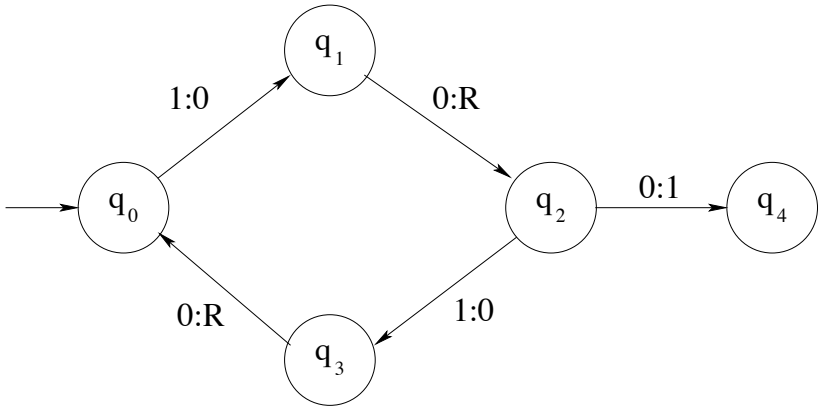
Start: One block of 1's, scanning leftmost, otherwise blank. **Halt:** Added another 1 and halted scanning leftmost.



Exercise set 4

Design a TM that, started scanning the leftmost 1 *of an unbroken block of* 1*'s on an otherwise blank tape, eventually halts, scanning a square on an otherwise blank tape, where the square contains a blank or a* 1 *depending on whether there were an even or an odd number of strokes in the original block. Hint: recall the parity-checker DFA.*

Idea: In state q_0 the machine has deleted an even number of 1's. If no 1 is left then halt. In state q_2 machine has deleted an odd number of 1's. If no 1 is left write a 1 and halt.

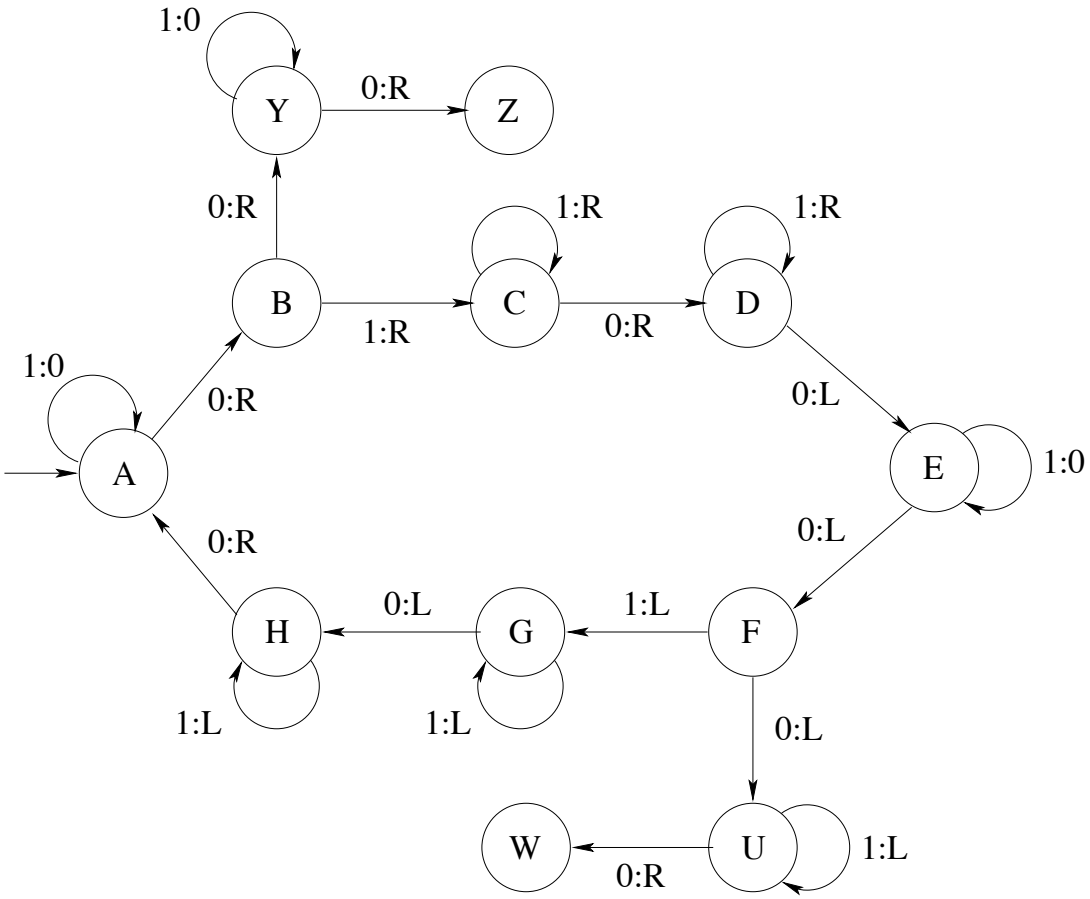


Exercise set 5

Design a TM that will do the following. Given a tape containing a block of n *strokes, followed by a blank, followed by a block of* m *strokes, and otherwise blank, if the machine is started scanning the leftmost* 1 *on the tape, it will eventually halt leaving a block of* $m - n$ *strokes if* $m \geq n$ *or* $n - m$ *strokes if* $n \geq m$*, scanning the leftmost stroke on the tape if any stroke is left.*

This question is harder than the others. The machine I will give as an answer is not necessarily the smallest machine possible but should be relatively easy to follow.

Start: Two blocks of 1's, the first of length n and the second of length m, otherwise blank, scanning leftmost.
Halt: If $m \geq n$ then halt scanning leftmost 1 of a block of $m - n$ 1's. If $n \geq m$ then halt scanning leftmost 1 of a block of $n - m$ 1's.
Idea: The Turing machine deletes the leftmost 1 (A) then the rightmost 1 (E) and then goes back to A. We are deleting pairs of 1's. The process stops when one of the two blocks runs out of 1's. This is decided in state B if the first block becomes empty or state F if the second does. In the first case we then delete another 1 from the second block in state Y and halt in state Z. In the second case we move back to the leftmost 1 (W).



If you didn't manage to design a machine for this task or you are unsure how the above machine works then try simulating some inputs. Some good basic test cases would be 1_A011, 1_A101 and 1_A1011.