This tutorial contains some revision questions to help you refresh your memory on some discrete mathematics topics. Most of the questions are extracted from chapter 0 of "Introduction to the Theory of Computation", by Michael Sipser.

Problem 1. Examine the following formal descriptions of sets so that you understand which members they contain. Write a short informal English description of each set.

a) $\{1, 3, 5, 7, ...\}$

b) $\{..., -4, -2, 0, 2, 4, ...\}$

c) $\{n \mid n = 2m \text{ for some } m \in \mathbb{N}\}$

d) $\{n \mid n = 2m \text{ for some } m \in \mathbb{N}, \text{ and } n = 3k \text{ for some } k \in \mathbb{N}\}$

e) $\{w \mid w \text{ is a string of 0s and 1s and } w \text{ equals the reverse of } w\}$

f) $\{n \mid n \text{ is an integer and } n = n + 1\}$

Solution 1. We do not include 0 in the set $\mathbb{N}$ of natural numbers.

a) The set of odd natural numbers.

b) The set of even integers.

c) The set of even natural numbers.

d) The set of natural numbers that are multiples of 2 and of 3.

e) The set of binary strings that are palindromes (i.e., read the same forward as backwards).

f) The empty set (note that the set of integers that are equal to their successor has no members at all).

Problem 2. Write formal descriptions of the following sets.

a) The set containing the numbers 1, 10, and 100

b) The set containing all integers that are greater than 5

c) The set containing all natural numbers that are less than 5

d) The set containing the string `aba`

e) The set containing the empty string

f) The set containing nothing at all

Solution 2.

a) $\{1, 10, 100\}$

b) $\{n \mid n \in \mathbb{Z}, n > 5\}$

c) $\{n \mid n \in \mathbb{N}, n < 5\}$, or simply $\{1, 2, 3, 4\}$.

d) $\{\texttt{aba}\}$

e) $\{\varepsilon\}$ (we will frequently use ε to denote the empty string in this course).

f) $\emptyset$, or $\{\}$

Problem 3. Let A be the set $\{x, y, z\}$ and B be the set $\{x, y\}$.

a) Is A a subset of B?

b) Is B a subset of A?

c) What is $A \cup B$?

d) What is $A \cap B$?

e) What is $A \times B$?

f) What is the power set of B?

Solution 3.

a) No

b) Yes

c) $\{x, y, z\}$

d) $\{x, y\}$

e) $\{(x, x), (x, y), (y, x), (y, y), (z, x), (z, y)\}$

f) $\{\emptyset, \{x\}, \{y\}, \{x, y\}\}$

Problem 4. If A has a elements and B has b elements, how many elements are in $A \times B$? Explain your answer.

Solution 4. ab elements. The reason is that for every element a of A, there are b pairs in $A \times B$.

Problem 5. If C is a set with c elements, how many elements are in the power set of C? Explain your answer.

Solution 5. 2^c elements. The elements of the power set of C are exactly the subsets of C. Moreover, each subset can be identified by a series of c-many binary choices, one for each element of C, to decide whether to include it or exclude it from the subset.

Problem 6. Let X be the set $\{1,2,3,4,5\}$ and Y be the set $\{6,7,8,9,10\}$. The unary function $f : X \to Y$ and the binary function $g : X \times Y \to Y$ are described in the following tables.

n	$f(n)$
1	6
2	7
3	6
4	7
5	6

g	6	7	8	9	10
1	10	10	10	10	10
2	7	8	9	10	6
3	7	7	8	8	9
4	9	8	7	6	10
5	6	6	6	6	6

a) What is the value of $f(2)$?

b) What are the range and domain of f?

c) What is the value of $g(2,10)$?

d) What are the range and domain of g?

e) What is the value of $g(4, f(4))$?

Solution 6.

a) 7

b) The range of f is Y, and the domain of f is X

c) 6

d) The range of g is Y, the domain is $X \times Y = \{(x,y) \mid x \in X, y \in Y\}\}$

e) 8

Problem 7. For each part, give a relation that satisfies the stated condition.

a) Reflexive and symmetric but not transitive.

b) Reflexive and transitive but not symmetric.

c) Symmetric and transitive but not reflexive.

Solution 7. Recall that a relation $R \subseteq D \times D$ is

1. *reflexive* if $(x, x) \in R$ for every $x \in D$
2. *symmetric* if $(x, y) \in R$ implies $(y, x) \in R$, for all $x, y \in D$.
3. *transitive* if $(x, y) \in R$ and $(y, z) \in R$ implies $(x, z) \in R$, for all $x, y, z \in D$.

a) The relation $R = \{(a, b), (b, c), (a, a), (b, b), (c, c), (b, a), (c, b)\}$ over the set $\{a, b, c\}$. This relation is not transitive since $(a, b) \in R, (b, c) \in R, (a, c) \notin R$.

b) The relation $\geq$ over the set $\mathbb{N}$. This relation is not symmetric since $1 \leq 2$ but $2 \not\leq 1$.

c) The relation $R = \{(a, b), (b, a), (a, a), (b, b)\}$ over the set $\{a, b, c\}$. This relation is not reflexive since $(c, c) \notin R$.

If you would like a refresher on some of the basic discrete mathematics assumed knowledge, please review the "tutorial 0" exercise sheet prior to working on this one.

Problem 1. Discussion: what is the difference between *syntax* and *semantics*?

Solution 1. Syntax refers to the rules for how to construct expressions/formulas, while semantics refers to the rules for how to evaluate expressions/formulas.

Problem 2. Are the following expressions propositional logic formulas (according to the definition given in Lecture 1)?

a) $(p \wedge q)$

b) $\neg(p \wedge q)$

c) $(p \oplus q)$

d) $(p \rightarrow q)$

e) $(p \rightarrow (q \vee r))$

f) $p \wedge q$

g) $\neg\neg\neg\neg\neg p$

h) (p)

Solution 2.

a) Yes.

b) Yes.

c) No, because the symbol $\oplus$ is not one of the defined connectives, nor a shorthand that has been introduced. However, you may recognise that it usually means "exclusive or". *You* can introduce it as an abbreviation (in the same way as we did on pg 15) by saying that we introduce $(F_1 \oplus F_2)$ as an abbreviation of $((F_1 \vee F_2) \wedge \neg(F_1 \wedge F_2))$.

d) Yes. $\rightarrow$ is not part of the original definition, but we have defined $(F \rightarrow G)$ as shorthand for $(\neg F \vee G)$.

e) Yes. $(q \vee r)$ is a formula, and $\neg p$ is a formula, so $(\neg p \rightarrow (q \vee r))$ is a formula, which is the expanded version of this formula.

f) No, the rule which defines formulas using $\wedge$ requires parentheses.

g) Yes.

h) No, we only add parentheses when joining two formulas with a binary connective (e.g. $\wedge$, $\vee$, etc.).

Problem 3. List all the subformulas of the following formula. Which of them are *atomic propositions*?

$$(\neg p \rightarrow (p \wedge q))$$

Solution 3. Strictly speaking, $(\neg p \rightarrow (p \wedge q))$ is an abbreviation of the formula $(\neg\neg p \vee (p \wedge q))$, and so its subformulas are:

1. p (atomic)
2. q (atomic)
3. $\neg p$
4. $\neg\neg p$
5. $(p \wedge q)$
6. $(\neg\neg p \vee (p \wedge q))$

Aside. If one doesn't consider $\rightarrow$ as an abbreviation, then the subformulas are:

1. p (atomic)
2. q (atomic)
3. $\neg p$
4. $(p \wedge q)$
5. $(\neg p \rightarrow (p \wedge q))$

Problem 4. Write a recursive function $\mathbf{SubForms}(G)$ that returns the set of subformulas of G (i.e. in a similar style to the way that the truth value function tv on slide 23 of lecture 1 is defined).

Solution 4. The set of subformulas of a formula G is denoted $\mathbf{SubForms}(G)$, and is defined as follows:

1. If G is an atom, then $\mathbf{SubForms}(G) = \{G\}$.
2. If G is a formula of the form $\neg F$, then $\mathbf{SubForms}(G) = \{G\} \cup \mathbf{SubForms}(F)$.
3. If G is a formula of the form $(E \vee F)$ or $(E \wedge F)$ then $\mathbf{SubForms}(G) = \{G\} \cup \mathbf{SubForms}(E) \cup \mathbf{SubForms}(F)$.

Aside. If one doesn't consider $\rightarrow, \leftrightarrow, \top, \bot, \bigvee, \bigwedge$ as abbrevations, then one also needs rules for these, i.e.,

4. If G is a formula of the form $\top$ or $\bot$, then $\mathbf{SubForms}(G) = \{G\}$.
5. If G is a formula of the form $(E \rightarrow F)$ or $(E \leftrightarrow F)$, then $\mathbf{SubForms}(G) = \{G\} \cup \mathbf{SubForms}(E) \cup \mathbf{SubForms}(F)$.

6. If G is a formula of the form $(\bigwedge_{i=1}^{n} F_i)$ or $(\bigvee_{i=1}^{n} F_i)$ then $\textbf{SubForms}(G) = \{G\} \cup \textbf{SubForms}(F_1) \cup \cdots \cup \textbf{SubForms}(F_n)$.

This illustrates the usefuleness of having a definition with just a few connectives, i.e., there is less to write.

Problem 5. This formula $(\neg p \rightarrow q)$ uses the $\rightarrow$ shorthand. Rewrite the formula using the negation, disjunction and/or conjunction operators.

Solution 5. $(\neg\neg p \vee q)$ Note that $(p \vee q)$ is semantically equivalent to $(\neg\neg p \vee q)$ (see Lecture 2), but not syntactically equal. Because we defined $(F \rightarrow G) = (\neg F \vee G)$, it is necessary to add a negation in front of the antecedent when converting from implication to disjunction, rather than removing one.

Problem 6. Let $G = (\neg p \rightarrow (p \wedge q))$, $\alpha(p) = 1$, $\alpha(q) = 0$. Compute $\mathrm{tv}(G, \alpha)$ recursively by computing $\mathrm{tv}(F, \alpha)$ for each subformula F of G, starting with the atomic propositions.

Solution 6. Removing the abbreviations, the formula G is $(\neg\neg p \vee (p \wedge q))$. It's subformulas are given in the solution above, i.e., $p, q, \neg p, \neg\neg p, (p \wedge q)$, and G itself. So:

1. $\mathrm{tv}(p, \alpha) = 1$.
2. $\mathrm{tv}(q, \alpha) = 0$.
3. $\mathrm{tv}(\neg p, \alpha) = 1 - \mathrm{tv}(p, \alpha) = 1 - 1 = 0$.
4. $\mathrm{tv}(\neg\neg p, \alpha) = 1 - \mathrm{tv}(\neg p, \alpha) = 1 - 0 = 1$.
5. $\mathrm{tv}((p \wedge q), \alpha) = \min\{\mathrm{tv}(p, \alpha), \mathrm{tv}(q, \alpha)\} = 0$
6. $\mathrm{tv}((\neg\neg p \vee (p \wedge q)), \alpha) = \max\{\mathrm{tv}(\neg\neg p, \alpha), \mathrm{tv}((p \wedge q), \alpha)\} = \max\{1, 0\} = 1$.

Problem 7. Extend the definition of the function $\mathrm{tv}(F, \alpha)$ to have appropriate rules to directly compute the truth value of formulas written using the $\rightarrow$ or $\leftrightarrow$ shorthand notation.

Solution 7.

$$\mathrm{tv}((F \rightarrow G), \alpha) = \max\{1 - \mathrm{tv}(F, \alpha), \mathrm{tv}(G, \alpha)\}$$
$$\mathrm{tv}((F \leftrightarrow G), \alpha) = 1 - |\,\mathrm{tv}(F, \alpha) - \mathrm{tv}(G, \alpha)|$$

Problem 8. A formula F is *satisfiable* if there is some assignment α such that $\mathrm{tv}(F, \alpha) = 1$.

Write (full) truth tables for each of the following formulas, and thereby determine whether each of them is satisfiable or not.

a) $(p \to (q \to p))$

b) $((p \wedge q) \wedge \neg p)$

c) $((\neg p \vee \neg q) \leftrightarrow (p \to \neg q))$

d) $(p \wedge (q \vee p))$

e) $(p \wedge \neg(q \to p))$

Solution 8.

a) satisfiable:

p	q	$(q \to p)$	$(p \to (q \to p))$
0	0	1	1
0	1	0	1
1	0	1	1
1	1	1	1

b) not satisfiable:

p	q	$(p \wedge q)$	$\neg p$	$((p \wedge q) \wedge \neg p)$
0	0	0	1	0
0	1	0	1	0
1	0	0	0	0
1	1	1	0	0

c) satisfiable, d) satisfiable, e) not satisfiable

Problem 9. Rewrite the following pseudocode[1] in an equivalent form that uses only a single conditional (if-else statement):

```
if x == True:
    if y == True:
        foo()
    else if z == True:
        bar()
    else:
        foo()
else:
    bar()
```

[1] ok, I admit it, it's Python. Same thing really.

Solution 9. One possible answer is:

```
if (x and (y or not z)):
    foo()
else:
    bar()
```

Problem 10. Examine and run the code in `arithmetic_1.py`. See how it demonstrates that arithmetic expressions are composed of other arithmetic expressions, in a similar way to propositional logic. Also look at how the `eval` method works, passing a context (an *assignment* of values to variables) through the data structure to help it recursively compute the expression.

a) Write a set of rules to inductively define simple arithmetic expressions.

b) Arithmetic expressions usually are not so strict about parentheses placement. Try adjusting your definition to allow formulas like 1+2+3 to be constructed.

c) Write a similar program to build and evaluate propositional logic formulas (e.g. with classes `Atom`, `Negation`, `Disjunction`, `Conjunction`, etc.)

Problem 1. Show that $((p \vee (q \vee r)) \wedge (r \vee \neg p)) \equiv ((q \wedge \neg p) \vee r)$. Justify each step with one of the Equivalences from the table of equivalences in the lecture slides.

Problem 2. A propositional formula F is in *negation normal form (NNF)* if negations only occur immediately infront of atoms. For instance, $\neg(p \wedge q)$ is not in NNF but $(p \wedge \neg q)$ is.

Which of the following propositional formulas are in NNF?

a) p

b) $\neg p$

c) $\neg\neg p$

d) $(\neg q \vee p)$

e) $\neg(q \wedge \neg p)$

f) $\neg(q \wedge \neg(p \vee \neg q))$

Solution 2. yes, yes, no, yes, no, no

Problem 3. The reason that NNF is called a *normal form* is that every propositional formula is equivalent to one in NNF. For instance, $\neg(p \wedge q)$ is not in NNF, but it is equivalent to $(\neg p \vee \neg q)$ which is in NNF.

Here is an algorithm that converts every propositional formula F into an equivalent one in NNF. Intuitively, the algorithm "pushes negations inwards": Substitute in F every occurrence of a subformula of the form

$$\neg\neg G \text{ by } G \tag{1}$$
$$\neg(G \wedge H) \text{ by } (\neg G \vee \neg H) \tag{2}$$
$$\neg(G \vee H) \text{ by } (\neg G \wedge \neg H) \tag{3}$$

until no such subformulas occur, and return the result, lets call it F'.

a) For each of the three substitutions, state the law of equivalence which justifies it.

b) Apply the algorithm to put the following formulas into NNF: i) p, ii) $\neg\neg p$, iii) $\neg(q \wedge \neg p)$, iv) $\neg(q \wedge \neg(p \vee \neg q))$

Solution 3.

a) The first substitution is justified by the law of Double Negation; the remaining two by de Morgan's Laws (All three steps may make use of the Substitution Rule).

b) i) p, ii) p, iii) $(\neg q \vee p)$ iv) $(\neg q \vee (p \vee \neg q))$

Problem 4. What do you need to do to show that F is not a logical consequence of $\{E_1, \cdots, E_k\}$? Illustrate with concrete formulas.

Problem 5. Express logical equivalence $\equiv$ in terms of logical consequence $\models$.

Solution 5. $F \equiv G$ if and only if $F \models G$ and $G \models F$.

Problem 6. Short discussion: What is the difference between $(F \leftrightarrow G)$ and $F \equiv G$? How are these two related?

Solution 6. $(F \leftrightarrow G)$ is a formula of propositional logic, while $F \equiv G$ is statement about formulas of propositional logic. They are related as follows.

A formula F is *valid* if its truth-value is $lastweek'stut1$ under every assignment. Then: $F \equiv G$ if and only if $(F \leftrightarrow G)$ is valid.

Problem 7. Argue why

$$\bot \models F$$

for every propositional formula F.

Solution 7. We give two similar arguments (either one is fine).

1. The statement $\bot \models F$ says for every assignment α, (*): if α makes $\bot$ true then it also makes F true. So take an arbitrary assignment α. Since α makes $\bot$ false, the implication (*) is true.

2. The statement $\bot \models F$ says for every assignment α, if α makes $\bot$ true then it also makes F true. This is exactly the same as saying (**): for every assignment α, either α doesn't make $\bot$ true or it makes F true. We will show (**) holds. To do this, take an arbitrary assignment α. We must show that either a) α doesn't make $\bot$ true or b) α makes F true (inclusive or). But a) holds since no assignment makes $\bot$ true.

Problem 8. Discussion points on performing proofs using Natural Deduction

a) What is the difference between the "assumptions" and "references" columns?

b) When do you know that a proof of the consequence $S \vdash F$ is finished?

c) What assumptions do most proofs start with?

d) Can you assume G and, later in the same proof, also assume $\neg G$?

e) Are there any restrictions on the assumptions you introduce?

Solution 8.

a) The "assumptions" column lists the line numbers of the set of assumptions, i.e., formulas introduced by the "Assumption Introduction" Rule, that the formula in that line depends on. The "references" column lists : the lines referenced by the Inference Rule used to deduce the formula in that step.

b) When you have deduced the consequent formula F, on a line which is only dependent on the assumptions in S.

c) An assumption for each of the formulas in the set S.

d) Yes, this is totally fine (and often useful)!

e) No, although not all assumptions will necessarily be useful.

Problem 9. In the rest of this tutorial, you will be providing proofs in Natural Deduction. **Note that instead of writing $\{E_1, \cdots, E_k\} \vdash F$ we will sometimes be lazy and drop the parentheses and simply write $E_1, \cdots, E_k \vdash F$.**

Prove that $\wedge$ and $\vee$ have the idempotence property, by proving the following four consequences. (Hint: the first three are very simple, the fourth will require an extra assumption.)

a) $A \vdash (A \wedge A)$

b) $(A \wedge A) \vdash A$

c) $A \vdash (A \vee A)$

d) $(A \vee A) \vdash A$

Solution 9.

a)

Line	Assumptions	Formula	Justification	References
1	1	A	Asmp. I	
2	1	$(A \wedge A)$	$\wedge$ I	1, 1

b)

Line	Assumptions	Formula	Justification	References
1	1	$(A \wedge A)$	Asmp. I	
2	1	A	$\wedge$ E	1

c)

Line	Assumptions	Formula	Justification	References
1	1	A	Asmp. I	
2	1	$(A \vee A)$	$\vee$ I	1

d)

Line	Assumptions	Formula	Justification	References
1	1	$(A \vee A)$	Asmp. I	
2	2	A	Asmp. I	
3	1	A	$\vee$ E	1, 2, 2, 2, 2

Unpacking the references for line 3: the disjunction (1), the assumption for the first case (2), the conclusion for the first case (2), the assumption for the second case (2), the conclusion for the second case (2). That's slightly ridiculous, but arises from the fact that both our cases are actually the same, and are so trivial.

Problem 10. $\wedge$ and $\vee$ also have the associativity property. We'll just show one direction of the equivalence, as the proof of the converses of each of these two consequences are almost identical.

a) $(A \wedge (B \wedge C)) \vdash ((A \wedge B) \wedge C)$

b) $(A \vee (B \vee C)) \vdash ((A \vee B) \vee C)$

Solution 10.

a)

Line	Assumptions	Formula	Justification	References
1	1	$(A \wedge (B \wedge C))$	Asmp. I	
2	1	A	$\wedge$ E	1
3	1	$(B \wedge C)$	$\wedge$ E	1
4	1	B	$\wedge$ E	3
5	1	C	$\wedge$ E	3
6	1	$(A \wedge B)$	$\wedge$ I	2, 4
7	1	$((A \wedge B) \wedge C)$	$\wedge$ I	5, 6

b)

Line	Assumptions	Formula	Justification	References
1	1	$(A \vee (B \vee C))$	Asmp. I	
2	2	A	Asmp. I	
3	3	$(B \vee C)$	Asmp. I	
4	4	B	Asmp. I	
5	5	C	Asmp. I	
6	2	$(A \vee B)$	$\vee$ I	2
7	2	$((A \vee B) \vee C)$	$\vee$ I	6
8	4	$(A \vee B)$	$\vee$ I	4
9	4	$((A \vee B) \vee C)$	$\vee$ I	8
10	5	$((A \vee B) \vee C)$	$\vee$ I	5
11	3	$((A \vee B) \vee C)$	$\vee$ E	3, 4, 9, 5, 10
12	1	$((A \vee B) \vee C)$	$\vee$ E	1, 2, 7, 3, 11

For the ($\vee$ E) on line 11, we're referencing disjunction $(B \vee C)$ (line 3), the assumption of B (4), the conclusion drawn from B (9), the assumption of C (5), and the conclusion from C (10).

For the ($\vee$ E) on line 12, we're referencing disjunction $(A \vee (B \vee C))$ (line 1), the assumption of A (2), the conclusion drawn from A (7), the assumption of $(B \vee C)$ (3), and a conclusion from $(B \vee C)$ (11).

We ought to prove the converses of these too (e.g. that $((A \wedge B) \wedge C) \vdash (A \wedge (B \wedge C))$ etc.,) if we want to show that these are logical equivalences, but the proofs would be almost identical.

Problem 11. Prove $\emptyset \vdash (A \vee \neg A)$.

Hint: Proof by contradiction using Reductio Absurdium (RA)) works well here (covered in the slides and video of Lecture 2b). Start by assuming the negation of the thing you're trying to prove.

Solution 11.

Line	Assumptions	Formula	Justification	References
1	1	$\neg(A \vee \neg A)$	Asmp. I	
2	2	A	Asmp. I	
3	2	$(A \vee \neg A)$	$\vee$ I	2
4	1, 2	$\bot$	$\neg$ E	1, 3
5	1	$\neg A$	$\neg$ I	2, 4
6	1	$(A \vee \neg A)$	$\vee$ I	5
7	1	$\bot$	$\neg$ E	1, 6
8		$(A \vee \neg A)$	RA	1, 7

Problem 12. Prove de Morgan's Laws:

a) $(\neg A \vee \neg B) \vdash \neg(A \wedge B)$
Hint: ($\neg$ I) works well here

b) $\neg(A \vee B) \vdash (\neg A \wedge \neg B)$
Hint: assume A, and try to deduce $\neg A$ while cancelling that assumption

c) $(\neg A \wedge \neg B) \vdash \neg(A \vee B)$
Hint: assume $(A \vee B)$

d) $\neg(A \wedge B) \vdash (\neg A \vee \neg B)$

Note: for this question you should use some of the axioms that were not discussed in the lecture. Read the slides or watch the video to learn about the rules involving $\neg$ and $\bot$.

Solution 12.

a)

Line	Assumptions	Formula	Justification	References
1	1	$(\neg A \vee \neg B)$	Asmp. I	
2	2	$\neg A$	Asmp. I	
3	3	$\neg B$	Asmp. I	
4	4	$(A \wedge B)$	Asmp. I	
5	4	A	$\wedge$ E	4
6	2, 4	$\bot$	$\neg$ E	2, 5
7	2	$\neg(A \wedge B)$	$\neg$ I	4, 6
8	4	B	$\wedge$ E	4
9	3, 4	$\bot$	$\neg$ E	3, 9
10	3	$\neg(A \wedge B)$	$\neg$ I	4, 9
11	1	$\neg(A \wedge B)$	$\vee$ E	1, 2, 7, 3, 10

b)

Line	Assumptions	Formula	Justification	References
1	1	$\neg(A \vee B)$	Asmp. I	
2	2	A	Asmp. I	
3	2	$(A \vee B)$	$\vee$ I	2
4	1, 2	$\bot$	$\neg$ E	1, 3
5	1	$\neg A$	$\neg$ I	2, 4
6	6	B	Asmp. I	
7	6	$(A \vee B)$	$\vee$ I	6
8	1, 6	$\bot$	$\neg$ E	1, 7
9	1	$\neg B$	$\neg$ I	1, 8
10	1	$(\neg A \wedge \neg B)$	$\wedge$ I	5, 9

c)

Line	Assumptions	Formula	Justification	References
1	1	$(\neg A \wedge \neg B)$	Asmp. I	
2	1	$\neg A$	$\wedge$ E	1
3	1	$\neg B$	$\wedge$ E	1
4	4	$(A \vee B)$	Asmp. I	
...	...	...	...	
9	1, 4	A	via slide 23 (DS)	...
10	1, 4	$\bot$	$\neg$ E	2, 9
11	1	$\neg(A \vee B)$	$\neg$ I	4, 10

d)

Line	Assumptions	Formula	Justification	References
1	1	$\neg(A \wedge B)$	Asmp. I	
2	2	$\neg(\neg A \vee \neg B)$	Asmp. I	
...	...	...	...	
3	2	$(\neg\neg A \wedge \neg\neg B)$	via part (b)	2
4	2	$\neg\neg A$	$\wedge$ E	3
...	...	...	...	
5	2	A	via DN (slide 20)	4
6	2	$\neg\neg B$	$\wedge$ E	3
...	...	...	...	
7	2	B	via DN (slide 20)	6
8	2	$(A \wedge B)$	$\wedge$ I	5, 7
9	1, 2	$\bot$	$\neg$ E	1, 8
10	1	$(\neg A \vee \neg B)$	RA	2, 9

Problem 13. Now we will see why contraposition arguments work in this system, i.e. why we can make arguments like "Whenever it is raining I carry my umbrella, therefore if I am not carring my umbrella then it must not be raining".

To keep our these proofs short, you can use two of the consequences proven in the lecture slides, rather than repeating their proofs here.

- Modus Tollens (MT): $(F \to G), \neg G \vdash \neg F$

- Double Negation (DN): $\neg\neg F \vdash F$ and $F \vdash \neg\neg F$

Prove the following:

a) $(A \to B) \vdash (\neg B \to \neg A)$

b) $(\neg B \to \neg A) \vdash (A \to B)$

Solution 13.

a)

Line	Assumptions	Formula	Justification	References
1	1	$(A \to B)$	Asmp. I	
2	2	$\neg B$	Asmp. I	
3	1,2	$\neg A$	Modus Tollens	1, 2
4	1	$(\neg B \to \neg A)$	$\to$ I	2, 3

b)

Line	Assumptions	Formula	Justification	References
1	1	$(\neg B \to \neg A)$	Asmp. I	
2	2	A	Asmp. I	
3	2	$\neg\neg A$	Double Negation	2
4	1,2	$\neg\neg B$	Modus Tollens	1, 3
5	1,2	B	Double Negation	4
6	1	$(A \to B)$	$\to$ I	2, 5

Problem 1. Give an example of a ternary predicate, i.e., one that takes three arguments.

Solution 1. E.g., the ternary predicate of three arguments x, y, z defined by $z = x + y$.

Problem 2. Write the following as predicate logic formulas:

1. P is a reflexive binary relation.
2. P is a transitive binary relation.
3. f is an injective function.
4. f is a surjective function.

Solution 2.

1. $\forall x P(x, x)$
2. $\forall x \forall y \forall z((P(x, y) \land P(y, z)) \to P(x, z))$
3. $\forall x \forall y(\neg E(x, y) \to \neg E(f(x), f(y)))$
4. $\forall y \exists x E(f(x), y)$

Problem 3. Let $\texttt{child}(x, y)$ be a binary relation expressing that x is a child of y. Write predicate logic formulas expressing the following:

1. the sibling relation,
2. the cousin relation,
3. the second-cousin relation (i.e., their parents are cousins),
4. no pair of siblings has any children,
5. no pair of first cousins has any children,
6. there are two second cousins (i.e., their grandparents are siblings) each of which have children.

Solution 3.

1. the sibling relation $\texttt{siblings}(x_1, x_2)$ can be expressed by
$$\exists y(\texttt{child}(x_1, y) \land \texttt{child}(x_2, y))$$
2. the cousin relation $\texttt{cousins}(x_1, x_2)$ can be expressed by
$$\exists y_1 \exists y_2(\texttt{siblings}(y_1, y_2) \land \texttt{child}(x_1, y_1) \land \texttt{child}(x_2, y_2))$$
3. the second-cousin relation can be expressed by
$$\exists z_1 \exists z_2(\texttt{cousins}(z_1, z_2) \land \texttt{child}(x_1, z_1) \land \texttt{child}(x_2, z_2))$$

Problem 4. Recall the structure of integers $\mathcal{Z}$ and of strings $\mathcal{S}$ from lectures.

1. How is $\texttt{add}^{\mathcal{Z}}$ different from $\texttt{add}^{\mathcal{S}}$?
2. Find a sentence which is true in the structure of integers $\mathcal{Z}$ and of strings $\mathcal{S}$.
3. Consider the structure $\mathcal{T} = (\mathbb{H}, \texttt{parent\_of}^{\mathcal{T}}, \texttt{man}^{\mathcal{T}}, \texttt{woman}^{\mathcal{T}})$, where $\texttt{parent\_of}^{\mathcal{T}}$ is binary relation, and man is a unary predicate, and $\texttt{woman}^{\mathcal{T}}$ is a unary predicates. The formula $(\texttt{parent\_of(x,y)} \wedge \texttt{man(x)})$ means that x is the father of y. Write formulas that express that x is the sister of y, that x is the uncle of y, and other family relations.
4. Give a structure with 3 elements in the domain and one binary predicate.
5. Consider the formula $\exists x \forall y\, \texttt{eq}(\texttt{add}(y, x), y)$ that says that there is an element x such that when it is added to any element y we just get y. Is this formula true about strings? Is this formula true about integers?

Solution 4.

1. $\texttt{add}^{\mathcal{Z}}$ is a function on the domain of integers, while $\texttt{add}^{\mathcal{S}}$ is a function on the domain of strings.
2. $\exists x \forall y\, \texttt{eq}(\texttt{add}(x, y), \texttt{add}(y, x))$.

Problem 5. What are the subformulas of the following predicate logic formulas?

1. $\forall x \forall y (P(x, y) \wedge \neg P(y, x))$
2. $\forall x (P(x, f(x)) \wedge \exists y \neg P(y, x))$
3. $\forall x \forall y (P(x, y) \leftrightarrow \neg R(x, y))$

Solution 5.

1. The subformulas are $\forall x \forall y (P(x, y) \wedge \neg P(y, x))$, $\forall y (P(x, y) \wedge \neg P(y, x))$, $(P(x, y) \wedge \neg P(y, x))$, $P(x, y)$, $\neg P(y, x))$, and $P(y, x)$.

Problem 6. If F is a formula and F occurs as part of the formula G then F is called a *subformula* of G. Give a recursive procedure for determinining if F is a subformula of G.

Problem 7. Let $\text{Free}(F)$ be the set of all variables that occur free in F. Define $\text{Free}(F)$ by a recursive procedure.

Solution 7. The set $\text{Free}(F)$ of free variables of F is defined by the following recursive procedure:

1. for an atomic formula F, $\text{Free}(F)$ is the set of variables occuring in F.
2. $\text{Free}(\neg F) = \text{Free}(F)$
3. $\text{Free}((F \vee G)) = \text{Free}((F \wedge G)) = \text{Free}(F) \cup \text{Free}(G)$

4. $\text{Free}(\exists x F) = \text{Free}(\forall x F) = \text{Free}(F) \setminus \{x\}$.

Problem 8. Consider the formula $\forall x\, E(f(x))$ where E is a unary predicate-symbol and f is a unary function-symbol.

1. Provide a structure $\mathcal{A}$ in which the truth-value of this formula is 1.
2. Provide a structure $\mathcal{B}$ in which the truth-value of this formula is 0.
3. Provide a structure $\mathcal{C}$ in which the truth-value of the atomic formula $E(f(x))$ depends on the given assignment.

Problem 9. Consider the structure $\mathcal{R} = (\mathbb{R}, P^{\mathcal{R}}, N^{\mathcal{R}}, S^{\mathcal{R}})$ where

- $P^{\mathcal{R}}(x)$ expresses that x is positive,
- $N^{\mathcal{R}}(x)$ expresses that x is a natural number,
- $S^{\mathcal{R}}(x, y)$ expresses that $x = y^2$.

Express the following formulas in English, and decide if they are true or false:

1. $\exists x \exists y \big(P(x) \wedge S(x, y)\big)$
2. $\forall x \neg P(x)$
3. $\exists x \neg P(x)$
4. $\neg \forall x P(x)$
5. $\forall x \exists y \Big(\big(N(x) \wedge S(x, y)\big) \rightarrow P(x)\Big)$
6. $\big(\forall x P(x) \vee N(x)\big)$
7. $\exists x S(x, y)$

Do the same for the structure $\mathcal{Z} = (\mathbb{Z}, P^{\mathcal{Z}}, N^{\mathcal{Z}}, S^{\mathcal{Z}})$ where

- $P^{\mathcal{Z}}(x)$ expresses that x is positive,
- $N^{\mathcal{Z}}(x)$ expresses that x is a natural number,
- $S^{\mathcal{Z}}(x, y)$ expresses that $x = y^2$.

Solution 9. Here is the solution for $\mathcal{R}$:

1. There is a positive real number that is the square of another real number
2. All real numbers are not positive
3. There is a real number that is not positive
4. Not all real numbers are positive (this is equivalent to the previous formula)

5. On first glance you might read this as meaning "If a natural number is the square of a number, then it is positive", but actually this interpretation isn't correct! For example, let $y = \pi$, then $(N(x) \wedge S(x, y))$ will be false for any value of x, which makes the formula trivially true.

   A more appropriate formula for this statement would be:

   $$\forall x \Big(\big(N(x) \wedge \exists y S(x, y)\big) \rightarrow P(x) \Big)$$

   Or, equivalently: $\forall x \forall y \Big(\big(N(x) \wedge S(x, y)\big) \rightarrow P(x) \Big)$

6. All numbers are positive, or x is a a natural number (note: the second x in this formula is a *free variable*, so it refers to a specific number, unlike the first x, which is bound to the the $\forall$ quantifier.)

7. y is the square of a number

Problem 10. Suppose the domain D is the universe. Using the following predicates:

- $C(x)$: x is a child
- $B(x)$: x is a book
- $L(x, y)$: x likes y
- $E(x)$: x is educational

Express the following sentences in predicate logic:

1. All children like books
2. Some child likes every single book
3. Not all children like all books
4. Some child does not like any book
5. There is a book that all children like
6. Books are always educational
7. Educational books are liked by children
8. Books are not always educational
9. No book, except for the educational ones, is liked by every child.
10. There is a child who likes all books that are not educational

Solution 10.

1. All children like books

   $$\forall x (C(x) \rightarrow \exists y (B(y) \wedge L(x, y)))$$

2. Some child likes every single book

$$\exists x\Big(C(x) \land \forall y\big(B(y) \to L(x,y)\big)\Big)$$

3. Not all children like all books

$$\neg\forall x\forall y\Big(\big(C(x) \land B(y)\big) \to L(x,y)\Big)$$

4. Some child does not like any book

$$\exists x\forall y\Big(C(x) \land \big(B(y) \to \neg L(x,y)\big)\Big)$$

5. There is a book that all children like

$$\exists y\forall x\Big(B(y) \land \big(C(x) \to L(x,y)\big)\Big)$$

6. Books are always educational

$$\forall x(B(x) \to E(x))$$

7. (All) Educational books are liked by (all) children

$$\forall x\forall y\Big(\big((C(x) \land B(y)) \land E(y)\big) \to L(x,y)\Big)$$

8. Books are not always educational

$$\neg\forall x(B(x) \to E(x))$$

9. No book, except for the educational ones, is liked by every child.

$$\forall y\exists x\Big(\big(B(y) \land \neg E(y)\big) \to \big(C(x) \land \neg L(x,y)\big)\Big)$$

10. There is a child who likes all books that are not educational

$$\exists x\forall y\Big(C(x) \land \big((B(y) \land \neg E(y)) \to L(x,y)\big)\Big)$$

Problem 11. For each of the following expresions, indicate if it is a formula of predicate logic. If it is, then indicate the free occurrences of variables, the bound occurrences of variables, the constant symbols, the function symbols, and the predicate symbols.

1. $(\forall x P(x) \land P(c))$
2. $\forall x(P(x) \land P(y))$
3. $\forall x(P(x) \land Q(y,x))$
4. $\forall x(P(x,a) \land (\exists y))$
5. $(\forall x\exists y P(y,x) \land P(x,y))$

6. $(\forall x P(y,x) \wedge \exists y P(x,y))$

Solution 11.
P and Q are the predicate symbols. There are no function symbols.

1. Correct syntax, x is bound, c is a constant
2. Correct syntax, x is bound, y is free
3. Correct syntax, the x's are bound, y is free
4. Incorrect syntax, the $\exists y$ quantifier does not introduce a formula
5. Correct syntax, the first x and y are bound, but the second ones are free.
6. Correct syntax, the first x is bound but the second is free, the first y is free but the second is bound.

Problem 12. Consider the sentence:

$$\forall x \forall y \Big(\big(P(x,y) \wedge (P(x,c) \vee Q(x,c))\big) \rightarrow \big(P(c,f(x,y)) \vee Q(c,f(x,y))\big) \Big)$$

and the structure $\mathcal{A}$ with domain $\mathbb{Z}$, and

- $c^{\mathcal{A}} = 0$,
- $f^{\mathcal{A}}(x,y) = x + y$,
- $P^{\mathcal{A}}$ is the set of pairs (a,b) with $(a < b)$
- $Q^{\mathcal{A}}$ is the set of pairs (a,b) with $(a = b)$

What is the truth-value of the setence in $\mathcal{A}$?

Solution 12. It evaluates as False, e.g., $x = -100, y = 1$

Problem 1. The following are not equivalence laws of predicate logic. Show this by providing counterexamples.

1. $(\forall x F \vee \forall x G) \not\equiv \forall x(F \vee G)$
2. $(\exists x F \wedge \exists x G) \not\equiv \exists x(F \wedge G)$
3. $\forall x \exists y F \not\equiv \exists y \forall x F$

Problem 2. For each of the formulas below, indicate the free and bound occurrences of the variables, and put the formulas into prenex normal form, justifying each step.

1. $(\forall x Q(x) \to \exists z \forall y R(z, y))$
2. $(\forall x Q(x) \to \forall y \exists z R(z, y))$
3. $\forall x\big(R(x, y) \wedge \neg \forall y R(x, y)\big)$

Solution 2.

1. All variables are bound. No renaming necessary

$$\begin{array}{ll} (\forall x Q(x) \to \exists z \forall y R(z, y)) & \\ (\neg \forall x Q(x) \vee \exists z \forall y R(z, y)) & \text{Def Imp} \\ (\exists x \neg Q(x) \vee \exists z \forall y R(z, y)) & \text{Q Neg} \\ \exists x(\neg Q(x) \vee \exists z \forall y R(z, y)) & \text{Q Extr} \\ \exists x \exists z(\neg Q(x) \vee \forall y R(z, y)) & \text{Q Extr} \\ \exists x \exists z \forall y(\neg Q(x) \vee R(z, y)) & \text{Q Extr} \end{array}$$

   Note: $\exists z \exists x \forall y, \exists z \forall y \exists x$ are also correct orders, but we could not extract y before z.

2. Similar to before, but note the different order of y and z.

$$\exists x \forall y \exists z(\neg Q(x) \vee R(z, y))$$

   Similarly these orderings are also correct: $\forall y \exists x \exists z, \forall y \exists z \exists x$

3. All occurrences of x are bound, the first y is free, but the second is bound.

$$\begin{array}{ll} \forall x\big(R(x, y) \wedge \neg \forall y R(x, y)\big) & \\ \forall x\big(R(x, y) \wedge \neg \forall z R(x, z)\big) & \text{Relabelling} \\ \forall x\big(R(x, y) \wedge \exists z \neg R(x, z)\big) & \text{Q Neg} \\ \forall x \exists z\big(R(x, y) \wedge \neg R(x, z)\big) & \text{Q Extr} \end{array}$$

   Note that we had to relabel the *bound* y. We *cannot* relabel the free y.

Problem 3.

1. For formula $F = (\forall x(P(x,z) \wedge \exists y Q(y)) \rightarrow P(y,z))$ and term $t = f(x,y)$, write $F[t/y]$ and say whether t is free to replace y in F.
2. For formula $F = (\forall x(P(x,z) \wedge \exists y Q(y)) \rightarrow P(y,z))$ and term $t = f(x,y)$, write $F[t/z]$ and say whether t is free to replace z in F.
3. For formula $F = (\forall x(P(x,z) \wedge \exists y Q(y)) \rightarrow P(y,z))$ and term $t = f(x,y)$, write $F[t/x]$ and say whether t is free to replace x in F.

Solution 3.

1. Yes.
2. No, because x occurs in t and a free occurrence of z is in the scope of the $\forall x$ quantifier.
3. ??

Problem 4. The following proof segment contains MISTAKES. Say what is wrong and why.

Asmp. I	Line	Formula	Justification	References
1	1	$\exists x P(y,x)$	Asmp. I	
1	2	$P(y,c)$	$\exists$-elim	1
1	3	$\forall y P(y,c)$	$\forall$-intro	2

Solution 4. Line 2 is an incorrect application of ($\exists$E) because to use ($\exists$E) correctly one needs two premises: a formula of the form $\exists x F$ and a formula of the form G proved from an assumption of the form $F[x/c]$.
Line 3 is an incorrect application of ($\forall$I) because to use ($\forall$I) correctly one needs to replace a constant symbol by a variable, e.g., one can pass from $P(y,c)$ to $\forall x P(y,x)$ as long as c does not occur in the assumption set for $P(y,c)$.

Problem 5. A sentence over vocabulary σ is *valid* if its is true in every structure over vocabulary σ. Show that the following sentence is not valid:

$$\Big(\exists x P(x) \wedge \big(\exists x((P(x) \rightarrow Q(x)) \rightarrow \exists x Q(x))\big)\Big)$$

Hint: Consider any structure $\mathcal{A}$ where $P^{\mathcal{A}}$ is empty. Then the sentence will be false in that structure.

Problem 6. Consider the following argument: Every dog likes people or hates cats. Rover is a dog. Rover likes cats. Therefore, some dog likes people.

1. Express the English sentences in predicate logic over a suitably chosen vocabulary.
2. The argument itself can be expressed as the consequence $E_1, E_2, E_3 \vdash F$. Prove this consequence in Natural Deduction.

Solution 6.

1. Take the vocabulary with unary predicates D (for the set of dogs), C (for the set of dogs that like cats), P (for the set of dogs that like people), and constant symbol *rover*.

Problem 7. Prove the following consequences in Natural Deduction:

1. $\forall x P(x) \vdash \exists x P(x)$
2. $\exists y \forall x P(x,y) \vdash \forall x \exists y P(x,y)$
3. $\forall x(P(x) \to \neg Q(x)),\ \forall x(R(x) \to P(x)) \vdash \forall x(Q(x) \to \neg R(x))$
4. $\forall x P(x), \forall x(P(x) \to Q(x)) \vdash \forall x Q(x)$
5. $\vdash (\exists x(P \wedge Q) \to (\exists x P \wedge \exists x Q))$
6. $\vdash \Big(\big(\forall x \neg P(x,x) \wedge \forall x \forall y \forall z((P(x,y) \wedge P(y,z)) \to P(x,z))\big) \to \forall x \forall y \neg \big(P(x,y) \wedge P(y,x)\big)\Big)$
7. $\vdash \Big(\big(\forall x(P(x) \to Q(x)) \wedge \exists x P(x)\big) \to \exists x Q(x)\Big)$
8. $\vdash \big(\exists y \forall x P(x,y) \to \forall x \exists y P(x,y)\big)$

Solution 7.

1.

Line	Asmp.	Formula	Justification	References
1	1	$\forall x P(x)$	Asmp. I	
2	1	$P(c)$	$\forall$-elim	1
3	1	$\exists x P(x)$	$\exists$-intro	2

5.

Line	Asmps	Formula	Justification	Refs
1	1	$\exists x(P \wedge Q)$	Asmp. I	
2	2	$(P \wedge Q)[x/c]$	Asmp. I	
3	2	$P[x/c]$	$\wedge$E	2
4	2	$Q[x/c]$	$\wedge$E	2
5	2	$\exists x P$	$\exists$I	3
6	2	$\exists x Q$	$\exists$I	4
7	2	$(\exists x P \wedge \exists x Q)$	$\wedge$I	5,6
8	1	$(\exists x P \wedge \exists x Q)$	$\exists$E	1,2,7
9		$(\exists x(P \wedge Q) \to (\exists x P \wedge \exists x Q))$	$\to$I	1,8

Problem 1. Let $\Sigma = \{0, 1\}$. Write regular expressions over Σ for the following languages.

1. The set of strings containing at least one 0 and at least one 1.
2. The set of strings whose third symbol from the right is 1.
3. The set of strings whose length is a multiple of three.
4. The set of strings whose number of 0's is a multiple of three.
5. The set of strings with at most one pair of consecutive 1's.
6. The set of strings not containing 101 as a substring.

Solution 1.

1. $((0 \cup 1)^*0(0 \cup 1)^*1(0 \cup 1)^* \cup (0 \cup 1)^*1(0 \cup 1)^*0(0 \cup 1)^*)$.
2. $(0 \cup 1)^*1(0 \cup 1)(0 \cup 1)$.

Problem 2. Which of the languages described by the following regular expressions over the alphabet $\{a, b, c\}$ is infinite?

1. $(a \cdot (b \cdot c^*))$
2. $((a \cup b) \cdot c)$
3. $(a \cup b)^*$
4. $\emptyset$
5. $\emptyset^*$
6. ϵ^*

Solution 2. Only 1) and 3) are infinite.

Problem 3. Argue that every finite language $\{s_1, s_2, \cdots, s_n\}$ is the language of some regular expression.

Solution 3. For every string $s = a_1a_2 \cdots a_k \in \Sigma^*$, we have $L(a_1 \cdot a_2 \cdot a_3 \cdots a_k) = \{s\}$. Since we don't need to write the concatenation symbol $\cdot$, we have that $L(s) = \{s\}$, for every string s. The language $\{s_1, s_2, \cdots, s_n\}$ is the language of the regular expression

$$s_1 \cup s_2 \cup \cdots \cup s_n$$

Problem 4. If R is a regular expression, and $k \geq 0$ is an integer, we introduce the following shorthand: R^k is a regular expression whose language is $\overbrace{L(R) \cdot L(R) \cdots L(R)}^{k}$. Give a recursive definition of $L(R^k)$. By default, we have that $L(R^0) = \{\epsilon\}$. Use this shorthand to give a regular expression over $\{0, 1\}$ whose tenth last digit is a 1.

Solution 4. Define $L(R^0) = \{\epsilon\}$, and for $k \geq 1$, define $L(R^k) = (L(R^{k-1}) \cdot L(R))$. For instance, $(0 \cup 1)^* 1 (0 \cup 1)^9$ is a regular expression denoting the language of strings whose tenth last digit is a 1.

Problem 5. Discussion. Look at the definition of a deterministic finite automaton (DFA).

1. What does "deterministic" refer to?
2. What does "finite" refer to?
3. What does "automaton" refer to?

Problem 6. If $R_1, R_2, \cdots R_n$ are regular expressions then we write $(\cup_{i=1}^n R_i)$ and $(R_1 \cup R_2 \cdots \cup R_n)$ as shorthand for $(\ldots(R_1 \cup (R_2 \cup R_3)) \cdots \cup R_n)$. For instance, $(0 \cup 1 \cup 00)$ is shorthand for $((0 \cup 1) \cup 00)$. Why are we justified in doing this?

Solution 6. Because $\cup$ is an associate operation on languages. That is, $(R \cup (S \cup T)) \equiv ((R \cup S) \cup T))$, and so writing $(R \cup S \cup T)$ does not leave any ambiguity as to the language being represented.

Problem 7. If $R_1, R_2, \cdots R_n$ are regular expressions then we write $(R_1 \cdot R_2 \ldots \cdot R_n)$ as shorthand for $(\ldots(R_1 \cdot (R_2 \cdot R_3)) \ldots \cdot R_n)$. For instance, $(0 \cup 1)^* 001$ is shorthand for $(0 \cup 1)^* \cdot 0 \cdot 0 \cdot 1$. Why are we justified in doing this?

Problem 8. Two expressions R, S are called *equivalent* if $L(R) = L(S)$. Here are some equivalence laws for regular expressions. For all regular expressions R, S, T:

$$(R \cup (S \cup T)) \equiv ((R \cup S) \cup T \tag{1}$$
$$(R \cup S) \equiv (S \cup R) \tag{2}$$
$$(R(ST)) \equiv ((RS)T) \tag{3}$$
$$(R \cup \emptyset) \equiv R \tag{4}$$
$$(R \cup R) \equiv R \tag{5}$$
$$(\epsilon \cdot R) \equiv (R \cdot \epsilon) \equiv R \tag{6}$$
$$(R(S \cup T)) \equiv ((RS) \cup (RT)) \tag{7}$$
$$((R \cup S)T \equiv ((RT) \cup (ST)) \tag{8}$$
$$(\emptyset \cdot R) \equiv (R \cdot \emptyset) \equiv \emptyset \tag{9}$$
$$(\epsilon \cup (R \cdot R^*)) \equiv R^* \tag{10}$$
$$(\epsilon \cup (R^* \cdot R)) \equiv R^* \tag{11}$$

Use these equivalences to show that $(1 \cup 01 \cup 001)^*(\epsilon \cup 0 \cup 00)$ is equivalent to $((\epsilon \cup 0)(\epsilon \cup 0)1)^*(\epsilon \cup 0)(\epsilon \cup 0)$.

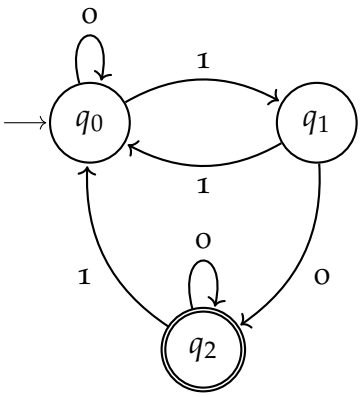
Solution 8.

$$\begin{aligned}(1 \cup 01 \cup 001)^*(\epsilon \cup 0 \cup 00) &= ((\epsilon 1 \cup 01) \cup 001)^*((\epsilon \cup 0) \cup 00)) && (6)\\ &\equiv ((\epsilon \cup 0)1 \cup 001)^*((\epsilon \cup 0) \cup 00) && (8)\\ &\equiv (((\epsilon \cup 0) \cup 00)1)^*((\epsilon \cup 0) \cup 00) && (8)\end{aligned}$$

Now we are left to show that $((\epsilon \cup 0) \cup 00) \equiv ((\epsilon \cup 0)(\epsilon \cup 0))$. We start with the Right-Hand-Side (RHS). We drop outermost brackets (for readability):

$$\begin{aligned}
(\epsilon \cup 0)(\epsilon \cup 0) &\equiv (\epsilon \cup 0)\epsilon \cup (\epsilon \cup 0)0 && (7)\\
&\equiv (\epsilon\epsilon) \cup (0\epsilon) \cup (\epsilon 0) \cup (00) && (8)\\
&\equiv \epsilon \cup 0 \cup 0 \cup 00 && (6)\\
&\equiv \epsilon \cup 0 \cup 00 && (4)
\end{aligned}$$

Problem 9. Consider the automaton drawn below:



1. Give $(Q, \Sigma, \delta, q_0, F)$ for this automaton.
2. What is language accepted by this automaton?

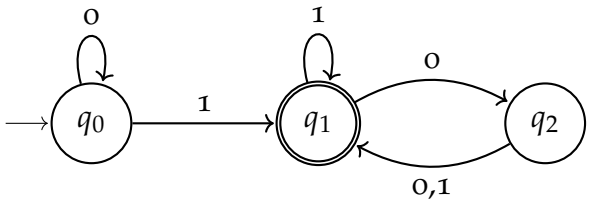
Solution 9.

1. $Q = \{q_0, q_1, q_2\}$
   $\Sigma = \{0, 1\}$
   $q_0 = q_0$
   $\delta : Q \times \Sigma \to Q$ is given by:

	0	1
q_0	q_0	q_1
q_1	q_2	q_0
q_2	q_2	q_0

   $F = \{q_2\}$
2. Strings of 1s and 0s which end in 0 and which contain an odd number of 1s

Problem 10.
Consider the automaton drawn below:



1. Give $(Q, \Sigma, \delta, q_0, F)$ for this automaton.
2. What is language accepted by this automaton?

Solution 10.

1. $Q = \{q_0, q_1, q_2\}$
   $\Sigma = \{0, 1\}$
   $q_0 = q_0$
   $\delta : Q \times \Sigma \to Q$ is given by:

	0	1
q_0	q_0	q_1
q_1	q_2	q_1
q_2	q_1	q_1

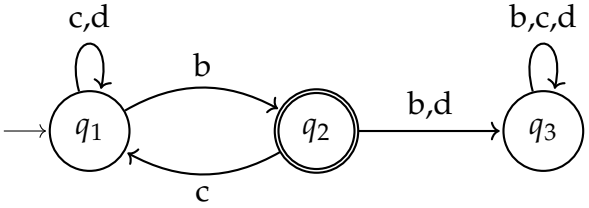
   $F = \{q_1\}$
2. Strings of 1s and 0s which contain at least one 1 and which do not end with an odd number of 0s

Problem 11.
Draw the following automaton and give the language that it accepts:
$Q = \{q_1, q_2, q_3\}$ is the set of states
$\Sigma = \{b, c, d\}$ is the alphabet
$\delta : Q \times \Sigma \to Q$ is given by:

	b	c	d
q_1	q_2	q_1	q_1
q_2	q_3	q_1	q_3
q_3	q_3	q_3	q_3

q_1 is the start state
$F = \{q_2\}$ is the set of accept states

Solution 11.



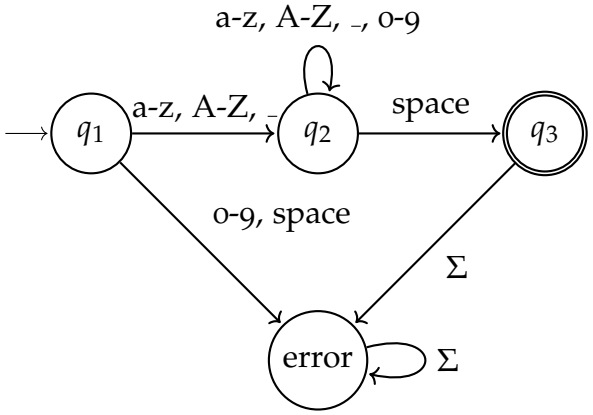
Accepts strings which end in b and in which b is never followed by b or d.

Problem 12.

1. Construct a DFA that accepts valid identifiers. A valid identifier begins with a letter or with an underscore '_' and contains any combination of letters, underscores, or digits, followed by a blank space.
2. How would you change your automaton if the length of the identifier (excluding the blank) should not be longer than 6 symbols?

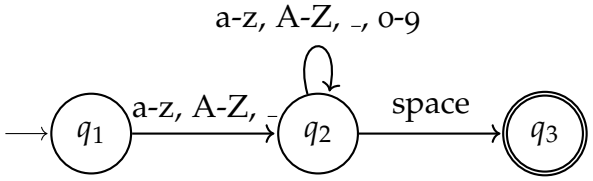
Solution 12.

1. If we don't need to count the length, it's fairly simple:



   Often we're lazy, and omit to draw the 'error states', states from which no path leads to an accept state. If we do this we must write 'Error state not shown' on our diagram:

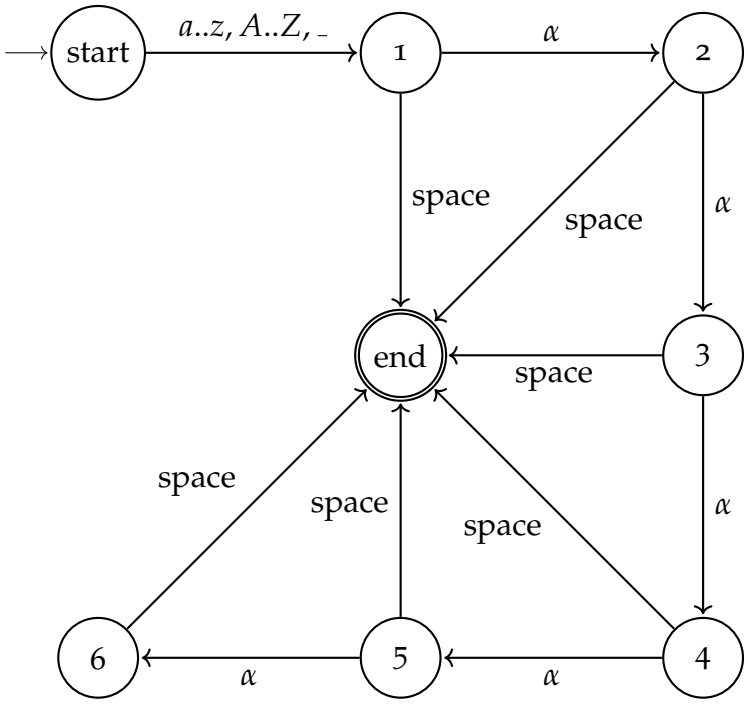
   (Error state not shown)



2. To be able to count the length of the identifier (up to to six characters) we will need to add more states, which can be used to count how many symbols we have read:

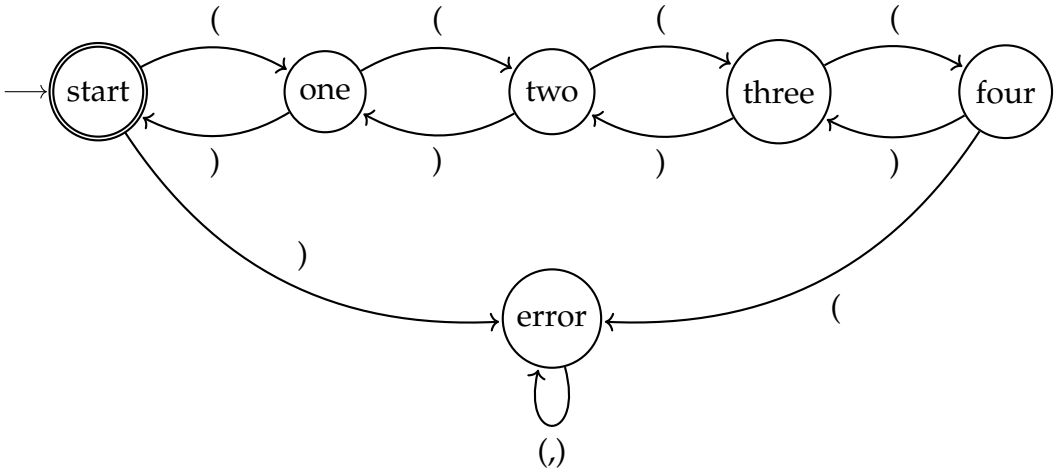
   Let $\Sigma = \{a..z, A..Z, 0..9, \_, space\}$ and $\alpha = \Sigma \setminus \{space\}$

   (Error state not shown)



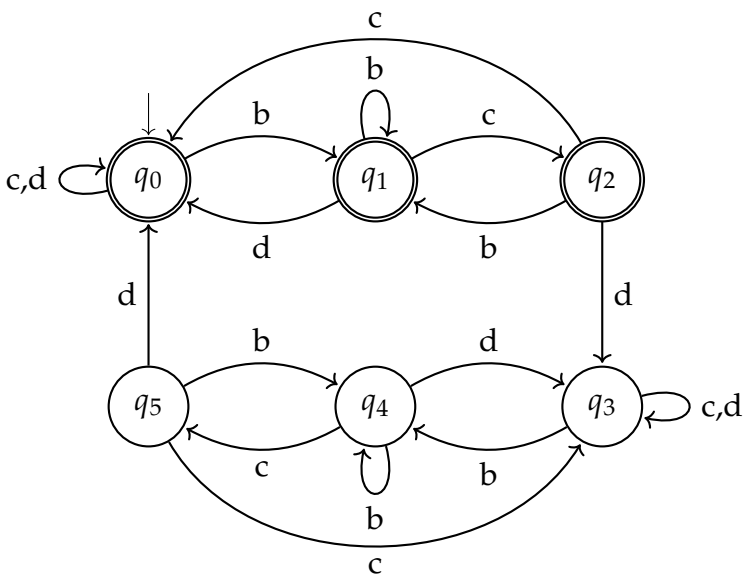
Problem 13. Construct a DFA which accepts strings of valid bracket expressions with at most 4 levels of nesting. For example, it should accept ((((())))) or ()(()()) but not (() or)(.

Solution 13.



Problem 14. Draw an automaton which accepts strings over $\{b, c, d\}$ containing an even number of occurrences of the substring bcd. Recall that 0 is an even number.

Hint: think about what the automaton needs to remember as it moves through the input. Add a state for each of these conditions, then add appropriate transitions between them.



Solution 14.

In q_0, q_1, q_2 we've read an even number of occurrneces of bcd. In the other states we've seen an odd number. In q_1 or q_4 the last symbol read was a b. In q_2 or q_5 the last two symbols read were bc.

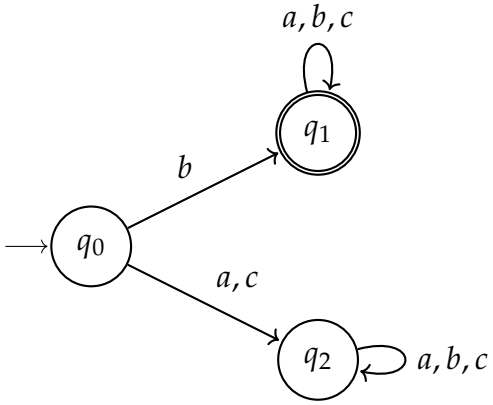
Problem 15. Prove that the following languages are regular, by drawing finite automata which recognise them. The alphabet for all the languages is $\{a, b, c\}$.

1. Start with b

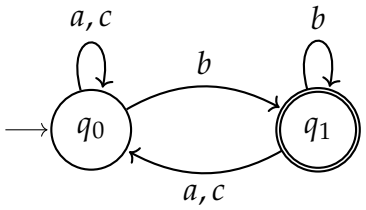
2. End with b
3. Contain exactly one b
4. Contain at least one b
5. Contain at least one b and are of length exactly 2
6. Are of length at most 2
7. Are of length at most n for some integer n

Solution 15.

1. Start with b

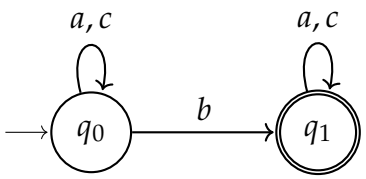


2. End with b

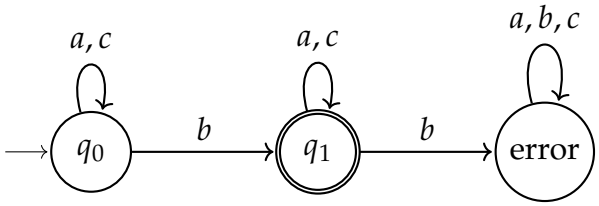


3. Contain exactly one b

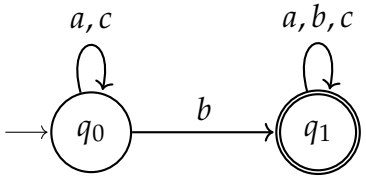
(Error states not shown)



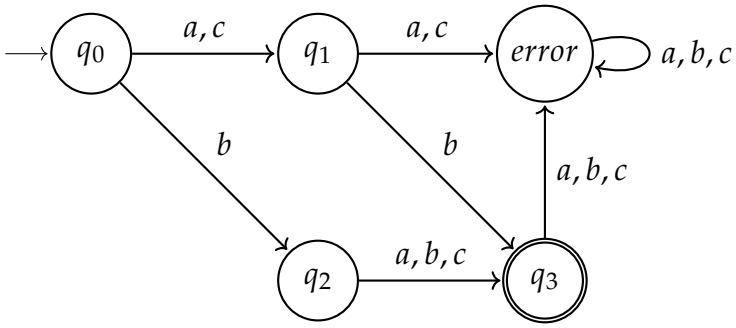
Or, with an error state:



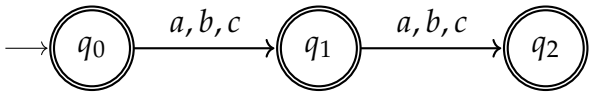
4. Contain at least one b



5. Contain at least one b and are of length exactly 2



6. Are of length at most 2
   (error states not shown)

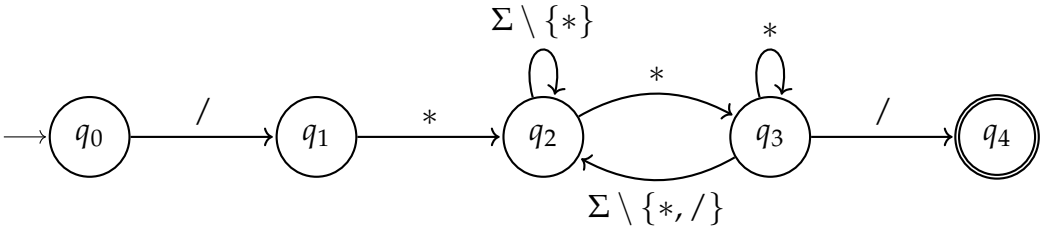


7. Are of length at most n for some integer n
   As above, but with the set of $n \cup 2$ states $Q = \{q_0, ..., q_n, error\}$

Problem 16. Draw an automaton for comments in C or C++. These are strings which

1. begin with /*
2. end in */ (where the * cannot be the same as the first one)
3. have no other occurrences of */

Solution 16. (Error states not shown)

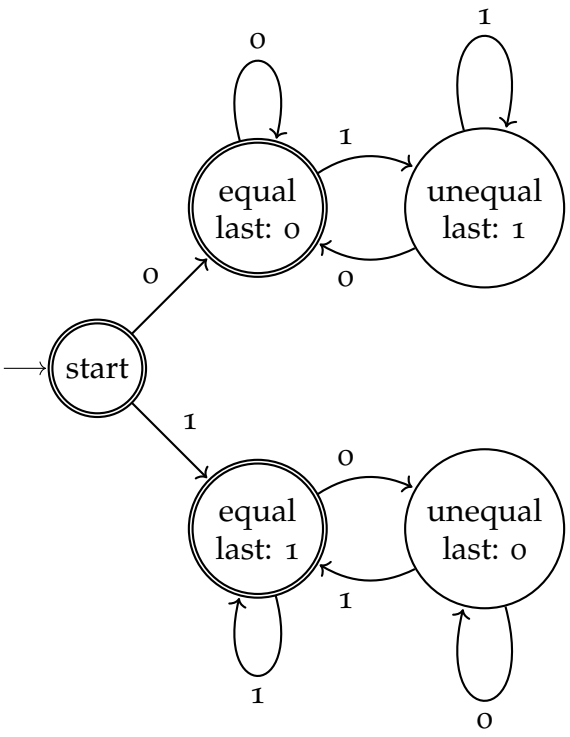


Problem 17. Try construct a DFA M such that

$L(M) = \{w | w$ contains an equal number of occurrences of the substrings 01 and $10\}$

For example, $101 \in L(M)$, but $1010 \notin L(M)$. If you cannot do so, what is the difficulty?

Solution 17.
Observe that in any string of 1s and 0s, it's impossible for the number of occurrences of 01 and of 10 to differ by more than one at any point of scanning through the string. So, we just need our states to represent all the combinations of {equal count, unequal count} and {last seen 0, last seen 1, seen nothing}



Problem 18. Try to devise a finite automaton M' such that

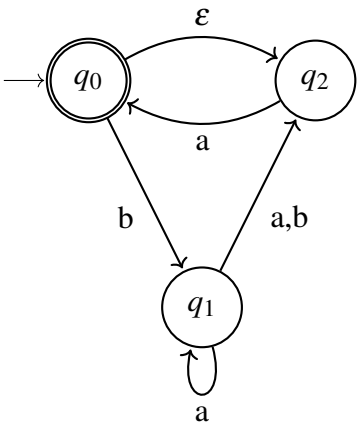
$$L(M') = \{w | w \text{ contains an equal number of occurrences of 0 and 1}\}$$

For example, $1010 \in L(M)$, but $101 \notin L(M)$. If you cannot do so, what is the difficulty?

Solution 18. The difference in counts could be arbitrarily large at any point of reading through the string. Suppose we used n states to represent a difference of up to n 0s more than 1s in the count. Let s be the string 0^m1^m (i.e. m 0s followed by m 1s), where $m > n$. Then s is in $L(M')$, but our automaton would fail to recognise it. No matter how large we make n, we will always be able to construct a string in the language for which the automaton would need to have been a little bit larger. Therefore, *this particular approach* will not allow us to create a deterministic *finite* automaton which could recognise the language.

In fact, there is *no* deterministic finite automata which will recognise this language (i.e. the language is not regular.) Here is a proof of this fact. The proof is by contradiction. Suppose there were a DFA, say M, that recognised the set of all strings (over the alphabet $\{0, 1\}$ with the same number of 0s as 1s. For every n, the string 0^n labeles a path starting in the initial state and ending in some state, call it q_n. But since there are only finitely many states, there must exist $n \neq m$ such that $q_n = q_m$. But since 0^n1^n is accepted by M, there is a path from q_n labeled 1^n to a final state (this is the only way for 0^n1^n to be accepted by M). Thus there is a path from the initial state to a final state labeled 0^m1^n, i.e., the string 0^m1^n is accepted by M. But 0^m1^n has a different number of 0s as 1s. This contradicts the assumption that M accepts strings with the same number of 0s as 1s'.

Problem 1. Consider the following NFA:



Does the NFA accept:

1. ε
2. a
3. baba
4. baa
5. aaab
6. aabb

Solution 1.

1. ε ACCEPTED (by path q_0)
2. a ACCEPTED (by path q_0, q_2, q_0)
3. baba ACCEPTED (by path q_0, q_1, q_1, q_2, q_0)
4. baa ACCEPTED (by path q_0, q_1, q_2, q_0)
5. aaab REJECTED (ended in the set of states $\{q_1\}$, which does not contain an accept state)
6. aabb REJECTED (ended in the set of states $\{q_2\}$, which does not contain an accept state)

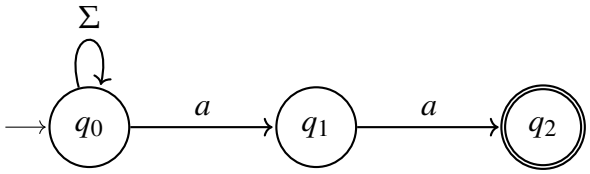
Problem 2. Draw an NFA for each of the following languages. The alphabet is $\Sigma = \{a, b, c\}$. You may use epsilon transitions.

1. Strings that end with *aa* (use only 3 states)
2. Strings that contain the substring 'bbc' (use only 4 states)
3. Strings that start with a or end in c (use only 4 states)
4. Strings that start with a and end in c (use only 3 states)
5. Strings of length 4 starting with b

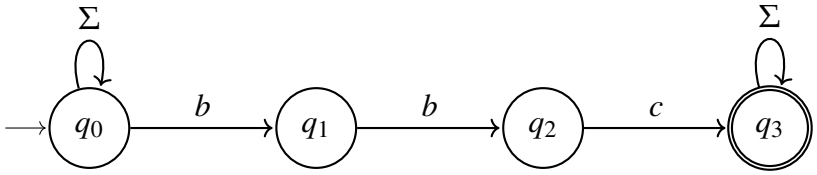
6. Strings of length at most 4.

Solution 2.

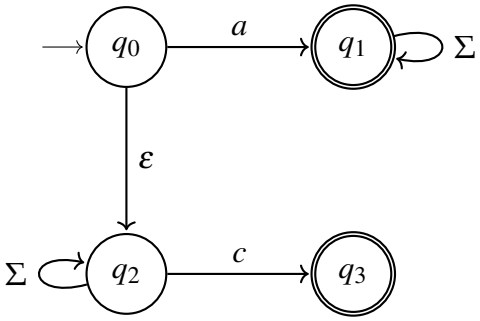
1. Strings that end with *aa* (use only 3 states)



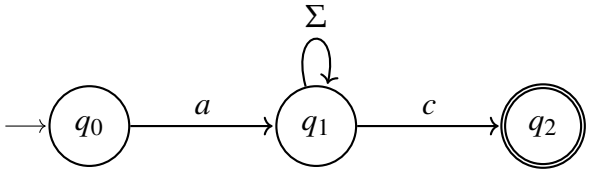
2. Strings that contain the substring 'bbc' (use only 4 states)



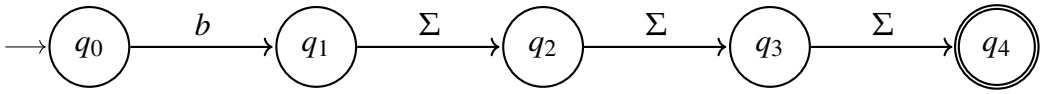
3. Strings that start with a or end in c (use only 4 states)



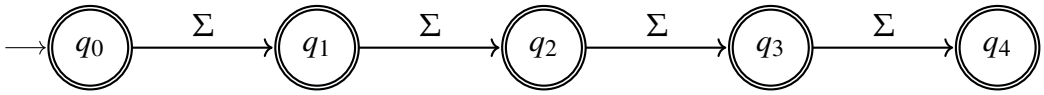
4. Strings that start with a and end in c (use only 3 states)



5. Strings of length 4 starting with b



6. Strings of length at most 4.



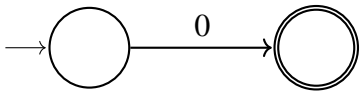
Problem 3. Use the construction demonstrated in lectures to construct an NFA for each of the following RegExs:

1. $0^\star 110^\star$
2. $((00)^\star(11) \cup 10)^\star$

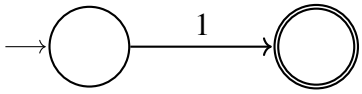
Using the construction demonstrated in lectures, remove ε-transitions from each of the final NFAs.

Solution 3.

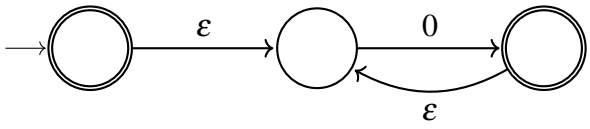
1. NFA for 0:



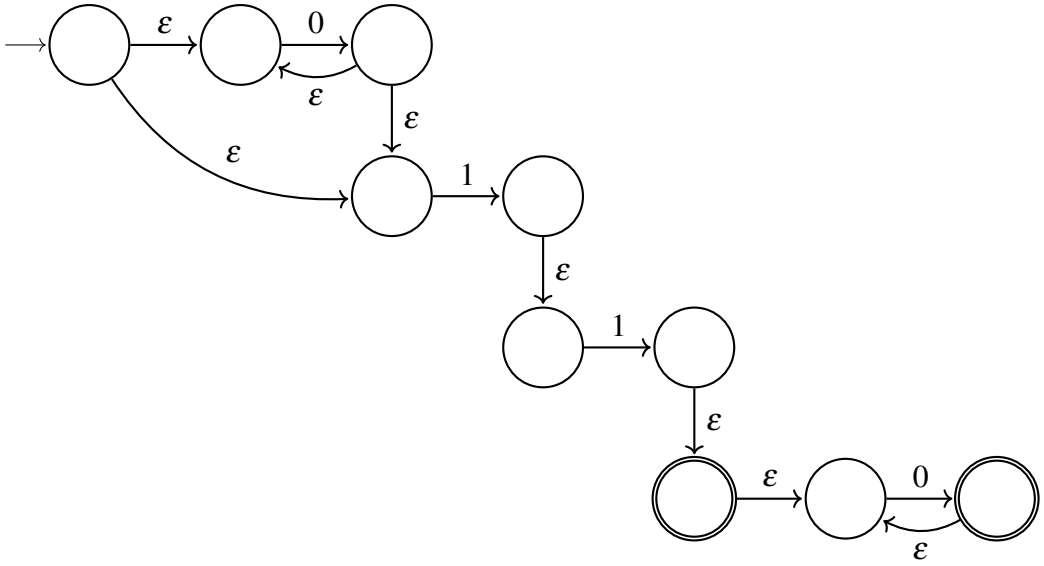
NFA for 1:



NFA for $0^\star$:

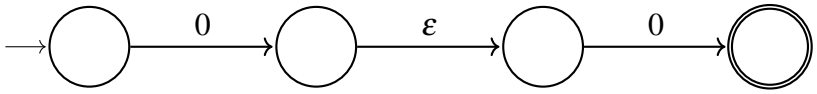


Concatenate the various machines to get the final answer:

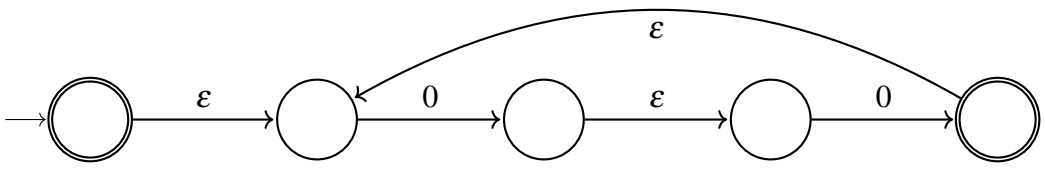


2. $((00)^\star(11) \cup 10)^\star$

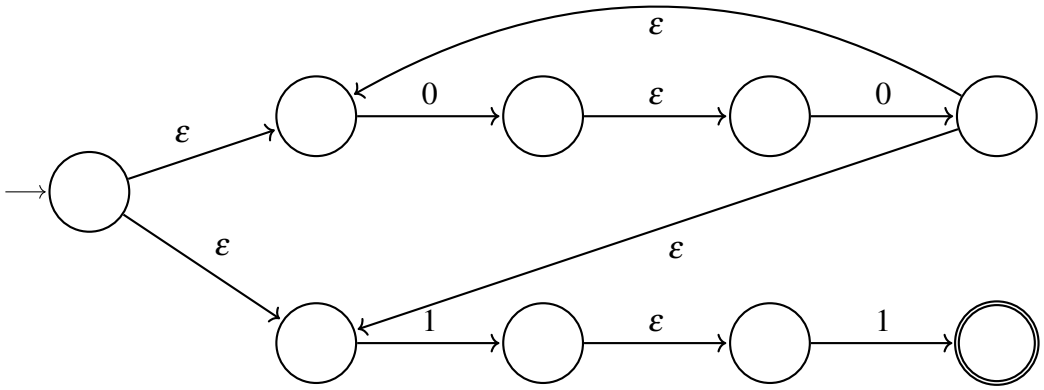
NFA for 00 (similarly for 11, 10):



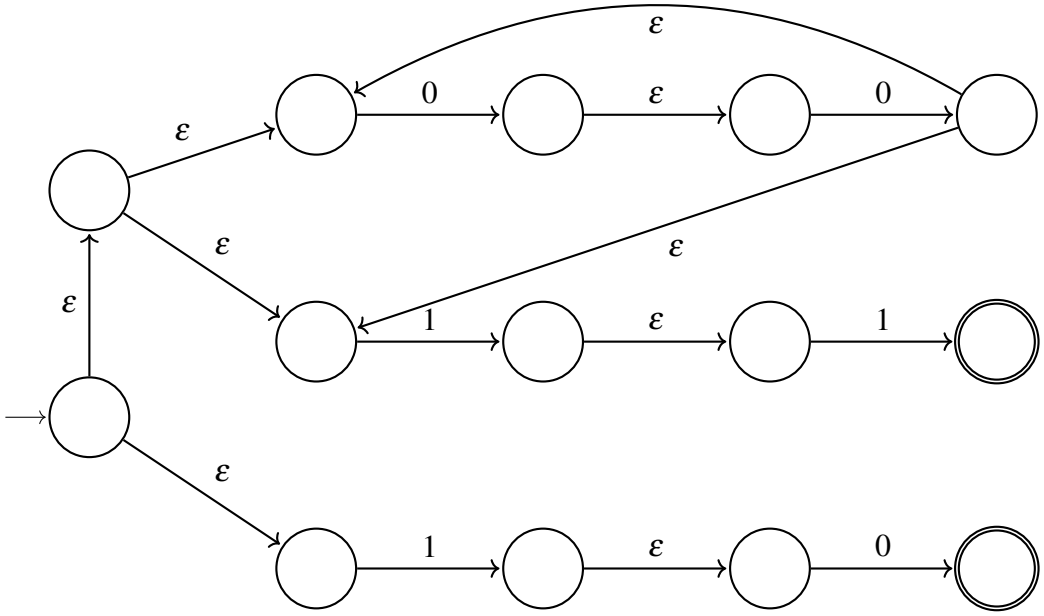
NFA for $(00)^\star$



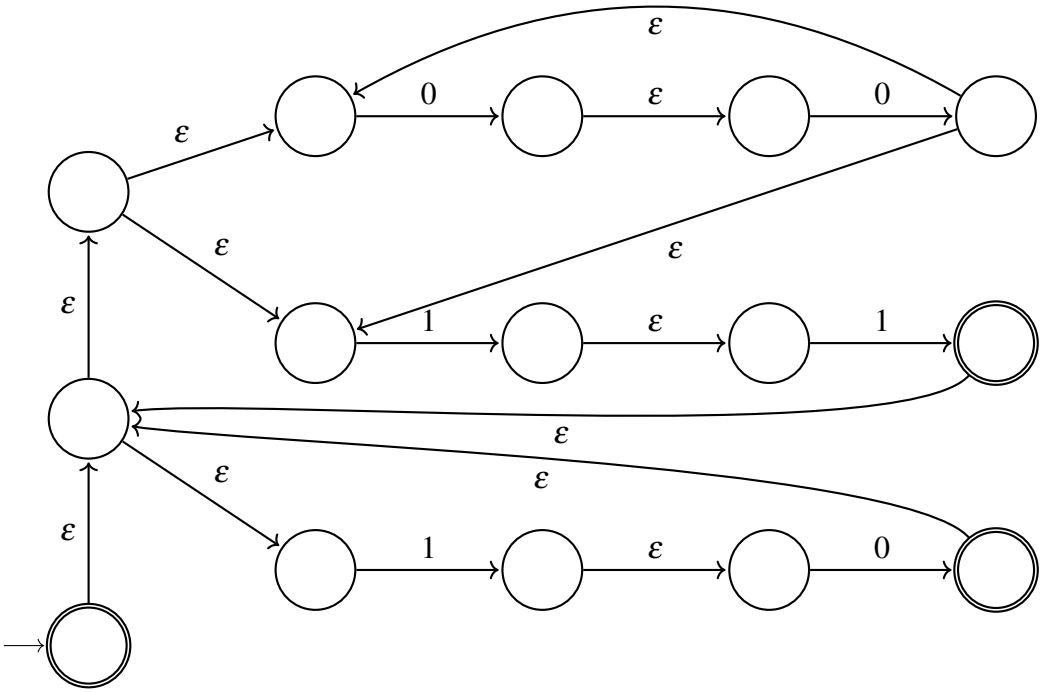
NFA for $(00)^\star(11)$



NFA for $(00)^\star(11)\cup 10$



NFA for $((00)^\star(11)\cup 10)^\star$



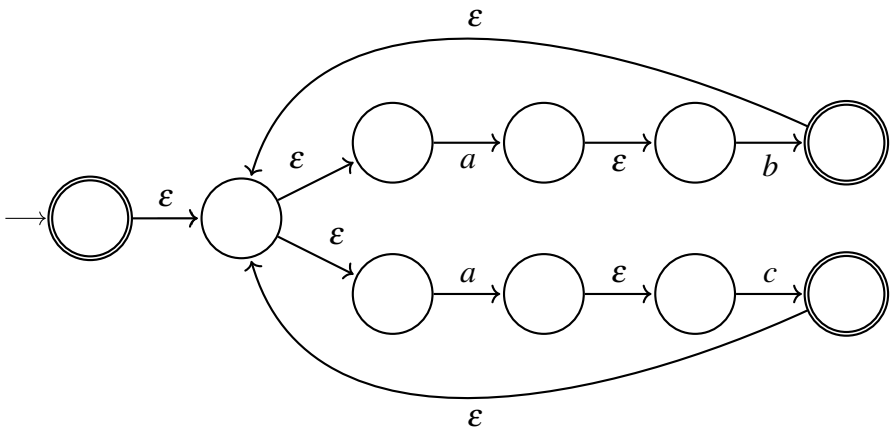
Problem 4.

1. Give examples of strings described by the RegEx $(ab \cup ac)^\star$
2. Rewriting the RegEx in an equivalent form, construct two NFA (from the two regular expressions) which recognise the same language

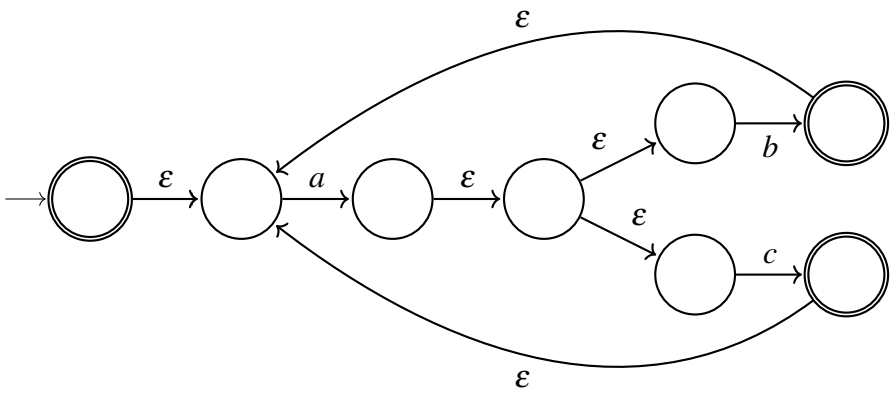
Solution 4.

1. $\{\varepsilon, ab, ac, abab, abac, acab, acac, ababab, ababac, ...\}$
2. $\big(a(b \cup c)\big)^\star$

   NFA for $(ab \cup ac)^\star$



NFA for $\big(a(b \cup c)\big)^\star$



Problem 5.

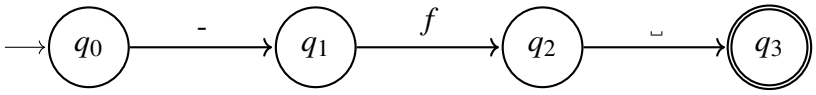
1. Write a regular expression that matches strings composed of: "-F␣" or "-f␣", followed by filenames consisting only any number of "a" (possibly none) followed by the extension ".tmp"

   Example: "-f aaaaa.tmp", "-f a.tmp", "-f .tmp" are all accepted, but "-fa.tmp", "-f b.temp", "-f aaa.pdf" are not.

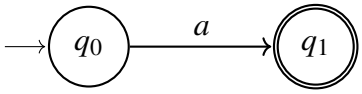
2. Use the construction methods shown in lectures to convert the regular expression into an NFA.

Solution 5.

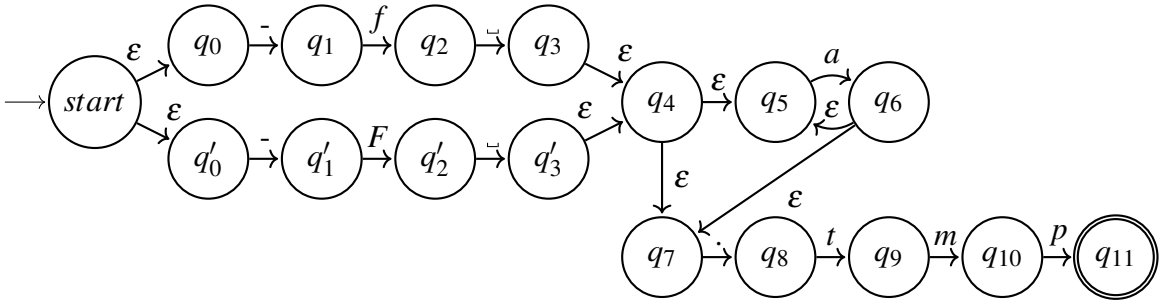
1. One such regular expression is $(((-F\text{␣})\cup(-f\text{␣}))\circ a^*\circ(.tmp))$
2. (a) Here is an automaton recognising only the string "-f␣"



   (b) Here is an automaton recognising only the string "a"



   (c) Here is the NFA:



Problem 6. Let N_1, N_2 be NFAS.

1. Devise an algorithm for testing whether or not $L(N_1) = \emptyset$.

2. Devise an algorithm for testing whether or not $L(N_1) \subseteq L(N_2)$.

Solution 6.

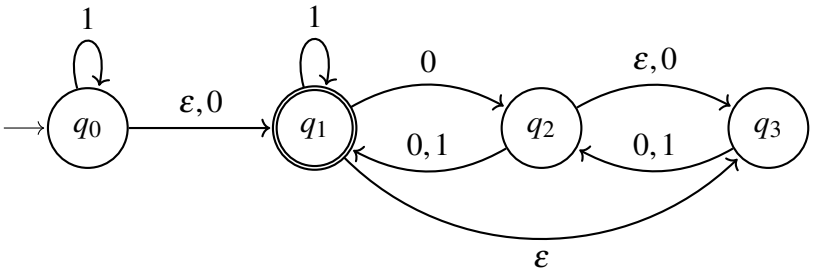
1. Check if there are any accept states in the *connected component* of the graph containing the start state.
2. Build an NFA N recognising $L(N_1) \setminus L(N_2)$ and check if $L(N) = \emptyset$. Use that $L(N) = \emptyset$ iff $L(N_1) \subseteq L(N_2)$.

Problem 7. For an NFA N, the notation $\xrightarrow{\varepsilon^*}$ refers to a binary relation on the states of N where $q \xrightarrow{\varepsilon^*} q'$ means that either $q = q'$ or there is a path from q to q' whose transitions are labeled with ε. Give a recursive definition of $\xrightarrow{\varepsilon^*}$.

Solution 7.

1. The base case: $q \xrightarrow{\varepsilon^*} q$ for every $q \in Q$.
2. If $q \xrightarrow{\varepsilon^*} q'$ and $q'' \in \delta(q', \varepsilon)$ then $q \xrightarrow{\varepsilon^*} q''$.

Problem 8. Consider the following NFA:

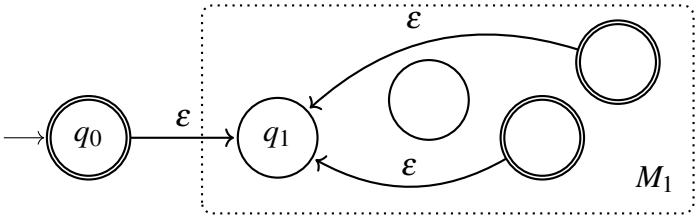


For each state q, write the set $E(q)$ of states q' such that $q \xrightarrow{\varepsilon^*} q'$. The set $E(q)$ is called the *epsilon-closure* of the state q.
Using the method from lectures, draw an equivalent NFA that has no ε-transitions.

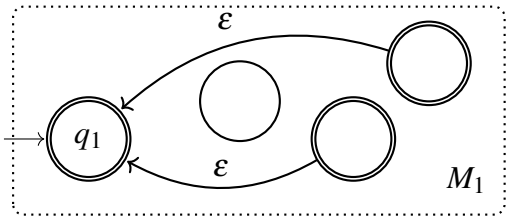
Solution 8.

1. $E(q_0) = \{q_0, q_1, q_3\}$
2. $E(q_1) = \{q_1, q_3\}$
3. $E(q_2) = \{q_2, q_3\}$
4. $E(q_3) = \{q_3\}$

Problem 9. Recall the construction of an NFA M recognising $L(M_1)^*$ given an NFA M_1:

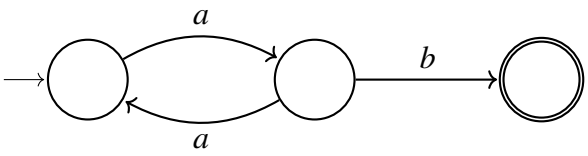


Let us now consider the following construction instead, where we do not add a new start state, and instead make the original start state accepting:



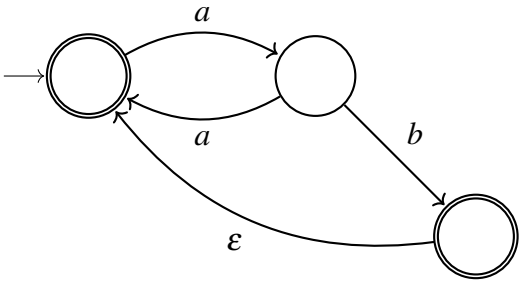
Show that the second construction does not work, i.e., find a finite automaton M_1 for which the automaton M built using the second construction method does *not* recognise $L(M_1)^*$

Solution 9. Let M_1 be:

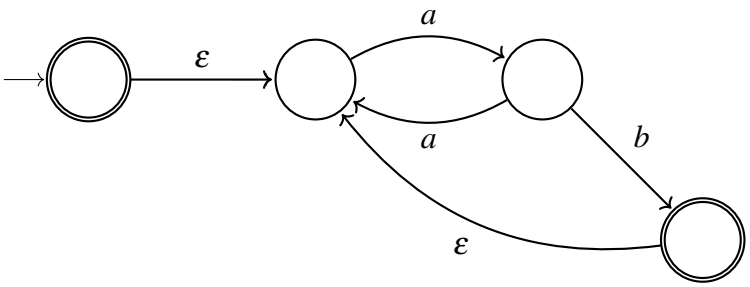


Then $L(M_1) = \{aa^{2k}b | k \geq 0\}$ (an odd number of a's followed by a b)

If we make an automata using the second approach, it will accept aa, but this is not in $L(M_1)^*$



The approach described in lectures would give us the following automata, which does work:

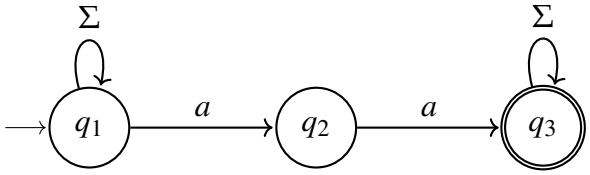


Problem 1.

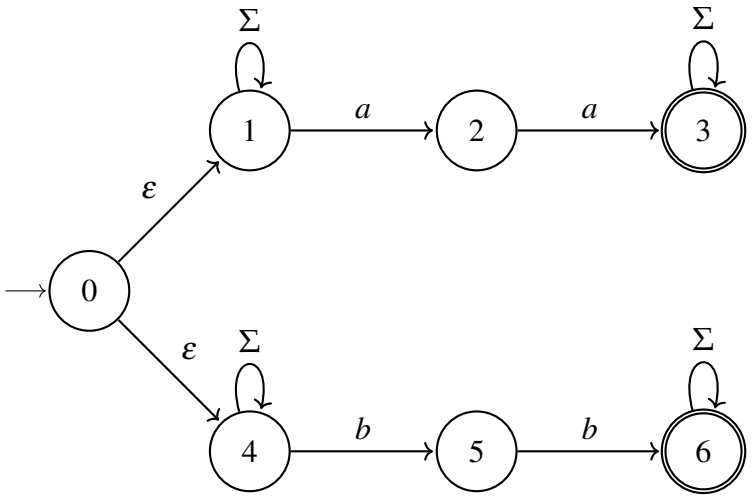
1. Devise an NFA which accepts strings containing two consecutive a's
2. Apply the union construction to deduce an NFA which accepts all strings over $\{a,b\}$ with two consecutive a's or two consecutive b's
3. Transform the NFA into an equivalent DFA, using the method shown in lectures

Solution 1.

1. Devise an NFA which accepts strings containing two consecutive a's

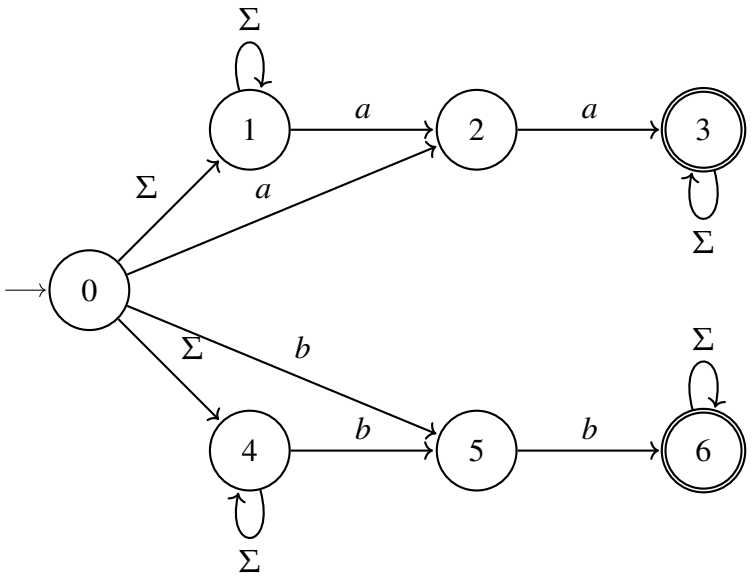


2. Apply the union construction to deduce an NFA which accepts all strings over $\{a,b\}$ with two consecutive a's or two consecutive b's



3. Transform the NFA into an equivalent DFA, using the method shown in lectures.

   First we remove the ε-transitions. To do this, we compute the epsilon-closure $E(q)$ of every state q: $E(0) = \{0,1,4\}$, and $E(q) = \{q\}$ for every other state q. So, after removing the ε-transitions we have the following NFA:



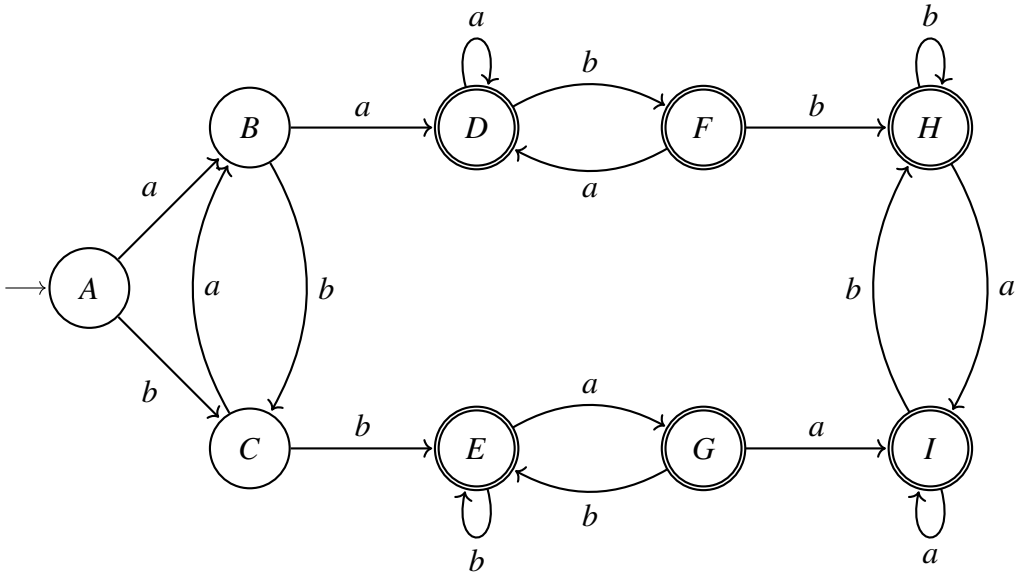
state	a	b
$\{0\}$	$\{1,2,4\}$	$\{1,4,5\}$
$\{1,2,4\}$	$\{1,2,3,4\}$	$\{1,4,5\}$
$\{1,4,5\}$	$\{1,2,4\}$	$\{1,4,5,6\}$
$\underline{\{1,2,3,4\}}$	$\{1,2,3,4\}$	$\{1,3,4,5\}$
$\underline{\{1,4,5,6\}}$	$\{1,2,4,6\}$	$\{1,4,5,6\}$
$\underline{\{1,3,4,5\}}$	$\{1,2,3,4\}$	$\{1,3,4,5,6\}$
$\underline{\{1,2,4,6\}}$	$\{1,2,3,4,6\}$	$\{1,4,5,6\}$
$\underline{\{1,3,4,5,6\}}$	$\{1,2,3,4,6\}$	$\{1,3,4,5,6\}$
$\underline{\{1,2,3,4,6\}}$	$\{1,2,3,4,6\}$	$\{1,3,4,5,6\}$

Relabel the states to make it easier to read:

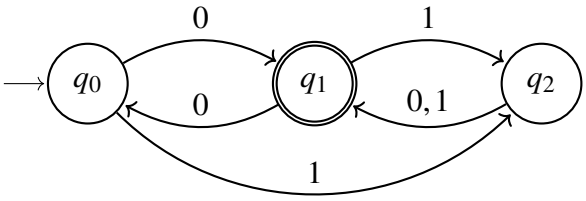
state	a	b
A	B	C
B	D	C
C	B	E
$\underline{D}$	D	F
$\underline{E}$	G	E
$\underline{F}$	D	H
$\underline{G}$	I	E
$\underline{H}$	I	H
$\underline{I}$	I	H

Start state is A

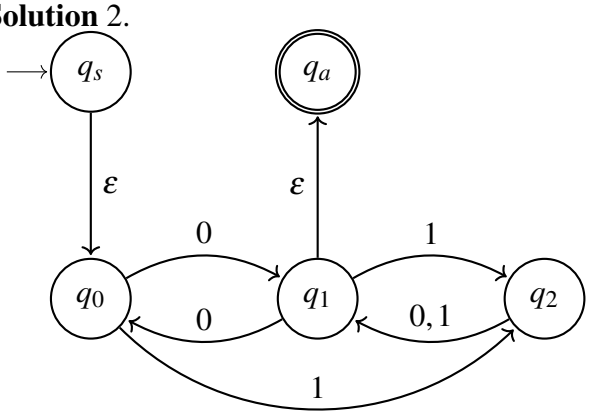
Set of accept states is $\{D,E,F,G,H,I\}$



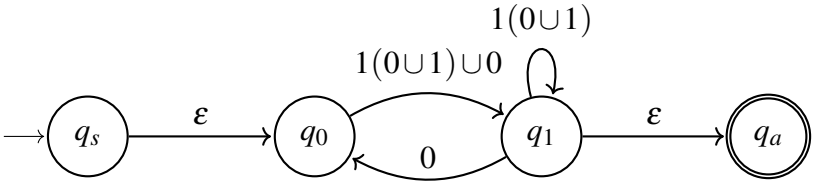
Problem 2. Convert the automaton to a regular expression.



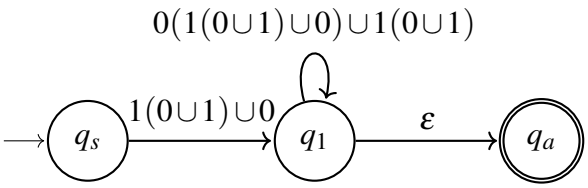
Solution 2.



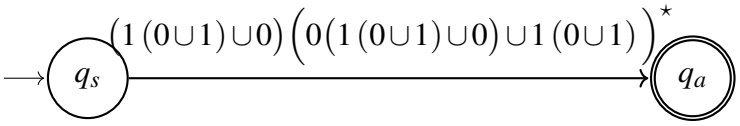
Eliminate q_2



Eliminate q_0



Eliminate q_0

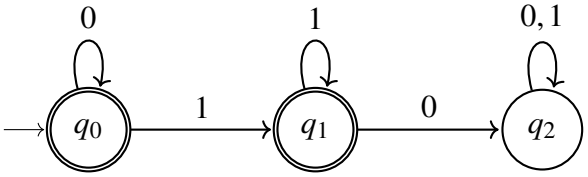


This is a very ugly RegEx. We should simplify it a bit.

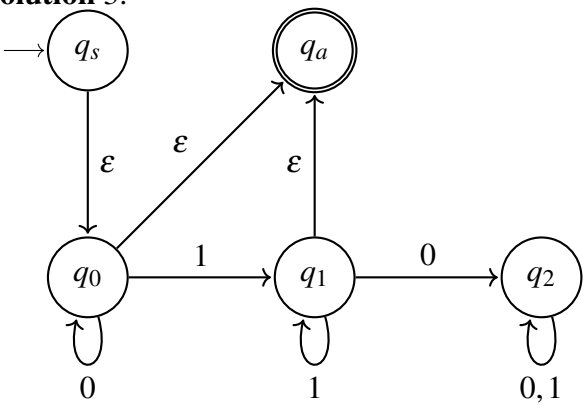
$$
\begin{aligned}
&(1(0\cup1)\cup0)\left(0(1(0\cup1)\cup0)\cup1(0\cup1)\right)^\star \\
&=(10\cup11\cup0)\left(0(1(0\cup1)\cup0)\cup1(0\cup1)\right)^\star \\
&=(10\cup11\cup0)\left(0(10\cup11\cup0)\cup1(0\cup1)\right)^\star \\
&=(10\cup11\cup0)\left((010\cup011\cup00)\cup1(0\cup1)\right)^\star \\
&=(10\cup11\cup0)\left((010\cup011\cup00)\cup(10\cup11)\right)^\star \\
&=(10\cup11\cup0)(010\cup011\cup00\cup10\cup11)^\star \\
&=(0\cup10\cup11)(00\cup010\cup011\cup10\cup11)^\star \\
&=(0\cup10\cup11)\left(00\cup(01\cup1)(0\cup1)\right)^\star
\end{aligned}
$$

When you have a fairly simple RegEx like this, it's a lot easier to see that it's correct.

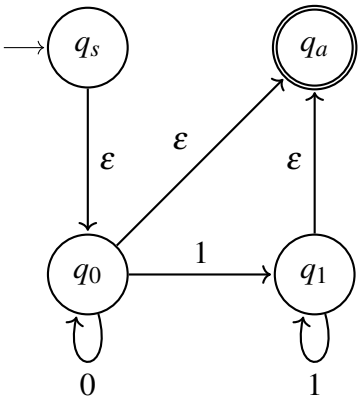
Problem 3. Convert the automaton to a regular expression.



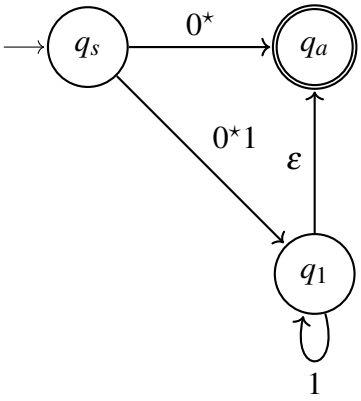
Solution 3.



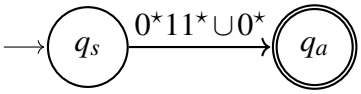
Eliminate q_2 (no outgoing transitions, very easy!)



Eliminate q_0



Eliminate q_1



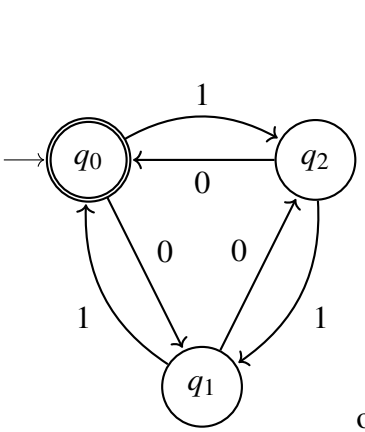
RegEx is $0^\star 11^\star \cup 0^\star$

Problem 4.

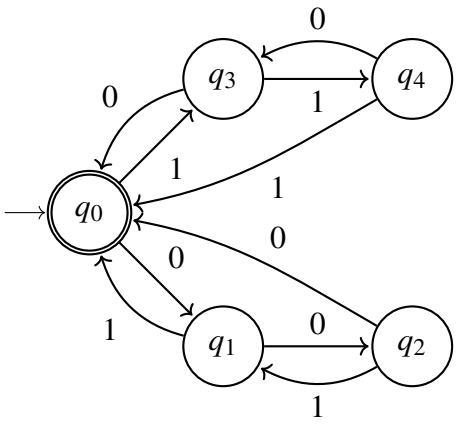
Construct a DFA which accepts strings of 0's and 1's, where the number of 1's modulo 3 equals the number of 0's modulo 3.

Give the corresponding RegEx

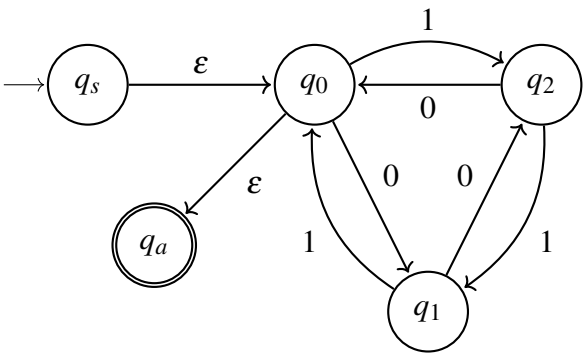
Solution 4. a)



or

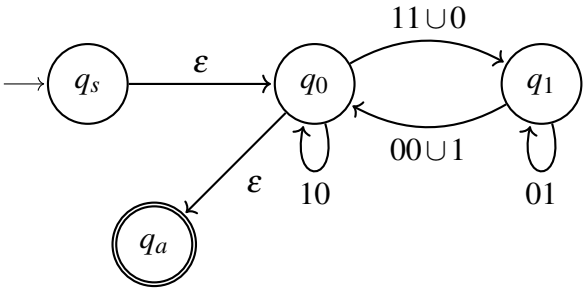


Using the first (smaller) automaton:



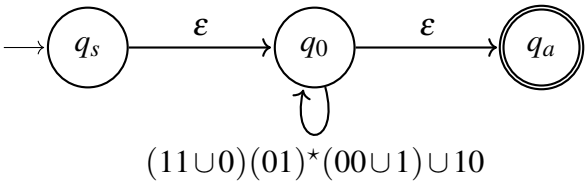
eliminate q_2:

$$
\begin{aligned}
(q_0,q_0) &: R_1 = 1, R_2 = \emptyset, R_3 = 0, R_4 = \emptyset, (R_1)(R_2)^\star(R_3)\cup(R_4) &&= 10\\
(q_0,q_1) &: R_1 = 1, R_2 = \emptyset, R_3 = 1, R_4 = 0, (R_1)(R_2)^\star(R_3)\cup(R_4) &&= 11\cup 0\\
(q_1,q_0) &: R_1 = 0, R_2 = \emptyset, R_3 = 0, R_4 = 1, (R_1)(R_2)^\star(R_3)\cup(R_4) &&= 00\cup 1\\
(q_1,q_1) &: R_1 = 0, R_2 = \emptyset, R_3 = 1, R_4 = \emptyset, (R_1)(R_2)^\star(R_3)\cup(R_4) &&= 01
\end{aligned}
$$

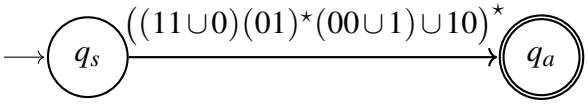


eliminate q_1:

$$
\begin{aligned}
(q_0,q_0) &: R_1 = 11\cup 0, R_2 = 01, R_3 = 00\cup 1, R_4 = 10\\
&\quad (R_1)(R_2)^\star(R_3)\cup(R_4) = (11\cup 0)(01)^\star(00\cup 1)\cup 10
\end{aligned}
$$



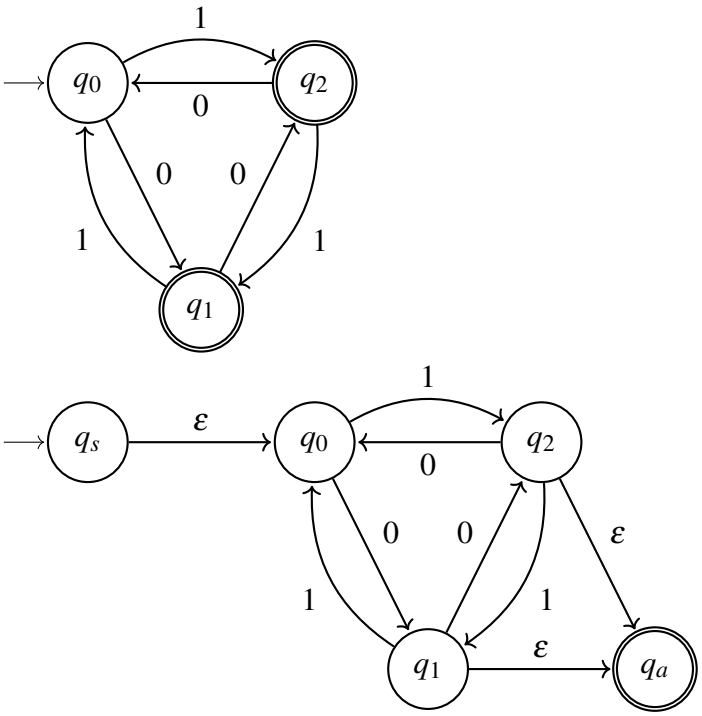
eliminate q_0



Problem 5. Construct a DFA which accepts strings of 0's and 1's, where the number of 1's modulo 3 does not equal the number of 0's modulo 3.

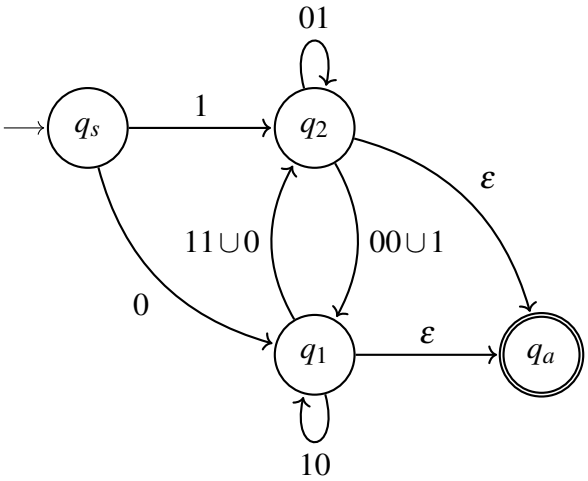
Give the corresponding RegEx

Solution 5.



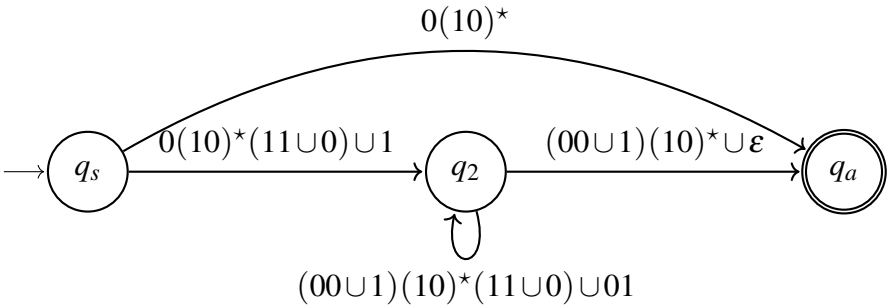
eliminate q_0:

$$
\begin{aligned}
(q_s,q_1) &: R_1=\varepsilon, R_2=\emptyset, R_3=0, R_4=\emptyset, (R_1)(R_2)^\star(R_3)\cup(R_4) &&= 0\\
(q_s,q_2) &: R_1=\varepsilon, R_2=\emptyset, R_3=1, R_4=\emptyset, (R_1)(R_2)^\star(R_3)\cup(R_4) &&= 1\\
(q_1,q_1) &: R_1=1, R_2=\emptyset, R_3=0, R_4=\emptyset, (R_1)(R_2)^\star(R_3)\cup(R_4) &&= 10\\
(q_1,q_2) &: R_1=1, R_2=\emptyset, R_3=1, R_4=0, (R_1)(R_2)^\star(R_3)\cup(R_4) &&= 11\cup 0\\
(q_2,q_1) &: R_1=0, R_2=\emptyset, R_3=0, R_4=1, (R_1)(R_2)^\star(R_3)\cup(R_4) &&= 00\cup 1\\
(q_2,q_2) &: R_1=0, R_2=\emptyset, R_3=1, R_4=\emptyset, (R_1)(R_2)^\star(R_3)\cup(R_4) &&= 01
\end{aligned}
$$



eliminate q_1:

$$(q_s,q_a): R_1=0, R_2=10, R_3=\varepsilon, R_4=\emptyset, (R_1)(R_2)^\star(R_3)\cup(R_4) = 0(10)^\star$$
$$(q_s,q_2): R_1=0, R_2=10, R_3=11\cup 0, R_4=1, (R_1)(R_2)^\star(R_3)\cup(R_4) = 0(10)^\star(11\cup 0)\cup 1$$
$$(q_2,q_a): R_1=00\cup 1, R_2=10, R_3=\varepsilon, R_4=\varepsilon, (R_1)(R_2)^\star(R_3)\cup(R_4) = (00\cup 1)(10)^\star\cup\varepsilon$$
$$(q_2,q_2): R_1=00\cup 1, R_2=10, R_3=11\cup 0, R_4=01, (R_1)(R_2)^\star(R_3)\cup(R_4) = (00\cup 1)(10)^\star(11\cup 0)\cup 01$$



Which gets us to

$$R_1 = 0(10)^\star(11\cup 0)\cup 1$$
$$R_2 = (00\cup 1)(10)^\star(11\cup 0)\cup 01$$
$$R_3 = (00\cup 1)(10)^\star\cup\varepsilon$$
$$R_4 = 0(10)^\star$$
$$(R_1)(R_2)^\star(R_3)\cup(R_4) = \big(0\,(10)^\star\,(11\cup 0)\cup 1\big)\big((00\cup 1)\,(10)^\star\,(11\cup 0)\cup 01\big)^\star\big((00\cup 1)\,(10)^\star\cup\varepsilon\big)\cup 0\,(10)^\star$$

Yikes. That's a pretty ugly RegEx! It looks like it's probably correct. Lets see how sane it is.

Let n be the number of 0's - number of 1's

$R_1 = 0(10)^\star(11\cup 0)\cup 1 \Rightarrow$	$n\%3 = 2$
$R_2 = (00\cup 1)(10)^\star(11\cup 0)\cup 01 \Rightarrow$	$n\%3 = 0$
$R_3 = (00\cup 1)(10)^\star\cup\varepsilon \Rightarrow$	$n\%3 = 2$ or 0
$R_4 = 0(10)^\star \Rightarrow$	$n\%3 = 1$
$(R_1)(R_2)^\star(R_3) \Rightarrow$	$n\%3 = 2$ or 1
$(R_1)(R_2)^\star(R_3)\cup R_4 \Rightarrow$	$n\%3 = 2$ or 1

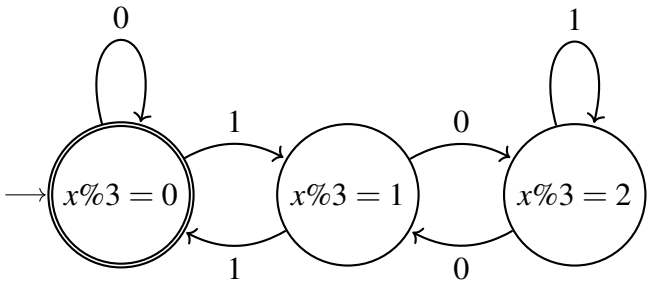
So, any string matching the RegEx does not have the same number of 0s and 1s mod 3. It's harder to confirm that this RegEx matches all such strings, but if there aren't any errors in the transformation from the NFA, then it will.

Problem 6.

1. Construct a DFA which accepts binary numbers that are multiples of 3, scanning the most significant bit (i.e. leftmost) first. Hint: consider the value of the remainder so far, as we scan across the number.
2. Deduce an automaton which accepts binary numbers which are NOT multiples of 3.

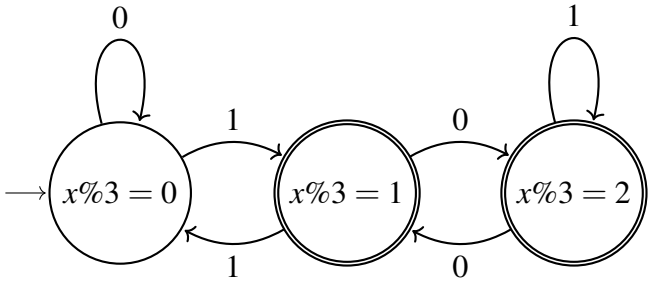
Solution 6. (For simplicity, we decided to treat empty binary strings as representing 0.)

1. Each state will remember what the remainder is so far.



   To understand the transitions, use the following reasoning. Suppose the automaton has read a string u corresponding to number n and is in the state numbered $i = n\%3$. Then if the next symbol read is a 1, the string read so far is $u1$, which corresponds to the number $2n+1$, which has remainder $2i+1\%3$ when divided by three (this explains the transition on input 1 from state 0 to state 1, from state 1 to state 0, and from state 2 to state 2). Similarly, if the next symbol read is a 0, the string read so far is $u0$, which corresponds to the number $2n$, which has remainder $2n\%3$ when divided by three (this explains the transition on input 0 from state 0 to itself, from state 1 to state 2, and from state 2 to state 1).

2. If we set $F' = Q \setminus F$ (i.e. invert the accept / non-accepting status of all the states), then the automata will accept the complement of the original language.



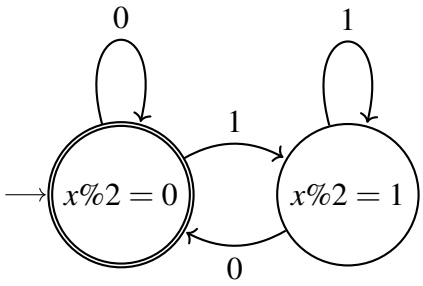
   Inverting the accept states of any *DFA M* will always result in a machine which recognises the complement of $L(M)$. However, this approach does *not* work for all NFA (think about why!)

Problem 7.

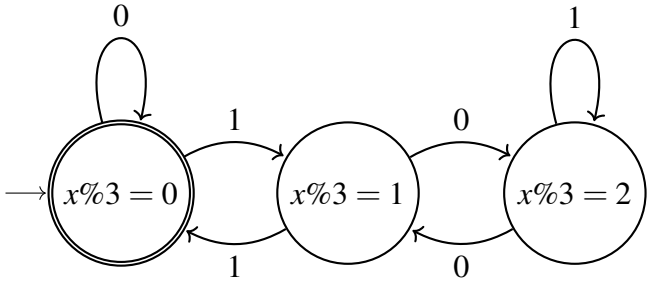
1. Construct a DFA which recognises binary numbers which are multiples of 2.
2. Construct a DFA which recognises binary numbers which are multiples of 3 (previous exercise)
3. Using these two DFA, construct an NFA which accepts binary numbers which are either not a multiple of 2, or not a multiple of 3, or both. (i.e. 2 is accepted, 3 is accepted, but 6 is not accepted)
4. Convert it to a DFA
5. Swap the accept and non-accept states. What is the language accepted by this new DFA?

Solution 7. (For simplicity, we decided to treat empty binary strings as representing 0.)

1. Construct a DFA which recognises binary numbers which are multiples of 2.

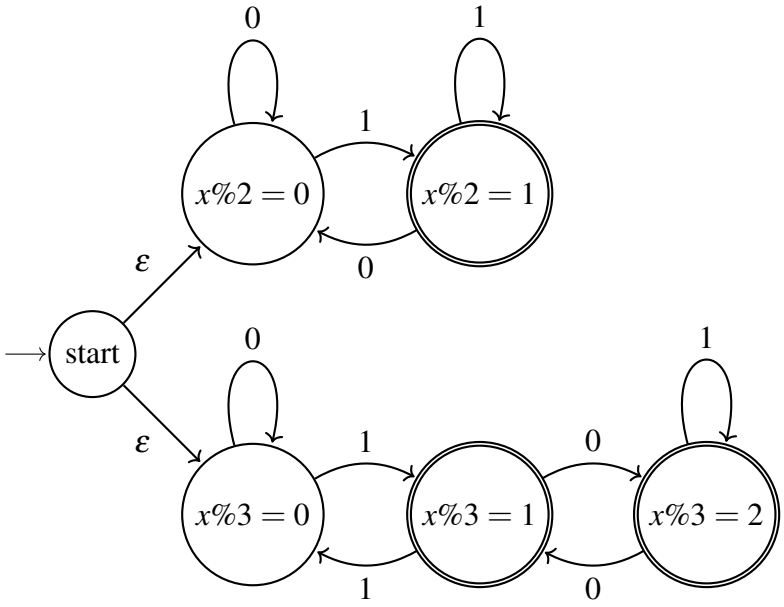


2. Construct a DFA which recognises binary numbers which are multiples of 3 (previous exercise)

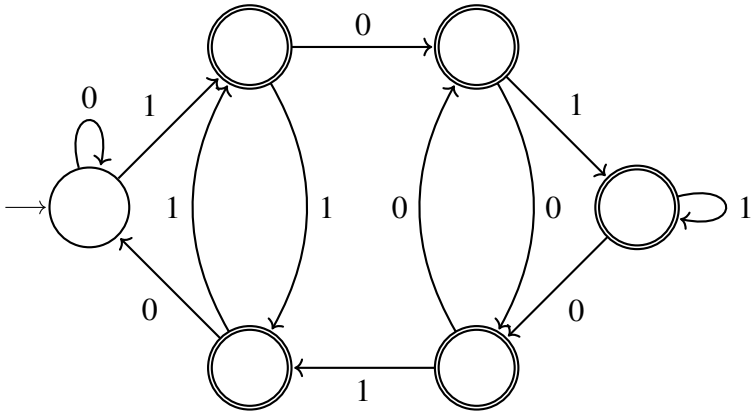


3. Using these two DFA, construct an NFA which accepts binary numbers which are either not a multiple of 2, or not a multiple of 3, or both. (i.e. 2 is accepted, 3 is accepted, but 6 is not accepted)

   We invert the accept/non accept states, then construct the union of the two DFA



4. Convert it to a DFA



5. Swap the accept and non-accept states. What is the language accepted by this new DFA?
   Binary numbers divisible by both 2 and 3 (i.e. binary numbers divisible by 6)
   We could have predicted this outcome using *DeMorgan's Law*: $(\neg A \vee \neg B) \equiv \neg(A \wedge B)$

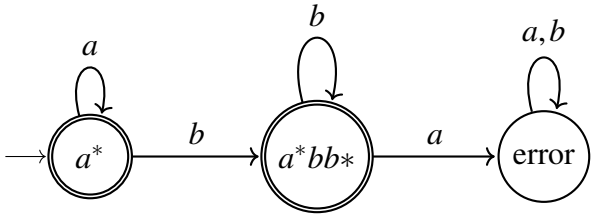
Problem 8.

1. The product technique for building a DFA recognisining the union of the languages of two DFAs can be slightly modified to get a DFA recognising the intersection of the languages of two DFAs. How?
2. Use the product technique to build a DFA for this language over $\{a,b\}$:

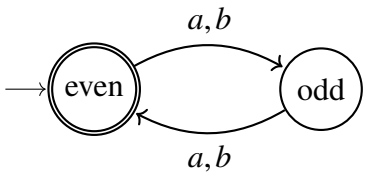
$$L = \{x| \text{ there is no } b \text{ before an } a \text{ in } x \text{ and } |x| \text{ is even}\}$$

Solution 8.

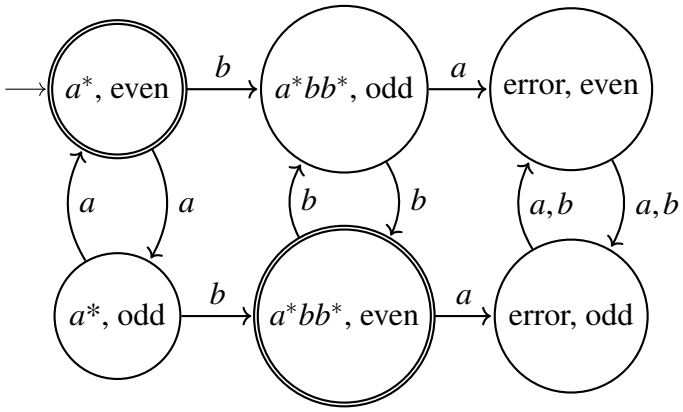
1. The only change is the set F' of final states: these should be the pairs (q_1,q_2) such that $q_1 \in F_1$ *and* $q_2 \in F_2$.
2. $L = \{x| \text{ there is no } b \text{ before an } a \text{ in } x\}$



$L = \{x||x| \text{ is even}\}$



Cross product:



Problem 9. Consider the decision problem which takes as input a regular expression R and string w, and outputs 1 if $w \in L(R)$, and 0 otherwise. This problem is called the *Regular-expression membership problem*. Give an algorithm that solves this decision problem.

Solution 9. Using the techniques from lectures:

1. Convert R into a DFA M such that $L(R) = L(M)$.
2. Check if $w \in L(M)$ using the algorithm for the DFA membership problem.

Problem 10. Consider the decision problem which takes as input regular expressions R_1, R_2, and outputs 1 if $L(R_1) = L(R_2)$, and 0 otherwise. This problem is called the *Regular-expression equivalence problem*. Give an algorithm that solves this decision problem.

Solution 10. Using the techniques from lectures:

1. Convert R_i into a DFA M_i such that $L(R_i) = L(M_i)$.
2. Check if $L(M_1) = L(M_2)$ using the algorithm for the DFA equivalence problem.

Problem 11. Show that $L = \{a^i b^j : i > j\}$ is not regular.

Solution 11.

1. Assume there is a DFA M that recognises L.
2. Write $f(w)$ for the state that M reaches after reading input w.
3. By PHP, there exists $m < n$ such that $f(a^n) = f(a^m)$.
4. But $a^n b^{n-1} \in L$ means that the path from state $f(a^n)$ labeled b^{n-1} ends in a final state.
5. So there is a path from the initial state to a final state labeled $a^m b^{n-1}$. So $a^m b^{n-1}$ is accepted by M.
6. But since $m \not> n-1$ (since, in fact, $m \leq n-1$), M accepts a word not in L.
7. This contradicts the assumption that M recognises L.

Problem 12. Show that $L = \{a^{n^2} : n \geq 0\}$ is not regular.

Solution 12. Use the fact that $(n+1)^2 = n^2 + 2n + 1$.

1. Suppose there is a DFA M that recognises L.
2. Write $f(w)$ for the state that M reaches after reading input w.
3. By PHP, there exists $n < m$ such that $f(a^{n^2}) = f(a^{m^2})$.
4. But $a^{(n+1)^2} \in L$ means that the path from state $f(a^{n^2})$ labeled a^{2n+1} ends in a final state.
5. So there is a path from the initial state to a final state labeled a^{m^2+2n+1}. So a^{m^2+2n+1} is accepted by M.
6. But a^{m^2+2n+1} is not in L since $m^2 < m^2+2n+1 < m^2+2m+1 = (m+1)^2$. So M accepts a word not in L.
7. This contradicts the assumption that M recognises L.
8. Conclude that no DFA recognises L.

Problem 1. Consider the following grammar:

$$
\begin{aligned}
E &\to E + T \mid E - T \mid T \\
T &\to F \times T \mid T / T \mid F \\
F &\to (E) \mid V \mid C \\
V &\to a \mid b \mid c \\
C &\to 1 \mid 2 \mid 3
\end{aligned}
$$

1. Indicate the set of variables, terminals, production rules, and the start variable.
2. Give a left-most derivation of the string $a + b \times c$
3. Give a right-most derivation of the string $a + b \times c$
4. Give a parse tree for $a \times b - 2 \times c$
5. Give a parse tree for $a \times (b - 2 \times c)$

Solution 1.

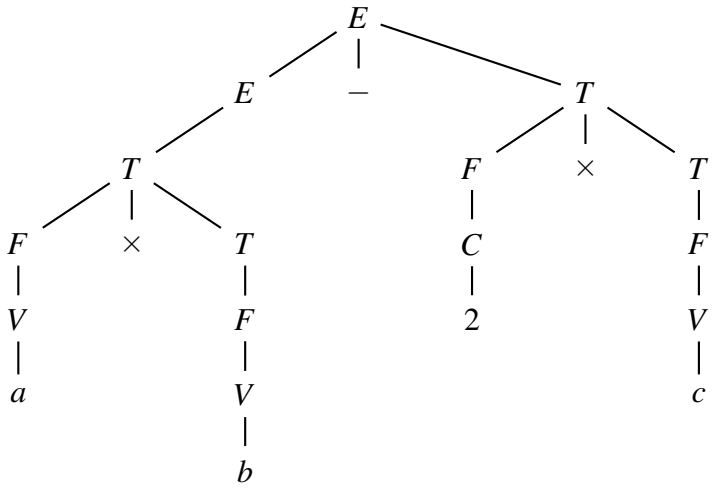
1. $V = \{E, T, F, V, C\}$
   $T = \{a, b, c, 1, 2, 3, (,), \times, /, +, -\}$
   Production rules as above (note there are 15 rules!)
   Start variable as E (if it's not stated we assume it's the first one, or S)
2.

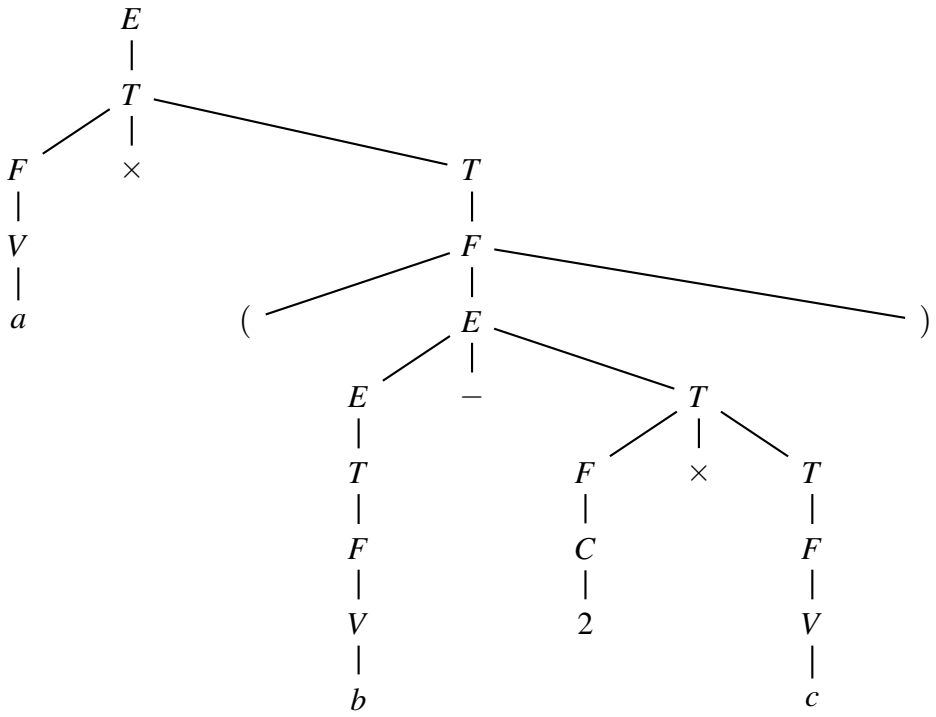
$$
\begin{aligned}
E &\Rightarrow E + T \\
&\Rightarrow T + T \\
&\Rightarrow F + T \\
&\Rightarrow V + T \\
&\Rightarrow a + T \\
&\Rightarrow a + F \times T \\
&\Rightarrow a + V \times T \\
&\Rightarrow a + b \times T \\
&\Rightarrow a + b \times F \\
&\Rightarrow a + b \times V \\
&\Rightarrow a + b \times c
\end{aligned}
$$

3.

$$
\begin{aligned}
E &\Rightarrow E + T \\
&\Rightarrow E + F \times T \\
&\Rightarrow E + F \times F \\
&\Rightarrow E + F \times V \\
&\Rightarrow E + F \times c \\
&\Rightarrow E + V \times c \\
&\Rightarrow E + b \times c \\
&\Rightarrow T + b \times c \\
&\Rightarrow F + b \times c \\
&\Rightarrow V + b \times c \\
&\Rightarrow a + b \times c
\end{aligned}
$$

4.





5.

Problem 2. Give a context-free grammar which accepts valid identifiers. A valid identifier begins with a letter or underscore, contains only letters, underscores, or digits, and is ended by a blank.

Solution 2.

$\langle Identifier \rangle \rightarrow \langle First \rangle \langle Rest \rangle \langle Space \rangle$ (i.e. the first char, the rest, then the terminating space)

$\langle First \rangle \rightarrow \langle Underscore \rangle \mid \langle Letter \rangle$

$\langle Rest \rangle \rightarrow \varepsilon \mid \langle Character \rangle \langle Rest \rangle$ (Recursively derive the middle of the string)

$\langle Character \rangle \rightarrow \langle Underscore \rangle \mid \langle Letter \rangle \mid \langle Digit \rangle$

$\langle Underscore \rangle \rightarrow \_$

$\langle Letter \rangle \rightarrow a \mid b \mid ... \mid Z$

$\langle Digit \rangle \rightarrow 0 \mid 1 \mid 2 \mid 3 \mid ... \mid 9$

$\langle Space \rangle \rightarrow \sqcup$

Problem 3. Some programming languages define the `if` statement in similar ways to the grammar given below. Show that this grammar is ambiguous:

$Conditional \rightarrow$ if $Condition$ then $Statement \mid$ if $Condition$ then $Statement$ else $Statement$

$Statement \rightarrow Conditional \mid S_1 \mid S_2$

$Condition \rightarrow C_1 \mid C_2$

Show that the program you provided can be interpreted in two different ways, resulting in programs with different behaviours.

What if we make the following change?

$Conditional \rightarrow$ if $Condition$ then $Statement$ endif $\mid$ if $Condition$ then $Statement$ else $Statement$ endif

Solution 3. Show that a string exists with two parse trees, or two leftmost derivations, or two rightmost derivations.

e.g. the string "if C1 then if C2 then S1 else S2" is ambiguous on this grammar. In particular, if $C1$ is true and C_2 is false, then one of the programs does S_2, and the other doesn't do S_2.

The suggested change would make this ambiguity disappear.

Problem 4. Extend the previous grammar to allow a statement to also be an assignment or a while loop.

Solution 4.

$$\begin{aligned}
\mathit{Conditional} &\to \text{ if } \mathit{Condition} \text{ then } \mathit{Statement} \text{ endif} \\
\mathit{Conditional} &\to \text{ if } \mathit{Condition} \text{ then } \mathit{Statement} \text{ else } \mathit{Statement} \text{ endif} \\
\mathit{Statement} &\to \mathit{Conditional} \mid S_1 \mid S_2 \mid \mathit{Assignment} \mid \mathit{Loop} \\
\mathit{Loop} &\to \text{ while } \mathit{Condition} \text{ do } \mathit{Statement} \text{ endloop} \\
\mathit{Assignment} &\to V := E \\
\mathit{Condition} &\to C_1 \mid C_2
\end{aligned}$$

(V and E also need rules. V might, for example, derive the $\langle \mathit{Identifier} \rangle$ rules, and E the arithmetic rules, from the earlier exercises.)

Problem 5. Find a context-free grammar for each of the following languages:

1. $\{bb, bbbb, bbbbbb, ...\} = \{(bb)^{n+1} \mid n \in \mathbb{N}\}$
2. $\{a, ba, bba, bbba, ...\} = \{b^n a \mid n \in \mathbb{N}\}$
3. $\{\varepsilon, ab, abab, ...\} = \{(ab)^n \mid n \in \mathbb{N}\}$
4. $\{ac, abc, abbbc, ...\} = \{ab^n c \mid n \in \mathbb{N}\}$
5. $\{a^m b^n \mid n, m \in \mathbb{N}\}$
6. $\{a^m b^n \mid n, m \in \mathbb{N}, m > 0\}$
7. $\{a^m b^n \mid n, m \in \mathbb{N}, m > 0, n > 0\}$
8. $\{a^n b^n : n > 0\}$

Solution 5.

1. $\{bb, bbbb, bbbbbb, ...\} = \{(bb)^{n+1} \mid n \in \mathbb{N}\}$

$$S \to bb \mid bbS \qquad (bSb, Sbb \text{ would also work fine})$$

2. $\{a, ba, bba, bbba, ...\} = \{b^n a \mid n \in \mathbb{N}\}$

$$S \to a \mid bS$$

3. $\{\varepsilon, ab, abab, ...\} = \{(ab)^n \mid n \in \mathbb{N}\}$

$$S \to \varepsilon \mid abS \qquad (Sab \text{ would also work fine})$$

4. $\{ac, abc, abbbc, ...\} = \{ab^n c \mid n \in \mathbb{N}\}$

$$\begin{aligned} S &\to aBc && \text{(this puts the a and c on the ends)} \\ B &\to bB \mid \varepsilon && \text{(derives any number of b's)} \end{aligned}$$

5. $\{a^m b^n \mid n, m \in \mathbb{N}\}$

$$\begin{aligned} S &\to AB \\ A &\to aA \mid \varepsilon && \text{(derives any number of a's)} \\ B &\to bB \mid \varepsilon && \text{(derives any number of b's)} \end{aligned}$$

6. $\{a^m b^n \mid n, m \in \mathbb{N}, m > 0\}$

$$\begin{aligned} S &\to AB \\ A &\to aA \mid a && \text{(derives at least one a)} \\ B &\to bB \mid \varepsilon && \text{(derives any number of b's)} \end{aligned}$$

7. $\{a^m b^n \mid n, m \in \mathbb{N}, m > 0, n > 0\}$

$$\begin{aligned} S &\to AB \\ A &\to aA \mid a && \text{(derives at least one a)} \\ B &\to bB \mid b && \text{(derives at least one b)} \end{aligned}$$

8. $\{a^n b^n : n > 0\}$

$$\begin{aligned} S &\to aTb \\ T &\to aTb \mid \varepsilon \end{aligned}$$

Problem 6. Show that the following grammars are ambiguous. In other words, find a string which has two different parse trees (equivalently two left most derivations, or two right most derivations)

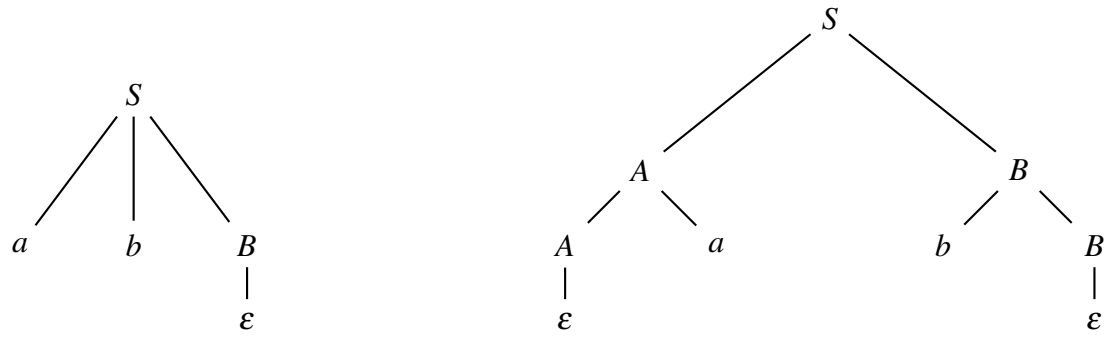
1. $S \to a \mid SbS$
2. $S \to abB \mid AB$
   $A \to \varepsilon \mid Aa$
   $B \to \varepsilon \mid bB$
3. $S \to aS \mid Sa \mid a$
4. $S \to S[S]S \mid \varepsilon$

Solution 6.

1. "ababa" is ambiguous. (i.e. it has two parse trees (you need to show them, or give two leftmost derivations, or two rightmost deriviations))

   e.g. these are both leftmost derivations of "ababa"
   $S \Rightarrow SbS \Rightarrow abS \Rightarrow abSbS \Rightarrow ababS \Rightarrow ababa$
   $S \Rightarrow SbS \Rightarrow SbSbS \Rightarrow abSbS \Rightarrow ababS \Rightarrow ababa$

2. "ab" has two parse trees



3. "aa" has two parse trees, because it has two left derivations:

$$S \Rightarrow aS \Rightarrow aa$$

$$S \Rightarrow Sa \Rightarrow aa$$

4. "[][]" has two parse trees, because it has two left derivations:

$$S \Rightarrow S[S]S \Rightarrow S[S]S[S]S \Rightarrow ... \Rightarrow [][]$$

$$S \Rightarrow S[S]S \Rightarrow [S]S \Rightarrow []S \Rightarrow []S[S]S \Rightarrow ... \Rightarrow [][]$$

Problem 7. For each of the grammars in the previous exercise, first describe in English the language generated, and try to find an equivalent context-free grammar which is not ambiguous.

Solution 7.

1. The grammar generates the language described by $(ab)^\star a$, so $S \to a \mid abS$ is equivalent (and is not ambiguous)
2. The grammar generates the langauge of strings of any number of a's followed by any number of b's. The first production rule for S was unnecessary, since the second one could also derive every string it generated:
   $S \to AB$
   $A \to \varepsilon \mid Aa$
   $B \to \varepsilon \mid bB$
3. Strings of one or more a's:
   $S \to aS \mid a$
4. Balanced square brackets:
   $S \to S[S] \mid \varepsilon$

Problem 8. Write a context-free grammar for the language of balanced parentheses. For example, $(()())$ and $(())()$ are in the language but $)($ is not.

Solution 8.

$$S \to SS \mid (S) \mid \varepsilon$$

Problem 1. Transform the following grammar into Chomsky Normal Form (CNF). Let w be the string $();$. Show derivations in both grammars of w. Then apply the CYK algorithm to the grammar in CNF show that w is generated by it.

$$S \to B;$$
$$B \to (B)B \mid \varepsilon$$

Solution 1.

1. START step: nothing to do
2. TERM step:

$$S \to BN_1$$
$$B \to N_2BN_3B \mid \varepsilon$$
$$N_1 \to ;$$
$$N_2 \to ($$
$$N_3 \to)$$

3. BIN step:

$$S \to BN_1$$
$$B \to N_2B_1 \mid \varepsilon$$
$$B_1 \to BB_2$$
$$B_2 \to N_3B$$
$$N_1 \to ;$$
$$N_2 \to ($$
$$N_3 \to)$$

4. DEL step:

$$S \to BN_1 \mid N_1$$
$$B \to N_2B_1$$
$$B_1 \to BB_2 \mid B_2$$
$$B_2 \to N_3B \mid N_3$$
$$N_1 \to ;$$
$$N_2 \to ($$
$$N_3 \to)$$

5. UNIT step:

$$
\begin{aligned}
S &\to BN_1 \mid ; \\
B &\to N_2 B_1 \\
B_1 &\to BB_2 \mid N_3 B \mid) \\
B_2 &\to N_3 B \mid) \\
N_1 &\to \ ; \\
N_2 &\to \ (\\
N_3 &\to \ )
\end{aligned}
$$

A derivation of w in the original grammar is $S \Rightarrow B; \Rightarrow (B)B; \Rightarrow ()B; \Rightarrow ();$.
A derivation of w in the new grammar is $S \Rightarrow BN_1 \Rightarrow B; \Rightarrow N_2B_1; \Rightarrow (B_1; \Rightarrow ();$.
The CYK table for w is:

S		
B	$\emptyset$	
N_2	N_3, B_2, B_1	N_1, S
(	)	;

Problem 2. Convert the following grammar into CNF:

$$
\begin{aligned}
S &\to ASA \mid aB \\
A &\to B \mid S \\
B &\to b \mid \varepsilon
\end{aligned}
$$

Then draw the CYK table for the string *abb* and decide if it is generated by the grammar or not.

Problem 3. The following process takes a CFG G as input and returns a set $\mathbf{X}$ of variables as output.

1. Let $\mathbf{X}$ consist of all variables A such that $A \to \varepsilon$ is a rule.
2. Repeat the following until $\mathbf{X}$ no longer changes:
   (a) For every rule $A \to u$ in R such that A is a variable and u only consists of variables,
   (b) if A is not in $\mathbf{X}$ and every variable in u is in X, then
   (c) add A to X.
3. Return $\mathbf{X}$.

What does the process do? i.e., which variables end up in $\mathbf{X}$ and which do not? Explain how to use this procedure to transform a CFG G that does not generate the empty-string into a CFG G' that does not have any epsilon-rules, i.e., rules of the form $A \to \varepsilon$.

Solution 3. This process returns the set $\mathbf{X}$ of variables A such that $A \Rightarrow^* \varepsilon$, i.e., such that A derives the empty-string.

Let G be a CFG that does not generate the empty-string. Compute the set $\mathbf{X}$ of variables that derive the empty-string. For every rule $A \to v$ where v is a string of terminals and non-terminals, add to G all rules of the form $A \to v'$ where v' is any non-empty string obtained by removing one or more occurences of variables in $\mathbf{X}$ from v. Finally, remove all rules of the form $A \to \varepsilon$ from G.

In this tutorial you will get practice designing/programming Turing Machines to do certain tasks. Think of this as if you had a low-level programming language that has some finite internal memory and just a single data structure, a tape.

Problem 1. Consider the following Turing machine $M = (Q, \Sigma, \Gamma, \delta, q_0, q_{accept}, q_{reject})$ where

- $Q = \{q_0, q_1, q_2, q_3, q_{accept}, q_{reject}\}$
- $\Sigma = \{0, 1\}$
- $\Gamma = \{0, 1, X, Y, \sqcup\}$
- $\delta : Q \times \Gamma \to Q \times \Gamma \times \{\text{L}, \text{R}\}$ is given by

	0	1	X	Y	$\sqcup$
q_0	(q_1, X, R)			(q_3, Y, R)	
q_1	$(q_1, 0, \text{R})$	(q_2, Y, L)		(q_1, Y, R)	
q_2	$(q_2, 0, \text{L})$		(q_0, X, R)	(q_2, Y, L)	
q_3				(q_3, Y, R)	$(q_{accept}, \sqcup, \text{R})$

Blank entries are of the form $(q_{reject}, 0, \text{R})$.

1. For each of the following input strings, show the computation (sequence of configurations) of M, and say whether the machine accepts, rejects, or diverges: 01, 10, and 00111.
2. Give pseudo-code or an English-description of how M operates.
3. What language does M recognise?
4. Is M a decider?

Solution 1.

1. (a) The sequence of configuration of M on input 01 is:

$$q_0 01$$
$$X q_1 1$$
$$q_2 X Y$$
$$X q_0 Y$$
$$X Y q_3 \sqcup$$
$$X Y \sqcup q_4$$

The input string 01 is accepted.

(b) 10

$$q_0 10$$
$$0 q_{rejected} 0$$

The input string is rejected.

(c) 00111

$$
\begin{aligned}
&q_0 00111\\
&X q_1 0111\\
&X 0 q_1 111\\
&X q_2 0 Y 11\\
&q_2 X 0 Y 11\\
&X q_0 0 Y 11\\
&X X q_1 Y 11\\
&X X Y q_1 11\\
&X X q_2 Y Y 1\\
&X q_2 X Y Y 1\\
&X X q_0 Y Y 1\\
&X X Y q_3 Y 1\\
&X X Y Y q_3 1\\
&X X Y Y 0 q_{rejected}
\end{aligned}
$$

The input string is rejected.

2. (a) If we did not start on a 0, reject
   (b) Replace the 0 with an X
   (c) Move right until a 1 occurs
   (d) Replace the 1 with a Y
   (e) Move left until we find the X marker
   (f) Move right once. If we are on a 0, goto (b)
   (g) Move right across the tape, skipping X and Y.
   - If a 1 or 0 is scanned, reject
   - If a blank is scanned, accept
3. M recognises the language $\{0^n 1^n : n > 0\}$.
4. M is a decider, i.e., it does not diverge on any input string. The reason is that the only loop is between steps (b) and (f). However, each time the machine does a pass it reduces the number of 0s. At some point there will be no more 0's, and so the loop will end.

Problem 2.

1. Can a Turing Machine ever write the blank symbol (⊔) on its tape?
2. Can the tape alphabet Γ be the same as the input alphabet Σ?
3. Can a Turing Machine ever be in the same location in two successive steps?
4. What happens if a Turing machine enters an accepting or rejecting state? Can it continue its computation?
5. Does a TM always halt on every input?

Solution 2.

1. Yes, e.g., the transition $\delta(q,a) = (q', \sqcup, \text{L})$ writes the blank symbol.
2. No, because the blank symbol cannot be in the input alphabet
3. If the Turing machine's tape is infinite in both directions, then it will always move. However, there are also a definitions of Turing machines in which the tape is only unbounded to the right. In these Turing machines the machine will not move if it tries to move left when it is already at the left edge of the tape. Also, in the variant of TM in which there is a "stay put" option (in addition to moving left and moving right), then yes, the machine can be in the same location in two successive steps.
4. No, it halts.
5. No, it may never reach the accepting state or the rejecting state, in which case it runs forever. It is said to *diverge*.

Problem 3. In this problem you will see that TMs can also compute things (and not just decide things).

Devise a TM M that, given an input string of the form 0^n10^m reaches an accepting configuration with 0^{n+m} written on the tape.

1. Give the pseudo-code or an English description of M.
2. Give the formal description of M, i.e., states, transitions, etc.
3. Give the computation, i.e., sequence of configuration, of M on input 00001000.

Solution 3.

1. (a) Move to the right until we reach a 1 (skipping 0s)
   (b) Rewrite the 1 with a 0
   (c) Move to the right until we reach a blank (skipping 0s)
   (d) Move to the left once, to rewrite the last 0 with a blank
2. - $Q = \{q_0, q_1, q_2, q_{accept}, q_{reject}\}$
   - $\Sigma = \{0,1\}$
   - $\Gamma = \{0,1,\sqcup\}$
   - $q_0 \in Q$ is the start state
   - $\delta : Q \times \Gamma \to Q \times \Gamma \times \{\text{L},\text{R}\}$ is given by:

	0	1	$\sqcup$
q_0	$(q_0, 0, \text{R})$	$(q_1, 0, \text{R})$	$(q_{accept}, \sqcup, \text{R})$
q_1	$(q_1, 0, \text{R})$		$(q_2, \sqcup, \text{L})$
q_2	$(q_{accept}, \sqcup, \text{R})$		

Blank entries are of the form $(q_{reject}, 0, R)$.

This machine takes a string of the form 0^n10^m as input and when it finnishes computing it has written 0^{n+m} on the tape. In other words, TM can also be used to *compute functions* (not just to decide languages). Note the similarity with computers.

3.

$q_0$00001000
0$q_0$0001000
00$q_0$001000
000$q_0$01000
0000$q_0$1000
00000$q_1$000
000000$q_1$00
0000000$q_1$0
00000000q_1
0000000$q_2$0
0000000q_3␣

Problem 4. Devise a TM that recognises the language $\{w\#w \mid w \in \{0,1\}^\star\}$. You might find it easier to write a semi-formal description first, then write the transition table in full.

Solution 4.

q_{start}	if read 0, write X, move right to state q_{0a} if read 1, write X, move right to state q_{1a} if read #, write #, move right to state q_{end} if read X, move right else reject
q_{0a}	move right through any 0's and 1's if read #, move right and change to state q_{0b} if read ␣, reject
q_{0b}	move right through any X's if read 0, write X, move left to state q_{return} if read 1 or ␣, reject
q_{1a}	move right through any 0's and 1's if read #, move right and change to state q_{1b} if read ␣, reject
q_{1b}	move right through any X's if read 1, write X, move left to state q_{return} if read 0 or ␣, reject
q_{end}	if read ␣, write ␣, move right to state q_{accept} if read X, move right else reject
q_{return}	move left until read a ␣, then move right to state q_{start}

Problem 5. Devise a turing machine that takes as input a unary number outputs a tape with that many Quokka on it.

e.g. input tape 0000␣... output tape: *quokkaquokkaquokkaquokka*␣...

Solution 5.

q_{start}	if read 0, write ⊔, move right to state q_{right} if read q, move left to state q_{end}
q_{right}	move right until read ⊔, then write q and move to state q_{uokka1}
q_{uokka1}	write u and move right to state q_{uokka2}
q_{uokka2}	write o and move right to state q_{uokka3}
q_{uokka3}	write k and move right to state q_{uokka4}
q_{uokka4}	write k and move right to state q_{uokka5}
q_{uokka5}	write a and move left to state q_{reset}
q_{reset}	move left until read ⊔, then move right to state q_start
q_{end}	accept

This TM will repeatedly replace the 0's with blanks one by one, each time moving to the right end of the tape to write a quokka, then back to the left to the start of the remaining input. When the original input is gone, our tape will contain nothing but quokkas.

Problem 6. Make a turing machine that multiplies two unary numbers.
e.g. input tape 0000⊔000⊔... output tape: 000000000000⊔...

Solution 6.

1	if there is a 0, write ⊔and go right to (2) otherwise write ⊔and go to (a)
2	move right through the 0's to the next blank, go right to (3)
3	(start to copy the second unary number once) if there is a 0, rewrite it as X and go right to (4), otherwise go left to (8)
4	move right through the 0's to the next blank, go right to (5)
5	move right through the 0's to the next blank, rewrite with 0, go left to (6)
6	move left through 0's, until read a blank, move left to (7)
7	move left through 0's, until read an X, move right to (3)
8	(there weren't any 0's left) move left, rewriting all the X's to 0 when you read ⊔, move left to (9)
9	move left until read ⊔, at which point move right to (1)
a	move right replacing everything with ⊔, until reading a ⊔ (halt)

- States 1,2 deletes one digit from the first unary number
- States 3 – 9 is a subroutine which appends the second unary number to the end of the tape (important - while it marks off the number with X's, it changes them back to 0's before returning)
- State a is reached when the first unary number is gone. It wipes the second unary number from the tape then halts

Because it type-sets better, we use B instead of ⊔.
Example $3*2=6$:

```
 1 000B00BBBBBBBB
B 2 00B00BBBBBBBB
B0 2 0B00BBBBBBBB
B00 2 B00BBBBBBBB
B00B 3 00BBBBBBBB
B00BX 4 0BBBBBBBB
B00BX0 4 BBBBBBBB
```

```
B00BX0B 5 BBBBBBB
B00BX0B 6 0BBBBBB
B00BX0 6 B0BBBBBB
B00BX 7 0B0BBBBBB
B00B 7 X0B0BBBBBB
B00BX 3 0B0BBBBBB
B00BXX 4 B0BBBBBB
B00BXXB 5 0BBBBBB
B00BXXB0 5 BBBBBB
B00BXXB 6 00BBBBB
B00BXX 6 B00BBBBB
B00BX 7 XB00BBBBB
B00BXX 3 B00BBBBB
B00BXX 3 B00BBBBB
B00BX 8 XB00BBBBB
B00B 8 X0B00BBBBB
B00 8 B00B00BBBBB
B0 9 0B00B00BBBBB
B 9 00B00B00BBBBB
9 B00B00B00BBBBB
B 1 00B00B00BBBBB
...
BB 1 0B00B0000BBB
...
BBB 1 B00B000000B
BBBB a 00B000000B
BBBBB a 0B000000B
BBBBBB a B000000B
halt
```

Problem 7.
Devise a turing machine which takes a binary number as input and either

- outputs a factor of that number, and rejects
- or, accepts if the number was prime

e.g.

- input 1101␣... accepts (13 is prime).
- input 1111␣... rejects with tape 11␣... because 3 is a factor of 15

You may find it easier to first design subroutines of the machine, e.g.

- convert the number to unary
- determine if a given unary number is a factor of another unary number
- ...

It is sufficient to list the states and briefly explain the purpose of each one, rather than filling out the entire table.

Problem 8. Fill in the blanks.
A TM M recognises a language L means that for every input string $w \in \Sigma^*$:

1. if $w \in L$ then M ...
2. if $w \notin L$ then M ...

A TM M decides a language L means that for every input string $w \in \Sigma^*$:

1. if $w \in L$ then M ...
2. if $w \notin L$ then M ...

Solution 8.
A TM M recognises a language L means that for every input string $w \in \Sigma^*$:

1. if $w \in L$ then M accept w.
2. if $w \notin L$ then M does not accept w.

A TM M decides a language L means that for every input string $w \in \Sigma^*$:

1. if $w \in L$ then M accepts w.
2. if $w \notin L$ then M rejects w.

Note that a TM might not accept w but also not reject w, i.e., it might diverge.

Problem 9. In this problem we will see how to build new decidable/recognisable languages from old ones.
Give reasons for the following facts:

1. The decidable/recognisable languages are closed under union.
2. The decidable languages are closed under complement.
3. The decidable languages are closed under intersection.
4. A language is decidable iff it and its complement are recognisable.

Solution 9.

1. Let L_1 and L_2 be languages that are decided (recognised) by TMs M_1 and M_2 respectively. We construct a TM that decides (recognised) $L_2 \cup L_2$ as follows.
   On input x:
   (a) Run M_1 and M_2 in parallel on x.
   (b) Accept if either M_1 or M_2 accepts.
2. Suppose L is decided by M.
   (a) Define M' to be the same as M with q_{accept} and q_{reject} swapped.
   (b) M' decides $\Sigma^* \setminus L$.
3. Follow from the previous two facts by De Morgan.
4. There are two directions to handle.
   (a) $\Rightarrow$: follows from the closure properties of decidable languages.

(b) $\Leftarrow$: We construct a decider D for L from a TM M that recognises L and a TM M' that recognises the complement of L as follows. On input x:

i. Run M and M' on x in parallel.

ii. If M accepts, then accept.

iii. If M' accepts, then reject.

Problem 10. Is the intersection of two recognisable languages recognisable?

Problem 11.

1. Argue that every regular language is Turing-decidable.
2. Argue that every context-free language is Turing-decidable.

Solution 11.

1. Given a DFA $D = (Q, \Sigma, \delta, q_0, F)$ build a TM M_D that takes $w \in \Sigma^*$ as input and simualtes D as follows. It does this with the following transitions: for every transition $\delta(q,a) = q'$ of the DFA we add a transition of M_D that maps (q,a) to (q',a,R), i.e., don't change the input, move to the right, and change state. In addition, we have a transition that maps $(q,\sqcup)$ to $(q_{accept},\sqcup,R)$ for every $q \in F$, and a transition that maps $(q,\sqcup)$ to $(q_{reject},\sqcup,R)$ for every $q \notin F$.
2. There are a number of ways to do this. One way is to convert the CFG into a pushdown automaton (i.e., NFA with stack), and simulate the stack with the tape. Another way is to take a CFG in CNF for the grammar, and to have the TM implement the CYK algorithm. Another way is to take a CFG for the grammar, and to build a nondeterministic Turing-machine that simulates derivations of the grammar.

1 Turing machines, decidability and undecidability

Problem 1. State the Church-Turing thesis.

Problem 2. Give a direct argument that the decidable languages are closed under intersection. Does your argument work for recognisable languages?

Solution 2. If TM M_i decides language L_i, for $i = 1, 2$, define a TM M that decides $L_1 \cap L_2$ as follows:

- On input x:
  - Run M_1 on x. If it rejects then reject.
  - Run M_2 on x. If it rejects then reject.
  - Accept.

This also works for recognisable languages. Indeed, if both machines accept then also M accepts. And if at least one machine does not accept, either because it rejects or diverges, then so does M.

Problem 3. Is it the case that the complement of a recognisable language is recognisable?

Solution 3. No. For instance, A_{TM} is recognisable, but its complement, the set of all strings that either do not encode inputs of the form $\langle M, w \rangle$, or if they do then M does not accept w, is not recognisable. Because, if it were, then A_{TM} would be decidable (since if a language and its complement are both recognisable, then the language is decidable).

Problem 4. Is it the case that the intersection of a decidable language and a recognisable language is recognisable?

2 Lambda calculus

Problem 5. Give a recursive definition of free and bound variables for lambda-terms.

Solution 5. The free variables can be defined as follows:

- $FV(x) = \{x\}$
- $FV(MN) = FV(M) \cup FV(N)$
- $FV(\lambda x.M) = FV(M) \setminus \{x\}$

A variable occuring in the term that is not free is called bound. Note that, as usual, a variable may have both free and bound occurrences in a term.

Problem 6. In each of the following expressions, indentify which occurences of variables are free and which are bound.

1. x
2. (xy)
3. $\lambda x.x$

4. $\lambda x.y$
5. $\lambda x.(xy)$
6. $(x(\lambda x.(xy)))$
7. $\lambda x.(\lambda x.(xy))$
8. $\lambda x.(\lambda y.(xy))$

Solution 6.

1. x is free
2. x and y are both free
3. x is bound. Note that there is only one *occurence* of x in this formula (λx is the abstraction, not an occurence).
4. y is free. Note that there is no *occurence* of x in this formula!
5. x is bound, y is free
6. first x is free, second x is bound, y is free
7. x is bound *to the second* λx, y is free
8. x is bound, y is bound

Problem 7. Rewrite the following expressions to remove name clashes (we call this "α-conversion")

1. $(\lambda x.(xx))(\lambda x.(xx))$
2. $x(\lambda x.(xy))$
3. $\lambda x.(\lambda x.(xy))$

Solution 7.

1. $(\lambda x.(xx))(\lambda y.(yy))$
2. $x(\lambda z.(zy))$
3. $\lambda z.(\lambda x.(xy))$ or $\lambda x.(\lambda z.(zy))$ are both good answers.

Problem 8. Reduce the following expressions, using β-reductions. You will find it a bit easier to do if you perform α-conversions whenever relevant.

1. $(\lambda x.x)y$
2. $(\lambda x.y)z$
3. $(\lambda x.(xy))y$
4. $(((\lambda x.(\lambda x.(xy)))a)b)$

5. $((\lambda x.(xy))(\lambda z.z))$
6. $(((\lambda x.(xx))(\lambda x.x))x)$
7. $((\lambda x.(xx))(\lambda x.(xx)))$

Solution 8.

1. y
2. y
3. (yy)
4. $$\begin{aligned} & (((\lambda x.(\lambda x.(xy)))a)b) \\ \longrightarrow\ & ((\lambda x.(xy))b) && (\beta\text{-reduction}) \\ \longrightarrow\ & (by) && (\beta\text{-reduction}) \end{aligned}$$

   Explanation of the first step: We want to apply the axiom:

   $$(\lambda x.M)N = M[N/x] \qquad (\beta\text{-reduction})$$

   To the subexpression:

   $$(\lambda x.(\lambda x.(xy)))a$$

   Let $N = a$, and $M = \lambda x.(xy)$

   There are no free occurences of x in M, so $M[N/x] = M$

   To make it a bit more obvious, we could've used α-conversion (renaming) first, like this:

   $$\begin{aligned} & (((\lambda x.(\lambda x.(xy)))a)b) \\ \longrightarrow\ & (((\lambda x.(\lambda z.(zy)))a)b) && (\alpha\text{-reduction}) \\ \longrightarrow\ & ((\lambda z.(zy))b) && (\beta\text{-reduction}) \\ \longrightarrow\ & (by) && (\beta\text{-reduction}) \end{aligned}$$

5. $$\begin{aligned} & ((\lambda x.(xy))(\lambda z.z)) \\ \longrightarrow\ & ((\lambda z.z)y) && (\beta\text{-reduction}) \\ \longrightarrow\ & y && (\beta\text{-reduction}) \end{aligned}$$

6. $$\begin{aligned} & (((\lambda x.(xx))(\lambda x.x))x) \\ \longrightarrow\ & (((\lambda x.x)(\lambda x.x))x) && (\beta\text{-reduction}) \\ \longrightarrow\ & ((\lambda x.x)x) && (\beta\text{-reduction}) \\ \longrightarrow\ & x && (\beta\text{-reduction}) \end{aligned}$$

That wasn't too bad, but it's perhaps a little easier to follow with some α-reductions:

$$
\begin{aligned}
& (((\lambda x.(xx))(\lambda x.x))x) && \\
\longrightarrow\ & (((\lambda y.(yy))(\lambda x.x))x) && (\alpha\text{-reduction}) \\
\longrightarrow\ & (((\lambda y.(yy))(\lambda z.z))x) && (\alpha\text{-reduction}) \\
\longrightarrow\ & (((\lambda z.z)(\lambda z.z))x) && (\beta\text{-reduction}) \\
\longrightarrow\ & (((\lambda z.z)(\lambda a.a))x) && (\alpha\text{-reduction}) \\
\longrightarrow\ & ((\lambda a.a)x) && (\beta\text{-reduction}) \\
\longrightarrow\ & x && (\beta\text{-reduction})
\end{aligned}
$$

7.

$$
\begin{aligned}
& ((\lambda x.(xx))(\lambda x.(xx))) && \\
\longrightarrow\ & ((\lambda x.(xx))(\lambda x.(xx))) && (\beta\text{-reduction}) \\
\longrightarrow\ & ((\lambda x.(xx))(\lambda x.(xx))) && (\beta\text{-reduction}) \\
\longrightarrow\ & ((\lambda x.(xx))(\lambda x.(xx))) && (\beta\text{-reduction})
\end{aligned}
$$

... hmmm

Does α-reduction help?

$$
\begin{aligned}
& ((\lambda x.(xx))(\lambda x.(xx))) && \\
\longrightarrow\ & ((\lambda x.(xx))(\lambda y.(yy))) && (\alpha\text{-reduction}) \\
\longrightarrow\ & ((\lambda y.(yy))(\lambda y.(yy))) && (\beta\text{-reduction}) \\
\longrightarrow\ & ((\lambda x.(xx))(\lambda y.(yy))) && (\alpha\text{-reduction})
\end{aligned}
$$

... oh dear

This is an example of an expression that cannot be reduced to a normal form.

Problem 9. For each of the following formulae expand the notation to include all parentheses and operators. E.g., $(\lambda xyz.xy)a = ((\lambda x.(\lambda y.(\lambda z.(xy))))a)$

1. $\lambda xy.x$
2. $abcd$
3. $\lambda x.xyz$
4. $\lambda x.x(yz)$
5. $(\lambda x.xy)(\lambda x.xy)$

Solution 9.

1. $(\lambda x.(\lambda y.x))$
2. $(((ab)c)d)$
3. $(\lambda x.((xy)z))$
4. $(\lambda x.(x(yz)))$

5. $\Big((\lambda x.(xy))(\lambda x.(xy))\Big)$

Problem 10. Here are some encodings of Propositional Logic in the Lambda Calculus:

- TRUE = $\lambda xy.x$
- FALSE = $\lambda xy.y$
- NOT = $\lambda fxy.fyx$
- AND = $\lambda xy.xyx$

Prove the following equalities about these encodings (hint: it's often wise to avoid rewriting macros (like 'TRUE') as their full formulae until you actually need to, as this reduces both writing and confusion!)

1. NOT TRUE = FALSE
2. AND (NOT TRUE) TRUE = FALSE

Solution 10.

1. NOT TRUE = FALSE

$$
\begin{aligned}
& NOT\ TRUE \\
&= (\lambda fxy.fyx)TRUE && \text{(encoding for } NOT\text{)} \\
&\xrightarrow{\beta} \lambda xy.TRUE\ yx && (\beta\text{-reduction}) \\
&= \lambda xy.(\lambda xy.x)\ yx && \text{(encoding for } TRUE\text{)} \\
&\xrightarrow{\alpha} \lambda xy.(\lambda ay.a)\ yx && (\alpha\text{-reduction}) \\
&\xrightarrow{\alpha} \lambda xy.(\lambda ab.a)\ yx && (\alpha\text{-reduction}) \\
&\xrightarrow{\beta} \lambda xy.(\lambda b.y)x && (\beta\text{-reduction}) \\
&\xrightarrow{\beta} \lambda xy.y && (\beta\text{-reduction}) \\
&= FALSE && \text{(encoding for } FALSE\text{)}
\end{aligned}
$$

2. AND (NOT TRUE) TRUE = FALSE

$$
\begin{aligned}
& AND\ (NOT\ TRUE)\ TRUE \\
=_{\beta}\ & AND\ FALSE\ TRUE && \text{(previous qn)} \\
=\ & (\lambda xy.xyx)\ FALSE\ TRUE \\
\xrightarrow{\beta}\ & (\lambda y.FALSE\ y\ FALSE)\ TRUE \\
\xrightarrow{\beta}\ & FALSE\ TRUE\ FALSE \\
=\ & (\lambda xy.y)\ TRUE\ (\lambda xy.y) \\
\xrightarrow{\alpha}\ & (\lambda xy.y)\ TRUE\ (\lambda ab.b) \\
\xrightarrow{\beta}\ & (\lambda y.y)(\lambda ab.b) \\
\xrightarrow{\beta}\ & \lambda ab.b \\
=\ & FALSE
\end{aligned}
$$