Fall 2018

TÖL301G Formal Languages and Computability

Esa Hyytiä
Department of Computer Science
University of Iceland

Example Solutions to Weekly Exercises

Contents

The second draft version

Problems:

1. Let $Q_n = 1^2 + 2^2 + \ldots + n^2$. This sum reduces to a polynomial of third degree (cf. tutorial). Determine the polynomial and prove that it gives the correct answer for all $n \geq 1$.

   Answer:

   Proceeding "mechanically", let $P_n = an^3 + bn^2 + cn + d$ denote the polynomial with unknown coefficients. As $Q_0 = 0$, we immediately have $d = 0$. Then, with $n = 1$, $Q_1 = 1^2$ and

   $$P_1 = a + b + c = 1.$$

   With $n = 2$, $Q_2 = 1^2 + 2^2 = 5$ and

   $$\begin{aligned} P_2 = 8a + 4b + 2c &= 5 \\ 6a + 2b + &= 3. \end{aligned}$$

   Finally, with $n = 3$, $Q_3 = 1^2 + 2^2 + 3^2 = 14$ and

   $$\begin{aligned} P_3 = 27a + 9b + 3c &= 14 \\ 24a + 6b &= 11 \end{aligned}$$

   Combining the last two, gives $b = 1/2$, and consequently, $a = 1/3$ and $c = 1/6$. Hence, P_n is correct at the first four points if

   $$P_n = \frac{2n^3 + 3n^2 + n}{6}.$$

   For $n \geq 3$, consider the difference $P_{n+1} - P_n$, which gives

   $$P_{n+1} - P_n = \ldots = (n+1)^2. \tag{1}$$

   Proof by induction: (i) claim holds for $n = 0, \ldots, 3$. Suppose it holds for $0, \ldots, n$, then by (1) it holds for $n + 1$. Claim holds for any given n.

2. Consider the two DFAs M_1 and M_2 discussed in Tutorial 1.

   a) Does M_1 accept string $abba$?
   b) Does M_2 accept string $abba$?

   Answer:

   Following the "arrows" accordingly we accept if the final state is an accept state, and otherwise we reject. a) yes, b) no.

3. Figure 1 shows a DFA for binary input. Let us interpret "accept" states as "1" and others as "0", so that each input sequence (string) generates an output sequence (string) of equal length.

   a) What is the output sequence corresponding to 010101?
   b) What is the output sequence corresponding to 010100?
   c) What are the last six output bits for 000111010101101110101010101110110100010100?

   Note: Answer by giving a binary number of 6 bits, e.g., 111000.

   Answer:

   Again we follow the arrows and if after each move we consider the state we are in, if it is an accept state the output is 1 otherwise it is 0

a) 001010

b) 001010

c) 001010

(staring the input-output pairs, and applying our mental "machine-learning algorithm"*, it seems that the output is equal to the input string with one step delay . . . and analysis of the state transition diagram proves that this indeed is the case!)*

4. Design a DFA corresponding to the classical paper-scissors-stone game. The alphabet consists of three symbols, say (a, b, c) for paper, scissors and stone, respectively, and each game is a sequence of pairs $x_1y_1x_2y_2x_3y_3\ldots$ where (x_i, y_i) denote the choice of player 1 and 2 at round i. The game ends once $x_i \neq y_i$ (remaining moves can be ignored).

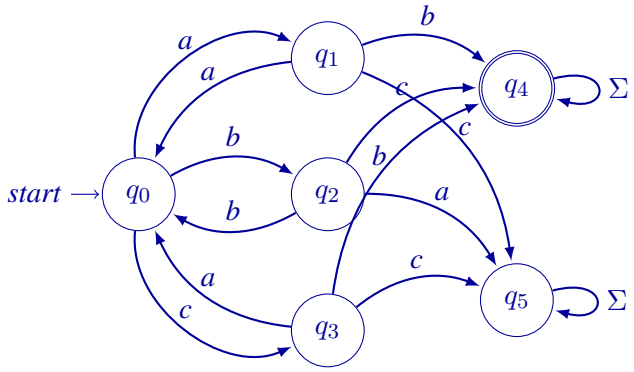
   a) Draw the corresponding state diagram.

   b) How many states are needed?

   c) Suppose player 1 chooses *a stone* always, whereas player 2 chooses a symbol uniformly at random. What is the probability that the game lasts more than 3 rounds?

   Answer:

   a) Let us interpret the question so that a string is accepted only if player 1 wins:



   b) 6 in above, if the winner is irrelevant, then 5.

   c) More than 3 rounds: $1 - 2/3 - (1/3) \cdot (2/3) = 1/3$.

5. Draw a state diagram for a DFA that *accepts* any binary string (e.g., 0100011) that has three or more consecutive zeroes.

   Answer:

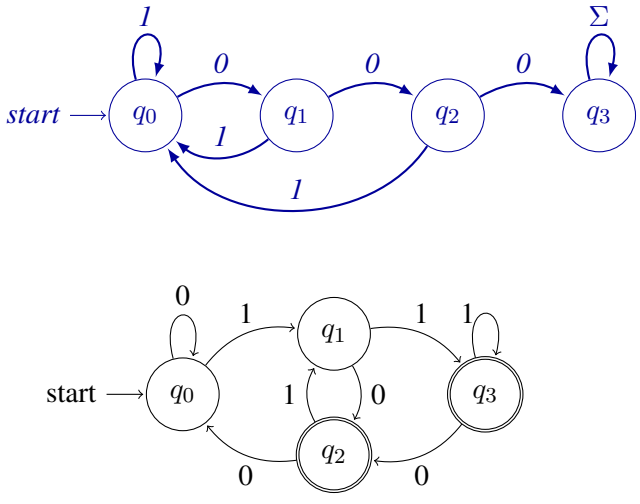


Figure 1: DFA with 4 states.

Problems:

1. Let $\mathcal{P}$ denote the set of polynomials with integer coefficients, i.e., functions $\mathbb{R} \to \mathbb{R}$ of form $a_0 + a_1 x + a_2 x^2 + \ldots + a_n x^n$, where the a_i are integers. Hence, e.g., $f(x) = 1 - x^2$ belongs to $\mathcal{P}$, while $x/2$ and $\sin(x)$ do not. Define mappings (functions)

$$I(f) = \int_0^x f(s)\,ds, \qquad D(f) = \frac{d}{dx} f(x) = f'(x),$$

i.e., a definite integral and derivative of a given function f, yielding another function as a result.

   a) Is $\mathcal{P}$ closed under integral $I(f)$?
   b) Is $\mathcal{P}$ closed under derivation $D(f)$?

   Answer:

   a) No. Consider e.g., $f(x) = x$, which maps to $g(x) = (1/2)x^2 \notin \mathcal{P}$.
   b) Yes. $f'(x) = a_1 + 2a_2 x + \ldots + n a_n x^{n-1} \in \mathcal{P}$.

2. Let A be a regular language, $B = A^*$, and $C = A \circ B$.

   a) Is $A = B$?
   b) Is $B = C$?
   c) Is $C = A$?

   In each case, argue briefly why.

   Answer:

   a) No, because B includes, e.g., an empty string which may not be in A.
   b) Only if an empty string belongs to A. In general, no.
   c) Similar as a), i.e., no.

3. Give regular expressions (cf. tutorial and `grep`) corresponding to the following criterion:

   a) Lines which lengths are multiples of three (3,6,9,...)
   b) Lines with three fields (CSV, comma separated files, i.e., look for lines with two commas)
   c) Lines where the second field matches your email address (again a CSV file)

   You can assume that the alphabet consists of letters a-z, numbers 0-9 and few necessary special symbols such as "," and "@".

   Answer:

   For example as follows:

```
grep "^\(...\)\{1,\}$"
grep "^[^,]*,[^,]*,[^,]*$"
grep "^[^,]*,foo@bar.com,[^,]*$"
```

4. Design the corresponding DFAs (either give a formal definition, 5-tuple, or draw a state diagram).

   Answer:

   a) Trivial, 4 states, of which last three repeat. One accepting state.
   b) Similarly, 4 states needed, of which one is an accept state.
   c) Expand the state in b) corresponding to the 2nd field to check the email address.

Problems:

1. Consider the two NFAs of Problem 1.16 from the book. Which of the following strings the machines accept:

string	Machine (a)	Machine (b)
$aaaaa$		
$bbbbb$		
$abba$		
$baba$		

(Indicate which of the four strings each machine accepts)

Answer:

string	*Machine (a)*	*Machine (b)*
aaaaa	✓	✓
bbbbb	✗	✗
abba	✓	✓
baba	✗	✗

2. Suppose $\Sigma = \{0, 1\}$ and let M_i denote an NFA that recognizes the strings with the ith last symbol 1, and $\bar{M}_i$ its complement, i.e., an NFA that recognizes string where the ith last bit is zero.

   a) Does $M_1 \cap M_2 \cap \bar{M}_3 \cap M_4$ recognize string 11101101011?
   b) Does $M_1 \cup M_2 \cup \bar{M}_3 \cup M_4$ recognize string 11111111111?
   c) Does $(M_1)^*$ recognize string 1011?

Answer:

Yes, yes, yes.

3. Convert the NFA depicted in Fig. 2 into an equivalent DFA.

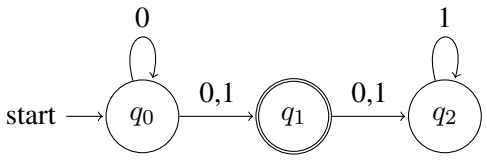


Figure 2: NFA with 3 states.

Answer:

In principle, the DFA corresponding to the given NFA looks as in Figure 3:

However, as the right side of the NFA is irrelevant for whatever a string is accepted or not (there are no transitions to accept states from q_2), we can reduce the DFA considerably. The result is depicted in Figure 4.

4. Consider the Exercise 1.24 from the book and the finite state transducer (FST) T_1. Here each transition generates also an output symbol (the second parameter).

   a) What is the output on input 011:

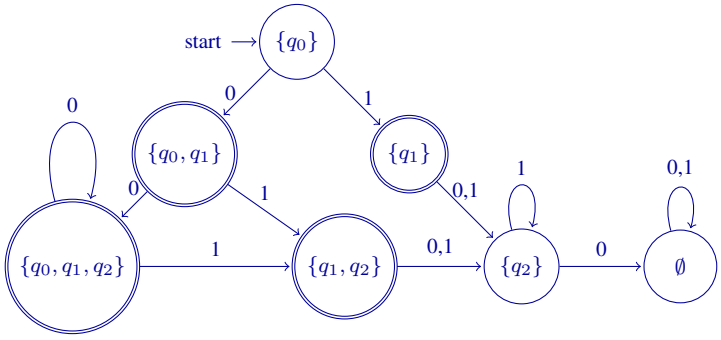


Figure 3: First solution.

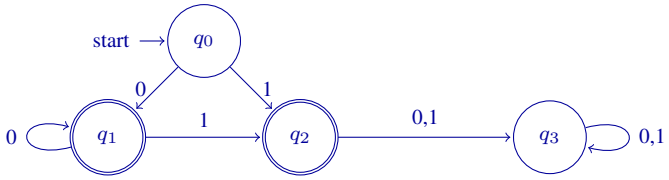


Figure 4: Second solution.

b) What is the output on input 211:

c) What is the output on input 0202:

Answer:

a) 000

b) 111

c) 0101

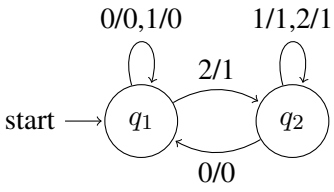


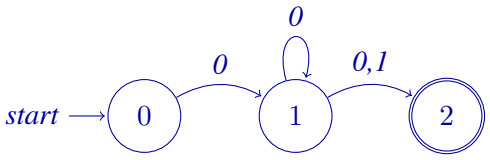
Figure 5: FST T_1 (book, problem 1.24).

Problems:

1. Find an NFA that accepts the language $L(00^*(0 \cup 1))$ (i.e., the language the given RE defines).

   Answer:

   One brief solution is the following NFA (the loop back can be at state 0, too):



2. In tutorial 1, we discussed the DFA that determines whether a *binary string* is divisible by three or not. The state diagram is depicted below. Convert it to a corresponding regular expression.

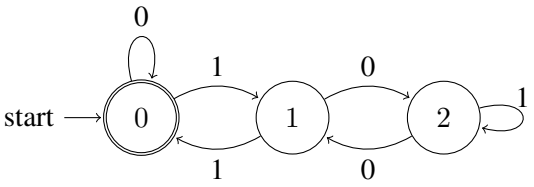


Figure 6: DFA to check whether a binary number is divisible by 3.

   Answer:

   Let's work with the corresponding GNFA and start removing states. First, remove state 2 and compensate that with an arrow 01^*0 from state 1 back to itself. Then, remove state 1 and compensate that with an arrow $1(01^*0)^*1$ from state 0 back to itself. These yield a general regular expression (alternate forms):

$$R = (0 \cup 1(01^*0)^*1)^*$$
$$R = (0^*(1(01^*0)^*1)^*)^*$$

3. Consider the ternary number system (i.e., base 3, where the digits are $0, 1, 2$).

   a) Draw a state diagram for a DFA that accepts ternary strings that are *not* divisible by three.

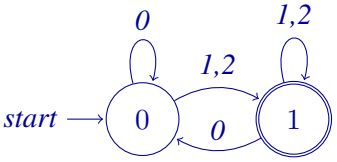
   b) What is the corresponding regular expression?

   c) Draw a state diagram for a DFA that accepts ternary strings that are *not* divisible by nine.

   d) What is the corresponding regular expression?

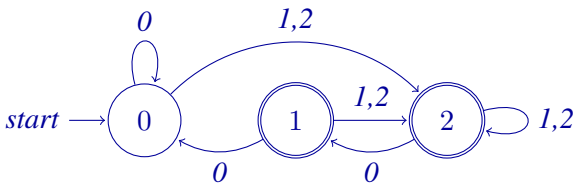
   Answer:

   a) A ternary number is divisable by three if the last digit (if any) is zero.



b) Strings that end with 1 *or* 2*:* $R = \Sigma^*(1 \cup 2)$

c) A ternary number is divisable by 9 iff last two digits are zero:



d) $\Sigma^*(1 \cup 2)$ *matches the strings which end with non-zero digit, and* $\Sigma^*(1 \cup 2)\Sigma$ *matches the strings which second last digit is non-zero. Our language accepts both, so*

$$(\Sigma^*(1 \cup 2)) \cup (\Sigma^*(1 \cup 2)\Sigma) = \Sigma^*(1 \cup 2))(\epsilon \cup \Sigma)$$

4. Let $w = x_k \dots x_0$ be a string of ternary digits (MSB first on the left).

   a) Draw a state diagram for a DFA that accepts strings that are divisible by two.

   b) Convert the given DFA to a regular expression.

   c) Let $n_i(x)$ denote the number of digits in some number x that have value i. For example, for $x = 1120$, corresponding to the decimal number $2 \cdot 3 + 3^2 + 3^3 = 42$, we have $n_0(x) = 1$, $n_1(x) = 2$ and $n_2(x) = 1$. Analyze the DFA and determine a simple rule based on the counts $n_i(x)$ for the divisibility by two. That is, suppose you only know the triple (n_0, n_1, n_2), can you determine whatever the unknown number x can be divided by two?

Answer:

a) Let n_i *denote the number after reading* i *symbols. Clearly,*

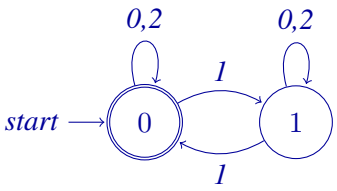
$$n_{i+1} = 3n_i + x,$$

where x *is the last symbol read. Now we are asked about divisability by two, so we need to check if* $n \mod 2 = 0$*. We can take "* $\mod 2$*" at every step, so each* $\tilde{n}_i$ *is either* 0 *or* 1*, i.e., we need two states, and the update rule is*

$$\tilde{n}_{i+1} = (3\tilde{n}_i + x) \mod 2,$$

which gives

		$n_i \mod 2$	
x		0	1
$(0,2)$	0	0	1
(1)	1	1	0



b) Hence, we are checking whether a string has an even number of 1*:s. So for example,*

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

c) The term corresponding to the i*th digit,* $i = 0, 1, \dots$*, is* $a_i 3^i$*. Taking modulo 2 we notice that*

$$a_i 3^i \mod 2 = \begin{cases} 0, & \text{if } a_i = 0 \text{ or } a_i = 2 \\ 1, & \text{if } a_i = 1 \end{cases}$$

Hence, digits 0 *and* 2 *in ternary presentation are irrelevant for divisability by* 2*. Instead, as long as there are an even number of* 1 *digits, the number is divisable by* 2*.*
Now, given we know only the counts $\{n_0, n_1, n_2\}$*, the criteria for divisability by two is whether* n_1 *is even or odd. This can be deduced also directly from the DFA depicted above.*

Problems:

1. Prove that the following languages are not regular

   a) $\{0^{n+m}1^m \mid n \geq 0, m \geq 0\}$

   b) $\{0^n1^m0^n \mid n \geq 0, m \geq 0\}$

   c) $\{www \mid w \in \{0,1\}^*\}$

   Answer:

   These three cases can be proven with the pumping lemma. **A complete proof** *starts by stating an assumption that A is regular, and therefore Pumping Lemma holds, and let p denote the corresponding pumping length. Then you choose a suitable s with $|s| \geq p$, which can be split into xyz so that the 3 conditions of the Lemma are valid. Then by pumping down (removing y) or up (adding more y:s) one obtains a string that does not belong to A even though it was suppose to. This contradiction then means that A is not regular.*

   Below we just briefly give examples how an appropriate string s can be chosen.

   a) Suppose the language is regular, let p be the pumping length, and choose $s = 0^p1^p$. Pumping Lemma says $s = xyz$ such that $|xy| < p$ and $|y| > 0$, i.e., y is a string of one or more zeroes. Pumping down gives $s_2 = xz$, which thus has less zeroes than ones, and $s_2 \notin A$. Contradiction and A is not regular.

   b) Here we choose $s = 0^p10^p$, and pumping up y (or down) yields a contradiction.

   c) Here we can choose $s = 0^p10^p10^p1$, i.e., the substring w is 0^p1. This produces a contradiction when pumped up.

2. Let $\Sigma = \{0, 1, +, =\}$ and define a language A as follows:

   $$A = \{x = y + z \mid x, y, z \text{ are binary integers, and } x \text{ is the sum of } y \text{ and } z\}.$$

   Show that A is not regular.

   Answer:

   Suppose A is regular and pumping lemma thus holds. Let p denote the pumping length and choose $s =$"$\mathbf{1^p = 0 + 1^p}$" (hyphens are for clarity), where 1^p is a regular expression and expands to a string of p 1:s. The expression is mathematically true and hence $s \in A$. As $|s| \geq p$, the three conditions of Pumping Lemma hold. First, $s = xyz$ where y is a non-empty string of 1:s (cf. conditions (ii) and (iii)). Now pumping up or down, with xy^iz, adds or removes 1-bits from the left-hand side, making the mathematical equation false, and thus the corresponding strings no longer belong to A. This is a contradiction and hence A must be nonregular.

3. Let $F = \{a^ib^jc^k \mid i, j, k \geq 0, \text{ and if } i = 1 \text{ then } j = k\}$.

   a) Show that F is not regular.

   b) Show that F acts like a regular language in the pumping lemma: Give a pumping length p and demonstrate that F satisfies the three conditions of the lemma for this p.

   c) Explain why parts a) and b) do not contradict the pumping lemma

   Answer:

a) **Show that F is not regular:**

As in b) we are asked to show that F satisfies the pumping lemma, we choose to not take that avenue. *Here the essential part is the case when $i = 1$, when the extra condition (equality) on j and k is imposed to. As a counter assumption, suppose that F is regular. In addition to pumping lemma, this means that*

- *DFA, NFA and RE for it exists*
- *as the class of regular languages is closed under the regular operations, union, intersection and star, applying such to F should yield a regular language.*

The latter sounds promising, so let's proceed with it and define a new language

$$X = \{ab^j c^k \mid j, k \geq 0\},$$

i.e. a language of all strings with a single 'a' followed by some number of b:s and then some number of c:s (both can be omitted too). Clearly, X is regular as RE $R = ab^*c^*$ defines it. However,

$$Y = X \cap F = \{ab^j c^j \mid j \geq 0\},$$

which is known to be nonregular. As X is regular, then F must be nonregular.

b) Show that F acts like a regular language in the pumping lemma:

Let us consider first different cases for i when splitting the string from F:

- $i = 0$*: no a:s, so let's choose*
  $x = \epsilon$
  $y =$ *"the next symbol"*
  $z =$ *"the rest"*
- $i = 1$*: one 'a', let's get rid of it!*
  $x = \epsilon$
  $y = a$ $z =$ *"the rest"*
- $i \geq 2$*: two or more a:s, let's preserve at least two of them!*
  $x = aa$
  $y =$ *"the next symbol"*
  $z =$ *"the rest"*

In each case the string resulting from pumping (up or down) never has a single a. Moreover, string is always of form $a^i b^j c^k$, and thus belongs to F. Minimum pumping length is $p = 3$ as in the last case the string must have at least 3 symbols.

c) Pumping Lemma is valid for any regular language. It says nothing about being valid or not if the language is nonregular. (even though in many cases, it does trigger a contradiction, making it a useful result).

4. Let $\Sigma = \{a, b\}$ and define

$$C_k = \Sigma^* a \Sigma^{k-1}, \quad k = 1, 2, \ldots$$

i.e., C_k consists of strings where the kth last symbol is a.

a) Describe an NFA with $k + 1$ states that recognizes C_n in terms of both a state diagram and a formal description.

b) Prove that for each k no DFA with fewer than 2^k states can recognize C_k

Answer:

a) The state diagram is depicted in Figure 7 (formal description straightforward and omitted).

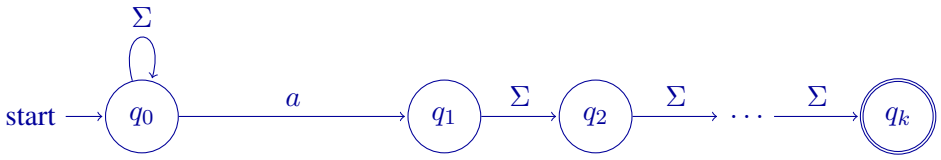


Figure 7: NFA for checking if the kth symbol from the end of the string was a or not.

b) At no point, the DFA may not know how many more symbols will be read before the string ends. Consequently, it needs to keep track of the last k symbols read, whether they would be "ok" or "not ok". The only memory DFA has is in its states, and as the number of possible configurations for the last k symbols is 2^k, we need 2^k states to represent them all. Initial state will be the one where all "last" symbols were not ok. This corresponds to a shift register.

Problems:

1. Recall the CFG G4 from Example 2.4. For convenience, let's rename its variables with single letters as follows.

$$E \to E + T \mid T$$
$$T \to T \times F \mid F$$
$$F \to (E) \mid a$$

Give parse trees and derivations for the following strings:

a) a
b) $a + a$
c) $a + a + a$
d) $((a))$

Answer:

a) Derivation for a:

$$E \to T \to F \to a$$

Parse tree looks exactly the same (in vertical), and omitted.

b) Derivation for $a + a$:

$$E \to E + T \to T + T \to F + T \to a + T \to a + F \to a + a.$$

Parse tree looks exactly the same (in vertical), and omitted.

c) Derivation for $a + a + a$ repeats the same pattern:

$$E \to E + T \to E + T + T \to T + T + T \to \ldots$$

(not leftmost derivation)

d) The deriation for $((a))$ is

$$E \to T \to F \to (E) \to \ldots \to (F) \to ((E)) \to \ldots \to ((F)) \to ((a)).$$

Parse tree is exactly the same linear sequence.

2. Convert the following CFG into an equivalent CFG in Chomsky normal form, using the procedure given in Theorem 2.9.

$$A \to BAB \mid B \mid \epsilon$$
$$B \to 00 \mid \epsilon$$

Answer:

We proceed as requested:

(a) New start variable:

$$S_0 \to A$$

(b) Remove ϵ-rules:
First, rule $A \to \epsilon$, which is compensated with the following:

$$A \to BB$$

Then, rule $B \to \epsilon$: add the following

$$A \to AB$$
$$A \to BA$$

(Note, no point to add rule $A \to A$)

(c) Remove unit rules:
First, removal of $A \to B$ is compensated by adding

$$A \to 00$$

and $S_0 \to A$ is compensated by adding all "variable A" rules also for S_0, so our complete set of rules at this stage is

$$S_0, A \to BAB|BB|AB|BA|00$$
$$B \to 00$$

(d) Finally we fix the "long rules":
First, $S_0, A \to BAB$ becomes $S_0, A \to BC$ and $C \to AB$.
Then $S_0, A, B \to 00$ are replaced by $S_0, A, B \to ZZ$ and $Z \to 0$.

3. Let $F = \{a^i b^j c^k \mid i, j, k \geq 0$, and if $i = 1$ then $j = k\}$. We know that F is not a regular language (see Homework 5, Problem 3). Show that F is a context-free language.

Answer:

We have three options: i) construct a CFG, ii) construct a PDA, or iii) utilize the fact that the class of CFLs is closed under regular operations.

In this case, the last option seems tempting. Thus, we define

$$F_1 = \{a^0 b^j c^k \mid j, k \geq 0\}$$
$$F_2 = \{ab^j c^j \mid j \geq 0\}$$
$$F_3 = \{a^i b^j c^k \mid i \geq 2 \text{ and } j, k \geq 0\}$$

Now F_1 and F_3 are regular languages and thus also CFLs. Similarly, F_2 is known to be a CFL. Therefore their union, which is equal to F, is also a CFL.

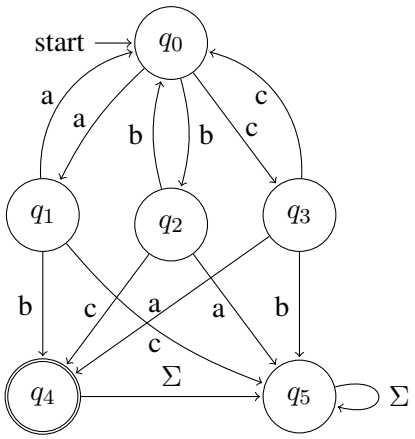
4. Let $A/B = \{w \mid wx \in A$ for some $x \in B\}$. Show that if A is context free and B is regular, then A/B is context free.
NOTE: This problem is postponed to next week, to be solved using the PDAs! Please SKIP!

Answer:

5. Consider again the paper-scissors-stone game, with the DFA depicted below.

   a) Describe a corresponding CFG.

   b) Draw a parse tree for game (string) *bbbc*.



Answer:

This is easiest probably by starting from the corresponding Regexp,

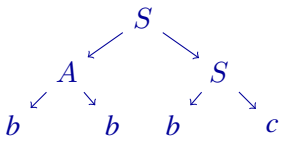
$$(aa, bb, cc)^*(ab, bc, ca)$$

a) Hence, one CFG generating the language is as follows:

$$S \to ab|bc|ca|AS$$
$$A \to aa|bb|cc$$

where A corresponds to a tie round.

b) A parse tree for game (string) bbbc:



Problems:

1. Let $\Sigma = \{0, 1\}$ and consider the following languages:

   a) $\{1011 \mid n \geq 0\}$
   b) $\{1^n011^n \mid n \geq 0\}$
   c) $\{10^n11 \mid n \geq 0\}$
   d) $\{(1011)^n \mid n \geq 0\}$

   For each, first **show** if the given language is regular or not. For nonregular languages, try to give a CFG or a PDA, either of which would imply that the given language is still context free.

   Answer:

   a) This is regular, e.g., RE $R = 1011$
   b) This is nonregular. Assume that A is regular so that Pumping Lemma should hold and let p denote the corresponding pumping length . . . choose $s = 1^n011^n$ so that $s \in A$ and $|s| \geq p$ . . . pumping up/down adds/removes 1:s from the start of s, and thus the resulting string is not in A; contradiction and thus A must be nonregular.
   c) Regular as generated by RE $R = 10^*11$
   d) Similarly, regular and generated by $R = (1011)^*$

2. Let $\Sigma = \{a, b, c\}$. We know that

   - $\{w \mid w \in \{a, b\}^*\}$ is regular as it can be recognized by a FA
   - $\{ww^R \mid w \in \{a, b\}^*\}$ is context free as it can be recognized by a PDA
   - $\{ww \mid w \in \{a, b\}^*\}$ is not context free as it cannot be recognized by a PDA

   At the same time, we recall that

   - NFAs were generalized to PDAs by adding one stack.
   - Maybe PDAs in turn can be generalized by adding more stacks?

   Suppose we have two stacks, so that the transitions in the state diagram are indicated with

   $$b, (c_1, c_2) \rightarrow (e_1, e_2),$$

   where b is the input symbol read, (c_1, c_2) denote the symbols at the top of the stack 1 and 2, respectively, and (e_1, e_2) are symbols pushed back to the corresponding stacks. As before, any of the (b, c_1, c_2, e_1, e_2) can be ϵ indicating that the corresponding parameter/operation is omitted. Draw a state diagram for two-stack PDA that recognizes $\{ww \mid w \in \{a, b\}^*\}$.

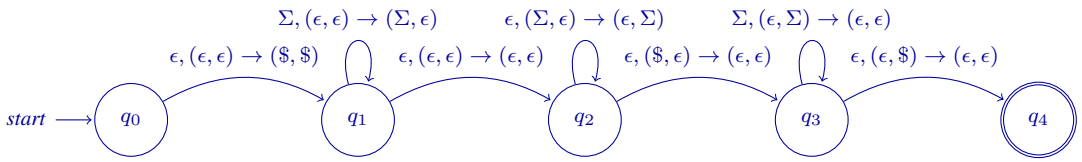
   Note: consequently, you show that the second stack improves the computational capacity!

   Answer:

   High level idea:

   - *First store the first string to stack #1*
   - *Then guess the end, and transfer the contents of stack #1 to stack #2, thereby reversing the order (now first symbol read is at the top of the stack)*
   - *Compare the contents of stack #2 to the remaining input*

The corresponding PDA is as follows:



3. Let $A/B = \{w \mid wx \in A \text{ for some } x \in B\}$. Show that if A is context free and B is regular, then A/B is context free.

   Answer:

   Let P a PDA for A, and M be a DFA for B. We show that A/B is a CFL by constructing PDA P' that recognizes it.

   PDA P' is a modified version of P that will run simultaneously the finite state control of DFA M's in addition to its own finite state control (cf. the cartesian product). Initially, only the original logic of P is run and M waits at the starting state.

   After w has been read, P' continues also with M. From now on, P' guesses *string x by using nondeterminism, and processes it by both M and P. The machine will* accept *only if both machines simultaneously* accept.

4. If A and B are languages, define $A \diamond B = \{xy \mid x \in A \text{ and } y \in B \text{ and } |x| = |y|\}$. Show that if A and B are regular languages, then $A \diamond B$ is a CFL.

   Answer:

   We have NFA N_1 for A and another NFA N_2 for B. We need to construct a PDA, so we are free to define how stack is utilized. The corresponding PDA is constructed as follows:

   - *Start state is the start state of N_1*
   - *Modify all transitions of N_1 so that they also add 1 to stack (count up)*
   - *Add ϵ transitions from every accept state of N_1 to start state of N_2*
   - *Modify all transitions of N_2 so that they* consume *one 1 from the stack. (if stack empties, transitions will not take place).*
   - *Add a new accept state q_a and $\epsilon, \$ \to \epsilon$ transitions from every accept state of N_2 to q_a.*

   We have constructed a PDA for $A \diamond B$, and therefore it is a CFL.

5. Let $\Sigma = \{0, 1\}$ and let B be the set of strings that contain at least one 1 in their second half, i.e., $B = \{uv | u \in \Sigma^*, v \in \Sigma^* 1 \Sigma^*, \text{ and } |u| \geq |v|\}$.

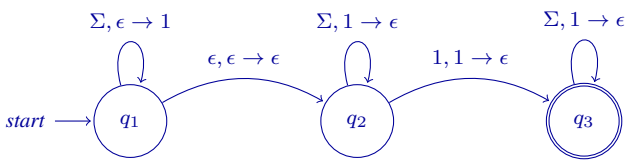
   a) Give a PDA that recognizes B

   b) Give a CFG that generates B

   Hint: utilize the nondeterminism to "find out" where the half-way is.

   Answer:

   a) **PDA:**



b) **CFG:**

Acceptable strings can be divided into 4 parts:

(1 or more symbols) (n symbols) (tagged 1) (n symbols)

where $n \geq 0$. This ensures that the tagged 1 is on the right half. The corresponding CFG is

$$S \to AB$$
$$A \to AA|0|1$$
$$B \to CBC|1$$
$$C \to 0|1$$

Here, A generates the first part and B corresponds to the other 3 parts. B again expands into C^n1C^n, where each C is any symbol.

Problems:

1. Give the sequence of configurations M_2 of Example 3.7 enters when started on $w = 000$.

   Answer:

   0. q_1000
   1. $\sqcup q_200$
   2. $\sqcup xq_30$
   3. $\sqcup x0q_4\sqcup$
   4. $\sqcup x0\sqcup q_{reject}$

2. Let $\Sigma = \{a, b, c\}$ corresponding to three candidates in an election. Give an implementation level description of a TM that recognizes if candidate a wins, i.e., she/he gets more votes than other candidates. Is your TM a decider?

   Answer:

   For example, simply as follows:

   $C = C(w)$:

   (a) Sweep from left to right, and cross out the **first instance** *of each symbol (in the first round, also mark the start of the tape)*
   (b) If the sweep had only a:s, then halt on `accept`
   (c) If the sweep had other symbols but no a:s, then halt on `reject`
   (d) Go back to start (i.e., sweep found a and some other symbol)

   (every other sweep can be also from right to left . . .)

3. Let $\Sigma = \{0, 1, \#\}$. As a simple integrity check, the files in a file system are stored three times so that a file w, consisting of $\{0, 1\}$, is stored as $w\#w^R\#w$. Let A denote the corresponding language, i.e., files (strings) that satisfy the elementary integrity check. Give an implementation level description of a TM that decides on A.

   Answer:

   $M = M(w)$:

   (a) **Init:** *Mark the start of the tape ($c \to \bar{c}$), and sweep from left to right to check that string has exactly two #:s; If not, Halt on* `reject`*. Otherwise move tape head back to start.*
   (b) **Repeat:**
      i. If the symbol is #, then **break** *to final check*
      ii. Otherwise, cross it off with x (and keep the value in the current state)
      iii. Scan right over the two #:s, and continue over the x:s (if any)
      iv. If the symbol is different, then Halt on `reject`
      v. Otherwise, cross it off with x, scan left over one #, and the following x:s (if any)
      vi. If the symbol is different, then Halt on `reject`

vii. *Otherwise, cross it off with x, scan left over one $\#$, and continue until x*

viii. *Move right (the next symbol to be verified)*

(c) **Final check:**

- *Sweep from left to right to check that all symbols have been crossed off*
- *Halt on* `reject` *if not; Otherwise Halt on* `accept`

4. Show that the language $\{a^n b^n c^n \mid n \geq 0\}$ is Turing recognizable. Is it also Turing-decidable?

   Answer:

   $M = M(w)$:

   (a) *While a, repeat the following steps:*

   i. *Write $\bar{a}$ over the a*

   ii. *Scan right over possible a:s, and then over possible x:s*

   iii. *Halt on* `reject` *if not b; Otherwise cross it off with an x*

   iv. *Scan right over possible b:s, and then over possible x:s*

   v. *Halt on* `reject` *if not c; Otherwise cross it off with an x*

   vi. *Scan left until $\bar{a}$ is found, and then move one back right*

   (b) *Scan right over the possible x:s*

   (c) *Halt on* `reject` *if not blank ($\sqcup$); Otherwise Halt on* `accept`

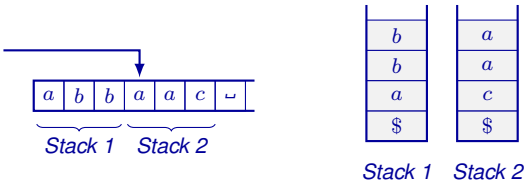
   This machine is a decider, and therefore the language is Turing-decidable. Other approach would be to first check that the input string complies with the format $a^*b^*c^*$, and then start crossing them off (without worrying about the order anymore).

5. Show that an arbitrary TM can be simulated with a PDA with two stacks.

   Answer:

   The basic idea is to utilize the stacks as follows:

   - *Stack 1: contents on tape left of the tape head*
   - *Stack 2: contents on tape right of the tape head, including the current position*



   Moreover, both stacks have been initilized so that $ marks the bottom of the stack.

   Initially, the input string is thus in Stack 2.

   Before every step: If $ at the top of Stack 2, push $\sqcup$ there

   Movement left:

   - *If $ at the top of Stack 1, ignore*
   - *Otherwise, pop one symbol from Stack 1 and push it to Stack 2*

   Movement right: Pop one symbol from Stack 2 and push it to Stack 1

   Reading & Writing: In stack 2, with pop & push

Problems:

1. Let A_k denote the language of Rice encoded strings with parameter $k > 0$, i.e., strings of form

$$\underbrace{b_0 b_1 \ldots b_{k-1}}_{k \text{ LSBs}} \underbrace{11\ldots1}_{m \text{ times}} 0,$$

where k is some fixed positive integer number, e.g., $k = 3$, and $b_0, b_1, \ldots, b_{k-1} \in \{0, 1\}$ and $m \geq 0$ depend on the number x that was encoded (the b_i define lowest k bits; the possible 1 bits define the higher bits if x was exceptionally large).

   a) Is the language A_3 (i) finite, (ii) infinite but countable, or (iii) uncountable?

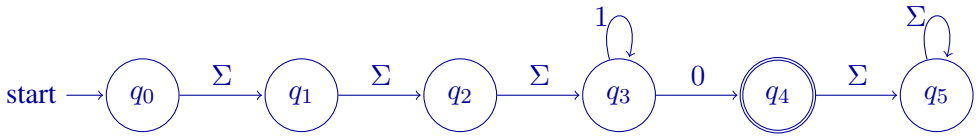
   b) Design a DFA M that recognizes language A_3

   c) Is A_3 decidable?

   Answer:

   a) Language A_3 is infinitely countable: all languages are countable, and here m has no upper bound.

   b) We give a state diagram:



   c) Yes, it's a regular language (which are decidable).

2. Let $\mathbb{B}$ be the set of all infinite sequences over $\{0, 1\}$. Show that $\mathbb{B}$ is uncountable using a proof by diagonalization.

   Answer:

   Proof by contradiction. Suppose $\mathbb{B}$ is countable and let $S = \{B_1, B_2, B_3, \ldots\}$ denote a sequence of $B_i \in \mathbb{B}$ that includes all infinite sequences, $S = \mathbf{B}$. Define $\hat{B}$ such that the ith bit is the opposite of the ith bit in B_i:

$$\hat{b}_i = 1 - b_{i,i} = b_{i,i} \oplus 1.$$

   Now $\hat{B}$ is an infinite binary sequence, i.e., $\hat{B} \mathbb{B}$, but by construction, $\hat{B}$ does not appear in the sequence S. This is a contradiction, and $\mathbb{B}$ is thus uncountable.

   Visually, it is instructive to consider the infinite matrix:

$$\begin{array}{c} \\ B_1: \\ B_2: \\ B_3: \\ \\ \end{array} \begin{array}{c} \begin{array}{cccc} 1 & 2 & 3 & \\ \end{array} \\ \begin{bmatrix} b_{1,1} & b_{1,2} & b_{1,3} & \cdots \\ b_{2,1} & b_{2,2} & b_{2,3} & \cdots \\ b_{3,1} & b_{3,2} & b_{3,3} & \cdots \\ \vdots & & & \end{bmatrix} \end{array}$$

   flip the bits on the diagonal!

3. Let $T_k = \{(n_1, \ldots, n_k) \mid n_1, \ldots, n_k \in \mathbb{N}\}$. Show that T_k is countable for all $k = 1, 2, \ldots$.

   Answer:

   Two example answers:

   (a) **Using primes:** *The set of primes is countably infinite, $\mathbb{P} = \{2, 3, 5, \ldots\}$. Let p_i denote the ith prime, and define*

   $$f(n_1, \ldots, n_k) = p_1^{n_1} \cdots p_k^{n_k}.$$

   This function is **an injection** $\mathbb{N}^k \to \mathbb{N}$*, and thus T_k is countable. (integer numbers have an unique factorization).*

   Remark: f is not a bijection, i.e., mapping is "sparse" – we skip numbers divisable by some higher prime p_i with $i > k$.

   (b) **By induction:** *Case $k = 1$ is trivial with $f_1(n) = n$. Let $f_2(n, m) : \mathbb{N}^2 \to \mathbb{N}$ be* **a bijection** *(or injection). $f_2(n, m)$ can be easily constructed (see the tutorial), and case $k = 2$ is covered.*

   Induction hypothesis: *Suppose that the claim holds for $1, 2, \ldots, k - 1$.*

   Consider the case $k > 2$. Define

   $$f_k(n_1, \ldots, n_k) \triangleq= f_{k_1}(n_1, \ldots, n_{k-2}, f_2(n_{k-1}, n_k)).$$

   This, according to the induction hypothesis, provides a necessary bijection (injection) from $\mathbb{N}^k \to \mathbb{N}$, and therefore also T_k is countable. By induction, T_k is countable for any k.

4. Let $A = \{\langle R, S\rangle \mid R$ and S are regular expressions and $L(R) \subseteq L(S)\}$. Show that A is decidable.

   Answer:

   So we need to construct a decider for A.

   $D = M(\langle R, S\rangle)$:

   (a) Halt on reject if R and S are not valid regular expressions
   (b) Construct a DFA M_1 for $L(R)$
   (c) Construct a DFA M_2 for $L(R) \cap L(S)$
   (d) Invoke a decider for EQ_{DFA} with input $\langle M_1, M_2\rangle$;
   and halt accordingly

   This provides the requested decider because if $L(R) \subseteq L(S)$ then $L(R) \cap L(S) = L(R)$.

Problems:

1. Let us consider the set $\mathcal{M}_n$ of Turing machines with alphabet Σ having $m = |\Sigma|$ symbols, and n non-halting states, i.e., with the accept and reject states, machines in $\mathcal{T}_n$ have $n + 2$ states. What is the size of the set $\mathcal{M}_n$, i.e., the number of Turing machines with n states?

   Answer:

   Suppose states are labeled $1, 2, \ldots, n + 2$ and we start from state 1. (we exclude starting from halting states, as those are trivial special cases for which no TM is needed).

   Moreover, we let the tape alphabet has k symbols, whereas the input alphabet Σ has m symbols.

   Next we focus on the number of possible ways to define the transition function

   $$\delta : Q \times \Gamma \to Q \times \Gamma \times \{L, R\}$$

   For each TM, we need to decide on action in nk "state-input" pairs (action are immaterial for the halting states). Decision must be one of the $2(n + 2)k$ possible actions. Hence, we have

   $$(2(n + 2)k)^{nk}$$

   essentially different Turing machines.

2. Let A denote the set of even numbers $2, 4, 6, \ldots$ and B the set of odd numbers $1, 3, 5, \ldots$. Show that A is mapping reducible from B by giving a computable function that provides the needed reduction.

   Answer:

   Let $f(n) = n + 1$, which provides the mapping reduction.

3. Suppose you have a TM with three tapes.

   a) Tape 1 contains a binary number x (LSB first), and tape 2 a binary number y (LSB first), and tape 3 is initially blank. Construct a TM that computes the sum $x + y$ on Tape 3 and halts on *accept*. (implementation level description)

   b) Suppose next that the numbers are unary numbers, i.e., on tape 1 there are x 1s and then a blank, and tape 2 has y 1s and then a blank. Construct a TM that computes the sum of x and y on Tape 3.

   Answer:

   We need to implement carry:

Tape 1	*Tape 2*	*Carry*	*Tape 3*	*Carry*
0	*0*	*0*	*0*	*0*
1	*0*	*0*	*1*	*0*
0	*1*	*0*	*1*	*0*
1	*1*	*0*	*0*	*1*
0	*0*	*1*	*1*	*0*
1	*0*	*1*	*0*	*1*
0	*1*	*1*	*0*	*1*
1	*1*	*1*	*1*	*1*

   and scan from left to right until blanks on both Tape 1 and 2. Finish with writing the carry to Tape 3, if set.

   With unary numbers just copy the two strings to Tape 3.

4. Show that EQ_{CFG} is undecidable.

   Answer:

   Define CFG $G0 = (V, \Sigma, R, S)$, where $V = \{S\}$ and S is the starting variable. For each terminal $x \in \Sigma$, the CFG $G0$ has a rule $S \to xS$. $G0$ includes also the rule $S \to \epsilon$. Hence, $L(G0) = \Sigma^*$.

   Let R be a TM that decides EQ_{CFG}. We shall use it to construct TM S to decide ALL_{CFG}.

   $S = S(\langle G \rangle)$, where G is a CFG:

   (a) Run R on input $\langle G, G0 \rangle$
   (b) If R accepts, `accept`; Otherwise `reject`.

   In stage 1, TM R determines if $L(G) = L(G0)$. As the latter is Σ^*, we have a decider for ALL_{CFG}, which is undecidable. Contradiction, and therefore R cannot exist and EQ_{CFG} must be undecidable.

5. Let

$$T = \{\langle M \rangle \mid M \text{ is a TM that accepts } w^R \text{ whenever it accepts } w\}.$$

   Show that T is undecidable.

   Answer:

   Proof by contradiction. Let TM S be a decider for T. We will show that A_{TM} reduces to T, thus denying the existance of S.

   $A = \langle M, w \rangle$, where M is a TM and w a string:

   (a) Check that input is valid; `reject` if not.
   (b) Construct the following TM M_2 from M and w
      $M_2 = M_2(\langle x \rangle)$:
      i. If $x \in L(00^*11^*)$, then `accept`
      ii. Otherwise, run M on input w
      `accept` if accept, otherwise reject
   (c) Run S with input $\langle M_2 \rangle$
   (d) `accept` if accept; otherwise `reject`

   Note:

   - *If $w \in L(00^*11^*)$, then $w^R \notin L(00^*11^*)$*
   - *Similarly, for language of all strings: $w \in L(\Sigma^*)$ means that $w^R \in L(\Sigma^*)$*

   Two cases:

   *(a) M **accepts** w: In this case, M_2 accepts any string, $\langle M_2 \rangle \in T$ and S accepts it, and TM A `accepts`*
   *(b) M **does not accept** w: In this case, M_2 accepts only strings from $L(00^*11^*)$. Consequently, $\langle M_2 \rangle \notin T$ and S rejects it ... and TM A `rejects`*

   This means A would decide on A_{TM}, contradiction!

Problems:

1. Let $f(n) = 7n^2 + 2n \log_2 n + e^{-n}$. Which of the following are true?

   a) $f(n) = o(n \log_2 n)$
   b) $f(n) = o(2^n)$
   c) $f(n) = o(n^4)$
   d) $f(n) = O(n \log_2 n^n)$
   e) $f(n) = O(2^n + 1)$

   Answer:

   As n increases, the first term in $f(n)$ turns out the be the important one. (e.g., $n \log n$ grows slower than n^2, etc.).

   a) ✗ *($f(n)/(n \log_2 n) \to \infty$ as n increases)*
   b) ✓
   c) ✓
   d) ✓
   e) ✓

2. Which of the following pairs of numbers are relatively prime? Show the calculations that led to your conclusions.

   a) 1234 and 431
   b) 444 and 123

   Answer:

   Idea would be to use the Euclidean algorithm to determine the gcd. It turns out that gcd$(1234, 431) = 1$ and gcd$(444, 123) = 3$. Therefore, the former are, and the latter are not.

3. Let CONNECTED $= \{\langle G \rangle \mid G$ is a connected undirected graph$\}$. Analyze the algorithm given on page 186 to show that this language is in P.

   Answer:

   - **Step 1:** *Takes $O(1)$ time.*
   - **Step 2:** *This step may repeat at most $n - 1$ times.*
   - **Step 3:** *Each run of step 3 checks all nodes, then checks all of its neighbors. In the worst case, a node can have $n-1$ neighbors. The running time of step 3 is $O(n \cdot (n-1)) = O(n^2)$.*
   - **Step 4:** *Scanning all nodes takes $O(n)$ time.*

   The greatest factor is steps 2-3. Since step 3 is run n times in the worst case, the total running time is $O(n^3)$, and therefore CONNECTED is in P.

4. Let $G = (V, E)$ be a directional graph where nodes correspond to bus stops and an edge $v_i \to v_j$ indicates that node v_j can be reached from node v_i within *"a reasonable time"*. Suppose $\{(s_1, d_1), \ldots, (s_n, d_n)$ is a set of n transportation requests where s_i is the origin and d_i the destination node. We have a single vehicle with unit capacity, i.e., it can complete a single request at time (cf. taxi rides). Initially the vehicle is at node v_0, the graph G is connected, each edge takes unit time to travel and a feasible solution satisfies the n requests in time $4n$.

a) Devise a brute-force algorithm that determines if a feasible solution exists.
b) Devise a more efficient algorithm and analyze it.
c) Based on your algorithm, is this problem in P and/or NP?

Answer:

The problem is about a single vehicle trying to fulfill n ride requests. (There are numerous acceptable answers, so consider this as an example only!)

a) **(Naïve) Brute force algorithm:**

For every n-tuple of $\mathbf{x} = (x_1, \ldots, x_n)$, where $x_i \in \{1, \ldots, n\}$

(a) Check if the route defined by $\mathbf{x}$ includes all requests ($1, \ldots, n$ exist in it)

(b) Compute the length of route $(v_0, s_{x_1}, d_{x_1}, \ldots, s_{x_n}, d_{x_n})$

(c) If length is less than $4n$, accept

Reject (as no path was feasible)

b) **More efficient algorithm:** *The brute force algorithm leaves a lot for improvement. For example, instead of n-tuples (n^n), we could check only the valid permutations ($n!$).*

However, we can do even better:

(a) Invoke (e.g.) Dijktra's algorithm for G (for all pairs)
Set $C = 4n - \sum_i \ell(s_i, d_i)$. If $C < 0$, reject.
Construct a new graph $G' = (V', E')$, with $n + 1$ vertices:
- Vertex 0 represents v_0, and vertices $1, \ldots, n$ represent rides (s_i, d_i).
- Edges $e_{0,i}$ have weights $\ell(v_0, s_i)$
- Edges $e_{i,j}$ have weights $\ell(d_i, s_j)$

(b) Remaining problem: check if hamiltonian path from 0 exists with length at most C

In practice, you may implement something like this:

Iterate$(i, n, \boldsymbol{\pi}, t)$*:*

If $t > t_{\max}$*:*
- **Return** `Reject` *(prune search tree)*

Else If $i = n$ **Or Iterate**$(i + 1, n, \boldsymbol{\pi}, t + \ell(\pi_i, \pi_{i+1}))$
- *Return* `Accept`

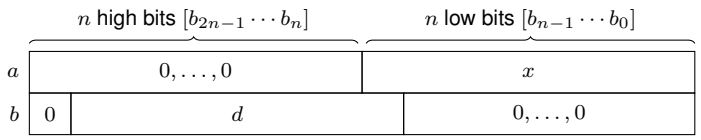
For $j = i + 2, \ldots, n$*:*
- **Swap**(π_{i+1}, π_j)
- **If Iterate**$(i + 1, n, \boldsymbol{\pi}, t + \ell(\pi_i, \pi_{i+1}))$
  - **Return** `Accept`
- **Swap**(π_{i+1}, π_j)

CheckRides$(G, \{s_i, d_i\})$*:*

Construct G' as in above: n rides, $n + 1$ vertices (G' is a complete graph)
Setup $\ell(i, j)$: the travel times from i to j (matrix for all pairs / compute as needed)
Return Iterate$(0, n, (0, \ldots, n), 0)$ *(the first node 0 is fixed, permutate others)*

c) **In class P?:** *This is in NP, as a solution can be verified. For P, we need a better algorithm!*

5. Given two binary numbers x and d, both of the same length n, and $d > 0$, an algorithm to divide x by d uses two registers a and b of length $2n$ initialized as follows.



Then the following two steps are repeated n times to get the result:

i) If $a \geq b$: $a \leftarrow a - b$ and $c \leftarrow 1$; Else $c \leftarrow 0$ (c is carry)
ii) $a \leftarrow 2 \cdot a + c$ (shift bits to left, with carry in)

Note that:

- Step i) is a comparison and a possible subtract operation on bits $[b_{2n-1} \cdots b_{n-2}]$
- Step ii) is a simple bit shift left operation (with carry in)

At the end, the high bits of a contain the remainder r, and the low bits the quotient q, $x = qd + r$.

a) Analyze the time complexity of the above algorithm (polynomial time or not?).
b) A naïve algorithm would repeatedly subtract d from x until $x < d$ and count the number of rounds. What is the time complexity of this algorithm?

Answer:

a) Step i) has a comparison and a possible subtraction of n bits: $O(n)$
Step ii) has a shift left (with carry) operation for $2n$ bits: $O(n)$

Steps i) and ii) are repeated n times, so the running time of the algorithm is $O(n^2)$, which is a polynomial time, and therefore the algorithm is in P.

b) The worst case of this algorithm is when x is all 1s (n times) and d is 1. Then the algorithm would apply $2^n - 1$ subtractions on x, each $O(n)$, so that the total running time is horrible $O(n2^n)$.

Problems:

1. You have implemented an algorithm using a 3-tape Turing machine X that runs in time $O(n^2)$. What is the time complexity when X is simulated with a single-tape Turing machine?

   Answer:

   Theorem 7.8 says that every $t(n)$ time multitape TM has an equivalent $O(t^2(n))$ time single-tape TM. Hence, in this case we have $O(n^4)$.

2. Show that P is closed under union, concatenation, and complement.

   Answer:

   Let $A, B \in P$, with TMs M_1 and M_2 deciding in polynomial time.

   First $A \cup B$:

   TM $M = M(w)$:

   (a) Run M_1 on input w, and halt on `accept` *if M_1 accepted*
   (b) Run M_2 on input w, and halt on `accept` *if M_2 accepted*
   (c) Halt on `reject`

   The above is a polynomial time decider for $A \cup B$ as both M_1 and M_2 are such.

   Then $A \circ B$:

   TM $M = M(w)$:

   (a) For $i = 0$ to $|w|$:
      i. Run M_1 on substring $w[1 : i]$, and M_2 on substring $w[i + 1 : |w|]$
      ii. Halt on `accept` *if both accepted*
   (b) Halt on `reject`

   The above runs also in polynomial time as both M_1 and M_2 run in polynomial time.
   More precisely: suppose that the worst-case running times of M_1 and M_2 are $O(n^{k_1})$ and $O(n^{k_2})$. Then, the for-loop of the above M is run $n + 1$ times in the worst case, and the time complexity of M is $O(n + 1) \cdot (O(n^{k_1}) + O(n^{k_2})) = O(n^{1+\max\{k_1,k_2\}})$, which is a polynomial function of n.

   Finally the complement, A^c:

   TM $M = M(w)$:

   (a) Run M_1 on input w
   (b) Halt on `accept` *if M_1 rejected; otherwise halt on* `reject`

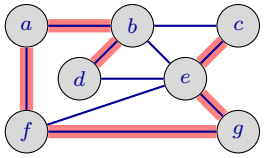
   The above runs in polynomial time as M_1 runs in polynomial time.

3. Consider the undirected graph G depicted below.

   a) Determine a Hamiltonian path for G (for some pair of nodes).
   b) How many edges can be removed so that such a Hamiltonian path still exists?
   c) Hamiltonian cycle is like Hamiltonian path, but at the end one still needs to return to the starting node. Show that no such cycle exists in given G.

   Answer:

a) E.g., $d - b - a - f - g - e - c$ works.



b) Path itself must contain $n - 1$ edges if there is n vertices. We can remove all other edges, i.e., four edges in this case.

c) Nodes c and d are degree two, and a potential circuit would visit them using the corresponding edges. This would form a smaller circuit, and such may not exist in Hamiltonian circuits. Therefore, no Hamiltonian cycle can exist.

4. Determine which of the following formulæ are satisfiable:

   a) $(x_1 \vee x_2) \wedge (x_1 \vee \bar{x}_2)$

   b) $(x_1 \vee x_2) \wedge (x_1 \vee \bar{x}_2) \wedge (\bar{x}_1 \vee x_2) \wedge (\bar{x}_1 \vee \bar{x}_2)$

   c) $(\bar{x}_1 \vee \bar{x}_2) \wedge (x_1 \vee x_2 \vee x_3) \wedge (\bar{x}_1 \vee \bar{x}_3)$

   Answer:

   a) ✓: set $x_1 = 1$ and x_2 can be anything.

   b) ✗: first two clauses imply that x_1 should be true; last two clause imply the same for $\bar{x}_1$. x_1 cannot be both so this expression is never true.

   c) ✓: e.g, set $x_1 = 0$ and the first and last clause are true. The middle clause is true if either x_2 or x_3 is true.

5. a) Let A be a DFA with m states. Argue that all "useful" states (the states that can be reached with some input string) can be reached with input strings whose lengths are $m - 1$ or less.

   b) Devise a Turing machine that determines if an arbitrary DFA rejects any strings, i.e., a TM that decides on

   $$\{\langle A \rangle \mid A \text{ is a DFA that rejects some string}\}.$$

   c) Analyze the time complexity of your Turing machine and determine if it is in P.

   Answer:

   (sketch)

   a) DFA is initially in starting state. Suppose that some state x can be reached only after m steps (or more). So the first $m - 1$ steps the DFA spends in other states, the number of which is $m - 1$. Pigeonhole principle says that at least one state is visited twice. Hence, a shorter path to state x exists, a contradiction, and therefore all (reachable) states are within $m - 1$ steps.

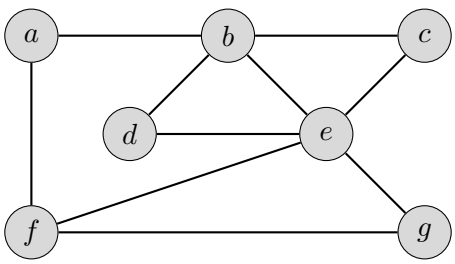


Figure 8: Graph G.

b) Let us describe briefly two TMs that both decide on the given language, denoted by B.

TM M_1 is build on the observation made in a). M_1 generates (sequentially) all strings of length $m-1$, and simulates A with them. If a reject state is reached during the simulation, i.e., A would reject the corresponding substring, then M_1 Halts on `accept`*. If no string reaches a reject state, M_1 Halts on* `reject`*. This is a decider, as the number of rounds is bounded.*

TM M_2 constructs/traverses are "reachability" graph G, where each (marked) vertex corresponds to a reachable state in the DFA. Initially, we start with one vertex corresponding the starting state of DFA A. At every step, M_2 Halts on `accept` *if a* reject state of the DFA *is reached. Otherwise, M_2 adds vertices to G corresponding to those states that are not in G but can be reached from the states currently in G. (cf. DFS and BFS). This is repeated until no more vertices is added, and M_2 Halts on* `reject`*.*

c) **First $\mathbf{M_1}$** *(brute force): Suppose that A is nontrivial and $|\Sigma| \geq 2$. Then, in the worst case, M_1 checks $|\Sigma|^{m-1} \geq 2^{m-1}$ strings. This means that M_1 is not in P. However, it is easy to see that B is in NP, as giving a string that A rejects serves as a trivial certificate that can be verified in polynomial time. Equivalently, checking the strings nondeterministically gives a polynomial running time.*

Then $\mathbf{M_2}$ *(dynamic programming): This algorithm is much more efficient and in P. The DFA can be seen as a graph, and checking the states that can be reached from a given state (with a suitable input symbol) takes a constant time (with Σ fixed). Each round takes at most time $O(m^2)$, and we can run at most m rounds, so the worst-case running time is $O(m^3)$, and therefore B is in P. (In fact, checking connectivity can be done in $O(\log n)$ time ...).*

Traveling Salesman Problem

You can find from UGLA a file named `tsp-2018.loc`, which contains (x, y) coordinates of locations in plane. Distance between two locations is their Euclidean distance,

$$d = \sqrt{(x_1 - x_0)^2 + (y_1 - y_0)^2}.$$

We experiment with the travelling salesman problem (TSP) using the corresponding complete graph, where the distance between two vertices is their Euclidean distance in plane. In TSP, we need to find a shortest circuit that visits every node and then returns back to the start. This problem is NP hard, and thus we resort to (simple) heuristics.

Problems:

1. Write a program that determines the length of the circuit if nodes are visited in the order $1, 2, \ldots, n, 1$. You can choose your favorite programming language (e.g., Python, Java, Perl). Report the length of the circuit.
2. Experiment with some random permutations and see how good solutions you can find by guessing. Report on your experiments and include a table of the lengths you determine.
3. Implement a greedy algorithm that works as follows:
   (a) Start with node 1, and tag it.
   (b) While there are untagged nodes left:
      i. Choose the shortest edge from the last tagged node to an untagged node
      ii. Add the corresponding edge and node to the circuit and tag the node
   (c) Add the edge from the last node to start (node 1).

   Report both the circuit and its length.
4. Analyze the above algorithm and determine its time complexity using big-O notation.
5. Develop a better algorithm. How short circuit can you find?

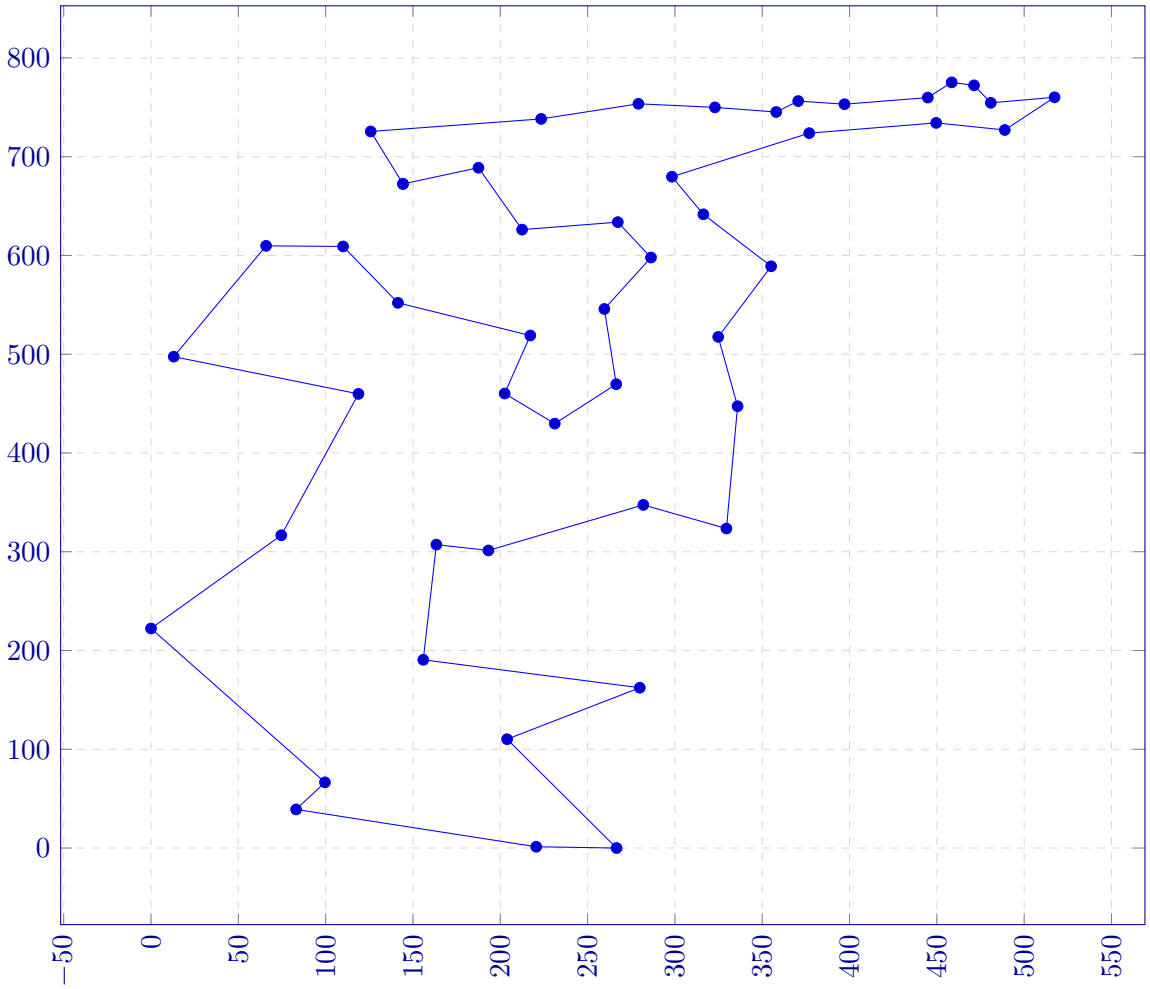
Note:

- **Return your answers in a short "report"**
  - State the numerical values clearly
  - Also explain "why" when appropriate
- **Also include your code in the report.**

You are encouraged to **publish the length** of the shortest circuit you can find (last question) in Piazza.

However, **do not publish the circuit** itself. Let's see how short a route we can find within one week!

Answer:



A relatively short tour, depicted above, is

$$\begin{array}{rrrrrrrrrr}
\{0, & 42, & 17, & 29, & 10, & 40, & 13, & 41, & 32, & 39, \\
11, & 16, & 15, & 27, & 14, & 43, & 25, & 31, & 9, & 8, \\
4, & 26, & 20, & 46, & 7, & 34, & 47, & 33, & 3, & 6, \\
28, & 44, & 24, & 45, & 22, & 37, & 36, & 18, & 30, & 5, \\
35, & 23, & 21, & 19, & 38, & 12, & 1, & 2 & \} &
\end{array}$$

with the length of about **3352.4**. *In visual inspection, the tour seems reasonably good as there is no random "zigzag:s".*