TÖL301G Formal Languages and Computability

Esa Hyytiä
Department of Computer Science
University of Iceland

Example Solutions to Weekly Exercises

Contents

Exercise 1: Deterministic Finite Automata (DFAs) **2**
Exercise 2: NFAs **5**
Exercise 3: Regular Expressions (RE) **8**
Exercise 4: Non-regular languages **10**
Exercise 5: CFG **12**
Exercise 6: Pushdown Automata **14**
Exercise 7: Turing Machines (TMs) **17**
Exercise 8: Decidability **22**
Exercise 9: Decidability and mapping reduction **25**
Exercise 10: Time complexity **27**
Exercise 11: NP-completeness **30**
Exercise 12: SAT, SPACE and Probabilistic TMs **33**

The complete set, v.0

Problems:

1. A game proceeds so that at every step one randomly chosen individual is removed from the game. Suppose that team A has $n > 0$ persons left and team B $m > 0$ persons left. Let $p(n, m)$ denote the probability that the m members of team B get removed before a single member of team A needs to leave. Show by induction (on m) that

$$p_n(m) = \frac{m!}{\prod_{i=1}^{m}(n+i)}.$$

Answer:

- *Let $n > 0$ be fixed.*
- *Case $m = 1$ is true (for every n), so the base is covered.*
- *Suppose that expressions holds for $m = 1, 2, \ldots, k-1$. For $p_n(k)$ we have*

$$p_n(k) = \frac{n}{n+k} \cdot 0 + \frac{k}{n+k} \cdot p_n(k-1),$$

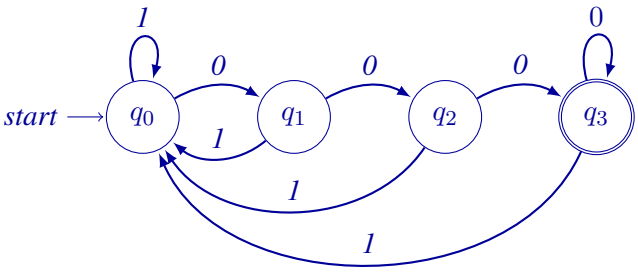
yielding

$$p_n(k) = \frac{k}{n+k} \cdot \frac{(k-1)!}{\prod_{i=1}^{k-1}(n+i)} = \frac{k!}{\prod_{i=1}^{k}(n+i)},$$

and therefore it also holds for $m = k$. By induction the expression holds for any (finite) m (and n).

2. Draw a state diagram for a DFA that *accepts* any binary string (e.g., 0101000) that ends with three or more consecutive zeroes. Then implement your DFA using the syntax of our simulator and submit it to Gradescope with filename `ThreeOrMoreOnesDFA.txt` for autograding!

Answer:



3. 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 form $x_1y_1x_2y_2x_3y_3\ldots$ where pairs (x_i, y_i) denote the choice of player 1 and 2 at round i. DFA accepts the game if player 1 wins. Games that player 2 wins, or if more moves follow after the end, are rejected.

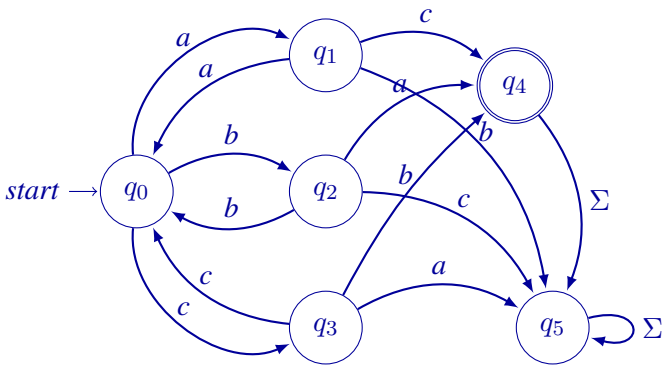
   a) Draw the corresponding state diagram.

   b) How many states are needed?

c) Implement your DFA to our simulator and submit the code to Gradescope with filename `PaperScissorRockDFA.txt` (important!).

Answer:

a) So a string is accepted only if player 1 wins:



b) 6 states as in above.

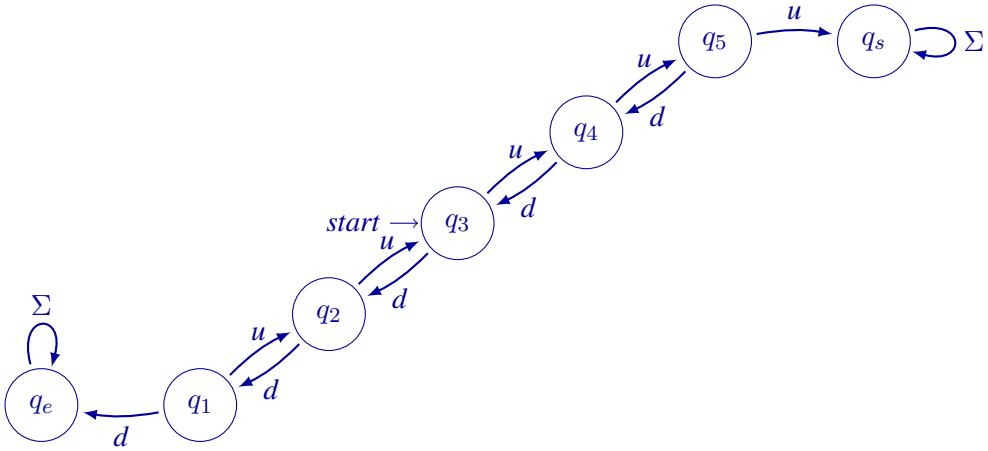
4. Let $\Sigma = \{u, d\}$, where u denotes a move "up" and d a move "down", and a string w describes commands given to a drone that is initially at altitude $h_0 = 3$. The altitude at time t is $h_t = h_{t-1} + 1$ if we read u, and with d we have $h_t = h_{t-1} - 1$. The drone should not land and neither should it fly at too high altitude, so string w is accepted if $0 < h_t \leq 5$ for all $t = 0, \ldots, |w|$.

a) Design a DFA that recognizes valid strings.

b) Implement it using the syntax of our simulator and submit your code to Gradescope for auto-grading using filename `FlyingDroneDFA.txt`. (important!)

Answer:

a) For example:



Note that states q_e and q_s can be combined. Here they are separaly just for clarity.

b) Left as an exercise!

Autograder – IMPORTANT

It is important that you submit your code using the given filenames! Additionally, you need to submit one pdf file that includes Problem 1 (and preliminary steps of other problems), e.g. scan an A4 paper.

Example: two state DFA

```
name: testing
init: q0
accept: q1

q0,0,q0
q0,1,q1
q1,0,q0
q1,1,q1
```

- First three lines describe the name of the DFA, the start state, and accept states (comma separated)
- The other lines describe the transitions: `[current state],[symbol],[new state]`

That's all!

Problems:

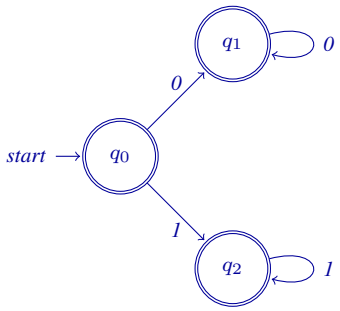
1. Let $\Sigma = \{0, 1\}$ and let A denote a language of strings that consist solely of zeroes, or ones (including the empty string). For example, strings 000 and 11 belong to A, whereas 11101 is not.

   a) Draw a state diagram of an NFA that recognizes A.
   b) Implement your NFA using the syntax of our simulator.
   c) How many states you need with an DFA to the same task?

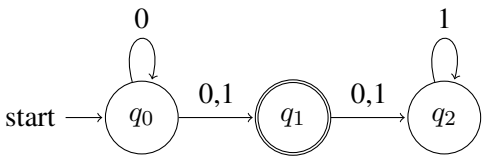
   Note: submit your code to autograder in file `OnlyOnesOrZeros`.

   Answer:

   The simple NFA depicted below does the job. For DFA, one needs one extra state (reject).



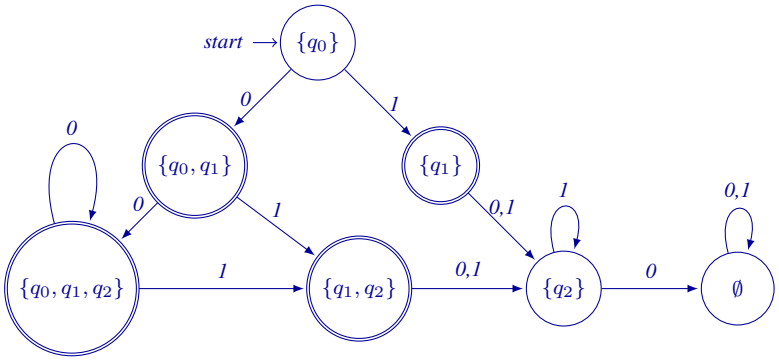
2. Convert the NFA depicted below into an equivalent DFA. Use the syntax of our simulator to describe your DFA. Name the file as `ToDFA` (**important!**).



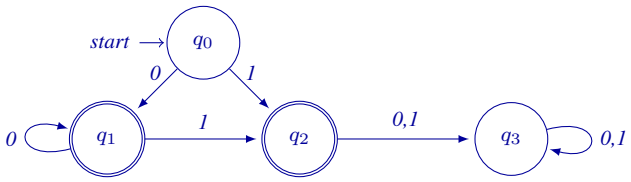
   Answer:

   In principle, the DFA corresponding to the given NFA looks as in the problem description. We give two solutions.

   In the first solution, each state is from the powerset of Q (i.e., a subset of states of the NFA):



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 below.



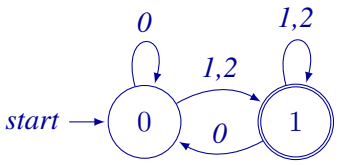
3. Consider the ternary number system (i.e., base 3, where the digits are $0, 1, 2$).

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

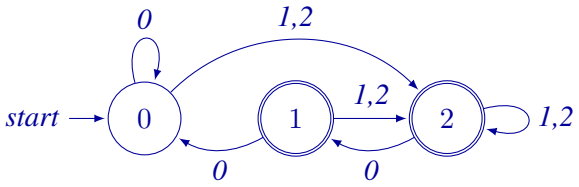
   Name your submissions as `NotBy3` and NotBy9 (filenames for autograder, important!).

   Answer:

   a) A ternary number is divisable by three if the last digit (if any) is zero, and vice versa:



   c) A ternary number is divisable by 9 iff last two digits are zero, so we need to exclude that:

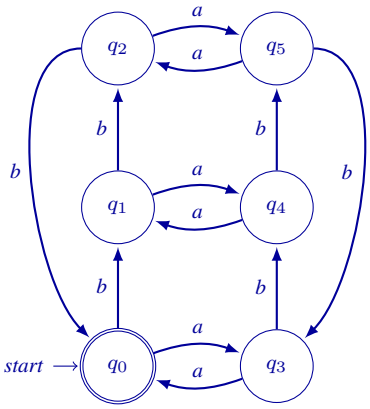


4. Let $\Sigma = \{a, b\}$ and let A denote a language of strings w where w consists of $2n$ a's and $3m$ b's, $n, m \geq 0$. Note that the symbols can be in any order, only their count matters. For example, strings $ababb$ and $bbbaa$ are in A, whereas $abba$ is not.

   a) Draw a state diagram of a DFA that recognizes A.
   b) How many states you need? Is there an NFA with a fewer number of states?
   c) Implement the DFA/NFA to our simulator and submit it with filename `TwoA-ThreeB`.

   Answer:

   a) We need to count the symbols separately.



b) Six states (see above), and working with an NFA does not help in reducing the number of states.

c) Below is one example solution:

```
name: TwoA-ThreeB
init: q0
accept: q0

q0, a, q1
q0, b, q2
q2, a, q3
q2, b, q4
q4, a, q5
q4, b, q0
q1, a, q0
q1, b, q3
q3, a, q2
q3, b, q5
q5, a, q4
q5, b, q1
```

Problems:

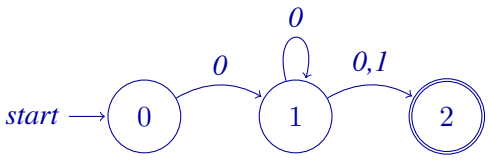
1. Let us consider the language $L(00^*(0 \cup 1))$.

   a) Draw the state diagram of an NFA that accepts it.

   b) Implement your NFA using the syntax of our simulator and submit your code with filename `RegexLang` for autograding.

   Answer:

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



   The corresponding RE is (for example!) `^0+(0|1)$`.

2. Suppose the inbut alphabet is binary numbers, $\Sigma = \{0, 1\}$, and language A consists of binary numbers x such that $x \bmod 3 = 1$. In other words,

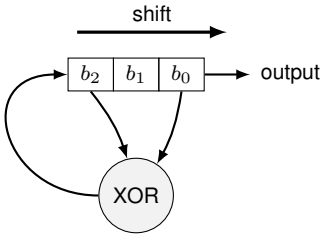
$$A = \{x \in \Sigma^* \mid x \bmod 3 = 1\}.$$

   Determine a regular expression for A, and submit it (one line) using the filename `Mod3`.

   Answer:

   This is similar to the tutorial problem where the criterion was $x \bmod 3 = 0$. Simply redefine the set of accept states (one state here), and redo the RE. (left as a homework exercise!)

3. Linear Feedback Shift Registers (LFSRs) are computationally efficient pseudo random number generators. Let us consider a very short 3 bit version depicted below:[1]



   In C, you could write:

```c
b = x & 1;     /* output bit */
x = (x>>1) | ( ((x<<0)^(x<<2)) & 4 ); /* update the state */
return b;
```

   So the new MSB b_2 is the result of XOR operation on bits b_0 and b_2, and at the same time all bits are shifted one right. The output bit is b_0 that gets shifted "out". The initial state (=seed) can be any other state than $x = 0$ (which is forbidden as then the state would never change!). Given the feedback structure is chosen carefully, LFSR achieves the maximal period of $N = 2^n - 1$, i.e., the state cycles through all possible bit combinations. In our case, $N = 7$.

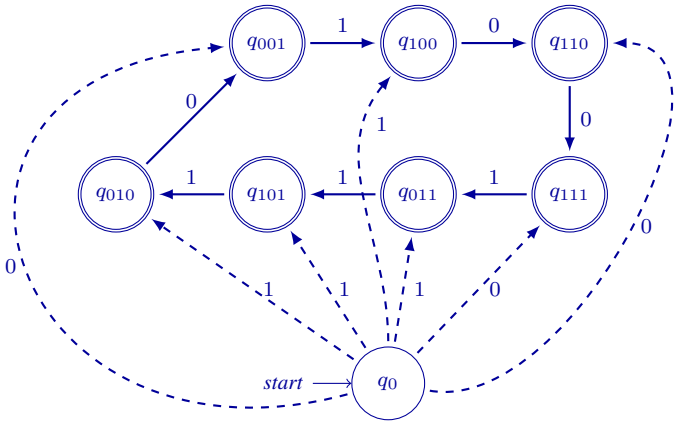
[1] For any real purposes, you probably want to use a larger size than that!

You are working for the MSA that is monitoring a huge number of bitstreams – and there is a need to identify those that are generated by the LFSR! So we define the language A as the finite bit strings that the above LFSR generates (with some non-zero initial value). We exclude the empty string ϵ from A. Starting, e.g., from state 001, it is easy to see that the next states are 100 and 110. Consequently, bit strings 1, 10 and 100 are in A. As the LFSR generates an infinite sequence of bits, language A is not finite.

Construct a NFA that recognizes A using the syntax of our simulator. Submit your code using filename `LFSR3NFA`.

Answer:

The NFA needs to have a state for each possible state of LFSR, and an additional start state q_0:



As said, the output bit string is from LFSR is periodic. In our case, the length of the period is $N = 7$ symbols,

$$\ldots \quad \underbrace{0\ 1\ 0\ 0\ 1\ 1\ 1\ 0}_{\text{one period}}\ 1\ 0\ 0\ 1\ 1\ 1\ 0 \quad \ldots$$

4. Determine a regular expression that generates the language A for the above LFSR and submit it (one line) using filename `LFSR3RE`.

Answer:

The answer is a bit tedious, but based on the periodic bit string, e.g.,

```
^((1|11|111|0111|00111|100111|10011|011|0011|10011|001|1001|00|100)|\
((0|10|110|1110|01110|001110|1001110)(1001110)*(|1|10|100|1001|10011|100111)))$
```

(two rows are concatenated, without the backslash). The first row lists all strings shorten than period that the second pattern does not include. The second pattern includes all strings longer than the period ($N = 7$), and some shorter strings.

The start of line ^ and the end of line $ symbols are important(!).

Problems:

1. Give an example where A, B and $A \cap B$ are all nonregular.

   Answer:

   For example, let $A = B = \{0^n 1^n \mid n \geq 0\}$.

2. Give an example where A, B are nonregular, but $A \cap B$ is regular.

   Answer:

   For example, $A = \{0^n 1^n \mid n \geq 0\}$ and $B = \{1^n 0^n \mid n \geq 0\}$. Then the intersection is the empty language, which is trivially regular!

3. Prove that the following languages are not regular

   a) $\{0^{n+m} 1^m \mid n \geq 0, m \geq 0\}$
   b) $\{0^n 1^m 0^n \mid n \geq 0, m \geq 0\}$
   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 $s = 0^p 1^p$. Pumping Lemma says $s = xyz$ such that $|xy| \leq p$ 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 $s = 0^p 10^p$, and pumping up y (or down) yields a contradiction.

   c) Here we can choose $s = 0^p 10^p 10^p 1$, i.e., the substring w is $0^p 1$. This produces a contradiction when pumped up.

4. Let $F = \{a^i b^j c^k \mid i, j, k \geq 0, \text{ and if } i = 1 \text{ then } j = k\}$.

   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).

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 the leftmost derivations and parse trees for the following strings:

a) $a \times (a)$
b) $a + a$

Answer:

a) Derivation for $a \times (a)$:

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

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.

2. The DFA depicted below determines whether the remainder after dividing a *binary number* is one. Convert it to a corresponding context-free grammar.

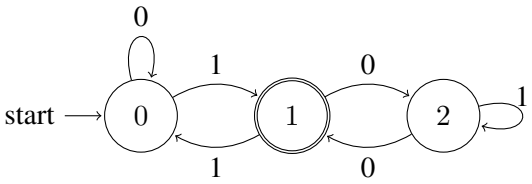


Figure 1: DFA to check whether $x \bmod 3 = 1$.

Answer:

Let us proceed similarly as in the slides.

We let R_0, R_1 and R_2 denote the three variables corresponding to the states q_0, q_1 and q_2, where R_0 is the starting variable. The production rules are as follows:

$$R_0 \to 0R_0 \mid 1R_1$$
$$R_1 \to 0R_2 \mid 1R_0 \mid \epsilon$$
$$R_2 \to 0R_1 \mid 1R_2$$

where the ϵ is due to the fact that q_1 is the (only) accept state.

3. 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 $Z \to 0$.

4. Let

$$A = \{u^i d^k u^{k-i} \ : \ k \geq i \geq 0\}.$$

a) Let us first visualize language A. Suppose that letter u corresponds to one step "up", and letter d one step "down". Describe informally with 1-2 sentences what kind of strings are in A.
b) Show that A is CFL.

Answer:

a) All strings in A first move i steps up, $i \geq 0$, and then "dive" back to the start level, and possibly below it, and then return back to the start level.
b) Substituting $j = k - i$ to the definition of A gives an alternative form,

$$A = \{u^i d^{i+j} u^j \ : \ i, j \geq 0\}.$$

From this it's easy to write down a CFG G:

$$S \to AB,$$
$$A \to uAd \mid \epsilon,$$
$$B \to dAu \mid \epsilon.$$

Clearly $L(G) = A$ and therefore A is a CFL.

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, give a CFG or a PDA, either of which implies 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. If A and B are languages, define $A \diamond B = \{xy \mid x \in B \text{ and } y \in A \text{ and } |x| = |y|\}$. Show that if A and B are regular languages, then $A \diamond B$ is a CFL.

   Answer:

   We have NFA N_1 for B and another NFA N_2 for A. 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 $\epsilon, \$ \rightarrow \epsilon$ 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.

3. Flying drones, part 2! Let $\Sigma = \{u, U, d, D\}$ denote the alphabet, where the four symbols correspond to integers $1, 2, -1, -2$, respectively (u and U for up, d and D for down). Strings over Σ correspond to flying instructions (program). Let A denote the language that consists of strings w that correspond to flights that take-off immediately and remain in the air until the last step that brings the drone back to the ground level (but not below it!). So for example, uuD corresponds to $(1) + (1) + (-2)$, and correspond to a valid trajectory. In contrast, uud is not in A as the drone has not landed yet!

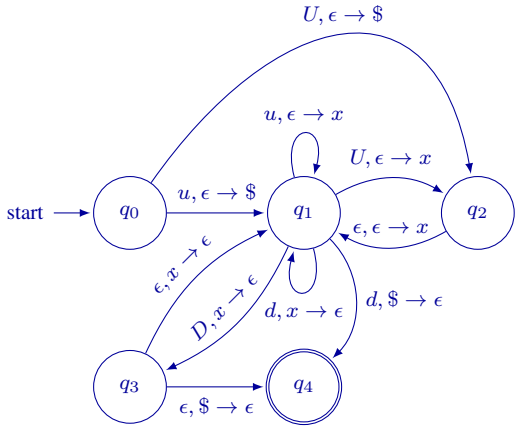
   a) Formulate the above condition mathematically (give mathematical expressions that a string $w = a_1a_2 \cdots a_k$ must satisfy).
   b) Show that A is a CFL by giving a state diagram of PDA that recognizes A.

Answer:

a) Let variables correspond to the given numerical values, $-2, -1, 1, 2$. Then we have

$$\sum_{i=1}^{j} a_j > 0, \qquad \textit{forall } j = 1, \ldots, k-1,$$
$$\sum_{i=1}^{k} a_j = 0.$$

b) The state diagram below depicts one solution. Each token in the stack corresponds to unit height (up).



4. Let a and b denote two binary numbers of the same length, $|a| = |b|$. Their Hamming distance is equal to the number of positions where bits are different. This is the same as counting the one bits in $a \oplus b$ (i.e., the bitwise XOR of a and b). Let A denote the language that corresponds to checking whether the Hamming distance between two binary strings is at least three,

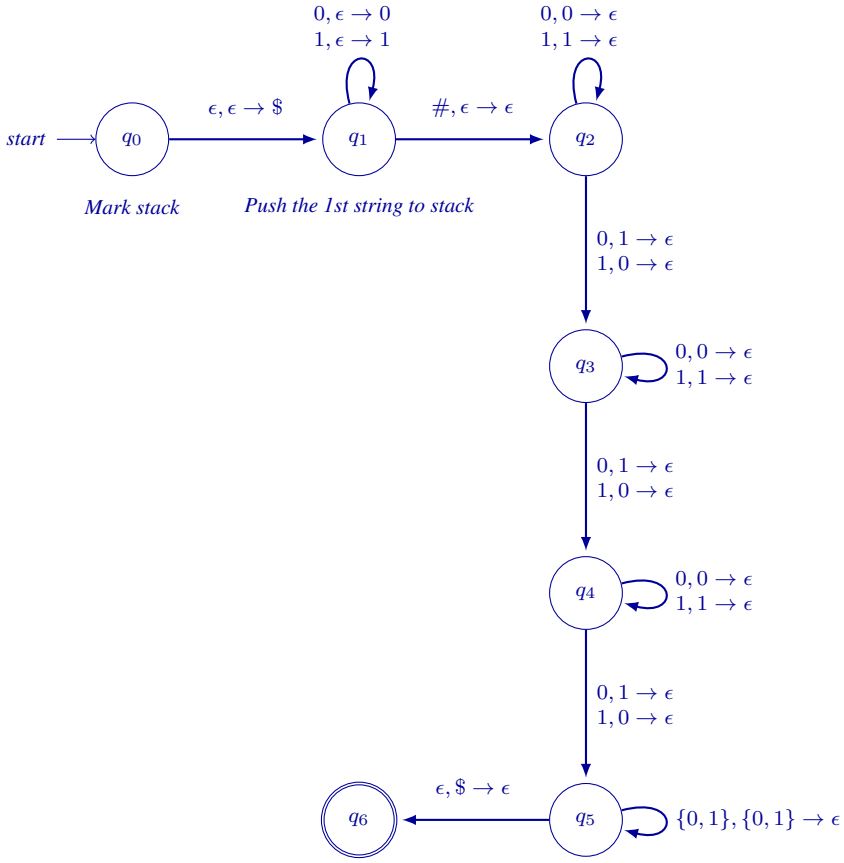
$$A = \{a\#b \quad | \quad a, b \in \{0,1\}^*,\ |a| = |b| \text{ and } \text{dist}(a, b) \geq 2\}.$$

where $\text{dist}(a, b)$ denotes the Hamming distance between a and b. Show that A is a CFL by constructing a PDA that recognizes it. You can assume that the input alphabet is $\Sigma = \{0, 1, \#\}$, and that b is in reverse order to facilitate the computations with the stack. For example, $1011\#1101$ is not in A because $\text{dist}(1011, 1101) = 0$, whereas $1011\#1000$ is in A as $\text{dist}(1011, 0001) = 2$. **Draw a state diagram of your PDA and submit it!**

Hamming distance is an important measure in coding theory quantifying the difference between two codewords. For example, one can require that the Hamming distance between any two codewords is three. Then, a received bit string w is decoded to a symbol whose codeword is "closest" to w. Given one bit has flipped during the transmission, one can still recover the correct symbol.

a	*00000*
b	*10110*
c	*01101*

Answer:



Note that

- *States $q_2, \ldots, q_5$ include the information on how many differences have been encountered so far: q_2=none, q_3=one, ..., q_5=three or more*
- *We "sit" in state q_5 to ensure that the bit strings have equal lengths*
- *PDA moves to state q_6 once the bottom of the stack is reached (special symbol pushed there at the start), and the string is accepted if there is no further input. Otherwise ... REJECT!*

Problems:

1. Let us consider binary numbers with alphabet $\Sigma = \{0, 1, \#\}$, where $\#$ is used as a delimiter between two binary numbers. Then define language A,

$$A = \{x\#y \ : \ x, y \in \{0,1\}^+ \text{ and } 2x = y\}.$$

So, e.g., $0\#0000 \in A$ and $1011\#10110 \in A$, but $1011\#1011 \notin A$. Show that A is decidable by constructing an appropriate TM using the syntax of TM simulator. Submit your code with filename `2xybin.txt`.

Answer:

We give an implementation level solution here and leave the actual implementation to the reader.

$M = M(w)$:

(a) While scanning from left to right, (i) replace the initial zeroes in x and y with $\#$ and (ii) validate the format. If there is a problem, then halt on `reject`.
(b) If the last bit of y is 1, halt on `reject`. Otherwise write a blank symbol over it.
(c) Rewind and compare the remaining strings x' and y'. If strings are equivalent, halt on `accept`. Otherwise, halt on `reject`.

So instead of multiplying x by 2, we decided to divide y by 2.

2. Let us consider the same language but now x and y are given as ternary numbers, so that the alphabet is $\Sigma = \{0, 1, 2, \#\}$. In this case, $0\#0000 \in A$ and $12\#101 \in A$, but $11\#110 \notin A$. Show that A is decidable by constructing an appropriate TM using the syntax of TM simulator. Submit your code with filename `2xyter.txt`.

Answer:

In this case, we cannot avoid carrying out the actual multiplication. Here we give an implementation level solution again, and leave the actual implementation to the reader. Note that the TM of simulators has infinite tape to both directions.

$M = M(w)$:

(a) While scanning from left to right, (i) replace the initial zeroes in x and y with z and (ii) validate the format. If there is a problem, then halt on `reject`.
(b) Next we carry out the multiplication by 2:
i. Rewind to $\#$ and move one more to left (the lowest digit of x)
ii. Carry bit can be stored "in state". Initially $C = 0$.
iii. Multiply the digits x_i one at a time from right to left using the following table

	x_i		
	0	1	2
$C = 0$	$(0,0)$	$(2,0)$	$(1,1)$
$C = 1$	$(1,0)$	$(0,1)$	$(2,1)$

where pairs (x, c) correspond to the new value $x_i \leftarrow (2x_i + C \bmod 3)$ and the new carry (1 if the result was 3 or more). It is not a problem if $2x$ is one digit longer than x as the tape extends also to the left (check the carry at the end!).
(c) Compare the strings and halt accordingly.

---

3. Let us consider language S consisting of binary representations of perfect squares,

$$S = \{x \mid \ x = n^2 \text{ for some integer } n \ \}.$$

So, e.g., 001 (i.e., one) and 100 (i.e., four) are in S, but 1011 (=eleven) is not. Give a high-level description of a TM that decides on S.

Answer:

Recall that a high-level description simply explains the steps in the algorithm without describing how everything is implemented using the tape. It is easy to come up with different kinds of algorithms that complete the task. We give two examples:

"Brute force" algorithm: *(with integer multiplication)*

i) *Set* $n = 0$ *and* $s = 0$

ii) *While* $(s < x)$*:*
$n \leftarrow n + 1$
$s \leftarrow n^2$

iii) *If* $(s = x)$ *return* Accept
Otherwise return Reject

For simplicity, the above algorithn avoids using floating point numbers and computing (integer) square roots, which can be utilized in practice. Instead, it computes squares of integer numbers and compares their values to x*. Note that the loop has a finite upper bound (for each* x*), and therefore the corresponding TM* **is a decider**.

An alternative solution: *(with integer addition)*

i) *Set* $j = 0$ *and* $s = 0$ $\quad (j = 2n$ *and* $s = n^2)$

ii) *While* $(s < x)$*:*
$s \leftarrow s + j + 1$
$j \leftarrow j + 2$

iii) *If* $(s = x)$ *return* Accept
Otherwise return Reject

The first two example solutions basically test all numbers $n = 0, 1, \ldots$ *until* $n^2 \geq 0$*, and then halt accordingly, i.e., a* "linear search"*. This takes about* $\sqrt{x}$ *rounds, which itself does not sound too bad. Still, when* x *is very large, also* $\sqrt{x}$ *is large.*

Fast-forwarding: *One crude way to "fast-forward" at start is as follows:*

i) *If* $x = 0$*, return* Accept

ii) *Set* $k = 0$ *and* $s = 4$

iii) *While* $(s \leq x)$*:*
$s \leftarrow 4 \cdot s$ $\quad$ *(shift left by 2)*
$k \leftarrow k + 1$

iv) *Take back the last shift:*
$s \leftarrow s/4$ $\quad$ *(shift right by 2)*

First we determine the largest k *for which* $2^{2k} \leq x$*. Then we can start the brute-force search from* $n = 2^k$*. For the second algorithm, we start with* $j = 2^{k+1}$ *and* $s = 2^{2k}$*.*

Good news: *The procedure is fast especially when using bit shift operations.*
Bad news: *It takes you only to "half-way": the average length of the search reduces to one third*

Remark: *The trivial upper bound for a feasible n is the x itself because $\sqrt{x} \leq x$ for all $x \geq 0$. However, we can easily come up with better bounds. For example,*

$$\lfloor \sqrt{x} \rfloor \leq \lfloor (x+1)/2 \rfloor =: m(x). \tag{1}$$

First, we can manually check that (1) *holds for $x = 0, \ldots, 3$:*

x	$\sqrt{x}$	$m(x)$
0	0	0
1	1	1
2	≈ 1.42	1
3	≈ 1.73	2

Then, for $x \geq 4$, we have

$$\sqrt{x} \leq \sqrt{x} \cdot \underbrace{\frac{\sqrt{x}}{2}}_{\geq 1} \leq \frac{x}{2} \leq \left\lfloor \frac{x+1}{2} \right\rfloor.$$

Hence, if x is a perfect square, then $\sqrt{x} \in \{0, \ldots, m(x)\}$.

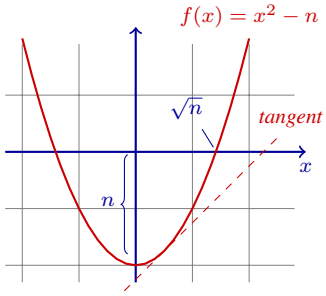
A silly idea: *Run two TMs in parallel, one checking numbers $1, 2, \ldots$, the other one numbers $m(x)$, $m(x)-1$, ... This is parallel computation! But does this help? Is there a better way to utilize two TMs?*

Third algorithm: **(with division)**

The third option is to compute the square root of x (approximately), and then check if the square of the nearest integer is equal to x.

Newton's algorithm is an efficient way to solve $f(x) = 0$, when $f(x)$ is differentiable.
In general, the iteration step is

$$x \leftarrow x - \frac{f(x)}{f'(x)}.$$



For computing $\sqrt{n}$, we define $f(x) = x^2 - n$.
Then the iteration step is

$$x \leftarrow \frac{x^2 + n}{2x} = \frac{x + n/x}{2}$$

which is repeated until a desired accuracy is reached. (Note that (1) is obtained with $x_0 = 1$).

Instead of floating point arithmetic, the same steps can be carried out using integer arithmetic with an appropriately modified stopping criterion. Either way, the above expression includes (integer) division, but otherwise it converges fast!

Fourth algorithm: binary search
Details are left to the reader.

Fifth idea: *Fiddle with Prime numbers*
First, handle the cases $x \in \{0, 1, 2, 3\}$ manually.
For $p = 2, 3, 5, \ldots$ (set of primes!)

(a) $(q, r) \leftarrow$ *x divided by p* *(quotient, remainder)*

(b) If $r = 0$ (i.e., p divides x):

- $(x, r) \leftarrow$ *q divided by p*
- *If $r \neq 0$, then Halt on* `reject`
- *If $x = 1$, then Halt on* `accept`
- *Otherwise, Repeat step (a)*

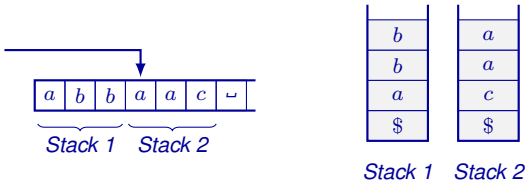
Q: Why does this work? Or does it work? This seems to be efficient in some cases. For example, it says "no" to $x = 42 \cdot 11^{42}$ immediately during the first round!

4. 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*



   Both stacks have been initilized so that $ marks the bottom of the stack. Initially, the input string is thus in Stack 2.

   Otherwise, the different actions of a TM can be emulated as follows:

   Before every step: *If $ at the top of Stack 2, push ␣ there*

   Movement left:

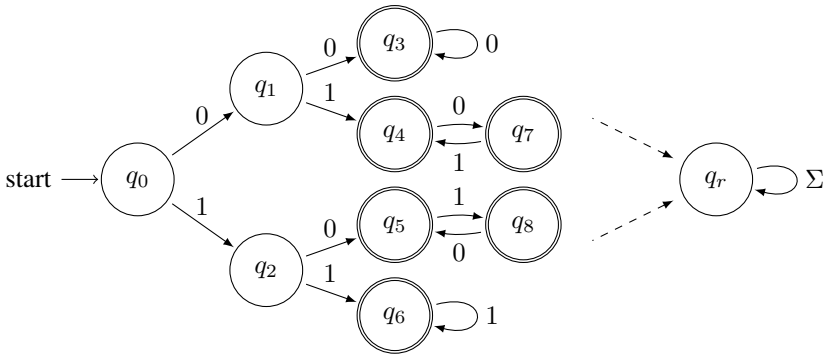
   - *If $ at the top of Stack 1, ignore*
   - *Otherwise, pop one symbol from Stack 1 and push it to Stack 2*

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

   Reading: *The symbol on the top of Stack 2*
   Writing: *On the top of Stack 2 with pop & push*

Problems:

1. Consider the following DFA M. The omitted transitions lead to the (reject) state q_r.



   a) Is $1011 \in L(M)$?
   b) Is $\langle M, 1011\rangle \in A_{\text{DFA}}$?
   c) Is $\langle M, 1010\rangle \in A_{\text{DFA}}$?
   d) Is $\langle M\rangle \in EQ_{\text{DFA}}$?

   In each case, argue briefly why! (one sentence is often enough)

   Answer:

   a) No, the given DFA rejects string 1011.
   b) No, for the same reason. A_{DFA} *includes only pairs* $\langle M, w\rangle$ *where* M *accepts* w.
   c) Yes, M *accepts* 1010 *and thus* $\langle M, 1010\rangle \in A_{DFA}$.
   d) No. Input is of wrong type. It should describe two DFAs.

2. Let A and B be countable sets. Show that $S = A \times B \times B \times A$ is countable.

   Answer:

   First we show that the Cartesian product $H \times K$,

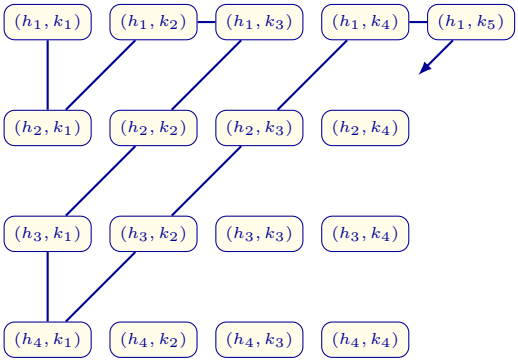
$$H \times K = \{(h, k) \,:\, h \in H \text{ and } k \in K\},$$

   of two countable sets H *and* K *is countable.*

   The fact that H *and* K *are countable, the existance of the bijection to* $\mathbb{N}$ *means that we can write* $H = \{h_1, h_2, \ldots\}$ *and* $K = \{k_1, k_2, \ldots\}$. *The pairs can be now placed to an infinite matrix/table*

$$\begin{pmatrix} (h_1, k_1) & (h_1, k_2) & \cdots \\ (h_2, k_1) & (h_2, k_2) & \\ \vdots & & \ddots \end{pmatrix},$$

   which can be enumerated in a similar "diagonal zig-zag" manner as the fractional numbers (cf. lectures). For example, according to the following curve which visits all pairs in a specific order.



Hence, $H \times K$ is also countable. This means that $A' = A \times B$ and $B'B \times A$ is countable, and consequently, $S = A' \times B'$ is countable.

3. Let us consider binary numbers with a possible binary point (cf. decimal point). So the alphabet is $\Sigma = \{0, 1, '.'\}$ and the language may include both whole numbers such as 01001 and 101, as well as fractional numbers such as 1.011, where the latter corresponds to $1 + 3/2^3 = 1.375$. The language A we are interested in is

$$A = \{w \mid (w)_2 \leq \sqrt{8}\}.$$

i.e. A includes strings w that correspond to binary numbers (with a possible binary point) such that $(w)_2 \leq \sqrt{8}$, where $\sqrt{8} \approx (10.110101)_2$. E.g., 1.011 is thus in A, but 11.011 is not.

a) Is A (i) finite, (ii) countably infinite, or (iii) uncountable set?
b) Is $w = 1.010101010\ldots$ corresponding to $\sum_{i=0}^{\infty} 2^{-2i} = 4/3$, in A?
c) Is A decidable?

In each case, argue why! Simple yes or no is not enough.

Answer:

a) Countably infinite. All languages are countable. As strings of form 0^+ are in A, A cannot be finite.
b) No, the given w is not a string but an infinite sequence.
c) A is decidable as it is straightforward to construct a TM that decides on it:

$M = M(w)$:

(a) First check the format and halt on `reject` if the input string w does not correspond to a binary number (with at most one binary point).
(b) Next compute the square of the input, which is also a binary string with a possible binary point
(c) Then compare the result to $8 = (1000.000)_2$, and halt accordingly.

All steps are guaranteed to finish and we have a decider for A.

4. Let $A = \{\langle R, S\rangle \mid R \text{ and } S \text{ are regular expressions and } L(S) \subseteq L(R)\}$. 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 $L(S)$*

(c) *Construct a DFA M_2 for $L(S) \cap L(R)$*

(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 $L(S) \subseteq L(R)$ then $L(S) \cap L(R) = L(S)$.

Problems:

1. Let A, B and C denote three languages with alphabet Σ. Suppose f and g are two (poorly chosen) computable functions with the following properties:

   a) For $w \in A$, either $f(w) \in B$ or $g(w) \in C$ (or both).
   b) For $w \notin A$, $f(w) \notin B$ and $g(w) \notin C$.

   Show that A is also decidable.

   Answer:

   Let M_B and M_C be deciders for B and C, respectively. We construct a decider for A:

   $M = M(w)$:

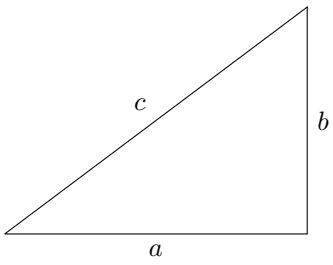
   (a) Run M_B with input $f(w)$; If M_B accepted, halt on `accept`
   (b) Run M_C with input $g(w)$; If M_C accepted, halt on `accept`
   (c) Halt on `reject`

   TM M halts on every input and it is thus a decider. Moreover, M will halt on `accept` *iff $w \in A$. It is thus a decider for A and A is decidable.*

2. Let us consider right triangles with sides of lengths a, b and c (see figure below). Language A is defined as follows:

$$A = \{\langle a, b\rangle \ : \ a, b \in \mathbb{N} \text{ and } \exists\, c \in \mathbb{N} \text{ such that } a^2 + b^2 = c^2\},$$

   where $\mathbb{N} = \{1, 2, 3, \ldots\}$ is the set of natural numbers. So for example $(a, b) = (3, 4)$ is in A because $3^2 + 4^2 = 5^2$. Show that A is decidable.



   Answer:

   The condition is thus that

$$a^2 + b^2 = c^2,$$

   for some $c \in \mathbb{N}$. Let us construct a decider for A:

   $M = M(\langle a, b\rangle)$:

   (a) Set $k = 1$
   (b) If $k^2 = a^2 + b^2$, halt on `accept`
   (c) If $k^2 > a^2 + b^2$, halt on `reject`
   (d) Increment k by one and go back to step 2

   TM M is clearly a decider for A, and thus A is decidable.

3. Suppose $\Sigma = \{0, 1\}$ and input strings correspond to binary numbers, e.g. 1011 corresponds to eleven. Let $f(x) = 3x + 1$. Show that $f(x)$ is a computable function by constructing a TM that implements it (an empty input string can be left as it is). Note that TM is suppose to leave the tape head to the start of the resulting string at the end of computation. **Submit a state diagram.**
Suggestion: use any TM simulator to verify your design.

Answer:

Instead of state diagram, we give a implementation level description:

$M = M(x)$:

(a) Halt on `accept`if input is ϵ
(b) Move the input string 3 positions to right
At the same time, write \$ to start of the tape, and two blank symbols after that
(c) Move right to the LSB, set carry $C = 1$ (in state), and start the multiplication
(d) While bits left (0 or 1 under tape head)
    i. $y = 3x_i + C$ (x_i is the bit under the tape head)
    ii. Write $y \bmod 2$ to tape, set $C = \lfloor y/2 \rfloor$, and move one left
(e) If $C = 1$, write 1 over the blank symbol
(f) If $C = 2$, write 0 over the blank symbol, move one left, and write 1 over the second blank symbol
(g) Move the binary string back to start of the tape
(cf. the \$ marker we placed there)

---

4. Show that EQ_{CFG} is undecidable.
Hint: Construct a reduction from ALL_{CFG}.

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, $L(G0) = \Sigma^*$.

Proof by contradiction: *Let R be a TM that decides EQ_{CFG}. We shall use it to construct a TM S that decides on 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.

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. Let $A = \{0^n 1^m \mid n, m \geq 0\}$ and consider TM M that decides on A by scanning the tape from left to right. In particular, M halts on `reject` immediately if the illegal pattern 10 is detected. Otherwise, upon reaching the end of the string, it halts on `accept`.

   a) What is the worst-case running time $f(n)$?
   b) What is the probability of accepting a random string W with length n given the uniform distribution of symbols (i.i.d.)?
   c) What is the average running time $g(n)$ in that case?

   Answer:

   a) The worst case running time happens when the string is accepted, i.e., all n symbols pass the condition. One scan across the tape takes $n+1$ steps (including the action related to the blank symbol at the end of the string),

   $$f(n) = n + 1.$$

   b) So here we assume that each symbol of the string is chosen independently uniformly in random. Let us condition on the number of 0's,

   $$\mathbf{P}\{\textit{accept}\} = \sum_{i=0}^{n} \underbrace{\left(\frac{1}{2}\right)^i}_{\textit{zeroes}} \underbrace{\left(\frac{1}{2}\right)^{n-i}}_{\textit{ones}} = \frac{n+1}{2^n}.$$

   Note that the above is valid also for the trivial cases $n = 0, 1$.

   c) Using the above, we can write

   $$g(n) = \sum_{i=0}^{n-2} \underbrace{\frac{i+1}{2^i}}_{\textit{first } i \textit{ ok}} \cdot \underbrace{\left(\frac{1}{2}\right)^2}_{\textit{10 pattern}} \cdot \underbrace{(i+2)}_{\textit{time to halt}} + \underbrace{\frac{n+1}{2^n} \cdot (n+1)}_{\textit{all ok (accept)}}$$

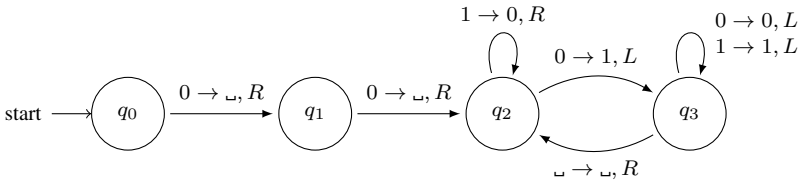
which reduces into

$$g(n) = 4 - (n+3)/2^n.$$

Also this expression holds for the trivial cases when $n < 2$.

Remark: *$g(n) < 4$ for all n, whereas $f(n)$ increases linearly to the infinity.*

3. Consider the following TM M with input alphabet is $\Sigma = \{0\}$. The machine "thinks hard" before accepting every string, i.e., $L(M) = \Sigma^*$ and **the omitted transitions lead to the accept state.**



We are interested in its *exact* running time $r(n)$ for non-empty strings, $w = 0^n$ with $n > 0$.

a) Implement the TM for a simulator and tabulate the running times for $n = 1, \ldots, 7$.

b) Determine a rule for $r(n+1)$ as a function $r(n)$, which is of form (a linear difference equation),

$$r(n+1) = ar(n) + bn + c. \qquad (n > 0)$$

c) Then solve $r(n)$. (Determine the exact closed-form expression for it).

Answer:

It is easy to see that $r(n+1) = 2r(n) + n - 4$ for all these cases.
(The next step would be to argue that this really holds for any n).
The general solution to the homogeneous equation $r(n+1) = 2r(n)$ is obviously $r(n) = A \cdot 2^n$, where A is some constant. Then a special solution to the difference equation is $r'(n) = 1 - n$, as then $r'(n+1) = 1 - n - 1 = -n$ and also

$$2r'(n) + n - 2 = 2 - 2n + n - 2 = -n.$$

Hence, the general solution is $r(n) = A2^n - n + 1$. Taking the initial condition (e.g.) that $r(1) = 2$ into account gives $A = 1$. Hence,

$$r(n) = 2^n - n + 1.$$

n	$r(n)$
0	*1*
1	*2*
2	*3*
3	*6*
4	*13*
5	*28*
6	*59*
7	*122*

```
Name:   slow
Init:   q0
Accept: qa

q0,_
qa,1,>
q0,0
q1,_,>

q1,0
q2,_,>
q1,_
qa,_,<

q2,1
q2,0,>
q2,0
q3,1,<
q2,_
qa,_,<

q3,0
q3,0,<
q3,1
q3,1,<
q3,_
q2,_,>
```

4. Let $G = (V, E)$ be a directional graph where nodes correspond to taxi 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"*, $\Delta t = 1$. 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 taxi with unit capacity, i.e., it can complete a single request at a time. Initially the taxi is at node v_0, the graph G is guaranteed to be connected, each edge takes unit time to travel ($\Delta t = 1$), 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!*

Problems:

1. 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.

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:

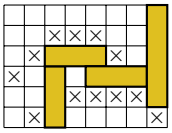
   TM $M = M(w)$:

   (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. Let us consider a board game called the Geothermal Park. This game is played on $n \times m$ board. The game master first blocks some squares by placing mud pots to them. The player is then given a set of k construction blocks with sizes $1 \times h_i$ that can be rotated. The task is to build a continuous path ("bridge") using non-overlapping blocks through the geothermal area. It is not required to use all blocks. The path must start from a square at the bottom and it must end to the top row (see picture below for a 8×6 board game).

   a) Formulate the corresponding language PARK.
   b) Show that PARK is in NP.
   c) Show that PARK is NP-complete.



Answer:

a)

$$\text{PARK} = \{\langle B, h_1, \ldots, h_k \rangle \ : \ \textit{path across the board exists}$$

i.e., on condition that such a placement of blocks exists that a path across the board is realized in a way that no block is placed on top of a mud pot. Parameter B describes the board and its mud pots.

b) Let certificate c desribe a feasible placement of some of the blocks AND a path across the board (for simplicity; it would be easy to determine this path also). It is clearly possible to check in polynomial time if (i) placement is feasible and (ii) the given path is valid (from bottom to top, using the given blocks). Thus a deterministic verifier exists and the language is in NP.

c) Reduction from SUBSET-SUM. *Let $x_1, \ldots, x_k$ denote the integers of the subset sum problem, and S the target sum. Then construct the corresponding* PARK *problem $\langle B, h_1, \ldots, h_k \rangle$ as follows:*

- *Construction blocks are $1 \times x_i$, $i = 1, \ldots, k$*
- *Board is $1 \times S$, and has not mud pots.*

Clearly $\langle B, h_1, \ldots, h_k \rangle \in$ PARK iff $\langle S, x_1, \ldots, x_k \rangle \in$ SUBSET-SUM (i.e., we have established a mapping reduction). The reduction (construction the instance of PARK *problem) runs in polynomial time, and therefore we have a polynomial time reduction,* SUBSET-SUM $\leq_P$ PARK. *We already showed that* PARK *is in NP, and thus it is NP-complete.*

4. Several different vaccines against a new COVID-19 variant are going to be available soon, but their effectiveness is unknown. A company called METO has been offered m different vaccination programs (=different vaccines). They have nominated Thomas Gardner as their chief medical officer (CMO). His task is to design a vaccination scheme for their n employees, where each employee may take (at most) one vaccine due to safety regulations. Thomas identifies k critical project teams $G_1, \ldots, G_k$ (a team is a subset of employees) in the company that should remain operational under all circumstances (a team is considered to be operational if at least one member is healthy). Each employee may belong to several teams, i.e., $G_i \cap G_j$ is not necessarily an empty set. Thomas defines the so-called *diversity criteria*, which states that each critical project team G_i should have members from three vaccination programs. The fundamental question is if it is possible to treat the n employees with the m vaccines in such a way that the diversity criteria is satisfied. The CMO realizes that the same question is asked everywhere, and wants to get rich by designing a Turing machine for it! Unfortunately, he has never studied TÖL301G, and thus asks your help!

   a) Formulate the corresponding language COVID.
   b) Show that COVID is in NP.
   b) Show that COVID is NP-complete. (Hint: consider a reduction from 3COLOR)

Answer:

Let $\langle m, n, G_1, \ldots, G_k \rangle$ denote a string describing a problem instance. Let T denote a vaccination scheme, $T = (t_1, \ldots, t_n)$ where all $t_i \in \{1, \ldots, m\}$. We say that a scheme is feasible if it satisfies the diversity criteria, that is if each set

$$T_i = \bigcup_{i \in G_i} \{t_i\},$$

has a sufficient number of elements, $|T_i| \geq 3$.

$$\text{COVID} = \{\langle m, n, G_1, \ldots, G_k \rangle | \text{ a feasible vaccination scheme exists}\}$$

A valid scheme T serves as a certificate for a deterministic TM that can verify the given solution in polynomial time simply by verifying the size of each set T_i (k groups, each group has at most n members). Therefore COVID *is in NP.*

A reduction from 3COLOR*: For each vertex v_i in graph G of* 3COLOR*, we add a new employee e_i. If vertices v_i and v_j are neighbors, then we add a group $G_{ij} = \{i, j\}$. Each group should have members with at least three different vaccines. Next we add a single manager to set of employees, who is member of all groups. Hence, each group has exactly 3 members, and a valid vaccination scheme means that they all must be given a different vaccine. Finally, we set $m = 4$, i.e., there is exactly 4 vaccines (=colors) available, of which one is reserved for the manager. The remaining three are for the employees.*

A valid vaccination scheme maps to a valid 3-coloring, and vice versa. Given constructing the instance of COVID *problem runs in polynomial time, we have* 3COLOR $\leq_p$ COVID*, and as* 3COLOR *is a well-known NP-complete problem, also* COVID *is NP-complete.* □

Problems:

1. Let us consider the DPLL algorithm with

   $$\Phi = (x_1 \vee \bar{x}_4 \vee x_3) \wedge (\bar{x}_1 \vee x_2 \vee \bar{x}_4) \wedge (\bar{x}_3 \vee \bar{x}_4 \vee x_2) \wedge (\bar{x}_2 \vee x_3 \vee x_4) \wedge (x_4) \wedge (\bar{x}_1 \vee \bar{x}_3 \vee \bar{x}_4).$$

   a) Carry out the unit clause propagation step for one (or more) unit clauses.
   b) Identify pure literals, set them true, and simplify the expression further.

   Answer:

   a) (x_4) is a unit clause and hence x_4 has to be true. Expression simplifies to

   $$\Phi = (x_1 \vee x_3) \wedge (\bar{x}_1 \vee x_2) \wedge (\bar{x}_3 \vee x_2) \wedge (\bar{x}_1 \vee \bar{x}_3).$$

   b) Now x_2 is a pure literal, so we can *set x_2 also to true, thus removing the clauses where it is present,*

   $$\Phi = (x_1 \vee x_3) \wedge (\bar{x}_1 \vee \bar{x}_3).$$

   (the above can be easily solved, but that was not asked.)

2. Consider a simple model for an airplane that consists of multiple components (sub-systems). The airplane has three engines, for which we define three boolean variables (e_1, e_2, e_3) indicating if a particular engine is functional or not. Similarly, airplane has two pilots, corresponding to p_1 and p_2, n navigational aids $a_1, \ldots, a_n$, as well as, numerous other sub-systems with respective variables. (often duplicated as redundancy is the key element for improved safety.)

   Let F denote some complicated boolean expression involving all these variables. In particular, F is true when the airplane can still fly safely(!). F should include clauses such as $(e_1 \vee e_2 \vee e_3)$ and $(p_1 \vee p_2)$. Note that the engines provide electricity to some other sub-systems, so these variables appear also in other clauses of F. By using F (which exact form is unspecified at this point) and the above variables, write down a boolean expression for the airplane being safe to fly for when

   a) The main engine 1 is out-of-order (i.e., e_1 is false).
   b) Engines 2 and 3, and one pilot, are out-of-order

   Hint: The failures are extra conditions imposed on top of F, there is no need to modify F (cf. how DPLL proceeds). That is, determine such $\phi = F \wedge (\text{something})$ that can be fed to a SAT solver to determine if the plane is still airworthy in the given scenario according to the safety model F.

   Answer:

   So F includes all the necessary information that specify the states when plane is considered safe to fly. We obviously home that the engineers did their job and $F \in SAT$.

   a) Let us impose the "requirement" that exactly one engine is out of order, $E_1 = (\bar{e}_1)$ and the answer is $F \wedge E_1$.
   b) Now $E_2 = \bar{e}_2 \wedge \bar{e}_3 \wedge (p_1 \vee p_2) \wedge (\bar{p}_1 \vee \bar{p}_2)$ and the answer is $F \wedge E_2$.

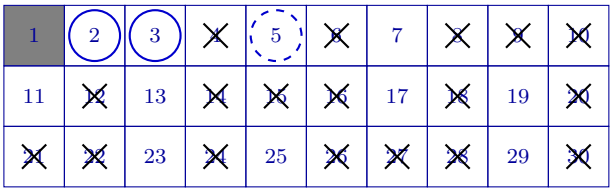
   The next step would be to feed the above expressions to company's SAT solver in order to convince the authorities about the air worthiness of the plane!

3. Suppose the TM M decides on primality (if x is a prime or not) using the sieve of Eratosthenes. The input string is a binary number, and the sieve stores its data so that one tape cell corresponds to one number ($i = 2, \ldots, x$).

a) Does M run in polynomial space?

b) What is the space complexity if the input is a unary number?

Answer:



Eratosthenes sieve. The new prime is 5 and the next step is to cross out its integer multiples.

a) To check whether p is prime, we need an array with length p. The maximum value of n bit number is $2^n - 1$, so the space (cells of the tape) needed increases exponentially, not polynomially.

b) With unary input, the size of the array is basically of the same length. So $f(n) = O(n)$. This runs in polynomial space.

4. Let us consider language A,

$$A = \{w \,:\, |w| \geq 4\}.$$

A probabilistic Turing machine M works as follows:

PTM $M(w)$:

(a) Move tape-head N times right

(b) Halt on `reject`, if the current symbol is blank; otherwise halt on `accept`.

The parameter N above is a random variable. Ideally, $N = 3$ always (i.e. a constant) and PTM M would do the perfect job. However, this Turing machine was cheap, and $N = 2$ with probability $1/2$, and otherwise $N = 3$.

a) Give a bound for $\mathbf{P}\{\texttt{accept}\}$ when $w \in A$

b) Give a bound for $\mathbf{P}\{\texttt{reject}\}$ when $w \notin A$

Answer:

So this TM checks the symbol either at position 3, or at position 4.

a) If $w \in A$, then both positions have a valid symbol and $\mathbf{P}\{\texttt{accept}\} = 1$.

b) In contrast, when $w \notin A$, it is possible that position 3 still has a valid symbol. Therefore, with certainty, we can only say that $\mathbf{P}\{\texttt{reject}\} \geq 1/2$.