

PROBLEM SET 10

MARC PICKETT I

- (1) Write all the 4 letter strings that are elements of the language described by the regular expression: $A*B*(AB)^*$

Answer: There are 16 possible 4 letter strings:

AAAA AAAB AABA AABB ABAA ABAB ABBA ABBB

BAAA BAAB BABA BABB BBAA BBAB BBBA BBBB

Of these, the following are in the language:

AAAA AAAB AABB ABAB BBAB BBBB

- (2) Consider a Finite State Automaton that has 3 states: 1, 2, and 3. State 1 is the start state, and state 3 is the accepting state. The state transitions are as follows:

In state	and reading	goto state
1	A	2
1	B	3
2	A	1
2	B	3
3	A	1
3	B	3

- (a) Write a regular expression that has the same language as this FSA.

Answer: If we ignore the transition from state 3 to state 1, then our regexp would be $A*BB^*$, but since we have the option of looping back to 1 with an A, we get $A*BB*(AA*BB)^*$.

Alternatively, we can note that a B will always transfer you to 3, the accepting state, and an A is never undefined. Since we'll need at least one B to get to state 3, this means that any string that ends with B (and only strings that end with a B) will be accepted. So we get $(A|B)^*B$, which has the same language as the regular expression above.

- (b) Write the transitions for a 2 state FSA that has the same language as the 3 state FSA above. Be sure to say which states are accepting states, and which state is the start state.

Answer: We can do this by noting that we can merge states 1 and 2. State 1 would be our start state, and state 3 would be the accepting state.

In state	and reading	goto state
1	A	1
1	B	3
3	A	1
3	B	3

- (3) Construct a Turing machine that takes in a string of As and Bs (followed by empty tape), and writes a B at the right of the string (the halts) if there are an even number of As, and writes an A if the number of As in the string is odd. You can assume that the reader head is at the leftmost character of the string.

Answer: The Turing machine will need only 2 states. One that says that the number of As read so far is even, and one that says it's odd. We'll start in state 1, which means we have an even number of As. The rules would look like this:

In state	and reading	write	move	and goto state
1	A	A	Right	2
1	B	B	Right	1
1	empty	B	Right	halt
2	A	A	Right	1
2	B	B	Right	2
2	empty	A	Right	halt

- (4) Count Vlad Urr gives Ossub Ull, the court Imp (not to be confused with Ido Urr, the court Gog), the task of writing the function `boolean printsA(Program P)` that takes an arbitrary program P, and in finite time returns whether P (when executed) prints the letter A. Can you prove that Imp Ossub Ull's task is impossible?

Answer: We can use proof by contradiction to prove this impossible. Suppose we had a *working* function `boolean printsA(Program P)` that did what Vlad wants it to do. We can construct the program named M

M:

```
if printsA(M)
```

```

    exit
else print A

```

Thus, if `printsA` predicts that `M` will print an `A`, then `M` won't print an `A` and vice versa. This is in direct contradiction with our assumption that `printsA` is working, and therefore it's impossible to construct a working `printsA`.

- (5) BONUS: Upon being again countered, Count Urr orders that *Gog Ido Urr go summon The High Programmer*. Upon arrival, The High Programmer states “This might not be as hard as *I think*. *Therefore, I am* inclined to say that Imp Osub Ull's task is possible if the input `P` will never have more than 256 lines of code and never use more than 24 bits of memory (including registers).”. With these restrictions on `P`, can one construct `printsA`? If so, how would `printsA` work? (An outline of the algorithm is sufficient.) If not, please explain why.

Answer: There are “only” 2^{24} possible memory states for `P`, and only 256 ($=2^8$) positions for the program counter to be in. This gives a total of $2^{32} = 4,294,967,296$ total possible states for `P`. `printsA` would create an array of 4,294,967,296 bits and set them all to 0. Then `printsA` would simulate running `P`, and change these bits to 1 as `P` entered the respective states. If `P` ever prints an `A`, then `printsA` would return `true` and halt. Otherwise if `P` enters a state that's already marked 1 (without printing `A`), then `printsA` would return `false` and halt. At worst, `printsA` will go through less than 4.3 billion states before halting.

Note that the proof by contradiction above no longer applies because `M` uses well over 24 bits of memory.