Evan Edelstein
EN.605.621.84.FA25
HW 6
The work in this exercise is mine alone without un-cited help. No AI was used to answer these questions.

1) We define the compliment of a language L to be $\bar{L} = \sum{}^{*} - L$, that is all combinations of the symbols in the language except those in L. We would like to show that if $\bar{L}$ is undecidable then L is also undecidable. We say a language is undecidable if there are no algorithms A that accepts all strings (including empty string) in L, and rejects all strings not in L. Note that in all cases A returns a value.

For the sake of contradiction, let there be a language L which is decidable but has a compliment that is undecidable. We can construct an algorithm to decide its compliment, $\bar{L}$ by flipping the answer given by an algorithm that decides L. So, if an algorithm A (that decides L) rejects an input s in $\bar{L}$, we accept s, and if it accepts s, we reject. This contradicts the assumption that $\bar{L}$ is undecidable. Thus, if $\bar{L}$ is undecidable, L is undecidable as well.

2)
$s^n = s\ concatanated\ with\ itself\ n\ times$

Language 1: $L1 = \{a^n b^n \mid 0 \leq n \leq 1000\}$
L1 is a finite set of languages composed of n 'a's followed by n 'b's. Since it is a finite set it can be decided by a finite-state automaton, so it is a regular language and type 3.

Language 2: $L2 = \{a^n b^n \mid n \geq 0\}$
L2 is like L1, except it is an infinite set. Since it is unbounded and requires that an equal number of 'b's follow an equal number of 'a's we cannot construct a FSA that determines if a string is in L2. We could use a stack to hold the number of 'a's and then pop from the stack when a 'b' is seen. In this way we could determine if a string is in the language by ensuring that the stack is empty right after we finish reading the string. Therefore, L2 requires a pushdown automaton (FSA + stack) which makes it a context-free or type 2 language.

Language 3: $L3 = \{a^n b^m \mid n, m \geq 0\}$
L3 is like L2 but does not require that there are an equal number of 'a's and 'b's, so we can represent this language with a finite state automaton that ensures we start at 'a', 'a's transition to 'a' or 'b' and 'b's transitions to only 'b'. Therefore, L3 is decided by FSA so it is regular and type 3. A depiction of this FSA is shown below.

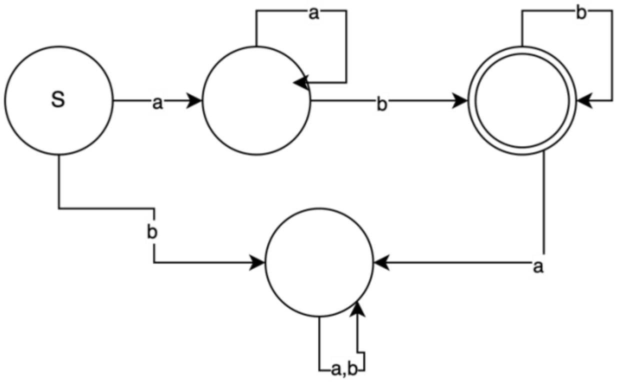

Figure 1: FSA to decide L3.

3)
a) Since the language provided is finite over all 3 operations, we can construct an FSA that can represent this language and decide if an input is in it. Therefore, it is a regular language and type 3.

b) This language will always halt since there are no unbounded strings or infinite recursive rules generated by the 3 operations. Each operation only expands the language by a finite number of non-terminal states. (also, it says it will always accept or reject an input in the question).

c) We can compose the 3 wildcard operations using the 3 basic operations. '.' can be rewritten as the concatenation of *a*, an alternation of all the single letter symbols ($\sum$), and *c*.

$$a.c \rightarrow a\ U \sum U\ c \rightarrow\ a\ (\ a\ U\ b\ U\ c\ \ldots)\ c$$

The '*' wildcard expression a*c can be written as the concatenation of *a*, $\Sigma^*$and *c*, since Σ^* includes the empty string as well.

$$a * c \rightarrow a\sum^* c$$

The a{m,n}c matching can be rewritten by alternating all the Kleene stars of the single symbol *a* up to from m to n times.

$$a\{m,n\}c \rightarrow (a\ U\ aa\ U\ aaa\ \ldots)c$$

Since we can show that these operations can be decomposed to regular operations, the regularity of the language is maintained. Therefore, it is still a type 3 language, and we can be certain that this language will halt on any input.

d) Drawing out the production rule for the backreference operator shows that it requires recursion and fits into the rubric of a context-sensitive grammar. Specifically, the rule for matching a backreference would look like:

$$aAnb \rightarrow aAA(n-1)b \quad |\ when\ n > 0$$
$$aAnb \rightarrow aAb \qquad\qquad |\ when\ n = 0$$

Where *a* and *b* represent a string of terminals and non-terminals, *A* being a string of terminals to match, and *n* being the number of matches. To decide on this extended version of regular expressions we require a machine that has a memory hierarchy to store and compare strings of arbitrary length, and a system that supports recursion to use the production rules above. This points to the language requiring a Linear Bounded Automata and therefore is a context-sensitive language. Since *n* only decreases, we have a finite number of elements to match. Since *A* only contains terminals, there will be no recursive backreferencing, and therefore no infinite recursion. Therefore the program will halt.

Citations:

[1]https://www.youtube.com/watch?v=p1peR1Qbp0s&list=PL1BaGV1cIH4WFbOgfefWr3nXCEIeqp6br&index=5

[2] https://www.youtube.com/watch?v=glTmP0IWff0