HW 06

Elizabeth Hunt (A02364151)

November 2, 2023

1 Problem One

1.1 1 - { a^i b^j c^k | i > j > k }

We assume that there exists a Context Free Language, L such that $L = a^i b^j c^k | i > j > k$. Then by the Pumping Lemma for Context Free Languages, for each $z \in L$ of sufficient length n we can split z into the sequence of substrings $uvwxy$ with $|vx| \geq 1$ and $|vwx| \leq n$. Any such "pumped" string $uv^i wx^i y \forall i \geq 0$ is also in L.

Consider the string $a^{n+2}b^{n+1}c^n \in L$ which is always longer than n (the integer of the Pumping Lemma) so it is of sufficient length. Splitting this up according to the aforementioned Lemma, we know $|vwx| \leq n$ and thus the string vx contains either all of one letter or two different letters.

If vx contains only a's or b's then we can "pump" vx 0 times to produce $z^* = uv^0wx^0y$ and since $|vx| \geq 1$ then we lose $p \geq 1$ a's or b's. Thus, in the case of a, $(n+2) - p \leq n+1$ and z^* is not in L, and in the case of b, $(n+1) - p \leq n$, with the same conclusion.

If vx contains only c's then we can "pump" vx n times to produce $z^* = uv^n wx^n y$ at least $2n$ the number of $c's$ (which is always $\geq n$), and there would be equal or more c's to b's and thus $z^* \notin L$.

If vx contains both a's and b's then we can "pump" vx 0 times similar to the case of only a's and b's to produce a string with an equal or less number of b's than c's.

If vx contains both b's and c's then we can "pump" vx $3n$ times to come up with a string that will always have b's and c's greater or eqaul to the number of a's.

1.2 2 - { a^i b^j | i = j^2 }

We assume that there exists a Context Free Language, L such that $L = a^i b^j | i = j^2$.

Then by the Pumping Lemma for Context Free Languages, for each $z \in L$ of sufficient length n we can split z into the sequence of substrings $uvwxy$ with $|vx| \geq 1$ and $|vwx| \leq n$. Any such "pumped" string $uv^i wx^i y \forall i \geq 0$ is also in L.

Consider the string $a^{n^2}b^n \in L$ which is always longer than n (the integer of the Pumping Lemma) so it is of sufficient length. Splitting this up according to the aforementioned Lemma, we know $|vwx| \leq n$ and thus the string vx contains either all of one letter or two different letters.

1. If vx contains only a's then we can "pump" vx 0 times to produce $z^* = uv^0wx^0y$ and since $|vx| \geq 1$ then we lose $p \geq 1$ a's and thus $z^* = a^{n^2-p}b^n \notin L$ as $n^2 - p \neq n^2$.

2. If vx contains only b's then we can "pump" vx 2 times to produce $z^* = uv^2wx^2y$ and since $|vx| \geq 1$ then we add $p \geq 2$ b's and thus $z^* = a^{n^2}b^{n+p} \notin L$ as $n^2 \neq (n+p)^2$.

3. If vx contains both b's and a's then we can prove that we can construct a number of times to pump such that the relationship for $i = j^2$ cannot be satisfied:

- v must contain only a's and x must contain only b's; else we could pump and obtain a string in which the order of a's and b's is disrupted.
- Let $y = |v|$ and $u = |x|$ with $1 \leq y + u \leq n$ since $|vwx| \leq n$ and $|vx| \geq 1$. Then we can pump $p \geq 1$ times to create the string $z^* = a^{(n^2-y)+py}b^{(n-u)+pu} = a^{n^2+p(y-1)}b^{n+p(u-1)}$.
- Then for any such n, u, y we can find a p such that $(n + p(u-1))(n + p(u-1)) = n^2 + 2pn(u-1) + p^2(u-1)^2 \neq n^2 + p(y-1)$ since u and y are positive.

1.3 3 - { a^i | i is prime }

We assume that there exists a Context Free Language, L such that $L = a^i b^j | i = j^2$.

Then by the Pumping Lemma for Context Free Languages, for each $z \in L$ of sufficient length n we can split z into the sequence of substrings $uvwxy$ with $|vx| \geq 1$ and $|vwx| \leq n$. Any such "pumped" string $uv^iwx^iy \forall i \geq 0$ is also in L.

Consider the string $z = a^{p(n)} \in L$ where $p(n)$ is a function returning the first prime number greater than n; this string is always longer than n (the integer of the Pumping Lemma) so it is of sufficient length. Splitting this up according to the aforementioned Lemma, we know $|vwx| \leq n$ and thus the string vx is made up of one or more a's.

Given $t = |v|$ and $u = |x|$, and $1 \leq t + u \leq n$; we can simply pump vx $p(n)$ times to obtain the string $z^* = uv^{tp(n)}wx^{up(n)}y$ which is $a^{p(n)+tp(n)+up(n)}$. And $p(n) + tp(n) + up(n)$ is not prime since it can be factored into $p(n)(1 + t + u)$; as either t or u is greater than one, then $z^* \notin L$.

2 Problem Two

`https://bit.ly/cs5000-simponic-hw06-p02`

3 Problem Three

`https://bit.ly/cs5000-simponic-hw06-p03`

In english(ish):

1. If the current symbol is B, move right and goto 2
2. If the current symbol is a, b, c, move right and goto 2. If the current symbol is B, move Left and goto 3. If the current symbol is * we move left and goto 6.
3. We're at the last c in the string. If the current symbol is c then we print \$ and go left until we see an a. Then, move Left and go to 4.
4. If the current symbol is b then we print * and go right until we see \$. At which we move left and goto 3. If the symbol is * then we move right and goto 5.
5. If the current symbol is b, c, *, we move right and goto 5. If it's \$ we move left and goto 2 to do the next iteration.
6. We move left until we hit B at which we move right and goto 7.
7. We're now at the beginning of the a's after replacing all c's with \$ and b's with *. If we read an a we print + and goto 8. If we see a $-$ (which we replace $*$'s with), we accept.
8. If the current symbol is $+, a, *$, we move right and goto 8. If we see \$ or $-$ we move left and goto 9.

9. If the current symbol is $*$ then we print $-$ and goto 9. If we see $-$ we move left and goto 10.

10. If the current symbol is $*, a$ then we move left and goto 10. If we see $+$ we move right and goto 7 to do another iteration.

4 Problem Four

We can write the control tuples for the two-tape turing machine in the form: $(q_{\text{from}}, s_1, s_2, a_1, a_2, q_{\text{to}})$ where s_n is the expected symbol on tape n, a_n is the action (R / L / new symbol) to perform on tape n, q_{from} is the current state, and q_{to} is the next state.

At a high level we can translate these control tuples to a single tape by separating the single tape into two sections; the first section simulates the first tape, and the other the second tape. These can be split by a single symbol `+` not in the alphabet (with the first cell also being `+`, representing the beginning of the input on the first tape). Additionally, the position of the tape head of the turing machine being simulated in its section can be delimited by some symbol not in the input alphabet in each section; call this symbol `*`.

Thus, to build the initial tape we can construct the tape `+ * <input on tape 1> + * <input on tape 2>` (the head being at the beginning). Then, for each tuple in the two-tape turing machine $(q_{\text{from}}, s_1, s_2, a_1, a_2, q_{\text{to}})$ we construct a set of new states that will:

1. Search until we find `*` (we're at the head of the first "tape"), at which we go right one, and if we read s_1 goto 2.

2. Search until we find `*` (we're at the head of the second "tape"), at which we go right one, and if we read s_2 goto 3.

3. We perform a_2.

   - If s_2 is the `+` symbol, we perform a new set of states to analyze the next symbol after; if it's non-`+` we shift all symbols proceeding to the right (this is a catch for 4 in the case we need to overwrite the `+` cell delimiting the two sections) and print a blank symbol. Then continue.
   - If a_2 is "R" then we go left one (to the `*`), write s_2, and go right one.
   - If a_2 is "L" then we go back 2 (to the symbol preceding `*`) and from a new set of states constructed from all symbols in the alphabet go right and copy that symbol. Then we go back one and print "*".
   - Else, we print a_2.
   - Finally, go left and goto 4.

4. Move left until we find `*` (we're now at the head of the first "tape") again and perform the same algorithm as 3 but with $s_2 \to s_1, a_2 \to a_1$. Then goto 5.

5. Move left until we find the `+` symbol (we're at the head of the first tape) and move to the state corresponding to 1 with q_{to}.

5 Problem Five

We can write the control quadruples for the 3D turing machine in the form: $(q_{\text{from}}, s, a, q_{\text{to}})$ where s is the expected symbol and a is the movement action $\{+x, -x, +y, -y, +z, -z\}$ or a symbol to print; where each movement action is a single step on the specified axis by either positive or negative one.

At a high level we will apply our strategy in Problem 4 of delimination to construct several rows of tape into a "matrix" and then scale those into several matrices to produce a 3-dimensional tape; as one could emulate with `(make-array (list z y x))`.

Then if we construct a one dimensional tape such that each "row" is a "tape" in the z axis then we "paste" the input into a one dimensional tape in the form `- = + * _ <...[(0, 0, 0), (0, 0, 1)...]>` with the first five unique symbols not in the alphabet; the `-` indicating a new "matrix" in the z-y plane, and each `+` symbol a new row in the z direction. But now, we need to keep track of where we are in the z and y planes for every x. To do so we can also create another three new symbols not in the alphabet; say `*` to stick with Problem 4 for the z plane, `=` for the y plane, and `_` for the actual placement of the current head of the machine.

Now, for each control tuple of the 3D turing machine we can construct a new set of states that will, at a high level:

1. Search until we find `_` (we're at the head of the tape), at which we go right one, and if we read s goto 2.

2. If a is $\pm z$ then we move right or left.

   - If this symbol is `+` we must then traverse to the very beginning of the tape and "increase the size" of every single z since the tape has the potential to now go in the x or y axes; we do this by searching for the first `+`, move right, shift the entire tape right one and print a blank symbol (to add an extra cell at the beginning), then continue searching for `+` and for each: shift the entire tape right one, move left, print a blank symbol, move right two, shift everything to the right one, move left, and print a blank symbol (adding a blank cell before and after each `+`).
   - Then we go back to the beginning of the tape and continue moving right until we see `_`.

3. If a is $\pm y$ or $\pm x$ then move left or right until the first instance of either `=` or `-` respectively and perform a similar kind of algorithm to copy the rows or matrices with blank z rows of the same size, adding rows / matrices if necessary should we hit the beginning or end of the tape (for x) or a new "matrix" (`-` for y)..