William Fenton
CNC - Formal Languages
Winter Midterm

1. Use the pumping lemma to show that the set of strings over {a, b} in which the number of a's is a perfect cube is not regular.

To Prove: the set of strings over {a, b} in which the number of a's is a perfect cube is not regular.

Let L = the set of strings over {a, b} in which the number of a's is a perfect cube
Let k be the number from the pumping lemma
Let s = ababa = $a^{k-2}ba^{k-2}ba^{k-2}$, where k = 3

By the pumping lemma s = uvw where v ≠ **λ** and |uv| ≤ k

Since |uv| ≤ k, v must consist of aba | ba | a
Since v ≠ **λ**, v must consist at least one a

Case: u = **λ**, v = aba, w = ba

Suppose we pump
By the pumping lemma:

s = aabaaaabaaba, u = **λ**, v = aabaaaabaa

the number of a's = 9
9 is not a perfect cube

Case: u = a, v = ba, w = ba

Suppose we pump
By the pumping lemma:

s = abaabaaba, u = a, v = baabaa
the number of a's = 6
6 is not a perfect cube

Case: u = ab, v = a, w = ba

Suppose we pump
By the pumping lemma:

s =abaaba, u = ab, v = aa
the number of a's = 4

For all of the possible v's, s is not in L
Contradiction! L is not regular

2. A context free grammar G = (V, Σ, P, S) is called left-linear if each rule is of the form:

> **$A \rightarrow u$**
> **$A \rightarrow Bu$**

where A, B ∈ V and u ∈ Σ*.
Show that the left linear grammars generate precisely the regular sets.

S -> A
A -> Au | u

Since every non-terminal must end with a terminal value, we can make a regular expression:
A regular expression for this language would be u*u
Or we could build a DFA

State	**Input = u**
S1	S2
S2	S2

3. Let M be the PDA in Example 7.1.3.

(a) Give the transition table of M.

State	Input	New State	Pop	Push
q0	a	q0	λ	A
q0	b	q0	λ	B
q0	λ	q1	λ	λ
q1	a	q1	A	λ
q1	b	q1	B	λ

(b) Trace all computations of the strings ab, abb, abbb in M.

NOTE: Probably need to convert to an empty stack accepting machine

s = ab:
(q0, ab, {}) |- (q0, b, {A})
|- (q0, λ, {BA}) ; not accepting
s = abb:
(q0, abb, {}) |- (q0, bb, {A})
|- (q0, b, {BA})
|- (q1, b, {BA}); lambda transition
|- (q1, λ, {A}) ; error: stack is not empty
s = abbb
(q0, abbb, {}) |- (q0, bbb, {A})
|- (q0, bb, {BA})
|- (q1, bb, {BA}); lambda transition
|- (q1, b, {A})
|- (q1, λ, {A}); error reading B from stack

(c) Show that aaaa, baab ∈ L(M)

s = aaaa
(q0, aaaa, {}) |- (q0, aaa, {A})
|- (q0, aa, {AA})
|- (q1, aa, {AA}); lambda transition
|- (q1, a, {A})
|- (q1, λ, {}); accept
s = baab
(q0, baab, {}) |- (q0, aab, {B})
|- (q0, ab, {AB})
|- (q1, ab, {AB}); lambda transition
|- (q1, b, {B})
|- (q1, λ, {}); accept

(d) Show that aaa, ab ∉ L(M)

NOTE: Need to convert to empty stack accepting machine

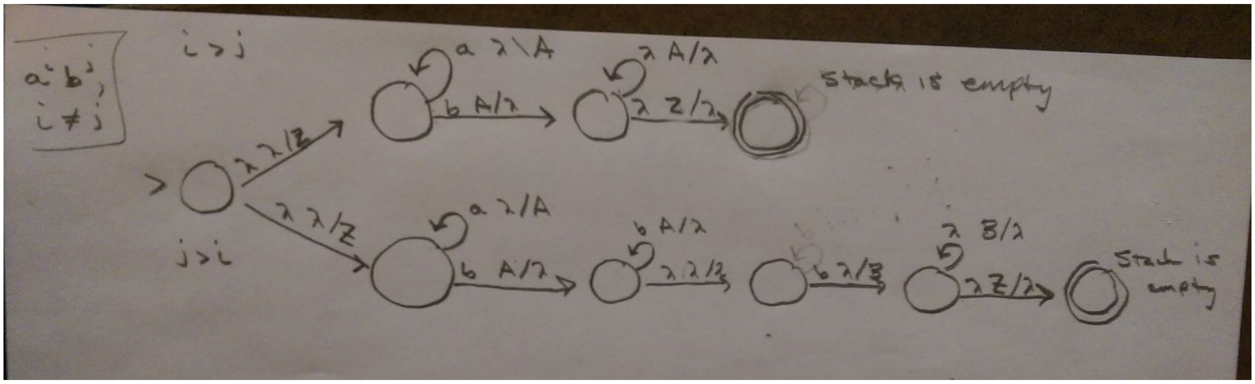
s = aaa

(q0, aaa, {})	|- (q0, aa, {A})
	|- (q0, a, {AA})
	|- (q0, **λ**, {AAA}); error, stack is not empty

s = ab

(q0, ab, {})	|- (q0, b, {A})
	|- (q0, **λ**, {BA}); error, stack is not empty

4. Construct a PDA that accepts the language $\{a^i b^j \mid i \neq j\}$.



5. Let L = {$a^{2i}b^i$| i ≥ 0}

(a) Construct a PDA M1 with L(M1) = L

State	Input	New State	Pop	Push
q0	a	q0	λ	A
q0	λ	q1	λ	λ
q1	b	q2	A	λ
q2	λ	q3	A	λ
q3	b	q2	A	λ

(b) Construct an atomic PDA M2 with L(M2) = L

State	Input	New State	Pop	Push
q0	a	q1	λ	λ
q1	λ	q2	λ	A
q2	a	q1	λ	λ
q2	b	q3	λ	λ
q3	λ	q4	A	λ
q4	λ	q5	A	λ
q5	b	q3	λ	λ

(c) Construct an extended PDA M3 with L(M3) = L that has fewer transitions than M1.

State	Input	New State	Pop	Push
q0	a	q0	λ	A
q0	λ	q1	λ	λ
q1	b	q3	AA	λ
q3	b	q3	AA	λ

(d) Trace the computation that accepts the string aab in each of the PDAs that you constructed.

M1:

(q0, aab, {}) |- (q0, ab, {A})
|- (q0, b, {AA})
|- (q1, b, {AA})
|- (q2, λ, {A})
|- (q3, λ, {})

M2:

(q0, aab, {}) |- (q1, ab, {})
|- (q2, ab, {A})
|- (q1, b, {A})
|- (q2, b, {AA})
|- (q3, λ, {AA})
|- (q4, λ, {A})
|- (q5, λ, {})

M3:

(q0, aab, {}) |- (q0, ab, {A})
|- (q0, b, {AA})
|- (q1, b, {AA})
|- (q2, λ, {})

6. Use the pumping lemma to prove that $\{ww^Rw | w \in \{a, b\}^*\}$ is not context free.

To Prove: $\{ww^Rw | w \in \{a, b\}^*\}$ is not context free.
Assume: L is a context free grammar

Let L = $\{ww^Rw | w \in \{a, b\}^*\}$
Let k be the number from the pumping lemma
Let s = abbaab

By the pumping lemma s = *uvwxy*

with substrings *u*, *v*, *w*, *x* and *y*, such that

1. $|vwx| \leq k$,
2. $|vx| \geq 1$, and
3. uv^nwx^ny is in L for all $n \geq 0$.

Since $|vwx| \leq k$, there are 4 possibilities for vwx:
u = **λ**, vwx = abb, y = aab
u = a, vwx = bba, y = ab
u = ab, vwx = baa, y = b
u = abb, vwx = aab, y = **λ**

Case 1:
Suppose we pump once
By the pumping lemma:
vwx = aabbb, s = aabbbaab
s is not in the language

Case 2:
Suppose we pump once
By the pumping lemma:
vwx = bbbaa, s = abbbaaab
s is not in the language

Case 3:
Suppose we pump once
By the pumping lemma:
vwx = bbaaa, s = abbbaaab
s is not in the language

Case 4:
Suppose we pump once
By the pumping lemma:
vwx = aaabb, s = abbaaabb
s is not in the language

Therefore, L is not a context free grammar

7. Construct a Turing machine (with no macros) to compute the following number-theoretic functions:

(a) even(n) = if n is even: 1, otherwise: 0

M = (Q, Σ, Γ, δ, q_0, F)
Q = {$q_0 q_1$}xF, F= {$q_a q_h$} where q_a is an accepting state, and q_h is a rejecting state
Σ = {1}, note: this is a unary number representation
Γ = Σ x B where B is the blank symbol on the tape

δ =

Current State	Current Symbol	New Symbol	Direction	New State
q0	B	B	R	qa
q0	1	1	R	q1
q1	B	B	R	qh
q1	1	1	R	q0

Example 1, input = 111:
$q_0 111 \Rightarrow 1q_1 11 \Rightarrow 11q_0 1 \Rightarrow 111q_1 B \Rightarrow 111Bq_h B$
111 is odd and the machine halts in state q_h signifying a rejection

Example 2, input = 11:
$q_0 11 \Rightarrow 1q_1 1 \Rightarrow 11q_0 B \Rightarrow 11Bq_a B$
11 is even and the machine halts in state q_a signifying a acceptance

(b) lt(n, m) = if n < m: 1, otherwise: 0

M = (Q, Σ, Γ, δ, q_0, F)
Q = $\{q_0,q_1,q_2,q_3,q_4,q_5\}$ x F, F= $\{q_a q_h\}$ where q_a is an accepting state, and q_h is a rejecting state
Σ = {1}, note: this is a unary number representation
Γ = Σ x {e,<,_} where '_' is the blank symbol on the tape and 'e' signifies the left edge of the tape

δ =

Current State	Current Symbol	New Symbol	Direction	New State
0	e	e	r	1
1	1	x	r	2
1	x	x	r	1
1	<	<	r	5
1	_	_	r	halt
2	1	1	r	2
2	<	<	r	3
2	_	_	r	halt
3	x	x	r	3
3	1	x	r	4
3	_	_	r	halt
4	x	x	l	4
4	1	1	l	4
4	<	<	l	4
4	_	_	l	4
4	e	e	r	1
5	x	x	r	5
5	1	1	r	accept
5	_	_	r	halt

Example 1: Input = e1<11
$q_0e1<11 \rightarrow eq_11<11 \rightarrow exq_2<11 \rightarrow ex<q_311 \rightarrow ex<xq_41 \rightarrow ex<q_4x1 \rightarrow exq_4<x1 \rightarrow$
$eq_4x<x1 \rightarrow q_4ex<x1 \rightarrow eq_1x<x1 \rightarrow exq_1<x1 \rightarrow ex<q_5x1 \rightarrow ex<xq_51 \rightarrow ex<x1q_{accept\_}$

Example 2: Input = e1<1
$q_0e1<1 \rightarrow eq_11<1 \rightarrow exq_2<1 \rightarrow ex<q_31 \rightarrow ex<xq_{4\_} \rightarrow ex<q_4x \rightarrow exq_4<x \rightarrow eq_4x<x \rightarrow$
$q_4ex<x \rightarrow eq_1x<x \rightarrow exq_1<x \rightarrow ex<q_5x \rightarrow ex<xq_{5\_} \rightarrow ex<x\_q_{halt\_}$

For each machine provide one example showing the action of your machine on a sample input.

Design a machine that computes: gt(n,m) = if n > m: 1; otherwise: 0

If we were doing greater than or equal (or or macro was for less than or equal), this would be easier, as we could just flip the sign on the input tape to <, then run the lt macro, and use the property of negation to map the macro accept to a rejecting state and rejecting state to a accepting state. But it's not that easy. We could also swap the numbers on the input tape then run the lt macro in a similar way, but that seems about as hard to program as starting from scratch. So I'm starting from scratch.

M = (Q, Σ, Γ, δ, q_0, F)
Q = {q_0,q_1,q_2,q_3,q_4,q_5} x F, F= {$q_a q_h$} where q_a is an accepting state, and q_h is a rejecting state
Σ = {1}, note: this is a unary number representation
Γ = Σ x {e,>,_} where '_' is the blank symbol on the tape and 'e' signifies the left edge of the tape

δ =

Current State	**Current Symbol**	**New Symbol**	**Direction**	**New State**
0	e	*	r	1
1	1	x	r	2
1	x	x	r	1
1	>	>	r	5
1	_	_	r	halt
2	1	1	r	2
2	>	>	r	3
2	_	_	r	halt
3	x	x	r	3
3	1	x	r	4
3	_	_	r	halt-accept
4	x	x	l	4
4	1	1	l	4
4	>	>	l	4
4	_	_	l	4
4	e	e	r	1
5	x	x	r	5
5	1	1	r	halt
5	_	_	r	halt

Example 1: Input = e11>1

q_0e11>1 → e$q_1$11>1 → ex$q_2$1>1 → ex1q_2>1 → ex1>$q_3$1 → ex1>xq_4_ →
ex1>q_4x → ex1q_4>x → ex$q_4$1>x → eq_4x1>x → $q_4$ex1>x → eq_1x1>x →
ex$q_1$1>x → exxq_2>x → exx>q_3x → exx>xq_3_ → exx>x_q_{accept}_

Example 2: Input = e> (0>0)

q_0e> → eq_1> → e>q_5_ → e>_q_{halt}_

9. Trace the actions of the machine MULT for computations with input:

(a) $n = 0$, $m = 4$

(b) $n = 1$, $m = 0$

(c) $n = 2$, $m = 2$

10. Let F be a Turing machine that computes a total unary number-theoretic function f. Design a machine to compute the function:

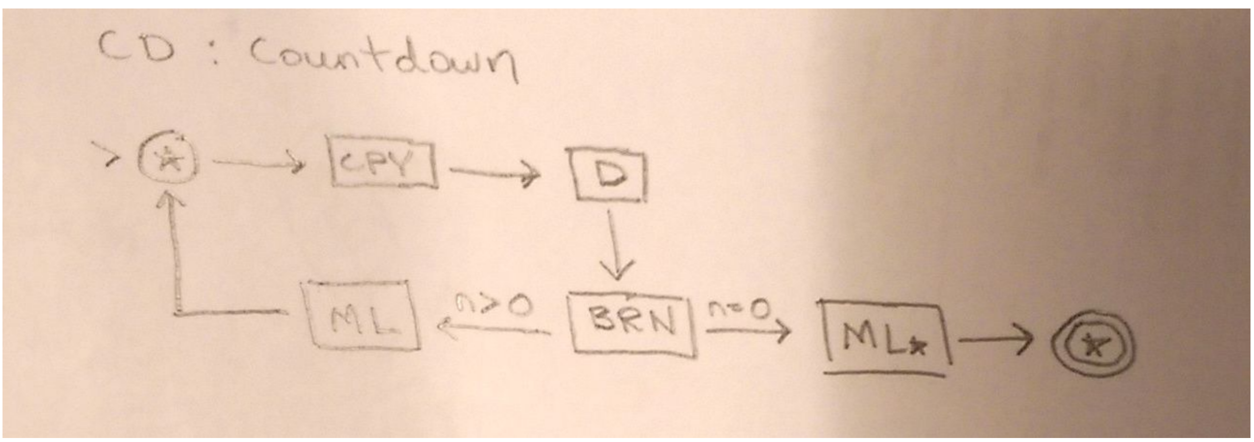
g(n) = ∑ i=0 : n f(i)

Using the following prebuilt macros:

CPY = copy 1 word and move the beginning of the copied word
D = decrement a unary number and move the the beginning of that word
BRN = Branch on Zero
ML = Move left 1 word
MR = Move right 1 word
ML* = Move all the way to the left boundary
FN = some given total unary number-theoretic function f

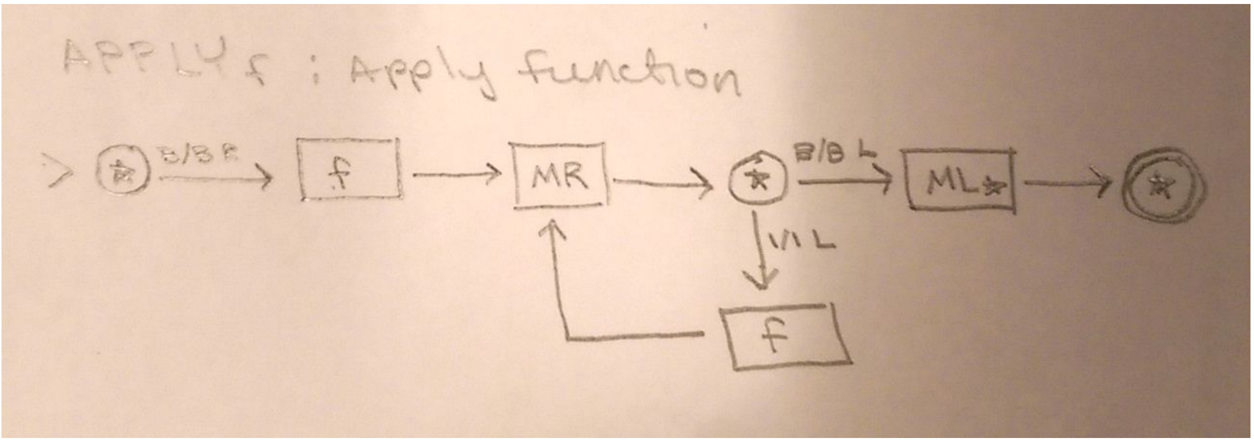
And the following custom macros:

CD = countdown. Takes a single unary n on the tape and writes n … pred(n) … 0 on the tape.
Example: 111→ 111 11 1



APPLY$_f$ = Apply function. Applies a function to each unary number on the tape and resets the tape head to the left boundary.

Example: Assume f = successor(x), _111 11 1 → _1111 111 11



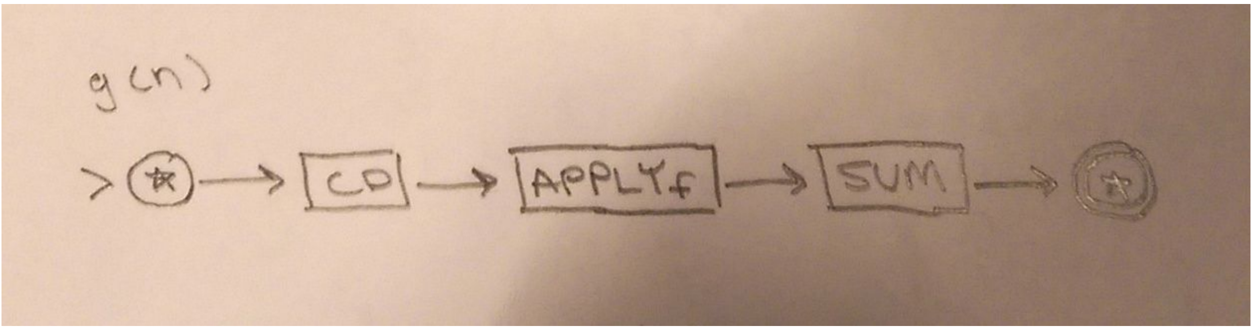
SUM = Summation function. Sums each unary number (separated by a blank) on the tape.

Example: _111 11 1 → _111111

Note: I programmed and tested this machine

```
; SUM macro
; Sums multiple unary numbers
0 _ _ r 1
1 1 1 r 1
1 _ 1 r 2
2 1 1 r 2
2 _ _ l 3
3 1 1 l 4
4 1 1 l 4
4 _ _ r 5
5 1 _ r 6
6 1 1 r 6
6 _ _ r 7
7 _ _ l 10
7 1 1 l 8
8 _ _ l 9
9 1 1 l 9
9 _ _ * 0
10 _ _ l 11
11 1 1 l 11
11 _ _ * halt
```

The final machine would be the composition of machines:
$g(n) = SUM(APPLY_f(CD(n)))$ or $g = SUM . APPLY_f . CD$


g(n)
CD
APPLY f
SUM

11. Let G be the context-sensitive grammar:

G : $\quad$ **S $\rightarrow$ SBA | a**
BA $\rightarrow$ AB
aA $\rightarrow$ aaB
B $\rightarrow$ b

(a) Give a derivation of aabb

S $\rightarrow$ SBA
SBA $\rightarrow$ aBA ;using S $\rightarrow$ A
aBA $\rightarrow$ aAB ;using BA $\rightarrow$ AB
aAB $\rightarrow$ aaBB ;using aA $\rightarrow$ aaB
aaBB $\rightarrow$ aabB ;using B $\rightarrow$ b
aabB $\rightarrow$ aabb ;using B $\rightarrow$ b

(b) What is L(G)?

By looking at successive applications of the recursive S $\rightarrow$ SBA|a rule:
0 applications: s = a = a^1b^0 (base case)
1 applications: s = aabb, a^2b^2
2 applications: s = aaabbbb = a^3b^4
3 applications: s = aaaabbbbbb = a^4b^6

Note: I can give you a screenshot of a page full of derivations, if need be.

It looks like the number of a's in the string equal the number of applications of the recursive rule plus one
It looks like the number of b's in the string equal the number of applications of the recursive rule times two
Where L(G) = $\{a^{n+1}b^{2n} \mid n \geq 0\}$

(c) Construct a context-free grammar that generates L(G)

G: $\quad$ S $\rightarrow$ aSA | a
A $\rightarrow$ BB
B $\rightarrow$ b

0 applications of the recursive rule S $\rightarrow$ aSA | a:
S $\rightarrow$ a = a^1b^0, matches
1 applications of the recursive rule S $\rightarrow$ aSA | a:
S $\rightarrow$ aSA $\rightarrow$ aaA $\rightarrow$ aaBB $\rightarrow$ aabb = a^2b^2, matches
2 applications of the recursive rule S $\rightarrow$ aSA | a:
S $\rightarrow$ aSA $\rightarrow$ aaSAA $\rightarrow$ aaaAA $\rightarrow$ aaaBBBB $\rightarrow$ aaabbbb = a^3b^4, matches
2 applications of the recursive rule S $\rightarrow$ aSA | a:
S $\rightarrow$ aSA $\rightarrow$ aaSAA $\rightarrow$ aaaSAAA $\rightarrow$ aaaaAAA $\rightarrow$ aaaaBBBBBB $\rightarrow$ aaaabbbbbb = a^4b^6, matches

I feel pretty confident that these generate the same language.

12. Design a two tape TM that determines if two strings u and v over {0, 1} are identical. The computation begins with BuBvB on the tape and should require no more than 3(length(u) + 1) transitions. (The limitation is guidance only: if you have a longer one, don't worry. But consider this lower bound.)

```
//2-Tape Compare String

//-------DELTA FUNCTION:
//[current_state],[read_symbol_top][read_symbol_bottom],[new_state],[write_symbol],[>|<
|-]

// < = left
// > = right
// - = hold
// use underscore for blank cells

// Input: 2 binary strings separated by a space _
// Ouput:
// Example: accepts 10101 10101, rejects 111 010
//
// --------- States ---------------------------|
// qCopy - copy 1st word to second tape        |
// qLeft - move to left boundary               |
// qMoveR - move first tape to next word       |
// qCheck - compare first tape and second tape |
// qReject - rejecting state                   |
// qaccept - accepting state                   |
//---------------------------------------------|

name: Compare 2 Strings
init: qCopy
accept: qAccept, qReject

qCopy,0,_,qCopy,0,0,>,>
qCopy,1,_,qCopy,1,1,>,>
qCopy,_,_,qLeft,_,_,<,<

qLeft,0,0,qLeft,_,0,<,<
qLeft,0,1,qLeft,_,1,<,<
qLeft,1,1,qLeft,_,1,<,<
qLeft,1,0,qLeft,_,0,<,<
qLeft,_,_,qMoveR,_,_,>,>

qMoveR,_,0,qMoveR,_,0,>,-
qMoveR,_,1,qMoveR,_,1,>,-
qMoveR,0,0,qCheck,0,0,-,-
qMoveR,0,1,qCheck,0,1,-,-
qMoveR,1,0,qCheck,1,0,-,-
```

```
qMoveR,1,1,qCheck,1,1,-,-

qCheck,0,0,qCheck,0,0,>,>
qCheck,0,1,qReject,0,1,-,-
qCheck,0,_,qReject,0,_,-,-

qCheck,1,1,qCheck,1,1,>,>
qCheck,1,0,qReject,1,0,-,-
qCheck,1,_,qReject,1,_,-,-

qCheck,_,_,qAccept,_,_,-,-
qCheck,_,0,qReject,_,0,-,-
qCheck,_,1,qReject,_,1,-,-
```

13. Let L be the language $\{a^i b^{2i} a^i \mid i > 0\}$.

(a) Use the pumping lemma for context-free languages to show that L is not context-free.

To Prove: $\{a^i b^{2i} a^i \mid i > 0\}$ is not context free.
Assume: L is a context free grammar

Let L = $\{a^i b^{2i} a^i \mid i > 0\}$
Let k be the number from the pumping lemma
Let s = abba

By the pumping lemma s = *uvwxy* with substrings *u*, *v*, *w*, *x* and *y*, such that
1. $|vwx| \leq k$,
2. $|vx| \geq 1$, and
3. uv^nwx^ny is in L for all $n \geq 0$.

Since $|s| \leq k$ and there are 4 possibilities for vwx:
1. u = **λ**, vwx = a, y = bba
2. u = a, vwx = b, y = ba
3. u = ab, vwx = b, y = a
4. u = abb, vwx = a, y = **λ**

Case 1:
Suppose we pump once
By the pumping lemma:
vwx = aa, s = aabba
s is not in the language

Case 2:
Suppose we pump once
By the pumping lemma:
vwx = bb, s = abbba
s is not in the language

Case 3:
Suppose we pump once
By the pumping lemma:
vwx = bb, s = abbba
s is not in the language

Case 4:
Suppose we pump once
By the pumping lemma:
vwx =aa, s = abbaaa
s is not in the language

Therefore, L is not a context free grammar

(b) Construct a context-sensitive grammar G that generates L.

S → abba | aSBa
aB → Ba
bB → bbb

(c) Give the derivation of aabbbbaa in G

S → aSBa → aabbaBa → aabbBaa → aabbbbaa

(d) Construct an LBA M that accepts L.

S → abba:

```
;initial tape: >  S  <
;initial state: q0
q0 > > r q1
q1 _ _ r q1
q1 S S * q2

q2 S X r q4

q4 a a r q4
q4 b b r q4
q4 a a r q4
q4 _ _ l q5

q5 a _ r q6
q5 b _ r q7
q5 X X r q9

q6 _ a l q10

q7 _ b l q10

q10 _ _ l q5

q9 _ a l q10
q10 X b l q11
q11 _ b l q12
q12 _ a l halt
```

S → aSBa:

```
;initial tape: >  S  <
;initial state: q0

q0 > > r q1
q1 _ _ r q1
q1 S S * q2
```

q2 S X r q4

q4 a a r q4
q4 B B r q4
q4 S S r q4
q4 a a r q4
q4 _ _ l q5

q5 a _ r q6
q5 B _ r q7
q5 S _ r q8
q5 X X r q9

q6 _ a l q10
q7 _ B l q10
q8 _ S l q10

q10 _ _ l q5

q9 _ a l q11
q11 X B l q12
q12 _ S l q13
q13 _ a l halt

aB → Ba:
;initial tape: > aB <
;initial state: q0

q0 > > r q1
q1 _ _ r q1
q1 a a * q2

q2 a X r q3
q3 B X r q4

q4 B B r q4
q4 a a r q4
q4 _ _ l q5

q5 a _ r q6
q5 B _ r q7
q5 X X r q9

q6 _ a l q10
q7 _ B l q10
q10 _ _ l q5

```
q9 _ a l q10
q10 X B l q11
q11 X _ r halt
```

bB → bbb:

```
;initial tape: >bB <
;initial state: q0

q0 > > r q1
q1 _ _ r q1
q1 b b * q2

q2 b X r q3
q3 B X r q4

q4 b b r q4
q4 _ _ l q5

q5 b _ r q6
;q5 B _ r q7
q5 X X r q9

q6 _ b l q10
;q7 _ B l q10
q10 _ _ l q5

q9 _ b l q10
q10 X b l q11
q11 X b l q12
q12 > > r halt
```

(e) Trace the computation of M with input aabbbbaa