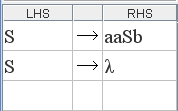
**Theory of Computation:**

**Problems on Context-Free Languages, Grammars and Push Down Automata.**

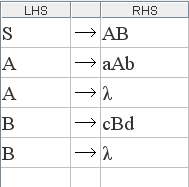
1. **Construct a context-free grammar for each of the following languages:**
2. *a*2n*b*n

Ans:



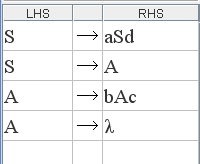
1. *a*n*b*n*c*m*d*m

Ans:



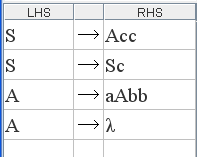
1. *a*n*b*m*c*m*d*n

Ans:



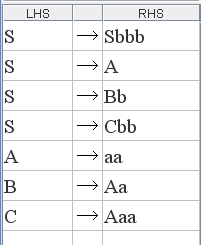
1. *a*n*b*m*c*kwhere 2*n* = *m* and *k* ≥ 2.

Ans:



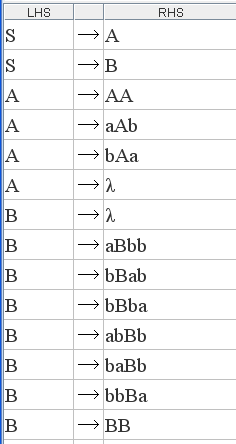
1. *a*n*b*m where *n* = 2 + (*m* mod 3). Is this a regular language? If so, make sure that the grammar is regular.

Ans: Yes, this is a regular language. The grammar below is left-linear.



1. All strings over {*a, b*} with either equal numbers of *a* and *b* or twice as many *b* s as *a* s.

Ans:



**B. Answer the following**

1. Prove that the set of all regular languages is a proper subset of the set of all context-free languages.

Ans: Every regular language is context-free since every regular language has a right-linear grammar which is also a context-free grammar. Thus, regular languages are a subset of context-free languages. To show that they are a proper subset, we must show that at least one context-free language is not regular. We have already seen that several languages are not regular. Some of them, for example, *wcw*R are in fact context-free. Thus, regular languages are a proper subset of context-free languages.

1. We have seen that elements of the syntax of high-level programming languages such as arithmetic expressions, nested parentheses and nested if-then-else statements are all context-free languages. Assuming that programs written in the language are made up of a sequence of such expressions and statements each of which is context-free, show that the set of all such programs is also context-free.

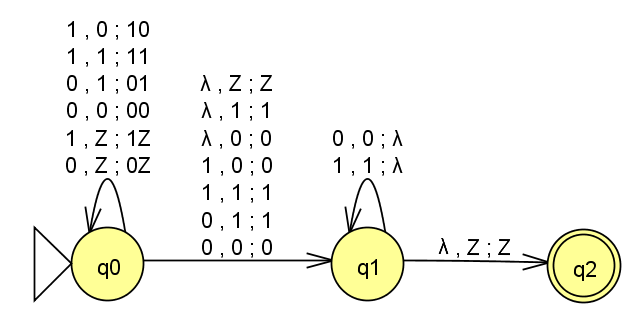
Ans: We know that context-free languages are closed under concatenation and \* closure. Therefore the above statement is true.

1. Show that the set of all binary strings which are palindromes and, when interpreted as positive integers, are divisible by 3, is context-free.

Ans: The set of all binary strings which are palindromes is context-free:

*S* → 0*S*0 | 1*S*1 | 0 | 1 | *λ*

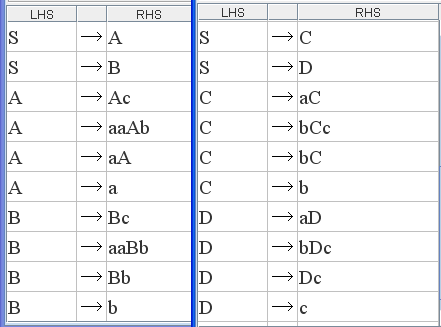
We can also construct a PDA to accept all odd and even palindromes:



The set of all binary strings that are divisible by 3 is regular. We know that the intersection of a context-free language with a regular language is context-free. Hence the given statement is true.

1. Give an example of a context-free language whose complement is not context-free

Ans: We know that *a*2m*b*m*c*m is not context-free this can be shown using the Pumping Lemma; intuitively, this language involves two comparisons of counts: number of *a* s to be twice the number of *b* s and number of *b* s to be equal to number of *c* s; both are not possible using the single stack of a PDA. Its complement is the subset of *a*\**b*\**c*\* where either of the two pairs of numbers do not match i.e., it has more *a* s than twice the number of *b* s, or less *a* s than twice the number of *b* s, or more *b* s than the number of *c* s, or less *b* s than the number of *c* s. This language is context-free since we can construct a CFG for it:



Thus, we have a context-free language *a*2m*b*n*c*k, *m* ≠ *n*, or *n* ≠ *k,* whose complement *a*2m*b*m*c*m is not context-free.

1. Show that the set of all strings over {*a, b, c*} that are either even palindromes or odd palindromes is context-free.

Ans: The set of all even palindromes over the given alphabet is context-free:

*S* → *aSa* | *bSb* | *cSc* | *λ*

The set of all odd palindromes is also context-free:

*S* → *aSa* | *bSb* | *cSc* | *a* | *b* | *c*

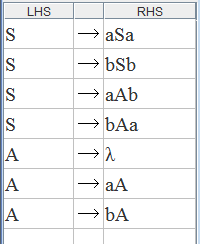
We know that the union of two context-free languages is context-free. Therefore the given language is context-free.

1. Can we show that the set of all strings over {*a, b, c*} that are neither even palindromes nor odd palindromes is context-free? Explain.

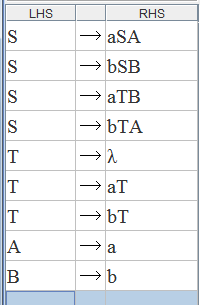
Ans: Yes, but not using the idea of complementation. Consider the set of all strings that are either even palindromes or odd palindromes. This language is context-free since we can easily construct either a CFG or a PDA for it. Its complement, the set of all strings that are neither even palindromes nor odd palindromes may or may not be context-free since there is no guarantee that the complement of a context-free language is context-free.

However, for this problem, it is possible to construct a PDA or a CFG directly for the set of all strings that are neither even nor odd palindromes. for a PDA for all strings that are not odd palindromes assuming that *c* occurs only as a separator for the two parts of the input string. Modifying this so that it accepts only those that are neither even nor odd palindromes is not straightforward. This is due to the nondeterministic nature of even palindromes. A nondeterministic PDA for even palindromes cannot be complemented easily by exchanging its final and non-final states.

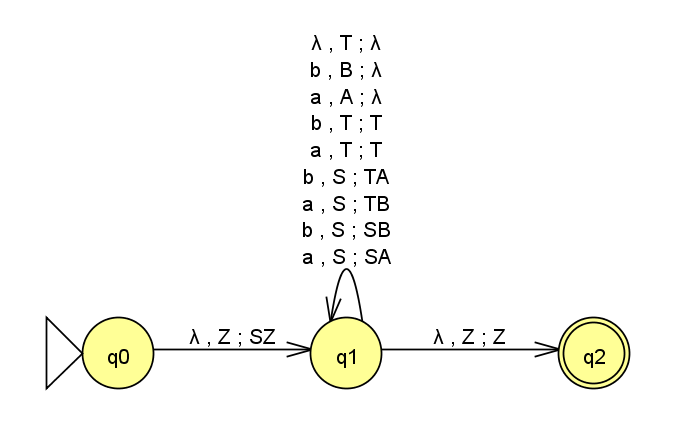
However, we will see how to construct a CFG and a PDA directly. A CFG for all strings that are not odd or even palindromes is with only *an* and *b*:



This grammar can be converted to GNF:



A corresponding PDA is:



Since we could construct both a CFG and a PDA, the given language is certainly context-free, but not because its complement is context-free.

1. If a programming language is context-free, every program that can be written in the language can be generated from the grammar of the language. Can we show that the set of all invalid programs, that is, those with syntax errors, is not context-free? Explain.

Ans: This is not possible since the complement of a context-free language may or may not be context-free. As such, the set of all programs with syntax errors may or may not be context-free. We either have to construct a CFG/PDA for it to show that it is context-free or we have to use the Pumping Lemma for context-free languages to show that it is not context-free. Both are not easy, in general.

1. We know that the set of strings *w* = (*a* + *b*)\* is a regular language and therefore also a context-free language. We also know that the concatenation of two context-free languages is a context-free language. Consider the set of concatenations *w.w*. We showed that this language is not context-free. Is this a contradiction? Is this the language of the following grammar? Explain.

*S*  *AA*, *A*  *aA* | *bA* | *λ*

Ans: There is no contradiction. The language of *w.w* is the set of all strings where the first and second halves are identical. By simply concatenating any string from (*a* + *b*)\* with any (other) string from the same set, we do not get *w.w*. Instead, we get (*a* + *b*)\* itself! The language of the above grammar is also just (*a* + *b*)\*, not *w.w*. In the grammar, there is no control over how the first *A* is expanded in relation to how the second *A* is expanded; there is no way to construct a CFG for *ww*.

1. What if in *w.w*, *w* = (*ab*)\* + (*ba*)\*? Is this language context-free or not?

Ans: Yes, this language is context-free. In fact, it is regular! A RegEx for this language is:

(*abab*)\* + (*baba*)\*

A right-linear grammar for this language is:

*S*  *A* | *B*, *A*  *ababA* | *λ*, *B*  *babaB* | *λ*

1. Is there a deterministic language that is not context-free?

Ans: Yes. Languages such as *a*n*b*n*c*n or *a*n*b*n*a*n*b*n, *n* > 0, are deterministic but not context-free. They are deterministic context-sensitive languages.

In fact, it is not known whether deterministic and non-deterministic linear-bounded automata that is the machines of context-sensitive languages, are equivalent.

**C. Construct a PDA for the following languages:**

1. Odd palindromes *wcw*R over {*a*, *b*, *c*} where *w* = (*a* + *b*)\*. Show an accepting sequence of configurations for the input *abbcbba*. Show how it rejects *abbcbb*.

Ans: The accepting sequence of configurations for *abbcbba* is:

(*q*0, *abbcbba*, *Z*), (*q*0, *bbcbba*, 0*Z*), (*q*0, *bcbba*, 10*Z*), (*q*0, *cbba*, 110*Z*), (*q*1, *bba*, 110*Z*),

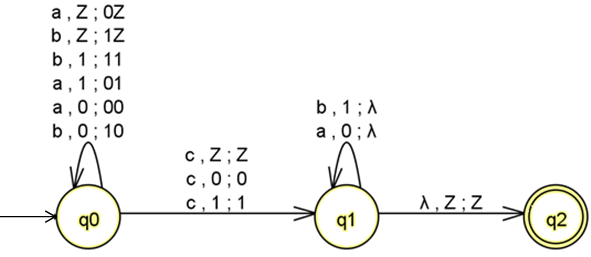
(*q*1, *ba*, 10*Z*), (*q*1, *a*, 0*Z*), (*q*1, *λ*, *Z*), (*q*2, *λ*, *Z*)

For *abbcbb*, the rejecting sequence is:

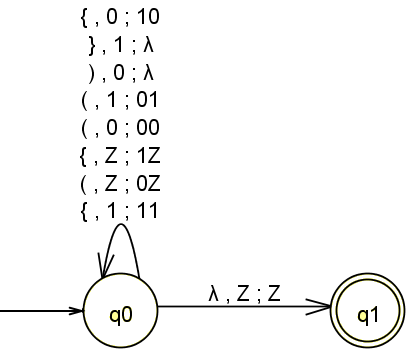
(*q*0, *abbcbb*, *Z*), (*q*0, *bbcbb*, 0*Z*), (*q*0, *bcbb*, 10*Z*), (*q*0, *cbb*, 110*Z*), (*q*1, *bb*, 110*Z*),

(*q*1, *b*, 10*Z*), (*q*1, *λ*, 0*Z*)

That is, the PDA halts in a non-final state; the stack is also not empty and the string is rejected.



1. Proper nesting of parentheses and flower brackets. For example, {(())(){{()}{}}}. Show how it rejects {(){{(})}}.

Ans: 

It can be seen that {(())(){{()}{}}} is accepted with the sequence (showing only successive stack contents):

*Z*, 1*Z*, 01*Z*, 001*Z*, 01*Z*, 1*Z*, 01*Z*, 1*Z*, 11*Z*, 111*Z*, 0111*Z*, 111*Z*, 11*Z*, 111*Z*, 11*Z*, 1*Z*, *Z*

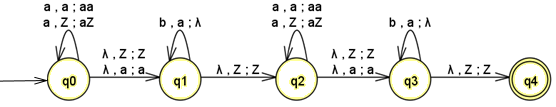
The string {(){{(})}} is rejected with the sequence:

*Z*, 1*Z*, 01*Z*, 1*Z*, 11*Z*, 111*Z*, 0111*Z*,

At this point, there is a mismatch between the next input symbol } and the symbol 0 on the stack and the PDA halts right there, rejecting the input.

1. *a*n*b*n*a*m*b*m, *n* ≥ 0, *m* ≥ 0. Show, along with two different accepting sequences of configurations, how non-determinism works to accept the string *aaabbb* in two different ways.

Ans:



Accepting sequence 1 for *aaabbb*:

(*q*0, *aaabbb*, *Z*), (*q*0, *aabbb*, *aZ*), (*q*0, *abbb*, *aaZ*), (*q*0, *bbb*, *aaaZ*), (*q*1, *bbb*, *aaaZ*),

(*q*1, *bb*, *aaZ*), (*q*1, *b*, *aZ*), (*q*1, *λ*, *Z*), (*q*2, *λ*, *Z*), (*q*3, *λ*, *Z*), (*q*4, *λ*, *Z*).

Accepting sequence 2 for *aaabbb*:

(*q*0, *aaabbb*, *Z*), (*q*1, *aaabbb*, *Z*), (*q*2, *aaabbb*, *Z*), (*q*2, *aabbb*, *aZ*), (*q*2, *abbb*, *aaZ*), (*q*2, *bbb*, *aaaZ*), (*q*3, *bbb*, *aaaZ*), (*q*3, *bb*, *aaZ*), (*q*3, *b*, *aZ*), (*q*3, *λ*, *Z*), (*q*4, *λ*, *Z*).

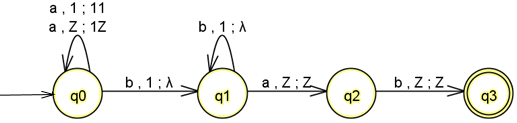
In fact, there is a third accepting sequence:

(*q*0, *aaabbb*, *Z*), (*q*0, *aabbb*, *aZ*), (*q*0, *abbb*, *aaZ*), (*q*0, *bbb*, *aaaZ*), (*q*1, *bbb*, *aaaZ*),

(*q*2, *bbb*, *aaaZ*), (*q*3, *bbb*, *aaaZ*), (*q*3, *bb*, *aaZ*), (*q*3, *b*, *aZ*), (*q*3, *λ*, *Z*), (*q*4, *λ*, *Z*).

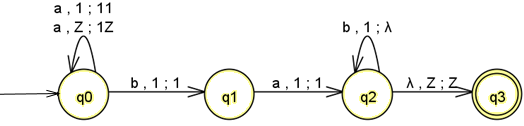
1. *a*n*b*n*ab*, *n* > 0. Make sure that the PDA is deterministic.

Ans: Note that the video solution has a lambda transition; this is not quite deterministic. The solution below corrects this by eliminating the lambda transition.



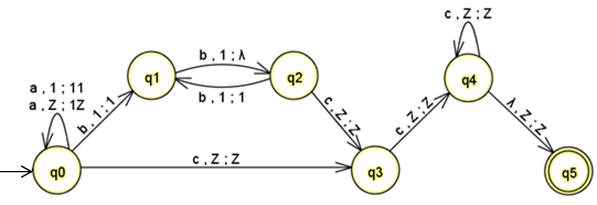
1. *a*n*bab*n, *n* > 0. Make sure that the PDA is deterministic.

Ans:



1. *a*n*b*m*c*k where 2*n* = *m* and *k* ≥ 2. Make sure that the PDA is deterministic.

Ans:



7. Consider the PDA shown in Fig. which accepts all strings over {*a, b, c*} that are *not* odd palindromes with *c* occurring only as the separator. Since we could construct a PDA for it, this language must be context-free. However, its complement – the set of odd palindromes – is also context-free since we can easily construct a PDA or a CFG for it. We also know that the complement of any context-free language may not be context-free. Is there a contradiction here? Explain.

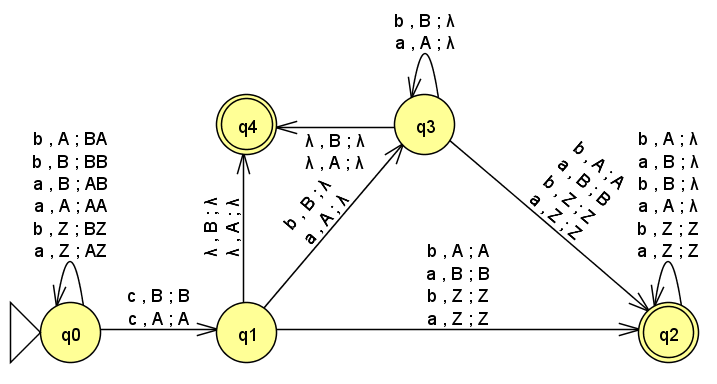


Figure PDA for all strings that are not odd-palindromes

Ans: There is no contradiction here. The complement of a context-free language may very well be also context-free. There is just no guarantee that the complement of every context-free language is context-free. In fact, there are context-free languages whose complements are not context-free.

8. Are the languages of the following two grammars with start symbols *S*1 and *S*2 the same? How can we be sure?

*S*1  *aS*1*b* | *λ*

*S*2  *ATB* | *λ*, *T*  *CTD* | *S*2, *C*  *A*, *D*  *B*, *A* , *B* 

Ans: Applying the method for eliminating unit productions the second grammar can be reduced to:

*S*2  *ATB* | *λ*, *T*  *ATB* | *S*2, *A* , *B* 

and then further by applying the substitution rule:

*S*2  *aTb* | *λ*, *T*  *aTb* | *S*2

Now, substituting for *T* in *S*2 , we get:

*S*2  *aS*2*b* | *λ* | *aaS*2*bb*

This is the same as the other grammar. Hence the languages of the two grammars are the same.

Note: It may be noted however that there is no general method to compare two context-free grammars