FLC exams questions

In this document are reported some questions that can be asked during the exam of Formal Languages and Compilers. The answers are not complete, but they can be used as a starting point to prepare the exam.

In the Chomsky hierarchy, what is the relation between type-n languages and type-(n-1) languages?

For each set of grammar, you have that these grammars can describe a set of languages, so the same hierarchy that you have for grammars is also the hierarchy of the languages. This means that type-(n-1) language is a larger set of languages compared to type-n languages: when you go from *type-n* to *type-(n-1)*, you increase the set of languages that you can have.

If a language can be generated by a type-1 grammar, can the same language be generated also by a type-2 grammar? Explain why.

A type-2 grammar is **more restricted** than a type-1 grammar, so it is possible that the same language can be described by a type-2 grammar but maybe cannot; it depends on which language it is.

How can we prove that the language L=Σ+ (positive closure of the alphabet) is regular?

This language includes all the strings of the alphabet, so it is very large, and it is also very simple. You can show that it is a regular language by building a grammar: in this grammar, the alphabet of terminal symbols is Σ, then there is the start symbol S (which is a non-terminal symbol), and you have to create some rules that can generate any string. The ideal way is to use a simple recursion, so the grammar can be built by using a recursive approach.

Explain why type-1 grammars are also called monotonic grammars.

In type-1 grammar there is a restriction: the right hand side of the rule **cannot be smaller** than the lefter side of the rule. For this reason, you cannot reduce the number of symbols, but you can only increase it by performing substitutions.

According to the given definitions, can a type-n grammar with n>0 generate a language that includes the empty string ?

It is **not possible**, because *type-1*, *type-2* and *type-3* grammars are monotonic and, in order to generate the empty string, you need a rule that has the empty string on the right hand side. According to the Chomsky hierarchy, the only grammar that can generate the empty string is the type-0 one.

Is it possible to introduce further restrictions on regular grammars? What happens if we do it?

It **is possible** to introduce further restrictions on regular grammars (for example, by removing the non-terminal symbols on the right hand side of the rules); if you do this, you obtain grammars that can just generate a single symbol (from the start symbol, you can only generate one symbol).However, these languages are not useful from a practical point of view, so the regular grammars are considered the most simple ones for practical applications.

What is the "state explosion" problem that may occur when building a DFA equivalent to a given NFA?

This problem occurs for particular languages, for which the **number of states of the DFA is much larger than the number of states of the corresponding NFA**. It depends on the features of the language.

If we build a DFA equivalent to a given NFA, and the NFA has m states, what is the worst-case (i.e., maximum) number of states that we can have in the DFA?

As each state of the DFA actually represents a set of states of the NFA, the number of different subsets of different states is 2^m, where m is the number of states of the NFA. The cardinality of the power set is 2^m and it also represents the **worst case**, that is the maximum number of states that a DFA can have when you build the DFA such that it is equivalent to an NFA and, of course, this can be a very large number, because it grows exponentially with the number of states of the NFA.

Explain advantages and disadvantages of using an NFA instead of a DFA for solving the membership problem for a regular language.

The most important advantage of using an NFA instead of a DFA is that **there isn't the problem of state explosion**. There is also a disadvantage, because the execution time increases: you don't have the most performant way of solving the problem. Another advantage of using NFA instead of a DFA is that you have the possibility to implement extra operators, which could not be possible with the DFA approach

(GEN) What is the difference between a DFA and a DFA with -moves?

The difference is that the DFA with -moves can have transitions that are not labelled by a symbol of the alphabet, but they are labelled by the empty string . This means that, when you are in a state, you can move to another state without reading any symbol of the alphabet.

(GEN) What is the membership problem?

The membership problem is the problem of deciding whether a given string belongs to a given language or not.

What can we say about the intersection of two regular languages and the complement of a regular language? Are they regular or not? How can we prove it?

Yes, they are regular languages. Considering for example the complement of a language, if you have a DFA representation of a regular language, you can easily build another DFA that represents the complement of the language represented by the first DFA (it is enough to change final to non final states). The fact that you have this algorithm to build the complement of a language means that the complement of a language is also a regular language.The intersection can be expressed by means of the *De Morgan theorem*, in terms of complement and union, but as the complement of a regular language and the union of a regular language are regular languages, of course the intersection of regular languages is also a regular language.

What is a possible way to check if a regular language L1 is included into another regular language L2?

To check if *L1* is included into *L2*, you can use the equivalence: you can compute the intersection between *L1* and *L2* and check if it is equivalent to *L1*; if it is not equivalent to L1 it means that there is a part of *L1* that is not inside *L2*.You can also compute the complement of *L2* and check that the intersection between the complement of L2 and L1 is empty. Another way to check the inclusion is to compute the union of L1 and L2 and see if it is equivalent.

Is it possible to turn an ambiguous context-free grammar into a non-ambiguous one? Are there any limitations?

It is possible to turn an ambiguous context-free grammar into a non-ambiguous one, but not for any grammar: there are languages that are said to be inherently ambiguous for which this is not possible.

What is the relation between the context-free languages accepted by PDAs and the ones accepted by DPDAs (i.e., deterministic PDAs)?

DPDAs are a special case of non-deterministic PDAs and the class of languages accepted by *DPDAs* is a strict subset of the class of languages accepted by non-deterministic PDAs.

How is it possible to show that context-free languages are closed under union?

It is possible to build a grammar that is context-free by computing the union of the languages generated by two context-free grammars. The same goes for the closure and the concatenation operations.

Why is the deterministic context-free languages class interesting? What is its most important property, with reference to the membership problem?

The computational complexity of solving the *membership problem* for a deterministic and a nondeterministic PDA is different: for the deterministic one, you can solve the problem in the **linear** time in the length of the input string.

(GEN) What is the difference between a right linear grammar and a left linear grammar?

In a right linear grammar, the right hand side of the rules is always a single terminal symbol followed by a single non-terminal symbol, while in a left linear grammar, the right hand side of the rules is always a single non-terminal symbol followed by a single terminal symbol.

What is the relation between the languages accepted by Turing Machines (TMs) and the ones accepted by nondeterministic TMs?

A non-deterministic TM is a generalization of a deterministic one, but differently from what happens to PDAs, if you add non-determinism to a TM the class of languages that the machine can accept does not change (you still have the class of type-0 languages).

What is the main difference between recursively enumerable sets and recursive sets for what concerns the membership problem?

The class of recursive sets is the class of languages for which you can build a TM that always halts (it always tells if the string is accepted or not) and these recursive sets are the languages for which the membership problem is decidable (you can build an algorithm that solves the problem).

If a language can be generated by a type-3 grammar, can the same language be generated also by a type-2 grammar? Explain why.

Yes, if a language can be generated by a type-3 grammar, it can also be generated by a type-2 grammar simply because a type-3 grammar is also a type-2 grammar.

What is a turing machine? A Turing machine is a mathematical model of computation that defines an abstract machine, which manipulates symbols on a strip of tape according to a table of rules. Despite the model's simplicity, given any computer algorithm, a Turing machine capable of simulating that algorithm's logic can be constructed.

What is the difference between a deterministic and a nondeterministic Turing machine?

A deterministic Turing machine is a Turing machine that never has more than one move to make for a given situation. A nondeterministic Turing machine is a Turing machine that can have more than one move for a given situation.

What are the main advantage and the main disadvantage of using NFA instead of DFA for solving the membership problem for a regular language?

The main advantage is that NFA do not suffer from state explosion. The main disadvantage is that the algorithms to solve the membership problem using NFA are computationally more complex than those that use DFA.

(GEN) What is the halting problem related to a Turing Machine?

Te halting problem is the fact that a Turing Machine always halts when is in accepting state, in addition is not always possible to require that a TM halts when is in rejecting state.

(GEN) Which kind of conflicts can be present in a parsing table?

A parsing table can have shift/reduce conflicts and reduce/reduce conflicts.

(GEN) What is a rightmost derivation?

A rightmost derivation is a sequence of grammar rule applications that transforms the start symbol into a string of terminal symbols.

[2022-07-04] In what case is a context-free grammar said to be ambiguous? Is the ambiguity of a context-free grammar decidable? What is the meaning of decidability?

A CFG is ambiguous if there is at least one string in its language having two different parse trees. The ambiguity of a CFG is not decidable. Decidability of a problem means there is an algorithm that can always solve it (i. e., for any inputs).

[2022-07-19] Is it possible that a language generated by a type-2 grammar is also generated by another type-1 grammar? Explain why.

Yes, it is possible because any type-2 language, generated by a type-2 grammar, is also a type-1 language, hence it can be generated by a type-1 grammar.

[2022-09-16] What is the relation between the languages generated by linear grammars and the languages generated by regular grammars?

The languages generated by regular grammars are a subset of the languages generated by linear grammars.

[2023-07-03] What is the difference, in terms of computational complexity, between the membership problem for context-free languages and the membership problem for deterministic context-free languages?

The time complexity of the membership problem for context-free languages is cubic in the length of the input string, while for deterministic context-free languages it is linear in the length of the input string.

What is the relation between right-linear grammars and regular grammars?

The key relationship between right-linear languages and regular languages is that they are equivalent:

1. Every right-linear language is regular.
2. Every regular language can be represented by a right-linear grammar.

What is the relation between recursively enumerable sets and type 0 languages?

Recursively enumerable sets are the same as type 0 languages. This means that every recursively enumerable set can be generated by a Turing machine, and every Turing machine can be represented by a recursively enumerable set.

What is an undecidable problem?

An undecidable problem is a problem for which there is no algorithm that can always solve it. This means that there are inputs for which the algorithm will not terminate or will not produce the correct result.

An example of an undecidable problem is the membership problem for context-free grammars: given a context-free grammar and a string, it is undecidable to determine whether the string is in the language generated by the grammar.

Explain why it is not possible to obtain a deterministic bottom-up parser for an ambiguous grammar

If a grammar is ambiguous, an input string may have more than one parse tree. For that reason, a bottom-up parser for that grammar does not have a unique way to build the parse tree for any input string, which produces conflicts in the parsing table and makes the parser nondeterministic. Only if we choose a priori which of the alternative parse trees is the right one and we exclude the others (conflict resolution) the parsing process can be made deterministic.