True/False Questions

True or False. T = true, F = false, and O = open, meaning that the answer is not known science at this time. In the questions below,

(i) **F** Let L be the language over $\{a, b, c\}$ consisting of all strings which have more a's than b's and more b's than c's. There is some PDA that accepts L.

L is not context-free.

(ii) **T** The language $\{a^n b^n \mid n \geq 0\}$ is context-free.

(iii) **F** The language $\{a^n b^n c^n \mid n \geq 0\}$ is context-free.

(iv) **T** The language L $\left\{a^i b^j c^k \mid j = i + k\right\}$ is context-free.

L is the concatentation of $\left\{a^i b^i\right\}$ and $\left\{b^k c^k\right\}$. The concatenation of two CF languages is CF.

(v) **T** The intersection of any two regular languages is regular.

(vi) **T** The intersection of any regular language with any context-free language is context-free.

I never proved this for you, but I told you to remember it.

(vii) **F** The intersection of any two context-free languages is context-free.

T If L is a context-free language over an alphabet with just one symbol, then L is regular.

(viii) **T** There is a deterministic parser for any context-free grammar.

However, there may be no deterministic PDA. The language of all palindromes over $\{a, b\}$ is CF, but has no is not accepted by any DPDA.

(ix) **T** The set of strings that your high school algebra teacher would accept as legitimate expressions is a context-free language.

At least that's true for **my** high school algebra teacher!

(x) **T** Every language accepted by a non-deterministic machine is accepted by some deterministic machine.

A deterministic machine emulates every possible computation of the non-deterministic machine. This could increase the time complexity exponentially.

(xi) **T** The problem of whether a given string is generated by a given context-free grammar is decidable.

The CYK algorithm can be used.

(xii) **T** If G is a context-free grammar, the question of whether $L(G) = \emptyset$ is decidable.

Even though the CFG grammar equivalence problem is undecidable, this special case is decidable.

(xiii) **F** Every language generated by an unambiguous context-free grammar is accepted by some DPDA.

Palindromes, again.

(xiv) **T** The language $\{a^n b^n c^n d^n \mid n \geq 0\}$ is recursive.

After one month (or less) of a programming course, you could write a program to decide this language.

(xv) **T** The language $\{a^n b^n c^n \mid n \geq 0\}$ is in the class $\mathcal{P}$-TIME.

After one month (or less) of a programming course, you could write a program to decide this language in polynomial time.

(xvi) **O** There exists a polynomial time algorithm which finds the factors of any positive integer, where the input is given as a binary numeral.

One of the most important open questions in complexity theory. If it is true, RSA coding can be broken in polynomial time. (A disaster.) But "everyone" believes it is false.

(xvii) **F** Every undecidable problem is $\mathcal{NP}$-complete.

All $\mathcal{NP}$ languages are decidable.

(xviii) **F** Every problem that can be mathematically defined has an algorithmic solution.

The halting problem is mathematically defined, yet has no algorithmic solution.

(xix) **F** The intersection of two undecidable languages is always undecidable.

This is trivial. If L is undecidable, then L', its complement, is undecidable, and $L \cap L' = \emptyset$.

(xx) **T** Every $\mathcal{NP}$ language is decidable.

But a machine that decides it might take exponential time.

(xxi) **T** The intersection of two $\mathcal{NP}$ languages, L_1 and L_2, must be $\mathcal{NP}$.

A string is in $L_1 \cap L_2$ if it has two certificates, one L_1, the other for L_2

(xxii) **O** If L_1 and L_2 are $\mathcal{NP}$-complete languages and $L_1 \cap L_2$ is not empty, then $L_1 \cap L_2$ must be $\mathcal{NP}$-complete.

This is true if $\mathcal{P} = \mathcal{NP}$, but false otherwise.

(xxiii) **O** There exists a $\mathcal{P}$-TIME algorithm which finds a maximum independent set in any graph.

The independent set problem is $\mathcal{NP}$–complete.

(xxiv) **T** There exists a $\mathcal{P}$-TIME algorithm which finds a maximum independent set in any **acyclic** graph G.

Tricky, tricky!

(xxv) **O** $\mathcal{NC} = \mathcal{P}$.

Important open problem.

(xxvi) **O** $\mathcal{P} = \mathcal{NP}$.

Important open problem.

(xxvii) **O** $\mathcal{NP} = \mathcal{P}$-SPACE

Important open problem.

(xxviii) **O** $\mathcal{P}$-SPACE = EXP-TIME

Important open problem.

(xxix) **O** EXP-TIME = EXP-SPACE

Important open problem.

(xxx) **F** EXP-TIME = $\mathcal{P}$-TIME.

By the time hierarchy theorem.

(xxxi) **F** EXP-SPACE = $\mathcal{P}$-SPACE.

By the space hierarchy theorem.

(xxxii) **T** The traveling salesman problem (TSP) is known to be $\mathcal{NP}$–complete.

One of the best known $\mathcal{NP}$–complete problems.

(xxxiii) **T** The language consisting of all satisfiable Boolean expressions is known to be $\mathcal{NP}$-complete.

This is the Cook-Levin theorem.

(xxxiv) **T** The Boolean Circuit Problem is $\mathcal{P}$.

It is actually $\mathcal{P}$–complete.

(xxxv) **O** The Boolean Circuit Problem is in $\mathcal{NC}$.

If true, all polynomial time algorithms can be executed quickly using multi-processor machines, since Boolean Circuit is $\mathcal{P}$–complete. Of immense practical importance.

(xxxvi) **F** If L_1 and L_2 are undecidable langugages, there must be a recursive reduction of L_1 to L_2.

Maybe, maybe not.

(xxxvii) **T** 2-SAT is $\mathcal{P}$-TIME.

(xxxviii) **O** 3-SAT is $\mathcal{P}$-TIME.

3-SAT is $\mathcal{NP}$–complete.

(xxxix) **T** Primality is $\mathcal{P}$-TIME.

Published in 2004.

(xl) **F** There is a $\mathcal{P}$-TIME reduction of the halting problem to 3-SAT.

Since 3-SAT is decidable, this would imply that the halting problem is decidable.

(xli) **T** Every context-free language is in $\mathcal{P}$.

By the CYK algorithm.

(xlii) **T** Every context-free language is in $\mathcal{NC}$.

I have found this statement on the internet, but have not found a proof or a citation to a proof. Can you help?

(xliii) ________ Addition of binary numerals is in $\mathcal{NC}$.

Using the tournament method.

(xliv) **O** Every context-sensitive language is in $\mathcal{P}$.

The CSL membership problem is known to be $\mathcal{P}$–SPACE complete, which means that whether it is $\mathcal{P}$–TIME is an open question.

(xlv) **F** Every language generated by a general grammar is recursive.

Every language generated by a general grammar is recursively enumerable, and vice versa, so HALT is generated by a general grammar.

(xlvi) **F** The problem of whether two given context-free grammars generate the same language is decidable.

It's co-$\mathcal{RE}$ and undecidable.

(xlvii) **T** The language of all fractions (using base 10 numeration) whose values are less than π is decidable. (A *fraction* is a string. "314/100" is in the language, but "22/7" is not.)

π is a recursive real number.

(xlviii) **T** Any context-free language over the unary alphabet is regular.

(xlix) ________ Any context-sensitive language over the unary alphabet is regular.

I need to check this one. I believe the answer is **T**.

(l) **F** Any recursive language over the unary alphabet is regular.

For example, the set of unary numerals for powers of 2 is recursive but not regular.

(li) **T** There exists a polynomial time algorithm which finds the factors of any positive integer, where the input is given as a unary numeral.

This is very easy. Just try to divide the number by every smaller number.

(lii) **T** For any two languages L_1 and L_2, if L_1 is undecidable and there is a recursive reduction of L_1 to L_2, then L_2 must be undecidable.

If L_2 were decidable, then L_1 would be decidable.

(liii) **F** For any two languages L_1 and L_2, if L_2 is undecidable and there is a recursive reduction of L_1 to L_2, then L_1 must be undecidable.

You can easily reduce an easy language to a hard language.

(liv) **F** If P is a mathematical proposition that can be written using a string of length n, and P has a proof, then P must have a proof whose length is $O(2^{2^n})$.

This theorem is false even if you use **any** computable function of n. For example, even if you replace 2^{2^n} by the Ackermann function, the statement is false.

(lv) **T** If L is any $\mathcal{NP}$ language, there must be a $\mathcal{P}$–TIME reduction of L to the partition problem.

The partition problem is $\mathcal{NP}$–complete.

(lvi) **F** Every bounded function is computable.

If S is a set of integers, then $f(n) = \begin{cases} 1 \text{ if } n \in S \\ 0 \text{ otherwise} \end{cases}$ is bounded. The membership problem for a set of integers is, in general, undecidable, since there are uncountably many sets of integers, and only countably many C++ programs.

(lvii) **O** If L is $\mathcal{NP}$ and also co-$\mathcal{NP}$, then L must be $\mathcal{P}$.

Another important open problem. Factorization of binary numerals is in $\mathcal{NP}$ and also in co-$\mathcal{NP}$, but is not known to be in $\mathcal{P}$.

(lviii) **T** If L is $\mathcal{RE}$ and also co-$\mathcal{RE}$, then L must be decidable.

Suppose L and its complement L' are both $\mathcal{RE}$. Let M and M' be machines which accept L and L', respectively. Run the machines simultaneously on a string w. Eventually, one of them halts.

(lix) **T** Every language is enumerable.

But you might not be able to compute an enumeration.

(lx) **F** If a language L is undecidable, then there can be no machine that enumerates L.

Some undecidable languages, such as HALT, are $\mathcal{RE}$, but not all are.

(lxi) **T** There exists a mathematical proposition which is true, but can be neither proved nor disproved.

I cannot give you an example, because if I knew it was true, I must have a proof of it. I can only point to a set of propositions and prove that one of them is true, yet neither provable nor disprovable.

(lxii) ________ There is a non-computable function which grows faster than any computable function.

One example is the Busy Beaver function. `https://en.wikipedia.org/wiki/Busy_beaver`

(lxiii) **T** There exists a machine that runs forever and outputs the string of decimal digits of π (the well-known ratio of the circumference of a circle to its diameter).

π is a recursive real number.

(lxiv) **F** For every real number x, there exists a machine that runs forever and outputs the string of decimal digits of x.

Not every real number is recursive.

(lxv) **F** **Rush Hour**, the puzzle sold in game stores everywhere, generalized to a board of arbitrary size, is known to be $\mathcal{NP}$–complete.

Maybe it is, but its an open problem. It is known to be $\mathcal{P}$–SPACE complete.

(lxvi) **O** There is a polynomial time algorithm which determines whether any two regular expressions are equivalent.

(lxvii) **O** If two regular expressions are equivalent, there is a polynomial time proof that they are equivalent.

(lxviii) **F** Every subset of a regular language is regular.

The **idiot** question I like to ask! **Every** language is a subset of the regular language Σ^*, where Σ is the alphabet of the language.

(lxix) **T** Every subset of any enumerable set is enumerable.

(lxx) **T** The computer language Pascal has Turing power.

Any computation that can be done by any machine can be done by some Pascal program.

(lxxi) **T** Computing the square of an integer written in binary notation is an $\mathcal{NC}$ function.

All arithmetic problems are in $\mathcal{NC}$.

(lxxii) **T** If L is any $\mathcal{P}$-TIME language, there is an $\mathcal{NC}$ reduction of L to the Boolean circuit problem.

That's the definition of $\mathcal{P}$–completeness.

(lxxiii) ________ If an abstract Pascal machine can perform a computation in polynomial time, there must be some Turing machine that can perform the same computation in polynomial time.

By the Church-Turing thesis.

(lxxiv) **T** The binary integer factorization problem is co-$\mathcal{NP}$.

That means the set of binary numerals for compsite numbers is $\mathcal{NP}$. IF n is composite and m is a non-trivial factor of n, then $\langle m \rangle$ is a certificate that n is composite.

(lxxv) **O** There is a polynomial time reduction of the subset sum problem to the binary numeral factorization problem. It is not known whether that factorization problem is $\mathcal{P}$–complete.

(lxxvi) **T** There is a polynomial time reduction of the binary numeral factorization problem to the subset sum problem.

There is a $\mathcal{P}$–TIME reduction of any $\mathcal{NP}$ problem to any $\mathcal{NP}$–complete problem.

(lxxvii) **F** For any real number x, the set of fractions whose values are less than x is $\mathcal{RE}$.

There are uncountably many real numbers, but the set of $\mathcal{RE}$ languages over a given alphabet is countable.

(lxxviii) **T** For any recursive real number x, the set of fractions whose values are less than x is recursive (decidable).

By definition of recursive real number.

(lxxix) **T** The union of any two deterministic context-free languages must be a DCFL.

Connect the output of one DPDA to the input of the other.

(lxxx) **F** The intersection of any two deterministic context-free languages must be a DCFL.

Let $L_1 = \left\{a^i b^i c^j\right\}$ and $L_2 = \left\{a^i b^j c^j\right\}$. the intersection $L_1 \cap L_2 = \{a^n b^n c^n\}$ is not context-free.

(lxxxi) **T** The complement of any DCFL must be a DCFL.

Proving this is rather difficult.

(lxxxii) **T** The membership problem for a DCFL is in the class $\mathcal{P}$-TIME.

Use the CYK algorithm.

(lxxxiii) **T** Every finite language is decidable.

Trivially. Just compare a string w with every member of L.

(lxxxiv) Duplicate question.

(lxxxv) **F** 2-SAT is known to be $\mathcal{NP}$-complete.

T 2-SAT is $\mathcal{P}$–TIME

(lxxxvi) **T** The complement of any $\mathcal{P}$-TIME language is $\mathcal{P}$-TIME.

(lxxxvii) **T** The complement of any $\mathcal{P}$–SPACE language is $\mathcal{P}$–SPACE.

The *jigsaw puzzle problem* is, given a set of various polygons, and given a rectangular table, is it possibe to assemble those polygons to exactly cover the table?

The *furniture mover's problem* is, given a room with a door, and given a set of objects outside the room, it is possible to move all the objects into the room through the door?

(lxxxviii) **T** The jigsaw puzzle problem is known to be $\mathcal{NP}$-complete.

(lxxxix) **F** The jigsaw puzzle problem is known to be $\mathcal{P}$–SPACE complete.

It might be, but it's not known.

(xc) **F** The furniture mover's problem is known to be $\mathcal{NP}$ complete.

It might be, but it's not known.

(xci) **T** The furniture mover's problem is known to be $\mathcal{P}$–SPACE complete.

(xcii) **T** The complement of any recursive language is recursive.

If you decide whether $w \in L$, you have also decided whether $w \notin L$.

(xciii) **T** The complement of any undecidable language is undecidable.

See the previous question.

(xciv) **F** Every undecidable language is either $\mathcal{RE}$ or co-$\mathcal{RE}$.

There are lots more.

(xcv) **T** For any infinite countable sets A and B, there is a 1-1 correspondence between A and B.

(xcvi) **T** A language L is recursively enumerable if and only if there is a machine which accepts L.

(xcvii) **T** Every $\mathcal{NP}$ language is reducible to the independent set problem in polynomial time.

(xcviii) **T** If a Boolean expression is satisfiable, there is a polynomial time proof that it is satisfiable.

Use a certificate.

(xcix) **T** The general sliding block problem is $\mathcal{P}$–SPACE complete.

(c) **T** The regular expression equivalence problem is $\mathcal{P}$–SPACE complete.

(ci) **T** The context-sensitive membership problem is $\mathcal{P}$–SPACE complete.

By Savitch's theorem (1970).

(cii) **T** The Post correspondence problem is undecidable.

(ciii) **F** The set of real numbers is countable.

Use a diagonalization argument.

(civ) **T** The set of recursive real numbers is countable.

(cv) **T** A finite set has only finitely many subsets.

A set of size n has 2^n subsets.

(cvi) **F** A countable set has only countably many subsets.

Use a diagonalization argument.

(cvii) **F** Suppose some machine writes a convergent sequence of rational numbers $x_1, x_2, \ldots$. Then $lim_{i\to\infty} x_i$ must be a recursive real number.

Proving this is a challenge problem in CS656.

(cviii) **T** There are infinitely many prime integers.

Known to the ancient Greeks.

(cix) **O** There are infinitely many Mersenne primes. (Look it up.)

This has been an open question at least since the 17th century.