CS Exit Ethiopia

Questions & Worked Answers

Computer Science Exit Examination

156 complete questions

1 selected course | 18 November 2026

Selected courses

01. Automata and Complexity Theory

https://csexitethiopia.com

abelasnake2003@gmail.com | Telegram: @Abechu7

Course contents

Each course begins on a new page. PDF bookmarks provide question, study-note section and revision-topic navigation.

Automata and Complexity Theory 3

Reading the answers

Every question includes its answer, explanation, choice-by-choice contrasts and available evidence. Primary note proof quotes the study notes. Related reading and module passages are labelled separately; a module reference is not presented as primary note proof.

Automata and Complexity Theory

Questions & Worked Answers

156 questions

Question 1 | Q00015

Which of the following term refers to the number of symbols composing a string?

A. Length

B. Symbols

C. Alphabet

D. Word

Hint

Distinguish symbols, finite strings, their lengths and sets of strings. Apply the operation to its stated operands, including empty cases.

Answer A: Length

Count symbols rather than distinct symbols: that count is the string's length, $|w|$. Alphabet is the set from which symbols are drawn, not the count in a particular string.

Why each choice is right or wrong

A - Correct: This is precisely the defined count $|w|$.

B - Incorrect: Symbols are the individual objects being counted.

C - Incorrect: An alphabet is a finite set of symbols.

D - Incorrect: A word/string is the sequence whose length is measured.

Primary note proof

Study-note section AUT01

> An alphabet Σ is a finite set of symbols. A word/string is a finite ordered sequence of symbols from Σ; a language is a set of words, $L \subseteq \Sigma^*$. An empty word ε, also written λ, has length zero. The empty language $\varnothing$ contains no words; $\{\varepsilon\}$ contains one.

Study-note section AUT01

> Length counts occurrences, not distinct symbols: $|caddba|=6$, so $|caddba|^2=36$.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Automata and Complexity Theory module | PDF page 2 | lines 26-26

> The number of symbols in a string w is called the length of a string. It is denoted by $|w|$.

Automata and Complexity Theory module | PDF page 2 | lines 15-15

> An alphabet finite set of non-empty symbols. It is denoted by ∑ (sigma).

Automata and Complexity Theory module | PDF page 2 | lines 21-21

> It is a finite collection of symbols from the alphabet. The string is denoted by w.

Question sources

MOE Computer Science Exit Exam · 2016 · Question 15

Question 2 | Q00024

Let L_1={c,d} and L_2={e,fgh}. Which lowercase word is NOT an element of L_2L_1?

A. fghc

B. ed

C. fghe

D. ec

Hint

Distinguish symbols, finite strings, their lengths and sets of strings. Apply the operation to its stated operands, including empty cases.

Answer C: fghe

Language concatenation preserves order: each word of L_2 is followed by a word of L_1. The four products are ec, ed, fghc and fghd. Thus fghe is the unique nonmember: its ending e is not one of the words available from L_1. The lowercase transcription repair makes the symbols consistent with the declared languages.

Why each choice is right or wrong

A - Incorrect: fghc is fgh followed by c, so it belongs to the product.

B - Incorrect: ed is e followed by d, so it belongs to the product.

C - Correct: fghe cannot be formed because e is not in L_1.

D - Incorrect: ec is e followed by c, so it belongs to the product.

Primary note proof

Study-note section AUT02

> Concatenation xy writes x followed by y: cddc·ababb=cddcababb; order generally matters. εx=xε=x. Define x^0=ε and x^n as n copies for n≥1. Reverse x^R reads symbols backward: caddba reverses to abddac; $(xy)^R$=y^Rx^R and $(x^R)^R$=x.

Study-note section AUT02

For languages, LM={xy:x∈L,y∈M}, L^0={ε}, and L^{n+1}=L^nL. Thus {e,fgh}{c,d}={ec,ed,fghc,fghd}; fghe is absent.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Automata and Complexity Theory module | PDF page 2 | lines 21-21

It is a finite collection of symbols from the alphabet. The string is denoted by w.

Automata and Complexity Theory module | PDF page 3 | lines 13-14

A grammar defines a set of rules, and with the use of these rules, valid sentences in a language are
generated.

Question sources

MOE Computer Science Exit Exam · 2016 · Question 24

Question 3 | Q00032

Which theory provides mechanisms for classifying combinatorial problems and measuring the computational resources necessary to solve them?

A. Computability theory

B. Set theory

C. Complexity theory

D. Automata theory

Hint

Check whether the required procedure must halt on every input, and distinguish solving a task from recognizing its yes-instances.

Answer C: Complexity theory

The requested quantity is the resources needed to solve a problem. Computational complexity estimates time and space requirements. Computability instead asks whether a membership algorithm exists at all.

Why each choice is right or wrong

A - Incorrect: Computability concerns existence of algorithms, rather than their resource costs.

B - Incorrect: Set theory does not name the study of time and space requirements.

C - Correct: Matches the resource-estimation definition directly.

D - Incorrect: Automata studies abstract machines; the specific resource classification asked here is complexity.

Primary note proof

Study-note section AUT-S01

> Computability theory asks whether an effective algorithm can solve a problem at all. For a decision problem, a decision procedure must terminate with the correct yes-or-no result on every input.

Study-note section AUT-S01

> Computational complexity theory studies the time and memory resources required to solve problems as functions of encoded input size. Automata theory studies abstract machine models and the languages they recognize; these subjects overlap but ask different questions.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Automata and Complexity Theory module | PDF page 15 | lines 33-35

> Algorithm analysis is an important part of a broader computational complexity theory, which provides theoretical estimates for the resources needed by any algorithm which solves a given computational problem.

Automata and Complexity Theory module | PDF page 15 | lines 24-26

> A set X is recursive if we have an algorithm to determine whether a given element belongs to X or not.

Question sources

MOE Computer Science Exit Exam · 2016 · Question 32

Question 4 | Q00192

Which language over {0,1} is denoted by (0+1)*0(0+1)*0(0+1)*? Here + means union.

A. Strings with substring 00

B. Strings with at least two 0's

C. Strings with at most two 0's

D. Strings with postfix 1.

Hint

Track mandatory symbols and factors that may repeat or disappear. Use short test words to check both directions of each proposed language description.

Answer B: Strings with at least two 0's

Each starred factor denotes any binary string, including the empty string. The two unstarred zeros therefore require at least two zero occurrences, while permitting arbitrary symbols before, between and after them. Conversely, any word with at least two zeros can be split at two selected zero positions to fit the expression. For example, 010 fits without containing adjacent zeros, and 000 fits although it has more than two zeros. No final 1 is required.

Why each choice is right or wrong

A - Incorrect: 010 is generated but does not contain the adjacent substring 00.

B - Correct: Two explicit zeros are required; the starred factors allow additional zeros and ones.

C - Incorrect: 000 is generated, so there is no upper bound of two zeros.

D - Incorrect: 00 is generated and ends in 0.

Primary note proof

Study-note section AUT13

> To require at least one a anywhere over {a,b}, use (a|b)*a(a|b)*. b*ab* requires exactly one a. At least two zeros uses (0|1)*0(0|1)*0(0|1)*; the zeros need not be adjacent. Testing 010 distinguishes it from a requirement for substring 00. For S→aS|ε, any number of a's is generated, giving a*.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Automata and Complexity Theory module | PDF page 5 | lines 22-22

> In a regular expression, x* means zero or more occurrence of x.

Automata and Complexity Theory module | PDF page 7 | lines 16-16

> A language is said to be a Regular language if and only if some finite state machine recognizes it

Question sources

Addis Ababa University mock exam · Question 94

Question 5 | Q00193

The transition function in a finite-state automaton maps which pair?

A. Q and Σ to Σ

B. Q and Q to Σ

C. Σ and Σ to Q

D. Q and Σ to Q

Hint

Read the transition domain and acceptance condition carefully. Track which inputs are consumed and evaluate the whole word under the stated model.

Answer D: Q and Σ to Q

For a deterministic finite automaton, finite control receives the current state and input symbol and chooses the next state. Thus $\delta:Q\times\Sigma\to Q$. A general NFA instead has a set of possible next states, so the stem tacitly uses the deterministic convention.

Why each choice is right or wrong

A - Incorrect: Outputs an input symbol, not a state.

B - Incorrect: The second input must be a symbol, not another state.

C - Incorrect: Omits the current state and uses two input symbols.

D - Correct: Correct deterministic domain and codomain.

Primary note proof

Study-note section AUT06

> A deterministic finite automaton (DFA) is $M=(Q,\Sigma,\delta,q_0,F)$, with finite nonempty Q, start $q_0\in Q$, accepting set $F\subseteq Q$ and total $\delta:Q\times\Sigma\to Q$. Exactly one next state is specified per state/symbol pair; this does not mean the whole machine has one transition. F may be empty, recognizing $\varnothing$.

Study-note section AUT06

> Start at q_0 and consume the entire input. Accept iff the ending state is in F; visiting an accepting state earlier is insufficient. ε is accepted exactly when $q_0\in F$. A nonaccepting sink loops on every symbol and cannot reach acceptance; it still finishes after finite input. Finite automata have finite memory encoded by states, not no state or no ability to remember any history.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Automata and Complexity Theory module | PDF page 3 | lines 25-25

> A finite automaton is a collection of 5-tuple (Q, ∑, δ, q0, F), where:

Automata and Complexity Theory module | PDF page 4 | lines 5-6

> Finite control: The finite control decides the next state on receiving particular input from input tape.

Automata and Complexity Theory module | PDF page 5 | lines 3-5

> DFA refers to deterministic finite automata. Deterministic refers to the uniqueness of the computation. In the DFA, the machine goes to one state only for a particular input character. DFA does not accept the null move.

Question sources

Addis Ababa University mock exam · Question 95

Question 6 | Q00194

With start variable A and productions A→aB|bA|bBa and B→bB|cA|b, which offered string is generated?

A. abbcabbcbbbcbbaca

B. bcbbbcabbcbaa

C. acbcbbbbcabaa

D. bbbbcbbbbcab

Hint

Apply the stated transition, closure or equivalence rule step by step; distinguish a guaranteed bound from a value for one particular machine.

Answer C: acbcbbbbcabaa

With A as the start variable, the candidate follows the production sequence A⇒aB⇒acA⇒acbBa⇒acbcAa⇒acbcbAa⇒acbcbbAa⇒acbcbbbAa⇒acbcbbbbBaa⇒acbcbbbbcAaa⇒acbcbbbbcaBaa⇒acbcbbbbcabaa. Each rewrite uses one of the displayed rules, including a final B→b that removes the remaining variable. The other three offered words fail the complete shrinking-residual membership test.

Why each choice is right or wrong

A - Incorrect: Every permitted prefix or wrapper decomposition eventually leaves a residual that neither required variable generates.

B - Incorrect: The required A/B prefix and wrapper decompositions do not reach the terminating B→b production.

C - Correct: The displayed derivation produces acbcbbbbcabaa from start A.

D - Incorrect: Following the required variable after each prefix does not produce this word.

Primary note proof

Study-note section AUT40

> For A→aB|bA|bBa and B→bB|cA|b, specify a start variable first. With A assumed, a terminating membership test recursively removes the required a/b prefix or b…a wrapper; B either matches b or removes b before B/c before A. Every recursive residual is shorter. Apply the membership test to every candidate word; none qualifies in this four-word example under the assumed start A. Use the membership test to justify whether a candidate word belongs to the language.

Study-note section AUT16

> In a context-free grammar (CFG) each production has one variable on the left, A→α with α∈(V∪Σ)*.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Automata and Complexity Theory module | PDF page 8 | lines 27-28

> R is a set of production rules where each production rule maps a variable to a string $s \in (V \cup \Sigma)*$;

Automata and Complexity Theory module | PDF page 9 | lines 2-2

> A sentential form is any string derivable from the start symbol.

Question sources

Addis Ababa University mock exam · Question 96

Question 7 | Q00195

When can we say two grammars are equivalent?

A. When the language derived from one grammar is a subset of the other

B. When the alphabets of the two grammars are the same

C. When the languages represented by both grammars are the same

D. When a single string is a member of both languages

Hint

Distinguish symbols, finite strings, their lengths and sets of strings. Apply the operation to its stated operands, including empty cases.

Answer C: When the languages represented by both grammars are the same

A grammar specifies its generated sentences. Equality of the generated sets, not equality of alphabets or overlap, is the criterion for semantic equivalence. For example {a} and {a,aa} have the same alphabet and a common string but different languages.

Why each choice is right or wrong

A - Incorrect: One-way inclusion allows extra strings in the other language.

B - Incorrect: Same terminal alphabet permits different sets of sentences.

C - Correct: Equality of the full generated languages establishes equivalence.

D - Incorrect: A single shared string establishes overlap only.

Primary note proof

Study-note section AUT04

> Write $G=(V,\Sigma,P,S)$: finite variables V, terminals Σ disjoint from V, productions P and start variable $S \in V$. A production replaces its left side by its right side. The language is $L(G)=\{w \in \Sigma^* : S \Rightarrow^* w\}$. A recognizer processes a word and accepts exactly the language members. Grammars generate languages; automata recognize languages, not grammars.

Study-note section AUT04

> Two grammars are equivalent when their generated languages are equal, regardless of rule counts or variable names. One common word, equal alphabets or one-way inclusion is insufficient.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Automata and Complexity Theory module | PDF page 3 | lines 13-14

> A grammar defines a set of rules, and with the use of these rules, valid sentences in a language are
> generated.

Question sources

Addis Ababa University mock exam · Question 97

Computer Science Exit Exam · Sene 2016 · Question 19-3

Computer Science Exit Exam practice collection · Question 262

Question 8 | Q00196

If L1 and L2 are two regular languages, which one of the following is true

A. LiU L2 is a regular language only if empty string is not a member

B. Li* is the same with L2* and are both regular languages

C. LiL is also a regular language

D. L:* is a regular language and is equivalent to L;

Hint

Track mandatory symbols and factors that may repeat or disappear. Use short test words to check both directions of each proposed language description.

Answer C: The concatenation L1L2 is regular.

Regular languages are closed under concatenation, so any two regular languages have a regular concatenation. Reversal preserves regularity but need not preserve the original language. Union needs no exclusion of the empty word, and arbitrary closures need not coincide.

Why each choice is right or wrong

A - Incorrect: Union is regular even if it contains the empty word; the only-if restriction is false.

B - Incorrect: Both stars are regular, but arbitrary regular languages need not have equal Kleene closures.

C - Correct: The image reads L1L2; regular languages are closed under this concatenation operation.

D - Incorrect: The image reads reversal: it remains regular but need not equal L1, for example {ab} reverses to {ba}.

Primary note proof

Study-note section AUT14

> Right-linear rules have form A→aB, A→a or permitted ε rules; left-linear rules put the single variable at the opposite end, A→Ba. Both consistent orientations generate REG; arbitrarily mixing them can generate nonregular languages, for example S→aA|ε, A→Sb generates $a^n b^n$. A CFG rule A→aBA is legal but not regular-grammar form.

Study-note section AUT14

> Regular languages are closed under union, concatenation, star, reversal, intersection, difference and complement.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Automata and Complexity Theory module | PDF page 7 | lines 16-16

> A language is said to be a Regular language if and only if some finite state machine recognizes it

Automata and Complexity Theory module | PDF page 6 | lines 4-4

> Kleen closure: If L is a regular language then its Kleen closure L1* will also be a regular language.

Question sources

Addis Ababa University mock exam · Question 98

Question 9 | Q00197

Let G = {N, T, P, S} be a grammar where:
N={S, A}, T={ 0, 1}, S={S} and P= {S-11S|0A]|1A,A>0]|1] 0S}.
Which one of the following is a sentential form?

A. 111011110S

B. 11111101

C. 0000111001AS

D. OOO011L1I0A

Hint

Apply the stated transition, closure or equivalence rule step by step; distinguish a guaranteed bound from a value for one particular machine.

Answer B: 11111101

The grammar is S→11S|0A|1A, A→0|1|0S, and option D is 00001110A. Terminal sentences are also sentential forms. B has the derivation S⇒11S⇒1111S⇒111111S⇒1111110A⇒11111101. Every reachable form contains at most one nonterminal, at the right end. Before final termination, a return to S appends either 11, 00, or 10. A cannot be segmented into these pairs (its last prefix pair is 01); C has two variables; D has prefix ending at A which cannot be obtained by those pairs followed by 0 or 1.

Why each choice is right or wrong

A - Incorrect: Before S the terminal prefix must be a sequence of pairs 11,00,10; 111011110 contains a forbidden pair 01 and has odd length.

B - Correct: The explicit derivation ends at this terminal sentence, which is also a sentential form.

C - Incorrect: Two variables AS cannot arise from rules that never introduce two variables.

D - Incorrect: The form 00001110A would require an odd-length prefix before A (pairs followed by one symbol); its eight-symbol prefix has even length.

Primary note proof

Study-note section AUT40

> For S→11S|0A|1A and A→0|1|0S, every reachable form has at most one variable, at the right end. A complete terminal word is still a sentential form. For example S⇒11S⇒1111S⇒111111S⇒1111110A⇒11111101. Returning to S appends a pair 11,00 or 10; use that invariant to rule out impossible prefixes rather than guessing from string length.

Study-note section AUT16

> A sentential form is any string derived from S, including terminal-only sentences and S itself.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Automata and Complexity Theory module | PDF page 9 | lines 2-2

> A sentential form is any string derivable from the start symbol.

Automata and Complexity Theory module | PDF page 8 | lines 27-28

> R is a set of production rules where each production rule maps a variable to a string $s \in (V \cup \Sigma)*$;

Question sources

Addis Ababa University mock exam · Question 99

Question 10 | Q00198

In a derivation graph whose root is S, whose vertices are sentential forms, and whose directed edges are single production applications, which property records the steps generating a terminal string?

A. Root of the graph

B. Degree of the vertex

C. Path of a vertex from the root

D. Parent vertex

Hint

Apply the stated transition, closure or equivalence rule step by step; distinguish a guaranteed bound from a value for one particular machine.

Answer C: Path of a vertex from the root

A derivation is a sequence of production applications beginning at S and ending at a terminal word. Under the specified graph model, each application is one directed edge, so the root-to-terminal-vertex path records the complete sequence. A root alone gives only the start; degree and an immediate parent give local information. This graph is not being treated as a parse tree.

Why each choice is right or wrong

A - Incorrect: The root identifies S but contains no sequence of applications.

B - Incorrect: Degree counts incident edges rather than their ordered traversal.

C - Correct: A root-to-vertex path lists the successive production applications.

D - Incorrect: A parent identifies one predecessor rather than the entire derivation.

Primary note proof

Study-note section AUT16

> In a context-free grammar (CFG) each production has one variable on the left, $A \to \alpha$ with $\alpha \in (V \cup \Sigma)^*$. The left side therefore also belongs to $V \cup \Sigma$ and its string closure; alternatives stating these weaker memberships can all be true. A sentential form is any string derived from S, including terminal-only sentences and S itself.

Study-note section AUT16

> A leftmost derivation always replaces the leftmost variable; a rightmost one replaces the rightmost variable. This is not scanning the completed input in opposite directions. A complete parse tree has root S, internal variable nodes expanded by productions and terminal leaves, with ε leaves when that convention is used. Read the leaves left-to-right for the yielded word. The whole tree represents derivation structure; a path in a derivation graph is a different object.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Automata and Complexity Theory module | PDF page 8 | lines 30-31

> The nodes of the tree represent the
> symbols and the edges represent the use of production rules.

Automata and Complexity Theory module | PDF page 9 | lines 5-6

> The derivation tree is a graphical representation of the given production rules of context-free grammar (CFG).

Question sources

Addis Ababa University mock exam · Question 100

Question 11 | Q00329

A context free Grammar is called ambiguous if there exists a string that can have?

A. only one parse tree

B. no parse tree

C. Partial parse tree

D. more than one parse tree

Hint

Compare production restrictions, memory access and language-family containments. Distinguish an exact characterization from an included subclass.

Answer D: more than one parse tree

Ambiguity is witnessed by one terminal string having two distinct parse trees. It does not require every string to be ambiguous, and a string with no parse tree is simply not generated.

Why each choice is right or wrong

A - Incorrect: A unique tree is consistent with unambiguity.

B - Incorrect: No tree means failure to derive the string, not competing derivations.

C - Incorrect: A partial tree is unfinished and does not witness two full derivations.

D - Correct: Two or more full trees for the same string meet the definition.

Primary note proof

Study-note section AUT18

> A CFG is ambiguous when some word has two distinct parse trees, equivalently two different leftmost derivations. Different expansion orders that give the same tree do not prove ambiguity.

Study-note section AUT18

> Separate precedence levels: E→E+T|T, T→T*F|F, F→id|(E). Multiplication binds inside T before addition at E, and left recursion encodes left association. Uniqueness follows by the top-level last

operator split and induction. S→SS|ε is already ambiguous for ε. S→A|B is ambiguous if A→ab and B→abB|ε both produce ab. Ambiguous grammar and inherently ambiguous language differ: another grammar may remove ambiguity.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Automata and Complexity Theory module | PDF page 8 | lines 33-34

Grammar can be implemented with multiple parse
trees to get the same resulting string, this is said to be ambiguous.

Question sources

Computer Science model exam · Question 60

University of Gondar model exam · Question 60

Question 12 | Q00330

A context free grammar with production rules: S->aS / ¢ is equivalent to the regular expression

A. a*

B. (atb)*

C. a(at+b)*

D. a+

Hint

Track mandatory symbols and factors that may repeat or disappear. Use short test words to check both directions of each proposed language description.

Answer A: a*

The grammar is S→aS|ε. Applying the recursive rule n times followed by ε produces a^n for each n≥0, and no production creates any other terminal. This is exactly a*, including ε. Positive closure a+ omits ε.

Why each choice is right or wrong

A - Correct: Contains exactly all finite repetitions of a, including zero.

B - Incorrect: The expression (a+b)* also contains b.

C - Incorrect: The expression a(a+b)* contains ab and omits ε.

D - Incorrect: Requires at least one a, although S⇒ε is permitted.

Primary note proof

Study-note section AUT13

To require at least one a anywhere over {a,b}, use (a|b)*a(a|b)*. b*ab* requires exactly one a. At least two zeros uses (0|1)*0(0|1)*0(0|1)*; the zeros need not be adjacent. Testing 010 distinguishes it from a requirement for substring 00. For S→aS|ε, any number of a's is generated, giving a*.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Automata and Complexity Theory module | PDF page 5 | lines 22-22

In a regular expression, x* means zero or more occurrence of x.

Automata and Complexity Theory module | PDF page 8 | lines 27-28

R is a set of production rules where each production rule maps a variable to a string s∈(V∪Σ)∗;

Automata and Complexity Theory module | PDF page 2 | lines 24-25

A string with zero occurrences of symbols is known as an empty string. It is represented by ε.

Question sources

Computer Science model exam · Question 61

University of Gondar model exam · Question 61

Question 13 | Q00331

A push down automaton employs data structure.

A. Linked List

B. Stack

C. Hash Table

D. Queue

Hint

Compare production restrictions, memory access and language-family containments. Distinguish an exact characterization from an included subclass.

Answer B: Stack

A PDA augments finite control with a stack. Push and pop operations supply its unbounded LIFO memory.

Why each choice is right or wrong

A - Incorrect: A linked list is an implementation possibility, not the abstract PDA storage discipline.

B - Correct: A PDA uses stack memory.

C - Incorrect: Hash lookup is not the PDA memory mechanism.

D - Incorrect: A queue is FIFO, unlike the required pushdown stack.

Primary note proof

Study-note section AUT21

> A nondeterministic pushdown automaton (PDA) adds a stack to finite control: $(Q,\Sigma,\Gamma,\delta,q_0,Z_0,F)$. Its transition relation maps (state,input symbol or ε,stack top) to possible (new state,replacement stack string) pairs.

Study-note section AUT-S02

> A stack is last-in, first-out: push adds an item at the top and pop removes the top item, so the most recently pushed item is removed first. A PDA accesses this top rather than an arbitrary interior position.

Study-note section AUT-S02

> A queue is first-in, first-out: insertion at the rear and removal at the front returns the oldest waiting item. A linked list can implement a storage discipline, but the implementation container is different from the abstract push/pop or enqueue/dequeue access rule.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Automata and Complexity Theory module | PDF page 9 | lines 18-19

> Pushdown automata are nondeterministic finite state machines augmented with additional memory
> in the form of a stack,

Question sources

Computer Science model exam · Question 62

University of Gondar model exam · Question 62

Question 14 | Q00332

Which of the following Machine is Equivalent for Regular grammar?

A. Push down automata

B. Linear bounded automata

C. Finite state automata

D. Turing Machine

Hint

Compare production restrictions, memory access and language-family containments. Distinguish an exact characterization from an included subclass.

Answer C: Finite state automata

Regular grammars generate exactly the regular languages, which finite-state automata recognize. The visual hierarchy pairs the innermost regular class with finite-state automata. More powerful models can accept those languages too but are not equivalent in class-wide expressive power.

Why each choice is right or wrong

A - Incorrect: PDA languages include nonregular CFLs, so the whole model is more powerful.

B - Incorrect: The diagram pairs LBAs with the larger context-sensitive class.

C - Correct: Matches exactly the class generated by regular grammars.

D - Incorrect: The outer Turing machine class is larger.

Primary note proof

Study-note section AUT05

> Type 3 regular grammars correspond to finite automata; type 2 CFGs to nondeterministic PDAs; type 1 context-sensitive/noncontracting grammars to nondeterministic LBAs; type 0 unrestricted grammars to Turing recognizers.

Study-note section AUT05

> Use REG ⊊ CFL ⊊ CSL ⊊ DECIDABLE ⊊ RE, under the usual convention allowing S→ε with S absent from right sides when needed.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Automata and Complexity Theory module | PDF page 8 | lines 2-2

> The language defined by regular grammar is known as regular language.

Automata and Complexity Theory module | PDF page 7 | lines 16-16

> A language is said to be a Regular language if and only if some finite state machine recognizes it

Automata and Complexity Theory module | PDF page 10 | lines 6-6

> Fig: The traditional Chomsky hierarchy

Question sources

Computer Science model exam · Question 63

University of Gondar model exam · Question 63

Question 15 | Q00333

For a CFG G=(V,Σ,P,S), consider a production α→β. Treat a symbol also as its length-one string. Which statements hold? Select all that apply.

A. α∈(V∪Σ)

B. α∈V

C. α∈(V∪Σ)*

D. β∈(V∪Σ)*

Hint

Apply the stated transition, closure or equivalence rule step by step; distinguish a guaranteed bound from a value for one particular machine.

Answer A, B, C, D: α∈(V∪Σ); α∈V; α∈(V∪Σ)*; β∈(V∪Σ)*

A context-free production has exactly one variable on its left and a finite string of variables and terminals on its right. Therefore α∈V. Since V⊆V∪Σ, that same α belongs to the union and, as a one-symbol string, to its Kleene closure. The right-hand side β belongs to (V∪Σ)*, including the possibility of ε. All four memberships hold.

Why each choice is right or wrong

A - Correct: A variable is also an element of V∪Σ.

B - Correct: The left side of a CFG rule is one variable.

C - Correct: That variable is a length-one string in (V∪Σ)*.

D - Correct: The right side is a finite string over variables and terminals.

Primary note proof

Study-note section AUT16

> In a context-free grammar (CFG) each production has one variable on the left, A→α with α∈(V∪Σ)*. The left side therefore also belongs to V∪Σ and its string closure; alternatives stating these weaker memberships can all be true. A sentential form is any string derived from S, including terminal-only sentences and S itself.

Study-note section AUT16

> A leftmost derivation always replaces the leftmost variable; a rightmost one replaces the rightmost variable. This is not scanning the completed input in opposite directions. A complete parse tree has root S, internal variable nodes expanded by productions and terminal leaves, with ε leaves when that convention is used. Read the leaves left-to-right for the yielded word. The whole tree represents derivation structure; a path in a derivation graph is a different object.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Automata and Complexity Theory module | PDF page 8 | lines 27-28

R is a set of production rules where each production rule maps a variable to a string $s \in (V \cup \Sigma)*$;

Automata and Complexity Theory module | PDF page 8 | lines 22-23

Context-free grammars have a start symbol and set of terminals, non-terminals, and production rule components.

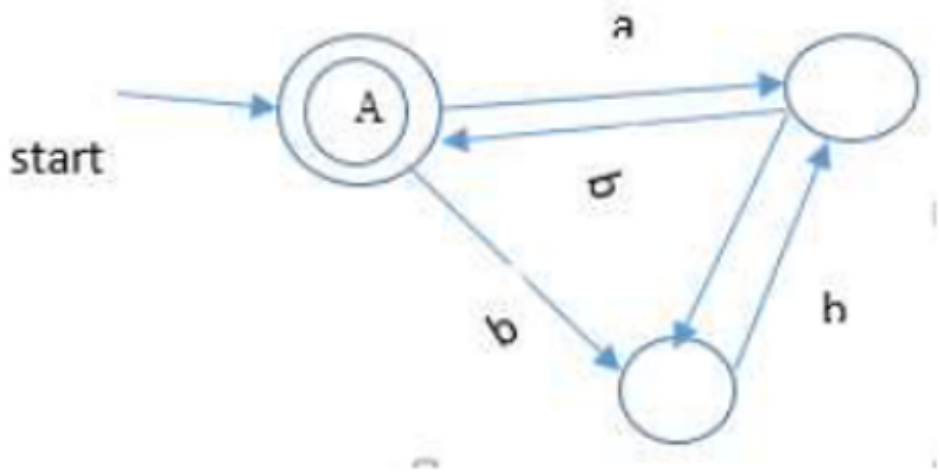
Question sources

Computer Science model exam · Question 64

University of Gondar model exam · Question 64

Question 16 | Q00334

For the diagram, name the top-right state B and bottom state C. Use A as start and sole accepting state; transitions are A-a→B, A-b→C, B-a→C, B-b→A, C-b→B. Unshown transitions reject. Which regular expression denotes its language? Here + means union.



A. a*(a+b)ab(ab+bb+aa*(a+b)ab)*

B. ((a+bb)(ab)*b)*

C. (a+b)ab(ab+bb+aa*(a+b)ab)*

D. a*ab(ab+bb+aa*(a+b)ab)*

Hint

Apply the stated transition, closure or equivalence rule step by step; distinguish a guaranteed bound from a value for one particular machine.

Answer B: ((a+bb)(ab)*b)*

A return to A first reaches B either directly by a or via C by bb. While at B, the machine may make any number of ab detours through C. It returns to A by b. Thus one return has form (a+bb)(ab)*b, and any number of returns gives ((a+bb)(ab)*b)*. This also accepts ε and ab, as the accepting start and shortest return require.

Why each choice is right or wrong

A - Incorrect: Its mandatory prefix excludes ε, which must be accepted.

B - Correct: Its starred blocks describe exactly all excursions from A back to A.

C - Incorrect: Its mandatory prefix excludes the empty word.

D - Incorrect: Its mandatory ab suffix excludes ε.

Primary note proof

Study-note section AUT12

> Classical expressions use ∅, ε, symbols, union, concatenation and star. A union B is written A|B or A+B; postfix r^+ means one-or-more r repetitions and is a different use of +.

Study-note section AUT13

> For automaton→RE, eliminate a state k while updating edge i→j as $r_{ij} \mid r_{ik}(r_{kk})^*r_{kj}$; include new start/final nodes where required.

Study-note section AUT06

> Start at q_0 and consume the entire input. Accept iff the ending state is in F; visiting an accepting state earlier is insufficient. ε is accepted exactly when $q_0 \in F$.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Automata and Complexity Theory module | PDF page 3 | lines 25-25

> A finite automaton is a collection of 5-tuple (Q, ∑, δ, q0, F), where:

Automata and Complexity Theory module | PDF page 5 | lines 22-22

> In a regular expression, x* means zero or more occurrence of x.

Automata and Complexity Theory module | PDF page 7 | lines 16-16

> A language is said to be a Regular language if and only if some finite state machine recognizes it

Question sources

Computer Science model exam · Question 65

University of Gondar model exam · Question 65

Question 17 | Q00634

Which one of the following regular expressions gives zero or one?

A. r + =rr*

B. r* = r+ | ε

C. r? =r|ε

D. All

Hint

Apply the stated transition, closure or equivalence rule step by step; distinguish a guaranteed bound from a value for one particular machine.

Answer C: The optional expression r? = r|ε.

AUT12 defines r? as r|ε: one occurrence of a word matching r or no occurrence. Positive closure requires one or more occurrences; star permits arbitrarily many including zero. Their identities may be valid, but only the optional expression denotes the requested zero-or-one repetition count, so the aggregate is false.

Why each choice is right or wrong

A - Incorrect: Positive closure rr* requires at least one repetition and also allows more than one.

B - Incorrect: Star allows zero or arbitrarily many repetitions, not just zero or one.

C - Correct: The union with ε makes r optional: either one r match or none.

D - Incorrect: The positive and Kleene closures do not impose the zero-or-one bound.

Primary note proof

Study-note section AUT12

> Classical expressions use ∅, ε, symbols, union, concatenation and star. A union B is written A|B or A+B; postfix r^+ means one-or-more r repetitions and is a different use of +. Parenthesize when conventions might clash. Star binds tightest, concatenation next, union last. r? means r|ε; r^+=rr*.

Study-note section AUT12

> (a|b)*a denotes words ending a; 0*10* denotes exactly one 1; a*(b|c) gives zero-or-more a's then exactly one b or c, so aab belongs but aaa and aaabc do not.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Automata and Complexity Theory module | PDF page 5 | lines 22-22

> In a regular expression, x* means zero or more occurrence of x.

Automata and Complexity Theory module | PDF page 5 | lines 25-25

> means one or more occurrence of x. It can generate {x, xx, xxx, xxxx,

Question sources

Computer Science model exam · 2015 · Question 103

Question 18 | Q00751

If a grammar produces more than one leftmost derivation for a sentence, it is called as:

A. Ambiguous grammar

B. Perfect grammar

C. Associative grammar

D. Precedence

Hint

Compare production restrictions, memory access and language-family containments. Distinguish an exact characterization from an included subclass.

Answer A: Ambiguous grammar

Each parse tree determines a unique leftmost sequence of expansions. Two distinct leftmost derivations of the same sentence therefore encode distinct parse trees and witness ambiguity under this definition.

Why each choice is right or wrong

A - Correct: Two leftmost derivations of one sentence witness two parse trees.

B - Incorrect: Perfect is not the relevant defined grammar property.

C - Incorrect: Associativity is an operator grouping convention, not this definition.

D - Incorrect: Precedence concerns ordering of operators; it does not name multiple leftmost derivations.

Primary note proof

Study-note section AUT18

> A CFG is ambiguous when some word has two distinct parse trees, equivalently two different leftmost derivations. Different expansion orders that give the same tree do not prove ambiguity.

Study-note section AUT18

> Separate precedence levels: E→E+T|T, T→T*F|F, F→id|(E). Multiplication binds inside T before addition at E, and left recursion encodes left association. Uniqueness follows by the top-level last operator split and induction. S→SS|ε is already ambiguous for ε. S→A|B is ambiguous if A→ab and B→abB|ε both produce ab. Ambiguous grammar and inherently ambiguous language differ: another grammar may remove ambiguity.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Automata and Complexity Theory module | PDF page 8 | lines 33-34

> Grammar can be implemented with multiple parse
> trees to get the same resulting string, this is said to be ambiguous.

Automata and Complexity Theory module | PDF page 9 | lines 7-8

> in the leftmost derivation, the input is
> scanned and replaced with the production rule from left to right.

Question sources

Computer Science model exam 2 · Question 92

Question 19 | Q00752

The intersection of a context-free language and a regular language is _____

A. context-free

B. neither context-free nor regular.

C. regular but not context-free

D. both regular and context-free.

Hint

Compare production restrictions, memory access and language-family containments. Distinguish an exact characterization from an included subclass.

Answer A: Always context-free, but not necessarily regular.

AUT23 constructs the intersection by combining PDA control with DFA control and retaining the original stack. It therefore guarantees a CFL. Regularity does not follow: intersecting $\{a^n b^n\}$ with a*b* gives the same nonregular CFL. Some individual intersections can be regular, but the question's general result cannot assert that extra property.

Why each choice is right or wrong

A - Correct: Closure under intersection with a regular language guarantees that the result is context-free.

B - Incorrect: The stated closure theorem directly rules out a result outside the context-free family.

C - Incorrect: Every regular language is context-free, so regular but not context-free is impossible.

D - Incorrect: Some intersections are regular, but the counterexample in AUT23 shows regularity is not guaranteed.

Primary note proof

Study-note section AUT23

> CFLs are closed under union, concatenation, star, reversal and intersection with a regular language. For CFL∩REG combine PDA control with DFA control, using the original stack; each consumed symbol updates both states, while an ε move leaves the DFA state unchanged. The result is guaranteed context-free, not necessarily regular: intersect a^nb^n with a*b* to get a^nb^n.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Automata and Complexity Theory module | PDF page 9 | lines 23-23

> Pushdown automata accept context-free languages, which include the set of regular languages.

Automata and Complexity Theory module | PDF page 7 | lines 16-16

> A language is said to be a Regular language if and only if some finite state machine recognizes it

Question sources

Computer Science model exam 2 · Question 93

Question 20 | Q00753

Which one of the following is not part of Turing machine?

A. Finite control

B. Input

C. Tape

D. Head

Hint

Compare production restrictions, memory access and language-family containments. Distinguish an exact characterization from an included subclass.

Answer B: Input is data, not a separate hardware component.

AUT27 lists finite control, a read/write head and an unbounded tape as the standard TM architecture. Input is initial tape content rather than a separate hardware organ. Thus input is the distinction requested by not part, under that structural reading; the other three options are actual machine components, not merely values processed by them.

Why each choice is right or wrong

A - Incorrect: Finite control is explicitly part of the standard machine's architecture.

B - Correct: Input is initial tape content rather than a separate machine hardware component.

C - Incorrect: The unbounded tape is the standard TM's storage component.

D - Incorrect: The read/write head is an explicit machine component used for tape access.

Primary note proof

Study-note section AUT27

> For a Turing machine (TM), use $M=(Q,\Sigma,\Gamma,\delta,q_0,q\text{accept},q\text{reject})$, with finite state/tape alphabets, $\Sigma\subseteq\Gamma$ and blank $\square\in\Gamma-\Sigma$. The blank symbol belongs to the tape alphabet Γ, not the input alphabet Σ. The standard machine has finite control, a read/write head and an unbounded tape. Input is initial tape content, not a separate hardware organ; no separate stack is required.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Automata and Complexity Theory module | PDF page 11 | lines 9-9

> It has a tape head that can read and write symbols and move around on the tape.

Automata and Complexity Theory module | PDF page 11 | lines 20-22

> δ is the transition function, mapping (q, x) onto (q', y, D) where D denotes the direction of movement of R/W head: D = L or R according as the movement is to the left or right.

Question sources

Computer Science model exam 2 · Question 94

Question 21 | Q00891

Which automaton is powerful enough to recognize the language L = $\{a^nb^nc^n \mid n > 1\}$, where n is an integer?

A. DFA

B. Linear Bounded Automaton (LBA)

C. NFA

D. Pushdown Automaton (PDA)

Hint

Respect the quantifier order and length constraints. A valid nonmembership argument must address every allowed decomposition, not one convenient split.

Answer B: Linear Bounded Automaton (LBA).

The language contains equal numbers of a, b and c in that order. This family is context-sensitive but not context-free. Restricting n to integers greater than 1 does not remove the unbounded equal-count requirement. An LBA checks the order and repeatedly marks one a, one b and one c, using only the input-sized tape region. B therefore recognizes the stated language; DFA, NFA and single-stack PDA cannot.

Why each choice is right or wrong

A - Incorrect: A DFA recognizes regular languages; this three-equal-count language is not even context-free.

B - Correct: An LBA can mark matching a, b and c symbols while staying within space proportional to the input length.

C - Incorrect: An NFA recognizes exactly the regular languages, so nondeterminism does not give it the memory needed here.

D - Incorrect: A standard single-stack PDA recognizes context-free languages; the stated three-equal-count language is not context-free.

Primary note proof

Study-note section AUT25

A context-sensitive rule $\alpha A\beta \to \alpha\gamma\beta$ replaces A in context, with γ nonempty. Equivalently, noncontracting grammar presentations preserve/increase total length, with the isolated ε exception. The condition is on the complete sides, not an unrelated comparison $|\beta| \geq |\alpha|$.

Study-note section AUT25

A linear bounded automaton (LBA) is a TM with O(n) tape cells on length-n input; the standard CSL characterization uses nondeterministic LBAs. For $a^n b^n c^n$ first verify a*b*c* order, repeatedly mark one unmarked a, b and c, and reject missing/excess symbols.

Study-note section AUT25

Thus every CSL is decidable.

Study-note section AUT24

If L is context-free, some p ensures each sufficiently long $s \in L$ has $s=uvxyz$ with $|vxy| \leq p$, $|vy| \geq 1$ and $uv^i xy^i z \in L$ for every $i \geq 0$. The two pieces v and y are pumped together; either may be empty, not both. A non-CFL proof must defeat every allowed decomposition.

Study-note section AUT24

For $s=a^p b^p c^p$, the window vxy cannot contain both an a and a c. Pumping affects at most two blocks and at least one count. Use i=0 or 2: the unaffected third block retains p while another count changes, or symbol order breaks. Hence $a^n b^n c^n$ is not CFL.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Automata and Complexity Theory module | PDF page 10 | lines 6-6

Fig: The traditional Chomsky hierarchy

Automata and Complexity Theory module | PDF page 9 | lines 18-19

Pushdown automata are nondeterministic finite state machines augmented with additional memory
in the form of a stack,

Question sources

Computer Science Exit Exam · 2018 · Question 4

Question 22 | Q00917

A recognizer reads input once from left to right and stores all its working memory in a fixed finite set of states. Which universal statement correctly explains why its language can also be described by a context-free grammar?

A. Every regular language is context-free.

B. Every context-free language is regular.

C. Every context-sensitive language is regular.

D. Every recursively enumerable language is recursive

Hint

Check whether the required procedure must halt on every input, and distinguish solving a task from recognizing its yes-instances.

Answer A: Every regular language is context-free.

A recognizer whose memory is a fixed finite state has regular-language power. Every regular language is context-free, so a context-free grammar can also describe its language. The reverse inclusion fails: equal-count matching such as {0^n1^n} is context-free but not regular. Context-sensitive languages need not be regular either. Finally, the halting language separates recognizable from decidable languages, so not every recursively enumerable language is recursive.

Why each choice is right or wrong

A - Correct: Every regular language has an equivalent context-free grammar.

B - Incorrect: {0^n1^n : n≥0} is context-free but not regular.

C - Incorrect: The context-sensitive family includes nonregular languages, including equal-count matching languages.

D - Incorrect: The halting language is recursively enumerable but undecidable, disproving this converse.

Primary note proof

Study-note section AUT05

> Type 3 regular grammars correspond to finite automata; type 2 CFGs to nondeterministic PDAs; type 1 context-sensitive/noncontracting grammars to nondeterministic LBAs; type 0 unrestricted grammars to Turing recognizers.

Study-note section AUT05

> Use REG ⊊ CFL ⊊ CSL ⊊ DECIDABLE ⊊ RE, under the usual convention allowing S→ε with S absent from right sides when needed.

Study-note section AUT29

A decider halts on every input and accepts exactly its language; these languages are recursive/decidable. A recognizer must halt and accept every member, but on a nonmember may halt-reject or loop. Its languages are recursively enumerable (RE), Turing-recognizable or semidecidable. Looping is nonacceptance, not a rejecting halt. An undecidable language may still have a recognizer.

Study-note section AUT29

Every decidable language is RE, and the containment is strict.

Study-note section AUT06

Finite automata have finite memory encoded by states, not no state or no ability to remember any history.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Automata and Complexity Theory module | PDF page 8 | lines 20-21

A context-free grammar
can describe all regular languages and more, but they cannot describe all possible languages.

Automata and Complexity Theory module | PDF page 10 | lines 6-6

Fig: The traditional Chomsky hierarchy

Automata and Complexity Theory module | PDF page 15 | lines 30-30

All recursive set is recursively enumerable.

Question sources

Computer Science Exit Exam · 2018 · Question 30

Question 23 | Q00934

In the CNF-first route for converting a CFG into Greibach normal form, which initial normalization is performed before the later ordering, substitution and left-recursion work?

A. Convert the grammar into CNF

B. If any production rule in the grammar is not in GNF form, convert it.

C. .
Convert the grammar into GNF

D. If the grammar exists left recursion, eliminate it.

Hint

Compare production restrictions, memory access and language-family containments. Distinguish an exact characterization from an included subclass.

Answer A: Convert the grammar into CNF

The specified route begins by putting the grammar into Chomsky normal form. Later ordering and substitution arrange terminal-leading productions and eliminate left recursion as required for Greibach form. CNF is an intermediate normalization in this particular route, not a universal prerequisite for every GNF construction. The other conversion choices state the goal broadly rather than identify this route's initial normalization.

Why each choice is right or wrong

A - Correct: CNF conversion is the initial normalization in the selected route.

B - Incorrect: Converting arbitrary offending rules does not identify the route's initial normalization.

C - Incorrect: GNF conversion is the overall goal, not this preliminary operation.

D - Incorrect: Left recursion must be handled later in this chosen route.

Primary note proof

Study-note section AUT20

> Some algorithms first convert to CNF; this is a method choice, not a universally mandatory first step.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Automata and Complexity Theory module | PDF page 8 | lines 27-28

> R is a set of production rules where each production rule maps a variable to a string $s \in (V \cup \Sigma)*$;

Automata and Complexity Theory module | PDF page 9 | lines 2-2

> A sentential form is any string derivable from the start symbol.

Automata and Complexity Theory module | PDF page 9 | lines 15-16

> Simplification essentially comprises the following steps; Reduction of CFG, Removal of Unit Productions, and Removal of Null Productions.

Question sources

Computer Science Exit Exam · 2018 · Question 47

Question 24 | Q00953

Among the listed grammar families in the Chomsky hierarchy, which has the greatest expressive power?

A. Context-free

B. Context-sensitive

C. Unrestricted (Type 0)

D. Regular

Hint

Compare production restrictions, memory access and language-family containments. Distinguish an exact characterization from an included subclass.

Answer C: Unrestricted (Type 0)

Unrestricted, Type 0 grammars generate the recursively enumerable languages, the family recognized by Turing machines. Context-sensitive, context-free and regular grammars generate successively smaller families. Asking for the most expressive listed family makes this an exact hierarchy comparison. It does not claim that every conceivable set of strings is recursively enumerable or that weaker grammars cannot generate complete languages of their own.

Why each choice is right or wrong

A - Incorrect: Context-free grammars generate a smaller family than RE.

B - Incorrect: Context-sensitive grammars generate a smaller family than RE.

C - Correct: Unrestricted grammars generate exactly RE.

D - Incorrect: Regular grammars generate the least expressive listed family.

Primary note proof

Study-note section AUT05

> Type 3 regular grammars correspond to finite automata; type 2 CFGs to nondeterministic PDAs; type 1 context-sensitive/noncontracting grammars to nondeterministic LBAs; type 0 unrestricted grammars to Turing recognizers.

Study-note section AUT05

> Use REG ⊊ CFL ⊊ CSL ⊊ DECIDABLE ⊊ RE, under the usual convention allowing S→ε with S absent from right sides when needed.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Automata and Complexity Theory module | PDF page 3 | lines 13-14

> A grammar defines a set of rules, and with the use of these rules, valid sentences in a language are
> generated.

Automata and Complexity Theory module | PDF page 8 | lines 20-21

> A context-free grammar
> can describe all regular languages and more, but they cannot describe all possible languages.

Automata and Complexity Theory module | PDF page 10 | lines 6-6

> Fig: The traditional Chomsky hierarchy

Question sources

Computer Science Exit Exam · 2018 · Question 66

Question 25 | Q00977

ou have given that Grammar G1 -({S, A, B}, {a, b}, S, {S - AB, A- a, B-> b}).
Which one is terminal symbol in the give?

A. b

B. S

C. .
S- AB,A--a,B-b

D. S,A,B

Hint

Distinguish symbols, finite strings, their lengths and sets of strings. Apply the operation to its stated operands, including empty cases.

Answer A: b

The given terminal alphabet is {a,b}, so b is a terminal. S, A and B are variables, and the arrow expressions are productions. These categories match the four-component CFG definition.

Why each choice is right or wrong

A - Correct: b belongs to the given terminal alphabet.

B - Incorrect: S belongs to the variables and is the start symbol.

C - Incorrect: This is a set/list of productions, not one terminal symbol.

D - Incorrect: These are the variables, not the terminals.

Primary note proof

Study-note section AUT04

> Write G=(V,Σ,P,S): finite variables V, terminals Σ disjoint from V, productions P and start variable S∈V. A production replaces its left side by its right side. The language is L(G)={w∈Σ*:S⇒*w}. A recognizer processes a word and accepts exactly the language members. Grammars generate languages; automata recognize languages, not grammars.

Study-note section AUT04

> Two grammars are equivalent when their generated languages are equal, regardless of rule counts or variable names. One common word, equal alphabets or one-way inclusion is insufficient.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Automata and Complexity Theory module | PDF page 8 | lines 22-23

> Context-free grammars have a start symbol and set of terminals, non-terminals, and production rule components.

Automata and Complexity Theory module | PDF page 8 | lines 27-28

> R is a set of production rules where each production rule maps a variable to a string $s \in (V \cup \Sigma)*$;

Question sources

Computer Science Exit Exam · 2018 · Question 90

Question 26 | Q00981

Over $\Sigma=\{a,b\}$, classify $L=\{ww^R \mid w \in \Sigma^*\}$, the unmarked even-length palindrome language, by the most specific listed description.

A. Regular

B. Non-deterministic CFL but not deterministic

C. Deterministic context-free

D. Context-sensitive but not context-free

Hint

Compare production restrictions, memory access and language-family containments. Distinguish an exact characterization from an included subclass.

Answer B: Non-deterministic CFL but not deterministic

An NPDA can push a guessed first half, choose its midpoint nondeterministically, and pop matching symbols while reading the reverse half. This binary unmarked palindrome language is not deterministic context-free. It is context-free and nonregular. With exponent R denoting reversal, ww^R is the stated palindrome language; the forward-copy language ww would instead fall outside CFL.

Why each choice is right or wrong

A - Incorrect: Binary unmarked even palindromes are nonregular.

B - Correct: An NPDA recognizes them, but a DPDA cannot recognize this unmarked binary language.

C - Incorrect: The missing center marker prevents deterministic context-free recognition.

D - Incorrect: The language has a CFG, so the claim that it is not context-free is false.

Primary note proof

Study-note section AUT22

> For even palindromes ww^R over {a,b}, push the first half, nondeterministically guess its midpoint, then pop matching symbols. No center marker makes that language context-free but not deterministic context-free. With a marker wcw^R the switch point is visible and a DPDA suffices. Unary even palindromes are merely even-length words and regular. A PDA cannot compare two arbitrary halves in the same forward order: {ww:w∈{a,b}*} is the copy language, not the palindrome language.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Automata and Complexity Theory module | PDF page 8 | lines 27-28

> R is a set of production rules where each production rule maps a variable to a string s∈(V∪Σ)∗;

Automata and Complexity Theory module | PDF page 10 | lines 6-6

> Fig: The traditional Chomsky hierarchy

Question sources

Computer Science Exit Exam · 2018 · Question 94

Question 27 | Q01185

Which of the following statements is false about classes of language in automaton theory?

A. Context-free language is the subset of context-sensitive language

B. Regular language is the subset of context-sensitive language

C. Recursively enumerable language is the superset of regular language

D. Context-sensitive language is a subset of context-free language

Hint

Compare production restrictions, memory access and language-family containments. Distinguish an exact characterization from an included subclass.

Answer D: Context-sensitive language is a subset of context-free language

The inspected diagram nests regular inside context-free inside context-sensitive inside recursively enumerable. Thus A, B and C follow the depicted inclusions, while D reverses the middle inclusion. This uses the traditional hierarchy convention, which suppresses technical empty-word qualifications.

Why each choice is right or wrong

A - Incorrect: Context-free is drawn inside context-sensitive.

B - Incorrect: Regular is inside context-free and hence inside context-sensitive.

C - Incorrect: RE is the outer class surrounding regular.

D - Correct: Reverses the diagram: context-sensitive is not contained in the smaller CFL class.

Primary note proof

Study-note section AUT05

> Type 3 regular grammars correspond to finite automata; type 2 CFGs to nondeterministic PDAs; type 1 context-sensitive/noncontracting grammars to nondeterministic LBAs; type 0 unrestricted grammars to Turing recognizers.

Study-note section AUT05

> Use REG ⊊ CFL ⊊ CSL ⊊ DECIDABLE ⊊ RE, under the usual convention allowing S→ε with S absent from right sides when needed.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Automata and Complexity Theory module | PDF page 10 | lines 6-6

> Fig: The traditional Chomsky hierarchy

Automata and Complexity Theory module | PDF page 8 | lines 20-21

> A context-free grammar
> can describe all regular languages and more, but they cannot describe all possible languages.

Question sources

Computer Science model exam 2 · Question 6

Question 28 | Q01206

Suppose you are asked to construct a DFA for the strings ending with an 'n'
length substring. How many minimal states may the DFA require?

A. n

B. n+1

C. n+2

D. n*2

Hint

Read the transition domain and acceptance condition carefully. Track which inputs are consumed and evaluate the whole word under the stated model.

Answer B: n+1 states for a fixed valid suffix word.

AUT08 gives n+1 states for a valid fixed suffix of length n: states remember the longest suffix of the consumed input that is a prefix of the target. Distinguishable prefixes establish the lower bound. The answer presumes the target word uses the declared alphabet; no extra rejecting sink is needed for a suffix matcher.

Why each choice is right or wrong

A - Incorrect: n states omit one of the n+1 necessary prefix-match stages.

B - Correct: The note gives one state for each matched-prefix length from zero through n.

C - Incorrect: An extra rejecting sink is unnecessary because later input may always form the required suffix.

D - Incorrect: Doubling n is not the general suffix-matching state bound; accidental numeric equality does not define it.

Primary note proof

Study-note section AUT08

> For a fixed valid word w of length n, a DFA for strings ending in w needs n+1 states representing the longest consumed suffix that is a prefix of w. Overlapping prefixes determine fallback transitions. Distinct prefixes are distinguishable by suitable continuations, proving the lower bound; do not add an unnecessary rejecting sink.

Study-note section AUT08

> For strings starting with w, a complete DFA ordinarily has n matched-prefix stages, an accepting continuation state and a rejecting sink: n+2 when n≥1 and the alphabet allows mismatches.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Automata and Complexity Theory module | PDF page 5 | lines 3-5

> DFA refers to deterministic finite automata. Deterministic refers to the uniqueness of the computation. In the DFA, the machine goes to one state only for a particular input character. DFA does not accept the null move.

Automata and Complexity Theory module | PDF page 7 | lines 16-16

> A language is said to be a Regular language if and only if some finite state machine recognizes it

Question sources

Computer Science model exam 2 · Question 27

Question 29 | Q01223

Under the standard total-DFA model, which statements about finite automata are NOT correct? Select all that apply.

A. Finite automaton accept a string or word by final state

B. Finite Automaton has at least one final state

C. Finite Automaton has at least one initial state

D. Deterministic Finite Automaton has a single transition.

Hint

Read the transition domain and acceptance condition carefully. Track which inputs are consumed and evaluate the whole word under the stated model.

Answer B, D: Finite Automaton has at least one final state; Deterministic Finite Automaton has a single transition.

A DFA requires a designated start state and accepts when the state after the complete input is accepting. Its accepting set may nevertheless be empty, producing the empty language, so B is false. Determinism requires one successor for each state/symbol pair, not one transition in the entire automaton, making D false as well. The two remaining statements describe standard acceptance and initialization.

Why each choice is right or wrong

A - Incorrect: Final-state acceptance after all input is consumed is correct, so A is not selected.

B - Correct: The accepting set can be empty, making the asserted minimum of one false.

C - Incorrect: A designated initial state is required, so C is not selected.

D - Correct: One successor per state/symbol pair does not mean one transition in the entire machine.

Primary note proof

Study-note section AUT06

> A deterministic finite automaton (DFA) is $M=(Q,\Sigma,\delta,q_0,F)$, with finite nonempty Q, start $q_0 \in Q$, accepting set $F \subseteq Q$ and total $\delta: Q \times \Sigma \to Q$. Exactly one next state is specified per state/symbol pair; this does not mean the whole machine has one transition. F may be empty, recognizing $\varnothing$.

Study-note section AUT06

> Start at q_0 and consume the entire input. Accept iff the ending state is in F; visiting an accepting state earlier is insufficient. ε is accepted exactly when $q_0 \in F$. A nonaccepting sink loops on every symbol and cannot reach acceptance; it still finishes after finite input. Finite automata have finite memory encoded by states, not no state or no ability to remember any history.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Automata and Complexity Theory module | PDF page 3 | lines 25-25

A finite automaton is a collection of 5-tuple (Q, ∑, δ, q0, F), where:

Automata and Complexity Theory module | PDF page 5 | lines 3-5

DFA refers to deterministic finite automata. Deterministic refers to the uniqueness of the computation. In the DFA, the machine goes to one state only for a particular input character. DFA does not accept the null move.

Question sources

Computer Science model exam 2 · Question 44

Question 30 | Q01253

Finite automaton recognizes:

A. Any class of language.

B. Context-free language.

C. Context-sensitive languages.

D. Regular languages.

Hint

Read the transition domain and acceptance condition carefully. Track which inputs are consumed and evaluate the whole word under the stated model.

Answer D: Regular languages.

Regular languages are exactly those recognized by finite-state automata. This is an exact language-family question: some CFLs and context-sensitive languages happen to be regular, but the whole larger families are beyond finite automata.

Why each choice is right or wrong

A - Incorrect: The hierarchy shows larger families outside finite-state power.

B - Incorrect: All CFLs is too broad, though regular examples are CFLs.

C - Incorrect: All context-sensitive languages is too broad.

D - Correct: Exactly matches the finite-state/regular equivalence.

Primary note proof

Study-note section AUT06

A deterministic finite automaton (DFA) is $M=(Q,\Sigma,\delta,q_0,F)$, with finite nonempty Q, start $q_0 \in Q$, accepting set $F \subseteq Q$ and total $\delta: Q \times \Sigma \to Q$. Exactly one next state is specified per state/symbol pair; this does not mean the whole machine has one transition. F may be empty, recognizing ∅.

Study-note section AUT06

Start at q_0 and consume the entire input. Accept iff the ending state is in F; visiting an accepting state earlier is insufficient. ε is accepted exactly when $q_0 \in F$. A nonaccepting sink loops on every

> symbol and cannot reach acceptance; it still finishes after finite input. Finite automata have finite memory encoded by states, not no state or no ability to remember any history.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Automata and Complexity Theory module | PDF page 7 | lines 16-16

> A language is said to be a Regular language if and only if some finite state machine recognizes it

Automata and Complexity Theory module | PDF page 8 | lines 20-21

> A context-free grammar
> can describe all regular languages and more, but they cannot describe all possible languages.

Automata and Complexity Theory module | PDF page 10 | lines 6-6

> Fig: The traditional Chomsky hierarchy

Question sources

Computer Science model exam 2 · Question 74

Question 31 | Q01273

Which one of the following grammars is unambiguous?

A. E → I; E → E + E; E → E * E; E → (E); I → ε | 0 | 1 | 2 | ... | 9

B. S → aSb | SS; S → ε

C. E → E + T / T; T → T x F / F; F → id

D. S → A / B; A → aAb / ab; B → abB / ∈

Hint

Compare production restrictions, memory access and language-family containments. Distinguish an exact characterization from an included subclass.

Answer C: E → E + T / T; T → T x F / F; F → id

In C, F generates id, T is a nonempty sequence of Fs separated by ×, and E a nonempty sequence of Ts separated by +. The last + uniquely determines the top E→E+T split; each T is uniquely split at its last ×. Induction on operator count gives a unique parse. A allows different groupings of id-like operands through E+E and E*E. B already has two trees for ε: S→ε and S→SS→εε. D is also ambiguous: ab is generated by S→A→ab and S→B→abB→ab.

Why each choice is right or wrong

A - Incorrect: For 1+1*1, E→E+E permits a top + parse while E→E*E permits a top * parse; I→1 supplies the operands. These are different trees for the same string (PDF page 8).

B - Incorrect: ε has both a direct leaf expansion and an SS expansion with two empty children.

C - Correct: Operator levels fix the unique split at each stage, giving a unique tree.

D - Incorrect: ab is obtained via either A or B, yielding distinct root expansions.

Primary note proof

Study-note section AUT18

> A CFG is ambiguous when some word has two distinct parse trees, equivalently two different leftmost derivations. Different expansion orders that give the same tree do not prove ambiguity.

Study-note section AUT18

> Separate precedence levels: E→E+T|T, T→T*F|F, F→id|(E). Multiplication binds inside T before addition at E, and left recursion encodes left association. Uniqueness follows by the top-level last operator split and induction. S→SS|ε is already ambiguous for ε. S→A|B is ambiguous if A→ab and B→abB|ε both produce ab. Ambiguous grammar and inherently ambiguous language differ: another grammar may remove ambiguity.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Automata and Complexity Theory module | PDF page 8 | lines 33-34

> Grammar can be implemented with multiple parse
> trees to get the same resulting string, this is said to be ambiguous.

Automata and Complexity Theory module | PDF page 8 | lines 27-28

> R is a set of production rules where each production rule maps a variable to a
> string s∈(V∪Σ)*;

Question sources

Computer Science model exam 2 · Question 94

Question 32 | Q01418

Which language family is recognized exactly by nondeterministic pushdown automata?

A. Type 0 language

B. Type 1 language

C. Type 2 language

D. Type 3 language

Hint

Compare production restrictions, memory access and language-family containments. Distinguish an exact characterization from an included subclass.

Answer C: Type 2 language

Nondeterministic pushdown automata recognize exactly the context-free languages, corresponding to Type 2 grammars. Regular languages are included in that family, but Type 3 is too narrow to describe all PDA languages. Context-sensitive and unrestricted languages include non-context-free languages that an ordinary single-stack PDA cannot recognize. The exact-family wording therefore makes Type 2 the unique classification.

Why each choice is right or wrong

A - Incorrect: Type 0 includes languages beyond PDA recognition.

B - Incorrect: Type 1 includes non-context-free languages beyond PDA recognition.

C - Correct: Type 2 is exactly the context-free family characterized by NPDAs.

D - Incorrect: Type 3 covers only the regular subfamily of PDA languages.

Primary note proof

Study-note section AUT05

> Type 3 regular grammars correspond to finite automata; type 2 CFGs to nondeterministic PDAs; type 1 context-sensitive/noncontracting grammars to nondeterministic LBAs; type 0 unrestricted grammars to Turing recognizers.

Study-note section AUT05

> Use REG ⊊ CFL ⊊ CSL ⊊ DECIDABLE ⊊ RE, under the usual convention allowing S→ε with S absent from right sides when needed.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Automata and Complexity Theory module | PDF page 9 | lines 23-23

> Pushdown automata accept context-free languages, which include the set of regular languages.

Automata and Complexity Theory module | PDF page 10 | lines 6-6

> Fig: The traditional Chomsky hierarchy

Question sources

Compiler Design review questions · Question 26

Question 33 | Q01419

Among context-sensitive, context-free and regular grammar families, which is most expressive under the usual isolated-start ε convention?

A. Context-sensitive grammar

B. Context-free grammar

C. Regular grammar

D. All of these

Hint

Compare production restrictions, memory access and language-family containments. Distinguish an exact characterization from an included subclass.

Answer A: Context-sensitive grammar

The hierarchy places regular languages strictly inside context-free languages and context-free languages strictly inside context-sensitive languages. Context-sensitive grammars are therefore the most expressive of the three named families. Unrestricted grammars would be stronger if included, but the question explicitly limits the comparison. The three named classes are not tied for maximum expressive power, so the aggregate alternative does not apply.

Why each choice is right or wrong

A - Correct: Context-sensitive is the largest of the three compared families.

B - Incorrect: Context-free is strictly below context-sensitive.

C - Incorrect: Regular is strictly below both other named families.

D - Incorrect: The three families are not equally most expressive.

Primary note proof

Study-note section AUT05

> Type 3 regular grammars correspond to finite automata; type 2 CFGs to nondeterministic PDAs; type 1 context-sensitive/noncontracting grammars to nondeterministic LBAs; type 0 unrestricted grammars to Turing recognizers.

Study-note section AUT05

> Use REG ⊊ CFL ⊊ CSL ⊊ DECIDABLE ⊊ RE, under the usual convention allowing S→ε with S absent from right sides when needed.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Automata and Complexity Theory module | PDF page 3 | lines 13-14

> A grammar defines a set of rules, and with the use of these rules, valid sentences in a language are
> generated.

Automata and Complexity Theory module | PDF page 10 | lines 6-6

Fig: The traditional Chomsky hierarchy

Question sources

Compiler Design review questions · Question 27

Question 34 | Q01426

In which derivation the right-most non-terminal symbol is replaced at each step?

A. Right look ahead

B. Right claim

C. Rightmost

D. Right non-terminal

Hint

Apply the stated transition, closure or equivalence rule step by step; distinguish a guaranteed bound from a value for one particular machine.

Answer C: Rightmost

At every derivation step choose the rightmost remaining variable and replace it by a production right side. That is a rightmost derivation.

Why each choice is right or wrong

A - Incorrect: Lookahead is not the name of the expansion order.

B - Incorrect: Right claim does not name a derivation method.

C - Correct: Names the specified rightmost expansion rule.

D - Incorrect: Names a symbol position but not the derivation method.

Primary note proof

Study-note section AUT16

In a context-free grammar (CFG) each production has one variable on the left, $A \to \alpha$ with $\alpha \in (V \cup \Sigma)^*$. The left side therefore also belongs to $V \cup \Sigma$ and its string closure; alternatives stating these weaker memberships can all be true. A sentential form is any string derived from S, including terminal-only sentences and S itself.

Study-note section AUT16

A leftmost derivation always replaces the leftmost variable; a rightmost one replaces the rightmost variable. This is not scanning the completed input in opposite directions. A complete parse tree has root S, internal variable nodes expanded by productions and terminal leaves, with ε leaves when that convention is used. Read the leaves left-to-right for the yielded word. The whole tree represents derivation structure; a path in a derivation graph is a different object.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Automata and Complexity Theory module | PDF page 9 | lines 9-10

> in the rightmost derivation, the input is scanned and
> replaced with the production rule from right to left.

Question sources

Compiler Design review questions · Question 34

Question 35 | Q01427

In which derivation the leftmost non-terminal symbol is replaced at each step?

A. Left recursion

B. Left non-terminal

C. Left pushdown

D. Leftmost

Hint

Apply the stated transition, closure or equivalence rule step by step; distinguish a guaranteed bound from a value for one particular machine.

Answer D: Leftmost

Choose the leftmost variable at each step, rather than arbitrarily choosing any remaining variable. This is precisely leftmost derivation.

Why each choice is right or wrong

A - Incorrect: Left recursion is a form of production, not the choice of variable at each derivation step.

B - Incorrect: A left nonterminal is a location, not the derivation name.

C - Incorrect: Left pushdown is not the defined method.

D - Correct: Matches left-to-right variable expansion.

Primary note proof

Study-note section AUT16

> In a context-free grammar (CFG) each production has one variable on the left, A→α with α∈(V∪Σ)*. The left side therefore also belongs to V∪Σ and its string closure; alternatives stating these weaker memberships can all be true. A sentential form is any string derived from S, including terminal-only sentences and S itself.

Study-note section AUT16

A leftmost derivation always replaces the leftmost variable; a rightmost one replaces the rightmost variable. This is not scanning the completed input in opposite directions. A complete parse tree has root S, internal variable nodes expanded by productions and terminal leaves, with ε leaves when that convention is used. Read the leaves left-to-right for the yielded word. The whole tree represents derivation structure; a path in a derivation graph is a different object.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Automata and Complexity Theory module | PDF page 9 | lines 7-8

in the leftmost derivation, the input is
scanned and replaced with the production rule from left to right.

Question sources

Compiler Design review questions · Question 35

Question 36 | Q01429

Which symbol is not related to context-free grammar?

A. End symbol

B. Start symbol

C. Non-terminal symbol

D. Terminal symbol

Hint

Distinguish symbols, finite strings, their lengths and sets of strings. Apply the operation to its stated operands, including empty cases.

Answer A: End symbol

A CFG consists of a start symbol, terminals, nonterminals and productions. A dedicated end symbol is not one of the four CFG components, even though a parser may add an input end marker for a separate purpose.

Why each choice is right or wrong

A - Correct: Not a required CFG component in the four-element definition.

B - Incorrect: The distinguished S is required.

C - Incorrect: Variables/nonterminals form V.

D - Incorrect: The alphabet Σ supplies terminals.

Primary note proof

Study-note section AUT04

Write G=(V,Σ,P,S): finite variables V, terminals Σ disjoint from V, productions P and start variable S∈V. A production replaces its left side by its right side. The language is L(G)={w∈Σ*:S⇒*w}. A recognizer processes a word and accepts exactly the language members. Grammars generate languages; automata recognize languages, not grammars.

Study-note section AUT04

Two grammars are equivalent when their generated languages are equal, regardless of rule counts or variable names. One common word, equal alphabets or one-way inclusion is insufficient.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Automata and Complexity Theory module | PDF page 8 | lines 22-23

Context-free grammars have a start symbol and set of terminals, non-terminals, and production rule components.

Question sources

Compiler Design review questions · Question 37

Question 37 | Q01432

Which of the following tree is the pictorial identification of the derivation?

A. The oct tree

B. The parse tree

C. The binary tree

D. None of the above

Hint

Apply the stated transition, closure or equivalence rule step by step; distinguish a guaranteed bound from a value for one particular machine.

Answer B: The parse tree

A parse/derivation tree labels nodes with grammar symbols and uses edges to show applications of productions. That is the pictorial representation asked for; it need not be a binary tree because productions may have arbitrary finite right sides.

Why each choice is right or wrong

A - Incorrect: An octree is not the grammar representation defined here.

B - Correct: A parse tree is this graphical representation.

C - Incorrect: Production right sides need not have two children, so binary tree is not the general answer.

D - Incorrect: The parse-tree option already supplies the defined representation.

Primary note proof

Study-note section AUT16

> In a context-free grammar (CFG) each production has one variable on the left, A→α with α∈(V∪Σ)*. The left side therefore also belongs to V∪Σ and its string closure; alternatives stating these weaker memberships can all be true. A sentential form is any string derived from S, including terminal-only sentences and S itself.

Study-note section AUT16

> A leftmost derivation always replaces the leftmost variable; a rightmost one replaces the rightmost variable. This is not scanning the completed input in opposite directions. A complete parse tree has root S, internal variable nodes expanded by productions and terminal leaves, with ε leaves when that convention is used. Read the leaves left-to-right for the yielded word. The whole tree represents derivation structure; a path in a derivation graph is a different object.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Automata and Complexity Theory module | PDF page 9 | lines 5-6

> The derivation tree is a graphical representation of the given production rules of context-free grammar (CFG).

Automata and Complexity Theory module | PDF page 8 | lines 30-31

> The nodes of the tree represent the
> symbols and the edges represent the use of production rules.

Question sources

Compiler Design review questions · Question 40

Question 38 | Q01435

Which grammar forms are guaranteed to have an equivalent DFA? Select all that apply. Use consistently linear orientations throughout each grammar.

A. Generic grammar (arbitrary unrestricted grammar)

B. Left Linear Grammar

C. Right-linear grammar

D. All of the above

Hint

Track mandatory symbols and factors that may repeat or disappear. Use short test words to check both directions of each proposed language description.

Answer B, C: Left Linear Grammar; Right-linear grammar

Consistently left-linear and consistently right-linear grammars both generate regular languages. Every regular language has an equivalent DFA; a right-linear grammar can first be converted to an NFA and then determinized. Arbitrary unrestricted grammars can generate nonregular languages and have no such guarantee. The all-of-the-above alternative fails because that generic case is not constrained to regular languages.

Why each choice is right or wrong

A - Incorrect: An arbitrary unrestricted grammar need not generate a regular language.

B - Correct: A consistently left-linear grammar generates a regular language.

C - Correct: A consistently right-linear grammar generates a regular language.

D - Incorrect: The generic-grammar case prevents all of the above from being true.

Primary note proof

Study-note section AUT14

> Right-linear rules have form A→aB, A→a or permitted ε rules; left-linear rules put the single variable at the opposite end, A→Ba. Both consistent orientations generate REG; arbitrarily mixing them can generate nonregular languages, for example S→aA|ε, A→Sb generates $a^n b^n$. A CFG rule A→aBA is legal but not regular-grammar form.

Study-note section AUT14

> Regular languages are closed under union, concatenation, star, reversal, intersection, difference and complement.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Automata and Complexity Theory module | PDF page 8 | lines 2-2

> The language defined by regular grammar is known as regular language.

Automata and Complexity Theory module | PDF page 7 | lines 16-16

> A language is said to be a Regular language if and only if some finite state machine recognizes it

Question sources

Compiler Design review questions · Question 43

Question 39 | Q01437

Classify FORTRAN's computational power under an idealized model with ordinary loops and unbounded indexed read/write storage. Which description is intended, rather than a classification of its token patterns or source syntax?

A. Turing-complete language

B. Context-sensitive language

C. Context-free language

D. Regular language

Hint

Compare production restrictions, memory access and language-family containments. Distinguish an exact characterization from an included subclass.

Answer A: Turing-complete language

Under the stated idealization, indexed writable storage can represent a Turing tape, with an index recording the head position and variables recording the finite control. A loop implements successive read, write, move and state-update steps. This gives the intended Turing-complete computational classification. The other choices classify word-language families rather than naming this general simulation capability; the conclusion depends on the explicit storage model.

Why each choice is right or wrong

A - Correct: Turing-complete describes the specified general computation capability.

B - Incorrect: Context-sensitive classifies formal word languages rather than this simulation capability.

C - Incorrect: Context-free syntax does not itself characterize general computation power.

D - Incorrect: Regular token structure does not describe Turing-machine simulation power.

Primary note proof

Study-note section AUT28

> An off-line TM has a read-only input tape and separate read/write work tape. Multitape, multihead, two-way-infinite-tape and nondeterministic TMs have the same language-recognition power as standard TMs, though simulation costs differ. Two unbounded stacks can simulate a tape. Turing completeness means ability to simulate general TM computation, not polynomial speed or literally using infinitely many cells in a finite run. The Church-Turing thesis connects effective procedures to this formal model; it is not a proved runtime theorem.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Automata and Complexity Theory module | PDF page 8 | lines 20-21

> A context-free grammar
> can describe all regular languages and more, but they cannot describe all possible languages.

Automata and Complexity Theory module | PDF page 3 | lines 13-14

> A grammar defines a set of rules, and with the use of these rules, valid sentences in a language are
> generated.

Question sources

Compiler Design review questions · Question 45

Question 40 | Q01451

DFA is an abbreviation of.

A. Non Deterministic Finite set Automata

B. Deterministic Finite Automata

C. Non Deterministic Finite Automata

D. Deterministic Finite set Automata

Hint

Read the transition domain and acceptance condition carefully. Track which inputs are consumed and evaluate the whole word under the stated model.

Answer B: Deterministic Finite Automata

DFA stands for deterministic finite automaton; deterministic specifies a unique next state for a given current state and input.

Why each choice is right or wrong

A - Incorrect: Adds nondeterministic and set, neither in the abbreviation.

B - Correct: This is the exact expansion of the abbreviation.

C - Incorrect: This is NFA, not DFA.

D - Incorrect: Inserts set, which is absent from the expansion.

Primary note proof

Study-note section AUT06

> A deterministic finite automaton (DFA) is $M=(Q,\Sigma,\delta,q_0,F)$, with finite nonempty Q, start $q_0 \in Q$, accepting set $F \subseteq Q$ and total $\delta: Q \times \Sigma \to Q$. Exactly one next state is specified per state/symbol pair; this does not mean the whole machine has one transition. F may be empty, recognizing $\varnothing$.

Study-note section AUT06

> Start at q_0 and consume the entire input. Accept iff the ending state is in F; visiting an accepting state earlier is insufficient. ε is accepted exactly when $q_0 \in F$. A nonaccepting sink loops on every symbol and cannot reach acceptance; it still finishes after finite input. Finite automata have finite memory encoded by states, not no state or no ability to remember any history.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Automata and Complexity Theory module | PDF page 5 | lines 3-5

> DFA refers to deterministic finite automata. Deterministic refers to the uniqueness of the computation. In the DFA, the machine goes to one state only for a particular input character. DFA does not accept the null move.

Question sources

Compiler Design review questions · Question 59

Question 41 | Q01452

NFA is an abbreviation of.

A. Non Deterministic Finite set Automata

B. Deterministic Finite Automata

C. Non Deterministic Finite Automata

D. Deterministic Finite set Automata

Hint

Read the transition domain and acceptance condition carefully. Track which inputs are consumed and evaluate the whole word under the stated model.

Answer C: Non Deterministic Finite Automata

NFA stands for nondeterministic finite automaton; it permits multiple possible next states and null moves.

Why each choice is right or wrong

A - Incorrect: Inserts set, which is absent from the abbreviation.

B - Incorrect: This expands DFA instead.

C - Correct: This is the exact expansion of the abbreviation.

D - Incorrect: This describes a deterministic model and inserts set.

Primary note proof

Study-note section AUT09

> A nondeterministic finite automaton (NFA) permits zero, one or many successors: $\delta:Q\times\Sigma\to P(Q)$. An ε-NFA additionally permits moves without consuming input, using domain $Q\times(\Sigma\cup\{\varepsilon\})$. Some books include ε moves in the term NFA; state the convention. An NFA accepts iff at least one finite path consumes the complete word and finishes in F, including permitted ε moves before/after symbols.

Study-note section AUT09

An NFA is not more expressive than a DFA: both recognize exactly REG. Nondeterminism can make a description smaller. A DFA is a special NFA after interpreting its unique successor as a singleton set.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Automata and Complexity Theory module | PDF page 5 | lines 7-7

NFA stands for non-deterministic finite automata.

Question sources

Compiler Design review questions · Question 60

Question 42 | SM0018

Which branch of study deals with whether a problem can be solved at all or not, regardless of the resources required?

A. Computability theory

B. Automata theory

C. Set theory

D. Complexity theory

Hint

Check whether the required procedure must halt on every input, and distinguish solving a task from recognizing its yes-instances.

Answer A: Computability theory

The resource-independent question is whether an algorithm can decide the problem at all. Computability concerns membership algorithms and the limits of what Turing machines compute; computational complexity concerns resource estimates.

Why each choice is right or wrong

A - Correct: Matches existence of an algorithm regardless of cost.

B - Incorrect: Automata concerns abstract machines; computability specifically asks the solvability question.

C - Incorrect: Computability, rather than set theory, studies algorithmic solvability.

D - Incorrect: Complexity concerns time and space resources once solving a problem is considered.

Primary note proof

Study-note section AUT-S01

> Computability theory asks whether an effective algorithm can solve a problem at all. For a decision problem, a decision procedure must terminate with the correct yes-or-no result on every input.

Study-note section AUT-S01

> Computational complexity theory studies the time and memory resources required to solve problems as functions of encoded input size. Automata theory studies abstract machine models and the languages they recognize; these subjects overlap but ask different questions.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Automata and Complexity Theory module | PDF page 15 | lines 24-26

> A set X is recursive if we have an algorithm to
> determine whether a given element belongs to
> X or not.

Automata and Complexity Theory module | PDF page 15 | lines 33-35

> Algorithm analysis is an important part of a broader computational complexity theory, which provides theoretical estimates for the resources needed by any algorithm which solves a given computational problem.

Automata and Complexity Theory module | PDF page 11 | lines 7-7

> A Turing machine can do everything that a real computer can do.

Question sources

MOE Computer Science Exit Exam · Question 5-2

National Computer Science Exit Exam · Question 38

National Computer Science Exit Exam

MOE Computer Science Exit Exam

Computer Science Exit Exam practice collection · Question 246

Question 43 | SM0024

Let a Grammar G = ({S}, {a, b}, S, P), where {S} is set of variables, {a, b} is set of terminal symbols, S is start variable and P is set of productions. Then which one of the following is not an element of the language (L) (L) which is generated from G. Given:
P:S-- aSb,
S-A

A. aabb

B. ab

C. X

D. a

Hint

Apply the stated transition, closure or equivalence rule step by step; distinguish a guaranteed bound from a value for one particular machine.

Answer D: a is not generated.

The image restores S→aSb|λ and option C as λ, not the extracted X. AUT17 derives exactly $a^n b^n$ for n≥0. Two recursions generate aabb, one generates ab, and the base rule gives λ. A lone a has unequal counts and cannot arise. The readable source resolves the OCR damage rather than changing the grammar by assumption.

Why each choice is right or wrong

A - Incorrect: Two recursive pair additions followed by the empty rule derive aabb.

B - Incorrect: One application of S→aSb followed by S→λ derives ab.

C - Incorrect: The image reads lambda here, and the empty base production derives it directly.

D - Correct: A lone a has no matching b and is outside the generated equal-count language.

Primary note proof

Study-note section AUT17

> Use a recursive rule to enforce matched structure and a base rule to stop. S→aSb|ε generates exactly $a^n b^n$ for n≥0: n recursive steps add n a's on the left and n b's on the right. Derive aabb by S⇒aSb⇒aaSbb⇒aabb. For n≥1 replace ε by base ab.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Automata and Complexity Theory module | PDF page 8 | lines 27-28

> R is a set of production rules where each production rule maps a variable to a string s∈(V∪Σ)*;

Automata and Complexity Theory module | PDF page 2 | lines 24-25

> A string with zero occurrences of symbols is known as an empty string. It is represented by ε.

Question sources

MOE Computer Science Exit Exam · Question 7-1

National Computer Science Exit Exam · Question 49

National Computer Science Exit Exam

MOE Computer Science Exit Exam

Question 44 | SM0056

Which type of Turing machine has two tapes: one tape read-only and the other read-write?

A. Multi-head Turing machine

B. Off-line Turing machine

C. Non-deterministic Turing machine

D. Multi-dimensional Turing machine

Hint

Compare production restrictions, memory access and language-family containments. Distinguish an exact characterization from an included subclass.

Answer B: An off-line Turing machine.

AUT28 directly defines an off-line TM as having a read-only input tape and a separate read/write work tape. The question gives that architecture. Multiple heads, nondeterministic transition choices and multidimensional tape geometry are independent variants; none defines this input/work separation. The number of tape roles alone does not imply those other variants.

Why each choice is right or wrong

A - Incorrect: Multi-head describes head multiplicity rather than this read-only input and writable work-tape separation.

B - Correct: The off-line model explicitly separates a read-only input tape from a read/write work tape.

C - Incorrect: Nondeterministic describes transition choices, not which tape permits writes.

D - Incorrect: Multidimensional describes tape geometry rather than the two tape roles specified here.

Primary note proof

Study-note section AUT28

> An off-line TM has a read-only input tape and separate read/write work tape. Multitape, multihead, two-way-infinite-tape and nondeterministic TMs have the same language-recognition power as standard TMs, though simulation costs differ. Two unbounded stacks can simulate a tape. Turing completeness means ability to simulate general TM computation, not polynomial speed or literally using infinitely many cells in a finite run. The Church-Turing thesis connects effective procedures to this formal model; it is not a proved runtime theorem.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Automata and Complexity Theory module | PDF page 11 | lines 9-9

> It has a tape head that can read and write symbols and move around on the tape.

Automata and Complexity Theory module | PDF page 11 | lines 20-22

δ is the transition function, mapping (q, x) onto (q', y, D) where D denotes the direction of movement of R/W head: D = L or R according as the movement is to the left or right.

Question sources

MOE Computer Science Exit Exam · Question 16-3

National Computer Science Exit Exam · Question 36

National Computer Science Exit Exam

MOE Computer Science Exit Exam

Computer Science Exit Exam practice collection · Question 248

Question 45 | SM0094

Given input alphabet Σ = {a, b, c, d} and empty string λ, which one of the following is true about Σ*?

A. $\Sigma^* = \Sigma^1 \cup \Sigma^2 \cup \Sigma^3 \cup \Sigma^4$

B. $\Sigma^* = \Sigma^+$

C. $\Sigma^* = \Sigma^+ \cup \{\lambda\}$

D. $\Sigma^* = \{\lambda\}$

Hint

Distinguish symbols, finite strings, their lengths and sets of strings. Apply the operation to its stated operands, including empty cases.

Answer C: $\Sigma^* = \Sigma^+ \cup \{\lambda\}$

Σ* consists of all finite symbol sequences including zero occurrences; Σ+ supplies the positive-length sequences. Since the alphabet has no empty symbol, exactly λ separates the two sets. Thus Σ*=Σ+∪{λ}.

Why each choice is right or wrong

A - Incorrect: Only lengths 1-4 are included; both λ and words of length 5 are missing.

B - Incorrect: Positive closure omits λ, unlike star.

C - Correct: Adds exactly the missing zero-length word.

D - Incorrect: Star also contains a and every other nonempty finite word over the alphabet.

Primary note proof

Study-note section AUT03

Σ^n contains words of exactly n symbols; for $|\Sigma|=k$, $|\Sigma^n|=k^n$, including $\Sigma^0=\{\varepsilon\}$. $\Sigma^*=\bigcup n\geq 0\Sigma^n$ and $\Sigma^+=\bigcup n\geq 1\Sigma^n$. Over a nonempty alphabet, Σ^+ excludes ε and $\Sigma^*=\Sigma^+\cup\{\varepsilon\}$. Σ^2 over {a,b} is {aa,ab,ba,bb}.

Study-note section AUT03

> For an arbitrary language L, $L^*=\cup n\geq 0 L^n$ and $L^+=LL^*$. Unlike alphabet positive closure, L^+ can contain ε when $\varepsilon \in L$. $\varnothing^*=\{\varepsilon\}$, $\varnothing^+=\varnothing$, and $\{\varepsilon\}^*=\{\varepsilon\}$. Union means membership in either set; intersection requires the same word in both, not concatenation. Complement is Σ^*-L and needs a fixed alphabet.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Automata and Complexity Theory module | PDF page 5 | lines 22-22

> In a regular expression, x* means zero or more occurrence of x.

Automata and Complexity Theory module | PDF page 5 | lines 25-25

> means one or more occurrence of x. It can generate {x, xx, xxx, xxxx,

Automata and Complexity Theory module | PDF page 2 | lines 24-25

> A string with zero occurrences of symbols is known as an empty string. It is represented by ε.

Question sources

MOE Computer Science Exit Exam · Question 27-1

National Computer Science Exit Exam · Question 55

National Computer Science Exit Exam

MOE Computer Science Exit Exam

Question 46 | SM0095

Given input alphabet Σ = {a, b, c, d} and empty string λ, which one of the following is equivalent to Σ^0?

A. Σ^+

B. Σ^*

C. Σ

D. $\{\lambda\}$

Hint

Distinguish symbols, finite strings, their lengths and sets of strings. Apply the operation to its stated operands, including empty cases.

Answer D: The singleton set {λ}.

AUT03 defines Σ^n as all words of length n, including $\Sigma^0=\{\varepsilon\}$; AUT01 identifies λ with ε. There is exactly one length-zero word, so its set is {λ}. Σ, positive closure and Kleene closure contain nonempty words for the given nonempty alphabet and therefore are not equal to the zero power.

Why each choice is right or wrong

A - Incorrect: Σ^+ contains positive-length words over the alphabet and excludes the empty word.

B - Incorrect: Σ^* contains every finite-length word, not just the single empty word.

C - Incorrect: Σ contains four alphabet symbols, each forming a length-one word rather than a length-zero word.

D - Correct: λ denotes the empty word, and its singleton set is exactly the alphabet's zero power.

Primary note proof

Study-note section AUT03

> Σ^n contains words of exactly n symbols; for $|\Sigma|=k$, $|\Sigma^n|=k^n$, including $\Sigma^0=\{\varepsilon\}$. $\Sigma^*=\bigcup n\geq 0\Sigma^n$ and $\Sigma^+=\bigcup n\geq 1\Sigma^n$. Over a nonempty alphabet, Σ^+ excludes ε and $\Sigma^*=\Sigma^+\cup\{\varepsilon\}$. Σ^2 over {a,b} is {aa,ab,ba,bb}.

Study-note section AUT03

> For an arbitrary language L, $L^*=\bigcup n\geq 0L^n$ and $L^+=LL^*$. Unlike alphabet positive closure, L^+ can contain ε when $\varepsilon\in L$. $\varnothing^*=\{\varepsilon\}$, $\varnothing^+=\varnothing$, and $\{\varepsilon\}^*=\{\varepsilon\}$. Union means membership in either set; intersection requires the same word in both, not concatenation. Complement is Σ^*-L and needs a fixed alphabet.

Study-note section AUT01

> An alphabet Σ is a finite set of symbols. A word/string is a finite ordered sequence of symbols from Σ; a language is a set of words, $L\subseteq\Sigma^*$. An empty word ε, also written λ, has length zero. The empty language $\varnothing$ contains no words; $\{\varepsilon\}$ contains one.

Study-note section AUT01

> Length counts occurrences, not distinct symbols: $|caddba|=6$, so $|caddba|^2=36$.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Automata and Complexity Theory module | PDF page 5 | lines 22-22

> In a regular expression, x* means zero or more occurrence of x.

Automata and Complexity Theory module | PDF page 2 | lines 24-25

> A string with zero occurrences of symbols is known as an empty string. It is represented by ε.

Question sources

MOE Computer Science Exit Exam · Question 27-2

National Computer Science Exit Exam · Question 30

National Computer Science Exit Exam

MOE Computer Science Exit Exam

Question 47 | SM0206

a - b, L state transition in a Turing Machine means,

A. read a, write b and move left

B. delete a, read b and move left

C. write a, read b and move left

D. replace 6 by a and move left

Hint

Compare production restrictions, memory access and language-family containments. Distinguish an exact characterization from an included subclass.

Answer A: read a, write b and move left

In an edge label (a,b,L), the first symbol is the scanned symbol, the second is written in its cell, and L moves the head left. The finite control also moves to the destination state of the arrow.

Why each choice is right or wrong

A - Correct: Exactly follows read, write, move-left order.

B - Incorrect: The scanned symbol is a, and b is written, not read.

C - Incorrect: Reverses reading and writing.

D - Incorrect: Reverses the direction of replacement; the cell changes a to b.

Primary note proof

Study-note section AUT27

> δ(q,a)=(p,b,L) means read a in q, write b, move one cell left and enter p. Accept/reject halting states have no further moves under this convention.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Automata and Complexity Theory module | PDF page 11 | lines 20-22

> δ is the transition function, mapping (q, x) onto (q', y, D) where D denotes the direction of movement of R/W head: D = L or R according as the movement is to the left or right.

Automata and Complexity Theory module | PDF page 11 | lines 9-9

> It has a tape head that can read and write symbols and move around on the tape.

Automata and Complexity Theory module | PDF page 14 | lines 8-8

> As a result the symbol is β written in the cell under the R/W head.

Question sources

Computer Science Exit Exam · Sene 2016 · Question 2-2

Question 48 | SM0234

Which of the following IS NOT a component of a Turing machine?

A. Read-write head

B. Control Unit

C. Stack

D. Tape

Hint

Compare production restrictions, memory access and language-family containments. Distinguish an exact characterization from an included subclass.

Answer C: Stack

The standard Turing machine uses an unbounded tape, a read/write head, and finite state control. Stack is the distinctive added memory of a PDA and is not a separate structural component required by this standard TM model.

Why each choice is right or wrong

A - Incorrect: Explicitly named as the reading/writing/moving device.

B - Incorrect: Finite states and transition rules provide control.

C - Correct: The standard TM model has tape storage, whereas the PDA description explicitly adds a stack.

D - Incorrect: The tape provides the TM's memory.

Primary note proof

Study-note section AUT27

> For a Turing machine (TM), use $M=(Q,\Sigma,\Gamma,\delta,q_0,q\text{accept},q\text{reject})$, with finite state/tape alphabets, $\Sigma\subseteq\Gamma$ and blank $\square\in\Gamma-\Sigma$. The blank symbol belongs to the tape alphabet Γ, not the input alphabet Σ. The standard machine has finite control, a read/write head and an unbounded tape. Input is initial tape content, not a separate hardware organ; no separate stack is required.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Automata and Complexity Theory module | PDF page 11 | lines 9-9

> It has a tape head that can read and write symbols and move around on the tape.

Automata and Complexity Theory module | PDF page 11 | lines 20-22

> δ is the transition function, mapping (q, x) onto (q', y, D) where D denotes the direction of movement of R/W head: D = L or R according as the movement is to the left or right.

Automata and Complexity Theory module | PDF page 9 | lines 18-19

> Pushdown automata are nondeterministic finite state machines augmented with additional memory
> in the form of a stack,

Question sources

Computer Science Exit Exam · Sene 2016 · Question 11-2

Computer Science Exit Exam practice collection · Question 258

Question 49 | SM0246

In a grammar, if the production S -> lambda (where S is a non-terminal) exists, such a production is called:

A. Null production

B. Unit production

C. Secondary production

D. Sigma production

Hint

Distinguish symbols, finite strings, their lengths and sets of strings. Apply the operation to its stated operands, including empty cases.

Answer A: A null (epsilon) production.

AUT19 defines a null production as A→ε, while AUT01 identifies λ as another notation for the empty word. Substituting S for A gives the displayed S→λ rule. A unit production instead has one nonterminal on the right; secondary and sigma production are not the defined names for an empty right-hand side.

Why each choice is right or wrong

A - Correct: The right-hand side is the empty word, exactly the definition of a null production.

B - Incorrect: A unit production has one nonterminal on the right, not the empty word.

C - Incorrect: Secondary production is not the defined term for a rule deriving the empty word.

D - Incorrect: Sigma denotes an alphabet conventionally, not the null-production category described here.

Primary note proof

Study-note section AUT19

A null production is A→ε; a unit production is A→B with one variable.

Study-note section AUT01

An alphabet Σ is a finite set of symbols. A word/string is a finite ordered sequence of symbols from Σ; a language is a set of words, L⊆Σ*. An empty word ε, also written λ, has length zero. The empty language ∅ contains no words; {ε} contains one.

Study-note section AUT01

Length counts occurrences, not distinct symbols: |caddba|=6, so $|caddba|^2=36$.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Automata and Complexity Theory module | PDF page 2 | lines 24-25

A string with zero occurrences of symbols is known as an empty string. It is represented by ε.

Automata and Complexity Theory module | PDF page 8 | lines 27-28

R is a set of production rules where each production rule maps a variable to a string s∈(V∪Σ)∗;

Automata and Complexity Theory module | PDF page 9 | lines 15-16

Simplification essentially comprises the following steps; Reduction of CFG, Removal of Unit Productions, and Removal of Null Productions.

Question sources

Computer Science Exit Exam · Sene 2016 · Question 15-2

Computer Science Exit Exam practice collection · Question 261

Question 50 | SM0256

Which string from the following is a member of the language represented by the regular expression a*(b+c)?

A. aab

B. bc

C. aaa

D. aaabc

Hint

Apply the stated transition, closure or equivalence rule step by step; distinguish a guaranteed bound from a value for one particular machine.

Answer A: aab

a* permits any number of initial a symbols; (b+c) then contributes exactly one b or c. aab factors as aa followed by b. No further suffix is allowed after that final symbol.

Why each choice is right or wrong

A - Correct: aa is in a*, followed by the permitted b.

B - Incorrect: Contains two symbols from the final alternative; the a* prefix cannot consume b.

C - Incorrect: Lacks the compulsory final b or c.

D - Incorrect: After aa a and b there is an extra c; the expression allows only one final alternative.

Primary note proof

Study-note section AUT12

> Classical expressions use ∅, ε, symbols, union, concatenation and star. A union B is written A|B or A+B; postfix r^+ means one-or-more r repetitions and is a different use of +. Parenthesize when conventions might clash. Star binds tightest, concatenation next, union last. r? means r|ε; r^+=rr*.

Study-note section AUT12

> (a|b)*a denotes words ending a; 0*10* denotes exactly one 1; a*(b|c) gives zero-or-more a's then exactly one b or c, so aab belongs but aaa and aaabc do not.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Automata and Complexity Theory module | PDF page 5 | lines 22-22

> In a regular expression, x* means zero or more occurrence of x.

Automata and Complexity Theory module | PDF page 7 | lines 16-16

> A language is said to be a Regular language if and only if some finite state machine recognizes it

Question sources

Computer Science Exit Exam · Sene 2016 · Question 18-2

Computer Science Exit Exam practice collection · Question 259

Question 51 | SM0303

Which class of languages is most suited to be represented by pushdown automata?

A. Context-free language

B. Regular language

C. Recursively enumerable language

D. Unrestricted language

Hint

Compare production restrictions, memory access and language-family containments. Distinguish an exact characterization from an included subclass.

Answer A: Context-free language

PDAs and CFGs characterize the same family of context-free languages. Regular languages are included, but the question most suited asks for the full family rather than a subclass.

Why each choice is right or wrong

A - Correct: Exactly the class characterized by PDAs.

B - Incorrect: A smaller included family, already recognizable by finite automata.

C - Incorrect: The hierarchy places RE at the larger Turing machine level.

D - Incorrect: Unrestricted languages are not the PDA/CFG class.

Primary note proof

Study-note section AUT21

> A nondeterministic pushdown automaton (PDA) adds a stack to finite control: $(Q,\Sigma,\Gamma,\delta,q_0,Z_0,F)$. Its transition relation maps (state,input symbol or ε,stack top) to possible (new state,replacement stack string) pairs. Fix which end denotes the top before tracing. Replacing X by X leaves the stack unchanged; ε input means consume no symbol, while replacing X by ε pops it. These are different uses of ε.

Study-note section AUT05

> Type 3 regular grammars correspond to finite automata; type 2 CFGs to nondeterministic PDAs; type 1 context-sensitive/noncontracting grammars to nondeterministic LBAs; type 0 unrestricted grammars to Turing recognizers.

Study-note section AUT05

> Use REG ⊊ CFL ⊊ CSL ⊊ DECIDABLE ⊊ RE, under the usual convention allowing S→ε with S absent from right sides when needed.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Automata and Complexity Theory module | PDF page 9 | lines 23-23

> Pushdown automata accept context-free languages, which include the set of regular languages.

Question sources

Computer Science Exit Exam · Sene 2016 · Question 33-3

Computer Science Exit Exam practice collection · Question 257

Question 52 | AM0744

Which statement correctly describes the generation and recognition roles of formal grammars and automata?

A. Formal grammars generate formal languages, and automata recognize those languages.

B. Formal languages recognize automata, which are generated by formal grammars.

C. Formal grammars generate automata, which recognize formal languages.

D. Formal grammars recognize formal languages, which generate automata.

Hint

Distinguish symbols, finite strings, their lengths and sets of strings. Apply the operation to its stated operands, including empty cases.

Answer A: Formal grammars generate formal languages, and automata recognize those languages.

A formal grammar generates terminal words, collectively its language. An automaton processes words and accepts exactly those in the language it recognizes. Thus languages are generated by grammars and recognized by automata. The relationship concerns the same set of strings, not a claim that automata recognize grammar descriptions or that languages themselves perform machine operations. Option A states those roles explicitly.

Why each choice is right or wrong

A - Correct: It correctly assigns language generation to grammars and recognition to automata.

B - Incorrect: Languages do not perform recognition or generate machines.

C - Incorrect: Grammars generate word languages rather than automata in this relationship.

D - Incorrect: Grammars generate languages; languages do not generate automata.

Primary note proof

Study-note section AUT04

> Write G=(V,Σ,P,S): finite variables V, terminals Σ disjoint from V, productions P and start variable S∈V. A production replaces its left side by its right side. The language is L(G)={w∈Σ*:S⇒*w}. A recognizer processes a word and accepts exactly the language members. Grammars generate languages; automata recognize languages, not grammars.

Study-note section AUT04

> Two grammars are equivalent when their generated languages are equal, regardless of rule counts or variable names. One common word, equal alphabets or one-way inclusion is insufficient.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Automata and Complexity Theory module | PDF page 3 | lines 13-14

A grammar defines a set of rules, and with the use of these rules, valid sentences in a language are
generated.

Automata and Complexity Theory module | PDF page 7 | lines 16-16

A language is said to be a Regular language if and only if some finite state machine recognizes it

Automata and Complexity Theory module | PDF page 10 | lines 6-6

Fig: The traditional Chomsky hierarchy

Question sources

Computer Science MCQ practice collection · Question 744

Question 53 | AM0745

Which offered item is a precisely specified formal language, rather than a natural-language name?

A. English

B. Spanish

C. L={w∈{a,b}* | w ends in a}, denoted by (a|b)*a

D. French

Hint

Apply the stated transition, closure or equivalence rule step by step; distinguish a guaranteed bound from a value for one particular machine.

Answer C: L={w∈{a,b}* | w ends in a}, denoted by (a|b)*a

A formal language is a set of strings with precisely defined membership. Option C gives the language of all words over {a,b} that end in a, denoted by (a|b)*a. English, Spanish and French are natural-language names in this comparison. A regular expression alone names a specification notation; explicitly naming its denoted word set provides the actual language example.

Why each choice is right or wrong

A - Incorrect: English is a natural-language name.

B - Incorrect: Spanish is a natural-language name.

C - Correct: This specifies a set of words over a fixed alphabet with an exact ending condition.

D - Incorrect: French is a natural-language name.

Primary note proof

Study-note section AUT04

Write G=(V,Σ,P,S): finite variables V, terminals Σ disjoint from V, productions P and start variable S∈V. A production replaces its left side by its right side. The language is L(G)={w∈Σ*:S⇒*w}. A

recognizer processes a word and accepts exactly the language members. Grammars generate languages; automata recognize languages, not grammars.

Study-note section AUT04

Two grammars are equivalent when their generated languages are equal, regardless of rule counts or variable names. One common word, equal alphabets or one-way inclusion is insufficient.

Study-note section AUT12

Classical expressions use ∅, ε, symbols, union, concatenation and star. A union B is written A|B or A+B; postfix r^+ means one-or-more r repetitions and is a different use of +. Parenthesize when conventions might clash. Star binds tightest, concatenation next, union last. r? means r|ε; r^+=rr*.

Study-note section AUT12

(a|b)*a denotes words ending a; 0*10* denotes exactly one 1; a*(b|c) gives zero-or-more a's then exactly one b or c, so aab belongs but aaa and aaabc do not.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Automata and Complexity Theory module | PDF page 3 | lines 13-14

A grammar defines a set of rules, and with the use of these rules, valid sentences in a language are
generated.

Automata and Complexity Theory module | PDF page 7 | lines 16-16

A language is said to be a Regular language if and only if some finite state machine recognizes it

Question sources

Computer Science MCQ practice collection · Question 745

Question 54 | AM0746

Which is the least expressive listed machine family that recognizes every regular language?

A. Turing machine

B. Pushdown automaton

C. Finite automaton

D. Nondeterministic Turing machine

Hint

Compare production restrictions, memory access and language-family containments. Distinguish an exact characterization from an included subclass.

Answer C: Finite automaton

Finite automata recognize exactly the regular languages. Pushdown automata and Turing machines can also recognize regular languages, but their families recognize strictly more languages and are not the least sufficient choice. The nondeterministic Turing-machine alternative is likewise stronger than finite automata. This compares expressive power across machine families, rather than implementation size or execution speed.

Why each choice is right or wrong

A - Incorrect: Turing machines recognize a strictly larger family than REG.

B - Incorrect: Pushdown automata recognize all CFLs, exceeding REG.

C - Correct: Finite automata characterize REG exactly and are least sufficient here.

D - Incorrect: Nondeterministic Turing machines exceed finite-automaton recognition power.

Primary note proof

Study-note section AUT05

> Type 3 regular grammars correspond to finite automata; type 2 CFGs to nondeterministic PDAs; type 1 context-sensitive/noncontracting grammars to nondeterministic LBAs; type 0 unrestricted grammars to Turing recognizers.

Study-note section AUT05

> Use REG ⊊ CFL ⊊ CSL ⊊ DECIDABLE ⊊ RE, under the usual convention allowing S→ε with S absent from right sides when needed.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Automata and Complexity Theory module | PDF page 7 | lines 16-16

> A language is said to be a Regular language if and only if some finite state machine recognizes it

Automata and Complexity Theory module | PDF page 9 | lines 23-23

> Pushdown automata accept context-free languages, which include the set of regular languages.

Automata and Complexity Theory module | PDF page 10 | lines 6-6

> Fig: The traditional Chomsky hierarchy

Question sources

Computer Science MCQ practice collection · Question 746

Question 55 | AM0747

Which statements follow from the Chomsky hierarchy and decidability? Select all that apply.

A. Context-free languages are a subset of regular languages.

B. Context-sensitive languages are a subset of recursive languages.

C. Regular languages are a subset of recursive languages.

D. Context-free languages are a subset of recursive languages.

Hint

Compare production restrictions, memory access and language-family containments. Distinguish an exact characterization from an included subclass.

Answer B, C, D: Context-sensitive languages are a subset of recursive languages.; Regular languages are a subset of recursive languages.; Context-free languages are a subset of recursive languages.

The hierarchy is REG⊊CFL⊊CSL⊊DECIDABLE⊊RE under the stated ε convention. Recursive means decidable, so context-sensitive languages are recursive, and transitivity gives the same conclusion for regular and context-free languages. A reverses the first strict inclusion: every regular language is context-free, but not every context-free language is regular. Therefore the three recursive-containment statements are all true.

Why each choice is right or wrong

A - Incorrect: The REG/CFL inclusion is reversed.

B - Correct: CSL is contained in the decidable or recursive family.

C - Correct: REG is contained in CSL and therefore in the recursive family.

D - Correct: CFL is contained in CSL and therefore in the recursive family.

Primary note proof

Study-note section AUT05

> Type 3 regular grammars correspond to finite automata; type 2 CFGs to nondeterministic PDAs; type 1 context-sensitive/noncontracting grammars to nondeterministic LBAs; type 0 unrestricted grammars to Turing recognizers.

Study-note section AUT05

> Use REG ⊊ CFL ⊊ CSL ⊊ DECIDABLE ⊊ RE, under the usual convention allowing S→ε with S absent from right sides when needed.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Automata and Complexity Theory module | PDF page 10 | lines 6-6

> Fig: The traditional Chomsky hierarchy

Automata and Complexity Theory module | PDF page 8 | lines 20-21

> A context-free grammar
> can describe all regular languages and more, but they cannot describe all possible languages.

Automata and Complexity Theory module | PDF page 15 | lines 24-26

> A set X is recursive if we have an algorithm to
> determine whether a given element belongs to
> X or not.

Automata and Complexity Theory module | PDF page 15 | lines 30-30

> All recursive set is recursively enumerable.

Question sources

Computer Science MCQ practice collection · Question 747

Question 56 | AM0748

Which statement is true of ordinary DFAs? In D, the claim concerns every context-free language.

A. DFAs can recognize all regular languages.

B. DFAs have multiple possible transitions for each input symbol.

C. DFAs can have empty transitions.

D. DFAs can recognize every context-free language.

Hint

Respect the quantifier order and length constraints. A valid nonmembership argument must address every allowed decomposition, not one convenient split.

Answer A: DFAs can recognize all regular languages.

Ordinary DFAs recognize exactly the regular languages. Each state/input pair has one next state, and transitions consume an input symbol rather than ε. Some context-free languages are regular, but others, such as a^nb^n, are not and cannot be recognized by a DFA. Making D explicitly universal removes its weaker existential reading and leaves A as the unique true statement.

Why each choice is right or wrong

A - Correct: Every regular language has a DFA.

B - Incorrect: A DFA has a unique successor for each state/symbol pair.

C - Incorrect: Ordinary DFAs do not have ε moves.

D - Incorrect: Nonregular CFLs such as a^nb^n cannot be recognized by DFAs.

Primary note proof

Study-note section AUT06

A deterministic finite automaton (DFA) is $M=(Q,\Sigma,\delta,q_0,F)$, with finite nonempty Q, start $q_0 \in Q$, accepting set $F \subseteq Q$ and total $\delta: Q \times \Sigma \to Q$. Exactly one next state is specified per state/symbol pair; this does not mean the whole machine has one transition.

Study-note section AUT09

An NFA is not more expressive than a DFA: both recognize exactly REG.

Study-note section AUT15

For $L=\{a^n b^n : n \geq 0\}$, choose $a^p b^p$. Every allowed $y=a^k$ with $k \geq 1$; pumping $i=0$ leaves fewer a's than b's. The contradiction excludes every DFA.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Automata and Complexity Theory module | PDF page 5 | lines 3-5

DFA refers to deterministic finite automata. Deterministic refers to the uniqueness of the computation. In the DFA, the machine goes to one state only for a particular input character. DFA does not accept the null move.

Automata and Complexity Theory module | PDF page 7 | lines 16-16

A language is said to be a Regular language if and only if some finite state machine recognizes it

Automata and Complexity Theory module | PDF page 8 | lines 20-21

A context-free grammar
can describe all regular languages and more, but they cannot describe all possible languages.

Question sources

Computer Science MCQ practice collection · Question 748

Question 57 | AM0749

Which listed automaton family has recognition power strictly between finite automata and Turing machines and uses one unbounded stack as its auxiliary memory?

A. Pushdown automaton

B. Non-deterministic finite automaton

C. Turing machine

D. Nondeterministic linear bounded automaton

Hint

Compare production restrictions, memory access and language-family containments. Distinguish an exact characterization from an included subclass.

Answer A: Pushdown automaton

A nondeterministic PDA adds one stack to finite control and recognizes the context-free languages. This family strictly contains REG and lies strictly below Turing-recognizable languages. NFAs have the same power as DFAs, while Turing machines are the upper endpoint. A nondeterministic LBA also lies between those endpoints, but uses bounded tape rather than the specified single-stack model.

Why each choice is right or wrong

A - Correct: A PDA has one unbounded stack and the required intermediate expressive power.

B - Incorrect: An NFA has no stack and recognizes only REG.

C - Incorrect: A TM is the upper endpoint, not strictly below it.

D - Incorrect: An LBA uses linearly bounded tape rather than the specified single stack.

Primary note proof

Study-note section AUT05

> Type 3 regular grammars correspond to finite automata; type 2 CFGs to nondeterministic PDAs; type 1 context-sensitive/noncontracting grammars to nondeterministic LBAs; type 0 unrestricted grammars to Turing recognizers.

Study-note section AUT05

> Use REG ⊊ CFL ⊊ CSL ⊊ DECIDABLE ⊊ RE, under the usual convention allowing S→ε with S absent from right sides when needed.

Study-note section AUT21

> A nondeterministic pushdown automaton (PDA) adds a stack to finite control: $(Q,\Sigma,\Gamma,\delta,q_0,Z_0,F)$.

Study-note section AUT25

> A linear bounded automaton (LBA) is a TM with O(n) tape cells on length-n input; the standard CSL characterization uses nondeterministic LBAs.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Automata and Complexity Theory module | PDF page 9 | lines 21-21

> machines-that can do more than a finite state machine, but less than a Turing machine.

Automata and Complexity Theory module | PDF page 10 | lines 6-6

> Fig: The traditional Chomsky hierarchy

Question sources

Computer Science MCQ practice collection · Question 749

Question 58 | AM0751

Which statements about NP-complete decision problems are true? Select all that apply. In C, exponential time is an upper bound.

A. NP-complete problems are only solvable by nondeterministic algorithms.

B. NP-complete problems are a subset of NP problems.

C. NP-complete problems can be solved in exponential time.

D. NP-complete problems have no practical applications.

Hint

Separate resource bounds, certificate verification and reduction requirements. Use proved containments without assuming unresolved class separations.

Answer B, C: NP-complete problems are a subset of NP problems.; NP-complete problems can be solved in exponential time.

NP-completeness includes membership in NP and hardness for every NP language under polynomial reductions. Polynomial-size certificates permit finite deterministic exhaustive search, giving a 2^poly(n) upper bound. Thus B and C are true together. This does not show that exponential time is necessary. SAT, coloring and tour-decision formulations illustrate meaningful problems, so the blanket denial of applications is also false.

Why each choice is right or wrong

A - Incorrect: Deterministic exhaustive certificate search also decides these problems.

B - Correct: Membership in NP is part of the definition.

C - Correct: Enumerating polynomial-size certificates gives a deterministic exponential-time upper bound.

D - Incorrect: The note gives substantive SAT, coloring and tour-decision examples.

Primary note proof

Study-note section AUT36

> X is NP-hard if every A∈NP satisfies A≤pX. It need not belong to NP and can even be undecidable. X is NP-complete if X∈NP and X is NP-hard. NP-complete languages lie inside NP and can be decided by finite exhaustive certificate search in exponential time. They are not known to require exponential time.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Automata and Complexity Theory module | PDF page 17 | lines 27-28

> P problems are set of problems which can be solved in
> polynomial time by deterministic algorithms.

Automata and Complexity Theory module | PDF page 17 | lines 29-30

> NP problems are the problems which can be solved in non-deterministic polynomial time.

Automata and Complexity Theory module | PDF page 17 | lines 33-35

> The problem belongs to NP, if it‘s easy to check a solution that may have been very tedious to find.

Question sources

Computer Science MCQ practice collection · Question 751

Question 59 | AM0752

Which of the following complexity classes represents the hardest problems in NP?

A. P

B. NP-complete

C. NP-hard

D. PSPACE

Hint

Separate resource bounds, certificate verification and reduction requirements. Use proved containments without assuming unresolved class separations.

Answer B: NP-complete: the problems both in NP and NP-hard.

AUT36 defines NP-complete as NP membership together with hardness for every NP problem under polynomial reductions. That is the precise class described by hardest problems in NP. NP-hard alone can extend beyond NP, even to undecidable problems; PSPACE is a larger resource class, and P does not impose the required hardness.

Why each choice is right or wrong

A - Incorrect: P means deterministic polynomial time and does not define hardness for every NP language.

B - Correct: NP-completeness combines being in NP with being at least as hard as every NP problem.

C - Incorrect: NP-hardness alone does not ensure membership in NP, so it is broader than the class requested.

D - Incorrect: PSPACE is defined by polynomial space and can contain languages outside the specified NP-complete class.

Primary note proof

Study-note section AUT36

X is NP-hard if every A∈NP satisfies A≤pX. It need not belong to NP and can even be undecidable. X is NP-complete if X∈NP and X is NP-hard. NP-complete languages lie inside NP and can be decided by finite exhaustive certificate search in exponential time. They are not known to require exponential time.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Automata and Complexity Theory module | PDF page 17 | lines 29-30

NP problems are the problems which can be
solved in non-deterministic polynomial time.

Automata and Complexity Theory module | PDF page 17 | lines 33-35

The problem belongs to NP, if it‘s easy to check
a solution that may have been very tedious to
find.

Question sources

Computer Science MCQ practice collection · Question 752

Question 60 | AM0753

Which of the following is true regarding the halting problem?

A. The halting problem can be solved by a Turing machine.

B. The halting problem is decidable for all programs.

C. The halting problem is an undecidable problem.

D. The halting problem is only relevant to regular languages.

Hint

Check whether the required procedure must halt on every input, and distinguish solving a task from recognizing its yes-instances.

Answer C: The general halting problem is undecidable.

AUT30 constructs a machine that contradicts a hypothetical universal halting decider when run on its own description. Hence no decider answers halting correctly for all machine/input pairs. HALT is recognizable by simulation, but that process need not terminate on nonhalting instances; recognizing some yes-instances is not solving the general decision problem.

Why each choice is right or wrong

A - Incorrect: Simulation recognizes halting yes-instances but no TM decides the general halting problem on every input.

B - Incorrect: The diagonal argument rules out a decider correct for all programs and inputs.

C - Correct: Undecidability is the exact result proved by the halting diagonalization argument in AUT30.

D - Incorrect: HALT concerns arbitrary computations and is not confined to regular languages or finite automata.

Primary note proof

Study-note section AUT30

> HALT={⟨M,w⟩:M halts on w} is recognizable: simulate and accept upon any halt. Suppose a decider H determines halting. Construct D(x) that loops if H(x,x) says halt, otherwise halts. Running D on its own description contradicts H in both cases. Thus HALT is undecidable and witnesses RE beyond DECIDABLE. Its complement is not RE, or both recognizers would yield a decider.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Automata and Complexity Theory module | PDF page 14 | lines 19-20

> -It Is impossible to construct a single algorithm
> that always leads to a correct yes-or-no answer.

Automata and Complexity Theory module | PDF page 15 | lines 24-26

> A set X is recursive if we have an algorithm to
> determine whether a given element belongs to
> X or not.

Question sources

Computer Science MCQ practice collection · Question 753

Question 61 | AM0754

Which of the following is a regular language?

A. {anbn | n ≥ 0}

B. {anbncn | n ≥ 0}

C. {an | n is a prime number}

D. {anb | n ≥ 0}

Hint

Compare production restrictions, memory access and language-family containments. Distinguish an exact characterization from an included subclass.

Answer D: {anb | n ≥ 0}

Reading the powers as a^n b^n, a^n b^n c^n, a^n for prime n, and a^n b. D is a*b, a regular expression. A and B fail the pumping lemma's condition: choose a^p b^p (or a^p b^p c^p); any nonempty pumped block in the first p positions consists only of a, so pumping changes only its count. For C choose a prime q at least p. If the pumped block has k≥1 a symbols, pumping with i=q+1 gives length q+qk=q(k+1), composite. Thus no pumping decomposition works. The argument uses the arithmetic fact that arbitrarily large primes exist.

Why each choice is right or wrong

A - Incorrect: Pumping only a symbols destroys equality with the b count.

B - Incorrect: Pumping only a symbols destroys equality with both later blocks.

C - Incorrect: Pumping the unary block q+1 times makes a composite length q(k+1).

D - Correct: a*b produces any number of a symbols followed by exactly one b.

Primary note proof

Study-note section AUT26

a*b* and {$a^n b$:n≥0} are regular. $a^n b^n$ is CFL but not regular. Over a two-symbol alphabet, ww^R is CFL, ww is not CFL; both are decidable. $a^n b^n c^n$ is CSL but not CFL, and therefore decidable. Every finite language is regular. A stronger grammar or TM can still recognize any of these lower-family languages.

Study-note section AUT26

Unary prime length {a^p:p prime} is nonregular: choose prime q above pumping length p; if pump piece has length k≥1, using exponent q+1 gives length q(k+1), composite. Every unary CFL is regular, so this language is not CFL either. It is nevertheless context-sensitive: count the input in bounded workspace and test divisors. The balanced-three-count, prime-length, palindrome and $a^n b$ examples above are all recursive languages.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Automata and Complexity Theory module | PDF page 8 | lines 12-12

(1) |uv| ≤ n (2) |v| ≥ 1 (3) for all i ≥ 0: uviw ∈ L

Automata and Complexity Theory module | PDF page 5 | lines 22-22

In a regular expression, x* means zero or more occurrence of x.

Automata and Complexity Theory module | PDF page 8 | lines 27-28

R is a set of production rules where each production rule maps a variable to a string s∈(V∪Σ)*;

Question sources

Computer Science MCQ practice collection · Question 754

Question 62 | AM0755

Which is the least expressive listed grammar family capable of generating every context-free language?

A. Regular grammar

B. Context-free grammar

C. Context-sensitive grammar

D. Unrestricted grammar

Hint

Compare production restrictions, memory access and language-family containments. Distinguish an exact characterization from an included subclass.

Answer B: Context-free grammar

Context-free grammars generate exactly the context-free languages. Regular grammars are insufficient because CFL contains nonregular languages such as a^nb^n. Context-sensitive and unrestricted grammars can also generate CFLs under the stated ε convention, but their expressive power is greater than required. The least-family qualifier distinguishes the exact defining family from stronger families that merely have the same capability as a subset.

Why each choice is right or wrong

A - Incorrect: Regular grammars fail for nonregular CFLs such as a^nb^n.

B - Correct: Context-free grammars cover exactly the required family.

C - Incorrect: Context-sensitive grammars suffice but are stronger than required.

D - Incorrect: Unrestricted grammars suffice but are stronger than required.

Primary note proof

Study-note section AUT05

> Type 3 regular grammars correspond to finite automata; type 2 CFGs to nondeterministic PDAs; type 1 context-sensitive/noncontracting grammars to nondeterministic LBAs; type 0 unrestricted grammars to Turing recognizers.

Study-note section AUT05

> Use REG ⊊ CFL ⊊ CSL ⊊ DECIDABLE ⊊ RE, under the usual convention allowing S→ε with S absent from right sides when needed.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Automata and Complexity Theory module | PDF page 8 | lines 20-21

A context-free grammar
can describe all regular languages and more, but they cannot describe all possible languages.

Automata and Complexity Theory module | PDF page 10 | lines 6-6

Fig: The traditional Chomsky hierarchy

Question sources

Computer Science MCQ practice collection · Question 755

Question 63 | AM0756

Under the usual isolated-start ε convention, which statements about CFLs and CSLs are true? Select all that apply.

A. Every context-sensitive language is also a context-free language.

B. Every context-free language is also a context-sensitive language.

C. Context-free languages are a strict subset of context-sensitive languages.

D. Context-free languages are unrelated to context-sensitive languages.

Hint

Compare production restrictions, memory access and language-family containments. Distinguish an exact characterization from an included subclass.

Answer B, C: Every context-free language is also a context-sensitive language.; Context-free languages are a strict subset of context-sensitive languages.

Every context-free language is context-sensitive under the stated ε convention. The containment is strict: $a^n b^n c^n$ is context-sensitive but not context-free. Hence both the inclusive statement B and the stronger proper-subset statement C are true. A reverses the containment, and D denies an established relationship. Selecting the stronger statement alone would incorrectly discard a weaker statement that it logically entails.

Why each choice is right or wrong

A - Incorrect: The three-equal-count language is a CSL outside CFL.

B - Correct: Every CFL belongs to CSL.

C - Correct: CSL contains additional languages, making the inclusion strict.

D - Incorrect: The two families are related by strict containment.

Primary note proof

Study-note section AUT05

Type 3 regular grammars correspond to finite automata; type 2 CFGs to nondeterministic PDAs; type 1 context-sensitive/noncontracting grammars to nondeterministic LBAs; type 0 unrestricted grammars to Turing recognizers.

Study-note section AUT05

> Use REG ⊊ CFL ⊊ CSL ⊊ DECIDABLE ⊊ RE, under the usual convention allowing S→ε with S absent from right sides when needed.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Automata and Complexity Theory module | PDF page 10 | lines 6-6

> Fig: The traditional Chomsky hierarchy

Automata and Complexity Theory module | PDF page 8 | lines 20-21

> A context-free grammar
> can describe all regular languages and more, but they cannot describe all possible languages.

Question sources

Computer Science MCQ practice collection · Question 756

Question 64 | AM0757

Which listed machine family can recognize every context-sensitive language? Interpret D as a nondeterministic finite automaton.

A. Finite automaton

B. Pushdown automaton

C. Turing machine

D. Nondeterministic finite automaton

Hint

Compare production restrictions, memory access and language-family containments. Distinguish an exact characterization from an included subclass.

Answer C: Turing machine

Context-sensitive languages are recognized by nondeterministic linear bounded automata. An unrestricted Turing machine can simulate their bounded-tape computations, so it can recognize every CSL. Finite automata and NFAs recognize only regular languages, and ordinary PDAs recognize only context-free languages. Those weaker families cannot cover the three-equal-count CSL, so they fail the universal capability requirement.

Why each choice is right or wrong

A - Incorrect: Finite automata cannot recognize nonregular CSLs.

B - Incorrect: PDAs cannot recognize CSLs outside CFL.

C - Correct: A TM can simulate the LBA model for every CSL.

D - Incorrect: NFAs recognize only REG, not every CSL.

Primary note proof

Study-note section AUT05

Type 3 regular grammars correspond to finite automata; type 2 CFGs to nondeterministic PDAs; type 1 context-sensitive/noncontracting grammars to nondeterministic LBAs; type 0 unrestricted grammars to Turing recognizers.

Study-note section AUT05

Use REG ⊊ CFL ⊊ CSL ⊊ DECIDABLE ⊊ RE, under the usual convention allowing S→ε with S absent from right sides when needed.

Study-note section AUT25

A context-sensitive rule αAβ→αγβ replaces A in context, with γ nonempty. Equivalently, noncontracting grammar presentations preserve/increase total length, with the isolated ε exception. The condition is on the complete sides, not an unrelated comparison $|\beta| \geq |\alpha|$.

Study-note section AUT25

A linear bounded automaton (LBA) is a TM with O(n) tape cells on length-n input; the standard CSL characterization uses nondeterministic LBAs. For $a^n b^n c^n$ first verify a*b*c* order, repeatedly mark one unmarked a, b and c, and reject missing/excess symbols.

Study-note section AUT25

Thus every CSL is decidable.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Automata and Complexity Theory module | PDF page 10 | lines 6-6

Fig: The traditional Chomsky hierarchy

Automata and Complexity Theory module | PDF page 11 | lines 7-7

A Turing machine can do everything that a real computer can do.

Question sources

Computer Science MCQ practice collection · Question 757

Question 65 | AM0758

Which listed term names exactly the recursively enumerable language family, rather than a proper subfamily?

A. Regular language

B. Context-free language

C. Context-sensitive language

D. Turing-recognizable language

Hint

Check whether the required procedure must halt on every input, and distinguish solving a task from recognizing its yes-instances.

Answer D: Turing-recognizable language

Recursively enumerable, Turing-recognizable and semidecidable are names for the same language family. Regular, context-free and context-sensitive languages are all contained in RE, but each is a proper subfamily. The exact-family wording therefore singles out Turing-recognizable. Recognizers must halt and accept members; on a nonmember they may either reject or loop, unlike deciders which halt on every input.

Why each choice is right or wrong

A - Incorrect: REG is a proper subfamily of RE.

B - Incorrect: CFL is a proper subfamily of RE.

C - Incorrect: CSL is a proper subfamily of RE.

D - Correct: Turing-recognizable is an exact synonym for RE.

Primary note proof

Study-note section AUT29

> A decider halts on every input and accepts exactly its language; these languages are recursive/decidable. A recognizer must halt and accept every member, but on a nonmember may halt-reject or loop. Its languages are recursively enumerable (RE), Turing-recognizable or semidecidable. Looping is nonacceptance, not a rejecting halt. An undecidable language may still have a recognizer.

Study-note section AUT29

> Every decidable language is RE, and the containment is strict.

Study-note section AUT05

> Type 3 regular grammars correspond to finite automata; type 2 CFGs to nondeterministic PDAs; type 1 context-sensitive/noncontracting grammars to nondeterministic LBAs; type 0 unrestricted grammars to Turing recognizers.

Study-note section AUT05

> Use REG ⊊ CFL ⊊ CSL ⊊ DECIDABLE ⊊ RE, under the usual convention allowing S→ε with S absent from right sides when needed.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Automata and Complexity Theory module | PDF page 10 | lines 6-6

> Fig: The traditional Chomsky hierarchy

Automata and Complexity Theory module | PDF page 15 | lines 30-30

All recursive set is recursively enumerable.

Question sources

Computer Science MCQ practice collection · Question 758

Question 66 | AM0759

Which class of languages is the most powerful in terms of generative power?

A. Regular languages

B. Context-free languages

C. Context-sensitive languages

D. Recursively enumerable languages

Hint

Compare production restrictions, memory access and language-family containments. Distinguish an exact characterization from an included subclass.

Answer D: Recursively enumerable languages

Reading most powerful as largest generative family in the traditional hierarchy, the outermost region is recursively enumerable. It properly surrounds context-sensitive, context-free and regular classes.

Why each choice is right or wrong

A - Incorrect: Innermost family, hence smallest listed.

B - Incorrect: Contains regular but is inside context-sensitive.

C - Incorrect: Larger than CFL but still inside RE.

D - Correct: Outermost family in the diagram.

Primary note proof

Study-note section AUT05

> Type 3 regular grammars correspond to finite automata; type 2 CFGs to nondeterministic PDAs; type 1 context-sensitive/noncontracting grammars to nondeterministic LBAs; type 0 unrestricted grammars to Turing recognizers.

Study-note section AUT05

> Use REG ⊊ CFL ⊊ CSL ⊊ DECIDABLE ⊊ RE, under the usual convention allowing S→ε with S absent from right sides when needed.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Automata and Complexity Theory module | PDF page 10 | lines 6-6

> Fig: The traditional Chomsky hierarchy

Question sources

Computer Science MCQ practice collection · Question 759

Question 67 | AM0760

Which statements about recursive and recursively enumerable languages are true? Select all that apply.

A. Every recursive language is also a recursively enumerable language.

B. Every recursively enumerable language is also a recursive language.

C. Recursive languages are a strict subset of recursively enumerable languages.

D. Recursive languages are unrelated to recursively enumerable languages.

Hint

Check whether the required procedure must halt on every input, and distinguish solving a task from recognizing its yes-instances.

Answer A, C: Every recursive language is also a recursively enumerable language.; Recursive languages are a strict subset of recursively enumerable languages.

Every decider is also a recognizer, so each recursive language is recursively enumerable. The containment is proper: the halting language can be recognized by simulation but cannot be decided on every input. Thus both the inclusion statement and the strict-subset statement are true. The converse would turn every recognizer into a total decider, contradicting the halting example; describing the two families as unrelated also contradicts their inclusion.

Why each choice is right or wrong

A - Correct: A decider recognizes its language and halts on every input, establishing inclusion.

B - Incorrect: The recognizable but undecidable halting language is a counterexample.

C - Correct: The halting language belongs to RE but not to the recursive family, making inclusion strict.

D - Incorrect: The families are related by proper containment.

Primary note proof

Study-note section AUT29

> A decider halts on every input and accepts exactly its language; these languages are recursive/decidable. A recognizer must halt and accept every member, but on a nonmember may halt-reject or loop. Its languages are recursively enumerable (RE), Turing-recognizable or semidecidable. Looping is nonacceptance, not a rejecting halt. An undecidable language may still have a recognizer.

Study-note section AUT29

> Every decidable language is RE, and the containment is strict.

Study-note section AUT30

> HALT={⟨M,w⟩:M halts on w} is recognizable: simulate and accept upon any halt. Suppose a decider H determines halting. Construct D(x) that loops if H(x,x) says halt, otherwise halts. Running D on its own description contradicts H in both cases. Thus HALT is undecidable and

> witnesses RE beyond DECIDABLE. Its complement is not RE, or both recognizers would yield a decider.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Automata and Complexity Theory module | PDF page 15 | lines 30-30

> All recursive set is recursively enumerable.

Automata and Complexity Theory module | PDF page 15 | lines 24-26

> A set X is recursive if we have an algorithm to
> determine whether a given element belongs to
> X or not.

Question sources

Computer Science MCQ practice collection · Question 760

Question 68 | AM0761

Which of the following is an example of an undecidable problem?

A. Checking if a given number is prime

B. Sorting a list of numbers in ascending order

C. Solving a system of linear equations

D. Determining if a Turing machine halts on a given input

Hint

Check whether the required procedure must halt on every input, and distinguish solving a task from recognizing its yes-instances.

Answer D: Determining whether an arbitrary TM halts on its given input.

AUT30 proves HALT undecidable, directly matching D. The other choices describe ordinary finite mathematical computation: primality can test finitely many candidate divisors, a finite list can be sorted, and an exact finite linear system can be processed by elimination. Their cost can vary, but no general undecidability follows from those ordinary formulations.

Why each choice is right or wrong

A - Incorrect: For an integer, testing finitely many candidate divisors decides primality; AUT26 uses bounded divisor testing.

B - Incorrect: A finite numeric list can be sorted by a terminating comparison procedure.

C - Incorrect: An ordinarily encoded finite linear system can be decided and solved by finite elimination steps.

D - Correct: This is the general halting decision problem proved undecidable in AUT30.

Primary note proof

Study-note section AUT30

Thus HALT is undecidable and witnesses RE beyond DECIDABLE.

Study-note section AUT26

It is nevertheless context-sensitive: count the input in bounded workspace and test divisors.

Study-note section AUT-S04

Sorting a finite numeric list is a terminating computation. Quicksort, for example, chooses a pivot, partitions the other entries and recursively sorts the smaller sublists. Each recursive subproblem has fewer entries. A large running time does not make the task undecidable.

Study-note section AUT-S04

A finite linear system with exactly represented rational coefficients can be processed by Gaussian elimination and back substitution. Row exchanges handle unsuitable pivots; row reduction detects inconsistency or leaves free variables describing multiple solutions. These finite algebraic procedures differ from a general halting decider, which does not exist.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Automata and Complexity Theory module | PDF page 14 | lines 19-20

-It Is impossible to construct a single algorithm
that always leads to a correct yes-or-no answer.

Automata and Complexity Theory module | PDF page 15 | lines 24-26

A set X is recursive if we have an algorithm to
determine whether a given element belongs to
X or not.

Question sources

Computer Science MCQ practice collection · Question 761

Question 69 | AM0762

Which offered language is nonrecursive (undecidable)? Use finite standard encodings and a fixed finite alphabet for mirror words.

A. HALT={⟨M,w⟩ | M halts on w}

B. {an | n is a prime number}

C. {wwR | w is any string}

D. {anb | n ≥ 0}

Hint

Check whether the required procedure must halt on every input, and distinguish solving a task from recognizing its yes-instances.

Answer A: HALT={⟨M,w⟩ | M halts on w}

The halting language HALT contains encoded machine/input pairs for which the machine eventually halts. The diagonal argument in the note proves that no algorithm decides HALT on every input. The remaining examples are decidable: unary primality uses finite divisor testing, mirror words can be compared finitely, and a^nb is regular. Thus nonregularity or an expensive test alone does not imply nonrecursiveness.

Why each choice is right or wrong

A - Correct: HALT is recognizable but undecidable by the diagonal argument.

B - Incorrect: Finite counting and divisor testing decide unary prime length.

C - Incorrect: A finite mirror comparison decides palindrome membership.

D - Incorrect: The language a^nb is regular and hence decidable.

Primary note proof

Study-note section AUT30

> HALT={⟨M,w⟩:M halts on w} is recognizable: simulate and accept upon any halt. Suppose a decider H determines halting. Construct D(x) that loops if H(x,x) says halt, otherwise halts. Running D on its own description contradicts H in both cases. Thus HALT is undecidable and witnesses RE beyond DECIDABLE. Its complement is not RE, or both recognizers would yield a decider.

Study-note section AUT26

> a*b* and $\{a^nb:n\geq 0\}$ are regular. a^nb^n is CFL but not regular. Over a two-symbol alphabet, ww^R is CFL, ww is not CFL; both are decidable. $a^nb^nc^n$ is CSL but not CFL, and therefore decidable. Every finite language is regular. A stronger grammar or TM can still recognize any of these lower-family languages.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Automata and Complexity Theory module | PDF page 15 | lines 24-26

> A set X is recursive if we have an algorithm to
> determine whether a given element belongs to
> X or not.

Automata and Complexity Theory module | PDF page 11 | lines 7-7

> A Turing machine can do everything that a real computer can do.

Question sources

Computer Science MCQ practice collection · Question 762

Question 70 | AM0763

At the level of generated language families, what criterion orders the Chomsky hierarchy?

A. It categorizes formal languages based on their computational complexity.

B. It categorizes formal languages based on their expressive power.

C. It categorizes formal languages based on their alphabet size.

D. It categorizes languages by the surface structure of individual input strings.

Hint

Compare production restrictions, memory access and language-family containments. Distinguish an exact characterization from an included subclass.

Answer B: It categorizes formal languages based on their expressive power.

The hierarchy orders families by expressive power: regular languages form a proper subfamily of CFLs, then CSLs, then the stronger computable families. Grammar production restrictions define the corresponding types, but the resulting language-family comparison concerns generative capability. It is not a ranking of algorithm runtimes, alphabet sizes, or surface appearance of particular strings. That scope makes B the unique criterion.

Why each choice is right or wrong

A - Incorrect: The hierarchy is not a runtime or workspace ranking.

B - Correct: It compares the language families that the models can describe.

C - Incorrect: Alphabet size does not determine the family.

D - Incorrect: Surface structure of individual strings is not the language-family ordering criterion.

Primary note proof

Study-note section AUT05

> Type 3 regular grammars correspond to finite automata; type 2 CFGs to nondeterministic PDAs; type 1 context-sensitive/noncontracting grammars to nondeterministic LBAs; type 0 unrestricted grammars to Turing recognizers.

Study-note section AUT05

> Use REG ⊊ CFL ⊊ CSL ⊊ DECIDABLE ⊊ RE, under the usual convention allowing S→ε with S absent from right sides when needed.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Automata and Complexity Theory module | PDF page 10 | lines 6-6

> Fig: The traditional Chomsky hierarchy

Automata and Complexity Theory module | PDF page 3 | lines 13-14

A grammar defines a set of rules, and with the use of these rules, valid sentences in a language are
generated.

Question sources

Computer Science MCQ practice collection · Question 763

Question 71 | AM0764

What is the least expressive listed grammar family that generates $L=\{a^nb^n \mid n\geq 0\}$? Use the usual isolated-start ε convention.

A. Regular grammar

B. Context-free grammar

C. Context-sensitive grammar

D. Unrestricted grammar

Hint

Respect the quantifier order and length constraints. A valid nonmembership argument must address every allowed decomposition, not one convenient split.

Answer B: Context-free grammar

The CFG $S\to aSb|\varepsilon$ generates exactly equal blocks of a and b: every recursive step adds one of each and the base supplies n=0. The pumping argument rules out regularity. Context-sensitive and unrestricted grammars can also generate this language, but are stronger than necessary under the ε convention. The least expressive sufficient family is therefore context-free.

Why each choice is right or wrong

A - Incorrect: Regular grammars cannot enforce these unbounded equal counts.

B - Correct: The displayed matched recursive CFG generates the language.

C - Incorrect: Context-sensitive grammars are stronger than necessary.

D - Incorrect: Unrestricted grammars are stronger than necessary.

Primary note proof

Study-note section AUT17

Use a recursive rule to enforce matched structure and a base rule to stop. $S\to aSb|\varepsilon$ generates exactly a^nb^n for $n\geq 0$: n recursive steps add n a's on the left and n b's on the right. Derive aabb by $S\Rightarrow aSb\Rightarrow aaSbb\Rightarrow aabb$. For $n\geq 1$ replace ε by base ab.

Study-note section AUT15

If L is regular, some $p\geq 1$ satisfies: every $s\in L$ with $|s|\geq p$ has a split $s=xyz$, $|xy|\leq p$, $|y|\geq 1$, and $xy^iz\in L$ for every $i\geq 0$.

Study-note section AUT15

For L={$a^n b^n$:n≥0}, choose $a^p b^p$. Every allowed y=a^k with k≥1; pumping i=0 leaves fewer a's than b's. The contradiction excludes every DFA.

Study-note section AUT05

Type 3 regular grammars correspond to finite automata; type 2 CFGs to nondeterministic PDAs; type 1 context-sensitive/noncontracting grammars to nondeterministic LBAs; type 0 unrestricted grammars to Turing recognizers.

Study-note section AUT05

Use REG ⊊ CFL ⊊ CSL ⊊ DECIDABLE ⊊ RE, under the usual convention allowing S→ε with S absent from right sides when needed.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Automata and Complexity Theory module | PDF page 8 | lines 27-28

R is a set of production rules where each production rule maps a variable to a string s∈(V∪Σ)∗;

Automata and Complexity Theory module | PDF page 8 | lines 12-12

(1) |uv| ≤ n (2) |v| ≥ 1 (3) for all i ≥ 0: uviw ∈ L

Automata and Complexity Theory module | PDF page 10 | lines 6-6

Fig: The traditional Chomsky hierarchy

Question sources

Computer Science MCQ practice collection · Question 764

Question 72 | AM0765

Which listed language is context-sensitive but NOT context-free? Interpret the notation as powers and use the usual ε convention.

A. {$a^n b^n$ | n≥0}

B. {$a^n b^m$ | n,m≥0}

C. {a^n | n is prime}

D. {$a^n b$ | n≥0}

Hint

Compare production restrictions, memory access and language-family containments. Distinguish an exact characterization from an included subclass.

Answer C: {a^n | n is prime}

Unary prime length is context-sensitive because bounded-workspace counting and divisor testing decide it. It is not regular by the prime-length pumping argument, and every unary CFL is regular; consequently it is not context-free. In contrast, a^nb^n is context-free, while independently chosen a/b counts and any number of a followed by one b are regular. These weaker examples are excluded by the explicit non-CFL condition.

Why each choice is right or wrong

A - Incorrect: Equal two-block counts form a CFL.

B - Incorrect: Independent counts give the regular language a*b*.

C - Correct: Unary prime length is CSL but not CFL.

D - Incorrect: Any number of a followed by one b is regular.

Primary note proof

Study-note section AUT26

> a*b* and {a^nb:n≥0} are regular. a^nb^n is CFL but not regular. Over a two-symbol alphabet, ww^R is CFL, ww is not CFL; both are decidable. $a^nb^nc^n$ is CSL but not CFL, and therefore decidable. Every finite language is regular. A stronger grammar or TM can still recognize any of these lower-family languages.

Study-note section AUT26

> Unary prime length {a^p:p prime} is nonregular: choose prime q above pumping length p; if pump piece has length k≥1, using exponent q+1 gives length q(k+1), composite. Every unary CFL is regular, so this language is not CFL either. It is nevertheless context-sensitive: count the input in bounded workspace and test divisors. The balanced-three-count, prime-length, palindrome and a^nb examples above are all recursive languages.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Automata and Complexity Theory module | PDF page 8 | lines 27-28

> R is a set of production rules where each production rule maps a variable to a string s∈(V∪Σ)∗;

Automata and Complexity Theory module | PDF page 5 | lines 22-22

> In a regular expression, x* means zero or more occurrence of x.

Automata and Complexity Theory module | PDF page 10 | lines 6-6

> Fig: The traditional Chomsky hierarchy

Question sources

Computer Science MCQ practice collection · Question 765

Question 73 | AM0766

What is the least expressive listed machine family that recognizes $L=\{a^nb^n \mid n\geq 0\}$? Interpret D as a nondeterministic finite automaton.

A. Finite automaton

B. Pushdown automaton

C. Turing machine

D. Nondeterministic finite automaton

Hint

Respect the quantifier order and length constraints. A valid nonmembership argument must address every allowed decomposition, not one convenient split.

Answer B: Pushdown automaton

A PDA pushes one stack symbol for each a, then pops one for each b, accepting exactly when the blocks finish together. The language is nonregular, so neither deterministic nor nondeterministic finite automata suffice. A TM can simulate the matching procedure, but has greater expressive power than required. The least sufficient listed family is the pushdown automaton.

Why each choice is right or wrong

A - Incorrect: Finite automata cannot recognize the nonregular matched-count language.

B - Correct: A PDA stack matches each a with one later b.

C - Incorrect: A TM works but is stronger than necessary.

D - Incorrect: An NFA recognizes only regular languages.

Primary note proof

Study-note section AUT22

> For a^nb^n, push one X per a, switch to a b-reading phase, pop one X per b, and accept when input ends at the bottom marker. Reject a after entering the b phase and b when no X remains. Handle ε explicitly if n=0 is permitted. A stack counts without storing the full input as random-access memory.

Study-note section AUT15

> If L is regular, some $p\geq 1$ satisfies: every $s\in L$ with $|s|\geq p$ has a split $s=xyz$, $|xy|\leq p$, $|y|\geq 1$, and $xy^iz\in L$ for every $i\geq 0$.

Study-note section AUT15

> For $L=\{a^nb^n : n\geq 0\}$, choose a^pb^p. Every allowed $y=a^k$ with $k\geq 1$; pumping $i=0$ leaves fewer a's than b's. The contradiction excludes every DFA.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Automata and Complexity Theory module | PDF page 8 | lines 27-28

> R is a set of production rules where each production rule maps a variable to a string $s \in (V \cup \Sigma)*$;

Automata and Complexity Theory module | PDF page 9 | lines 23-23

> Pushdown automata accept context-free languages, which include the set of regular languages.

Automata and Complexity Theory module | PDF page 8 | lines 12-12

> (1) $|uv| \leq n$ (2) $|v| \geq 1$ (3) for all $i \geq 0$: $uviw \in L$

Automata and Complexity Theory module | PDF page 10 | lines 6-6

> Fig: The traditional Chomsky hierarchy

Question sources

Computer Science MCQ practice collection · Question 766

Question 74 | AM0767

Over $\Sigma=\{a,b\}$, what is the least expressive listed grammar family generating $L=\{ww^R \mid w \in \Sigma^*\}$? Use the usual isolated-start ε convention.

A. Regular grammar

B. Context-free grammar

C. Context-sensitive grammar

D. Unrestricted grammar

Hint

Compare production restrictions, memory access and language-family containments. Distinguish an exact characterization from an included subclass.

Answer B: Context-free grammar

The grammar $S \to aSa \mid bSb \mid \varepsilon$ generates precisely binary even-length palindromes: matching symbols wrap a shorter mirror word until the empty center is reached. This language is context-free and nonregular over the stated two-symbol alphabet. Context-sensitive and unrestricted grammars also suffice but exceed the minimum expressive power required. Fixing the alphabet excludes the unary even-length case, which would be regular.

Why each choice is right or wrong

A - Incorrect: Binary unmarked even palindromes are nonregular.

B - Correct: The wrapping CFG generates exactly this mirror language.

C - Incorrect: Context-sensitive grammars are stronger than necessary.

D - Incorrect: Unrestricted grammars are stronger than necessary.

Primary note proof

Study-note section AUT22

For even palindromes ww^R over {a,b}, push the first half, nondeterministically guess its midpoint, then pop matching symbols. No center marker makes that language context-free but not deterministic context-free. With a marker wcw^R the switch point is visible and a DPDA suffices. Unary even palindromes are merely even-length words and regular. A PDA cannot compare two arbitrary halves in the same forward order: {ww:w∈{a,b}*} is the copy language, not the palindrome language.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Automata and Complexity Theory module | PDF page 8 | lines 27-28

R is a set of production rules where each production rule maps a variable to a string $s \in (V \cup \Sigma)*$;

Automata and Complexity Theory module | PDF page 9 | lines 23-23

Pushdown automata accept context-free languages, which include the set of regular languages.

Automata and Complexity Theory module | PDF page 10 | lines 6-6

Fig: The traditional Chomsky hierarchy

Automata and Complexity Theory module | PDF page 8 | lines 12-12

(1) $|uv| \leq n$ (2) $|v| \geq 1$ (3) for all $i \geq 0$: $uviw \in L$

Question sources

Computer Science MCQ practice collection · Question 767

Question 75 | AM0768

Over Σ={a,b}, what is the least expressive listed machine family recognizing $L=\{ww^R \mid w \in \Sigma^*\}$? There is no center marker; D means NFA.

A. Finite automaton

B. Nondeterministic pushdown automaton

C. Turing machine

D. Nondeterministic finite automaton

Hint

Compare production restrictions, memory access and language-family containments. Distinguish an exact characterization from an included subclass.

Answer B: Nondeterministic pushdown automaton

An NPDA pushes the first half, guesses the midpoint and matches the remaining symbols against the stack in reverse order. That recognizes the binary even-palindrome language, which is nonregular. Finite automata and NFAs therefore fail, whereas a TM can recognize it but is stronger than necessary. The unmarked midpoint requires nondeterminism in the PDA family, as distinguished in the note.

Why each choice is right or wrong

A - Incorrect: A finite automaton cannot recognize binary unmarked even palindromes.

B - Correct: A nondeterministic PDA guesses the midpoint and matches using its stack.

C - Incorrect: A TM is sufficient but stronger than necessary.

D - Incorrect: An NFA still recognizes only regular languages.

Primary note proof

Study-note section AUT22

> For even palindromes ww^R over {a,b}, push the first half, nondeterministically guess its midpoint, then pop matching symbols. No center marker makes that language context-free but not deterministic context-free. With a marker wcw^R the switch point is visible and a DPDA suffices. Unary even palindromes are merely even-length words and regular. A PDA cannot compare two arbitrary halves in the same forward order: {ww:w∈{a,b}*} is the copy language, not the palindrome language.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Automata and Complexity Theory module | PDF page 8 | lines 27-28

> R is a set of production rules where each production rule maps a variable to a string $s \in (V \cup \Sigma)^*$;

Automata and Complexity Theory module | PDF page 9 | lines 23-23

> Pushdown automata accept context-free languages, which include the set of regular languages.

Automata and Complexity Theory module | PDF page 10 | lines 6-6

> Fig: The traditional Chomsky hierarchy

Automata and Complexity Theory module | PDF page 8 | lines 12-12

> (1) $|uv| \leq n$ (2) $|v| \geq 1$ (3) for all $i \geq 0$: $uviw \in L$

Question sources

Computer Science MCQ practice collection · Question 768

Question 76 | AM0769

What is the least expressive listed grammar family generating $L=\{0^n1^n2^n \mid n\geq 0\}$? Use the usual isolated-start ε convention.

A. Regular grammar

B. Context-free grammar

C. Context-sensitive grammar

D. Unrestricted grammar

Hint

Respect the quantifier order and length constraints. A valid nonmembership argument must address every allowed decomposition, not one convenient split.

Answer C: Context-sensitive grammar

Renaming the symbols in $a^nb^nc^n$ gives $0^n1^n2^n$ without changing its classification. The three matched blocks form a context-sensitive language but not a context-free one. The isolated-start ε convention covers n=0. Regular grammars are weaker still, and unrestricted grammars can generate it but exceed what is needed. Thus the least sufficient listed grammar family is context-sensitive.

Why each choice is right or wrong

A - Incorrect: Regular grammars cannot generate this nonregular language.

B - Incorrect: The language lies outside the context-free family.

C - Correct: The context-sensitive family suffices and is the least listed family that does.

D - Incorrect: Unrestricted grammars suffice but exceed the required expressive power.

Primary note proof

Study-note section AUT25

> A context-sensitive rule $\alpha A\beta\rightarrow\alpha\gamma\beta$ replaces A in context, with γ nonempty. Equivalently, noncontracting grammar presentations preserve/increase total length, with the isolated ε exception. The condition is on the complete sides, not an unrelated comparison $|\beta|\geq|\alpha|$.

Study-note section AUT25

> A linear bounded automaton (LBA) is a TM with O(n) tape cells on length-n input; the standard CSL characterization uses nondeterministic LBAs. For $a^nb^nc^n$ first verify a*b*c* order, repeatedly mark one unmarked a, b and c, and reject missing/excess symbols.

Study-note section AUT25

> Thus every CSL is decidable.

Study-note section AUT24

If L is context-free, some p ensures each sufficiently long $s \in L$ has $s=uvxyz$ with $|vxy| \le p$, $|vy| \ge 1$ and $uv^ixy^iz \in L$ for every $i \ge 0$. The two pieces v and y are pumped together; either may be empty, not both. A non-CFL proof must defeat every allowed decomposition.

Study-note section AUT24

For $s=a^pb^pc^p$, the window vxy cannot contain both an a and a c. Pumping affects at most two blocks and at least one count. Use i=0 or 2: the unaffected third block retains p while another count changes, or symbol order breaks. Hence $a^nb^nc^n$ is not CFL.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Automata and Complexity Theory module | PDF page 8 | lines 27-28

R is a set of production rules where each production rule maps a variable to a string $s \in (V \cup \Sigma)*$;

Automata and Complexity Theory module | PDF page 10 | lines 6-6

Fig: The traditional Chomsky hierarchy

Question sources

Computer Science MCQ practice collection · Question 769

Question 77 | AM0770

Which listed machine family can recognize $L=\{0^n1^n2^n \mid n \ge 0\}$? Interpret D as a nondeterministic finite automaton.

A. Finite automaton

B. Pushdown automaton

C. Turing machine

D. Nondeterministic finite automaton

Hint

Respect the quantifier order and length constraints. A valid nonmembership argument must address every allowed decomposition, not one convenient split.

Answer C: Turing machine

A Turing machine verifies the 0*1*2* order, repeatedly marks one symbol from each block, and checks that none remains unmatched. Each round marks additional symbols, so the procedure halts. This is the three-equal-count language with renamed symbols, which is not context-free. Consequently neither a finite automaton, an NFA nor a single-stack PDA recognizes it, leaving the TM as the sufficient listed model.

Why each choice is right or wrong

A - Incorrect: Finite automata cannot recognize this nonregular language.

B - Incorrect: A single-stack PDA cannot recognize this non-CFL language.

C - Correct: A TM implements the terminating membership test described in the note.

D - Incorrect: An NFA recognizes only REG and cannot recognize this language.

Primary note proof

Study-note section AUT25

A context-sensitive rule $\alpha A\beta \to \alpha\gamma\beta$ replaces A in context, with γ nonempty. Equivalently, noncontracting grammar presentations preserve/increase total length, with the isolated ε exception. The condition is on the complete sides, not an unrelated comparison $|\beta| \geq |\alpha|$.

Study-note section AUT25

A linear bounded automaton (LBA) is a TM with O(n) tape cells on length-n input; the standard CSL characterization uses nondeterministic LBAs. For $a^n b^n c^n$ first verify a*b*c* order, repeatedly mark one unmarked a, b and c, and reject missing/excess symbols.

Study-note section AUT25

Thus every CSL is decidable.

Study-note section AUT24

If L is context-free, some p ensures each sufficiently long $s \in L$ has $s=uvxyz$ with $|vxy| \leq p$, $|vy| \geq 1$ and $uv^i xy^i z \in L$ for every $i \geq 0$. The two pieces v and y are pumped together; either may be empty, not both. A non-CFL proof must defeat every allowed decomposition.

Study-note section AUT24

For $s=a^p b^p c^p$, the window vxy cannot contain both an a and a c. Pumping affects at most two blocks and at least one count. Use i=0 or 2: the unaffected third block retains p while another count changes, or symbol order breaks. Hence $a^n b^n c^n$ is not CFL.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Automata and Complexity Theory module | PDF page 11 | lines 7-7

A Turing machine can do everything that a real computer can do.

Automata and Complexity Theory module | PDF page 15 | lines 24-26

A set X is recursive if we have an algorithm to
determine whether a given element belongs to
X or not.

Automata and Complexity Theory module | PDF page 10 | lines 6-6

Fig: The traditional Chomsky hierarchy

Question sources

Computer Science MCQ practice collection · Question 770

Question 78 | AM0771

What is the least expressive listed grammar family generating $L=\{a^n \mid n \text{ is prime}\}$? Use the usual isolated-start ε convention.

A. Regular grammar

B. Context-free grammar

C. Context-sensitive grammar

D. Unrestricted grammar

Hint

Compare production restrictions, memory access and language-family containments. Distinguish an exact characterization from an included subclass.

Answer C: Context-sensitive grammar

Unary prime length is nonregular by the pumping argument. Since every unary context-free language is regular, it is not context-free either. Bounded-workspace counting and divisor testing establish context sensitivity. Unrestricted grammars also suffice but are more expressive than necessary. The least sufficient listed family is therefore context-sensitive, rather than one of the two weaker grammar families.

Why each choice is right or wrong

A - Incorrect: Regular grammars cannot generate this nonregular language.

B - Incorrect: The language lies outside the context-free family.

C - Correct: The context-sensitive family suffices and is the least listed family that does.

D - Incorrect: Unrestricted grammars suffice but exceed the required expressive power.

Primary note proof

Study-note section AUT26

> a^*b^* and $\{a^nb : n \geq 0\}$ are regular. a^nb^n is CFL but not regular. Over a two-symbol alphabet, ww^R is CFL, ww is not CFL; both are decidable. $a^nb^nc^n$ is CSL but not CFL, and therefore decidable. Every finite language is regular. A stronger grammar or TM can still recognize any of these lower-family languages.

Study-note section AUT26

> Unary prime length $\{a^p : p \text{ prime}\}$ is nonregular: choose prime q above pumping length p; if pump piece has length $k \geq 1$, using exponent q+1 gives length q(k+1), composite. Every unary CFL is regular, so this language is not CFL either. It is nevertheless context-sensitive: count the input in bounded workspace and test divisors. The balanced-three-count, prime-length, palindrome and a^nb examples above are all recursive languages.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Automata and Complexity Theory module | PDF page 8 | lines 12-12

> (1) $|uv| \leq n$ (2) $|v| \geq 1$ (3) for all $i \geq 0$: $uviw \in L$

Automata and Complexity Theory module | PDF page 10 | lines 6-6

> Fig: The traditional Chomsky hierarchy

Automata and Complexity Theory module | PDF page 15 | lines 24-26

> A set X is recursive if we have an algorithm to
> determine whether a given element belongs to
> X or not.

Question sources

Computer Science MCQ practice collection · Question 771

Question 79 | AM0772

Which listed machine family can recognize $L=\{a^n \mid n \text{ is prime}\}$? Interpret D as a nondeterministic finite automaton.

A. Finite automaton

B. Pushdown automaton

C. Turing machine

D. Nondeterministic finite automaton

Hint

Compare production restrictions, memory access and language-family containments. Distinguish an exact characterization from an included subclass.

Answer C: Turing machine

A Turing machine can count the input and test finitely many divisors to determine whether its length is prime. This procedure halts on every input. The language is nonregular, and every unary CFL is regular, so it is also outside CFL. Finite automata, NFAs and single-stack PDAs therefore fail, while the TM provides the required recognition procedure.

Why each choice is right or wrong

A - Incorrect: Finite automata cannot recognize this nonregular language.

B - Incorrect: A single-stack PDA cannot recognize this non-CFL language.

C - Correct: A TM implements the terminating membership test described in the note.

D - Incorrect: An NFA recognizes only REG and cannot recognize this language.

Primary note proof

Study-note section AUT26

a*b* and {$a^n b$:n≥0} are regular. $a^n b^n$ is CFL but not regular. Over a two-symbol alphabet, ww^R is CFL, ww is not CFL; both are decidable. $a^n b^n c^n$ is CSL but not CFL, and therefore decidable. Every finite language is regular. A stronger grammar or TM can still recognize any of these lower-family languages.

Study-note section AUT26

Unary prime length {a^p:p prime} is nonregular: choose prime q above pumping length p; if pump piece has length k≥1, using exponent q+1 gives length q(k+1), composite. Every unary CFL is regular, so this language is not CFL either. It is nevertheless context-sensitive: count the input in bounded workspace and test divisors. The balanced-three-count, prime-length, palindrome and $a^n b$ examples above are all recursive languages.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Automata and Complexity Theory module | PDF page 11 | lines 7-7

A Turing machine can do everything that a real computer can do.

Automata and Complexity Theory module | PDF page 15 | lines 24-26

A set X is recursive if we have an algorithm to
determine whether a given element belongs to
X or not.

Automata and Complexity Theory module | PDF page 8 | lines 12-12

(1) |uv| ≤ n (2) |v| ≥ 1 (3) for all i ≥ 0: uviw ∈ L

Question sources

Computer Science MCQ practice collection · Question 772

Question 80 | AM0773

What is the least expressive listed grammar family generating L={$a^n b$ | n≥0}?

A. Regular grammar

B. Context-free grammar

C. Context-sensitive grammar

D. Unrestricted grammar

Hint

Track mandatory symbols and factors that may repeat or disappear. Use short test words to check both directions of each proposed language description.

Answer A: Regular grammar

A right-linear grammar S→aS|b generates any number of a symbols followed by exactly one b, including the word b for n=0. The language is therefore regular. Context-free, context-sensitive and unrestricted grammars can all generate regular languages too, but are more expressive than necessary. The least-family question uniquely selects regular grammar without denying the capabilities of the stronger classes.

Why each choice is right or wrong

A - Correct: The right-linear rule S→aS|b suffices.

B - Incorrect: CFGs work but are stronger than necessary.

C - Incorrect: Context-sensitive grammars work but are stronger than necessary.

D - Incorrect: Unrestricted grammars work but are stronger than necessary.

Primary note proof

Study-note section AUT14

> Right-linear rules have form A→aB, A→a or permitted ε rules; left-linear rules put the single variable at the opposite end, A→Ba. Both consistent orientations generate REG; arbitrarily mixing them can generate nonregular languages, for example S→aA|ε, A→Sb generates $a^n b^n$. A CFG rule A→aBA is legal but not regular-grammar form.

Study-note section AUT14

> Regular languages are closed under union, concatenation, star, reversal, intersection, difference and complement.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Automata and Complexity Theory module | PDF page 5 | lines 22-22

> In a regular expression, x* means zero or more occurrence of x.

Automata and Complexity Theory module | PDF page 8 | lines 2-2

> The language defined by regular grammar is known as regular language.

Automata and Complexity Theory module | PDF page 10 | lines 6-6

> Fig: The traditional Chomsky hierarchy

Automata and Complexity Theory module | PDF page 8 | lines 27-28

> R is a set of production rules where each production rule maps a variable to a string s∈(V∪Σ)⁎;

Question sources

Computer Science MCQ practice collection · Question 773

Question 81 | AM0774

Which class is defined as the decision languages decidable by a deterministic algorithm in polynomial time?

A. P

B. NP

C. PSPACE

D. EXPTIME

Hint

Separate resource bounds, certificate verification and reduction requirements. Use proved containments without assuming unresolved class separations.

Answer A: P

P is defined by deterministic polynomial-time decidability. NP, PSPACE and EXPTIME all contain P, so broad containment would not distinguish the choices. They have different defining bounds: nondeterministic polynomial time, deterministic polynomial workspace and deterministic exponential time, respectively. The explicit deterministic-time definition uniquely identifies P without assuming any unresolved separation among neighboring classes.

Why each choice is right or wrong

A - Correct: P is defined by deterministic polynomial-time decisions.

B - Incorrect: NP is defined by nondeterministic polynomial time or polynomial verification.

C - Incorrect: PSPACE is defined by polynomial workspace.

D - Incorrect: EXPTIME is defined by exponential-time bounds rather than precisely the stated polynomial-time bound.

Primary note proof

Study-note section AUT34

> Complexity classes here contain decision languages. P consists of deterministic polynomial-time decidable languages; NP consists of nondeterministic polynomial-time decidable languages, equivalently polynomially verifiable yes-instances with polynomial-size certificates. A verifier must accept some certificate exactly for yes-inputs; no certificate may falsely establish a no-instance. NP means nondeterministic polynomial, not non-polynomial.

Study-note section AUT34

> P⊆NP because a verifier can ignore its certificate and run the P algorithm. P=NP remains open; NP does not mean definitely hard or definitely outside P.

Study-note section AUT39

> Use L⊆NL⊆P⊆NP⊆PSPACE⊆EXPTIME⊆EXPSPACE. Here L/NL use logarithmic workspace; EXPTIME is ⋃kTIME(2^(n^k)), an upper-bound class that also contains P, not a statement that every member requires exponential time.

Study-note section AUT39

> P versus NP, NP versus PSPACE and PSPACE versus EXPTIME remain unresolved. coNP contains complements of NP languages; P⊆NP∩coNP, and NP=coNP is open.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Automata and Complexity Theory module | PDF page 17 | lines 27-28

> P problems are set of problems which can be solved in
> polynomial time by deterministic algorithms.

Automata and Complexity Theory module | PDF page 17 | lines 29-30

> NP problems are the problems which can be
> solved in non-deterministic polynomial time.

Question sources

Computer Science MCQ practice collection · Question 774

Question 82 | AM0775

Which class consists of decision problems whose yes-instances have polynomial-size certificates checkable by a deterministic polynomial-time verifier?

A. P

B. NP

C. PSPACE

D. EXPTIME

Hint

Separate resource bounds, certificate verification and reduction requirements. Use proved containments without assuming unresolved class separations.

Answer B: NP

NP is defined by polynomially bounded certificates and deterministic polynomial-time verification, equivalently nondeterministic polynomial-time decision. P instead names deterministic polynomial-time decision; its members also admit efficient verification, but P is not the name for the full verifier-defined class. PSPACE bounds deterministic working space by a polynomial, and EXPTIME bounds deterministic time exponentially. These are different resource-based definitions; their inclusion relationships do not make them alternative names for NP.

Why each choice is right or wrong

A - Incorrect: P is defined by deterministic polynomial-time decision, not by the full certificate-verification characterization.

B - Correct: The certificate-and-verifier definition is precisely NP.

C - Incorrect: PSPACE is defined by a polynomial bound on deterministic space.

D - Incorrect: EXPTIME is defined by exponential deterministic time bounds.

Primary note proof

Study-note section AUT34

> Complexity classes here contain decision languages. P consists of deterministic polynomial-time decidable languages; NP consists of nondeterministic polynomial-time decidable languages, equivalently polynomially verifiable yes-instances with polynomial-size certificates. A verifier must accept some certificate exactly for yes-inputs; no certificate may falsely establish a no-instance. NP means nondeterministic polynomial, not non-polynomial.

Study-note section AUT34

> P⊆NP because a verifier can ignore its certificate and run the P algorithm. P=NP remains open; NP does not mean definitely hard or definitely outside P.

Study-note section AUT38

> PSPACE is ⋃kSPACE(n^k), using deterministic polynomial workspace. NP⊆PSPACE because candidate polynomial witnesses can be enumerated and checked one at a time, reusing memory; exponentially many candidates do not require retaining them all.

Study-note section AUT39

> Use L⊆NL⊆P⊆NP⊆PSPACE⊆EXPTIME⊆EXPSPACE. Here L/NL use logarithmic workspace; EXPTIME is ⋃kTIME(2^(n^k)), an upper-bound class that also contains P, not a statement that every member requires exponential time.

Study-note section AUT39

> P versus NP, NP versus PSPACE and PSPACE versus EXPTIME remain unresolved. coNP contains complements of NP languages; P⊆NP∩coNP, and NP=coNP is open.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Automata and Complexity Theory module | PDF page 17 | lines 29-30

> NP problems are the problems which can be
> solved in non-deterministic polynomial time.

Automata and Complexity Theory module | PDF page 17 | lines 33-35

> The problem belongs to NP, if it‘s easy to check
> a solution that may have been very tedious to
> find.

Automata and Complexity Theory module | PDF page 17 | lines 27-28

> P problems are set of problems which can be solved in
> polynomial time by deterministic algorithms.

Question sources

Computer Science MCQ practice collection · Question 775

Question 83 | AM0776

Which class is defined as the decision languages decidable using deterministic polynomial workspace?

A. P

B. NP

C. PSPACE

D. EXPTIME

Hint

Separate resource bounds, certificate verification and reduction requirements. Use proved containments without assuming unresolved class separations.

Answer C: PSPACE

PSPACE consists of decision languages decidable with deterministic workspace bounded by a polynomial in input length. P and NP use polynomial-time definitions, whereas EXPTIME uses an exponential-time definition and contains PSPACE. Those containments do not change which class the workspace condition defines. Nor is it necessary to assume unresolved strict separations among the classes to identify the definition.

Why each choice is right or wrong

A - Incorrect: P has a deterministic polynomial-time definition.

B - Incorrect: NP has a nondeterministic polynomial-time or certificate-verification definition.

C - Correct: PSPACE has exactly the stated deterministic polynomial-workspace definition.

D - Incorrect: EXPTIME contains PSPACE but is defined by a time bound.

Primary note proof

Study-note section AUT38

PSPACE is $\bigcup_k \mathrm{SPACE}(n^k)$, using deterministic polynomial workspace. NP⊆PSPACE because candidate polynomial witnesses can be enumerated and checked one at a time, reusing memory; exponentially many candidates do not require retaining them all.

Study-note section AUT34

Complexity classes here contain decision languages. P consists of deterministic polynomial-time decidable languages; NP consists of nondeterministic polynomial-time decidable languages, equivalently polynomially verifiable yes-instances with polynomial-size certificates. A verifier must accept some certificate exactly for yes-inputs; no certificate may falsely establish a no-instance. NP means nondeterministic polynomial, not non-polynomial.

Study-note section AUT34

P⊆NP because a verifier can ignore its certificate and run the P algorithm. P=NP remains open; NP does not mean definitely hard or definitely outside P.

Study-note section AUT39

Use L⊆NL⊆P⊆NP⊆PSPACE⊆EXPTIME⊆EXPSPACE. Here L/NL use logarithmic workspace; EXPTIME is ⋃kTIME(2^(n^k)), an upper-bound class that also contains P, not a statement that every member requires exponential time.

Study-note section AUT39

P versus NP, NP versus PSPACE and PSPACE versus EXPTIME remain unresolved. coNP contains complements of NP languages; P⊆NP∩coNP, and NP=coNP is open.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Automata and Complexity Theory module | PDF page 16 | lines 4-5

Space Complexity: Determine the approximate memory required to solve a problem of size n.

Automata and Complexity Theory module | PDF page 17 | lines 27-28

P problems are set of problems which can be solved in polynomial time by deterministic algorithms.

Automata and Complexity Theory module | PDF page 17 | lines 29-30

NP problems are the problems which can be solved in non-deterministic polynomial time.

Question sources

Computer Science MCQ practice collection · Question 776

Question 84 | AM0777

Which concept is used to show that one problem can be reduced to another problem in order to prove hardness or completeness?

A. Reductions

B. Hierarchy

C. Completeness

D. Complexity classes

Hint

Separate resource bounds, certificate verification and reduction requirements. Use proved containments without assuming unresolved class separations.

Answer A: Reductions.

AUT35 defines a polynomial reduction as a computable transformation preserving yes/no membership and explains how reducing a known hard problem to a new target proves target hardness. AUT32 supplies the computable version for undecidability. Thus reductions name the proof concept itself, while hierarchy, completeness and classes describe relationships or resulting properties rather than the transformation method.

Why each choice is right or wrong

A - Correct: A reduction transforms instances while preserving membership, enabling hardness or completeness arguments with the stated resource bound.

B - Incorrect: A hierarchy organizes classes; it does not itself supply the instance transformation used in a hardness proof.

C - Incorrect: Completeness is the property proved using membership and hardness, not the reduction method itself.

D - Incorrect: Complexity classes group problems by resources; reductions establish the required relationships between individual problems.

Primary note proof

Study-note section AUT35

> $A \leq_p B$ means f is computable in polynomial time and $x \in A$ iff $f(x) \in B$. Solving B then solves A, so B is at least as hard under this reduction. If $B \in P$ then $A \in P$; if A is NP-hard then B is NP-hard. $A \in P$ does not force $B \in P$, and the reduction alone does not place both problems in the same class. Polynomial reductions compose.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Automata and Complexity Theory module | PDF page 15 | lines 33-35

> Algorithm analysis is an important part of a broader computational complexity theory, which provides theoretical estimates for the resources needed by any algorithm which solves a given computational problem.

Automata and Complexity Theory module | PDF page 17 | lines 29-30

> NP problems are the problems which can be solved in non-deterministic polynomial time.

Question sources

Computer Science MCQ practice collection · Question 777

Question 85 | AM0778

Which listed term denotes decision problems that belong to NP and are at least as hard as every NP problem under polynomial-time many-one reductions?

A. P

B. NP-complete

C. PSPACE

D. EXPTIME

Hint

Separate resource bounds, certificate verification and reduction requirements. Use proved containments without assuming unresolved class separations.

Answer B: NP-complete

NP-completeness combines two conditions: the target problem belongs to NP, and every NP language reduces to it in polynomial time. That gives a precise meaning to hardest within NP under the chosen reductions. P, PSPACE and EXPTIME instead name resource-bounded language classes and do not by themselves assert that each member is hard for NP or belongs to NP.

Why each choice is right or wrong

A - Incorrect: P names a resource class, not the membership-plus-hardness condition.

B - Correct: NP-complete states both NP membership and hardness for all NP languages.

C - Incorrect: PSPACE membership alone does not give the specified NP completeness.

D - Incorrect: EXPTIME membership alone does not give the specified NP completeness.

Primary note proof

Study-note section AUT36

> X is NP-hard if every A∈NP satisfies A≤pX. It need not belong to NP and can even be undecidable. X is NP-complete if X∈NP and X is NP-hard. NP-complete languages lie inside NP and can be decided by finite exhaustive certificate search in exponential time. They are not known to require exponential time.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Automata and Complexity Theory module | PDF page 17 | lines 27-28

> P problems are set of problems which can be solved in
> polynomial time by deterministic algorithms.

Automata and Complexity Theory module | PDF page 17 | lines 29-30

> NP problems are the problems which can be
> solved in non-deterministic polynomial time.

Automata and Complexity Theory module | PDF page 15 | lines 33-35

Algorithm analysis is an important part of a broader computational complexity theory, which provides theoretical estimates for the resources needed by any algorithm which solves a given computational problem.

Question sources

Computer Science MCQ practice collection · Question 778

Question 86 | CS0247

Let a Grammar G = ({S}, {a,b}, S, P), where P: S -> aSb, S -> lambda. Which one of the following is NOT an element of the language L generated from G?

A. aabb

B. ab

C. lambda (empty string)

D. a

Hint

Apply the stated transition, closure or equivalence rule step by step; distinguish a guaranteed bound from a value for one particular machine.

Answer D: a

After n applications of S→aSb and a final S→λ the word is a^n b^n. Induction preserves equal counts in block order, including n=0. a has one a and no b and cannot be generated.

Why each choice is right or wrong

A - Incorrect: S⇒aSb⇒aaSbb⇒aabb.

B - Incorrect: S⇒aSb⇒ab.

C - Incorrect: S⇒λ directly.

D - Correct: Unequal counts violate the invariant of every terminal derivation.

Primary note proof

Study-note section AUT17

> Use a recursive rule to enforce matched structure and a base rule to stop. S→aSb|ε generates exactly $a^n b^n$ for n≥0: n recursive steps add n a's on the left and n b's on the right. Derive aabb by S⇒aSb⇒aaSbb⇒aabb. For n≥1 replace ε by base ab.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Automata and Complexity Theory module | PDF page 8 | lines 27-28

> R is a set of production rules where each production rule maps a variable to a string s∈(V∪Σ)*;

Automata and Complexity Theory module | PDF page 2 | lines 24-25

> A string with zero occurrences of symbols is known as an empty string. It is represented by ε.

Question sources

Computer Science Exit Exam practice collection · Question 247

Question 87 | CS0249

Given input Alphabet sigma = {a, b, c, d} and empty string lambda. Which one of the following is true about sigma*?

A. sigma* = sigma+

B. sigma* = sigma+ union {lambda}

C. sigma* = sigma^1 union sigma^2 union sigma^3 union sigma+

D. sigma* = {lambda}

Hint

Distinguish symbols, finite strings, their lengths and sets of strings. Apply the operation to its stated operands, including empty cases.

Answer B: sigma* = sigma+ union {lambda}

Star allows every finite number of alphabet symbols, including zero. Positive closure allows all positive numbers. Their union with the singleton zero-length word gives exactly star. In C, unioning positive powers with Σ^+ still gives only Σ^+, missing λ.

Why each choice is right or wrong

A - Incorrect: Σ^+ omits λ, so it is smaller than Σ^*.

B - Correct: Partitions all words by zero versus positive length.

C - Incorrect: All terms already have positive length; λ is absent.

D - Incorrect: Omits every nonempty word such as a.

Primary note proof

Study-note section AUT03

> Σ^n contains words of exactly n symbols; for $|\Sigma|=k$, $|\Sigma^n|=k^n$, including $\Sigma^0=\{\varepsilon\}$. $\Sigma^*=\bigcup n\geq 0\Sigma^n$ and $\Sigma^+=\bigcup n\geq 1\Sigma^n$. Over a nonempty alphabet, Σ^+ excludes ε and $\Sigma^*=\Sigma^+\cup\{\varepsilon\}$. Σ^2 over {a,b} is {aa,ab,ba,bb}.

Study-note section AUT03

> For an arbitrary language L, $L^*=\bigcup n\geq 0 L^n$ and $L^+=LL^*$. Unlike alphabet positive closure, L^+ can contain ε when $\varepsilon\in L$. $\varnothing^*=\{\varepsilon\}$, $\varnothing^+=\varnothing$, and $\{\varepsilon\}^*=\{\varepsilon\}$. Union means membership in either set; intersection requires the same word in both, not concatenation. Complement is Σ^*-L and needs a fixed alphabet.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Automata and Complexity Theory module | PDF page 5 | lines 22-22

> In a regular expression, x* means zero or more occurrence of x.

Automata and Complexity Theory module | PDF page 5 | lines 25-25

means one or more occurrence of x. It can generate {x, xx, xxx, xxxx,

Automata and Complexity Theory module | PDF page 2 | lines 24-25

A string with zero occurrences of symbols is known as an empty string. It is represented by ε.

Question sources

Computer Science Exit Exam practice collection · Question 249

Question 88 | CS0250

Given input Alphabet sigma = {a, b, c, d} and empty string lambda. Which one of the following is equivalent to sigma+?

A. sigma

B. sigma+

C. {lambda}

D. sigma*

Hint

Distinguish symbols, finite strings, their lengths and sets of strings. Apply the operation to its stated operands, including empty cases.

Answer B: sigma+

Taken literally, B is an identity: Σ+ equals itself. Since the alphabet is nonempty and contains no empty symbol, Σ contains only length-one words, Σ* additionally contains λ, and {λ} contains no positive-length word. The identical expressions make B a definite answer as printed.

Why each choice is right or wrong

A - Incorrect: aa is in Σ+ but not in Σ.

B - Correct: Identical expression on both sides.

C - Incorrect: λ has length zero and is not a positive repetition.

D - Incorrect: Σ* includes λ, unlike Σ+.

Primary note proof

Study-note section AUT03

Σ^n contains words of exactly n symbols; for $|\Sigma|=k$, $|\Sigma^n|=k^n$, including $\Sigma^0=\{\varepsilon\}$. $\Sigma^*=\bigcup n\geq 0\Sigma^n$ and $\Sigma^+=\bigcup n\geq 1\Sigma^n$. Over a nonempty alphabet, Σ^+ excludes ε and $\Sigma^*=\Sigma^+\cup\{\varepsilon\}$. Σ^2 over {a,b} is {aa,ab,ba,bb}.

Study-note section AUT03

For an arbitrary language L, $L^*=\bigcup n\geq 0L^n$ and $L^+=LL^*$. Unlike alphabet positive closure, L^+ can contain ε when $\varepsilon\in L$. $\varnothing^*=\{\varepsilon\}$, $\varnothing^+=\varnothing$, and $\{\varepsilon\}^*=\{\varepsilon\}$. Union means membership in either set;

> intersection requires the same word in both, not concatenation. Complement is Σ*−L and needs a fixed alphabet.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Automata and Complexity Theory module | PDF page 5 | lines 22-22

> In a regular expression, x* means zero or more occurrence of x.

Automata and Complexity Theory module | PDF page 5 | lines 25-25

> means one or more occurrence of x. It can generate {x, xx, xxx, xxxx,

Automata and Complexity Theory module | PDF page 2 | lines 24-25

> A string with zero occurrences of symbols is known as an empty string. It is represented by ε.

Question sources

Computer Science Exit Exam practice collection · Question 250

Question 89 | CS0251

If Z=caddba, what is $|Z|^2$, the square of the numeric length of Z?

A. 3

B. 4

C. 2

D. 36

Hint

Distinguish symbols, finite strings, their lengths and sets of strings. Apply the operation to its stated operands, including empty cases.

Answer D: 36

String length counts every symbol occurrence, including repetitions. The word caddba has six positions, so $|Z|=6$ and $|Z|^2=6^2=36$. Counting distinct symbols would give four, which is not string length. Concatenating two copies gives length twelve, but that computes $|Z^2|$ instead; the revised stem expressly preserves the original placement of the exponent outside the bars.

Why each choice is right or wrong

A - Incorrect: Three is neither the length nor its square.

B - Incorrect: Four counts distinct letters rather than the squared length.

C - Incorrect: Two is the exponent, not the squared length.

D - Correct: Six positions squared gives 36.

Primary note proof

Study-note section AUT01

> An alphabet Σ is a finite set of symbols. A word/string is a finite ordered sequence of symbols from Σ; a language is a set of words, L⊆Σ*. An empty word ε, also written λ, has length zero. The empty language ∅ contains no words; {ε} contains one.

Study-note section AUT01

> Length counts occurrences, not distinct symbols: |caddba|=6, so $|caddba|^2$=36.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Automata and Complexity Theory module | PDF page 2 | lines 26-26

> The number of symbols in a string w is called the length of a string. It is denoted by |w|.

Automata and Complexity Theory module | PDF page 5 | lines 22-22

> In a regular expression, x* means zero or more occurrence of x.

Question sources

Computer Science Exit Exam practice collection · Question 251

Question 90 | CS0252

What will be the value of Z^R (reverse) if string Z = caddba?

A. abddac

B. cad

C. abd

D. caddba

Hint

Distinguish symbols, finite strings, their lengths and sets of strings. Apply the operation to its stated operands, including empty cases.

Answer A: abddac

Reversal reads the characters in the opposite order without dropping any: c a d d b a becomes a b d d a c.

Why each choice is right or wrong

A - Correct: Produces every symbol in reverse order: abddac.

B - Incorrect: Only a three-symbol prefix; does not reverse the full word.

C - Incorrect: Only the first three reversed symbols; drops the remaining characters.

D - Incorrect: Original word differs from its reverse at the first symbol.

Primary note proof

Study-note section AUT02

> Concatenation xy writes x followed by y: cddc·ababb=cddcababb; order generally matters. εx=xε=x. Define $x^0=\varepsilon$ and x^n as n copies for n≥1. Reverse x^R reads symbols backward: caddba reverses to abddac; $(xy)^R=y^Rx^R$ and $(x^R)^R=x$.

Study-note section AUT02

> For languages, LM={xy:x∈L,y∈M}, $L^0=\{\varepsilon\}$, and $L^{n+1}=L^nL$. Thus {e,fgh}{c,d}={ec,ed,fghc,fghd}; fghe is absent.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Automata and Complexity Theory module | PDF page 2 | lines 21-21

> It is a finite collection of symbols from the alphabet. The string is denoted by w.

Automata and Complexity Theory module | PDF page 2 | lines 26-26

> The number of symbols in a string w is called the length of a string. It is denoted by |w|.

Question sources

Computer Science Exit Exam practice collection · Question 252

Question 91 | CS0253

If we have two strings Z = cddc and W = ababb, then the concatenation ZW will be?

A. ZW = cddcababb

B. ZW = cdab

C. ZW = ababbcddc

D. ZW = ca

Hint

Distinguish symbols, finite strings, their lengths and sets of strings. Apply the operation to its stated operands, including empty cases.

Answer A: ZW = cddcababb.

AUT02 defines xy as all of x followed by all of y, preserving order and every symbol. Appending ababb to cddc yields cddcababb, of length 4+5=9. Reversing the operands produces a different string; selecting prefixes or first letters discards symbols and is not concatenation.

Why each choice is right or wrong

A - Correct: This preserves cddc followed by ababb, giving the required nine-symbol concatenation.

B - Incorrect: cdab discards symbols from both operands instead of retaining both complete strings.

C - Incorrect: ababbcddc is WZ, reversing the specified operand order.

D - Incorrect: ca retains only initial symbols and is not the concatenation of the full strings.

Primary note proof

Study-note section AUT02

> Concatenation xy writes x followed by y: cddc·ababb=cddcababb; order generally matters. εx=xε=x. Define $x^0=\varepsilon$ and x^n as n copies for n≥1. Reverse x^R reads symbols backward: caddba reverses to abddac; $(xy)^R=y^Rx^R$ and $(x^R)^R=x$.

Study-note section AUT02

> For languages, LM={xy:x∈L,y∈M}, $L^0=\{\varepsilon\}$, and $L^{n+1}=L^nL$. Thus {e,fgh}{c,d}={ec,ed,fghc,fghd}; fghe is absent.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Automata and Complexity Theory module | PDF page 2 | lines 21-21

> It is a finite collection of symbols from the alphabet. The string is denoted by w.

Automata and Complexity Theory module | PDF page 2 | lines 26-26

> The number of symbols in a string w is called the length of a string. It is denoted by |w|.

Question sources

Computer Science Exit Exam practice collection · Question 253

Question 92 | CS0254

Given input alphabet sigma = {a, b, c, d} and empty string lambda. Which one of the following is FALSE about lambda?

A. lambda*ab = ab

B. For any string Z, Z^0 = lambda

C. lambda is an element of sigma*

D. lambda = 0

Hint

Distinguish symbols, finite strings, their lengths and sets of strings. Apply the operation to its stated operands, including empty cases.

Answer D: The empty word is not numeric zero.

AUT01 distinguishes the empty word λ from the numeric symbol 0. AUT02 gives λx=x and Z^0=λ, while AUT03 includes the empty word in Σ*. Thus the first three claims express the standard empty-word identities under the displayed concatenation notation, and D is the false equality.

Why each choice is right or wrong

A - Incorrect: Reading the displayed star as concatenation, the empty word followed by ab is ab.

B - Incorrect: Every string's zeroth concatenation power is the empty word.

C - Incorrect: Kleene closure of the alphabet includes its zero-length word λ.

D - Correct: This equality is false: λ is a zero-length word, not the numeric symbol 0.

Primary note proof

Study-note section AUT01

> An alphabet Σ is a finite set of symbols. A word/string is a finite ordered sequence of symbols from Σ; a language is a set of words, L⊆Σ*. An empty word ε, also written λ, has length zero. The empty language ∅ contains no words; {ε} contains one.

Study-note section AUT01

> Length counts occurrences, not distinct symbols: |caddba|=6, so $|caddba|^2$=36.

Study-note section AUT02

> Concatenation xy writes x followed by y: cddc·ababb=cddcababb; order generally matters. εx=xε=x. Define x^0=ε and x^n as n copies for n≥1. Reverse x^R reads symbols backward: caddba reverses to abddac; $(xy)^R$=y^Rx^R and $(x^R)^R$=x.

Study-note section AUT02

> For languages, LM={xy:x∈L,y∈M}, L^0={ε}, and L^{n+1}=L^nL. Thus {e,fgh}{c,d}={ec,ed,fghc,fghd}; fghe is absent.

Study-note section AUT03

> $Σ^n$ contains words of exactly n symbols; for |Σ|=k, $|Σ^n|$=k^n, including $Σ^0$={ε}. Σ*=∪n≥0$Σ^n$ and $Σ^+$=∪n≥1$Σ^n$. Over a nonempty alphabet, $Σ^+$ excludes ε and Σ*=$Σ^+$∪{ε}. $Σ^2$ over {a,b} is {aa,ab,ba,bb}.

Study-note section AUT03

> For an arbitrary language L, L*=∪n≥0L^n and L^+=LL*. Unlike alphabet positive closure, L^+ can contain ε when ε∈L. ∅*={ε}, $∅^+$=∅, and {ε}*={ε}. Union means membership in either set; intersection requires the same word in both, not concatenation. Complement is Σ*−L and needs a fixed alphabet.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Automata and Complexity Theory module | PDF page 2 | lines 24-25

A string with zero occurrences of symbols is known as an empty string. It is represented by ε.

Automata and Complexity Theory module | PDF page 5 | lines 22-22

In a regular expression, x* means zero or more occurrence of x.

Question sources

Computer Science Exit Exam practice collection · Question 254

Question 93 | CS0255

Which one of the following is FALSE about any language L?

A. L^3 = L^1 * L^1 * L^1

B. L^0 = lambda

C. L* = L^0 union L^1 union L^2 union L^3 ...

D. L+ = L^1 union L^2 union L^3 ...

Hint

Distinguish symbols, finite strings, their lengths and sets of strings. Apply the operation to its stated operands, including empty cases.

Answer B: L^0 is {λ}, not the word λ itself.

AUT02 defines $L^0=\{\varepsilon\}$; AUT01 equates λ with ε. Thus the literal equality $L^0=\lambda$ confuses a set with its sole word and is false unless braces are explicitly understood. Reading the binary stars in A as the displayed concatenation separators, A is the third-power definition; C and D are the unions defining star and positive closure in AUT03.

Why each choice is right or wrong

A - Incorrect: With the displayed binary stars denoting concatenation, three copies of L form L^3, so this is true.

B - Correct: This is the false statement: the zeroth power is the singleton language {λ}, not the word λ.

C - Incorrect: The union of all nonnegative language powers is the definition of Kleene closure, so this is true.

D - Incorrect: The union of positive language powers is positive closure, even if L itself contains ε.

Primary note proof

Study-note section AUT03

Σ^n contains words of exactly n symbols; for $|\Sigma|=k$, $|\Sigma^n|=k^n$, including $\Sigma^0=\{\varepsilon\}$. $\Sigma^*=\bigcup n\geq 0\Sigma^n$ and $\Sigma^+=\bigcup n\geq 1\Sigma^n$. Over a nonempty alphabet, Σ^+ excludes ε and $\Sigma^*=\Sigma^+\cup\{\varepsilon\}$. Σ^2 over {a,b} is {aa,ab,ba,bb}.

Study-note section AUT03

For an arbitrary language L, $L^*=\bigcup_{n\geq 0}L^n$ and $L^+=LL^*$. Unlike alphabet positive closure, L^+ can contain ε when $\varepsilon\in L$. $\varnothing^*=\{\varepsilon\}$, $\varnothing^+=\varnothing$, and $\{\varepsilon\}^*=\{\varepsilon\}$. Union means membership in either set; intersection requires the same word in both, not concatenation. Complement is Σ^*-L and needs a fixed alphabet.

Study-note section AUT02

Concatenation xy writes x followed by y: cddc·ababb=cddcababb; order generally matters. εx=xε=x. Define $x^0=\varepsilon$ and x^n as n copies for n≥1. Reverse x^R reads symbols backward: caddba reverses to abddac; $(xy)^R=y^Rx^R$ and $(x^R)^R=x$.

Study-note section AUT02

For languages, LM={xy:x∈L,y∈M}, $L^0=\{\varepsilon\}$, and $L^{n+1}=L^nL$. Thus {e,fgh}{c,d}={ec,ed,fghc,fghd}; fghe is absent.

Study-note section AUT01

An alphabet Σ is a finite set of symbols. A word/string is a finite ordered sequence of symbols from Σ; a language is a set of words, $L\subseteq\Sigma^*$. An empty word ε, also written λ, has length zero. The empty language ∅ contains no words; {ε} contains one.

Study-note section AUT01

Length counts occurrences, not distinct symbols: $|caddba|=6$, so $|caddba|^2=36$.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Automata and Complexity Theory module | PDF page 6 | lines 4-4

Kleen closure: If L is a regular language then its Kleen closure L1* will also be a regular language.

Automata and Complexity Theory module | PDF page 2 | lines 24-25

A string with zero occurrences of symbols is known as an empty string. It is represented by ε.

Question sources

Computer Science Exit Exam practice collection · Question 255

Question 94 | CS0256

Given input alphabet sigma = {a, b, c, d}, which one of the following is an element of set sigma^2?

A. b

B. c

C. ac

D. aabbccdd

Hint

Distinguish symbols, finite strings, their lengths and sets of strings. Apply the operation to its stated operands, including empty cases.

Answer C: ac.

AUT03 defines Σ^n as the words containing exactly n alphabet symbols. The given alphabet includes a and c, so ac has two permitted symbols and belongs to Σ^2. Single-symbol choices have length one; aabbccdd has length eight. Membership depends on length and symbol validity, not on including all alphabet symbols.

Why each choice is right or wrong

A - Incorrect: b uses an allowed symbol but has length one rather than the required two.

B - Incorrect: c is a one-symbol word and therefore belongs to Σ^1, not Σ^2.

C - Correct: ac has exactly two symbols, both drawn from the given alphabet.

D - Incorrect: aabbccdd has eight symbols; using each symbol twice does not make it a length-two word.

Primary note proof

Study-note section AUT03

> Σ^n contains words of exactly n symbols; for $|\Sigma|=k$, $|\Sigma^n|=k^n$, including $\Sigma^0=\{\varepsilon\}$. $\Sigma^*=\bigcup n\geq 0\Sigma^n$ and $\Sigma^+=\bigcup n\geq 1\Sigma^n$. Over a nonempty alphabet, Σ^+ excludes ε and $\Sigma^*=\Sigma^+\cup\{\varepsilon\}$. Σ^2 over {a,b} is {aa,ab,ba,bb}.

Study-note section AUT03

> For an arbitrary language L, $L^*=\bigcup n\geq 0L^n$ and $L^+=LL^*$. Unlike alphabet positive closure, L^+ can contain ε when $\varepsilon\in L$. $\varnothing^*=\{\varepsilon\}$, $\varnothing^+=\varnothing$, and $\{\varepsilon\}^*=\{\varepsilon\}$. Union means membership in either set; intersection requires the same word in both, not concatenation. Complement is Σ^*-L and needs a fixed alphabet.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Automata and Complexity Theory module | PDF page 2 | lines 15-15

> An alphabet finite set of non-empty symbols. It is denoted by ∑ (sigma).

Automata and Complexity Theory module | PDF page 2 | lines 26-26

> The number of symbols in a string w is called the length of a string. It is denoted by |w|.

Automata and Complexity Theory module | PDF page 2 | lines 21-21

> It is a finite collection of symbols from the alphabet. The string is denoted by w.

Question sources

Computer Science Exit Exam practice collection · Question 256

Question 95 | CS0260

In a Turing Machine, the transition a -> b, L means:

A. Read a, write b and move left

B. Delete a, read b and move left

C. Write a, read b and move left

D. Replace b by a and move left

Hint

Compare production restrictions, memory access and language-family containments. Distinguish an exact characterization from an included subclass.

Answer A: Read a, write b and move left

The TM transition reads the scanned a, replaces it with b, and moves the head one cell left. This is the transition ordering (read symbol, write symbol, movement), with next state specified by the destination vertex.

Why each choice is right or wrong

A - Correct: Matches the scanned symbol, replacement, and L direction.

B - Incorrect: b is written rather than read; no distinct delete-then-read instruction is specified.

C - Incorrect: Swaps read and write roles.

D - Incorrect: Replaces the wrong symbol in the wrong direction.

Primary note proof

Study-note section AUT27

> δ(q,a)=(p,b,L) means read a in q, write b, move one cell left and enter p. Accept/reject halting states have no further moves under this convention.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Automata and Complexity Theory module | PDF page 11 | lines 20-22

> δ is the transition function, mapping (q, x) onto (q', y, D) where D denotes the direction of movement of R/W head: D = L or R according as the movement is to the left or right.

Automata and Complexity Theory module | PDF page 11 | lines 9-9

> It has a tape head that can read and write symbols and move around on the tape.

Automata and Complexity Theory module | PDF page 14 | lines 8-8

> As a result the symbol is β written in the cell under the R/W head.

Question sources

Computer Science Exit Exam practice collection · Question 260

Question 96 | NS0161

What is the difference between a DFA and an NFA? Are they equally powerful?

Hint

Apply the stated transition, closure or equivalence rule step by step; distinguish a guaranteed bound from a value for one particular machine.

Answer: DFA has one successor per state/symbol; NFA permits sets. They recognize the same languages.

AUT09 permits an NFA to have zero, one or several next states and states that DFA and NFA recognize exactly REG. Subset construction simulates the simultaneous NFA possibilities by DFA states.

Primary note proof

Study-note section AUT06

> A deterministic finite automaton (DFA) is $M=(Q,\Sigma,\delta,q_0,F)$, with finite nonempty Q, start $q_0 \in Q$, accepting set $F \subseteq Q$ and total $\delta: Q \times \Sigma \to Q$. Exactly one next state is specified per state/symbol pair; this does not mean the whole machine has one transition.

Study-note section AUT09

> A nondeterministic finite automaton (NFA) permits zero, one or many successors: $\delta: Q \times \Sigma \to P(Q)$.

Study-note section AUT09

> An NFA is not more expressive than a DFA: both recognize exactly REG. Nondeterminism can make a description smaller.

Study-note section AUT10

> ε-closure(T) contains T and every state reachable through ε moves. Start the DFA at ε-closure($\{q_0\}$); for each current subset T and symbol a compute ε-closure($\bigcup_{q \in T} \delta(q,a)$). A subset accepts iff it intersects F.

Question sources

Automata & Complexity Theory review questions · Question 1

Question 97 | NS0162

What is the pumping lemma used for? Demonstrate with an example.

Hint

Respect the quantifier order and length constraints. A valid nonmembership argument must address every allowed decomposition, not one convenient split.

Answer: The regular pumping lemma can disprove regularity, for example for $a^n b^n$.

AUT15 chooses $a^p b^p$ after assuming a pumping length p. Every legal nonempty y in the first p symbols contains only a, so pumping it out changes the a count without changing b. That contradicts regularity.

Primary note proof

Study-note section AUT15

> If L is regular, some $p \geq 1$ satisfies: every $s \in L$ with $|s| \geq p$ has a split $s=xyz$, $|xy| \leq p$, $|y| \geq 1$, and $xy^i z \in L$ for every $i \geq 0$.

Study-note section AUT15

> For $L=\{a^n b^n : n \geq 0\}$, choose $a^p b^p$. Every allowed $y=a^k$ with $k \geq 1$; pumping $i=0$ leaves fewer a's than b's. The contradiction excludes every DFA.

Question sources

Automata & Complexity Theory review questions · Question 2

Question 98 | NS0163

What is the difference between P and NP? Why does it matter?

Hint

Separate resource bounds, certificate verification and reduction requirements. Use proved containments without assuming unresolved class separations.

Answer: P is deterministic polynomial-time decision; NP admits polynomial certificates. Equality remains open.

AUT34 defines P by deterministic polynomial running time and NP by nondeterministic polynomial time or polynomial-size verifiable certificates. P is included in NP, and equality is unresolved. This matters for converting efficiently checked witnesses into efficiently found decisions.

Primary note proof

Study-note section AUT34

> Complexity classes here contain decision languages. P consists of deterministic polynomial-time decidable languages; NP consists of nondeterministic polynomial-time decidable languages, equivalently polynomially verifiable yes-instances with polynomial-size certificates. A verifier must accept some certificate exactly for yes-inputs; no certificate may falsely establish a no-instance. NP means nondeterministic polynomial, not non-polynomial.

Study-note section AUT34

> P⊆NP because a verifier can ignore its certificate and run the P algorithm. P=NP remains open; NP does not mean definitely hard or definitely outside P. Finding versus verifying is an intuition for witnesses, not a rule that all NP instances need exponential search.

Study-note section AUT36

> X is NP-hard if every A∈NP satisfies A≤pX. It need not belong to NP and can even be undecidable. X is NP-complete if X∈NP and X is NP-hard.

Question sources

Automata & Complexity Theory review questions · Question 3

Question 99 | NS0164

Why is the halting problem undecidable? Explain the contradiction.

Hint

Check whether the required procedure must halt on every input, and distinguish solving a task from recognizing its yes-instances.

Answer: A proposed halting decider yields a program contradicting its own prediction.

AUT30 constructs D(x) to loop when H(x,x) predicts halting and to halt otherwise. On D's own description, either prediction is contradicted. Therefore no general halting decider exists, although simulation recognizes halting instances.

Primary note proof

Study-note section AUT30

> HALT={⟨M,w⟩:M halts on w} is recognizable: simulate and accept upon any halt. Suppose a decider H determines halting. Construct D(x) that loops if H(x,x) says halt, otherwise halts. Running D on its own description contradicts H in both cases. Thus HALT is undecidable and witnesses RE beyond DECIDABLE. Its complement is not RE, or both recognizers would yield a decider.

Question sources

Automata & Complexity Theory review questions · Question 4

Question 100 | NS0165

What is the Chomsky hierarchy and why are the containments proper?

Hint

Check whether the required procedure must halt on every input, and distinguish solving a task from recognizing its yes-instances.

Answer: REG⊊CFL⊊CSL⊊RE by increasing generative power; separating examples witness proper inclusions.

AUT05 supplies the proper family hierarchy. Equal pairs separate CFL from REG; equal triples separate CSL from CFL. CSLs are decidable by AUT25, while HALT is RE but undecidable under AUT30, separating CSL from RE.

Primary note proof

Study-note section AUT05

> Type 3 regular grammars correspond to finite automata; type 2 CFGs to nondeterministic PDAs; type 1 context-sensitive/noncontracting grammars to nondeterministic LBAs; type 0 unrestricted grammars to Turing recognizers.

Study-note section AUT05

> Use REG ⊊ CFL ⊊ CSL ⊊ DECIDABLE ⊊ RE, under the usual convention allowing S→ε with S absent from right sides when needed.

Study-note section AUT26

> a*b* and $\{a^n b : n \geq 0\}$ are regular. $a^n b^n$ is CFL but not regular. Over a two-symbol alphabet, ww^R is CFL, ww is not CFL; both are decidable. $a^n b^n c^n$ is CSL but not CFL, and therefore decidable. Every finite language is regular. A stronger grammar or TM can still recognize any of these lower-family languages.

Study-note section AUT30

> Thus HALT is undecidable and witnesses RE beyond DECIDABLE.

Question sources

Automata & Complexity Theory review questions · Question 5

Question 101 | NS0166

Which of the following correctly orders automata from LEAST to MOST powerful?

A. NFA < DFA < PDA < TM

B. PDA < NFA < DFA < TM

C. DFA = NFA < PDA < LBA < TM

D. TM < LBA < PDA < DFA

Hint

Compare production restrictions, memory access and language-family containments. Distinguish an exact characterization from an included subclass.

Answer C: DFA=NFA < PDA < LBA < TM in language-recognition power.

AUT09 equates DFA and NFA power. AUT05 assigns finite automata, nondeterministic PDAs, nondeterministic LBAs and Turing recognizers to successively stronger language families. That gives C under the standard nondeterministic PDA/LBA family convention. The comparison is expressiveness, not a statement that stronger models always run faster.

Why each choice is right or wrong

A - Incorrect: NFA and DFA have equal power rather than the strict order shown.

B - Incorrect: PDAs recognize more than regular languages, so placing PDA below finite automata reverses the hierarchy.

C - Correct: This gives equal finite-automaton power followed by the standard strictly increasing PDA/LBA/TM families.

D - Incorrect: This reverses the standard language-recognition power hierarchy.

Primary note proof

Study-note section AUT05

> Type 3 regular grammars correspond to finite automata; type 2 CFGs to nondeterministic PDAs; type 1 context-sensitive/noncontracting grammars to nondeterministic LBAs; type 0 unrestricted grammars to Turing recognizers.

Study-note section AUT05

> Use REG ⊊ CFL ⊊ CSL ⊊ DECIDABLE ⊊ RE, under the usual convention allowing S→ε with S absent from right sides when needed.

Question sources

Automata & Complexity Theory review questions · Question 1

Question 102 | NS0167

Complexity theory primarily studies:

A. How to design finite automata

B. Resources (time and space) required to solve computational problems

C. How to convert NFA to DFA

D. The syntax of formal grammars

Hint

Check whether the required procedure must halt on every input, and distinguish solving a task from recognizing its yes-instances.

Answer B: Time and space resources for computational problems.

AUT33 defines time as computation steps and space as memory cells, and AUT34 groups decision problems by polynomial resource bounds. Complexity studies these resource requirements as a function of encoded input size. Designing automata or transforming grammars may be objects of study, but neither is the general defining purpose.

Why each choice is right or wrong

A - Incorrect: Finite-automaton design is one specific task rather than the general study of computational resources.

B - Correct: Complexity theory classifies time and space requirements relative to input size.

C - Incorrect: NFA determinization is a particular construction, not the broad subject of complexity theory.

D - Incorrect: Grammar syntax concerns formal language description rather than resource requirements in general.

Primary note proof

Study-note section AUT-S01

> Computability theory asks whether an effective algorithm can solve a problem at all. For a decision problem, a decision procedure must terminate with the correct yes-or-no result on every input.

Study-note section AUT-S01

> Computational complexity theory studies the time and memory resources required to solve problems as functions of encoded input size. Automata theory studies abstract machine models and the languages they recognize; these subjects overlap but ask different questions.

Question sources

Automata & Complexity Theory review questions · Question 2

Question 103 | NS0168

Given $\Sigma = \{a, b\}$, which of the following is in Σ^2 but NOT in Σ^1?

A. a

B. b

C. ε

D. ab

Hint

Distinguish symbols, finite strings, their lengths and sets of strings. Apply the operation to its stated operands, including empty cases.

Answer D: ab.

AUT03 defines Σ^n by exact word length. Over {a,b}, ab contains two permitted symbols and is not length one. The single letters a and b belong to Σ^1; ε has length zero. Thus only ab satisfies both membership in Σ^2 and exclusion from Σ^1.

Why each choice is right or wrong

A - Incorrect: a has length one and belongs to Σ^1 rather than Σ^2.

B - Incorrect: b has length one, violating the required exact length two.

C - Incorrect: Epsilon has length zero and is not in Σ^2.

D - Correct: ab has two permitted symbols and is not a one-symbol word.

Primary note proof

Study-note section AUT03

> Σ^n contains words of exactly n symbols; for $|\Sigma|=k$, $|\Sigma^n|=k^n$, including $\Sigma^0=\{\varepsilon\}$. $\Sigma^*=\bigcup n\geq 0\Sigma^n$ and $\Sigma^+=\bigcup n\geq 1\Sigma^n$. Over a nonempty alphabet, Σ^+ excludes ε and $\Sigma^*=\Sigma^+\cup\{\varepsilon\}$. Σ^2 over {a,b} is {aa,ab,ba,bb}.

Study-note section AUT03

> For an arbitrary language L, $L^*=\bigcup n\geq 0 L^n$ and $L^+=LL^*$. Unlike alphabet positive closure, L^+ can contain ε when $\varepsilon\in L$. $\varnothing^*=\{\varepsilon\}$, $\varnothing^+=\varnothing$, and $\{\varepsilon\}^*=\{\varepsilon\}$. Union means membership in either set; intersection requires the same word in both, not concatenation. Complement is Σ^*-L and needs a fixed alphabet.

Question sources

Automata & Complexity Theory review questions · Question 3

Question 104 | NS0169

The set Σ^* for $\Sigma = \{0, 1\}$ includes which of the following?

A. ε (empty string)

B. All binary strings of length ≥ 1 only

C. Only finite-length strings starting with 1

D. All strings NOT over {0,1}

Hint

Distinguish symbols, finite strings, their lengths and sets of strings. Apply the operation to its stated operands, including empty cases.

Answer A: The empty string is included.

AUT03 defines Σ^* as the union of all nonnegative powers, including $\Sigma^0=\{\varepsilon\}$. Therefore the empty word belongs to the binary closure. Positive-length-only and starting-with-one descriptions exclude valid members, and strings using symbols outside {0,1} are outside this universe. The wording of B must not erase its restrictive only.

Why each choice is right or wrong

A - Correct: The zero power contributes epsilon to the Kleene closure.

B - Incorrect: The word only excludes epsilon, so this is not a complete description of Σ^*.

C - Incorrect: Binary strings starting with 0 and epsilon also belong, so the starting-with-1 restriction is false.

D - Incorrect: Σ^* consists of words over the specified binary alphabet, not words outside it.

Primary note proof

Study-note section AUT03

> Σ^n contains words of exactly n symbols; for $|\Sigma|=k$, $|\Sigma^n|=k^n$, including $\Sigma^0=\{\varepsilon\}$. $\Sigma^*=\bigcup n\geq 0\Sigma^n$ and $\Sigma^+=\bigcup n\geq 1\Sigma^n$. Over a nonempty alphabet, Σ^+ excludes ε and $\Sigma^*=\Sigma^+\cup\{\varepsilon\}$. Σ^2 over {a,b} is {aa,ab,ba,bb}.

Study-note section AUT03

> For an arbitrary language L, $L^*=\bigcup n\geq 0 L^n$ and $L^+=LL^*$. Unlike alphabet positive closure, L^+ can contain ε when $\varepsilon\in L$. $\varnothing^*=\{\varepsilon\}$, $\varnothing^+=\varnothing$, and $\{\varepsilon\}^*=\{\varepsilon\}$. Union means membership in either set; intersection requires the same word in both, not concatenation. Complement is Σ^*-L and needs a fixed alphabet.

Question sources

Automata & Complexity Theory review questions · Question 4

Question 105 | NS0170

What is the difference between Σ^* and Σ^+?

A. Σ^* excludes strings of length 2 or more

B. Σ^+ includes the empty string but Σ^* does not

C. Σ^* includes the empty string (ε) but Σ^+ does not

D. They are identical

Hint

Distinguish symbols, finite strings, their lengths and sets of strings. Apply the operation to its stated operands, including empty cases.

Answer C: Σ^* includes ε; Σ^+ excludes it.

AUT03 explicitly gives $\Sigma^*=\Sigma^+\cup\{\varepsilon\}$ for a nonempty alphabet. Both operations allow arbitrarily long finite words, but only star includes length zero. This statement concerns alphabet closure; the note separately cautions that positive closure of an arbitrary language containing epsilon can itself include epsilon.

Why each choice is right or wrong

A - Incorrect: Σ^* contains words of every finite length, including length two and above.

B - Incorrect: This reverses which alphabet closure contains the empty word.

C - Correct: Star includes the zero-length word and positive alphabet closure omits it.

D - Incorrect: They differ at least by epsilon for the nonempty alphabet convention.

Primary note proof

Study-note section AUT03

> Σ^n contains words of exactly n symbols; for $|\Sigma|=k$, $|\Sigma^n|=k^n$, including $\Sigma^0=\{\varepsilon\}$. $\Sigma^*=\bigcup n\geq 0\Sigma^n$ and $\Sigma^+=\bigcup n\geq 1\Sigma^n$. Over a nonempty alphabet, Σ^+ excludes ε and $\Sigma^*=\Sigma^+\cup\{\varepsilon\}$. Σ^2 over {a,b} is {aa,ab,ba,bb}.

Study-note section AUT03

> For an arbitrary language L, $L^*=\bigcup n\geq 0L^n$ and $L^+=LL^*$. Unlike alphabet positive closure, L^+ can contain ε when $\varepsilon\in L$. $\varnothing^*=\{\varepsilon\}$, $\varnothing^+=\varnothing$, and $\{\varepsilon\}^*=\{\varepsilon\}$. Union means membership in either set; intersection requires the same word in both, not concatenation. Complement is Σ^*-L and needs a fixed alphabet.

Question sources

Automata & Complexity Theory review questions · Question 5

Question 106 | NS0171

In the five-tuple $M = (Q, \Sigma, \delta, q_0, F)$ of an FSM, which component determines how the machine moves from state to state?

A. Q - the set of states

B. δ - the transition function

C. F - the set of accepting states

D. q_0 - the initial state

Hint

Read the transition domain and acceptance condition carefully. Track which inputs are consumed and evaluate the whole word under the stated model.

Answer B: δ, the transition function.

AUT06 defines $\delta:Q\times\Sigma\to Q$, mapping a current state and input symbol to the next state. That is precisely the movement rule asked for. Q lists available states, F marks acceptance and q_0 fixes the starting state; none of those alone specifies the transitions between states.

Why each choice is right or wrong

A - Incorrect: Q supplies state names but not the next-state rule for input symbols.

B - Correct: Delta maps the current state and input symbol to the next state.

C - Incorrect: F identifies acceptance rather than transitions between states.

D - Incorrect: The initial state specifies where processing starts, not how each symbol changes state.

Primary note proof

Study-note section AUT06

> A deterministic finite automaton (DFA) is $M=(Q,\Sigma,\delta,q_0,F)$, with finite nonempty Q, start $q_0\in Q$, accepting set $F\subseteq Q$ and total $\delta:Q\times\Sigma\to Q$. Exactly one next state is specified per state/symbol pair; this does not mean the whole machine has one transition. F may be empty, recognizing $\varnothing$.

Study-note section AUT06

> Start at q_0 and consume the entire input. Accept iff the ending state is in F; visiting an accepting state earlier is insufficient. ε is accepted exactly when $q_0\in F$. A nonaccepting sink loops on every symbol and cannot reach acceptance; it still finishes after finite input. Finite automata have finite memory encoded by states, not no state or no ability to remember any history.

Question sources

Automata & Complexity Theory review questions · Question 6

Question 107 | NS0172

What is the defining characteristic that makes an automaton 'deterministic'?

A. It has exactly one accepting state

B. It processes input from right to left

C. It never enters a dead state

D. For every (state, input) pair, there is exactly ONE next state

Hint

Read the transition domain and acceptance condition carefully. Track which inputs are consumed and evaluate the whole word under the stated model.

Answer D: One next state for every state/input-symbol pair.

AUT06 explicitly defines exactly one successor per state/symbol pair. Determinism does not constrain the number of final states, input-reading direction or whether a rejecting sink is reachable. The uniqueness concerns each transition choice, not the number of states or transitions in the entire automaton.

Why each choice is right or wrong

A - Incorrect: A DFA may have zero, one or several accepting states.

B - Incorrect: Right-to-left processing is not the definition of determinism.

C - Incorrect: Deterministic automata may legitimately have reachable rejecting sinks.

D - Correct: The next state is uniquely determined by the current state and current input symbol.

Primary note proof

Study-note section AUT06

> A deterministic finite automaton (DFA) is $M=(Q,\Sigma,\delta,q_0,F)$, with finite nonempty Q, start $q_0 \in Q$, accepting set $F \subseteq Q$ and total $\delta: Q \times \Sigma \to Q$. Exactly one next state is specified per state/symbol pair; this does not mean the whole machine has one transition. F may be empty, recognizing $\varnothing$.

Study-note section AUT06

> Start at q_0 and consume the entire input. Accept iff the ending state is in F; visiting an accepting state earlier is insufficient. ε is accepted exactly when $q_0 \in F$. A nonaccepting sink loops on every symbol and cannot reach acceptance; it still finishes after finite input. Finite automata have finite memory encoded by states, not no state or no ability to remember any history.

Question sources

Automata & Complexity Theory review questions · Question 7

Question 108 | NS0173

Over the alphabet Σ={a,b,c}, what is the minimum number of states in a complete DFA accepting exactly the strings that start with abc?

A. 3

B. 5

C. 4

D. 6

Hint

Read the transition domain and acceptance condition carefully. Track which inputs are consumed and evaluate the whole word under the stated model.

Answer B: 5

Before completing abc, the DFA must distinguish no matched symbol, a matched, and ab matched. It then needs an accepting state for all continuations after abc and a rejecting sink for any earlier mismatch. These five distinct situations require five states in a complete DFA. Including c in the alphabet makes the intended prefix language nonempty and the standard prefix-state construction applicable.

Why each choice is right or wrong

A - Incorrect: Three omits the accepting continuation and rejecting-mismatch situations.

B - Correct: Three incomplete-prefix stages plus accepting continuation and sink give five.

C - Incorrect: Four omits a necessary distinct situation in the complete DFA.

D - Incorrect: Six exceeds the five-state construction and is not minimal.

Primary note proof

Study-note section AUT08

> For a fixed valid word w of length n, a DFA for strings ending in w needs n+1 states representing the longest consumed suffix that is a prefix of w. Overlapping prefixes determine fallback transitions. Distinct prefixes are distinguishable by suitable continuations, proving the lower bound; do not add an unnecessary rejecting sink.

Study-note section AUT08

> For strings starting with w, a complete DFA ordinarily has n matched-prefix stages, an accepting continuation state and a rejecting sink: n+2 when n≥1 and the alphabet allows mismatches.

Question sources

Automata & Complexity Theory review questions · Question 8

Question 109 | NS0174

In a DFA, what happens after reaching a nonaccepting trap state while processing a finite input?

A. It immediately accepts the input

B. It restarts processing from the initial state

C. It stays there for every remaining input symbol and rejects after the input ends.

D. It transitions to the initial state on the next input

Hint

Read the transition domain and acceptance condition carefully. Track which inputs are consumed and evaluate the whole word under the stated model.

Answer C: It stays there for every remaining input symbol and rejects after the input ends.

A nonaccepting trap is a sink: every subsequent input symbol leaves the machine in that same state. It therefore cannot recover an accepting state. Once the finite remaining input is consumed, the DFA finishes and rejects. A self-loop transition does not imply an infinite computation after the input ends, nor does it restart processing or transfer to the start state.

Why each choice is right or wrong

A - Incorrect: The trap is nonaccepting, so it does not accept.

B - Incorrect: A sink stays in place rather than restarting.

C - Correct: It remains in the trap for the remaining symbols and rejects when input ends.

D - Incorrect: The next transition returns to the trap itself.

Primary note proof

Study-note section AUT06

> A deterministic finite automaton (DFA) is $M=(Q,\Sigma,\delta,q_0,F)$, with finite nonempty Q, start $q_0 \in Q$, accepting set $F \subseteq Q$ and total $\delta: Q \times \Sigma \to Q$. Exactly one next state is specified per state/symbol pair; this does not mean the whole machine has one transition. F may be empty, recognizing $\varnothing$.

Study-note section AUT06

> Start at q_0 and consume the entire input. Accept iff the ending state is in F; visiting an accepting state earlier is insufficient. ε is accepted exactly when $q_0 \in F$. A nonaccepting sink loops on every symbol and cannot reach acceptance; it still finishes after finite input. Finite automata have finite memory encoded by states, not no state or no ability to remember any history.

Question sources

Automata & Complexity Theory review questions · Question 9

Question 110 | NS0175

The transition function of a DFA is defined as:

A. $\delta: Q \times \Sigma \to Q$

B. $\delta: Q \times \Sigma \to 2^Q$

C. $\delta: Q \times \Sigma \times \Gamma \to Q \times \Gamma^* \times \{L,R\}$

D. $\delta: Q \times (\Sigma \cup \{\varepsilon\}) \to 2^Q$

Hint

Read the transition domain and acceptance condition carefully. Track which inputs are consumed and evaluate the whole word under the stated model.

Answer A: $\delta:Q\times\Sigma\to Q$.

AUT06 defines a total function from a state/input-symbol pair to one state. A set-valued codomain describes NFA choices; allowing epsilon in the input domain describes epsilon-NFA transitions. Stack alphabets and write/direction outputs belong to richer models and are not components of the ordinary DFA transition signature.

Why each choice is right or wrong

A - Correct: This is the exact state/symbol to unique next-state signature of a DFA.

B - Incorrect: A powerset codomain allows nondeterministic next-state choices.

C - Incorrect: The extra storage alphabet and write/direction outputs do not describe an ordinary finite automaton.

D - Incorrect: Epsilon input and set-valued successors describe an epsilon-NFA rather than a DFA.

Primary note proof

Study-note section AUT06

> A deterministic finite automaton (DFA) is $M=(Q,\Sigma,\delta,q_0,F)$, with finite nonempty Q, start $q_0\in Q$, accepting set $F\subseteq Q$ and total $\delta:Q\times\Sigma\to Q$. Exactly one next state is specified per state/symbol pair; this does not mean the whole machine has one transition. F may be empty, recognizing $\varnothing$.

Study-note section AUT06

> Start at q_0 and consume the entire input. Accept iff the ending state is in F; visiting an accepting state earlier is insufficient. ε is accepted exactly when $q_0\in F$. A nonaccepting sink loops on every symbol and cannot reach acceptance; it still finishes after finite input. Finite automata have finite memory encoded by states, not no state or no ability to remember any history.

Question sources

Automata & Complexity Theory review questions · Question 10

Question 111 | NS0176

Which operation on regular languages gives all strings that are NOT in the original language?

A. Reverse (L^R)

B. Complement (Σ* - L)

C. Kleene Star (L*)

D. Concatenation

Hint

Track mandatory symbols and factors that may repeat or disappear. Use short test words to check both directions of each proposed language description.

Answer B: Complement, relative to the fixed alphabet universe.

AUT03 defines complement as $\Sigma^* - L$ and requires a fixed alphabet. This selects precisely the words in the universe absent from L. Reversal changes symbol order, star concatenates copies and concatenation joins words; none performs the required membership negation. Completing a DFA before swapping final states is the recognition construction in AUT14.

Why each choice is right or wrong

A - Incorrect: Reversal changes word order rather than selecting all words outside the language.

B - Correct: Complement contains exactly the universe's words that are not members of L.

C - Incorrect: Kleene star concatenates zero or more language words and does not negate membership.

D - Incorrect: Concatenation joins words from languages rather than forming the universe-relative exclusion.

Primary note proof

Study-note section AUT03

> Σ^n contains words of exactly n symbols; for $|\Sigma|=k$, $|\Sigma^n|=k^n$, including $\Sigma^0=\{\varepsilon\}$. $\Sigma^*=\bigcup n\geq 0\Sigma^n$ and $\Sigma^+=\bigcup n\geq 1\Sigma^n$. Over a nonempty alphabet, Σ^+ excludes ε and $\Sigma^*=\Sigma^+\cup\{\varepsilon\}$. Σ^2 over {a,b} is {aa,ab,ba,bb}.

Study-note section AUT03

> For an arbitrary language L, $L^*=\bigcup n\geq 0L^n$ and $L^+=LL^*$. Unlike alphabet positive closure, L^+ can contain ε when $\varepsilon\in L$. $\varnothing^*=\{\varepsilon\}$, $\varnothing^+=\varnothing$, and $\{\varepsilon\}^*=\{\varepsilon\}$. Union means membership in either set; intersection requires the same word in both, not concatenation. Complement is Σ^*-L and needs a fixed alphabet.

Study-note section AUT14

> Right-linear rules have form A→aB, A→a or permitted ε rules; left-linear rules put the single variable at the opposite end, A→Ba. Both consistent orientations generate REG; arbitrarily mixing them can generate nonregular languages, for example S→aA|ε, A→Sb generates a^nb^n. A CFG rule A→aBA is legal but not regular-grammar form.

Study-note section AUT14

> Regular languages are closed under union, concatenation, star, reversal, intersection, difference and complement.

Question sources

Automata & Complexity Theory review questions · Question 11

Question 112 | NS0177

The language L* (Kleene Star of L) is defined as:

A. $L^1 \cup L^2 \cup L^3 \cup ...$

B. $L \cap \Sigma^*$

C. All strings not in L

D. $L^0 \cup L^1 \cup L^2 \cup ...$ (includes empty string because $L^0 = \{\varepsilon\}$)

Hint

Distinguish symbols, finite strings, their lengths and sets of strings. Apply the operation to its stated operands, including empty cases.

Answer D: The union of all nonnegative language powers.

AUT03 defines L* by union over every n≥0, and AUT02 gives $L^0=\{\varepsilon\}$. Thus the zero power guarantees the empty word even when L is empty. A omits the zero power and instead describes positive closure; intersection with the universe and complement are different operations.

Why each choice is right or wrong

A - Incorrect: This omits L^0 and defines positive closure rather than general Kleene star.

B - Incorrect: Intersecting a language with its word universe simply retains L.

C - Incorrect: All words outside L define complement, not repeated concatenation.

D - Correct: The nonnegative-power union is the definition and its zeroth term contributes epsilon.

Primary note proof

Study-note section AUT03

> Σ^n contains words of exactly n symbols; for $|\Sigma|=k$, $|\Sigma^n|=k^n$, including $\Sigma^0=\{\varepsilon\}$. $\Sigma^*=\bigcup n\geq 0\Sigma^n$ and $\Sigma^+=\bigcup n\geq 1\Sigma^n$. Over a nonempty alphabet, Σ^+ excludes ε and $\Sigma^*=\Sigma^+\cup\{\varepsilon\}$. Σ^2 over {a,b} is {aa,ab,ba,bb}.

Study-note section AUT03

> For an arbitrary language L, $L^*=\bigcup n\geq 0L^n$ and $L^+=LL^*$. Unlike alphabet positive closure, L^+ can contain ε when $\varepsilon\in L$. $\varnothing^*=\{\varepsilon\}$, $\varnothing^+=\varnothing$, and $\{\varepsilon\}^*=\{\varepsilon\}$. Union means membership in either set; intersection requires the same word in both, not concatenation. Complement is Σ^*-L and needs a fixed alphabet.

Study-note section AUT02

Concatenation xy writes x followed by y: cddc·ababb=cddcababb; order generally matters. εx=xε=x. Define $x^0=\varepsilon$ and x^n as n copies for n≥1. Reverse x^R reads symbols backward: caddba reverses to abddac; $(xy)^R=y^Rx^R$ and $(x^R)^R=x$.

Study-note section AUT02

For languages, $LM=\{xy:x\in L,y\in M\}$, $L^0=\{\varepsilon\}$, and $L^{n+1}=L^nL$. Thus {e,fgh}{c,d}={ec,ed,fghc,fghd}; fghe is absent.

Question sources

Automata & Complexity Theory review questions · Question 12

Question 113 | NS0178

In an NFA, the transition function maps each (state, input) pair to:

A. Exactly one next state

B. The initial state always

C. A SET of possible next states (including the empty set)

D. A symbol and a direction

Hint

Read the transition domain and acceptance condition carefully. Track which inputs are consumed and evaluate the whole word under the stated model.

Answer C: A set of possible successors, possibly empty.

AUT09 defines an NFA transition as a mapping into the powerset of Q, so zero, one or several successors are permitted. The empty set represents a stuck branch, not an invalid machine. A uniquely determined singleton is only a deterministic special case; fixed resets and tape write/direction outputs do not describe the NFA signature.

Why each choice is right or wrong

A - Incorrect: One successor is permitted but not required; NFA transitions may have zero or several.

B - Incorrect: An NFA transition need not return to the initial state.

C - Correct: The powerset codomain permits any subset of states, including the empty set.

D - Incorrect: Write-symbol and direction outputs are tape-machine operations, not NFA state-set transitions.

Primary note proof

Study-note section AUT09

> A nondeterministic finite automaton (NFA) permits zero, one or many successors: $\delta:Q\times\Sigma\to P(Q)$. An ε-NFA additionally permits moves without consuming input, using domain $Q\times(\Sigma\cup\{\varepsilon\})$. Some books include ε moves in the term NFA; state the convention. An NFA accepts iff at least one finite path consumes the complete word and finishes in F, including permitted ε moves before/after symbols.

Study-note section AUT09

> An NFA is not more expressive than a DFA: both recognize exactly REG. Nondeterminism can make a description smaller. A DFA is a special NFA after interpreting its unique successor as a singleton set.

Question sources

Automata & Complexity Theory review questions · Question 13

Question 114 | NS0179

An NFA accepts a string if:

A. ALL computation paths lead to an accepting state

B. AT LEAST ONE computation path leads to an accepting state

C. The majority of paths lead to accepting states

D. The last state visited is the accepting state

Hint

Read the transition domain and acceptance condition carefully. Track which inputs are consumed and evaluate the whole word under the stated model.

Answer B: At least one path consumes the whole word and finishes accepting.

AUT09 states existential acceptance over finite computation paths, with the entire word consumed and permitted epsilon moves included. B expresses that existential condition, interpreted with the full-input requirement. Failed branches do not cancel an accepting branch, and counting a majority or asking every path to accept gives different, incorrect conditions.

Why each choice is right or wrong

A - Incorrect: Requiring all paths would wrongly reject words with one accepting branch and another failing branch.

B - Correct: One complete accepting path suffices; it must consume the whole input.

C - Incorrect: Path majority is irrelevant; the acceptance condition is existential rather than statistical.

D - Incorrect: There is no single global last state across nondeterministic branches, and unread input cannot be ignored.

Primary note proof

Study-note section AUT09

> A nondeterministic finite automaton (NFA) permits zero, one or many successors: $\delta:Q\times\Sigma\to P(Q)$. An ε-NFA additionally permits moves without consuming input, using domain $Q\times(\Sigma\cup\{\varepsilon\})$. Some books include ε moves in the term NFA; state the convention. An NFA accepts iff at least one finite path consumes the complete word and finishes in F, including permitted ε moves before/after symbols.

Study-note section AUT09

> An NFA is not more expressive than a DFA: both recognize exactly REG. Nondeterminism can make a description smaller. A DFA is a special NFA after interpreting its unique successor as a singleton set.

Question sources

Automata & Complexity Theory review questions · Question 14

Question 115 | NS0180

Which statement about DFAs and NFAs is CORRECT?

A. Every NFA can be converted to an equivalent DFA recognizing the same language

B. NFAs are strictly more powerful than DFAs

C. DFAs can recognize languages that NFAs cannot

D. NFAs cannot recognize regular languages

Hint

Apply the stated transition, closure or equivalence rule step by step; distinguish a guaranteed bound from a value for one particular machine.

Answer A: Every NFA has an equivalent DFA.

AUT09 says both machine types recognize exactly regular languages, and AUT10 constructs equivalent DFA states as reachable NFA-state subsets. Nondeterminism may reduce representation size but does not add expressive power. Therefore A is true while each claim that one family recognizes languages inaccessible to the other is false.

Why each choice is right or wrong

A - Correct: Subset construction produces a DFA accepting exactly the NFA's language.

B - Incorrect: Nondeterminism may improve succinctness but does not exceed regular-language power.

C - Incorrect: Every DFA is also an NFA special case, so no such exclusive DFA language exists.

D - Incorrect: NFAs recognize exactly the regular languages, the opposite of this claim.

Primary note proof

Study-note section AUT09

> A nondeterministic finite automaton (NFA) permits zero, one or many successors: $\delta:Q\times\Sigma\to P(Q)$. An ε-NFA additionally permits moves without consuming input, using domain $Q\times(\Sigma\cup\{\varepsilon\})$. Some books include ε moves in the term NFA; state the convention. An NFA accepts iff at least one finite path consumes the complete word and finishes in F, including permitted ε moves before/after symbols.

Study-note section AUT09

> An NFA is not more expressive than a DFA: both recognize exactly REG. Nondeterminism can make a description smaller. A DFA is a special NFA after interpreting its unique successor as a singleton set.

Study-note section AUT10

> ε-closure(T) contains T and every state reachable through ε moves. Start the DFA at ε-closure($\{q_0\}$); for each current subset T and symbol a compute ε-closure($\bigcup q\in T\delta(q,a)$). A subset accepts iff it intersects F. Explore only reachable subsets; the empty subset is the rejecting sink when reached. An n-state NFA yields at most 2^n DFA states, not necessarily exactly 2^n.

Question sources

Automata & Complexity Theory review questions · Question 15

Question 116 | NS0181

What is an ε-transition in an NFA?

A. A transition that rejects the current input

B. A transition that only occurs at the end of input

C. A transition to the dead state

D. A transition that occurs without consuming any input symbol

Hint

Read the transition domain and acceptance condition carefully. Track which inputs are consumed and evaluate the whole word under the stated model.

Answer D: A move that consumes no input symbol.

AUT09 explicitly defines epsilon moves as transitions without consuming input. They may occur before, between or after symbol-consuming transitions when the machine permits them. They do not inherently mean rejection or movement to a sink, and they do not waive the requirement that an accepting computation consume the full word.

Why each choice is right or wrong

A - Incorrect: An epsilon move is not intrinsically rejecting; its destination and later path determine acceptance.

B - Incorrect: Permitted epsilon moves can occur before or between input symbols as well as at the end.

C - Incorrect: Its destination need not be a dead state.

D - Correct: The machine changes state without consuming a symbol from the input.

Primary note proof

Study-note section AUT09

> A nondeterministic finite automaton (NFA) permits zero, one or many successors: $\delta:Q\times\Sigma\to P(Q)$. An ε-NFA additionally permits moves without consuming input, using domain $Q\times(\Sigma\cup\{\varepsilon\})$. Some books include ε moves in the term NFA; state the convention. An NFA accepts iff at least one finite path consumes the complete word and finishes in F, including permitted ε moves before/after symbols.

Study-note section AUT09

> An NFA is not more expressive than a DFA: both recognize exactly REG. Nondeterminism can make a description smaller. A DFA is a special NFA after interpreting its unique successor as a singleton set.

Question sources

Automata & Complexity Theory review questions · Question 16

Question 117 | NS0182

When converting an NFA with n states to an equivalent DFA, the DFA may have at most how many states?

A. n states

B. n^2 states

C. 2^n states (the power set of Q)

D. n+1 states

Hint

Apply the stated transition, closure or equivalence rule step by step; distinguish a guaranteed bound from a value for one particular machine.

Answer C: At most 2^n states.

AUT10 states the subset-construction bound explicitly. An n-element state set has 2^n subsets, and each DFA state denotes one subset. Only reachable subsets need be included, so the bound need not be attained. Linear, quadratic and n+1 bounds are not general worst-case guarantees for NFA determinization.

Why each choice is right or wrong

A - Incorrect: A deterministic equivalent may require more than the original n states.

B - Incorrect: A quadratic bound does not cover the general powerset worst case.

C - Correct: There are 2^n possible subsets of the n NFA states.

D - Incorrect: An n+1 bound is not sufficient for general subset construction.

Primary note proof

Study-note section AUT10

> ε-closure(T) contains T and every state reachable through ε moves. Start the DFA at ε-closure({q_0}); for each current subset T and symbol a compute ε-closure(∪q∈Tδ(q,a)). A subset accepts iff it intersects F. Explore only reachable subsets; the empty subset is the rejecting sink when reached. An n-state NFA yields at most 2^n DFA states, not necessarily exactly 2^n.

Question sources

Automata & Complexity Theory review questions · Question 17

Question 118 | NS0183

What does the regular expression (0+1)*1 describe?

A. All binary strings

B. All binary strings that END with '1'

C. All binary strings starting with '1'

D. All binary strings containing exactly one '1'

Hint

Apply the stated transition, closure or equivalence rule step by step; distinguish a guaranteed bound from a value for one particular machine.

Answer B: All binary strings ending in 1.

With + as union, (0+1)* generates any finite binary prefix, including empty. Concatenating the final 1 forces the whole word to end in 1. It does not constrain the beginning or total number of ones: 01 and 11 are included, while epsilon and words ending in 0 are not.

Why each choice is right or wrong

A - Incorrect: Epsilon and words ending in zero are binary strings excluded by the final mandatory 1.

B - Correct: An arbitrary binary prefix followed by 1 is precisely a binary word ending in 1.

C - Incorrect: The arbitrary prefix may begin with zero, as in 01.

D - Incorrect: The prefix may contain additional ones, so exactly-one is too restrictive.

Primary note proof

Study-note section AUT12

> Classical expressions use ∅, ε, symbols, union, concatenation and star. A union B is written A|B or A+B; postfix r^+ means one-or-more r repetitions and is a different use of +. Parenthesize when conventions might clash. Star binds tightest, concatenation next, union last. r? means r|ε; r^+=rr*.

Study-note section AUT12

> (a|b)*a denotes words ending a; 0*10* denotes exactly one 1; a*(b|c) gives zero-or-more a's then exactly one b or c, so aab belongs but aaa and aaabc do not.

Question sources

Automata & Complexity Theory review questions · Question 18

Question 119 | NS0184

Which regular expression describes the language of all strings over {a,b} with at least one 'a'?

A. (a+b)*a(a+b)*

B. b*ab*

C. (a+b)+

D. a(a+b)*

Hint

Track mandatory symbols and factors that may repeat or disappear. Use short test words to check both directions of each proposed language description.

Answer A: (a+b)*a(a+b)*.

AUT13 gives this construction explicitly for at least one a. The middle a guarantees existence, while both stars permit additional a or b anywhere. b*ab* permits exactly one a; positive closure also permits all-b words, and a(a+b)* requires the first symbol rather than merely some symbol to be a.

Why each choice is right or wrong

A - Correct: The required middle a guarantees existence while unrestricted sides allow all other positions.

B - Incorrect: b*ab* excludes words with multiple a symbols and therefore denotes exactly one a.

C - Incorrect: Positive closure admits nonempty all-b words, which contain no a.

D - Incorrect: This requires a at the beginning and excludes valid words such as ba.

Primary note proof

Study-note section AUT13

> To require at least one a anywhere over {a,b}, use (a|b)*a(a|b)*. b*ab* requires exactly one a. At least two zeros uses (0|1)*0(0|1)*0(0|1)*; the zeros need not be adjacent. Testing 010 distinguishes it from a requirement for substring 00. For S→aS|ε, any number of a's is generated, giving a*.

Question sources

Automata & Complexity Theory review questions · Question 19

Question 120 | NS0185

The regular expression 0*10* represents:

A. All binary strings

B. All binary strings starting with 1

C. All binary strings containing exactly one '1'

D. All binary strings ending with '0'

Hint

Apply the stated transition, closure or equivalence rule step by step; distinguish a guaranteed bound from a value for one particular machine.

Answer C: Exactly one 1, with any number of zeros on either side.

AUT12 explicitly interprets 0*10* as exactly one 1. Both stars contribute only zeros, including none, so they cannot add another 1. The word 1 is accepted, while 11 and epsilon are not. No constraint forces the first symbol to be 1 or the final symbol to be zero.

Why each choice is right or wrong

A - Incorrect: Many binary words, including epsilon and 11, are excluded.

B - Incorrect: Leading zeros are permitted, so accepted words need not start with 1.

C - Correct: Only the mandatory middle symbol is 1; all other generated symbols are zeros.

D - Incorrect: The trailing zero star can be empty, so accepted words may end in 1.

Primary note proof

Study-note section AUT12

> Classical expressions use ∅, ε, symbols, union, concatenation and star. A union B is written A|B or A+B; postfix r^+ means one-or-more r repetitions and is a different use of +. Parenthesize when conventions might clash. Star binds tightest, concatenation next, union last. r? means r|ε; r^+=rr*.

Study-note section AUT12

> (a|b)*a denotes words ending a; 0*10* denotes exactly one 1; a*(b|c) gives zero-or-more a's then exactly one b or c, so aab belongs but aaa and aaabc do not.

Question sources

Automata & Complexity Theory review questions · Question 20

Question 121 | NS0186

Which of the following languages CANNOT be described by a regular expression?

A. All strings over {a,b} starting with 'a'

B. All binary strings of even length

C. All strings containing at least two 'b's

D. $\{a^n b^n \mid n \geq 1\}$ - strings with equal a's and b's

Hint

Respect the quantifier order and length constraints. A valid nonmembership argument must address every allowed decomposition, not one convenient split.

Answer D: The equal-count language $a^n b^n$ is nonregular.

AUT15 uses $a^p b^p$ and shows that pumping any permitted initial a block breaks equal counts. Classical regular expressions cannot describe this language. Starting with a fixed symbol, even length and a minimum number of b occurrences require only finite-state information, so the other descriptions are regular.

Why each choice is right or wrong

A - Incorrect: A fixed starting symbol can be recognized by finitely many prefix states.

B - Incorrect: Even length needs only two parity states and is regular.

C - Incorrect: Counting b occurrences up to a threshold of two needs finitely many states.

D - Correct: Unbounded equality of separated counts defeats regular pumping and therefore classical regular expressions.

Primary note proof

Study-note section AUT15

> If L is regular, some $p \geq 1$ satisfies: every $s \in L$ with $|s| \geq p$ has a split $s=xyz$, $|xy| \leq p$, $|y| \geq 1$, and $xy^i z \in L$ for every $i \geq 0$.

Study-note section AUT15

> For $L=\{a^n b^n : n \geq 0\}$, choose $a^p b^p$. Every allowed $y=a^k$ with $k \geq 1$; pumping $i=0$ leaves fewer a's than b's. The contradiction excludes every DFA.

Question sources

Automata & Complexity Theory review questions · Question 21

Question 122 | NS0187

In the Chomsky hierarchy, which grammar type corresponds to Finite Automata?

A. Type 0 - Unrestricted Grammar

B. Type 3 - Regular Grammar

C. Type 2 - Context-Free Grammar

D. Type 1 - Context-Sensitive Grammar

Hint

Compare production restrictions, memory access and language-family containments. Distinguish an exact characterization from an included subclass.

Answer B: Type 3 regular grammars.

AUT05 directly pairs Type 3 regular grammars with finite automata. Other grammar types can generate regular sublanguages, but the question asks which family corresponds to the finite-automaton model, an exact-family comparison. Type 2, Type 1 and Type 0 correspond to progressively stronger stack, bounded-tape and unrestricted-tape recognition families.

Why each choice is right or wrong

A - Incorrect: Type 0 corresponds to unrestricted Turing recognition rather than exactly finite-automaton power.

B - Correct: Type 3 is the regular grammar family exactly matched by finite automata.

C - Incorrect: Type 2 corresponds to nondeterministic pushdown recognition.

D - Incorrect: Type 1 corresponds to nondeterministic linear-bounded recognition.

Primary note proof

Study-note section AUT05

> Type 3 regular grammars correspond to finite automata; type 2 CFGs to nondeterministic PDAs; type 1 context-sensitive/noncontracting grammars to nondeterministic LBAs; type 0 unrestricted grammars to Turing recognizers.

Study-note section AUT05

> Use REG ⊊ CFL ⊊ CSL ⊊ DECIDABLE ⊊ RE, under the usual convention allowing S→ε with S absent from right sides when needed.

Question sources

Automata & Complexity Theory review questions · Question 22

Question 123 | NS0188

A formal grammar is defined as a 4-tuple G = (V, Σ, P, S). What does 'P' represent?

A. The set of all accepting states

B. The power set of the terminal alphabet

C. The set of production rules (A → β)

D. The pushdown stack alphabet

Hint

Distinguish symbols, finite strings, their lengths and sets of strings. Apply the operation to its stated operands, including empty cases.

Answer C: The set of production rules.

AUT04 defines V as variables, Σ as terminals, P as productions and S as the start variable. Productions specify the allowed rewriting steps used to generate terminal words. Accepting states and stack alphabets belong to machine definitions, while an alphabet powerset is not the meaning of P in this grammar tuple.

Why each choice is right or wrong

A - Incorrect: Accepting states are a recognizer component, not the production component of a grammar.

B - Incorrect: P is not the powerset notation for the alphabet in this tuple.

C - Correct: P denotes the allowed grammar-production rewrite rules.

D - Incorrect: A stack alphabet belongs to PDA storage rather than the grammar's production set.

Primary note proof

Study-note section AUT04

> Write G=(V,Σ,P,S): finite variables V, terminals Σ disjoint from V, productions P and start variable S∈V. A production replaces its left side by its right side. The language is L(G)={w∈Σ*:S⇒*w}. A recognizer processes a word and accepts exactly the language members. Grammars generate languages; automata recognize languages, not grammars.

Study-note section AUT04

> Two grammars are equivalent when their generated languages are equal, regardless of rule counts or variable names. One common word, equal alphabets or one-way inclusion is insufficient.

Question sources

Automata & Complexity Theory review questions · Question 23

Question 124 | NS0189

In a RIGHT-LINEAR grammar, production rules have the form:

A. A → aB or A → a (terminal before optional non-terminal on right)

B. A → Ba or A → a (non-terminal before terminal)

C. A → α where α is any string of terminals and non-terminals

D. αAβ → αγβ

Hint

Track mandatory symbols and factors that may repeat or disappear. Use short test words to check both directions of each proposed language description.

Answer A: A→aB or A→a, with permitted ε rules where needed.

AUT14 places the optional single variable at the right end after the terminal. That gives A; B is the opposite left-linear orientation. An unrestricted string on the right is CFG form rather than necessarily regular, and a context-sensitive rewrite with surrounding context is a different grammar restriction. Epsilon conventions are handled separately.

Why each choice is right or wrong

A - Correct: The optional nonterminal follows the terminal at the right end of the production.

B - Incorrect: Putting the nonterminal before the terminal gives a left-linear orientation.

C - Incorrect: An arbitrary right-hand string can contain multiple variables and need not be regular-grammar form.

D - Incorrect: The context-sensitive contextual rewrite is not the defining right-linear production form.

Primary note proof

Study-note section AUT14

> Right-linear rules have form A→aB, A→a or permitted ε rules; left-linear rules put the single variable at the opposite end, A→Ba. Both consistent orientations generate REG; arbitrarily mixing them can generate nonregular languages, for example S→aA|ε, A→Sb generates $a^n b^n$. A CFG rule A→aBA is legal but not regular-grammar form.

Study-note section AUT14

> Regular languages are closed under union, concatenation, star, reversal, intersection, difference and complement.

Question sources

Automata & Complexity Theory review questions · Question 24

Question 125 | NS0190

Which production rule form is VALID in a Context-Free Grammar but NOT in a Regular Grammar?

A. A → a

B. A → aB

C. A → ε

D. A → aBA (two non-terminals on right-hand side)

Hint

Track mandatory symbols and factors that may repeat or disappear. Use short test words to check both directions of each proposed language description.

Answer D: A→aBA is context-free but not regular-grammar form.

AUT14 explicitly identifies A→aBA as legal CFG form but not regular form. A CFG restricts the left side to one variable while permitting a variable/terminal string on the right. The other choices are regular productions or permitted epsilon cases. Two right-hand nonterminals violate the regular linear restriction.

Why each choice is right or wrong

A - Incorrect: A→a is an ordinary regular production as well as context-free.

B - Incorrect: A→aB is exactly a right-linear regular production.

C - Incorrect: An epsilon rule is allowed under the note's regular-grammar convention.

D - Correct: The single-variable left side is CFG-valid, but two right-side variables violate regular linear form.

Primary note proof

Study-note section AUT14

> Right-linear rules have form A→aB, A→a or permitted ε rules; left-linear rules put the single variable at the opposite end, A→Ba. Both consistent orientations generate REG; arbitrarily mixing them can generate nonregular languages, for example S→aA|ε, A→Sb generates $a^n b^n$. A CFG rule A→aBA is legal but not regular-grammar form.

Study-note section AUT14

> Regular languages are closed under union, concatenation, star, reversal, intersection, difference and complement.

Question sources

Automata & Complexity Theory review questions · Question 25

Question 126 | NS0191

A grammar is said to be AMBIGUOUS if:

A. It generates infinitely many strings

B. Some string in the language has more than one parse tree

C. It contains left-recursive production rules

D. It uses both left-linear and right-linear rules

Hint

Compare production restrictions, memory access and language-family containments. Distinguish an exact characterization from an included subclass.

Answer B: Some generated word has more than one parse tree.

AUT18 defines ambiguity by two distinct parse trees for at least one word, equivalently two leftmost derivations. Infinite size, left recursion and mixed linear orientations concern other properties and do not establish this condition. One must exhibit competing trees, not merely different expansion orders of a single tree.

Why each choice is right or wrong

A - Incorrect: Infinite languages can have unambiguous grammars, so infinitude does not define ambiguity.

B - Correct: Multiple parse trees for one generated word are the precise ambiguity criterion.

C - Incorrect: Left recursion can encode unambiguous left associativity; it does not alone prove ambiguity.

D - Incorrect: Mixing linear orientations concerns grammar form and is not the parse-tree multiplicity definition.

Primary note proof

Study-note section AUT18

> A CFG is ambiguous when some word has two distinct parse trees, equivalently two different leftmost derivations. Different expansion orders that give the same tree do not prove ambiguity.

Study-note section AUT18

> Separate precedence levels: E→E+T|T, T→T*F|F, F→id|(E). Multiplication binds inside T before addition at E, and left recursion encodes left association. Uniqueness follows by the top-level last operator split and induction. S→SS|ε is already ambiguous for ε. S→A|B is ambiguous if A→ab and B→abB|ε both produce ab. Ambiguous grammar and inherently ambiguous language differ: another grammar may remove ambiguity.

Question sources

Automata & Complexity Theory review questions · Question 26

Question 127 | NS0192

In a derivation tree (parse tree), what do the LEAF NODES represent?

A. Non-terminal symbols that haven't been expanded yet

B. The start symbol S

C. Terminal symbols - the actual characters in the generated string

D. ε-transitions in the grammar

Hint

Apply the stated transition, closure or equivalence rule step by step; distinguish a guaranteed bound from a value for one particular machine.

Answer C: Terminal symbols, with an explicit ε leaf where that convention is used.

AUT16 describes complete parse trees with terminal leaves and sometimes an epsilon leaf for an empty production. Reading leaves left to right yields the generated word. Unexpanded variables belong to incomplete trees, and a start symbol is normally the root. Epsilon productions are not automaton transitions in this representation.

Why each choice is right or wrong

A - Incorrect: Unexpanded nonterminals indicate an incomplete tree rather than leaves of the completed derivation.

B - Incorrect: S is normally the root, not the general category of terminal leaves.

C - Correct: Terminal leaves spell the derived word; epsilon may be displayed for an empty production.

D - Incorrect: A parse tree records productions, not automaton epsilon-transition edges.

Primary note proof

Study-note section AUT16

> In a context-free grammar (CFG) each production has one variable on the left, $A \to \alpha$ with $\alpha \in (V \cup \Sigma)^*$. The left side therefore also belongs to $V \cup \Sigma$ and its string closure; alternatives stating these weaker memberships can all be true. A sentential form is any string derived from S, including terminal-only sentences and S itself.

Study-note section AUT16

> A leftmost derivation always replaces the leftmost variable; a rightmost one replaces the rightmost variable. This is not scanning the completed input in opposite directions. A complete parse tree has root S, internal variable nodes expanded by productions and terminal leaves, with ε leaves when that convention is used. Read the leaves left-to-right for the yielded word. The whole tree represents derivation structure; a path in a derivation graph is a different object.

Question sources

Automata & Complexity Theory review questions · Question 27

Question 128 | NS0193

What is the key difference between left derivation and right derivation?

A. Left derivation always expands the LEFTMOST non-terminal; right derivation expands the RIGHTMOST

B. Left derivation produces left-linear grammars; right produces right-linear grammars

C. Left derivation is used only for regular grammars

D. They produce different final strings for the same grammar

Hint

Apply the stated transition, closure or equivalence rule step by step; distinguish a guaranteed bound from a value for one particular machine.

Answer A: Expand the leftmost versus rightmost remaining nonterminal.

AUT16 defines the two derivations by which remaining variable is rewritten. Both can describe the same final word and even the same parse tree. They do not determine whether the grammar itself is left-linear or right-linear, and the distinction applies to CFGs generally rather than only regular grammars.

Why each choice is right or wrong

A - Correct: The choice of the leftmost or rightmost remaining variable is the defining difference.

B - Incorrect: Derivation order does not convert the production system into a left-linear or right-linear grammar.

C - Incorrect: Leftmost derivation is used for context-free grammars generally, not only regular ones.

D - Incorrect: Different expansion orders can generate the same terminal word and the same tree.

Primary note proof

Study-note section AUT16

> In a context-free grammar (CFG) each production has one variable on the left, $A \to \alpha$ with $\alpha \in (V \cup \Sigma)^*$. The left side therefore also belongs to $V \cup \Sigma$ and its string closure; alternatives stating these weaker memberships can all be true. A sentential form is any string derived from S, including terminal-only sentences and S itself.

Study-note section AUT16

> A leftmost derivation always replaces the leftmost variable; a rightmost one replaces the rightmost variable. This is not scanning the completed input in opposite directions. A complete parse tree has root S, internal variable nodes expanded by productions and terminal leaves, with ε leaves when that convention is used. Read the leaves left-to-right for the yielded word. The whole tree represents derivation structure; a path in a derivation graph is a different object.

Question sources

Automata & Complexity Theory review questions · Question 28

Question 129 | NS0194

The language $\{a^n b^n \mid n \geq 1\}$ is:

A. Regular - recognized by a DFA

B. Not a formal language

C. Context-Free but NOT Regular - recognized by a PDA, not a DFA

D. Context-Sensitive but not Context-Free

Hint

Respect the quantifier order and length constraints. A valid nonmembership argument must address every allowed decomposition, not one convenient split.

Answer C: Context-free but not regular.

AUT17 generates $a^n b^n$ with matched recursive additions, using ab as the base for n≥1. AUT15 proves the family nonregular by pumping the initial a block. Thus a PDA suffices but a DFA does not. It remains a formal language and is also CSL, so a description denying context-freeness is wrong.

Why each choice is right or wrong

A - Incorrect: The regular pumping argument rules out any DFA for unbounded equal counts.

B - Incorrect: This is a precisely specified set of words and therefore a formal language.

C - Correct: A CFG/PDA handles matched counts, while the language is not regular.

D - Incorrect: It is context-sensitive too, but denying its context-free status is false.

Primary note proof

Study-note section AUT17

> Use a recursive rule to enforce matched structure and a base rule to stop. $S \to aSb \mid \varepsilon$ generates exactly $a^n b^n$ for n≥0: n recursive steps add n a's on the left and n b's on the right. Derive aabb by $S \Rightarrow aSb \Rightarrow aaSbb \Rightarrow aabb$. For n≥1 replace ε by base ab.

Study-note section AUT15

> If L is regular, some p≥1 satisfies: every $s \in L$ with $|s| \geq p$ has a split $s=xyz$, $|xy| \leq p$, $|y| \geq 1$, and $xy^i z \in L$ for every i≥0.

Study-note section AUT15

> For $L=\{a^n b^n : n \geq 0\}$, choose $a^p b^p$. Every allowed $y=a^k$ with k≥1; pumping i=0 leaves fewer a's than b's. The contradiction excludes every DFA.

Study-note section AUT05

Type 3 regular grammars correspond to finite automata; type 2 CFGs to nondeterministic PDAs; type 1 context-sensitive/noncontracting grammars to nondeterministic LBAs; type 0 unrestricted grammars to Turing recognizers.

Study-note section AUT05

Use REG ⊊ CFL ⊊ CSL ⊊ DECIDABLE ⊊ RE, under the usual convention allowing S→ε with S absent from right sides when needed.

Question sources

Automata & Complexity Theory review questions · Question 29

Question 130 | NS0195

Every regular language is a context-free language. This means:

A. Every CFL can be described by a regular expression

B. Any language recognizable by a DFA/NFA can also be recognized by a PDA

C. Regular grammars are more powerful than CFGs

D. CFGs and regular grammars generate the same set of languages

Hint

Compare production restrictions, memory access and language-family containments. Distinguish an exact characterization from an included subclass.

Answer B: A PDA can also recognize every DFA/NFA language.

AUT05 places regular languages inside context-free languages, and AUT09 equates DFA with NFA power. Thus a PDA can recognize every finite-automaton language, for example by ignoring its extra stack. The containment is strict; it does not give regular expressions for every CFL or make regular and CFG language families equal.

Why each choice is right or wrong

A - Incorrect: The proper inclusion leaves CFLs that no classical regular expression describes.

B - Correct: Finite-automaton languages are included in the PDA-recognizable context-free family.

C - Incorrect: Regular grammars are less expressive, not more expressive, than CFGs.

D - Incorrect: The note gives a proper containment rather than equality of the two families.

Primary note proof

Study-note section AUT05

> Type 3 regular grammars correspond to finite automata; type 2 CFGs to nondeterministic PDAs; type 1 context-sensitive/noncontracting grammars to nondeterministic LBAs; type 0 unrestricted grammars to Turing recognizers.

Study-note section AUT05

> Use REG ⊊ CFL ⊊ CSL ⊊ DECIDABLE ⊊ RE, under the usual convention allowing S→ε with S absent from right sides when needed.

Question sources

Automata & Complexity Theory review questions · Question 30

Question 131 | NS0196

What additional component does a Pushdown Automaton (PDA) have that a Finite Automaton lacks?

A. A set of accepting states

B. A transition function

C. An input tape

D. A stack (LIFO memory)

Hint

Compare production restrictions, memory access and language-family containments. Distinguish an exact characterization from an included subclass.

Answer D: A stack.

AUT21 defines a PDA by adding a stack to finite control, with its own alphabet and initial symbol. Finite automata already have states, accepting states and transition rules and process input. The distinguishing unbounded LIFO storage is what supports matched-count recognition beyond finite-state memory.

Why each choice is right or wrong

A - Incorrect: Finite automata also have an accepting-state set.

B - Incorrect: Finite automata already use a transition function.

C - Incorrect: Both models process input; an input representation is not the added PDA memory.

D - Correct: The stack is the extra LIFO storage that distinguishes the PDA from finite control alone.

Primary note proof

Study-note section AUT21

> A nondeterministic pushdown automaton (PDA) adds a stack to finite control: $(Q,\Sigma,\Gamma,\delta,q_0,Z_0,F)$. Its transition relation maps (state,input symbol or ε,stack top) to possible (new state,replacement stack string) pairs.

Study-note section AUT-S02

> A stack is last-in, first-out: push adds an item at the top and pop removes the top item, so the most recently pushed item is removed first. A PDA accesses this top rather than an arbitrary interior position.

Study-note section AUT-S02

> A queue is first-in, first-out: insertion at the rear and removal at the front returns the oldest waiting item. A linked list can implement a storage discipline, but the implementation container is different from the abstract push/pop or enqueue/dequeue access rule.

Question sources

Automata & Complexity Theory review questions · Question 31

Question 132 | NS0197

The stack in a PDA operates as a:

A. Last-In-First-Out (LIFO) data structure

B. First-In-First-Out (FIFO) data structure

C. Random access memory

D. Circular queue

Hint

Compare production restrictions, memory access and language-family containments. Distinguish an exact characterization from an included subclass.

Answer A: Last in, first out.

AUT21 defines each PDA move using the current stack top and its replacement string. This top-only discipline means the most recently pushed item is the next removed: LIFO. A queue removes oldest items, random-access storage permits arbitrary addressing, and circular-queue organization is not the PDA stack model.

Why each choice is right or wrong

A - Correct: Top-only push/pop access removes the most recently pushed item first.

B - Incorrect: FIFO removes the oldest item and describes queue rather than stack behavior.

C - Incorrect: A PDA cannot arbitrarily access interior stack positions like random-access memory.

D - Incorrect: Circular queue organization does not express the top-only LIFO stack discipline.

Primary note proof

Study-note section AUT21

> A nondeterministic pushdown automaton (PDA) adds a stack to finite control: $(Q,\Sigma,\Gamma,\delta,q_0,Z_0,F)$. Its transition relation maps (state,input symbol or ε,stack top) to possible (new state,replacement stack string) pairs.

Study-note section AUT-S02

> A stack is last-in, first-out: push adds an item at the top and pop removes the top item, so the most recently pushed item is removed first. A PDA accesses this top rather than an arbitrary interior position.

Study-note section AUT-S02

> A queue is first-in, first-out: insertion at the rear and removal at the front returns the oldest waiting item. A linked list can implement a storage discipline, but the implementation container is different from the abstract push/pop or enqueue/dequeue access rule.

Question sources

Automata & Complexity Theory review questions · Question 32

Question 133 | NS0198

A PDA formally defined as a 7-tuple includes which of the following that a DFA does NOT have?

A. A set of states Q

B. An input alphabet Σ

C. A stack alphabet Γ and initial stack symbol Z

D. A set of accepting states F

Hint

Compare production restrictions, memory access and language-family containments. Distinguish an exact characterization from an included subclass.

Answer C: A stack alphabet and initial stack symbol.

AUT21 gives the PDA tuple $(Q,\Sigma,\Gamma,\delta,q_0,Z_0,F)$. Compared with a DFA, Γ and Z_0 describe its added stack storage. States, input symbols and accepting states occur in both models. Their presence is not what expands the recognizer beyond finite-state control.

Why each choice is right or wrong

A - Incorrect: Both DFA and PDA models have a finite control-state set Q.

B - Incorrect: Both machines process symbols from an input alphabet Σ.

C - Correct: Γ and Z_0 specify stack storage, which an ordinary DFA lacks.

D - Incorrect: Accepting-state sets occur in both models and do not distinguish stack memory.

Primary note proof

Study-note section AUT21

> A nondeterministic pushdown automaton (PDA) adds a stack to finite control: $(Q,\Sigma,\Gamma,\delta,q_0,Z_0,F)$. Its transition relation maps (state,input symbol or ε,stack top) to possible (new state,replacement stack string) pairs. Fix which end denotes the top before tracing. Replacing X by X leaves the stack unchanged; ε input means consume no symbol, while replacing X by ε pops it. These are different uses of ε.

Question sources

Automata & Complexity Theory review questions · Question 33

Question 134 | NS0199

The transition function of a PDA takes which inputs and produces which outputs?

A. (state, input) → next state

B. (state, input symbol or ε, top of stack) → (new state, new stack string)

C. (state, input) → set of states

D. (state, stack top, next input) → (write symbol, direction)

Hint

Compare production restrictions, memory access and language-family containments. Distinguish an exact characterization from an included subclass.

Answer B: State, input-or-ε and stack top map to possible new states and replacement stack strings.

AUT21 defines a PDA move by the current state, consumed symbol or epsilon and top stack symbol. Its possible outputs combine a new state with a replacement stack string. Set-valued nondeterminism is understood even though B abbreviates an individual result. The alternatives lacking stack access or using tape directions describe other models.

Why each choice is right or wrong

A - Incorrect: This omits the stack top and replacement string essential to PDA behavior.

B - Correct: This includes the current stack top and replacement string; nondeterminism permits a set of such results.

C - Incorrect: A state-set output without stack operations describes finite-state nondeterminism instead.

D - Incorrect: Writing a tape symbol and moving a head is TM-style machinery, not PDA stack replacement.

Primary note proof

Study-note section AUT21

> A nondeterministic pushdown automaton (PDA) adds a stack to finite control: $(Q,\Sigma,\Gamma,\delta,q_0,Z_0,F)$. Its transition relation maps (state,input symbol or ε,stack top) to possible (new state,replacement stack string) pairs. Fix which end denotes the top before tracing. Replacing X by X leaves the stack unchanged; ε input means consume no symbol, while replacing X by ε pops it. These are different uses of ε.

Question sources

Automata & Complexity Theory review questions · Question 34

Question 135 | NS0200

Which of the following languages can a PDA recognize but a DFA CANNOT?

A. All binary strings ending with '1'

B. All strings over {a,b} starting with 'ab'

C. $\{a^n b^n \mid n \geq 1\}$ - strings with equal number of a's and b's

D. All strings of even length over {a,b}

Hint

Respect the quantifier order and length constraints. A valid nonmembership argument must address every allowed decomposition, not one convenient split.

Answer C: The nonregular equal-count language $a^n b^n$.

AUT22 supplies the push-one/pop-one PDA procedure for equal a/b counts, and AUT15 excludes a DFA. Fixed suffixes, fixed prefixes and length parity require only finite memory, so both machine types can recognize those. C alone describes a language in PDA capability but outside DFA capability.

Why each choice is right or wrong

A - Incorrect: A fixed final symbol is a regular condition recognized by a DFA.

B - Incorrect: A fixed starting prefix needs only finitely many match stages and is regular.

C - Correct: A PDA matches unbounded equal counts using its stack, while regular pumping rules out a DFA.

D - Incorrect: Even length needs only a finite parity state, so a DFA already suffices.

Primary note proof

Study-note section AUT22

> For $a^n b^n$, push one X per a, switch to a b-reading phase, pop one X per b, and accept when input ends at the bottom marker. Reject a after entering the b phase and b when no X remains. Handle ε explicitly if n=0 is permitted. A stack counts without storing the full input as random-access memory.

Study-note section AUT15

> If L is regular, some $p \geq 1$ satisfies: every $s \in L$ with $|s| \geq p$ has a split $s=xyz$, $|xy| \leq p$, $|y| \geq 1$, and $xy^i z \in L$ for every $i \geq 0$.

Study-note section AUT15

> For $L=\{a^n b^n : n \geq 0\}$, choose $a^p b^p$. Every allowed $y=a^k$ with $k \geq 1$; pumping i=0 leaves fewer a's than b's. The contradiction excludes every DFA.

Question sources

Automata & Complexity Theory review questions · Question 35

Question 136 | NS0201

The language $\{a^n b^n c^n \mid n \geq 1\}$ is NOT a context-free language. This means:

A. It can be recognized by a PDA

B. It can be described by a context-free grammar

C. It is not a formal language

D. No PDA can recognize it; it requires at least a Linear Bounded Automaton (LBA)

Hint

Respect the quantifier order and length constraints. A valid nonmembership argument must address every allowed decomposition, not one convenient split.

Answer D: No PDA recognizes it; an LBA suffices in the listed hierarchy.

AUT24 proves three equal counts non-context-free, and AUT25 recognizes them by marking matched triples within the input region. Thus no ordinary single-stack PDA recognizes the language, while an LBA does. It is still a precisely defined formal language; non-context-freeness is a family boundary rather than failure to be a language.

Why each choice is right or wrong

A - Incorrect: PDA-recognizable languages are context-free, contradicting the stated non-CFL property.

B - Incorrect: A CFG would make it context-free, which the pumping proof rules out.

C - Incorrect: It remains a formal language despite lying outside the context-free family.

D - Correct: No PDA suffices; bounded triple marking provides an LBA recognizer in the next listed family.

Primary note proof

Study-note section AUT25

> A context-sensitive rule $\alpha A \beta \to \alpha \gamma \beta$ replaces A in context, with γ nonempty. Equivalently, noncontracting grammar presentations preserve/increase total length, with the isolated ε exception. The condition is on the complete sides, not an unrelated comparison $|\beta| \geq |\alpha|$.

Study-note section AUT25

> A linear bounded automaton (LBA) is a TM with O(n) tape cells on length-n input; the standard CSL characterization uses nondeterministic LBAs. For $a^n b^n c^n$ first verify a*b*c* order, repeatedly mark one unmarked a, b and c, and reject missing/excess symbols.

Study-note section AUT25

> Thus every CSL is decidable.

Study-note section AUT24

If L is context-free, some p ensures each sufficiently long $s \in L$ has $s=uvxyz$ with $|vxy| \le p$, $|vy| \ge 1$ and $uv^ixy^iz \in L$ for every $i \ge 0$. The two pieces v and y are pumped together; either may be empty, not both. A non-CFL proof must defeat every allowed decomposition.

Study-note section AUT24

For $s=a^pb^pc^p$, the window vxy cannot contain both an a and a c. Pumping affects at most two blocks and at least one count. Use i=0 or 2: the unaffected third block retains p while another count changes, or symbol order breaks. Hence $a^nb^nc^n$ is not CFL.

Question sources

Automata & Complexity Theory review questions · Question 36

Question 137 | NS0202

In a Context-Sensitive Grammar, the production rule $\alpha A \beta \rightarrow \alpha \gamma \beta$ requires:

A. $|\gamma| < |A|$ (right side shorter than left)

B. $|\alpha\gamma\beta| \geq |\alpha A \beta|$ (right side no shorter than left side)

C. α and β must be empty strings

D. γ must be a single terminal symbol

Hint

Compare production restrictions, memory access and language-family containments. Distinguish an exact characterization from an included subclass.

Answer B: The complete right-hand side cannot be shorter.

AUT25 explains contextual rules with a nonempty replacement γ, equivalently noncontracting full-side length under the isolated epsilon convention. The comparison must include the complete left and right sides, making B correct. Context α and β need not vanish, and γ may contain more than a single terminal.

Why each choice is right or wrong

A - Incorrect: A shorter replacement contradicts the noncontracting context-sensitive requirement.

B - Correct: The full right-hand side must be at least as long as the full left-hand side.

C - Incorrect: Context can be present; requiring empty α and β is not the context-sensitive rule.

D - Incorrect: Gamma is nonempty but need not be exactly one terminal.

Primary note proof

Study-note section AUT25

> A context-sensitive rule $\alpha A \beta \rightarrow \alpha \gamma \beta$ replaces A in context, with γ nonempty. Equivalently, noncontracting grammar presentations preserve/increase total length, with the isolated ε exception. The condition is on the complete sides, not an unrelated comparison $|\beta| \geq |\alpha|$.

Study-note section AUT25

> A linear bounded automaton (LBA) is a TM with O(n) tape cells on length-n input; the standard CSL characterization uses nondeterministic LBAs. For $a^n b^n c^n$ first verify a*b*c* order, repeatedly mark one unmarked a, b and c, and reject missing/excess symbols.

Study-note section AUT25

> Thus every CSL is decidable.

Question sources

Automata & Complexity Theory review questions · Question 37

Question 138 | NS0203

The Turing Machine transition function δ(q, a) = (p, b, L) means:

A. In state q reading symbol a: go to state p, write b, move head LEFT

B. In state q reading symbol a: go to state p, write b, move head RIGHT

C. In state q reading symbol a: go to state p, push b onto stack, move LEFT

D. In state q reading symbol a: delete b, move LEFT, enter state p

Hint

Compare production restrictions, memory access and language-family containments. Distinguish an exact characterization from an included subclass.

Answer A: Write b, move left, and enter p after reading a in q.

AUT27 directly interprets δ(q,a)=(p,b,L). The current scanned symbol a is replaced with b, the head shifts one cell left and control enters p. L fixes the direction; there is no stack push operation, and deletion is not the meaning of writing b.

Why each choice is right or wrong

A - Correct: This exactly preserves the specified new state, written symbol and leftward head movement.

B - Incorrect: R would indicate right movement; the given L specifies left.

C - Incorrect: The TM writes b on tape rather than pushing onto a separate stack.

D - Incorrect: The transition writes b; it does not mean delete b.

Primary note proof

Study-note section AUT27

> δ(q,a)=(p,b,L) means read a in q, write b, move one cell left and enter p. Accept/reject halting states have no further moves under this convention.

Question sources

Automata & Complexity Theory review questions · Question 38

Question 139 | NS0204

Which of the following is TRUE about Turing Machines?

A. A TM has a finite tape of fixed length

B. A TM can only move the head in one direction

C. A TM can simulate any algorithm - it is the most powerful computational model

D. A TM is equivalent in power to a PDA

Hint

Compare production restrictions, memory access and language-family containments. Distinguish an exact characterization from an included subclass.

Answer C: A TM models general effective algorithmic computation.

AUT28 identifies Turing completeness with simulating general TM computation and relates effective procedures to the Church-Turing thesis. C expresses this ordinary model comparison, not omnipotence over undecidable tasks or a speed guarantee. A standard tape is unbounded, heads may move both directions, and TMs exceed single-stack PDA recognition power.

Why each choice is right or wrong

A - Incorrect: A standard TM tape is unbounded rather than fixed to a finite length.

B - Incorrect: Standard TM transitions may move the head left or right.

C - Correct: Under the effective-algorithm interpretation, TM computation captures general algorithmic procedures.

D - Incorrect: TMs recognize languages beyond the context-free languages of ordinary PDAs.

Primary note proof

Study-note section AUT28

> An off-line TM has a read-only input tape and separate read/write work tape. Multitape, multihead, two-way-infinite-tape and nondeterministic TMs have the same language-recognition power as standard TMs, though simulation costs differ. Two unbounded stacks can simulate a tape. Turing completeness means ability to simulate general TM computation, not polynomial speed or literally using infinitely many cells in a finite run. The Church-Turing thesis connects effective procedures to this formal model; it is not a proved runtime theorem.

Question sources

Automata & Complexity Theory review questions · Question 39

Question 140 | NS0205

Which option summarizes the pumping lemma for regular languages, rather than a separate consequence for a chosen recognizing DFA?

A. Every string in L can be pumped to produce longer strings still in L

B. Any sufficiently long string s in L can be split as xyz where the middle part y can be 'pumped' (repeated any number of times) and stay in L

C. L is infinite if and only if a string in L is longer than the number of states

D. All strings in L have the same length

Hint

Respect the quantifier order and length constraints. A valid nonmembership argument must address every allowed decomposition, not one convenient split.

Answer B: Any sufficiently long string s in L can be split as xyz where the middle part y can be 'pumped' (repeated any number of times) and stay in L

For each regular language there is a pumping length p. Every accepted word of length at least p has a decomposition xyz with $|xy| \le p$ and $|y| > 0$ such that xy^iz remains accepted for every $i \ge 0$. The threshold and nonempty repeated part matter; short words need not be pumpable. The infinitude criterion involving a DFA's state count is a separate consequence, while a regular language can contain words of many different lengths.

Why each choice is right or wrong

A - Incorrect: The lemma applies only above a length threshold and pumps a suitable part, not every word without qualification.

B - Correct: This states the sufficiently-long-word split and repeatable middle part; the formal bounds are supplied in the explanation.

C - Incorrect: A fixed-DFA infinitude criterion is a consequence rather than the pumping lemma's quantified statement.

D - Incorrect: Regular languages need not have constant word length; 0* contains arbitrarily many lengths.

Primary note proof

Study-note section AUT15

> If L is regular, some $p \ge 1$ satisfies: every $s \in L$ with $|s| \ge p$ has a split $s=xyz$, $|xy| \le p$, $|y| \ge 1$, and $xy^iz \in L$ for every $i \ge 0$.

Study-note section AUT15

> For $L=\{a^nb^n : n \ge 0\}$, choose a^pb^p. Every allowed $y=a^k$ with $k \ge 1$; pumping $i=0$ leaves fewer a's than b's. The contradiction excludes every DFA.

Question sources

Automata & Complexity Theory review questions · Question 40

Question 141 | NS0206

The language $L = \{a^n b^n c^n \mid n \geq 1\}$ is NOT context-free. Which technique proves this?

A. The Pumping Lemma for Regular Languages

B. The Chomsky hierarchy alone

C. Constructing a PDA and showing it fails

D. The Pumping Lemma for Context-Free Languages - any pumping violates the equal-count constraint

Hint

Respect the quantifier order and length constraints. A valid nonmembership argument must address every allowed decomposition, not one convenient split.

Answer D: The CFL pumping lemma, defeating every permitted decomposition.

AUT24 chooses $a^p b^p c^p$ and shows the bounded vxy window cannot cover both a and c blocks. Pumping changes at most two counts or breaks block order, defeating every allowed decomposition. A failed individual PDA is not a proof against all PDAs, and regular pumping excludes only regularity, not the stronger CFL property.

Why each choice is right or wrong

A - Incorrect: Regular pumping disproves regularity and does not by itself establish non-context-freeness.

B - Incorrect: The hierarchy alone does not prove the specific language lies outside CFL.

C - Incorrect: Failure of one proposed PDA does not rule out every possible PDA.

D - Correct: The CFL pumping proof breaks every legal decomposition by changing counts or symbol order.

Primary note proof

Study-note section AUT24

> If L is context-free, some p ensures each sufficiently long $s \in L$ has $s=uvxyz$ with $|vxy| \leq p$, $|vy| \geq 1$ and $uv^i xy^i z \in L$ for every $i \geq 0$. The two pieces v and y are pumped together; either may be empty, not both. A non-CFL proof must defeat every allowed decomposition.

Study-note section AUT24

> For $s=a^p b^p c^p$, the window vxy cannot contain both an a and a c. Pumping affects at most two blocks and at least one count. Use i=0 or 2: the unaffected third block retains p while another count changes, or symbol order breaks. Hence $a^n b^n c^n$ is not CFL.

Question sources

Automata & Complexity Theory review questions · Question 41

Question 142 | NS0207

'Turing Completeness' means a system:

A. Can simulate a Turing Machine and thus compute anything computable

B. Solves problems in polynomial time

C. Uses an infinite amount of memory

D. Is equivalent to a Pushdown Automaton

Hint

Compare production restrictions, memory access and language-family containments. Distinguish an exact characterization from an included subclass.

Answer A: Ability to simulate general Turing-machine computation.

AUT28 defines Turing completeness by general TM simulation. This says which effective computations can be expressed, not whether they run in polynomial time. A finite execution uses finitely many cells even though the model permits unbounded memory. Single-stack PDA equivalence is weaker than Turing universality.

Why each choice is right or wrong

A - Correct: Simulation of general TM computation is the definition of Turing completeness.

B - Incorrect: Universality does not guarantee a polynomial-time algorithm for every computable problem.

C - Incorrect: A finite run need not and cannot literally use infinitely many cells.

D - Incorrect: An ordinary PDA has less expressive recognition power than a TM.

Primary note proof

Study-note section AUT28

> An off-line TM has a read-only input tape and separate read/write work tape. Multitape, multihead, two-way-infinite-tape and nondeterministic TMs have the same language-recognition power as standard TMs, though simulation costs differ. Two unbounded stacks can simulate a tape. Turing completeness means ability to simulate general TM computation, not polynomial speed or literally using infinitely many cells in a finite run. The Church-Turing thesis connects effective procedures to this formal model; it is not a proved runtime theorem.

Question sources

Automata & Complexity Theory review questions · Question 42

Question 143 | NS0208

The complexity class P contains problems that are:

A. Hard to solve and hard to verify

B. Solvable in polynomial time by a deterministic Turing Machine

C. Solvable only by nondeterministic machines

D. Requiring exponential space to solve

Hint

Separate resource bounds, certificate verification and reduction requirements. Use proved containments without assuming unresolved class separations.

Answer B: Deterministic polynomial-time decidable problems.

AUT34 defines P by deterministic polynomial-time decision relative to input length. It is a solvability upper bound, not a requirement of hardness, nondeterminism or exponential space. A problem in P may also have nondeterministic algorithms, but it is the existence of a deterministic polynomial-time decider that establishes membership.

Why each choice is right or wrong

A - Incorrect: P is defined by efficient deterministic decision, not by hardness of solving or verification.

B - Correct: This is the exact deterministic polynomial-time definition of P.

C - Incorrect: Problems in P have deterministic algorithms and are not solvable only nondeterministically.

D - Incorrect: Exponential space is not a requirement for a polynomial-time computation.

Primary note proof

Study-note section AUT34

> Complexity classes here contain decision languages. P consists of deterministic polynomial-time decidable languages; NP consists of nondeterministic polynomial-time decidable languages, equivalently polynomially verifiable yes-instances with polynomial-size certificates. A verifier must accept some certificate exactly for yes-inputs; no certificate may falsely establish a no-instance. NP means nondeterministic polynomial, not non-polynomial.

Study-note section AUT34

> P⊆NP because a verifier can ignore its certificate and run the P algorithm. P=NP remains open; NP does not mean definitely hard or definitely outside P.

Question sources

Automata & Complexity Theory review questions · Question 43

Question 144 | NS0209

The key property of problems in class NP is:

A. They can be solved in polynomial time by a deterministic TM

B. They require exponential time to verify solutions

C. Given a potential solution, it can be VERIFIED in polynomial time

D. They require polynomial space but exponential time

Hint

Separate resource bounds, certificate verification and reduction requirements. Use proved containments without assuming unresolved class separations.

Answer C: Polynomial-time verification of polynomial-size yes-certificates.

AUT34 equates NP with polynomially verifiable yes-instances using polynomial-size certificates. A verifier must not certify no-instances. Polynomial-time deterministic solving defines P and is not known for every NP problem; exponential verification or execution bounds are not part of NP's defining certificate condition.

Why each choice is right or wrong

A - Incorrect: Deterministic polynomial-time solution defines P; whether it covers all NP is unresolved.

B - Incorrect: NP verification is polynomial for an appropriate certificate rather than requiring exponential time.

C - Correct: A polynomial-size yes-certificate can be checked in polynomial time, the equivalent NP definition.

D - Incorrect: NP does not require exponential running time; it includes the polynomial-time languages.

Primary note proof

Study-note section AUT34

> Complexity classes here contain decision languages. P consists of deterministic polynomial-time decidable languages; NP consists of nondeterministic polynomial-time decidable languages, equivalently polynomially verifiable yes-instances with polynomial-size certificates. A verifier must accept some certificate exactly for yes-inputs; no certificate may falsely establish a no-instance. NP means nondeterministic polynomial, not non-polynomial.

Study-note section AUT34

> P⊆NP because a verifier can ignore its certificate and run the P algorithm. P=NP remains open; NP does not mean definitely hard or definitely outside P.

Question sources

Automata & Complexity Theory review questions · Question 44

Question 145 | NS0210

If a problem is NP-COMPLETE, it means:

A. It is in NP but not in P

B. It is the easiest problem in NP

C. It is NP-hard but might not be in NP

D. It is both in NP AND NP-hard; every problem in NP reduces to it in polynomial time

Hint

Separate resource bounds, certificate verification and reduction requirements. Use proved containments without assuming unresolved class separations.

Answer D: Both NP membership and NP-hardness under polynomial reductions.

AUT36 defines NP-complete as being in NP and hard for every NP language under polynomial reduction. It does not assert exclusion from P, since P versus NP is open. NP-hard alone need not even be decidable, while complete problems are the hardest in NP under the stated reduction, not its easiest members.

Why each choice is right or wrong

A - Incorrect: NP-completeness does not prove exclusion from P; that would settle the open P-versus-NP question.

B - Incorrect: Completeness concerns hardness for the whole class, not easiest membership.

C - Incorrect: This describes NP-hardness without the required NP-membership condition.

D - Correct: Both membership and polynomial-reduction hardness for every NP problem are required.

Primary note proof

Study-note section AUT36

> X is NP-hard if every $A \in NP$ satisfies $A \leq_p X$. It need not belong to NP and can even be undecidable. X is NP-complete if $X \in NP$ and X is NP-hard. NP-complete languages lie inside NP and can be decided by finite exhaustive certificate search in exponential time. They are not known to require exponential time.

Question sources

Automata & Complexity Theory review questions · Question 45

Question 146 | NS0211

Which is an established reason for the foundational significance of Boolean satisfiability (SAT) in NP-completeness theory?

A. It was the FIRST problem proven NP-complete (Cook-Levin theorem); every NP problem reduces to SAT

B. It can be solved in polynomial time

C. It is in class P

D. It requires exponential space to verify

Hint

Distinguish a foundational hardness proof, efficient certificate verification and an unproved efficient solving algorithm.

Answer A: It was the FIRST problem proven NP-complete (Cook-Levin theorem); every NP problem reduces to SAT

The Cook-Levin theorem made SAT the first problem proved NP-complete: its proof encodes an arbitrary nondeterministic polynomial-time computation as a Boolean formula. Hence every NP decision problem has a polynomial-time reduction to SAT. This starts the chain of later hardness reductions. A satisfying assignment can be checked in polynomial time. Whether SAT itself has a deterministic polynomial-time algorithm is unresolved, so neither membership in P nor such an algorithm is an established reason.

Why each choice is right or wrong

A - Correct: Cook-Levin established the first NP-complete problem and reductions from every problem in NP.

B - Incorrect: A general deterministic polynomial-time SAT algorithm is not established; one would imply P=NP.

C - Incorrect: SAT membership in P is unresolved, so this is not an established statement.

D - Incorrect: An assignment can be verified in polynomial time, without requiring exponential verification space.

Primary note proof

Study-note section AUT-S03

> Boolean satisfiability was the first problem proved NP-complete, through the Cook-Levin theorem. Its hardness proof starts directly from an arbitrary nondeterministic polynomial-time computation and constructs a Boolean formula that is satisfiable exactly when an accepting computation exists.

Study-note section AUT-S03

> This starting point matters: once one NP-complete problem is available, later hardness proofs can reduce that known problem to a new target. Membership in NP remains a separate

obligation. Checking a satisfying assignment is efficient; whether finding one is always polynomial-time remains unresolved.

Study-note section AUT36

SAT asks whether a Boolean formula has a satisfying assignment; Cook-Levin establishes SAT NP-complete. CNF-SAT and 3-SAT are NP-complete versions.

Question sources

Automata & Complexity Theory review questions · Question 46

Question 147 | NS0212

Which containment chain is established unconditionally, without assuming answers to the open equalities among these classes? Here EXP means EXPTIME.

A. NP ⊆ P ⊆ PSPACE ⊆ EXP

B. P ⊆ NP ⊆ PSPACE ⊆ EXP

C. PSPACE ⊆ NP ⊆ P ⊆ EXP

D. EXP ⊆ PSPACE ⊆ NP ⊆ P

Hint

Separate resource bounds, certificate verification and reduction requirements. Use proved containments without assuming unresolved class separations.

Answer B: P⊆NP⊆PSPACE⊆EXPTIME.

The established resource-class chain is P⊆NP⊆PSPACE⊆EXPTIME. These subset relations need not all be strict: the adjacent equalities named in the note remain unresolved. A reverse relation such as NP⊆P cannot therefore be selected as an unconditional theorem. Reversing the entire chain would additionally put EXPTIME inside P, contradicting the known strict separation P⊊EXPTIME. The revised scope asks for proved inclusion, not for a speculative equality.

Why each choice is right or wrong

A - Incorrect: This includes NP⊆P, which would settle the open P-versus-NP question rather than state a known unconditional chain.

B - Correct: P⊆NP⊆PSPACE⊆EXPTIME is the established chain.

C - Incorrect: This requires reverse containments that are not established unconditionally.

D - Incorrect: Reversing the entire chain would imply EXPTIME⊆P, contradicting the known strict time-hierarchy separation.

Primary note proof

Study-note section AUT39

> Use L⊆NL⊆P⊆NP⊆PSPACE⊆EXPTIME⊆EXPSPACE. Here L/NL use logarithmic workspace; EXPTIME is ⋃kTIME(2^(n^k)), an upper-bound class that also contains P, not a statement that every member requires exponential time.

Study-note section AUT39

> P versus NP, NP versus PSPACE and PSPACE versus EXPTIME remain unresolved. coNP contains complements of NP languages; P⊆NP∩coNP, and NP=coNP is open.

Question sources

Automata & Complexity Theory review questions · Question 47

Question 148 | NS0213

If problem A is REDUCIBLE to problem B in polynomial time ($A \leq_p B$), what can we conclude?

A. A is at least as hard as B

B. A and B are in the same complexity class

C. B is at least as hard as A; an efficient algorithm for B gives an efficient algorithm for A

D. B can be solved in the same time as A

Hint

Separate resource bounds, certificate verification and reduction requirements. Use proved containments without assuming unresolved class separations.

Answer C: B is at least as hard as A; solving B efficiently solves A efficiently.

AUT35 defines A≤pB by a polynomial-time transformation preserving membership. Compute that transformation and run B's algorithm to solve A. Therefore the target B is at least as hard under this reduction. The reduction does not force equal complexity classes, a reverse reduction or an algorithm for B from A alone.

Why each choice is right or wrong

A - Incorrect: This reverses the hardness direction implied by reducing A to B.

B - Incorrect: A one-way reduction does not place both languages in the same complexity class.

C - Correct: The polynomial mapping followed by a fast B solver gives a fast A solver.

D - Incorrect: A solver for A alone does not invert the reduction to provide a B solver.

Primary note proof

Study-note section AUT35

> A≤pB means f is computable in polynomial time and x∈A iff f(x)∈B. Solving B then solves A, so B is at least as hard under this reduction. If B∈P then A∈P; if A is NP-hard then B is NP-hard. A∈P does not force B∈P, and the reduction alone does not place both problems in the same class. Polynomial reductions compose.

Question sources

Automata & Complexity Theory review questions · Question 48

Question 149 | NS0214

For standard finite encoded instances, how is the exact optimization task of finding a shortest tour visiting every city once and returning to the start classified?

A. P - solvable efficiently

B. PSPACE-complete

C. Not a formal computational problem

D. NP-hard; this classification does not establish an exponential-time lower bound.

Hint

Separate resource bounds, certificate verification and reduction requirements. Use proved containments without assuming unresolved class separations.

Answer D: NP-hard; this classification does not establish an exponential-time lower bound.

Finding the shortest tour is an optimization problem and is NP-hard. The related question whether a tour of cost at most a given bound exists is a decision problem and is NP-complete. These statements do not prove that every exact algorithm must take exponential time or resolve P versus NP. The option retains the established hardness classification without overstating a lower bound.

Why each choice is right or wrong

A - Incorrect: A general polynomial-time exact algorithm is not the classification established in the note.

B - Incorrect: PSPACE-complete is not the supplied classification of this optimization task.

C - Incorrect: The finite encoded task is a well-defined computational optimization problem.

D - Correct: Shortest-tour optimization is NP-hard; hardness alone gives no exponential lower bound.

Primary note proof

Study-note section AUT36

> TSP decision asks whether a tour of cost at most B exists; it is NP-complete for standard finite encoded inputs. Finding the shortest tour is optimization and NP-hard; call it NP-complete only after specifying a decision formulation.

Question sources

Automata & Complexity Theory review questions · Question 49

Question 150 | NS0215

Which statement about P vs NP is CORRECT?

A. It has been proven that P ≠ NP

B. It is an UNSOLVED open question - one of the most important problems in computer science

C. It has been proven that P = NP

D. The question is irrelevant since all real-world problems are in P

Hint

Separate resource bounds, certificate verification and reduction requirements. Use proved containments without assuming unresolved class separations.

Answer B: P versus NP remains open.

AUT34 explicitly says P=NP remains open, and AUT39 lists the separation among unresolved questions. Therefore neither equality nor inequality is proved in this source. The inclusion P⊆NP is known, but does not decide equality. Real-world problem encodings and resource requirements do not make the theoretical question irrelevant.

Why each choice is right or wrong

A - Incorrect: The note states that the separation is unresolved, not proved.

B - Correct: Whether P and NP are equal is an open question.

C - Incorrect: The note does not establish equality and explicitly calls it unresolved.

D - Incorrect: The note gives hard and undecidable problems, refuting the claim that all real-world problems fall in P.

Primary note proof

Study-note section AUT34

> Complexity classes here contain decision languages. P consists of deterministic polynomial-time decidable languages; NP consists of nondeterministic polynomial-time decidable languages, equivalently polynomially verifiable yes-instances with polynomial-size certificates. A verifier must accept some certificate exactly for yes-inputs; no certificate may falsely establish a no-instance. NP means nondeterministic polynomial, not non-polynomial.

Study-note section AUT34

> P⊆NP because a verifier can ignore its certificate and run the P algorithm. P=NP remains open; NP does not mean definitely hard or definitely outside P.

Study-note section AUT39

> Use L⊆NL⊆P⊆NP⊆PSPACE⊆EXPTIME⊆EXPSPACE. Here L/NL use logarithmic workspace; EXPTIME is ⋃kTIME(2^(n^k)), an upper-bound class that also contains P, not a statement that every member requires exponential time.

Study-note section AUT39

P versus NP, NP versus PSPACE and PSPACE versus EXPTIME remain unresolved. coNP contains complements of NP languages; P⊆NP∩coNP, and NP=coNP is open.

Question sources

Automata & Complexity Theory review questions · Question 50

Question 151 | NS0287

Which automaton is powerful enough to recognize the language L = {$a^n b^n c^n$ | n ≥ 1}?

A. DFA

B. Linear Bounded Automaton (LBA)

C. NFA

D. Pushdown Automaton (PDA)

Hint

Respect the quantifier order and length constraints. A valid nonmembership argument must address every allowed decomposition, not one convenient split.

Answer B: A linear bounded automaton.

An LBA checks block order then repeatedly marks one a, one b and one c, rejecting unmatched symbols. The language is not context-free, ruling out a PDA and therefore finite automata too.

Why each choice is right or wrong

A - Incorrect: A DFA recognizes only regular languages, too weak for this non-CFL language.

B - Correct: An LBA can match the three counts by marking symbols within the bounded input region.

C - Incorrect: NFA power equals DFA power and cannot reach this context-sensitive non-CFL language.

D - Incorrect: A PDA recognizes CFLs, whereas this equal-three-count language is not context-free.

Primary note proof

Study-note section AUT25

A context-sensitive rule αAβ→αγβ replaces A in context, with γ nonempty. Equivalently, noncontracting grammar presentations preserve/increase total length, with the isolated ε exception. The condition is on the complete sides, not an unrelated comparison |β|≥|α|.

Study-note section AUT25

A linear bounded automaton (LBA) is a TM with O(n) tape cells on length-n input; the standard CSL characterization uses nondeterministic LBAs. For $a^n b^n c^n$ first verify a*b*c* order, repeatedly mark one unmarked a, b and c, and reject missing/excess symbols.

Study-note section AUT25

Thus every CSL is decidable.

Study-note section AUT24

If L is context-free, some p ensures each sufficiently long $s \in L$ has $s=uvxyz$ with $|vxy| \le p$, $|vy| \ge 1$ and $uv^ixy^iz \in L$ for every $i \ge 0$. The two pieces v and y are pumped together; either may be empty, not both. A non-CFL proof must defeat every allowed decomposition.

Study-note section AUT24

For $s=a^pb^pc^p$, the window vxy cannot contain both an a and a c. Pumping affects at most two blocks and at least one count. Use i=0 or 2: the unaffected third block retains p while another count changes, or symbol order breaks. Hence $a^nb^nc^n$ is not CFL.

Question sources

Computer Science Exit Exam · 2018 · Question 4

Question 152 | NS0309

Which of the following statements is TRUE?

A. Every regular language is context-free.

B. Every context-free language is regular.

C. Every context-sensitive language is regular.

D. Every recursively enumerable language is recursive

Hint

Compare production restrictions, memory access and language-family containments. Distinguish an exact characterization from an included subclass.

Answer A: Every regular language is context-free.

AUT05 gives REG⊊CFL⊊CSL⊊DECIDABLE⊊RE, immediately establishing A. Reverse inclusions for CFL or CSL into REG fail. AUT29 explains that RE strictly exceeds decidable/recursive languages, so not every RE language is recursive. Strictness is essential to rejecting the three converse-style claims.

Why each choice is right or wrong

A - Correct: REG is a proper subfamily of CFL, so every regular language is context-free.

B - Incorrect: Matched equal counts supply CFLs that are not regular.

C - Incorrect: CSL includes nonregular and even non-CFL languages.

D - Incorrect: Some Turing-recognizable languages, such as HALT, are not recursive.

Primary note proof

Study-note section AUT05

> Type 3 regular grammars correspond to finite automata; type 2 CFGs to nondeterministic PDAs; type 1 context-sensitive/noncontracting grammars to nondeterministic LBAs; type 0 unrestricted grammars to Turing recognizers.

Study-note section AUT05

> Use REG ⊊ CFL ⊊ CSL ⊊ DECIDABLE ⊊ RE, under the usual convention allowing S→ε with S absent from right sides when needed.

Question sources

Computer Science Exit Exam · 2018 · Question 30

Question 153 | NS0322

In the CNF-first route for converting a CFG into Greibach normal form, which initial normalization is performed before the later ordering, substitution and left-recursion work?

A. Convert the grammar into CNF

B. If any production rule in the grammar is not in GNF form, convert it.

C. Convert the grammar into GNF

D. If the grammar exists left recursion, eliminate it.

Hint

Compare production restrictions, memory access and language-family containments. Distinguish an exact characterization from an included subclass.

Answer A: Convert the grammar into CNF

The specified route begins by putting the grammar into Chomsky normal form. Later ordering and substitution arrange terminal-leading productions and eliminate left recursion as required for Greibach form. CNF is an intermediate normalization in this particular route, not a universal prerequisite for every GNF construction. The other conversion choices state the goal broadly rather than identify this route's initial normalization.

Why each choice is right or wrong

A - Correct: CNF conversion is the initial normalization in the selected route.

B - Incorrect: Converting arbitrary offending rules does not identify the route's initial normalization.

C - Incorrect: GNF conversion is the overall goal, not this preliminary operation.

D - Incorrect: Left recursion must be handled later in this chosen route.

Primary note proof

Study-note section AUT20

> Some algorithms first convert to CNF; this is a method choice, not a universally mandatory first step.

Question sources

Computer Science Exit Exam · 2018 · Question 47

Question 154 | NS0336

A recognizer for a language L halts and accepts every member, but may reject or run forever on nonmembers. L is known to be undecidable. Which listed grammar family can nevertheless generate exactly L?

A. Context-free

B. Context-sensitive

C. Unrestricted (Type 0)

D. Regular

Hint

Check whether the required procedure must halt on every input, and distinguish solving a task from recognizing its yes-instances.

Answer C: Unrestricted (Type 0)

The recognizer accepts every member, so L is Turing-recognizable even though no decider exists for it. Type 0 unrestricted grammars generate exactly the Turing-recognizable languages and can therefore generate L. Regular, context-free and context-sensitive languages are all decidable under the stated hierarchy, so none of those grammar families can generate this known undecidable language. The distinction is recognition versus guaranteed decision, not running speed.

Why each choice is right or wrong

A - Incorrect: Every context-free language is decidable, whereas the given L is not.

B - Incorrect: Every context-sensitive language is decidable, excluding the stated L.

C - Correct: Type 0 grammars correspond to Turing recognizers, including recognizable undecidable languages.

D - Incorrect: Every regular language is decidable, whereas the given L is not.

Primary note proof

Study-note section AUT29

> A decider halts on every input and accepts exactly its language; these languages are recursive/decidable. A recognizer must halt and accept every member, but on a nonmember may halt-reject or loop. Its languages are recursively enumerable (RE), Turing-recognizable or semidecidable. Looping is nonacceptance, not a rejecting halt. An undecidable language may still have a recognizer.

Study-note section AUT29

> Every decidable language is RE, and the containment is strict.

Study-note section AUT05

Type 3 regular grammars correspond to finite automata; type 2 CFGs to nondeterministic PDAs; type 1 context-sensitive/noncontracting grammars to nondeterministic LBAs; type 0 unrestricted grammars to Turing recognizers.

Study-note section AUT05

Use REG ⊊ CFL ⊊ CSL ⊊ DECIDABLE ⊊ RE, under the usual convention allowing S→ε with S absent from right sides when needed.

Question sources

Computer Science Exit Exam · 2018 · Question 66

Question 155 | NS0353

ou have given that Grammar G1 −({S, A, B}, {a, b}, S, {S → AB, A → a, B → b}).
Which one is terminal symbol in the give?

A. a,b

B. S

C. S → AB, A → a, B → b

D. S,A,B

Hint

Distinguish symbols, finite strings, their lengths and sets of strings. Apply the operation to its stated operands, including empty cases.

Answer A: a and b are the terminal symbols.

AUT04 separates the variable set from the terminal alphabet in G. The stem explicitly declares {S,A,B} as variables and {a,b} as terminals. The production rules are rewrites, not symbols, and uppercase S is the start variable. Therefore a,b is the terminal-symbol choice without needing any derivation calculation.

Why each choice is right or wrong

A - Correct: These are exactly the two symbols in the supplied terminal alphabet {a,b}.

B - Incorrect: S is the start variable rather than a terminal.

C - Incorrect: These are production rules, not terminal symbols.

D - Incorrect: S,A,B is the declared variable set rather than the terminal alphabet.

Primary note proof

Study-note section AUT04

> Write G=(V,Σ,P,S): finite variables V, terminals Σ disjoint from V, productions P and start variable S∈V. A production replaces its left side by its right side. The language is L(G)={w∈Σ*:S⇒*w}. A recognizer processes a word and accepts exactly the language members. Grammars generate languages; automata recognize languages, not grammars.

Study-note section AUT04

> Two grammars are equivalent when their generated languages are equal, regardless of rule counts or variable names. One common word, equal alphabets or one-way inclusion is insufficient.

Question sources

Computer Science Exit Exam · 2018 · Question 90

Question 156 | NS0356

The language $L = \{ww^R \mid w \in \{a, b\}^*\}$ (even-length palindromes) is:

A. Regular

B. Non-deterministic CFL but not deterministic

C. Deterministic CFL

D. Context-sensitive but not context-free

Hint

Compare production restrictions, memory access and language-family containments. Distinguish an exact characterization from an included subclass.

Answer B: A nondeterministic CFL that is not deterministic context-free.

AUT22 gives the NPDA for binary ww^R by guessing the midpoint and states that without a marker it is not deterministic context-free. The alphabet {a,b} is explicit here, excluding the unary regular exception. A visible center marker would allow a DPDA, but no such marker occurs in the given language.

Why each choice is right or wrong

A - Incorrect: Binary unmarked even palindromes are not regular; the unary exception does not apply.

B - Correct: An NPDA guesses the midpoint, while the unmarked binary language is not deterministic context-free.

C - Incorrect: Deterministic recognition requires a visible midpoint marker for the contrasting construction in AUT22.

D - Incorrect: The language is context-free, so denying context-freeness is false even though it is also CSL.

Primary note proof

Study-note section AUT22

> For even palindromes ww^R over {a,b}, push the first half, nondeterministically guess its midpoint, then pop matching symbols. No center marker makes that language context-free but not deterministic context-free. With a marker wcw^R the switch point is visible and a DPDA suffices. Unary even palindromes are merely even-length words and regular. A PDA cannot compare two arbitrary halves in the same forward order: $\{ww : w \in \{a,b\}^*\}$ is the copy language, not the palindrome language.

Question sources

Computer Science Exit Exam · 2018 · Question 94