CS Exit Ethiopia

Complete Study Notes

Computer Science Exit Examination

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 6

How to use these study notes

START01 A practical study cycle

Start with a small set of questions before reading.

Mark the exact thing that stopped you: an unknown term, a rule, a calculation, a trace or a choice between alternatives.

Use the question lookup to open the relevant section.

Read one section, close it, and explain the rule in your own words.

Solve the question again and say why each other choice fails.

Then change one assumption or input and solve the variation.

This checks understanding beyond recognition.

A useful 60-minute session: 10 minutes of cold questions, 20 minutes of focused reading, 20 minutes of closed-book solving and 10 minutes updating an error log.

Adjust the timing to your needs.

Revisit mistakes after a gap, for example the next day, several days later and the following week.

These intervals are a practical starting plan, not a fixed learning law.

Record the section ID, your mistaken rule, the corrected rule and one boundary case.

Keep the log short enough to review.

Mix related courses during revision: programming with data structures; databases with security; networking with administration.

Explain how the same word changes meaning across contexts.

START02 What to emphasize

Emphasize definitions, key differences, algorithm costs, SDLC models, databases, networking and security.

Use short oral questions to check understanding and correct common misconceptions.

Combine those habits with the tracing, calculation, design and diagnosis demanded by the Ministry learning outcomes.

Allocate revision by both exam share and your error log.

A course with fewer items can still deserve more time if your foundations are weak.

The blueprint is a planning baseline, not a prediction of exact questions.

Definitions: distinguish nearby concepts, not just their names.

Calculations: write units, formulas, assumptions and intermediate steps.

Code and SQL: check validity before tracing the text; state any changes before analyzing a corrected version.

Algorithms and protocols: know the input, decision rule, state change, termination and failure cases.

Design choices: state the criterion.

A choice can change when cost, reliability, consistency or latency matters more.

Course group	Items in the 100-item blueprint	Revision implication
Web Programming	9	Reserve extra practice for HTML, CSS, JavaScript, PHP, client/server behavior and safe data handling.
Data Structures and Algorithms	7	Trace operations and compare their assumptions and costs.
Each of the other fourteen courses	6 each	Study all sixteen; do not abandon a whole course based on a short note or an alleged repeated-question list.

START03 How to handle exam traps

Underline NOT, FALSE, BEST and EXCEPT before comparing choices.

Check whether the stem supplies enough facts to select one answer.

A long scenario is not automatically an analysis question, and a familiar answer letter is not evidence.

When code is invalid, a diagram is absent, or two choices express the same rule, identify the clarification needed before answering.

Learn the correct rule and the smallest clarification needed.

Do not memorize an arbitrary key as a universal fact.

Distinguish worst case from average case, and an upper bound from a tight bound.

Check language version, SQL dialect, addressing assumptions, graph weights and tie-breaking rules.

For subnetting and scheduling, calculate rather than guess from a familiar number.

Use timed mixed papers only after you can explain your errors.

Revisit the reading sections behind recurring mistakes.

Automata and Complexity Theory

Complete Study Notes

AUT01 | Symbols, alphabets, words and length

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.

Neither ε nor $\varnothing$ is the numeric symbol 0.

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

By contrast $|(caddba)^2|=12$ because the word is concatenated twice.

Distinguish $|Z|^2$, the square of a word's length, from $|Z^2|$, the length of its repeated concatenation; keep the exponent in its stated position.

Automata and Complexity Theory module

AUT02 | Concatenation, powers and reversal

Concatenation xy writes x followed by y: cddc·ababb=cddcababb; order generally matters. $\varepsilon x = x\varepsilon = x$. Define $x^0=\varepsilon$ and x^n as n copies for $n \geq 1$.

Reverse x^R reads symbols backward: caddba reverses to abddac; $(xy)^R=y^Rx^R$ and $(x^R)^R=x$.

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.

A language power is a set, so $L^0=\varepsilon$ is a type error unless braces are understood.

Symbols are case-sensitive; uppercase symbols do not equal lowercase symbols unless a conversion is explicitly specified.

Automata and Complexity Theory module

AUT03 | Language operations and empty-word edge cases

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 $\{aa,ab,ba,bb\}$.

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 $\Sigma^* - L$ and needs a fixed alphabet.

Automata and Complexity Theory module

AUT04 | Grammars generate; automata recognize

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.

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.

A regular expression denotes a language but is not itself the set of matched words.

Natural languages such as English are not the intended formal-language examples; give a precisely specified set instead.

Automata and Complexity Theory module

AUT05 | Chomsky hierarchy and the epsilon convention

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.

In an unrestricted production $\alpha \to \beta$, α contains at least one variable; unrestricted does not mean any meaningless rewrite is allowed.

Use $\text{REG} \subsetneq \text{CFL} \subsetneq \text{CSL} \subsetneq \text{DECIDABLE} \subsetneq \text{RE}$, under the usual convention allowing $S \to \varepsilon$ with S absent from right sides when needed.

Strict noncontracting grammars otherwise cannot generate ε.

Stronger families can describe lower-family languages: asking which grammar is capable is often nonunique.

Ask for the least sufficient family or the exact family equivalent to a machine model.

This is an expressiveness hierarchy, not a runtime ranking.

Automata and Complexity Theory module

AUT06 | DFA definition and acceptance

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 $\varnothing$.

State names $q_0, q_1, \ldots$ do not determine accepting status.

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.

Automata and Complexity Theory module

AUT07 | Design and trace a finite-state recognizer

Choose states with invariant meanings: the information about the consumed prefix needed to decide future acceptance.

For binary words ending 01 use q_0=no useful suffix, q_1=last suffix 0, q_2=last suffix 01; only q_2 accepts.

Transitions on (0,1) are $q_0 \to (q_1, q_0)$, $q_1 \to (q_1, q_2)$, $q_2 \to (q_1, q_0)$.

Input 10101 follows $q_0, q_0, q_1, q_2, q_1, q_2$ and accepts.

For length at least two use meanings 0 symbols, 1 symbol, at least 2 symbols.

Every input advances toward the third state, which loops and accepts.

Calling it q_2 is a naming choice: without a diagram or state definitions no numbered-state answer follows.

To verify a design, test ε, shortest accepted/rejected words and overlaps; sample agreement alone is not a proof.

Automata and Complexity Theory module

AUT08 | Minimum states: prefix, suffix and valid alphabets

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.

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.

Over a unary alphabet the sink may be unreachable.

If w contains a symbol outside Σ, the language is empty, whose minimal complete DFA has one rejecting state.

Thus starting abc over {a,b} needs one state, not five; five requires an alphabet containing a,b,c.

Automata and Complexity Theory module

AUT09 | NFA and epsilon transitions

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.

Other paths may get stuck.

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. ε is a zero-input move, not an ordinary input character or permission to accept with unread input.

Automata and Complexity Theory module

AUT10 | Subset construction with epsilon closure

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.

Example over $\Sigma=\{a\}$: states p,r, start p, final r; $\delta(p,a)=\{p,r\}$, $\delta(r,a)=\varnothing$.

The subset machine starts {p}, reaches {p,r} on a, and stays there on further a's. It accepts a^+.

With an ε edge p→r, the starting subset would already contain r and ε would also be accepted.

Automata and Complexity Theory module

Sipser MIT Lecture 2: nondeterminism and closure

AUT11 | Equivalence, minimization and diagram checks

First discard unreachable DFA states.

Partition remaining states into accepting/nonaccepting blocks; repeatedly split a block when two states transition on some symbol into different blocks.

Stable blocks are equivalence classes and can be merged.

Equivalent states have identical acceptance for every continuation; current acceptance alone is insufficient.

For DFA equivalence build pairs of states and search from the start pair.

Any reachable pair with exactly one accepting component supplies a distinguishing word; if none exists, the languages agree.

A partial diagram must first be completed consistently with a sink.

To reject a proposed RE quickly, compare ε and the shortest cycles: an accepting start plus an ab return cycle forces ε and ab into the language.

Every equivalent expression must contain both.

Automata and Complexity Theory module

Cornell CS412/413: DFA minimization

AUT12 | Regular-expression notation and precedence

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*.

(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. (0|ε)(0|1)* is simply all binary words because ε permits any suffix.

Programming regex features such as backreferences may exceed classical regular-language power; wildcard dot is an extra notation, not a basic alphabet-independent theorem.

Automata and Complexity Theory module

AUT13 | Build expressions and convert representations

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*.

RE→automaton uses ε-NFA fragments: parallel alternatives for union, connect accepting ends to the next fragment for concatenation, and add bypass/repetition ε moves for star.

Then apply subset construction.

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.

These operations preserve the whole language, not just tested examples.

Automata and Complexity Theory module

Sipser MIT Lecture 3: FA/RE and pumping

AUT14 | Regular grammars and closure constructions

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.

For right-linear grammar→NFA, variables become states, A→aB becomes an a edge, A→a leads to a fresh final state, and A→ε makes A accepting.

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

Reverse edges and exchange start/final roles using an ε-NFA to obtain reversal.

Complement requires a complete DFA; then swap accepting/nonaccepting states.

Product-state constructions implement intersection/difference.

Automata and Complexity Theory module

Sipser MIT Lecture 2: closure constructions

AUT15 | Regular pumping lemma: use the quantifiers

If L is regular, some p≥1 satisfies: every s∈L with $|s|\geq p$ has a split s=xyz, $|xy|\leq p$, $|y|\geq 1$, and $xy^iz\in L$ for every i≥0.

To disprove regularity, assume an arbitrary pumping length p, choose s, then show every allowed split can be broken by some i. Finding one bad split proves nothing; satisfying this necessary condition alone does not prove regularity.

For $L=\{a^nb^n : n\geq 0\}$, choose a^pb^p.

Every allowed $y=a^k$ with k≥1; pumping i=0 leaves fewer a's than b's. The contradiction excludes every DFA.

For an n-state DFA, an accepted word of length at least n repeats a state along its path, giving an accepted cycle that can be repeated indefinitely.

Thus a fixed DFA's language is infinite iff it has an accepted word of length at least its state count.

Automata and Complexity Theory module

AUT16 | CFG derivations, sentences and parse trees

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.

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.

Automata and Complexity Theory module

AUT17 | Design and validate context-free grammars

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 $n \geq 0$: 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.

For $a^n b^m$ with $n > m \geq 0$, use $S \to aSb \mid A$ and $A \to aA \mid a$: each S recursion adds a matched pair, then A adds at least one extra a. For balanced parentheses use $S \to (S)S \mid \varepsilon$.

Validate soundness-each rule preserves the intended property-and completeness-decompose any desired nonempty word into pieces matching a production.

Merely producing several examples does not establish exact language equality.

Automata and Complexity Theory module

AUT18 | Ambiguity and precedence

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.

E→E+E|E*E|id allows id+id*id to group as (id+id)*id or id+(id*id).

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.

Automata and Complexity Theory module

AUT19 | Simplification, CNF and membership

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

Mark nullable variables and add alternatives omitting nullable occurrences, preserving ε through a fresh start when needed.

Remove unit rules by copying nonunit productions reachable through unit chains.

Remove nongenerating variables, then unreachable symbols.

Do not delete a nullable variable's every use and lose valid nonempty words.

Chomsky normal form uses A→BC or A→a, with an optional isolated $S_0 \to \varepsilon$.

Replace terminals in long right sides by variables and split long variable sequences into binary steps: A→aBC becomes $A \to XC_1$, X→a, $C_1 \to BC$.

For CNF membership, CYK fills variables generating each substring, combining splits with A→BC.

For a fixed grammar its time is $O(n^3)$; n>0 derivations contain n terminal rules and n−1 binary rules.

Automata and Complexity Theory module

Virginia Tech OpenDSA: transforming grammars

AUT20 | Greibach form and left recursion

Greibach normal form requires $A \to aA_1 \ldots A_k$, k≥0: one leading terminal followed only by variables.

Handle ε separately through a fresh start that never appears on a right side.

A nonempty word of length n takes exactly n leftmost rule applications because each contributes one terminal to the prefix.

A constructive route removes ε/unit/useless rules, orders variables, substitutes rules beginning with earlier variables, removes left recursion, and substitutes until every rule begins with a terminal.

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

Direct $A \to A\alpha \mid \beta$ can become $A \to \beta R$, $R \to \alpha R \mid \varepsilon$, then further normalization handles ε.

GNF cannot retain direct or indirect left recursion.

A question demanding the first step must name its chosen algorithm.

Automata and Complexity Theory module

Duke CPS140: CNF and Greibach conversion

AUT21 | PDA configuration and acceptance

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.

A configuration records state, unread input and stack.

Final-state acceptance requires all input consumed and a final state; empty-stack acceptance requires all input consumed and an empty stack.

For NPDAs the modes characterize the same language family, through a construction, not necessarily the same unmodified machine.

Deterministic-PDA acceptance conventions need extra care and are not automatically interchangeable.

Automata and Complexity Theory module

AUT22 | Construct PDA recognizers and compare determinism

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.

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.

Automata and Complexity Theory module

Jeannette Wing, CMU: Formal methods for computer systems

AUT23 | CFL closure and counterexamples

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.

CFLs are not closed under arbitrary intersection.

Both $\{a^nb^nc^m\}$ and $\{a^mb^nc^n\}$ are context-free; their intersection $a^nb^nc^n$ is not.

Nor are CFLs closed under complement or arbitrary difference.

CFL minus a regular language remains CFL by intersecting with the regular complement.

A failed construction of one proposed PDA proves only that construction fails, not that every PDA fails.

Automata and Complexity Theory module

Old Dominion University CS390: CFL properties

AUT24 | CFL pumping proof and copy versus mirror

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.

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 $a^n b^n c^n$ is not CFL.

The copy language ww is also non-CFL over two symbols, whereas ww^R has the explicit grammar $S \to aSa \mid bSb \mid \varepsilon$.

An expression that writes ww and calls it a palindrome is contradictory.

Automata and Complexity Theory module

AUT25 | Context-sensitive rules and linear bounded automata

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 $|\beta| \geq |\alpha|$.

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.

All marks stay within the input region.

A fixed input has finitely many configurations, at most $|Q| m |\Gamma|^m$ for m cells.

Reachability search in that finite graph decides acceptance, even if some machine branches loop.

Thus every CSL is decidable.

Automata and Complexity Theory module

Illinois CS373: unrestricted grammars and LBAs

AUT26 | Classify languages with justified boundaries

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.

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.

Automata and Complexity Theory module

UMass, David Barrington: unary context-free languages

AUT27 | TM components, transitions and configurations

For a Turing machine (TM), use $M=(Q,\Sigma,\Gamma,\delta,q_0,q_{accept},q_{reject})$, 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.

$\delta(q,a)=(p,b,L)$ 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.

In ID u q av the head scans a; the left-move successor, when u=u'c, is u' p cbv.

For a right move it is ub p v, showing a blank if the head enters unused tape.

State and transition names must follow the diagram's declared convention.

Automata and Complexity Theory module

AUT28 | TM design, variants and computation limits

To decide $a^nb^nc^n$, repeatedly mark matching triples, resetting the head between scans, then verify no unmatched symbol remains.

Each round marks additional input cells, so finitely many rounds suffice.

To decide ww, reject odd length, locate the midpoint using marks/counters, and compare corresponding symbols of the two halves; writable tape permits access a single stack lacks.

State both the acceptance test and why every input halts.

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.

Automata and Complexity Theory module

AUT29 | Deciders, recognizers and complements

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.

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

If L and its complement are both RE, simulate their recognizers in dovetailed steps until one accepts, obtaining a decider.

Conversely complementing a decider works because it always halts.

Flipping final states in an arbitrary recognizer fails to handle nonmembers that loop.

Recursive function theory's total computable functions are not merely programs written with recursive self-calls; partial computable functions may be undefined where computation does not halt.

Automata and Complexity Theory module

AUT30 | Halting, diagonalization and Rice’s theorem

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.

Rice's theorem makes every nontrivial semantic property of a TM's recognized language undecidable.

Nontrivial means some recognized languages have it and some do not.

Empty language, contains ε and at least three accepted words qualify.

Number of states and whether a particular symbol is written are machine-behavior/syntactic properties, not automatically language properties; prove such claims separately.

Undecidable means no general decider exists, not merely no algorithm is currently known.

Automata and Complexity Theory module

AUT31 | Which decision questions are solvable?

DFA/NFA word membership is decidable by simulation or reachable state sets.

FA emptiness is reachable-accepting-state search; DFA equivalence searches the product for an acceptance disagreement.

CFG membership is decidable by parsing/CYK and emptiness by marking variables that can generate terminal strings.

These procedures halt and are not the same as deciding arbitrary TM membership.

TM acceptance on input is RE-complete and undecidable; HALT is undecidable.

General CFG ambiguity and CFG equivalence are undecidable even though membership for a given CFG and word is decidable.

A concrete grammar's two parse trees can establish that particular grammar is ambiguous; this does not give an algorithm deciding ambiguity for every grammar.

FORTRAN/programming-language questions must distinguish CFG-style syntax, semantic restrictions and Turing-complete computation; a language's name alone does not choose one formal classification.

Automata and Complexity Theory module

Sipser MIT Lecture 7: decision problems

AUT32 | Computable reductions for undecidability

A many-one reduction $A \leq_m B$ is a total computable transformation f satisfying $x \in A$ iff $f(x) \in B$. If B had a decider, compute f and invoke it to decide A. Therefore known-undecidable A reduced to B proves B undecidable.

Showing $B \leq_m A$ alone does not establish B's hardness.

Example $HALT \leq_m A_{TM}$: given $\langle M,w \rangle$, construct N that ignores its input, simulates M(w), and accepts if M ever halts, including rejecting halts.

Then $\langle N,\varepsilon \rangle$ is accepted iff M(w) halts.

Construction of N's finite description terminates; the simulation inside N may not.

Reductions require both yes and no directions and a computable mapping, not just a resemblance between two problems.

Automata and Complexity Theory module

Sipser MIT Lecture 9: reducibility

AUT33 | Measure resources with the right input size

Time counts computation steps; space counts cells/memory used, specifying whether read-only input is excluded.

For a deterministic decider use the maximum over all length-n inputs; for a time-bounded nondeterministic decider every branch meets the bound.

Big-O is an asymptotic upper bound, not inherently a worst-case label: it can describe a worst-, average- or best-case function. Θ adds a matching lower bound.

Growth typically orders $1, \log n, n, n \log n, n^2, 2^n, n!$.

A fixed polynomial n^k differs from exponent 2^n.

Encoding matters: the number N in binary has Θ(log N) bits, so O(N) work is not polynomial in its binary input length.

Pseudopolynomial algorithms depend on numeric magnitude rather than encoding length.

A polynomial of enormous degree need not be practical.

Automata and Complexity Theory module

AUT34 | P, NP and polynomial verification

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.

For CLIQUE, the certificate lists k vertices; verify they are distinct and every pair is an edge.

This is polynomial in the graph encoding.

$P \subseteq NP$ 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.

Automata and Complexity Theory module

Sipser MIT Lecture 14: P, NP, certificates

Clay Mathematics Institute: P versus NP

AUT35 | Polynomial reductions: direction and consequences

$A \leq_p B$ 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.

To prove a new target X hard, reduce a known hard problem Y to X, not X to Y. The mapping and target instance must have polynomial size.

For undecidability, general computable reductions suffice; for NP-completeness, an arbitrarily slow computable transformation is insufficient.

Always state which reduction and resource bound are being used.

Automata and Complexity Theory module

AUT36 | NP-hard versus NP-complete; decision versus optimization

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.

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.

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.

Fixed 2-colorability is in P, while fixed k-colorability for $k \geq 3$ is NP-complete.

Automata and Complexity Theory module

AUT37 | Work a hardness proof: 3-SAT to CLIQUE

First CLIQUE belongs to NP by the pairwise-edge certificate check.

For a 3-CNF formula with m clauses, make one vertex for each literal occurrence.

Connect two vertices iff they come from different clauses and are not complementary literals.

Ask whether the graph has a clique of size m. Construction takes polynomial time and space.

A satisfying assignment supplies one true literal per clause; they are mutually consistent and form a clique.

Conversely an m-clique selects one occurrence per clause, with no variable chosen both positively and negatively, so assign those literals true and extend remaining variables arbitrarily.

Both implications prove equivalence.

For $(x_1 \vee \neg x_2) \wedge (\neg x_1 \vee x_2 \vee x_3) \wedge \neg x_3$, the assignment (true,true,false) satisfies all clauses because ¬false is true.

Automata and Complexity Theory module

Sipser MIT Lecture 15: NP-completeness

AUT38 | PSPACE, quantifiers and space reuse

PSPACE is $\bigcup_k \text{SPACE}(n^k)$, 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.

Savitch's theorem gives $\text{NSPACE}(s) \subseteq \text{DSPACE}(s^2)$ for appropriate $s \geq \log n$, hence NPSPACE=PSPACE.

It does not prove P=NP.

TQBF asks whether a fully quantified Boolean formula is true and is PSPACE-complete.

Evaluate ∃x by trying one branch or the other; evaluate ∀x by requiring both.

Depth-first evaluation stores assignments and recursion state using polynomial space despite potentially exponential time.

For example $\forall x \exists y (x \leftrightarrow y)$ is true, but $\exists y \forall x (x \leftrightarrow y)$ is false.

PSPACE-complete means membership plus hardness for every PSPACE language under the stated reduction.

Automata and Complexity Theory module

Sipser MIT Lecture 17: PSPACE and Savitch

AUT39 | Known hierarchy and open separations

Use L⊆NL⊆P⊆NP⊆PSPACE⊆EXPTIME⊆EXPSPACE.

Here L/NL use logarithmic workspace; EXPTIME is $\bigcup_k \text{TIME}(2^{(n^k)})$, an upper-bound class that also contains P, not a statement that every member requires exponential time.

Suitable hierarchy theorems prove P⊊EXPTIME and PSPACE⊊EXPSPACE.

They do not establish every adjacent containment strictly.

P versus NP, NP versus PSPACE and PSPACE versus EXPTIME remain unresolved. coNP contains complements of NP languages; $P \subseteq NP \cap coNP$, and NP=coNP is open.

If any NP-complete language is in P, all NP collapses to P. This does not make every decidable language polynomial or make undecidable problems decidable.

Avoid blanket claims about all cryptography or all problems becoming easy; the class and encoding matter.

Automata and Complexity Theory module

Sipser MIT Lecture 21: hierarchy theorems

AUT40 | Derive languages from grammar productions

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.

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.

Automata and Complexity Theory module

AUT41 | Specify languages, states and start symbols

Use the exact readable language formula and alphabet when testing membership.

State labels are arbitrary: use the transition diagram, initial state and accepting markings, or a definition assigning meanings to states.

Language theory alone cannot identify an arbitrary state label.

An accepting start state and an ab return cycle require a corresponding regular expression to include both ε and ab.

Declare the start variable before testing grammar membership; with A assumed in the grammar above, none of the four candidate words qualifies.

Distinguish the non-CFL copy language ww from the even-palindrome language ww^R.

Generic capable, contains and most difficult questions need a least-family, exact-class or reduction qualifier.

For GNF first-step questions identify the conversion algorithm.

Evaluate each description against the stated language and machine conventions.

Automata and Complexity Theory module

AUT-S01 | Computability, complexity and abstract machines

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.

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.

Automata and Complexity Theory module · page 15

Automata and Complexity Theory module · page 2

Automata and Complexity Theory module · page 16

AUT-S02 | The stack discipline and its queue contrast

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.

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.

Data Structure and Algorithms module · page 6

Data Structure and Algorithms module · page 7

Oracle Java SE 17: Deque stack and queue semantics

AUT-S03 | Why satisfiability was the first NP-complete problem

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.

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.

University of Toronto CSC373: Complexity, slide 28

AUT-S04 | Finite procedures are different from undecidable tasks

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.

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.

Data Structure and Algorithms module · page 11

MIT 18.06: Gaussian elimination

MIT OpenCourseWare - Solving Ax = b: row reduced form R