The final exam will take place during the officially scheduled period, 1:30-4:30pm on Tuesday, May 7.

The exam will cover material from the entire course, with some greater emphasis on material covered since the last exam (sections 4.3–4.4, 4.6, and 5.1–5.3 — parsing, parse trees, pushdown automata, general grammars, Turing machines, and computability). It will be a bit longer than the midterms but is intended to take most people less than two hours.

Many of the questions on the exam will be similar to problems on the homeworks and previous exams. There many be some "short essay" questions that ask you to define something, or discuss something, or explain something, and so on. There may be some proofs but there won't be any questions about logic circuits or about real computers (Java, BNF, regular expressions on computers), though you may want to address these things in your response to the final essay question.

The last question will be an essay question asking you to bring together various things that have been covered in the course and to reflect on what you have learned about the nature of computation. You should be prepared to write about things like logic and logic circuits, different types of automata and languages, and ideas about computability and uncomputability.

A reference for propositional logic rules, the rules of deduction, and any definitions needed in proofs (similar to what was provided on the first exam) will be provided as needed. In addition, you may use one page of notes (one side of an 8.5×11 sheet of paper) on the exam. This page should be personally prepared by you (not copied from a friend) and will be handed in with your exam. Preparing notes is a useful study and review activity if you engage with the process of creating it — you are encouraged to handwrite or type elements from scratch rather than copying and pasting.

Terms and ideas from the last part of the course that you should be familiar with:

- parsing
- parse tree
- left derivations and right derivations
- how a parse tree corresponds to a derivation
- pushdown automaton
- transition diagram of a pushdown automaton
- definition of a pushdown automaton as a list of six things $(Q, \Sigma, \Delta, q_0, \partial, F)$ and what each thing means
- what it means for a pushdown automaton to accept a language
- deterministic pushdown automaton
- deterministic context-free language
- general grammars

- derivations using a general grammar
- how a general grammar generates a language
- Turing machine
- transition diagram for a Turing machine; table of rules for a Turing machine
- how a Turing machine operates
- how a Turing machine computes a function
- evidence that Turing machines are general-purpose computing devices
- Turing-acceptable language, also known as a recursively enumerable language
- Turing-decidable language, also known as a recursive language
- a language L is recursive if and only if both L and L are recursively enumerable
- a language is recursively enumerable if and only if it is generated by a grammar
- the language hierarchy: finite, regular, context-free, recursive, recursively enumerable
- the Church-Turing Thesis
- Turing machines (up to equivalence) can be assigned numbers: T_0, T_1, T_2, T_3, ...
- the set $K = \{\ n \mid T_n \text{ halts when run with input } n\ \}$
- the set K is recursively enumerable, but it is not recursive
- the set K is not recursively enumerable
- the Halting Problem and its computational unsolvability
- the nature of computation

Things from earlier in the course that you should be familiar with:

- translations from logic to English, and from English to logic
- propositional logic; the logical operators "and" ($\wedge$), "or" ($\vee$), and "not" ($\neg$)
- truth table
- logical equivalence ($\equiv$)
- the conditional ("implies") operator ($\rightarrow$)
- the negation of $p \rightarrow q$ is $p \wedge \neg q$
- the biconditional operator ($\leftrightarrow$)
- tautology
- some basic laws of Boolean algebra (double negation, DeMorgan's, commutative, etc.)
- converse of an implication
- contrapositive of an implication
- predicate logic; quantifiers, "for all" ($\forall$) and "there exists" ($\exists$)
- negation of a statement that uses quantifiers (DeMorgan's laws for predicate logic)
- the difference between $\forall x \exists y$ and $\exists y \forall x$
- arguments, valid arguments, and deduction
- formal proof of the validity of an argument
- Modus Ponens and Modus Tollens
- direct proof (to prove $P \rightarrow Q$, assume P is true, and prove Q)
- proof by contradiction

- sets
- set notations: $\{a, b, c\}$, $\{1, 2, 3, \ldots\}$, $\{\ x \mid P(x)\ \}$, $\{\ x \in A \mid P(x)\ \}$
- the empty set, $\emptyset$ or $\{\ \}$
- element of a set: $a \in A$
- subset: $A \subseteq B$
- union, intersection, and set difference: $A \cup B$, $A \cap B$, $A \setminus B$
- definition of set operations in terms of logical operators
- power set of a set: $\mathcal{P}(A)$
- universal set; complement of a set (in a universal set): $\overline{A}$
- ordered pair: (a, b); cross product of sets: $A \times B$
- one-to-one correspondence
- infinite set
- countably infinite set (in one-to-one correspondence with $\mathbb{N}$)
- uncountable set (that is, uncountably infinite)
- for any set A, there is no one-to-one correspondence between A and $\mathcal{P}(A)$
- alphabet (finite, non-empty set of "symbols")
- string over an alphabet Σ
- string operations: length ($|w|$), concatenation (xy), reverse (w^R), w^n, $n_\sigma(w)$
- language over an alphabet Σ (a subset of Σ^*)
- the set of strings over Σ is countable; the set of languages over Σ is uncountable
- set operations on languages: union, intersection, set difference, and complement
- concatenation (LM), power (L^n), and Kleene star (L^*) of languages
- regular expression over an alphabet Σ
- regular language; the language $L(r)$ generated by a regular expression r
- DFA (Deterministic Finite Automaton)
- transition diagram of a DFA
- definition of a DFA as a list of five things $(Q, \Sigma, q_0, \delta, F)$ and what each thing means
- what it means for a DFA to accept a language
- nondeterminism; NFA (Non-deterministic Finite Automaton)
- what it means for an NFA to accept a string
- the language, $L(M)$, accepted by a DFA or NFA
- algorithm for converting an NFA to an equivalent DFA
- algorithm for converting a regular expression to an equivalent NFA
- DFAs, NFAs, and regular expressions all define the same class of languages
- CFGs (Context-Free Grammars); context-free language
- definition of a CFG as a 4-tuple $G = (V, \Sigma, P, S)$
- derivation of a string from the start symbol of a CFG