- Inductively defined sets: lists (and hence natural numbers and strings) and trees are inductively defined. Inductive proofs and recursive definitions of functions are possible on inductively defined sets.

- Automata Theory the essence of computation is decision procedures over strings. Simple automata can be imagined and programed to make decisions and useful problems can be solved this way.

- Deterministic Finite Automata: A DFA has a finite set of states and a finite set of input symbols.

  Gradiance Question ID #74, #75, #76.

- Transition Diagrams: It is convenient to represent automata by a graph in which the nodes are the states, and the arcs are labeled by input symbols, indicating the transitions of that automaton. The start state is designated by an arrow, and the accepting states by double circles.

- Language of an Automaton: The automaton accepts strings. A string is accepted if, starting in the state state, that transitions caused by processing the symbols of that string one-at-a-time lead to an accepting state.

- Nondeterministic Finite Automata: The NFA differs from the DFA in that the NFA can have any number of transitions (including zero) to next states from a given state on a given input.

- The Subset Construction: By treating sets of states of an NFA as states of a DFA, it is possible to convert any NFA to a DFA that accepts the same language.

- ε-Transitions: We can extend the NFA by allowing transitions on an empty input, i.e., no input symbol at all. These extended NFA's can be converted to DFA's accepting the same language.

- Regular Expressions: This algebraic notation describes exactly the same languages as finite automata: the regular languages. The regular-expressions operators are union, concatenation (or "dot"), and closure (or "star").

  Gradiance Question ID #121, #122.

- Equivalence of Regular Expressions and Finite Automata: We can convert a DFA to a regular expression by an inductive construction in which expressions for the labels of paths are allowed to pass through increasingly larger sets of states are constructed. Alternatively, we can use a state-elimination procedure to build the regular expression for a DFA. In the other direction, we can construct recursively an ε-NFA from regular expressions, and then convert the ε-NFA to a DFA, if we wish.

  Gradiance Question ID #77, #84 [problem 3.2, page 124], #86 [problem 3.4, page 124], #123 [problem 3.9, page 125].

- The Algebra of Regular Expressions: Regular expressions obey many of the algebraic laws of arithmetic, although there are differences. Union and concatenation are associative, but only union is commutative. Concatenation distributes over union. Union is idempotent.

  Gradiance Question ID #87 [problem 3.5, page 124], #114 [problem 3.6, page 124].

- The Pumping Lemma: If a language is regular, then every sufficiently long string in the language has a nonempty substring that can be "pumped," that, repeated any number of times while the resulting strings are also in the language. This fact can be used to prove that many different languages are *not* regular.

  Gradiance Question ID #88, #91 (requires CFL).

- Operations That Preserve the Property of Being a Regular Language: There are many operations that, when applied to regular languages, yield a regular language as a result. Among these are union, concatenation, closure, intersection, complementation, difference, reversal, homomorphism (replacement of each symbol by an associated string), and inverse homomorphism.

  Gradiance Question ID #124.

- Testing Emptiness of Regular Languages: There is an algorithm that given a representation of a regular languages, such as an automaton or regular expression, tells whether or not the represented language is the empty set.

- Testing Membership of Regular Languages: There is an algorithm that, given a string and a representation of a regular language, tells whether or not the string is in the language.

Classic Test Questions:

- Exercise 2.2.4, page 53.
- Exercise 2.3.2, page 66.
- Exercise 2.4.1, page 71–72.
- Exercise 2.5.1, page 79.
- Exercise 3.1.1, page 91.
- Exercise 3.1.3, page 92.
- Exercise 3.2.1, page 107.
- Exercise 4.1.1, page 131.