The exam will be held in the Testing Center from August 10 to 11. Note that the Testing Center closes at 2:00 on August 11.

The homeworks are great preparation. Pick up your graded homeworks from the TAs and understand any mistakes you made.

Finite State Machines
- Be able to define a finite state machine to recognize a given language and to describe the language recognized by a given finite state machine.
- Be able to convert from a diagram of a finite state machine to its formal definition and vice versa.

Grammars
- Know the difference between Simple $LL(1)$, $LL(1)$ without $\epsilon$, $LL(1)$, and context free grammars. Be able to specify whether or not a given grammar is any of these.
- Be able to produce strings from a grammar and describe the language produced by a grammar.
- Be able to create an $LL(1)$ grammar with tails using $\epsilon$-rules (for numbers, parameter lists, variable names, etc.).
- Be able to recognize problems in a grammar, such as ambiguity, incorrect precedence, incorrect associativity, unreachable rules, unproductive non-terminals, etc.
- Understand the $FIRST$ function, be able to compute it for a given string, and be able to explain how it applies to parsing algorithms.
- Be able to create a parse table for an $LL(1)$ grammar and simulate the steps of the table-driven parsing algorithm.

Propositional Logic
- Understand and be able to use the concepts of formal logic: proposition, $\lor$, $\land$, $\neg$, $\Rightarrow$, $\Leftrightarrow$, $\exists$, $\forall$, etc.
- Be able to manipulate expressions, justifying each step with a law. Be able to justify each step either by naming or stating (in general form) the appropriate law.
- Be able to identify whether any logical statement is a tautology, a contradiction, or contingent. Be able to use both techniques: simplifying expressions and making truth tables.
- Be able to produce a CNF or DNF expression for any expression (using laws) or for any function in a truth table (using minterms or maxterms).

Proof Techniques
- Given a set of premises and a conclusion in the form of propositions, be able to create a formal proof. Be able to justify each step either by naming or stating (in general form) the appropriate law.
- Be able to prove theorems involving natural numbers or real numbers.
- Be able to perform proof by induction, proof by contradiction, proof by contrapositive, and proof by the Deduction Theorem.

Predicate Calculus
- Understand and be able to apply concepts from predicate calculus: predicate, universe of discourse, the universal and existential quantifiers, and free and bound variables.
- Be able to standardize variables apart (i.e., rectify an expression).
- Given a set of premises and a conclusion in the form of predicates, be able to create a formal proof. Be able to justify each step either by naming or stating the appropriate law.
- Understand and be able to use universal instantiation and generalization, existential instantiation and generalization, and unification.

Resolution
- Be able to recognize whether an expression is a Horn clause. Be able to convert a conjunction of Horn clauses into a Datalog program.
• Be able to prepare a theorem to be proved by resolution by converting the premises and conclusion into a list of disjunctive clauses in Prenex Normal Form.

• Be able to create resolution proofs (justifying each step).

**Sets and Relations**

• Know basic set and string concepts: union, intersection, cardinality, set membership, power set, alphabet, string, language, $V^3$, $V^*$, $V^+$, etc.

• Be able to use set builder notation to describe a set and to perform proofs involving the subset, superset, and set equality operators.

• Be able to convert between the representations of relations: set of tuples with schema, table, and set of sets of attribute-value pairs.

• Understand relational operators, including union, intersection, difference, cross product, select, rename, project, natural join.

• Be able to write queries using relational algebra and convert Datalog into relational algebra.

• Be able to convert between the representations of binary relations: directed graph, adjacency matrix, and set of pairs (edges).

• Understand binary relation concepts: domain, domain space, range, range space, reflexivity, symmetry, transitivity, equivalence relation, partition, equivalence class (block), inverse, composition, reflexive closure, symmetric closure, and transitive closure.

• Know the difference between a partial order (weak vs. strict) and a total order and be able to produce a Hasse diagram for any partial order (note that a total order is also a partial order).

• Know what a poset is, but *don’t* worry about well-founded posets, upper bounds, maximal elements, etc. (important but not enough time).

• Be able to identify whether a binary relation is a function and whether a function is injective (one-to-one), surjective (onto), or bijective. Know the difference between composition of functions vs. relations.

**Graph Algorithms**

• Know the following graph concepts: formal definition of graph ($G = (V, E)$), vertex (node), edge, path, cycle, cyclic, connected, reachable, adjacent, subgraph, fully connected (complete).

• Be able to create a depth-first or breadth-first tree and to classify edges as forward, backward, or cross.

• Know the difference between preorder and postorder depth-first traversal of a tree, and be able to produce a topological sort using postorder traversal.

• Be able to describe, compare, contrast, execute, and apply the following algorithms: Warshall’s, Floyd’s, Dijkstra’s, Prim’s, and Kruskal’s.

• Be able to create inductive proofs for graph algorithms (understand structural induction, loop invariants, etc.).