EECS4302 Compilers and Interpreters
Fall 2026

Digest of Self-Guided Lecture, October 19

Formulating Regular Language Tools

Jackie Wang

How to use this Digest

Use this digest as a **roadmap and checklist**. It highlights the main ideas, important examples, common pitfalls, practice targets, and where to return in the course materials. It does **not** replace attending or reviewing the session, using the recording when needed, studying Jackie's annotated iPad notes and other official course materials, completing assigned work, or practising independently. Use it to identify what you understand, what you need to revisit, and what you should be able to do on your own.

If time is limited, start with the sections most relevant to your current review and return to the rest later; selective use is better than skipping the digest because it looks long.

This is an AI-assisted study roadmap reviewed by Jackie and grounded in the session transcript and Jackie's annotated iPad notes. Like any course document, it may still contain an error or omission. If something seems technically doubtful or inconsistent with the original course materials, return to those materials and ask Jackie for clarification rather than treating the digest as final authority.

Table of Contents

Main Ideas 2

Roadmap 2

Key Terms, Notation, and Structures 6

Key Examples and Demonstrations 7

Common Pitfalls and Misconceptions 9

What You Should Be Able to Do 10

Suggested Practice 11

Where to Review 11

Closing Reminder 12

Main Ideas

- **Regular-language operations give a set-theoretic foundation for regular expressions.** Jackie developed union, concatenation, and Kleene closure directly on languages, then used those operations to define the formal meaning of regular-expression operators.
- **The boundary cases matter.** The lecture repeatedly used ε and $\varnothing$ to distinguish an empty string from an empty language. In particular, $L^0 = \{\varepsilon\}$ and $\varnothing^* = \{\varepsilon\}$, not $\varnothing$.
- **DFA and NFA are closely related, but their transition functions have different result types.** A DFA transition returns one state, while an NFA transition returns a subset of states. This difference drives the way strings are traced, how acceptance is formulated, and how the generalized transition function $\hat{\delta}$ is defined.
- **NFA acceptance is an intersection condition.** After reading the whole input, an NFA accepts when at least one possible ending state is an accept state, equivalently when the ending-state set has nonempty intersection with F. Jackie explicitly rejected subset-based conditions as too strong or otherwise nongeneral.
- **Subset construction turns nondeterministic state sets into DFA states.** The theoretical output may contain as many as $2^{|Q|}$ subset states, but Jackie's algorithm uses lazy evaluation and constructs only reachable subsets, tracked through `Reachable` and `ToDiscover`.
- **An ε-NFA adds one more layer to the same formalism.** Its ordinary transition function explicitly allows ε in addition to alphabet symbols, while its generalized transition function incorporates ε-closure. The final acceptance condition remains the same intersection test used for an NFA.
- **This self-guided lecture is meant to be worked through, not merely watched.** Jackie repeatedly asked students to reconstruct formulas, trace automata, study the subset-construction pseudocode, and compare the DFA/NFA/ε-NFA formulations. He indicated that more exercises and clarification would follow on Thursday.

Roadmap

1. Regular-language union and concatenation

The session starts from the idea that a language is a set of strings. Page 2 of Jackie's annotated iPad notes therefore treats language operations as set operations. For languages L and M, union is

$$L \cup M = \{w \mid w \in L \vee w \in M\},$$

while language concatenation is

$$LM = \{xy \mid x \in L \wedge y \in M\}.$$

The worked example takes $L = \{ab, bc, ca\}$ and $M = \{ba, cb\}$ and forms every ordered concatenation of one string from L with one from M. Jackie also connects this exhaustive pairing to the counting rule $|LM| = |L||M|$ for this finite example.

2. Language powers and Kleene closure

Page 3 develops Kleene closure by first defining repeated concatenation of a language. The ordinary cases follow the expected pattern:

$$L^1 = L, \qquad L^2 = LL,$$

and, more generally, L^i contains strings formed by concatenating i strings chosen from L. Jackie emphasized the boundary case

$$L^0 = \{\varepsilon\},$$

because zero repetitions produce the length-zero string rather than the empty set.

The Kleene star is then the generalized union

$$L^* = \bigcup_{i \geq 0} L^i.$$

Two short exercises sharpen the notation. First, $|L|^i$ counts ordered choices of i strings from finite L; hence $|L^i| \leq |L|^i$, with equality when distinct choices give distinct concatenations. Second, if the language itself is $L = 0^* = \{\varepsilon, 0, 00, \ldots\}$, then taking the star of that language still gives $L^* = L$. Jackie explicitly warned that this last equality is a special case, not a general law for arbitrary languages.

3. Giving regular expressions a formal meaning

Page 4 turns the language operations into semantics for regular expressions. If E and F are regular expressions, Jackie develops the recursive cases

$$\begin{aligned} L(E + F) &= L(E) \cup L(F), \\ L(EF) &= L(E)L(F), \\ L(E^*) &= (L(E))^*, \\ L((E)) &= L(E). \end{aligned}$$

The key distinction is the level at which an operator is being used. For example, juxtaposing E and F is regular-expression concatenation; on the right-hand side, $L(E)L(F)$ is concatenation of two languages.

The base cases are equally important:

$$L(\varepsilon) = \{\varepsilon\}, \qquad L(\varnothing) = \varnothing, \qquad L(a) = \{a\}.$$

A student question prompted Jackie to stress that $L(a)$ contains exactly the one-symbol string a: neither ε, another symbol, nor aa is in that language.

4. Algebra with $\varnothing$ and ε

Page 5 uses the definitions rather than memorized identities. The four target simplifications are

$$\varnothing + L = L, \qquad \varnothing L = \varnothing, \qquad \varnothing^* = \{\varepsilon\}, \qquad \varnothing^* L = L.$$

The most important reasoning is the difference between having no string to choose and choosing the empty string. Unfolding the star shows that $\varnothing^0 = \{\varepsilon\}$ while every positive power remains $\varnothing$. For $\varnothing L$, the set-comprehension predicate requires an element from $\varnothing$; that conjunct is always false, so the resulting language is empty. For $\varnothing^* L$, the only left-hand string is ε, which acts as the identity for string concatenation.

5. DFA as a 5-tuple and why δ is total

Pages 6–7 formulate a DFA as

$$M = (Q, \Sigma, \delta, q_0, F),$$

with transition function

$$\delta : Q \times \Sigma \to Q.$$

Jackie used the six-state example on page 6 to identify the state set, alphabet, start state, accept-state set, and transition function, and then traced the input 0101 to an accepting state.

A central study question appears on page 7: the ordinary DFA transition function is a **total** function. For every state and every input symbol, exactly one destination state must be defined. Jackie contrasted this with a partial-function notation and asked students to decide whether that alternative would still be appropriate for a DFA.

6. Two equivalent ways to formulate DFA string processing

Page 6 first describes acceptance by introducing the sequence of states visited while the characters $a_1, \ldots, a_n$ are read. The transition condition is

$$\delta(q_{i-1}, a_i) = q_i$$

for the chosen indexing convention, followed by the requirement $q_n \in F$. Jackie noted that other indexing conventions are also valid if their ranges are adjusted consistently.

Page 8 then introduces the generalized transition function $\hat{\delta} : Q \times \Sigma^* \to Q$, defined recursively by

$$\hat{\delta}(q, \varepsilon) = q, \qquad \hat{\delta}(q, xa) = \delta(\hat{\delta}(q, x), a).$$

For $w = 010$, the recursive step processes the prefix 01 first and then the last symbol 0. This makes the accepted language compact to state:

$$L(M) = \{w \in \Sigma^* \mid \hat{\delta}(q_0, w) \in F\}.$$

Jackie said students should be comfortable with both the indexed and $\hat{\delta}$-based formulations.

7. NFA transitions and the parallel-state trace

Page 9 changes the codomain of the transition function:

$$\delta : Q \times \Sigma \to \mathcal{P}(Q).$$

A state-symbol pair can therefore lead to no state, one state, or several states. The page-9 trace processes 00101 by carrying forward a set of possible states and unioning the outcomes from each current state. The ending set is $\{q_0, q_2\}$, and because q_2 is accepting, the input is accepted.

This is the decisive NFA perspective: all currently possible states are kept alive in parallel. An undefined branch contributes $\varnothing$; it does not cancel the other branches.

8. The correct NFA acceptance condition

Page 10 isolates the logic. Let E be the set of possible ending states after the entire input and F the set of accept states. The acceptance condition is

$$E \cap F \neq \varnothing,$$

equivalently, there exists a state that belongs to both E and F.

Jackie explicitly tested two tempting alternatives, $E \subseteq F$ and $F \subseteq E$, and showed by counterexample that they are not general acceptance criteria. The requirement is only that at least one possible run end in an accepting state.

9. Relating DFA and NFA, then defining $\hat{\delta}$ for NFA

Pages 11–12 compare the two machine classes. If every NFA transition result is a singleton set, the machine behaves as a DFA. If a transition result has cardinality zero or greater than one, nondeterminism is present. Jackie therefore drew DFA as the more specialized class inside the more general class of NFA.

Page 13 extends the recursive string-processing idea to an NFA. The base case is now

$$\hat{\delta}(q, \varepsilon) = \{q\},$$

and the recursive case must apply δ separately to each state reachable after the prefix and union all the results:

$$\hat{\delta}(q, xa) = \bigcup_{q' \in \hat{\delta}(q,x)} \delta(q', a).$$

The language definition becomes

$$L(M) = \{w \in \Sigma^* \mid \hat{\delta}(q_0, w) \cap F \neq \varnothing\}.$$

Page 14 briefly reviews generalized union because this set-level flattening is central to the recursive definition.

10. NFA-to-DFA subset construction with lazy evaluation

Pages 15–17 move from formulation to conversion. Each state of the output DFA represents a subset of states from the input NFA. If the NFA has n states, there are at most 2^n different subsets, so a three-state NFA has at most eight subset states and a ten-state NFA has an upper bound of 1024.

Jackie's practical message is that this is only a theoretical upper bound. The algorithm constructs **reachable** subset states on demand. Page 15 uses three set-valued variables/roles:

- `Reachable`: subset states already known to be reachable;
- `ToDiscover`: reachable subset states whose outgoing transitions still need to be processed;
- `NotYetDiscovered`: new successor subset states after removing those already in `Reachable`.

The first iteration starts from the singleton subset $\{q_0\}$. Its successors on inputs 0 and 1 are kept as distinct subset states, not flattened together; after removing the already-known $\{q_0\}$, the new state $\{q_0, q_1\}$ remains to be discovered. The loop stops only when `ToDiscover` becomes empty.

11. ε-NFA: extending the transition domain

Page 18 introduces the final machine variant. The alphabet itself still does **not** contain ε. Instead, the transition function explicitly extends its input domain:

$$\delta : Q \times (\Sigma \cup \{\varepsilon\}) \to \mathcal{P}(Q).$$

The example table adds a separate ε-column; among the entries shown, $\delta(q_0, \varepsilon) = \{q_1\}$ and $\delta(q_3, \varepsilon) = \{q_5\}$, while several other states have no ε-transition.

12. $\hat{\delta}$ for ε-NFA and the final comparison

Page 19 makes the contrast across the three machine types explicit. The base case for an ε-NFA is

$$\hat{\delta}(q, \varepsilon) = \mathrm{ECLOSE}(q),$$

not merely q or $\{q\}$. For the recursive case, first process the prefix x, then consume the final symbol a from every reachable state, and then include all states reachable by ε-transitions from those results:

$$\hat{\delta}(q, xa) = \bigcup \left\{ \mathrm{ECLOSE}(q'') \;\middle|\; q' \in \hat{\delta}(q, x) \wedge q'' \in \delta(q', a) \right\}.$$

The acceptance condition itself is unchanged from the NFA case:

$$L(M) = \{w \in \Sigma^* \mid \hat{\delta}(q_0, w) \cap F \neq \varnothing\}.$$

Jackie's recommended way to study this final part is to compare the DFA, NFA, and ε-NFA recursive definitions side by side and identify exactly what changes at each level.

Key Terms, Notation, and Structures

Language A set of strings. The lecture uses this set view to define union, concatenation, powers, and Kleene closure.

Language concatenation $LM = \{xy \mid x \in L \wedge y \in M\}$: choose a string from L, choose one from M, then concatenate them in that order.

Language power	L^i is the language formed by concatenating i strings chosen from L. The boundary case is $L^0 = \{\varepsilon\}$.
Kleene closure / star	$L^* = \bigcup_{i \geq 0} L^i$: all strings formed by any number of repetitions, including zero repetitions.
ε	The empty string, of length zero. It is a string and must not be confused with the empty set $\varnothing$.
$\varnothing$	The empty language/set, containing no strings.
Regular-expression semantics	The mapping $L(E)$ that assigns to a regular expression E the language it denotes.
DFA	A deterministic finite automaton $M = (Q, \Sigma, \delta, q_0, F)$ with total transition function $\delta : Q \times \Sigma \to Q$.
NFA	A nondeterministic finite automaton with $\delta : Q \times \Sigma \to \mathcal{P}(Q)$. A transition can therefore return a set of possible destination states.
Generalized transition function	$\hat{\delta}$, which processes a whole string rather than one symbol. Its return type and recursion differ across DFA, NFA, and ε-NFA.
Ending-state set	For an NFA or ε-NFA, the set of all states that may be reached after the entire input has been processed. Acceptance requires nonempty intersection with F.
Power set $\mathcal{P}(Q)$	The set of all subsets of Q. NFA transitions return members of this power set, and subset construction uses such subsets as DFA states.
Subset construction	Conversion from NFA to DFA in which each output DFA state represents a subset of input NFA states.
Lazy evaluation	Construct only subset states that are actually reachable from the initial subset, instead of enumerating all $2^{\lvert Q \rvert}$ theoretical possibilities.
ε-closure / ECLOSE	The set of states reachable from a state using only zero or more ε-transitions. It appears in the ε-NFA base case and after each ordinary symbol step.

Key Examples and Demonstrations

1. Concatenating two finite languages

Setup: Page 2 uses $L = \{ab, bc, ca\}$ and $M = \{ba, cb\}$.

Main point: Form one result for every ordered pair (x, y) with $x \in L$ and $y \in M$. The six resulting strings illustrate both the formal set-comprehension definition and the product-rule count.

What to practise: Reconstruct the six strings without copying the page, and explain why reversing the order to ML would in general produce a different language.

2. Why L^0 and $\varnothing^*$ contain ε

Setup: Pages 3 and 5 treat zero concatenations as the boundary case.

Main point: Zero repetitions do not mean that there is no resulting string; they produce the unique length-zero string. This is why $L^0 = \{\varepsilon\}$ and, after unfolding all powers, $\varnothing^* = \{\varepsilon\}$.

What to practise: Derive $\varnothing^*$, $\varnothing^* L$, and $\varnothing L$ from the definitions rather than memorizing the final identities.

3. DFA trace and the two formulations of acceptance

Setup: Pages 6 and 8 use the same DFA to process inputs such as 0101 and 010.

Main point: The character-by-character state sequence and the recursive $\hat{\delta}$ definition describe the same computation at different levels of abstraction.

What to practise: Trace 0101 using ordinary δ, then write the corresponding $\hat{\delta}$ expression. Be able to explain why the DFA transition function must be total.

4. NFA trace on 00101

Setup: Page 9 starts in $\{q_0\}$ and carries a set of possible states through each symbol.

Main point: Each current state produces a destination set, and the results are unioned. The final set $\{q_0, q_2\}$ is accepted because it has nonempty intersection with the accept-state set.

What to practise: Reproduce the set after every input symbol, then use the page-10 logic to justify acceptance formally. Jackie also left the nearby rejecting-input trace as an exercise rather than something to memorize from the recording.

5. Why subset conditions do not define NFA acceptance

Setup: Page 10 tests candidate logical conditions using small sets of ending and accepting states.

Main point: Acceptance does not require all ending states to be accepting, nor all accepting states to appear among the endings. It requires at least one overlap.

What to practise: Produce a small counterexample to $E \subseteq F$ and to $F \subseteq E$, then state the correct condition $E \cap F \neq \varnothing$.

6. First lazy subset-construction iteration

Setup: Page 15 starts with `Reachable` = $\{\{q_0\}\}$ and `ToDiscover` = $\{\{q_0\}\}$.

Main point: Processing $S = \{q_0\}$ gives separate successor subset states for inputs 0 and 1. Keeping these results as a *set of subsets* lets the algorithm remove already-known subset

states without losing their identity. The newly discovered $\{q_0, q_1\}$ is then added to both tracking sets for later processing.

What to practise: Continue the remaining iterations from the table on page 15 until `ToDiscover` is empty, checking your result against the three reachable subset states shown there.

7. Extending NFA reasoning to ε-NFA

Setup: Pages 18–19 add an ε-transition column and then redefine $\hat{\delta}$.

Main point: ε is not placed inside Σ; it is added explicitly to the transition domain. In the generalized computation, ε-closure is used in the base case and after each ordinary symbol transition.

What to practise: Starting from the diagram on page 18, reproduce the ε-column of the transition table, then compare the DFA, NFA, and ε-NFA base/recursive cases side by side.

Common Pitfalls and Misconceptions

- **Confusing ε with $\varnothing$.** The first is one string of length zero; the second is a set containing no strings.
- **Treating Σ^* and L^* as the same operation on the same kind of object.** Both use star notation, but one starts from an alphabet and the other from a language. Page 3 explicitly separates these contexts.
- **Generalizing the special example $L = 0^* \Rightarrow L^* = L$.** Jackie marked this as a special case, not a law for arbitrary L.
- **Assuming $L(a)$ also contains ε.** The base regular expression a denotes exactly the singleton language $\{a\}$.
- **Assuming $\varnothing^* = \varnothing$.** The zero-repetition term contributes $\{\varepsilon\}$, so the star is not empty.
- **Using a partial transition function for a DFA without checking the definition.** Jackie's formulation requires one defined destination for every state-symbol pair; that is why δ is total.
- **Mixing δ and $\hat{\delta}$.** Ordinary δ processes one symbol. The generalized version processes a string and is recursively defined using the ordinary transition step.
- **Applying NFA δ directly to a set of states.** The ordinary function takes one state and one symbol. For a whole current-state set, apply δ to each member and union the returned sets.
- **Using subset inclusion as the NFA acceptance test.** The source-grounded condition is nonempty intersection with F, not $E \subseteq F$ or $F \subseteq E$.
- **Assuming subset construction must build all $2^{|Q|}$ subsets.** That value is an upper bound. Jackie's lazy algorithm keeps only subsets reachable from the start subset.

- **Flattening a set of subset states too early.** In the lazy algorithm, $\{\{q_0, q_1\}, \{q_0\}\}$ must preserve two distinct macro-states. Replacing it by $\{q_0, q_1\}$ loses the information needed to track what has already been discovered.
- **Putting ε inside Σ for an ε-NFA.** Jackie explicitly treated $\varepsilon \notin \Sigma$ and extended the transition domain by $\Sigma \cup \{\varepsilon\}$.

What You Should Be Able to Do

After this session, you should be able to:

- define language union, concatenation, powers, and Kleene closure using set notation and distinguish their roles from regular-expression operators;
- explain why $L^0 = \{\varepsilon\}$, why $|L|^i$ counts ordered choices of i strings from finite L, and why $|L^i| \leq |L|^i$ in general;
- use $L(E)$ to give the recursive and base-case semantics of the regular-expression forms developed in the session;
- derive the four page-5 identities involving $\varnothing$, ε, union, concatenation, and star;
- write the DFA 5-tuple, explain why its ordinary transition function is total, and trace an input through the machine;
- state and use both the indexed DFA-processing formulation and the recursive generalized transition function $\hat{\delta}$;
- explain precisely how the NFA transition-function codomain differs from the DFA case and trace an NFA by maintaining a set of possible states;
- justify NFA acceptance using $E \cap F \neq \varnothing$ and reject incorrect subset-based alternatives with counterexamples;
- explain in what sense every DFA can be viewed as an NFA whose destination sets are all singletons;
- write and interpret the recursive NFA definition of $\hat{\delta}$, including the generalized union across intermediate states;
- explain the $2^{|Q|}$ subset-state upper bound and carry out lazy subset construction by updating `Reachable`, `ToDiscover`, and newly discovered subset states;
- define the ordinary transition function for an ε-NFA, explain why $\varepsilon \notin \Sigma$, and reproduce the role of ε-closure in its generalized transition function;
- compare the DFA, NFA, and ε-NFA base cases, recursive cases, and acceptance conditions without collapsing their distinct return types.

Suggested Practice

1. Rework pages 2–3 of Jackie's annotated iPad notes from a blank page: define $L \cup M$, LM, L^0, L^i, and L^*, then check your notation against the source.
2. Re-derive the page-5 results $\varnothing + L$, $\varnothing L$, $\varnothing^*$, and $\varnothing^* L$ using the definitions and the logical predicate in the set comprehension. Do not rely only on the final simplified forms.
3. Use the page-6 DFA to retrace 0101, then rewrite the same kind of processing using $\hat{\delta}$ as developed on page 8. Also revisit Jackie's study question about why the DFA transition function is total rather than partial.
4. Reproduce the page-9 NFA trace for 00101, and then do the rejecting exercise written at the top of that page independently. Finish by stating the acceptance condition in both intersection and existential form.
5. Work through the NFA formula for $\hat{\delta}$ on page 13 using the final-character split $w = xa$. Make sure you can explain why ordinary δ must be applied to each possible intermediate state separately.
6. Trace the remaining lazy subset-construction iterations after the first iteration shown on page 15. Keep the values as a set of subset states when required, and verify that the process stops when `ToDiscover` is empty.
7. Compare pages 8, 13, and 19 side by side. Write a three-row summary for DFA, NFA, and ε-NFA showing the base case of $\hat{\delta}$, the recursive step, the return type, and the acceptance condition.
8. Before the follow-up class, identify any step in the subset-construction pseudocode or ε-NFA formulation that you cannot yet justify from the definitions. Jackie said further exercises and clarification would follow on Thursday, so bring those specific questions forward.

Where to Review

- **Jackie's annotated iPad notes, pages 2–3:** regular-language union/concatenation, language powers, L^0, Kleene closure, counting, and the special $L = 0^*$ example.
- **Pages 4–5:** recursive/base-case semantics of regular expressions and the exercises involving $\varnothing$, ε, star, union, and concatenation.
- **Pages 6–8:** DFA 5-tuple, transition-function totality, the indexed acceptance formulation, and the generalized transition function $\hat{\delta}$.
- **Pages 9–10:** NFA string processing for 00101, the rejecting follow-up exercise, and the correct acceptance logic using ending states and accept states.

- **Pages 11–14:** DFA versus NFA transition-function types, DFA-as-specialized-NFA reasoning, NFA $\hat{\delta}$, and generalized union.
- **Pages 15–17:** subset-construction pseudocode, the `Reachable/ToDiscover` mechanism, the first lazy-evaluation iteration, and the $2^{|Q|}$ upper-bound intuition.
- **Pages 18–19:** ε-NFA transition domain, transition-table ε-column, ε-closure, recursive $\hat{\delta}$, and the final language-acceptance condition.
- Return to the corresponding self-guided lecture recording whenever a formula or trace in the notes is too compressed to reconstruct confidently. Jackie also referred students to the Math Review material for power sets, set operations, logic, functions, and related notation used throughout these formulations.

Closing Reminder

Because this was a self-guided lecture rather than a regular in-person meeting, work through Jackie's lecture recording actively rather than treating it as a video to play straight through. Pause at the important definitions, derivations, state traces, and subset-construction steps, and try to predict or reproduce the next step before continuing. If a transition or formal statement remains unclear, replay the relevant portion to repair that gap.

For structured review, start with Jackie's annotated iPad notes because they most closely reflect the actual development of the material. Work through them in manageable logical units. After each logical unit, study the corresponding part of the presentation slides alongside it. Jackie does not normally teach by reading the slides line by line, so the slides may contain relevant supporting material that was not explicitly read aloud.

Finally, choose a suitable selection of the important reasoning from this session and reproduce it independently: language-operation derivations, regular-expression semantics, DFA/NFA string traces, acceptance arguments, $\hat{\delta}$ recursions, subset-construction iterations, and ε-closure reasoning. The goal is not to memorize Jackie's recorded steps, but to determine whether you can reconstruct the reasoning from your own understanding and explain why each important step works.