

EECS4302 Compilers and Interpreters

Fall 2026

Digest of Lecture 6, September 29

Languages, Regular Expressions, and DFA/NFA Foundations

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	5
Key Examples and Demonstrations	6
Common Pitfalls and Misconceptions	7
What You Should Be Able to Do	7
Suggested Practice	8
Where to Review	8
Closing Reminder	9

Main Ideas

- Lexical analysis rests on a hierarchy of formal objects: an alphabet Σ , strings over that alphabet, languages $L \subseteq \Sigma^*$, and computational problems that ask whether a given string belongs to a language. Precise distinctions such as Σ versus Σ^1 , and $\{\varepsilon\}$ versus $\emptyset$, matter.
- Regular expressions, DFA, NFA, and ε -NFA are equivalent ways to specify regular languages. This session focused on regular-expression construction and precedence, then began the DFA/NFA comparison.
- A useful way to compare two regular expressions is to compare their languages. To show that two expressions are not equivalent, one witness string that belongs to one language but not the other is enough.
- DFA design requires a deterministic transition choice for every symbol in the alphabet from every state. State meanings and dead states help organize the construction. NFA design relaxes this restriction and can make some languages easier to model, even though NFAs and DFAs are equivalent in expressive power.
- Assignment 1 remains an immediate application of this material. Jackie's annotated iPad notes record the due date as October 7, and Jackie emphasized subset construction as one of the techniques used in the assignment.

Roadmap

1. Review scope and Assignment 1 priorities

Jackie opened by keeping the focus practical: cover enough lexical-analysis theory this week to support Assignment 1, while postponing some of the heavier formalism until later. The lecture digest and notes template remain the starting points for review. Jackie also said that he would try to release the project and Assignment 2 before next Thursday's class (before the reading week), followed by a short briefing on what to focus on.

2. Systematically enumerating strings and reading Σ notation

The session resumed the previous exercise with the alphabet $\{A, B, C\}$ and the set Σ^4 . Jackie used a four-layer tree: at every layer, branch once for each symbol in the alphabet. Every complete root-to-leaf path gives one string of length four. The point was not the specific alphabet, but a systematic method that avoids missing cases.

Jackie then separated several notations that look similar but represent different kinds of objects:

- Σ is the alphabet itself, a set of symbols.
- Σ^1 is the set of strings of length one over Σ .
- Σ^+ contains strings of positive length.

- Σ^* contains strings of length zero or more, so it includes ε .
- For any alphabet Σ , $\Sigma^0 = \{\varepsilon\}$.

A related bonus exercise asks you to prove or disprove

$$\Sigma_1 \subseteq \Sigma_2 \implies \Sigma_1^* \subseteq \Sigma_2^*.$$

The lecture supplied the interpretation and intuition; the proof was left for independent work.

3. From alphabets and strings to languages and problems

A language L over Σ is a set of strings, so $L \subseteq \Sigma^*$. Jackie also reviewed the distinction between subset and proper subset. A proper subset excludes equality; in practical language examples there are usually strings in Σ^* that are not valid members of the language.

The concrete example used Σ_{key} , the ASCII characters available from the keyboard, and Σ_{key}^* , all strings that can be typed from that alphabet. Jackie then described the language of compilable Java programs by set comprehension:

$$L_{\text{Java}} = \{p \mid p \in \Sigma_{\text{key}}^* \text{ and } p \text{ compiles in Eclipse}\}.$$

Here “compilable” was deliberately separated from runtime correctness.

Two small languages sharpened the notation:

$$L_1 = \{\varepsilon\}, \quad L_2 = \emptyset.$$

L_1 is a non-empty language containing exactly the empty string; L_2 contains no strings at all. Jackie suggested revisiting how to specify these two languages using regular expressions and automata.

Finally, the computational-problem viewpoint was phrased as membership: given a string $w \in \Sigma^*$ and a language $L \subseteq \Sigma^*$, decide whether $w \in L$. The Java-compilation example fits this form directly.

4. Regular expressions as a language specification

Regular expressions were introduced as a textual way to specify regular languages. Jackie emphasized that regular expressions, DFA, NFA, and ε -NFA are equally expressive for regular languages, so conversions among these forms are possible.

The three regular-expression operators emphasized were:

- Kleene star R^* : repeat pattern R zero or more times;
- concatenation $R_1 R_2$: match R_1 followed by R_2 ;
- union $R_1 + R_2$: match either R_1 or R_2 .

Without parentheses, precedence is

$$* \text{ (highest), } \quad \text{concatenation,} \quad + \text{ (lowest).}$$

Parentheses override this order.

5. Worked regular expression for alternating 0s and 1s

For the language of strings whose 0s and 1s alternate, Jackie first built a case-based expression:

$$(01)^* + (10)^* + 1(01)^* + 0(10)^*.$$

This includes the empty string as well as strings beginning with either symbol and ending after either parity of length. The expression was then refactored to reduce repetition:

$$(1 + \varepsilon)(01)^* + (0 + \varepsilon)(10)^*.$$

The less-factored version is still acceptable when it is correct; the main task is to cover exactly the intended language.

6. Operator precedence and witness strings

Jackie compared

$$10^* \quad \text{and} \quad (10)^*.$$

Because star has the highest precedence, 10^* means $1(0^*)$, whereas $(10)^*$ means zero or more repetitions of the whole block 10.

To show two regular expressions are not equivalent, compare their languages and find one witness in one language but not the other. Jackie's annotated iPad notes record two clean witnesses for this pair:

$$100 \in L(10^*) \text{ but } 100 \notin L((10)^*),$$

while

$$\varepsilon \in L((10)^*) \text{ but } \varepsilon \notin L(10^*).$$

One valid witness is already sufficient to disprove equivalence.

7. DFA design: determinism, state meanings, and a dead state

The DFA exercise used the language

$$\{ w \mid w \neq \varepsilon \text{ and } w \text{ has equal numbers of alternating 0s and 1s} \}.$$

Jackie emphasized that, for a DFA, each state must have exactly one outgoing transition for each symbol in Σ .

The partial construction in Jackie's annotated iPad notes illustrates how state meanings guide transitions. Starting from s_0 , reading 0 moves to s_1 ; reading 1 from s_1 reaches accept-state s_3 . A second consecutive 0 leads to dead state s_5 , because the alternation condition has been permanently violated. From s_3 , another 0 returns to s_1 , while another 1 goes to the dead state. The symmetrical branch for strings beginning with 1 was left for students to complete.

8. Why an NFA can be easier to design

The final example considered strings over $\{0, 1\}$ that end with 01. Jackie showed an NFA in which s_0 can keep consuming the arbitrary prefix by looping on 0 or 1, while the same input 0 can also nondeterministically move from s_0 to s_1 to guess that the final 01 has begun. Then

$$s_1 \xrightarrow{1} s_2,$$

where s_2 is accepting.

This illustrates the modeling convenience of nondeterminism. It does not create a more powerful class of languages: the NFA can be converted to an equivalent DFA, and Jackie connected that conversion to subset construction in Assignment 1.

Key Terms, Notation, and Structures

Σ	An alphabet: a finite, non-empty set of symbols.
Σ^n	The set of strings of length n over Σ .
Σ^0	The singleton set $\{\varepsilon\}$.
Σ^+	All non-empty strings over Σ .
Σ^*	All strings over Σ , including ε .
ε	The empty string, with length zero. It is a string, not the empty set and not a “null character.”
$L \subseteq \Sigma^*$	A language over Σ is a set of strings drawn from Σ^* .
$L(R)$	The language denoted by regular expression R .
Kleene star	The regular-expression operator for zero or more repetitions of a pattern.
Concatenation	Matching one regular-expression pattern immediately after another.
Union +	Choosing either of two regular-expression alternatives.
DFA	Deterministic finite automaton: each state has exactly one transition for each symbol in the alphabet.
Dead state	A non-accepting state from which no future input can recover acceptance; it typically loops on all alphabet symbols.
NFA	Nondeterministic finite automaton: a state may have missing transitions or multiple transitions for the same input symbol.
Subset construction	The conversion method from an NFA to an equivalent DFA; Jackie noted that it is used in Assignment 1.

Key Examples and Demonstrations

Enumeration tree for $\{A, B, C\}^4$

Setup: Four character positions, each with the choices A , B , or C .

Main point: Build one tree layer per position. A complete root-to-leaf path chooses exactly one symbol at each layer and therefore corresponds to one length-four string.

What to practise: Reproduce the method for a different alphabet or with a restriction on one or more positions. See page 3 of Jackie's annotated iPad notes for the layered-tree sketch.

Compilable Java programs as a language

Setup: Start with all keyboard-ASCII strings Σ_{key}^* , then select the strings that compile in Eclipse.

Main point: The compiler's recognition task can be viewed as a language membership question. "Compiles" is the predicate used in the set comprehension; it is not a claim about runtime behavior.

What to practise: Explain why an invalid Java source string can still be in Σ_{key}^* while not being in L_{Java} .

Alternating 0s and 1s

Setup: Construct a regular expression for strings whose 0s and 1s alternate.

Main point: First cover all structural cases, then optionally factor common subexpressions. Compare the two source-grounded forms

$$(01)^* + (10)^* + 1(01)^* + 0(10)^*$$

and

$$(1 + \varepsilon)(01)^* + (0 + \varepsilon)(10)^*.$$

What to practise: Test edge cases such as ε , one-symbol strings, and longer alternating strings against your expression.

Distinguishing 10^* from $(10)^*$

Setup: Apply precedence before trying to compare the languages.

Main point: $10^* = 1(0^*)$, while $(10)^*$ repeats the two-symbol block. A single witness proves non-equivalence; page 8 of Jackie's annotated iPad notes uses 100 in one direction and ε in the other.

What to practise: For another pair of similar-looking regular expressions, write down $L(R_1)$ and $L(R_2)$ conceptually and search for the shortest useful witness.

DFA and NFA constructions

DFA example: Revisit page 9 of Jackie’s annotated iPad notes and explain what s_0 , s_1 , s_3 , and dead state s_5 mean before following the arrows. Then complete the symmetric branch that begins by reading 1.

NFA example: Revisit page 10. Explain why s_0 can both keep consuming the arbitrary prefix and, on input 0, also branch toward the final 01. Then explain why this nondeterministic convenience does not change expressive power.

Common Pitfalls and Misconceptions

- **Confusing Σ with Σ^1 , Σ^+ , or Σ^* .** Σ contains symbols; the powered/starred forms contain strings.
- **Confusing ε , $\{\varepsilon\}$, and $\emptyset$.** ε is one string, $\{\varepsilon\}$ is a one-element language, and $\emptyset$ is the empty language.
- **Treating “compilable” as the same as runtime correctness.** The L_{Java} example uses compilation as its membership predicate; it does not claim that every compiling program behaves correctly at runtime.
- **Ignoring regular-expression precedence.** Star binds tighter than concatenation, which binds tighter than union. Therefore 10^* and $(10)^*$ denote different languages.
- **Writing $w \in R$ when R is a regular expression.** Membership is in the language denoted by the expression: write $w \in L(R)$.
- **Leaving a DFA transition undefined or giving two choices for one symbol.** For each DFA state and each symbol in Σ , exactly one next state must be determined. Use a dead state when a condition has been permanently violated.

What You Should Be Able to Do

After this session, you should be able to:

- distinguish Σ , Σ^n , Σ^+ , Σ^* , Σ^0 , ε , $\{\varepsilon\}$, and $\emptyset$;
- enumerate fixed-length strings systematically using a layered tree and explain why each root-to-leaf path corresponds to one string;
- explain a language as a subset of Σ^* and formulate a simple language using set comprehension;
- frame compilation or another recognition task as a membership problem $w \in L$;
- use star, concatenation, and union to construct a regular expression and apply the correct operator precedence;

- compare two regular expressions through $L(R_1)$ and $L(R_2)$ and use a witness string to disprove equivalence;
- explain the determinism requirement of a DFA, assign meanings to states, and use a dead state when needed;
- explain how an NFA can use multiple interpretations/transitions to simplify a design, and trace the NFA for strings ending in 01;
- explain why DFA, NFA, ε -NFA, and regular expressions can specify the same regular languages, and identify subset construction as an NFA-to-DFA conversion technique.

Suggested Practice

1. Complete Jackie's bonus exercise: prove or disprove $\Sigma_1 \subseteq \Sigma_2 \Rightarrow \Sigma_1^* \subseteq \Sigma_2^*$.
2. After reviewing the relevant regular-expression and automata material, specify $L_1 = \{\varepsilon\}$ and $L_2 = \emptyset$ using a regular expression and the automata forms Jackie named in class.
3. Try the two additional operator-precedence/equivalence exercises shown on page 8 of Jackie's annotated iPad notes. For any non-equivalence claim, give a valid witness in one language but not the other.
4. Draw a DFA for the language of binary strings that contain 01 as a substring, as Jackie suggested during the DFA portion.
5. Complete the symmetric half of the DFA on page 9 for the case in which the first symbol read is 1. Label each state by the property it represents before drawing transitions.
6. Continue Assignment 1 with the lecture material beside you, especially the regular-language equivalences and subset construction. Jackie's annotated iPad notes list the due date as October 7.

Where to Review

- **Page 2 of Jackie's annotated iPad notes:** current reminders, including the October 7 Assignment 1 due date.
- **Page 3:** the length-four enumeration tree, Σ versus Σ^1 , and the subset/Kleene-star proof-or-disproof exercise.
- **Page 4:** language notation, subset versus proper subset, and the L_{Java} example over Σ_{key}^* .
- **Pages 5–6:** $\{\varepsilon\}$ versus $\emptyset$, $\Sigma^0 = \{\varepsilon\}$, and language-membership problems.
- **Page 7:** the alternating-binary regular-expression construction and its two source-grounded forms.

- **Page 8:** regular-expression operator precedence and witness strings for comparing 10^* with $(10)^*$; the remaining highlighted pairs are practice targets.
- **Page 9:** the partially developed DFA, state meanings, accepting state, and dead-state reasoning.
- **Page 10:** DFA-versus-NFA comparison and the NFA for binary strings ending with 01.
- Use the session recording when you need the spoken reasoning behind a transition, witness, or notation choice, especially where Jackie develops the diagrams incrementally.

Closing Reminder

Use this digest as a roadmap for active review, not as a substitute for the learning process itself. Attend Jackie's in-person sessions and pay attention to how he develops ideas and works through the important reasoning step by step. If a transition, explanation, example, or reasoning step remains unclear afterwards, return to the recording to fill that gap.

For structured review, start with Jackie's annotated iPad notes because they most closely reflect the actual flow and development of the class. 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 work through it again on your own. The goal is not to memorize Jackie's recorded steps, but to determine whether you can reproduce the reasoning from your own understanding and explain why the important steps work.