EECS4302 Compilers and Interpreters
Fall 2026

Digest of Lecture 12, October 27

DFA Minimization, Scanner Tracing, and Parser Context

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 4

Key Examples and Demonstrations 5

Common Pitfalls and Misconceptions 6

What You Should Be Able to Do 6

Suggested Practice 7

Where to Review 8

Closing Reminder 8

Main Ideas

- Jackie revisited the DFA-minimization algorithm through the roles of P, T, and a candidate subset S. The algorithm stops only at a true fixed point, $P = T$; merely having the same number of partition blocks is not enough.
- The lecture then completed the main lexical-analysis story by showing how a minimized DFA can be implemented as tables and used by a scanner to recognize token categories. The worked example used the register pattern $r[0 \ldots 9]^+$ and traced the input `r2EOF`.
- The scanner example has two sequential phases: scan forward while saving useful states on a stack, then roll back after reaching the error state until an accepting state or the sentinel `bad` is reached. In the trace, the final token type is `register` and the matched lexeme is `r2`.
- The session closed by placing the parser in context: the lexer maps a linear character sequence to a linear token sequence, while the parser maps that token sequence into a non-linear parse tree/AST. Top-down and bottom-up parsing are the next major topics.
- For immediate course preparation, Assignment 2 was due October 28, and the Programming Test was scheduled for November 5. Jackie said the minimization and scanner pseudocode would be supplied if needed on a test or exam; students should practise tracing rather than memorize the code line by line.

Roadmap

1. Revisit DFA minimization as an iterative fixed-point process

Jackie began by returning to the minimization example from the previous lecture. The trace table on page 3 of Jackie's annotated iPad notes tracks three pieces of information: the partition set P carried into an iteration, the updated partition set T, and a selected subset S used for refinement. Initially, the states are separated into accepting and non-accepting blocks, while T begins empty for the trace shown in class.

The important stopping condition is exact equality:

$$P = T.$$

If $P \neq T$, another iteration is required. Jackie explicitly distinguished this from comparing only $|P|$ and $|T|$: two partition collections can have the same cardinality without containing the same blocks.

For the first refinement of the running example, the selected subset is

$$S = \{S_2, S_4\},$$

which is separated from the original non-accepting block. Jackie emphasized the selection criteria used in the algorithm: the candidate must be splittable and a proper subset, and the algorithm chooses a maximal candidate satisfying the required behavior. Students do not need to memorize the pseudocode; they need to be able to follow it and trace the partition updates correctly.

2. Move from token patterns to a scanner

Jackie then shifted from minimizing the DFA to implementing lexical recognition. A scanner may need to recognize many syntactic categories, each described by a regular-expression pattern. Conceptually, those patterns are combined so that matching one of the permitted categories avoids a lexical error. When multiple patterns can match, priority matters; Jackie connected this idea to the ordering of lexical rules in ANTLR4.

The example category was a register. In Jackie's annotated notes, the regular expression is

$$r[0\ldots9]^+,$$

meaning a lower-case `r` followed by at least one digit. The lecture connects this regular expression to the familiar pipeline

$$\text{RE} \longrightarrow \varepsilon\text{-NFA} \longrightarrow \text{DFA} \longrightarrow \text{minimized DFA},$$

after which the minimized DFA can be represented by tables for execution.

3. Read the scanner tables before tracing the code

Pages 6–7 of Jackie's annotated iPad notes contain the essential scanner data. A character-category table maps `r` to *Register*, digits to *Digit*, and EOF/other inputs to *Other*. A transition table then maps the current DFA state and category to the next state. The token-type table shows that S_2 is the accepting state associated with token type `register`; S_0, S_1, and the error state S_e are invalid as final results.

The corresponding DFA is a useful visual check, but Jackie stressed that the transition table is sufficient for executing the scanner. For tracing, keep the current state separate from the stack contents.

4. Trace the first scanner loop on `r2EOF`

The pseudocode begins with

$$\textit{state} = S_0, \qquad \textit{word} = \varepsilon,$$

and an initially empty stack seeded with the sentinel `bad`. The first while-loop continues until the current state becomes S_e.

For `r2EOF`, the decisive state progression is

$$S_0 \xrightarrow{\text{Register}} S_1 \xrightarrow{\text{Digit}} S_2 \xrightarrow{\text{Other}} S_e.$$

After reading `r`, the word is `r`; after reading `2`, it is `r2`. When EOF is read, S_2 is accepting, so the stack is reset before S_2 is pushed. EOF is categorized as *Other*, which sends the current state to S_e and ends the first loop. At this point, Jackie highlighted a subtle but important distinction: the current state is S_e, while the top of the stack is still S_2.

5. Roll back to the most recent accepting state

The second while-loop runs while the current state is neither accepting nor the sentinel `bad`. In the example, the scanner pops S_2 from the stack, sets the current state back to S_2, and truncates the final EOF item from the accumulated word. The word therefore returns from `r2EOF` to `r2`.

Because $S_2 \in F$, the loop exits. The token-type table maps S_2 to `register`. Keep the two outputs conceptually separate:

- token type/category: `register`;
- lexeme/word: `r2`.

Jackie described this scanner as deliberately simplified: the purpose is to understand table-driven recognition, stack use, and rollback rather than to study a production scanner implementation.

6. Connect lexical analysis to parsing

The final page of Jackie's annotated iPad notes places the parser in the compiler pipeline. Lexical analysis transforms a source program viewed as a sequence of characters into a sequence of tokens. Syntactic analysis then takes that linear token sequence and constructs a non-linear parse tree/AST.

Jackie previewed two parsing directions, top-down and bottom-up, and said the next material will review context-free grammars, derivations, and parse trees before developing the parsing algorithms. He also connected the resulting tree structures to later semantic processing and to the Composite/Visitor ideas already studied in the course.

Key Terms, Notation, and Structures

P and T
Partition collections used during DFA minimization. In Jackie's trace, P records the partitioning carried into an iteration and T records the refinement being produced. The fixed point is reached only when $P = T$.

S in the minimization trace
A candidate proper subset selected for refinement. Splittability and proper-subset status are conditions on the candidate; the algorithm then chooses a maximal candidate satisfying the required transition behavior.

Character category
The classification used before a DFA transition. In the scanner example, `r` is *Register*, digits are *Digit*, and EOF/other inputs are *Other*.

S_e
The scanner's explicit error state. The first scanning loop exits when the current state becomes S_e.

F
The accepting-state set. For the register example, $F = \{S_2\}$.

`bad`
A sentinel pushed onto the scanner stack at initialization. It gives the rollback phase a stopping case when no accepting state can be recovered.

Token type vs. lexeme
The token type is the recognized category, such as `register`; the lexeme is the actual matched character sequence, such as `r2`.

Lexer / scanner
The phase that maps a linear sequence of source characters to a linear sequence of tokens.

Parser
The next phase, which consumes the token sequence and constructs a non-linear parse tree/AST according to the language grammar.

Key Examples and Demonstrations

DFA minimization trace table

Setup: Revisit page 3 of Jackie's annotated iPad notes. The initial partition separates $\{S_3, S_5\}$ from $\{S_e, S_0, S_1, S_2, S_4\}$, and T starts empty in the displayed trace.

Main point: The first selected subset is $S = \{S_2, S_4\}$, producing three blocks in the refined partition. Since the resulting T differs from P, the algorithm must continue. The formal stopping test is equality of the partition collections, not equality of their sizes.

What to practise: Continue the table independently until $P = T$, using the supplied minimization pseudocode rather than trying to memorize it.

Scanner trace: `r2EOF`

Setup: Use the category table, transition table, token-type table, and pseudocode together on page 6 of Jackie's annotated iPad notes.

Main point: The forward scan reaches S_2 after `r2`, then EOF is classified as *Other* and drives the current state to S_e. The stack still preserves S_2. Rollback pops S_2, truncates EOF, and returns the recognized token.

Result:

token type = `register`, lexeme = `r2`.

What to practise: Track at least the word, current state, input category, and stack after each important operation. Do not mentally merge the current state with the stack top.

Lexer-to-parser workflow

Setup: Page 9 of Jackie's annotated iPad notes shows the compiler pipeline around the scanner and parser.

Main point:

$$\text{characters} \longrightarrow \text{tokens} \longrightarrow \text{parse tree / AST}.$$

The first transformation is linear-to-linear; the parser then constructs the non-linear syntactic structure that later phases can process.

Common Pitfalls and Misconceptions

- **Stopping minimization because $|P| = |T|$.** Jackie explicitly clarified that the formal condition is $P = T$. Equal cardinalities do not by themselves establish that the partition blocks are identical.
- **Treating the whole partition as a valid split.** A refinement requires a proper subset; moving an entire block together does not create a new partition. The algorithm also selects a maximal valid candidate rather than arbitrarily splitting compatible states one by one.
- **Confusing current state with stack top.** In the `r2EOF` trace, the current state becomes S_e while the top of the stack remains S_2. That difference is exactly what makes rollback work.
- **Misreading the second while-loop condition.** Jackie used De Morgan's law to explain the exit cases: leave the loop when the recovered state is accepting or when the sentinel `bad` is reached.
- **Confusing token type and matched text.** `register` is the category/type; `r2` is the lexeme.
- **Trying to memorize the pseudocode line by line.** Jackie said the minimization/scanner code would be supplied if used in a test or exam. The study target is being able to trace and explain it correctly.

What You Should Be Able to Do

After this session, you should be able to:

- trace the DFA-minimization variables P, T, and S across iterations and justify why the process continues or stops;
- distinguish exact partition equality $P = T$ from merely having equal cardinalities;
- explain the role of splittability, proper-subset status, and maximal selection when refining a DFA partition;

- translate the register pattern $r[0 \ldots 9]^+$ into the intended token-recognition behavior and relate it to the minimized-DFA pipeline;
- use a character-category table and DFA transition table to update the scanner state step by step;
- trace the two scanner loops, including stack reset/push/pop behavior, the error state S_e, the `bad` sentinel, and word truncation;
- explain why `r2EOF` yields token type `register` with lexeme `r2`;
- distinguish the lexer's character-to-token transformation from the parser's token-to-tree transformation; and
- describe the upcoming roles of context-free grammars, derivations, parse trees, and top-down/bottom-up parsing at the level previewed in class.

Suggested Practice

1. Re-run the DFA-minimization example from the previous lecture using the $P/T/S$ table on page 3 of Jackie's annotated iPad notes. Continue beyond the first refinement until the exact fixed point $P = T$ is reached.
2. Complete the two posted DFA-minimization exercises and compare your work with the written solutions Jackie said were available on the lecture site.
3. Reproduce the `r2EOF` scanner trace from page 6 without looking at Jackie's annotations. Record the word, current state, category, and stack whenever they change.
4. Trace the slightly longer `r24EOF` example that Jackie recommended. Check that you can explain why the accepted lexeme is recovered after the forward scan reaches the error state.
5. Trace `r24*3` in two stages. For the first `NextWord()` call, follow the rollback logic and verify that the recognized prefix `r24` is returned with token type `register`. Then consider the next scan beginning with the remaining `*3`: because `*` sends the initial state directly to the error state before any accepting state is saved, that scan returns `invalid`. This distinction explains Jackie's statement that the complete input is ultimately invalid, while the shown single-token procedure first recognizes the valid prefix `r24`.
6. Before moving into parsing, review the self-guided formal notation material Jackie asked students to complete before the Written Test, and be ready to revisit context-free grammars, derivations, and parse trees.

Where to Review

- **Jackie's annotated iPad notes, page 3:** the $P/T/S$ minimization trace and the first refinement of the running DFA example.
- **Pages 4–5:** scanner categories/patterns and the visual connection from a character sequence to a token sequence and then toward a parse tree.
- **Pages 6–8:** the register regular expression, character- category table, transition table, token-type table, scanner pseudocode, `r2EOF` annotations, DFA visualization, and the independent `r24*3` practice setup.
- **Page 9:** "Parser in Context" - the compiler pipeline from scanner output to parser input, plus the top-down/bottom-up parsing preview and the review targets of context-free grammars, derivations, and parse trees.
- **Lecture site:** the two DFA-minimization exercises and their written solutions, as referenced by Jackie at the start of the session.
- **Recording:** return to the scanner trace when the timing of stack reset, push/pop, state update, or truncation is unclear; those details are easiest to repair by replaying the step-by-step execution alongside the annotated notes.

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.