EECS4302 Compilers and Interpreters
Fall 2026

Digest of Lecture 8, October 6

Subset Construction, Recursive Systems, and Polymorphism

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 8

What You Should Be Able to Do 8

Suggested Practice 9

Where to Review 10

Closing Reminder 10

Main Ideas

- **Subset construction makes nondeterminism explicit as sets.** Each state of the output DFA represents a *set* of possible states of the input NFA. Jackie used lazy evaluation: start from the initial subset, compute only reachable subset states, and stop once every transition leads to a subset already discovered.
- **Acceptance and determinism must both be preserved.** A DFA subset state is accepting whenever it contains at least one accepting NFA state, equivalently when its intersection with F is non-empty. For every DFA state, there must be exactly one transition for each symbol in the alphabet.
- **ε-NFA conversion adds ε-closure at the beginning and after each symbol transition.** The output DFA starts from $\mathrm{ECLOSE}(q_0)$. For each input symbol, first union the ordinary transition results from every state in the current subset; then take the union of the ε-closures of the resulting states.
- **The automata work connects directly to the lexer toolchain.** Jackie described the behind-the-scenes progression used by ANTLR4 as regular expression → ε-NFA → DFA → minimized DFA. The course focus here is applying the constructions and notation rather than proving their equivalence formally.
- **The lecture then shifted to recursive software structures.** A hardware-component tree motivated the distinction between base components and composite components. This is the kind of recursive organization that also appears in compiler structures such as abstract syntax trees, leading into the Composite pattern and later the Visitor pattern.
- **Static typing controls compile-time expectations; dynamic type controls runtime dispatch.** The declared type determines which attributes and methods may legally follow the dot operator. Static type does not change after declaration, whereas the object currently referenced may change. Dynamic binding chooses the overridden method using that current runtime object.
- **Immediate course actions matter.** Assignment 1 was due Wednesday, October 7; Assignment 2 and the project had been released. Jackie asked students to read the new instructions, begin forming project teams of two or three, and be ready for a high-level walkthrough in Thursday's class.

Roadmap

1. Immediate assessment and project preparation

Jackie began with several time-sensitive reminders. Assignment 1 was due the next day, Wednesday, October 7, and the instruction was to submit the best attempt on time in order to claim the available credit. Assignment 2 and the project had just been released. Jackie's annotated iPad notes record that Assignment 2 was recommended to be completed alone,

while students were asked to find project teammates as soon as possible. In class, Jackie suggested teams of two or three and said that one team-member file should be submitted per team by next Wednesday. He also planned to skim the Assignment 2 and project instructions on Thursday rather than immediately diving into all technical details.

Jackie pointed students to the project specification, examples from previous iterations, and the AI-usage policy on eClass. He also said that he would try to make himself available during Reading Week for questions. These logistics are worth acting on before the rest of the technical review.

2. NFA to DFA by subset construction and lazy evaluation

The first technical goal was to finish the conversion from an NFA to an equivalent DFA. The central principle, written prominently in Jackie's annotated iPad notes, is:

each output-DFA state corresponds to a set of input-NFA states.

For the worked NFA on page 3 of the notes,

$$\Sigma = \{0, 1\}, \qquad Q = \{q_0, q_1, q_2\}, \qquad F = \{q_2\},$$

with q_0 as the initial state. Although three NFA states have up to $2^3 = 8$ subsets in principle, Jackie did not enumerate them all. Instead, he used **lazy evaluation**: construct a subset only when a transition actually reaches it, and distinguish subsets that are already discovered from those that still need to be expanded.

Start with the singleton subset $\{q_0\}$. The worked transitions are:

$$\delta_D(\{q_0\}, 0) = \{q_0, q_1\},$$
$$\delta_D(\{q_0\}, 1) = \{q_0\}.$$

The new subset $\{q_0, q_1\}$ must then be expanded by unioning the NFA transitions of both members. In particular, the undefined transition from q_1 on 0 contributes $\varnothing$ rather than being silently ignored:

$$\delta(q_0, 0) \cup \delta(q_1, 0) = \{q_0, q_1\} \cup \varnothing = \{q_0, q_1\}.$$

On input 1, the same subset reaches $\{q_0, q_2\}$. Because $q_2 \in F$, $\{q_0, q_2\}$ is an accepting DFA state. Its outgoing transitions return only to subsets already discovered, so the construction stops. The complete output DFA therefore uses only three reachable subset states, not all eight possible subsets.

3. Extending the construction to an ε-NFA

The next adjustment was conceptually small but important. With ε-transitions, the initial DFA subset is not merely $\{q_0\}$; it is

$$\mathrm{ECLOSE}(q_0).$$

In the page-4 example, this gives $\{q_0, q_1\}$.

For a current subset T and input symbol a, Jackie split each transition into two stages. First compute the ordinary transition results from every member of T:

$$S = \bigcup_{q \in T} \delta(q, a).$$

Then expand every resulting state through ε-transitions:

$$\delta_D(T, a) = \bigcup_{s \in S} \text{ECLOSE}(s).$$

This is the key procedural difference from the earlier NFA-only construction. Jackie left entries in the displayed transition table for students to complete so that the two-stage computation could be practised directly.

4. Why this matters for lexical analysis

Jackie explicitly kept the focus practical rather than proof-heavy. The goal is to apply the conversion and understand the notation, because this forms part of the scanner/lexer foundation. He described the conceptual ANTLR4 pipeline as

$$\text{regular expression} \rightarrow \varepsilon\text{-NFA} \rightarrow \text{DFA} \rightarrow \text{minimized DFA}.$$

From a tool user's point of view, the regular expression is what is written; the automata construction and later minimization happen behind the scenes. Jackie said DFA minimization and the exponential-growth issue would be revisited after Reading Week.

5. Recursive systems as motivation for design patterns

The lecture then pivoted from automata to design patterns. Jackie used the hardware hierarchy on page 5 of his annotated iPad notes: a cabinet can contain chassis, a chassis can contain other components, and leaf components such as a card, hard drive, DVD/CD-ROM unit, or power supply cannot themselves host subcomponents. He labelled the two roles as **composite components** (internal nodes) and **base components** (leaves or external nodes).

The point was not the hardware itself. The important structural idea is that a system can be recursive: a composite contains children that may themselves be composites or leaves. Jackie connected this directly to compiler work, where abstract syntax trees must be traversed recursively. This sets up the Composite pattern, with Visitor to follow later and in the ANTLR4 tutorial.

6. Static type and compile-time expectations

Before going further with Composite and Visitor, Jackie reviewed polymorphism and dynamic binding. On page 6, the Student class is the superclass, with resident-student and non-resident-student subclasses. The central compile-time notion was **expectation**: given a reference such as `x`, what attributes or methods are legal after `x.`?

The answer depends on the variable's declared, or static, type. A `Student` reference can use members declared for `Student`; a `ResidentStudent` reference inherits those expectations and adds its own resident-specific members; a `NonResidentStudent` reference similarly adds its own non-resident-specific members. Members that belong only to one sibling subclass are not automatically available through a reference of the other sibling type.

7. Polymorphic assignment and the safety argument

Page 7 develops the assignment direction explicitly. If `s` has static type `Student` and `rs` has static type `ResidentStudent`, then assigning `s = rs;` is valid, while `rs = s;` should not compile.

Jackie justified the rejected direction using a contradiction-style runtime argument. Suppose `rs = s;` were allowed. The static type of `rs` would still promise resident-specific members such as `premiumRate`. But after the assignment, `rs` could point to an ordinary `Student` object that has no such member. A later access through that expectation could therefore fail at runtime. The compiler prevents this unsafe situation by rejecting the assignment.

8. Static type, dynamic type, and dynamic binding

The final development on page 8 separated two notions:

- the **static type** is the declared type and never changes; and
- the **dynamic type** is determined by the actual object currently referenced and may change after assignments.

For the trace

```
C obj;
obj = new C1();
obj.m();
obj = new C2();
obj.m();
```

the static type remains `C` throughout. Immediately after declaration, the reference is `null`; after the first assignment it points to a `C1` object, and after the second assignment it points to a `C2` object. Dynamic binding means that the call at line 3 uses the overridden `m()` from `C1`, while the call at line 5 uses the version from `C2`. Jackie emphasized that this mechanism will be used repeatedly when tracing Composite and Visitor designs.

Key Terms, Notation, and Structures

Subset state	A DFA state whose label is a set of states from the input NFA. It represents all NFA states that may be active after the same input.
Subset construction	The conversion technique that builds a DFA from an NFA by computing transitions between subset states.

Lazy evaluation	Construct only subset states that are actually reachable, instead of enumerating all $2^{\|Q\|}$ subsets in advance.
Discovered / not discovered	Bookkeeping for whether a reachable subset state has already had all of its outgoing DFA transitions computed.
Accepting subset	A DFA subset T is accepting when $T \cap F \neq \varnothing$.
ε-closure	$\mathrm{ECLOSE}(q)$ is the set of states reachable from q using zero or more ε-transitions, including q itself. Extended subset construction uses closures at the start and after each ordinary symbol transition.
Recursive system	A structure in which a component may contain other components of the same general family. The hardware hierarchy and compiler syntax trees are motivating examples from this session.
Base vs. composite component	A base component is a leaf/external node with no children; a composite component is an internal node that can contain child components.
Expectation	The set of attributes and methods that may legally appear after the dot operator for a reference, determined by its static type.
Static type	The declared type of a variable. Jackie emphasized that it does not change after declaration.
Dynamic type	The type of the actual runtime object currently being referenced. It may change as a reference is reassigned.
Dynamic binding	Runtime selection of the overridden method implementation according to the current dynamic type of the receiver.

Key Examples and Demonstrations

1. Complete subset construction from three NFA states

Setup: $Q = \{q_0, q_1, q_2\}$, $F = \{q_2\}$, and $\Sigma = \{0, 1\}$. The input NFA is nondeterministic because a single input may lead to more than one possible state.

Main point: Follow the page-3 table and diagram together. Start from $\{q_0\}$, compute transitions by union, create a new node only when a new subset is reached, and mark $\{q_0, q_2\}$ accepting because it intersects F.

What to practise: Reconstruct the three-state output DFA without copying the final drawing, and explain why no fourth reachable subset is needed.

2. Extended subset construction with ε-closure

Setup: Page 4 uses the floating-point-style ε-NFA from the previous lecture and displays a transition table.

Main point: The initial DFA subset is $\mathrm{ECLOSE}(q_0)$. For each input category, first union the ordinary δ results and then union the ε-closures of those resulting states.

What to practise: Complete the omitted table entries using the two-step procedure, then compare against the completed material in the course slides.

3. Hardware hierarchy as a recursive system

Setup: Page 5 shows a cabinet containing chassis and leaf hardware components.

Main point: A composite can contain children, including other composites; leaves cannot. The same recursive shape motivates how compiler data structures such as ASTs are organized and traversed.

What to practise: Point to each pictured component and classify it as a base component or a composite component, then explain how a total-price operation would naturally recurse over the tree.

4. The Student expectation matrix

Setup: Page 6 compares `Student`, `ResidentStudent`, and `NonResidentStudent` references against superclass members and subclass-specific members.

Main point: Compilation of dot notation is decided from the static type. Inherited members remain available in subclasses, while a member unique to one sibling subclass is not an expectation of the other sibling.

What to practise: Recreate the green-check/red-cross matrix from the class hierarchy before looking back at the annotated page.

5. Why the reverse assignment is rejected

Setup: Page 7 assumes, for contradiction, that a `ResidentStudent` reference can be assigned an ordinary `Student` reference.

Main point: The static type of `rs` promises resident-specific members, but the actual `Student` object may not contain them. The assignment is rejected to prevent an unsafe runtime access.

What to practise: Redraw the two reference/object boxes and explain the failure using the specific `premiumRate` expectation.

6. Five-line dynamic-binding trace

Setup: Page 8 declares `C obj`, assigns a `C1` object, calls `m()`, then assigns a `C2` object and calls `m()` again.

Main point: Static type remains `C`. The runtime object changes, and therefore dynamic binding selects `C1.m()` at the first call and `C2.m()` at the second.

What to practise: Reproduce the static/dynamic-type table for all five lines and annotate which implementation of `m()` is selected.

Common Pitfalls and Misconceptions

- **Enumerating every theoretical subset instead of using lazy evaluation.** The upper bound may be exponential, but the worked DFA only needs reachable subset states. Expand newly discovered states and stop when every result is already known.
- **Computing a subset transition imprecisely.** Every NFA member in the current subset contributes to the union. An undefined transition contributes $\varnothing$; dropping or inventing a member changes the DFA.
- **Forgetting accepting-state propagation.** A DFA subset must be marked accepting whenever it contains at least one state from F. Jackie explicitly warned that otherwise the converted DFA could end up with no accepting state where one is required.
- **Using ordinary subset construction unchanged for an ε-NFA.** Start from $\mathrm{ECLOSE}(q_0)$, and after each symbol transition take the ε-closure of every resulting state before forming the next DFA subset.
- **Letting runtime assignments change the static type.** They do not. The static type is fixed by the declaration; reassignment changes which object is referenced, not the declared type.
- **Treating subclass expectations as interchangeable.** A `ResidentStudent` does not gain members unique to `NonResidentStudent`, and vice versa. Sibling-specific expectations do not cross the inheritance tree.
- **Assuming both assignment directions in an inheritance hierarchy are equally safe.** Assigning a subclass object to a superclass reference is allowed in the example, but assigning a general superclass reference to a more specific subclass reference would make subclass-specific expectations unsafe.
- **Choosing an overridden method from the static type.** Static type controls compile-time member availability; dynamic binding chooses the actual overridden method using the runtime object's dynamic type.

What You Should Be Able to Do

After this session, you should be able to:

- convert a small NFA to a DFA using subset construction and lazy evaluation;
- compute subset transitions by unioning the member-state transitions, including $\varnothing$ when a transition is undefined;

- identify accepting DFA subset states using $T \cap F \neq \varnothing$;
- adapt the construction to an ε-NFA by using ε-closure for the start subset and after every symbol transition;
- explain the conceptual regular-expression $\rightarrow$ ε-NFA $\rightarrow$ DFA $\rightarrow$ minimized-DFA pipeline Jackie associated with ANTLR4;
- distinguish base components from composite components in a recursive hierarchy and explain why this shape is relevant to compiler ASTs;
- determine which attributes and methods are legal after the dot operator from a variable's static type and inheritance relationships;
- justify why `s = rs;` is allowed in the worked hierarchy while `rs = s;` is rejected;
- track static and dynamic types through a short execution trace; and
- determine which overridden method executes under dynamic binding.

Suggested Practice

1. **Finish the immediate course actions.** Submit Assignment 1 by Wednesday, October 7 using your best attempt. Read the newly released Assignment 2 and project instructions before Thursday's walkthrough. Jackie recommended completing Assignment 2 alone and asked students to begin forming project teams of two or three.
2. **Redo the page-3 subset construction from scratch.** Do not copy the final DFA. Maintain a discovered/not-discovered list, compute both input transitions for each subset, and mark acceptance only after checking the subset against F.
3. **Complete the ε-NFA transition table on page 4.** Jackie explicitly left entries for practice. For each cell, perform the two steps separately: ordinary δ union first, then the union of ε-closures. His strongest recommendation was to work through the whole table if time permits.
4. **Rebuild the type reasoning rather than memorizing the checks.** Recreate the page-6 expectation matrix, the page-7 `rs = s;` contradiction, and the page-8 static/dynamic-type trace from the class hierarchy and code alone.
5. **Prepare for the project and ANTLR4 work.** Read the project specification and AI-usage policy, inspect the previous-project examples if useful, submit one project-team member file per team by next Wednesday, and take notes on how the Visitor pattern is used when working through the ANTLR4 tutorial during Reading Week.

Where to Review

- **Jackie's annotated iPad notes, page 2:** Assignment 1 deadline, release of Assignment 2 and the project, recommendation for Assignment 2, and the reminder to find project teammates.
- **Jackie's annotated iPad notes, page 3:** the complete NFA-to-DFA subset-construction example, including lazy evaluation, the transition table, accepting-subset marking, and the final three-state DFA drawing.
- **Jackie's annotated iPad notes, page 4:** extended subset construction for an ε-NFA, especially the start state from $\mathrm{ECLOSE}(q_0)$ and the two-stage transition calculation.
- **Jackie's annotated iPad notes, page 5:** the recursive hardware system, with base components/leaves and composite components/internal nodes.
- **Jackie's annotated iPad notes, page 6:** the three-class Student hierarchy and the expectation matrix governed by static types.
- **Jackie's annotated iPad notes, page 7:** the polymorphic assignment example and the contradiction argument explaining why `rs = s;` should not compile.
- **Jackie's annotated iPad notes, page 8:** the distinction between static and dynamic type, the five-line trace, and dynamic binding of `m()` to the `C1` or `C2` implementation.
- **Session recording:** return to the spoken discussion for the project/team logistics, Jackie's practical explanation of the ANTLR4 automata pipeline, and the transition from recursive systems into the Composite/Visitor motivation.

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.