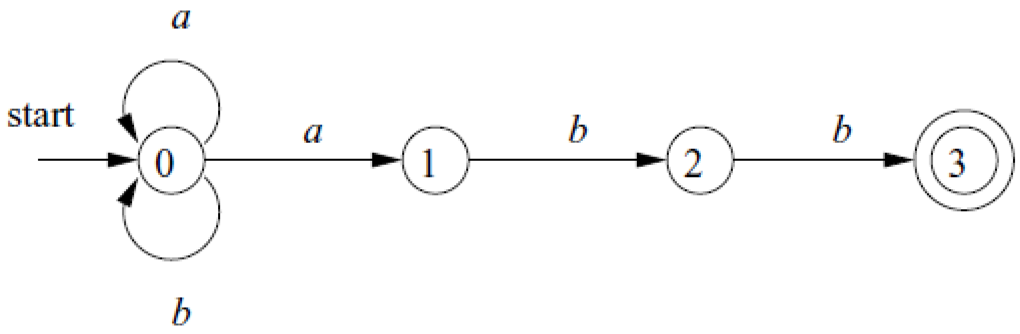
CS 420 - Compilers

Dr. Chen-Yeou (Charles) Yu

- Finite Automata (3.6)
  - ~~Nondeterministic Finite Automata (NFA) (3.6.1)~~
  - ~~Transition Tables (3.6.2)~~
  - ~~Acceptance of Input Strings by Automata (3.6.3)~~
  - Deterministic Finite Automata (3.6.4)
- From Regular Expressions to Automata (3.7)
  - Conversion of an NFA to a DFA (3.7.1)

Finite Automata

- A deterministic finite automaton (DFA) is a special case of an NFA where:
  - There are no moves on input *epsilon*, and (Epsilon is not allowed anymore)
  - For each state *s* and input symbol *a*, there is **exactly one edge** out of *s* labeled *a*
- So, just take a look at the previous translation table for NFA
  - From state 0, if we use 'a', it can bring us back to state 0 or state 1.
  - So it is {0, 1}



STATE	a	b	ϵ
0	$\{0,1\}$	$\{0\}$	$\emptyset$
1	$\emptyset$	$\{2\}$	$\emptyset$
2	$\emptyset$	$\{3\}$	$\emptyset$
3	$\emptyset$	$\emptyset$	$\emptyset$

Finite Automata

- Now, for DFA, if we are using a transition table to represent a DFA, then each entry is a single state.
- We may therefore represent this state **without** the curly braces { }, that we use to form sets.
  - While the NFA is an **abstract representation** of an algorithm to recognize the strings of a certain language, **the DFA is a simple, concrete algorithm for recognizing strings.**
  - Every regular expression and every NFA can be converted to a DFA accepting the same language

Finite Automata

- Simulating a DFA (An algorithm)
  - Indeed a DFA is so reasonable there is an obvious algorithm for simulating it (i.e., reading a string and deciding whether or not it is in the language accepted by the DFA).
  - **INPUT**:
    - An input string x terminated by an end-of-file character **eof**.
    - A DFA D (D means the whole graph) with start state $\mathbf{S_0}$, accepting states **F**, and **transition** function **move()**.
  - **OUTPUT**: Answer “yes" if D accepts x; “no" otherwise
  - **METHOD**: Apply the algorithm in the next page, to the input string x.
    - The function move(s, c) gives the state to which there is an edge from state s on input c.
      - c is just a character
    - The function nextChar() returns the next character of the input string x

Finite Automata

- An easy algorithm in the book

```
s = s0;
c = nextChar();
while ( c != eof ) {
        s = move(s, c);
        c = nextChar();
}
if ( s is in F ) return "yes";
else return "no";
```

- Based on the previous algorithm, we had an example like this:
  - Given a language (a|b)*abb, same as we mentioned before in NFA
  - Given the input string “ababb”, this DFA enters a sequence of states: 0,1,2,1,2,3
  - And return a “yes”, in the accepting state “3”
  - See the next page for detail

Finite Automata

- DFA accepting (a|b)*abb

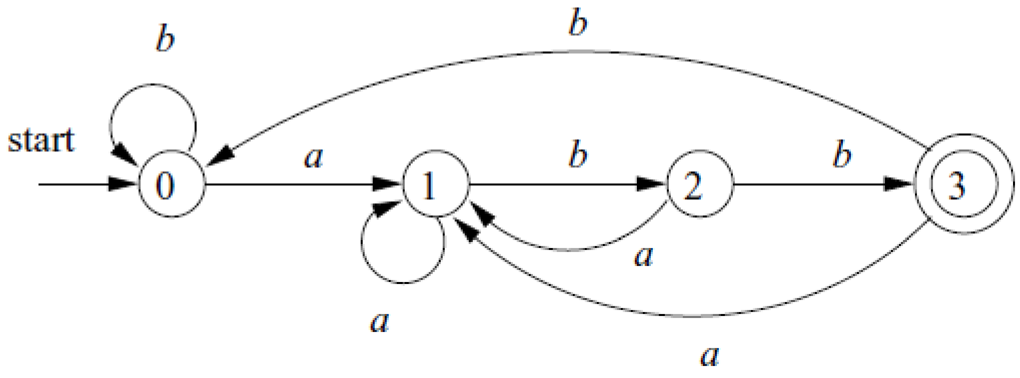
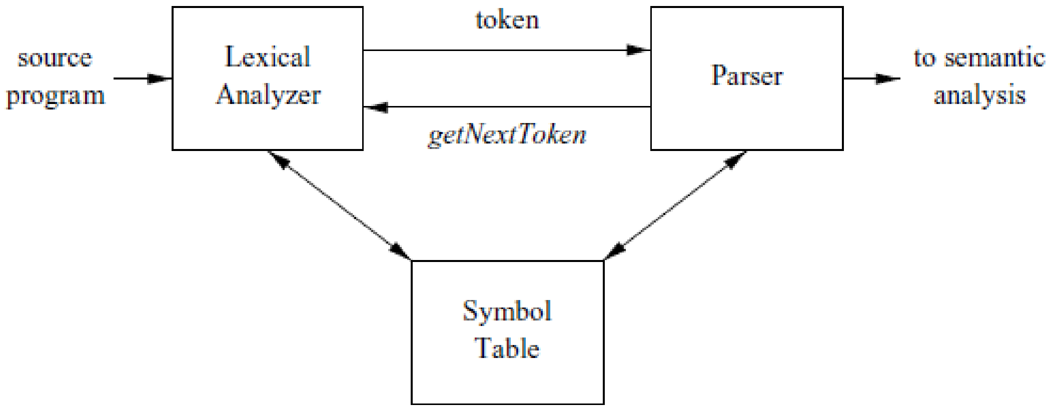


Figure 3.28: DFA accepting $(\mathbf{a}|\mathbf{b})^*\mathbf{abb}$

From Regular Expressions to Automata

- Do not forget the goal of this chapter is to understand the lexical analysis
- NFA often has a **choice** of move on an input symbol but DFA doesn't
- DFA looks more reasonable in the implementation of software
- Our first job is to convert NFA to DFA (the subset construction)
- We had a couple of jobs to do
    - Convert the Regular Expression (RE) to NFA
    - **Convert the NFA to DFA**
- The book, introduced the "jobs to do" in, reversed order



Conversion of an NFA to a DFA

- The general idea behind the subset construction is that each state of the constructed DFA **corresponds** to a **set** of **NFA states**.
- DFA states would be the subset of NFA states!
- Performance analysis:
  - The algorithm from the book presented is awful since for a set with *k* elements, there are 2^k subsets.
  - Fortunately, normally only a small fraction of the possible subsets occur in practice.

Conversion of an NFA to a DFA

- Algorithm: subset construction of DFA from NFA
- INPUT: An NFA, N.
- OUTPUT: A DFA, D accepting the same language as N.
- This is the example of the input NFA, N

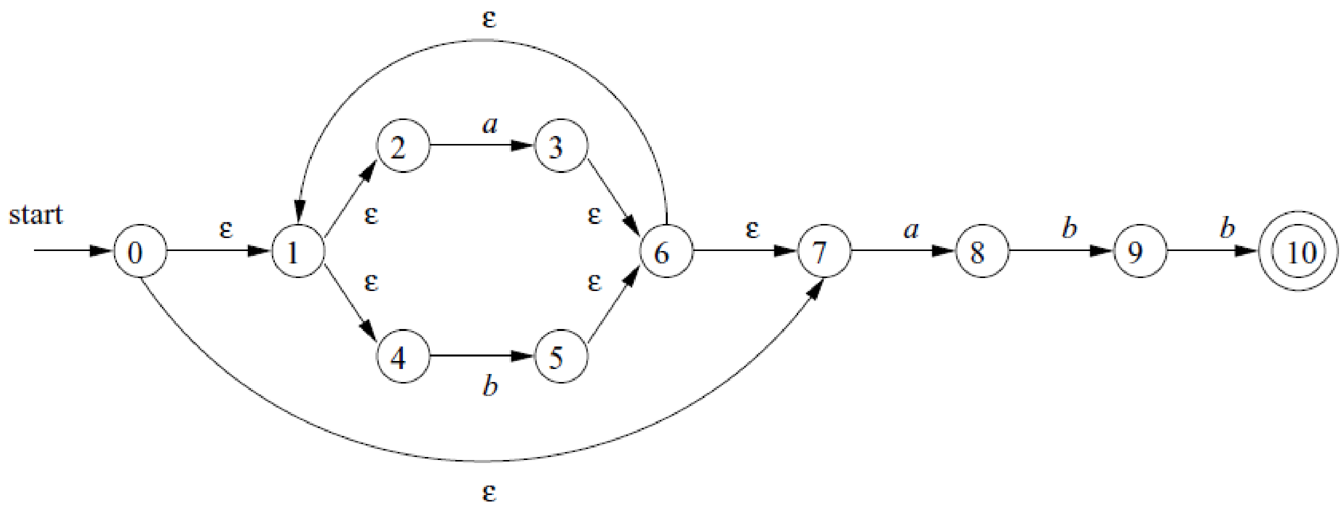


Figure 3.34: NFA N for $(\mathbf{a}|\mathbf{b})^*\mathbf{abb}$

Conversion of an NFA to a DFA

- This is the tool (Transition table, Dtran) we are going to use in this example, the Dtran for DFA, D

NFA STATE	DFA STATE	a	b
$\{0, 1, 2, 4, 7\}$	A	B	C
$\{1, 2, 3, 4, 6, 7, 8\}$	B	B	D
$\{1, 2, 4, 5, 6, 7\}$	C	B	C
$\{1, 2, 4, 5, 6, 7, 9\}$	D	B	E
$\{1, 2, 4, 5, 6, 7, 10\}$	E	B	C

Figure 3.35: Transition table *Dtran* for DFA D

Conversion of an NFA to a DFA

- This is the output example of output DFA, D
  - See? The epsilons are removed!

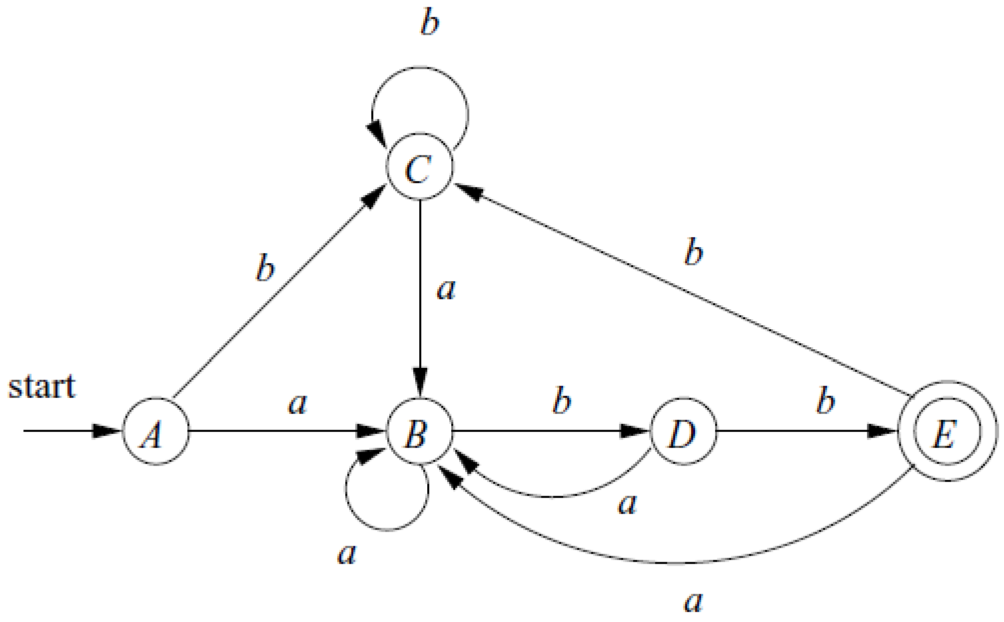


Figure 3.36: Result of applying the subset construction to Fig. 3.34

Conversion of an NFA to a DFA

- As for the detail, it involves complicated set operations.
  - We will talk about this in the next time

Plans for the rest of the class

- So far, we only have 35% of the semester scores.
- We might still need (either way)
  - 2 more HW(s) + 1 quiz, or
  - 2 more Quizzes + 1 HW
- But if it is a quiz, I will let you know one week earlier.
- The bottom line is, if I don’t understand the content of the book, it won’t show up in the exam ^_^