CSE443
Compilers

Dr. Carl Alphonce
alphonce@buffalo.edu
343 Davis Hall

Announcements

- Project document posted on website
- Team formation
    - We can take a few minutes to do it now
    - Make private Piazza post with UBIT username and corresponding GitHub username.

Phases of a compiler

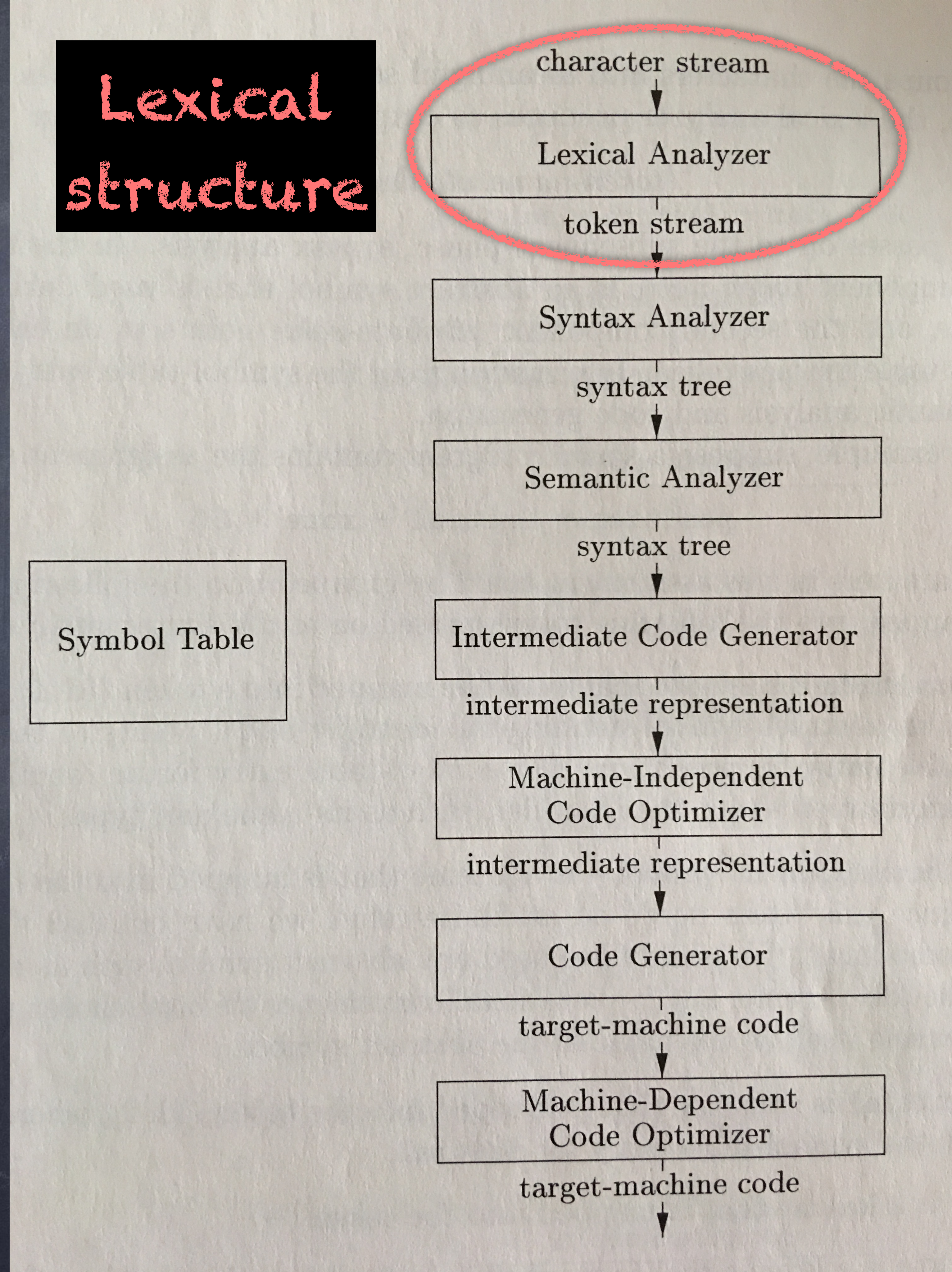


Figure 1.6, page 5 of text

Languages & grammars

Formally, a grammar $G = (N, \Sigma, P, S)$ is defined by 4 items:

1. N, a set of non-terminals

   $N = \{ X, Y \}$

2. Σ, a set of terminals (alphabet)

   $\Sigma = \{ a, b \}$ <- for example

   $N \cap \Sigma = \{\}$ <- general grammar constraints

3. P, a set of productions of the form (right linear)

   X -> aY

   Y -> bX

   Y -> a <- a right linear grammar describing a regular language

   X -> ε

   $X \in N$, $Y \in N$, $a \in \Sigma$, ε denotes the empty string

4. S, a start symbol

   S = Y

   $S \in N$

languages & grammars

[...] a regular grammar is a grammar that is right-regular or left-regular:

- all production rules have at most one non-terminal symbol
- that symbol is either always at the end or always at the start of the rule's right hand side.

https://en.wikipedia.org/wiki/Regular_grammar

languages & grammars

Given a string αA, where $\alpha \in \Sigma^*$ and $A \in N$,
and a production
$A \rightarrow \beta \in P$
we write $\alpha A \Rightarrow \alpha\beta$ to indicate that αA derives $\alpha\beta$ in one step.

$\Rightarrow^k$ and $\Rightarrow^*$ can be used to indicate k or arbitrarily many derivation steps, respectively.

languages & grammars

$\mathcal{L}(G)$ is the set of all strings derivable from G starting with the start symbol; i.e. it denotes the language of G.

Languages & grammars

Given a grammar G the language it generates, $\mathcal{L}(G)$, is unique.

Given a language L there are many grammars H such that $\mathcal{L}(H) = L$.

Languages & grammars

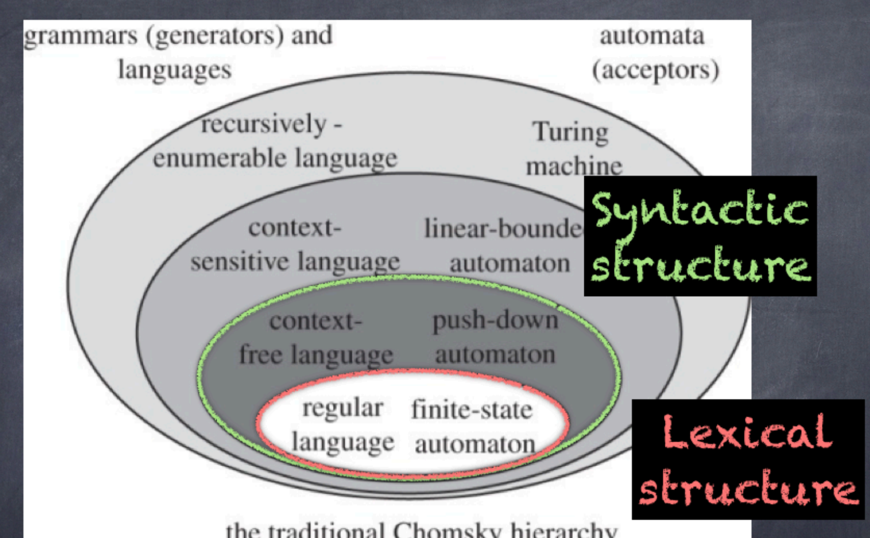
Think about what this means for us: there is no single "correct" grammar for a language.

In fact, grammars for users vs. tool writers vs. compiler writers can all be different.

Given a language L there are many grammars H such that $\mathcal{L}(H) = L$.

Lexical Analysis

- Lexical structure described by regular grammar
- Deterministic finite state machine performs analysis



SOURCE: https://openi.nlm.nih.gov/detailedresult.php?img=PMC3367694_rstb20120103-g2&req=4

AUTHORS: Fitch WT, Friederici AD - Philos. Trans. R. Soc. Lond., B, Biol. Sci. (2012)

How is a regular language defined?

- Recall that a language is a set of strings. This set can be finite or infinite.
- The possible regular languages over a given alphabet are defined inductively - construction given on next two slides.

LANGUAGE operations

base cases

- $\{ \varepsilon \}$ is a regular language
- $\forall\ a \in \Sigma$, $\{ a \}$ is a regular language

ε is the empty string

LANGUAGE operations

If L and M are regular, so are:

- $L \cup M = \{ s \mid s \in L \text{ or } s \in M \}$ UNION
- $LM = \{ st \mid s \in L \text{ and } t \in M \}$ CONCATENATION
- $L^* = \bigcup_{i=0,\infty} L^i$ KLEENE CLOSURE
- No other languages are regular

L^i is L concatenated with itself i times:
$L^0 = \{\varepsilon\}$, by definition
$L^1 = L$
$L^2 = LL$
$L^3 = LLL$, etc.
L^* is the union of all these sets!

Example of L^*

Suppose L is {a, bb}

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

$L^1 = L = \{a, bb\}$

$L^2 = LL = \{aa, abb, bba, bbbb\}$

$L^3 = LLL = \{$ aaa, aabb, abba, abbbb,
bbaa, bbabb, bbbba, bbbbbb$\}$

$L^4 =$

...and so so...

$L^* = \bigcup_{i=0,\infty} L^i = \{\varepsilon$, a, bb, aa, abb, bba, bbbb, aaa,
aabb, abba, abbbb, bbaa, bbbba, bbaa, bbabb,
bbbba, bbbbbb, abbbb, bbabb, ... $\}$

Some regular languages over $\Sigma = \{0,1\}$

The base cases yield these regular languages:

$\{\varepsilon\}$, $\{0\}$, $\{1\}$

The inductive cases yield many more. Some are:

$\{0, 1\}$, $\{01\}$, $\{10\}$, $\{01, 10\}$, $\{0, 01\}$, $\{1, 01\}$, $\{0, 10\}$, $\{1, 10\}$, $\{0, 1, 01\}$, $\{0, 1, 10\}$, $\{0, 01, 10\}$, $\{1, 01, 10\}$, $\{00\}$, $\{000\}$, $\{0000\}$, $\{11\}$, $\{111\}$, $\{1111\}$, and many many more.

Can you demonstrate how each of these is regular?

Why use grammars?

- Recall that a language is a possibly infinite set of strings.
- A grammar gives us a way to describe, using finite means, an infinite set.
- Regular expressions are equivalent to regular grammars in expressive power: both regular grammars and regular expressions describe regular languages.
- If X is a regular expression, $\mathcal{L}(X)$ denotes the set of strings recognized by X.

Inductive definition of REGular EXpressions (regex) over a given alphabet Σ

ε is a regex

$\mathcal{L}(\varepsilon) = \{\varepsilon\}$

For each $a \in \Sigma$, a is a regex

$\mathcal{L}(a) = \{a\}$

Regular expressions (regex) Inductive definition

Assume r and s are regexes.

r|s is a regex denoting $\mathcal{L}(r) \cup \mathcal{L}(s)$

rs is a regex denoting $\mathcal{L}(r)\mathcal{L}(s)$

r* is a regex denoting $(\mathcal{L}(r))^*$

(r) is a regex denoting $\mathcal{L}(r)$

Precedence: Kleene closure > concatenation > union

Associativity: all left-associative (minimize use of parentheses: (r|s)|t = r|s|t)

Algebraic laws

Assume r and s are regexes.

Commutativity	r|s = s|r
Associativity	r|(s|t) = (r|s)|t and r(st) = (rs)t
Disributivity	r(s|t) = rs|rt and (s|t)r = sr|tr
Identity	εr = rε = r
Idempotency	r** = r*

We can describe a regular language using a regular expression

Why do we care?

- We will be using a tool called FLEX to construct a lexical analyzer (a lexer) for the programming language we're constructing a compiler for.
- If we give FLEX a regular expression describing the lexical structure of our language, FLEX will produce a C program which acts as our lexer.
- The next step for us to understand (at a high level) how FLEX converts a regex to a C program.

A regular expression can be implemented using a finite state machine.

Finite state machines can be deterministic or non-deterministic:

DFA
deterministic finite automaton

NFA
non-deterministic finite automaton

Process of building lexical analyzer

1) spell out the language

language

Process of building lexical analyzer

2) formulate a regular expression

$\boxed{\text{language}} \rightarrow \boxed{\text{regex}}$

Process of building lexical analyzer

3) build an NFA

language → regex → NFA

Process of building lexical analyzer

4) transform NFA to DFA

language → regex → NFA → DFA

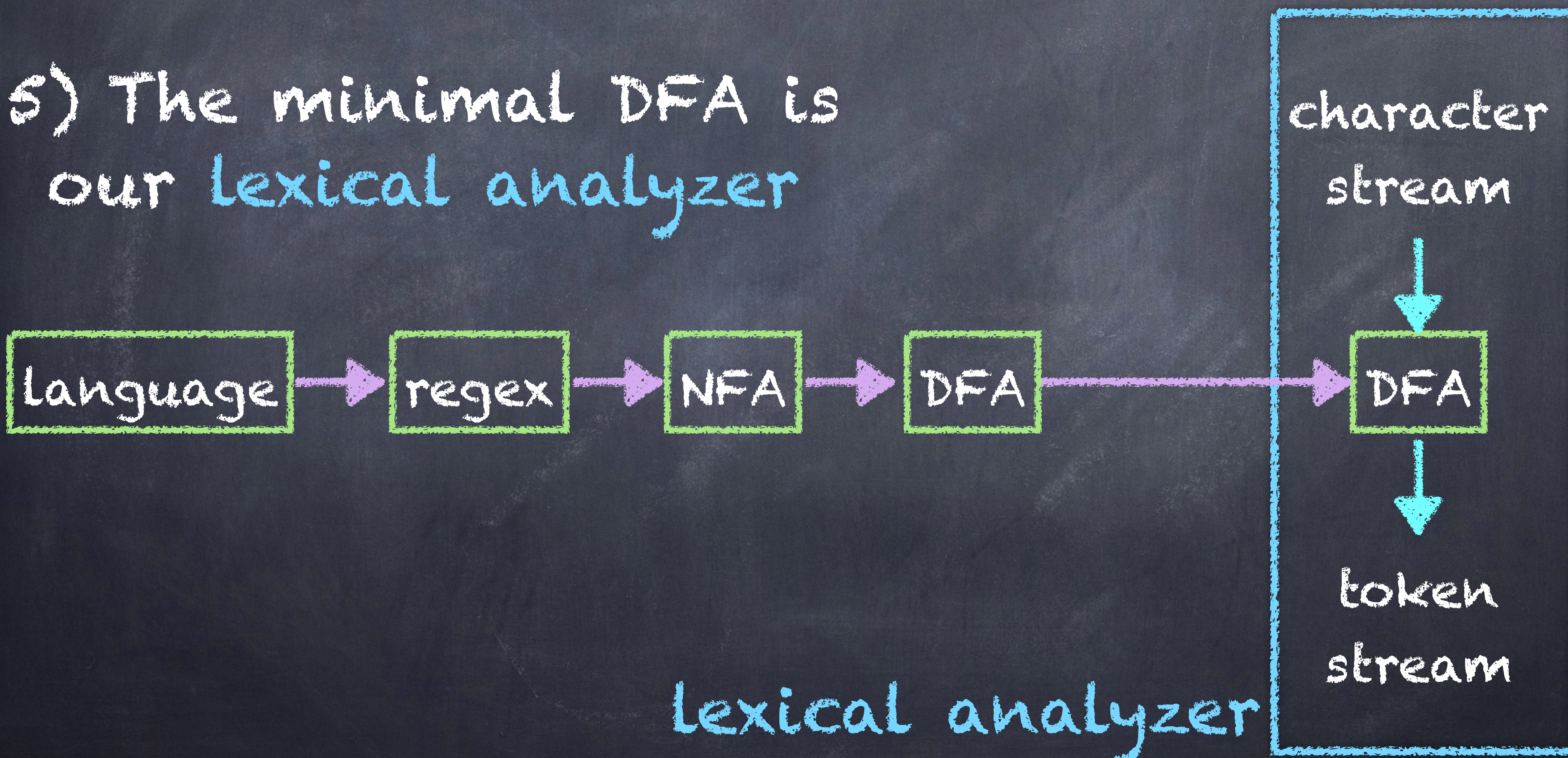
Process of building lexical analyzer

5) transform DFA to a minimal DFA

language → regex → NFA → DFA → DFA

Process of building lexical analyzer

5) The minimal DFA is our lexical analyzer



FLEX generates a C program which implements the minimized DFA

Step 1:
Construct NFA from regex

Nondeterministic Finite Automata (NFA)

- A finite set of states S
- An alphabet Σ, $\varepsilon \notin \Sigma$
- $\delta \subseteq S \times (\Sigma \cup \{\varepsilon\}) \times \mathcal{P}(S)$ (transition function)
- $s_0 \in S$ (a single start state)
- $F \subseteq S$ (a set of final or accepting states)

Deterministic Finite Automata (DFA)

- A finite set of states S
- An alphabet Σ, $\varepsilon \notin \Sigma$
- $\delta \subseteq S \times \Sigma \times S$ (transition function)
- $s_0 \in S$ (a single start state)
- $F \subseteq S$ (a set of final or accepting states)

NFA vs DFA

transition function

- $\delta \subseteq S \times \boxed{(\Sigma \cup \{\varepsilon\})} \times \boxed{\mathcal{P}(S)}$
- $\delta \subseteq S \times \boxed{\Sigma} \times \boxed{S}$

no ε-transitions

no multiple transitions

A state is a circle with its state number written inside.

Initial state has an arrow from nowhere pointing in. State 0 is often the initial state.

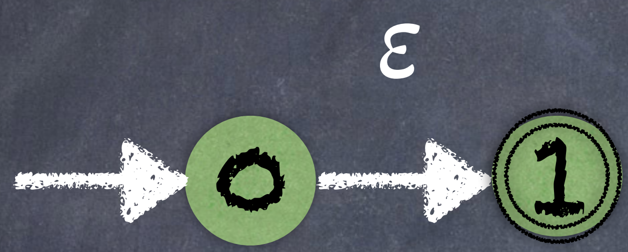
A final (accepting) state is drawn with a double circle.

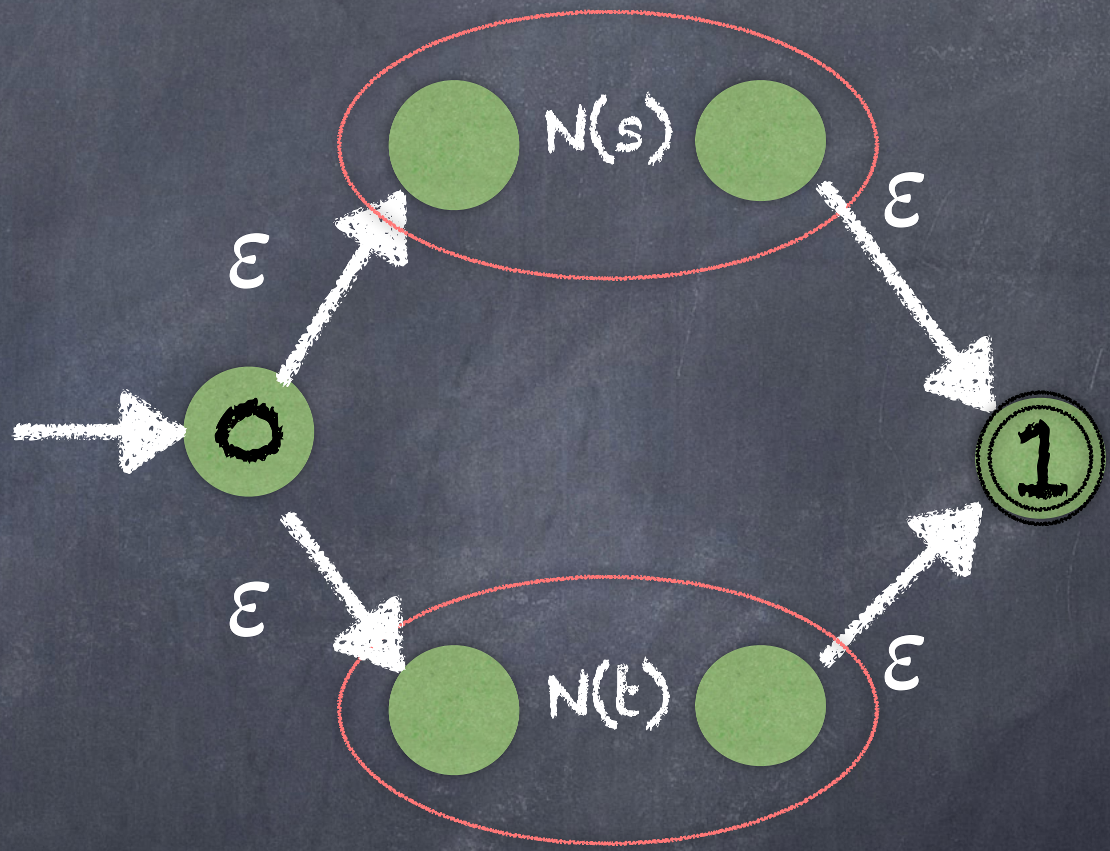
Arrows are labeled with ε ...

... or $a \in \Sigma$.

for each $a \in \Sigma$

Regex -> NFA



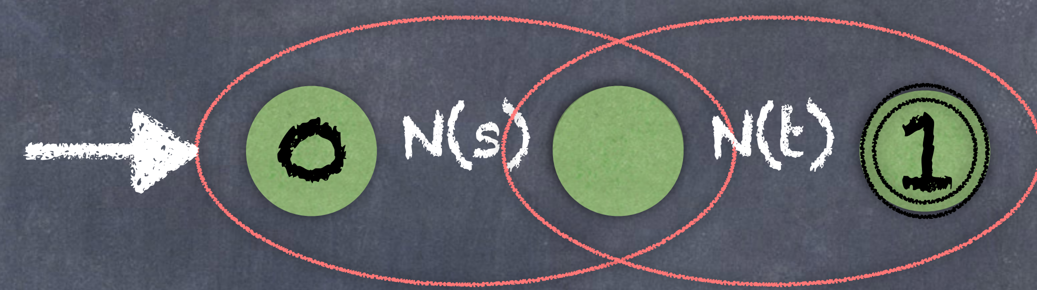


for each $a \in \Sigma$

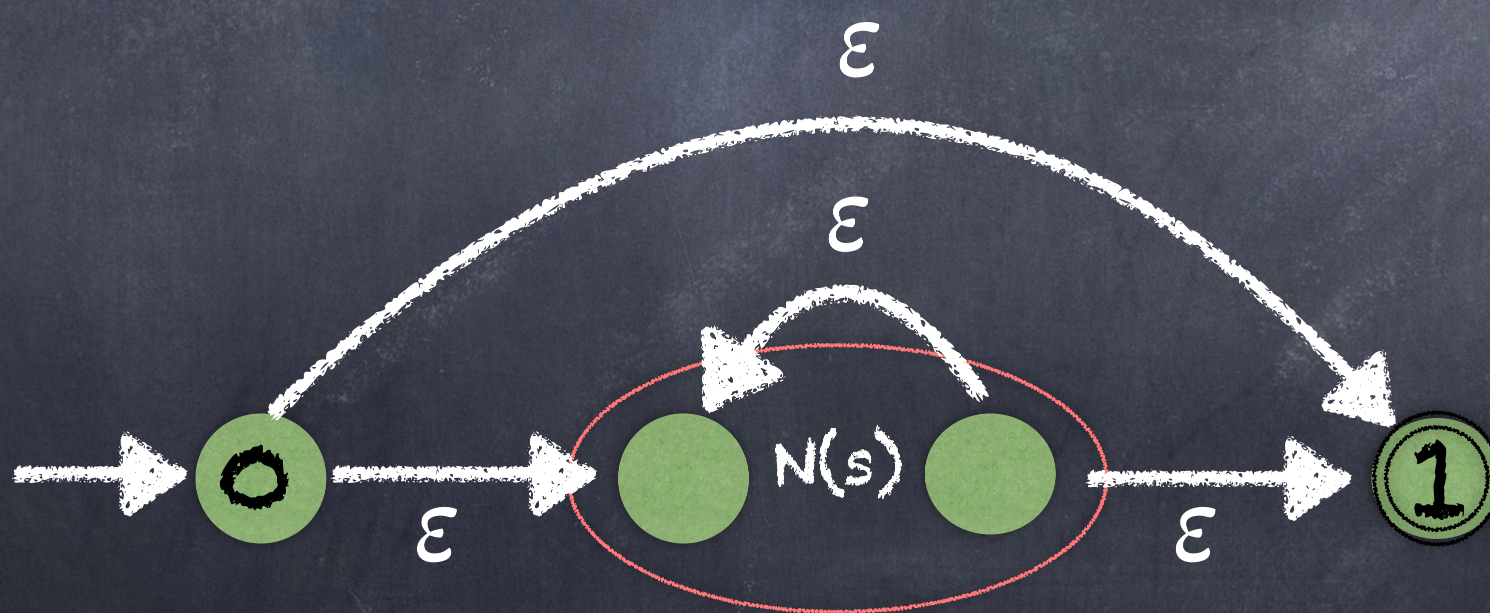
s | t

Regex -> NFA

st



s*



Simple example

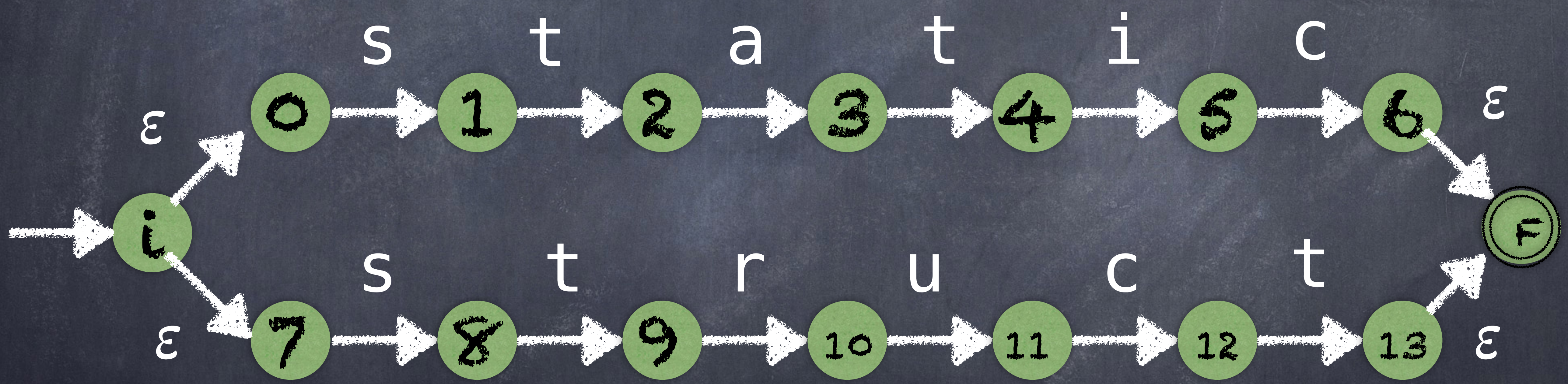
```
static
```

Simple example

static

Simple example

```
static
struct
```



Step 2:
Construct DFA equivalent to NFA

$\boxed{\text{NFA}} \rightarrow \boxed{\text{DFA}}$

Next time!

Step 3: DFA minimization

Next time!