CSE443
Compilers

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

Syllabus

- Academic Integrity (esp wrt teamwork)
- Values Statement

Textbook

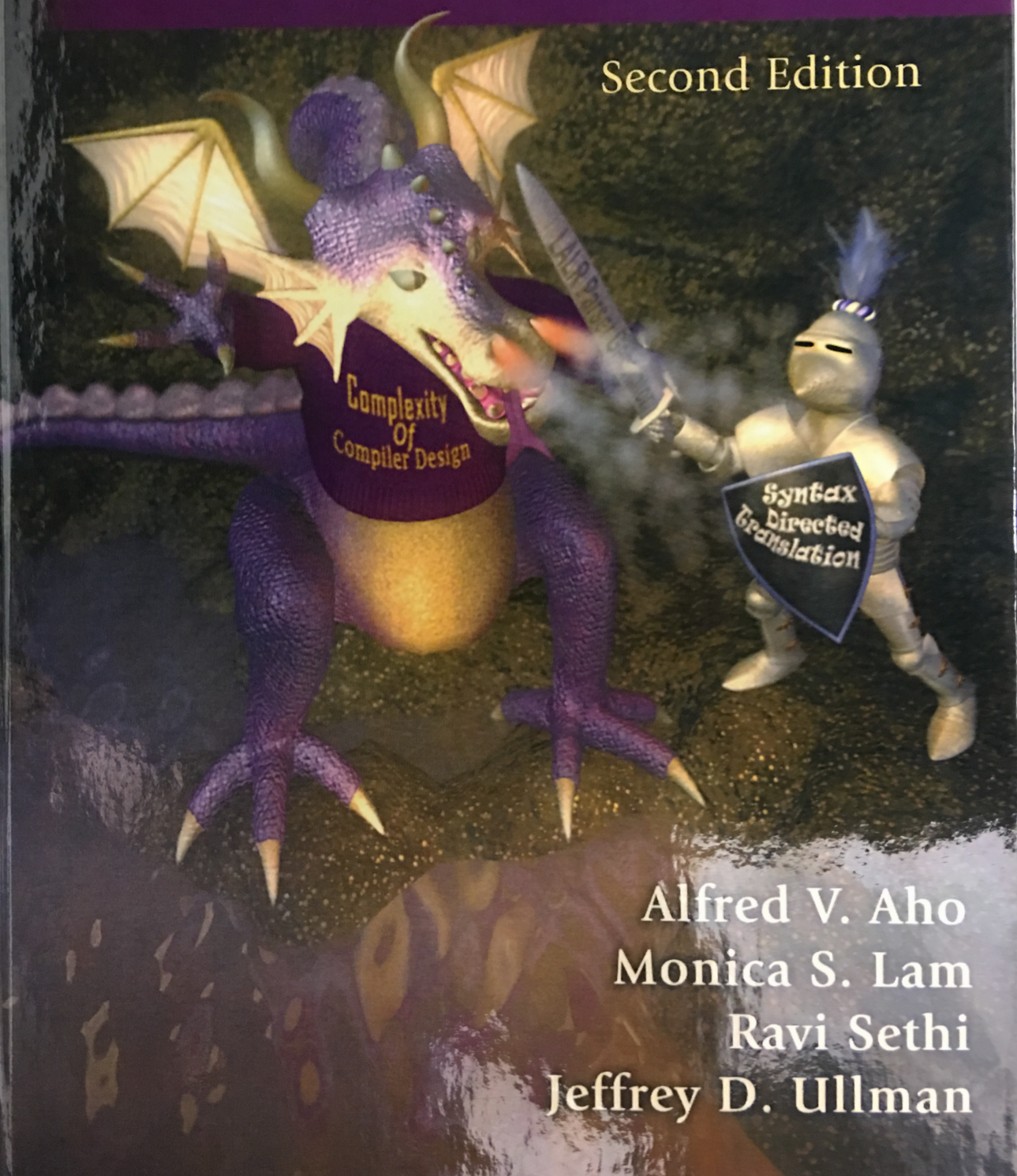
Classic text.

You should hang on to this one.

Compilers

Principles, Techniques, & Tools

Second Edition



Alfred V. Aho
Monica S. Lam
Ravi Sethi
Jeffrey D. Ullman

Team information

- If you have a team, please list members in Piazza post.
- PM meetings in recitation starting Tuesday next week

Project structure

- 4 sprints (see course schedule)
  - PMs assess teamwork/process
    - you provide peer evaluation
    - there is a non-binding "grading" of project functionality (think of it as a progress bar)
- Project grade determined by assessment at end of semester (your sprint 4 submission)

Teamwork process

- User stories
- Task breakdown
- GitHub usage
- etc.

THE "USER STORY"

- A construct of the agile methodology, this paradigm frames requirements as user-centric features, each addressing a specific need from a user perspective, with a rationale
- As a <user type> I want <functionality> so that <benefit>
- Each user story should also contain the acceptance criteria – how do you know when this story has been fulfilled, and how?
- Each user story will be broken into one or more **development** tasks

Slide courtesy of Alan Hunt

Deep understanding - ex 2

```
f() + g() * h();
```

What is the order of the function calls?

Must g be called before f?

Deep understanding - ex 2

```
f() + f() * f();
```

How many times will
f be called?

Could it be just once?

If it cannot be just once, is
order important?

Deep understanding - ex 2

```
f() + f() * f();
```

If the value of f() depends on mutable persistent state, then the value returned by each call can be different.

Deep understanding - ex 2

```
f() + f() * f();
```

If f is known to be
referentially transparent, then each
call to f() will produce the same value.

We can then compute f once, and use
its value multiple times.

Referential transparency and referential opacity are properties of parts of computer programs. An expression is called referentially transparent if it can be replaced with its corresponding value without changing the program's behavior. This requires that the expression be pure, that is to say the expression value must be the same for the same inputs and its evaluation must have no side effects. An expression that is not referentially transparent is called referentially opaque.

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

If f is known to be
referentially transparent, then each
call to f() will produce the same value.

We can then compute f once, and use
its value multiple times.

What determines program semantics?

```
#include <stdio.h>

int main() {
   int i = 0;
   int sum = 0;
   while (i <= 10) {
      sum = sum + i;
      printf("sum of integers from 0 to %d is %d.\n",i,sum);
      i = i + 1;
   }
}
```

What is this?

```
#include <stdio.h>

int main() {
   int i = 0;
   int sum = 0;
   while (i <= 10) {
      sum = sum + i;
      printf("sum of integers from 0 to %d is %d.\n",i,sum);
      i = i + 1;
   }
}
```

What is this?

```
#include <stdio.h>

int main() {
   int i = 0;
   int sum = 0;
   while (i <= 10) {
      sum = sum + i;
      printf("sum of integers from 0 to %d is %d.\n",i,sum);
      i = i + 1;
   }
}
```

```
/* La suite de Syracuse est définie ainsi :
- on part d'un entier ;
- s'il est pair, on le divise par 2 ;
- sinon, on le multiplie par 3 et on ajoute 1 ;
- on recommence la même opération sur l'entier obtenu, et ainsi de suite ;
- la suite s'arrête si on arrive à 1. */

syracuse :
   durée est un nombre
   e est un nombre
   début
e prend 14
tant que e != 1 lis
   durée prend durée + 1
   si (e mod 2) = 0, e prend e / 2
   sinon e prend e * 3 + 1
   affiche e
ferme
affiche "durée = {durée}"
```

More information

```
/* The Syracuse sequence is defined as follows:
- it starts with any natural number > 0
- if it is even, we divide by 2
- else we multiply by 3 and add 1
- the process is repeated on the result
- the process ends when the result is 1 */

void syracuse() {
  int iterations;
  int e;

  iterations = 0;
  e = 14;
  while (e != 1) {
    iterations = iterations + 1;
    if ( (e % 2) == 0 ) e = e / 2;
    else e = e * 3 + 1;
    printf("%d\n",e);
  }
  printf("iterations = %d\n",iterations);
}
```

Linotte
French keywords

```
syracuse :
   durée est un nombre
   e est un nombre
   début
      e prend 14
      tant que e != 1 lis
         durée prend durée + 1
      si (e mod 2) = 0, e prend e / 2
      sinon e prend e * 3 + 1
      affiche e
ferme
affiche "durée = {durée}"
```

C
English keywords

```
void syracuse() {
  int iterations = 0;
  int e;

  e = 14;
  while (e != 1) {
    iterations = iterations + 1;
    if ( (e % 2) == 0 ) e = e / 2;
    else e = e * 3 + 1;
    printf("%d\n",e);
  }
  printf("iterations = %d\n",iterations);
}
```

Linotte
French keywords

```
syracuse :
   durée est un nombre
   e est un nombre
   début
      e prend 14
      tant que e != 1 lis
         durée prend durée + 1
      si (e mod 2) = 0, e prend e / 2
      sinon e prend e * 3 + 1
      affiche e
   ferme
   affiche "durée = {durée}"
```

C
English keywords

```
void syracuse() {
  int iterations = 0;
  int e;

  e = 14;
  while (e != 1) {
    iterations = iterations + 1;
    if ( (e % 2) == 0 ) e = e / 2;
    else e = e * 3 + 1;
    printf("%d\n",e);
  }
  printf("iterations = %d\n",iterations);
}
```

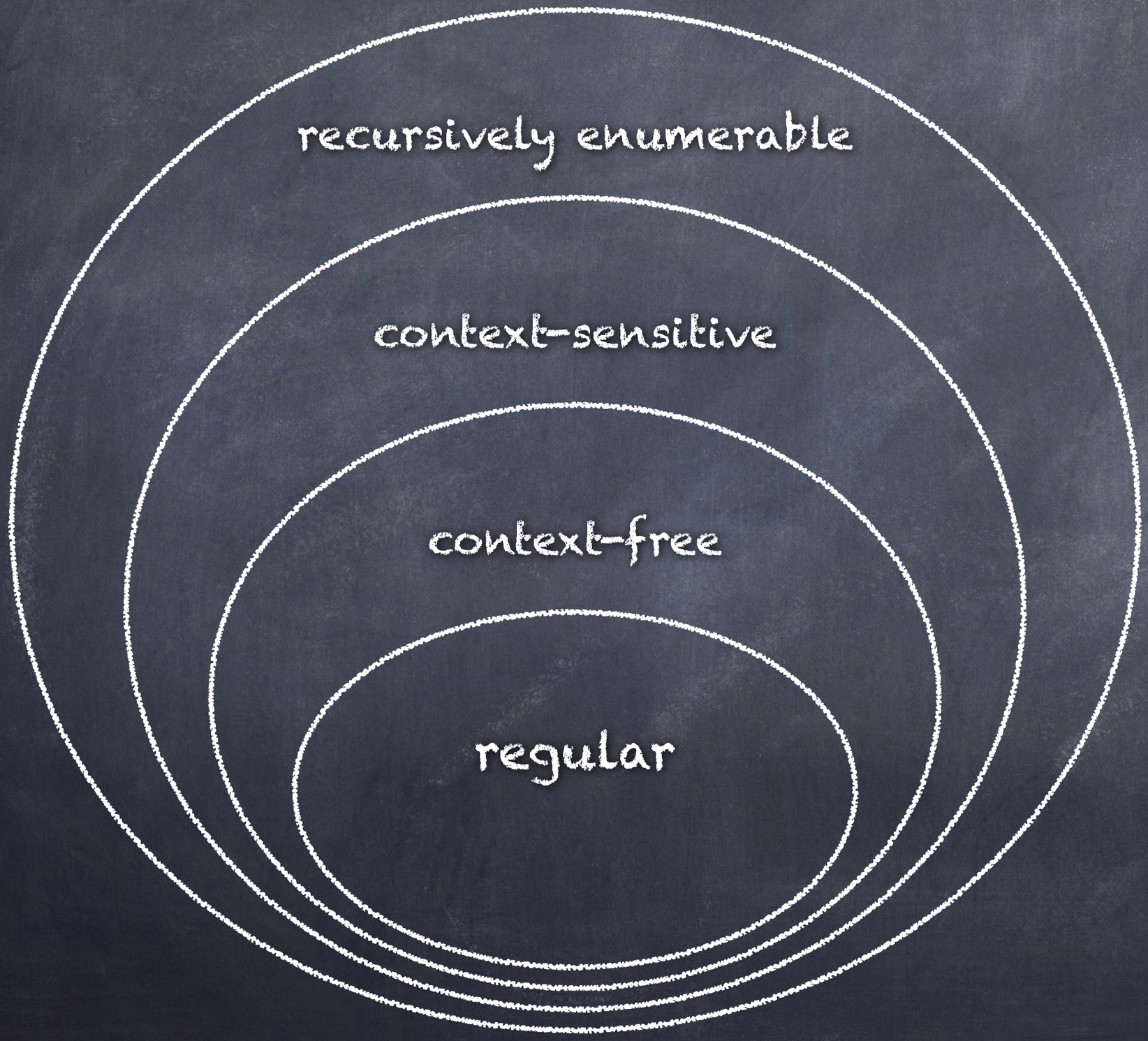
- Keywords have no inherent meaning.
- Program meaning is given by formal semantics.
- Compiler must preserve semantics of source program in translation to low level form.

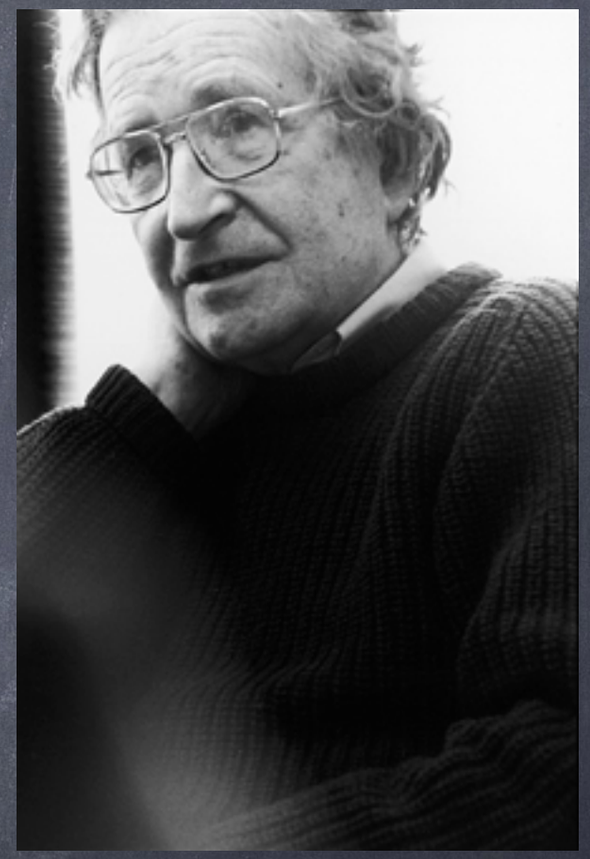
Syntax and semantics

- Syntax: program structure
- Semantics: program meaning
- Semantics are determined (in part) by program structure.

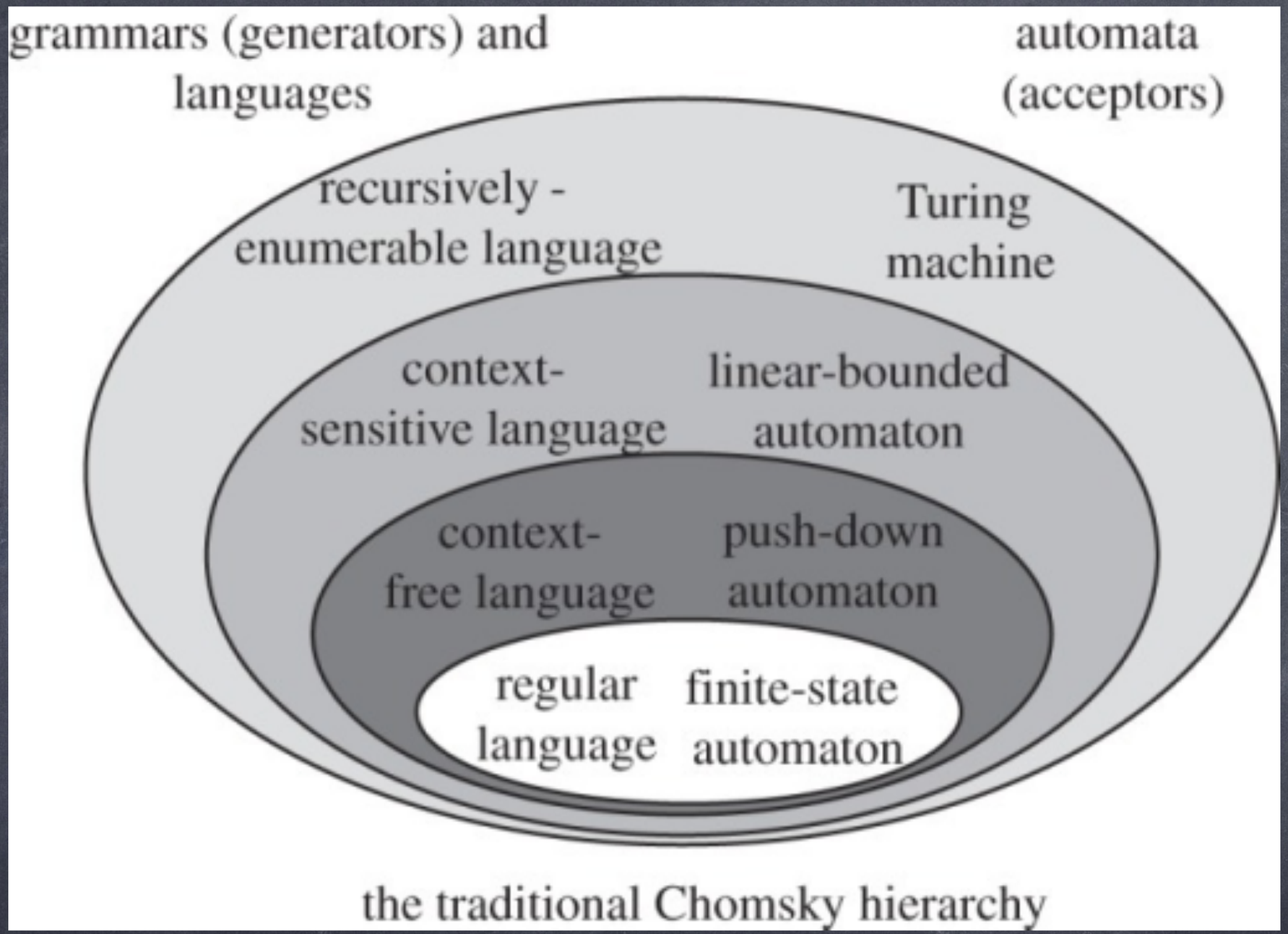
Languages: the Chomsky hierarchy

"On Certain Formal Properties of Grammars" published 1959





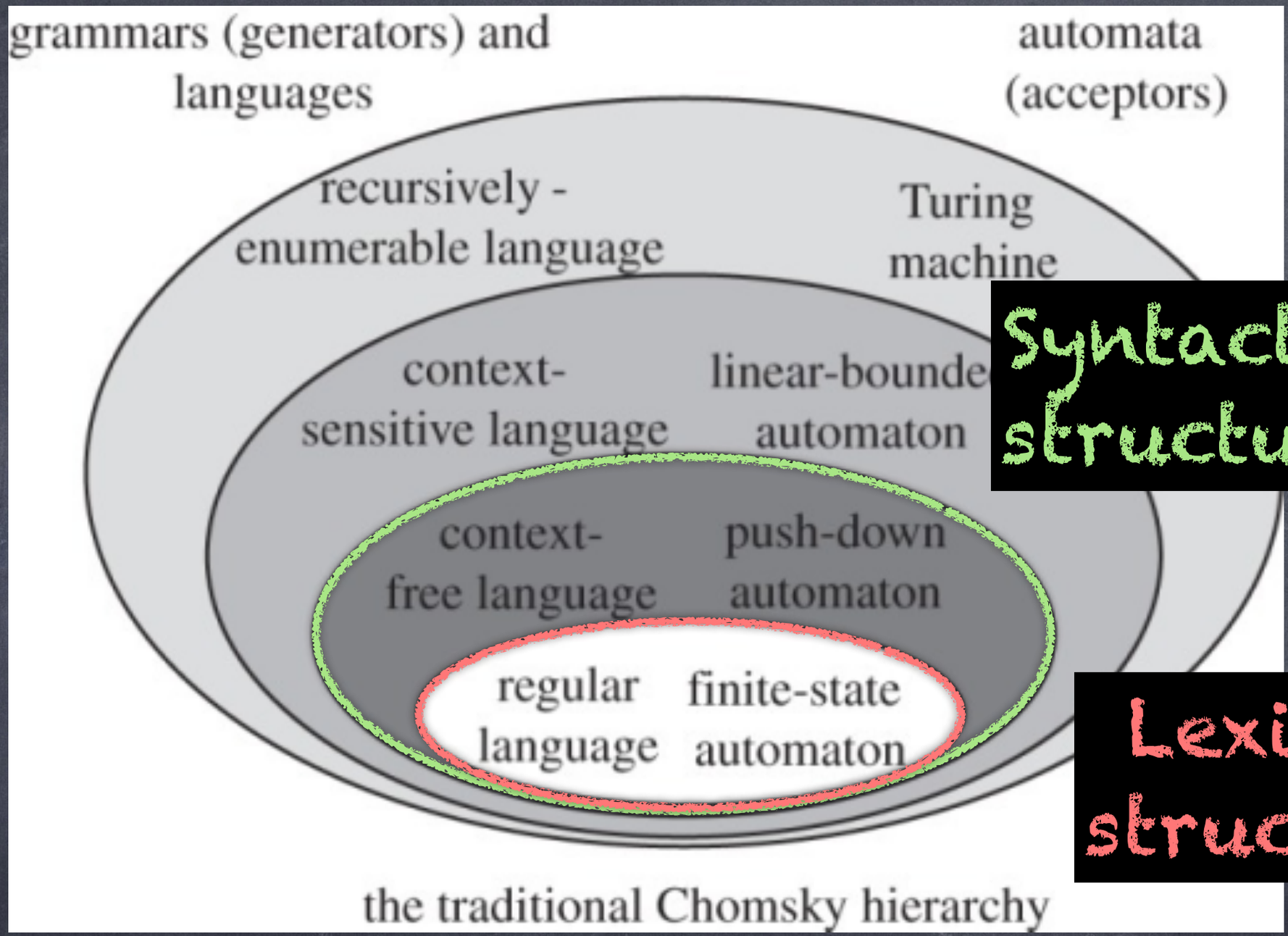
https://upload.wikimedia.org/wikipedia/commons/8/86/Noam_chomsky.jpg



the traditional Chomsky hierarchy

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)



Syntactic structure

Lexical structure

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)

Phases of a compiler

Lexical structure

Syntactic structure

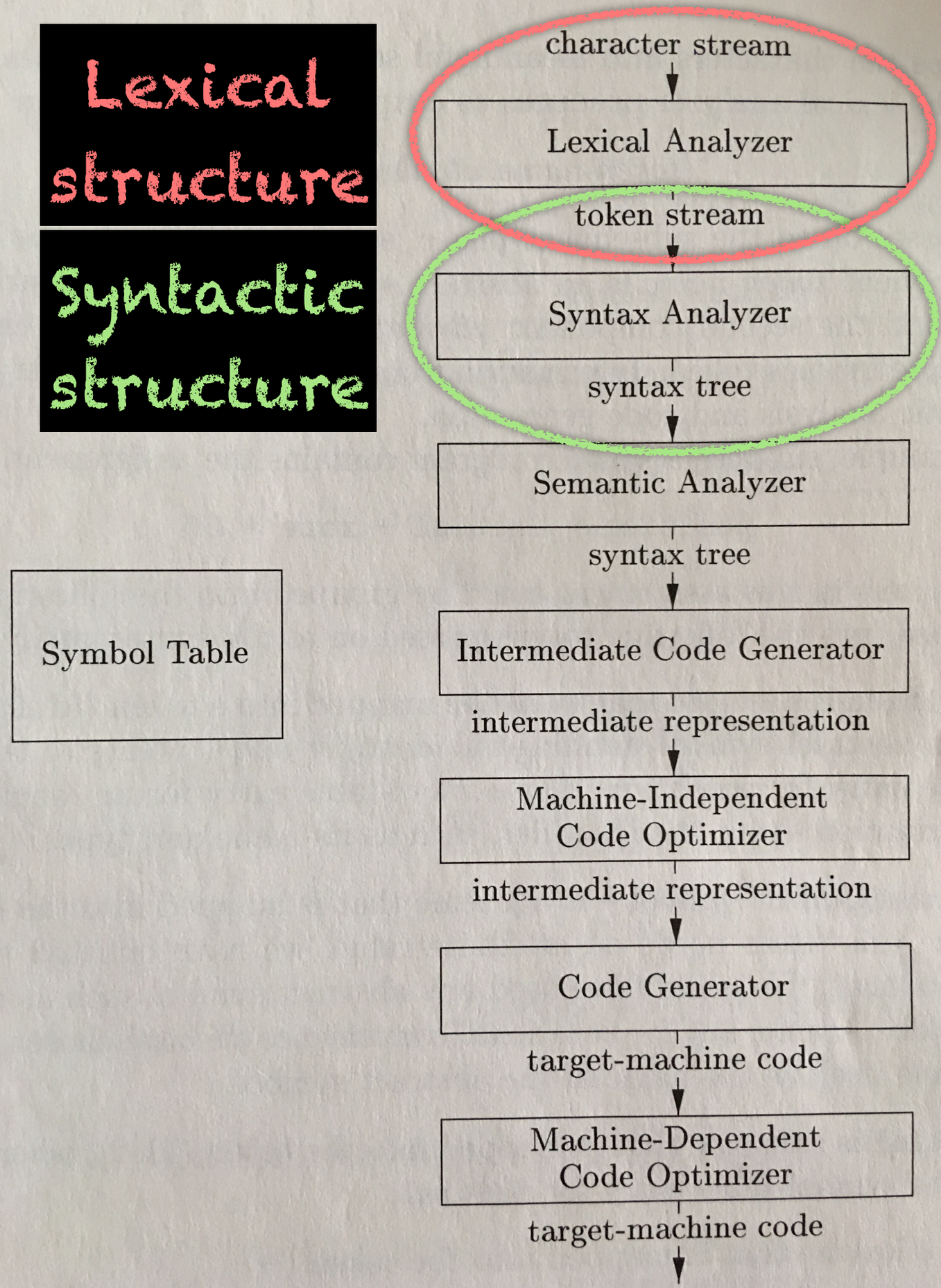


Figure 1.6, page 5 of text

Phases of a compiler

Lexical structure

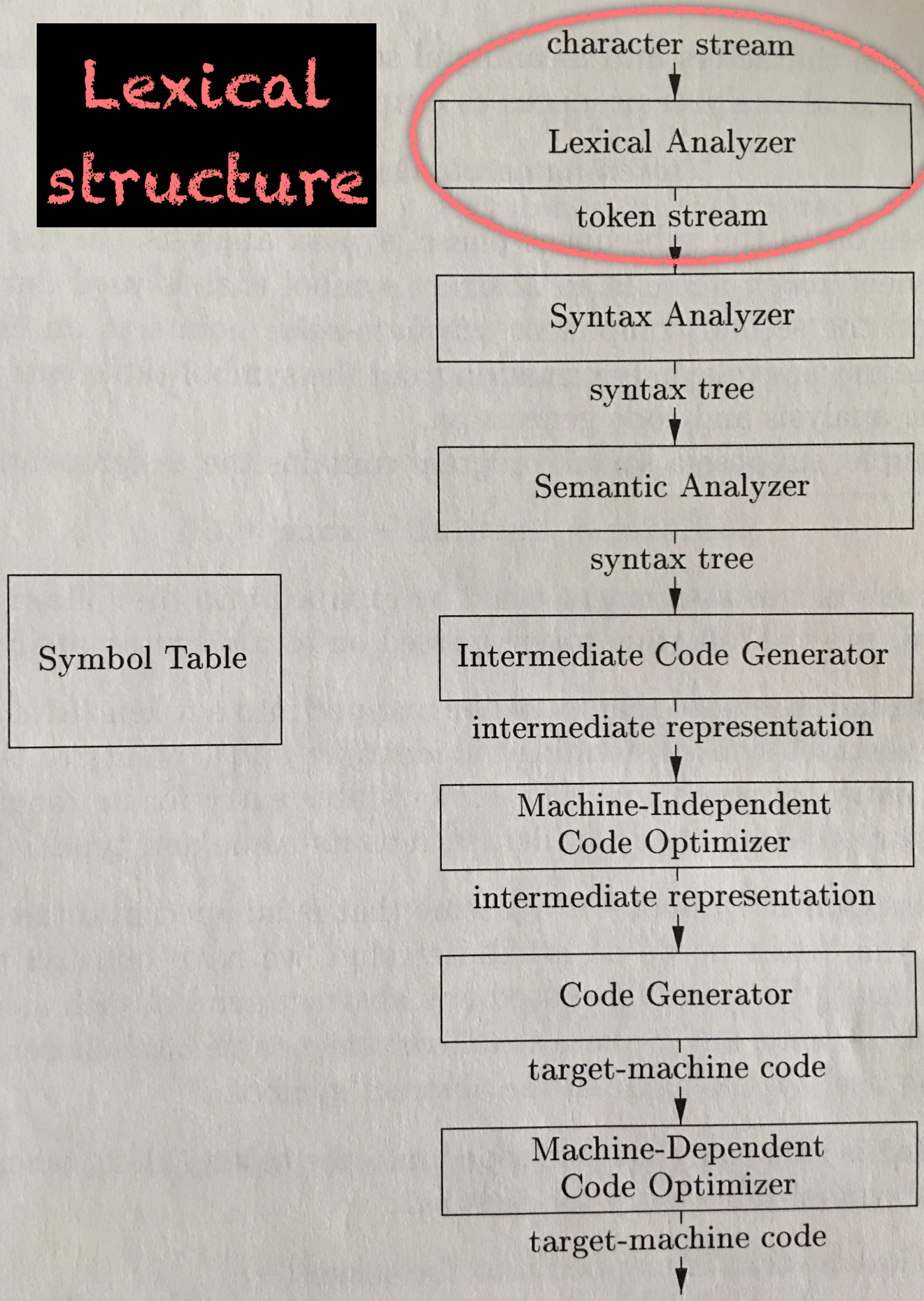


Figure 1.6, page 5 of text

Lexical Structure

```
int main(){
```

Lexical Structure

```
int main(){
```

character stream

```
i n t   m a i n ( ) {
```

Lexical Structure

```
int main(){
```

character stream -> token stream

```
i n t ␣ m a i n ( ) {
```

id(“int”) id(“main”) LPAR RPAR LBRACE

Lexical Structure

tokens

- keywords (e.g. static, for, while, struct)
- operators (e.g. <, >, <=, =, ==, +, -, &, .)
- identifiers (e.g. foo, bar, sum, mystery)
- literals (e.g. -17, 34.52E-45, true, 'e', "Serenity")
- punctuation (e.g. { , } , (,) , ;)

Describing lexical structure

- We need some formal way of describing the lexical structure of a language.

meta vs object language

- object language: the language we are describing
- meta language: the language we use to describe the object language

meta vs object language

- How do we distinguish between the two?

meta vs object language

- use quotes (meta vs 'object')
  - punctuation (e.g. '{' , '}' , '(' , ')' , ';')

- use font or font property (meta vs object)
  - punctuation (e.g. { , } , (,) , ;)

Languages & grammars

- Formally, a language is a set of strings over some alphabet
- Ex. {00, 01, 10, 11} is the set of all strings of length 2 over the alphabet {0, 1}
- Ex. {00, 11} is the set of all even parity strings of length 2 over the alphabet {0, 1}

Languages & grammars

- Formally, a grammar is defined by 4 items:
  1. N, a set of non-terminals
  2. Σ, a set of terminals
  3. P, a set of productions
  4. S, a start symbol
- $G = (N, \Sigma, P, S)$

Lexical analysis: a bird's eye view

{ for, while, x, factorial, ... }

language: a set of strings

$G = (N, \Sigma, P, S)$

grammar: rules for generating language

finite automaton

a machine for language

regular expression

regex: a form of grammar

C program

generated by FLEX

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)

   $X \rightarrow aY$

   $Y \rightarrow bX$

   $Y \rightarrow a$ <- a right linear grammar describing a regular language

   $X \rightarrow \varepsilon$

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

4. S, a start symbol

   $S = Y$

   $S \in N$