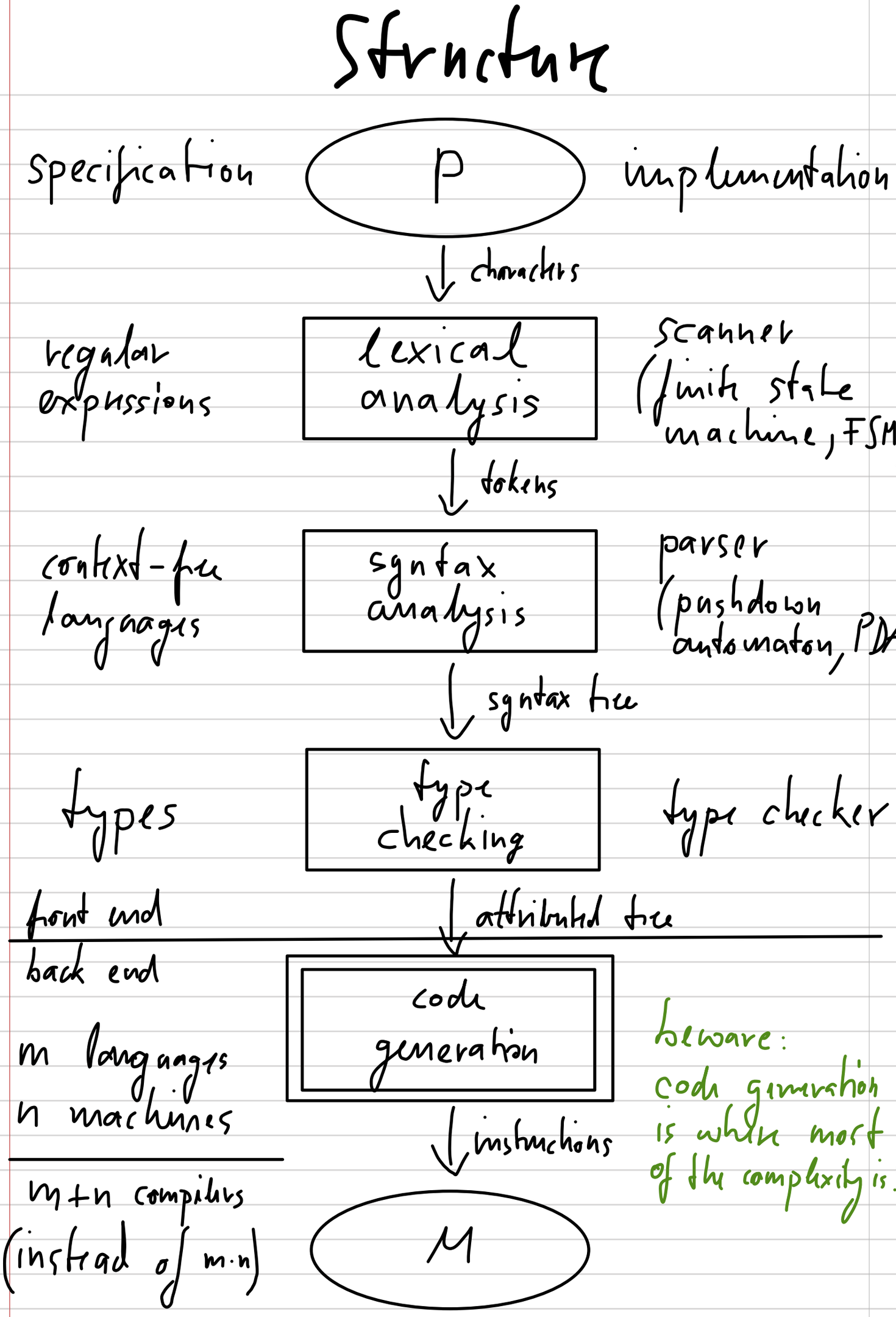
Structure


specification
P
implementation
characters
regular expressions
lexical analysis
scanner (finite state machine, FSM)
tokens
context-free languages
syntax analysis
parser (pushdown automaton, PDA)
syntax tree
types
type checking
type checker
front end
attributed tree
back end
code generation
beware: code generation is where most of the complexity is!
m languages
n machines
instructions
m+n compilers (instead of m·n)
M

Scanning and Parsing

```
main() {
 int i;
 i = 0;
 while (i < 10) {
  i = i + 1;
 }
 i = 11;
}
```

```
main()_{|
__int_i;|
__i_=_0;|
__while_(i_<_10)_{|
_____i_=_i_+_1;|
__}|
__i_=_11;|
}
```

sequence of symbols

scanning

whitespace

token for int keyword

```
main ( ) { int i ; i = 0 ; while ( i < 10 ) { i = i + 1 ; } i = 11 ; }
```

=

sequence of tokens

```
24-"main" 8 9 27
 43 24-"i" 17
 24-"i" 21 16-'0' 17
 33 8 24-"i" 18 16-'10' 9 27
  24-"i" 21 24-"i" 11 16-'1' 17
 28
 24-"i" 21 16-'11' 17
28
```

parsing

```
main
__int
  i
__=
  i
  0
__while
  <
    i
    10
  =
    i
    +
      i
      1
__=
  i
  11
```

token for number

token for identifier

token for while keyword

a scanner recognizes symbols (sequences of characters) and returns tokens (integers) that represent symbols uniquely

How to Construct a Scanner!

the scanner maintains a global variable currentCharacter which is initialized to the first character of the input program by invoking a library procedure:

readCharacter(); ← implemented in runtime

which reads the next character from the input program. the scanner is invoked by the parser through a procedure:

getSymbol(); (keep track of location in source (line, col.))

which returns the token that represents the next symbol (in another global variable). for each invocation of getSymbol() the scanner checks if currentCharacter already constitutes a valid symbol. if yes, the scanner invokes readCharacter() (to prepare for the next invocation of getSymbol()) and returns the appropriate token. if not, the scanner keeps invoking readCharacter() until it recognizes a valid symbol or returns an error.

1. define the set of valid symbols
(identifiers are sequences of letters and digits that start with a letter; numbers are sequences of digits; strings are sequences of printable characters in quotes)
2. define the set of keywords
3. define what a comment is
4. define symbol-to-token mapping
5. implement in your language

// or /* */

distinguish "=" from "==" !

group symbols of of the same "class"

check identifiers if they are keywords using a lookup table or code

Summary:

- scanning vs. parsing
  words sentences
- focus on scanner, no parsing yet
- write test input for unit testing (very important!)
- get started today! even if you have to rewrite later

including meaningless input, syntactically correct but also syntactically incorrect!