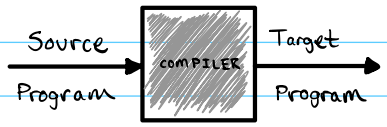
1. Introduction

a **computer program** is expressed in a **programming language** and transforms an input to an output via the **semantics** of the **symbols** of the **programming language**

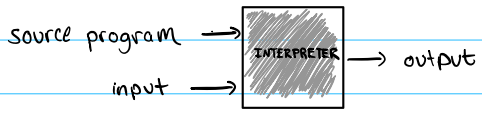
syntax: there is a formal set of rules that describes the set of all valid programs that can be written in terms of the **symbols** of the language

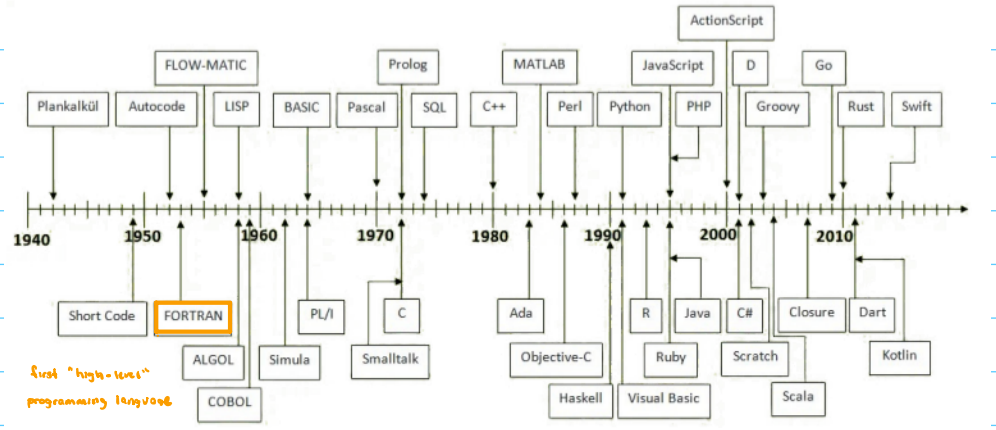
semantics: gives formal meaning to the **symbols** in a language

compiler translates programs written in one programming language into a program written in another programming language

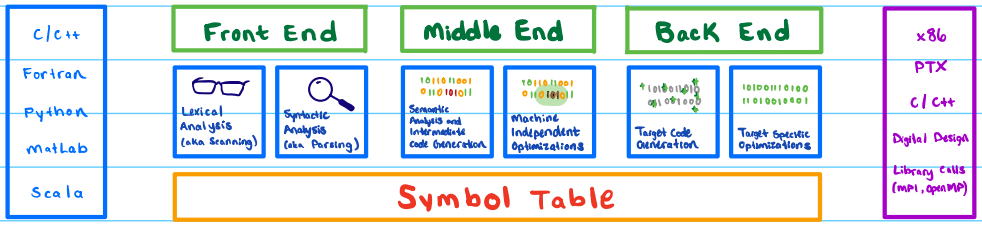


interpreter directly executes the semantics of the symbols in a source program in response to input as its being supplied





compilation overview



2. Lexical Analysis

the process of breaking down a large text into smaller parts
(such as words, phrases, or symbols) "source tokenization"

deals with "lower level" syntactic structure
strings with same structure get aggregated into the same class
implemented as the scanner
scanner invoked by parser

can be written
- by hand OR - using a scanner generator with reg-ex

SCANNING input: source program output: stream of tokens

- decompose the input file (stream of chars) into a stream of strings (lexemes)
- ignore white space (tab, return, etc...)
- assign a token (category) to each string
- job performed by the scanner
- automatic tools take regular expressions and produce compilable code (a DFA)
  (scanner generators)
  (e.g. Lex, Flex, JFlex, etc...)

symbol table keeps track of variable names + values

```
for (int i = 0; i < 10; i++)
```

① → "for", "(", ..., "++", ")"

② → KEYWORD_FOR, LEFT_PAR, ..., PLUS_PLUS, RIGHT_PAR

numbers?: "(+|-)?[0-9]+\.?[0-9]+" --> NUMBER

removes comments and reports compiler errors

RE Regular Expressions

denoted by:
- a character
- the empty string ε
- concatenation / "|" or operator
- * Kleene star

number → integer | real
integer → digit digit *
real → integer exponent | decimal (exponent | e)
decimal → digit *(. digit | digit .) digit *
exponent → (e | E)(+ | - | e) integer
digit → [0-9]

SCANNER RULES

recognizable strings:

1. Key words — for, while, if, switch, return
   identifiers — i, my_sum, _count_, sum
   numbers — integers + floating point
   operators — =, +=, +, ++, <, <=, !=, ...
   other — (,), {, }, [,], ;

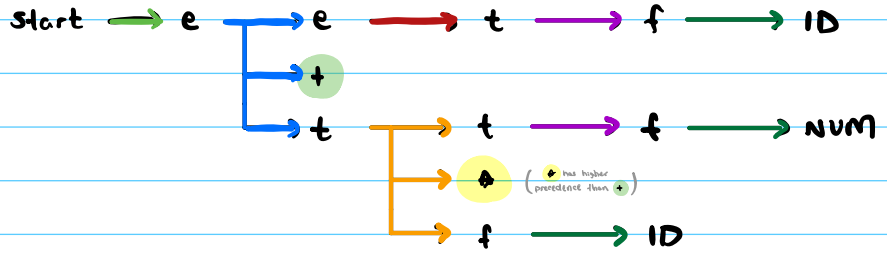
deterministic finite automation

A: semi-mechanical, pure DFA (usually realized as nested case statements)
B: table-driven DFA: write rules, generator produces code
C: ad-hoc: very case specific

3. Syntactic Analysis (parsing)

goal: match stream of tokens to the grammar of the programming language

input: a stream of tokens output: abstract syntax tree



rules: Start → e

e → t | e → e + t

t → f | t → t * f

f → NUM | f → ID

CONTEXT FREE GRAMMAR: rules applicable independently from "context"

(similar to english tuples: WORD → char | char char* PUNCTUATION → .|,|:|;|?|!|)

four tuples (VTPS): Variables (non-terminals) Terminals

Production Rules Starting Symbol

expression

expr → id | number | -expr | (expr) | expr op expr (recursively defined)

operation

op → + | - | * | /

1. begin with starting symbol
2. chose a production with the starting symbol on the left-hand side
3. replace starting symbol with right-hand side of production
4. chose a non-terminal A in resulting string
5. chose a production P with A on the left-hand side, and replace A with the right-hand side of P
6. return to 4 recursively until parsing is complete

LL parser (Left to Right, Leftmost derivation) top-down parse

LR parser (Left to Right, Rightmost derivation) bottom-up parse

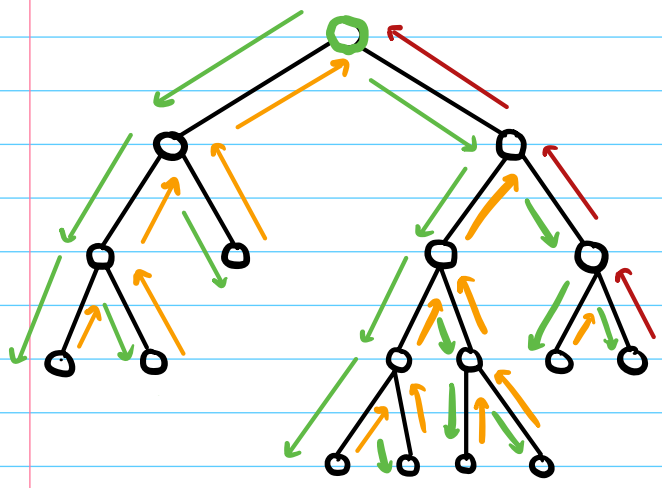
Leftmost derivation: always derive from the first non-terminal { L → R

Rightmost derivation: found in the sentential form, moving from: { R → L

(Sentential form: only terminal symbols)

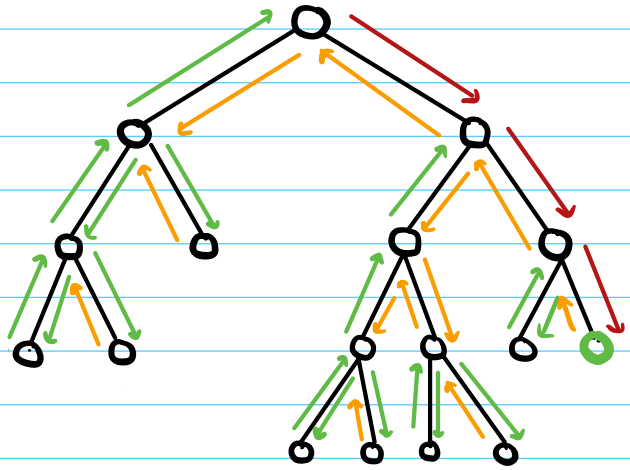
"capturing structure" A-B-C ⇒ (A-B)-C Left Associativity | A-(B-C) Right Associativity

LL parser



predictive, recursive-descent

LR parser



shift-reduce

LL PARSING TREE EXAMPLE

to parse: n * n

production rules

1. E → TR
2. R → ε
3. R → +E
4. T → FS
5. S → ε
6. S → *T
7. F → n
8. F → (E)

Table D

	n	+	*	(	)	$
E	1			1		
R		3	2		2	2
T	4			4		
S		5	6		5	5
F	7			8		

D(E, n) = 1

D(T, n) = 4

D(F, n) = 7

✓↓

D(S, *) = 6

✓↓

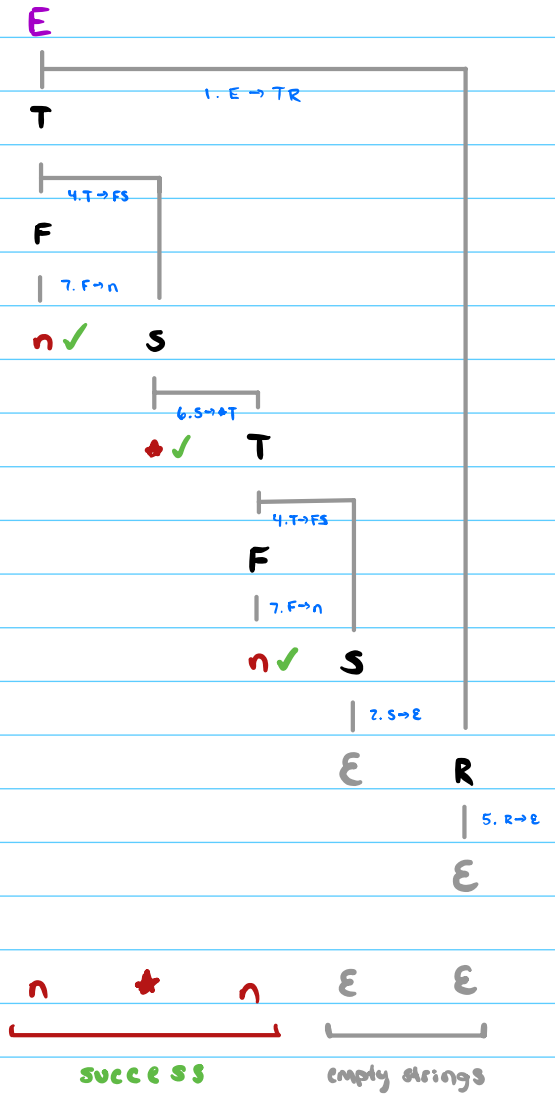
D(T, n) = 4

D(F, n) = 7

✓↓

2. R → ε

5. S → ε



LR PARSING EXAMPLE

productions marked with a (.) become LR items

the LR item tells us in which production we can be, and in what position

1. e → .e + t	1. e → e + t	1. e → e + t	1. e → e + t	1. e → e + t
2. e → t	2. e → .t	2. e → t	2. e → t	2. e → t
3. t → t * f	3. t → t * f	3. t → .t * f	3. t → t * f	3. t → t * f
4. t → f	4. t → f	4. t → f	4. t → .f	4. t → f
5. f → (e)	5. f → (e)	5. f → (e)	5. f → (e)	5. f → .(e)
6. f → ID	6. f → ID	6. f → ID	6. f → ID	6. f → .ID
7. f → NUM	7. f → NUM	7. f → NUM	7. f → NUM	7. f → .NUM

basis

closure

* I DONT UNDERSTAND THE REST OF LR PARSING *

Quiz 1

1. what compilation phase is responsible for detecting and reporting syntax errors
   a. Syntactic Analysis

2. what compiler component permits the use of specific identifiers
   a. Lexical Analyzer (aka Scanner)

3. the decision to replace (or not) expressions occurs in what compilation phase
   a. Target Specific Optimizations

4. sort the compilation phases
   a. 1. Lexical Analysis 2. Syntactic Analysis 3. Syntactic Analysis + Intermediate Code Generation
      4. Machine Independent Optimizations 5. Target Code Generation 6. Target Specific Optimization

5. what is considered the first "high level" programming language
   a. FORTRAN

6. what is the output of lexical analyzer
   a. set of tokens

7. a VPN ____
   1a. stands for Virtual Private Network
   2a. is used to appear as though you're physically on a network you're not

8. on Linux, return to home directory from anywhere with _____
   a. cd /~

9. command to compile hello.java using CLI
   a. javac hello.java

10. command to get from directory home/A/B/C to home/A/E/F
    1a. cd ../../E/F
    2a. cd ~/A/E/F

11. command to create new text file

a. touch book.txt

12. which are lexemes

a. Identifiers, constants, keywords

13. token category of 3 in sum = 3 + 2

a. Integer Literal

14. lexical analyzer takes _____ as input, returns _____

1a. Source Program

2a. Stream of Tokens

15. what goes over the characters of the lexeme to produce a value

a. Scanner

Quiz 2

for 1-6 consider the following grammar:

1) v → v[expr]
2) v → ID
3) expr → NUM
4) expr → v

① top-down, right-most derivation of the string "ID [NUM] [ID]"

a. v ← STARTING SYMBOL

1. v → v[expr]

v[expr]

4. expr → v

v[v]

(was not included in answer on quiz?)

2. v → ID

v[ID]

1. v → v[expr]

v[expr][ID]

3. expr → NUM

v[NUM][ID]

2. v → ID

ID[NUM][ID]

SUCCESS

← terminal values