E.g. ② "if8" matches both identifier as well as keyword "if" followed by number "8".

According to the priority, it will be matched as keyword "if" followed by number "8".

but solution: longest match rule

What we want: set of regular expression (language dependent) → LEX analyzer generator → LEX analyzer → automatic generated

the one-to-one (token type) & (regular expression) table

Input program → LEX analyzer → tokens

E.g. "lex" in Unix

Language independent

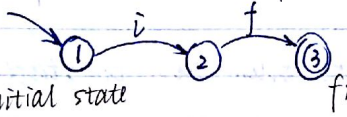
Finite Automata (FA) (finite state machine)

Regular expressions essentially can be transformed into FA

○ start state

◎ final state

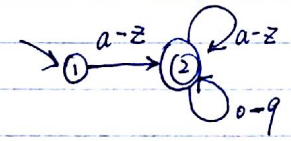
FA for IF



initial state

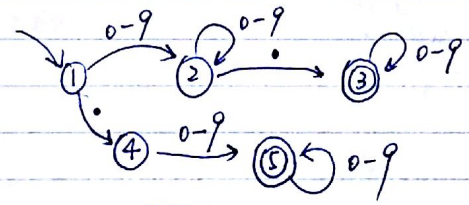
final state

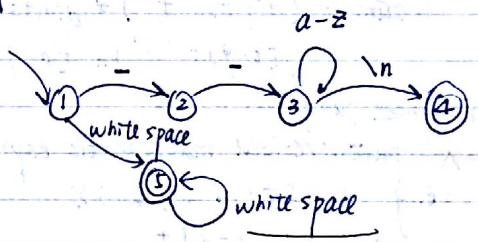
FA for ID



NUM

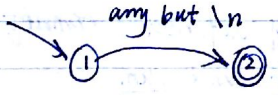
REAL





means space, new line, tab

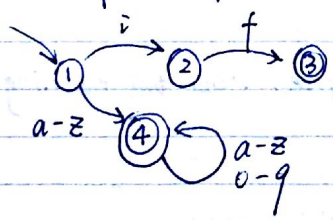
/* do nothing */



error

Each of the finite state machines can be implemented by switch in C.

Combine IF & ID FAs



This cannot work, since it cannot determine which way to go when see a "i" in state 1.

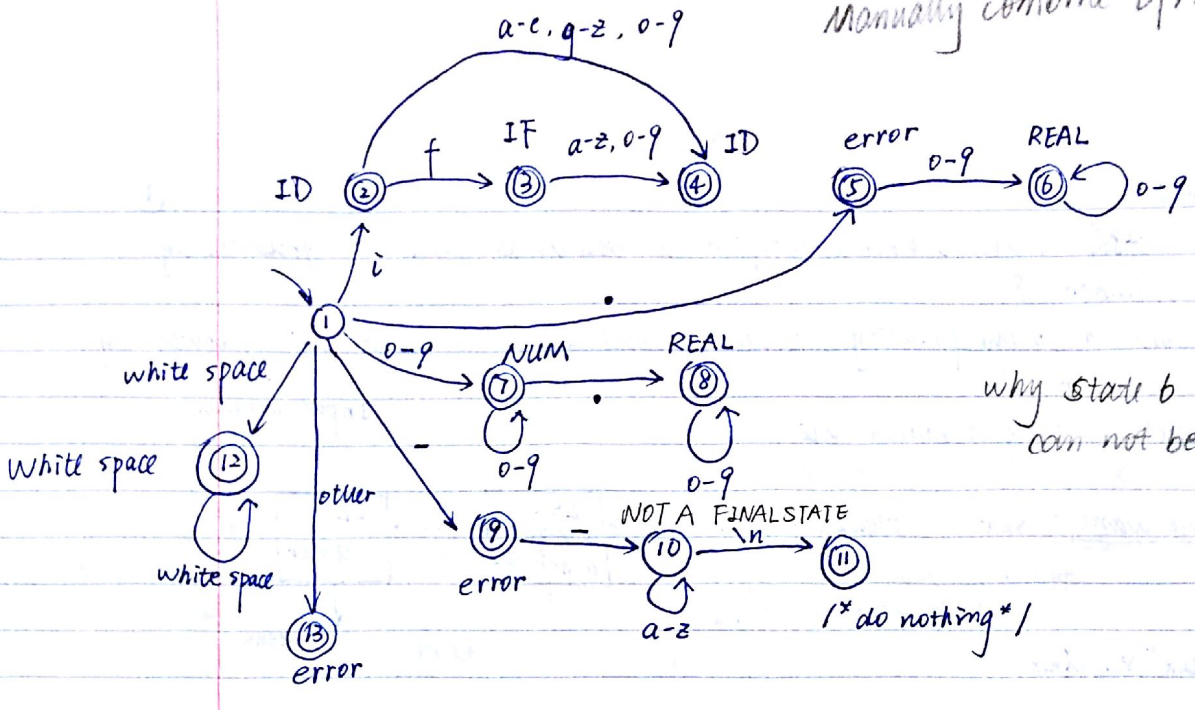
Non-deterministic FA (NFA)

→ Unimplementable

What we need is DFA (Dedermistic FA)

It is possible to convert NFA to DFA, (but will not be covered in this course)

Manually combine DFA in exam



why state 6 and state 8 can not be combined ?

In this combined FA, what happens when there is no transition for the character we see next ?

→ Take any action in the current state and then move to state 1, ~~meanwhile~~ then, look at the same character again.

2.12 digits = [0-9]+ How to recognize sum

sum = (digits "+") * digits not regular expression

= ([0-9]+ "+") * [0-9]+ regular expression

How to recognize fully parenthesized mathematic expression

E.g. ((15+17)+(14+3))

digits = [0-9]+

sum = expr "+" expr

expr = "(" sum ")" | digits > recursively defined, cannot get rid of variables like expr and sum

It is impossible to recognize fully parenthesized mathematic expression in regular expressi
Because we you have to track the number of open parenthesis, ~~but~~ which can be infinite.

Context-free grammers (CFGs)

~~the mean~~ natural languages are not context free, since the meaning of one sentence may depend on the previous sentences.

CFGs → Terminals (usually tokens in language)

→ Non-terminals (variable) e.g. expr, sum

→ Production rules which define non-terminals in terms of terminals as well as non-terminals.

e.g. 1. sum = expr "+" expr
expr = "(" sum ")" | digits → { 2. expr = "(" sum ")" | 3. expr = digits } | eliminate alternation "|"

1, 2, 3 are three production rules.

1. sum → expr plus expr
(expr: non-terminals; plus: terminals; expr: non-terminals)
2. expr → LPAREN sum RPAREN
(LPAREN: terminals; sum: non-terminals; RPAREN: terminals)
3. expr → digits

※ CFGs can be recognized by stack machine, which is infinite.

Derivation

Example of a complete CFG (Page 41)

S → S ; S S stands for statement or statements
S → id := E → assignment in the language
S → print (L)

E → id
E → num
E → E + E
E → (S, E)

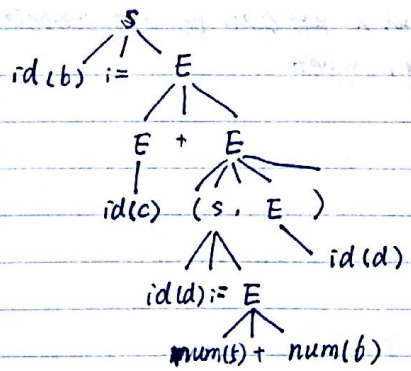
comma expression: b = c + (d = 5+b, d); ⇒ b = c + 11
(d = 5+b: statement; d: expression)

L → E
L → L, E

derivation:
S
id := E
id := E + E
id := id + E
id := id + (S, E)
⋮

derivation: left most / right most / neither

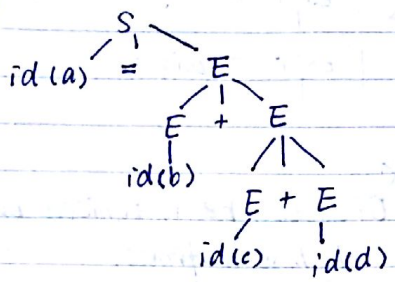
parse tree



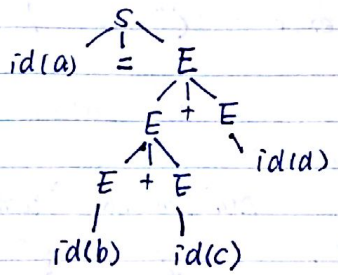
Don't want ambiguous grammar where ∃ more than one parse tree for a context.
∃ there exists

That grammar is ambiguous

E.g. $a = b + c + d$



$a = b + (c + d)$



$a = (b + c) + d$

You may end up with different results when using floating point.
In C, associate by left is always assumed.

2.19 A grammar is ambiguous if there exists a string, which has more than one parse trees.

grammar on P44:

Expand on expressions (E)

$E \to id$
$E \to num$
$E \to E \times E$
$E \to E / E$
$E \to E + E$
$E \to E - E$
$E \to (E)$

ambiguous example: $a = 1 - 2 - 3$

which means the first expression will be correct

can be: $a = (1-2)-3 = -4$

or: $a = 1-(2-3) = 2$

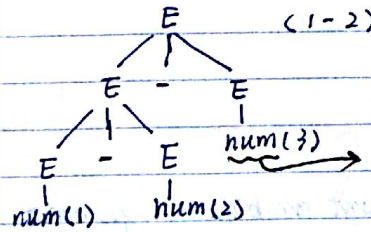
We say an operator $\oplus$ is associative if: $(a \oplus b) \oplus c = a \oplus (b \oplus c)$

+ and × are associative, − and / are not associative

However, when operating float, + and × can be not associative.

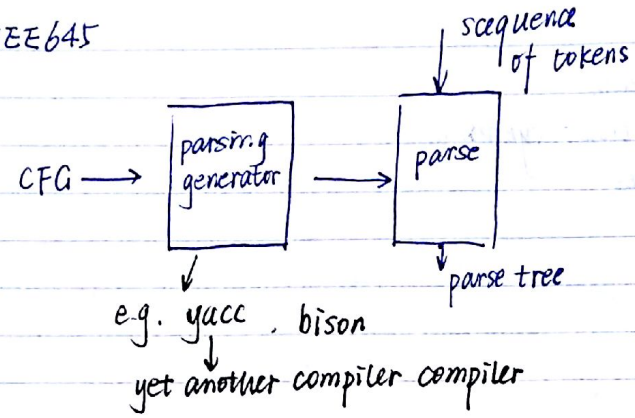
extremely tiny number

$(1-2)-3$



num is a terminal since num is a token

ENEE645



Replace $E \to E - E$ by $E \to E - T$ encapsulate associativity from left

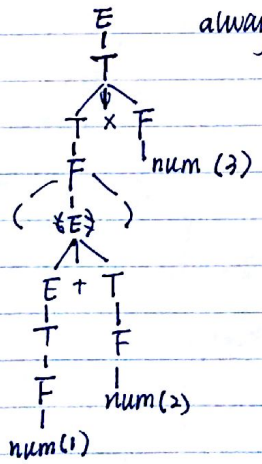
Unambiguous grammar | CFG : Context-free Grammar

$E \to E + T$	$T \to T \times F$	$F \to id$
$E \to E - T$	$T \to T / F$	$F \to num$
$E \to T$	$T \to F$	$F \to (E)$ parenthesized expression
lowest priority	next lower priority	highest priority

give you associativity from left

the three layers give the order of ~~express~~ operation

$(1+2) \times 3$



always starts from expression

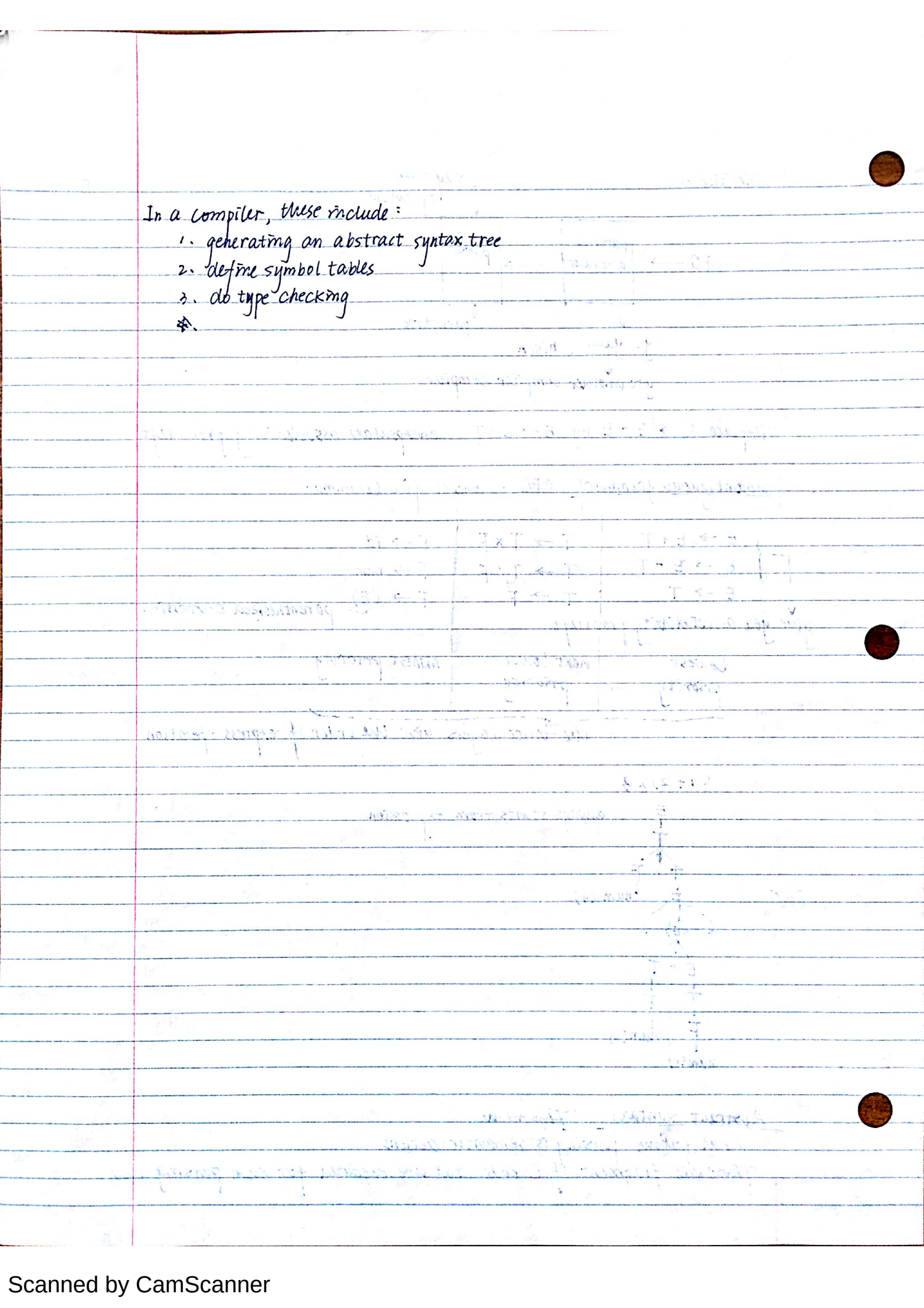
Abstract syntax | ~~This is w~~

What follows parsing is semantic actions.

Those are fragment of C code that are executed for each parsing rule.

In a compiler, these include:

1. generating an abstract syntax tree
2. define symbol tables
3. do type checking
4.



```
int *p, a ;
 p = &a ;
   = a ;   ⟶ must demote a from register to memory
*p =
```

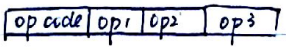
(compiler register, instead of hardware register)

pointers and arrays are the only two things that are considered as memory in compiler.

Assembly language: human readable one-to-one mapping machine code.

e.g. MIPS ISA

instructions are 32 bit



→ machine code

ADD R3, R2, R6; → assembly code (nice for human, but horrible for computer)

↘ (if translate in ~~ASCII~~ ASCII ☺, will be more than 104 bits)

LEX operation (The input of LEX is a sequence of ASCII characters)

Token type: P17 (the name) ←	Token type	Example
	ID (identifier)	foo, n14, a, last (if not a keyword)
	NUM	73, 0, 00, 082 (=82)
	REAL	66.1, 0.5, .5, 10.0, 10., 1e67, 5.5e-10
	IF	if
	FOR	for
	COMMA	,
	NOTEQ	!=
	LPAREN	(
	RPAREN	)
	⋮	⋮

Language for rules of tokens is "regular expressions"

2.5 Example on P17

```
float match0(char* s) /* find a zero */
{
    if (!strncmp(s, "0.0", 3))
        return 0.;
}
```

preprocessor will remove comments before LEX

Token:

```
FLOAT ID(match0) LPAREN CHAR STAR ID(s) RPAREN
LBRACE
    IF LPAREN BANG ID(strncmp) ...
```

an ID that has value "match0"

won't be recognized as ID(match) NUM because of the longest matching rule. (token with longest match wins)

Regular expressions

Language for regular expressions has:

① Symbols (E.g. a is symbol for itself)

② Alternation : ~~M/N~~ M|N (either rule M or rule N)

③ Concatenation: M.N or MN (rule for M followed by rule N)

④ Epsilon (ε) matches empty string " "

⑤ Repetition (M^*) Concatenation of zero or more instances of M → the repetition of the rule

⑥ M+ Concatenation of one or more instances of M

M+ ≡ MM*

E.g. (0|1)*0 → taking the rule "0|1" and ~~re~~ concatenation of zero or more ~~inst~~ times

0100 ✓ a member of this language

0 ✓

(0|1)*0 0101 ✗ not a member of this language

is called regular expression 0a10 ✗

E.g. b*(abb*)*(a|ε) doesn't have to end with a

strings of a's and b's with no concatenation a's

The difference between (a|ε) and (a|b) is that the latter one cannot have empty string.

set of regular expression ← (a|b)*aa(a|b)* : strings of a's and b's that contain aa somewhere

highest priority

Token type	regular expression	
IF	if	
ID	[a-z][a-z 0-9]*	(cannot start with a number, cannot have capital letters and underscore)
NUM	[0-9]+	(zero and zero onwards)
REAL	([0-9]+"."[0-9]*) | ([0-9]*"."[0-9]+)	not a concatenation (avoid . case)
/*do nothing*/ return no token	("--"[a-z]* "\n") | (" " | "\n" | "\t")+	space; nothing before and after the dot
error()	. → means any single character except new line	

lowest

2.10

Ambiguity

E.g. ① "if" matches both keyword & identifier

solution: rule priority

ENEE 645

If you get tons of files (Seperate compilation) compiling source files seperately into object files

```
gcc -c file1.c -o file1.o
...
gcc file1.o file2.o ... -o file.exe
```

(Linking takes much less time than compiling)

a make file can detect the files that have been changed and just re-compile them (not the others) and do the linking.

Reasons to learn about compilers:

→ profilenation of non-standard architectures
→ parallelism
- multi-core parallelism
- hyper threading (two threading inside a core)
- multi-media extension (MMX → → AVX)
- gaming console : IBM cell
- GPU

→ embedded system (a sample computer system in a device whose primary purpose is not computing)
→ ~~exclusive~~ explicit memory hierarchy
→ cybersecurity
(the tool to detect ~~cybersecurity~~ S/W vulnerability use the same theory as compiler)
→ ASIC (Application Sepecific Integreted Circuit)
→ power reduction (make computers / phones consume less energy)

Parallism

→ Detect dependent and independent instructions

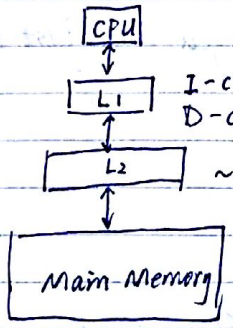
```
a = b + 1
...
c = a + 2
```
dependent instructions, cannot run in parallel (data dependency)

```
A[i] = ...
...
... = A[j]
```
may or may not be dependent, since i might equal to j

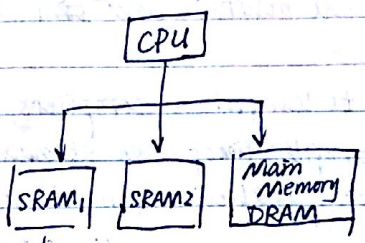
```
*p = ...
...
... = *q
```
if p and q point to the same thing, there might be dependence

```
for i = 1 to 100 {
    A[i] = A[i] + 3;        This is parallel
}
```

Affine theory
↳ linear functions.

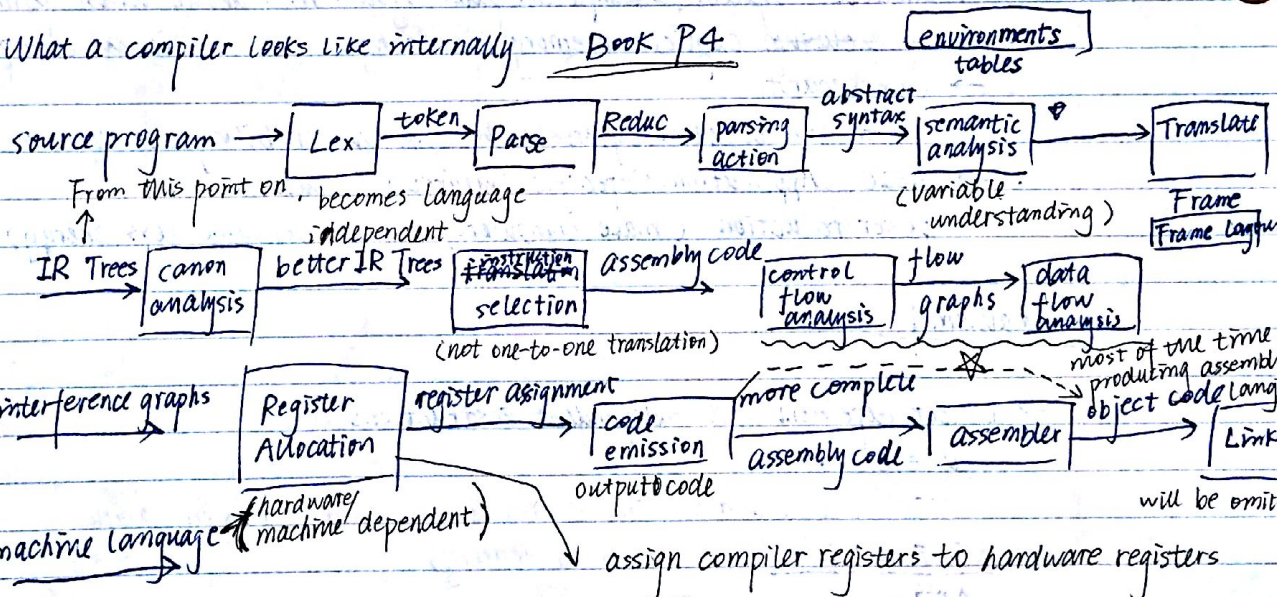


embedded system doesn't have cache



scratch pad memory
compiler will determine the addresses of variables
(put the most frequently used to SRAM 1)

2.3 What a compiler looks like internally — Book P4



variables { global variable
stack variable
heap variable

if you name a location directly, that's called register index accessing called memory

pointer can only point to memory location, but not register

ENEE 645

1.27 Prof. Rajeev B A.V.W. 1431

piazza.com (for disscussion)

4-5 homeworks 20%

1 midterm exam 30%

1 software project 15% (use LLVM) open-source compiler

1 final exam 35%

compiler is a piece of software that translate high-level language into machine code (HLL)

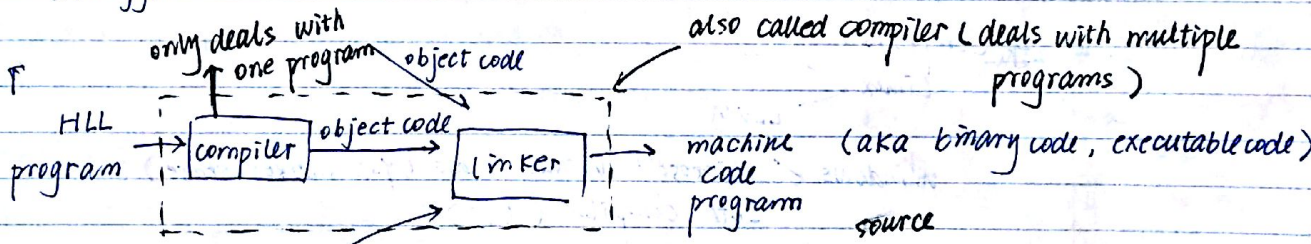
software development process:

Compiler (assembler may be part of compiler)

Linker

Loader

Debugger



(assume the program consists of 3 files)

(must include all program files in order to generate machine code)

Object code:

Like machine code, but

→ corresponds to only one source file

→ addresses of external symbols are missing

Linker: (Linker is running statically because it's ~~part of~~ run during compile time)

→ Joins multiple object files together

→ Resorb external addresses (Link different object files)

Loader:

→ Not a part of the compiler

→ Part of the OS.

cannot use standard libraries, as a result it
statically linked will cause redundancy in file system and main memory

→ Copies executable from file system to main memory & transfers control to it.
→ Links dynamically linked libraries (DLLs)
dynamically = run time statically = compile time

Library: (are not necessarily part of compilers)
→ a set of software routines that are often reuse in programs.
Reasons why written:
→ convenience & reuse
→ often call low-level resources
→ system calls (trap)
→ Assembly code
e.g. printf() is an open source code that makes system call →

Locations where linker filling addresses are specified in the relocation table.

1.29 Debugger
• part of SDK (Software Development kit)
SDK = Compiler + Linker + Debugger + GUI

Compilers:
Linux < gcc
LLVM
Windows < Microsoft Visual Studio (95% market share)
Intel compiler (ICC)

gcc
source to object file (xx.c → xx.o)
command: gcc -c file.c -o file.o
stop at object file
interpritive language: Java .net php has intellectual property issue

source to executable file
command: gcc file.c -o file.exe

if you have more than one file
gcc file1.c file2.c ... -o file.exe