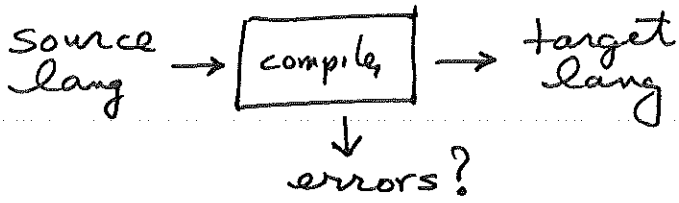
CMPS-104A

ch1: p 1

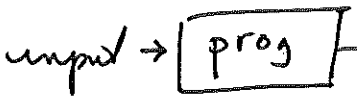
CH.1 Intro

Dragon 1: p 1

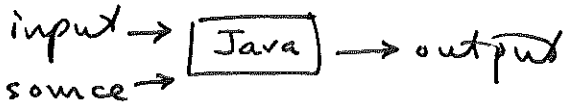
Language Processor



mach. lang



interpreter



Java

```
a=b*c+d;
```

Sparc RISC

```
ld [b], r1
ld [c], r2
mul r1, r2, r1
ld [d], r2
add r1, r2, r1
st r1, [a]
```

Intel CISC

```
mov b,eax
mul c,eax
add d,eax
mov eax, a
```

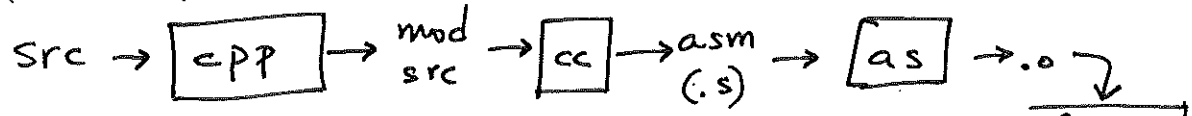
JVM

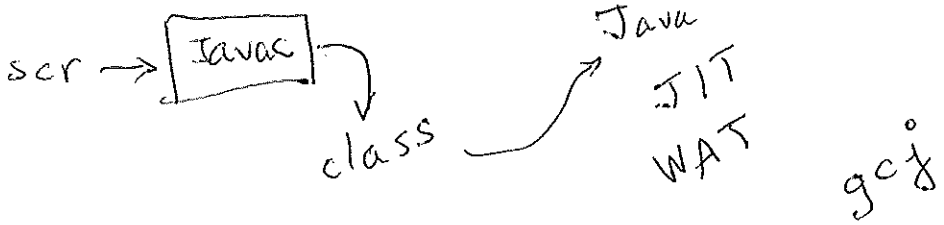
```
push b
push c
mul
push d
add
pop a.
```

Interpretation
shell:
Perl:
Java:

Compilation

~~cpp → cc → as → ld~~





CMPS -104A

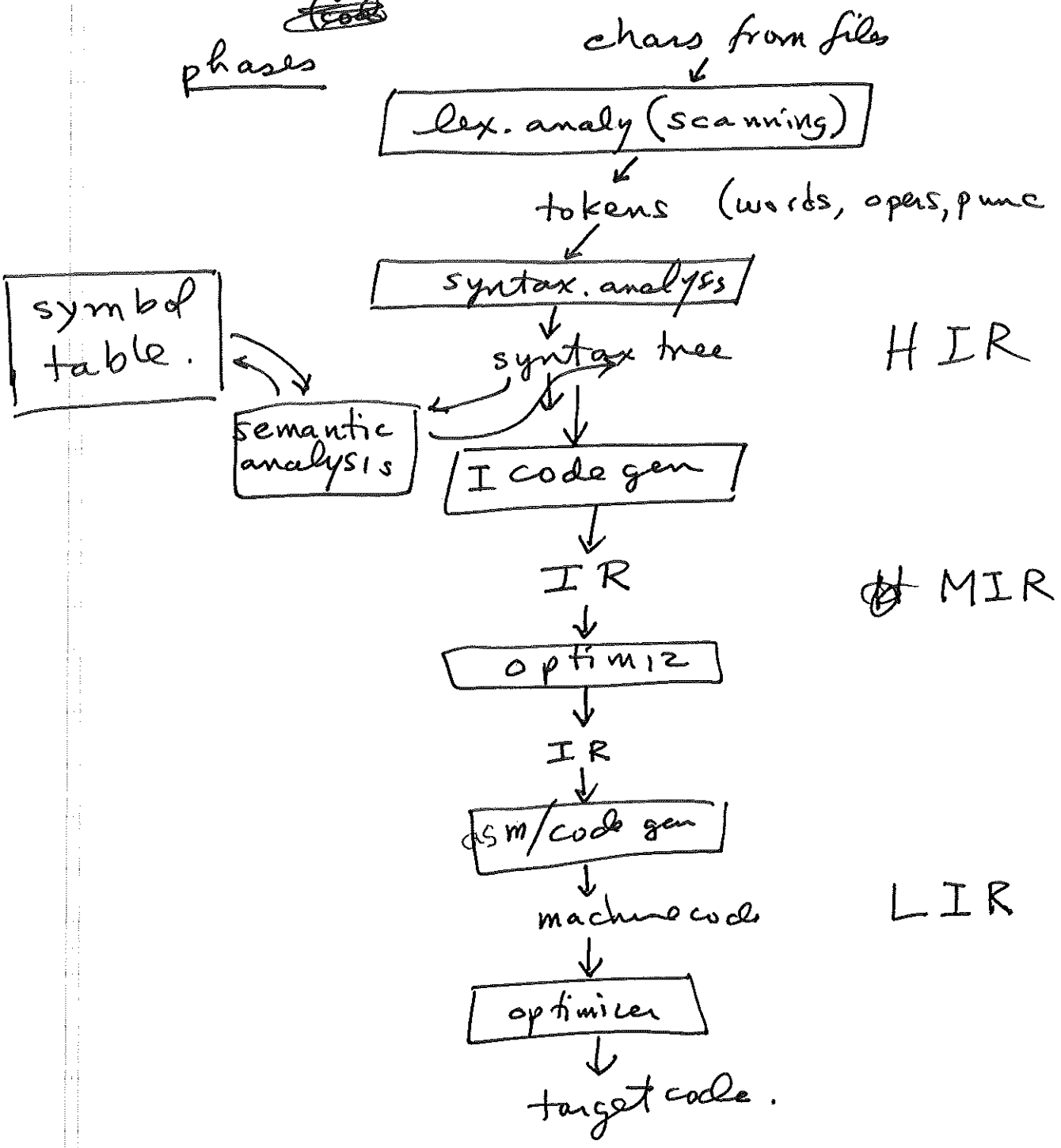
Compiler Structure

ch 1: p2

Dragon 1:5

1. front end = analysis (scan; parse)
2. "middle" end = optimization
3. back end = synthesis (code gen)

phases



Lexical Analysis

CMPS-104A

ch 1: p 3

Dragon 1:6

scanning

~~char~~ strings → tokens.

~~a~~ len = len + 1 ;
I I I I I I

constants

whitspc comments

find words, opers, punct, reserved words

elim white space, comments

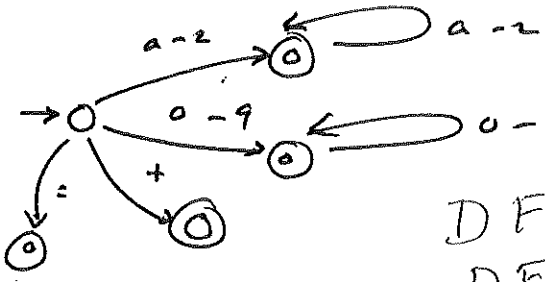
token = struct { lexinfo string ; token category }

make use of: regular expr to specify
: finite state m/c to operate (FA, DFA, NFA)

ex

```
[a-z]+
[0-9]+
"+"
"="
```

flex



DFSM
DFA

table driven
hand coded.

tool: flex (lex)

In English: I scream
ice cream.

↓

words
punctuation

CMPS -104A

ch 1: p 4

Syntax Analysis

parsing

token stream → abstract syntax tree (Dragon 1:8)

↳ parse tree

uses context free grammar.

usually LALR(1) - bottom up

tool: bison (yacc)

other tools: LL(1) - top down.

Earley is LR(k)

E T F

$E \to E+T$

$E \to T$

$T \to T*F$

$T \to F$

$F \to (E)$

$F \to i$

$G = \langle V_T, V_N, P, S \rangle$

$V_T = \{ + * ( ) i \}$

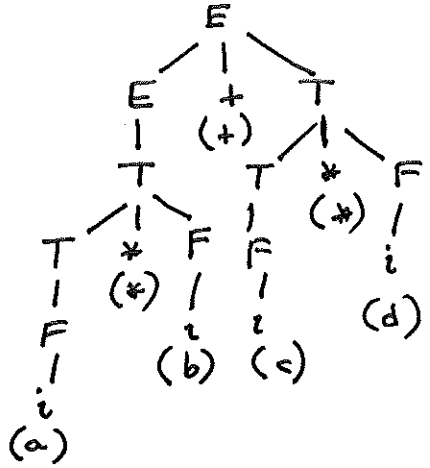
$V_N = \{ E\ T\ F \}$

P

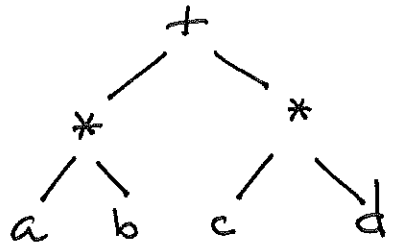
$S = E$

a*b+c*d

parse tree



AST



Silly English example

(1) Time flies like an arrow.

Fruit flies like a banana

(2) Colorless green ideas sleep furiously.

CMPS-104A
1-5
Dragon
1:8

Semantic Analysis

- type checking
  - each oper matching opnds
- implicit coercions
  - ex 3 + 4.0 → 3.0 + 4.0

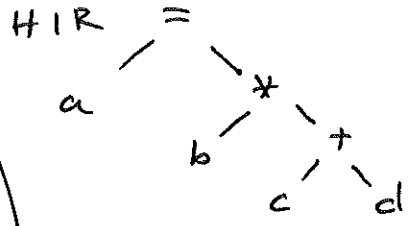
Intermed Code Gen

high	HIR	- AST
medium	MIR	- 3A C
low	LIR	- machine or asm.

source

a = b * (c+d);

HIR



MIR

t1 = c * d
t2 = b * t1
a = t2

Code Opt

- improve code
- can't change bubble → quick

Code Generation

- map IR → target lang
- alloc regs & mem loc

ex (above) t1 = t2 ⇒ r1
c = [fp - 4]
d = [fp - 8]

retargetable

CMPS-104A

ch 1.6

Dragon 1:11

Symbol Tables

map idents → attrs.

Tools

- parser generators: (bison)
- scanner generators (flex)
- syntax-directed transl.
- code gen generators.
- data flow analysis
- toolkits

Prog Languages ⟹ CMPS-112.

- asm lang
- procedural
- object oriented
- functional
- scripting

Building a Compiler

- math techniques
- finite automata & regexes.
  - lex units
- deterministic context free G.
- optimization - architecture
  - theory
  - but: engr: not math

CMPS-104A
ch 1.7
Dragon
1:17

Applications

- most people don't write compil.
- so why?
  - LITTL & languages.
  - understand how to use
- Impl of High Level PL
  - data abstraction
  - inheritance
- Optimization
  - parallelism ⇒ CMPS-111
    - IA64 compilation
    - out of order exec.
    - multiprocessors.
  - Mem hierarchies

	speed	cost	capy
regs	H	H	L
L1 cache	↓	↓	↓
L2 cache			
RAM			
disk cache			
disk swap	L	L	H

- Architectures
  - RISC
  - SIMD
  - VLIW

CMPS - 104A
1.8
Dragon 1 : 23

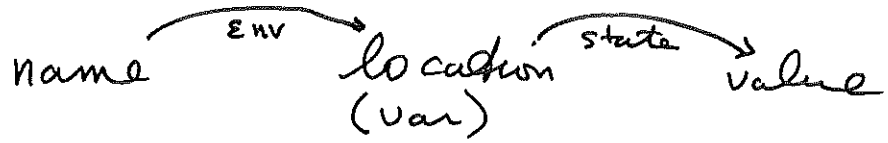
Productivity

type check – none
– static
– dynamic.
– data flow analysis
– bounds checking a[i]
– mem mgt : free vs gcol

PL Basics

static = compile time
dynamic = run time

- Scope : static vs dynamic
- Environment & state



env: map : name → locn
state : map : locn → value
name : lex chars
variable: locn in memory

- function = proc, method, subroutine
- Static Scope
  – block structure ~~ {....}
  – nesting
  – decl hiding (nest)

CMPS-104A
1.9
Dragon 1:31

Explicit Access Ctl

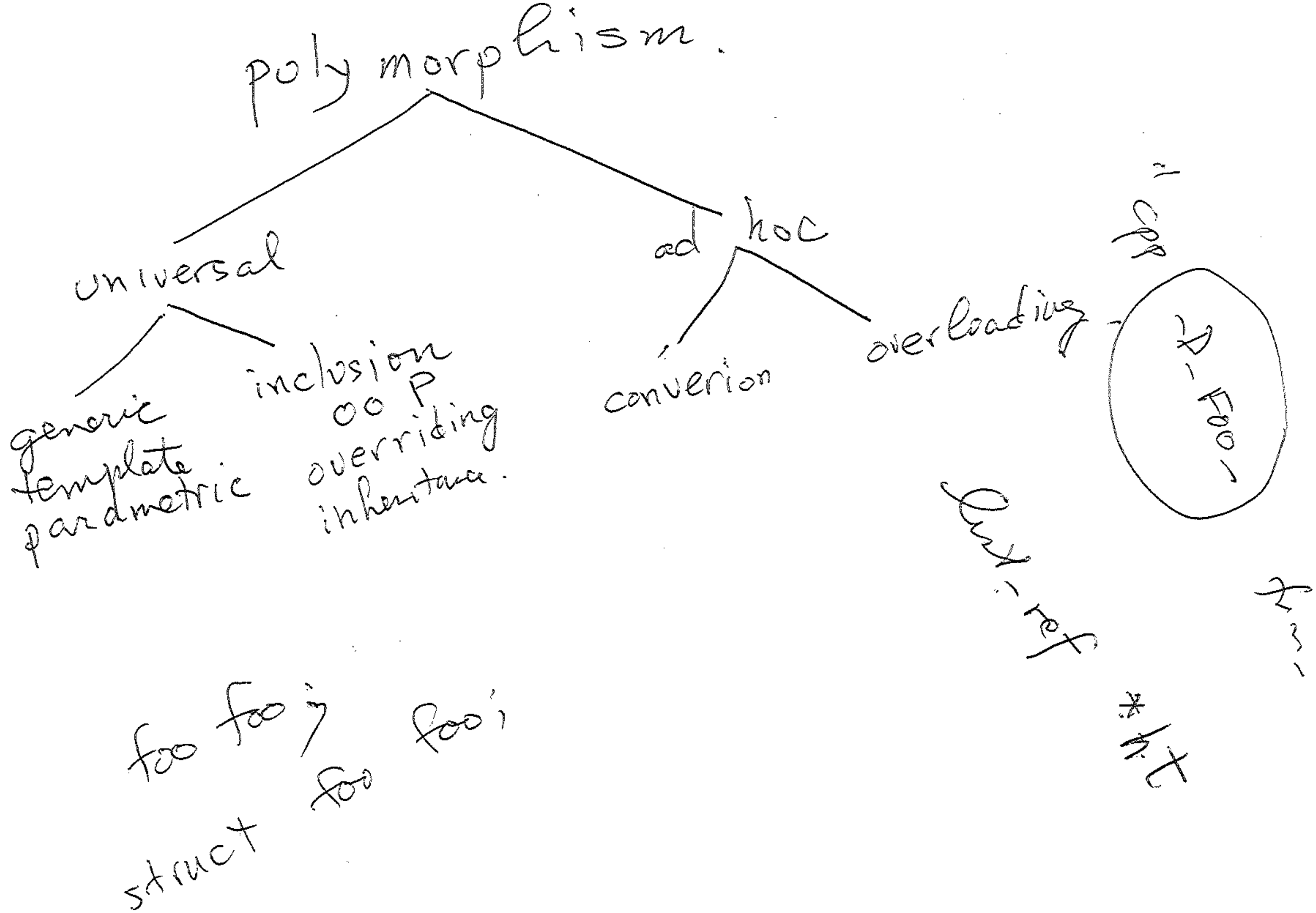
- Fields of structs
- x.f ~~~ x must be struct
  f must be field.

- public, protected, private, friend, package
  ~~~ → symtab problem

Parameter Passing

args — actual — exp used in a call
params — formal — decl of var in param list

- mechanism
  - call by value — copied
  - call by reference — pass addr.
  - call by name = thunk
- aliasing (call by ref)
  - pass — two params → same variable
  - global vs param aliasing
  ~~~


polymorphism.
universal
ad hoc
generic
template
parametric
inclusion
OOP
overriding
inheritance.
conversion
overloading
cpp
_D_Foo_
lust_ref
*ht
foo foo;
foo;
struct foo