2/5/18
CS4250
Page 2

- Lexical Analyzer: Follow mathematical models to recognize

3 categories

Automata

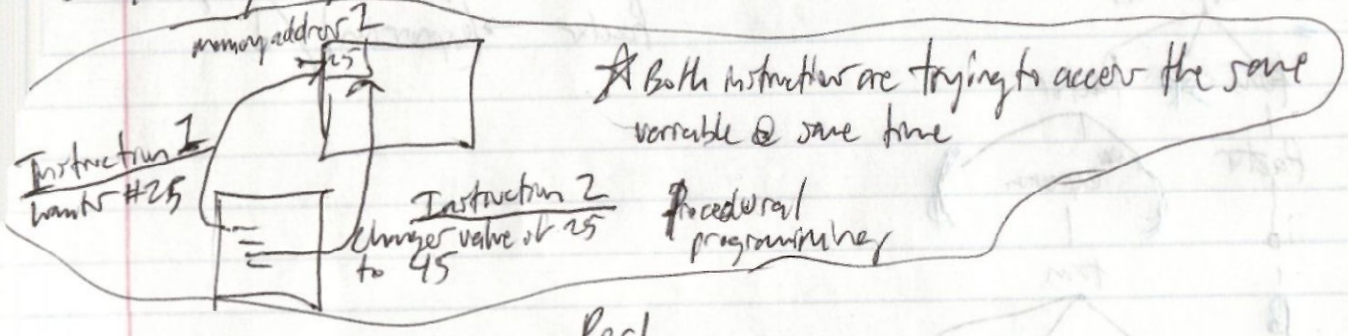
Push Down Automata
- some complex strings

Turing Machine
- Can process each & everything
→ Lamda Calculus, used for functional programs

- 3 paradigms for writing programs ① Procedural (OOP) ② Functional ③ Logic Based Program

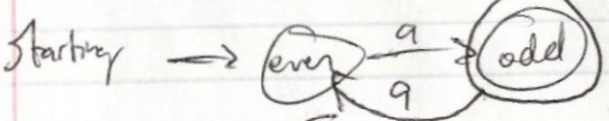
① Procedural: Follow steps. Have to check where variable is stored, how to get it / output (can produce incorrect result)

② Functional: Completely mathematical. Only need to provide input parameter / output (Lisp/Haskell/F#) Not concerned w/ where variables are stored.



③ Logic Based (Perl)

example: Language = [a, aaa, aaaaa...] = $[a^{2n+1} \mid n \geq 0]$



→ Using 2 state to recognize a whole language

Starting State [S, symbols, transitions, Final States]

2/7/18
CS 4250

- Attribute Grammar → Knuth
  - In some programming language, you have to follow constraints
    - program you have to write program name & end with program name.

STATIC SEMANTICS → constraints you have to follow to write program.

- wherever writing assignment operator:

<assignment> → <var> = <expr>
<expr> → ~~<var>~~ <var> + <var> / <var> - <var> / <var>
<var> → A / B / C

+C

Let us suppose var A → int
" " var B/C → float or real

So A = B + C is NOT allowed in this language

"Let us suppose" A = B + C and A = int, B = int; C = int
then A = B + C is VALID in this language

Grammar = { S, P, T, N, A } ← Attributes

- 2 Types of Attributes 1) Inherited
  2) Synthesized

- Static Semantics

- How do we define the grammer for "something"

Test Mon Feb 26

2/12/18
CS 4250

- Want to define the grammer of procedure. either function
  - Arithmetic statements
  - loops
  - control unit

→ (S, T, N, P)

- CFG can NOT describe all of a programming language

example procedure name
- procedure (x, y)

* putting f(x) or procedure name on top of Pda is semantics *

* To define functions / procedures use ATTRIBUTE GRAMMAR

- Attribute grammar : (S, T, N, P, A) → Attributes
- A(x) → X is non-terminal
- Synthesized Attribute or ~~intrinsic~~ Attribute

example - In particular programming language... we want to make an assignment statement
Valid → only assignment statements as type of variable on (L) side must match type of variable on (R) side.
Assignment → <var> = <expression>

type matching type

- Format of assignment statement

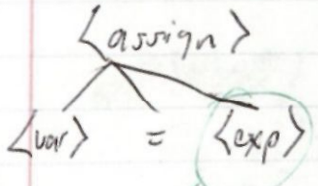
<assignment> → <var> = <exp>
<exp> → <exp> + <var> / <exp> * <var> // <var>
<var> → A / B / C / D

?, If A = B + C + D valid ?, ~~Yes~~

- int a → int is an attribute

A = B + C + D
int // = int if statement is valid

type[A] → int
type[B] → float
type[C] → int
type[D] → int



Attach an attribute called expected type. The sibling of the expression is determining the type. This is called an Inherited Attribute.

if the value of attribute comes from the two kids it is called Synthesized

2/12/18 Page 2
CS 4250

- Continuing example w/ grammar on previous page...

<exp> actual type <exp> ≠ <var>

A = B + C + D is NOT valid b/c A's type is int & B is float.

Defining Attribute grammar: Syntax Rule & Semantic Rule.

Expr <assign> → <var> = <exp>

<exp> → <var> + <var> / <var>

<var> → A / B / C

inherited

in Book

Rules for this grammar

1) <assign> → <var> = <exp> → Syntax Rule

Semantic Rule → <exp> expected type = <var> actual type

synthesized

2) <exp> → <var> + <var> → Syntax Rule

Semantic Rule → iff (<var> 1 actual type = int) and
iff (<var> 2 actual type = int) then
<expr> · actual type = int;
else
<expr> · actual type = float;

3) <exp> → <var> → Syntax Rule

Semantic Rule <exp> actual type = <var> actual type

4) <var> → A / B / C

Semantic Rule lookup string A;

~~[illegible]~~

▷ Add 1 more rule to 2nd syntax rule

<exp> · actual type = <exp> · expected type

p. 159 question 19

Test Mon 2/26/18

2/12/18 CS4250 Page 3

- Write Att. Grammar for same example as before but w/ newer rules

Rules: ① Data types can not be mixed in expression

② Data types cannot be mixed in exp but assignment statement need not be same

1) <assign> → <var> = <exp> DOES NOT need new rule

2) <exp> → <var> + <var>

semantic Rule: iff (<var>1.actualtype ≠ <var>2.actualtype)

then print ("Expression is NOT valid")

else

<exp>.actualtype = <var>.type

3) <exp> → <var> DOES NOT need new rule

4) All good w/ this one...?

New Example

<assign> → <id> = <exp>

<id> → A/B/C

<exp> → <id> + <exp> / <id> * <exp> / <exp> / <id>

Same Rules as are Above. Write grammar. w/ <id>type = <exp>type

1) <assign> → <id> = <exp>

<exp>.expectedtype = <id> actualtype

2 <id> → A/B/C

3) <exp> → <id> + <exp>

<exp>.actualtype

iff (<id> actualtype = <exp>1.actualtype)

then (<exp>.actualtype = <id>.actualtype)

else (<exp>.actualtype = float)

$a^n b^n c^n$ → Can you write CFG for this?

$a^n b^n$ → Can you write CFG for this?

↳ #A = #B so define recursively

S → a S b / ∈

↳ Sentences of this { ∈, ab, aabb, aaabbb ... }

S → abS / ∈

↳ Sentences of this { ∈, ab, abab, ababab ... }

Continued on Back

2/26/18
CS 4250

Attribute grammar → (CFG) → (T, N, S, P, A)

What are attributes? → attributes are related to Non-Terminals

Must write 2 rules → ① Syntax Rule & ② Semantics Rule

- Suppose we are writing program language where assignment will be valid only if the type of variable matches type of expression

CHAPTER 3

Assignment → var = exp

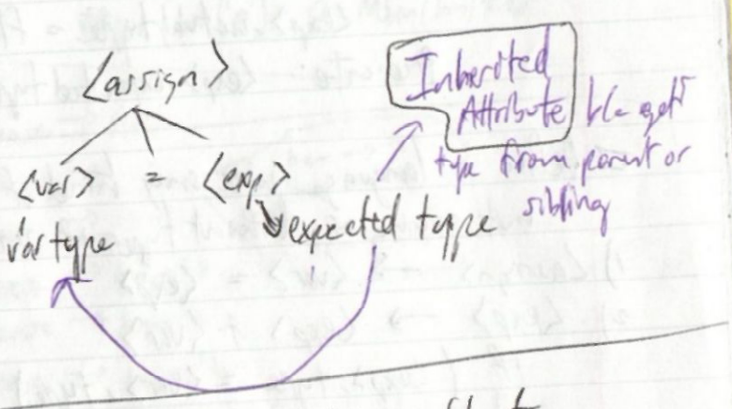
type is an attribute of <var>

<assign> → <var> = <exp>

<exp> → <var> + <var> / <var>

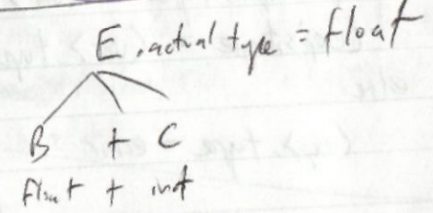
<var> → A / B / C

A	B	C
int	float	int



We want to match ~~exp.expectedtype~~ exp.expected type = exp.actual type

Synthesized Attributes are those attributes which get their attributes from their children or from the program



Intrinsic Attribute (a type of synthesized attribute)

1) <assign> → <var> = <exp> /// this is inherited attribute specifically Intrinsic

Semantic Rule: exp.expected type = var.actual type

2) <exp> → <var> + <var> /// var → int. or var → float

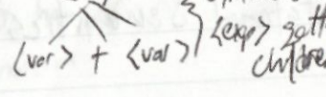
Semantic Rule: if (<var 1>.type == <var 2>.type)

<exp>.actual type = <var 1>.type

else

<exp>.actual (type) = float

Predicate exp.actual type = exp.expected type



3) <exp> → <var>

Semantic Rule <exp>.actual type = <var>.actual type

Predicate: exp.actual type = exp.expected type

2/26/18
CS4250

New Grammar: <assign> → <var> = <exp>
<exp> → <exp> + <var> / <var>
<var> → A/B/C

"Attribute grammar is best when semantic rules are minimized"

1) <assign> → <var> = <exp>
Semantic Rule: <exp>.expected type = <var>.actual type
2) <exp> → <exp1> + <var>
Semantic Rule: if (<exp>.actual type == <var>.actual type)
<exp>.actual type = <var>.actual type
else
<exp>.actual type = Float
Predicate: <exp>.expected type = <exp>.actual type ? ?

□ Define a language where any kind of assignment operator is allowed but mixing of different types of variables is NOT allowed.

1) <assign> → <var> = <exp>
2) <exp> → <exp> + <var>
if (<exp>.type = <var>.type)
<exp>.type = <var>.type
else
<exp>.type = error

- semantic information

Binary	Decimal
00	0
01	1
10	2
11	3

Mathematical = Recursive

bin → bin '1'
bin → '0'
bin → bin '1'
bin → bin '0'

- Denotational Semantics

bin → '1'
$M_b('1') = 1$
$M_b('0') = 0$
$M_b(bin\ '0') =$
$M_b(bin\ '1') = 2 * M_b(bin) + 1$

$111 = 2x(11)+1 = 2\cdot(2\cdot(11)+1)+1 = 2\cdot(2\cdot(2$

$11 = (11)1 = (2x3)+1$
$110 = (11)0 = 2x3+0$

2/28/18
CS4250

How to check the program? → Use Semantics

- Denotational Semantics powerpoint slide → Chapter 3
- Writing a program to convert binary # to decimal #... How do you know if it's correct?
  Apply Denotational Semantics to write a mathematical function that can convert any binary # to decimal #. Math F(x) → F(x) that has to be recursive. A language based on math F(x): Lambda Calculus.

bin → (0)
bin → (1)
bin → bin (1)
bin → bin (0)
} create any binary # w/ this

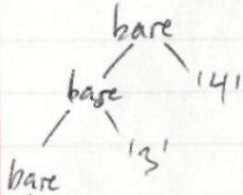
decimal#
123 → (12)x10 + 3

binary
$101_2 = (10_2) \times 2 + 1$

Mathematical Function → $M_{bin}('0') = 0$
$M_{bin}('1') = 1$
$M_{bin}(bin('1')) = 2 \cdot M_{bin}(bin) + 1$
$M_{bin}(bin('0')) = 2 \cdot M_{bin}(bin) + 0$

- Any # in base B will have B digits

{0; 1 ... 77 734



base → '7'

base → '0'
base → '1'
base → 'b-1'
base → base '0'
base → base '1'
base → base 'b-1'

base → '2'
base → '3'
base → 'b-1'

$M_b('0') = 0$
$M_b('1') = 1$
$M_b('2') = 2$
Skip some digits & $M_b('b-1') = b-1$

$M_b(base\ '0') = b \cdot M_b(base) + 0$
$M_b(base\ '1') = b \cdot M_b(base) + 1$
$M_b(base\ 'b-1') = b \cdot M_b(base) + (b-1)$

111 → '11' 1
b · p(11) + 1

LISP - Functional Programming → write everything in mathematical f(x)'s.

- Diff between f(x) & imperative programming: $x = y = (+) = x+y$ vs $1\ 2\ (+) = 3$

program → var → states so program changes variables states so we have to write mathematical f(x)'s that show states.

Chapter 3 example PAGE 164 example under 3.5.2.3 header

- Writing a denotational expression for a mathematical expression

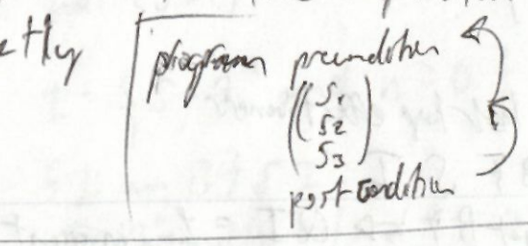
⋮

- Axiomatic Semantics → Reverse Engineering

3/5/18
CS 4250

Axiomatic Semantics; we use axiomatic verification to check if program ran correctly

Precondition
post = Postcondition



-1 ⇒ post statement is $a < 10$; $a < 10$ is output of $a = \frac{b}{2} - 1$

10 ⇒ $b < 22$

example: $x = 2y - 3$ $\{x > 25\}$ $2y - 3 > 25 \Rightarrow 2y > 28 \Rightarrow y > 14$ is the precondition

example: $y = 3x + 1$ (S1)
$x = y + 3$ (S2)
post statement: $x < 10$

$y + 3 < 10$
$y < 7$ → Precondition of S_2 which becomes the post condition of (S1)
$3x + 1 < 7$
$x < 2$ → Precondition of (S1)

(S1) $a = 2b + 1$
(S2) $b = a - 3$ $\{b < 0\}$

precondition: $a < 3$ that becomes post condition of (S1)
$2b + 1 < 3$
$2b < 2 \Rightarrow b < 1$

If $(X > 0)$ then
  $Y = Y - 1$ → post condition $Y > 0$ so $Y > 1$
else
  $Y = Y + 1$ → post condition $Y > 0$ so $Y > -1$
post condition = $Y > 0$

"for either statement the value of Y should be greater than 0"

$Y > 1$ is the answer ?? have to cover the weakest case

if $(X < Y)$
  $X = X + 1;$
else $X = 3 * X;$
$\{X < 0\}$

precondition
$X < 0$ → $X = 3X$, $X < 0$
$X < -1$ → $X = X + 1;$, $X < 0$

if $(a == b)$
  $b = 2 * a + 1;$
else $= 2 * a;$
$\{b > 1\}$

→ $a > \frac{1}{2}$ → $a > 0$

$a > \frac{1}{2}$ is correct.

Variables -- slides powerpoint
- How to write a program w/ complex # as input??

Mix of Ch.5 & Ch.6

3/14/18
CS 4250

Functional Programming $\lambda(X)\ X*X*X$

- Mathematical Function → MUST DEFINE EVERYTHING RECURSIVELY
- CAR or CDR functions
  - List (ABC)
    - CAR (LIST)
    - CDR (LIST) → (B,C)
  - CADR (ABC)
  - CAR (CDR (ABC)) = B
  - CDAR → CDR (CAR (ABC))
  - CADDR (ABCDE)
    - CAR (CDR (CDR (ABCDE))) = C
- Primitive Function Evaluation $\lambda X = X^2+3$

(language) LISP → atom: any value/fcn/operator
→ list: list of atoms
( )
→ (+ 2 3) = 5 (2+3)=5
1st atom

(* (+ 2 3) 7) = 5*7 = 35
5

data list: ① (A B C)

(λX X*X*X)

(lambda * (+ (* X X)))

3/19/18
CS 4250

Type Binding

static binding → global variables mostly

- local variable in any f(x)? → use stack
- pointer? → use heap

LET (e (+ (+ x y) 3) x+y+3)

Chapter 5

a+b/c-d → top/bottom

(LET ((top (+ a b))
(bottom (- c d)))
(/ top bottom))

top
a+b-3 / c+d-4
(name for each)
bottom

→ (Let ((top (- (+ a b) 3))
(bottom (- (+ c d) 4)))
(/ top bottom)
)

- ML → a functional programming language → No parentheses in ML

Let
val name1 = expression 1
: val namen = expression n
in expression
end;

Let
val top = a + b
val bottom = c - d
in top / bottom
end;

Chapter 6

Advantage of primitive type?? (in any language) Can define something your self. ALREADY DEFINED OPERATORS

- if you are worried your data is critical & you want that data private you must use Object Oriented Programming

3/19/18
4250 C
Page 2

int a[10]; → Static array declaration

know a[3] don't know a[0] want to find a[10]

upper lower

a[0] → a[3] → a[10]

int a[2][3] row column

a[0][0] "Row major representation"

What will be address of a[1][2] → rmr → cmr

int a[3][3]

base address a[0][0] = 2000H

$$\begin{bmatrix} 0,0 & 0,1 & 0,2 \\ 1,0 & 1,1 & 1,2 \\ 2,0 & 2,1 & 2,2 \end{bmatrix}$$

→ a[0][0] + 6

= 6000 + 6 * 2 = 2000C

cmr → 20004B

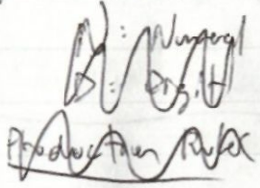
Calculate address of let a[4][3]

$$\begin{bmatrix} 0,0 & 0,1 & 0,2 \\ 1,0 & 1,1 & 1,2 \\ 2,0 & & 2,2 \\ 3,0 & & 3,2 \end{bmatrix}$$

3/21/18
CS4250

1) What is attribute grammar?
   - an extension of CFG where we add a symbol called Attribute.

2) Write denotational semantics for octal numbers.



Numeral ::= Digit | Numeral Digit

Binary numbers ⟶ (meaning) ⟶ Decimal value
Octal # ⟶ (meaning) ⟶ Decimal value

$B \rightarrow$ '0'	$M_B('0') = 0$
$B \rightarrow$ '1'	$M_B('1') = 1$
$B \rightarrow B$'0'	$M_B(B'0') = 2 \times M_B(B) + 0$
$B \rightarrow B$'1'	$M_B(B'1') = 2 \times M_B(B) + 1$

For octal:

$O \rightarrow$ '0' | '1' | '2' | '3' | '4' | '5' | '6' | '7'

$O \rightarrow O$ '0' | '1' | '2' | '3' | '4' | '5' | '6' | '7'

$0 \le d \le 7$

$M_O('d') \rightarrow d$

$M_O('0') \rightarrow 0 \quad M_O('1') \rightarrow 1 \ldots M_O('7') \rightarrow 7$

$M_O(O\,'d') \rightarrow 8 \times M_O(O) + d$

$M_B('0') = 0$

$M_B('1') = 1$

$M_B(B'0') = 2 \cdot M_B(B) + 0$

$M_B(B'1') = 2 \cdot M_B(B) + 1$

"Denotational Semantics"

3/21/18
CS 4250

$M_B(1101) \rightarrow M_B(110\,'1')$

$\rightarrow 2 \cdot M_B(110) \rightarrow 2 \cdot (2 \cdot M_B(11) + 0) + 1$ ★

★ $= 2^2 M_0(11) + 1 = 2^2 (2 \cdot \underbrace{M_B(1)}_{=1} + 1) + 1 =$ ☺

☺ $= 2^3 \cdot 1 + 2^2 \cdot 1$

Calculate decimal value of $(421)_8$:

START: $M_0(42\,'1') \rightarrow 8 \cdot M_0(4\,'2') + 1$ ★

★ $= 8 \cdot (8 \cdot M_0(4) + 2) + 1$ ☺

☺ ~~$= 8^2 \cdot 4 + 2 \cdot$ = 2005~~

↳ $\triangle = 8^2 \cdot M_0(4) + 8 \cdot 2 + 1$ ✡

✡ ~~[illegible]~~ $= 8^2 \cdot 4 + 8 \cdot 2 + 1$

Octal Rules

$M_0('0') = 0$

$M_0('1') = 1 \ldots M_0('7') = 7$

$M_0(0\,'0') = 8 \cdot M_0('0') + 0$

$M_0(0\,'7') = 8 \cdot M_0('0') + 7$

$M_B(1011)_8$?, ?

$M_0(1011) \rightarrow M_B(101\,'1') \rightarrow 8 \cdot M_0(\underbrace{1\,'0\,'1}_{?}) + 1 \rightarrow$

$8 \cdot (8 \cdot M_B(10) + 1) + 1 \rightarrow 8^2 M_B(10) + 2 \rightarrow 8^2 (8 \cdot \underbrace{M_B(1)}_{=1} + 0) + 1$

64.9

Write grammar

- Use denotational method to find meaning of # in VSOP denotational semantics

Attribute grammar

<assign> → <var> = <exp>

<exp> → <var> + <var> / <var>

<var> → A / B / C

i) exptype → vartype

- What will be intrinsic, inherited, & synthesized attributes in this grammar
- Specify attribute & show how to specify these conditions in grammar?

A → B + C

A → Real / float
B → Real / Float
C → int

{ expected type → 'exp'
  actual type → 'exp' }

<assign> → <var> = <exp>
exp.expectedtype = var.actualtype
<exp> → <var> + <var>
if (var1.actualtype = var2.actualtype)
exp.actualtype = var.actualtype
else
exp.actualtype = Real / float

A = C + D
PREDICATE → Check in textbook!!!!!

@ least 1 textbook question on Exam 2

"This is grammar, identify which will be synthesized attribute?"

"This is grammar... identify which will be inherited attribute?"

(5)

S → E B E
E → V o V / V
V → b / var
o → + / *
B → == / != / < / > / >= / <=

(6)

CAR grabs first element from the list
CAR (a, b, c) ⇒ a
CDR grabs everything but first element from a list
CDR (a, b, c) ⇒ b, c

Define (member atm a_list)
(name of function)
(2 inputs)

((cond
((null? a_list) #F) ← asking if a_list is null
((EQ atm (CAR a_list)) #T) ← asking if atm = list element of a_list
((Else (member atm (CDR a_list)))
))
(deliver atm as the remaining elements of a_list that is not the first 2)

f(x) definition not bound by an identifier

(8)
```
(lambda (a b c)
        (+ (+ (* a (* x x)) (* b x)) c))
```

- A lambda expression

(9)
explicit int a
implicit var type binding is done by a compiler.

Chapter 7

① variable
② data structure
③ Expression
} 3 parts of a program

4/2/18
CS4250

What is evaluation for operator for a+b*c+d?

→ a+(b*c)$_1$+d ① this signifies order of operation

→ (a+(b*c)$_1$)$_2$+d

→ ((a+(b*c)$_1$)$_2$+d)$_3$

- Highest level → *, /, NOT

Associativity: Left to Right

+, -, ~~/~~, mod

- (unary)

=, /=, <, <=, >=, >, and

Lowest level → OR, NOR

---

(1) a*b-1+c

→ (a*b)$_1$ -1 +c

→ ((a*b)$_1$ -1)$_2$ +c

→ (((a*b)$_1$ -1)$_2$ +c)$_3$

(2) a*(b-1)/c mod d

→ ~~a*(b-1)/c mod d~~

→ a*(b-1)$_1$ /c mod d

↔ (a*(b-1)$_1$)$_2$

→ ((a*(b-1)$_1$)$_2$ / c)$_3$ mod d

↔ (((a*(b-1)$_1$)$_2$ /c)$_3$ mod d)$_4$

4/2/18
CS4250 paper

TEST 2 Question like this. No messy operator stuff.

example $a > b$ XOR cor $d <= 17$

$\rightarrow (a > b)_1$ XOR cor $d <= 17$

$\rightarrow (a > b)_1$ XOR cor $(d <= 17)_2$

$\rightarrow ((a > b)_1$ XOR $C)_3$ OR $(d <= 17)_2)_4$

---

① $(a-b)/c$ ∄ $(d*e/a-3)$

$\rightarrow (a-b)_1/c$ ∄ $(d*e/a-3)$

$\rightarrow (a-b)_1/c$ ∄ $(((d*e)_2/a)_3 - 3)_4$

$((a-b)_1/c)_5$ ∄ $(((d*e)_2/a)_3 - 3)_4$

CAR ∄ CDR definitely ON exam.

- What complex types can be defined in what languages??

- No Chapter 4
What kind of complex data types can be be defined in what languages??
Implicit / explicit Show examples