Broghici Andreea-Maria | CR451A

C-minus-minus Project

1. C-- grammar

1.1 Rules

program: program varDecl | program fnDecl | /* empty */ ;

varDecl: type id ';' | type id '[' INTLITERAL ']' ';' ;

type: INT | BOOL | VOID ;

fnDecl: type id parameters block ;

parameters: '(' ')' | '(' formalsList ')' ;

formalsList: formalDecl | formalsList ',' formalDecl ;

formalDecl: type id ;

block: '{' declList stmtList '}'

declList: declList varDecl | /* empty */ ;

stmtList: stmtList stmt | /* empty */ ;

stmt: CIN READ id ';'
| CIN READ id '[' exp ']' ';'
| COUT WRITE exp ';'
| subscriptExpr '=' exp ';'
| id '=' exp ';'
| IF '(' exp ')' block
| IF '(' exp ')' block ELSE block
| WHILE '(' exp ')' block
| RETURN exp ';' | RETURN ';' | fnCallStmt ';'

Drăghici Andreea-Maria	CR 451A

```
exp : exp '+' exp
    | exp '-' exp
    | exp '*' exp
    | exp '/' exp
    | '!' exp
    | exp ANDAND exp | exp OROR exp | exp EQEQ exp
    | exp NOTEQ exp | exp '<' exp | exp '>' exp
    | exp LESSEQ exp | exp GREATEREQ exp
    | '-' otom | otom ;

otom : INTLITERAL | STRINGLITERAL | TRUE | FALSE
     | '(' exp ')' | fnCallExpr | subscriptExpr | id ;

fnCallExpr : id '(' ')' | id '(' actualList ')' ;

fnCallStmt : id '(' ')' | id '(' actualList ')' ;

actualList : exp | actualList ',' exp ;
subscriptExpr : id '[' exp ']' ;
id : ID ;
```

Drăghici Andreea-Maria	CR4S1A

1.2 Terminal symbols:

INT

BOOL

VOID

EQ => "="

EQEQ => "=="

OROR => "||"

ANDAND => "&&"

NOTEQ => "!="

GREATEREQ => ">="

LESSEQ => "<="

"+"

"-"

"*"

"/"

"!"

"<"

">"

TRUE

FALSE

WHILE

IF

ELSE

RETURN

TYPE

Drăghici Andreea-Maria. | CR4S1A.

";"
"("
")"
"["
"]"
","
"|"

ID

INT_LITERAL

STRING_LITERAL

"{"
"}"
"."
"<<"
">>"

CIN

COUT

Drăghici Andreea - Maria	CR4 51 A

1.3. Non-terminal symbols

< program >
< varDecl >
< fnDecl >
< parameters >
< formalsList >
< formalDecl >
< block >
< declList >
< stmtList >
< stmt >
< exp >
< term >
< fnCallExpr >
< fnCallStmt >
< actualList >
< subscriptExpr >

Dăghici Andreea - Maria	CR 4 S1A

1. Lexical analysis

1.1 Regular definitions

Digit -> [0-9]

Number -> [1-9][0-9]*

character -> {Digit} | [A-Za-z]

String -> {character}+

Boolean = true | false

Integer = {Number}

ident -> [A-Za-z][A-Za-z0-9]*

int-const -> 0

int-const -> [1-9][0-9]*

1.2 Finite automata used for lexical analysis

fig. 1: finite automaton for the identifier lexical category

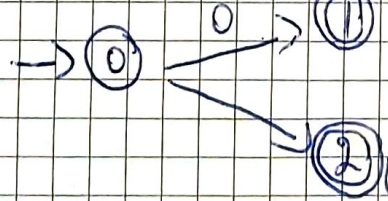

fig. 2: finite automaton for the integer-constant category

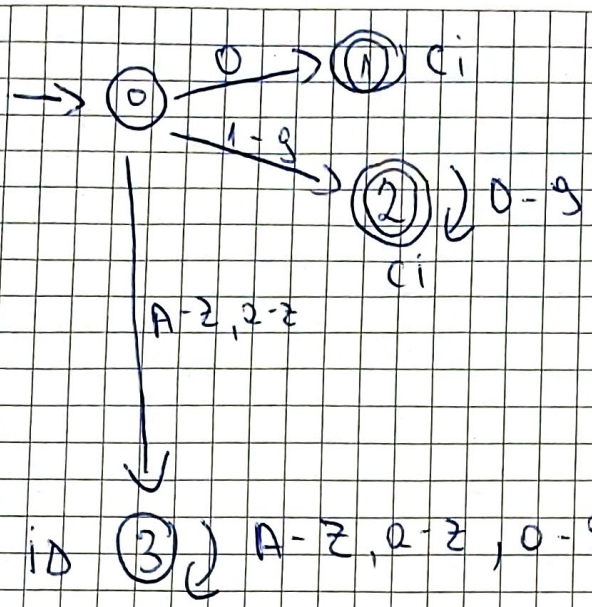

Fig. 3. deterministic finite automaton for ident and integer - constant lexical categories.

Drăghici Andreea-Maria CR4 S1A

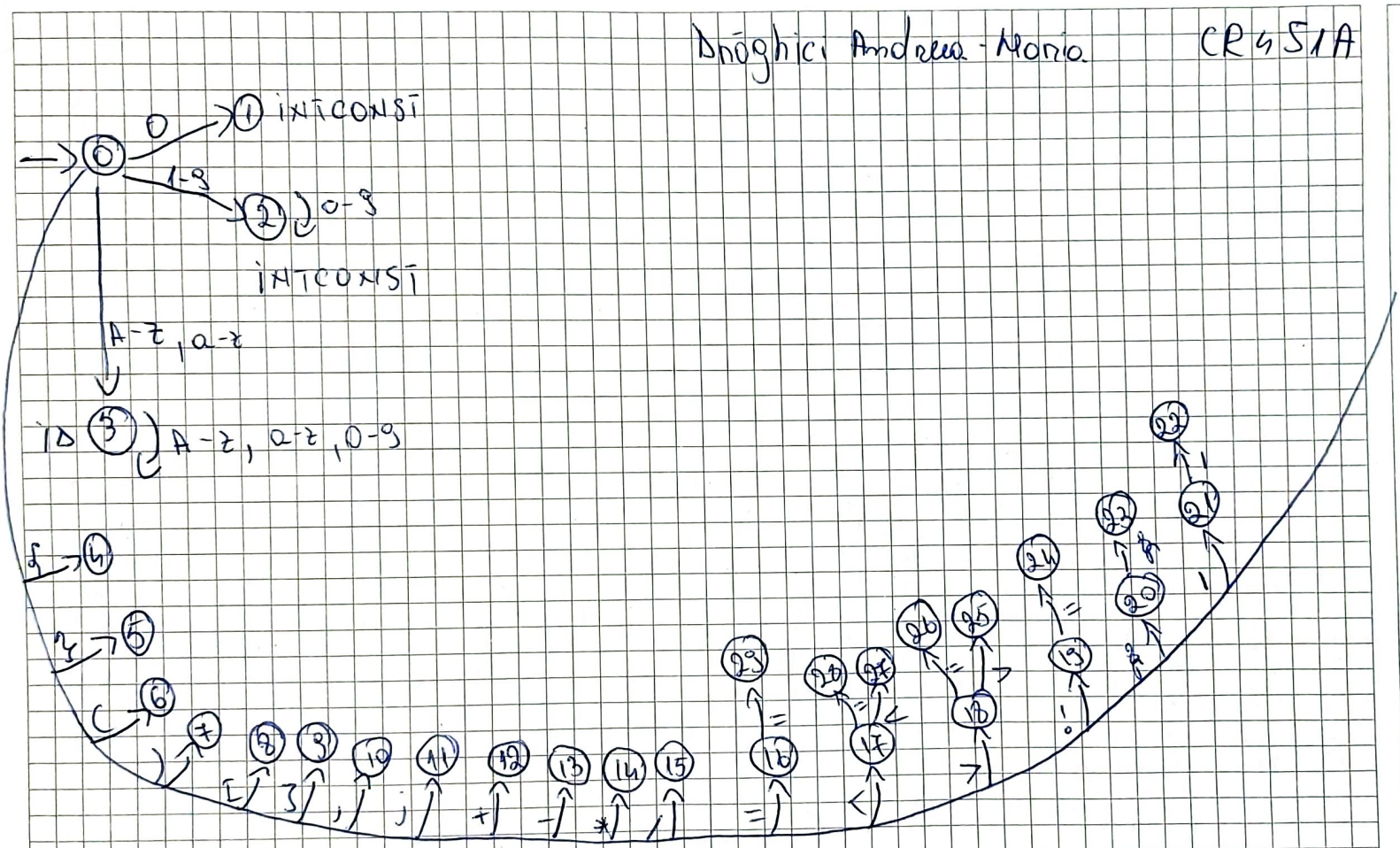

fig 4. Deterministic finite automaton for all lexical categories.

Dnoghici Andreea-Maria	CR4S1A.

The names of some final states in this DFA are:

STATE 4: LBRACE,

STATE 5: RBRACE,
STATE 6: LPAREN,
STATE 7: RPAREN,
STATE 8: LSQBKT,
STATE 9: RSQBKT,
STATE 10: COMMA,
STATE 11: SEMICOLON,
STATE 12: PLUS,
STATE 13: MINUS,
STATE 14: MULTIPLICATION,
STATE 15: DIVISION,
STATE 16: EQ,
STATE 17: LT,
STATE 18: GT,
STATE 19: NOT,
STATE 20: AND,
STATE 21: OR,
STATE 22: OROR,
STATE 23: ANDAND,
STATE 24: NEQ,
STATE 25: RSHIFT,

STATE 26: GREATEREQ,
STATE 27: LSHIFT,
STATE 28: LESSEQ,
STATE 29: EQEQ.