Type	Examples
ID	`foo  n14  last`
NUM	`73  0  00  515  082`
REAL	`66.1  .5  10.  1e67  5.5e-10`
IF	`if`
COMMA	`,`
NOTEQ	`!=`
LPAREN	`(`
RPAREN	`)`

comment	`/* try again */`
preprocessor directive	`#include<stdio.h>`
preprocessor directive	`#define NUMS 5 , 6`
macro	`NUMS`
blanks, tabs, and newlines	

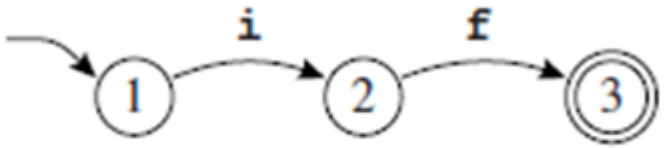
```
float match0(char *s) /* find a zero */
{if (!strncmp(s, "0.0", 3))
  return 0.;
}
```

```
FLOAT ID(match0) LPAREN CHAR STAR ID(s) RPAREN
LBRACE IF LPAREN BANG ID(strncmp) LPAREN ID(s)
COMMA STRING(0.0) COMMA NUM(3) RPAREN RPAREN
RETURN REAL(0.0) SEMI RBRACE EOF
```

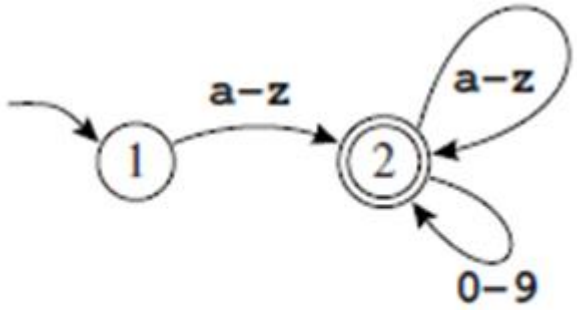
Regular expression notation.

a	An ordinary character stands for itself.
ϵ	The empty string.
	Another way to write the empty string.
$M \mid N$	Alternation, choosing from M or N.
$M \cdot N$	Concatenation, an M followed by an N.
MN	Another way to write concatenation.
M^*	Repetition (zero or more times).
M^+	Repetition, one or more times.
$M?$	Optional, zero or one occurrence of M.
[a – zA – Z]	Character set alternation.
.	A period stands for any single character except newline.
`"a.+*"`	Quotation, a string in quotes stands for itself literally.

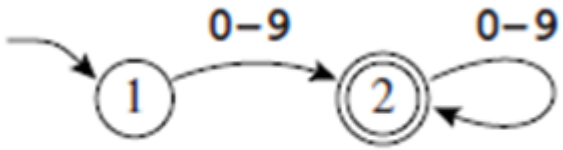
```
if                                          {return IF;}
[a-z][a-z0-9]*                              {return ID;}
[0-9]+                                      {return NUM;}
([0-9]+"."[0-9]*)|([0-9]*"."[0-9]+)         {return REAL;}
("--"[a-z]*"\n")|(" "|"\n"|"\t")+           { /* do nothing */ }
.                                           {error();}
```



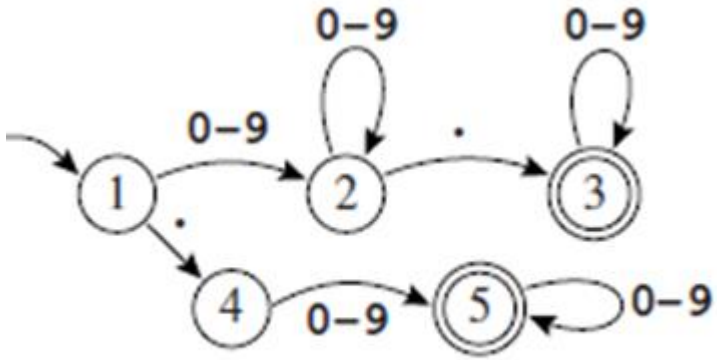
IF



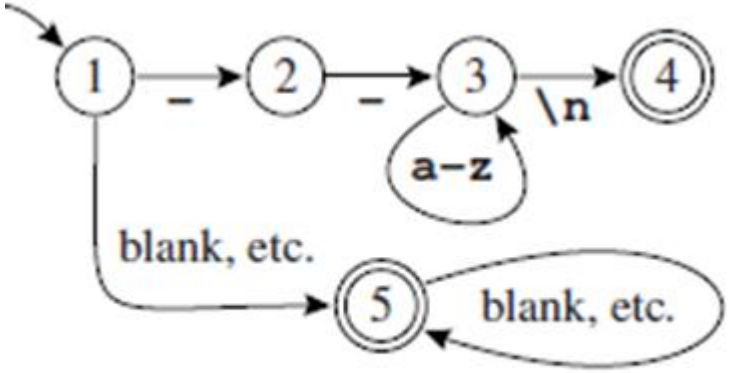
ID



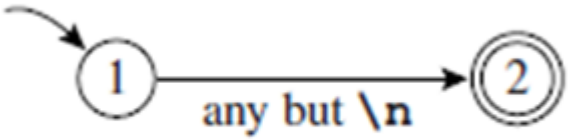
NUM



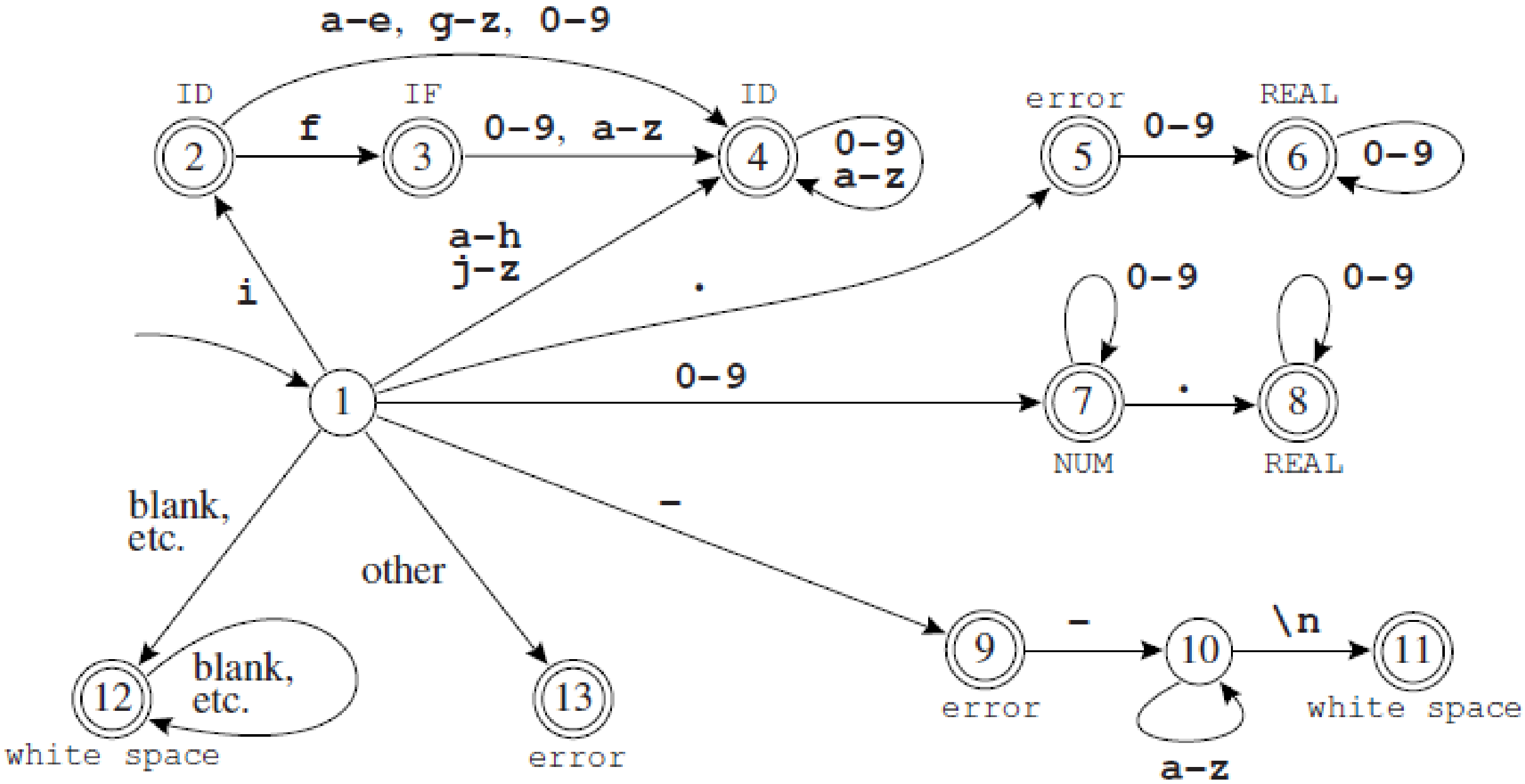
REAL



white space

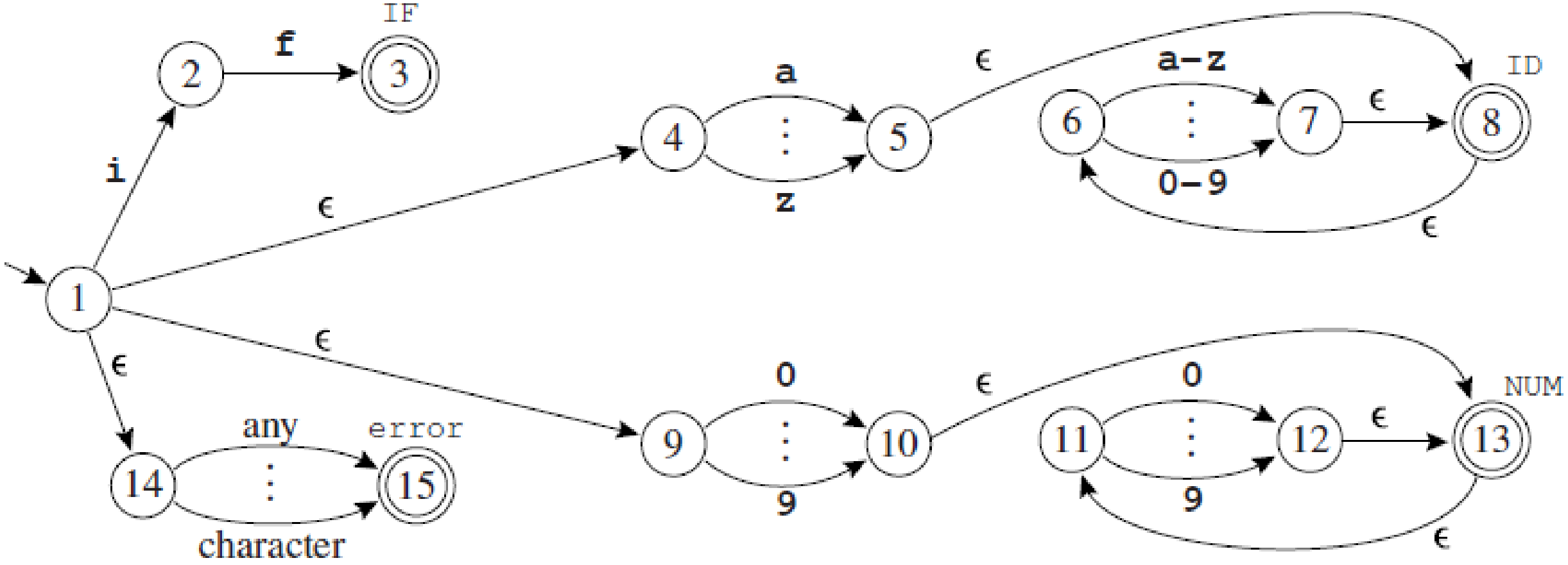


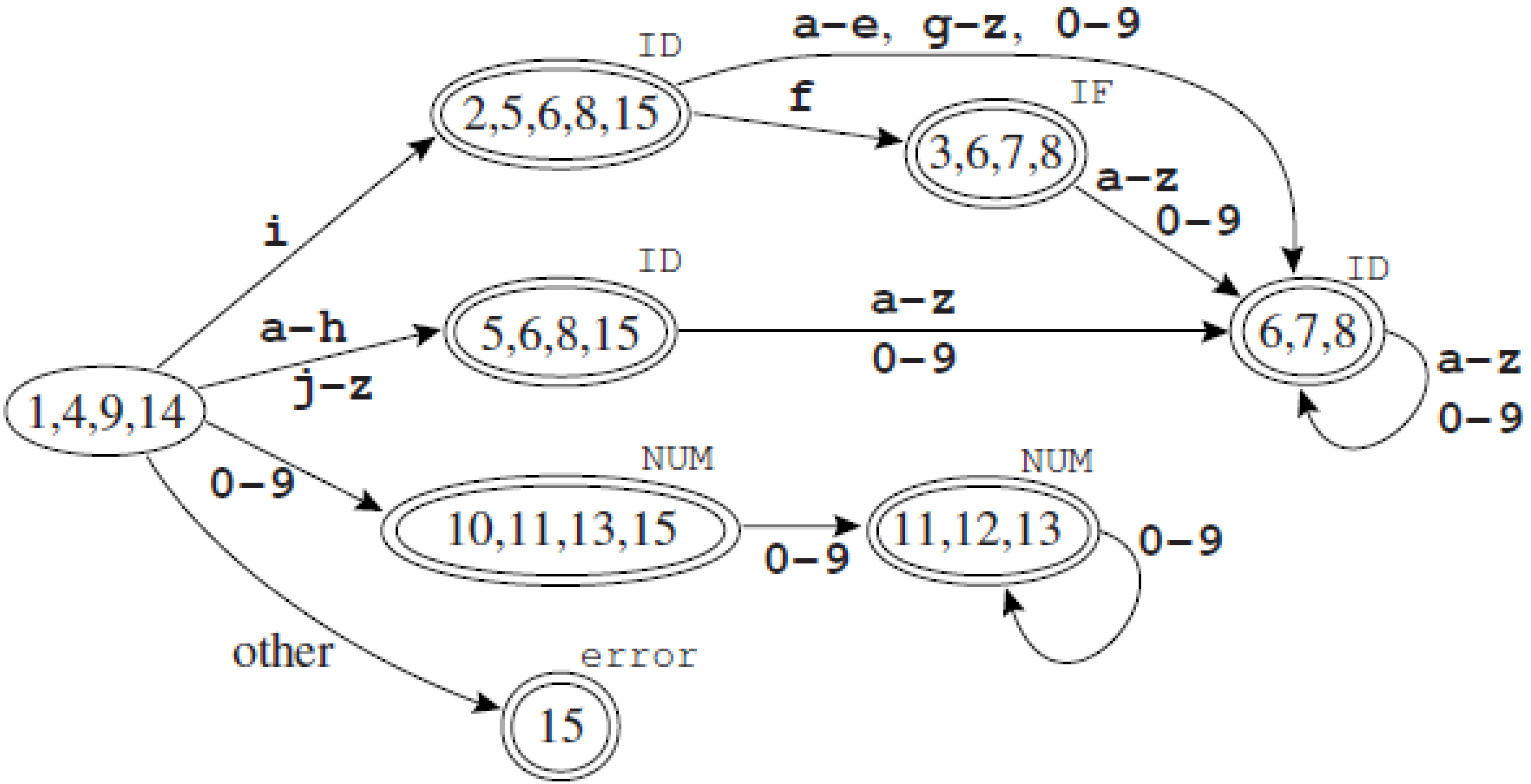
error


a-e, g-z, 0-9
ID
IF
ID
error
REAL
2
f
3
0-9, a-z
4
0-9
a-z
5
0-9
6
0-9
a-h
j-z
i
.
0-9
0-9
1
0-9
7
.
8
NUM
REAL
blank,
etc.
-
other
9
-
10
\n
11
12
blank,
etc.
13
error
white space
error
a-z
white space

Last Final	Current State	Current Input	Accept Action
0	1	`|if --not-a-com`	
2	2	`|i|f --not-a-com`	
3	3	`|if| --not-a-com`	
3	0	`|if| --not-a-com`	*return* IF
0	1	`if| --not-a-com`	
12	12	`if| --not-a-com`	
12	0	`if| -|-not-a-com`	*found white space; resume*
0	1	`if |--not-a-com`	
9	9	`if |-|-not-a-com`	
9	10	`if |-|n|ot-a-com`	
9	10	`if |-|n|ot-a-com`	
9	10	`if |-|no|t-a-com`	
9	10	`if |-|not|-a-com`	
9	0	`if |-|not-|a-com`	*error, illegal token '-'; resume*
0	1	`if -|-not-a-com`	
9	9	`if -|-|n|ot-a-com`	
9	0	`if -|-|n|ot-a-com`	*error, illegal token '-'; resume*

NONDETERMINISTIC FINITE AUTOMATA




1,4,9,14
i
a-h
j-z
0-9
other
2,5,6,8,15
ID
5,6,8,15
ID
10,11,13,15
NUM
15
error
a-e, g-z, 0-9
f
3,6,7,8
IF
a-z
0-9
a-z
0-9
6,7,8
ID
a-z
0-9
11,12,13
NUM
0-9

```
%{
/* C Declarations: */
#include "tokens.h"    /* definitions of IF, ID, NUM, ... */
#include "errormsg.h"
union {int ival; string sval; double fval;} yylval;
int charPos=1;
#define ADJ  (EM_tokPos=charPos, charPos+=yyleng)
%}
/* Lex Definitions: */
digits  [0-9]+
%%
/* Regular Expressions and Actions: */
if                        {ADJ; return IF;}
[a-z][a-z0-9]*            {ADJ; yylval.sval=String(yytext);
                            return ID;}
{digits}                 {ADJ; yylval.ival=atoi(yytext);
                            return NUM;}
({digits}"."[0-9]*)|([0-9]*"."{digits})     {ADJ;
                            yylval.fval=atof(yytext);
                            return REAL;}
("--"[a-z]*"\n")|(" "|"\n"|"\t")+    {ADJ;}
.                         {ADJ; EM_error("illegal character");}
```

```
    :
    :  the usual preamble ...
%Start INITIAL COMMENT
%%
<INITIAL>if       {ADJ; return IF;}
<INITIAL>[a-z]+ {ADJ; yylval.sval=String(yytext); return ID;}
<INITIAL>"(*"     {ADJ; BEGIN COMMENT;}
<INITIAL>.        {ADJ; EM_error("illegal character");}
<COMMENT>"*)"     {ADJ; BEGIN INITIAL;}
<COMMENT>.        {ADJ;}
.                 {BEGIN INITIAL; yyless(1);}
```