Assignment 1 (Lexical Analyzer)

Compiler Construction CS4435 (Spring 2015)

University of Lahore

Maryam Bashir

Assigned: Saturday, March 14, 2015. Due: Monday 23rd March 2015 11:59 PM

Lexical analysis

Lexical analysis is the process of reading in the stream of characters making up the source code of a program and dividing the input into tokens. In this assignment, you will use regular expressions and DFAs to implement a lexical analyzer for a subset of C programming language.

Your Task

Your task is to write a program that reads an input text file, and constructs a list of tokens in that file. Your program may be written in C, C++, Java or any other programming language. Assuming that the input file contains the following code string:

2.1 Sample Input (C++ code)

```
void main ()
    {
        int sum = 0;
        for(int j=0; j < 10; j=j+1)
        {
            sum = sum + j + 10.43 + 34E4 + 45.34E-4 + E43 + .34;
        }
    }
```

2.2 Sample Output

The output of the program should be similar to the following:

Class : Lexeme
keyword : void
identifier : main

(: (
) :)
{ : {
keyword : int
identifier : sum
= : =
num : 0
; : ;
keyword : for
(: (
keyword : int
identifier : j
= : =
num : 0
; : ;
identifier : j
< : <
num : 10
; : ;
identifier : j
= : =
identifier : j
+ : +
num : 1
) :)
{ : {
identifier : sum
= : =
identifier : sum
+ : +
identifier : j
+ : +
num : 10.43
+ : +
num : 34.E4
+ : +
num : 45.34E-4
+ : +
identifier : E43
+ : +
Error : .
num : 34
; : ;
} : }
} : }

Token Type	**Lexical Specification**
keyword	One of the strings **while**, **if**, **else**, **return**, **break**, **continue**, **int** , **float**, **void**
identifier	Token Id for identifiers matches a letter followed by letters or digits or underscore: **letter** $\rightarrow$ **[A-Z** $\mid$ **a-z]** **digit** $\rightarrow$ **[0-9]** **id** $\rightarrow$ **letter (letter** $\mid$ **digit** $\mid$ _)*
num	Token num matches unsigned numbers: **digits**$\rightarrow$ **digit digit*** **optional-fraction** $\rightarrow$ (**.** **digits**) $\mid \epsilon$ **optional-exponent** $\rightarrow$ ($E(+ \mid - \mid \epsilon)$ **digits**) $\mid \epsilon$ **num**$\rightarrow$ **digits optional-fraction optional-exponent**
addop	+ , -
mulop	$*, /$
relop	$<, >, <=, >=, ==, != $
and	&&
or	||
not	!
)	)
(	(
{	{
}	}
[	[
]	]

Table 1: Table of lexical specifications for tokens

Valid Tokens

Programs in this language are composed of tokens displayed in table 1 :

Construction of Deterministic Finite Automata (DFA) or Transition Diagrams

You can construct single DFA (also called transition diagram) for recognizing all tokens in this language by combining individual DFAs for each type of token. For example you can construct transition diagram for identifiers and numbers and them merge the start states of two diagrams to create a single transition diagram that recognizes both numbers and identifiers.

For identifying keywords, you can store keywords in some data structure (hashmap or string array). Whenever your program recognizes a token of identifier, you can check your map or string

array if this identifier matches any keyword. If it matches any keyword then consider it keyword otherwise consider it identifier.

Implementation of Deterministic Finite Automata (DFA)

I have written a small lexical analyser for a simple DFA shown in figure 1. You can find its source code ("Lexer.java") on Piazza site. You can either make changes to this code or write your own lexical analyser from scratch. You can also write C++ version of this Java code if you want to program in C++.

Please note that this DFA is not correct DFA for this assignment. It is just a sample DFA to give you an idea of how to convert your DFA to code. For example this DFA only allows letters in an identifier whereas in your C language you can use digits and underscore in identifier as well.

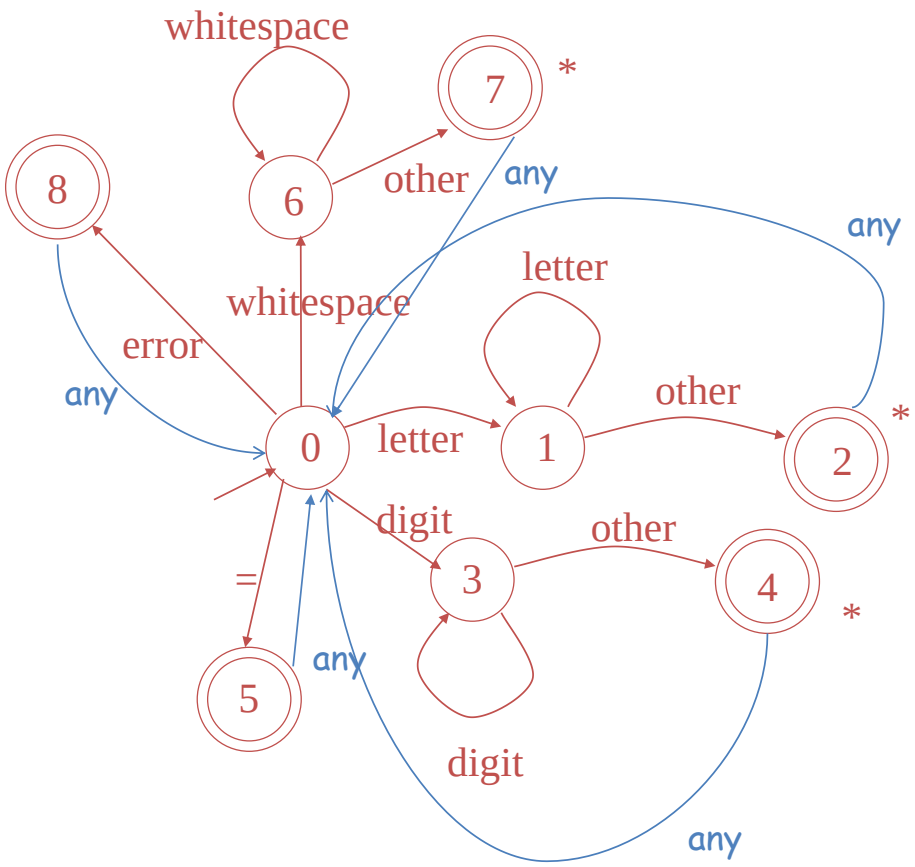


Figure 1: DFA

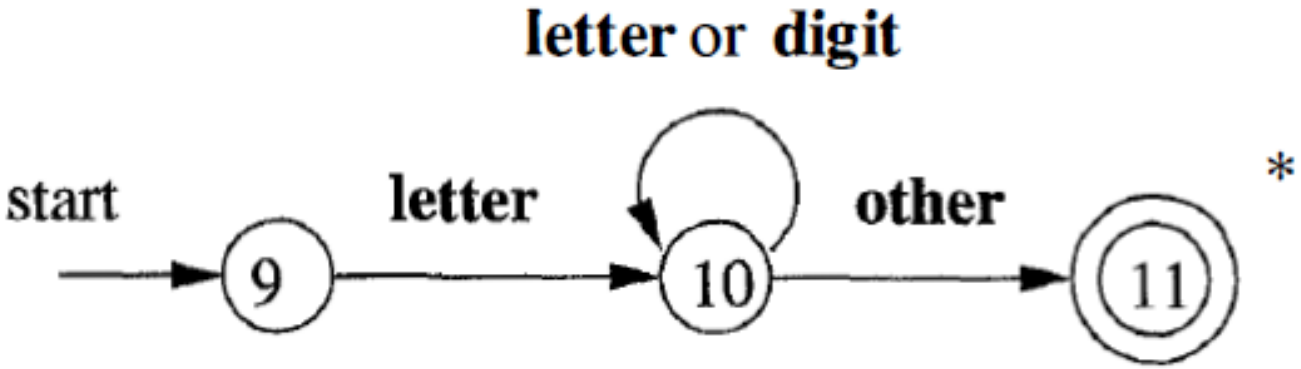
Submission

Email a zip file containing your complete project (all source files, along with whatever other files are needed to compile them; sample input and output files) to the following address:
maryam.bashir@cs.uol.edu.pk

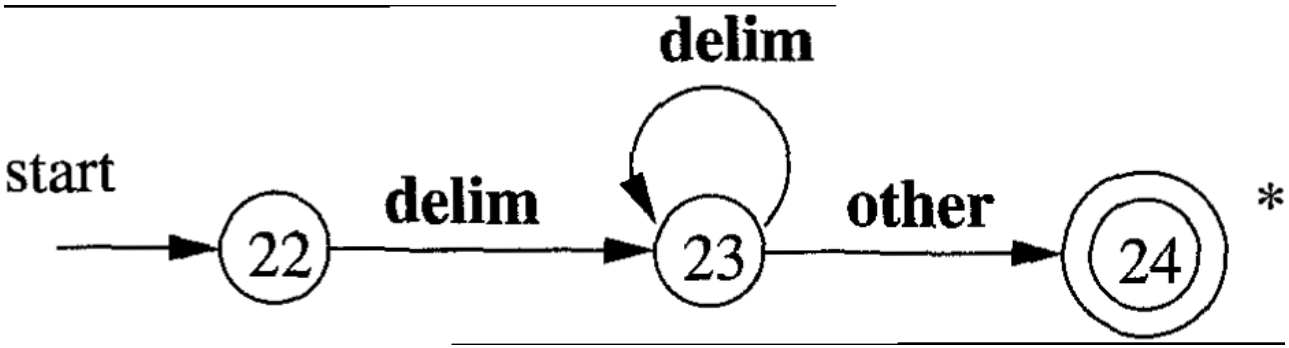
The name of zip file should be roll numbers of all students in the group as follows:
RollNumber1-RollNumber2

You should try to work on this assignment individually. If you think your programming is very weak then you can work in group size of maximum 2 students.

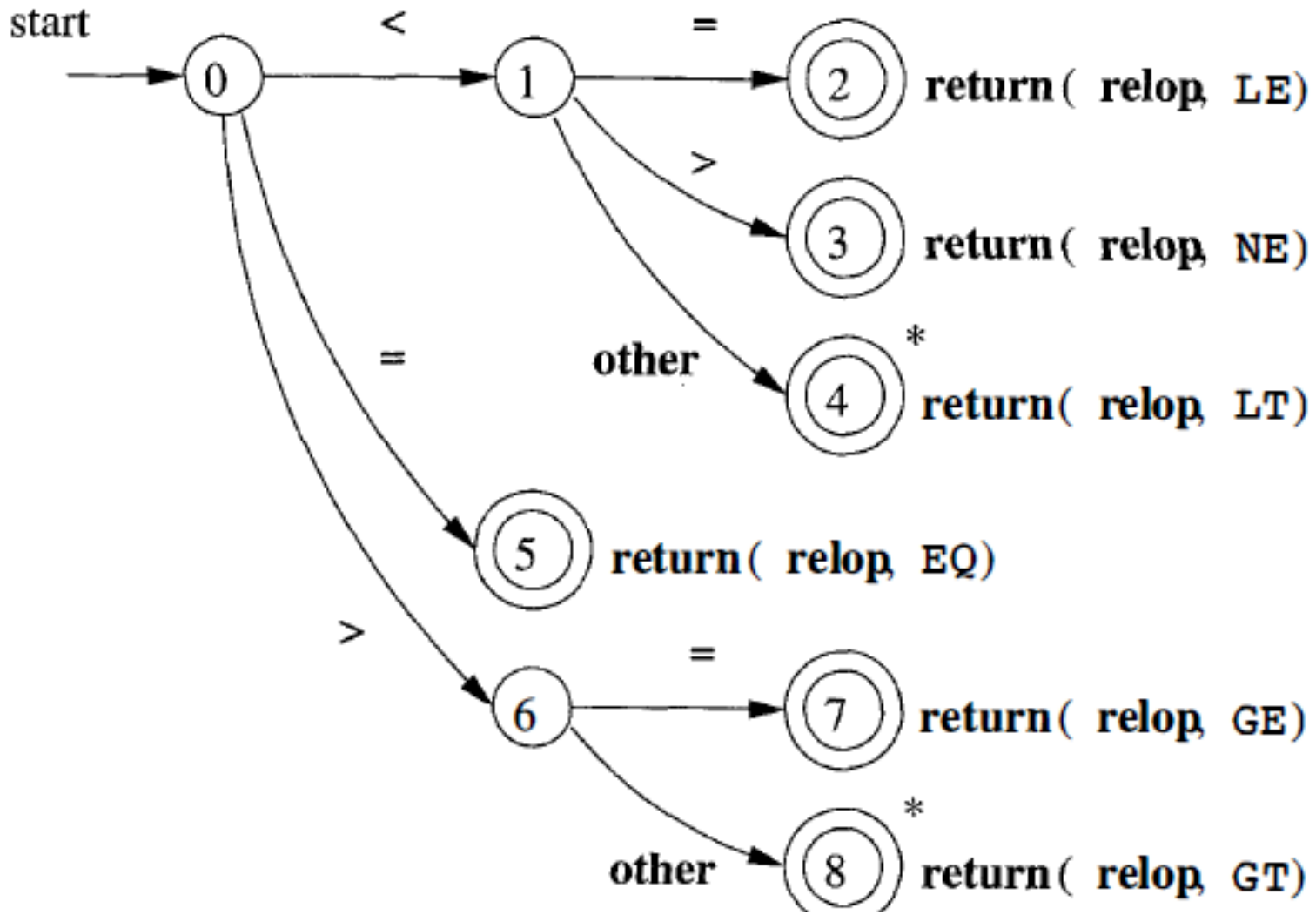
A Transition diagram for identifiers



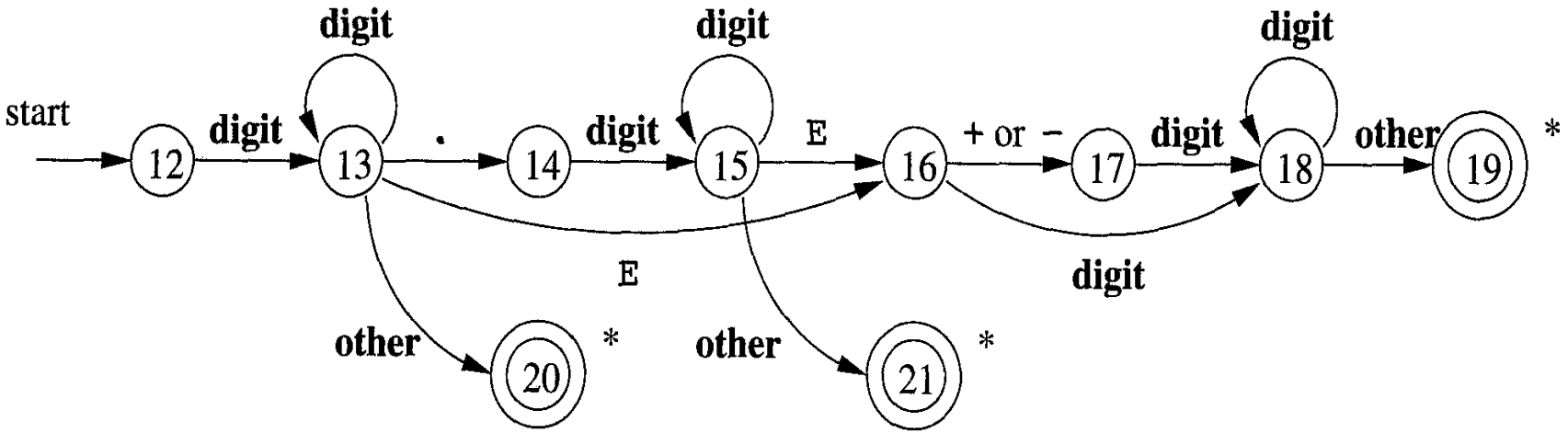
A transition diagram for whitespace



A transition diagram for relational operators



A transition diagram for unsigned numbers

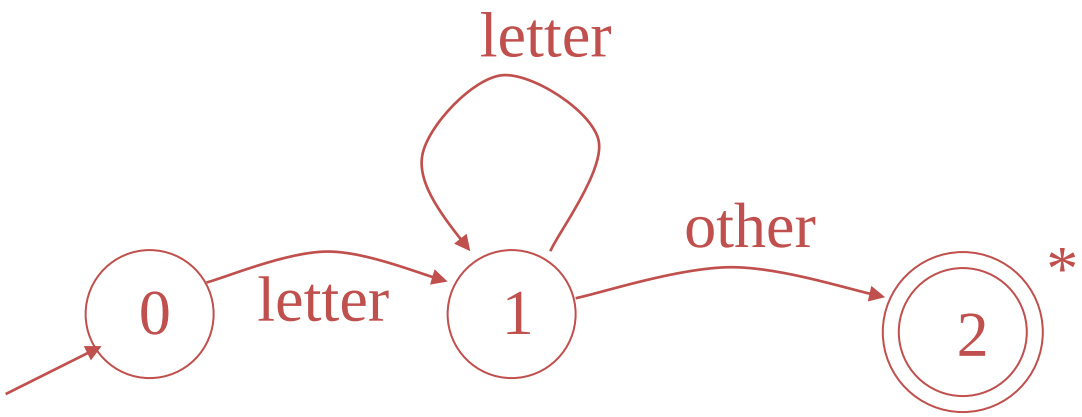


How to Merge Multiple Transition Diagrams

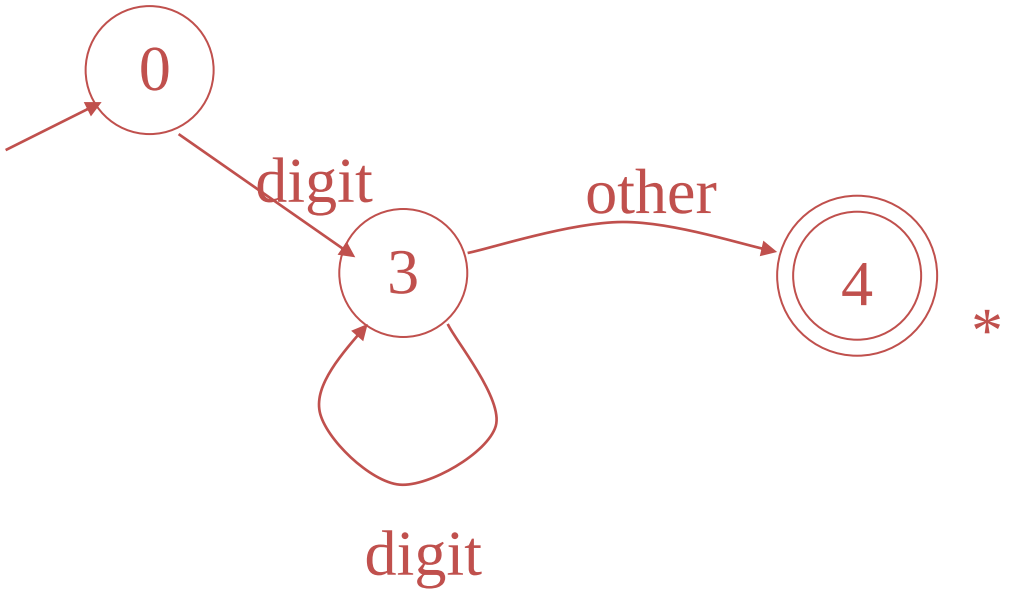
- Step 1: Merge start states of all transition diagrams

- Step2: Make a new transition on any input from final state of each transition diagram to the new start state

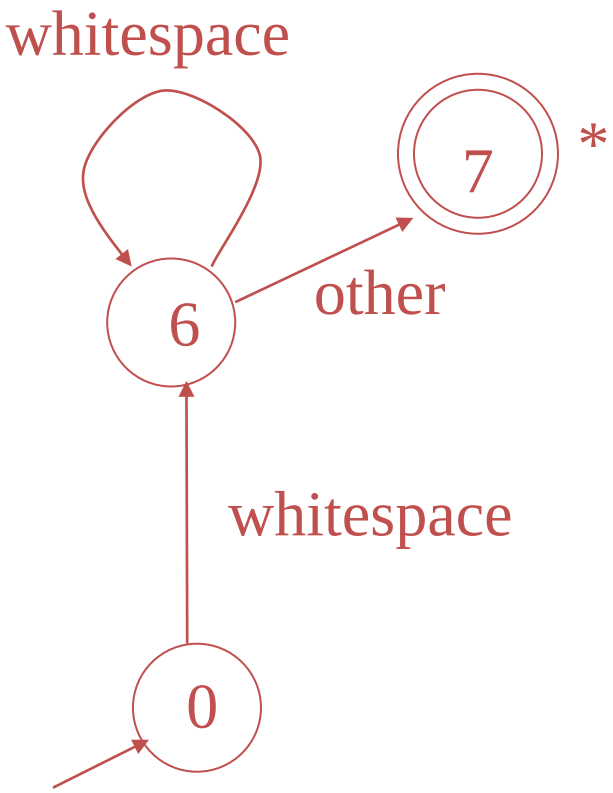
Transition Diagram for Identifiers made of only letters



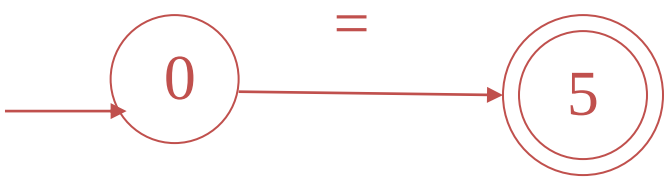
Transition Diagram for integers



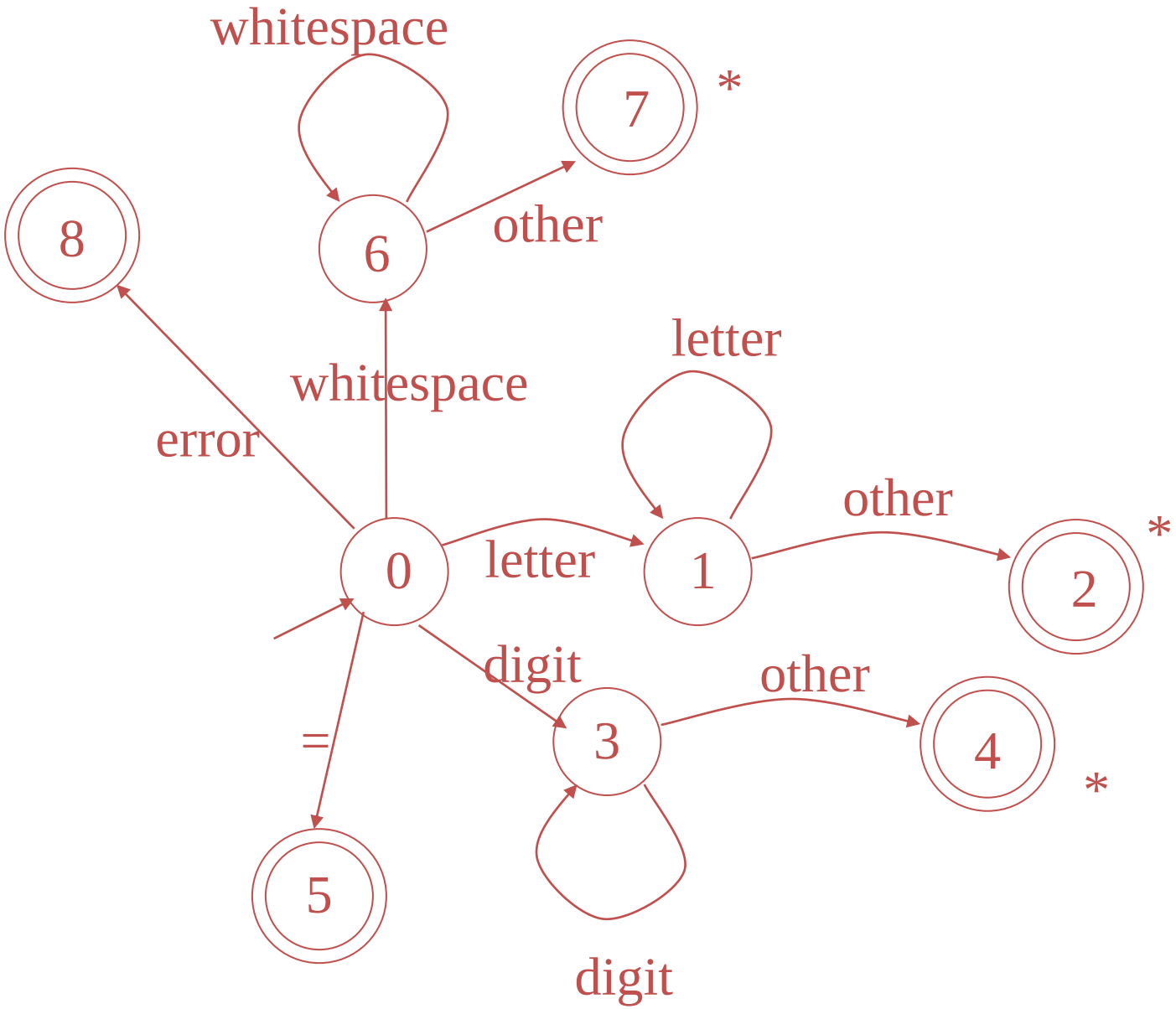
Transition Diagram for whitespace

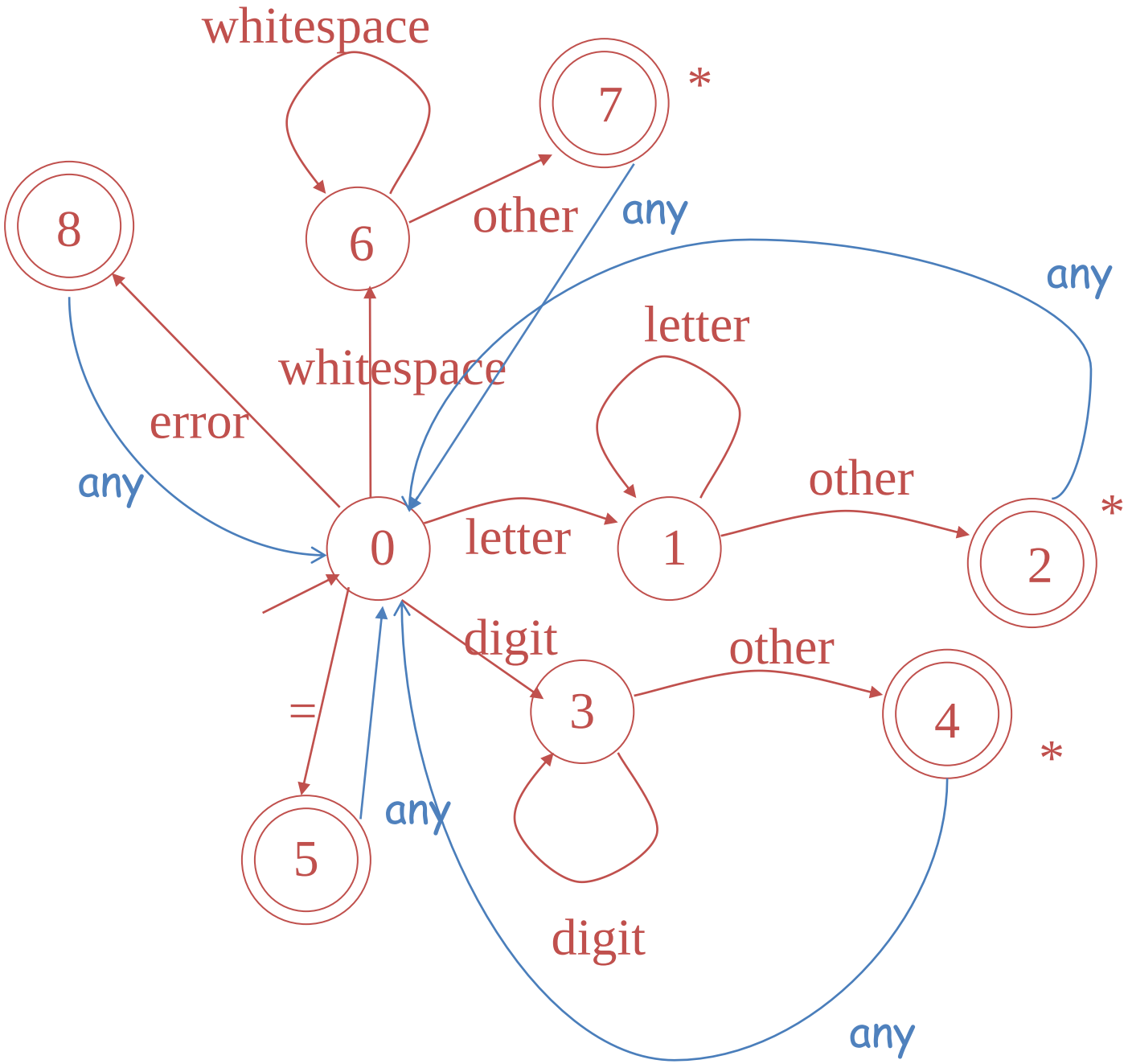


Transition Diagram for = operator



Transition Diagram for error


whitespace
7
*
8
6
other
letter
whitespace
error
other
*
0
letter
1
2
digit
other
=
3
4
*
5
digit


whitespace
7
*
8
6
other
any
any
letter
whitespace
error
any
other
*
0
letter
1
2
digit
other
=
3
4
*
any
5
digit
any