Sean Cox
Cory Caprioli
Caitlin San Pedro

CS323 Documentation

About 2 pages

1. Problem Statement

To write a lexical analyzer that reads an input file and tokenize the contents, printing out the token and lexeme in an output file using finite state machines

2. How to use your program

(how i did it on tuffix)

1. download zip folder on to desktop
2. unzip folder
3. open folder in editor
4. open terminal and run commands
5. "cd Desktop"
6. "cd CPSC323-project1-main/"
7. "clang++ -std=c++11 main.cpp lexer.cpp -o main"
8. execute with ./main
9. enter the input file and the output will be printed in output.txt
10. add your own input files

3. Design of your program

Data Structures: vector, 2d array

REs:
id = l(l | d | _)*
real = d+.d+
int = d+

2 DFSMs:

- one for integers and real numbers
- one for identifiers

NFSM (Thompson Construction Method): next page

Sean Cox
Cory Caprioli
Caitlin San Pedro

Documentation (Design)

Individual ε-Closures:

integer: d^+

$$\begin{tikzcd}
1 \arrow[r, "d"] & 2 \arrow[l, bend right=60, "\varepsilon"']
\end{tikzcd}$$

	d	ε
1	[2]	[ ]
2	[ ]	[1]

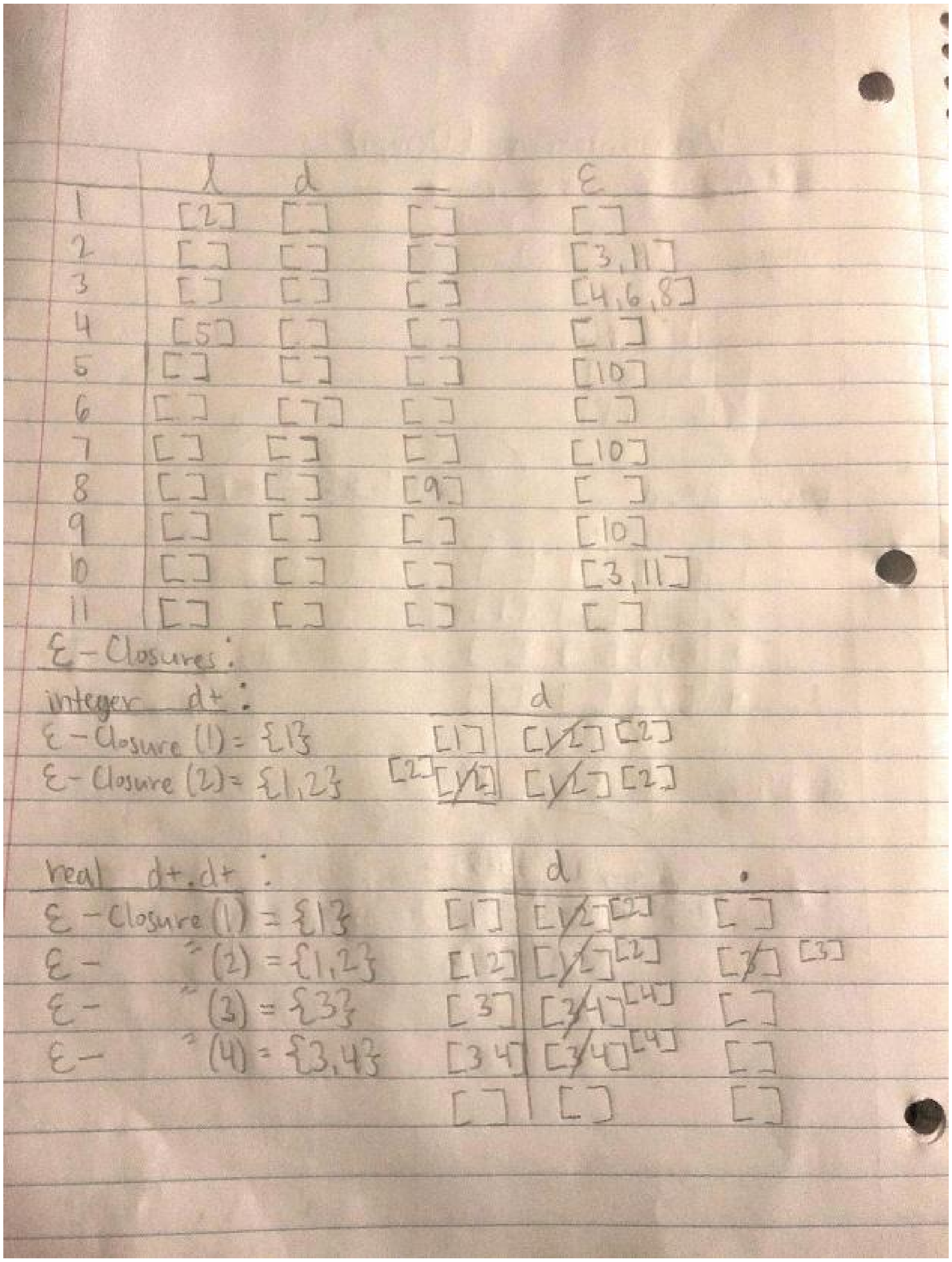
real: $d^+ . d^+$

$$\begin{tikzcd}
1 \arrow[r, "d"] & 2 \arrow[r, "\cdot"] \arrow[l, bend right=60, "\varepsilon"'] & 3 \arrow[r, "d"] & 4 \arrow[l, bend right=60, "\varepsilon"']
\end{tikzcd}$$

	d	•	ε
1	[2]	[ ]	[ ]
2	[ ]	[3]	[1]
3	[4]	[ ]	[ ]
4	[ ]	[ ]	[3]

identifier: $l(l \mid d \mid \_)^*$

$$\begin{tikzcd}
 & & & 4 \arrow[r, "l"] & 5 \arrow[dr] & & \\
1 \arrow[r, "l"] & 2 \arrow[r, "\varepsilon"] \arrow[rrrrr, bend right=60, "\varepsilon"'] & 3 \arrow[ur, "\varepsilon"] \arrow[r, "\varepsilon"] \arrow[dr, "\varepsilon"'] & 6 \arrow[r, "d"] & 7 \arrow[r] & 10 \arrow[r] \arrow[lll, bend right=80, "\varepsilon"'] & 11 \\
 & & & 8 \arrow[r, "\_"] & 9 \arrow[ur] & &
\end{tikzcd}$$

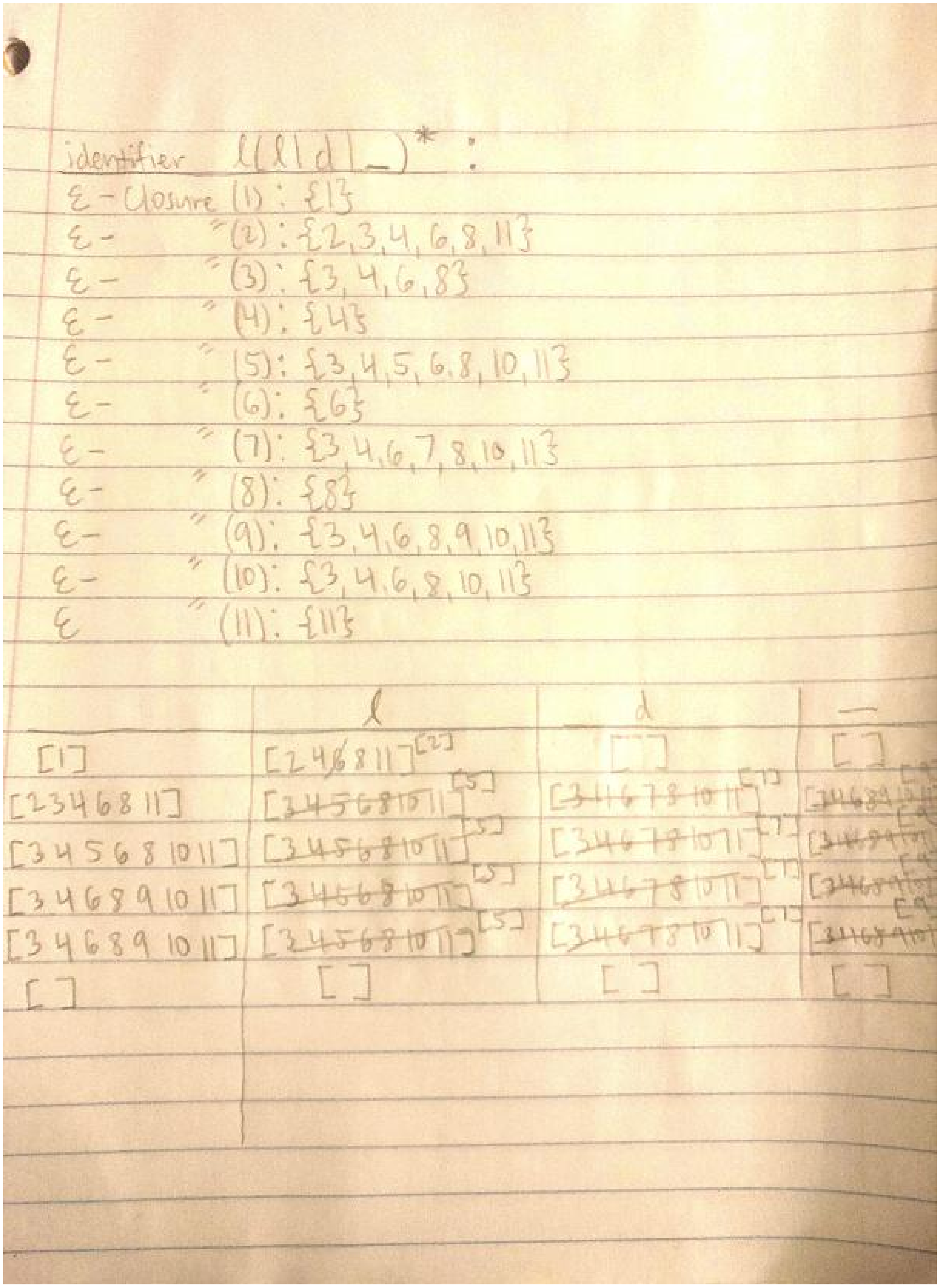


	l	d	_	ε
1	[2]	[ ]	[ ]	[ ]
2	[ ]	[ ]	[ ]	[3, 11]
3	[ ]	[ ]	[ ]	[4, 6, 8]
4	[5]	[ ]	[ ]	[ ]
5	[ ]	[ ]	[ ]	[10]
6	[ ]	[7]	[ ]	[ ]
7	[ ]	[ ]	[ ]	[10]
8	[ ]	[ ]	[9]	[ ]
9	[ ]	[ ]	[ ]	[10]
10	[ ]	[ ]	[ ]	[3, 11]
11	[ ]	[ ]	[ ]	[ ]

ε – Closures:

integer d+:		d
ε – Closure (1) = {1}	[1]	[~~1~~2] [2]
ε – Closure (2) = {1, 2}	[2] [~~1~~2]	[~~1~~2] [2]

real d+.d+:		d	.
ε – Closure (1) = {1}	[1]	[~~1~~2] [2]	[ ]
ε – " (2) = {1, 2}	[1 2]	[~~1~~2] [2]	[~~3~~] [3]
ε – " (3) = {3}	[3]	[~~3~~4] [4]	[ ]
ε – " (4) = {3, 4}	[3 4]	[~~3~~4] [4]	[ ]
	[ ]	[ ]	[ ]



identifier l(l|d|_)* :

ε-Closure (1) : {1}
ε- " (2) : {2, 3, 4, 6, 8, 11}
ε- " (3) : {3, 4, 6, 8}
ε- " (4) : {4}
ε- " (5) : {3, 4, 5, 6, 8, 10, 11}
ε- " (6) : {6}
ε- " (7) : {3, 4, 6, 7, 8, 10, 11}
ε- " (8) : {8}
ε- " (9) : {3, 4, 6, 8, 9, 10, 11}
ε- " (10) : {3, 4, 6, 8, 10, 11}
ε " (11) : {11}

	l	d	_
[1]	[2 4 6 8 11] [2]	[ ]	[ ]
[2 3 4 6 8 11]	~~[3 4 5 6 8 10 11]~~ [5]	~~[3 4 6 7 8 10 11]~~ [7]	~~[3 4 6 8 9~~ [9
[3 4 5 6 8 10 11]	~~[3 4 5 6 8 10 11]~~ [5]	~~[3 4 6 7 8 10 11]~~ [7]	~~[3 4 6 9~~ [9
[3 4 6 8 9 10 11]	~~[3 4 5 6 8 10 11]~~ [5]	~~[3 4 6 7 8 10 11]~~ [7]	~~[3 4 6 8 9~~ [9
[3 4 6 8 9 10 11]	~~[3 4 5 6 8 10 11]~~ [5]	~~[3 4 6 7 8 10 11]~~ [7]	~~[3 4 6 8 9 10~~ [9
[ ]	[ ]	[ ]	[ ]

4. Any Limitation

1. some keywords, operators, or separators may be missing from the lists, we included as many as we could think of

2. cannot read the “::” and ‘\n’ operators (eg. std::cout << “hello\n”)

3. “++” and “--” operators need a space between former identifier or identifier won’t be read (eg. i++ needs to be i ++ to read the i)

5. **Any shortcomings**
   none