41080 Theory of Computing Science Assignment 1 – Finite Automata

By Matthew Mallos 13255587

Submitted September 6, 2019

EXECUTIVE SUMMARY

A report detailing and recounting an implementation of a deterministic finite automation/transducer that takes in a string representing an arithmetic statement and produces an output sequence of tokens that represent the numbers and operators. The transducer also stores the correct values of these numbers within these tokens. Finally, the program handles errors, including incorrect representations of numbers, and incorrect representations of expressions.

HOW DID YOU EXTRACT FINITE AUTOMATA FROM THE SPECIFICATION? **/** WHAT IS THE FORMAL SPECIFICATION OF YOUR AUTOMATA?

Reducing the assignment into the base elements of an automata form crucially beneficial building blocks for my would-be transducer. I began by attempting to decipher the formal language of my would-be automaton. Using the formatting definition of a number and the list of valid expressions, I created the following:

L = {x ∈ B∗ | x is a valid number OR a valid expression, followed by an operator, followed by a valid number. A valid number is a string of digits, possibly followed by a decimal point and a non-empty string of digits. If there is a decimal point, the string before the decimal point should consist of “0”, If the number starts with “0”, it is either “0” or “0.[something]” where [something] = a string of digits}

Somewhat messy however it did lay the groundwork for what my Java program would need to recognise and assign as tokens. The next and most useful method for extracting a finite automata was to build a Deterministic Finite Automata graphical representation. Originally, I planned to do this on a whiteboard; however, very quickly ran out of space. Thankfully, I found an extremely useful [builder online](http://madebyevan.com/fsm/) (Wallace, 2010) that allowed me to create the DFA and contort it into a semi-understandable format.

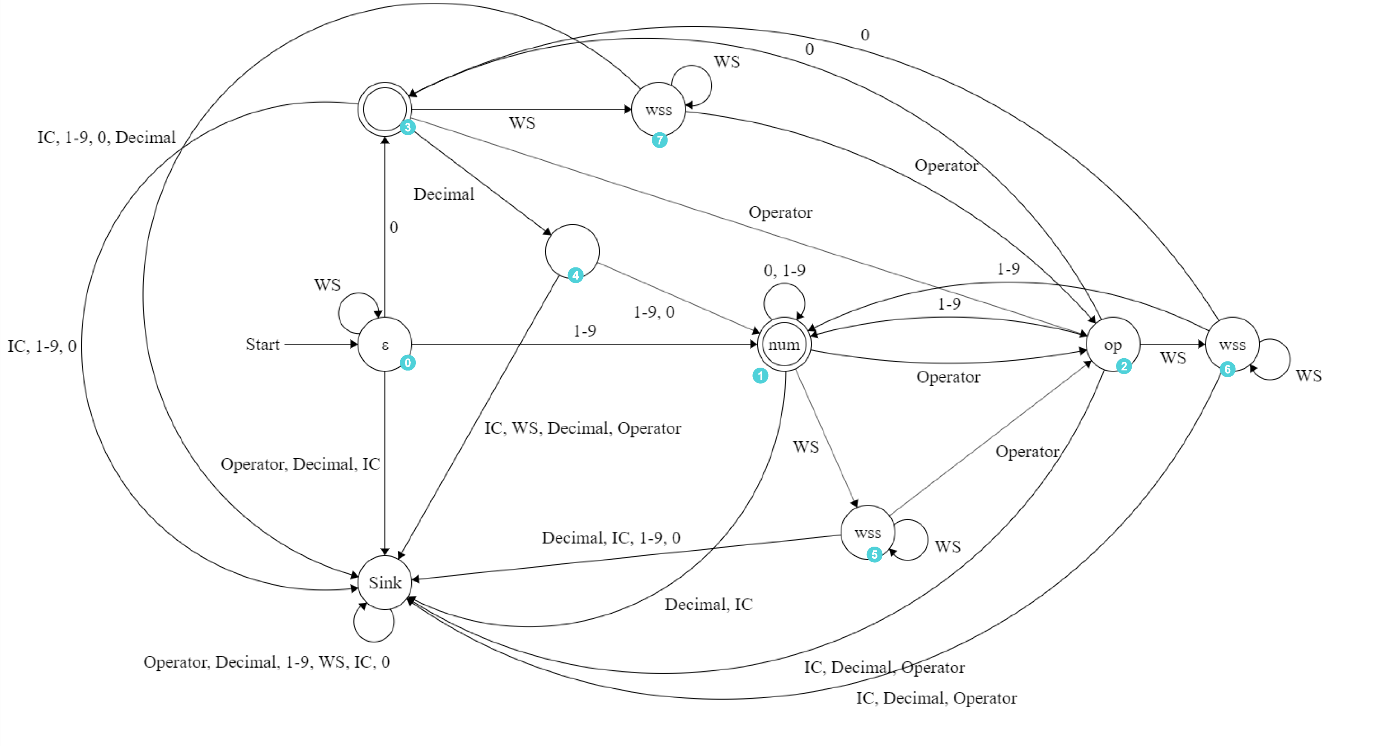


Figure 1 DFA Graphical Representation

This DFA contains a total of nine states, only two of which are accept states (when the expression ends in a digit, either 0 or 1-9 respectively). Three of the states are to handle whitespace; there are also an empty state, sink state and one special state that handles a decimal following a 0.

Extracting finite automata was a great challenge for this specification due to the increased number of options and transitions a state could have. To help keep track, I summarised all possible inputs into six categories:

* Decimal (.)
* Operator (+,-,/,\*)
* Number > 0 (1,2,3,4,5,6,7,8,9)
* Zero (0)
* Whitespace ( )
* Invalid Character (i.e. a letter or invalid symbol, e.g. @, B, “)

Using these six categories, I looked at every state and determined what the transition would be depending on the character the program was reading. Due to the many fault points of a mathematical expression, the sink state was frequently transitioned to by all states, all in multiple ways. To further assist with understanding the DFA, I created a transition/equivalence table and described as simplistically as possible, the representation for each state. These are pictured below:

**States**

**>0:** “I have seen nothing yet or any whitespace before an expression.”

**1:** “I have just seen a starting digit from 1-9 or a non-starting digit from 0-9.”

**2:** “I have just seen an operator.”

**3:** “I have just seen a zero, and it is the start of a number

**4:** “I have seen a decimal after zero.”

**5:** “I have just seen whitespace following a valid number.”

**6:** “I have just seen whitespace after an operator.”

**7:** “I have just seen whitespace following a number containing only zero.”

**Sink:** “I have seen an incorrect expression.”

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **State** | **Num**  **(0)** | **Num (1-9)** | **Operator** | **Decimal** | **White Space** | **Invalid Character** |
| **>0** | 3 | 1 | Sink | Sink | 0 | Sink |
| **1** | 1 | 1 | 2 | Sink | 5 | Sink |
| **2** | 3 | 1 | Sink | Sink | 6 | Sink |
| **3** | Sink | Sink | 2 | 4 | 7 | Sink |
| **4** | 1 | 1 | Sink | Sink | Sink | Sink |
| **5** | Sink | Sink | 2 | Sink | 5 | Sink |
| **6** | 3 | 1 | Sink | Sink | 6 | Sink |
| **7** | Sink | Sink | 2 | Sink | 7 | Sink |
| **Sink** | Sink | Sink | Sink | Sink | Sink | Sink |

Table 1 Equivalence Table

This table allows me to visualise succinctly and quickly the outcome of an expression at each character. Using this, I can build a program that evaluates the expression one character at a time and changes the “rules” by which the program will evaluate the following character.

HOW DOES YOUR CODE IMPLEMENT THE FINITE AUTOMATA THAT YOU DESIGNED?

Implementing code to replicate finite automata was a unique and unfamiliar concept to me. Hence my solution was very straightforward. The most readable and logical visualisation for me when it came to implementing code to replicate the automata was to treat each state as a switch allowing a large for loop to surround everything effectively “transitioning” every input-loop depending on the rules of each switch. This method allowed me to keep track of every state simplistically and “follow” the process of the program of reading the input. The skeleton provided allowed for an easy assigning of each character to the correct token through the line:

1. result.add(**new** Token(Token.typeOf(input.charAt(i))));
2. result.get(temp).setValue(value);

Figure 2 Code snippet of adding new tokens and setting values

That is, the result array list would (also using a switch) use the input character and match it to the correct token.

Implementing finite automata requires the generation of tokens; this takes place at states or transitions. My design relies on the transitions to avoid the generation of extra states (i.e. extra states for more specific examples of whitespace/number endings). This design implies more information being passed at transitions, for example, when my program transitions to the number state after seeing a digit, the execution changes depending on the transition it is taking due to the unique state. When the program transitions from the state where it sees a decimal just after a zero, it then assigns a token and prepares to record and overwrite numeric input. This assigning differs in the case the transition is from a start state, where the program assigns the token during the transition with the first digit in the possible multi-digit number. This design model required extended thought into each transition yet allowed the states to be more simplistic and efficient.

The sink state I had designed was, in hindsight, not a perfect representation of the given language. This incorrect definition is due to the need for two sink states, one for incorrect numerical exceptions, and another for incorrect expression exceptions. To circumvent this oversight, I built a final check into my lexical analyser that, before returning a result, will confirm what state it is at and decide whether to return the result. This check begins to conflict with other expressions I had already built into the switches/states and brings into question if my solution was a “perfect” representation of a DFA. Fortunately, this can also be looked at as an improvement to the automata as the program can define an incorrect representation without needing to waste processing power on continuing an expression already in a sink state, and should it fail to catch the error during the processing, find said error before returning a successful result.

DID YOU USE ANY ADDITIONAL TECHNIQUES TO IMPROVE THE AUTOMATA?

As implied above, the implementation of sink states within my Java program was supposed to be proactive and timesaving. For example, when given an incorrect expression

(i.e. “1 + + 2”), the transducer will return an expression error before it begins reading the whitespace after the second “plus” as a case has been set within the state to handle such error. This early query saves processing time as the program won’t loop through the remaining characters in a sink state as it already knows the expression is incorrect.

Furthermore, was the case of incorrect characters; these needed to be implemented specifically in the two accept states. A simple default clause for each resulted in ensuring expressions ended correctly; They were unnecessary in other switches due to the already existing sink state that would catch them. This method, on the other hand, is thought of as “implementing in two areas” and may confuse and muddy up the representation of a DFA as the premise of improving the automata becomes a game of ensuring I catch every error before it needs to enter the sink state.

DID YOU ENCOUNTER ANY CHALLENGES OR LIMITATIONS, EITHER TECHNICAL OR CONCEPTUAL, IN IMPLEMENTING A THEORETICAL CONSTRUCT IN A CONCRETE PROGRAMMING LANGUAGE?

This lexical analyser was one of the most challenging theoretical-based projects I have worked on, and so, came with a slew of challenges. Particularly with ensuring the DFA did not attempt to “look forward or backward” and especially when attempting to add tokens, I needed to ensure it was adding them in the correct order. To accomplish this, I made a “temp” variable that would serve the purpose of keeping the system aware of not just what token it was at, but also what character it was reading. The ‘temp’ variable’s use adjusts according to the state it is being used in (however may have made the program less confusing if I just added another variable). For example, when the program was adding numbers to expression and suddenly encountered an operator, the temp variable would be assigned to the length of the token array list after adding said operator. The assigning of the array length to the temp variable executes so that when the program encounters a new number, it sets the correct value to the token (which is due to a custom “setValue” method I created in the token class, see below).

1. **public** **void** setValue(**double** newValue) {
2. value = newValue;
3. }

Figure 3 The setValue method I added to the token class

As mentioned earlier, implementing a sink state becomes confusing due to the multiple areas I was writing this function into existence; instead of making the sink state its specific case outright. This overload of exception catches was due to an attempt to improve the concept of a sink state without wasting time running through a DFA when an expression was doomed to fail. I thought heavily on if this would almost make my program provide NFA attributes to its DFA intentions however concluded that the program when returning these errors, was only interpreting the character it was currently reading, as it should.

Even though I intended to make an easily readable interpretation of a DFA; The mass of switches and cases became difficult to follow when running lengthy test cases. The equivalence table and the graphical representation were much simpler to understand, and unfortunately, I don’t see a way to make a java program as readable as these. What assisted me, was labelling my switches and variables through frequent commenting to connect the table, graph and program’s elements visually.

In sum, this challenge showcased the limitations and capabilities of a standard coding language, while this DFA was able to be transferred to Java, other, more complex DFAs may be near impossible or require immense coding to convert.

REFERENCES

Wallace, 2010 *Finite State Machine Designer*, viewed 25 August 2019

<<http://madebyevan.com/fsm/>>

Mathieson, L. 2019 “Regular Expressions”

UTS online Subject 41080, Lecture notes, UTS, Sydney, viewed 1st September 2019

DFA IN SVG

<?xml version="1.0" standalone="no"?>

<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 1.1//EN" "http://www.w3.org/Graphics/SVG/1.1/DTD/svg11.dtd">

<svg width="800" height="600" version="1.1" xmlns="http://www.w3.org/2000/svg">

<ellipse stroke="black" stroke-width="1" fill="none" cx="443.5" cy="388.5" rx="30" ry="30"/>

<text x="439.5" y="394.5" font-family="Times New Roman" font-size="20">&#949;</text>

<ellipse stroke="black" stroke-width="1" fill="none" cx="888.5" cy="388.5" rx="30" ry="30"/>

<text x="870.5" y="394.5" font-family="Times New Roman" font-size="20">num</text>

<ellipse stroke="black" stroke-width="1" fill="none" cx="888.5" cy="388.5" rx="24" ry="24"/>

<ellipse stroke="black" stroke-width="1" fill="none" cx="443.5" cy="653.5" rx="30" ry="30"/>

<text x="425.5" y="659.5" font-family="Times New Roman" font-size="20">Sink</text>

<ellipse stroke="black" stroke-width="1" fill="none" cx="1254.5" cy="388.5" rx="30" ry="30"/>

<text x="1244.5" y="394.5" font-family="Times New Roman" font-size="20">op</text>

<ellipse stroke="black" stroke-width="1" fill="none" cx="443.5" cy="127.5" rx="30" ry="30"/>

<ellipse stroke="black" stroke-width="1" fill="none" cx="443.5" cy="127.5" rx="24" ry="24"/>

<ellipse stroke="black" stroke-width="1" fill="none" cx="748.5" cy="127.5" rx="30" ry="30"/>

<text x="737.5" y="133.5" font-family="Times New Roman" font-size="20">ws</text>

<ellipse stroke="black" stroke-width="1" fill="none" cx="981.5" cy="597.5" rx="30" ry="30"/>

<text x="966.5" y="603.5" font-family="Times New Roman" font-size="20">wss</text>

<ellipse stroke="black" stroke-width="1" fill="none" cx="1387.5" cy="388.5" rx="30" ry="30"/>

<text x="1372.5" y="394.5" font-family="Times New Roman" font-size="20">wss</text>

<ellipse stroke="black" stroke-width="1" fill="none" cx="652.5" cy="285.5" rx="30" ry="30"/>

<polygon stroke="black" stroke-width="1" points="331.5,388.5 413.5,388.5"/>

<text x="288.5" y="394.5" font-family="Times New Roman" font-size="20">Start</text>

<polygon fill="black" stroke-width="1" points="413.5,388.5 405.5,383.5 405.5,393.5"/>

<polygon stroke="black" stroke-width="1" points="473.5,388.5 858.5,388.5"/>

<polygon fill="black" stroke-width="1" points="858.5,388.5 850.5,383.5 850.5,393.5"/>

<text x="652.5" y="379.5" font-family="Times New Roman" font-size="20">1-9</text>

<path stroke="black" stroke-width="1" fill="none" d="M 875.275,361.703 A 22.5,22.5 0 1 1 901.725,361.703"/>

<text x="865.5" y="312.5" font-family="Times New Roman" font-size="20">0, 1-9</text>

<polygon fill="black" stroke-width="1" points="901.725,361.703 910.473,358.17 902.382,352.292"/>

<polygon stroke="black" stroke-width="1" points="443.5,418.5 443.5,623.5"/>

<polygon fill="black" stroke-width="1" points="443.5,623.5 448.5,615.5 438.5,615.5"/>

<text x="260.5" y="527.5" font-family="Times New Roman" font-size="20">Operator, Decimal, IC</text>

<path stroke="black" stroke-width="1" fill="none" d="M 1225.171,394.8 A 801.692,801.692 0 0 1 917.829,394.8"/>

<polygon fill="black" stroke-width="1" points="1225.171,394.8 1216.361,391.426 1218.278,401.241"/>

<text x="1035.5" y="430.5" font-family="Times New Roman" font-size="20">Operator</text>

<path stroke="black" stroke-width="1" fill="none" d="M 448.971,682.878 A 22.5,22.5 0 1 1 423.51,675.712"/>

<text x="174.5" y="747.5" font-family="Times New Roman" font-size="20">Operator, Decimal, 1-9, WS, IC, 0</text>

<polygon fill="black" stroke-width="1" points="423.51,675.712 414.132,676.744 420.328,684.593"/>

<path stroke="black" stroke-width="1" fill="none" d="M 889.863,418.455 A 286.119,286.119 0 0 1 469.188,668.97"/>

<polygon fill="black" stroke-width="1" points="469.188,668.97 473.896,677.145 478.599,668.32"/>

<text x="755.5" y="683.5" font-family="Times New Roman" font-size="20">Decimal, IC</text>

<path stroke="black" stroke-width="1" fill="none" d="M 917.147,379.604 A 568.952,568.952 0 0 1 1225.853,379.604"/>

<polygon fill="black" stroke-width="1" points="917.147,379.604 926.203,382.246 923.491,372.621"/>

<text x="1058.5" y="349.5" font-family="Times New Roman" font-size="20">1-9</text>

<polygon stroke="black" stroke-width="1" points="443.5,358.5 443.5,157.5"/>

<polygon fill="black" stroke-width="1" points="443.5,157.5 438.5,165.5 448.5,165.5"/>

<text x="448.5" y="264.5" font-family="Times New Roman" font-size="20">0</text>

<polygon stroke="black" stroke-width="1" points="473.5,127.5 718.5,127.5"/>

<polygon fill="black" stroke-width="1" points="718.5,127.5 710.5,122.5 710.5,132.5"/>

<text x="581.5" y="148.5" font-family="Times New Roman" font-size="20">WS</text>

<path stroke="black" stroke-width="1" fill="none" d="M 416.069,376.646 A 22.5,22.5 0 1 1 436.224,359.516"/>

<text x="364.5" y="328.5" font-family="Times New Roman" font-size="20">WS</text>

<polygon fill="black" stroke-width="1" points="436.224,359.516 440.601,351.159 430.63,351.92"/>

<polygon stroke="black" stroke-width="1" points="900.696,415.909 969.304,570.091"/>

<polygon fill="black" stroke-width="1" points="969.304,570.091 970.619,560.749 961.483,564.815"/>

<text x="897.5" y="508.5" font-family="Times New Roman" font-size="20">WS</text>

<polygon stroke="black" stroke-width="1" points="1005.321,579.264 1230.679,406.736"/>

<polygon fill="black" stroke-width="1" points="1230.679,406.736 1221.288,407.629 1227.366,415.57"/>

<text x="1123.5" y="513.5" font-family="Times New Roman" font-size="20">Operator</text>

<polygon stroke="black" stroke-width="1" points="951.661,600.606 473.339,650.394"/>

<polygon fill="black" stroke-width="1" points="473.339,650.394 481.813,654.539 480.778,644.593"/>

<text x="618.5" y="609.5" font-family="Times New Roman" font-size="20">Decimal, IC, 1-9, 0</text>

<path stroke="black" stroke-width="1" fill="none" d="M 414.072,659.251 A 271.32,271.32 0 1 1 414.072,121.749"/>

<polygon fill="black" stroke-width="1" points="414.072,659.251 405.461,655.397 406.834,665.302"/>

<text x="23.5" y="396.5" font-family="Times New Roman" font-size="20">IC, 1-9, 0</text>

<polygon stroke="black" stroke-width="1" points="472.058,136.691 1225.942,379.309"/>

<polygon fill="black" stroke-width="1" points="1225.942,379.309 1219.859,372.099 1216.795,381.618"/>

<text x="848.5" y="247.5" font-family="Times New Roman" font-size="20">Operator</text>

<path stroke="black" stroke-width="1" fill="none" d="M 778.396,129.972 A 706.144,706.144 0 0 1 1235.155,365.573"/>

<polygon fill="black" stroke-width="1" points="1235.155,365.573 1233.619,356.265 1226.115,362.875"/>

<text x="1033.5" y="195.5" font-family="Times New Roman" font-size="20">Operator</text>

<path stroke="black" stroke-width="1" fill="none" d="M 414.209,647.068 A 324.299,324.299 0 1 1 728.367,105.274"/>

<polygon fill="black" stroke-width="1" points="414.209,647.068 407.78,640.164 405.185,649.821"/>

<text x="56.5" y="164.5" font-family="Times New Roman" font-size="20">IC, 1-9, 0, Decimal</text>

<path stroke="black" stroke-width="1" fill="none" d="M 469.727,112.942 A 569.82,569.82 0 0 1 1241.686,361.378"/>

<polygon fill="black" stroke-width="1" points="469.727,112.942 479.132,113.679 474.511,104.811"/>

<text x="911.5" y="66.5" font-family="Times New Roman" font-size="20">0</text>

<path stroke="black" stroke-width="1" fill="none" d="M 1244.561,416.801 A 526.058,526.058 0 0 1 468.233,670.472"/>

<polygon fill="black" stroke-width="1" points="468.233,670.472 472.245,679.01 477.666,670.607"/>

<text x="910.5" y="753.5" font-family="Times New Roman" font-size="20">IC, Decimal, Operator</text>

<polygon stroke="black" stroke-width="1" points="1284.5,388.5 1357.5,388.5"/>

<polygon fill="black" stroke-width="1" points="1357.5,388.5 1349.5,383.5 1349.5,393.5"/>

<text x="1306.5" y="409.5" font-family="Times New Roman" font-size="20">WS</text>

<path stroke="black" stroke-width="1" fill="none" d="M 914.701,373.897 A 485.811,485.811 0 0 1 1361.299,373.897"/>

<polygon fill="black" stroke-width="1" points="914.701,373.897 924.104,374.66 919.507,365.779"/>

<text x="1124.5" y="310.5" font-family="Times New Roman" font-size="20">1-9</text>

<path stroke="black" stroke-width="1" fill="none" d="M 1376.318,416.335 A 607.143,607.143 0 0 1 467.534,671.449"/>

<polygon fill="black" stroke-width="1" points="467.534,671.449 471.169,680.155 476.952,671.997"/>

<text x="968.5" y="787.5" font-family="Times New Roman" font-size="20">IC, Decimal, Operator</text>

<path stroke="black" stroke-width="1" fill="none" d="M 469.803,113.078 A 687.267,687.267 0 0 1 1372.341,362.614"/>

<polygon fill="black" stroke-width="1" points="469.803,113.078 479.208,113.821 474.592,104.95"/>

<text x="973.5" y="50.5" font-family="Times New Roman" font-size="20">0</text>

<path stroke="black" stroke-width="1" fill="none" d="M 1010.312,589.573 A 22.5,22.5 0 1 1 1005.317,615.548"/>

<text x="1055.5" y="619.5" font-family="Times New Roman" font-size="20">WS</text>

<polygon fill="black" stroke-width="1" points="1005.317,615.548 1007.135,624.805 1014.435,617.97"/>

<path stroke="black" stroke-width="1" fill="none" d="M 1417.309,386.406 A 22.5,22.5 0 1 1 1407.295,410.887"/>

<text x="1457.5" y="429.5" font-family="Times New Roman" font-size="20">WS</text>

<polygon fill="black" stroke-width="1" points="1407.295,410.887 1407.253,420.321 1415.756,415.058"/>

<path stroke="black" stroke-width="1" fill="none" d="M 762.546,101.124 A 22.5,22.5 0 1 1 777.979,122.605"/>

<text x="809.5" y="80.5" font-family="Times New Roman" font-size="20">WS</text>

<polygon fill="black" stroke-width="1" points="777.979,122.605 785.953,127.647 786.006,117.647"/>

<polygon stroke="black" stroke-width="1" points="467.431,145.591 628.569,267.409"/>

<polygon fill="black" stroke-width="1" points="628.569,267.409 625.202,258.596 619.172,266.573"/>

<text x="475.5" y="227.5" font-family="Times New Roman" font-size="20">Decimal</text>

<polygon stroke="black" stroke-width="1" points="679.995,297.5 861.005,376.5"/>

<polygon fill="black" stroke-width="1" points="861.005,376.5 855.673,368.717 851.672,377.882"/>

<text x="719.5" y="358.5" font-family="Times New Roman" font-size="20">1-9, 0</text>

<polygon stroke="black" stroke-width="1" points="637.685,311.586 458.315,627.414"/>

<polygon fill="black" stroke-width="1" points="458.315,627.414 466.614,622.926 457.918,617.988"/>

<text x="554.5" y="487.5" font-family="Times New Roman" font-size="20">IC, WS, Decimal, Operator</text>

</svg>

DFA IN Latex

\documentclass[a3paper, landscape]{article}

\usepackage{tikz}

\begin{document}

\pdfpagewidth 18.25in

\pdfpageheight 10.75in

\begin{center}

\begin{tikzpicture}[scale=0.2]

\tikzstyle{every node}+=[inner sep=0pt]

\draw [black] (44.3,-38.8) circle (3);

\draw (44.3,-38.8) node {$\epsilon$};

\draw [black] (87.8,-39.5) circle (3);

\draw (87.8,-39.5) node {$num$};

\draw [black] (87.8,-39.5) circle (2.4);

\draw [black] (44.3,-65.3) circle (3);

\draw (44.3,-65.3) node {$Sink$};

\draw [black] (125.4,-38.8) circle (3);

\draw (125.4,-38.8) node {$op$};

\draw [black] (44.3,-12.7) circle (3);

\draw [black] (44.3,-12.7) circle (2.4);

\draw [black] (74.8,-12.7) circle (3);

\draw (74.8,-12.7) node {$ws$};

\draw [black] (98.1,-59.7) circle (3);

\draw (98.1,-59.7) node {$wss$};

\draw [black] (138.7,-38.8) circle (3);

\draw (138.7,-38.8) node {$wss$};

\draw [black] (65.2,-28.5) circle (3);

\draw [black] (33.1,-38.8) -- (41.3,-38.8);

\draw (32.6,-38.8) node [left] {$Start$};

\fill [black] (41.3,-38.8) -- (40.5,-38.3) -- (40.5,-39.3);

\draw [black] (47.3,-38.85) -- (84.8,-39.45);

\fill [black] (84.8,-39.45) -- (84.01,-38.94) -- (83.99,-39.94);

\draw (66.06,-38.62) node [above] {$1-9$};

\draw [black] (86.477,-36.82) arc (234:-54:2.25);

\draw (87.8,-32.25) node [above] {$0,\mbox{ }1-9$};

\fill [black] (89.12,-36.82) -- (90,-36.47) -- (89.19,-35.88);

\draw [black] (44.3,-41.8) -- (44.3,-62.3);

\fill [black] (44.3,-62.3) -- (44.8,-61.5) -- (43.8,-61.5);

\draw (43.8,-52.05) node [left] {$Operator,\mbox{ }Decimal,\mbox{ }IC$};

\draw [black] (122.476,-39.469) arc (-78.12071:-99.74618:84.58);

\fill [black] (122.48,-39.47) -- (121.59,-39.14) -- (121.8,-40.12);

\draw (106.67,-41.84) node [below] {$Operator$};

\draw [black] (44.847,-68.238) arc (38.28175:-249.71825:2.25);

\draw (31.11,-73.11) node [below] {$Operator,\mbox{ }Decimal,\mbox{ }1-9,\mbox{ }WS,\mbox{ }IC,\mbox{ }0$};

\fill [black] (42.3,-67.52) -- (41.36,-67.62) -- (41.98,-68.41);

\draw [black] (87.99,-42.492) arc (0.53405:-119.18939:27.665);

\fill [black] (46.83,-66.9) -- (47.29,-67.73) -- (47.78,-66.86);

\draw (79.82,-67.05) node [below] {$Decimal,\mbox{ }IC$};

\draw [black] (90.654,-38.578) arc (106.46993:75.66318:59.983);

\fill [black] (90.65,-38.58) -- (91.56,-38.83) -- (91.28,-37.87);

\draw (106.53,-35.59) node [above] {$1-9$};

\draw [black] (44.3,-35.8) -- (44.3,-15.7);

\fill [black] (44.3,-15.7) -- (43.8,-16.5) -- (44.8,-16.5);

\draw (44.8,-25.75) node [right] {$0$};

\draw [black] (47.3,-12.7) -- (71.8,-12.7);

\fill [black] (71.8,-12.7) -- (71,-12.2) -- (71,-13.2);

\draw (59.55,-13.2) node [below] {$WS$};

\draw [black] (41.557,-37.615) arc (274.36072:-13.63928:2.25);

\draw (37.92,-33.16) node [above] {$WS$};

\fill [black] (43.57,-35.9) -- (44.01,-35.07) -- (43.01,-35.14);

\draw [black] (89.16,-42.17) -- (96.74,-57.03);

\fill [black] (96.74,-57.03) -- (96.82,-56.09) -- (95.93,-56.54);

\draw (92.26,-50.73) node [left] {$WS$};

\draw [black] (100.48,-57.88) -- (123.02,-40.62);

\fill [black] (123.02,-40.62) -- (122.08,-40.71) -- (122.69,-41.51);

\draw (115.81,-49.75) node [below] {$Operator$};

\draw [black] (95.12,-60.01) -- (47.28,-64.99);

\fill [black] (47.28,-64.99) -- (48.13,-65.4) -- (48.03,-64.41);

\draw (69.54,-61.32) node [above] {$Decimal,\mbox{ }IC,\mbox{ }1-9,\mbox{ }0$};

\draw [black] (41.357,-65.875) arc (-82.10975:-277.89025:27.132);

\fill [black] (41.36,-65.88) -- (40.5,-65.49) -- (40.63,-66.48);

\draw (10,-39) node [left] {$IC,\mbox{ }1-9,\mbox{ }0$};

\draw [black] (47.16,-13.62) -- (122.54,-37.88);

\fill [black] (122.54,-37.88) -- (121.94,-37.16) -- (121.63,-38.11);

\draw (88.39,-25.09) node [above] {$Operator$};

\draw [black] (77.79,-12.947) arc (84.05522:41.37448:70.614);

\fill [black] (123.47,-36.51) -- (123.31,-35.58) -- (122.56,-36.24);

\draw (106.87,-19.91) node [above] {$Operator$};

\draw [black] (41.371,-64.657) arc (-105.03548:-315.17894:32.43);

\fill [black] (41.37,-64.66) -- (40.73,-63.97) -- (40.47,-64.93);

\draw (21.08,-15.83) node [left] {$IC,\mbox{ }1-9,\mbox{ }0,\mbox{ }Decimal$};

\draw [black] (46.923,-11.244) arc (117.52455:26.79651:56.982);

\fill [black] (46.92,-11.24) -- (47.86,-11.32) -- (47.4,-10.43);

\draw (91.59,-6.99) node [above] {$0$};

\draw [black] (124.406,-41.63) arc (-20.98491:-122.82475:52.606);

\fill [black] (46.77,-67) -- (47.17,-67.85) -- (47.72,-67.01);

\draw (99.95,-73.66) node [below] {$IC,\mbox{ }Decimal,\mbox{ }Operator$};

\draw [black] (128.4,-38.8) -- (135.7,-38.8);

\fill [black] (135.7,-38.8) -- (134.9,-38.3) -- (134.9,-39.3);

\draw (132.05,-39.3) node [below] {$WS$};

\draw [black] (90.413,-38.027) arc (117.71171:63.8641:50.396);

\fill [black] (90.41,-38.03) -- (91.35,-38.1) -- (90.89,-37.21);

\draw (113.15,-31.73) node [above] {$1-9$};

\draw [black] (137.582,-41.583) arc (-23.30185:-125.33713:60.714);

\fill [black] (46.7,-67.09) -- (47.07,-67.97) -- (47.65,-67.15);

\draw (105.75,-77.06) node [below] {$IC,\mbox{ }Decimal,\mbox{ }Operator$};

\draw [black] (46.93,-11.258) arc (117.48611:31.60345:68.727);

\fill [black] (46.93,-11.26) -- (47.87,-11.33) -- (47.41,-10.44);

\draw (97.75,-5.43) node [above] {$0$};

\draw [black] (100.981,-58.907) arc (133.11447:-154.88553:2.25);

\draw (105.46,-61.27) node [right] {$WS$};

\fill [black] (100.48,-61.5) -- (100.66,-62.43) -- (101.39,-61.75);

\draw [black] (141.681,-38.591) arc (121.75098:-166.24902:2.25);

\draw (145.69,-42.26) node [right] {$WS$};

\fill [black] (140.68,-41.04) -- (140.68,-41.98) -- (141.53,-41.46);

\draw [black] (76.205,-10.062) arc (179.69571:-108.30429:2.25);

\draw (80.87,-7.38) node [right] {$WS$};

\fill [black] (77.75,-12.21) -- (78.55,-12.71) -- (78.55,-11.71);

\draw [black] (46.69,-14.51) -- (62.81,-26.69);

\fill [black] (62.81,-26.69) -- (62.47,-25.81) -- (61.87,-26.61);

\draw (50.86,-21.1) node [below] {$Decimal$};

\draw [black] (67.9,-29.81) -- (85.1,-38.19);

\fill [black] (85.1,-38.19) -- (84.6,-37.39) -- (84.16,-38.29);

\draw (73.7,-34.52) node [below] {$1-9,\mbox{ }0$};

\draw [black] (63.72,-31.11) -- (45.78,-62.69);

\fill [black] (45.78,-62.69) -- (46.61,-62.24) -- (45.74,-61.75);

\draw (55.41,-48.12) node [right] {$IC,\mbox{ }WS,\mbox{ }Decimal,\mbox{ }Operator$};

\end{tikzpicture}

\end{center}

\end{document}