Compilers-II
Language Specification Document
Professor Rama Krishna Upadrasta

Automaton

V Sathwik - Project manager

Nalari Sushma - System Architect

P Rashmitha - System Integrator

L Harsha Vardhan - Language Guru

Danda Sarat Chandra - System Integrator

K Lakshmi Sravya - Tester

K Sri Teja - Tester

Contents

1 Introduction to Automaton

Our idea for the **Automaton** is to enable a programming language that emulates DFA (Deterministic Finite Automata).

The purpose of the language is to shrink the gap between the paper model of DFA and the running code. Automaton is organized and executed similar to an Automata diagram, emphasizes modularity and organization of code into short nodes that transition to each other until reaching one of the end states. Our language blends the functional and imperative programming styles to allow programmers to abstract away implementation details.

1.1 States in Automaton

Automaton programs contain nodes/states, and the transition between these states takes place based on the conditions we check.

Transition nodes include transition statements, which check the expressions, and use them if the statement is true. All transition nodes should be kept up to date. End node.

The state of an automata is defined by name, followed by flower brackets , which contains multiple functions inside it, with either the transition or return statements.

Start state: All the Automaton codes must begin with the "start" state, where the 1st state to be chosen is given.

Accepting state: If a finite state machine finishes an input string and is in an accepting state, the string is considered to be valid.

1.2 Modularity

The key to a good Automaton program is extreme flexibility. Being able to draw a program, on paper, like an automata means that you are probably on the right track. Nodes are systematic in a short concise manner along with ability to create new variables, the decisions mainly contain global information, and local variables are also available primarily for simplicity, efficiency, and shortening code length.

2 Lexical Conventions

2.1 Comments

Single line comments: They start with these characters.
Ex: Automaton is a domain specific language.
Multi line comments: They start with these characters /* and end with these characters */ . Between these characters /* & */ there can be any characters between them except the end symbol.
Ex: /* Automaton is a domain specific language, based on DFA */

2.2 Identifiers

An identifier is a sequence of letters, underscores, and digits. the first letter must be small and it is a case sensitive language.

2.3 Keywords

These are the keywords and can't be used as variables :
main, int, float, char, DFA, void, stack, start, exit, return, true, false, boolean, print.

2.4 Constants

These are some of the constants used in Automaton

2.4.1 Integer Constants

It consists of one or more than one digits and also one optional '-' (negative) sign.
Ex:
Valid : 10, 0, -12, etc.
Invalid : +10, 0.01, 00, etc.

2.4.2 Boolean Values

While no explicit Boolean constant type is expressed, any empty value (such as an empty sequence or list) or zero will evaluate to false and any other value will be evaluated as true.

2.4.3 Float Constants

It is a 64-bit signed floating point. It contains one or more digits followed by a decimal and another integer or a decimal point followed by an integer(positive integer).
Ex:
Valid: 2.43, 90.8, .6, -2.3.
Invalid: 42, 0, .-3.

2.4.4 String Constants

It contains a fixed store of characters, a string literal consisting of characters plus a final null character in between the double quotes.
Ex: "Automaton is domain specific language"

2.5 Punctuations

Punctuation	Use	Example
,	Multiple variables, array literal separation	function(int a, int b)
;	End of statement	int a = 10;
{}	Indicate function call, list of parameters for a state	int DFA main {}
()	Function arguments, expression precedence	void DFA2(string str)
""	String literal declaration	string s = "Hello, World!

2.6 Operators

All the operators in this language are similar to C++ operators. But the arrow operator(->) is used for state transition.

2.6.1 Arithmetic Operators

Operator	Use
+	Addition
-	Subtraction
*	Multiplication
/	Division
%	Modulo
++	Increment Operator
–	Decrement Operator

2.6.2 Relational Operators

Operator	Use
==	Test equivalence
!=	Test inequality
>	Greater than
<	Less than
≥	Greater than or equal to
≤	Less than or equal to

2.6.3 Logical Operators

Operator	Use
&&	AND
||	OR
.	Access

2.6.4 Assignment Operator

= is assignment Operator. It assigns values from right side operand to left side operand.

2.6.5 Arrow Operator

-> is Arrow Operator.This operator is used for transition between the states in the Definite Finite Automata.

2.7 Precedence and Associativity

The following table is the order of the precedence and Associativities are mentioned against them.

Operator	Associativity
->	Left to Right
,	Left to Right
=	Right to Left
||	Left to Right
&&	Left to Right
== and !=	Left to Right
> < >= <=	Left to Right
+ -	Left to Right
* / %	Left to Right
++ –	Right to Left
(){}[]	Left to Right

3 Types

3.1 Type Declaration

A set of values with similar characteristics are considered as data types. Primitive data types are pre-defined by the language and reserved by a keyword. Derived data types are the aggregation of fundamental data types.

3.2 Primitive Types

The basic data types in Automaton are:

- int - It is a keyword used for the integer data type. It is represented in 32-bits and in signed two's complement form. Its range is from

  $$-2^{31} \; and \; 2^{31} - 1$$

  .
  Ex: int a = 3, b = -2;

- char - It is a keyword used for the character data type. It is represented in 8-bits. It stores all the ASCII characters in a fixed-length field.
  Ex: char ch1 = 'X, ch2 = ' ';

- float - It is a keyword for 64-bit signed which also includes decimal/exponential portion.
  Ex: float f = 1.32;

- double - It is a keyword for 32- bit floating bit number.
  Ex: double d = 1.7 e-3;

- bool - It is a keyword used for the data type Boolean. It represents the correctness of a statement/expression. It has 2(binary) possibilities - TRUE or FALSE.
  Ex: bool var = true;

3.3 Derived Types

The derived data types in Automaton are:

- stack - It is a collection of data types. It follows LIFO(last in first out). It uses push(), pop(), top() to manipulate a given stack.
- push - Used to push a given item to the top of a stack.
- pop - Used to remove the top element of a stack.
- top - Used to return the top element of a stack.
  Ex:
  stack<int> s;
  s.push(2);
  s.top();
  s.pop();
- string - It is a sequence of characters. It is represented in double-quotes.
  Ex: string s = “AbC”;
- vector - It is used to store a collection of same data types.
  Ex: vector<int> v;
  push_back(), pop() , size() is used to add an element, remove an element, return the size of a vector.

4 Different From Other Languages

4.1 Conditional Statements

Conditional statements are those statements where a hypothesis is followed by a conclusion.

- In automaton, if and else statements are not required. A condition is checked and the process is directed towards the corresponding process based on the boolean output.
  Ex:- For an int n,

```
start
{
$->A}

A {
  (n % 2 == 0)->B;
  (n % 2 == 1)->C;
}

B {
  print ("n is an even integer");
  return;
}

C {
  print ("n is an odd integer");
  return;
}
```

- Similarly, while loops are not required in automaton. Iterators or conditions are used to repeat a given state(loop).

4.2 Domain-Specific

We know that C, C++ are generic-purpose languages. But, automaton is basically a domain-specific language. Our language focuses on the logic behind DFA. It also enables us to connect the input strings to it to check its acceptance. Though Domain-specific, we can also code simple generic programs using the fact that with two or more stacks we can design a Turing machine that can accept any particular language.

5 Language Examples

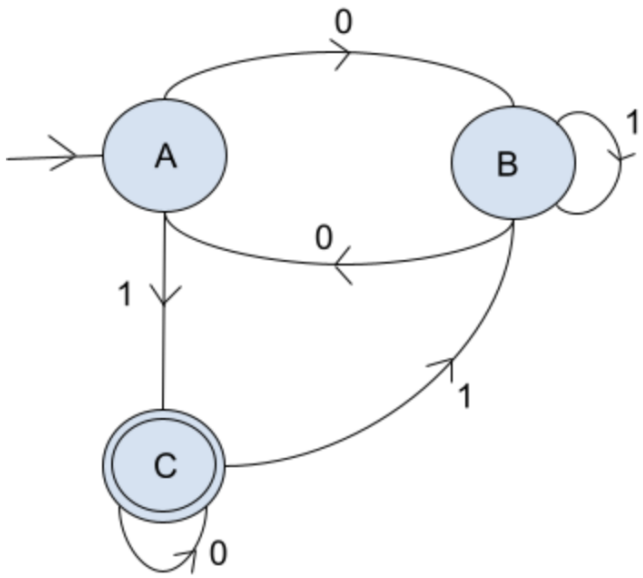
- **Example-1**

```
int DFA main ()
{
  int a, b;
  // Input of two numbers
  a = int (input ());
  b = int (input ());

  // start state
  start
  {
      // default transition : jump to A;
      $->A;
  }
  //  state B
  A
  {
      // if any one of the number is zero goto state B or C
      (a == 0) -> B
      (b == 0) -> C

      // if a id greater than B then a = a- b; and goto state A and vice versa
      (a >= b) && (a = a - b) -> A
      (a < b) && (b = b - a) -> A
  }
  // state B
  B
  {
      // print GCD
      print("The GCD of a & b are : ", b);
      $ -> exit;
  }
  // state C
  C
  {
      // print GCD
      print("The GCD of a & b are : ", a);
      $ -> exit;
  }
  exit
  {
      // return the main function
      return 0;
  }

}

```

- **Example-2**

```
int DFA main(){

    int n;
    n = int(input());       //input for n is taken using input()

    start
    {
        $ ->A;           //A is taken as the starting stage
    }

    A
    {
        (n>0 && n-- && print("Hello World") ) ->A;     //n is decremented and hello world is printed
        (n<=0) ->B;      //checking if required number of times is printed
    }

    B
    {
       return 0;         //program ends when n is equal to 0
    }

}
```

- **Example-3**



The above DFA is coded in Automaton as follows. Any string is taken as input and checked whether the string is accepted by DFA or not.

```
int DFA main(){

    string s = input();             //input string consisting of the alphabets 1, 0 is taken as input

    int i=0, n = s.length();        //length of the string is stored in n

    //intially A is taken as the start state
    start{
        $ -> A;
    }

    A{
       (i==n) -> reject;            //if the end of string is at transition state,the string is considered as rejected
       (s[i]==0 && i++) -> B;
       (s[i]==1 && i++) -> C;
    }

    B{
        (i==n) -> reject;           //if the end of string is at transition state,the string is considered as rejected
        (s[i]==1 && i++) -> B;
        (s[i]==0 && i++) -> A;
    }

    C{
        (i==n) -> accept;           //if the end of string is at end state, then the string is considered as accepted
        (s[i]==1 && i++) -> B;
        (s[i]==0 && i++) -> C;
    }

    //the string is accepted
    accept{
        print("Given string is accepted by the DFA");
        return 0;
    }

    //the string is rejected
    reject{
        print("Given string is not accepted by the DFA");
        return 0;
    }
}
```