A Z-Notation Formalization of Deterministic Finite Automata

[1]Amjad Farooq,[2]M. Junaid Arshad, [3]Muhammad Abuzar Fahiem and [4]Ayesha Jalal
[1,2]University of Engineering and Technology, Lahore-Pakistan
[3,4]Lahore College for Women University, Lahore-Pakistan

***Abstract*— A deterministic finite automaton (DFA) helps to design a system in a structured way and may be used to demonstrate visual flows between system functionalities. Formal methods (FM) are a mathematical formalization mechanism for the mature structuring of software / hardware products. FM based analysis of software / hardware design yields a reliable and robust system. In this paper, we provide FM for DAFs for the system development based on Z-Notation formalization. Z-Notation is widely used to describe the system properties and operations. This formalization may provide a guaranty to an efficient and accurate system design.**

***Keywords*—** Deterministic Finite Automaton, Formal Methods, Z-Notation, System Design, Formal Specifications

I. INTRODUCTION

The functionalities of any system can be captured in terms of operations and constraints. A deterministic finite automaton (DFA) helps to design a system in a structured way and may be used to demonstrate visual flows between system functionalities. DFAs have various applications in the field of computer science and engineering such as modeling of control systems, construction of a compiler, optimization programs, verification of protocols, human computer interaction and pattern matching.

Formal methods (FM) are a mathematical formalization mechanism for the mature structuring of software / hardware products. FM based analysis of software / hardware design yields a reliable and robust system. However, the high cost of using FM means that these are usually suitable for the development of high integrity safety critical secure systems.

In this paper, we provide FM for DAFs for the development of a system based on Z-Notation formalization. Z-Notation is widely used to describe the system properties and operations. This formalization may provide a guaranty to an efficient and accurate system design.

In section 2, we describe the related work while the proposed work is presented in section 3. Section 4 gives the summary and the future extensions to our work are suggested in section 5.

II. RELATED WORK

A DFA is a 5-tuple state machine consisting of a finite set of states (s), a finite set of inputs (Σ), a transition function ($T : S \times \Sigma \rightarrow S$), a start state (S), and a set of accept states (A). DFAs are abstract models of machines which are represented by diagrams that are based on mathematical notations and techniques. These diagrams are used to demonstrate outputs from computations on input based on a transition function. DFAs are particularly useful for sequential logic and control functions and are the most practical computational models. Due to versatility in its methodologies, DFAs are difficult to adopt for a system.

FM are practical and precise way of solving problems by specifying a system through its properties and associated semantics. The system properties may include functional behavior, performance characteristics, or internal structure. FM can be applied at various stages (verification, specification, development and automated proofs) of a development process. FM provide means to symbolically examine the entire state of a digital design and establish correctness at all possible inputs. FM can be applied at different levels; at high level designs where most of the details are in abstract form, at only the most critical components, and at only code verification and testing levels. FM provide a well defined syntax, semantics and specifications of a system in terms of mathematical notations derived from set theory, discrete structures and graph representations [1]. FM have been proved to be useful to ensure the correctness of a system at different stages [2], [3]. The validation and verification techniques offered by FM may be applied at each phase of the development process [4]. Traditional development approaches lack the ability as whereas prove of specifications is concerned as the errors may be concealed under graphical interfaces / specifications [5], and can be usually identified during implementation and testing phases. Implementation errors are difficult and costly to fix. On the other hand, the mathematical nature of specifications, as in FM, enables to carry out proofs. Various studies have suggested that the use of FM has a great potential to improve the clarity and precision of a system development process as well as in finding critical errors and thus making the system more reliable [6]. FM have various fields of application such as nuclear controls systems, mobile communication systems and, portable video games. In the past, various FM based efforts have been made for system development. Earlier, a distributed real-time information system to support air traffic controllers was developed [7]. Another effort was to develop a safety critical radiation therapy machine [8]. FM based model checking for smart cards [9], verification and validation approaches [10], and multi-agent systems were also introduced [11]. FM may be divided into two main categories; model oriented and property oriented [12].

Z-Notation is a model oriented approach derived built on the fundamentals of set theory and first order predicate logic. In Z-Notation, system states are described in terms of mathematical structures (sets, relations, and functions). Z-Notation consists of linear notations, definitions and declarations, logical and relational operators and functions, orders and equivalences, sequences, and schema notations for expressing states and operations. Z-Notation is a good match to procedural as well as object-oriented paradigms.

III. PROPOSED WORK

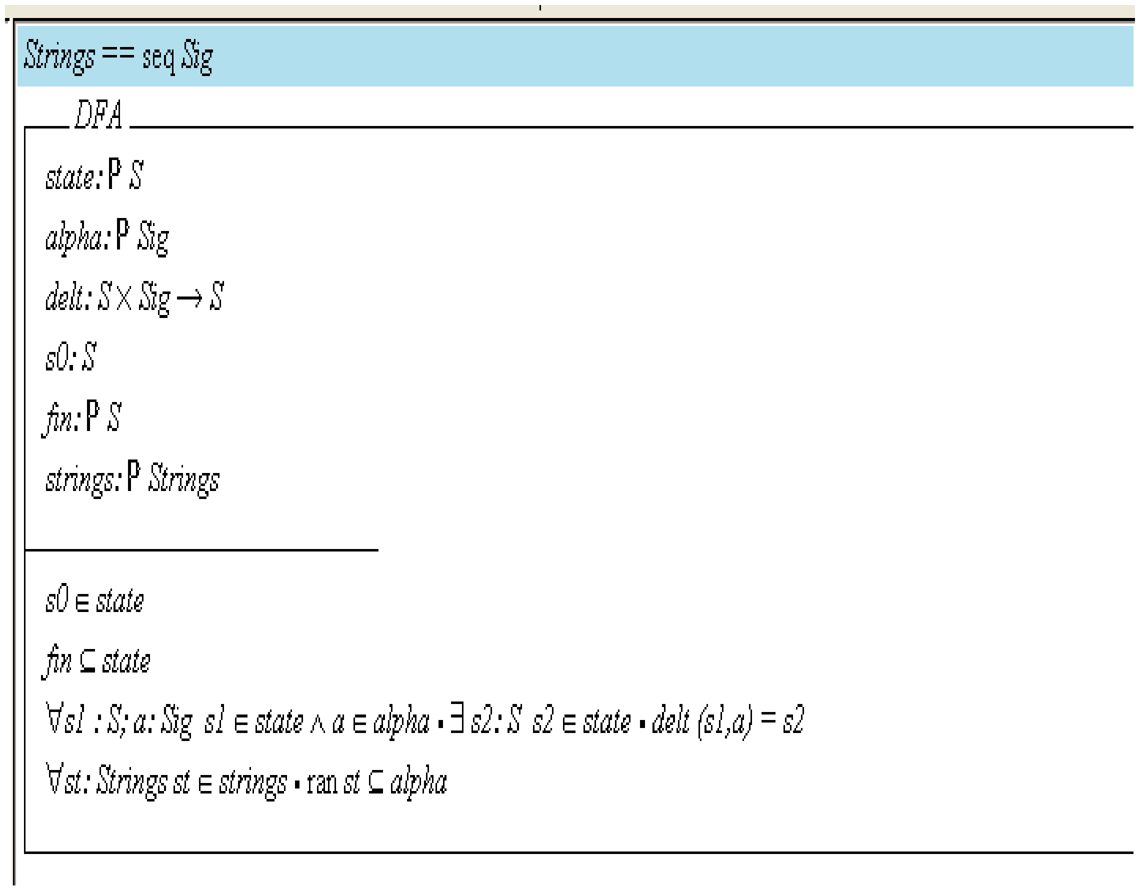
In this section, we will first give formal specifications of a DFA and then a string acceptor followed by a language acceptor, in terms of Z-Notation.

We select a DFA with transition function T $(s_1, a) = s_2$ for states (S) s_1, s_2 and input (Σ) a. For Z-notation formalization of the DFA, S and Σ are symbolized as 'S' and 'Sig' respectively [S, Sig]. T is a function for each input (s_1, a), where s_1 is a state and a is an input that generates a unique output s_2 of type S. Now, we can defined T as: delt: $S \times Sig \rightarrow S$. A variable "state" is defined for a set of states of DFA. Since a given state is of type S, so state must be a type of power set of S. Variable 'alpha' is declared for a set of inputs which is type of power set of 'Sig'.

The initial state s_0 is of type S and the set of final states is defined as "fin" and is a type of power set of S. This constraints relationship will be used by the composition of these objects, for which a schema structure is used. The name of the schema structure is defined as "DFA". Figure 1 shows the formal specifications of "DFA" using Z-Notation.

A string decider is designed in which the DFA takes inputs which generate a Boolean output showing whether the given string is accepted or not. It takes a string $v = v_1v_2. . . v_n$ for each input v_i where i = 1, 2,…, n. The DFA accepts the string t if we reach at the final state using the transition function. The steps followed by the DFA to accept the string are; the DFA starts from 'q_1' which is a start state, it goes from state to state according to the transition function T, and it accepts the string t if machine ends up at a final state. The first part of the schema shows the inputs while the constraints are shown in the second part.

In formal specifications of string decider two inputs, DFA and a string v? is taken as input. Schema name of the string decider is represented as "AcceptString". Figure 2 shows the formal specifications of "AcceptString" using Z-Notation.
For formal specifications of language accepter in Z- notation, language and DFA are taken as inputs and if the language is accepted then it returns the true value. The name of the schema structure is defined as "AcceptLang". Figure 3 shows the formal specifications of "AcceptLang" using Z-Notation.



```
Strings == seq Sig

DFA
state: ℙ S
alpha: ℙ Sig
delt: S × Sig → S
s0: S
fin: ℙ S
strings: ℙ Strings
-------------------------
s0 ∈ state
fin ⊆ state
∀s1 : S; a: Sig s1 ∈ state ∧ a ∈ alpha • ∃ s2: S s2 ∈ state • delt (s1,a) = s2
∀st: Strings st ∈ strings • ran st ⊆ alpha
```

Fig. 1: Formal Specifications of "DFA"

AcceptString

ΞDFA

$v? : Strings$

$v? \in strings$

$\exists q : \mathrm{seq}\, S\ \mathrm{ran}\, q \subseteq state \wedge \#q = \#v? + 1 \bullet \#q \geqslant 1 \Rightarrow q1 = q0 \wedge q(\#q) \in fin \wedge (\forall j : \mathbb{N}\ \#v? \geqslant j+1$

$\bullet (j \in 1 .. \#q\text{-}1 \Rightarrow ((qj, v?(j+1)), q(j+1)) \in delt))$

Fig. 2: Formal Specifications of “AcceptString”

AcceptLang

ΞDFA

$lang? : \mathbb{P}\ Strings$

$\forall v : Strings\ v \in lang? \bullet v \in strings$

$\forall v : Strings\ v \in lang? \bullet \exists q : \mathrm{seq}\, S\ \mathrm{ran}\, q \subseteq state \wedge \#q = \#v + 1 \bullet \#q \geqslant 1 \Rightarrow q1 = q0$

$\wedge q(\#q) \in fin \wedge (\forall j : \mathbb{N}\ \#v \geqslant j+1 \bullet (j \in 1 .. \#q\text{-}1 \Rightarrow ((qj, v(j+1)), q(j+1)) \in delt))$

Fig. 3: Formal Specifications of “AcceptLang”

IV. CONCLUSION AND FUTURE WORK

In this research work, we have proposed formal specification of DFAs based on FM using Z-Notation. We have formalized the DFA itself, string acceptance and language acceptance. This research may prove to be useful in various industrial applications. In future, our proposed work can be extended to formalization of union, intersection and closure operations of DFAs using Z-Notations. Moreover, Z-Notations can be applied to pushdown automaton, linear bounded automaton and deterministic Turing machine, as well.

REFERENCES

[1] Ciapessoni, E., Mirandola, P., Coen-Porisini, A., Mandrioli, D., Morzenti, A.; From Formal Models to Formally Based Methods: An Industrial Experience, ACM Transactions on Software Engineering and Methodology, vol. 8(1), (1999), 79-113.

[2] Antoniou, P., Holub, J., Illiopoulos, C. S., Melichar, B., Peterlongo, P.; Finding Common Motifs with Gaps Using Finite Automata, Lecture Notes in Computer Science, vol. 4094, (2006), 69-77.

[3] Heitmeyer, C.; On the Need for Practical Formal Methods, Lecture Notes in Computer Science, vol. 1486, (1998), 18-26.

[4] Bowen, J. P., Hinchey, M. G.; Ten commandments of Formal Methods, IEEE Computer , vol. 28(4), (2006), 56-63.

[5] Bowen, J. P., Hinchey, M. G.; The Use of Industrial-Strength Formal Methods, Proceedings of the Twenty-First Annual International Conference on Computer Software and Applications Conference, (1997).

[6] Easterbrook, S., Lutz, R., Covington, R., Kelly, J., Ampo, Y., Hamilton, D.; Experiences Using Lightweight Formal Methods for Requirements Modeling, IEEE Transactions on Software Engineering, vol. 24(1), 4-14, (1998).

[7] Hall, A.; Using Formal Methods to Develop an ATC Information System, IEEE Software, vol. 13(2), 66-76, (1996).

[8] Jacky, J.; Specifying a Safety-Critical Control System in Z, IEEE Transactions on Software Engineering, vol. 21(2), 99-106, (1995).

[9] Lanet, J. L., Lartigue, P.; The Use of Formal Methods for Smart Cards, a Comparison between B and SDL to Model the T=1 Protocol, Proceedings of International Workshop on Comparing Systems Specification Techniques, (1998).

[10] Frey, G., Litz, L.; Formal Methods in PLC Programming, Proceedings of IEEE International Conference on Systems, Man, and Cybernetics, (2000).

[11] Hilaire, V., Koukam, A., Gruer, P., Muller, J. P.; Formal Specification and Prototyping of Multi-agent Systems, Lecture Notes in Computer Science, vol. 1972, 114-127, (2000).

[12] Wing, J. M.; A Specifier's Introduction to Formal Methods, IEEE Computer , vol. 23(9), 8, 10-22, 24, (1990).