A Refinement: Integration of Algebraic Automata and Z Conforming Some Important Structures

Nazir Ahmad Zafar, Ajmal Hussain and Amir Ali

Abstract*— Automata theory has proved to be a cornerstone in theoretical computer science since last couple of decades. It is playing an important role in modeling behavior of complex systems. The algebraic automaton which is an advanced form of automata, having properties and structures from algebraic theory, is emerging with several modern applications. Optimization of logic based programs, design and development of model checkers are couple of examples of it. Design of a complex system not only requires the functionality but it also needs to capture its control behavior. Z notation is an ideal specification language used for describing state space of a system and operations over it. Consequently, an integration of algebraic automata and Z will be an effective tool for modeling purposes. In this paper, we have established a relationship between few fundamentals of algebraic automata and Z notation. At first, some important concepts of automata are transformed to Z notation. Then, we have given a formal construction of algebraic automata. Next, fundamental concepts of algebraic automata, for example, monoid, semi-group and group are formalized and refined. Finally, an important notion of homomorphism for verifying similarity between algebraic structures is described. Formal specification of this linkage is analyzed and validated using Z/EVES tool.

***Index Terms*—Integration of approaches, Algebraic automata, Formal methods, Z notation, Validation.**

I. Introduction

As computers are being used almost in every machine or in electrical equipment, that is, machines are being controlled by computer based systems. And, of course, computers are controlled by software systems. When software is used in controlling a complex system, for example, safety critical system its failure may cause a big loss in terms of wealth, deaths, injuries and environmental damages. Consequently, constructing correct software is as important as its other counterparts, for example, hardware or electro-mechanical systems [1]. Formal methods are mathematical based techniques used for specification of properties of software and hardware systems for insuring correctness of a system [2]. By applying formal methods, we can describe a mathematical model of a system and then it can be analyzed and validated increasing confidence over a system [3].

At the current stage of development in software engineering approaches, it is not possible to develop a system using a single technique and as a result various techniques have to be integrated for different purposes at different levels of development of software. This is the reason that today integration of approaches for software engineering, more broadly in computing systems, has become a well-researched area. Further, an integration of approaches is an open research area to bridge the gap between different methodologies in computer science and engineering leading to development of automated computer tools and techniques.

Design of a complex system, not only requires functionality but it also needs to model its control behavior. There are a large variety of techniques for software specification which are suitable for specific aspects in the process of the software development. For example, Z notation, Vienna Development Methods, B Method and algebraic techniques are usually used for defining the data type while Petri nets, process algebras, automata and statecharts are some of the examples which are best suited for capturing dynamic aspects of a system [4]. All of the above examples have a well-defined mathematics based syntax and semantics. Therefore it is required to identify relationship between static and dynamic modeling techniques for development of a complete and consistent system. Further, this integration will reduce the complexity of the systems.

Automata theory has proved to be a cornerstone of theoretical computer science since last couple of decades. Modeling of finite state systems, defining a regular set of finite words, compiler construction and control behavior are few traditional applications of automata. The algebraic automaton which is an advanced form of automata, having properties and structures from algebraic theory, is emerged with several modern applications. For example, optimization of logic based programs, specification and verification of protocols, design and development of model checkers, and human computer interaction are some applications areas of it. The applications of algebraic theory are not limited to computer but are being seen in other disciplines, for example, modeling physical phenomena in chemistry and biology.

In this paper, a refinement of a relationship between formal methods and algebraic automata identified in [5] is done, and some inconsistencies are removed. There are several applications of algebraic theory, particularly, in defining static part of a system. For example, modeling behavior of distributed systems, the objects are usually concatenated and hence the associative property is satisfied there at. After adding an identity element, the structure produced is called a monoid which is an abstract algebraic data type. In modeling computerized systems using algebraic structures, we must be able to represent it by some data types. Representation of

Manuscript received November 12, 2008. This work was supported by Centre for Research in Computer Science, University of Central Punjab (UCP), Lahore, PAKISTAN.

Nazir A. Zafar is with the Faculty of Information Technology, UCP, Lahore, Pakistan (telephone: +92-42-5755314; fax: +92-42-5710881; email: dr.zafar@ucp.edu.pk). Dr Zafar is on leave from Pakistan Institute of Engineering and Applied Sciences, Islamabad, PAKISTAN (telephone: +92-51-9290273-4; fax: +92-51-2208070; email: nazafar@pieas.edu.pk).

A. Hussain is Associated Dean of the Faculty of Information Technology, UCP, Lahore, Pakistan (telephone: +92-42-5755314; fax: +92-42-5710881; email: ajmal@ucp.edu.pk).

A. Ali is a student of MS in Faculty of Information Technology, UCP, (e-mail: amiralishahid@ucp.edu.pk).

static part of a system is important because time complexity of an algorithm depends on it. The major objectives of this paper are to (i) propose an integration of automata and formal methods enhancing modeling power, (ii) provide a syntactic and semantic well-defined relationship between Z notation and algebraic automata.

In section 2, related work is discussed. In section 3, an introduction to formal methods is given. In section 4, an overview of automata is provided. Formal construction of algebraic automata is described in section 5. Finally, conclusion and future are discussed in section 6.

II. Related Work

Although integration of approaches for software development is a well-researched area [6], [7], [8], [9], [10], [11], [12] but there does not exist much work on formalization of graphical based notations. The work [13], [14] of Dong et al., in which they have linked Object Z and timed automata for some aspects of these approaches, is close to ours. Their work is assumed as a starting point for this research. Another piece of good work is reported in [15], [16] in which R. L. Constable has proposed a constructive formalization of some important concepts of automata using Nuprl. In [17], a combination of Z notation with statecharts is established. A relationship is investigated between Petri-nets and Z notation in [18], [19]. An integration of B method and UML is presented in [20], [21]. Wechler, W. has introduced some important algebraic structures in fuzzy automata [22]. In [23], a treatment of fuzzy automata and fuzzy language theory is discussed when the set of possible values is a closed interval [0, 1]. Ito, M., has described automata and formal languages from the algebraic point of view. Firstly, he investigated the algebraic structure of automata and then treated a kind of global theory [24]. Kaynar, D. K at al. has proposed a modeling framework which is a basic set of mathematical models to support description and analysis of real timed computing systems [25]. Godsil, C. at al. [26] has given some ideas of algebraic graphs with an emphasis on current rather than classical theory of graphs. Their work is interesting because of usefulness of graph-based notations in modeling of various problems of computer science and engineering.

Most of the researchers listed above have either taken some examples in proposing integration of approaches or have addressed only some aspects of these approaches. Further, there is a lack of formal analysis which can be supported by computer tools. Our work is different from others because we have given a generic approach to link Z notation and algebraic automata. Further, a computer tool support is provided for analysis and validation of this relationship.

III. An Introduction to Formal Methods

Formal methods are mathematical approaches used for describing and analyzing properties of software and hardware systems [27]. That is, descriptions of a system are written using symbols and notations which are mathematical expressions rather than informal explanations. These notations are based on discrete mathematics such as logic, set theory and graph theory. Formal methods may be classified in several ways. One frequently-made distinction is between property oriented and model oriented methods [28]. Property oriented methods are used to describe software in terms of properties or constraints that must be satisfied on it. Model oriented methods are used to construct a model of a system's behavior [29]. For example, state transition diagrams are used to model the behavior of a system as a set of states and then transitions are defined between these states.

Formal methods are used to improve quality of software systems by means of documenting and specifying in a precise and structured manner. Z notation is one of the most popular specification languages in formal method. The Z [30] is a model oriented approach, which is based on set theory and first order predicate logic. It is also used for specifying the behavior of systems as an abstract data types. Sequential programs can also be modeled using it. The Z notation is used in this work for specification and validation because it describes a state space of a system and a set of operations that may be performed on it [28].

IV. Algebraic Automata

Automata theory has become a basis in the theoretical computer science since last couple of decades because of its various applications and having a vital role in science and engineering [31]. Modeling control behavior, modeling of finite state systems, compiler constructions, defining a regular set of finite words are some of the traditional applications of automata [32], [33], [34]. We can classify automata because of its deterministic and nondeterministic nature. Both types of automata have their own pros and cons in modeling and specification of systems but are equal in power. That means if a language is recognized by one, it can also recognized by the other. Nondeterministic finite automata (NFA) are sometimes useful because constructing an NFA is much easier than constructing deterministic finite automata (DFA) for a particular problem without going into details. Further, complexity of system is reduced and many important properties can be established easily using NFA. On the other hand, DFA is much easier to implement as compared to its sibling NFA. Consequently, both of the automata have their merits and demerits and any one can be used based on the requirements and nature of a problem.

Finite automata are abstract models of machines based on mathematical notations which can be represented using diagrams as well. These models can be used to perform computations on input and an output can be generated if required, by moving through a sequence of configurations. If we are able to reach any of the accepting configuration of a finite automata by using a series of computation then the given input is accepted. Of course, there must be a guide called the transition function which computes the next state based on the current state and an alphabet, at the every step of its computation. The starting point is called an initial state. A set of alphabets is required as one of the inputs to transition function to move from one state to another of a DFA.

The algebraic automaton is an advanced form of automata having properties and structures from algebraic theory of mathematics. Algebraic automata have emerged with several modern applications. Optimization of logic based programs, specification and verification of protocols, design and

development of model checkers, and human computer interaction are few examples of it. The applications of algebraic theory are not limited to computers but are also being seen in other disciplines of science. Representation of characteristics of physical phenomena in chemistry and biology is one of the important examples of it.

On the other hand, diagrams in algebraic automata have been difficult to use except very trivial cases, which is one of the issues in representation of automata diagrammatically. Further, automata may have different implementations and consequently its time and space complexity must also be different, which is another issue in modeling using algebraic automata similar to other automata. Because of such weaknesses and limitations of modeling using automata, it is argued that this single approach cannot be used for modeling of a complete system and consequently its integration is required with other useful approaches. Based on the reasoning given in the first section, a linkage between Z and algebraic automata will be useful in modeling using integrated approaches. If we are able to formalize and map this relationship, then it would be useful tool not only at academic but at an industrial level as well. A formal verified linkage of algebraic automata and Z is given in the next section.

V. Integration of Algebraic Automata and Z

In this section, an integration of some important concepts of algebraic automata and Z notation is given. It is to be mentioned that the definitions used are based on a well known book with title "Algebraic Theory of Automata and Languages" [24]. The set of structures used to give the formal relationship, as discussed above, between Z and algebraic automata is: (i) automaton, (ii) extended automaton, (iii) homomorphism and their extended forms, (iv) monoid endomorphisms, and (v) group automorphism.

A. Design of Algebraic Automaton

We start with the definition of algebraic automata which is a 3-tuple (Q, ∑, δ), where (i) Q is a finite nonempty set of states, (ii) ∑ is a finite set of alphabets and (iii) δ is a transition function which takes a state and an alphabet as input and produces a new or the same state as an output. The above tuple is a deterministic algebraic automata (DAA) if for each state q1 and for every alphabet a, there is a unique state q2 such that: δ(q1, a) = q2. To formalize DAA in Z, Q and ∑ are denoted by S and X respectively. [S, X]

In modeling systems using sets in Z notation, we do not impose any restriction upon the number of elements of a set and a high level of abstraction is supposed there. Further, we do not insist upon any procedure for deciding whether an element is a member of the given collection. As a consequent, our S and X, defined above, are sets over which we cannot define any operation. For example, cardinality to know the number of elements of a set cannot be defined. Similarly, union, intersection, complement, subset and cartesian product operations over these sets are not defined as well.

To describe a set of states for the above DAA, a variable *states* is introduced. Since, a given state q is of type S therefore *states* is a type of power set of S. Similarly, for alphabets the variable *alphabets* is of type of power set of X. As we know that δ relation is a function because for each input (q1, a), where q1 is a state and a belongs to set of *alphabets* there must be a unique output state q2, which is image of (q1, a) under the transition function δ (*delta*). Consequently, we can declare transition function δ as, $delta: S \times X \rightarrow S$.

For a moment, we have used mathematical language of Z notation which is used to describe various objects of a system. It is to be mentioned here that the same language can be used to define the relationships between these objects. This relationship will be used in terms of constraints over a schema after composing the objects. The schema structure is used for composition because it is very powerful at an abstract level of specification and it helps in describing a well organized specification approach. All of the above three components, *states*, *alphabets* and *delta* of DAA are encapsulated and put in the schema named as *Automaton*. The formal description of *Automaton* is given in Z notation as below.

Automaton

$$states: \mathbb{F}\ S$$
$$alphabets: \mathbb{F}\ X$$
$$delta: S \times X \rightarrow S$$

$$states \neq \{\}$$
$$\forall s: S; x: X \mid s \in states \wedge x \in alphabets \bullet \exists t: S \mid t \in states \bullet delta\ (s, x) = t$$

Invariants: (i) The set of states is finite and non-empty. (ii) For each input (s, x) for the transition function *delta* where s is an element of *states* and x is a member of *alphabets*, there is a unique state t such that: $delta\ (s, x) = t$.

After formalizing DAA, its extended form is described. In the extended form, three new components are added and two components of the algebraic automaton, defined above, are reused. In the schema as given below, the variables *states* and *alphabets* are reused and have the same meaning. The *delta* function defined above is refined. In the extended form, the *delta* function takes a states and a string as inputs and produces the same state or new state as output. Since we need to compute the set of all the strings which can be generated from the set of alphabets and hence a fourth variable is used and denoted by *strings* which is of type of power set of set of all the sequences. As a sequence can be empty and hence a fifth variable is used for it.

AutomatonExt

$$Automaton$$
$$strings: \mathbb{F}\ (\text{seq}\ X)$$
$$deltae: S \times \text{seq}\ X \rightarrow S$$
$$epsilon: \text{seq}\ X$$

$$epsilon \in strings$$
$$\forall s: S \mid s \in states \bullet deltae\ (s, epsilon) = s$$
$$\forall s: S; a: X; u: \text{seq}\ X \mid s \in states \wedge a \in alphabets \wedge u \in strings \bullet deltae\ (s, (\langle a \rangle \frown u)) = deltae\ ((deltae\ (s, \langle a \rangle)), u)$$

Invariants: (i) The null string *epsilon* is an element of *strings*. (ii) If the transition function takes a state and the null string *epsilon* as input then it produces the same state of DAA. (iii) For each input $(s, (\langle a \rangle \frown u))$, where s is an element of states, a is an element of alphabets and u is an element of *strings*, the *delta* function is defined as: $delta\ (s, (\langle a \rangle \frown u)) = delta\ ((delta\ (s, \langle a \rangle)), u)$.

B. Homomorphism and Its Extended Forms

Formal specification of homomorphism, isomorphism, endomorphism and automorphism over algebraic automata is described in this section. All of these morphisms are important in algebraic theory and are well-defined on some fundamental structures of this theory. In abstract algebra, a homomorphism is a structure preserving a mapping between two given algebraic structures. Monoid, semi-groups, groups, semi-rings, rings and vector spaces are some of the examples of algebraic structures where the concept of homomorphism and its variants can be defined. This concept was extended in [34] and was defined on algebraic automata. In fact, the word homomorphism is meant by "same shape" and is an interesting one because the similarity of the structures can be verified and tested by defining over the given structures. Formal description of homomorphism and its variants over the algebraic automaton is given below.

Let AA1 = (Q1, ∑1, δ1) and AA2 = (Q2, ∑2, δ2) be two given algebraic automata. Let ρ be a mapping from Q1 into Q2. If ρ (δ1(s, x)) = δ2(ρ(s), x) holds for any s ∈ Q1 and any x ∈ ∑1, then ρ is called a homomorphism of set Q1 into Q2. The formal specification of AA1 and AA2 is described by reusing the automaton defined above and are represented by schemas *AutomatonA* and *AutomatonB* respectively.

The automaton *AutomatonA* is created by replacing the variables *states*, *alphabets* and *delta* of the automaton *Automaton* with the new variables *states1*, *alphabets1* and *delta1* respectively. The definitions of these components and variants over it are same as defined in *Automaton*.

AutomatonA ≅
Automaton[*states1*/*states*, *alphabets1*/*alphabets*, *delta1*/*delta*]

Similar to the definition AA1, the formal description of AA2 is given in terms of a schema which is represented by *AutomatonB*. It is to be noted that the invariants over *AutomatonA* and *AutomatonB* hold as in *Automaton* because these are created not defined.

AutomatonB ≅
Automaton[*states2*/*states*, *alphabets2*/*alphabets*, *delta2*/*delta*]

After defining AA1 and AA2, we can give a formal definition of homomorphism from AA1 into AA2. The homomorphism is represented by a schema *Homomorphism* as given below. It consists of three components *AutomatonA*, *AutomatonB* and a variable *row*. The variable *row* is a mapping from *S* to itself used to represent the homomorphism. The invariants which must hold on homomorphism are defined in terms of predicates in second part of the schema.

Homomorphism

AutomatonA
AutomatonB
row: $S \rightarrow S$

$\forall s: S; x: X \mid s \in states1 \wedge x \in alphabets1$
• *row* (*delta1* (*s*, *x*)) = *delta2* ((*row s*), *x*)

Invariant: (i) For every s in the set of states and x in the set of alphabets of the first automata *AutomatonA*, if the mapping *row* satisfies the condition: *row* (*delta1* (*s*, *x*)) = *delta2* ((*row s*), *x*). then it conforms a homomorphism from the first automata *AutomatonA* into *AutomatonB*.

If AA1 = AA2 in the homomorphism defined above, then it is called an endomorphism. To give a formal description of the endomorphism, the mapping *row* is defined from the set of states *S* to itself. It can be observed that endomorphism is a variant more precisely a special case after reduction of homomorphism. That is why we have induced it from the definition of homomorphism given above.

Endomorphism

Automaton
row: $S \rightarrow S$

$\forall s: S; x: X \mid s \in states \wedge x \in alphabets$
• *row* (*delta* (*s*, *x*)) = *delta* ((*row s*), *x*)

Invariant: (i) For every state s in the set of states and x in the set of alphabets of the given automata *Automaton*, if the mapping *row* satisfies the condition: *row* (*delta1* (*s*, *x*)) = *delta2* ((*row s*), *x*), then it conforms an endomorphism from automata *Automaton* into itself.

If X and Y are two nonempty sets then a mapping η from set X to set Y is called one to one if different elements of X have different images in Y. Mathematically, ∀ x1, x2 ∈ X; y ∈ Y • η(x1) = y and η(x2) = y ⇒ x1 = x2. The mapping η is called onto if each element of Y is an image of some element of X that is range of η = Y. If a mapping is a one to one (injective) as well as onto (surjective) then it is called a bijective mapping. Coming back to our definition of homomorphism, if it is a bijective mapping from algebraic automata AA1 to AA2 then it is called an isomorphism and the automata are isomorphic. Now we give a formalization of isomorphism from *AutomatonA* to *AutomatonB* using a schema given below. For this purpose, we simply define constraints of bijection over the homomorphism which results an isomorphism.

Isomorphism

Homomorphism

$\forall s1, s2, s: S \mid s1 \in states1 \wedge s2 \in states1 \wedge s \in states2$
• $(s1, s) \in row \wedge (s2, s) \in row \Rightarrow s1 = s2$
ran *row* = *states2*

Invariants: (i) For all s1 and s2 in states of automata *AutomatonA* and s in states of *AutomatonB*, if the images of s1 and s2 in the second automata are same under the mapping *row* then s1 and s2 must be same. (ii) Each element of the set of states of automata *AutomatonB* is an image of some element of automata *AutomatonA* under the mapping *row*.

In the formal definition of isomorphism given above, if the algebraic automata *AutomatonA* and *AutomatonB* are equal, that is, their corresponding components are same and constraints are also applied then such an isomorphism is called an automorphism. A formal description of an automorphism is given below after defining some invariants over the endomorphism defined above.

Automorphism

Ξ*Endomorphism*

$\forall s1, s2, s: S \mid s1 \in states \wedge s2 \in states \wedge s \in states$
• $(s1, s) \in row \wedge (s2, s) \in row \Rightarrow s1 = s2$
ran *row* = *states*

Invariants: (i) For all s1 and s2 in set of states and x in the set of alphabets of some automata, if the images of s1 and s2 are same under the mapping *row* then the elements s1 and s2 are same. (ii) Each element of the set of states of the given automata is an image of some element of the set of states of the same automata under the mapping *row*..

C. Formalizing Endomorphisms over Monoid

Let A be a given non-empty set then the algebraic structure (A, *) is a monoid if the following conditions are satisfied.

i. $\forall$ x, y $\in$ A, the element x*y $\in$ A. The symbol * is a binary operation over A.

ii. $\forall$ x, y, z $\in$ A, (x*y)*z = x*(y*z), that is associative property is satisfied.

iii. $\forall$ x $\in$ A, there exists an element e $\in$ A, such that x*e = e * x = x. The element e is called an identity of A.

Let us suppose that E (A) = set of all the endomorphisms over the algebraic automata A. It is already proved in [34] that E (A) forms a monoid under the binary operation defined in section 5.2. Here we describe the formal specification of it by defining a schema *MonoidEndomorphisms*. To formalize this structure, two variables are assumed. The first one is a set of all endomorphism which is of type of power set of *Endomorphism* and is denoted by *endomorphisms*. The second one is a binary operation denoted by *boperation*. It takes two endomorphisms as input and produces a new endomorphism as an output. The components of *MonoidEndomorphisms* are defined in first part and invariants are defined in the second part of it.

MonoidEndomorphisms

endomorphisms: $\mathbb{F}$ *Endomorphism*
boperation: *Endomorphism* $\times$ *Endomorphism* $\rightarrow$ *Endomorphism*

$\forall$ *endo1, endo2*: *Endomorphism* | *endo1* $\in$ *endomorphisms* $\wedge$ *endo2* $\in$ *endomorphisms*
- $\exists$ *endo3*: *Endomorphism* | *endo3* $\in$ *endomorphisms*
  - *boperation* (*endo1*, *endo2*) = *endo3*

$\forall$ *endo1, endo2, endo3*: *Endomorphism* | *endo1* $\in$ *endomorphisms* $\wedge$ *endo2* $\in$ *endomorphisms* $\wedge$ *endo3* $\in$ *endomorphisms*
- *boperation* ((*boperation* (*endo1*, *endo2*)), *endo3*) = *boperation* (*endo1*, (*boperation* (*endo3*, *endo3*)))

$\forall$ *endo*: *Endomorphism* | *endo* $\in$ *endomorphisms*
- $\exists$ *endoe*: *Endomorphism* | *endoe* $\in$ *endomorphisms*
  - *boperation* (*endo*, *endoe*) = *endo* $\wedge$ *boperation* (*endoe*, *endo*) = *endo*

Invariants: (i) For any two endomorphisms *endo1* and *endo2* there exists an endomorphism *endo3* such that: *boperation* (*endo1*, *endo2*) = *endo3*. This property defines the binary operation over the monoid structure *MonoidEndomorphisms*. (ii) For any three endomorphisms *endo1*, *endo2* and *endo3*, the binary operation satisfies the conditions: *boperation* ((*boperation* (*endo1*, *endo2*)), *endo3*) = *boperation* (*endo1*, (*boperation* (*endo3*, *endo3*))). This is the associative property defined over the set of all the endomorphisms. (iii) For any endomorphisms *endo*, there exists an endomorphism *endoe* such that: *boperation* (*endo*, *endoe*) = *endo* $\wedge$ *boperation* (*endoe*, *endo*) = *endo*. This property ensures the unique existence of left and right identity of E (A).

D. Formalizing Automorphisms over Group

The algebraic structure (A, *) is a group if it satisfies the following: (i) A is a monoid, (ii) for each element x in the set A, there exists a unique element y in A such that x*y = y*x = e, that is, the inverse of each element of A exists and is unique.

Now let us suppose that A (A) is a set of all the automorphisms over the algebraic automata A. It is also proved in [34] that A (A) forms a group. Here we describe its formal specification by a schema *GroupAutomorphisms* as was done for E(A) in Section C. To formalize this structure, three variables are assumed. The first one is a set of all the automorphisms which is of type of power set of *Automorphism* and is denoted by *automorphisms*. The second one is an identity element under the same binary operation. And the last one is the binary operation itself denoted by *boperation*. It takes two automorphisms as input and produces the same or a new automorphism as an output. The components of the schema *GroupAutomorphisms* are given in first part and its invariants are defined in the second part of the schema *GroupAutomorphisms*.

GroupAutomorphisms

automorphisms: $\mathbb{F}$ *Automorphism*
autoe: *Automorphism*
boperation: *Automorphism* $\times$ *Automorphism* $\rightarrow$ *Automorphism*

$\forall$ *auto1, auto2*: *Automorphism* | *auto1* $\in$ *automorphisms* $\wedge$ *auto2* $\in$ *automorphisms*
- $\exists$ *auto3*: *Automorphism* | *auto3* $\in$ *automorphisms*
  - *boperation* (*auto1*, *auto2*) = *auto3*

$\forall$ *auto1, auto2, auto3*: *Automorphism* | *auto1* $\in$ *automorphisms* $\wedge$ *auto2* $\in$ *automorphisms* $\wedge$ *auto3* $\in$ *automorphisms*
- *boperation* ((*boperation* (*auto1*, *auto2*)), *auto3*) = *boperation* (*auto1*, (*boperation* (*auto3*, *auto3*)))

$\forall$ *auto*: *Automorphism* | *auto* $\in$ *automorphisms*
- *boperation* (*auto*, *autoe*) = *auto* $\wedge$ *boperation* (*autoe*, *auto*) = *auto*

$\forall$ *auto*: *Automorphism* | *auto* $\in$ *automorphisms*
- $\exists$ *autoi*: *Automorphism* | *autoi* $\in$ *automorphisms*
  - *boperation* (*auto*, *autoi*) = *autoe* $\wedge$ *boperation* (*autoi*, *auto*) = *autoe*

Invariants: (i) For any two automorphisms *auto1*and *auto2* there exists an automorphism *auto3* such that: *boperation* (*auto1*, *auto2*) = *auto3*. This property defines the binary operation over the structure *GroupAutomorphisms*. (ii) For any three automorphisms *auto1*, *auto2* and *auto3*, the binary operation satisfies the conditions: *boperation* ((*boperation* (*auto1*, *auto2*)), *auto3*) = *boperation* (*auto1*, (*boperation* (*auto2*, *auto3*))). This is the associative property over the set of all the automorphisms. (iii) For any automorphism *auto*, there exists an automorphism *endoe* such that: *boperation* (*auto*, *autoe*) = *auto* $\wedge$ *boperation* (*autoe*, *auto*) = *auto*. This property ensures the existence of unique identity element of the given collection. (iv) For any automorphism *auto*, there exists an automorphism *endoi* such that: *boperation* (*auto*, *autoi*) = *autoe* $\wedge$ *boperation* (*autoi*, *auto*) = *autoe*. This property is used to prove the unique existence of the inverse of each element in the set A (A) of all the automorphisms.

VI. Conclusion

The main objective of this research was proposing an integration of fundamental concepts in algebraic automata and Z notation. To achieve this objective, first we described formal specification of an algebraic automaton and then its extended form was described. A linkage was identified and formalized between various structures of algebraic automata and Z structures. Few important concepts of homomorphism and isomorphism were described between these algebraic structures. Finally its extended form, endomorphism and automorphism, over monoid and groups automata were described using Z notation.

Our idea is important and original because we have observed after integrating that a natural relationship exists between these approaches. This work is also important because formalizing graph based notation is not easy as there has been little tradition of formalization of graph theory due to concreteness of the graphs [35]. Our work is useful for researchers interested in integration of approaches for modeling the complex systems. We believe that this research is also useful because it is focused on general principles and concepts and this integration can be used for modeling systems after required reduction.

An extensive survey of existing work was done before initiating this research. Some interesting work [21], [22], [26], [36], [37], [38] was found but our work and approach are different because of conceptual and abstract level integration of Z and automata. Why and what kind of integration is required, were two basic questions in our mind before initiating this research. Since automaton is best suited for modeling behavior while Z is an ideal one used for describing state of a system. This distinct in nature but supporting behavior of Z encouraged us to integrate Z with automata. We believe that this work will be useful in development of integrated tools increasing their modeling power. Formalization of some other concepts in algebraic automata is under progress and will appear soon.

References

[1] A. Hall, Correctness by Construction: Integrating Formality into a Commercial Development Process, Praxis Critical Systems Limited, U.K. Springer-Verlag, London. ISBN:3-540-43928-5.

[2] C. J. Burgess, The Role of Formal Methods in Software Engineering Education and Industry, University of Bristol, UK. 1995.

[3] B. A. L. Gwandu, D. J. Creasey, The importance of Formal Specification in the Design of Hardware Systems, School of Electron. & Electr. Eng., Birmingham University, 1994.

[4] H. A. Gabbar, Fundamentals of Formal Methods, Modern Formal Methods and Applications, Springer, ISBN, 9781-4020-4222-5, 2006.

[5] N. A. Zafar, A. Hussain and A. Ali, "Integration of Algebraic Automata and Z Notation Conforming Some Important Structures", 4th International Computer Engineering Conference, Egypt, 2008.

[6] E. A. Boiten, J. Derrick, and G. Smith, Integrated Formal Methods (IFM04), Canterbury, UK, Springer-Verlag, 2004.

[7] J. Davies, and J. Gibbons, Integrated Formal Methods (IFM07), Oxford, UK, Springer-Verlag, 2007.

[8] J. Romijn, G. Smith, and J. v .d Pol, Integrated Formal Methods (IFM 05), Eindhoven, The Netherlands, Springer-Verlag, 2005.

[9] K. Araki, A. Galloway, and K. Taguchi, Integrated Formal Methods (IFM99), York, UK, Springer-Verlag, 1999.

[10] M. Butler, L. Petre, and K. Sere, Integrated Formal Methods (IFM02), Turku, Finland, Springer-Verlag, 2002.

[11] W. Grieskamp, T. Santen, and B. Stoddart, Integrated Formal Methods (IFM 2000), Dagstuhl Castle, Germany, Springer-Verlag, 2000.

[12] T. B. Raymond, Integrating Formal Methods by Unifying Abstractions, Intec, Ghent University, Belgique, 2006.

[13] J. S. Dong, R. Duke, and P. Hao, Integrating Object-Z with Timed Automata (ICECCS05), pp 488-497, 2005.

[14] J. S. Dong, et al., Timed Patterns: TCOZ to Timed Automata, The 6th International Conference on Formal Engineering Methods (ICFEM'04), LNCS, pp 483-498, 2004.

[15] R. L. Constable, P. B. Jackson, Naumov, P., Uribe, J.: Formalizing Automata II: Decidable Properties, Cornell University, 1997.

[16] R. L. Constable, P. B. Jackson, P. Naumov, J. Uribe,: Constructively Formalizing Automata Theory, Foundations Of Computing Series, MIT Press, ISBN:0-262-16188-5, 2000.

[17] R. Bussow, and W. Grieskamp, A Modular Framework for the Integration of Heterogeneous Notations and Tools, Integrated Formal Methods (IFM 99), York, UK, Springer-Verlag, 211–230, 1999.

[18] M. Heiner, and M. Heisel, Modeling Safety Critical Systems with Z and Petri nets, International Conference on Computer Safety, Reliability and Security, LNCS, Springer, pages 361–374, 1999.

[19] X. He, Pz nets a Formal Method Integrating Petri nets with Z, Information & Software Technology, 43(1):1–18, 2001.

[20] H. Leading, J. Souquieres,: Integration of UML and B Specification Techniques: Systematic Transformation from OCL Expressions into B, Proceedings of Asia-Pacific Software Engineering Conference (APSEC02), Australia, 2002.

[21] H. Leading, J. Souquieres,: Integration of UML Views using B Notation, Proceedings of Workshop on Integration and Transformation of UML models (WITUML02), Spain, 2002.

[22] W. Wechler, The Concept of Fuzziness in Automata and Language Theory, Akademic-Verlag, Berlin, 1978.

[23] N. M. John, and S. M. Davender, Fuzzy Automata and Languages: Theory and Applications, Chapman & HALL, CRC, United States of America, ISBN: 1-58488-225-5, 2002.

[24] M. Ito, Algebraic Theory of Automata and Languages, World Scientific Publishing Co., ISBN: 981-02-4727-3, 2004.

[25] D. K. Kaynar, and N. Lynchn, The Theory of Timed I/O Automata, Morgan & Claypool Publishers, 2006.

[26] C. Godsil, and G. Royle, Algebraic Graph Theory, Springer-Verlag, ISBN 0-387-95220-9, 2001.

[27] D. Conrad, B. Hotzer, Selective Integration of Formal Methods in the Development of Electronic Control Units, Research Institute for Automotive Engineering and Vehicle Engines, 1998.

[28] M. Brendan, and J. S. Dong, Blending Object-Z and Timed CSP: An Introduction to TCOZ, 1998.

[29] J. M. Spivey, The Z notation: A Reference Manual, Englewood Cliffs, NJ, Printice-Hall, 1989.

[30] J. M. Wing, A Specifier, Introduction to Formal Methods, IEEE Computer, Vol.23, No.9, pp.8-24, 1990.

[31] J. A. Anderson, Automata Theory with Modern Applications, Cambridge University Press. ISBN**:** 9780511223013, 2006

[32] L. L. Claudio, et al., Applications of Finite Automata Representing Large Vocabularies, John Wiley & Sons, ISBN:0038-0644, (1993)

[33] Y. V. Moshe, Nontraditional Applications of Automata Theory, Theoretical Aspects of Computer Software, ISBN: 3540578870, 1994.

[34] D. I. A. Cohen, Introduction to Computer Theory, 2ns Edition, City University of New York, pp. 149-152, 2000.

[35] C. T. Chou, A Formal Theory of Undirected Graphs in Higher Order Logic, 7th Int'l Workshop on Higher Order Logic Theorem Proving and Application, pp.144-157, 1994.

[36] D. P. Tuan, Computing with Words in Formal Methods, University of Canberra, ACT 2601, Australie, 2000.

[37] J. P. Bowen, Formal Specification and Documentation Using Z: A Case Study Approach, International Thomson Computer Press, 1996.

[38] S. A. Vilkomir, J. P. Bowen, Formalization of Software Testing Criterion, South Bank University, London, ISBN: 0-7695-1372-7, 2001.