Full Length Research Paper

Formal partitioning analysis and verification of extended algebraic automata

Nazir Ahmad Zafar
Department of Computer Science, King Faisal University, Hofuf, Saudi Arabia. E-mail: nazafar@kfup.edu.sa. Tel: +966-35887082.

Accepted 3 November, 2011

Algebraic automata is getting much importance in theoretical computer science because of its various applications, for example, in optimization of programs, verification of protocols, cryptography and modeling biological phenomena. Design of a complex system not only requires functionality but it also needs to capture its control behaviour. This paper is a part of our ongoing research on integration of algebraic automata and formal methods. Algebraic automaton is a powerful tool in modeling behaviour while Z is an ideal specification language used for describing statics of a system. Consequently, an integration of algebraic automata and Z will be a useful tool for modeling of complex systems. In this paper, we have described formal partitioning analysis of extended algebraic automata because of its use in components based modeling. At first, formal specification of constructing sub-automata for given automata is presented. Then equality of two given automata is verified. In next, cycles are identified and finally formal partitioning analysis of extended algebraic automata is provided. The formal specification is checked, analyzed and validated using Z/Eves tool.

Key words: Algebraic automata, partitioning automata, component based modeling, Z notation, validation.

INTRODUCTION

When software is used in modern systems, its failure may cause a big loss in terms of deaths, injuries, financial or environmental damages. Consequently, construction of correct software is as important as hardware or electro-mechanical systems (Hall, 2002). At the current stage of development it is not possible to develop a system using a single formal technique. And, hence, almost any software system must be modified or reconfigured to provide enhanced integrated solution (Isazadeh, 2004). Various techniques have to be integrated at different levels of development to describe and model a complete, consistent and accurate system. This is the reason that integration of approaches has become a well-researched area in theoretical computer science. Further, an integration of approaches leads to development of automated tools. Software specification is an important tool improving quality of software systems. If the specification is given using diagrams or in textual notations then many drawbacks are usually identified at the later stages of the development of software processes. On the other hand, design of a complex system not only requires functionality, it also needs to model the control behavior. There are various specification techniques suitable for specific aspects in software. For example, algebraic techniques, Z, VDM, and RAISE are usually used for defining data types while process algebras, Petri-nets and automata are best suited for capturing dynamics of systems (Burgess, 1995). Formal methods are mathematical techniques for describing properties of software systems. Their use provides a basis for theorem provers for verification of systems. Formal methods can be applied to remove ambiguities from specification at early stages of the software process because of having rigorous
mathematical and computer tool support. Using formal methods, we can describe a mathematical model to analyze and validate the system increasing confidence over the development (Gwandu and Creasey, 1994).

On the other side, originally, automata was used in switching circuits, programming languages, modeling control behavior, compiler construction, modeling of finite state systems. But in recent years, applications of automata have extended in many areas of computer science. Algebraic automaton is an advanced form of automata having properties and structures from algebraic theory of mathematics. The use of algebraic automata has emerged with several modern applications, for example, optimizing programs, developing model checkers, verifying protocols, cryptography, computer graphics, modeling biological phenomena. Although automata is a powerful tool in modeling of computerized systems but it has its limitations as well. It has only a finite number of states and, hence, it can only be used for a limited number of systems. It has a lack of computer tools which can be used to analyze and prove the abstract models of systems. In addition, there is a need of linking graphical representation of automata to produce descriptive models by developing optimal approaches resulting automated tools supporting a complete and efficient development process of software systems. That is why it requires integrating automata with formal techniques which are rich with computer tools for analyzing the mathematical models proving correctness of systems.

The initial results of this research were presented in (Zafar et al., 2008a, b, c, d) whereas some other similar work of the author(s) is discussed in (Sabir et al., 2008).

In this paper, algebraic automata and formal methods are linked in terms of Z notation to achieve the objective of integrating the approaches. Partitioning analysis is required in developing of complex systems supporting component based modeling. To address this issue, at first, formal specification of constructing sub-automata for a given automaton is presented. For subset construction, the equality of two automata is formalized. Then cycles are identified to partition the automata proving the resultant sub-automata to be disjoints. Finally, formal partitioning analysis of the extended algebraic automata is provided. The formal specification of partitioning of automata is checked, analyzed and validated using Z/Eves tool and proofs of the models are presented under certain assumptions as discussed in conclusive part of this paper. We made the use of schema structure in specification because it is very powerful in defining static part of a system and defining operations over it. Rest of the paper is organized as follows; an overview of related work; introduction to automata is provided; formal partitioning analysis is done using Z notation followed by model analysis and finally conclusion and future work are discussed.

RELATED WORK

In the area of software engineering, software technology is growing rapidly because of the complexity of modern systems and almost any software system is required to be modified or reconfigured to provide enhanced integrated solutions (Gabbar, 2006). This is reason that integration of approaches has become a well-researched area in computer science and engineering (Beek et al., 2004; Hasan and Tahar, 2007; Chin, 1997; Araki et al., 1999; Akbarpour et al., 2002; Derrick and Smith 2000; Kim et al., 2004). In (Dong et al., 2004; 2005) it is integrated object Z and timed automata by considering some aspects of these approaches. Another piece of good work is listed in (Constable et al., 1997, 2000) in which a constructive formalization of some important concepts of automata using Nuprl was presented. In (Bussow and Grieskamp, 1999), a combination of Z notation and Petri-nets in Heiner and Heisel, (1999) and He (2001). An integration of unified modeling language (UML) and B method is given in (Leading and Souquieres, 2002a, b). It introduced the algebraic structures using fuzzy automata theory in (Wechler, 1978). In John and Davender, (2002), a mathematical treatment using fuzzy automata and fuzzy language theory is given when the set of possible values is a closed interval [0, 1]. It described formal languages and automata from the algebraic point of view in (Ito, 2004). At first, it investigated the algebraic structure of automata and then treated a kind of global theory for generalization of approaches. In Kaynar and Lynchn (2006), it developed a new approach and presented a modeling framework which is in fact a basic set of mathematical models to support description and analysis of timed computing systems. In Godsil and Royle (2001), it presented some important concepts of algebraic graph theory with an emphasis on current structures rather than classical topics of graph theory.

AUTOMATA AND APPLICATIONS

There are various applications of automata theory and it has become a basis for modeling systems in computer science and engineering (Claudio and Tomasz 1993). Modeling control behavior, compiler construction, finite state systems are some application areas of automata theory (Cohen, 2000). Finite automata are abstract mathematical models of machines which can be represented using diagrams, and are used for modeling and viewing the complex systems. These models can be used to perform computation 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 configurations then the given input is accepted by the automata.

The algebraic automaton is an advanced form of automata having properties and structures from algebraic theory of mathematics. It has emerged with several modern applications, for example, optimization of logic based programs. Design and development of model checkers, specification of protocols and human computer interaction are few other examples of it. Further, applications of automata are not limited to it but are being seen in many other disciplines of computer science, for example, cryptography, modeling biological phenomena and computer graphics. One of the major drawbacks of automata is that the diagrams have been difficult to use except the very trivial cases. A given automata may have different implementation for the same mathematical model and consequently its time and space complexity must be different which is another issue in modeling using automata. Because of these limitations, it is observed that this single approach cannot be used for complete modeling of a system and consequently its integration is required with other useful approaches.

Based on the reasoning given above, linking Z notation and algebraic automata will be a very useful tool for modeling using this integrated approach.

**FORMAL PARTITIONING ANALYSIS**

In this section, the algebraic automata structures linked with algebraic theory are formalized to produce formalized algebraic automata. Formal notation in terms of Z is used producing the formal models which are then validated and verified using Z/EVES toolset. Formal specification of partitioning analysis is given in this section. At first, we describe formal specification of the algebraic automata then the subset construction procedure is formalized to generate and verify the subsets of a given automata. Based on the reasoning given above, linking Z notation and algebraic automata will be a very useful tool for modeling using this integrated approach.

**Extended algebraic automata**

An extended automaton, denoted by X-automaton, is a 2-tuple $T = (S, X)$, where (i) $S$ is a finite non-empty set of states, (ii) $X$ is a finite non-empty set of inputs. The above automaton $T$ is called an extended automaton if there exists a function $\delta$ of $S \times X^*$ into $S$, called a state transition function, such that $\delta(s, uv) = \delta(\delta(s, u), v)$ and $\delta(s, \varepsilon) = s$, $\forall s \in S$ and $\forall u, v \in X^*$, where $X^*$ is a free monoid generated by $X$ and $\varepsilon$ is its identity called empty word. That is, the transition function takes a state and a string as input and produces a state as an output. Here, $S$ and $X$ are assumed sets and denoted by $S$ and $X$ respectively.$[S, X]$.

In modeling using sets in Z, a high level of abstraction is supposed. For example, we do not insist upon any effective procedure for deciding whether an arbitrary element is a member of a set. Consequently, $S$ and $X$ are sets over which the operations of cardinality, subset and complement operators cannot be defined.

To describe set of states of $T$, a variable state is introduced. Since, a given state $q$ is of type $S$ therefore states must be of type of power set of $S$. For the set of inputs, a variable input is used. Since, a given string is of type seq $X$ therefore input is of type of power set of seq $X$. The null string is denoted by null. As we know that $\delta$ relation is a function because for each input $(q, s)$, where $q$ is a state and $s$ is in set of input there must be a unique output $q'$ of type $S$ which is image of $(q, s)$ under the transition function $\delta$. Hence, we can declare $\delta$ as of type: $S \times \text{seq } X \rightarrow S$. All the above components are encapsulated and put in the schema Extended, its formal description is given as follows:

**Extended**

| states: F S |
| input: F (seq X) |
| null: seq X |
| transition: $S \times \text{seq } X \rightarrow S$ |

**Invariants**

(i) The sets states and input are non-empty. (ii) The string concatenation is a binary operator defined on the set of input. (iii) The associative property must be satisfied under the concatenation operator. (iv) The
null string *null* is the identity of the set of all the input. (v) The transition function results the same state if null string is read in moving from one state to another. The function gives the same state reading concatenation of two strings together or separately. For each \((q, s)\) in domain of transition function, \(q\) is an element of states and \(s\) is an element of strings of the given automata. The range of the transition function is a subset of set of all the states.

**Creating objects from schema**

In this aspect, creation procedure of objects from a given schema is described. For example, let \(A\) and \(B\) are two extended algebraic automata to be created which are denoted by \(\text{Extendeda}\) and \(\text{Extendedb}\) respectively. Renaming is an important concept of \(Z\) which we have used here. Now we create \(\text{Extendeda}\) and \(\text{Extendedb}\) from \(\text{Extended}\) with the same pattern but with different components using renaming rather than creating it from scratch. Renaming is sometimes useful because in this way we are able to introduce a different collection of variables with the same pattern. The components, states, input, null and transition of the automaton \(\text{Extended}\) are replaced by \(\text{statesa}\), \(\text{inputa}\), nulla and transitiona producing the automaton \(\text{Extendeda}\). Similarly, the same components, states, input, null and transition of the same automaton \(\text{Extended}\) are replaced by \(\text{statesb}\), \(\text{inputb}\), nullb and transitionb producing the other automaton \(\text{Extendedb}\). The definitions of the variables of the automata \(\text{Extendeda}\) and \(\text{Extendedb}\) are same as defined for the \(\text{Extended}\). A formal procedure of creating the schema is given as follows:

\[
\begin{align*}
\text{Extendeda} &= \text{Extended}\left[\text{statesa}/\text{states}, \text{inputa}/\text{input}, \text{nulla}/\text{null}, \text{transitiona}/\text{transition}\right] \\
\text{Extendedb} &= \text{Extended}\left[\text{statesb}/\text{states}, \text{inputb}/\text{input}, \text{nullb}/\text{null}, \text{transitionb}/\text{transition}\right]
\end{align*}
\]

**Subset construction**

In this part, a formal procedure of creating subsets of an extended algebraic automaton is described and verified using \(Z\). To define this operation, schema named \(\text{SubsetConstruction}\) as an operator is introduced as given below. The universal set of type automata is given as input and collection of automata of type of power set of automata is created as output of the schema. The variables are defined in first part and subsets relationship with the universal set is given in second part of schema.

\[
\begin{align*}
\text{SubsetConstruction} & \\
\text{universal: Extended} & \\
\text{subsets: F Extended}
\end{align*}
\]

\[
\begin{align*}
\forall \text{ex}: \text{Extended} | \text{ex} \in \text{subsets} \\
& \cdot \forall q: S \cdot q \in \text{ex} \cdot \text{states} \Rightarrow q \in \text{universal} \cdot \text{states} \\
& \forall \text{ex}: \text{Extended} | \text{ex} \in \text{subsets} \\
& \cdot \forall q: S; s_1, s_2: \text{seq} X \\
& \mid q \in \text{ex} \cdot \text{states} \land s_1 = \text{ex} \cdot \text{input} \land s_2 = \text{ex} \cdot \text{input} \\
& \land (\text{ex} \cdot \text{transition}, (q, \text{ex} \cdot \text{null})) = \text{appliesSTo} \\
& \land (\text{ex} \cdot \text{transition}, (q, s_1 \sim s_2)) = \text{appliesSTo} \\
& \land (\text{ex} \cdot \text{transition}, (q, s_1)) = \text{appliesSTo} \\
& \land (\text{ex} \cdot \text{transition}, (q, s_2)) = \text{appliesSTo} \\
& \Rightarrow (\text{ex} \cdot \text{transition}, (q, s_1 \sim s_2)) = \text{appliesSTo} \\
& \land (\text{ex} \cdot \text{transition}, (q, \text{ex} \cdot \text{null})) = q \\
& \land (\text{ex} \cdot \text{transition}, (q, \text{ex} \cdot \text{null})) = \text{appliesSTo} \\
& \land \text{transitiona} \\
& \land (\forall q: S; s: \text{seq} X | (q, s) \in \text{dom} \text{ex} \cdot \text{transition} \\
& \cdot (q \in \text{ex} \cdot \text{states} \land s \in \text{ex} \cdot \text{input}) \\
& \land (\forall q: S | q \in \text{ran} \text{ex} \cdot \text{transition} - q \in \text{ex} \cdot \text{states}) \\
\forall \text{ex}: \text{Extended} | \text{ex} \in \text{subsets} \\
& \cdot \forall q: S; s: \text{seq} X | q \in \text{ex} \cdot \text{states} \land s \in \text{ex} \cdot \text{input} \\
& \land (q, s) \in \text{dom} \text{ex} \cdot \text{transition} \land (q, s) \in \text{dom} \text{universal} \cdot \text{transition} \\
& \exists q_1: S | q \in \text{ex} \cdot \text{states} \land q \in \text{ran} \text{ex} \cdot \text{transition} \\
& \cdot (\text{ex} \cdot \text{transition}, (q, s)) = \text{appliesSTo} \\
& \land \text{transitionb} \\
& \land (\forall q: S | q \in \text{ran} \text{ex} \cdot \text{transition} - q \in \text{ex} \cdot \text{states}) \\
& \land \text{universal} \cdot \text{transition}(q, s) = q_1
\end{align*}
\]

**Invariants**

(i) For any automaton, the set of all the states of it is a subset of states of universal automata. (ii) For any automaton in the collection, the transition function results the same state if null input is read in moving from one state to another. The transition function results the same state in reading concatenation of two inputs together or separately. For each \((q, s)\) in domain of transition function, \(q\) is an element of states and \(s\) is an element of input of the given automata. The range of the transition function is a subset of set of all the states. (iii) For any automaton in the collection, the transition function of the automaton behaves same as of transition function of the universal automaton.
Equality operator

In this part, formal specification of checking two automata to be equal is checked by using the Equality operator as given below. There are two inputs of the schema that is universal automaton and collection of automata. In the schema it is verified if any two sub-automata of the universal automaton are equal.

Identifying cycles of automata

Formal specification of identifying cycles of a given automaton is described by using the Cycles operator as given below. There are two inputs of the schema that is collection of automata and set of cycles. In the schema it is verified that set of cycles is a subset of collections and are in fact sub-automata. The definition of a cycle is same as in graph theory in addition to transition function in moving from one state to the other.

Formal partitioning analysis

In this aspect, formal specification of partitioning analysis is given by using the Partitioning operator as given below. There are two inputs of the schema that is collection of automata and set of relations based on
the extended automata. In the schema, it is verified that for any given relation and for any order pair of the relation the sub-automata of the given automaton are disjoint.

MODEL ANALYSIS

As we know there is hardly any computer tool which may guarantee about the complete correctness of a computer model. That is why we believe that even the specification is written, in any of the formal languages, it may contain potential errors. These errors may range from syntax errors to hazardous inconsistencies. The Z/Eves is one of the most powerful tools which can be used for analyzing the specification written in Z. It is integrated with various facilities which provide rigorous analysis of the system to be developed and has automated deduction capability. Because of abstract expressive power and model checking facilities, Z is a most popular among all the formal techniques. The syntax and type checking are used in this research, which do not require interaction with the theorem proving facility.

Domain checking facility of Z/Eves allows one to write the meaningful expressions. Z/Eves is used to check the specifications for the complete domain errors. It is observed that domain checking is much harder than the syntax and type checking. This is because the syntax and type check is performed automatically whereas one has to interact with the theorem prover to perform the domain checking. It is also observed that proof ‘by reduce’ in the proof window was enough for most of our specification for domain checking. If the specification passes the domain, we get a YES otherwise NO. The schema expansion facility was used to unravel the complex schemas. This facility simplified the results which were not easy to understand otherwise by showing detailed meaning of the given schema. Prove by reduce is one of the most important facility in the Z/Eves toolsset and is used for analyzing the specification of the system. In Table 1, the first column represents name of the schema which was evaluated, the second for syntax and type check, third for domain checking, fourth for reduction and the last one for the proof by reduction. The * symbol annotated with y represents that it required manual interaction with tool for analyzing the specification.

CONCLUSION

In this paper, we have identified and established a link between algebraic automata and Z notation. For this purpose, formal partitioning analysis of the algebraic automata was provided. As algebraic automaton is best suited for modeling behavior while Z is an ideal notation used for describing state space of a system and, hence, it was required to integrate these approaches for modeling of complex systems. An extensive survey of existing work was done before initiating this research. Some interesting work was found but our work and approach are different because of abstract and conceptual level integration. This work is different from others because we have given a generic approach to link Z and algebraic automata with a computer tool support. The idea is important and original because we have observed after integrating that a natural relationship exists between algebraic
automata and Z notation. 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 it.

REFERENCES


