International Journal of Information, Control and Computer Sciences
ISSN: 2517-9942
Vol:2, No:4, 2008

Construction of Intersection of Nondeterministic Finite Automata using Z Notation

Nazir Ahmad Zafar, Nabeel Sabir, and Amir Ali

***Abstract*—**Functionalities and control behavior are both primary requirements in design of a complex system. Automata theory plays an important role in modeling behavior of a system. Z is an ideal notation which is used for describing state space of a system and then defining operations over it. Consequently, an integration of automata and Z will be an effective tool for increasing modeling power for a complex system. Further, nondeterministic finite automata (NFA) may have different implementations and therefore it is needed to verify the transformation from diagrams to a code. If we describe formal specification of an NFA before implementing it, then confidence over transformation can be increased. In this paper, we have given a procedure for integrating NFA and Z. Complement of a special type of NFA is defined. Then union of two NFAs is formalized after defining their complements. Finally, formal construction of intersection of NFAs is described. The specification of this relationship is analyzed and validated using Z/EVES tool.

***Keywords*—**Modeling, Nondeterministic finite automata, Z notation, Integration of approaches, Validation.

I. Introduction

IN this paper, a relationship between automata and Z notation is investigated. Automata have various applications in many areas of computer science and engineering. Modeling control behavior, compiler constructions, modeling of finite state systems, defining a regular set of finite words are some of the traditional applications of automata. Automata have emerged with several modern applications, for example, optimization of logic based programs, verification of protocols [1] and human computer interaction. The Z notation [2] is a model oriented approach based on set theory and first order predicate logic. It is used for specifying the abstract data types and sequential programs. Z notation can also be used to define state of a system and then defining operations over it.

The design of a complex system, not only requires the techniques for capturing functionalities but it also needs to model control behavior [3]. Functions over any of the systems can be decomposed in terms of operations and the constraints, and, hence, Z notation is an ideal application for this purpose.

Control over a system can be viewed in terms of visual flows in between the system's functions. Automata theory is very powerful thereat. Consequently, it requires an integration of automata and Z to increase modeling power for a complex system, which is one of the objectives of this research.

The design of a complex system, not only requires the techniques for capturing functionalities but it also needs to model control behavior [3]. Functions over any of the systems can be decomposed in terms of operations and constraints, and hence Z notation is an ideal application for this purpose. Control over a system can be viewed in terms of visual flows in between the system's functions. Automata theory is very powerful thereat. Consequently, it requires an integration of automata and Z to increase modeling power for a complex system, which is one of the objectives of this research.

As we know that nondeterministic and deterministic finite automata are equivalent in power, in a sense, that if a language is recognized by one, it is also recognized by the other. Nondeterministic finite automata (NFA) are sometimes useful because constructing an NFA is easier than constructing deterministic finite automata (DFA). This is because the complexity of mathematical work is reduced using NFA. Further, many important properties in automata can be established easily using NFA. For example, to prove that a union or concatenation of regular languages is regular using NFA is much easier than using DFA [4]. This is another reason that NFA is selected to be integrated with Z notation.

Nondeterministic finite automata are abstract models of machines which can be represented using diagrams. These models can be used to perform computations on inputs by moving through a sequence of configurations. An NFA consumes the entire input of symbols and for each input symbol it transforms to a new state until all symbols have been consumed. If we are able to reach any of the accepting state after consuming whole input then the input is accepted.

At this level of integration, we have defined two NFAs and their complements are described. As we know that complement of an NFA is not well defined in general therefore, in this paper, we have proposed it only for particular cases. Union of the complemented NFAs is constructed and formal specification of their relationships is given. Finally intersection of the given NFAs is constructed by taking complement of the resultant. Formal specification of the whole

N. A. Zafar is with the Faculty of Information Technology, University of Central Punjab, Lahore, on leave from Pakistan Institute of Engineering Applied Sciences, Islamabad, Pakistan (phone: +92-51-9290273-4; fax: +92-51-2208070; e-mail: nazafar@pieas.edu.pk).

N. Sabir is in Faculty of Information Technology, University of Central Punjab, Lahore, Pakistan (e-mail: nabeel.bloch@ucp.edu.pk).

A. Ali is a student of PhD in Faculty of Information Technology, University of Central Punjab (e-mail: amiralishahid@ucp.edu.pk).

set of activities and the relationship between Z and automata is analyzed and validated using Z/EVES tool [5]. The main objectives of this paper are: (i) an integration of automata and Z notation by giving a syntactic and semantics relationship, (ii) linking constructs of NFA and Z notation such that both of these notations can be used in a cohesive way and (iii) reducing implementation issues of NFA.

Although integration of approaches is a well researched area [6], [7], [8], [9], [10], [11] but there does not exist much work on formalization of graphical based notations. The work [12], [13] of Dong et al. is close to ours in which they have integrated Object Z and Timed Automata for some aspects of automata. Another piece of good work is listed in [14], [15] in which R. L. Constable has given a constructive formalization of some important concepts of automata using Nuprl. Some work of interest is also reported in [16]. In [17], a combination of Z with statecharts is established. A relationship is investigated in between Z and Petri Nets in [18], [19]. An integration of UML and B is given in [20], [21].

In section 2, applications of formal methods are discussed. In section 3, applications and limitations of NFA are analyzed. Integration of NFA and Z is given in section 4. Conclusion and future work are discussed in section 5.

II. Applications of Formal Methods

Formal Methods (FM) refers to mathematically rigorous techniques and tools for the specification, design and verification of software and hardware systems [22]. FM uses mathematical notations for writing specifications of the system to be developed. These mathematical notations are particularly derived from the area of set theory, discrete mathematics or graph theory. Thus formal specifications are mathematical expressions with well-defined syntax and semantics [23]. Once formal specifications are written it can be refined into actually implemented system by a process of stepwise refinement. The validation and verification technique offered by FM is applied at each phase of the development process, which ensures the correctness and consistency by giving a high confidence in the system to be developed. Unlike traditional approaches, formal specification uses mathematical notations those have same interpretation throughout the globe [24]. The use of mathematics in writing specifications helps having deeper insight of a system to be developed and provides an excellent medium for its modeling.

One of the major limitations of traditional approaches is that they lack the ability to prove the specifications. The errors and inconsistencies are hidden behind graphical requirements specifications [25], and are usually identified only during implementation and testing phases. Implementation errors are difficult and costly to fix [26]. On the other hand, the mathematical nature of specifications enables to carry out proves. The worth of conducting proves is that it explores the entire state space of the system. FM makes it possible to prove and analyze certain properties of the system during early stages of the development process so that errors in the requirement specifications can be identified and removed. Studies have suggested that FM have tremendous potential for improving the clarity and precision of requirements specification, and in finding important and subtle errors [27]. Therefore FM is an emerging and future technology with its focus to develop high quality and reliable systems [28].

There are several ways in which formal methods may be classified. One frequently-made distinction is between model oriented and property oriented methods [29]. Model oriented methods are used to construct a model of a system's behavior. Property oriented methods are used to describe software in terms of a set of properties, or constraints, that must be satisfied. The Z notation [30] is a model oriented approach, which is based on set theory and first order predicate logic. Although formal methods are being applied successfully in many research areas of computer science and engineering but at the current stage of development, it requires an integration of formal and informal approaches.

III. Limitations of Nondeterministic Finite Automata

Nondeterministic finite automata are abstract models based on mathematical notations which can be represented using diagrams. These models can be used to perform computations on inputs by moving through a sequence of configurations. If we are able to reach any of the accepting state by using a series of computation then the input is accepted.

An extension of NFA is the NFA with ε (epsilon, a null string) defined by NFA $\cup$ {ε} in which the transition function is allowed to a new state without consuming any input symbol. For example, it can move from state A to the next state B by reading ε (without consuming any input symbol) and it creates an ambiguity. To remove this ambiguity, it is more understandable to talk of a set of possible states in which the transition function enters. We have supposed that our nondeterministic finite automaton is based on the set of alphabets in addition to the epsilon symbol and is denoted by NFA. The addition of epsilon, in the set of alphabets of NFA, increases more complexity in conversion from NFA to DFA.

Further, diagrams in NFA have been difficult to be used except the very trivial cases, which is one of the major issues in representation of NFA diagrammatically. It is a fact that a given NFA may have different implementation methodologies and consequently its time and space complexity may vary for different implementation, which is another issue in modeling using NFA. Further, automata cannot be used for defining functions and constraints and consequently it is not possible to model a complete system by this single approach. As a result, its integration will be very useful with Z notation increasing modeling power for a complex system. If we are able to formalize this relationship, then it would be very useful tool not only at academic but at an industrial level as well. This is because the study of automata in class room, after this integration, will increase clarity of concepts. A formal linkage between these approaches is given in the next section

IV. Formal Construction of Intersection of NFAs

A formal construction of intersection of two NFAs is demonstrated. An NFA is a five tuple (Q, ∑, δ, q0, F), where (i) Q is a finite non-empty set of states, (ii) ∑ is a finite set of alphabets, (iii), δ is a transition function, (iv) q0 is the initial state and (v) F is a finite set of final states.

The above 5-tuple is an NFA because for each state q_1, and for every alphabet a, there is a set of states s, such that $\delta(q_1,a)$ = s. The definitions used here are based on well known books on Automata and Computation Theory [31], [32].

Let us suppose that L is a language over a set of alphabets ∑, and is accepted by a machine NFA = (Q, ∑, δ, q0, F). We define complement of language L as the language of all the strings that are not words in L. Mathematically we define as:

$L^c = \{s, s \text{ is a string based on set of alphabets of } \sum \mid s \notin L\}$.

In order to take compliment of deterministic automata we simply swap the accepting and non-accepting states but this is not true in case of an NFA. For example, the NFA1 in Fig. 1 accepts all strings of length greater than or equal to 2. The NFA2, in Fig., is obtained by swapping the final and non-final states of NFA1 which accepts all the strings of any length and hence it is not complement of NFA1. If we suppose that our NFA accepts all the strings of length n and no self loop is allowed on a state then we can take compliment of it by simply swapping the final and non-final states. In this paper such NFAs for constructing intersection are supposed.

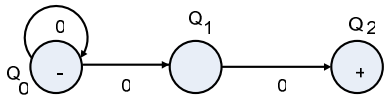

Fig. 1 NFA accepting strings of length at least two

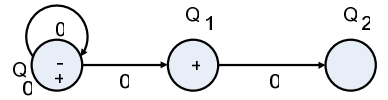

Fig. 2 Complement of the NFA given in Fig. 1

Let NFA1 and NFA2 be two NFAs accepting the languages L1 and L2 respectively. We construct the NFAs accepting the languages $L1^c$ and $L2^c$. Then a new NFA will be designed accepting all the words of $L1^c$ and $L2^c$. By deMorgon's Law: $L1 \cap L2 = (L1^c \cup L2^c)^c$ is the intersection of two given languages for which a new NFA is required.

A. Complementing First NFA

The first non-deterministic finite automata consists of 5-tuple (Q1, ∑1, δ1, q01, F1), where Q1 and ∑1 are represented as `Q` and `Sigma` respectively.

```
[Q, Sigma]
```

In modeling using sets in Z, we do not impose any restriction upon the number of elements and a high level of abstraction is supposed. As a consequent, our `Q` and `Sigma` are sets over which we cannot define any operation, for example, cardinality to know the number of elements in a set.

To describe a set of states, a variable `states1` is introduced. Since a given state q is of type Q therefore `states1` is of type of power set of Q. Similarly, a set of alphabets `alphabets1` is of type of power set of `Sigma`. As we know that δ1 relation is a function because for each input (q1, a), where q1 is a state and a is in set of `alphabet1` there must be a unique output s of type PQ, which is image of (q1, a) under the transition function δ1. Hence we can declare δ1 as, `δ1: Q x Sigma→PQ`. The initial state `q01` is of type Q. The F1, set of final states, is represented by `finals1` and is a type of power set of Q.

The schema structure is used here for composition of these objects because it is very powerful at abstract level of specification. All of the components of NFA1 are encapsulated and put in the schema named as *Nondeterministic1*. We also need to compute the set of all the strings generated by a given alphabets which is declared as: *Strings == seq Sigma.*

∪_*Nondeterministic1*_____
→*states1:* Π *Q*
→*alphabets1:* Π *Sigma*
→*apsi1: Sigma*
→*delta1: Q* ξ *Sigma* φ Π *Q*
→*q01: Q*
→*null1:* Π *Q*
→*finals1:* Π *Q*
→*strings1:* Π *Strings*
∩_____
→*apsi1* ε *alphabets1*
→*q01* ε *states1*
→*finals1* ζ *states1*
→A*q1, q2: Q; a: Sigma | q1* ε *states1 f q2* ε *states1 f a* ε *alphabets1*
→∞ E*s1, s2:* Π *Q | s1* ζ *states1 f s2* ζ *states1*
→ *f* ((*q1*, *a*), *s1*) ε *delta1*
→ *f* ((*q2*, *a*), *s2*) ε *delta1* ∞ (*q1*, *a*) = (*q2*, *a*) ⇒ *s1* = *s2*
→A*st: Strings | st* ε *strings1* ∞ ran *st* ζ *alphabets1*
∠_____

Invariants: (i) The empty string apsi1 is a member of set of alphabets1. (ii) The initial state q01 must be an element of set of states1. (iii) The set of final states is a subset of set of total states. (iv) For each (q, a), where q is an element of states1 and a is member of alphabets1 there is a unique set of states s such that: *delta1(q, a) = s*. (v) Any string given as input to an NFA must be based on the set of alphabets of the same NFA.

After designing NFA1, we need to take its complement. For this purpose a schema *ComplementOfNFA1* is defined. It contains NFA1 and some other components in addition to it, which are required in defining complement of an NFA. A relation is defined between the NFA and its complement.

∪_*ComplementOfNFA1* _____
→Ξ*Nondeterministic1*
→*states1c:* Π *Q*
→*alphabets1c:* Π *Sigma*
→*apsi1c: Sigma*
→*delta1c: Q* ξ *Sigma* φ Π *Q*
→*q01c: Q*
→*null1c:* Π *Q*
→*finals1c:* Π *Q*
→*strings1c:* Π *Strings*
∩_____
→*states1c = states1 f alphabets1c = alphabets1*
→*apsi1c = apsi1 f q01c = q01 f null1c = null1*

→*finals1c = states1 \ finals1*
→*strings1c = strings1*
→*apsi1c* ε *alphabets1c*
→*q01c* ε *states1c ƒ finals1c* ζ *states1c*
→A*q1, q2: Q; a: Sigma | q1* ε *states1c ƒ q2* ε *states1c ƒ a* ε *alphabets1c*
→∞ E*s1, s2:* Π *Q | s1* ζ *states1c ƒ s2* ζ *states1c ƒ* ((*q1*, *a*), *s1*) ε *delta1c*
→ *ƒ* ((*q2*, *a*), *s2*) ε *delta1c* ∞ (*q1*, *a*) = (*q2*, *a*) ⇒ *s1 = s2*
→A*st: Strings | st* ε *strings1c* ∞ ran *st* ζ *alphabets1c*
∠_____

Invariants: (i) The set of states and alphabets in the given NFA and its complement are same. (ii) The null strings, initial states and the sets of dead states in the NFAs and its complement are identical. (iii) The set of final states in complemented NFA is equal to difference of the sets `states1` and `finals1`. (iv) The sets of strings generated by both, NFA and its complement, are equal because these are based on the same alphabets. (v) The empty string `apsi1` is a member of set of `alphabets1`. (vi) The initial state `q01` must be an element of set of `states1`. (vii) The set of final states is a subset of set of total states.

B. Complementing the Second NFA

Let NFA2 = (Q2, ∑2, δ2, q02, F2) be a 5-tuple where all components have the same meaning as defined in case of NFA1. The NFA2 is represented by *Nondeterministic2* as given below and invariants over it are defined similar to NFA1.

∪*_Nondeterministic2*_____
→*states2:* Π *Q*
→*alphabets2:* Π *Sigma*
→*apsi2: Sigma*
→*delta2: Q* ξ *Sigma* φ Π *Q*
→*q02: Q*
→*null2:* Π *Q*
→*finals2:* Π *Q*
→*strings2:* Π *Strings*
∩_____
→*apsi2* ε *alphabets2*
→*q02* ε *states2*
→*finals2* ζ *states2*
→A*q1, q2: Q; a: Sigma | q1* ε *states2 ƒ q2* ε *states2 ƒ a* ε *alphabets2*
→∞ E*s1, s2:* Π *Q | s1* ζ *states2 ƒ s2* ζ *states2 ƒ* ((*q1*, *a*), *s1*) ε *delta2 ƒ*
→ ((*q2*, *a*), *s2*) ε *delta2* ∞ (*q1*, *a*) = (*q2*, *a*) ⇒ *s1 = s2*
→A*st: Strings | st* ε *strings2* ∞ ran *st* ζ *alphabets2*
∠_____

After designing NFA2, its complement is defined which is also a schema and represented as *ComplementOfNFA2* as below. The invariants over it are identified and defined as predicates in the second part of the schema. The informal description of the invariants is not given because it is nothing but a repetition of properties as we defined in the schema *ComplementOfNFA1*.

∪*_ComplementOfNFA2* _____
→Ξ*Nondeterministic2*
→*states2c:* Π *Q*
→*alphabets2c:* Π *Sigma*
→*apsi2c: Sigma*
→*delta2c: Q* ξ *Sigma* φ Π *Q*
→*q02c: Q*
→*null2c:* Π *Q*
→*finals2c:* Π *Q*
→*strings2c:* Π *Strings*
∩_____
→*states2c = states2 ƒ alphabets2c = alphabets2*
→*apsi2c = apsi2 ƒ q02c = q02 ƒ null2c = null2*
→*finals2c = states2 \ finals2 ƒ strings2c = strings2*
→*apsi2c* ε *alphabets2c ƒ q02c* ε *states2c ƒ finals2c* ζ *states2c*
→A*q1, q2: Q; a: Sigma | q1* ε *states2c ƒ q2* ε *states2c ƒ a* ε *alphabets2c*
→∞ E*s1, s2:* Π *Q | s1* ζ *states2c ƒ s2* ζ *states2c ƒ* ((*q1*, *a*), *s1*) ε *delta2c ƒ*
→((*q2*, *a*), *s2*) ε *delta2c* ∞ (*q1*, *a*) = (*q2*, *a*) ⇒ *s1 = s2*
→A*st: Strings | st* ε *strings2c* ∞ ran *st* ζ *alphabets2c*
∠_____

C. Union of Complements

Now we can give a formal definition of union of the complements of given NFAs. Mathematically, we can define: $L1^c \cup L2^c$ = {s, a string based on ∑1 | s ∉ L1 ∨ s ∉ L2}. The NFA accepting $L1^c \cup L2^c$, is given below following the set of properties defining the union relationship.

∪*_NFA1uNFA2*_____
→Ξ*ComplementOfNFA1*
→Ξ*ComplementOfNFA2*
→*states:* Π *Q*
→*alphabets:* Π *Sigma*
→*apsi: Sigma*
→*delta: Q* ξ *Sigma* φ Π *Q*
→*q0: Q*
→*null:* Π *Q*
→*finals:* Π *Q*
→*strings:* Π *Strings*
∩_____
→*alphabets1c = alphabets2c*
→*apsi1c = apsi2c*
→*strings1c = strings2c*
→*apsi* ε *alphabets*
→*q0* ε *states*
→*finals* ζ *states*
→A*q1, q2: Q; a: Sigma | q1* ε *states ƒ q2* ε *states ƒ a* ε *alphabets*
→ ∞ E*s1, s2:* Π *Q | s1* ζ *states*
→ *ƒ s2* ζ *state ƒ* ((*q1*, *a*), *s1*) ε *delta*
→ *ƒ* ((*q2*, *a*), *s2*) ε *delta* ∞ (*q1*, *a*) = (*q2*, *a*) ⇒ *s1 = s2*
→A*st: Strings | st* ε *strings* ∞ ran *st* ζ *alphabets*
→*states = states1c* Y *states2c* Y {*q0*}
→*alphabets = alphabets1c ƒ apsi = apsi1c*
→A*q: Q; a: Sigma; ss1:* Π *Q | q* ε *states ƒ a* ε *alphabets ƒ* ((*q*, *a*), *ss1*) ε
→*delta* ∞ (*q* ε *states1c* ⇒ (E*ss2:* Π *Q |* ((*q*, *a*), *ss2*) ε *delta1c* ∞ *ss1 = ss2*)) *ƒ*
→(*q* ε *states2c* ⇒ (E*ss3:* Π *Q |* ((*q*, *a*), *ss3*) ε *delta2c* ∞ *ss1 = ss3*))
→ *ƒ* (*q = q0 ƒ a = apsi* ⇒ *ss1* = {*q01c*, *q02c*})
→ *ƒ* (*q = q0 ƒ* ! *a = apsi* ⇒ *ss1 = null*)
→*null = null1c* Y *null2c*
→*finals = finals1c* Y *finals2c*
→*strings = strings1c*
∠_____

Invariants:(i) The set of alphabets in NFA1 and NFA2 are supposed to be same for simplicity of construction. (ii) The null strings in NFA1 and NFA2 must be same. (iii) Since the

alphabets are same for NFA1 and NFA2, therefore the possible strings in both of the automata are also same. (iv) The empty string apsi is a member of set of alphabets. (v) The initial state q0 must be an element of set of states. (vi) The set of final states is a subset of set of total states. (vii) For each (q, a) there is a unique set of states s such that: delta(q, a) = s. (viii) Any string given as input to an NFA must be based on the set of alphabets of the same NFA. The empty string apsi is a member of the set of alphabets of the resultant NFA. (ix) The set of states of the resultant NFA is equal to the union of the sets of states of NFA1, NFA2 and a set consisting of a single element q0. The q0 is a new state introduced at the time of union and is initial state in the resultant NFA. (x) The set of alphabets (including null string) in the resultant NFA is same as in the NFA1 or NFA2. (xi) For any state q of states and an element a of the alphabets of the NFA, the transition function holds the, (a) *delta (q, a) = delta1 (q, a), if q ∈ states1, (b) delta (q, a) = delta2 (q, a), if q ∈ states2, c) delta (q, a) = {q_{01}, q_{02}}, if q = q_0 and a = apsi, (d) delta (q, a) = null, if q = q_0 and a ∉ apsi.*(xii) The set of dead states of the resultant NFA is equal to the union of the sets of dead states of NFA1 and NFA2. (xiii) The set of final states of the resultant NFA is equal to the union of the sets of final states of NFA1 and NFA2. (xiv) As the alphabets are same in the given NFA and its complement, therefore the possible strings in these automata are also same.

D. Construction of Intersection

A formal construction of NFA accepting the language ($L1^c \cup L2^c$) is done in the sub-section 4.3. Now if we construct complement of ($L1^c \cup L2^c$) then the resultant automata will accept the language which is intersection of L1 and L2. The schema is represented by *ComplementOfUnionOfNFAs* as given below, which completes this formal construction.

∪_ComplementOfUnionOfNFAs_____
→ΞNFA1uNFA2
→statesc: Π *Q*
→alphabetsc: Π *Sigma*
→apsic: Sigma
→deltac: Q ξ *Sigma* φ Π *Q*
→q0c: Q
→nullc: Π *Q*
→finalsc: Π *Q*
→stringsc: Π *Strings*
∩_____
→statesc = states ƒ alphabetsc = alphabets
→apsic = apsi ƒ q0c = q0
→nullc = null
→finalsc = states \ finals
→stringsc = strings
→apsic ε *alphabetsc*
→q0c ε *statesc*
→finalsc ζ *statesc*
→A*q1, q2: Q; a: Sigma | q1* ε *statesc ƒ q2* ε *statesc ƒ a* ε *alphabetsc*
→∞ E*s1, s2:* Π *Q | s1* ζ *statesc ƒ s2* ζ *statesc ƒ* ((*q1*, *a*), *s1*) ε *deltac*
→ *ƒ* ((*q2*, *a*), *s2*) ε *deltac* ∞ (*q1*, *a*) = (*q2*, *a*) ⇒ *s1 = s2*
→A*st: Strings | st* ε *stringsc* ∞ ran *st* ζ *alphabetsc*
∠_____

Invariants: (i) The set of states in the given NFA and its complement are same. (ii) The set of alphabets in NFA and its complement are equal. (iii) The null strings in the NFAs are identical. (iv) The initial states must be same as well. (v) The set of dead states in both of the automata will remain same. (vi) The set of final states in the complemented NFA is equal to difference of sets `states` and `finals`. (vii) The sets of strings generated by both, NFA and its complement, are equal because these are based on same alphabets. (viii) The empty string `apsic` is a member of set of `alphabetsc`. (ix) The initial state `q0c` must be an element of set of `statesc`. (x) The set of final states is a subset of set of total states. (xi) For each `(q, a)`, where `q` is an element of `statesc` and `a` is member of `alphabetsc` there is a unique set of states `s` such that: `deltac(q, a) = s`. (xii) Any string given as input to an NFA must be based on the set of alphabets of the same NFA.

V. Conclusion

In this paper, we have identified a relationship between Z notation and automata. Our idea is original and important because we have observed, after integrating, that a natural relationship exists there. An extensive survey of existing work was done before initiating this research. There exists a lot of work on integration of approaches, as discussed in the introduction part of this paper. Some of the most relevant and interesting work reported in [14], [15], [18], [19], [21] was found but our work and approach are different because of abstract and conceptual level integration of Z and automata. Further, formalizing graph based notation is not easy, as there has been little tradition of formalization in it due to concreteness of the graphs [31].

Why and what kind of integration is required, were two basic questions in our mind before initiating this research. Capturing functionalities and modeling control behavior are both, equally, important in design of a complex system. Automata is best suited for modeling behavior of a system while Z is an ideal notation to be used describing state space and then defining operations over it. This proves a need and an importance of integration of Z notation with automata. We believe that this combined approach can be very useful in development of integrated tools.

In this paper, we have described formal specification of an algorithm which can be used to construct finite automata accepting a language which is an intersection of two given languages. Although a part of integration of automata and Z notation is treated but we have observed that this approach can be extended to give a formal specification of a more powerful tool. Our work is useful for researchers interested in integrated approaches because of the successful integration of Z notation with automata. We also believe that this research is useful at industrial as well as at academic level because it is focused on general principles and concepts and this integration can be used for modeling a particular application after an intelligent reduction of this specification.

Formalization of some other important concepts in automata is under progress and will appear soon. Further we have taken some assumptions in this integration, for example, it was assumed that the set of alphabets in both of the automata are same. These assumptions were taken for simplicity of construction. In our future work, a more generic integration will be proposed after relaxing such assumptions.

REFERENCES

[1] M. Y. Vardi, and T. Wilke, "Automata - from logic to algorithms," Logic and Automata - History and Perspectives, 2007.
[2] J. M. Spivey, "The Z notation, A Reference Manual," Englewood Cliffs, NJ, Prentice-Hall, 1989.
[3] I. J. Holub, "Finding Common Motifs with Gaps using Finite Automata," In Implementation and Application of Automata, Springer-Verlag, pp: 69-77, ISBN 3-540-37213-X, 2006.
[4] K. Brouwer, W. Gellerich and E. Ploedereder, "Myths and Facts about the Efficient Implementation of Finite Automata and Lexical Analysis," Springer-Berlin, ISBN 978-3-540-64304-3, 2006.
[5] I. Meisels and M. Saaltink, "The Z/EVES Reference Manual," TR-97-5493-03, ORA Canada, CANADA, 1997.
[6] E. A. Boiten, J. Derrick and G. Smith, "Integrated Formal Methods (IFM 2004)," Canterbury, UK, Springer-Verlag, 2004.
[7] J. Davies and J. Gibbons, "Integrated Formal Methods (IFM 2007)," Oxford, UK, Springer-Verlag, 2007.
[8] J. Romijn, G. Smith and J. v. d. Pol, "Integrated Formal Methods (IFM 2005)," Eindhoven, The Netherlands, Springer-Verlag, 2005.
[9] K. Araki, A. Galloway and K. Taguchi, "Integrated Formal Methods (IFM 99)," York, UK, Springer-Verlag, 1999.
[10] M. Butler, L. Petre and K. Sere, "Integrated Formal Methods (IFM 2002)," 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] J. S. Dong, R. Duke and P. Hao, "Integrating Object-Z with Timed Automata," 12th IEEE International Conference on Engineering Complex Computer Systems (ICECCS 2005), pp: 488-497, 2005.
[13] J. S. Dong et al, "Timed Patterns: TCOZ to Timed Automata," 6th International Conference on Formal Engineering Methods (ICFEM'04), LNCS, pp: 483-498, 2004.
[14] R. L. Constable, P. B. Jackson, P. Naumov and J. Uribe, "Formalizing Automata II: Decidable Properties," Cornell University, 1997.
[15] R. L. Constable, P. B. Jackson, P. Naumov and J. Uribe, "Constructively Formalizing Automata Theory," Foundations Of Computing Series, MIT Press, ISBN:0-262-16188-5, 2000.
[16] 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, pp: 211–230, 1999.
[17] R. Büssow, R. Geisler and M. Klar, "Specifying Safety-Critical Embedded Systems with Statecharts and Z: A Case Study," Fundamental Approaches to Software Engineering, Springer Berlin, ISBN, 978-3-540-64303-6, 2004.
[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, pp: 361–374, 1999.
[19] X. He, "Pz nets a formal method integrating petri nets with z," Information & Software Technology, 43(1), pp: 1–18, 2001.
[20] H. Leading and 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 and J. Souquieres, "Integration of UML Views using B Notation," Proceedings of Workshop on Integration and Transformation of UML models (WITUML02), Spain, 2002.
[22] C. Heitmeyer, "On the Need for Practical Formal Methods," Lecture Notes in Computer Science, Vol.1486, pp: 18-26, 1998.
[23] E. Ciapessoni, A. C. Porisini, E. Crivelli, D. Mandrioli, P. Mirandola and A. Morzenti, "From Formal Models to Formally-Based Methods: An Industrial Experience," TOSEM, Vol.8, No.1, pp: 79-113, 1999.
[24] J. P. Bowen, "Ten Commandments of Formal Methods," IEEE Computer, Vol.28, No.4, pp: 56-63, 1995.
[25] J. P. Bowen and M. G. Hinchey, "The Use of Industrial-Strength of Formal Methods," Proceedings of 21st International Computer Software & Application Conference (COMPSAC'97), pp: 332-337, 1997.
[26] M. Barjaktarovic, "The State-of-the-Art in Formal Methods," AFOSR Summer Research Technical Report for Rome Research Site, Formal Methods Framework-Monthly Status Report, F30602-99-C-0166, WetStone Technologies, 1998.
[27] R. W. Butler, "What is Formal Methods?," NASA LaRC Formal Methods Program, 2001.
[28] S. Easterbrook, R. Lutz, R. Covington, J. Kelly, Y. Ampo and D. Hamilton, "Experiences Using Lightweight Formal Methods for Requirements Modeling," IEEE Transactions on Software Engineering, Vol.24, No.1, pp: 4-14, 1998.
[29] J. M. Wing, "A Specifier's Introduction to Formal Methods," IEEE Computer, Vol.23, No.9, pp: 8-24, 1990.
[30] H. A. Gabbar, "Fundamentals of Formal Methods, Modern Formal Methods and Applications," Springer Netherlands, ISBN, 978-1-4020-4222-5, 2006.
[31] J. E. Hopcroft, R. Motwani and J. D. Ullman, "Introduction to Automata Theory, Language and Computation," Addison-Wesley, Reading, 2001.
[32] M. Sipser, "Introduction to the Theory of Computation," Course Technology, ISBN-13: 9780534950972, 2005.
[33] C. T. Chou, "A Formal Theory of Undirected Graphs in Higher Order Logic," 7th International Workshop on Higher Order Logic Theorem Proving and Application, pp: 144-157, 1994.

Nazir A. Zafar (QAU'91-KU'04) was born in Pakistan in 1969. He received his M.Sc. degree in mathematics from Quaid-i-Azam University (QAU), Islamabad, Pakistan in 1991. He did his M. Phil in mathematics from the same university in 1993. He also did his M.Sc. nuclear engineering from the QAU in 1994. He was awarded his PhD degree in computer science from Kyushu University (KU), Fukuoka, Japan, in 2004. Currently, he is working as a Full Professor at Faculty of Information Technology, University of Central Punjab (UCP), Lahore, Pakistan. Before, he was working as an Associate Professor at Department of Computer and Information Sciences, Pakistan Institute of Engineering Applied Sciences, Islamabad, PAKISTAN. He has served at various other universities and scientific organizations in Pakistan. He is the founder and Chair of formal methods research group at UCP. His current research interests are modeling of systems using formal approaches, integration of approaches, etc.

Dr. Zafar is an active member of Pakistan Mathematical Society. Since last few years, he has been involved, as a team member, in organizing the conference with title "International Pure Mathematics Conference (IPMC)" which is conducted every year and 9th IPMC is being held in August, 2008. He has worked for IEEE and some other international and national conferences as a member of technical committee. He has delivered various lectures at national level promoting use and applications of formal methods at academic as well as at industrial level in Pakistan.

Nabeel Sabir (UCP'03) was born in Pakistan in 1979. He received his master in computer science from University of Central Punjab (UCP), Lahore, Pakistan in 2003. He did his MS in computer science from the same university in 2007. Currently, he is a faculty member and PhD student of Dr. Zafar at University of Central Punjab, Lahore, Pakistan. Mr. Nabeel is an active member of various technical and non-technical committees working for UCP.

Amir Ali (BZU'03-QAU'05-NUML'07) was born in Pakistan in 1979. He received his M.Sc. degree in mathematics from Baha-ud-din Zakriya University (BZU), Multan in 2003. He did his post graduate diploma in computer science from Quaid-i-Azam University (QAU), Islamabad, Pakistan, in 2005. He did his master in computer science from National University of Modern Languages (NUML), Islamabad, Pakistan, in 2007. Currently, Mr. Ali is a PhD student of Dr. Zafar at University of Central Punjab, Lahore, Pakistan.