Formal Modeling towards the Context Free Grammar

Nazir Ahmad Zafar[1], Sher Afzal Khan[2], Fahad Alhumaidan[1], Bushra Kamran[3]

[1]Department of Computer Science, King Faisal University, Hofuf, Saudi Arabia
[2]Department of Computer Sciences, Abdul Wali Khan University, Mardan, Pakistan
[3]Faculty of Information Technology, University of Central Punjab, Lahore, Pakistan
nazafar@kfu.edu.sa;sher.afzal@awkum.edu.pk;falhumaidan@kfu.edu.sa;bushra.kamran@ucp.edu.pk

Abstract: The language to control objects is a primary requirement in design of a complex system. Context free grammar plays an important role in modeling control functionalities of a system by grammatical rules. This generates naturally the operation of a system by the language which having commands in the form of strings generated by variables which are nested inside variables arbitrarily deeply. The formal method 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 context free grammar and Z will be an effective tool for increasing modeling power for a complex system. In this paper, we have given a procedure for integrating CFG and Z. Formal definition of a CFG is defined. Then derivation of a string and further development of formal language is formalized. The specification of this relationship is analyzed and validated using Z/EVES tool.
[Zafar NA, Khan SA, Alhumaidan F, Kamran B. **Formal Modeling towards the Context Free Grammar**. *Life Sci J* 2012;9(4):988-993] (ISSN:1097-8135). http://www.lifesciencesite.com. 152

Keywords: Integration of approaches; Context free languages; Formal specification Z; Model checking.

1. Introduction

Machines are 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 or environmental damages. Consequently, constructing correct software is as important as its other counterparts, i.e., hardware or electro-mechanical systems (Hall, 2002). Formal methods are mathematical based techniques used for specification of properties of software and hardware systems for insuring their correctness (Burgess, 1995). We can describe a mathematical model of a system and then it can be analyzed and validated increasing confidence of development (Gwandu et al., 1994).

At the current stage of development in formal methods, it is not possible to develop a system using a single formal technique and as a result integration of approaches is required. That is why integration of approaches has become a well-researched area. Further, it is an open research area 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 data type while Petri nets, process algebras, automata and state-charts are best suited for capturing dynamic aspects 4. Therefore it is required to identify a relationship between static and dynamic modeling techniques for complete development of a system.

Although integration of approaches is a well researched are (Beek et al., 2004; Hasan et al., 2007; Gervais et al., 2005; Araki et al., 1999; Akbarpour et al., 2002; Raymond, 2004) but there does not exist much work on formalization of structures which generates formal languages. Dong et al (2004, 2005) described the integration of Object Z and timed automata. Another piece of good work is reported by Constable (1997, 2000) has proposed a constructive formalization of some important concepts of automata using Nuprl. A relationship is investigated between Petri-nets and Z notation in (Heiner et al., 1999; He, 2001). An integration of B method and UML is presented in (Leadinng et al., 2002,2002a). Wechler, W. has introduced some important algebraic structures in fuzzy automata (Wechler, 1978). In (John et al., 2002), 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. (Mansoor et al.,2007). Proposed, an algorithm to eliminate the useless productions of CFG.

In this paper, a relationship between Z notation and CFG is checked and verified after removing inconsistencies. Formal construction is given using Z notation, and it is analyzed and validated using Z/EVES tool. The major objectives of this research

are: (i) identifying and proposing an integration of context free grammar and formal methods enhancing modeling power of complex systems and (ii) providing a syntactic and semantic relationship between Z and CFG. In section 2, an introduction to formal methods is given. In section 3, an overview of context free grammar is provided. Formal specification of context free grammar is described in section 4. Section 5 describes the modeling analysis. Finally, conclusion and future work are discussed in section 6.

2 Formal Methods

Formal methods are mathematical approaches used for describing and analyzing properties of software systems (Khan et al., 2011, 2011(a), 2011(b); Zafar et al., 2012). These techniques are based on discrete mathematics such as logic, set theory and graph theory. Formal methods may be classified in several ways. Property and model oriented methods are two main classification of it (Brendan, 1998; Khan et al., 2007). Property oriented methods are used to describe software in terms of properties or constraints that must be satisfied. Model oriented methods are used to construct a model of a system's behavior (Spivey, 1989; Ahmad et al., 2011,; Ali et al., 2012, 2012(a)). Formal methods are used to improve quality of software systems by means of documenting and specifying in a precise and structured manner. Although formal methods are successfully applied in many research areas of computer science but at the current stage of their development, it requires an integration of formal and traditional approaches.

Z notation is one of the most popular specification languages in formal method. The Z (Spivey, 1989) 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 and sequential programs can also be modeled using it. In this paper, Z is selected to be integrated with algebraic automata because a natural relationship exists between these approaches. The Z is based upon set theory including standard set operators, set comprehensions, Cartesian products, and power sets. On the other hand the logic of Z is formulated using first order predicate calculus. The Z notation is used in our research because it allows organizing a system in smaller pieces known as schemas. The schema defines the way in which state of the system can be modified. A promising aspect of Z is the mathematical refinement which is a stepwise verifiable transformation of an abstract specification into a concrete executable program. Once formal specifications in Z are written it can be refined into actually implemented system by the process of stepwise mathematical refinements.

3 Context free grammar

A context-free grammar (CFG) generates a formal language where a clause is nested inside another clause making a best use of recursion. Every production of a context free grammar is of the form: $S \rightarrow t$, where S is a non-terminal consisting of a single character/symbol and t is a string which may contain only terminals or non-terminals on combination of both. Further, t might me an empty string. The notation: $S \rightarrow t$ is called a production or a rule. Context free grammar consists of such kind of rules which are applied one after other producing a parse tree. The tree ends with terminals which are leaves of the tree and each internal node is a non-terminal which produces one or more further nodes. The left hand side of a production rule of a context free grammar is always a single non-terminal. Because all rules only have non-terminals on the left hand side and it can easily be replaced with the string on the right hand side of this rule. Further the context in which the symbol occurs is therefore not important and hence the grammar is called context free grammar. It is to be noted that the context free grammar(s) are always recognized by finite state machines having a single infinite taps. For keeping track of nested units, the current parsing state is pushed at the start of the unit and it is recovered at the end. The context free grammars are very important in designing and description of the programming languages and their compilers. The syntax of natural languages can also be analyzed by using it.

The formalism of context-free grammar was developed by Noam Chomsky who described the linguistics in a grammatical form and finally converted into mathematical models providing a precise and simple mechanism for describing of languages. This way of description of languages makes the formalism producing rigorous mathematical studies. The context free grammars allow an efficient parsing of algorithms and their constructions in a simple way. Using the grammar, it can be determined whether a string can be generated or not. Further, the way of generation is also determined. On the other hand, context free languages have their own limitations. For example, some of the operators, which are well-defined in many models of automata theory, do not behave well in case of context free grammar. The intersection of two context free grammars, in general, is not context free, is an example of those operators. The complement of a context free language is not context free, is another example of it. However, union, concatenation and Kleene star operators produce context free language when applied to context-free language or languages.

4 Formal Specification of Context Free Grammar

In this section, an integration of some important concepts of CFG and Z notation is given. It is mentioned that the definitions used are based on the book with title "Algebraic Theory of Automata and Languages" (Brendan et al., 1998). The set of structures used are: (i) CFG (ii) *Derivation* (iii) *Derivations* (iv) *WorldOfCFG* and (v) *LanguageOfCFG*.

We start with the definition of context free grammar which is a 4-tuple $(V,\sum, R, S)$, where, V is a finite set of non-terminals, $\sum$ is the set of terminals, it is disjoint from V, this make the words for a language. S is the start non-terminal. R is the relation from V to $(V \cup \sum)^*$such that $\exists w \in (V \cup \sum)^*: (S,w) \in R$. In the specification denoted by CFG, we define the sets of non-terminal by V, set of terminal by Sigma, the set X denotes both the set of terminals and non-terminals. The notation rules define the relation between V and *seqX*. The *seqX*, denotes the set of all sequences containing terminals and non-terminals. The predicate part defines, *Sigma* is a finite set of terminals, it is disjoint from the set of non-terminals. The production rules are defined by the relation denoted by *rules* such that if ∃ a stringof type *seqX* then $(s0, w) \in rules$. Where *s0* is the start non-terminal used to represent the whole sentence or program.

$[X]$

$V == X$

$Sigma == X$

CFG

$variables: \mathbb{F}\ V$

$terminals: \mathbb{F}\ Sigma$

$alphabets: \mathbb{F}\ X$

$rules: V \leftrightarrow \text{seq}\ X$

$s0: V$

$s0 \in variables$

$\text{dom}\ rules \subseteq variables$

$variables \cap terminals = \{\}$

$alphabets = variables \cup terminals$

$\exists w: \text{seq}\ X \mid w \in \text{ran}\ rules \bullet (s0, w) \in rules$

$\forall st: \text{seq}\ X \mid st \in \text{ran}\ rules \bullet \text{ran}\ st \subseteq alphabets$

Invariants:

1. The variable *s0* must be an element of *variables*.
2. The domain of *rules* relation is subset of variables.
3. The terminals and non-terminals are disjoint sets.
4. The entire set of alphabets is union of terminals and non-terminals.
5. There exists at least one rule which contains start variable on the left hand side.
6. Elements of all the rules are members of alphabets.

Formal Construction of Productions

In this section, we describe the formal specification of production rules. Production rule is substitution rule perform recursively to derive new string of terminal and non-terminal from the string of terminal and non-terminal.

In the specification denoted by the *Derivation*, we specify the process of production of one string from another string of terminal and non-terminal. In the specification *st1* and *st2* are two strings of type *seqX*. We say *st1* yields *st2* if $\exists a \in V, b, st3$ and $st4 \in seqX$ such that:

$st1 = st3 \frown \langle a \rangle \frown st4 \wedge st2 = st3 \frown b \frown st4.$

Thus, *st2* is the result of obtained by the rule *(a, b)* to *st2*.

Derivation

ΞCFG

$drives: \text{seq}\ X \leftrightarrow \text{seq}\ X$

$\forall st1, st2: \text{seq}\ X \mid \text{ran}\ st1 \subseteq alphabets \wedge \text{ran}\ st2 \subseteq alphabets$

$\bullet\ (st1, st2) \in drives$

$\Rightarrow (\exists a: V; b: \text{seq}\ X; st3, st4: \text{seq}\ X$

$st3 \in \text{ran}\ rules \wedge st4 \in \text{ran}\ rules \wedge (a, b) \in rules$

$\bullet\ st1 = st3 \frown \langle a \rangle \frown st4 \wedge st2 = st3 \frown b \frown st4)$

Formal Construction of further Derivations

The specification schema denoted by Derivations is the extension of schema Derivation. This specify the generation of one string of non-terminals or terminals to the string of non-terminals or terminals. In the specification we consider two strings of non-terminals or terminals denoted by st1 and st2. The schema Derivations call the schema Derivation in successive manner and develop the productions: st1⇒st3∧ st3⇒st4 ∧ s4⇒st5∧ st5⇒st2 as specified.

In many fields of computer sciences words generated from grammar to code certain programs used to functionalize a system. The schema denoted by *WorldOfCFG* is used to generate strings of terminals from the strings of non-terminals and terminals. Word of CFG is define to be a string of terminals of type *seq Sigma* generated by successive production from the string of non-terminals or

terminals of type seqX. For generating word *WordOfCFG* uses operation of schema *Derivations* to perform the desire production.

Derivations

$$\Xi Derivation$$
$$drivess: \text{seq}\, X \leftrightarrow \text{seq}\, X$$

$$\forall st1, st2: \text{seq}\, X \mid \text{ran}\, st1 \subseteq alphabets \land \text{ran}\, st2 \subseteq alphabets$$
$$\bullet\ (st1, st2) \in drivess$$
$$\Rightarrow (\exists st3: \text{seq}\,(\text{seq}\, X) \mid 1 \leqslant \#st3 \land \text{ran}\, st3 \subseteq \text{ran}\, rules$$
$$\bullet\ ((st1, st3\ 1) \in drives$$
$$\Rightarrow (\exists a: V; b: \text{seq}\, X; st4, st5: \text{seq}\, X$$
$$st4 \in \text{ran}\, rules \land st5 \in \text{ran}\, rules \land (a, b) \in rules$$
$$\bullet\ st1 = st4 \frown \langle a \rangle \frown st5 \land st3\ 1 = st4 \frown b \frown st5))$$
$$\land (\forall i: \mathbb{N} \mid i \in 2 \mathinner{..} \#st3$$
$$\bullet\ ((st3\ (i - 1), st3\ i) \in drives$$
$$\Rightarrow (\exists a: V; b: \text{seq}\, X; st4, st5: \text{seq}\, X$$
$$st4 \in \text{ran}\, rules$$
$$\land st5 \in \text{ran}\, rules$$
$$\land (a, b) \in rules$$
$$\bullet\ st3\ (i - 1) = st4 \frown \langle a \rangle \frown st5$$
$$\land st3\ i = st4 \frown b \frown st5)))$$
$$\land ((st3\ (\#st3), st2) \in drives$$
$$\Rightarrow (\exists a: V; b: \text{seq}\, X; st4, st5: \text{seq}\, X$$
$$st4 \in \text{ran}\, rules \land st5 \in \text{ran}\, rules \land (a, b) \in rules$$
$$\bullet\ st3\ (\#st3) = st4 \frown \langle a \rangle \frown st5 \land st2 = st4 \frown b \frown st5)))$$

Worlds generated by CFG

WordOfCFG

$$\Xi Derivations$$
$$word?: \text{seq}\, Sigma$$

$$\text{ran}\, word? \subseteq terminals$$
$$(\langle s0 \rangle, word?) \in drivess$$
$$\Rightarrow (\exists st3: \text{seq}\,(\text{seq}\, X) \mid 1 \leqslant \#st3 \land \text{ran}\, st3 \subseteq \text{ran}\, rules$$
$$\bullet\ ((\langle s0 \rangle, st3\ 1) \in drives$$
$$\Rightarrow (\exists a: V; b: \text{seq}\, X; st4, st5: \text{seq}\, X$$
$$st4 \in \text{ran}\, rules \land st5 \in \text{ran}\, rules \land (a, b) \in rules$$
$$\bullet\ \langle s0 \rangle = st4 \frown \langle a \rangle \frown st5 \land st3\ 1 = st4 \frown b \frown st5))$$
$$\land (\forall i: \mathbb{N} \mid i \in 2 \mathinner{..} \#st3$$
$$\bullet\ ((st3\ (i - 1), st3\ i) \in drives$$
$$\Rightarrow (\exists a: V; b: \text{seq}\, X; st4, st5: \text{seq}\, X$$
$$st4 \in \text{ran}\, rules \land st5 \in \text{ran}\, rules \land (a, b) \in rules$$
$$\bullet\ st3\ (i - 1) = st4 \frown \langle a \rangle \frown st5$$
$$\land st3\ i = st4 \frown b \frown st5)))$$
$$\land ((st3\ (\#st3), word?) \in drives$$
$$\Rightarrow (\exists a: V; b: \text{seq}\, X; st4, st5: \text{seq}\, X$$
$$st4 \in \text{ran}\, rules \land st5 \in \text{ran}\, rules \land (a, b) \in rules$$
$$\bullet\ st3\ (\#st3) = st4 \frown \langle a \rangle \frown st5 \land word?$$
$$= st4 \frown b \frown st5)))$$

Language generated by CFG.

The set of words generated by a schema *WordOfCFG* is called a context-free language. In the specification *LanguageOfCFG* a sequence of terminal is denoted by *w* is of type *seq Sigma* belongs to the set *language* if there exist a set of derivations from s0 to w operated by the schema *Derivations* as specified in the following schema.

LanguageOfCFG

$$\Xi Derivations$$
$$language?: \mathbb{P}\,(\text{seq}\, Sigma)$$

$$\forall w: \text{seq}\, Sigma \mid w \in language? \bullet \text{ran}\, w \subseteq terminals$$
$$\forall w: \text{seq}\, Sigma \mid w \in language?$$
$$\bullet\ (\langle s0 \rangle, w) \in drivess \Rightarrow$$
$$(\exists st3: \text{seq}\,(\text{seq}\, X) \mid 1 \leqslant \#st3 \land \text{ran}\, st3 \subseteq \text{ran}\, rules$$
$$\bullet\ ((\langle s0 \rangle, st3\ 1) \in drives$$
$$\Rightarrow (\exists a: V; b: \text{seq}\, X; st4, st5: \text{seq}\, X$$
$$st4 \in \text{ran}\, rules \land st5 \in \text{ran}\, rules \land (a, b) \in rules$$
$$\bullet\ \langle s0 \rangle = st4 \frown \langle a \rangle \frown st5 \land st3\ 1 = st4 \frown b \frown st5))$$
$$\land (\forall i: \mathbb{N} \mid i \in 2 \mathinner{..} \#st3 \bullet ((st3\ (i - 1), st3\ i) \in drives$$
$$\Rightarrow (\exists a: V; b: \text{seq}\, X; st4, st5: \text{seq}\, X$$
$$st4 \in \text{ran}\, rules$$
$$\land st5 \in \text{ran}\, rules$$
$$\land (a, b) \in rules$$
$$\bullet\ st3\ (i - 1) = st4 \frown \langle a \rangle \frown st5$$
$$\land st3\ i = st4 \frown b \frown st5)))$$
$$\land ((st3\ (\#st3), w) \in drives$$
$$\Rightarrow (\exists a: V; b: \text{seq}\, X; st4, st5: \text{seq}\, X$$
$$st4 \in \text{ran}\, rules \land st5 \in \text{ran}\, rules \land (a, b) \in rules$$
$$\bullet\ st3\ (\#st3) = st4 \frown \langle a \rangle \frown st5 \land w = st4 \frown b \frown st5)))$$

---

5 Model Analysis

As we know that there does not exist any computer tool that may guarantee about the complete correctness of a computer model. That is why we can believe that even the specification is written, in any of the formal languages, it may contain potential errors. That means the art of writing formal specification never assures the consistencies, correctness and completeness of the system to be developed. But, on the other hand, if the formal specification is checked and analyzed with computer tools it certainly identifies, if exist, the potential errors in syntax and semantics of the formal description of a system. The Z/EVES is one of the most powerful tools which we have used for writing validating and analyzing the formal specification written in Z.

Conclusion

In this research, the approach of context free grammar is combined with Z notation which defining a relationship between fundamentals of these approaches. At first, we have described formally the structures of context free grammar then formal models of derivation process from a string to a string of non-terminals or terminals is presented. Further we specified the process of production in sequence by using the schema Derivations. At the end we presented the formal model to generate the words from CFG and further its language. Formal proofs of the above models are presented under the certain assumptions. Formal models of few interesting algebraic structures and its variants are proposed by reusing the definitions of the abstract structures. The specification of this integration is verified and validated using Z/Eves tool. An extensive survey of existing work was done before initiating this research. Some interesting work (Wechler, 1978; Tuan, 2000, Bowen, 1996, Vilkomir, 2001) was found but our work and approach are different because of conceptual and abstract level integration of Z and CFG. Why and what kind of integration is required, were two basic questions in our mind before initiating this research. Since, CFG is best suited for modeling system's behavior by using proper sequence of strings, 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 CFG.

Most of the researchers have either taken some examples in defining integration of approaches or have addressed only some aspects of it. Further, there is a lack of formal analysis supported by computer tools. Our work is different from others because we have given a generic approach to link Z and CFG. A computer tool support is provided for analysis and validation of this relationship as well.

Few benefits of using Z are: (i) Every object is assigned a unique type providing useful programming practice. (ii) Several type-checking tools exist to support the specification. (iii) The Z/Eves is a powerful free tool to prove and analyze the specification. (iv) the rich mathematical notations make it possible to reason about behavior of a specified system effectively.

References

1. A. Hall, Correctness by Construction: Integrating Formality into a Commercial Development Process, Praxis Critical Systems Limited, Springer, vol. 2391, pp. 139-157, 2002.
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 and 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, 2006.
5. H. Beek, A. Fantechi, S. Gnesi and F. Mazzanti, State/Event-Based Software Model Checking, Integrated Formal Methods, Springer, vol. 2999, pp. 128-147, 2004.
6. O. Hasan and S. Tahar, Verification of Probabilistic Properties in the HOL Theorem Prover, Integrated Formal Methods, Springer, vol. 4591, pp. 333-352, 2007.
7. F. Gervais, M. Frappier and R. Laleau, Synthesizing B specifications from EB3 Attribute Definitions, Integrated Formal Methods, Springer, vol. 3771, pp. 207-226, 2005.
8. K. Araki, A. Galloway and K. Taguchi, Integrated Formal Methods, Proceedings of the 1st International Conference on Integrated Formal Methods, Springer 1999.
9. B. Akbarpour and S. Tahar and A. Dekdouk, Formalization of Cadence SPW Fixed-Point Arithmetic in HOL, Integrated Formal Methods, Springer, vol. 2335, pp. 185-204, 2002.
10. J. Derrick and G. Smith, Structural Refinement of Object-Z/CSP Specifications, Integrated Formal Methods, Springer, vol. 1945, 2000.
11. T. B. Raymond, Integrating Formal Methods by Unifying Abstractions, Springer, vol. 2999, 2004.
12. J. S. Dong, R. Duke and P. Hao, Integrating Object-Z with Timed Automata, pp 488-497, 2005.
13. J. S. Dong, et al., Timed Patterns: TCOZ to Timed Automata, The 6th International Conference on Formal Engineering Methods, 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, 2000.
16. M. Heiner and M. Heisel, Modeling Safety Critical Systems with Z and Petri nets, International Conference on Computer Safety, Reliability and Security, Springer, pp. 361–374, 1999.

17. X. He, Pz nets a Formal Method Integrating Petri nets with Z, Information & Software Technology, vol. 43(1), pp.1–18, 2001.
18. H. Leading and J. Souquieres, Integration of UML and B Specification Techniques: Systematic Transformation from OCL Expressions into B, Asia-Pacific Software Engineering Conference, 2002.
19. H. Leading and J. Souquieres, Integration of UML Views using B Notation, Proceedings of Workshop on Integration and Transformation of UML Models, 2002a.
20. W. Wechler. The Concept of Fuzziness in Automata and Language Theory, Akademic-Verlag, Berlin, 1978.
21. N. M. John and S. M. Davender, Fuzzy Automata and Languages: Theory and Applications, Chapman & HALL, CRC, 2002.
22. D. Conrad and B. Hotzer, Selective Integration of Formal Methods in the Development of Electronic Control Units, Research Institute for Automotive Engineering and Vehicle Engines, 1998.
23. M. Brendan and J. S. Dong, Blending Object-Z and Timed CSP: An Introduction to TCOZ, Proceedings of International Conference on Software Engineering, 1998.
24. J. M. Spivey, The Z notation: A Reference Manual, Englewood Cliffs, NJ, Printice-Hall, 1989.
25. D. P. Tuan, Computing with Words in Formal Methods, University of Canberra, Australia, 2000.
26. J. P. Bowen, Formal Specification and Documentation Using Z: A Case Study Approach, International Thomson Computer Press, 1996.
27. S. A. Vilkomir and J.P. Bowen, Formalization of Software Testing Criterion, South Bank University, London, 2001.
28. Mansoor, A. A., Khan, A. A.: Removing Useless Productions of a Context Free Grammar through Petri Net, Journal of Computer Science, vol. 3 (7), pp. 494-498, 2007.
29. S.A. Khan and NA Zafar, Improving Moving Block Railway System using Fuzzy Multi-Agent Specification Language, Int. J. Innov. Computing, Inform. Control, 7(7(B)):4517-34, 2011.
30. S.A. Khan and NA Zafar and F Ahmad, Petri Net Modeling of Railway Crossing System using Fuzzy Brakes, International J. Phy. Sci, 6(14): 3389-3397, 2011(a).
31. Zafar, N.A, Khan, S.A and Araki, K, Towards the Safety Properties of Moving Block Railway Interlocking System, Int. J. Innov. Computing, Inform. Control, 8(8): 2012 .
32. S.A. Khan and NA Zafar, Extending promotion for the management of moving block interlocking components., International J. Phy. Sci, 6(31), 7262-70. 2011(b).
33. S.A. Khan and NA Zafar, Promotion of Local to Global Operation in Train Control System, Journal of Digital Information Management (page 228-233)., 2007.
34. F. Ahmad, SA Khan. Module-based Architecture for Periodic job-Shop Scheduling problem, Computers & Mathematics with Applications, 64(1), 1-10, 2012.
35. G Ali, SA Khan, F Ahmad and Zafar, N.A, Visualized and Abstract Formal Modeling towards the Multi-Agent Systems, International Journal of basic and Applied Sciences 2(8)8272-8284, 2012
36. G Ali, SA Khan, F Ahmad and NA Zafar, Formal Modeling towards a Dynamic Organization of Multi-Agent Systems Using Communicating X-Machine and Z-Notation, Indian Journal of Science and Technology, Vol. 5 No. 7, 2012(a).

9/19/2012