SIG on Formal Methods

NEWS LETTER VOLUME 6 , DECEMBER 2014

SIG on Formal Methods:

Formal Methods (FM) has been in existence since 1940's, when Alan Turing proved the logical analysis of sequential programs using the properties of program states; and Floyd, Hoare and Naur used axiomatic techniques to prove program correctness against the specifications in 1960's.

These initial successes helped inculcate an interest in applying FM to the field of computer science.

Academia has been instrumental in bringing this field to the forefront, through continued research and development. The use of Formal Methods requires an expert skill-set expertise and therefore, its use is limited to those trained in the field.

Mission Statement:

Computer Society of India wants the field of Formal Methods to have a wider audience and more people to benefit from the application of these methods to all spheres of life. There is a need to use effective, correct and reliable approaches to design, develop and qualify complex, high assurance system software with the rigid schedules and budget. For this we need advanced tools, techniques and methods. Industry standards like RTCA DO-178C (Civil Aerospace), ISO 26262 (Automotive), IEC 61513(Nuclear), EN50126 (Railways) have recommended the usage of formal –method based approach to be used in the various phases of engineering process to achieve the required levels of safety and security.

Today there are proven techniques and tools that can be used in specification, design and verification & validation phases to assure correct requirement-capture, implementation, software functionality and security. This helps in developing high assurance software for applications such as cyber-physical systems, net-centric warfare systems, autonomous robots and Next Generation Air Transportation.

- **One day workshop on FM was conducted during April 2013**

Objectives of the Special Interest Group (SIG) are:

- To bring together scientists, academicians active in the field of formal methods and willing to exchange their experience in the industrial usage of formal methods
- To coordinate efforts in the transfer of formal methods technology and knowledge to industry
- To promote research and development for the improvement of formal methods and tools with respect to their usage in industry.
- To bring out practical engineering methods where formal methods will be integrated with current engineering methods

Some of the known applications of formal methods are:

- Formal verification, including theorem proving, model checking, and static analysis
- Techniques and algorithms for scaling formal methods
- Use of formal methods in automated software engineering and testing
- Model-based formal development
- Formal program synthesis
- Formal approaches to fault tolerance
- Use of formal methods in safety cases
- Use of formal methods in human-machine interaction analysis
- Use of formal methods in compiler validation and object code verification

3. Committee Members

1. Ms. Bhanumathi K S, Convener
2. Mr.Chander Mannar
3. Prof.Anirban basu
4. Mr. Suman Kumar
5. Prof. Shyam Sundar
6. Ms. Manju Nanda
7. Ms. J. Jayanthi
8. Ms. Saroja Devi
9. Prof. Meenakshi D'Souza
10. Dr. Yoganand Jeepu
11. Dr. Swatnalatha Rao
12. Dr. Aditya Kanade
13. Dr. A. Indira
14. Mr. Dhinakaran Pillai

Convener:

Bhanumathi K S
“Ganadhakshya” #406, 8 C Main,
H R B R First Block
Kalyan Nagar
Bangalore 560043
Email:bhanushekar@gmail.com
Mobile:+91 95350 92589

Formal Methods for Safe and Secure Computer Systems

Compiled by Bhanumathi K S

Definition:

Formal methods can be seen as a scientist's reaction against empirical approaches, namely organizational approaches, which sometimes focus more on the design process than on the product itself, and technical approaches that rely heavily on testing to detect (certain but not all) design and programming mistakes.

Formal methods are multiple and diverse, so that it is difficult to give a unique definition that encloses and characterizes formal methods uniquely.

Formal methods in a broad sense are mathematically well-founded techniques designed to assist the development of complex computer-based systems; in principle, formal methods aim at building zero-defect systems, or at finding defects in existing systems, or at establishing that existing systems are zero-defect.

Three general traits common to most Formal Methods:

1. Language
2. Tools
3. Methodologies

1. Languages:

Formal methods are often associated with mathematical notations or computer languages with a formal semantics that can describe the properties expected from a system and/or the particular ways in which the system is designed (e.g., architecture, algorithms, etc.). Depending on the formal method considered, such descriptions can concern various phases of system development, from requirements, specification, and design to implementation and run-time execution. Whatever the phase considered, a central idea of formal methods is to consider systems, hardware, and/or software as mathematical objects that can be described and analyzed rigorously.

2. Tools:

Formal methods often come with software tools that ensure that the system under development will function as expected, under certain assumptions. This can be done either by guiding and assisting the development in such a way that the resulting system will function properly, correct-by-construction approach or by checking, at various phases, that the resulting system does not diverge from its initial expectations so as to detect, as soon as possible, any design or implementation mistake (formal verification approach, which is a branch of verification and validation).

An important difference between formal methods and traditional testing techniques is the emphasis of formal methods on analyzing (ideally) *all* possible executions of the system, and not

only a few ones. This is essential if the proper functioning of the system has to be mathematically demonstrated, and not only estimated with probabilities

3. Methodologies:

To be effective, formal methods should be well integrated within industrial practice. For this reason, most formal methods are equipped with methodological guidelines for a proper use in real-size system development.

Examples of failures affecting computer-based systems:

1. **Hardware-specific failures:** The Pentium floating-point division bug (1994) and the Cougar Point chipset flaw (2011), which costed Intel 475 million and one billion dollars, respectively.

2. **Software-specific failures:** The Therac 25 radiotherapy engine killed five persons in the 80s due to bad software design.

3. **Large-scale infrastructures:** The failure of the Denver airport automated baggage system (1994) delayed the airport's opening for 16 months with a cost overrun larger than 250 million dollars.

This list is by no means complete, as every week the Risks Digest forum reports new examples of risks to the public caused by computers and computer-based systems.

Reasons for failure or malfunctioning:

1. **Design errors** prevent a system from achieving its intended functionality. Such errors often occur during the early phases of system design and may be caused by inappropriate capture of system requirements, or inaccurate modeling of the actual environment in which the system is supposed to function, or mathematical errors in complex control equations, or errors in critical algorithms and data structures that the system is relying upon, or unexpected interactions between several functionalities that must be provided simultaneously, etc.
2. **Hardware faults** encompass physical or logical issues in microprocessors, microcontrollers, integrated circuits, sensors, actuators, etc. Certain issues come from hardware obsolescence and cannot be prevented from occurring; it is therefore mandatory that systems can detect, cope with, and recover from hardware faults.

3. **Software bugs** are logical mistakes when implementing the software part of a system. There are many kinds of bugs (e.g., run-time errors, non-terminating loops, deadlocks, etc.) depending whether the software is sequential, parallel, or distributed.

4. **Security issues** occur when a system is not robust enough to resist to malevolent users and/or intentional attackers. Nowadays, this has become a critical topic as more and more systems run in an open world connected to the internet.

5. **Performance issues** occur when a system cannot deliver its expected, quantitative performance, e.g., because it executes too slowly or because it consumes too much energy or other resources. There are many systems (e.g., image processing devices, broadcasting networks, consumer electronics, etc.) for which correct functionality is only moderately important, but whose added value and usability critically depend on performance criteria.

Thus the computer-based systems are increasingly assigned mission-and life-critical tasks. Their intrinsic complexity is steadily growing, at the same time, guaranteeing their safety is increasingly difficult, while they are exposed to a growing number of security threats.

Formal methods are a key enabling technology for building safe and secure computer-based systems. They help fighting the software quality crisis, in conjunction with related approaches, such as better technical education, design methodologies, computer languages, and development tools.
At present formal methods have gained industrial recognition, at least in the largest and most innovative companies, the point is no longer to question the usefulness of formal methods, but to discuss where and how formal specifications and verification methods can be introduced in design methodologies, and how the software tools developed in academia can be reused and adapted to various applicative contexts.

****