SIG on Formal Methods

NEWS LETTER VOLUME 5 , NOVEMEBR 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

Report of the Activity:

National Work Shop and Conference on Formal Methods for Software Engineering

Venue: M R C Auditorium, I I Sc , Bangalore, October 15-17, 2014

Report On National Workshop/ Conference On Formal Methods Held At Indian Institute Of Science , October 15 – 17 2014

BY

Special Interest Group On Formal Methods SIG-FM & Bangalore Chapter Of CSI

The Special Interest Group of CSI for FORMAL METHODS and CSI Bangalore Chapter organised a National Workshop/Conference on Formal Methods, NCFM 2014, from Oct 15-17, 2014, at MRC Auditorium, I I Sc, Bangalore.
This was the first National Workshop/Conference on Formal Methods in the country.

The advancement in the technologies, safety and security are key issues and there is a need to use effective and reliable approaches to design, develop and qualify the complex, high assurance system software's within a time-schedule and budget. Formal methods are proving effective in meeting these criteria. Therefore, the need to bring together all the scientists, academicians and industrialists who use formal methods and are willing to share their experiences, under one umbrella.
The NCFM brought together these bright minds who actively use formal methods and this resulted in an intensely enriching environment and where everyone was enlightened at the end of the event. FMSE (Formal Methods Software Engineering) strives to promote research and development for the improvement of formal methods and tools for industrial applications. This is the first initiative from the Special interest Group-SIG on Formal Methods. We believe, this is the first of the many successes we are going to see under the aegis of SIG.
The event was hosted by Indian Institute of Science under the leadership of Prof. R K Shyamsundar TIFR, Prof. Y Narahari, Chairman CSA, I I Sc and Prof. Deepak D'Souza, CSA, I I Sc.
This Workshop/Conference was a resounding success with 50 plus delegates participating from Industry, Academia, Defence and other government agencies.

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

Programs vs models

Compiled by Bhanumathi K S

Definition: Models

Operational specifications for systems and their hardware and/or software components are, in the standard approach, expressed as *models* written in some *modeling language*. Usually, models describe the individual behavior of each component as well as the composition of all components to form a system.

Models can be produced in two different ways:

- They can be developed *a priori*, to describe a system that is under construction. Such models are useful to experiment with a system that does not exist already, to get user feedback about it, to detect its potential design mistakes at soon as possible, and to predict its performance before it is built actually.
- They can be developed or generated *a posteriori*, to describe a system that already exists. Such models are helpful to better understand legacy systems, and to study in advance the impact of modifications or enhancements, without stopping nor perturbating running systems.

- The term *model* is often used with different meanings in mathematics and computer science. To avoid confusion, it should be noted that the models used in formal methods are distinct from three other notions of models:

- They are distinct from *mathematical logic models*, which are interpretations that assign the value true to a logic formula. The models used in formal methods are more general and exist by themselves, without reference to any logic formula. However, when verification is formulated in terms of property satisfaction, both notions of models coincide.

- They are also distinct from *data models*, which are used in software engineering and information systems to specify the structure, meaning, and handling of data. However, certain modeling languages borrow ideas from data models to formally describe data aspects in systems and components.

- They are richer and more general than the *models* and *metamodels* used in the so-called *model-driven* approaches promoted by the OMG (namely *model-driven architecture*, *model-driven engineering*, etc.).

In the context of formal methods, the term *model* has two distinct meanings:

- *High-level models* are mainly intended to humans for the purpose of describing systems. They aim at conciseness, expressiveness, readability, reusability, user-friendliness, etc.

• *Low-level models* are used for theoretical and computational purposes, in particular to specify the semantics of high-level models, — which are often defined by translation to lower-level models—and to be used as data structures by verification algorithms. They aim at expressiveness, mathematical elegance, minimality, simplicity, etc. Therefore, in principle, low-level models should not be directly used by humans to specify real-life systems, because this quickly gets too verbose.
Examples of low-level models are automata and all derived forms of state-transition models (traces, trees, etc.), binary decision diagrams, Boolean equation systems, term algebras, etc.

Notice that the distinction between low-and high-level is evolving because of the continuous trend toward higher-level specification languages. Certain formalisms initially used as high-level models are now considered to be low-level; we can mention, for instance, algebraic data types and Petri nets.

Rather than using models, it is also possible to describe systems and components using *programs* (i.e., executable descriptions written in some *programming language* or in *pseudocode*).

It is generally admitted that programs and models are two distinct concepts. We can mention two essential differences between programs and models:

• Models are, in general, more *abstract* than programs with the consequence that, for a given model, there may exist several alternative programs that (correctly) implement this model.

• Conversely, there may exist several models that, eventually, will be implemented by a single program, each model describing a particular aspect of the program. For instance, there may be different models, written in different modeling languages, to express the functional and non-functional properties of a system.

• Models may express realities that will never be described in programs. For instance, a model of a distributed system may include communication channels that can lose messages and computing nodes that can crash, although no implementation of such channels and nodes will contain explicit program code intended to cause losses or crashes.

Formal methods at large can operate either on models or programs, but the algorithms and methodologies for dealing with models and programs are not the same.

One may wonder whether models are really needed, and whether verification could not be performed directly on programs — with the obvious advantage that programs are closer than models to real systems. This is indeed a tempting approach, but there are also strong arguments supporting the use of models:

• Programs (contrary to higher-level models) are often a “flattened” set of individual components and do not ambition to represent the entire system, its architecture, and its environment. By focusing on programs only, one may lack a global view of design issues.

• Programs are usually more detailed than models and thus may be too complex for verification to be tractable. Moreover, verifying programs may require to consider low-level mechanisms specific to hardware (e.g., semantics of shared variables and locks) or operating system (e.g., synchronization and communication primitives), which increases the overall verification complexity. Furthermore, the frequent absence of architectural/environmental information in programs makes it difficult for verification to exploit the compositional structure of the system.

• Programs are only available during the last steps of system development, whereas models can be produced during the early steps and thus can be used to detect design mistakes as soon as possible. Such mistakes can be extremely costly when discovered lately (e.g., during integration testing or, even worse, after field deployment). Detecting and avoiding such mistakes is a central goal of development methodologies in general and formal methods in particular.

****