SIG on Formal Methods

NEWS LETTER VOLUME 3 , MARCH 2015

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

How are formal methods today?

Compiled by Bhanumathi K S

The vision of the current status of formal methods are given below: **A difficult problem**

Being more ambitious than traditional approaches, formal methods are naturally more complex and their associated tools are also more difficult to build. But there are deeper obstacles inherent to formal methods. These obstacles arise from fundamental results of *computational complexity theory*, which state that, by nature, most interesting verification problems are either impossible or very difficult to solve automatically.

A major obstacle comes from *undecidability* results. In the general case, there is no decision procedure (i.e., algorithm) that can decide whether any given program P may terminate or not (this is known as the *halting problem*). Similarly, there is no decision procedure that can decide whether a given instruction of program P will be actually executed, nor whether P will trigger a run-time error, nor if some given variable X of P will ever become null, etc. All these problems are known to be *undecidable*. Naturally, if a problem is undecidable, it is impossible to build a verification tool that always solves this problem for any system.

Formal verification of the vote-tallying part of the KOA open source software, which was formerly used for remote voting in Dutch public elections. The source code of the software was annotated with JML (Java Modeling Language) and analyzed using the ESC/Java2 [FLL+02] and the Forge checkers, which led to the discovery of specification errors and programming bugs undetected so far.

Formal verification of curved flight collision avoidance maneuvers using the KeYmaera verification tool for hybrid systems [PQ08, PC09a] and detection of an error in a traffic alert and collision avoidance system using the Euclide verification tool.

Formal verification of two operating system microkernels: the seL4 general-purpose commercial microkernel and a German academic microkernel, both verifications being tackled using the Isabelle/HOL theorem prover [NPW02

Formal verification using Coq [BC04] of the CompCert C compiler (front-end and back-end parts) that handles a realistic subset of the C language for critical embedded software.

Proof, using the Astrée static analyzer [BCC+02, BCC+03] based on abstract interpretation, of the absence of any runtime error in several safety-critical C programs of Airbus, namely the primary flight-control software for the A340 fly-by-wire system and, later, the electric flight-control codes for the A380 series [Cou07, DS07, SD07].

Formal proof using the ACL2 theorem prover that the microcode of the Rockwell Collins AAMP7 microprocessor respects a security policy corresponding to a static separation kernel; following this work, the microprocessor received a MILS Certificate from NSA to concurrently process information ranging from Unclassified to Top Secret [Mil08] [Kle09, Section 4.2] [WLBF09, Section 4.4].

Automated analysis of Lucent's CDMA base station call-processing software library (100,000's lines of C/C++ code) using the VeriSoft tool [God97, GHJ98] for systematic statespace exploration, enabling the detection of several critical bugs.

Development of a verification platform (based on static analysis and symbolic model checking) for analyzing the source code of Microsoft Windows drivers
— and more generally any source code written in the C language — so as to check whether the invocations of API (*Application Programming Interfaces*) primitives obey rules for proper use

Formal modeling of the EMV (*Europay-MasterCard-Visa*) protocol suite in the F# language, and automated analysis of these protocols by joint use of the FS2PV translator [BFGT06] and the ProVerif verification tool [Bla04].

****