SIG on Formal Methods

NEWS LETTER VOLUME 8, AUGUST 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

Formal Methods in the Verification & Validation Phase of Software Engineering

Compiled by Dr Manju Nanda

The limitations of testing -based approaches to V & V of safety - critical software are well known; it is impossible to do exhaustive testing, and therefore, to find all bugs through testing. Testing tends to find bugs late in the lifecycle, making changes costly, and testing is poor at finding rare faults even if they have catastrophic consequences. There clearly is a need to change our current V&V techniques and replace them with techniques that can truly increase safety and support innovation while decreasing overall development, maintenance, and evolution cost.

The software intensive systems research area is interested in contributing to the development of new software V&V techniques that can help automate current techniques (which should reduce cost and result in better code coverage), expand the applicability of advanced formal methods to software and design by making them more precise and more scalable, or, apply V&V techniques earlier in the development process, e.g., software design, (thus reducing the cost of fixing bugs and allowing them to target system-level safety properties). Solutions should take into account interactions with other NextGen components such as electronics hardware, other vehicle components such as sensors, actuators and power systems, vehicle dynamics, humans, and operational concepts. They should also have an impact throughout the software life cycle, including development (from design through coding) and maintenance as well as future system evolutions.

Formal methods offer the promise of significant improvements in verification and validation, and may be the only approach capable of demonstrating the absence of undesirable system behavior.

V&V has two principal objectives

The discovery of defects in the software;
The assessment of whether or not the software is useful and useable in an operational situation.

Verification and validation should establish confidence that the software is fit for purpose.

This does NOT mean completely free of defects.
Rather, it must be good enough for its intended use and the type of use will determine the degree of confidence that is needed.

Formal techniques can be used in the following places:

•Requirements policy –For a secure system, these may be the major security properties that must be preserved by the system (called the formal security policy model), such as confidentiality or integrity of data; for a system requiring high dependability, essential properties may include freedom from deadlock.

•Specifications–The formal specification is typically a mathematically based description of system behavior, using state tables or mathematical logic. It will not normally describe lowest level software, such as mathematical subroutine packages or data structure manipulation, but will describe the response of the system to events and inputs to a degree necessary to establish critical properties. Engineering judgment is required to determine the proper level of depth in the specification.

•Proof of correspondence between specification and requirements –It must be shown that the system, as described by the specification, establishes and preserves the properties in the requirements policy. If both are in a formal notation, rigorous proofs can be constructed, either manually or with machine assistance.

•Proof of correspondence between source code and specifications –Although many formal techniques were initially created to provide proof of correctness of code, this is rarely done because of the time and expense involved, but may be done for particularly critical portions of the system.

•Proof of correspondence between machine code and source code – This type of

proof is rare, both because of the expense involved and because modern compilers are very reliable

V&V analysis can be done in three phases :

1.Restate the requirements and conceptual model in a formal (or semi-formal) notation, typically a state table description.
2.Identify and correct ambiguities, conflicts, and inconsistencies.
3.Use a model checker or theorem prover to study system behavior, demonstrate properties, and produce traces of system behavior. Developers, users, and subject matter experts can then use these results to improve the conceptual model

Spectrum of Formal-Methods

Interactive theorem proving : requires great skill and resources
Model checking : analysis is automatic but must specify the model and property
Invisible formal methods : driven directly off model-based developments

Case studies :

The Aviation Safety Program supports the practical application of formal methods to improve safety in commercial aviation.

Rockwell Collins Advanced Technology Center - to develop extensions to existing methods and commercial off -the-shelf tools that enable (1) requirements modeling and analysis, (2) safety analysis and partitioning, (3) mode confusion detection, and (4) auto-generation of code. The project is seeking to develop anautomatic translator from RSML-e to PVS that can be used to verify safety properties and absence of features that have historically lead to mode confusion in flight guidance systems and flight management systems.

•Honeywell Engines and Systems -to develop a fault-tolerant integrated modular avionics architecture for a Full Authority Digital Engine Control (FADEC). This architecture is based on the Time Triggered Architecture (TTA) that has been developed over the past fifteen years at Vienna University of Technology. The challenge of this work is to develop formal proofs of the TTA architecture and to establish a basis for certification based on formal proof.

Model based intrusion detection. One of the most challenging problems in computer security is intrusion detection in networks. Most commercial intrusion detection systems use a “signaure based” approach, which scans network traffic for recognizable segments or hashes of known malicious software (e.g., viruses). The inherent limitation of this approach is that it cannot detect novel attacks for which no signature exists. A number of experimental

intrusion detection systems attempt to solve this problem by defining a formal model of expected system behavior, then checking network traffic for message sequences that deviate from normal.

Access control policy composition. Although formal methods have been applied to analyzing access control policies for more than 30 years, early efforts assumed a single, monolithic system. During the past two decades, a need for analyzing compositions of security policies has been recognized, due to the demands of distributed systems, "virtual enterprise" joint ventures in commerce, and coalition security in military systems. Bonatti, Vimercati, and Samarati [Bonatti et al, 2000] describe an approach to analyzing compositions of complex, independent access control policies. Security policy statements are translated into logic programs, which can then be executed to analyze the effects of policy composition

Secure Electronic Transaction Protocols –SET [Visa, 1997] is a very large collection of protocols designed by Visa and MasterCard to protect the confidentiality and authenticity of electronic commerce transactions. Many parts, though not all, of the SET specification have been subjected to formal verification, revealing several vulnerabilities [Meadows and Syverson, 1998], [Bella et al., 2000], [Bella, Massaci, Paulson, 2001]

The SCR model development tool TTM provides a consistency checker that checks the specification for defects such as type errors, missing cases, circular definitions and other application -independent errors. The TTM tool also provides a dependency graph browser that provides a graphical display of the dependencies among the variables in the specification. Running the SCR security specification model through these two processes helps in developing an internally consistent model and in verifying that all the relevant variables have been taken into account

Airbus uses formal analysis tools to compute the worst case execution time (WCET) and maximum stack usage of executables. For many other requirements, such as dataflow and functional properties, formal verification is only feasible via the source-code representation.

Dassault-Aviation has used formal verification techniques experimentally to replace integration robustness testing, where robustness is defined as "the extent to which software can continue to operate correctly despite abnormal inputs and conditions.

Industry standards also recommend using formal methods in the V&V phase.

Formal method based tools/ techniques in V&V phase:

Model checking tools face a combinatorial blow up of the state-space, commonly known as the state explosion problem, that must be addressed to solve most real-world problems. There are several approaches to combat this problem.

AlPiNA, stands for Algebraic Petri Nets Analyzer and is a model checker for Algebraic Petri Nets.

BLAST
CADP (Construction and Analysis of Distributed Processes) a toolbox for the design of communication protocols and distributed systems
CHESS
CHIC
CPAchecker, an open-source software model checker for C programs, based on the CPA framework
CTML (Computation Tree Measurement Language), a quantitative evaluation tool that covers PCTL and some PLTL that can't be expressed in PCTL.
ECLAIR, a platform for the automatic analysis, verification, testing and transformation of C and C++ programs
FDR2, a model checker for verifying real-time systems modeled and specified as CSP Processes
ISP code level verifier for MPI programs
Java Pathfinder - open source model checker for Java programs
LTSA - Labelled Transition System Analyser
LTSmin - open source model checker for various specification languages (Promela, mCRL2, UPPAAL language)
Markov Reward Model Checker (MRMC)
McErlang, a model checker for Erlang programs which can be distributed and fault-tolerant.
mCRL2 Toolset, Boost Software License, Based on ACP
MoonWalker - open source model checker for .NET programs
NuSMV, a new symbolic model checker
ompca, an interactive symbolic simulator with API control for C/C++ programs with OpenMP directives. The tool is built as an application of REDLIB.
PAT - an enhanced simulator, model checker and refinement checker for concurrent and real-time systems
Prism, a probabilistic symbolic model checker
Rabbit, a model checker for timed and hybrid automata
REDLIB, library for the model-checking of communicating timed automatas with BDD-like diagrams. Applications include a TCTL model-checker with timed fairness quantifications, fair simulation checker, and interactive symbolic simulator for C/C++ programs with OpenMP directives. GUI for model editing and symbolic simulation are also available.
Roméo, an integrated tool environment for modeling, simulation and verification of real-time systems modeled as parametric, time and stopwatch Petri nets
SMART Model checker, Symbolic Model checking Analyzer for Reliability and Timing
SPIN a general tool for verifying the correctness of distributed software models in a rigorous and mostly automated fashion.
Spot a library to implement the automata-theoretic approach for model checking. Has good translation of LTL into Büchi automata and also support the linear fragment of PSL. Must be interfaced with custom code that develop the state-space on-the-fly.
TAPAs: tool for the analysis of process algebra.
TAPAAL, an integrated tool environment for modeling, validation and verification of Timed-Arc Petri Nets
TLA+ model checker by Leslie Lamport

UPPAAL, an integrated tool environment for modeling, validation and verification of real-time systems modeled as networks of timed automata
Vereofy,[14] a software model checker for component-based systems for operational correctness
μCRL, GPL, Based on ACP

Automated theorem proving (also known as ATP or automated deduction) is a subfield of automated reasoning and mathematical logic dealing with proving mathematical theorems by computer programs.

Free software

Alt-Ergo
Automath
CVC
Gödel-machines
iProver
IsaPlanner
KED theorem prover
LCF
LoTREC
MetaPRL
NuPRL
Paradox
Simplify (GPL'ed since 5/2011)
Twelf
SPARK (programming language)

Proprietary software

Acumen RuleManager (commercial product)
ALLIGATOR
CARINE
KIV
Prover Plug-In (commercial proof engine product)
ProverBox
ResearchCyc
Spear modular arithmetic theorem prover

Some of the popular Formal static analysis techniques:

Model checking, considers systems that have finite state or may be reduced to finite state by abstraction;
Data-flow analysis, a lattice-based technique for gathering information about the possible set of values;

Abstract interpretation, to model the effect that every statement has on the state of an abstract machine (i.e., it 'executes' the software based on the mathematical properties of each statement and declaration). This abstract machine over-approximates the behaviours of the system: the abstract system is thus made simpler to analyze, at the expense of *incompleteness* (not every property true of the original system is true of the abstract system). If properly done, though, abstract interpretation is *sound* (every property true of the abstract system can be mapped to a true property of the original system). The Frama-c value analysis plugin and Polyspace heavily rely on abstract interpretation.
Hoare logic, a formal system with a set of logical rules for reasoning rigorously about the correctness of computer programs. There is tool support for some programming languages (e.g., the SPARK programming language (a subset of Ada) and the Java Modeling Language — JML — using ESC/Java and ESC/Java2, Frama-c WP (weakest precondition) plugin for the C language extended with ACSL (ANSI/ISO C Specification Language)).
Symbolic execution, as used to derive mathematical expressions representing the value of mutated variables at particular points in the code.

References:

Cost Effective Use of Formal Methods in Verification and Validation, D. Richard Kuhn Ramaswamy Chandramouli, Ricky W. Butler, http://csrc.nist.gov/groups/SNS/asft/documents/Foundations_2002.pdf

Research opportunities in aeronautics–2011, NASA Research Announcement (NRA): NNH10ZEA001N, OMB Approval Number 2700-0087

http://formalmethods.wikia.com/wiki/Safety-critical_systems

Formal Verification of Software, Bernhard Beckert, UNIVERSITAT KOBLENZ-LANDAU, 2006, http://formal.iti.kit.edu/~beckert/teaching/Verification-SS06/01intro.pdf

Formal Methods used in Software Verification, Robert T. Bauer, IBM/SWG/Rational/Beaverton, http://www.uces.csulb.edu/spin/media/pdf/20041201a_Bauer_LASPIN.pdf

Cost Effective Use of Formal Methods in Verification and Validation, D. Richard Kuhn, Ramaswamy Chandramouli and Ricky W. Butler, http://csrc.nist.gov/groups/SNS/asft/documents/Foundations_2002.pdf