FM'99 in Short

Alexandre David

September 27, 1999

FM'99 was definitive a unique event: never before have so many formal method papers been submitted (259), accepted (92) and presented. FM'99 had the following events: technical symposiums, an exhibition, user group meetings and workshops, industrial tutorials. The workshops had their own publications, which makes the number of papers presented even larger.

Concerning the exhibition, it was a very good opportunity to present the new version of Uppaal. Many visitors seemed to be interested and Paul and I made many demos. No less than 50 pamphlets have been distributed. It was a good opportunity to look at the other tools and compare them. Industrial tools are mature to spread formal methods and academic tools are competitive against them.

A general impression from the different presentations (symposium and workshops) was that formal methods converge. The different tools are not used competitively but rather in conjunction. Much effort has been spent to hide the technical aspects of the tools and methods to spread the tools in industry. Verification pushed from the source code itself (C or Java), or abstraction by experts from a higher level were presented.

The thread of FM'99 was the following points from Tony Hoare:
Maturity: use of a formal method is no longer an adventure, it is becoming routine.
Convergence: the choice of a formal method or tool is no longer controversial: they are chosen in relation to their purpose and they are increasingly used in effective combination.
Cumulative progress: promise of yet, further benefit is obtained by accumulation of tools, libraries, theories, and case studies based on the work of scientists from many schools which were earlier considered as competitors."

Following are remarks on some interesting presentations.

Error Detection with Directed Symbolic Model Checking

The goal is to explore as little state space as possible to find errors, with contrast to safety checking where the whole state space must be generated. An heuristic is presented to direct the search and to find errors faster.

This distinction between proofs and error detection was underlined in several presentations.

A GSM-MAP Protocol Experiment Using Passive Testing

This is about conformance testing: does a protocol implement some specification? The idea is to observe traces and test if they are accepted by the specifications.

Unfortunately, the coverage aspect was missing.

Model-Checking for Managers[1]

This is about using the SPIN model checker in industry. A remarkable point is to use patterns for the requirements to hide LTL formulas: patterns in plain English are configurable and mapped to LTL formulas without any prior knowledge of LTL.

[1] SPIN workshop

Xspin/Project-Integrated Validation Management for Xspin[2]

This is an integrated environment for the model checker SPIN. Distinction between debugging (weak abstraction, partial check: bit state hashing) and verification (strong abstraction, exhaustive) was stressed. The tool offers a Project Revision Control System similar to CVS with a repository which forces version integrity by reverification when changing a model.

PVS workshop - Amir Pnueli

This is in short deductive verification versus model checking. The message is deductive verification technology for hardware design is mature, and cumulative effort on verification is reusable making it worth the price.

Deduction is Forever - Amir Pnueli

This is the continuation of his previous talk. The fact that only deduction is symbolic is stressed and exploration is limited by the finite state systems. However there is a duality between exploration and deduction which makes it interesting.

The conclusion is the necessary conditions for the ultimate tool:

- to focus on the application area
- all verification conditions should be algorithmic resolvable (no interactive theorem proving)
- invalid verification conditions must yield usable and focused counter examples
- need for inginuity and cleverness, experts: the central issues in a model, system engineers: rule out bad counter examples

Mechanized Formal Methods: Where Next? - John Rushby

The reason for poor usage of formal methods is: "Bill Gates does not use our stuff"[3]. The reason is that correctness is often not a goal: a program must be just good enough. Coding is not a problem, the real problem is: requirements and design. Formal methods have their place at the requirement analysis and design exploration.

Mechanized formal method is the **only** important thing. It should calculate answer to a practical question. Combining theories yields to undecidable theories or to specialized theories where it is really fast to compute an answer: it is there that we should take advantage of combining methods.

The problem of PVS is that it is oriented to verification, not debugging and people are not prepared to this in contrast to model checking where people are prepared to reduce a problem to fit into a model checker because *it looks like design*. The proposal is to focus on construction of abstractions and invariants. The goal is to calculate an abstraction and to generate an invariant by modifying a model checker which would return a formula to feed the theorem prover.

The conclusion is: one needs powerful and well focused tools which really address the problems.

Extracting Verification Models from Source Code[4]

This was done in an industrial project while building a telephone switcher at Bells. The product is named "PathStar". The C source follows a `@` notation like labels. Automata are extracted from the core of the switcher and fed to SPIN. A database of properties is maintained with the results, the fixes, the traces... The check is nearly interactive: done on special hardware (parallel machine dedicated to SPIN), parallel verification of properties, optimization of models

[2] SPIN workshop
[3] John Rushby's words
[4] SPIN workshop

per properties, bitstate hashing for incremental verification. The equivalent processing power is a 8GHz PII and it will be upgraded.

The system to manage the verifications is called "Trailblazer" and it separates the user's side (web browser with updated database pages) from the database itself ("FeaVer").

Applying Model Checking in Java Verification[5]

Use the java PathFinder to have a Java-to-Promela translation. Application to a game server is presented. Conclusion is: program verification is worth attention.

IF: An Intermediate Representation and Validation Environment

IF is a structured program level intermediate format. The point is: translations $IF <-> Kronos, SDL-> IF, IF <-> SPIN$ and a static analyser from IF to improve the program. As we can see we have now $SDL-> IF-> Kronos$ or $SPIN$, which is SDL (design) to verification tools!

Last Words

We have to hide the formalism and formal methods should be fun to use like programming languages.

[5] SPIN workshop