Research and Application of Formal Methods

dr.ir. Loe M.G. Feijs

Dr.ir. Loe Feijs of Philips Research Laboratories has been working for many years in the area of formal specification methods. He is part of the team that developed COLD*, a wide-spectrum algebraic specification language. In this article he expresses his views on the pro's and con's of formal specification methods and his experience with the practical application of those methods within Philips' Product Divisions.*

What are formal methods?

Computing science is concerned with information processing systems, with designing these, and with the theories and principles that describe such systems. In computing science, when viewed as an engineering discipline, the issue is to apply the theory and the fundamental insights to make the design process of information processing systems happen in an orderly way.

The term formal methods refers to a number of languages and calculi, but at the same time it refers to a way of designing in which these languages and calculi play a role. The term 'formal' means 'according to the form', and is usually encountered in the context of legal and administrative matters. In such matters the term often has a negative interpretation: one could think of trials where someone guilty goes free because of procedural mistakes by an officer, or where loopholes can be found to circumvent the law. But also the advantages of the formal framework for legal and administrative matters are clear: written laws, in precise language, make it possible to avoid arbitrariness and too much subjectivity. It is not the case that all actions either before or afterwards are checked agains all laws and rules. Precisely the fact that in principle one could go to the bottom in case of a conflict or a breach of the law, makes that in most cases the conflict or breach of law does not occur in the first place.

Returning to the subject of computing science, consider the following header which could be part of the description of a software package containing arithmetical operations, written in the programming language C. Suppose that the designer of the division function has a clear mental image of the operation of `DIVIDE` and therefore does not add any further explanation, except for maybe a comment telling that it performs 'division'.

```
#define Nat int
Nat z,y,x;
void DIVIDE();
```

In the meantime other designers could start using this function. One designer feels that the first two variables `z` en `y`, provide the input for `DIVIDE` and that the input variables should not be changed. He will call the division function with saving `z` and `y` first; he does not know that these are also intended for the storage of intermediate results of the division. Yet another designer thinks that `x`, `y`, en `z` are the first input, the second input and the result, respectively. Moreover he considers `Nat` as a denotation of the natural numbers 0, 1, 2 etc. and therefore the division should work too if `y` equals zero. In his opinion, `z` should get the value zero in that case.

It will be clear that in the example sketched above the designers work according to an informal approach, i.e. not formal. Of course everybody will say that the header of `DIVIDE` was stupid in the first place, and moreover that with a little bit of communication amongst the designers it is easy to eliminate all misinterpretations. It also looks reasonable to assume that problems concerning the division function will be found when the larger system encompassing it will be tested. Some care is in order however: experience learns that when the systems grow larger and more complex, it turns out harder to have them well-tested.

Specification is important, but is quite non-trivial. It requires some experience with some techniques and it should be practiced. Special languages have been developed, such as VDM, Z, COLD, and LARCH. These languages have powerful expres-

sion means, which mostly have been taken from the field of mathematical logic. Using these, or other specification techniques, all unintended ambiguities can be eliminated from the specification of DIVIDE. Moreover such languages offer support to structure both the specifications themselves and the collection of domain-specific concepts (application-domain modeling). Just as is the case for a modular or an object-oriented programming language there are structuring concepts in the language. This seems superfluous when dealing with small examples, but it is essential for the large systems that happen to be made in practice. This style of specification is known as 'formal specification'. Instead of relying on intuition and implicit assumptions, agreements are made, and design decisions are taken on the basis of explicit descriptions of systems and subsystems.

It is also possible to go one or two steps further by not only approaching the specifications formally, but also the reasonings on the basis of which correct programs are distinguished from wrong programs. A formal system of reasoning is often called a 'calculus'. Using this, one can give correctness proofs for programs. This is a powerful technique, for as Prof. Dijkstra has pointed out already years ago: testing can demonstrate the presence of an error, but not its absence. The summum of formal verification is to use proofs which are detailed enough to have a computer (as a machine formal, of course) check for the correctness of the proof in a mathematical sense.

Before concluding that this ultimate option is to be chosen always, it is necessary to mention that with present day techniques and tools the construction of these machine-readable proofs is an extremely laborious task. Formal specifications however are already usable in an efficient way for designing real systems. With respect to applicability, the usage of calculi is in an intermediate position. In the remainder of this article the focus is on formal specification techniques.

Considerations in favor of formal methods

It is hard to understand that in computing science one cannot do what seems to succeed much better in other engineering disciplines: making a distinction between specification and implementation. Why do we accept a programmer who is not able to explain very precisely what his or her program will do, but instead starts programming on the basis of incomplete and ambiguous specifications? Of course this would be understandable if information processing systems would be fast and cheap to build. Although this is certainly the case for small tools for personal use, this does not hold for systems developed by a group of people over a longer period of time. In many companies the invested capital in software amounts to tens or hundreds of millions guilders -- so this is hardly to be called 'cheap'. When developing electrical, mechanical and electronic systems where a programmed computer is an essential part of the system ('embedded software'), the development of the programs often takes longer than the development of the rest of the system. If delivery time or timely introduction in the market are important, then program development is not fast any more -- at least, not fast enough.

Of course there are business areas where good work is done without formal methods; for example the movie-business, where the subjective element is very essential. But in computing science we are dealing with computers, and these are formal machines par excellence. If we think of system development by means of algorithmic and logic programming languages, then the development must always lead to a program which prescribes the steps to be followed by the computer in a very precise way. In this context, ambiguities, misunderstandings and implicit assumptions can -- enforcing each other -- easily lead to ununderstandable system behavior, bugs which are hard to eliminate, and of course as a consequence delay in the development projects.

Although good specification is not exactly easy, it does not require very rare capabilities. It can be learned. Some knowledge of logic is indispensable and also the usage of set theory is needed. The well-known Boolean algebra, also called proposition logic, is useful but not sufficient. Many developers of information processing systems have there head filled with facts and tricks concerning the commands of the operating system they use, names and parameter lists of the many functions they and their colleagues have made themselves, etc. Although remembering all this information is a difficult task, the value of all this knowledge is very local and very temporarily. Formal specification techniques however do not require much factual knowledge, but they require experience in the use of logic, in structuring facts and tricks to

orderly concepts, and most importantly, experience in writing these things down. It is clear that the inability to write down component specifications and system specifications is an obstacle for designing fast and efficiently. This pleads in favor of more education and training with formal methods than is usual at present.

Considerations against formal methods

One frequently mentioned objection against formal methods is that they are difficult and that they are too mathematical. There is much to be said about this. First of all that there is a kernel of truth in it. The usage of compact formula notation instead of natural language certainly is an essential ingredient of formal methods, and indeed the powerful expressive means call most for experience and for abstraction. It cannot be denied that the propagators of formal specification techniques themselves, sometimes caught by enthusiasm about recently gained insights, pour out more mathematics over their audience than needed. Finally, the specification languages of today are not perfect. They could not be, in view of the fact that the entire field of computing science is still much in development, and also in view of the fact that the effort which has been invested in the development of formal methods is small when compared with the development of for example programming languages (which are not perfect either).

Experience learns that writing a good specification takes time, sometimes even much time. The amount of time needed is of the same order of magnitude as the amount of time needed to make the corresponding program (or the time to test it). In any case, this time must be paid and planned and this calls for a rational cost-benefit analysis to decide whether or not to employ formal specification (or specification in general).

There are paradoxical dilemmas when deciding upon the right moment to start specifying a component or a system. When starting with a specification first, it is difficult in the early and abstract phase to have a good understanding of the efficiency problems with respect to the usage of scarce resources which are to be expected (scarce resources include execution time, memory usage, etc.). Nevertheless, efficiency is of central importance: how easy would many systems be to make if efficiency didn't count. When the specification is postponed until *after* the implementation, it requires an enormous discipline to make the specification yet. This is not an easy dilemma, but maybe the solution lies in the education of people who have a lot of experience in both specifying and implementing and who are able to see both expects in their mutual dependency.

Some care is in order with respect to the idea that *everything* should be specified formally. A team can get caught in a spiral of first making system specifications, component specifications, striving for a high degree of perfection with respect to form and structure of those specifications, factoring out common parts, etc. and where the usage of correctness proofs is an obvious next step. Of course these are honorable initiatives, but if it is forgotten (in each specific situation) to make a rational cost-benefit analysis, it is not strange when things go wrong from a business point of view.

Experiences in Philips

The experiences, gathered in the course of 10 years at Philips have turned the author into a convinced proponent of the use of formal specification techniques.

At Philips Research Laboratories in Eindhoven work on formal specification techniques has been done since the early 1980s. In the early years, work was done on language definition and fundamentals. In this period there was a fruitful cooperation with university specialists in mathematical logic, in particular with G. Renardel de Lavalette en C. Koymans. In this period COLD was developed, a kind of answer upon the VDM of Bjoerner and Jones, and broader with respect to its scope than the then existing algebraic specification languages. The chief designer of COLD is Dr. ir. Hans Jonkers of Philips Research.

At the end of the 1980 the research has taken a new direction, shifting focus from fundamentals to tools and applications. We have had the opportunity to experiment with the application of formal specification techniques in various product divisions of Philips, including Medical Systems, Data Systems, Image Tubes (calculations for shadow masks), Industrial Electronics (off-line scheduling and control of component-placement machines) and Consumer Electronics.

At present parts of TV sets are formally specified

according to a method called SPRINT. In particular these are the parts which are suitable candidates for reuse in different variants of the TV sets. For these parts, called audio/video components it is worthwhile to make significant investments in their specification. These components are addressed twice: they are explained formally, using logic, preconditions, and postconditions; they are implemented in a normal programming language. Other parts of the system are developed by means of COLD and a special compiler which generates code from the specification. Of course not all the powerful expressions from a specification language can be translated efficiently, but because of suitably (with respect to efficiency) chosen restrictions, it was even possible to meet high efficiency requirements. The executable subset of COLD is called PROTOCOLD.

In consumer electronics, products must be sold in large numbers and for attractive prices. An important role is played by efficiency requirements. During the development attention must be paid to the costs of the ROM, the RAM, and the processor. This is not a standard application of formal methods, trying to do everything as formal as possible; where the investment pays off, careful and formal specification is done, but where flexibility and costs are more important, a compromise is chosen, sacrificing abstractness of specification in favor of fast experimentation and efficiency.

I expect that in the future formal specification techniques will find increasingly more applications. But I also expect that these techniques will become more integrated into programming languages, graphical formalisms, and tools. I expect that after some time one does not talk anymore about formal methods, but that they will just be used. Moreover, many of the information processing systems which are being developed now are open systems, fitting into standardized user-interface frameworks (window systems), operating in communication networks (mobile phone, paging, e-mail, various new services) and exploiting multimedia technology. This will give rise to complex and long-living interfaces and standards which must be well-specified. □

Dr.ir. Loe M.G. Feijs works at the department Information and Software Technology of Philips Research Laboratories in Eindhoven.

Short News

Matrix

Eindhoven University of Technology (EUT) started the publication of a new three-monthly magazine called 'Matrix'. It is intended for all the relations of the university. Target groups are people in business and industry, politicians, and EUT graduates. It contains information about research and educational programs which take place at the university.

Currently 'Matrix' is sent to all those graduates which are member of a graduates association (such as XOOTIC). Everyone else who wishes to receive this magazine, can contact the editors: Lucy Holl, EUT, In- and External Relations, tel. +31 40 473330.

New IVO promotion video

The Institute for Ongoing Education (IVO) has made a second promotion video for the post-masters programmes in technological design of EUT. Whereas the first video was targeted at new students, this one is targeted at companies who should hire graduates. The title of the film is ''Technologic Designers - People that will make it''.

Two graduates play a role in this video: Coen van Beek, graduate of the course Computer-aided Design and Manufacture of Discrete Products, and Erik Jan Marinissen, graduate of OOTI.

The premiere of the video will take place during the IVO Information Day at April 8th. The video will be shown on many occasions and locations later on.