Abstract

I. Formal Methods Explained

A. Formal methods refers to mathematically based techniques for the specification, development and verification of software and hardware systems. The approach is especially important in high-integrity systems, for example where safety or security is important, to help ensure that errors are not introduced into the development process. Formal methods are particularly effective early in development at the requirements and specification levels, but can be used for a completely formal development of an implementation (e.g., a program).

B. Taxonomy
1. Level 0: Formal specification may be undertaken and then a program developed from this informally. This has been dubbed formal methods lite. This may be the most cost-effective option in many cases.

2. Level 1: Formal development and verification may be used to produce a program in a more formal manner. For example, proofs of properties or refinement from the specification to a program may be undertaken. This may be most appropriate in high-integrity systems involving safety or security.

3. Level 2: Theorem provers may be used to undertake fully formal machine-checked proofs. This can be very expensive and is only practically worthwhile if the cost of mistakes is extremely high (e.g., in critical parts of microprocessor design).

4. Denotational semantics, in which the meaning of a system is expressed in the mathematical theory of domains. Proponents of such methods rely on the well-understood nature of domains to give meaning to the system; critics point out that not every system may be intuitively or naturally viewed as a function.

5. Operational semantics, in which the meaning of a system is expressed as a sequence of actions of a (presumably) simpler computational model. Proponents of such methods point to the simplicity of their models as a means to expressive clarity; critics counter that the problem of semantics has just been delayed (who defines the semantics of the simpler model?).

6. Axiomatic semantics, in which the meaning of the system is expressed in terms of preconditions and postconditions which are true before and after the system performs a task, respectively. Proponents note the connection to classical logic; critics note that such semantics never really describe what a system does (merely what is true before and afterwards).

C. Lightweight versus Heavyweight
Some practitioners believe that the formal methods community has overemphasized full formalization of a specification or design (Jackson and Wing 1996), (George and Vaughn 2003). They contend that the expressiveness of the languages involved, as well as the complexity of the systems being modelled, make full formalization a difficult and expensive task. As an alternative, various lightweight formal methods, which emphasize partial specification and focused application, have been proposed. Examples of this lightweight approach to formal methods include the Alloy object modelling notation (Jackson 2002), Denney’s synthesis of some aspects of the Z notation with use case driven development (Denney 2005), and the IFAD VDM toolset (Agerholm and Larsen 1998).

D. Uses

1. Specification

Formal methods may be used to give a description of the system to be developed, at whatever level(s) of detail desired. This formal description can be used to guide further development activities (see following sections); additionally, it can be used to verify that the requirements for the system being developed have been completely and accurately specified.

The need for formal specification systems has been noted for years. In the ALGOL 60 Report, John Backus presented a formal notation for describing programming language syntax (later named Backus normal form or Backus-Naur form (BNF)); Backus also described the need for a notation for describing programming language semantics. The report promised that a new notation, as definitive as BNF, would appear in the near future; it never appeared.

2. Development

Once a formal specification has been developed, the specification may be used as a guide while the concrete system is developed (i.e. realized in software
and/or hardware). Examples:

- If the formal specification is in an operational semantics, the observed behavior of the concrete system can be compared with the behavior of the specification (which itself should be executable or simulateable). Additionally, the operational commands of the specification may be amenable to direct translation into executable code.

- If the formal specification is in an axiomatic semantics, the preconditions and postconditions of the specification may become assertions in the executable code.

E. Verification

Verification

Once a formal specification has been developed, the specification may be used as the basis for proving properties of the specification (and hopefully by inference the developed system). [edit]

Human-Directed Proof

Sometimes, the motivation for proving the correctness of a system is not the obvious need for re-assurance of the correctness of the system, but a desire to understand the system better. Consequently, some proofs of correctness are produced in the style of mathematical proof: handwritten (or typeset) using natural language, using a level of informality common to such proofs. A "good" proof is one which is readable and understandable by other human readers.

Critics of such approaches point out that the ambiguity inherent in natural language allows errors to be undetected in such proofs; often, subtle errors can be present in the low-level details typically overlooked by such proofs. Additionally, the work involved in producing such a good proof requires a high level of mathematical sophistication and expertise. [edit]

Automated Proof
In contrast, there is increasing interest in producing proofs of correctness of such systems by automated means. Automated techniques fall into two general categories:

* Automated theorem proving, in which a system attempts to produce a formal proof from scratch, given a description of the system, a set of logical axioms, and a set of inference rules. * Model checking, in which a system verifies certain properties by means of an exhaustive search of all possible states that a system could enter during its execution.

Neither of these techniques work without human assistance. Automated theorem provers usually require guidance as to which properties are "interesting" enough to pursue; model checkers can quickly get bogged down in checking millions of uninteresting states if not given a sufficiently abstract model.

Proponents of such systems argue that the results have greater mathematical certainty than human-produced proofs, since all the tedious details have been algorithmically verified. The training required to use such systems is also less than that required to produce good mathematical proofs by hand, making the techniques accessible to a wider variety of practitioners.

Critics note that such systems are like oracles: they make a pronouncement of truth, yet give no explanation of that truth. There is also the problem of "verifying the verifier"; if the program which aids in the verification is itself unproven, there may be reason to doubt the soundness of the produced results. [edit]

II. Criticism of Formal Methods

Criticisms

In addition to the internal criticisms mentioned above, the field of formal methods as a whole has its critics. At the current state of the art, proofs of correctness, whether handwritten or computer-assisted, need significant time (and thus money) to produce,
with limited utility other than assuring correctness. This makes formal methods more likely to be used in fields where the benefits of having such proofs, or the danger in having undetected errors, makes them worth the resources. Example: in aerospace engineering, undetected errors may cause death, so formal methods are more popular than in other application areas.

At times, proponents of formal methods have claimed that their techniques would be the silver bullet to the software crisis. Of course, it is widely believed that there is no silver bullet for software development, and some have written off formal methods due to those overstated, overreaching claims. Y Formal methods and notations

There are a variety of formal methods and notations available, including

* Abstract State Machines (ASMs) * Alloy * B-Method * Process calculi o CSP o ?-calculus * Actor model * Esterel * Lustre * Petri nets * RAISE * VDM * Z notation

[edit]

See also

* Automated theorem proving * Model checking * Software engineering * Software engineering disasters * Formal methods people

III. Integrating the Problem Formally

IV. The System

V. Conclusions