Verification by model checking

Motivation for verification

Formal verification methods have quite recently become usable by industry and there is a growing demand for professionals able to apply them. Formal verification techniques can be thought of as comprising three parts:

a framework for modelling systems, typically a description language of some sort;

a specification language for describing the properties to be verified;

a verification method to establish whether the description of a system satisfies the specification.

Approaches to verification can be classified according to the following criteria:

Proof-based vs. model-based.

In a proof-based approach, the system description is a set of formulas Γ (in a suitable logic) and the specification is another formula φ. The verification method consists of trying to find a proof that $\Gamma \vdash \varphi$. This typically requires guidance and expertise from the user. In a model-based approach, the system is represented by a model M for an appropriate logic. The specification is again represented by a formula φ and the verification method consists of computing whether a model M satisfies φ (written $M \models \varphi$). This computation is usually automatic for finite models.

Degree of automation.

Approaches differ on how automatic the method is; the extremes are fully automatic and fully manual. Many of the computer-assisted techniques are somewhere in the middle.

Full- vs. property-verification.

The specification may describe a single property of the system, or it may describe its full behaviour. The latter is typically expensive to verify.

Intended domain of application

which may be hardware or software; sequential or concurrent; reactive or terminating; etc. A reactive system is one which reacts to its environment and is not meant to terminate (e.g., operating systems, embedded systems and computer hardware).

Pre- vs. post-development.

Verification is of greater advantage if introduced early in the course of system development, because errors caught earlier in the production cycle are less costly to rectify. (It is alleged that Intel lost millions of dollars by releasing their Pentium chip with the FDIV error.)

Model checking is based on temporal logic. The idea of temporal logic is that a formula is not statically true or false in a model, as it is in propositional and predicate logic. Instead, the models of temporal logic contain several states and a formula can be true in some states and false in others. Thus, the static notion of truth is replaced by a dynamic one, in which the formulas may change their truth values as the system evolves from state to state. In model checking, the models M are transition systems and the properties φ are formulas in temporal logic. To verify that a system satisfies a property, we must do three things:

- model the system using the description language of a model checker, arriving at a model M;
- code the property using the specification language of the model checker, resulting in a temporal logic formula φ;
- Run the model checker with inputs M and φ.

Temporal logics have a dynamic aspect to them, since the truth of a formula is not fixed in a model, as it is in predicate or propositional logic, but depends on the time-point inside the model called Linear-time Temporal Logic (LTL), and another where time is branching, namely Computation Tree Logic (CTL). These logics have proven to be extremely fruitful in verifying hardware and communication protocols; and people are beginning to apply them to the verification of software.