Abstractions of Abstract Data Types

Ferucio Laurenţiu ŢIPLEA

Faculty of Computer Science
"Al.I.Cuza" University of Iaşi
6600 Iaşi, Romania
E-mail: `fltiplea@mail.dntis.ro`

Abstract

We report work in progress on a very general and powerful abstraction technique. It works on abstract data types and can be specified by equations. Property preservation results are obtained, which generalize similar results known for predicate abstraction, shape analysis, the duplicating predicate symbols technique etc.

Introduction. In the last decade a lot of progress has been made in the development of automated verification tools for formal verification, such as PVS, HOL, STEP, COSPAN, MURPHI, SMV, and SPIN. They have been successfully applied to analyze the correctness of a large variety of reactive systems, ranging from circuit designs to communication protocols. Although verification tools as those given above are currently in use by companies like Intel, Motorola, Siemens etc., industrial-size systems are still further to be satisfactorily checked. Traditional software testing is already a notoriously hard, time-consuming and expensive process, and stochastic systems, security protocols and web site testing present even greater challenges. All these show that additional research is needed to develop a number of new techniques that should enable larger hardware systems and certain types of software systems such as security protocols and probabilistic programs to be verified. Two such techniques, generally recognized as the only methods can ever scale up to handle industrial-size design and verification, are the *abstraction* and *modularization* which break the task of verifying a large system into several smaller tasks of verifying simpler systems.

Abstraction. Abstraction techniques, often based on abstract interpretation [2], provide a method for symbolically executing systems using the abstract instead the concrete domain. Familiar data-flow analysis algorithms are examples of abstract interpretation. In particular, abstract interpretation can be used for building the abstract state-space of the system. The abstraction is provided by the user and need not be dependent on the choice of properties to verify.

Many abstraction techniques have been proposed in the literature and many of them deal with *data type reduction*, that is the reduction of a large (possible unbounded) data type to a small one. Prominent examples include:

- *Shape analysis*, which is a data flow analysis technique [5]. It is used mainly for complex analysis of dynamically allocated data, and is based on representing the set of possible memory states ("stores") that arise at a given point in the program by *shape graphs*. In such a graph, heap cells are represented by shape-graph nodes and, in particular, sets on "indistinguishable" heap cells are represented by a single shape-graph node. In the past two decades, many shape-analysis algorithms have been developed. The parametric framework for shape analysis introduced in [6] covers almost all of the work done.

- *Predicate abstraction*, also called *boolean abstraction* or *existential abstraction*. This technique has been introduced by Graf and Saidi in [3] to provide a method for the automatic construction of an abstract state graph of an infinite system using the PVS theorem prover. The main idea of predicate abstraction is to map concrete objects (states of a transition system, data of a data type etc.) to "abstract objects" according to their evaluation under a finite set of predicates.

- The technique of *duplicating predicate symbols*, which is one of the intensively used technique in abstraction [1]. It is based on associating "copies" to each predicate symbol. Therefore, a formula φ gets several versions depending on the predicate copies are used. Usually, two copies for each predicate symbol are associated, and two versions of a formula are used: one of them for validation, and the other one for refutation.

In order to be useful from a practical point of view, a data type reduction (abstraction) should be *conservative* or *property-preserving* w.r.t. a specific set of properties. Three forms of property preservation are mostly studied in the literature: *weak*, *strong*, and *error preservation.* An abstraction is an weak (strong, error) preserving abstraction if a set of properties that are true (with truth-values true or false, false) in the abstract system has corresponding properties in the concrete system that are also true (with the same truth-values, also false).

Abstractions of ADTs. In order to address the problem of verifying or analyzing a particular program that uses large data structures we need a rigorous model for data types. Such a model can be obtained by using universal algebras, a widely accepted formalism for specifying data types. A data type defined by such a formalism, also called an *abstract data type* (ADT, for short), is a class of algebras of the same signature that is closed under isomorphism [1].

In our approach, we enrich signatures with logical symbols used to build formulas. Answers to questions about properties of data structures are obtained by evaluating such formulas. Then, an abstraction of an ADT is a congruence on the free algebra of the *initial specification* of that ADT. We distinguish between several types of abstractions and we get property preservation results for each case. As a conclusion we are able to generalize many abstraction techniques known from the literature, such as predicate abstraction, shape analysis, the technique of duplicating predicate symbols, the McMillan's approach [4] etc.

References

[1] E. Clarke, O. Grumberg, D.E. Long. *Model Checking and Abstraction*, ACM Transactions on Programming Languages and Systems, 1994, 1512–1542.

[2] P. Cousot, R. Cousot. *Abstract Interpretation: a Unified Lattice Model for Static Analysis of Programs by Construction or Approximation of Fixpoints*, in the 4th ACM Symposium on Principles of Programming Languages, 1977, 238–252.

[3] S. Graf, H. Saidi. *Construction of Abstract State Graphs with PVS*, Proceedings of the 9th International Conference on Computer Aided Verification, Lecture Notes in Computer Science 1254, 1997, 72–83.

[4] Kenneth McMillan. *Verification of Infinite State Systems by Compositional Model Checking*, Research Report, Cadence Berkeley Labs, 1999.

[5] F. Nielson, H.R. Nielson, Ch. Hankin. *Principles of Program Analysis*, Springer-Verlag, 1999.

[6] M. Sagiv, Th. Reps, R. Wilhelm. *Parametric Shape Analysis via 3-Valued Logic*, ACM Transaction on Programming Languages and Systems, 22, 2001.

[1]Informally, the closer under isomorphism corresponds to the fact that isomorphic algebras are "similar" in that they differ only by the nature of their carriers.