Domain Pattern Abstraction + Ptolemaic Abstract Domains = Environment Abstraction for Concurrent Systems

Murali Talupur[1] and Helmut Veith[2]

[1] Strategic CAD Labs, Intel Corporation, Portland, USA
[2] Formal Methods in Systems Engineering, FB Informatik, TU Darmstadt, Germany

1 Model Checking Systems with Replicated Processes

With the rapid onset of the multi-core era, the verification of multi-threaded systems and concurrent algorithms has become a pressing problem in the hardware and software industries. While traditional techniques like testing and simulation are often adequate for sequential software and hardware, they are not suited for validating concurrent systems; due to their their massive parallelism, concurrent systems have way too many possible interleavings for these informal techniques. Therefore, concurrent systems should be verified formally using techniques like model checking or theorem proving.

In this talk, we discuss environment abstraction [13, 4, 5], a novel model checking based approach for the verification of concurrent software. Environment abstraction is designed for systems with replicated processes, i.e., systems where the same process/algorithm is executed by multiple agents concurrently. Such systems often form the basic building blocks of larger systems, and tend to be combinatorially intricate; many of them, for instance cache coherence protocols, are also very large. At design time, the number of concurrent processes is unknown, and thus we speak of *parameterized verification*. We have applied environment abstraction successfully to a broad class of systems including cache coherence protocols and mutual exclusion protocols, and are currently working on environment abstraction for time triggered protocols.

2 Two Pillars of Environment Abstraction

Environment abstraction is based on two insights that allow to us build a uniform framework for a wide class of concurrent algorithms, namely, Ptolemaic abstraction and the use of domain-specific patterns which occur in real life protocols.

1. **Ptolemaic Abstraction.** When humans reason about complex systems, they tend to view themselves in the center of the system. While this Ptolemaic viewpoint is usually not adequate for systems that arise in nature, such as the Solar system, it is often implicit in systems *engineered* by humans. In particular, experience shows that distributed systems are often designed in such a way that the correctness of the algorithm can be assessed from the viewpoint of an individual process and its interference with the rest of the system. This insight naturally leads us to abstract

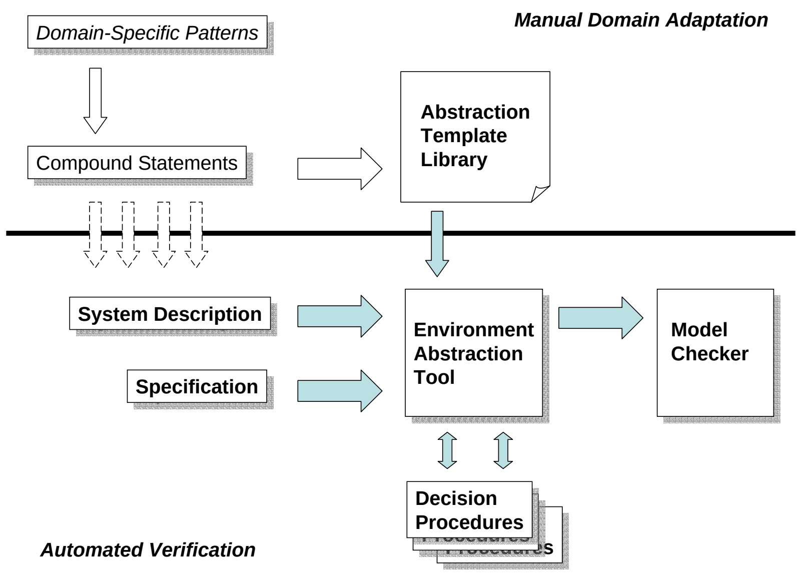

Fig. 1. The Environment Abstraction Tool Chain

domains which capture detailed information on one reference process and retain abstract information on the environment. Our case studies demonstrate that abstract domains for important classes of protocols (e.g. cache coherence protocols) can be constructed with little human effort.

2. **Domain Patterns.** To deal with the hard task of constructing the abstract model we make another crucial observation: Although the class of distributed protocols is rich and varied, we can confine our attention to a single class, like say the class of cache coherence protocols, and find that all natural protocols in that class can be expressed in terms of a few basic *domain patterns*. For each class of protocols, we can create a small number of compound statements that embody these basic domain patterns. Given that we know what the abstract states are, we manually supply an abstract *template transition invariant* for each compound statement. This transition invariant expresses the action of the compound statement in terms of the abstract domain. Given any protocol in the class, constructing the abstract model then reduces to instantiating the abstract template invariants appropriately with the actual parametrization corresponding to the statements in the protocol. Thus, for each class of protocols, we have a library containing compound statements and the corresponding abstract invariants.

Figure 1 illustrates the conceptual framework of environment abstraction. In order to adapt environment abstraction to a new class of protocols, we first need to analyze the domain and to define compound statements along with the appropriate abstract template invariants. From this point on, we can automatically verify protocols in the given class. We have used this method to verify protocols from cache coherence protocols to mutual exclusion protocols to semaphore based protocols. Some of the protocols we have verified, such as the simplified version of Flash protocol, are much larger than the protocols handled by other automatic verification methods. We are currently working on a fully configurable environment abstraction tool which supports easy adaption to new application domains, cf. Section 4.

3 Related Work

Several techniques have been proposed for protocol verification [11, 6, 7, 1, 10] and verification of concurrent software [8, 14]. Although these communities are addressing very similar issues, they have tended to work separately. While concurrent software tends to handle examples that are less combinatorially intricate, they have to deal with additional complications such as pointers. The protocol verification community deals with combinatorially intricate algorithms that don't use higher level language features such as pointers. One of the purposes of our work and this talk is to help bridge the gap between the two communities.

Within protocol verification, several approaches have been proposed to handle protocols with an unbounded number of agents (the so called parameterized verification problem). While these techniques have been successful for small protocols, like Bakery and Szymanski, they do not scale well to large protocols like the Flash cache protocol.

On the concurrent software verification side, TVLA [12, 14] has been used to verify different types of concurrent software from the Java Standard library. Recently, there have been attempts to extend this approach to handle linearizability properties of systems with unbounded number of threads [2], but the scalability of this method to large systems remains to be proven.

4 Status and Future Challenges

While the successful use of environment abstraction for a wide range of protocols is a witness to the applicability and scalability of the method, there are still several challenges which we are addressing in current and future work, in particular counterexample-guided abstraction refinement (CEGAR) [3] in combination with lazy abstraction [9] to further improve scalability, as well as specific support for liveness properties.

To fully realize the framework of Figure 1, we are currently developing a fully configurable environment abstraction tool where compound statements and their associated abstractions can be easily defined and adjusted for different application areas, and integrated with suitable decision procedures for the extraction of abstract models. Among other features, we are introducing convenient support for the data structures used in protocols such as message queues and pointer sets. This core framework will be oblivious to the exact class of protocols under consideration, and serve as a basis for future work.

References

1. T. Arons, A. Pnueli, S. Ruah, and L. Zuck. Parameterized Verification with Automatically Computed Inductive Assertions. In *Proc. CAV*, 2001.
2. J. Berdine, T. Lev-Ami, R. Manevich, G. Ramamlingam, and M. Sagiv. Thread quantification for concurrent shape analysis. In *Proc. CAV*, 2008. to appear.
3. E. Clarke, O. Grumberg, S. Jha, Y. Lu, and H. Veith. Counterexample-guided Abstraction Refinement. In *Twelfth Conference on Computer Aided Verification (CAV'00)*. Springer-Verlag, 200.
4. E. Clarke, M. Talupur, and H. Veith. Environment Abstraction for Parameterized Verification. In *Proc. VMCAI*, 2006.
5. E. Clarke, M. Talupur, and H. Veith. Proving Ptolemy Right: Environment Abstraction Principle for Parameterized Verification. In *Proc. TACAS*, 2008.
6. A. Emerson and V. Kahlon. Parameterized Verification of Ring Based Message Passing Systems. In *Proc. CSL*, 2004.
7. E. A. Emerson and K. S. Namjoshi. Reasoning about Rings. In *Proc. POPL*, 1995.
8. T. Henzinger, R. Jhala, and R. Majumdar. Race Checking with Context Inference. In *Proc. PLDI*, 2004.
9. T. Henzinger, R. Jhala, R. Majumdar, and G. Sutre. Lazy abstraction. In *Proc.* 29^{th} *Symp. on Principles of Programming Languages*, pages 58–70. ACM Press, 2002.
10. S. K. Lahiri and R. Bryant. Constructing Quantified Invariants. In *Proc. TACAS*, 2004.
11. A. Pnueli, J. Xu, and L. Zuck. Liveness with $(0, 1, \infty)$ Counter Abstraction. In *Proc. CAV*, 2002.
12. S. Sagiv, T. W. Reps, and R. Wilhelm. Parametric shape analysis via 3-valued logic. In *TOPLAS*, 2002.
13. M. Talupur. *Abstraction Techniques for Infinite State Verification*. PhD thesis, School of Computer Science, Carnegie Mellon, 2006.
14. E. Yahav. Verifying safety properties of concurrent java programs using 3-valued logic. In *In the Proceedings of 18th Symposium on Principles of Programming Languages*, 2001.