Tevfik Bultan: Research Summary

I lead the Verification Laboratory at UCSB. The research I conduct with my students focuses on automated verification techniques and their application to software. As computer systems become more pervasive in everyday life, their dependability becomes increasingly important. The size and the complexity of software systems nowadays inevitably lead to errors during both design and implementation phases. The goal of our research has been to develop automated verification techniques that will help developers in identifying errors in software systems before deployment. Our research can be grouped into three major areas: 1) design for verification, 2) specification and automated verification for service oriented computing, and 3) automated verification of infinite state systems.

Design for verification

There has been significant progress in automated verification techniques based on model checking. However, scalable software model checking remains a challenging problem [56]. I (and many other) researchers believe that scalability in automated software verification can be achieved using two key concepts: abstraction and modularity. Abstraction and modularity are also the key principles of software design. The question is: How can abstraction and modularity at the design level be better integrated with the verification techniques that depend on these principles? I have been actively exploring this question with my students in recent years [16, 17, 31, 26].

We developed a design for verification approach by creating a set of design patterns that support a modular verification strategy [8, 9, 12, 17]. This modular verification strategy is based on stateful, behavioral interfaces that decouple behavior of a module from its environment [7]. So far, we applied this approach to verification of concurrent [8, 10, 12, 13] and distributed [9, 11] systems, which are especially challenging to get right. One application area we focused on was verification of synchronization operations in concurrent programs [7, 8, 10, 12, 13, 60]. The case studies we conducted show that, using our design for verification approach, it is possible to achieve scalable software verification in this area. For example, we demonstrated that our design for verification approach was very effective in finding synchronization errors in a safety critical automated air traffic control software component called the Tactical Separation Assisted Flight Environment (TSAFE) [10, 13, 55].

Recently, we have been investigating specification of rich, behavioral interfaces using grammars [46, 47, 48, 49]. In this approach, interfaces for the components that are outside the scope of the current verification effort are specified using an interface specification language that we developed based on grammars. An interface grammar for a component specifies the sequences of method invocations that are allowed by that component. We built an interface compiler that takes the interface grammar for a component as input and automatically generates a stub for that component. This stub is basically a parser that uses the incoming method invocations as lookahead symbols. This stub/parser replaces the corresponding component during verification, either to assuage the state space explosion, or to provide an executable environment for the part that is being verified. We conducted a case study by writing an interface grammar for the Enterprise JavaBeans (EJB) persistence interface [47, 49]. Using our interface compiler we automatically generated an EJB stub. We used the JPF model checker to check EJB clients using this automatically generated EJB stub. Our results show that EJB clients can be verified efficiently with JPF using our approach, whereas they cannot be verified with JPF directly since JPF cannot handle the native code that is
More recently, we developed a framework for checking interface conformance for web services based on interface grammars [46]. Given a web service interface specification, we automatically generate web service server stubs (for client verification) and drivers (for server verification) and then use these stubs and drivers to check the conformance of the client and server to the interface specification. We implemented this framework by using interface grammars as the interface specification language. Our interface compiler automatically generates a web service stub or a web service driver from a given interface grammar. We conducted a case study by applying these techniques to the Amazon E-Commerce Service. We discovered two errors that were caused by the discrepancies between our understanding of the Amazon E-Commerce Service interface (based on the provided documentation) and its implementation.

**Specification and automated verification for service oriented computing**

An exciting part of my research in last several years has been specification and automated verification of web services. We all know that Web-based software applications, which enable user interaction through web browsers, have been extremely successful. Nowadays, one can look for and buy almost anything online, from a book to a car, using such applications. A promising extension to this framework is the service oriented computing paradigm, which promotes the development of Web accessible software applications that interact with each other through the Internet. Service oriented computing has the potential to have a huge impact on business-to-business applications similar to the impact interactive web software had on business-to-consumer applications.

Service oriented computing provides technologies that enable multiple organizations to integrate their businesses over the Internet. Typical execution behavior in this type of distributed system involves a set of autonomous peers interacting with each other through messages. A fundamental problem in developing dependable web services is analyzing their interactions. This is a challenging problem due to the distributed nature of web services [54]. Desired behaviors have to be specified as constraints on the interactions among different peers since no single party has access to the internal states of all the participating peers, and the interactions among peers are the only observable global behavior. Moreover, for this type of distributed systems, it might be worthwhile to specify the interactions among different peers before the services are implemented. Such a top-down design strategy may help different organizations to better coordinate their development efforts.

We developed a formal model for web services that is based on a set of peers that interact with each other via synchronous and/or asynchronous messages [19]. We model the interactions among a set of peers as conversations: the global sequences of messages exchanged among the peers [18]. A choreography specification identifies the set of allowable conversations for a composite Web service. An orchestration, on the other hand, is an executable specification that identifies the steps of execution for the peers.

Our conversation based modeling framework leads to the following interesting problems: realizability, synthesis, and synchronizability [20]. A choreography specification is realizable if the corresponding conversation set can be generated by a set of peers [38, 40, 43, 44, 28, 27]. This step is necessary to guarantee that the choreography specifications that are developed in a top-down manner are implementable. A related problem is automated synthesis of peer implementations from a given choreography specification. Synchronizability analysis, on the other hand investigates
the effects of asynchronous vs. synchronous communication to improve the efficiency of interaction analysis [39, 45]. A set of asynchronously communicating peers are synchronizable if their conversation set does not change when asynchronous communication is replaced with synchronous communication. Replacing asynchronous communication with synchronous communication enables more efficient analysis by removing the communication channels from the state space of the system.

We developed a tool called Web Service Analysis Tool (WSAT) [41] for analyzing conversations. WSAT verifies LTL properties of conversations, checks sufficient conditions for realizability and synchronizability, and synthesizes peer implementations from choreography specifications. In order to model XML data, WSAT uses a guarded automata model where the guards of the transitions are written as XPath expressions. WSAT uses the explicit-state model checker SPIN for LTL model checking by translating the guarded automata model to Promela [42]. WSAT has been used to analyze realizability and synchronizability of composite Web services specified using BPEL orchestration language and conversation protocols [40], which is a formalism for choreography specification and analysis.

Modeling, analysis and verification of web services is a very active area of research in the WWW community. Our work had a significant impact in this area and our papers on this topic are frequently cited.

Automated verification of infinite state systems

One direction of research that I have pursued for over a decade has been motivated by the need to address the fundamental limitations of automated verification techniques that hinder their applicability to software. One such limitation is the fact that software systems have data-types with arbitrarily large domains and well known theoretical results show that automated verification of such systems is impossible. My research on verification techniques for infinite state systems lead to heuristic solutions to this problem. The basic idea is to search an infinite state space using conservative approximation heuristics that over or under-approximate the behaviors of a system, and then, to use these approximations either to guarantee the absence of bugs or to identify the existing bugs.

In software systems, it is common to have linear arithmetic constraints on integer variables influence the control flow. Hence, automated verification of systems specified using linear arithmetic constraints is an important problem for software verification. My earlier results in this area showed that Presburger arithmetic formulas stored as polyhedra augmented with divisibility constraints are an effective symbolic representation for verification of such systems [23, 24]. Later on, we investigated the use of an efficient automata encoding as an alternative to the polyhedral representation, and compared the efficiency of these two approaches [1, 4]. We showed that image computation, which is a crucial and time consuming step in symbolic verification, can be done efficiently on automata encoding of arithmetic constraints for a class of systems that is common in practice [3]. We also developed a conservative approximation heuristic by defining a widening operator for the automata representation of arithmetic constraints [5], and showed that this approximation is precise for a restricted class of systems. We also investigated the efficiency of BDDs as a symbolic representation for linear arithmetic constraints on bounded integer variables [15] and we demonstrated that the performance of the well-known model checking tools SMV and NuSMV can be improved significantly based on our results [2, 6].

Automated verification techniques for software systems have to deal with a variety of vari-
able types. I developed a technique called composite model checking [21, 22], which combines type-specific symbolic representations and provides a uniform approach for verification of software systems with heterogeneous data types. My students and I implemented these ideas in a symbolic manipulator called Composite Symbolic Library [61, 62], which combines a BDD manipulator (for boolean and enumerated types) and a Presburger arithmetic manipulator (for integers) to handle manipulation of predicates on multiple types. We developed several heuristics for efficient manipulation of the disjunctive composite representation used by the Composite Symbolic Library [59]. We also integrated shape analysis to the composite model checking framework in order to analyze systems with recursive data types such as linked lists [58].

My students and I developed a tool called Action Language Verifier (ALV) [25, 57] that implements all the infinite state model checking techniques mentioned above. Action Language is a specification language for reactive and concurrent software systems [14] that supports both synchronous and asynchronous compositions and hierarchical specifications. ALV has been used in verification of parameterized cache coherence protocols [35], parameterized hierarchical state machines [63], workflow specifications [36, 37], requirements specifications [29, 30] and concurrency control components in Java programs [7, 8, 10, 12, 13, 60]. ALV is the leading infinite state verification tool for systems specified with linear arithmetic constraints on integer variables, and it has been used as a benchmark for comparing the performance of new verification tools and techniques in this domain.

In a recent work, we used ALV to analyze constraints about the sizes of collection types in Object Constraint Language (OCL) specifications [65]. We conducted a case study on the OCL specification of the Java Card API using the size analysis technique we developed. The OCL specification of the Java Card API consists of 31 classes and 150 methods. Verification of size constraints with ALV took only a few seconds per class and we revealed errors in 26 out of the 150 method specifications.

Recently, we have been working on extending our results on verification of infinite state systems to string analysis. Unsanitized string variables are a common cause of security vulnerabilities in Web applications. In typical interactive Web applications, user provided input strings are often used to query back-end databases. If the user input is not properly sanitized, the input strings that contain hidden destructive commands can be sent to back-end databases and cause damage. The goal of string analysis is to automatically check that the string variables are properly sanitized. Since strings can take arbitrarily large values, this is an infinite state verification problem. Existing approaches to this problem make very coarse approximations by setting the string values to all possible strings in order to achieve convergence. We have shown that our earlier results on widening automata [5] can be used in string analysis by building an automata based string analysis tool [64]. Since it is more precise, our string analysis tool is able to verify correctness of sanitization routines that existing tools are unable to verify. We are currently in the process of extending this approach to simultaneous analysis of string and integer variables, which can be useful in analyzing buffer overflows.

We have also worked on theoretical aspects of infinite state verification. We investigated the decidability of verification problems for restricted computation models. Our results identify several classes of infinite state systems for which sound and complete verification is possible. We showed that binary, forward, and backward reachability and safety problems for reversal-bounded counter machines are decidable [52, 53]. We also showed that when these machines are extended with an unrestricted pushdown stack, a restricted read/write tape or a restricted queue, the reachability properties of these extended systems are still decidable [50, 51]. Finally, we showed that the binary
reachability of timed automata with integer-valued clocks augmented with a pushdown stack is decidable [34] and this result also extends to pushdown timed automata with guards that can access the past values of finite state variables [33, 32].

Publication record

My publications appear in the top conferences in the three areas that relate to my research. In the area of software engineering my publications appear in top conferences such as ISSTA, ASE, FSE and ICSE (I published 12 papers in these four conferences). These conferences have very low acceptance rates (for example, 17% for FSE 2007, 12% for ASE 2007, 21% for ISSTA 2007). In the area of web services my publications appear again in top conferences such as WWW and ICWS (I published 5 papers in these two conferences) with similar acceptance rates (for example, 14% for WWW 2005, 19% for ICWS 2005). Finally, in the automated verification area, again my publications appear in top conferences such as CAV and TACAS (I published 10 papers in these two conferences). Other conferences and workshops I have published are also high quality international conferences and workshops but tend to be more focused (such as SPIN, MEMOCODE, CIAA, SAS, CONCUR, CP).

My publication record contains a total of 70 refereed publications, including 21 journal publications. Although my publication record is strong in terms of quantity, I am especially proud of the quality of my publications. It is common for program committees of computer science conferences to select the top 3 to 5 papers from conference proceedings and ask the authors to submit an extended version to a journal, to be published in a special issue. The extended journal versions are required to include substantial new material and go through the regular refereeing process before being accepted by the journals. I am very proud that my conference papers are consistently selected among the top papers in prestigious international conferences. 10 of my journal publications are extended versions of conference papers that were selected as one of the top-papers in the conferences they appeared, and were invited for a journal publication:

- [49] was selected as one of the top papers in ISSTA 2007 and the extended version will appear as [47]
- [28] was selected as one of the top papers in SOCA 2007 and the extended version will appear as [27]
- [30] was selected as one of the top papers in MEMOCODE 2006 and the extended version will appear as [29]
- [10] was selected as one of the top papers in ASE 2005 and the extended version appeared as [13], this paper also received the ACM SIGSOFT Distinguished Paper Award, and the ASE 2005 Best Paper Award
- [43] was selected as one of the top papers in ICWS 2004 and the extended version appeared as [44]
- [38] was selected as one of the top papers in CIAA 2003 and the extended version appeared as [40]
- [2] was selected as one of the top papers in TACAS 2003 and the extended version appeared as [6]
• [32] was selected as one of the top papers in CIAA 2001 and the extended version appeared as [33]

• [62] was selected as one of the top papers in TACAS 2001 and the extended version appeared as [61]

• [21] was selected as one of the top papers in ISSTA 1998 and the extended version appeared as [22]

References


